2011 Ninth Working IEEE/IFIP Conference on Software Architecture

Handling Variability in Software Architecture: Problems and Implications

Matthias Galster
University of Groningen, The Netherlands
m.r.galster@rug.nl

Paris Avgeriou
University of Groningen, The Netherlands
paris@cs.rug.nl

Abstract—**Variability helps manage differences and commonalities within and across software systems. As variability is reflected in and facilitated through the software architecture, it is important to understand the problems that architects face when carrying out their tasks. This would help us improve methods for architecting variability-intensive software systems. In this paper, we therefore present an exploratory study to identify problems that occur when performing variability-related tasks during software architecting. Our subjects were 27 graduate students. We identified eleven problems as experienced by the subjects of our study. The paper also presents implications of the findings for the software architecture field. In particular, we discuss implications for architecture description, methods and tools, and the training of architects.**

Keywords-software architecture; variability; exploratory study

I. INTRODUCTION

A. Research Problem

Variability in software systems is commonly understood as the ability of a software artifact to be changed (e.g., configured, customized, extended) for a specific context, in a preplanned manner [1]. Mechanisms to accommodate variability include software product lines, configuration wizards / tools in commercial software, configuration interfaces of software components, or dynamic runtime composition of web services [2-3].

So far, variability has primarily been addressed in the software product line domain [4]. However, variability imposes challenges on software development in general [4] as variability is a key fact of many, if not most, systems [2]. Moreover, variability is a relevant characteristic of the architectures of software systems [2, 5] and is explicitly reflected in and facilitated through the architecture [6]. Software architects encounter many situations where variability occurs. Examples include the deferral of design and implementation decisions to the latest point that is economically feasible, the configuration of single systems for customization, multiple deployment / operation and / or maintenance scenarios, the planned evolution of a system over its life cycle, self-* (-adaptive, -healing, -managing, etc.) systems, or the need to achieve system qualities related to change (e.g., adaptability) [2].

As with many system properties, identifying and managing variability of a system (either single systems, product lines, system of systems, etc.) early on, and in particular during architecting, is preferred over discovering and addressing variability later in the life cycle [7]. Moreover, as variability is pervasive, architects need proper support for dealing with it. It is essential for the architect to have suitable methods and tools for handling (i.e., representing, managing and reasoning about) variability. This is particularly true as the software architecture discipline acknowledges that variability affects the whole architecture and is a concern of many different stakeholders.

However, there are currently no commonly accepted approaches that deal with variability holistically at the architecture level [5]. Instead, dealing with variability is usually limited to feature modeling or product configuration [4]. Variability in the architecture is a complex concept and dealing with it is a multi-faceted activity. Consequently, in order to develop appropriate support for architects to deal with variability, we need to comprehend the various problems architects face when attempting to carry out variability-related tasks. Therefore, in this paper we address the question of *what problems are experienced while performing variability-related tasks when architecting a software-intensive system*.

B. Goals and Contribution

Rather than identifying any potential problems that occur during software architecting (and that stem from generic architecting tasks) and then selecting the subset of problems related to variability, we explicitly focus on problems that occur while performing variability-related tasks (e.g., identifying variation points). We first report on an exploratory study with 27 graduate students to investigate variability-related problems. Then, we discuss implications of the results from this study. Thus, we contribute a) a list of eleven empirically-grounded problems identified by the study participants, and b) a discussion of the implications of these problems on architecture description, methods and tools, and the training of architects.

C. Paper Structure

Section II of this paper discusses background and related work. In Section III we introduce our research method. Section IV presents the results of our study and in Section V we discuss their implications. Section VI presents threats to the validity before we conclude the paper in Section VII.

II. BACKGROUND AND RELATED WORK

Supporting variability is essential to manage commonalities and differences across software, and to accommodate reuse in different organizations and product versions. So far, variability has primarily been studied in the software product line (SPL) domain [4, 8]. In SPL, variability is understood as the ability of a software artifact to be configured, customized, extended, or changed for a specific context, in a preplanned manner [1]. Moreover, most definitions understand variability as "anticipated" change, i.e., change that is mostly foreseen, with predefined points of potential change and adaptation ("variation points"), as well as options for how to adapt software systems at variation points (i.e., "variants") [6]. For example, a variation point in terms of a feature in an email client could be the encryption algorithm, and variants to resolve this variation point could be DES or IDEA. In addition, there is evolution of variability which might not be anticipated. Also, in SPL, the concept of "product line architecture" exists: It describes concepts and structures to achieve variation in features of different products, while sharing as many parts as possible in the implementation [9]. Thus, the SPL architecture captures the central design of all products of the SPL, including variability and commonalities in product instances.

However, compared to software architectures, product line architectures have a limited scope with regard to variability. First, product line architectures focus on addressing variability explicitly as "features" and "decisions". On the other hand, variability in software architecture is often treated as a quality attribute [5]. In this sense, software architecture considers variability in a broader scope and acknowledges that variability is a concern of different stakeholders, and in turn affects multiple other concerns. Second, product line architectures encompass limited conceptual models, such as feature models, decision models or component-and-connector models [10]. However, variability and its impact on other models that are particularly relevant for software architecture (deployment models, information models, development models, etc.) has not yet been addressed sufficiently. When describing architectures, different architecture views (and their models) need to be managed, including the consistency between views (and between their models) [11]. Here, variability could either be a concern within a view, or we could have variability-specific views that address detailed variability-related concerns. This is particularly true for the impact of preplanned change on quality attributes. Third, a product line architecture assumes the existence of a product line infrastructure, including related processes (e.g., core asset development, product development, management) [12]. This is rarely the case for many software architectures which should support variability; as argued in Section I, product lines are only one way of facilitating variability.

