An Approach to Develop Frameworks from Feature Models

Matheus Viana, Rosângela Penteado, Antônio do Prado
Department of Computing
Federal University of São Carlos
13565-905, São Carlos, SP, Brazil
Email: {matheus_viana, rosangela, prado}@dc.ufscar.br

Rafael Durelli
Institute of Mathematical and Computer Sciences
University of São Paulo
13566-590, São Carlos, SP, Brazil
rdurelli@icmc.usp.br

***Abstract*—Frameworks are reusable software composed of concrete and abstract classes that implement the functionality of a domain. Applications can reuse framework design and code in order to improve their quality and be developed more efficiently. However, to develop software for reuse, such as a framework, is harder than to develop an application. Hence, in this paper we present an approach, named From Features to Framework (F3), to facilitate the development of white box frameworks. This approach is divided in two steps: Domain Modeling, in which the features of the framework are defined; and Framework Construction, in which the framework is designed and implemented according to its features and their relationships. We also present an experiment that evaluated the F3 approach showing that it makes framework development easier and more efficient.**

I. Introduction

Reuse is a practice that aims to reduce time spent in development process and to increase software quality. These advantages can be achieved because the software is not developed from scratch and the reusable artifacts were previously tested [1]. There are different levels of reuse. Copy/paste is the simplest one. Programming languages contain mechanisms, such as class inheritance, that provide reuse of code. Yet there are other more sophisticated forms of reuse, such as frameworks, that provide not only reuse of code, but also design and experience [2].

Frameworks are reusable software composed of concrete and abstract classes that implement the functionality of a domain [3]. Applications reuse the design and the implementation of a framework, adding their specific characteristics to its functionality [4], [5].

Despite the advantages frameworks offer, they are more complex to develop than applications [6]. Frameworks demand an adaptable design. Their classes will be reused by applications that are unknown during framework development, thereby frameworks need mechanisms to identify and to access application-specific classes. Thus, design patterns and advanced resources of programing languages, such as abstract classes, interfaces, polymorphism, generics and reflection, are often used in framework development. In addition to design and implementation complexities, it is also necessary to determine the domain of applications of the framework, the features that compose this domain and the rules that constraint these features [7].

In a previous paper we presented an approach for building Domain-Specific Modeling Languages (DSML) to facilitate framework reuse [8]. In that approach a DSML could be built by identifying the features of the framework domain and the information required to instantiate them. Then application models created with the DSML could be used as input for an application generator to transform them into application code. The experiment presented in this previous paper showed that, besides the gain of efficiency obtained from code generation, the use of DSML protects developers from framework complexities.

In order to promote reuse in application development, in this paper we propose an approach, named From Features to Framework (F3), that aims to facilitate the development of white box frameworks. In this approach framework development starts from defining its domain in a F3 model, which is an extended version of the feature model. Then a set of patterns, called F3 patterns, assist the design and the implementation of the framework according to the features defined in its F3 model.

We also have carried out an experiment in order to verify whether the F3 approach leads the developer to devise frameworks better than the adhoc one. The experiment showed that the F3 approach reduced the problems of incoherence, structure, bad smells and interface found in the outcome frameworks and, consequently, reduced the time spent to develop these frameworks.

The remainder of this paper is organized as follows: the background concepts applied in this research are discussed in Section II; the F3 approach is presented in Section III; an experiment to evaluate the F3 approach is show in Section IV; some related works are discussed in Section V; and conclusions and further works are presented in Section VI.

II. Background

The basic concepts applied in the F3 approach are about patterns, frameworks and domain engineering.

Patterns are successful solutions that can be reapplied to different contexts. They provide reuse of experience to help developers to solve common problems [3]. The documentation of a pattern usually contains its name, the context it can be applied, the problem it is intended to solve, the solution it proposes, illustrative class models and examples of use [9].

IEEE IRI 2013, August 14-16, 2013, San Francisco, California, USA
978-1-4799-1050-2/13/$31.00 ©2013 IEEE

Frameworks act like skeletons that can be instantiated to implement applications [3]. Their classes embody an abstract design to provide solutions for domains of applications [5]. Applications are connected to a framework by reusing its classes. Unlike library classes, whose execution flux is controlled by applications, frameworks control the execution flux accessing the application-specific code [4].

The fixed parts of the frameworks, known as frozen spots, implement common functionality of the domain that is reused by all applications. The variable parts, known as hot spots, can change according to the specifications of the desired application [5].

According to the way they are reused, frameworks can be classified as: white box, which are reused by class specialization; black box, which work like a set of components; and gray box, which are reused by the two previous ways [3].

Domain engineering represents software development related not to a specific application, but to a domain of applications that share common features [10], [11]. A feature is a distinguishing characteristic that aggregates value to applications. Thus, feature models are often used to model domains, illustrating the features that mandatory or optional, variations and require or exclude other features [12].

