Extending FragOP Domain Reusable Components to Support Product Customization in the Context of Software Product Lines

Daniel Correa[1(✉)], Raúl Mazo[2,3], and Gloria Lucia Giraldo[1]

[1] Universidad Nacional de Colombia, Medellín, Colombia
{dcorreab, glgiraldog}@unal.edu.co
[2] Université Panthéon Sorbonne - CRI, Paris, France
raul.mazo@univ-parisl.fr
[3] Universidad Eafit, GiDITIC, Medellín, Colombia

Abstract. Software product lines (SPL) have become an efficient paradigm for systematic reuse. SPL engineering is about the planned reuse of common assets for the rapid production of a software systems family. In SPL, an effective product derivation process is key to ensure that the effort required to develop the common assets will be lower than the benefits achieved through their use. While several approaches and tools are available on SPL engineering activities such as, variability management, component assembling, and product testing; most of the existing approaches do not present detailed information on the strategies for product customization (which affects the product derivation effectiveness). In a previous work, we introduced fragment-oriented programming (FragOP), which is a framework used to design, implement, and reuse domain components. In this paper, we enhanced the FragOP approach through the use of customization points and customization files to support the product customization activity. In order to gain preliminary insights into how VariaMos (the tool in which the approach is implemented) supports the FragOP approach, we designed a usability test by following the ISO/IEC 25062:2006 Common Industry Format for usability tests. Eight graduate students from the Universidad Nacional de Colombia took part and were asked to carry out a series of modifications to an e-commerce SPL. The usability test reported high subject performance results; however, we found some usability flaws that should be addressed.

Keywords: Software product lines · Usability tool test · Fragment-oriented programming · Product customization

1 Introduction

A software product line (SPL) is a collection of software systems that satisfy the specific needs of a particular market segment, and that are developed from a common set of core assets in a prescribed way [1]. Many software and systems product line (SPL) implementation approaches, such as CIDE, DeltaJ, Munge, Antenna, AspectJ, and AHEAD emerged during recent years [2]. These approaches focus on an effective

X. Peng et al. (Eds.): ICSR 2019, LNCS 11602, pp. 17–33, 2019.
https://doi.org/10.1007/978-3-030-22888-0_2

SPL component assembling (constructing and assembling a software product from the reusable SPL assets). However, existing approaches do not present detailed information on the strategies for product customization [3]. For example, de Souza *et al.*, [3] reported that much of the resources and effort of the product derivation process is spent on product customization. They analyzed some companies in which between 10% and 30% of each product instantiated from their platforms needs to be customized. Even, Montavillo *et al.*, [4] which developed a visualization tool to estimate the customization effort, deduced that less than 50% of an SPL example product code was reused as-is from the core-assets. The main difference between a component assembling and a product customization is that the first one reuses and assembles pre-developed core-assets to generate a new product. The second one is carried-out after the component assembling and it is commonly done manually, because each product customization is unique, so there are not pre-developed customized core-assets.

In previous work, we developed an SPL implementation approach called Fragment-oriented programming (FragOP) [5]. FragOP is a framework used to design, implement and reuse domain components in the context of an SPL. FragOP is a mix between SPL compositional and annotative approaches. In the original formulation of FragOP [5], it consisted on the definition of (i) domain components, (ii) fragmentations points, which are annotations over the domain components code; and (iii) fragments, a new type of file which alters the domain components code. In this paper, this approach is enhanced to support the SPL product customization. This enhancement was included as a new capability of the VariaMos tool [6], which is a software tool that supports the FragOP approach. Therefore, to gain insights into how VariaMos supports the enhanced FragOP approach, we decided to develop a usability test. Usability is defined by the International Standard Organization [7] as "the extent to which a product can be used by specified users to achieve specific goals with effectiveness, efficiency, and satisfaction in a specified context of use". That means that if a product (a software tool in our case) does not provide effectiveness, efficiency, and satisfaction to its users, it is not usable, and therefore will probably not be used. The rest of this paper is structured as follows. In Sect. 2, we present the FragOP approach including its enhanced metamodel and process; therefore, we present the enhanced VariaMos tool. In Sect. 3, we discuss the FragOP main two capabilities (assembling and customization) with a real SPL example. In Sect. 4, we present a usability test of VariaMos. In Sect. 5, we discuss the related work and finally Sect. 6 summarizes the contributions and presents future research directions.

2 Fragment-Oriented Programming

In this section, we recall the notion of FragOP as described in [5] and present the FragOP enhancement and its implementation in VariaMos. Fragment-oriented programming (FragOP) is a framework used to design, implement and reuse domain components in the context of an SPL. FragOP is based on the definition of six fundamental elements: (i) domain components, (ii) domain files, (iii) fragmentations points, (iv) fragments, (v) customization points, and (vi) customization files. The fragments act as composable units (compositional approach) and the fragmentation

points act as annotations (annotative approach). This mix of compositional and annotative approaches allows the FragOP to support multiple assets implemented over different languages, such as PHP, Java, JSP, CSS, HTML, and JavaScript, among others. The role of each FragOP element, their relationships, how are made up, and the information they store, it can be seen in the FragOP metamodel (see Fig. 1). Following, we present an overview of the FragOP metamodel elements:

SPL represents the software product line and contains an ID that represents the name of the corresponding SPL. **Domain requirements** represent SPL domain requirements (such as features and goals). **Domain components** represent SPL reusable domain components and contain an ID that represents a folder in which the component is stored. A **domain file** is a basic element of which most software components are made up; for instance, HTML, CSS, JavaScript, Java, and JSP files. A **fragment** is a special type of file which alters the application code. A **fragmentation point** is an annotation (a very simple mark) that specifies a "point" in which a domain file can be altered. A **customization file** is a file which specifies the domain files (for the current domain component) that should be customized. **Customization points** are annotations (very simple marks) that specify the "points" in which a domain file should be customized. **Product** represents a folder in which a new SPL product is derived. **Application files** are copies of domain files which are generated when a new product is derived. These files can be also modified by the fragments.