Chen and Babar identified a broad range of technical and non-technical challenges related to variability in SPL [8]. Challenges that are relevant in the context of software architecting include 1) handling complexity, 2) knowledge harvest and management, 3) variability modeling and documentation, and 4) design decisions management and enforcement. Our work complements [8] by a) focusing on software architecting of systems beyond SPL, and b) investigating concrete tasks that architects perform rather than any task in the software life cycle.

III. Research Method

A. Exploratory Study

Exploratory studies are used when "research looks for patterns, ideas, or hypotheses rather than research that tries to test or confirm hypotheses" [13]. Current research on variability in software architecture has been rather sporadic (e.g., Bachmann and Bass [6]) and there is not much empirically-grounded theory for variability-related issues in software architecture. Thus, an exploratory study was appropriate. As the study was conducted as part of a graduate course on software architecture, we followed the checklist for integrating empirical studies with research and teaching goals, proposed by Carver et al. [14]. The steps of our study are illustrated in Fig. 1.

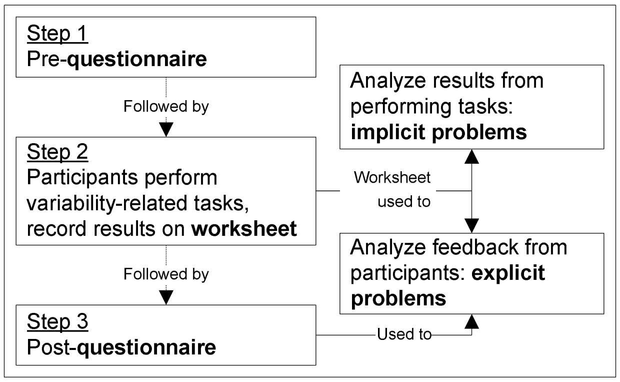
Step 1
Pre-questionnaire
Followed by
Step 2
Participants perform variability-related tasks, record results on worksheet
Followed by
Step 3
Post-questionnaire
Analyze results from performing tasks: implicit problems
Worksheet used to
Analyze feedback from participants: explicit problems
Used to

Figure 1. Steps followed in the exploratory study.

B. Research Question

The goal of the study was to identify problems that occur during software architecting when performing variability-related tasks. This leads to the research question of our study (already stated in Section I): *What problems are experienced while performing variability-related tasks when architecting a software-intensive system*? To answer this question, tasks related to variability were given to participants (see Section III.E and Step 2 in Fig. 1). We then approached the research question in two ways:

- Identification of implicit problems (problems that participants might not be aware of, but which are visible in their work results).
- Identification of explicit problems (problems that are stated by participants, and that may not be evident in the results of the tasks).

C. Study Participants

We used a combination of purposive and availability sampling [15] and involved graduate students in the study:

- Students were available and knowledgeable subjects with background in software engineering and architecture. As argued by Svahnberg et al., in some software engineering areas (e.g., requirements engineering), graduate students might act as substitutes for professionals [16]. Also, Kitchenham et al. argued that students are the next generation of software professionals. Thus, they are relatively close to the population of practitioners [17].

- As argued by Berander, students can be used to investigate phenomena that have not been studied extensively before [18] (e.g., variability in software architecture).
- As argued in Section I, variability is a concern of many systems. Thus, most likely not only the most experienced architects that have to deal with it. Inexperienced architects that work in small or medium-sized projects might also be required to design for variability.

The study was part of a software architecture course for graduate students at the University of Groningen, Netherlands. Participation was not mandatory and had no effect on the grade. In total, we used data from 27 out of 31 participants. When selecting participants, we excluded participants with no industrial software engineering experience, and no academic background in software architecture and software modeling. We included participants with a) at least 1 year software engineering experience in industry, and b) any software architecture experience in industry, and c) any modeling experience in industry. We used these criteria as we consider completely inexperienced students as invalid participants for our study. This resulted in the data of 4 students being discarded. On average, selected participants had 2.3 years of academic education in modeling software, 1 year of experience with modeling software in industry, 1.6 years of software architecture studies and 0.7 years of dealing with architecture in industry. All selected participants had an undergraduate degree in software engineering or computer science. We acknowledge that students have not been extensively trained on how to handle variability, but this is also the case for most practitioners. Threats to validity caused by using students as participants will be discussed in Section VI.

D. *The Software Project*

Participants were provided with a description of a public transportation system[1] (PTS) for buses, trains, subways, etc., including functional and non-functional requirements. This system has been chosen because a) it is a realistic system from the real world, b) it is comprehensible in a short period of time, c) it does not require specific domain knowledge as public transport systems can be experienced in daily life, and d) it provides reasonable variability, but also reasonable commonalities between different instances of the system. In the PTS, requirements are specified so that they address different instances. Instances of the PTS could be created for different cities, in different countries, with different infrastructure.

E. *Tasks Performed by Participants*

When architects in practice express their need for methodological support, they often think in terms of tasks that they need to perform [19]. Therefore, participants were expected to act in the role of a software architect responsible for handling variability issues during the design of the PTS. This means, the PTS was to be designed to accommodate planned variability. Resolving variability and evolution of variability beyond the design phase were not part of the assignment. To identify tasks, we chose four variability issues which, according to Capilla and Babar, are closely related to software architecting in practice [20]: 1) Common and specific requirements, 2) variability dependencies, 3) variability modeling, and 4) binding time.

1) *Tasks related to common and specific requirements*

The PTS includes requirements which are common across all instances of the system. Also, there are requirements which are specific to individual instances of the PTS. Therefore, participants were asked to

- determine possible common features (task 1.1),
- identify possible different features (task 1.2),
- derive variation points (task 1.3), and
- determine variants (task 1.4).