Different domain engineering approaches can be found in the literature [11], [13], [14]. Although there are differences between them, the basic idea of these approaches is to identify the features of a domain and to develop the artifacts that implement these features and are reused in application engineering.

Domains can also be modeled with metamodel languages, which are used to create Domain-Specific Languages (DSL) [15]. Metamodels are similar to class models, which makes them more appropriate to developers accustomed to the UML. While in feature models, only features and their constraints are defined, metaclasses in the metamodels can contain attributes and operations. On the other hand, feature models can define dependencies between features, while metamodels depend on declarative languages to do it [15].

III. Proposed Approach

The F3 approach provide mechanisms to define a domain at a high level of abstraction and to systematically construct a framework which implements this domain. Thereby, framework development is divided into two steps: A) Domain Modeling, in which a domain is defined and modeled; and B) Framework Construction, in which the framework is designed and implemented according to its domain. These two steps are illustrated in Figure 1.

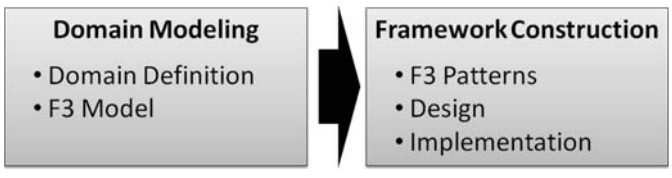

Fig. 1: Steps of the F3 approach.

A. Domain Modeling

The domain of applications that can be developed with the framework is defined and modeled in this step. Usually, a domain is defined by analyzing applications that belong to the desired domain or consulting an specialist in this domain [12]–[14]. The first features to be identified are the mandatory ones, because they represent the code asset of the domain and the minimum necessary to develop an application. Then optional features and variants can be added and the dependencies between them can be specified. It is possible to develop a framework with a small domain at first and keep increasing it as soon as more features become required.

In the F3 approach, domains are modeled in F3 models, which are feature models that includes some elements of metamodels, such as attributes, operations and multiplicity, so that frameworks can be developed from these models. As in conventional feature models, features in F3 models must be arranged in a tree-view, in which the main feature is decomposed in others. However, F3 models do not necessarily form a tree, since a feature can have a relationship targeting a sibling or even itself. Moreover, the graphical notation of F3 models is similar to the one of UML class models. This notation has been adopted because it allows that F3 models can be created by using any UML tool. The elements and relationships in F3 models are:

- **Feature**: graphically represented by a rounded square, it must have a name and it can contain any number of attributes and operations;
- **Decomposition**: relationship that indicates that a feature is composed of another feature. Its minimum multiplicity indicates whether the target feature is optional (`0`) or mandatory (`1`). Its maximum multiplicity indicates how many instances of the target feature can be associated to each instance of the source feature. In white box frameworks an instance of a feature is an application class that extends the framework class of this feature. The maximum multiplicity can assume the following values: `1` (simple), for a single feature instance; `*` (multiple), for a list of a single feature instance; and `**` (variant), for a list of different feature instances.
- **Generalization**: relationship that indicates that a feature is a variation and it can be generalized by another feature.
- **Dependency**: relationship that define constraints for feature instantiation. There are two types of dependency: `requires`, when the A feature requires the B feature, an application that contains the A feature has to include the B feature as well; and `excludes`, when the A feature excludes the B feature, no application can include both features at the same time.

A simplified F3 model for the domain of automated vehicles is shown in Figure 2. This domain is based on Lego Mindstorms NXT 2.0[1], whose hardware can be controlled by Lejos Java API[2]. The requirements of this domain are:

[1] http://mindstorms.lego.com/en-us/products/default.aspx#t
[2] http://lejos.sourceforge.net/#t

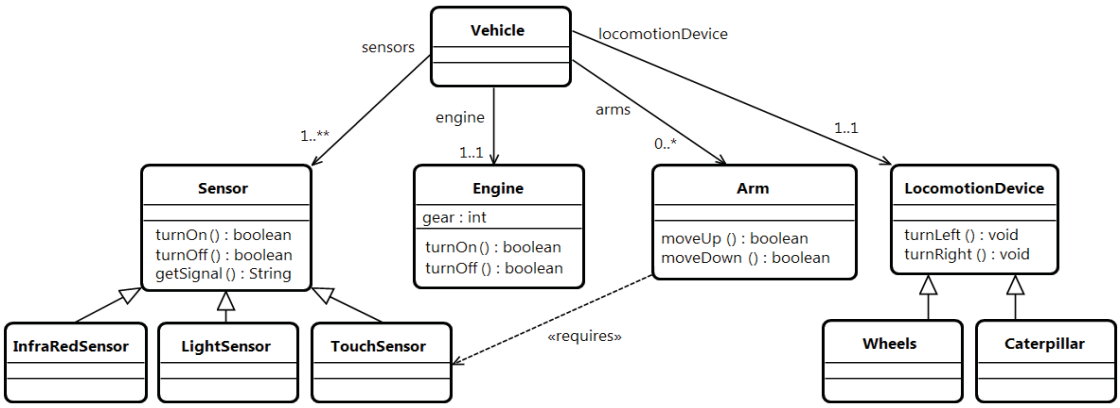

Fig. 2: F3 model for the domain of automated vehicles.

