5
Evaluation

The purpose of this Chapter is to present the evaluation of our proposed technique for engineering software product lines. In particular, the evaluation focuses on studying two distinct points: configuration knowledge specification quality; and configuration knowledge comprehension.

First, this Section describes the setting and results of a study evaluating the specification quality (Section 5.2). The goal of this study is to quantify and analyze the impact of domain knowledge modeling languages on configuration knowledge modularity and complexity, when compared to other code-oriented techniques. Quantifying the specification cost in order to determine the feasibility of the proposed technique is also the subject of this evaluation.

This Section also presents the design of an empirical study and the analyse of its results (Section 5.3). This study compares the specifications in the styles of annotative and general-purpose modeling with the domain-specific modeling one - based on DKMLs. The main goal is to measure correctness and response time of questions that require a thorough understanding of the configuration knowledge specification.

5.1
Selected Product Lines

As part of the experimental procedure, we have selected three distinct software product lines in order to conduct our study. Table 5.1 summarizes their main characteristics. The following characteristics were considered: name and domain of application; used frameworks; size, in terms of the number of features (mandatory, optional and alternative); and feature granularity (coarse- or fine-grained). In this set of studies, coarse-grained features are related to self-contained domain-specific concepts (e.g., beans, actions, agents and capability), and fine-grained ones are related to classes, attributes, methods.

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 applica-

Name	Domain	Frameworks	Size	Variability Granularity
E-Shop	Online stores	SpringDM SpringMVC iBatis	18 features • 9 Mandatory • 7 Optional • 2 Alternative	Coarse-grained Fine-grained
OLIS	Personal assistance software	Struts 2.0 Spring Hibernate JADE Jadex	21 features • 13 Mandatory • 5 Optional • 3 Alternative	Coarse-grained
Buyer Agent	e-Commerce agents	Jadex	12 features • 4 Mandatory • 1 Optional • 7 Alternative	Fine-grained

Table 5.1: Main characteristics of the target product lines.

tion frameworks; and (iv) their features vary in granularity. The inclusion of different granularity is important to enable us to observe if and when there is any effect of using domain-specific concepts on the specification of the 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. This ensure the correctness of the configuration knowledge, which is essential for our study.

Description of Selected Product Lines

Buyer-SPL The Buyer-SPL implements a family of buyer agents for the domain of e-Marketplaces. This domain in general contains buyers and sellers that interact in such way that sellers sell products to buyers. In this product line, customized buyer agents represent users and act in accordance with their preferences. The buyer agents can be derived from the following features: (i) Payment Type – Credit Card, Pay Pal and Pay upon Pick up; (ii) Shipping Type – Ground Shipping and Pick up at Store; and (iii) Store Selection Strategy – Cheaper and Faster. Note that the Pay upon Pick up feature can only be selected if the Pick up at Store is selected. Its architecture is based on the Web-MAS architectural pattern (Nunes et al. 2008) and encompasses different services provided for users, including the buying service, which is responsible for the customization of the buyer agent.

OLIS-SPL OLIS is a product line of web systems that supports the derivation of products with a set of core user services and optional ones whose goal is to automate tasks using software agents. There are four main services that compose OLIS, which are: (i) User Management service - allows users to register themselves and configure their account; (ii) Event Announcement service

- allows users to announce events to other users through an event board; (iii) Calendar service - allows users to schedule events in their calendars. Besides the information of events published on the events board, each calendar event contains a list of participants. Additionally, announced events can be imported into the users' calendar; and (iv) Weather service - provides information about the current weather conditions and a given location forecast. Examples of service variability are: (i) Event Reminder - sends notifications to the user about events that are about to begin; (ii) Event Scheduler - checks the event participants' schedule to verify if a new event conflicts with other existing ones. In this case, the system suggests a new date for a calendar event that is appropriate to the participants' schedule; and (iii) Events Suggestion - automatically recommends events based on user preferences. OLIS allows deriving products that are able to deal with three different event types: generic, academic and travel events. Likewise the Buyer-SPL, is based on the Web-MAS architectural pattern.

e-Shop-SPL The E-Shop is a software product line of online stores. It provides typical features such as search Store Catalog, Payment, and Shipment. The features were devised into four main modules: (i) eshop - core implementation of the e-Shop-SPL; (ii) eshop.payment - modularizes a set of payment methods and resources related to the payment process; (iii) eshop.customer - modularizes the customer service; (iv) eshop.shipment - encompasses services to calculate taxes and resolve the shipment process. The e-Shop is based on the typical layered architecture pattern.

5.2 Assessment of Modularity and Complexity

5.2.1 Study Phases and Analysis Procedures

The goal our evaluation is: (i) to quantify and analyze the modularity and complexity of the configuration knowledge specification compared to other code-oriented techniques; and (ii) to quantify the configuration knowledge specification cost of our proposed technique, in order to invastigate its feasibility. The study was organized in the following major phases: (i) specification of the product lines in GenArch$^+$, pure::variants and CIDE tools; (ii) quantification of the modularity and complexity metrics (Section 5.2.2) over the different investigated techniques; and (iii) quantitative analysis of the obtained results for the different metrics (Section 5.2.3 and Section 5.2.4).

The analysis was divided into two stages. First, we compare GenArch$^+$ against CIDE and pure::variants techniques from the perspective of the modularity and complexity of the configuration knowledge (Section 5.2.3). Second, we measured the cost of specifying each investigated product line using DKMLs. This support us to check if this issue (specification cost) does not compromise the gains on using domain knowledge models (Section 5.2.4). The results obtained from this evaluation allowed us to make a qualitative discussion about configuration knowledge specification in the context of framework-based software product lines.

5.2.2 Quantitative Metrics Suite

In order to compare GenArch$^+$ and other existing code-oriented techniques (CIDE and pure::variants), we used a suite of metrics to quantify the separation of concerns and the complexity needed to prepare the assets (models, configuration files) associated with the configuration knowledge specification.

Separation of Concern metrics A suite of separation of concern metrics was selected to quantify the modularization of the configuration knowledge. We consider measuring modularization because code instantiating framework-provided concepts often crosscuts several source code elements as mentioned in Chapter 2. As programming languages are not able to isolate the source code instantiating concepts, it tends to lead to configuration knowledge with poor modularity. The separation of concerns metrics are also interesting to predict the effort of developers in localizing and understanding the configuration knowledge specification. Previous empirical studies have observed that these metrics are also a useful indicator of maintenance effort in a wide range of software engineering tasks (Conejero et al. , Figueiredo et al. 2009, Eaddy et al. 2008).