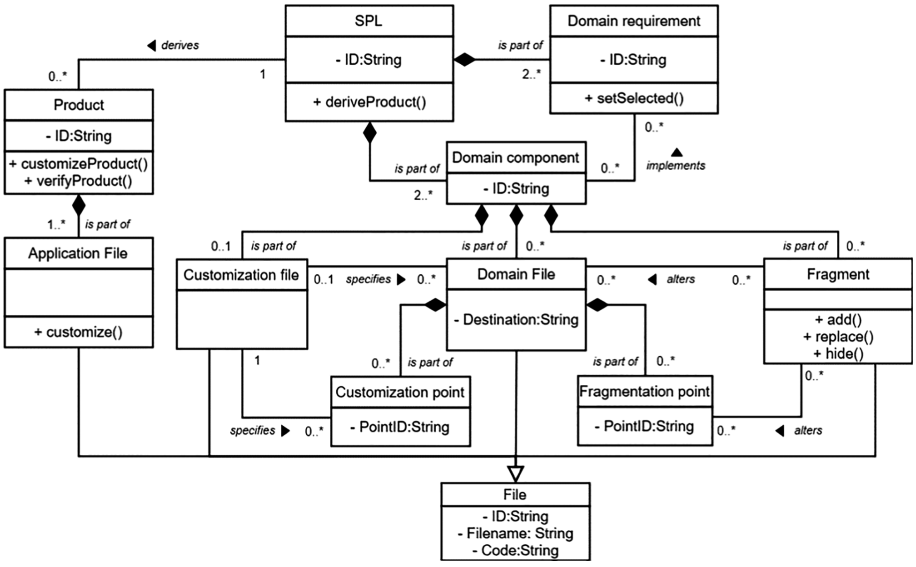

Fig. 1. FragOP metamodel (UML class diagram)

The FragOP metamodel presents the elements that must be used and understood in an SPL that implements a FragOP approach. However, it does not describe how to implement the SPL. That is the objective of the **FragOP process**. Following, we summarize the eight FragOP process activities (cf. Figure 2) including an example of its implementation within VariaMos (cf. Figure 3).

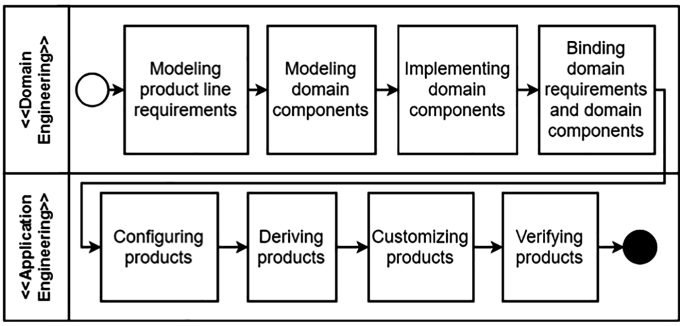

Fig. 2. FragOP process (UML activity diagram)

Domain Engineering

Modeling PL requirements is the activity in which variability models are used to graphically to represent the domain requirements. VariaMos allows specifying the PL requirements in the form of a "Feature model" (see Fig. 3a). **Modeling domain components** is an activity in which the PL domain components, their domain files and the relationship between these elements, are defined through a component model (see Fig. 3b). **Implementing domain components** is the activity in which the components and files are developed based on the domain component model. This activity implies: (i) to develop the domain components with their domain files code, (ii) to include the fragmentation points, (iii) to codify the fragments, (iv) to include the customization points, and (v) to codify the customization files. VariaMos does not support the implementation of domain components, so, the PL developer can use her/his preferred IDE (see Fig. 3c). The result of this activity is the development of a domain component pool that includes the reusable assets of the PL. This activity is detailed in Sect. 3. **Binding domain requirements and domain components**. The binding is an activity that links components and requirements; it allows specifying what domain requirements are realized by what domain components. VariaMos currently supports linking domain components with their corresponding domain requirements (see Fig. 3d). Later, this information is used in the configuration and derivation activities.

Application Engineering

Configuring products consists in selecting the specific features that a specific product will contain based on the stakeholder requirements. VariaMos permits configuring a product by selecting the specific leaf features that the SPL product will contain (see Fig. 3e). **Deriving products** consists in generating specific software products based on

the configured variability model (see Fig. 3f), the derivation activity is detailed in Sect. 3.1. **Customizing products** consists in modifying the derived products based on the customer's needs. For example, to parameterize configuration files or variables, to modify dummy texts, and to include specific customer requirements, among others (see Fig. 3g). The customization activity is detailed in Sect. 3.2. **Verifying products** is the last activity in the application engineering process. Due to the fact that FragOP allows injecting and modifying component file codes (through the use of fragments), it becomes relevant to verify the resulting products. VariaMos implemented ANTLR 4.7.1 and uses a series of parsers and lexers for languages, such as PHP, Java, CSS, MySQL, among others. Based on the derived application file extension, VariaMos analyses the grammar of each application file and generates alerts if errors are found (see Fig. 3h).

