Evaluating the Effort for Modularizing Multiple-Domain Frameworks Towards Framework Product Lines with Aspect-oriented Programming and Model-driven Development

Victor Hugo Santiago C. Pinto[1], Rafael S. Durelli[2], André L. Oliveira[2] and Valter V. de Camargo[1]
[1]*Advanced Research Group on Software Engineering (AdvanSE)- Computing Department, Federal University of São Carlos, Washington Luís highway - km 235, 13.565-905, São Carlos, SP, Brazil*
[2]*Institute of Mathematical and Computer Sciences, University of São Paulo, São Carlos, SP, Brazil*

Keywords: Multiple-Domain Frameworks, Framework Product Lines, Framework Modularization.

Abstract: Multiple-Domain Frameworks (MDFs) are frameworks that unconsciously involve variabilities from several domains and present two main problems: i) useless variabilities in the final releases and ii) architectural inflexibility. One alternative for solving this problem is to convert them into Framework Product Lines (FPL). FPL is a product line whose members are frameworks rather than complete applications. The most important characteristic of FPLs is the possibility of creating members (frameworks) holding just the desired variabilities. However, the process of converting an MDF into an FPL is very time-consuming and the choice for the most suitable technique may improve significantly the productivity. The main focus of this paper is an experiment that evaluates two techniques that are usually considered for dealing with features: model-driven development and aspect-oriented programming. Our experiment was conducted comparing the effort in converting an MDF called GRENJ into an FPL called GRENJ-FPL The results showed significant differences regarding the time spent and the occurrence of errors using both techniques.

1 INTRODUCTION

Frameworks are reuse infrastructures that aim to make the development of applications more productive by reusing both design and source-code from specific domains. The reuse process of a framework is known as instantiation, which consists in selecting variabilities to address application requirements (Johnson, 1991; Gamma et al., 1995). Regardless of the way a framework is instantiated, all of its variabilities are usually carried along with the application code in the final release. That is, irrespective whether we are developing a small or a huge application, the same set of features will be in the final release (Batory et al., 2000).

Frameworks are widely adopted when they offer a vast set of variabilities that cover as many domain requirements as possible. Consequently, during evolution processes, it is natural the inclusion of new variabilities aiming to attend clients from new domains. However, when the evolutions are not properly managed and designed, certain added variabilities may go beyond the borders of the original conceived domain, i.e., the new variabilities can belong to a domain/subdomain different from that one originally covered by the framework. When this happens, the framework becomes so broad that supports the development of applications in different domains; or at least in domains not previously thought (Batory et al., 2000; Oliveira et al., 2012). This brings many consequences, but the most evident one is that applications will include in their final release a lot of variabilities from other domains, which will never be useful. We call this kind of frameworks "Multiple-Domain Frameworks" (MDF), because they provide a wide set of variabilities for more than one domain/subdomain (Codenie et al., 1997). This kind of framework present problems for Framework Engineers (FE) and Application Engineers (AE). For AEs, it is necessary to select variabilities from a too vast set; what may decrease their productivity. For FEs, besides the maintenance become very complex, the MDF architectural inflexibility prevent them to compose smaller and more constrained frameworks.

Framework Product Line (FPL) is a Software Product Line (Clements and Northrop, 2001) whose composition of features does not result in

Hugo Santiago C. Pinto V., S. Durelli R., Oliveira A. and V. de Camargo V..
Evaluating the Effort for Modularizing Multiple-Domain Frameworks Towards Framework Product Lines with Aspect-oriented Programming and Model-driven Development.
DOI: 10.5220/0004894900600070
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 60-70
ISBN: 978-989-758-028-4
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

conventional applications, but in frameworks. So, the members of an FPL are frameworks that still need to be instantiated to create or to support the development of final applications. There are two main goals behind the idea of FPLs: i) allowing a more productive development of different frameworks that share a common set of reusable assets and ii) allowing the composition of frameworks that contain just the variabilities likely to be used in its domain, that is, preventing the creation of frameworks with useless variabilities (Batory et al., 2000; Kästner et al., 2009). As the main problem of MDFs is one of the main goals of FPLs (goal ii) we recognize that FPLs is a promising alternative for solving the problems of MDFs, that is, the conversion of an MDF into an FPL can solve the aforementioned problems.

However, the process of converting an MDF into an FPL is a time consuming and error-prone process involving several activities. During this process one important decision is regarding the technique which must be employed during the conversion process (Zanon et al., 2010). The productivity along this process as well as the quality of the resulting MDF is highly dependent on the technique used. Aspect-Oriented Programming (AOP) (Kiczales et al., 1997; Kiczales et al., 2001) and Model-Driven Development (MDD) are two promising technologies that are widely employed to deal with features in software product lines (Mezini and Ostermann, 2004; Trujillo et al., 2007; Voelter and Groher, 2007; Gottardi et al., 2013).

In this paper we present an experiment that compares the effort of converting an MDF into an FPL using AOP with AspectJ (Kiczales et al., 2001) and MDD with Acceleo Templates (Obeo, 2013). The MDF used in the experiment was GRENJ (Durelli et al., 2010), which is a framework originally conceived to support the business-transaction management domain. GRENJ is an MDF because it involves three subdomains, so applications that belong to one of those subdomains and that were developed with the support of this framework, carry with them a lot of variabilities which will never be used. We analyzed the time spent to modularize its subdomains and the problems found in the derived members from the FPL obtained.