2) *Tasks related to variability dependencies*

Variation points and variants may depend on other variation points and variants. These must be made explicit in the architecture. Thus, participants were asked to identify

- dependencies and types of dependencies between variation points (task 2.1),
- dependencies and types of dependencies between variants (task 2.2), and
- dependencies and types of dependencies between variation points and variants (task 2.3).

3) *Task related to variability modeling*

In task 3, participants were asked to create a variability model. Creating the variability model aggregates the results from all previous tasks into one architectural model. Participants did not have access to a tool. Instead, participants used a simple annotation of UML diagrams as suggested by Sun et al. [21].

4) *Task related to binding time*

In task 4, participants were asked to suggest binding times for variation points (compilation, integration, deployment, runtime). Fritsch et al. discuss techniques to resolve binding time [22]. As binding times are not documented in UML diagrams, participants were asked to keep a separate documentation of binding time information.

F. *Data Collection*

Information about participants was collected through a paper-based pre-questionnaire. This included questions on the background (industry, academia), industrial experience, experience in software engineering, software architecture, variability, and academic degrees obtained so far. These questions were not directly related to the research question but allowed us to filter participants (see Section III.C).

Participants recorded the results from performing the given tasks (Section III.E), time spent working on the tasks as well as problems observed while performing the tasks on a paper-based worksheet. The open questions on the worksheet included question WQ1 ("What problems did you encounter when creating the variability model?") and question WQ2 ("Please record challenges that you face when performing the given tasks. Also, please list the actions you

[1] http://score-contest.org/2011/Projects.php#lethbridge (last access: February 14, 2011)

chose to tackle these challenges."). The results from the tasks allowed us to identify implicit problems. The answers to WQ1 and WQ2 helped us elicit explicit problems.

Behavior and reasoning behind performing tasks were collected on a paper-based post-questionnaire. Questions included PQ1 ("What overall challenges did you encounter when handling variability issues?") and PQ2 ("What other information related to the concept of variability would you have found useful for better performing the given tasks?"). These helped us elicit explicit problems. We also asked how participants felt about the assignment and the tasks, using two questions and a 5-point Likert scale. Questions included PQ3 ("I distinguished variability in quality attributes and variability in functionality.") and PQ4 ("I performed the given tasks in the given order.").

G. Study Execution

The study took place in September 2010. For study execution, all study participants gathered in one room. The pre-questionnaire was distributed and each participant was given a unique identifier (Step 1 in Fig. 1). Anonymizing participants ensured that the researchers were not biased when analyzing the results. After the pre-questionnaire, participants were given an introduction to the assignment. A handout with information on the assignment and the worksheet to record the results were distributed (Step 2 in Fig. 1). The study was explained to participants to mitigate the risk of ambiguous or poorly understood tasks questions. One researcher was available to answer questions during the study. Questions from individuals were answered to all participants so that all participants had the same information. We did not answer questions related to performing the tasks or problems caused by tasks. We only answered questions related to the instructions of the assignment and the questionnaires. After completing the tasks, participants were given the post-questionnaire to record their experience about the assignment (Step 3 in Fig. 1). The study was scheduled for three hours but all participants finished earlier.

H. Data Analysis

We (the authors) analyzed implicit problems that became evident in the results of tasks, as well as explicit problems stated by participants while or after performing tasks.

1) Analysis of performed tasks

Descriptive statistics were used to analyze the data collected for each task (e.g., number of valid variation points, number of identified variants represented in variability model). The identification of problems in the performed tasks was based on the results for each task and each participant as recorded on the worksheet. As the goal was to unveil problems faced by participants, we a) analyzed the result of each task in detail (see Section IV) and b) used a set of quality criteria to evaluate each task (Table I). These criteria were derived based on the tasks given to participants. Some criteria in Table I are subjective (e.g., the number of variation points must be between 3 and 10). These numbers were chosen and refined a) based on looking at example specifications and implementations of the PTS, and b) after reviewing and piloting the study protocol.

TABLE I. QUALITY CRITERIA TO EVALUATE TASKS

#	Task	Description of criterion
1	1.3	VP (i.e., variation points) are realistic and within scope.
2	1.3	VP are related to differences identified in tasks 1.1 and 1.2.
3	1.3	Number of VP exceeds 3 but not 10.
4	1.4	Variants are realistic and within scope.
5	1.4	There are at least 2 variants per variation point.
6	2.1	Dependencies between all VP were identified.
7	2.2	Dependencies between all variants were identified.
8	2.3	Dependencies between all VP and variants were identified.
9	3	Variability model includes all VP, variants and common parts.

Each task that met the criteria was assigned a score of 2. Each task that was not completed was assigned a score of 0, and tasks that were solved partially were assigned a score of 1. This resulted in one score per task and participant. Please note that we did not identify quality criteria for task 1.1, 1.2 and task 4. This is because task 1.1 and 1.2 are the foundation for task 1.3 and task 1.4. We therefore analyzed how common and variable features had been translated into variation points and variants. For task 4 (identification of binding times) we found it difficult to define valid quality criteria as the binding times depend on the individual understanding of participants. Thus, we only judged if binding times are reasonable or not in the context of the results of a particular participant.

2) Analysis of explicit issues stated by participants

Based on the answers to questions WQ1 and WQ2 on the worksheet, and questions PQ1 and PQ2 on the post-questionnaire, a list of all problems was created. Thus, all answers to open questions were read, and phrases of interest were coded with labels to reflect the topic of that phrase [23]. For this purpose we used atlas.ti[2]. We also used constant comparison to analyze and to categorize data as starting points for codes [24]. The codes evolved during analysis. Analyzing qualitative data required integrating data where different participants might have used terms and concepts with different meanings, or different terms and concepts to express the same issue [25]. Thus, we used reciprocal translation [25]. Reciprocal translation helped summarize the newly identified issues that relate to similar issues by translating similar issues and problems into one another. This resulted in a list of unified problems. A frequency analysis of the occurrence of each problem (as formulated before reciprocal translation) indicated how often an issue had been mentioned.