1) An automated vehicle is composed of one engine, one locomotion device, zero or more arms and one or more sensors.
2) Engine provides power for the vehicle. It can be turned on and off and movement sense (forward/reverse) is controlled by its gear.
3) Locomotion device uses the power generated by the engine to move the vehicle and to change its direction. There are two types: wheels or caterpillar.
4) Sensors collect information from environment and return a signal. How the vehicle interprets this signal depends on the purpose of the vehicle. There are three types of sensor: infra-red, light and touch.
5) Arms can be used to grab objects. They can only move up and down and require a touch sensor.

As there are different types of sensor, the relationship between `Vehicle` and `Sensor` is a variant decomposition. A vehicle must have at least one sensor of any type. However, when it has arms, touch sensor becomes necessary, hence the `requires` dependency between `Arm` and `TouchSensor`.

B. Framework Construction

The F3 approach define a set of patterns to assist developers to design and implement a framework from the domain model. These patterns solves problems that go from the creation of classes for the features to the definition of the framework interface. Some of the F3 patterns are presented in Table I.

The documentation of the F3 patterns is organized into topics to help developers to identify when a certain pattern should be used. This documentation is described as follows:

- **Name**: identifies each pattern and summarizes its purpose;
- **Context**: describes a desired behavior for the framework/domain;
- **Scenario/Problem**: describes the arrangement of features and relationships in F3 models that can imply the use of the pattern;
- **Solution**: indicates the code units that should be created to implement the scenario identified by the pattern;
- **Model**: shows a generic graphical representation of the scenario/problem and the solution;
- **Implementation**: displays a fragment of code, in a programming language, that illustrates how the solution can be implemented.

For example, the third pattern listed in Table I, Optional Decomposition, suggests the creation of an operation that must be overridden in the instances of the source feature to specify which class is an instance of the target feature. The documentation of this pattern is:

Name: Optional Decomposition.

Context: when a target feature is optional to a source feature, every instance of the source feature may be associated with a instance of the target feature.

Scenario/Problem: a feature has a decomposition relationship with minimum multiplicity equals `0`.

TABLE I: The F3 patterns that are most commonly applied.

Pattern	Purpose
Domain Feature	Indicates the structures that should be created for a feature.
Mandatory Decomposition	Indicates the code units that should be created when there is a mandatory decomposition linking two features.
Optional Decomposition	Indicates the code units that should be created when there is an optional decomposition linking two features.
Simple Decomposition	Indicates the code units that should be created when there is a simple decomposition linking two features.
Multiple Decomposition	Indicates the code units that should be created when there is a multiple decomposition linking two features.
Variant Decomposition	Indicates the code units that should be created when there is a variant decomposition linking two features.
Variant Feature	Defines a hierarchy of classes for features with variants.
Modular Hierarchy	Defines a hierarchy of classes for features with common attributes and operations.
Requiring Dependency	Indicates the code units that should be created when a feature requires another one.
Excluding Dependency	Indicates the code units that should be created when a feature excludes another one.

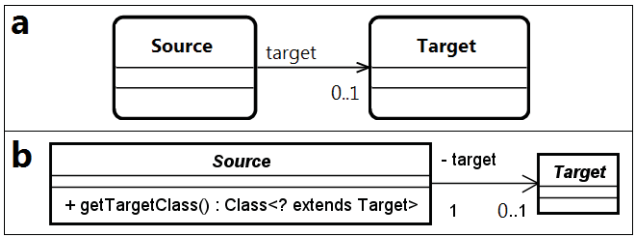

Fig. 3: The (a) pattern scenario and (b) its design solution.

Solution: the class that implements the source feature must have an operation that indicates what class implements the target feature in the applications. By default, this operations returns null indicating that the target feature is not being used.

Model: the (a) scenario and the (b) design solution of this pattern are shown in Figure 3.

Implementation:

```
public abstract class Source {
  public Class<? extends Target> getTargetClass() {
     return null;
  }
}
```

Considering the F3 model in Figure 2, the Optional Decomposition pattern should be applied for the decomposition relationship between `Vehicle` (source) and `Arm` (target). The solution created based on the pattern is shown in Figure 4.