Therefore, the main contributions of this paper are: i) providing directions for those who needs to decide which technology (AOP or MDD) is the most suitable for converting MDFs into FPLs. ii) revisiting the concept of FPL and MDF iii) provide a quick overview of the process for converting an MDF into an FPL.

In Section 2, the typical characteristics of the MDFs are discussed. In Section 3, the FPL concept is revisited and the main steps of our conversion process are briefly described. In Section 4, the structure of our experimental study and the results are presented, in which the GRENJ subdomains were modularized in order to obtain FPLs. In Section 5, we present the related work and finally in Section 6 the conclusions and future perspectives.

2 MULTIPLE-DOMAIN FRAMEWORKS

Multiple-Domain Frameworks (MDFs) is a term we have used to designate frameworks whose boundaries go beyond just one domain, that is, they provide variabilities to support the development of several domains of the applications.

Conventional frameworks become MDFs when they are submitted to a non-controlled and unmanaged evolution process. As a consequence, their architecture becomes inflexible avoiding the composition of frameworks targeted to the domain of applications. Since MDFs cover more than one domain, applications that are developed with their support involves in their final release variabilities that will never be used by these applications.

So, one inherent problem of MDFs is the presence of useless variabilities in specific sets of applications, that is, variabilities that are not likely to be used in the future (Batory et al., 2000). As a result, MDFs present problems for Application Engineers (AEs) and Framework Engineers (FEs). AEs need to live together with a vast set of variabilities and parts of them are useless to some specific domains, impacting negatively on their productivity. FEs do not manage to build smaller framework versions thanks to the architectural inflexibility.

These characteristics of the MDFs are common and real difficulties. For instance, if an application is developed using the Hibernate (JBoss-Community, 2013), the final release will include the object code of both the application and the whole framework, regardless of the amount of variabilities that is used.

Based on available documentation about Hibernate (Bauer and King, 2004; JBoss-Community, 2013), we identified that although there were parts separately available, such as: ORM (Object/Relational Mapping), Shards, Search, Tools, Validator, Matamodel Generator e OGM

(Object/Grid Mapper), there are other modules provided together with the core and are not always used in applications. These modules have variabilities to address certain technical subdomains regarding to platform of the applications.

For instance, "Envers" is highly common in SaaS (Software as a Service) in which the software deployment model allows the users to access a specific application or module that is hosted by the vendor as needed. Another example is "OSGi" that is designed for highly dynamics applications and thus they need to be frequently modified. In these applications, the components must be installed, deactivated, and uninstalled during runtime, without requiring a system restart. Furthermore, this framework supports the development in several databases, so it has many dialects and kinds of transactions and sessions.

In general, the whole framework is kept along with the application, even if few variabilities are actually used. However, since there is the possibility of using the other non-used variabilities when the application evolves, this is not recognized as a problem, because this is a framework characteristic.

On the other hand, in cases as Hibernate, there are variabilities not likely to be used in certain sets of applications. To clarify this idea, when we type for a period, it is possible select a variability of Hibernate from the list that is showed. Regardless of the domain or complexity of the application that we are developing, the same set of variabilities is presented and, as aforementioned, there are specific variabilities covered by Hibernate to address different application domains.

3 FRAMEWORK PRODUCT LINES

A "Framework Product Line" is a kind of Software Product Line whose members are frameworks rather than concrete and functional applications. Thus, the features composition in FPLs results in frameworks that still need to be instantiated or coupled to concrete applications to work properly (Zhang and Jacobsen, 2004; Camargo and Masiero, 2008; Batory et al., 2000; Oliveira et al., 2011).

Figure 1 illustrates the main idea of an FPL. In the part (a), there is a feature model which represents an FPL. In the part (b) there are framework members generated from the FPL. In the part (c), we can see the applications that were developed with framework members support.

In the part (a), the common part is called *Core* and the other features stand for the subdomains *S1*, *S2* and *S3*. The reason for creating this FPL is to allow the reuse of the *Core*, since there are common variabilities that may be used in both subdomains; if this is not the case, the ideal would be three entirely independent frameworks. In the case of this FPL, the subdomains are disjoint, i.e., although there is a common set of variabilities, there are specific variabilities to address only one of them. In that way, the three members from this FPL are possible to be built, as illustrated in the part (b). For instance, there is a set of applications which can be developed and evolved with a framework containing only the features *Core* and *S1* without the remaining variabilities, as it is shown in the part (c).

In this paper, we consider only FPLs with coarse-grained features, i.e., features in subdomain level. Thus, their variabilities are associated with subdomains or separated in a common part forming the feature *Core*. However, an FPL can also be developed in order to obtain fine-grained features. In this case, it will have a more flexible architecture, because with the variabilities separated into a greater set of features and combinations among them, it is possible to compose members more targeted and constrained to the applications requirements.