Our evaluation is based on the scattering and tangling attributes. We measured the scattering by counting the product line artifacts that contain at least one statement of configuration knowledge specification. We consider artifact as source code (e.g., Java classes and interfaces), configuration files (e.g., ADF files), and models (e.g,. agent-specific architecture model). The configuration knowledge statements are expressed in the techniques as: (i) mappings from features to product line assets (code, configuration files, model elements); and (ii) mechanisms used to configure the source code, such as template statements. Tangling was measured based on the Concern Diffusion over Lines of Code (CDLoC) metric (Garcia et al. 2005). The CDLoC counts the number of concern switches in the source code. In contrast, in our study, we count the number of configuration knowledge specification switches in the product line assets.

Effort estimation metrics In order to measure the complexity of the configuration knowledge specification we have adopted some size metrics. They count: (i) the number of elements in the configuration knowledge (NoECK); and (ii) number of features assignments (NoFA). We believe that a high number of elements and features assignments in the configuration knowledge might contribute to increase the effort (time) required for its specification.

Tools / Metrics		OLIS	Buyer	eShop
CIDE	Tang.	95	30	176
	Scatt.	156	16	52
pure::variants	Tang.	98	30	260
	Scatt.	15	4	41
GenArch	Tang.	0	0	0
	Scatt.	1	1	1

Table 5.2: Configuration Knowledge Modularization – separation of concerns metrics.

5.2.3 Results: Separation of Concerns

Table 5.2 summarizes the tangling and scattering metrics quantified in the product line assets of the investigated product lines, considering the GenArch$^+$, CIDE and pure::variant tools. As we can see, pure::variants presented the highest values (98/30/260) for the tangling metrics compared to the GenArch$^+$ and CIDE. In contrast, CIDE showed the highest values for the scattering metric. The reason for this is that CIDE requires introducing invasive annotations along all source code assigned to any feature. GenArch$^+$ and pure::variants presented low values for this metric because they specify the feature assignments in fewer artifacts. GenArch$^+$ adopts the configuration model and pure::variants uses the family models for this purpose. As a consequence, features assignments were found in GenArch$^+$ only in 1 places along all artifacts of the product lines. Therefore, we can conclude that the use of exclusive artifacts for representing the configuration knowledge, instead of only annotating the source code, tends to provide a better separation of the configuration knowledge concern.

Nevertheless, the use of templates by pure::variants technique to address fine-grained variability implied in some switches between configuration knowledge specification and source code not implementing features. For example, the OLIS-SPL Spring Application Context file (see Figure 2.11) contains some template statements that define which Beans is accountable to be part of derived products. This property affected (*increased*) the collected results for the separation of concern metrics. Without the use of templates, the results for scattering and tangling metrics is only 1, as the result achieved by GenArch$^+$. However, we believe that it has not compromised the achieved benefits by pure::variants. The difference from CIDE to pure::variants, for example, remains almost equals for the tangling metric. Based on these results, we can conclude that the model-based techniques offered by GenArch$^+$ and pure::variants tends to successfully prevent the scattering and tangling of the configuration knowledge specification over the framework-based source code. Therefore, even though they cannot replace cohesive modules, they exhibit characteristics that might facilitate the understanding of scattered feature implementations, thus contributing to the product line maintainability and evolvability.

5.2.4 Results: Size

The model-based techniques - pure::variants and GenArch$^+$ - are the solutions that better modularize the configuration knowledge specification. Nevertheless, as a drawback, they tend to increase the effort needed to specify the product line assets, as we could observe after the application of the size metrics. Table 5.3 summarizes the results for these metrics.

Tools / Metrics		OLIS	Buyer	eShop
CIDE	NoECK	245	53	501
	NoFA	245	53	501
pure::variants	NoECK	763	87	353
	NoFA	220	28	143
GenArch	NoECK	4119	1066	1859
	NoFA	153	20	138

Table 5.3: Configuration Knowledge Modularization – size metrics.

It is interesting to observe that the use of models substantially increases the number of elements in the configuration knowledge (NoECK). The annotation-based technique provided by CIDE requires the coloring of only 245 pieces of source code, this for specifying the configuration knowledge of OLIS-SPL; and 53 for Buyer-SPL. CIDE does not require the creation of any new asset. Therefore, as we will discuss further, this metric matches with the number of features assignments (NoFA) metric. On the other hand, pure::variants requires the creation of 763 model elements in the family model and the instrumentation of 49 source code elements with template statements, this for the specification of the OLIS-SPL. In the case of GenArch$^+$, we could observe that the use of domain knowledge models increases even more the NoECK metric. For example, for specifying the OLIS-SPL, GenArch$^+$ demands the creation of 4119 elements. For specifying the Buyer-SPL and eShop-SPL, a large number of elements in the domain knowledge models is also required, as shown in Table 5.3.

Even though GenArch$^+$ demands the creation of a large number of elements in the configuration knowledge of the product lines, the use of domain knowledge models helped to reduce the number of features assignments. In fact, GenArch$^+$ required only 51 and 20 assignments for specifying the OLIS and Buyer product lines, respectively. On the other hand, pure::variants for example, forced the creation of 220 feature assignments for specifying the OLIS-SPL, and 28 for the Buyer. This metric was measured by counting: (i) the number of mappings from features to solution space elements in the artifacts that specify the configuration knowledge; and (ii) the number of template statements inside code assets. The results demonstrated that GenArch$^+$ potentially contributes to reduce the replication in the configuration knowledge specification.

5.3 Empirical Evaluation

Our main goal is to investigate whether the different techniques influence the correct comprehension of the configuration knowledge. Similar to related efforts (Lange and Chaudron 2007, Cornelissen et al. 2011), two dimensions were evaluated in the empirical evaluation: (i) correctness; and (ii) time. That is, 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. Therefore, we distinguish the following research questions.

RQ1: Does the availability of domain-specific models increase the correct comprehension of the configuration knowledge?

RQ2: Does the availability of domain-specific models reduce the time that is needed to correctly comprehend the configuration knowledge?

RQ3: Does the individual differences among the expertise of product line engineers impact on the correct comprehension of the configuration knowledge?

RQ4: Which types of configuration knowledge comprehension task benefit most from the use of domain-specific and from other code-oriented techniques?