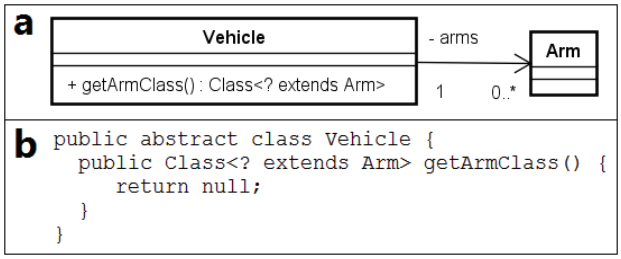

Fig. 4: Solution applied to the relationship between `Vehicle` and `Arm`.

IV. Evaluation

In this section we present an experiment in which the F3 approach has been compared with an adhoc approach. This experiment followed all steps described by Wohlin et al. (2000) and it can be defined as: (*i*) **Analyse** the F3 approach, described in Section III, (*ii*) **for the purpose of** evaluation, (*iii*) **with respect to** efficiency (time) and easiness (problems), (*iv*) **from the point of view of** the developer, and (*v*) **in the context of** MSc and PhD students of Computer Science.

The context of the experiment corresponds to multi-test within object study [16], since the experiment consisted of experimental tests executed by a group of subjects to study a single approach, which is the F3 approach.

A. Planning

The planning phase was divided into the six steps described in the following subsections:

1. Context Selection

The experiment has been performed in laboratory of Computer Science at an university environment. It involved the participation of MSc and PhD students of Computer Science with prior experience in software development using Java language, design patterns and frameworks.

2. Formulation of Hypotheses

The first question the experiment had to answer was: **RQ_1:** "**Which approach takes to a more efficient framework development in terms of time?**". In order to answer this question, the subjects had to measure the time spent (τ) to develop each framework. According to this, the following hypotheses were elaborated:

RQ_1, Null hypothesis, H_0: The F3 approach is not more efficient than the adhoc one in terms of time spent to develop a framework. It can be formalized as:

RQ_1H_0: $\tau_{F3} \geq \tau_{adhoc}$

RQ_1, Alternative hypothesis, H_1: The F3 approach is more efficient than the adhoc one in terms of time spent to develop a framework. It can be formalized as:

RQ_1H_1: $\tau_{F3} < \tau_{adhoc}$

The second question the experiment had to answer was: **RQ_2:** "**Which approach facilitates framework development reducing the number of problems in the frameworks during their development?**". In order to answer this question, we analyzed the reports of the subjects, in which they documented the problems found (ρ) in their frameworks during development, as well as the source-code of the outcome frameworks. By problems we mean defects and bad smells in the source-code of the frameworks. According to this, the following hypotheses were elaborated:

RQ_2, Null hypothesis, H_0: The F3 approach does not facilitate framework development, as the number of problems in the frameworks during their development is not reduced. It can be formalized as:

RQ_2H_0: $\rho_{F3} \geq \rho_{adhoc}$

RQ_2, Alternative hypothesis, H_1: The F3 approach facilitates framework development, reducing the number of problems in the frameworks during their development. It can be formalized as:

RQ_2H_1: $\rho_{F3} < \rho_{adhoc}$

3. Variables Selection

The dependent variables of this experiment were "**time spent to develop a framework**" and "**number of problems found in the frameworks**". The independent variables were:

- **Application**: Each subject had to develop two frameworks: one (Fw1) for the domain of trade and rental transactions and the other (Fw2) for the domain of automated vehicles. Both Fw1 and Fw2 were composed of 10 features.
- **Development Environment**: Eclipse 4.2.1, Astah Community 6.4.
- **Technologies**: Java version 6.

4. Selection of Subjects

Subjects were selected according to convenience sampling [16]. In this non-probabilistic technique, the selected participants were the closest and most convenient to conduct the experiment. Altogether, 26 Msc and PhD students voluntarily participated in the experiment.

5. Experiment Design

The experiment followed the design of grouping the subjects in homogeneous blocks [16], avoiding that their experience level could directly impact in the results. We used a Participant Characterization Form to determine the experience level of each subject. In this form the subjects had to answer multiple-choice questions about their knowledge regarding Java programming, design patterns and frameworks.