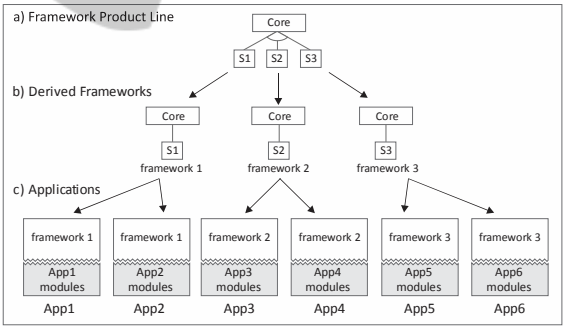

Figure 1: Framework Product Line.

Regardless of the granularity of the features, an FPL should be modularized in a way that does not allow the composition of frameworks with features that will never be used for certain sets of applications. We believe that the most common situation is the generation of large frameworks from FPLs with coarse-grained features. However, if necessary, someone can build FPLs with a larger number of fine-grained features for the provision of various frameworks with slightly different characteristics.

One important point to be highlighted is that in an FPL with fine-grained features, when applications evolve, they may ask for features that do not exist in

the restricted version of the framework. Thus, it is important to have an "on-demand feature selection and composition" strategy. Therefore, there must be a mechanism for searching new features in the FPL, checking them out and composing them to the framework. This is important, but it is out of the scope of this paper.

The main motivation for an FPL with coarse-grained features is to identify if a framework comprehends more than one subdomain, that is, when using it to develop applications that are specific to the framework subdomain. A set of features may not be used during instantiation; nonetheless, since they belong to the same subdomain of the developed application, they are likely to be used in the future. As the granularity of the FPL features is coarse, one may create wider frameworks. This means that applications can evolve without having to seek features that are not in the framework. The number of features that are carried along with the application is much higher, though; it is a trade-off.

3.1 Conversion Process

To transform MDF into FPL, we developed a conversion process that requires several activities. In this paper, we briefly describe only the relevant details of the process in the context of our experimental study.

First of all, it is important to identify the subdomains covered by MDF. A way to perform this task is to consider all available information about the framework, such as documentation, previous knowledge and developed applications with its support. For instance, GRENJ was developed based on a pattern language called GRN (Braga et al., 1999) and applications developed with its support involve rental, trade and maintenance transactions of business resources. In that way, we consider as subdomains: *Rental*, *Trade* and *Maintenance*.

Secondly, we suggest creating a feature model (Kang, 1990) to represent the subdomains. For this, it is necessary to make proper decisions based on domain knowledge to define the properties and relationship among features.

In case of GRENJ, it is important to note that this framework was developed to address applications in a constrained domain, i.e., small businesses as video store, etc. Then, we identified that applications from *Rental* subdomain can evolve and use the variabilities from *Trade* and vice-versa. Thus, *Rental* and *Trade* can stay together, but separated from *Maintenance*. Applications from *Rental* subdomain do not need to use the variabilities from *Maintenance* subdomain. However, the applications from *Trade* subdomain can evolve adding *Maintenance*. In that way, we established "or" relationships among features and we also defined an "excludes" constraint between *Rental* and *Maintenance* (Barreiros and Moreira, 2011). Figure 2 shows the feature model for GRENJ framework. Note that, besides the features to represent each subdomain, there is a feature called *Core* because there are common variabilities among the subdomains.

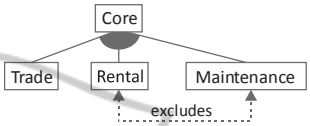

Figure 2: Features model for GRENJ.

Afterwards, we need to identify the modular units that collaborate with the implementation of each feature. Regarding to this issue, it is important to know the framework architecture in order to identify which pieces of code must be modularized to obtain an FPL. After this identification, a proper technique must be selected to modularize the features and obtain an FPL that is flexible in architectural terms.

4 EXPERIMENT

This section describes an experiment that compares the effort to convert an MDF into an FPL using two techniques: Aspect-Oriented Programming (AOP) using AspectJ (Kiczales et al., 2001) and Model-Driven Development (MDD) using Acceleo (Obeo, 2013). We have chosen AOP because it is the most well-known technique for modularizing crosscutting concerns and AspectJ language because is the most disseminated AOP language. Concerning MDD, we have chosen Acceleo because it allows associating pieces of code with their respective features and generating source-code from models. The MDF used as our case study was GRENJ.

The aim of the experiment is to assist domain engineers in choosing the most suitable technology to be used when converting an MDF into an FPL. It is important to highlight that the experiment does not aim neither to evaluate the conversion process nor to show that an FPL is better or worse than an MDF. As the time spent to conduct the conversion is so important as the quality of the obtained FPL, we also take into account the number of errors in

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

members derived from the FPL.

As previously discussed, the process to convert an MDF requires three high-level activities. Considering the complexity in performing these activities, we have decided to evaluate just the third activity. Thus, we had previously identified the MDF subdomains and provided the final feature model to the subjects. This feature model is the same illustrated in Figure 2.

4.1 Planning

The experiment was planned mainly to answer the following research questions: **RQ1: "Is the productivity different when using AOP or MDD to convert an MDF into an FPL?", and RQ2: "Is there difference in terms of structural and inconsistence errors when using AOP or MDD?"**

To answer the first question, we gathered and assessed the time spent to make the conversion. It is important to notice that the total time spent includes the time to handle errors found in the resultant FPLs' members. Similarly, to answer the second question, we analyzed a form that the subjects had filled informing the errors they had found. The planning phase was divided in six parts that are described in the next subsections.