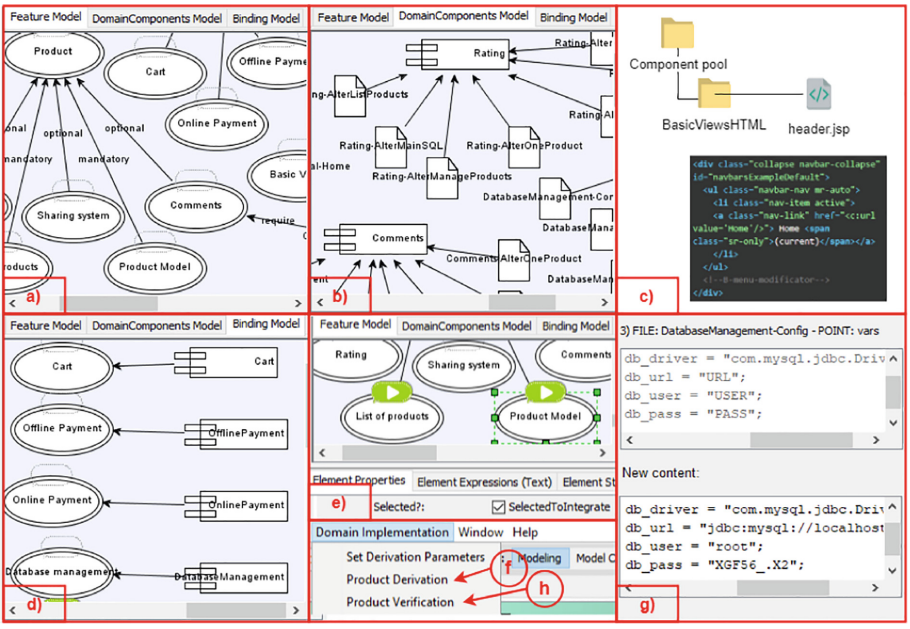

Fig. 3. FragOP process implemented with VariaMos

3 VariaMos (FragOP) Main Capabilities

In order to show the VariaMos (FragOP) main capabilities, and to describe the new FragOP elements, we took an existing e-commerce SPL called ClothingStores [5]. ClothingStores was improved to include the new FragOP elements: customization files and customization points. ClothingStores consists of 25 features and was developed covering most of the problems that SPL developers face when implementing an SPL; such as, **Crosscutting concerns** such as the `Login` component, that in case of being

part of a final product, it must be integrated transversally over multiple other product files. **Fine-grained extensions** such as, modify the header menu, modify specific parts of the product views, and SQL files, among others. **Coarse-grained extensions** such as, replace a validation method over the admin classes, and include class methods, among others. **Product customization** such as the database configuration variables, the name of each derived web store, and some default texts inside the product views must be customized. **Managing multiple language files**, it was designed as a real e-commerce web system which included domain files types, such as SQL, images (.jpg and .png), JavaScript, HTML, JSP, Java, and CSS.

Following, we will describe the two FragOP main capabilities (assembling and customization) with the use of the ClothingStores example.

3.1 Assembling Capability

To implement SPL efficiently, the domain component code has to be variable. Variability is defined as the ability to derive different products from a common set of artifacts [8]. This means the approach, tool, paradigm or methodology used to implement the SPL domain components should support the variability of code. The FragOP approach supports the domain component variability through the use of three FragOP elements (see Fig. 4): **domain files** that represent, for instance, HTML, CSS, JavaScript and Java files. Any file that could be reused for the development of multiple SPL products can be considered as a domain file. **Fragmentation points** are annotations (very simple marks) that specify "points" in which a domain file can be altered (they can be seen as variant points). They are contained inside language comments (similar to the Munge approach, in which the conditional tags are contained in Java comments, so they do not interfere with the development environments). Different to most annotative approaches, in FragOP the variable code is not contained inside the fragmentation points, it is located inside the fragments. And **fragments** which are a special type of file in which the SPL developers specify code alterations to the domain files (they can be seen as variants). Fragments are used to: (i) add, replace, or hide pieces of code over specific fragmentation points (even a piece of code can be injected over multiple locations); and (ii) add or replace entire files (which is useful for domain files that cannot be modified with the inclusion of fragmentation points, such as images or PDF files). Fragments also permit to specify the alteration order through the "fragment priority" property, and they work with multiple domain file types. Correa *et al.* [5] present the complete structure of fragments and fragmentation points.

The VariaMos (FragOP) assembling capability is carried out at application level through the Fig. 3f option. The product derivation consists of generating specific software products based on the configured variability model. The selected features and the variability model are taken as an input. Then, the binding is resolved to show what components should be assembled based on the selected features; and the components are assembled over a product folder (the output). In this activity, VariaMos executes the fragments which modify the product application file code. For example, in Fig. 4 a domain file (header.jsp) supports the code variability through the inclusion of a fragmentation point (menu-modificator). Additionally, a fragment (alterHeader.frag) specifies a code alteration (to include a new header menu element) in the previous

fragmentation point of the previous domain file. Once the product is derived, a copy of the header.jsp is included in the product folder (application file), and the alterHeader. frag injects the new menu element over the derived application file.