The design type of the experiment was **one factor with two treatments paired** [16]. The **factor** is the approach used to develop a framework and the **treatments** are the adhoc and the F3 approaches. Each subject had to develop two frameworks, one applying the adhoc approach and the other applying the F3 approach. The order in which the subjects applied the treatments had no effect in the result. Therefore, the subjects were divided into two blocks of 13 participants with two tasks, as follows:

- **Block 1**: Task 1, development of Fw1 applying the adhoc approach; and Task 2, development of Fw2 applying the F3 approach;
- **Block 2**: Task 1, development of Fw2 applying the adhoc approach; and Task 2, development of Fw1 applying the F3 approach;

6. Instrumentation

The subjects received all necessary materials to assist them during the execution of the experiment. These documents consist of: textual description and models of the framework domains; manual for creating F3 models; documentation of the F3 patterns; Data Collection Form, in which the subjects had to report the time spent to develop the frameworks and the problems found during their development; one Test Application for each framework, which should be used by the subjects to verify the correctness and the completeness of the outcome frameworks; and Feedback Form, in which the subjects should describe their difficulties and write their opinion after the experiment.

B. Operation

After defining and planning the experiment, its operation was carried out in two steps: (*1*) Preparation and (*2*) Execution.

1. Preparation

At first, the subjects signed a Consent Form, stating the objectives and confidentiality of the experiment, and filled the Participant Characterization Form in, reporting their experience in the concepts and technologies utilized in the experiment. After this, the subjects had a training in: adhoc framework development, in which they learned design patterns and code structures commonly used in frameworks to identify application-specific elements; and the F3 approach. After training, the subjects were able to carry out the experiment tasks.

2. Execution

Before starting the execution of the experiment, the subjects were positioned in the blocks and received the materials referent to their respective Task 1. Each subjects had access to an individual computer equipped with the tools required for framework development and the Test Applications.

When all subjects were commanded to execute Task 1 (applying the adhoc approach), they started to measure the time. They used the Astah Community to create a class model of the framework and then use the Eclipse IDE to implement its source-code. When they finished framework implementation, they executed its Test Application to verify whether or not it was developed as expected. If the Test Application showed a message of a problem, the subjects had to report it in the Data Collection Form and fix the problem(s) found. Only when the Test Application returned a successful message, the subject could stop measuring the time. Task 2 (applying the F3 approach) was performed in a similar way to Task 1. In the end, the subjects received the Feedback Form to comment the difficulties and advantages in applying each approach.

C. Analysis of Data

The experiment data is presented in Table II. In general, the groups developed the tasks satisfactorily and the collected data was within the expected limits. This means that the treatments were executed correctly and in accordance with the planning. The analysis of data is divided into two subsections: (*1*) Descriptive Statistics and (*2*) Hypotheses Testing.

1. Descriptive Statistics

In Table II, it can be seen that the F3 approach spent less time to develop a framework than the adhoc approach, i.e., approximately 38.7% against 61.3%. According to the feedback provided by the subjects in a form, this result is due to the fact that, in the F3 approach, although the subjects have spent part of the time trying to identify the F3 patterns that should be used, they saved some time because these patterns assisted them indicating the classes, attributes and operations that should be created. On the other side, when the subjects were developing the frameworks applying the adhoc approach, they spent part of the time trying to find out the code units they should implement. Moreover, most of the subjects reported that they spent a long time maintaining their frameworks, because the Test application returned lots of problem messages. The dispersion of time spent by the subjects are also represented graphically in a boxplot on left side of Figure 5.

In Table II it is also possible to visualize four types of problems that we analyzed in the outcome frameworks: incoherence, structure, bad smells and interface.

The problem of incoherence indicates that the subjects did not develop the frameworks with the correct features and constraints (mandatory, optional and alternative features) of the domain. In other words, they did not designed and implemented the classes, attributes and operations that could make the framework to behave as expected by its domain. In Table II it can be seen that the F3 approach helped the subjects to develop frameworks with less incoherence problems, approximately, 26% in opposition to 74% for the adhoc approach.

TABLE II: Results of the frameworks developed by the subjects.

