Configuration Knowledge of Software Product Lines: A Comprehensibility Study

Elder Cirilo, Ingrid Nunes, Alessandro Garcia, Carlos J.P. de Lucena
1Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil
{ecirilo, ionunes, afgarci, lucena}@inf.puc-rio.br
2King’s College London, Strand, London, WC2R 2LS, United Kingdom

ABSTRACT

The configuration knowledge is a key element to the success of software product lines, as it defines constraints on how product line variability should be composed to derive products. Even though configuration knowledge specification is a long standing problem in software product line engineering, the impact of different specification techniques on comprehensibility has never been studied. This paper presents an empirical study to evaluate and compare three techniques for configuration knowledge specification. Each of them is centered on different means to express the configuration constraints: annotations, general-purpose modeling, and domain-specific modeling. Our results suggest that: (i) the use of domain-specific abstractions tends to facilitate the comprehension of coarse-grained variability; (ii) the use of general-purpose models imposes certain restrictions on the location and comprehension of the configuration knowledge; and (iii) the correct comprehension of configuration constraints is not associated with individual expertise.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures—Domain-specific architectures; D.2.8 [Software Engineering]: Metrics

General Terms
Experimentation

Keywords
Software Product Lines, Configuration Knowledge, Controlled Experiment

1. INTRODUCTION

The adoption of configurable product lines for building families of enterprise systems is increasingly growing. A configurable product line [4] is a subclass of software product lines that is tailored into single products without designing or programming new completion code. Systematic variability management as configuration potentially results in significant gains in software development such as reduced time-to-market and costs, but it requires explicitly specifying the configuration knowledge. This knowledge defines variability implementation and how it should be composed to derive products. If the activity of localizing and understanding the configuration knowledge requires a significant effort and costs to be performed, there is no payoff for the upfront investment made to build reusable software assets. Therefore, an easy-to-comprehend specification of all variability implementation and valid compositions is essential to the success of configurable product lines.

A particularity of enterprise product lines is the reuse of a large number of frameworks in concert for addressing a diversity of concerns (e.g., service composition and graphical user interfaces), mainly because they are ready-to-use infrastructures. As frameworks typically rely on a set of domain-specific abstractions, these become part of mindset of product line engineers along the software development. However, implementing variability as domain-specific abstractions involves making completion choices, some of which cut across heterogeneous languages (e.g., Java and XML). Since the completion code is not found in a single location, it tends to result in variability implementation spread across the product line architecture. As a consequence, it can lead to a negative impact on the activities of localizing and understanding the configuration knowledge specification.

A diversity of specification techniques were introduced as a way to facilitate the comprehension of the configuration knowledge [1, 2, 3]. Nevertheless, there are limited investigations about the implications of the adopted technique on the comprehensibility of framework-based software product lines. In fact, as far as we know, there is no study that analyzes the impact of specifying the configuration knowledge using the different available techniques, namely annotations, general-purpose modeling and domain-specific modeling. Existing research work [5] has analyzed and compared the individual techniques in terms of modularity, but they have not observed their level of impact on the configuration knowledge comprehensibility measures. In addition, their findings are not applicable to the context of enterprise product lines built with multiple frameworks, which is the scenario we are addressing.

We have performed an experimental study to evaluate and compare three tools that support the aforementioned tech-
niques: CIDE [3], which promotes colored-based annotations; pure::variants [1], which relies on general-purpose abstractions; and GenArch+ [2], which exploits domain-specific abstractions. In this study, six participants (individually organized) answered questionnaires about three framework-based product lines, each of which with the configuration knowledge modeled with the three different tools. The results indicate that:

- there is no relevant difference between the techniques when we consider the absolute number of correct answers. However, the time spent for answering them is significantly different;
- the use of domain-specific abstractions in the configuration knowledge specification, in fact, seems to facilitate the understanding of coarse-grained variability;
- because general-purpose modeling is abstract and hides many relevant details, it imposes certain restrictions to product line engineers to quickly localize and comprehend the configuration knowledge; and
- the correct comprehension is not associated with individual expertise about the frameworks that are part of the target product line.

