# F3T: From Features to Frameworks Tool

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

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

***Abstract*—Frameworks are used to enhance the quality of applications and the productivity of development process, since applications can be designed and implemented by reusing framework classes. However, frameworks are hard to develop, learn and reuse, due to their adaptive nature. In this paper we present the From Features to Framework Tool (F3T), which supports framework development in two steps: Domain Modeling, in which the features of the framework domain are modeled; and Framework Construction, in which the source-code and the Domain-Specific Modeling Language (DSML) of the framework are generated from the features. In addition, the F3T also supports the use of the framework DSML to model applications and generate their source-code. The F3T has been evaluated in a experiment that is also presented in this paper.**

## I. Introduction

Frameworks are reusable software composed of abstract classes implementing the basic functionality of a domain. When an application is developed through framework reuse, the functionality provided by framework classes is complemented with the application requirements. As this application is not developed from scratch, the time spent in its development is reduced and its quality is improved [1]–[3].

Frameworks are often used in the implementation of common application requirements, such as persistence [4] and user interfaces [5]. Moreover, a framework is used as a core asset when many closely related applications are developed in a Software Product Line (SPL) [6], [7]. Common features of the SPL domain are implemented in the framework and applications implement these features reusing framework classes.

However, frameworks are hard to develop, learn and reuse. Their classes must be abstract enough to be reused by applications that are unknown beforehand. Framework developers must define the domain of applications for which the framework is able to be instantiated, how the framework is reused by these applications and how it accesses application-specific classes, among other things [7], [8]. Frameworks have a steep learning curve, since application developers must understand their complex design. Some framework rules may not be apparent in its interface [9]. A framework may contain so many classes and operations that even developers who are conversant with it may make mistakes while they are reusing this framework to develop an application.

In a previous paper we presented an approach for building Domain-Specific Modeling Languages (DSML) to support framework reuse [10]. A DSML can be built by identifying framework features and the information required to instantiate them. Thus, application models created with a DSML can be used to generate application source-code. Experiments have shown that DSMLs protect developers from framework complexities, reduce the occurrence of mistakes made by developers when they are instantiating frameworks to develop applications and reduce the time spent in this instantiation.

In another paper we presented the From Features to Framework (F3) approach, which aims to reduce framework development complexities [11]. In this approach the domain of a framework is defined in a F3 model, which is a extended version of the feature model. Then a set of patterns, called F3 patterns, guides the developer to design and implement a white box framework according to its domain. One of the advantages of this approach is that, besides showing how developers can proceed, the F3 patterns systematizes the process of framework development. This systematization allowed that this process could be automatized by a tool.

Therefore, in this paper we present the From Features to Framework Tool (F3T), which is a plug-in for the Eclipse IDE that supports the use of the F3 approach to develop and reuse frameworks. This tool provides an editor for developers to create a F3 model of a domain. Then, framework source-code and DSML can be generated from the domain defined in this model. Framework source-code is generated as a Java project, while the DSML is generated as a set of Eclipse IDE plug-ins. After being installed, a DSML can be used to model applications. Then, the F3T can be used again to generate the application source-code from models created with the framework DSML. This application reuses the framework previously generated.

We also have carried out an experiment in order to evaluate whether the F3T facilitates framework development or not. The experiment analyzed the time spent in framework development and the number of problems found the source-code of the outcome frameworks.

The remainder of this paper is organized as follows: background concepts are discussed in Section II; the F3 approach is commented in Section III; the F3T is presented in Section IV; an experiment that has evaluated the F3T is presented in Section V; related works are discussed in Section VI; and conclusions and future works are presented in Section VII.

## II. Background

The basic concepts applied in the F3T and its approach are presented in this section. All these concepts have reuse as their basic principle. Reuse is a practice that aims: to reduce time spent in a development process, because the software


DOI 10.1109/SBES.2013.15

is not developed from scratch; and to increase the quality of the software, since the reusable practices, models or code were previously tested and granted as successful [12]. Reuse can occur in different levels: executing simple copy/paste commands; referencing operations, classes, modules and other blocks in programming languages; or applying more sophisticated concepts, such as patterns, frameworks, generators and domain engineering [13].

Patterns are successful solutions that can be reapplied to different contexts [3]. They provide reuse of experience helping developers to solve common problems [14]. The documentation of a pattern mainly 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. There are patterns for several purposes, such as design, analysis, architectural, implementation, process and organizational patterns [15].

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 [9]. 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 [15]. 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 [9]. 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 [2].

Generators are tools that transform an artifact into another [16], [17]. There are many types of generators. The most common are Model-to-Model (M2M), Model-to-Text (M2T) and programming language translators [18]. Such as frameworks, generators are related to domains. However, some generators are configurable, being able to change their domain [19]. In this case, templates are used to define the artifacts that can be generated.

A domain of software consists of a set of applications that share common features. A feature is a distinguishing characteristic that aggregates value to applications [20]–[22]. For example, Rental Transaction, Destination Party and Resource could be features of the domain of rental applications. Different domain engineering approaches can be found in the literature [20], [22]–[24]. Although there are differences between them, their basic idea is to model the features of a domain and develop the components that implement these features and are reused in application engineering.