Subject	Time Spent (minutes)		Number of Problems									
			Incoherence		Structure		Bad Smells		Interface		Total	
	Adhoc	F3	Adhoc	F3	Adhoc	F3	Adhoc	F3	Adhoc	F3	Adhoc	F3
S1	108	72	5	1	2	0	2	2	7	0	16	3
S2	113	74	7	1	4	1	2	0	4	1	17	3
S3	139	83	9	3	11	1	3	1	12	2	35	7
S4	124	78	7	1	5	2	2	1	7	2	21	6
S5	101	67	4	0	3	0	1	0	3	0	11	0
S6	133	81	8	4	7	3	3	3	9	3	27	13
S7	131	79	5	3	3	1	2	1	6	2	16	7
S8	116	73	6	1	5	0	3	0	5	1	19	2
S9	109	79	7	1	4	2	2	1	7	2	20	6
S10	106	69	4	2	3	0	1	0	3	1	11	3
S11	119	71	4	1	4	1	2	0	7	0	17	2
S12	148	83	8	3	6	1	3	3	11	4	28	11
S13	110	74	4	1	2	1	3	1	5	0	14	3
S14	107	72	2	1	3	0	3	0	6	1	14	2
S15	117	76	5	3	5	2	2	1	4	2	16	8
S16	97	68	3	1	1	0	2	0	3	0	9	1
S17	137	80	8	5	9	4	3	3	10	3	30	15
S18	121	75	4	1	6	2	2	2	2	2	14	7
S19	115	73	3	0	4	1	2	0	4	0	13	1
S20	134	81	7	2	6	3	3	1	9	3	25	9
S21	144	86	9	3	7	3	3	3	12	6	31	15
S22	111	76	3	2	4	1	1	1	5	1	13	5
S23	129	83	7	4	8	3	3	2	11	3	29	12
S24	123	79	5	2	5	1	3	1	7	2	20	6
S25	127	77	5	3	3	1	1	0	4	1	13	5
S26	131	78	6	2	4	1	2	2	6	2	18	7
AVG	121.154	76.4231	5.57692	1.961538	4.76923	1.346154	2.26923	1.115385	6.5	1.692308		
%	61.3198	38.6802	73.9796	26.02041	77.9874	22.01258	67.0455	32.95455	79.3427	20.65728		

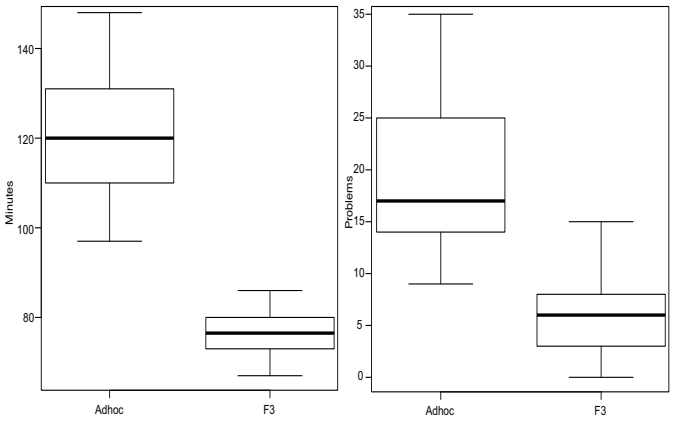

Fig. 5: Dispersion of the total time and number of problems.

The problem of structure indicates that the subjects did not implement the frameworks properly, for example, implementing classes with no constructor, non-abstract when they should be or incorrect relationships. In Table II it can be seen that the F3 approach helped the subjects to develop frameworks with less structure problems, i.e., 22% in opposition to 78%.

The problem of bad smells indicates design weaknesses that do not affect functionality, but make the frameworks harder to maintain. This problem is not a defect, so the Test Applications could not detect it and the subjects did not fixed it. We identified it by analyzing the source-code of the frameworks. In Table II we can remark that the use of the F3 approach resulted in a design with higher quality than the use of the adhoc approach, respectively, 33% against 67%.

The problem of interface indicates absence of getter/setter operations and the lack of operations that allows the applications to reuse the framework and so on. Usually, this kind of problem is a consequence of problems of structure, hence the number of problems of these two types are quite similar. As it can be observed in Table II that the F3 approach helped the subjects to design a better framework interface than when they developed the framework through the adhoc approach, respectively, 21% against 79%.

In the last two columns of Table II it can be seen that the F3T reduced the total number of problems found in the frameworks developed by the subjects. It is also graphically represented in the boxplot on right side of Figure 5.

2. Hypotheses Testing

The objective of this section is to verify with any degree of significance, whether it is possible to reject the nulls hypotheses (see Section IV-A) in favor of the alternative hypothesis based on the data set obtained. As we defined two nulls hypotheses this section is divided into two: (*1*) Hypotheses Testing - Time and (*2*) Hypotheses Testing - Problems.

1) **Hypotheses Testing - Time**: Since some statistical tests are applicable only if the population follows a normal distribution, we applied the Shapiro-Wilk test and created a Q-Q chart to verify whether or not the experiment data departs from linearity before choosing a proper statistical test. As it can be seen in the upper Q-Q charts in Figure 6, the experiment data related to the time spent in framework development is normally distributed. Thus, we decided to apply the Paired T-Test to the experiment data. According to *StatSoft*[3], we carried out this test by calculating: the difference of time between both approaches, $d = \{36, 39, 56, 46, 34, 52, 52, 43, 30, 37, 48, 65, 36, 35, 41, 29, 57, 46, 42, 53, 58, 35, 46, 44,$

[3]http://www.statsoft.com/textbook/distribution-tables/#t

50, 53}; the standard deviation of this difference, S_d = 9.357597; the number of degrees of freedom, $F = N - 1 = 26 - 1 = 25$, where N is the number of subjects; t_0 = 24.3741; and $t_{0.05,25} = 1.708141$. Since $t_0 > t_{0.05,25}$ it is possible to reject the null hypothesis with a two sided test at the 0.05 level. Therefore, statistically, we can assume that the F3 approach reduces the time spent in framework development when compared with the adhoc approach.