The remainder of this paper is organized as follows. Section 2 describes the problem we are investigating and the evaluated approaches. Section 3 details how we have performed our experimental study, followed by Section 4, which presents results and associated discussion. Section 5 shows threats to the validity of our study. Finally, Section 6 concludes this paper.

2. CONFIGURATION KNOWLEDGE

The configuration knowledge expresses relations between product line variability and code assets, and their interactions. Several ways of specifying the configuration knowledge have been proposed [1, 2, 3]. This section overviews three different techniques selected for our study: annotation, general-purpose modeling, and domain-specific modeling.

All investigated techniques assume that product lines are structured as a set of features. A feature model is used to represent the hierarchical arrangement of the features, together with the set of constraints that restrict their composition. Variability implementation is made using code configuration, i.e., the selection of specific parts of the code assets determines which parts are variable. These presence conditions are evaluated, in general, with respect to a feature model configuration, which is a subset of the available features. Therefore, given a valid feature model configuration, the evaluation of a configuration knowledge yields the code assets needed to build the corresponding product. Next we briefly describe what distinguishes each technique; further details can be found in their respective references.

**Annotation.** In annotative techniques, the configuration knowledge is specified directly into code assets similarly to `#ifdef` and `#endif` statements. Product line engineers annotate code fragments realizing variability inside the original source code. The annotations determine the presence of code fragments as part of the derived product. CIDE [3] is a representative tool that implements this technique. The advantage of CIDE in relation to other annotative approaches is its capability of warning product line engineers about annotations that do not respect the language syntax and type system. CIDE also provides views and navigation functionalities to support detailed understanding of the configuration knowledge, as the ability of hiding specific variable parts.

**General-purpose Modeling.** This technique specifies the configuration knowledge in one or more general-purpose models. Restrictions are attached to model elements to define when an element must be included or excluded from the derived product. These types of restrictions define traces that connect features and configurations on code assets. Developers are also able to create requires and excludes constraints among model elements. Observe that these last types of constraints are difficult to be expressed through source code annotation. Pure::variants [1] is a commercial product line tool that represents this kind of technique. The configuration knowledge is specified in one or more object-oriented models, called family model. This tool also uses templates to support fine-grained variability.

**Domain-specific Modeling.** This technique allows product line engineers to use domain-specific abstractions in the specification of the configuration knowledge. As opposed to general-purpose techniques, this style of configuration knowledge specification does not allow product line engineers to freely create requires and excludes constraints among solution space elements, but these types of compositions must follow predefined specifications, for instance, the programming interfaces of frameworks. GenArch+ [2] offers an implementation of this technique. The configuration knowledge is specified through the use of three kinds of model: domain-specific knowledge models, implementation model, and configuration model. Domain-specific knowledge models are expressed in terms of domain-specific abstractions, and model their composition rules. The implementation model represents the basic code assets (classes, aspects, files, folder, etc.), and the configuration model captures the restrictions that constrain the inclusion of domain-specific and implementation model elements.

3. STUDY SETTINGS

The configuration knowledge is a key element for the success of configurable product lines and is essential for product line engineers to understand it with little effort. Therefore, our aim is to investigate whether the different techniques influence the correct comprehension of the configuration knowledge.

3.1 Experiment Hypotheses

The experiment aimed at verifying three inter-dependent hypotheses:

H1: The correct comprehension of the configuration knowledge depends on the different specification techniques.

H2: The time to correctly comprehend the configuration knowledge depends on the different specification techniques.

H3: The individual difference among the expertise of product line engineers do not impact on the correct comprehension of the configuration knowledge.