5.3.1 Experiment Hypotheses

Associated to the first two research questions are two null hypotheses:

$H1_0$: *The correct comprehension of the configuration knowledge does not depend on the different specification techniques.*

$H2_0$: *The time to correctly comprehend the configuration knowledge does not depend on the different specification techniques.*

The alternative hypotheses are the following:

$H1_1$: *The correct comprehension of the configuration knowledge depends on the different specification techniques.*

$H2_1$: *The time to correctly comprehend the configuration knowledge depends on the different specification techniques.*

5.3.2 Background of the Participants

The first controlled experiment involved (Cirilo et al. 2011b) six post-graduate (MSc and PhD) students from PUC-Rio answering questions about the three previous presented software product lines. The second one involved fifteen post-graduate students from two different institutions – PUC-Rio and UFRN. All participants have knowledge in software product line engineering and in the languages Java and XML, but they were not familiar with the evaluated techniques. Therefore, they were given a short demonstration of pure::variants, CIDE, 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 of the first controlled experiment to fill in a background form after answering questionnaires. Our aim was to survey about the expertise of them in the frameworks

	Participant					
	P1	P2	P3	P4	P5	P6
Framework						
Spring	4	4	4	2	1	3
Struts	2	2	1	4	2	2
Spring MVC	4	4	4	5	3	5
Hibernate	1	1	1	1	4	1
iBatis	2	2	2	3	1	3
Spring DM	1	1	1	1	1	1
Jadex	2	1	1	3	1	1
Product Line						
eShop	1.5	1.25	1	2.25	2	1
OLIS	3	3	2.75	3.5	1.75	3.25
Buyer Agent	2	1	1	3	1	1

Table 5.4: Degree of Expertise of the participants

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 5.4 summarizes the participant's background. Rows 4-10 in the Table indicate the expertise that participants claimed to have about each technology used to implement the different product lines. Rows 12-14 indicate the resulting degree of expertise of the participants.

Most of participants of the first experiment 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. Overall, most of them have satisfactory expertise in the studied product lines.

5.3.3 Experimental Design

With the aid of a training section the participants had to answer three questionnaires, one for each product line. The questionnaires in the first experiment were composed of ten questions. Examples of questions are: "Which *concepts(s)/code asset(s)* is(are) related to the feature X?;" and "How many *concepts(s)/code asset(s)* is(are) mapped to the feature Y?". In the second evaluation the questionnaires were composed of nine questions. We devised the questionnaires into four typical comprehension tasks. Table 5.5 provides descriptions of the tasks and shows how each of the nine type of questions is covered by at least one activity. To render the questions more representative of real situations, they are open rather than multiple-choices, making it harder to subjects to resorting to guesting.

T1: Identifying all files in which source code of a feature occurs.
Q1. What file(s) in the source code the feature X were implemented? Q3. Which file(s) and respective source code fragment(s) are implementing the Feature X?
T2: Identifying all features that occur in a certain file.
Q2. Which other Feature(s) occurs in the file X? Q6. Which other Feature(s) occurs in the file(s) that implements the Feature X?
T3: Identifying all framework-concept instances that are implementing a certain feature.
Q4/Q8. Which framework-concepts are implementing the Feature X? Q5. What file(s) in the source code the framework-concept(s) instance(s) that implement the Feature X was/were implemented?
T4: Identifying all framework-concept instances that interact with other ones.
Q7. You have identified the framework-concept instances that implement the Feature X. For each one, which other framework-concept instances are interacting with them? Q9. Which framework-concept instances are interacting with the ones that implement the Feature X?

Table 5.5: Description of the second comprehension questionnaire.

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 in the first evaluation. In the second evaluation we replicated the square five times. Table 5.6 shows the configuration of the Latin square, presenting the allocation of participants, product lines and evaluated tools. This design is important to

Participants	E-Shop	OLIS	Buyer
P1 ... n	G+	PV	C
P2 ... n	C	G+	PV
P3 ... n	PV	C	G+

Table 5.6: Latin Square configuration

avoid some effects such as learning.

5.3.4 Variables and Analysis

The independent variable in our experiment is the availability of each product line tool. The first dependent variable is the time spent on each question. Since going back to earlier tasks is not allowed and the sessions are supervised, we just had to write down the time spent during each question. The second dependent variable is the correctness of each question.

To test our hypotheses, we first test whether the sample distribution is normal (Shapiro-Wilk) and have equal variance (Levene). If these tests pass, we use the ANOVA test to evaluate our hypotheses, that is, whether there is any evidence that the means of the populations differ; otherwise, we use the nonparametric Kruskal-Wallis test. If these tests lead to a conclusion that there is evidence that the group means differ, we then are interested in investigating which of the means are different. To this end, we use the Tukey multiple comparison test in the case of the sample distribution is normal and have equal variance. This test compares the difference between each pair of means with appropriate adjustment for the multiple testing. Otherwise, we use the Nemenyi-Damico-Wolfe-Dunn test. For the correctness and time variable, we maintain a typical confidence level of 95 percent (alfa = 0.05).

5.3.5 Fist Controlled Experiment: Results, Analysis and Discussion

In the first controlled experiment (Cirilo et al. 2011b) we resort to answer the RQ1, RQ2 and RQ3. This section presents the results and discusses some general observations. The analysis was decomposed into two categories regarding to: (i) correct answers and time; and (ii) expertise.

Hypotheses Testing

We start off by testing null hypothesis $H1_0$, which states that the correct comprehension of the configuration knowledge does not depend on the different specification techniques. Figure 5.1 shows a box plot for the scores that were obtained by the subjects. Note that we consider overall scores rather than scores per questions. The box plot shows that there is no explicit difference in terms of correctness. The Shapiro-Wilk test did not succeed for the answers data, which means that ANOVA cannot be used to test $H1_0$. Consequently, we used the nonparametric Kruskal-Wallis test. As a chi-squared = 0.0363, degree-of-freedom = 2, and a p-value = 0.982, the Kruskal-Wallis indicates that there

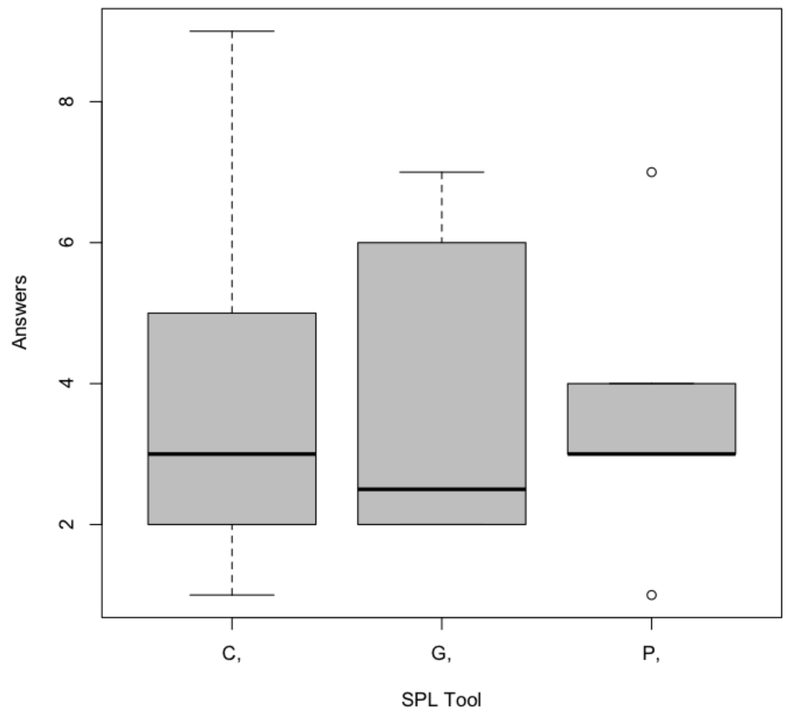


Figure 5.1: Box plot for correctness.