The features of a domain are defined in a feature model, in which they are arranged in a tree-view notation. They can be mandatory or optional, have variations and require or exclude other features. The feature that most represents the purpose of the domain is put in the root and a top-down approach is applied to add the other features. For example, the main purpose of the domain of rental applications is to perform rentals, so Rental is supposed to be the root feature. The other features are arranged following it.

Domains can also be modeled with metamodel languages, which are used to create Domain-Specific Modeling Languages (DSML). Metamodels, such as defined in the MetaObject Facility (MOF) [25], 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 [18]. A generator can be used along with a DSML to transform models created with this DSML into code. When these models represent applications, the generators are called application generators.

## III. The F3 Approach

The F3 is a Domain Engineering approach that aims to develop frameworks for domains of applications. It has two steps: 1) Domain Modeling, in which framework domain is determined; and 2) Framework Construction, in which the framework is designed and implemented according to the features of its domain.

In Domain Modeling step the domain is defined in a feature model. However, an extended version of feature model is used in the F3 approach, because feature models are too abstract to contain information enough for framework development and metamodels depend on other languages to define dependencies and constraints. This extended version, called F3 model, incorporates characteristics of both feature models and metamodels. As in conventional feature models, features in the F3 models can also be arranged in a tree-view, in which the root feature is decomposed in other features. However, features in the F3 models do not necessarily form a tree, since a feature can have a relationship targeting a sibling or even itself, as in metamodels. 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. This relationship specifies a minimum and a maximum multiplicity. The minimum multiplicity indicates whether the target feature is optional (`0`) or mandatory (`1`). The maximum multiplicity indicates how many instances of the target feature can be associated to each instance of the source feature. The valid values to the maximum multiplicity are: `1` (simple), for a single feature instance; `*` (multiple), for a list of a single feature instance; and `**` (variant), for any number of feature instances.
- **Generalization**: relationship that indicates that a feature is a variation generalized by another feature.
- **Dependency**: relationship that defines a condition for a feature to be instantiated. There are two types of dependency: `requires`, when the A feature requires the B feature, an application that contains the A feature also has to include the B feature; and `excludes`, when the A feature excludes the B feature, no application can include both features.

Framework Construction step has as output a white box framework for the domain defined in the previous step. The F3 approach defines a set of patterns to assist developers to design and implement frameworks from F3 models. The patterns treat 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.

TABLE I: Some of the F3 patterns.

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

In addition to indicate the code units that should be created to implement the framework functionality, the F3 patterns also determine how the framework can be reused by the applications. For example, some patterns suggest to include abstract operations in the classes of the framework that allows it to access application-specific information. In addition, the F3 patterns make the development of frameworks systematic, allowing it to be automatized. Thus, the F3T tool was created to automatize the use of the F3 approach, enhancing the processes of framework development.

## IV. The F3T

The F3T assists developers to apply the F3 approach in the development of white box frameworks and to reuse these frameworks through their DSMLs. The F3T is a plug-in for the Eclipse IDE. So developers can make use of the F3T resources,

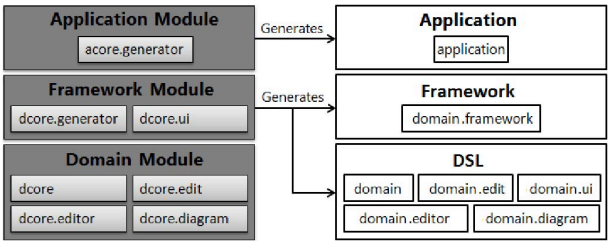


Fig. 1: Modules of the F3T.

such as domain modeling, framework construction, application modeling through framework DSML and application construction, as well the other resources provided by the IDE. The F3T is composed of three modules, as seen in Figure 1: 1) Domain Module (DM); 2) Framework Module (FM); and 3) Application Module (AM).

### A. Domain Module

The DM provides a F3 model editor for developers to define domain features. This module has been developed with the support of the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF) [18]. The EMF was used to create a metamodel, in which the elements, relationships and rules of the F3 models were defined as described in the Section III. The metamodel of F3 models is shown in Figure 2. From this metamodel, the EMF generated the source-code of the Model and the Controller layers of the F3 model editor.

GMF has been used to define the graphical notation of the F3 models. This graphical notation also can be seen as the View layer of the F3 model editor. With the GMF, the graphical figures and the menu bar of the editor were defined and linked to the elements and relationships defined in the metamodel of the F3 models. Then, the GMF generates the source-code of the graphical notation. The F3 model editor is shown in Figure 3 with an example of F3 model for the domain of trade and rental transactions.