3.2 Selected Product Lines

As part of the experimental procedure, we have selected three different software product lines in order to perform our study, whose characteristics are summarized in Table 1. In this table, each of them is described by: its name and domain of application; the frameworks used to implement them; its size in terms of the number of features (mandatory, optional and alternative); and the most common type of
configuration granularity (coarse- or fine-grained). Coarse-grained variability is related to classes, components, aspects or self-contained domain-specific abstractions (e.g., beans, actions, agents and capability), and fine-grained variability is related to attributes, methods, code segments or domain-specific abstractions that are part of some other abstraction (e.g., bean variants, beliefs, goals and plans). Finally, we also indicate the techniques adopted to support variability within the software product line architecture.

These product lines were chosen for several reasons: (i) they were implemented by experienced developers, which adopted widely used software development practices, such as design patterns, and traditional architectures; (ii) they vary in size; (iii) they take advantage of several commonly used application frameworks; and (iv) they vary in granularity. The inclusion of different granularity was important to enable us to observe if and when there was any effect of using abstractions on the specification of configuration knowledge. Moreover, we have selected product lines developed in our laboratory, due to the availability of developers. They helped us to model the configuration knowledge of these product lines in each of the tools, in order to assure the correctness of the configuration knowledge, which is essential for our study.

### 3.3 Background of the Participants

This initial evaluation involved six post-graduate (MSc and PhD) students answering questions about the three previous presented software product lines. All participants have knowledge in software product line engineering and in the languages Java and XML but they were not familiar with the evaluated approaches. Therefore, they were given a short 60-minute demonstration of pure::variants, CIDet, and GenArch. In this training session, we demonstrated specific functionalities of the tools and examples of configuration knowledge specification. We used a different product line to avoid biasing the experiment results.

In addition to training the participants, we asked each one to fill in a background form after answering questionnaires. Our aim was to survey about the expertise of participants in the frameworks used to implement the product lines. The expertise is a value ranging from 1 to 5, where 1 means no expertise in a given framework and 5 means a high expertise. After that, we calculated the degree of expertise of each participant for each product line. The degree of expertise in a given product line is the average of the expertise of the participant in the frameworks used to implement that product line. Table 2 summarizes the background of the participants. Rows 4-10 in Table 2 indicate the expertise that participants claimed to have about each technology used to implement the different product lines. Most of them claimed to have little knowledge about the development of agent-oriented software systems with Jadex. Additionally, all the participants have not previously worked with service-oriented development using Spring Dynamic Modules (SpringDM). However, in general all participants have at least basic skills in the relevant frameworks of the experiment. Rows 12-14 in Table 2 indicate the resulting degree of expertise of the participants. Overall, most of them have satisfactory expertise in the studied product lines.

### 3.4 Experimental Design

With the aid of a training session the participants had to answer three questionnaires, one for each product line. The questionnaires are composed of ten questions involving the configuration knowledge. Examples of questions are: “Which abstraction(s)/code asset(s) is(are) related to the feature X?” and “How many abstraction(s)/code asset(s) is(are) mapped to the feature Y?” Two dimensions were evaluated in the experiment: (i) correctness; and (ii) time. Therefore, we have evaluated not only if the subjects were able to correctly comprehend the configuration knowledge but also how fast they got the information they needed.

We designed our experimental study with the Latin square in order to control the test of each tool with each participant. The Latin square design gave us a random allocation of the tools in such a way that each one is used once for each participant (row) and once for each product line (column). Therefore, the size of the Latin square is 3 x 3, in which the x-axis is the participants and the y-axis is the product lines. We have replicated the square once. Table 3 shows the configuration of the Latin square, presenting the allocation of participants, product lines and evaluated tools.
<table>
<thead>
<tr>
<th>Participants</th>
<th>E-Shop</th>
<th>OLIS</th>
<th>Buyer</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 and P4</td>
<td>G+</td>
<td>PV</td>
<td>C</td>
</tr>
<tr>
<td>P2 and P5</td>
<td>C</td>
<td>G+</td>
<td>PV</td>
</tr>
<tr>
<td>P3 and P6</td>
<td>PV</td>
<td>C</td>
<td>G+</td>
</tr>
</tbody>
</table>

Table 3: Latin Square Configuration

4. ANALYSIS AND DISCUSSION

This section presents the results of our experiment and discusses general observations. The analysis was decomposed into two categories regarding to: (i) correct answers and time; and (ii) expertise.