For PQ3 and PQ4, we used simple frequency analysis to count the occurrences of scores provided on a 5-point Likert scale.

[2] http://www.atlasti.com/

IV. Results

In this section, we present the results from analyzing the worksheets and the post-questionnaires. We do not present the detailed outcomes of the tasks performed by participants, but a summary of findings for each task. Overall, we noticed that in some cases the task itself is a problem, whereas in other cases the task is a problem for a subset of participants, and in some cases certain parts of a task cause a problem. For example, in some cases less experienced participants performed worse than other participants. Here, "less" experienced was defined as having less than one year of industrial experience in software architecture. This characterization applied to 12 of the 27 participants.

As we had a single group of participants, we performed an initial cluster analysis on the time spent performing the tasks. However, we could not detect any grouping in participants that would reflect the experience or any other characteristic of participants. This means, the time spent on the tasks was not related to the background of participants. Furthermore, there was no correlation between the different times spent on the various tasks. We only found that participants who spent much time on identifying dependencies between variants also spent much time on identifying dependencies between variation points and variants (Kendall's tau = 0.614, Spearman's rho = 0.706, significant at the 0.01 level). To check how reliable our data are that were not determined as scores using the criteria in Table I, we calculated Cronbach's coefficient alpha as α = 0.78. Usually, $\alpha > 0.70$ is an indicator of reliable measurements [26]. For the score-based data, α was 0.72. Fig. 2 shows the frequency distribution of all scores achieved for the quality criteria listed in Table I.

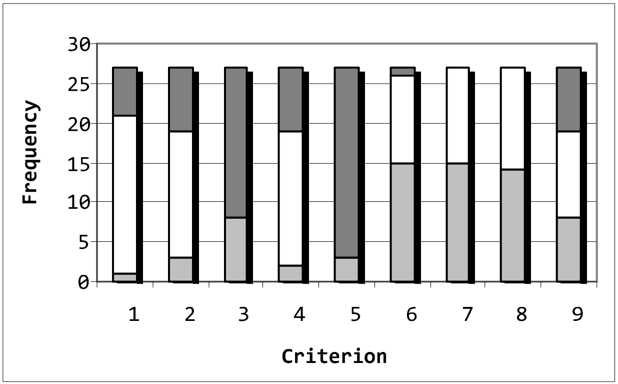

Figure 2. Scores for quality criteria achieved by participants.

Dark grey areas in Fig. 2 indicate how many participants achieved a score of 2, white areas indicate how many participants achieved a score of 1, and bright grey areas indicate the number of participants with a score of 0. We further discuss Fig. 2 in the following subsections.

A. *Tasks related to common and specific requirements*

1) *Common and varying features*

The task given to participants did not differentiate between functional and non-functional features. Yet, during analysis we separated both to get a better understanding of where exactly problems occurred. Table II shows a summary of valid items identified in task 1.1 and task 1.2. By valid we mean that commonalities and differences were realistic and within scope as well as reasonable with regard to the problem description of the PTS. σ denotes the standard deviation. Please note that we could not identify any correlation between valid common and varying features.

TABLE II. Results of Tasks 1.1 and 1.2

Item	Valid items (average)
Common non-functional properties (task 1.1)	51% (σ = 40%)
Varying (functional) features (task 1.2)	59% (σ = 38%)
Varying non-functional properties (task 1.2)	81% (σ = 38%)
Common (functional) features (task 1.1)	86% (σ = 30%)

Finding: Participants had problems to identify common non-functional properties and varying functional features. On the other hand, identifying varying non-functional properties and common features were performed better (51% versus 81% of valid items, see Table II). This result is surprising as it would have been more intuitive to find a difference between functional and non-functional requirements. One reason could be that the PTS had more problematic functional properties than the average software-intensive system. Moreover, we found that less experienced participants performed significantly worse than more experienced participants when identifying common functional features and varying functional features. To identify the difference between less experienced participants and more experienced participants, we used a Mann-Whitney test, which resulted in p = 0.07 for identifying common functional features (based on the percentage of valid common functional features), and p = 0.09 for varying functional features (based on the percentage of valid varying functional features). A small p indicates a significant difference between two groups. Based on the findings, we identified the following problems:

Problem 1: Identification of common non-functional characteristics based on the requirements description.

Problem 2: Identification of varying functional features based on the requirements description (particularly for less experienced participants).

2) *Identification of variation points*

As can be seen in Fig. 2, for the three criteria related to task 1.3, 20 participants reached a score of 1 for criterion 1, 16 participants a score of 1 for criterion 2 and for criterion 3, 19 participants reached a score of 2. When analyzing variation points in detail, we found the following:

- **Valid variation points:** On average, 61% of variation points were valid (σ = 26%). A Mann-Whitney test revealed a significant difference between experienced (70% valid variation points) and inexperienced participants (49% valid variation points), with p = 0.039.
- **Derivation of variation points:** On average, 75% of valid variation points were derived from different features

identified in tasks 1.1 and 1.2. This means, most participants successfully translated different features into variation points.

- **Variation points in variability model:** On average, 71% (σ = 38%) of variation points were translated into the variability model, with only 14 / 27 participants translating all their valid variability points in the variability model. This means, almost 50% of participants failed with translating all variation points into the variability model.

Finding: Many participants created valid variation points. Also, the number of variation points was reasonable. Less experienced participants performed worse when defining variation points. Also, even if variation points are known, their translation into the variability model is a problem. This leads to the following main problems:

Problem 3: Identification of valid variation points (particularly for less experienced participants).

Problem 4: Translation of variation points into variability model.