4.1.1 Context Selection

The experiment was conducted involving graduate students in Computer Science and it was performed in a laboratory at the university.

4.1.2 Hypothesis Formulation

The **RQ1** was formalized as follows: **Null hypothesis, H_0:** There is no difference between the AOP and MDD in terms of time spent to obtain an FPL from an MDF, that is, the techniques are equivalent.

Alternative hypothesis, H_1: There is difference between the AOP and MDD in terms of time to obtain an FPL. Thus, the techniques are not equivalent. Hypotheses for the **RQ1** can be formalized by equations 1 and 2:

$$\mathbf{H_0}: \mu_{AOP} = \mu_{MDD} \quad (1)$$

$$\mathbf{H_1}: \mu_{AOP} \neq \mu_{MDD} \quad (2)$$

Similarly, the **RQ2** was also formalized in two hypothesis, as follows: **Null hypothesis, H_0:** There is no significant difference between the AOP and MDD in terms of errors found in the outcome FPLs' members. Thus, the techniques are equivalent. **Alternative hypothesis, H_1:** There is difference between AOP and MDD in terms of errors found in the outcomes FPLs' members. Thus, the techniques are not equivalent. Similarly, the hypotheses for the **RQ2** can be formalized by equations 3 and 4:

$$\mathbf{H_0}: \mu_{AOP} = \mu_{MDD} \quad (3)$$

$$\mathbf{H_1}: \mu_{AOP} \neq \mu_{MDD} \quad (4)$$

4.1.3 Variable Selection

The dependent variables are: (i) the **"time spent to restructure an MDF into an FPL"** and (ii) the **"number of errors found in the outcome FPL members"**. The independent variables are: (i) **FPL:** The subjects were asked to create two FPLs from the given MDF. The only difference between the resultant FPLs was the employed techniques, i.e., either AOP or MDD; (ii) **Trade and Rental subdomains from GRENJ.**

4.1.4 Selection of Subjects

Subjects were selected according to convenience sampling (Wohlin et al., 2000). Fourteen students from the Computer Science post-graduate program participated in the experiment. The scope of their attendance was the "Topics in Software Engineering" course.

4.1.5 Experiment Design

The experiment followed the general design principle of grouping the subjects in homogeneous blocks (Wohlin et al., 2000). Thus, it was possible to avoid a direct impact of the experience level in the treatment outcomes of the restructuring technique factor, increasing the accuracy of the experiment.

In order to divide the subjects in balanced groups, we firstly asked them to fill out a *Categorization Form* with questions about their experience level in themes related to the experiment – this was the self-avaliation. Later, we asked them to solve some experiment-related exercises to check their solutions against the self-evaluation to verify if what they had said about themselves was really true. So, based on these data we divided them in two blocks of seven subjects. These groups were submitted to a pilot experiment, and based on the data gathered from the pilot we rearranged the groups again for the real experiment.

The *Categorization Form* included questions regarding to the knowledge about: Java, AspectJ, Acceleo, Developer Level, GRENJ, GRN and Eclipse IDE. Figure 3 describes the results of the application of this form in a grouped bar graph. This

graph illustrates the levels of experience of all subjects as well as the average level of them, i.e., the rectangles overlapped with labels. These levels were gathered to quantify the weight between the degrees of knowledge of each subject, e.g., scales 1 through 3 wherein: 1 (one) represents that the subject has basic level, 2 (two) represents that the subject has medium level, and 3 (three) represents that the subject has high level of experience. Based on these data, the subjects were separated into two balanced groups, considering the *Categorization Form*, exercises and data collected from pilot experiment. The subjects *S1* to *S7* belong to group 1 and the subjects *S8* to *S14* are from group 2.

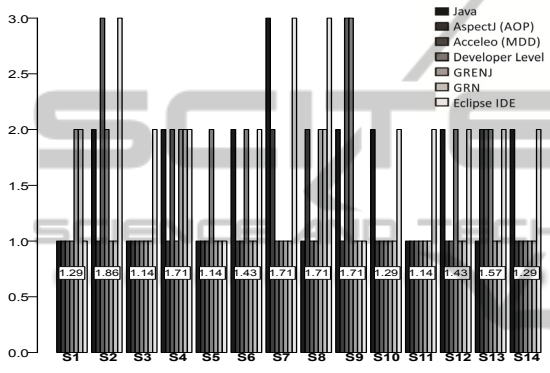

Figure 3: Experience level of all subjects.

Table 1: Experiment Design.

	Phase	Group 1	Group 2
Training	*1st Phase*	Development Techniques: AOP and MDD	
	2nd Phase	Restructuring of the GRENJ towards FPL using AOP and MDD	
Pilot	*1st Phase*	MDD	AOP
		*Modularizing the Trade Subdomain	*Modularizing the Rental Subdomain
	2nd Phase	AOP	MDD
		*Modularizing the Rental Subdomain	*Modularizing the Trade Subdomain
Real Experiment	*1st Phase*	MDD	AOP
		Modularizing the Trade Subdomain	Modularizing the Rental Subdomain
	2nd Phase	AOP	MDD
		Modularizing the Rental Subdomain	Modularizing the Trade Subdomain

Table 1 shows the experiment configuration. During the *Training*, every subject was introduced to both AOP, using AspectJ, and MDD, using Acceleo templates. Afterwards, they were taught on how to use these technologies to modularize the *Trade* and *Rental* subdomains of the GRENJ into features of the target FPL. Notice that the subjects had not converted the whole GRENJ, just two of its domains. However, we claim that this was enough to evaluate the given technologies.