2) **Hypotheses Testing - Problems**: Similarly, we used the Shapiro-Wilk test and Q-Q chart on the data shown in the last two columns of Table II, which represent the total number of problems found in the outcome frameworks by using the F3 approach and the adhoc one, respectively. As it can be seen in the lower Q-Q charts in Figure 6, the data depart from linearity, indicating a normal distribution of data. Thus, we also used a Paired T-Test in this case. Again according to *StatSoft*[4], we carried out this test by calculating: the difference of number of problems between both approaches, $d = \{$ 13, 14, 28, 15, 11, 14, 9, 17, 14, 8, 15, 17, 11, 12, 8, 8, 15, 7, 12, 16, 16, 8, 17, 14, 8, 11 $\}$; the standard deviation of this difference, S_d = 4.463183; the number of degrees of freedom, $F = N - 1 = 26 - 1 = 25$, where N is the number of subjects; t_0 = 4.463183; and $t_{0.05,25} = 1.708141$. Since $t_0 > t_{0.05,25}$, it is possible to reject the null hypothesis with a two sided test at the 0.05 level. Therefore, statistically, we can conclude that the F3 approach reduces the number of problems found in the outcome frameworks when compared with the adhoc approach.

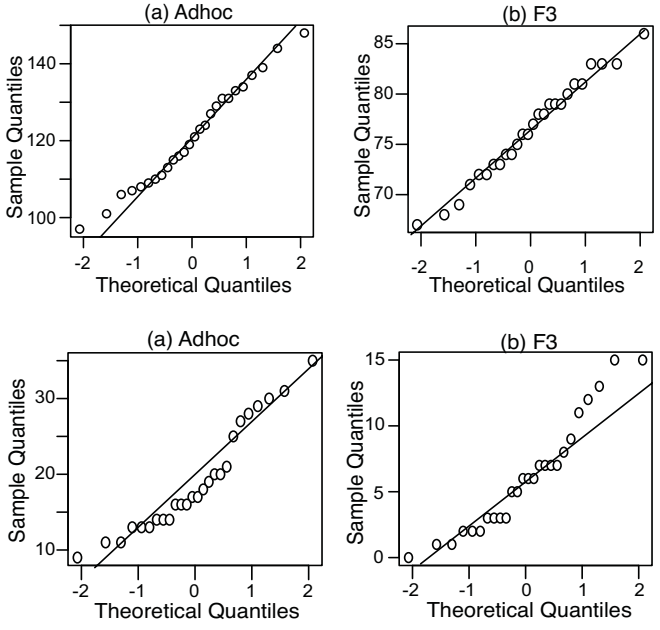

Fig. 6: Q-Q charts of time (upper) and problems found (lower).

[4]http://www.statsoft.com/textbook/distribution-tables/#t

D. Threats to Validity

Internal Validity:

- Experience level of participants: different levels of knowledge of the subjects could affect the collected data. To mitigate this threat, we divided the subjects into two balanced blocks based on their answers in the Participant Characterization Form. All subjects had prior experience in application development reusing frameworks and they were trained in the F3 approach.
- Productivity under evaluation: it might influence the experiment results because subjects often tend to think they are being evaluated. 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.

Validity by Construction:

- Hypothesis expectations: the subjects knew the researchers and knew that the F3 approach was supposed to ease framework development before the experiment. These issues could affect the collected data and cause the experiment to be less impartial. In order to keep impartiality, we enforced that the subjects had to keep a steady pace during the whole study.

External Validity:

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

Conclusion Validity:

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

V. Related Works

Xu and Butler [17] proposed an cascaded refactoring method to develop frameworks. In this method, a framework is specified by different models, sorted by abstraction level (from feature model to source-code). Refactorings are performed on these models following their sequence until the framework is completely developed. In the F3 approach the domain is also defined in feature models and framework design and

implementation are assisted by patterns, which provide more information to help developers than refactorings.

Zhang et al. [18] proposed a Feature-Oriented Framework Model Language (FOFML) to develop frameworks following the MDA approach. When this language is used, domain features are defined in a graphical metamodel and domain constraints are specified in a textual role model. Then, framework classes are implemented according to the features and roles defined in these models. In comparison, our approach define domain features and constraints in a single model.

Antkiewicz et al. [19] presented a framework-specific modeling language approach to modeling and instantiation framework. They used feature model to describe functional requirements of frameworks. However, they focused on framework instantiation level concepts and ignored framework design level concepts.