3) Identification of variants

As can be seen in Fig. 2, for the quality criteria related to task 1.4, 17 participants reached a score of 1 for criterion 4, and 24 participants reached a score of 2 for criterion 5. When analyzing variants, we noticed the following:

- **Valid variants:** On average, 67% of variants were valid (σ = 30%).
- **Variants in variability model:** Similar as with variation points, on average, only 59% of variants were translated into the variability model (σ = 40%). Moreover, participants who successfully translated variation points into the variability model tend to be more successful when translating variants.

Finding: The number of identified variants was reasonable. Translating variants into the variability model appears to cause more problems than defining valid variants. Thus, the following problem was derived:

Problem 5: Translation of variants into variability model.

B. *Tasks Related to Variability Dependencies*

1) Dependencies between variation points

As shown in Fig. 2, 15 participants reached a score of 0 for criterion 6 (task 2.1) and only 1 participant reached a score of 2. A detailed analysis of the dependencies identified by participants showed the following:

- **Valid dependencies:** When identifying dependencies between variation points, on average 34% of dependencies were valid (σ = 44%). This was calculated by determining the number of identified dependencies and the number of valid dependencies, per participant. The fact that σ is larger than the mean indicates highly scattered data related to dependencies between variation points. This means, the average is not a good indicator of how well the task was performed. Thus, a more thorough analysis showed that 16 participants could not identify any valid dependency, whereas 7 participants identified dependencies which were all valid. The remaining participants had some valid dependencies.
- **Valid dependency types:** On average, 57% of identified dependency types were valid (σ = 49%). Eleven participants did not identify any valid dependency type, 13 participants identified one type, 2 participants identified two types and 1 participant identified three dependency types.
- **Dependency types:** The following dependency types between variation points were identified: conflict (9x), requires (7x), and several other types which were named once.

Finding: Identifying dependencies between variation points caused major problems. Moreover, even though dependencies were identified, participants had problems characterizing these dependencies and describing how variation points would affect each other. Finally, a conflict appears to be the most common dependency between variation points.

Problem 6: Identification and characterization of dependencies between variation points.

2) Dependencies between variants

As shown in Fig. 2, 15 participants reached a score of 0 for criterion 7 (task 2.2). A more detailed analysis of dependencies between variants led to the following observations:

- **Valid dependencies:** On average, 46% of dependencies between variants were valid. However, similar to the average of valid dependencies between variation points (see previous subsection), σ = 48% indicates widely scattered results. Thirteen participants could not find any valid dependency whereas 11 participants identified dependencies which were all valid.
- **Valid dependency types:** On average, 48% of dependency types were valid (σ = 49%). Thus, a more detailed analysis showed that 13 participants could not identify any valid dependency type, 10 participants identified one valid dependency type and 4 participants identified two valid dependency types.
- **Dependency types:** Dependency types that were identified include conflicts (11x) and some other types named once. These types are similar as the types for dependencies between variation points (see previous subsection).

Finding: Dependencies between variants were identified even worse than dependencies between variation points. Again, a conflict between variants seems to be the most obvious and most frequently identified dependency. This leads to the following problem:

Problem 7: Identification and characterization of dependencies between variants.

3) Dependencies between variation points and variants

In Fig. 2 we can see that 14 participants reached a score of 0 for criterion 8 (task 2.3). In detail, we found the following:

- **Valid dependencies:** The average number of valid dependencies was 43% (σ = 49%). Again, the large standard derivation is an indicator for the spreading of results, as already observed for dependencies between variation points in Section IV.B.1) and between variants in Section IV.B.2). Fourteen participants could not identify any valid dependency whereas 11 participants identified only valid dependencies.

- **Valid dependency types:** On average, 38% of identified dependency types were valid (σ = 49%). Sixteen participants could not identify a single valid dependency type. Ten participants identified one valid dependency type and 1 participant identified four valid dependency types (e.g., implements, optional, supports).
- **Dependency types:** The dependency types identified by participants were as follows: implements (7x), conflict (2x), and several other types named once.

Finding: Identifying dependencies between variation points and variants causes problems. This is a surprising finding as the relationship between variation points and variants could be considered as trivial, given the fact that variants are assigned to variation points as implementation options. The resulting problem is:

Problem 8: Identification and characterization of dependencies between variation points and variants.

C. *Variability Modeling*

Some of the results related to variability modeling (e.g., if commonalities and differences were represented in the variability model) were already discussed in Section IV.A. Therefore, this section focuses on observations made from the variability models. Variability models should combine all information from the previous tasks. The distribution of quality scores for criterion 9 (task 3) can be seen in Fig. 2. In detail we found the following:

- Six participants introduced new components into the variability model that were neither identified as variation points, nor as common features.
- Sixteen variability models only show variable parts but ignore common parts, and only 1 participant completely included variable and constant parts of the PTS.
- Only 3 participants were able to include correct dependencies (derived from task 2.1 to 2.3).
- The most difficult problems reported by participants by answering WQ1 were modeling of non-functional variability (2x) and dependencies between variation points, variants, etc. (7x). This confirms the findings from the previous subsections, and in particular Problem 6, Problem 7 and Problem 8.

Finding: The main problem when creating the variability model appears to be the lack of linking different concepts (commonalities, differences, variation points, variants, etc.) together in the variability model. Thus, besides Problem 4 and Problem 5, the resulting problems are:

Problem 9: Modeling of common parts.

Problem 10: Modeling dependencies between variation points, variation points and variants, and variants.

D. *Binding Time*

In our study, 8 out of 27 participants could not identify a single reasonable binding time for variants. Only 5 participants identified valid binding times for all variation point – variant combinations. On average, 46% of binding times were reasonable (σ = 39%). This is an indicator that the concept of binding times was understood, but causes problems. The problem derived from this finding is essentially the whole task:

Problem 11: Identification of binding times for variation points and variants.