As can be seen in Table 1, the steps Pilot and Real Experiment have the following activities: "Modularizing the Trade Subdomain" and "Modularizing the Rental Subdomain", using either MDD or AOP. The result of each modularization is a "partial FPL" that allows generating some members. These members contain just the variabilities related to the modularized subdomain, avoiding the presence of variabilities that belong to others domains. For instance, the modularization of the *Rental* subdomain results in a partial FPL that allows generating a framework without the features of the *Trade* and *Maintenance* subdomains. In order to test FPL members, we provided a workspace with ready applications to use the members. Thus, the subjects added the member to the applications and then, they executed the test case.

The activities in the pilot and in the real experiment had the same descriptions. They were not exactly the same because this would threaten the validity of the obtained data in the real experiment. Thus, for each activity of the subdomains restructuration, we provided different versions of the *Trade* and *Rental* subdomains. To indicate this difference, in the activities of the pilot experiment there is "*", meaning a different version of the subdomains used in the real experiment.

The size and complexity in modularizing both subdomains were equivalent. For instance, the modularization of both *Trade* and *Rental* subdomains required the modularization of 6 full classes and 17 methods scattered over 7 classes. That is, the activity "Modularizing the Trade Subdomain" in the pilot required that 17 methods scattered over 7 classes and 6 whole classes should be modularized. This activity, in the real experiment, required the same effort, but with others methods and classes. The same is valid for the modularization of the *Rental* subdomain.

4.1.6 Instrumentation

To assist the subjects in the conversion process, we provided them with a document which shows the mapping between classes and features of the GRENJ, simulating as if they had already done these activities previously. Therefore, it was straightforward for the subjects to identify which classes collaborate with the implementation of the features. Table 2 illustrates part of this mapping document.

The lines represent the features and the columns represent the MDF classes. Each cell marked with

"X" indicates that class collaborates with the implementation of that feature. For instance, *AbstractCalculator* is a class that contributes to the implementation of the *Core* feature and *BasicDelivery* contributes to the *Trade* feature. Furthermore, we also had inserted comments in the source code of the classes to indicate which pieces of code were related to the features. In addition, we also provided the class documentation and a guide with the steps that must be followed during the conversion process.

Table 2: Mapping between Classes and Features of the GRENJ.

Classes / Features	AbstractCalculator	BasicDelivery	BasicMaintenance	BasicNegotiation	BasicPurchase	BasicSale	BusinessResourceQuotation	BusinessResourceTransaction	Cash	CashOnDelivery
Core	X			X			X	X	X	X
Trade		X			X	X				
Rental										
Maintenance			X							

Thus, with the feature model and the mapping document at hand, the subjects were asked to obtain two FPLs, each one with a different modularization technique, but equivalent in terms of functionally, complexity and composition alternatives.

4.2 Operation

Once the experiment had been defined and planned, it was performed according to the following steps: preparation, operation, and validation of the collected data.

4.2.1 Preparation

At this stage, the students got committed with the experiment and they were made aware its purpose. Thus, they accepted the terms regarding the confidentiality of the provided data, which would be only used for academic purposes, and their freedom to withdraw, by signing a *Consent Form*. In addition to this form, other objects were provided as follows:

- *Characterization Form*: Questionnaire in which the participants assessed their knowledge about on the technologies and concepts used in the experiment;
- *Support Material*: Roadmap describing the steps to restructure the *Trade* and *Rental* subdomains of the GRENJ;
- *Data Collection Form*: Document containing empty spaces to be filled by the participants to record the start and finishing time of each activity during the experiment.

In order to avoid interference between the time spent in learning AOP and MDD techniques, a 24 hours training, divided into six daily meetings of four-hours, was planned and provided to all participants. Thus, everybody was able to perform the activities proposed in the experiment.

The platform adopted to perform the experiment consisted of Java as implementation language, AspectJ as aspect-oriented language, Acceleo templates as a tool to create the rules of source code generation and the Eclipse IDE as development enviironment.

4.3 Data Analysis

This section presents our findings. The analysis is divided into two points: (*i*) descriptive statistics and (*ii*) hypotheses testing.

4.3.1 Descriptive Statistics

Herein we provide descriptive statistics of the experiment data. The data collected during the experiment are depicted in Table 3: (i) the time employed for each subject for restructuring the GRENJ into two FPLs, using both MDD and AOP and (ii); the types and number of problems found in the resultant FPL members.

Before applying statistical methods, we verified the quality of the input data (Wohlin et al., 2000). Incorrect data sets can be obtained due to systemantic errors or the presence of *outliers,* which are data values that are much higher or much lower than expected when compared with the remaining data. Therefore, we used box plot (Wohlin et al., 2000) as a way to identify *outliers*. Figure 4 shows the box plot based on time spent by all subjects.