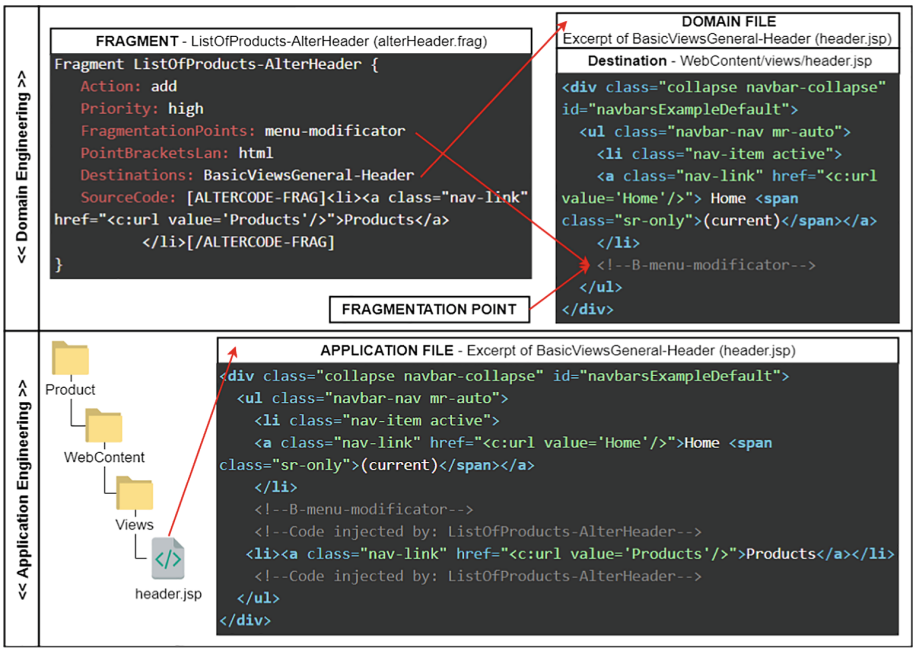

Fig. 4. An assembling scenario with the use of VariaMos (FragOP)

3.2 Customization Capability

Even when SPL products are derived based on the customer's needs, it is very common that these products require customization. For example, to parameterize configuration files or variables, to modify dummy texts, and to include specific customer requirements, among others. FragOP takes advantage of the customization points and customization files and facilitates the customization activity (see Fig. 5).

```
LanguageCommentBlock<BCP>-<PointID>LanguageCommentBlock
LanguageCommentBlock<ECP>-<PointID>LanguageCommentBlock
```

Listing. 1. Customization point shape.

Customization points are annotations that specify "points" in which a domain file should be customized. The customization points shape is similar to the fragmentation points shape, the main difference is that a customization point should contain a beginning part (`BCP`) and an ending part (`ECP`). Listing 1 shows the customization point shape. The code to be customized (at the application level) should be placed in the middle of both `BCP` and `ECP` parts. We use annotations because we want to support

the customization of most kinds of files, and we know that most product customizations are unique.

A **customization file** is a file which specifies the domain files (for the current domain component) that should be customized. Only one customization file is allowed per domain component, its filename must be customization.json, and it must respect the shape presented in Listing 2 and explained thereafter.

```
{
   "IDs": ["FileID1", "FileID2", "..."],
   "CustomizationPoints": ["PointID1", "PointID2", "..."],
   "PointBracketsLans": ["language1", "language2", "..."]
}
```

Listing. 2. Customization point shape.

IDs: <*FileID1*, *FileID2*, …>. It represents the domain files to be customized.
CustomizationPoints (optional): <*pointID1*, *pointID2*, …>. PointIDs are unique texts which serve to identify customization points.
PointBracketsLans (optional): <*language1*, *language2*, …>. It specifies the comment bracket languages in which the customization points are defined. For example, PHP, HTML and Java.

The customization points and the point brackets languages are optional; this way a customization file is able to specify entire domain files that must be customized (replaced) or specify customization points to be customized. Customizing an entire domain file is useful when it is not possible to include customization points. For example, when there is a domain file such as a default logo, that must be customized with the real client company logo.

The VariaMos (FragOP) customization capability is carried out at application level through a VariaMos option called “product customization”. Using this option, a popup shows (i) the customization points of the derived application files, and the SPL developer manually customizes the application file codes; and (ii) the derived application files that should be entirely customized (replaced), and the SPL developer uploads the customized files. For instance, the ClothingStores SPL contained a domain file (Config.java) that specified four variables which allow the communication with the database engine. As a domain file, these variables present sample values; however, for a final product, the value of each variable must be customized. As a consequence, we included a customization point (“vars”) inside the Config.java domain file (see Fig. 5). After the product assembling, the SPL developer customizes the Config.java file with real variable values, which generates the final application files. The content of these files is later verified through Fig. 3h VariaMos option.

VariaMos does not automatically customize the application files because this activity is customer-dependent. However, the activity is streamlined because without the use of customization points and customization files, the SPL developers should manually review each derived application file, trying to figure out what pieces of code and files should be customized. It is important to highlight that customization files and

customization points are very useful for simple customizations, such as parametrizing variables, changing a default text or replacing an image file; nevertheless, complex customization such as creating new components must be applied manually by developers.