E. *Problems Identified by Participants*

As part of performing the tasks, we also asked participants to document challenges that they faced while performing the tasks. Moreover, we asked participants to document how they tackled these challenges (question WQ2 in Section III.F). The challenges were grouped based on the tasks. In Fig. 3, we show a frequency distribution of how often a task was mentioned to impose a challenge. Fig. 3 confirms problems Problem 1 and Problem 2, as well as Problem 9 and Problem 10. However, Fig. 3 also shows how the results of participants and their own judgment of problems diverge. As we have shown before, identifying binding times was performed poorly. However, as shown in Fig. 3, only 1 participant stated problems with task 4.

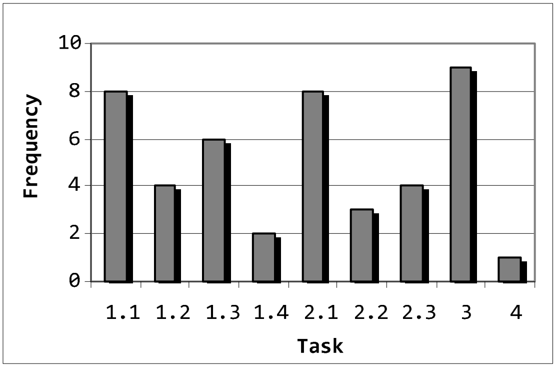

Figure 3. Problems imposed by tasks as stated by participants.

Some of the concrete problems reported by participant include problems on how to decompose the architecture to address variability sufficiently. This challenge might be considered as a general architecture problem, not specific to variability. However, this problem can be related to Problem 9 and Problem 10. Moreover, participants found it challenging to handle the vagueness in requirements which are the cause for variability. Vagueness in requirements can be considered as natural or even as a precondition that makes a design for variability necessary. Thus, we did not consider this problem as a problem stemming from variability-related tasks in software architecture. The actions that were taken to tackle challenges (second part of WQ2) were reported poorly and could not be used for further analysis. This is an indicator that participants did not know how to tackle the problems they faced. For example, 1 participant stated that s/he tried to keep things simple and assume things when identifying variants.

F. *Results from Post-questionnaire*

Due to space limitations, we only provide a summary of the results obtained from questions PQ1 to PQ4 on the post questionnaire (see Section III.F). The results of the open coding of information about overall challenges (PQ1) are as follows: Identification of variation points (5x), identification of variants (4x), identification of dependencies (5x), handling non-functional requirements (2x), determine level of abstraction (2x). Some other challenges occurred once. These findings confirm Problems 3, 6, 7 and 8.

Answers to PQ2 (what other information would have helped with performing the tasks) included the need for more information about the system (3x). Again and similar to the problem of decomposing the architecture discussed above, this is a generic problem and not specific to variability.

For PQ3 (distinguishing variability in quality attributes and functionality) we found that 11 participants clearly distinguished variability in non-functional properties from variability in functional properties. Sixteen participants indicated neutral. Yet, the results did not indicate any correlation between participants that claimed that they distinguished non-functional properties and their success in identifying common and different non-functional properties. This confirms Problem 1 and Problem 2.

For PQ4 (performing tasks in given order), 19 participants indicated that they performed the tasks in the given order; 8 participants indicated they changed the order. In a follow-up analysis we found that participants that followed the tasks in the given order performed better when identifying common features, when translating variation points in the variability model, and when translating variants in the variability model. This means, systematically moving from the problem space (common and different features) to the solution space (variability model) increases the quality of results, compared to developing the variability model without thoroughly identifying and analyzing common and different features.

V. Implications

The goal of our study is to identify problems that occur while performing variability-related tasks when architecting software-intensive systems. In this section we elaborate on the implications of these problems.

A. Architecture Description

Even though literature acknowledges that variability is a key issue in most systems [2], the problems we identified are currently not dealt with systematically when describing architectures. It is commonly agreed that the architecture description of a system must deal with many different concerns of various stakeholders [11]. One way to address different concerns in software architecture is to organize the architecture description according to multiple views. Views are created using predefined architecture viewpoints (i.e., conventions for constructing and interpreting views) [11]. A viewpoint usually describes the stakeholders, their concerns, model kinds, meta-models, modeling techniques, operations, notations, etc. used in the views. As we currently lack viewpoints that frame variability-specific concerns and ensure consistency among different views of a variability-intensive system, our results unveil a discrepancy between architecture practice and research. Even though variability has been addressed in other domains of software engineering (e.g., in terms of tracing volatile requirements in requirements engineering), identifying and characterizing variability-related dependencies and ensuring their consistency has not yet been acknowledged in software architecture research.

B. Methods and Tools

Lately, there has been research on variability in product lines [4, 8]. Moreover, as found in our previous work [19], software architects in practice think in terms of development activities that they perform when describing their needs for new methods or tools. Therefore, our findings that target architecting areas where variability has a particular impact could thus help improve methods and tools. As argued in [8], tool support for variability in product lines to model variability has been proposed. On the other hand, complexity, knowledge harvest and management, design decisions and enforcement are not addressed by approaches. As shown in Table III, these are also the areas where we identified our eleven problems. Therefore, the following types of support should be provided by methods and tools:

- Management of dependencies between variation points, variants, and variation points and variants.
- Transition from variation points and variants to architecture models.

Both rank high in terms of urgency, given the low quality score that related tasks achieved. New methods / tools might be based on existing approaches from the product line domain (e.g., feature modeling), but to fully address the needs of architects (see Section I), new approaches are needed. Moreover, given the results of participants, we found several problems as more difficult than others. For example, identifying binding times was performed poorly and thus needs special attention. On the other hand, new methods and tools for handling variability types or deriving variation points from varying requirements appear less difficult.