is no statistically significant difference among the investigated techniques in terms of correctness, meaning that $H1_0$ cannot be rejected in favor of our alternative hypothesis $H1_1$, stating that the correct comprehension of the configuration knowledge depends on the different specification techniques.

We next test the null hypothesis $H2_0$, which states that the time to correctly comprehend the configuration knowledge does depend on the different specification techniques. Figure 5.2 shows a box plot for the total time that the subjects spent on the questions. Differently from the correctness data, the requirements for the use of the ANOVA were met. Table 5.7, therefore, shows the results for ANOVA. The average time spent by the GenArch$^+$ was not clearly lower and the p-value $= 0.1325$ is bigger than 0.05, which means that $H2_0$ cannot be rejected in favor of the alternative hypothesis $H2_1$, stating that the time to correctly comprehend the configuration knowledge depends on the

Source	SS	DF	F	Sig.
Replica	142	141.5	0.0165	0.9011
SPL	1710	854.9	0.0994	0.9065
Technique	45246	22622.9	2.6298	0.1325
Replica:Subject	21864	5466.0	0.6354	0.6516
Residuals	68819	8602.4		

Table 5.7: ANOVA test for participants time.

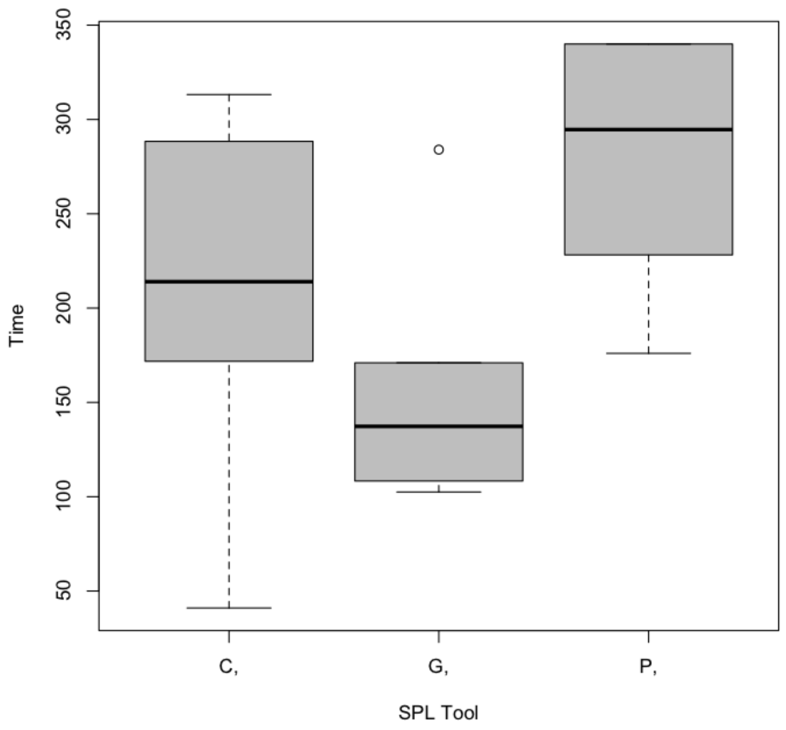


Figure 5.2: Box plot for time.

different specification techniques.

Analysis: Correct Answers, Time and Expertise

Although the results do not allow us to clearly confirm or reject the research hypotheses, next we discuss some implications that could be observed.

Product Lines vs. Correct Answers and Time Figure 5.3 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. 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 concepts. 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 time, consequently, in general negatively influenced the comprehensibility of the configuration knowledge.

Techniques vs. Correct Answers We observed that the correct comprehension of the configuration knowledge does 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 (Kästner et al. 2008). 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, the study indicates 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 observe from this data that alternative hypothesis $H1_1$ might hold. That is, product line engineers tend to better understand variability associated with concept instances 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

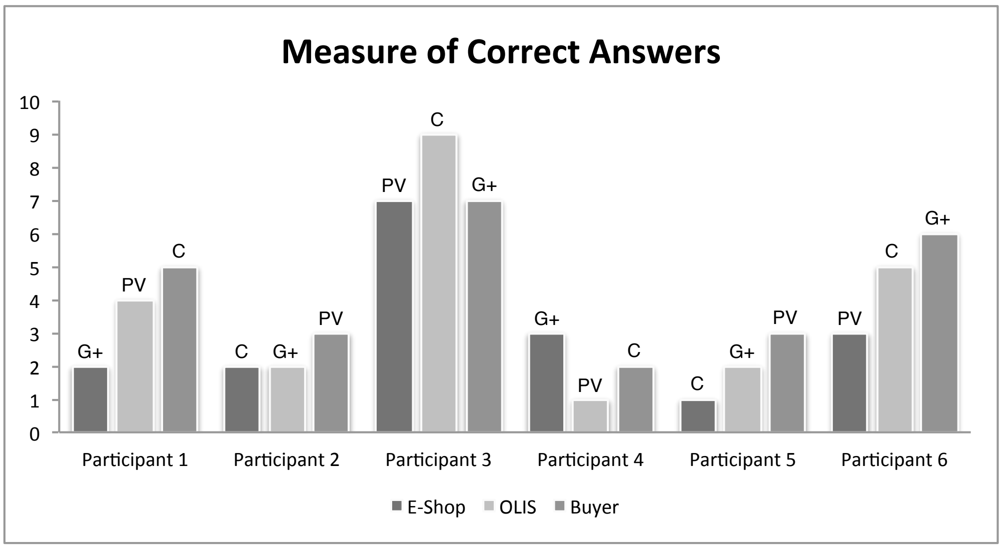


Figure 5.3: Measure of Correct Answers

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 also observe that hypothesis H2 might 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, it seems that there is no significant difference between GenArch$^+$ and CIDE in terms of the time needed to localize and understand the configuration knowledge specification.