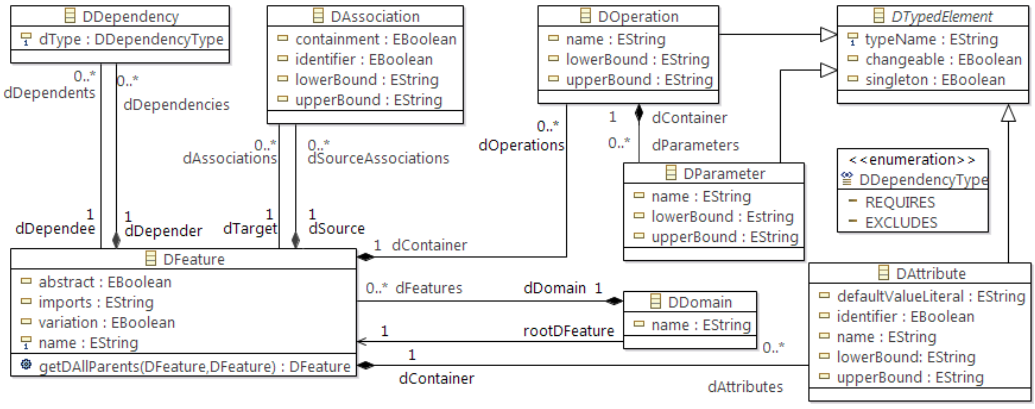


Fig. 2: Metamodel containing elements, relationships and rules of F3 models.

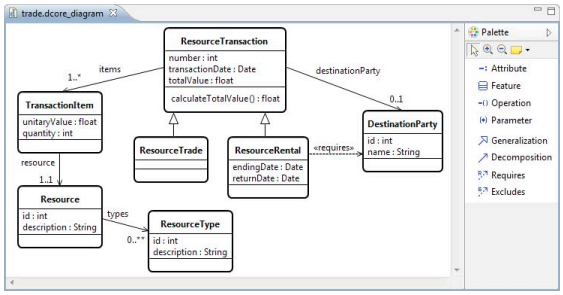


Fig. 3: F3 model for the domain of trade and rental transactions.

### B. Framework Module

The FM is a M2T generator that transforms F3 models into framework source-code and DSML. Despite their graphical notation, F3 models actually are XML files. It makes them more accessible to other tools, such as a generator. The FM was developed with the support of Java Emitter Templates (JET) in the Eclipse IDE [26].

JET contains a framework that is a generic generator and a compiler that translate templates into Java files. These templates are XML files, in which tags are instructions to generate an output based on information in the input and text is a fixed content inserted in the output independently of input. The Java files originated from the JET templates reuse the JET framework to compose a domain-specific generator. Thus, the FM depend on the JET plug-in to work.

The templates of the FM are organized in two groups: one related to framework source-code; and another related to framework DSML. Both groups are invoked from the main template of the DM generator. Part of the JET template which generates Java classes in the framework source-code from the features found in the F3 models can be seen as follows:

```
public
 <c:if test="($feature/@abstract)">abstract </c:if>
class <c:get select="$feature/@name"/> extends
 <c:choose select="$feature/@variation">
  <c:when test="'true'">DVariation</c:when>
  <c:otherwise> <c:choose>
    <c:when test="$feature/dSuperFeature">
      <c:get select="$feature/dSuperFeature/@name"/>
    </c:when>
    <c:otherwise>DObject</c:otherwise> </c:choose>
  </c:otherwise>
 </c:choose> { ... }
```

The framework source-code that is generated by the FM is put in a Java project identified by the domain name and the suffix ".framework". Its classes follow the patterns defined by the F3 approach. For example, the FM generates a class for each feature found in a F3 model. These classes contain the attributes and operations defined in its original feature. All generated classes also, directly or indirectly, extend the `DObject` class, which implements non-functional requirements, such as persistence and logging. Generalization relationships result in inheritances and decomposition relationships result in associations between the involving classes. Additional operations are included in framework classes to treat feature variations and constraints of the domains defined in the F3 models. For example, according to the *Variant Decomposition* F3 pattern, the `getResourceTypeClasses` operation was included in the code of the `Resource` class so that the framework can recognize which classes implement the `ResourceType` feature in the applications. Part of the code of the `Resource` class is presented as folows:

```
/** @generated */
public abstract class Resource extends DObject {

  /** @generated */
  private int id;

  /** @generated */
  private Sting name;

  /** @generated */
  private List<ResourceType> types;

  /** @generated */
  public abstract Class<?>[] getResourceTypeClasses();
```

Framework DSML is generated as a EMF/GMF project identified only by the domain name. The FM generates the EMF/GMF models of the DSML, as seen in Figure 4.a, which was generated from the F3 model shown in Figure 3. Then, source-code of the DSML must be generated by using the generator provided by the EMF/GMF in three steps: 1) using the EMF generator from the *genmodel* file (Figure 4.a); 2) using the GMF generator from the *gmfmap* file (Figure 4.b); and 3) using the GMF generator from the *gmfgen* file (Figure 4.c). After this, the DSML will be composed of 5 plug-in projects in the Eclipse IDE. The projects that contain the source-code and the DSML plug-ins of the framework for the trade and rental transactions domain are shown in Figure 4.d.