C. Training of Architects

To discuss the implications on the training of architects, we first compared the problems we identified with challenges related to managing variability in the product line domain [8], see Table III. This helped us identify training areas based on our findings, and confirm these with findings of other researchers. Please note that we only take challenges from [8] into consideration that are related to architecting. Unfortunately, challenges identified in [8] are quite generic, which is why we could not accurately assign our problems to challenges. However, it helped us structure the eleven problems we identified. Table III shows that most problems are related to variability modeling. This means, our participants and industrial SPL professionals considered modeling as a major challenge. Consequently, variability modeling should be a focus of training architects.

As shown before, there are specific problems that only certain participants experienced. Therefore, training architects for variability can be separated into training for less experienced architects and architects in general. For less experienced architects, training should focus on the identification of varying functional features, and the identification of valid variation points. Training for architects in general should focus on identifying and characterizing dependencies between variation points and variants.

TABLE III. MAPPING OF VARIABILITY-RELATED PROBLEMS TO CHALLENGES RELATED TO SPL IN INDUSTRY

Problem	Challenge (Chen and Babar)			
	Complexity	*Knowledge harvest / management*	*Variability modeling*	*Design decision management*
1: Identification of common non-functional characteristics	X	X		
2: Identification of varying functional features	X	X		
3: Identification of valid variation points			X	X
4: Translation of variation points into variability model			X	
5: Translation of variants into variability model			X	
6: Identification / characterization of dependencies between variation points	X		X	
7: Identification / characterization of dependencies between variants	X		X	
8: Identification / characterization of dependencies between variation points and variants	X		X	
9: Modeling of common parts			X	X
10: Modeling depencendies between variation points, variation points and variants, and variants			X	X
11: Identification of binding times for variation points and variants	X			

VI. THREATS TO VALIDITY

The validity of our results is subject to several threats.

Construct validity: Construct validity is concerned with how well an instrument measures a construct (theoretical concept) [27]. Our constructs are "problems" when performing variability-related tasks and were measured by a practical assignment. The assignment and the questions were piloted to determine if they help investigate our constructs. However, only one specific architecting experience was used to collect data. For other architecting projects, the perception of participants about "problems" might be different. Moreover, we judged the data ourselves; however, we used multiple sources of evidence (see also internal validity). To check if and how the constructs reflect their real-world counterparts, we related them to experiences of practicing architects (see Table III).

Internal validity: Internal validity relates to the extend to which the design and analysis may be compromised by confounding variables and other sources of bias [17]. In our study, some of the quality criteria used to evaluate the tasks are subjective. We tried to mitigate this risk by a) basing the criteria on existing implementations and specifications of the PTS, and b) refining them based on piloting and reviewing the study protocol, including data collection instruments and analysis. Moreover, we judged the data ourselves, rather than utilizing an expert reviewer. However, we cross-checked the results from different perspectives, e.g., checked if implicit problems were confirmed by explicit statements from participants. Also, it is sometimes argued that students lack commitment in academic studies. We tried to accommodate this problem by scheduling the study for September. During this time of the academic year, students tend to have less pressure to prepare assignments or for exams and thus are willing to invest effort. Furthermore, some participants might have misunderstood tasks. To mitigate this risk, researchers were available to answer questions throughout the study.

External validity: External validity is concerned with the ability to generalize the results of the study [28]. Due to the small sample size, results might be difficult to generalize. As the number of potential participants was restricted by the number of students enrolled in the software architecture class, we did not have any impact on the sample size. Consequently, we could not apply power analysis to determine the sample size needed to achieve statistically significant results. Moreover, we used availability sampling to recruit participants with a certain background. It would have been extremely difficult (if not impossible) to conduct such a study in industry using random sampling, given the required time commitment of participants. However, the use of students should not invalidate the results of this study; many important results have been found in other software engineering [16, 29] or software architecture [30-32] studies where students were used as subjects. Yet, we cannot claim that we completely mitigated the risk of using students and the resulting threat to external validity.

Reliability: To ensure reliability of our findings and the repeatability of the study, we piloted data collection instruments and reviewed the study protocol. However, some participants might have had problems to understand the PTS and its domain. However, the post-questionnaire explicitly asked for problems with the PTS but the responses did not indicate any problems with understanding the assignment.

VII. CONCLUSIONS AND FUTURE WORK

Variability is a concern of many, if not most software systems. To handle variability in the architecture, and understanding of the problems faced by architects is needed. Thus, this paper addressed the question of *what problems are experienced while performing variability-related tasks when architecting a software-intensive system*. We presented an exploratory study and identified eleven major problems based on variability-related tasks performed by participants. The problems were discussed in Section IV. We found that some of the problems are only relevant for some participants and some problems are identical to the tasks. Furthermore, we discussed implications of the problems on architecture description, methods and tools, and training of architects. Having empirically-grounded problems increases the quality and acceptance new methods and training plans.

Our future work is two-fold: First, based on these results and similar as in [8], we will do more systematic evaluations using practitioners as subjects to cross-check the problems identified in this study, and to identify more problems. Second, we will use the problems to explore concerns for variability viewpoints. When constructing viewpoints for variability, valid concerns need to be identified. By valid, we mean that evidence for the relevance of concerns must exist, in the sense that concerns a) represent real stakeholder interests in the system, and b) are related to variability. The problems we identified can be used to derive empirically-grounded concerns for variability viewpoints.

ACKNOWLEDGMENT

We thank the participants of the study. We also thank the anonymous reviewers for their comments. This work has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.207 SAS-LEG: Software as a Service for the varying needs of Local EGovernment.

REFERENCES

[1] F. Bachmann and P. C. Clements, "Variability in Software Product Lines," SEI CMU, Pittsburgh, PA, Technical Report CMU/SEI-2005-TR-012, 2005.