4.1 Correct Answers and Time Analysis

Figure 1 shows a chart that relates the number of correct answers and the evaluated techniques. Each bar in this chart indicates the number of questions correctly answered by the participants (y-axis) for each combination of product line and tool (x-axis). The contractions C, PV and G+ stand for CIDE, pure::variants and GenArch+ respectively.

Product Lines vs. Correct Answers. Overall, the Buyer agent presented a superior number of correct answers when compared with the OLIS and E-Shop. The Buyer agent is characterized to be a simple product line, which relies only on one framework. Therefore, it required small number of elements in the configuration knowledge specification. On the other hand, the OLIS and E-Shop are larger and more complex product lines when compared with the Buyer agent. The E-Shop, in particular, contains several fine-grained variability, and most of the features crosscut different code assets implementing the architectural layers of this product line. Therefore, the E-Shop configuration knowledge cannot be fully modularized only through the use of domain-specific abstractions. From this result, we concluded that the size (number of features and lines of code) and the number of fine-grained variability have negatively influenced the number of correct answers and consequently the comprehensibility of the configuration knowledge.

Techniques vs. Correct Answers. We observed that the correct comprehension of the configuration knowledge do not depend on the different specification natures of CIDE and pure::variants. For example, participants 2 and 5 achieved the lowest number of correct answers in the E-Shop product line. Based on this value we conclude that CIDE does not seem to contribute for the participants to correctly understand scattered configuration knowledge, as originally claimed by its authors [3]. On the other hand, the general-purpose modeling technique implemented by pure::variants seems to have helped participants 3 and 6 to better understand the E-Shop specification. The advantage of pure::variants is that it provides an overview of the coarse-grained variability through the family model and a detailed understanding of fine-grained variability via templates statements inside code assets. Pure::variants also provides searching mechanisms that were widely used by some participants. These mechanisms apparently helped participants to filter the configuration knowledge. However, the same result was not observed for the OLIS product line, as CIDE presented a higher number of correct answers than pure::variants for this product line. From these contrasting results, we conclude that there is no particularity in these tools that make them significantly better than other.

Nevertheless, we observed that, in general, the use of domain-specific abstractions provided the participants with the support to correctly understand the configuration knowledge. Participant 1 was the only exception, when pure::variants and CIDE were superior than GenArch+ in this aspect. To other participants, this tool presented intermediate results for the E-Shop and OLIS product lines and better results for the Buyer agent. Therefore, we conclude from this data that the hypothesis H1 holds. That is, product line engineers tend to better understand variability associated with abstractions in the presence of domain-specific modeling.

Approaches vs. Time. We also analyzed the time spent by the participants to localize and understand the configuration knowledge specifications. We observed that GenArch+ required the lowest time (3:40:45) for the participants to answer the questionnaires, followed by CIDE (4:19:56) and pure::variants (4:46:53). By analyzing only correct answers, we observed that GenArch+ was the tool that presented the lower average (0:02:57), and CIDE (0:03:10) and pure::variants (0:04:39) presented superior values.

Based on these results, we conclude that hypothesis H2 holds, but only with respect to some of the techniques. By comparing the time spent to answer questions correctly with GenArch+ and pure::variants, we observed that the use of the framework-provided abstractions improved the comprehension of the configuration knowledge specification, by allowing the participants localizing the configuration knowledge in a reduced time. However, there is no significant difference between GenArch+ and CIDE in terms of the time needed to localize and understand the configuration knowledge specification.
4.2 Expertise Analysis

Finally, we analyzed if the expertise of participant in the implementation frameworks was essential to correctly answer the questionnaires. Figure 2 shows the chart that relates the degree of expertise of each participant and his/her number of correct answers for each product line. Each bar in this chart indicates the expertise (x-axis) of participants (y-axis) about the product lines (bars). The bullets exhibit the total number of correct answers (CA/Total - secondary x-axis) of each participant. Note that there is no relation between the expertise and the number of correct answers. For example, participants 1, 2 and 3 claimed to have similar expertise; however, the participant 3 presented a superior number of correct answers. Correspondingly, participant 5, who has a limited expertise, presented a number of correct answers very close to the one achieved by participants 2 and 4, who claimed to have superior expertise.