As result, we identified just one subject, the "S11". Thus, we did not consider the data collected from this subject in the average time and in the average of the number of problems foud. It is also important to note that although the time consumed is unbalanced, the subjects were separated in a balanced way, as described earlier.

Table 3 shows that for most subjects, AOP technique spent more time to restructure the GRENJ subdomains than the MDD, i.e., aproximately 53% against 47%. This result is due to the fact that, in the AOP, the subjects obtained more errors regarding

the inconsistency and structure, 9 out of 13 subjects considered specifically, concluded the restructuring with more time. On the other hand, when the subjects used MDD, they performed it with less errors. Furthermore, in Table 3 it is possible visualize two kinds of problems that we have found in the FPL members: inconsistency and structure. It is evident by observing this table that the MDD technique guided all subjects to make less problems than the AOP, i.e., 25% against 75% for inconsistency and the same for structure, respectively. By observing Table 3 we can remark that the MDD, using templates conducts the developer to make a design with less problems than the AOP technique.

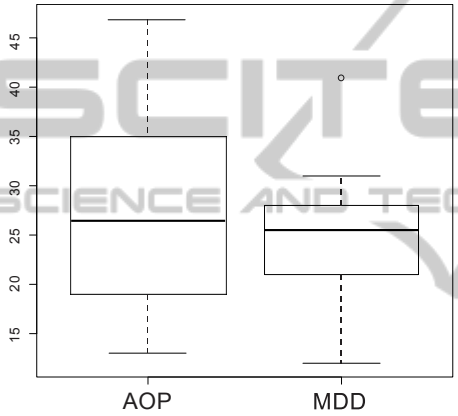

Figure 4: Box plot for the time spent by the subjects.

Table 3: Gathered Data.

		Time (min)		Problems: Inconsistency		Problems: Structure		Total number of problems	
G	S	MDD	AOP	MDD	AOP	MDD	AOP	Total MDD	Total AOP
1	S1	26	24	0	5	0	0	0	5
	S2	24	23	1	2	0	0	1	2
	S3	12	19	0	0	0	2	0	2
	S4	27	37	0	1	1	3	1	4
	S5	28	29	0	0	0	0	1	0
	S6	24	40	0	0	0	1	0	1
	S7	19	23	0	0	0	0	0	0
2	S8	25	13	1	0	2	0	3	0
	S9	12	15	0	1	0	1	0	2
	S10	21	19	1	1	0	2	1	3
	°*S11*	41	47	0	4	0	0	0	4
	S12	26	32	0	0	0	2	0	2
	S13	30	35	1	1	0	1	1	2
	S14	31	34	0	1	1	0	1	1
	Avg.	23.46	26.38	0.31	0.92	0.31	0.92		
	%	47%	53%	25%	75%	25%	75%		

4.3.2 Hypotheses Testing

Hypothesis Testing - Time: Since some statistical tests only apply if the population follows a normal distribution, before choosing a statistical test we examined whether our gathered data departs from linearity. Therefore, we have used Shapiro-Wilk test on the gathered time; this is shown on third and fourth column in Table 3 (group by *Time(min)*). These data represent the time that all subjects spent to devise an FPL by using both MDD and AOP. For these data the *p-value* is 0.8107, considering an $\alpha = 0.05$. As a consequence, we do not reject the hypothesis that the data are from a normally distributed population, as can be seen in the Q-Q plot which is plotted in Figure 5 (left side). In this plot we also showed the results without the presence of the outlier.

Afterwards, we have applied *Paired T-Test* in these data. In order to carry out the test with the data the following were calculated: d = {2, 1, -7, -10, -1, -16, -4, 12, -3, 2, -6, -5, -3}, *Sd* = 6.726336 and t_0= -1.5669. The number of degrees of freedom is f=n-1=13-1=12, and the confidence interval are -6.987761 to 1.141607. According to *Student's t Table*, it can be seen that t 0.025,9 = 2.16037. As for t0 < t 0.025,9 it is impossible to reject the null hypothesis with a two-sided test at the 0.05 level. Therefore, statistically, we may assume that the time needed to modularize an MDF into an FPL by using both AOP and MDD are approximately equal.

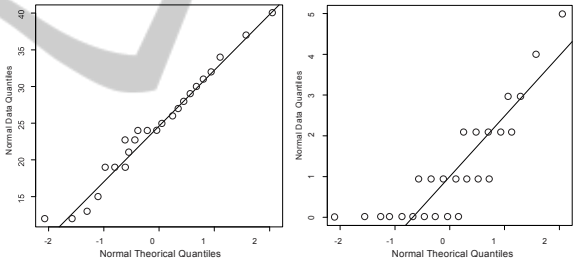

Figure 5: Normality test.

Hypothesis Testing - Problems: Similarly, we used Shapiro-Wilk test on both ninth and tenth column. These data represent the amount of problems that were found in the resultant FPL, by using both the AOP and MDD. Therefore, for these data the *p-value* is 0.001026, considering $\alpha = 0.05$. As *p-value* is less than alpha level we rejected the hypothesis that the data are from a normally distributed population. It is fairly evident by observing the Figure 5 (right side) that the problems lie nearly in a straight line, but not exactly, indicating that the problems may not be i.i.d normal. Thus, we used a non-parametric test, the Wilcoxon signed-rank test. The signed rank of these data are s/r = {-10, -2, -5.5, -8.5, -2, 8.5, -5.5, -5.5, -2}. As result we got a *p-value* = 0.06549227, once *p-value* is greater than 0.05, we can conclude that there is not considerable difference between the means of

the two treatments, considering the Wilcoxon signed-rank test. Thus, we were not able to reject H_0, even though the number of errors obtained with MDD were lesser (9 against 24, without *outlier*) than errors obtained with AOP.