[2] R. Hilliard, "On Representing Variation," Proc. Workshop on Variability in Software Product Line Architectures, ACM, 2010, pp. 312-315.

[3] J. van Gurp and J. Bosch, "Proceedings of the Software Variability Management Workshop," Groningen, The Netherlands, 2003, p. 142.

[4] L. Chen, M. A. Babar, and N. Ali, "Variability Management in Software Product Lines: A Systematic Review," Proc. 13th International Software Product Line Conference (SPLC), Carnegie Mellon University, 2009, pp. 81-90.

[5] M. Galster and P. Avgeriou, "The Notion of Variability in Software Architecture - Results from a Preliminary Exploratory Study," Proc. 5th International Workshop on Variability Modelling in Software-intensive Systems (VaMoS), ACM, 2011, pp. 59-67.

[6] F. Bachmann and L. Bass, "Managing Variability in Software Architectures," Proc. 2001 Symposium on Software Reusability, ACM, 2001, pp. 126-132.

[7] S. Thiel and A. Hein, "Modeling and Using Product Line Variability in Automotive Systems," *IEEE Software,* vol. 19, pp. 66-72, July / August 2002.

[8] L. Chen and M. A. Babar, "Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges," Proc. 14th International Software Product Line Conference, Springer Verlag, 2010, pp. 1-15.

[9] M. Jazayeri, F. van der Linden, and A. Ran, Software Architecture for Product Families: Principles and Practice. Reading, MA: Addison-Wesley, 2000.

[10] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud, "PuLSE: A Methodology to Develop Software Product Lines," Proc. Symposium on Software Reusability, ACM, 1999, pp. 122-131.

[11] ISO/IEC, "Systems and Software Engineering - Architecture Description." ISO/IEC 42010, 2010.

[12] P. Clements and L. Northrop, Software Product Lines - Practices and Patterns. Boston, MA: Addison-Wesley, 2001.

[13] P. Vogt, Dictionary of Statistics and Methodology - A Non-technical Guide for the Social Sciences. Thousand Oaks, CA: Sage Publications, 1998.

[14] J. C. Carver, L. Jaccheri, S. Morasca, and F. Shull, "A Checklist for Integrating Student Empirical Studies with Research and Teaching Goals," *Empirical Software Engineering,* vol. 15, pp. 35-59, February 2010.

[15] J. W. Creswell, Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Thousand Oaks, CA: Sage Publications, 2002, p. 246.

[16] M. Svahnberg, A. Aurum, and C. Wohlin, "Using Students as Subjects - An Empirical Evaluation," Proc. 2nd International Symposium on Empirical Software Engineering and Management, ACM, 2008, pp. 288-290.

[17] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 28, pp. 721-734, August 2002.

[18] P. Berander, "Using Students as Subjects in Requirements Prioritization," Proc. International Symposium on Empirical Software Engineering, IEEE Computer Society, 2004, pp. 167-176.

[19] T. B. C. Arias, P. America, and P. Avgeriou, "Defining and Documenting Execution Viewpoints for a Large and Complex Software-intensive System," *Journal of Systems and Software,* in press.

[20] R. Capilla and M. A. Babar, "On the Role of Architectural Design Decisions in Software Product Line Engineering," Proc. Second European Conference on Software Architecture, Springer Verlag, 2008, pp. 241-255.

[21] C. Sun, R. Rossing, M. Sinnema, P. Bulanov, and M. Aiello, "Modeling and Managing the Variability of Web-Service-based Systems," *Journal of Systems and Software,* vol. 83, pp. 502-516, March 2010.

[22] C. Fritsch, A. Lehn, and T. Strohm, "Evaluating Variability Implementation Mechanisms," Proc. International Workshop on Product Line Engineering, Fraunhofer IESE, 2002, pp. 59-64.

[23] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering,* vol. 25, pp. 557-572, July 1999.

[24] A. C. Strauss and J. Corbin, Basics of Qualitative Research: Grounded Theory Procedures and Techniques, 2nd ed. Thousand Oaks, CA: Sage Publications, 1990.

[25] G. W. Noblit and R. D. Hare, Meta-Ethnography: Synthesizing Qualitative Studies. Newbury Park, CA: Sage Publications, 1988.

[26] J. Rosenberg, "Statistical Methods and Measurement," in Guide to Advanced Empirical Software Engineering, F. Shull, J. Singer, and D. Sjoberg, Eds. London, UK: Springer Verlag, 2008, pp. 155-184.

[27] F. Shull, J. Singer, and D. Sjoberg, Guide to Advanced Empirical Software Engineering. Berlin / Heidelberg: Springer Verlag, 2007.

[28] C. Wohlin, M. Hoest, and K. Henningsson, "Empirical Research Methods in Software Engineering," in Empirical Methods and Studies in Software Engineering, R. Conradi and A. I. Wang, Eds. Berlin / Heidelberg: Springer Verlag, 2003, pp. 7-23.

[29] A. S. Danesh and R. Ahmad, "Study of Prioritization Techniques Using Students as Subjects," Proc. International Conference on Information Management and Engineering, IEEE Computer Society, 2009, pp. 390-394.

[30] J. A. Miller, R. Ferrari, and N. H. Madhavji, "Architectural Effects on Requirements Decisions: An Exploratory Study," Proc. Seventh Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, 2008, pp. 231-240.

[31] B. J. Williams and J. C. Carver, "Characterizing Software Architecture Changes: An Initial Study," Proc. First International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, 2007, pp. 410-419.

[32] B. Anders, J. Fellmann, M. Lindvall, and I. Rus, "Experimenting with Software Architecture Flexibility Using an Implementation of the Tactical Separation Assisted Flight Environment," Proc. 29th Annual IEEE/Nasa Software Engineering Workshop, IEEE Computer Society, 2005, pp. 275-284.