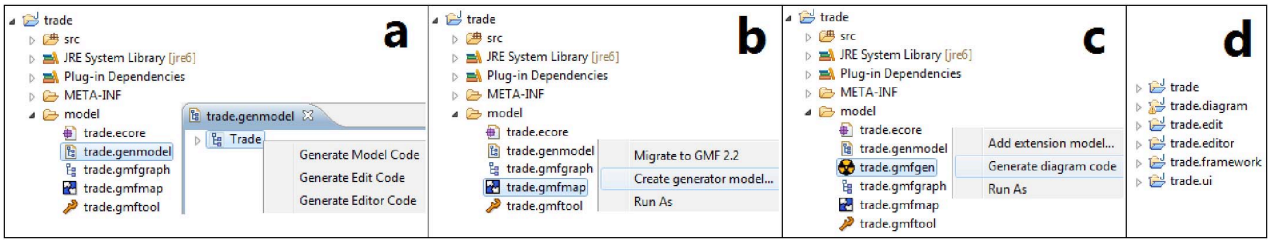


Fig. 4: Generation of the DSML plugins.

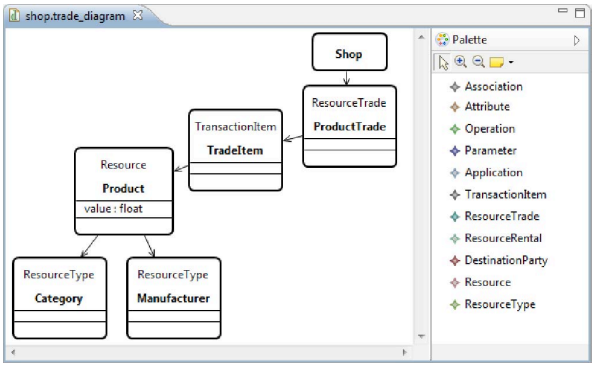


Fig. 5: Application model created with the framework DSML.

### C. Application Module

The AM has been also developed with the support of JET. It generates application source-code from an application model based on a framework DSML. The templates of the AM generate classes that extend framework classes and override operations that configure framework hot spots. After the DSML plug-ins are installed in the Eclipse IDE, the AM recognizes the model files created from the DSML. An application model created with the DSML of the framework for the domain of trade and rental transactions is shown in Figure 5.

Application source-code is generated in the source folder of the project where the application model is. The AM generates a class for each feature instantiated in the application model. Since the framework is white box, the application classes extend the framework classes indicated by the stereotypes in the model. It is expected that many class attributes requested by the application requirements have been defined in the domain. Thus, these attributes are in the framework source-code and they must not be defined in the application classes again. Part of the code of the `Product` class is presented as follows:

```
public class Product extends Resource {

  /** @generated */
  private float value;

  /** @generated */
  public Class<?>[] getResourceTypeClasses() {
    return new Class<?>[] {
      Category.class, Manufacturer.class };
  }
```

## V. Evaluation

In this section we present an experiment, in which we evaluated the use of the F3T to develop frameworks, since the use of DSMLs to support framework reuse has been evaluated in a previous paper [10]. The experiment was conducted following all steps described by Wohlin et al. (2000) and it can be summarized as: (*i*) **analyse** the F3T, described in Section IV; (*ii*) **for the purpose of** evaluation; (*iii*) **with respect to** time spent and number of problems; (*iv*) **from the point of view of** the developer; and (*v*) **in the context of** MSc and PhD Computer Science students.

### A. Planning

The experiment has been planned to answer two research questions:

- **$RQ_1$: Does the F3T reduce the effort to develop a framework?**
- **$RQ_2$: Does the F3T result in a outcome framework with a fewer number of problems?**

All subjects had to develop two frameworks, both applying the F3 approach, but one manually and the other with the support of the F3T. The context of our study corresponds to multi-test within object study [27], hence the experiment consisted of experimental tests executed by a group of subjects to study a single tool. In order to answer the first question, we measured the time spent to develop each framework. Then, to answer the second question, we analyzed the frameworks developed by the subjects, then we identified and classified the problems found in the source-code. The planning phase was divided into seven parts, which are described in the next subsections:

#### 1. Context Selection

26 MSc and PhD students of Computer Science have participated in the experiment, which has been made in an off-line situation. All participants had prior experience in software development, Java programming, patterns and framework reuse.

#### 2. Formulation of Hypotheses

The experiment questions have been formalized as follows:

**$RQ_1$, Null hypothesis, $H_0$:** Considering the F3 approach, there is no significant difference, in terms of time, between developing frameworks with the support of F3T and doing it manually. Thus, the F3T does not reduce the time spent to develop frameworks. This hypothesis can be formalized as:

**$H_0$:** $\mu_{F3T} = \mu_{manual}$

**$RQ_1$, Alternative hypothesis, $H_1$:** Considering the F3 approach, there is a significant difference, in terms of time, between developing frameworks with the support of F3T and doing it manually. Thus, the F3T reduces the time spent to develop frameworks. This hypothesis can be formalized as:

**$H_1$:** $\mu_{F3T} \neq \mu_{manual}$

**$RQ_2$, Null hypothesis, $H_0$:** Considering the F3 approach, there is no significant difference, in terms of problems found in the outcome frameworks, between developing frameworks using the F3T and doing it manually. Thus, the F3T does not reduce the mistakes made by subjects while they are developing frameworks. This hypothesis can be formalized as:

**$H_0$:** $\mu_{F3T} = \mu_{manual}$

**$RQ_2$, Alternative hypothesis, $H_1$:** Considering the F3 approach, there is a significant difference, in terms of problems found in the outcome frameworks, between developing frameworks using the F3T and doing it manually. Thus, the F3T reduces the mistakes made by subjects while they are developing frameworks. This hypothesis can be formalized as:

**$H_1$:** $\mu_{F3T} \neq \mu_{manual}$

*3. Variables Selection*

The dependent variables of this experiment were:

- **time spent to develop a framework**;
- **number of problems found in the frameworks**.

The independent variables were as follows:

- **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 automatic vehicles. Both Fw1 and Fw2 had 10 features.
- **Development Environment**: Eclipse 4.2.1, Astah Community 6.4, F3T.
- **Technologies**: Java version 6.

*4. Selection of Subjects*

The subjects has been selected through a non probabilist approach by convenience, so that the probability of all population elements belong to the same sample is unknown.

*5. Experiment Design*

The subjects were carved up in two blocks of 13 subjects:

- **Block 1**, development of Fw1 manually and development of Fw2 with the support of the F3T;
- **Block 2**, development of Fw2 manually and development of Fw1 with the support of the F3T.

We have chosen use block to reduce the effect of the experience of the students, that was measured through a form in which the students answered about their level of experience in software development. This form was given to the subjects one week before the pilot experiment herein described. The goal of this pilot experiment was to ensure that the experiment environment and materials were adequate and the tasks could be properly executed.

*6. Design Types*

The design type of this experiment was **one factor with two treatments paired** [27]. The **factor** in this experiment is the manner how the F3 approach was used to develop a framework and the **treatments** are the support of the F3T against the manual development.

*7. Instrumentation*

All necessary materials to assist the subjects during the execution of this experiment were previously devised. These materials consisted of forms for collecting experiment data, for instance, time spent to develop the frameworks and a list of the problems were found in the outcome frameworks developed by each subject. In the end of the experiment, all subjects received a questionnaire, in which they should report about the F3 approach and the F3T.

### B. Operation

The operation phase of the experiment was divided into two parts, Preparation and Execution, as described in the next subsections:

*1. Preparation*

Firstly, the subjects received a characterization form, containing questions regarding their knowledge about Java programming, Eclipse IDE, patterns and frameworks. Then, the subjects were introduced to the F3 approach and the F3T.

*2. Execution*

Initially, the subjects signed a consent form and then answered a characterization form. After this, they watched a presentation about frameworks, which included the description of some known examples and their hot spots. The subjects were also trained on how to develop frameworks using the F3 approach with or without the support of the F3T.

Following the training, the pilot experiment was executed. The subjects were split into two groups considering the results of the characterization forms. Subjects were not told about the nature of the experiment, but were verbally instructed on the F3 approach and its tool. The pilot experiment was intended to simulate the real experiments, except that the applications were different, but equivalent. Beforehand, all subjects were given ample time to read about approach and to ask questions on the experimental process. This could affect the experiment validity, then, the data from this activity was only used to balance the groups.

When the subjects understood what their had to do, they received the description of the domains and started timing the development of the frameworks. Each subject had to develop the frameworks applying the F3 approach, i.e., creating its F3 model from a document which describes its domain features and then applying the F3 patterns to implement it.

### C. Analysis of Data

This section presents the experimental findings. The analysis is divided into two subsections: (*1*) Descriptive Statistics and (*2*) Hypotheses Testing.

*1. Descriptive Statistics*

The time spent by each subject to develop a framework and the number of problems found in the outcome frameworks are shown in Table II. From this table, it can be seen that the subjects spent more time to develop the frameworks when they were doing it manually, approximately 72.5% against 27.5%. This result was expected, since the F3T generates framework source-code from F3 models. However, it is worth highlighting that most of the time spent in the manual framework development was due to framework implementation and the effort to fix the problems found in the frameworks, while most of the time spent in the framework development supported by the F3T was due to domain modeling. The dispersion of time spent by the subjects are also represented graphically in a boxplot on left side of Figure 6.

In Table II it is also possible to visualize four types of problems that we analyzed in the outcome frameworks: (*i*) incoherence, (*ii*) structure, (*iii*) bad smells, (*iv*) interface.

The problem of incoherence indicates that, during the experiment, the subjects did not model the domain of the framework as expected. Consequently, the subjects did not develop the frameworks with the correct domain features and

TABLE II: Development timings and number of problems.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

constraints (mandatory, optional, and alternative features). As the capacity to model the framework domains depend more on the subject skills than on tool support, incoherence problems could be found in equivalent proportions, approximately 50%, when the framework was developed either manually or with the support of the F3T.

The problem of structure indicates that the subjects did not implement the frameworks properly during the experiment. For example, they implemented classes with no constructor and incorrect relationships or when they forgot to declare the classes as abstract. This kind of problem occurred when the subjects did not properly follow the instructions provided by the F3 patterns. In Table II it can be seen that the F3T helped the subjects to develop frameworks with less structure problems, i.e., 10% in opposition to 90%.

The problem of bad smells indicates design weaknesses that do not affect functionality, but make the frameworks harder to maintain. In the experiment, this kind of problem occurred when the subjects forgot to apply some F3 patterns related to the organization of the framework classes, such as the *Modular Hierarchy* F3 pattern. By observing Table II we can remark that the F3T made a design with higher quality than the manual approach, i.e, 0% against 100%, because the F3T automatically identified which patterns should be applied from the F3 models.

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 F3T helped the subjects to design a better framework interface than when they developed the framework manually, i.e., 8.6% against 91.4%.

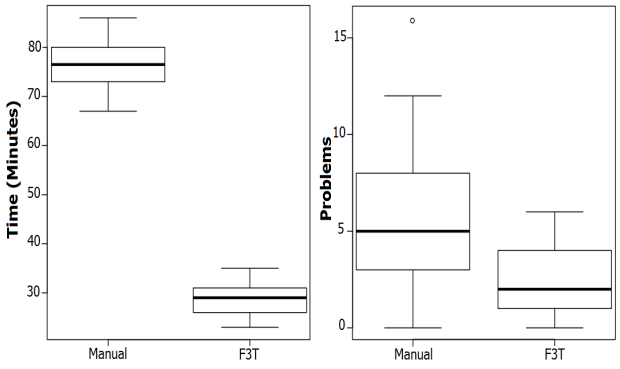


Fig. 6: Dispersion of the total time and number of problems.

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 6.

*2. Testing the Hypotheses*

The objective of this section is to verify, based on the data obtained in the experiment, whether it is possible to reject the null hypotheses in favor of the alternative hypotheses. 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. The tests has been carried out as follows:

1) **Time**: We have applied the Shapiro-Wilk test on the experiment data that represents the time spent by

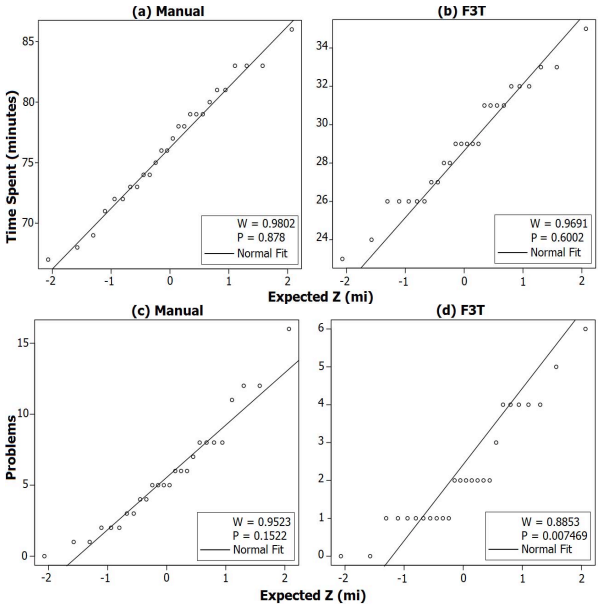


Fig. 7: Normality tests.

each subject to develop a framework manually or using the F3T, as shown in Table II. Considering an $\alpha$ = 0.05, the p-values are 0.878 and 0.6002 and Ws are 0.9802 and 0.9691, respectively, for each approach. The test results confirmed that the experiment data related to the time spent in framework development is normally distributed, as it can be seen in the Q-Q charts (a) and (b) in Figure 7. Thus, we decided to apply the Paired T-Test to these data. Assuming a Paired T-Test, we can reject $H_0$ if $\mid t_0 \mid > t_{\alpha/2,n-1}$. In this case, $t_{\alpha,f}$ is the upper $\alpha$ percentage point of the t-distribution with $f$ degrees of freedom. Therefore, based on the samples, $n$ = 26 and $d = \{46,42,52,49,41,49,55,50,53,42,42,52,48,43,45, 42,47,48,44,49,51,48,52,51,48,45\}$, $S_d$ = 9.95 and $t_0$ = 1.6993. The average values of each data set are $\mu_{manual} = 76.42$ and $\mu_{F3T} = 28.96$. So, $d = 76.42 - 28.96 = 47.46$, which implies that $S_d = 3.982$ and $t_0 = 60.7760$. The number of degrees of freedom is $f = n - 1 = 26 - 1 = 25$. We take $\alpha = 0.025$. Thus, according to *StatSoft*[1], it can be seen that $t_{\ 0.025,25} = 2.05954$. Since $\mid t_0 \mid > t_{0.025,25}$ **it is possible to reject the null hypothesis with a two sided test at the 0.025 level**. Therefore, statistically, we can assume that, when the F3 approach is applied, the time needed to develop a framework using F3T is less than doing it manually.

2) **Problems**: Similarly, we have applied the Shapiro-Wilk test on the experiment data shown in the last two columns of Table II, which represent the total number of problems found in the outcome frameworks that were developed whether manually or using the F3T. Considering an $\alpha$ = 0.05, the p-values are 0.1522 and 0.007469, and Ws are 0.9423 and 0.8853, respectively, for each approach. As it can be seen in the Q-Q charts (c) and (d) in Figure 7, the test results confirmed that date related to manual development is normally distributed, but the data related to the F3T can not be considered as normally distributed. Therefore we applied a non-parametric test, the Wilcoxon signed-rank test in these data. The signed rank of these data are $S/R$ of $\mid t_{problems_{manual}} - t_{problems_{F3T}} \mid$ = { +3.5, +7.5, +7.5, +16.5, -3.5, +23, +3.5, +3.5, +10.5, +10.5, +3.5, +18.5, +10.5, +14, +24, +18.5, +3.5, +21, +21, +14, +21, +10.5, +14, +16.5}, $S/R$ stand for "signed rank". As result we got a p-value = 0.001078 with a significance level of 1%. Based on these data, we conclude there is considerable difference between the means of the two treatments. We were able to reject H0 at 1% significance level. The p-value is very close to zero, which further emphasizes that the F3T reduces the number of problems found in the outcome frameworks.

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

### D. Opinion of the Subjects

We analyzed the opinion of the subjects in order to evaluate the impact of using the F3T. After the experiment operation, all subjects received a questionnaire, in which they could report their perception about applying the F3 approach manually or with the support of the F3T.

The answers in the questionnaire has been analyzed in order to identify the difficulties in the use of the F3 approach and its tool. As it can be seen in Figure 8, when asked if they encountered difficulties in the development of the frameworks by applying the F3 approach manually, approximately 52% of the subjects reported having significant difficulty, 29% mentioned partial difficulty and 19% had no difficulty. In contrast, when asked the same question with respect to the use of the F3T, 73% subjects reported having no difficulty, 16% subjects mentioned partial difficulty and only 11% of all subjects had significant difficulty.

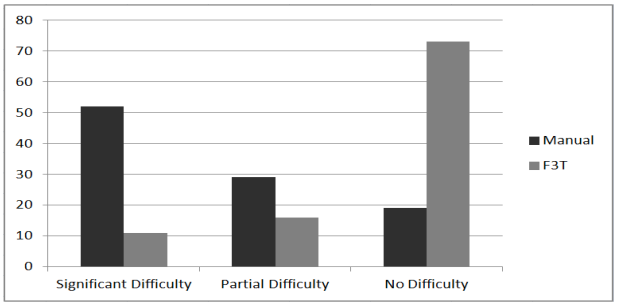


Fig. 8: Level of difficulty of the subjects.

The reduction of the difficulty to develop the frameworks, shown in Figure 8, reveals that the F3T assisted the subjects in this task. The subjects also answered in the questionnaire about the difficulties they found during framework development. The most common difficulties pointed out by the subjects when

they developed the frameworks manually were: 1) too much effort spent on coding; 2) mistakes they made due to lack of attention; 3) lack of experience for developing frameworks; and 4) time spent identifying the F3 patterns in the F3 models. In contrast, the most common difficulties faced by the subjects when they used the F3T were: 1) lack of practice with the tool; and 2) some actions in the tool interface, for instance, opening the F3 model editor, take many steps to be executed. The subjects said that the F3 patterns helped them to identify which structures were necessary to implement the frameworks in the manual development. They also said the F3T automatized the tasks of identifying which F3 patterns should be used and of implementing the framework source-code. Then, they could keep their focus on domain modeling.

### *E. Threats to Validity*

**Internal Validity:**

- Experience level of the subjects: the subjects had different levels of knowledge and it could affect the collected data. To mitigate this threat, we divided the subjects in two balanced blocks considering their level knowledge and rebalanced the groups considering the preliminary results. Moreover, all subjects had prior experience in application development reusing frameworks, but not for developing frameworks. Thus, the subjects were trained in common framework implementation techniques and how to use the F3 approach and the F3T.
- Productivity under evaluation: there was a possibility that this might influence the experiment results because 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. Thus, the subjects used the same hardware configuration and operating system.

**Validity by Construction:**

- Hypothesis expectations: the subjects already knew the researchers and knew that the F3T was supposed to ease framework development, which reflects one of our hypothesis. These issues 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.

**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 frameworks were developed and they had the same complexity. To mitigate this threat, the exercises were designed considering framework domains based on the real world.

**Conclusion Validity:**

- Measure reliability: it refers to metrics used to measuring the development effort. To mitigate this threat, we 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 this threat, 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.

## VI. Related Works

In this section some works related to the F3T and the F3 approach are presented.

Amatriain and Arumi [28] proposed a method for the development of a framework and its DSL through iterative and incremental activities. In this method, the framework has its domain defined from a set of applications and it is implemented by applying a series of refactorings in the source-code of 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 framework domain. However, the advantage of the F3 approach is the fact that the design and the implementation of the frameworks are supported by the F3 patterns and it is automatized by the F3T.

Oliveira et al. [29] presented the ReuseTool, which assists framework reuse by manipulating UML diagrams. The ReuseTool is based in the Reuse Description Language (RDL), a language created by these authors to facilitate the description of framework instantiation processes. Framework hot spots can be registered in the ReuseTool with the use of the RDL. In order to instantiate the framework, application models can be created based on the framework description. Application source-code is generated from these models. Thus, the RDL works as a meta language that registers framework hot spots and the ReuseTool provides a more friendly interface for developers to develop applications reusing the frameworks. In comparison, the F3T supports framework development through domain modeling and application development through framework DSML.

Pure::variants [30] is a tool that supports the development of applications by modeling domain features (Feature Diagram) and the components that implement these features (Family Diagram). Then the applications are developed by selecting a set of features of the domain. Pure::variants generates only application source-code, maintaining all domain artifacts in model-level. Besides, this tool has private license and its free version (Community) has limitations in its functionality. In comparison, the F3T is free, uses only one type of domain model (F3 model) and generates frameworks as domain artifacts. Moreover, the frameworks developed with he support of the F3T can be reused in the development of applications with or without the support of the F3T.

## VII. Conclusions

The F3T support framework development and reuse through code generating from models. This tool provides an F3 model editor for developers to define the features of the framework domain. Then, framework source-code and DSML can be generated from the F3 models. Framework DSML can be installed in the F3T to allow developers to model and to generate the source-code of applications that reuses the framework. The F3T is a free software available at: http://www.dc.ufscar.br/∼matheus_viana.

The F3T was created to semi-automatize the applying of the F3 approach. In this approach, domain features are defined in F3 models in order to separate the elements of the framework from the complexities to develop them. F3 models incorporate elements and relationships from feature models and properties and operations from metamodels.

Framework source-code is generated based on patterns that are solutions to design and implement domain features defined in F3 models. A DSML is generated along with the source-code and includes all features of the framework domain and in the models created with it developers can insert application specifications to configure framework hot spots. Thus, the F3T supports both Domain Engineering and Application Engineering, improving their productivity and the quality of the outcome frameworks and applications. The F3T can be used to help the construction of software product lines, providing an environment to model domains and create frameworks to be used as core assets for application development.

The experiment presented in this paper has shown that, besides the gain of efficiency, the F3T reduces the complexities surrounding framework development, because, by using this tool, developers are more concerned about defining framework features in a graphical model. All code units that compose these features, provide flexibility to the framework and allows it to be instantiated in several applications are properly generated by the F3T.

The current version of the F3T generates only the model layer of the frameworks and applications. In future works we intend to include the generation of a complete multi-portable Model-View-Controller architecture.

## Acknowledgment

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

## References

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

[2] M. Abi-Antoun. Making Frameworks Work: a Project Retrospective. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications*, 2007.

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

[4] JBoss Community. Hibernate. http://www.hibernate.org, Jan 2013.

[5] Spring Source Community. Spring Framework. http://www.springsource.org/spring-framework, Jan 2013.

[6] S. D. Kim, S. H. Chang, and C. W. Chang. A Systematic Method to Instantiate Core Assets in Product Line Engineering. In *11th Asia-Pacific Conference on Software Engineering*, pages 92–98, Nov 2004.

[7] David M. Weiss and Chi Tau Robert Lai. *Software Product Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, 1999.

[8] D. Parsons, A. Rashid, A. Speck, and A. Telea. A Framework for Object Oriented Frameworks Design. In *Technology of Object-Oriented Languages and Systems*, pages 141–151, Jul 1999.

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

[10] 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*, Jun 2012.

[11] M. Viana, R. Durelli, R. Penteado, and A. do Prado. F3: From Features to Frameworks. In *15th International Conference on Enterprise Information Systems*, Jul 2013.

[12] Sajjan G. Shiva and Lubna Abou Shala. Software Reuse: Research and Practice. In *Fourth International Conference on Information Technology*, pages 603–609, Apr 2007.

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

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

[15] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science, 7th edition, 2009.

[16] A. Sarasa-Cabezuelo, B. Temprado-Battad, D. Rodrguez-Cerezo, and J. L. Sierra. Building XML-Driven Application Generators with Compiler Construction. *Computer Science and Information Systems*, 9(2):485–504, 2012.

[17] S. Lolong and A.I. Kistijantoro. Domain Specific Language (DSL) Development for Desktop-Based Database Application Generator. In *International Conference on Electrical Engineering and Informatics (ICEEI)*, pages 1–6, Jul 2011.

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

[19] I. Liem and Y. Nugroho. An Application Generator Framelet. In *9th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'08)*, pages 794–799, Aug 2008.

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

[21] 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.

[22] 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, Nov 1990.

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

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

[25] OMG. OMG's MetaObject Facility. http://www.omg.org/mof, Jan 2013.

[26] The Eclipse Foundation. Eclipse Modeling Project, Jan 2013.

[27] 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.

[28] 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.

[29] T. C. Oliveira, P. Alencar, and D. Cowan. Design Patterns in Object-Oriented Frameworks. *ReuseTool: An Extensible Tool Support for Object-Oriented Framework Reuse*, 84(12):2234–2252, Dec 2011.

[30] Pure Systems. Pure::Variants. http://www.pure-systems.com/pure_variants.49.0.html, Fev 2013.