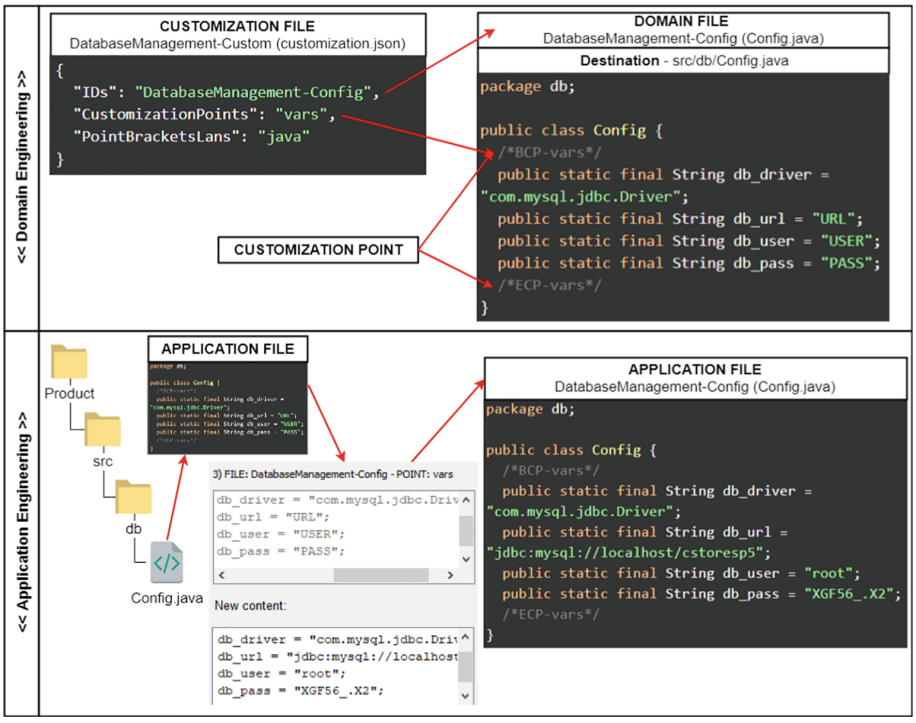

Fig. 5. A customization scenario with the use of VariaMos (FragOP)

3.3 Derivation Results

After following the FragOP process (see Fig. 2), we completed the derivation of the five ClothingStores products. We used the Koscielny *et al.*, [9] DeltaJ 1.5 case study (which presented a SimpleTextEditor SPL as the subject system) as the base to present the ClothingStores results. In comparison, the SimpleTextEditor consisted of 11 features, while the ClothingStores consists of 25 features. The results show that VariaMos (FragOP) is expressive enough to implement a real-world, variant-rich multi-language software system. An inspection of the product code shows that (see Fig. 6): (i) multiple assets of different types were automatically assembled and deployed in the respective project folder structure. (ii) Between 21 and 50 lines of code were manually customized (supported by the VariaMos tool) to complete each product finalization. Even, the database queries were automatically generated. (iii) Several LOC were derived and automatically injected. For instance, 27.72% of the P5 LOC were automatically

injected. This means that a P5 product derivation carried manually without the use of VariaMos will require to manually modify 560 LOC. (iv) If we try to derive the P5 with a compositional approach that is attached to a host language like Java (such as AspectJ, DeltaJ, AHEAD), 26 files must be manually included in the product folder structure, and a minimum of 284 LOC (14% of the total product LOC) must be manually modified to finalize the product derivation. Even, without counting the LOC of Java that implies fine-grained extensions that are not supported by these approaches. And (v) if we try to derive the P5 with annotative approaches, the results could vary depending on the annotative approach language support (for instance, Antenna only supports Java); however, annotative approaches inject all code variations inside the domain files, which is not the case in VariaMos (FragOP). It means that a domain file such as ListOfProducts-OneProduct (oneproduct.jsp) will contain at least 104 LOC in an annotative approach. Nevertheless, in VariaMos (FragOP) it only contains 31 LOC and the code variations are located in separated files (fragments). This characteristic makes domain files of annotative approaches difficult to maintain and evolve.

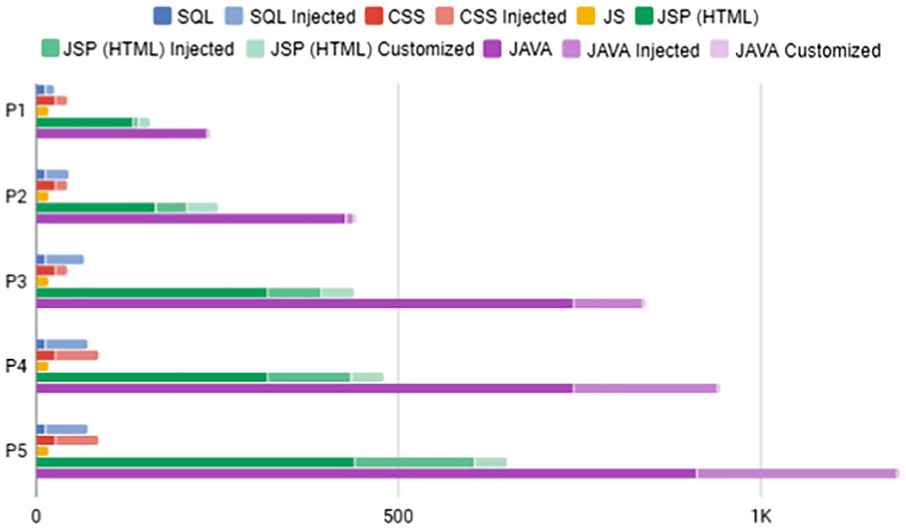

Fig. 6. LOC reused, automatically injected and customized of each derived ClothingStores product by file type

4 Usability Evaluation

This section presents a usability test of VariaMos (version 1.1.0.1). The main idea is to test the VariaMos usability to support the FragOP approach, and thus to gain insight into how easy or difficult it is to follow and understand the FragOP approach. To guide the usability test, we defined the next research question.