Amatriain and Arumi [20] also proposed a method to develop frameworks through iterative and incremental activities. In their method, the domain of the framework could be defined from existing applications and the framework could be implemented through a series of refactorings over these applications. The advantage of this method is a small initial investment and the reuse of the applications. Although it is not mandatory, the F3 approach can also be applied in iterative and incremental activities, starting from a small domain and then adding features. Applications can also be used to facilitate the identification of the features of the domain. However, the advantage of the F3 approach is the fact that the design and the implementation of the frameworks are performed with the support of patterns specific for framework development.

VI. Concluding Remarks and Future Work

In this paper we proposed the F3 approach to facilitate the development of white box frameworks. In this approach the framework domain is defined in F3 models, which include elements from feature models and metamodels. Then, the framework is designed and implemented with the support of F3 patterns, structuring code units according to the elements and relationships defined in F3 models.

Our approach promotes reuse in different levels. F3 models represent domains that can be reused and improved in different frameworks. The F3 patterns represent reuse of experience on framework development. Moreover, the outcome frameworks can be reused in the development of several applications, acting as a core asset for a software product line.

The experiment presented in this paper indicated that F3 approach facilitates framework development, because it shows developers how to proceed, making them less prone to insert defects and bad smells in the outcome frameworks. Our approach allowed that even subjects with no experience in framework development could execute this task correctly and spending less time.

In future works we intend to create more F3 patterns to deal with other scenarios in F3 models. Moreover, a tool with a F3 model editor and a code generator based in the F3 patterns is being developed to automatize the use of the F3 approach.

Acknowledgment

The authors would like to thank CAPES and FAPESP for sponsoring our research.

References

[1] S. G. Shiva and L. A. Shala. Software reuse: Research and practice. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 603–609, april 2007.

[2] W. Frakes and K. Kang. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*, 31(7):529–536, july 2005.

[3] R. E. Johnson. Frameworks = (Components + Patterns). *Communications of ACM*, 40(10):39–42, Oct 1997.

[4] M. Abi-Antoun. Making Frameworks Work: a Project Retrospective. In *Companion to the 22nd ACM SIGPLAN conference on Object-Oriented Programming Systems and Applications*, OOPSLA '07, pages 1004–1018, New York, NY, USA, 2007. ACM.

[5] S. Srinivasan. Design Patterns in Object-Oriented Frameworks. *Computer*, 32(2):24 –32, feb 1999.

[6] D. Kirk, M. Roper, and M. Wood. Identifying and Addressing Problems in Object-Oriented Framework Reuse. *Empirical Software Engineering*, 12(3):243–274, Jun 2007.

[7] V. Stanojevic, S. Vlajic, M. Milic, and M. Ognjanovic. Guidelines for Framework Development Process. In *Software Engineering Conference in Russia (CEE-SECR), 7th Central and Eastern European*, pages 1–9, Nov 2011.

[8] M. Viana, R. Penteado, and A. do Prado. Generating Applications: Framework Reuse Supported by Domain-Specific Modeling Languages. In *14th International Conference on Enterprise Information Systems (ICEIS'14)*, Jun 2012.

[9] M. Fowler. Patterns. *IEEE Software*, 20(2):56–57, 2003.

[10] K. Lee, K. C. Kang, and J. Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In *7th International Conference on Software Reuse: Methods, Techniques and Tools*, pages 62–77, London, UK, 2002. Springer-Verlag.

[11] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA): Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[12] J. M. Jezequel. Model-Driven Engineering for Software Product Lines. *ISRN Software Engineering*, 2012, 2012.

[13] H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley, 2004.

[14] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J. DeBaud. PuLSE: a Methodology to Develop Ssoftware Product Lines. In *Proceedings of the Symposium on Software Reusability*, pages 122–131. ACM, 1999.

[15] R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, 2009.

[16] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: an Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[17] L. Xu and G. Butler. Cascaded refactoring for framework development and evolution. *Software Engineering Conference, Australian*, pages 319–330, 2006.

[18] T. Zhang, X. Xiao, H. Wang, and L. Qian. A Feature-Oriented Framework Model for Object-Oriented Framework: An MDA Approach. In *9th IEEE International Conference on Computer and Information Technology*, volume 2, pages 199–204, 2009.

[19] M. Antkiewicz, K. Czarnecki, and M. Stephan. Engineering of Framework-Specific Modeling Languages. *Software Engineering, IEEE Transactions on*, 35(6):795–824, Nov-Dec 2009.

[20] X. Amatriain and P. Arumi. Frameworks Generate Domain-Specific Languages: a Case Study in the Multimedia Domain. *IEEE Transactions on Software Engineering*, 37(4):544–558, Jul-Aug 2011.