Expertise Finally, we analyzed if the expertise of participant in the implementation frameworks was essential to correctly answer the questionnaires. Figure 5.4 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.

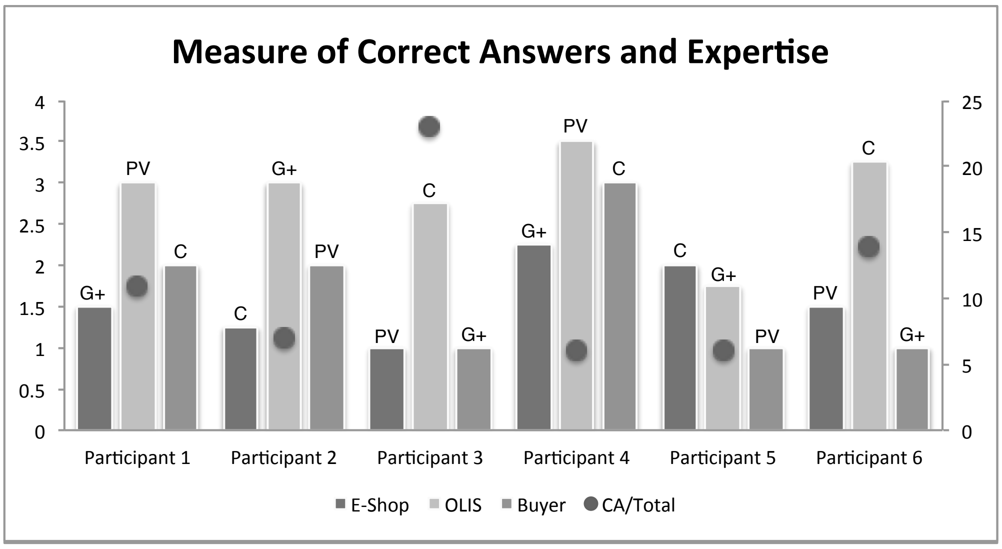


Figure 5.4: Measure of Correct Answers and Expertise

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 could observe that there is no correlation between the expertise and the number of correct answers.

5.3.6 Second Controlled Experiment: Analysis and Discussion

Now we present the results of our second experiment and discusses the observed results. In the same way, the analysis was decomposed into two categories regarding to: (i) correct answers and time; and (ii) expertise. Moreover, in this Section we also present a discussion about which types of comprehension tasks benefit most from the use of the different styles of configuration knowledge specification

Hypotheses Testing

In the same way, in the second experiment we start off by testing null hypothesis $H1_0$. Figure 5.5 shows a box plot for the scores that were obtained by the subjects (here we are also considering overall scores). Different from the first experiment, now the box plot shows that there is explicit difference in terms of correctness. Table 5.8 shows descriptive statistics of the measurements. The solution given by GenArch$^+$ subjects were 26.28% and 34.53% percent more accurate, averaging 5.82 out of 1.32 points compared to 3.81 points for the pure::variants and 4.29 points for the CIDE, respectively.

The Shapiro-Wilk and Levene tests succeeded for the answers data, which means that ANOVA may be used to test $H1_0$. As shown in Table 5.9, the ANOVA yields a statically significant result, that is, a p-value near to zero (p-value = 1.099e-06) indicates a statistically significant difference in the correctness measures associated with the three techniques. As the ANOVA test leads to a conclusion that there is evidence that the group means differ, we pass to investigate which of the means are different. To this end we use the post-hoc Tukey's HSD multiple comparison test. Table 5.10 shows the result for each pair as a p-value. The Tukey's HSD tests showed that GenArch$^+$ subjects had significantly higher scores than the other two groups at the 0.05 level of significance. Observe that other comparison (pure::variants and CIDE) was not significant. Therefore, we can reject $H1_0$ in favor of our alternative

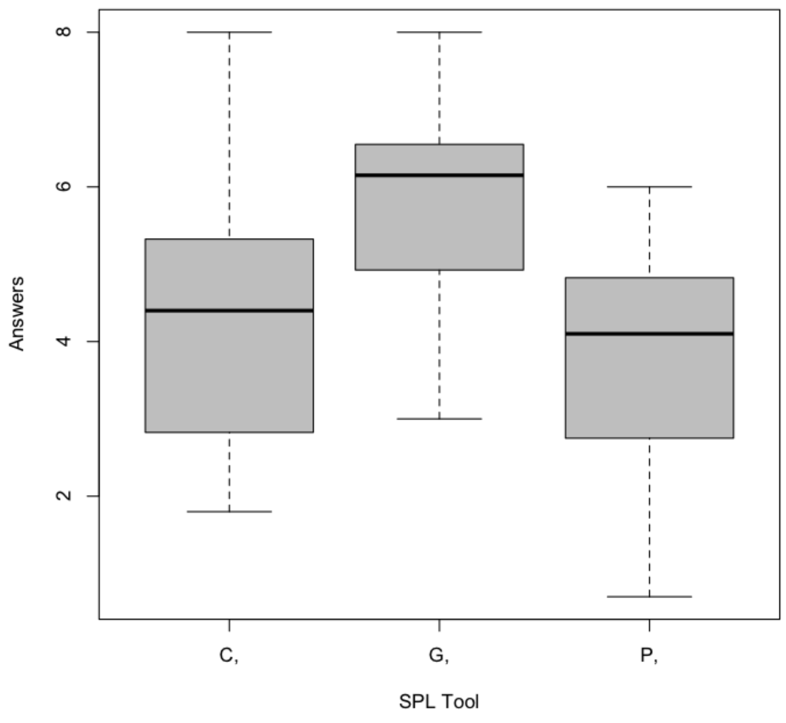


Figure 5.5: Box plot for correctness.

	Correctness			Time		
	G+	C	PV	G+	C	PV
mean	5.823333	4.296667	3.816667	737.978	915.768	1976.169
difference		-26.28%	-34.53%		+19.41%	+62.65%
min	3	1.8	0.7	239.24	296	454.14
max	8	8	6	1872.71	2345.45	10978.33
median	6.15	4.4	4.1	622.46	802.47	1234.17
stdev.	1.329724	1.814079	1.443293	449.202	567.3122	2687.005

Table 5.8: Descriptive statistics of the experimental results.

Source	SS	DF	F	Sig.
Replica	25.569	4	9.4522	7.428e-05
SPL	24.808	2	18.3421	1.075e-05
Technique	32.939	2	24.3539	1.099e-06
Replica:Subject	32.031	10	4.7364	0.000676
Residuals	17.583	26		

Table 5.9: ANOVA test for participants answers.

hypothesis indicating that there is a difference between the comprehension correctness at the 0.05 level of significance.