We also compared the degree of expertise, number of correct answers and the product line tools. A high degree of expertise in the frameworks used to implement the OLIS product line combined with the annotative approach provided by the CIDE tool may have helped participants to correctly answer the questionnaire, however the same behavior was not observed in the other two product lines with the same tool. In contrast, the participants that use pure::variants to answer questions about the E-Shop product line presented a high number of correct answers despite they claimed to have a low degree of expertise. However, for the other two product lines the participants presented the same behavior described above, i.e. claimed to have a high degree of expertise but achieved a low number of correct answers. For GenArch+, we observed the same behavior described above: the expertise in the frameworks was not fundamental to correctly answer the questionnaire. As a result, we can conclude that there is no relation between the expertise and the number of correct answers, accepting the hypothesis H3.

5. THREATS TO VALIDITY

This section discusses the study constraints. For each category, we list possible threats and the procedure we took to alleviate their risk.

Conclusion Validity. The major external risk here is related to the engagement of the subjects to be part of the experiments, due to the length (time) of the questionnaires (almost two hours and thirty minutes for each participant). However, there was a rotation of the approaches order given that we adopted the Latin square. Another threat is the heterogeneity of participants. We have not taken any special care to select the participants and so they may represent random choices. Although the heterogeneity of subjects can also be considered a threat to the conclusion validity, it helps to promote the external validity of the study. Finally, the quality of the investigated tools is also a risk for the conclusion validity. However, we did not observe bugs that hampered the understanding of specifications or forced the participants to spend more time answering a question.

Construct Validity. We identified the following threats to the construct validity: confounding questions, and insufficient training session. To minimize these problems, we answered questions from participants as they were emerging. To avoid biasing the experiment results, we limited the explanations about tool functionalities to what was demonstrated during the training session and about the questions to what clarifications were absolutely necessary.

Internal Validity. Threats to internal validity reside on how we have specified the configuration knowledge of the product lines with different techniques. We ensured that each product line has been specified following the same pattern in all tools by triple-checking each specification and by using the product line developers to model the configuration knowledge. It is important to be checked because the number of traceability links may increase depending on how the configuration knowledge is specified. In fact, the size and complexity of the product lines were two factors that have influenced the results.

External Validity. The major external risk here is related to the product lines. The selected product lines might not be representative of all industrial practices. To reduce this risk, we selected three product lines from different domains, which are heavily based on industry-strength frameworks. Although the size of the chosen product lines is limited, this decision allowed us to obtain more consistent results that could be interpreted in this specific context. Nevertheless, additional replications and statistical tests are necessary to determine if our findings can be generalized to other domains.

6. CONCLUSION

In this paper we presented an experimental study that compares three different product line tools (pure::variants, CIDE and GenArch+). The focus of our assessment is on the comprehensibility of configuration knowledge of framework-based software product lines. The experiment was performed based on questionnaires applied to six participants, following a Latin square configuration. As a result, GenArch+ tends to better support participants in the task of localizing and correctly answering questions about configuration knowledge. Therefore, based on current results, we can conclude that the correct understanding of the configuration knowledge is considerable dependent on the different specification techniques. Moreover, general-purpose modeling techniques might make it more difficult to product line engineers to quickly localize and comprehend the configuration knowledge. Interestingly, the individual expertise in the frameworks is not associated with the correct comprehension of the configuration knowledge. As future work, we intend to replicate the Latin Square in order to check whether the results found in this work prevail.

7. REFERENCES