4.4 Threats to Validity

4.4.1 Internal Validity

- Experience Level of Participants: One can argue that the heterogeneous knowledge of the subjects could have affected the collected data. To mitigate this threat, we divided the participants into two-balanced blocks considering the experience level and we rebalanced the groups considering the preliminary results. During the training, the subjects were trained on how to use the AspectJ and Acceleo to restructure frameworks in order to obtain FPLs;
- Productivity under evaluation: One can argue that the results were influenced because the subjects often tend to think they are being evaluated by experiment results. In order to mitigate this, we explained to the subjects that no one was being evaluated and their participation was considered anonymous;
- Facilities used during the study: different computers and installations could affect the recorded timings. However, the subjects used the same hardware configuration and operating system.

4.4.2 Validity by Construction

- Hypothesis expectations: the subjects already knew the researchers, in which reflects one of our hypotheses. This issue could affect the collected data and cause the experiment to be less impartial. In order to keep impartiality, we enforced that the participants had to keep a steady pace during the whole study.

4.4.3 External Validity

- Interaction between configuration and treatment: it is possible that the exercises performed in the experiment are not accurate for every framework development for real world applications. Only two FPLs were developed and they had the same complexity. To mitigate this threat, the exercises were designed considering framework domains based on the real world.

4.4.4 Conclusion Validity

- Measure reliability: it refers to the metrics used to measure the development effort. To mitigate this threat we have used only the time spent, which was captured in forms filled by the subjects;
- Low statistic power: the ability of a statistic test in reveals reliable data. To mitigate we applied two tests: *T-Tests* to statistically analyze the time spent to develop an FPL and Wilcoxon on signed-rank test to statistically analyze the number of problems found in the outcome FPL.

5 RELATED WORK

The most related work to ours is presented by (Batory and Shepherd, 2011), introducing the concept of Product Line of Software Product Line (SPL^2). The idea is to provide simpler specifications and prevent the generated applications from considering unnecessary features in product lines, which core is extensive. Thus, it is demonstrated that, from the analysis of the derived applications from a product line, some applications require only part of the core. Therefore, features can be dissociated from the core, generating a greater number of features, as well as a probably higher number of combinations with the new features of the line. This way, it is possible to generate applications that only contain the essential features. MDFs are fully considered by applications as an "indivisible core". They do not provide experiments comparing possible technologies to modularize the core of an SPL. In our paper, we present a solution for frameworks, but not for SPL. The solution avoids unnecessary features in applications and improves the architectural flexibility and the framework reuse.

Another approach was proposed by (Xu and Butler, 2006). The researchers presented a methodology for restructuring of frameworks in cascade. They consider that a framework can be specified by a set of models and, through these, a set of modularizations may be sequentially applied. The modularization starts in the feature model, then in the use case model and, after that, in the architectural model till it achieves the source code. In order to preserve the framework behavior after each change, trace maps should be used among the models. As a result of the process, decision records regarding the transformations are analysed in order to document and to completely restructure the framework, with improvements in terms of

modularity, which reflect more effective levels of maintenance. Considering their methodology, trace maps can assist the modularization of MDFs in FPLs. Their work presents just a strategy, but does not concern about modularization criteria and an empirical study comparing technologies that can be applied in the modularization process.

6 CONCLUSIONS

The main focus of this work was to investigate the impact AOP and MDD impose when converting a MDF into a FPL. Although FPL principles are technology and language independent, the process of converting an MDF into a FPL is influenced by the technology used. So, we concentrated on the i) the time employed to convert an MDF into an FPL and ii) the number of errors found in the resultant FPL members.

Thus, it must be emphasized that the time spent to make restructuration is not so important as the quality of FPL obtained. A package containing the tools, materials and more details about the experiment steps is available at *http:// www2.dc.ufscar.br/~victor.santiago/exp.zip*.

The main findings of our experiment showed that, in terms of productivity, there is no much difference of using AOP or MDD. However, the number of inconsistency and structural errors of the resulting FPL members were significantly influenced, that is, AOP got 50% more errors than MDD. So, taking into consideration all the limitations of our experiment, we conclude that MDD with Acceleo templates is a better option.

An FPL provides a flexible architecture that enables the creation of members that have a subset of the features. These members are frameworks completely aligned with the domain of these applications, so they need to be instantiated in order to obtain concrete applications. When converting MDFs into FPLs, one can achieve greater flexibility in the composition of features, which provide smaller frameworks and also better productivity levels by reducing errors in the instantiation process. The flexibility of composing features of an FPL enables to address the specific demands of applications.

Considering the concepts presented, one of the future perspectives of our work is to explore the possibility of developing an SPL from an FPL that was obtained from an application framework, including the classes that instantiate it and, then, compose and test the applications derived from this line. Besides, we also intend to explore the synergy between the concept of Software Ecosystems (SECOS) (Jansen and Cusumano, 2012) and FPL. This seems to be a very promising research field, since an FPL may consist of several developers on a distributed and open-source platform, that is, a collaborative network.