We next test null hypothesis $H2_0$. Figure 5.6 shows a box plot for the total time that the subjects spent on the questions. Table 5.8 indicates that, on average, the GenArch$^+$ required 19.41% percent less time than pure::variants and 62.65% less than CIDE. Since the Shapiro-Wilk test indicated deviations from normality, the Kruskal-Wallis test and Nemenyi-Damico-Wolfe-Dunn test were applied. While the Nemenyi-Damico-Wolfe-Dunn test allowed us to realize a pair wise comparison of the distributions, Kruskal-Wallis test allowed checking if there exist significant differences among the three techniques under investigation. As a chi-squared = 4.0495, a degree-of-freedom = 2 and a p-value = 0.1320, the Kruskal-Wallis indicates that there is no statistically significant difference in the effort need to correctly comprehend the configuration associated with the three techniques. Consequently, our initial intuition that the domain-specific based technique reduce the comprehension effort was not totally confirmed.

Comparisons	Diff	Lwr	Upr	Sig.
G+ - C	1.526667	0.7805027	2.2728306	0.0000777
PV - C	-0.480000	-1.2261640	0.2661640	0.2641892
PV - G+	-2.006667	-2.7528306	-1.2605027	0.0000013

Table 5.10: Tukey's HSD multiple comparison test results for answers.

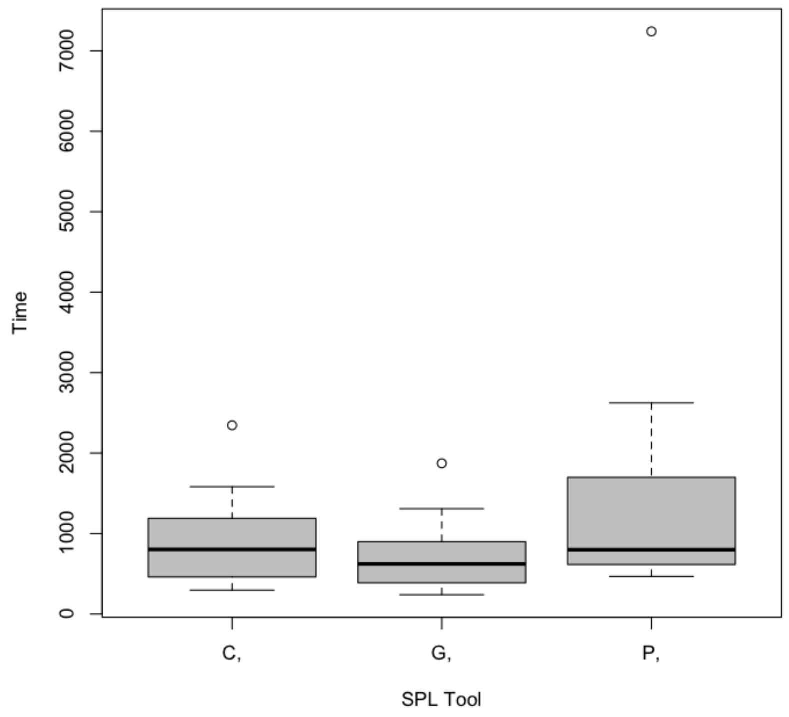


Figure 5.6: Box plot for time.

Analysis: Correct Answers and Time

Reasons for Correctness Differences. We attribute the added value of GenArch$^+$ domain-specific models to correctness to several factors. A first one is the appropriate abstraction level of such models. The fact that GenArch$^+$ shows domain-specific concepts without source-code details and in a modular way makes concept instances existence and assignment to features easier to understand. Section 3.3 discusses this point in more detail. In contrast, CIDE and pure::variants partially or totally mix this information with the source-code, which impose to the developers an extra mental effort to properly comprehend the configuration knowledge associated with framework-specific concepts.

Second, the results of the qualitative questionnaire show that GenArch$^+$ subjects liked tool features quite often: the subject estimate the facility to answer questions in an average of 2.54 points out of 1.06 points. The subject estimate the facility to answer questions using CIDE in an average of 2.7 points out of 1.19 points, and using pure::variants in an 3.21 average of, out of 1.09 points. Our observation shown that GenArch$^+$ features (see Section 4) was used frequently and that tool was actually found useful most of questions. This is a indication that the GenArch$^+$ subject generally did not experience a resistance to using it and were quite successful in their attempts. This observation is confirmed when we correlate the results of number of correct answers and time. We noted that in some cases the participants faced resistance to answer some questions using CIDE and pure::variants. In these cases, as they are not been successful in their attempts, they preferred to spent just few minutes trying to answer the request question, rather than keeping trying.

Reasons for Different Time. The lower time achieved by GenArch$^+$ subjects can be attributed to several factors. First of all, all-information required for correctly understanding the configuration knowledge is shown on domain-specific models, which eliminates the need for interpreting the source code and navigating through different files. In particular, the overview of the entire product line as domain-specific concepts saves time in comparison to conventional specifications in which multiple files typically have to be studied at once. It is also the case why CIDE subjects achieved a great time. Since this tool provides modular visualizations on feature and their interactions, it alleviates from the subject the need to navigate thought the entire product line source-code. However they still having to reasoning about feature assignment without an appropriated representation of the existing concept instances. Then, second, the need to imagine how certain features were assigned to concept instances or their interaction represents a substantial cognitive load on the part of the subjects. This is also alleviated by domain-specific models which show the actual concepts abstracted from the source-code. Examples of these assumptions were discussed in Chapter 2.

On the other hand, several factors may have had a negative impact on the time achieved by GenArch$^+$ subjects. For example, the fact the GenArch$^+$ represents the configuration knowledge on many models means that context switching is necessary, which may yield a certain amount of overhead on the part of the subjects. Moreover, although the activity of navigating from models to source code is easy with GenArch$^+$ (see Section 3.2), the subjects can get lost when many editors are opened. It means that the subject needs to choose the right editors, which often is not a direct decision. This last issue could be solved by keeping few editors opened. However, it should be noted that GenArch$^+$ would still require a substantial amount of models to represent the configuration knowledge effectively. We also observed the negative impact of the number of models on the time achieved by pure::variants subjects. But in this tool the things get worse because pure::variants does not support direct navigability from models to source code. The subjects had to resort on extra tools, like eclipse search mechanism to overcome this misconnection. Moreover, the use of general-purpose concepts require from subject a extra cognitive load.

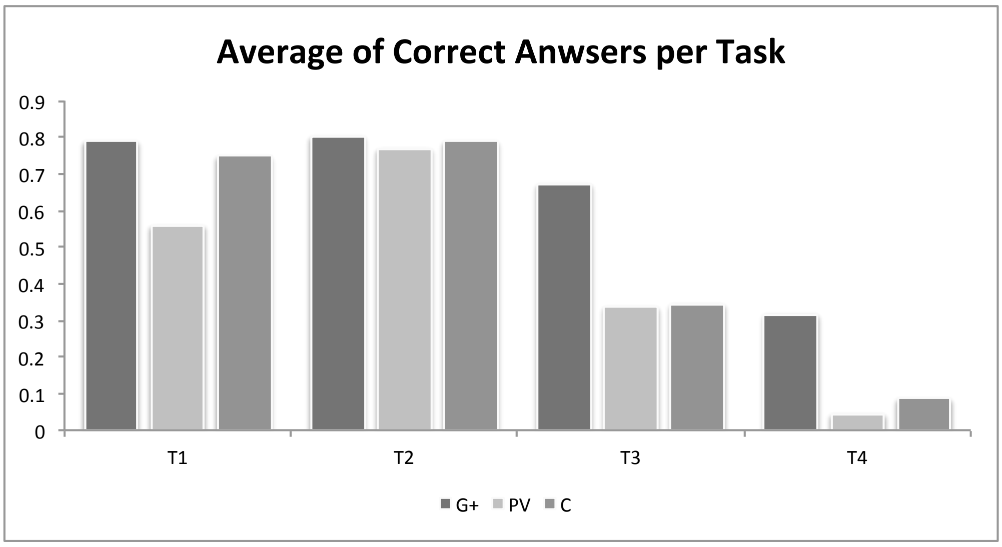


Figure 5.7: Measure of correct answers per tasks.

Individual Task Performance. Now we analyse which types of comprehension tasks benefit most from the use of the different styles of configuration knowledge specification.

Figure 5.7 and Figure 5.8 show the average scores and time spent by the participants in each product line tool from a task perspective. Figure 5.9 and Figure 5.10 show four box plots for the facility to answer questions in each tool estimated by the participants. The box plots are also separated from a task perspective.

	GenArch		CIDE		pure::variants	
	Correct.	Time	Correct.	Time	Correct.	Time
mean	0.78	420.42	0.74	397.49	0.55	1002.44
stdev	0.33	380.15	0.41	424.69	0.30	2594.13

Table 5.11: Descriptive statistics (Answers and Time) - Task 1

Identifying all files in which source code of a feature occurs. The goal of the first task was to identify and globally understand all files in which source code of a feature occurs. The performance difference here was quite subtle between GenArch$^+$ and CIDE groups, with the GenArch$^+$ apparently having had a very small advantage (see also Table 5.11). In terms of time, the difference between GenArch$^+$ and CIDE groups was also unnoticeable. However, it demanded the pure::variants group a huge among of time to complete this task. The CIDE group typically studied the *Interaction View* and/or used the *Views on Feature*. The GenArch$^+$ users mostly filtered the models content and studied the mappings to source code. Although, more laborious, this last solution proved effective and led to slightly more accurate

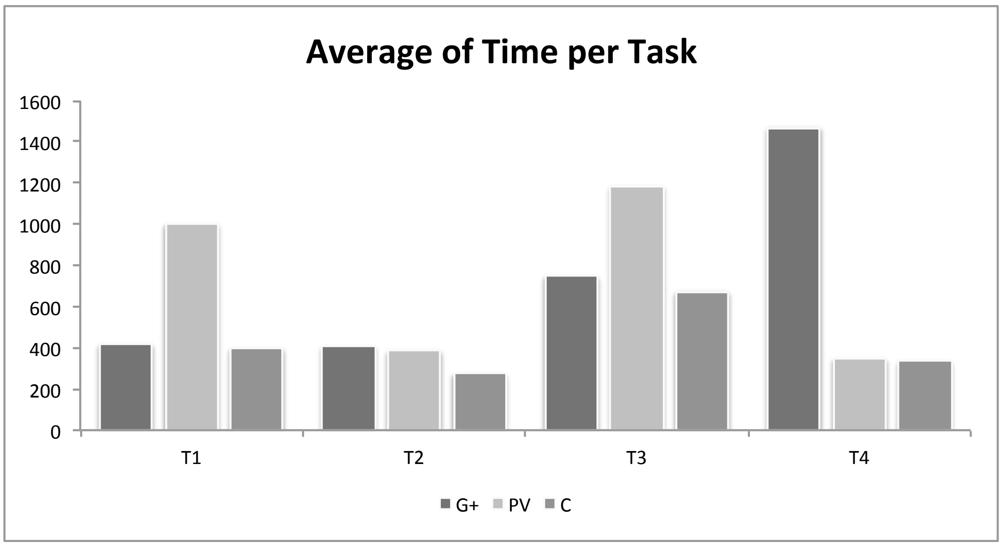


Figure 5.8: Measure of time per tasks.

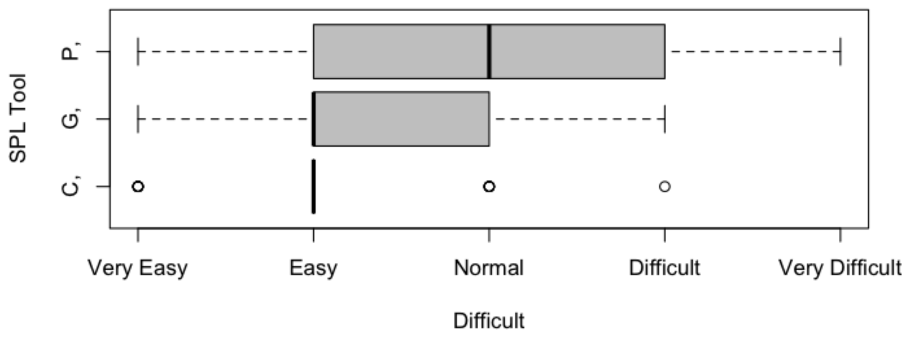
solutions. On the other hand, the pure::variants users in most of the time searched the source code and models for feature occurrence, which is fairly accurate once some results can be missed. To perform this task participants using CIDE and GenArch$^+$ liked tool features quite often more than the participants using pure::variants.

	GenArch		CIDE		pure::variants	
	Correct.	Time	Correct.	Time	Correct.	Time
mean	0.80	408.59	0.76	388.60	0.78	274.94
stdev	0.32	829.35	0.37	414.47	0.23	186.77

Table 5.12: Descriptive statistics (Answers and Time) - Task 2

Identifying all features that occur in a certain file. This task was similar to the previous one, but the focus was more on coupling, that is, identifying all features that occur in a certain file. All the subjects, either using GenArch$^+$, CIDE, or pure::variants scored equally well on this task and required similar amounts of time (see Table 5.12). The group using GenArch$^+$ resorted to the *Visualization on Source code* functionality to look for features assignment, while the CIDE group mostly observed the background color. The pure::variants group just have to studied the template statements referring to features. The fact that the information about features is directly exhibited on the source code account for the good effectiveness and accuracy. Moreover, the participants liked tool features equally.

Identifying all framework-concept instances that are implementing a certain feature. This task concerned a domain-specific analysis that turned out to be significantly easier for the GenArch$^+$ group (see



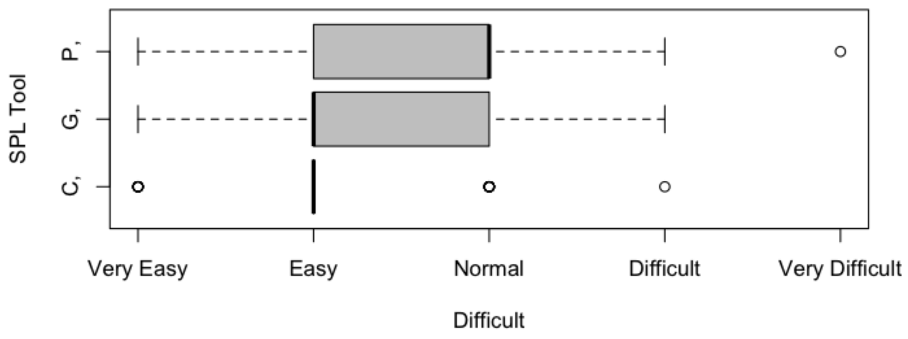


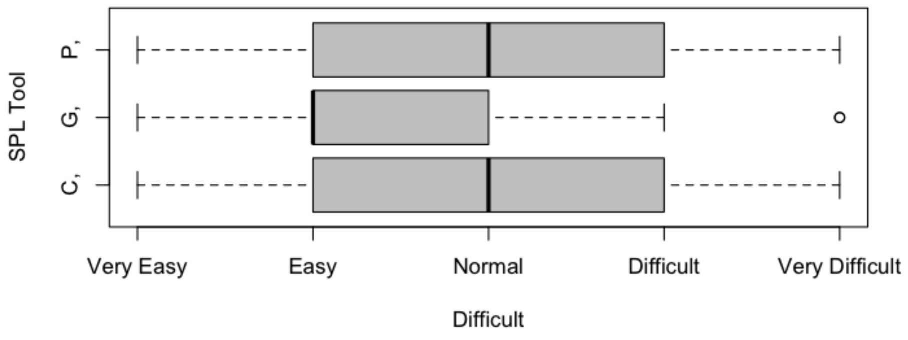
Figure 5.9: Box plot for difficult - Tasks T1 and T2.

Table 5.13). This is presumably explained by GenArch$^+$ domain knowledge models, from which all features assignments can be directly interpreted as domain-specific concepts. The other groups (CIDE and pure::variants) carried out a mental study of the source code, which is time-consuming and does not necessarily yield good results, specially when the framework-concepts are not well-known. Observe that in this case participants liked GenArch$^+$ features visibly more than they liked CIDE and pure::variants features.

	GenArch		CIDE		pure::variants	
	Correct.	Time	Correct.	Time	Correct.	Time
mean	0.67	749.34	0.33	1183.69	0.34	672.74
stdev	0.32	862.47	0.33	1674.59	0.33	1246.77

Table 5.13: Descriptive statistics (Answers and Time) - Task 3

Investigating dependencies between framework-concept instances. This task posed the challenging questions of which framework-provided concepts interact with each others. We can observe that GenArch$^+$ group completely out-performed in terms of correctness (see Table 5.14). An important reason might be that CIDE and pure::variants groups did not know exactly what to look for in the configuration knowledge because the



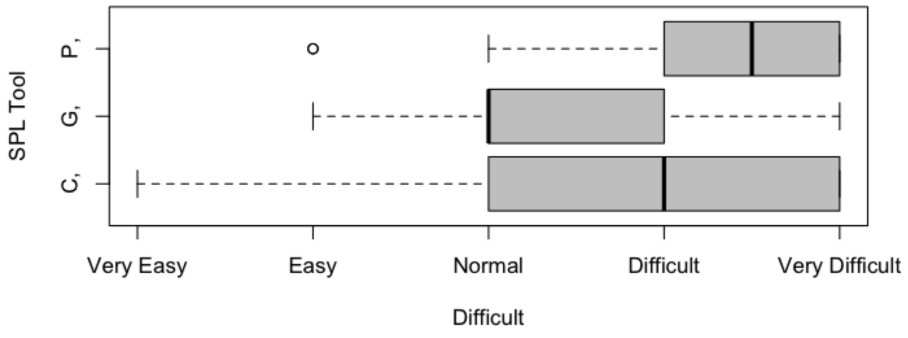


Figure 5.10: Box plot for difficult - Tasks T3 and T4.

dependencies between framework-concepts are rather obfuscated in the sense that no clear specification was given by such tools. Interestingly, even using GenArch$^+$ only a handful of subjects properly discovered the dependencies between framework-concept instances. However, the GenArch$^+$ group rated the difficulty of this task between "Normal" and "Difficult", whereas CIDE and pure::variants users rated the difficulty towards to "Difficult" and "Very Difficult".

	GenArch		CIDE		pure::variants	
	Correct.	Time	Correct.	Time	Correct.	Time
mean	0.31	1469.9	0.04	349.2	0.08	335.76
stdev	0.31	2187.17	0.11	824.46	0.23	1329.02

Table 5.14: Descriptive statistics (Answers and Time) - Task 4

5.4 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 patterns 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 are necessary to determine if our findings can be generalized to other domains.

5.5 Summary

In this chapter we have presented an evaluation of three different configuration knowledge specification techniques regarding some criteria: modularity, complexity, comprehensibility and effort required to correct comprehend product line. We provided evidence that the use of domain knowledge models reduces the number of features assignments but significantly increases the complexity of the configuration knowledge specification. We also showed that the use of domain knowledge models does not require less time to comprehend a product line, when compared to CIDE and pure::variants. Nevertheless, the results clearly illustrate domain-knowledge models usefulness for configuration knowledge comprehension. Our proposed technique added value was statistically significant, with the GenArch$^+$ group scoring more points on average.