RQ: *Is VariaMos a usable tool that supports the FragOP approach?*

In this evaluation, we decided to develop and conduct a usability test by using the ISO/IEC Common Industry Format (CIF) for usability tests [10]. We also applied three evaluation techniques: (i) one for the definition of the experimental tasks, (ii) another for evaluating user satisfaction by gathering their opinion through a survey, and finally (iii) a semi-structured interview to enrich this usability test. The complete usability format result can be found online [11]. The following subsections present: (i) the procedure, (ii) the metrics, (iii) the results and (iv) threats to validity.

4.1 Procedure

The usability test was designed as a process with nine activities (see Fig. 7), which is described next.

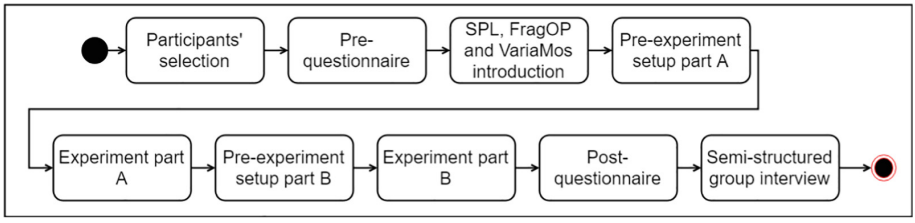

Fig. 7. Usability test process (UML activity diagram)

Participants' Selection. Eight graduate students from the *Universidad Nacional de Colombia* participated in this testing. Participants attended a postgraduate course in software modeling. The usability test was designed in two four-hour sessions. These participants are classified as "software developers who are interested in adopting an SPL methodology" which is one of the VariaMos user target population.

Pre-questionnaire (15 min). We requested the participants to complete a pre-questionnaire related to their background and software experience. The pre-questionnaire was designed using a Likert scale, which had a five-point format: (1) strongly disagree, (2) somewhat disagree, (3) neither agree nor disagree, (4) somewhat agree, and (5) strongly agree. The intention was to collect information about the participants' background and experience, and to confirm the participants' lack of knowledge with FragOP and VariaMos. The pre-questionnaire also showed the participants presented an average of 4 years of experience in software development.

SPL, FragOP and VariaMos Introduction (3 h). We designed a magistral class about the main concepts of SPL, FragOP, and VariaMos, and we developed a very small example of the use of FragOP and VariaMos. This introduction was important because the participants did not have knowledge of SPL, so, we introduced topics, such as software product lines, feature modeling, and product derivation.

Pre-experiment Setup Part A (30 min). The second session started with the "pre-experiment setup part A". Here, the participants were introduced to a document which presented a series of steps to set up an SPL project with the use of VariaMos.

Experiment Part A (Limit: 90 min). We shared with the participants a Google Drive folder with the experiment part A. Then, they were requested to complete five tasks. Therefore, two test administrators were observing and attending the participants' questions. The experiment part A tasks were about: (i) derivation and customization of a new SPL product, (ii) questions about the previous derived product, (iii) modification of a domain file, (iv) modification of the SPL which includes creating a feature, component, binding element, a fragment, and a fragmentation point. And (v) derivation of an additional SPL product.

Pre-experiment Setup Part B (15 min). The participants were introduced to a document which presented a series of steps to set up another SPL project.

Experiment Part B (Limit: 30 min). Participants started to complete two additional tasks. These tasks were about: (i) finding and fixing product derivation errors, and (ii) finding and fixing product verification errors.

Post-questionnaire (15 min). The participants were submitted to a post-questionnaire, which included questions about (i) experiment environment, (ii) overall satisfaction, (iii) VariaMos and FragOP performance, (iv) general question, and (v) specific questions about the VariaMos and FragOP theory.

Semi-structured Group Interview (25 min). We asked the participants four open questions about the tool usability, and we recorded the participants' answers. The questions were: (i) What did you like? (ii) What did you dislike/What should be improved? (iii) What are the opportunities when using this tool in daily business? and (iv) What are the risks when using this tool in daily business?

4.2 Metrics

We defined three usability metrics that tools must provide: effectiveness, efficiency, and satisfaction to its users. For the **effectiveness**, we recorded (i) completion rate (including assisted and unassisted completion), (ii) errors (defined as a task completed wrongly or not completed), and (iii) assists (defined as verbal help given by the test administrators to guide the participants to the next step in completing the task). For the **efficiency**, we recorded (i) task time (the amount of time to complete each task), and (ii) completion rate efficiency (mean completion task rate/mean task time). For the **satisfaction**, we used the post-questionnaire results and measured the participant's perception of ease of use, ease of learning, ease of remembering, and subjective satisfaction. Therefore, we take advantage of the semi-structured group interview results.

4.3 Results

Performance Results. All eight participants completed all of the seven tasks (see Table 1). Three of the participants completed all seven tasks without assistances.

A total of seven assistances were given to the participants, five of these assistances were requested to Task 4 – Part A, which was the most the complex task (participants spent a mean of 31 min to complete this task; see Fig. 8). Figure 8 also shows that the participants spend little time in the development of Task 5 – Part A and Task 2 – Part B. Task 5 – Part A was about a new product derivation, which took on average approximately 4 min; Task 2 – Part B focused on finding validation errors, we included a syntax error over a domain file and on average the participants only spent approximately 4 min in finding and fixing the error. The mean total time to complete all the seven tasks was approximately 72 min. Therefore, there were not errors because all the participants completed all the tasks properly.

Table 1. Participants' performance result summary

	Assisted task completion rate	Unassisted task completion rate	Total task time	Errors	Assistances	Mean task time	Efficiency
Mean	100.000	100.000	72.125	0.000	0.875	10.304	9.982
Standard dev	0.000	0.000	12.654	0.000	0.835	1.808	1.826
Standard error	0.000	0.000	4.474	0.000	0.295	0.639	0.646

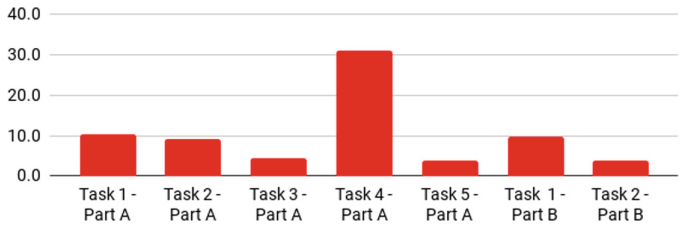

Fig. 8. Participants' average time (minutes) to complete each task

Finally, it is important to highlight that all participants were novice SPL developers and FragOP novice developers. So, the results in this test provide preliminary evidence that VariaMos *is a usable tool that properly supports the FragOP approach* (RQ). All of the participants completed all the tasks (effectiveness), and the mean task time was approximately 10 min (efficiency). The tool also provides errors notifications; which can help developers to easily find fragment errors or domain component errors.

Satisfaction Results. The satisfaction results were obtained from two sources. First, we analyzed 21 of the 26 post-questionnaire questions. Scores for the 21 questions were given for each participant, based on four usability attributes: ease of use, ease of learning, ease of remembering and subjective satisfaction. It is important to realize that usability is not a single, one-dimensional property of a user interface. Usability has multiple components and is traditionally associated with different usability attributes.

Second, we analyzed the semi-structured interview results which will be presented at the end of this section. Finally, the other five post-questionnaire questions results are used in Sect. 4.4 as a source of information for the validation threads. The summary for the 21 questions results can be seen in Fig. 9. The highest satisfaction result was about the "ease of use" of VariaMos with a mean of 4.153 (see Fig. 9a). Therefore, in average the participants had 4.6 correct answers of a total of 6 when asked about VariaMos and FragOP functionalities (see Fig. 9b).

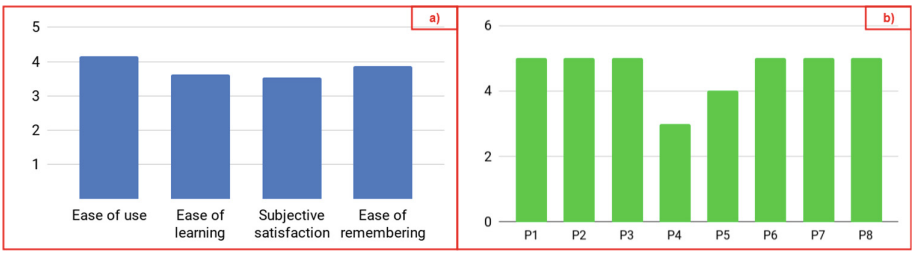

Fig. 9. Participants' satisfaction question average results - Participants' correct answers about VariaMos and FragOP

Finally, the semi-structured interview showed that in general the participants liked the software application and saw the potential of this tool and the FragOP approach. They mentioned that it is a good strategy to reuse the domain components and assembled them. Some participants think this tool could improve their work at their companies and appreciated the way the FragOP approach worked. There were also some recommendations to improve the tool: (i) the graphical interface could be improved. A participant mentioned that future work could be to move the graphical interface into a web project. Allowing the use cell phones or tablets to open the application or to avoid the installation of software programs. (ii) Another participant suggested to automatically generate the component model based on the component pool folder information, which will save time.

4.4 Threats to Validity

Participants sample. The number of subjects may seem relatively small. However, the ISO/IEC CIF for usability tests states "eight or more subjects are recommended" [10]. **Conclusion validity.** There is a threat that many of the results are not based on statistical relationships but on qualitative data. However, given that main aim of the study was to study the behavior and opinions of users of a tool, qualitative research methods are well suited. The analysis of the collected data still depends on our interpretation. The work was performed by a single researcher, but the result was carefully checked by two other researchers. **Project size.** We selected a basic SPL project due to target users that participated in this usability test; however, we have shown in previous sections that the tool also works with complex SPL projects. **Insufficient skills to execute the tasks.** This threat was discarded by the participants'

pre-questionnaire results. **External factors and lack of documentation.** They were discarded by the results of five post-questionnaire questions. Finally, rigorous experiments with complex SPL projects and SPL industry users should be developed in future work.

5 Related Work

There are many SPL implementation approaches that support the SPL component assembling, such as CIDE, DeltaJ, Munge, Antenna, AspectJ, and AHEAD, among others [2]. However, most of these approaches do not provide a product customization capability. Literature presents some customization strategies. Kim *et al.*, [12] propose three strategies: selection, plug-in, and external profile technique. However, these strategies only work with interface classes and are not applied in SPL scenarios. Rabiser *et al.*, [13] propose a decision-oriented software product line approach to support the end user personalization of a system based on its needs. However, the personalization is limited to what the decision model supports. Pleuss *et al.*, [14] propose the use of abstract UI models to bridge the gap between automated, traceable product derivation and customized, high-quality user interfaces. However, it requires to create abstract UI models with all possible scenarios, and this is only applied to user interfaces. Other strategies include inheritance, overloading, dynamic class loading, but not all assets are object-oriented. Finally, Montalvillo *et al.*, [4] developed CUSTOMS, a visualization utility for FeatureHouse that helps to estimate the product customization effort, broken down by product and core-asset.

In the usability testing field, Rabiser *et al.*, [15] presented an implementation of the capabilities in a configuration tool called DOPLER CW. They performed a qualitative investigation on the usefulness of the tool's capabilities for user guidance in product configuration by involving nine business-oriented experts of two industry partners from the domain of industrial automation. They also presented general implications for tool developers. Therefore, Teruel *et al.*, [16], presented a usability evaluation of the CSRML tool 2012; which is a Requirements Engineering CASE tool for the goal-oriented Collaborative Systems Requirements Modeling Language (CSRML). They involved 28 fourth-year Computer Science students in the evaluation, which was reported by following the ISO/IEC 25062:2006 Common Industry Format for usability tests. They obtained high usability levels, but they also revealed some usability flaws. We took as a base these reports to elaborate the VariaMos usability test.

6 Conclusions

This paper presents an enhanced version of FragOP, a framework used to design, implement and reuse domain components in the context of an SPL; which is a mix between a compositional and an annotative approach. The enhanced version supports the SPL product customization. Therefore, we improved an SPL tool called VariaMos to support the FragOP approach. We also included a usability test of VariaMos to gain insights into how VariaMos supports this approach. The key contributions of this paper

are (i) the FragOP and VariaMos enhancements, including an improved FragOP metamodel, FragOP process, and a new customization capability through the use of customization points and customization files. (ii) An SPL implementation through the use of the ClothingStores example; which included the derivation of five different products and an analysis of the derivation results. And (iii) the development of analysis of a usability test for the VariaMos tool. The test results provided preliminary evidence that VariaMos is a usable tool that properly supports the FragOP approach. All participants completed all of the tasks, and the mean task time was approximately 10 min. However, we found the VariaMos UI presented some minor issues (related to responsive design). In the short term, we plan to improve VariaMos to support complex binding relationships, support other variability models such as Orthogonal Variability Model (OVM), and improve the VariaMos UI. Finally, as a future work, we plan to develop more rigorous experiments: (i) to validate the approach benefits, (ii) to compare the different approaches to design and implement the domain components, and (ii) to develop an industrial case to provide valuable evidence about the benefits and limitations of VariaMos (FragOP).

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2001)
2. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: an extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014)
3. de Souza, L.O., O'Leary, P., de Almeida, E.S., de Lemos Meira, S.R.: Product derivation in practice. Inf. Softw. Technol. **58**, 319–337 (2015)
4. Montalvillo, L., Díaz, O., Azanza, M.: Visualizing product customization efforts for spotting SPL reuse opportunities. In: SPLC, pp. 73–80. ACM (2017)
5. Correa, D., Mazo, R., Goméz-Giraldo, G.L.: Fragment-oriented programming: a framework to design and implement software product line domain components. Dyna **85**(207), 74–83 (2018)
6. Mazo, R., Muñoz-Fernández, J.C., Rincón, L., Salinesi, C., Tamura, G.: VariaMos: an extensible tool for engineering (dynamic) product lines. In: SPLC, pp. 374–379. ACM (2015)
7. ISO 9241-11:1998: Ergonomic Requirements for Office Work with Visual Display Terminal (VDTs) – Part 11: Guidance on Usability (1998)
8. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Springer, Berlin (2013)
9. Koscielny, J., Holthusen, S., Schaefer, I., Schulze, S., Bettini, L., Damiani, F.: DeltaJ 1.5: delta-oriented programming for Java 1.5. In: PPPJ, pp. 63–74. ACM (2014)
10. ISO/IEC 25062, Software engineering—Software product Quality Requirements and Evaluation (SQuaRE) - Common Industry Format (CIF) for usability test reports (2006)
11. FragOP-Thesis GitHub repository. https://github.com/danielgara/FragOP-thesis. Accessed 21 Jan 2019
12. Kim, S.D., Min, H.G., Rhew, S.Y.: Variability design and customization mechanisms for COTS components. In: Gervasi, O., et al. (eds.) ICCSA 2005. LNCS, vol. 3480, pp. 57–66. Springer, Heidelberg (2005). https://doi.org/10.1007/11424758_7

13. Rabiser, R., Wolfinger, R., Grunbacher, P.: Three-level customization of software products using a product line approach. In: HICSS, pp. 1–10. IEEE (2009)
14. Pleuss, A., Hauptmann, B., Dhungana, D., Botterweck, G.: User interface engineering for software product lines: the dilemma between automation and usability. In: symposium on Engineering Interactive Computing Systems, pp. 25–34. ACM (2012)
15. Rabiser, R., Grünbacher, P., Lehofer, M.: A qualitative study on user guidance capabilities in product configuration tools. In: ASE, pp. 110–119. ACM (2012)
16. Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., González, P.: A CSCW requirements engineering CASE tool: development and usability evaluation. Inf. Softw. Technol. **56**(8), 922–949 (2014)