One current limitation FPLs is the lack of a comprehensive tool that supports the conversion process. With a complete tool, it would also be possible to investigate the impact of new features in the architecture of an FPL, by analysing the interferences they can cause in the existing ones. It is also believed that the creation of a plugin to visualize the mapping between features and classes can assist FEs in the conversion process, by showing which classes implement a particular feature and which are affected in case a feature is selected.

Another interesting tool which could be developed is one that could enable the creation of FPL members at various abstraction levels. Firstly, the FPL Engineer could create members with the features of a given domain and, then, if necessary, they would select a more specific set of features from this subdomain.

ACKNOWLEDGEMENTS

The authors would like to thank CNPq (Process 560241/2010-0) and Fapesp (Process 2011/04064-8) for financial support.

REFERENCES

Barreiros, J., Moreira, A., 2011. Soft Constraints in Feature Models. *In Proceedings of the 6th International Conference on Software Engineering Advances. ICSEA*, pp. 136-141.

Batory, D., Cardone, R., Smaragdakis, Y., 2000. Object Oriented Frameworks and Product Lines. *In 1st Software Product Lines Conference*. SPLC1, pp. 227-247.

Batory, D., Shepherd, C. T., 2011. Product Lines of Product Lines. Technical Report, University of Texas, Department of Computer Science.

Bauer, C., King, G., 2004. Hibernate in Action (In Action series), Manning Publications Co.. Greenwich, CT, 2nd edition.

Braga, R. T. V., Germano, F. S. R., Masiero, P. C., 1999. A Pattern Language for Business Resource Management. *In Proceedings of the 6th Pattern Language of Programs Conference*, pp. 1-33.

Camargo, V. V., Masiero P. C., 2008. An approach to

design crosscutting framework families. *In Proceedings of the 2008 AOSD workshop on Aspects*, components, and patterns for infrastructure software, pp. 1-6.

Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns, Addison-Wesley Professional, Boston, 3rd edition.

Codenie, W., Hondt, K., Steyaert, P., Vercammen, A., 1997. From Custom Applications to Domain-Specific Frameworks. Communications of the ACM, 40(10).

Durelli, V. H. S., Durelli, R. S., Braga, R. T. V., Borges, S. S., 2010. A Domain Specific Language for Lessening the Effort Needed to Instantiate Applications Using GRENJ Framework. In: Information Systems Brazilian Symposium, pp. 31-40.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: Elements of reusable object-oriented software. Addison Wesley.

Gottardi, T., Durelli, R., López, O., Camargo, V. V., 2013. Model-based reuse for crosscutting frameworks: assessing reuse and maintenance effort. *In Journal of Software Engineering Research and Development*, v. 1, p. 4-34.

Jansen, S., Cusumano, M., 2012. Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance. *In Proceedings of the 4th Workshop on Software Ecosystems*. IWSECO, pp. 41-58.

JBoss-Community, 2013. Hibernate http:// www.hibernate.org.

Johnson, R. E., 1991. Reusing Object-Oriented Design. University of Illinois, Technical Report.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E, Peterson, A. S., 1990. Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report.

Kästner, C., Apel, S., Rahman, S. S. ur, Rosenmüller, M., Batory, D., Saake, G., 2009. On the impact of the optional feature problem: analysis and case studies. *In Proceedings of the 13th International Software Product Line Conference*. SPLC, pp. 181-190.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irving, J., 1997. Aspect Oriented Programming. *In: Proceedings of the European Conference on Object-Oriented Programming. ECOOP*.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. G., 2001. An overview of AspectJ. In Object-Oriented Programming. *Springer Berlin Heidelberg*. ECOOP, pp. 327-353.

Mezini, M., Ostermann, K., 2004. Variability management with feature-oriented programming and aspects. In: *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*. SIGSOFT/FSE, pp. 127-136.

Obeo, 2013. "Acceleo" http://www.eclipse.org/acceleo/.

Oliveira, A. L., Ferrari, F. C., Penteado, R. A. D., Camargo, V. V., 2012. Investigating Framework Product Lines. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1177-1182.

Trujillo, S., Batory, D., Diaz, O., 2007. Feature Oriented Model Driven Development: A Case Study for Portlets. In *Proceedings of the 29th international conference on Software Engineering*, ICSE, pp. 44-53.

Voelter, M., Groher, I., 2007. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *11th International Software Product Line Conference*. SPLC, pp. 233-242.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., 2000. Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell.

Xu, L., Butler, G., 2006. Cascaded Refactoring for Framework Development and Evolution. In *Proceedings of the Australian Software Engineering Conference*. ASWEC, pp. 319-330.

Zanon, I. B., Camargo, V. V., Penteado, R. A. D., 2010. Reestructuring an Application Framework with a Persistence Crosscutting Framework. In *INFOCOMP Journal of Computer Science*, pp. 9-16.

Zhang, C., Jacobsen, H., 2004. Resolving feature convolution in middleware systems. In *Proceedings of the 19th Annual ACM SIGPLAN OOPSL Conference*, pp. 188-205.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS