Viana *et al. Journal of Software Engineering Research and Development* (2015) 3:4
DOI 10.1186/s40411-015-0017-y

Journal of Software Engineering Research and Development
a SpringerOpen Journal

RESEARCH Open Access

F3T: a tool to support the F3 approach on the development and reuse of frameworks

Matheus C Viana[1,2*], Rosângela AD Penteado[2], Antônio F do Prado[2] and Rafael S Durelli[3]

*Correspondence:
matheus_viana@dc.ufscar.br
[1]Federal Institute of Sao Paulo, Campus Sao Carlos, Rod. Washington Luis, km 235, Block AT6, 13565-905 Sao Carlos, Brazil
[2]Department of Computing, Federal University of Sao Carlos, Rod. Washington Luis, km 235, 13565-905 Sao Carlos, Brazil
Full list of author information is available at the end of the article

Abstract

Background: Frameworks are used to enhance the quality of applications and the productivity of the development process, since applications may be designed and implemented by reusing framework classes. However, frameworks are hard to develop, learn and reuse, due to their adaptive nature. From Feature to Frameworks (F3) is an approach that supports framework development in two steps: Domain Modeling, to model domain features of the framework; and Framework Construction, to develop framework source-code based on the modeled domain and on patterns provided by this approach.

Methods: In this article, it is presented the From Features to Framework Tool (F3T), which supports the use of the F3 approach on framework development.

Results: This tool provides an editor for domain modeling and generates framework source-code according to the patterns of the F3 approach. In addition, F3T also generates a Domain-Specific Modeling Language that allows the modeling of applications and the generation of their source-code. F3T has been evaluated in two experiments and the results are presented in this article.

Conclusions: F3T facilitates framework development and reuse by omitting implementation complexities and performing code generation.

Keywords: Reuse; Framework; Domain; Feature; Generator

1 Introduction

Frameworks are reusable software composed of abstract classes that implement the basic functionality of a domain. When an application is developed through framework reuse, the functionality provided by the framework classes is complemented with the application requirements. As this application has not been not developed from scratch, the time spent in its development was reduced and its quality was improved (Abi-Antoun 2007; Johnson 1997; Stanojevic et al. 2011).

Frameworks are often used to implement common application requirements, such as persistence Hibernate 2013 and user interface (Spring Framework 2013). Besides, frameworks are also used as core assets in the development of closely related applications in a Software Product Line (SPL) (Kim et al. 2004; Weiss and Lai 1999). The common features of the SPL domain are implemented in the framework and applications implement these features by reusing framework classes.

Despite those advantages, frameworks are hard to develop since their classes must be abstract enough to be reused by applications that are unknown beforehand. Therefore, it is necessary to define two things (Parsons et al. 1999; Weiss and Lai 1999): 1) the domain of applications the framework is able to instantiate; and 2) how the framework accesses application-specific classes. Frameworks are also hard to reuse because they have a steep learning curve (Srinivasan 1999). They may be very complex, composed by a large number of classes and modules that even developers who are conversant with it may make mistakes.

In a previous paper, an approach for building Domain-Specific Modeling Languages (DSML) was proposed to support framework reuse (Viana et al. 2012). A DSML can be built by identifying framework features and the information required to instantiate them. Then, the application source-code can be generated from models created with this DSML. Experiments have shown that DSMLs protect developers from framework complexities and reduce the time spent on framework instantiation.

In another paper, the From Features to Framework (F3) approach was presented. It aims to reduce framework development complexities (Viana et al. 2013). In this approach, framework domain is defined in an F3 model, as described in Section 2.1. A set of patterns guides the developer when designing and implementing a white box framework according to its domain. Besides showing how developers shall proceed, the F3 patterns systematizes framework development process, allowing it to be automatized by a tool.

In this article, the From Features to Framework Tool (F3T) is presented as a plug-in for the Eclipse IDE that supports the F3 approach on framework development and reuse. By using this tool, developers can define a domain in an F3 model. Then, framework source-code and DSML are generated from this model. This DSML can be used to model applications and to generate their source-code, which reuses the framework previously generated.

Two experiments have also been carried out in order to evaluate F3T. In the first one, it was analyzed whether F3T facilitates framework development or not and, in the second presents a comparison between F3T and Pure::variants (Pure::Variants 2013).

The remainder of this article is organized as follows: background concepts are discussed in Section 2; the F3 approach is described in Section 2.1; F3T is presented in Section 3; two experiments to evaluate F3T are presented in Section 4; the related works are discussed in Section 5; and the conclusions and future work are presented in Section 6.

2 Background

The basic concepts applied to F3T and its approach are presented in this section. Reuse is a practice that aims: to reduce the time spent in the development process, since software was not developed from scratch; and to increase the software quality, since reusable practices, models or code were previously tested (Shiva and Shala 2007). Patterns, frameworks, generators and domain engineering are common ways to apply reuse to software development (Frakes and Kang 2005).

Patterns are successful solutions that may be reapplied to different contexts (Johnson 1997). They provide reuse of experience, which helps developers to solve recurrent problems (Fowler 2003). A pattern documentation mainly contains its name, the context it may be applied, the problem it intends 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 (Pressman 2009).

Frameworks act like skeletons that can be instantiated to implement applications (Johnson 1997). Their classes embody an abstract design that provides solutions for application domains (Srinivasan 1999). Applications are connected to a framework by reusing its classes. According to the way a framework is reused, it is classified as: white box, when its classes need to be extended; black box, when it works like a set of components; and gray box, when it is reused on the two previous ways (Abi-Antoun 2007).

Despite the advantages frameworks offer, they are more complex to develop than applications (Kirk et al. 2007), since 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 programming 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 the framework will be able to instantiate, the features that compose this domain and the rules that constrain these features (Stanojevic et al. 2011). Some solutions to this issue propose adaptations of the traditional software development process (Amatriain and Arumi P 2011) or the refactoring of applications that share common features in order to implement a framework (Xu and Butler 2006).

The reuse of frameworks provides higher quality and efficiency to software development process. However, frameworks require the developer to have detailed knowledge about their internal structure and their hot spots so that they can be properly used (Abi-Antoun 2007; Srinivasan 1999). Some solutions have been applied in order to facilitate the difficulties in reusing frameworks, such as manuals, cookbooks and pattern languages. These solutions may guide the application developer through framework instantiation. However, the task of identifying and configuring the hot spots according to the application requirements is still executed by the developer and relies on his/her skills and knowledge (Antkiewicz et al. 2009).

Generators are tools that transform an artifact into another (Lolong and Kistijantoro 2011; Sarasa-Cabezuelo et al. 2012. There are many types of generators. As frameworks, generators are also related to domains, although some are configurable and may change their domain (Liem and Nugroho 2008). 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 (Bayer et al. 1999; Gomaa 2004; Jezequel 2012; Kang et al. 1990; Lee et al. 2002). Domain features are defined in feature models. Features may be mandatory or optional, have variations and require or exclude other features. The feature that represents the purpose of the domain is added to the root and a top-down approach is applied to add the other features.

Domains may also be modeled with metamodel languages, which are used to create Domain-Specific Modeling Languages (DSML). Metamodels, as defined in the MetaObject Facility (MOF) (OMG's MetaObject Facility 2013), are similar to class models, which makes them more appropriate to developers accustomed to UML. While in feature models, only features and their constraints are defined, metaclasses in the metamodels may contain attributes and operations. On the other hand, feature models can define

dependencies between features, while metamodels depend on declarative languages to do so (Gronback 2009).

2.1 F3 Approach

From Features to Framework (F3) is a Domain Engineering approach that aims to develop domain specific frameworks. It has two steps: 1) Domain Modeling, to define and model a domain of applications; and 2) Framework Construction, to design and implement a framework for the domain defined in the previous step.

In Domain Modeling step, the domain is defined as an extended version of feature model, called F3 model. This extended version is used in the F3 approach since feature models are too abstract to contain enough information for framework development and also because metamodels depend on other languages to define dependencies and constraints. F3 models incorporate characteristics from both feature models and metamodels. As in conventional feature models, features in the F3 models may also be arranged in a tree-view. The root feature is the main one and it is placed on top of the tree. However, F3 models do not necessarily form a tree, since a feature may 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 may 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 may be associated to each instance of the source feature. Valid values of the maximum multiplicity are: `1` (simple), for a single feature instance; `*` (multiple), for a list of instances of a single feature subclass; and `**` (variant), for a list of instances of different subclasses of a feature.
- **Generalization**: relationship that indicates 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 feature A requires feature B, an application that contains feature A also has to include feature B; and `excludes`, when feature A excludes feature B, no application may include both features.

Framework Construction step has a white box framework as output. The F3 approach defines patterns that assist developers to design and implement frameworks from F3 models as well as to know the code units that shall be created to implement domain functionality and its variability. F3 patterns address problems that range from the creation of classes that represent features to the definition of framework interface. Some of the F3 patterns are presented in Table 1.

Besides indicating the code units that shall be created to implement framework functionality, F3 patterns also determine how the framework may be reused by the applications. For instance, some patterns suggest implementing abstract operations that allow the framework to access application-specific information. In addition, F3 patterns make the process of framework development systematical, allowing it to be automatized. Thus,

Table 1 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.

F3T was created to automatize the use of the F3 approach, enhancing the processes of framework development.

3 From features to frameworks tool

Although all advantages provided by the F3 approach make framework development easier, it still requires developers to model domains and apply the F3 patterns properly to implement the frameworks. For instance, a developer could forget to apply an F3 pattern during Framework Construction step. Thus, computational support should be provided during code implementation, in order to improve productivity and reduce the occurrence of human errors.

F3T assists developers to apply the F3 approach in the development of white box frameworks and in the reuse of these frameworks through their DSML (Viana et al. 2012, 2013). In order to use the tool it is necessary to follow the steps of this approach. The tool provides an editor to F3 models and generates framework source-code based on the F3 patterns. The role of the framework DSML is to facilitate framework instantiation.

F3T is a plug-in for Eclipse IDE, so developers may make use of F3T resources, such as domain modeling, framework construction, application modeling through framework DSML, application construction and other resources provided by Eclipse IDE. F3T is composed of three modules, as seen in Figure 1: 1) Domain Module; 2) Framework Module; and 3) Application Module. Each module represents a resource that developers may use to create the artifacts required by the F3 approach to develop and reuse frameworks.

3.1 Domain module

Domain Module (DM) is an editor for developers to create an F3 model with the domain features, as illustrated in Figure 1. This module has been developed with the support of the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF) (Gronback 2009). EMF was used to create the metamodel that defines the elements, relationships and rules of F3 models, as described in the Section 2.1. This metamodel is shown

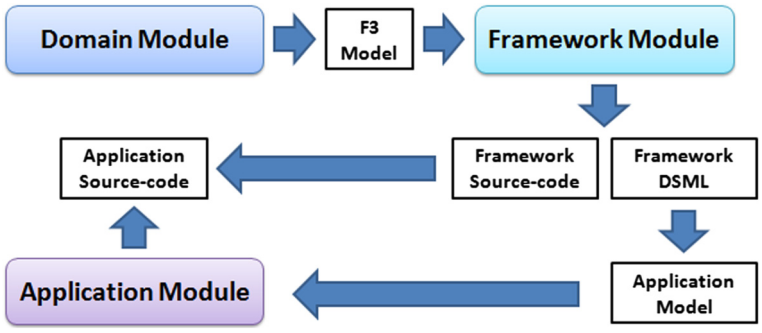

Figure 1 Modules of F3T. This figure shows the modules of F3T: Domain Module, Framework Module and Application Module. Each module is composed by a set of Eclipse IDE plug-ins.

in Figure 2. From this metamodel, 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 F3 models. This graphical notation may also be seen as the View layer of the F3 model editor, as it defines how features and relationships are graphically represented. Then, GMF generates the source-code of the graphical notation. The F3 model editor is shown in Figure 3 and it is composed of three parts: 1) editor panel, which is used to create and visualize F3 models; 2) menu bar, which provides the F3 elements and relationships to be included in the models; and 3)properties panel, which displays all properties of a selected element or relationship in the model.

For instance, the F3 model for the domain of rental and trade transactions is shown in Figure 3. This domain deals with rental and trade transactions of resources to destination parties. The root feature is a generic `ResourceTransaction`, specialized by the features `ResourceTrade` and `ResourceRental`. The `DestinationParty` feature represents the party that requires the transaction. For instance, a destination party may be treated as a customer in an application. `DestinationParty` is optional by default. However, once `ResourceRental` is used, `DestinationParty` is mandatory. That is why there is a `requires` relationship between these features. The `Resource` feature represents the resources that may be traded or rented. One or more resources participate in a transaction, so the `TransactionItem` feature was defined to represent

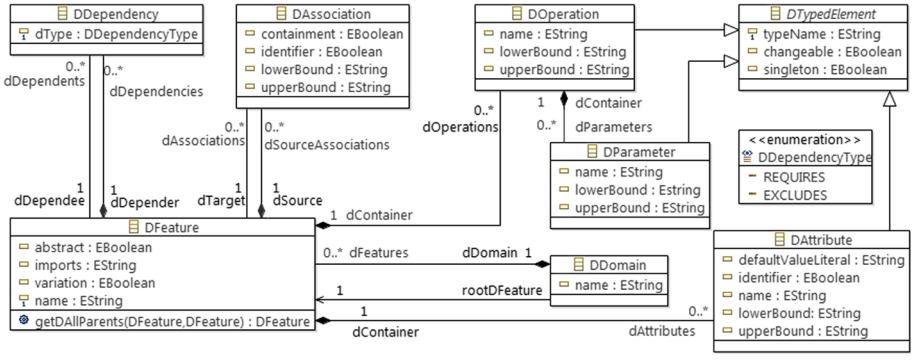

Figure 2 Metamodel containing elements, relationships and rules of F3 models. This figure shows the metamodel whose metaclasses and relationships define how F3 models can be created.

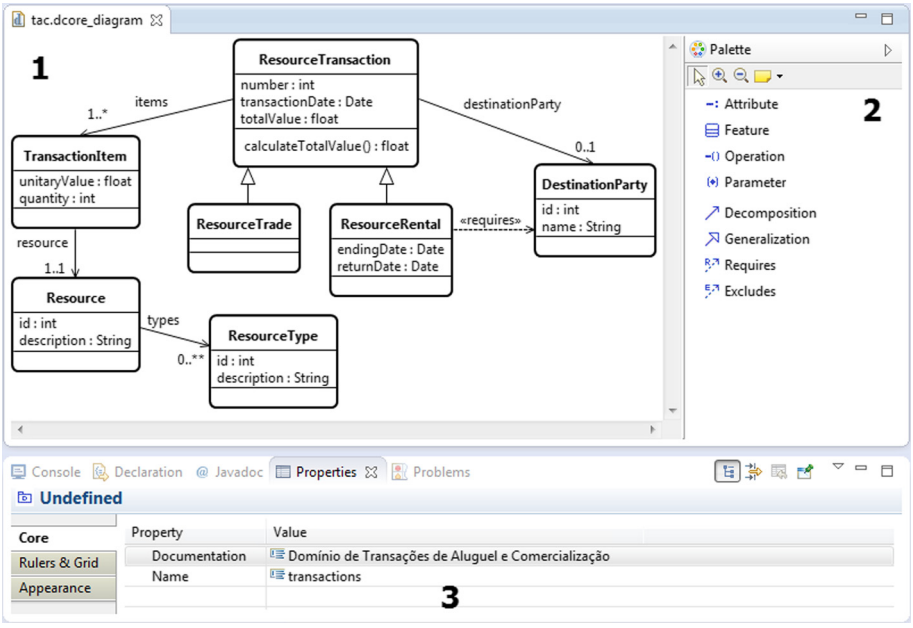

Figure 3 F3 model for the domain of trade and rental transactions. This figure shows an example of F3 model created by using F3T.

that. A resource can have classifications, so the `ResourceType` feature can be extended many times in an application to define any type of classification, such as category, genre, location and so on.

3.2 Framework module

Despite their graphical notation, F3 models actually are XML files, which makes them more accessible to other tools, such as generators. Therefore, Framework Module (FM) has been designed as a Model-to-Text (M2T) generator that transforms an F3 model into framework source-code and DSML.

FM has been developed with the support of Java Emitter Templates (JET) in the Eclipse IDE (The Eclipse Foundation Eclipse Modeling Project). JET contains a framework that works as 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 input information and text is a fixed content inserted in the output independently of the input. The Java files originated from the JET templates reuse the JET framework to compose a domain-specific generator. Thus, FM depend on the JET plug-in to work.

The hierarchy of the FM templates is shown in Figure 4. These templates are organized in two groups: one related to framework source-code (DSC); and the other related to framework DSML. Both groups are invoked from the Main template. The DSC template invokes the templates that originate the framework classes. Part of the JET template that generates Java classes in the framework source-code from the features found in the F3 models are 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>
```

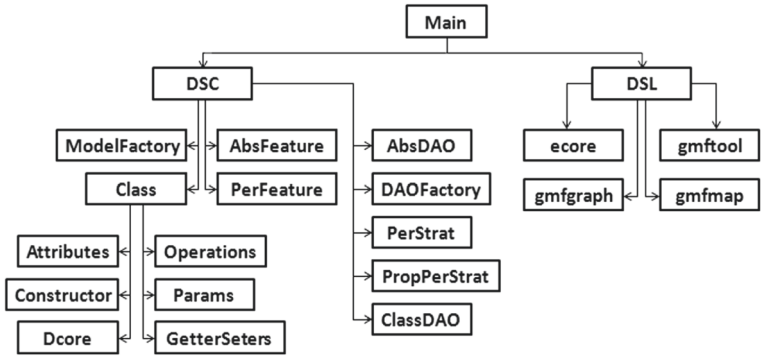

Figure 4 Hierarchy of the templates which compose FM. This figure shows how the FM templates are hierarchically organized.

```
   <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> { ... }
```

Framework classes are generated according to F3 patterns (Viana et al. 2013). For instance, FM generates a class for each feature found in an F3 model. The reason is that, since a white-box framework is generated, these classes are directly extended by the application classes. These classes also contain the attributes and operations, as specified in its correspondent feature. Besides the classes that represent features, others are also generated to provide code flexibility and to implement non-functional requirements, such as the `DObject` class that is, directly or indirectly, extended by all feature classes in order to provide data persistence functionality to them. Generalization relationships result in inheritances, whereas decomposition relationships result in associations between the involved classes. Additional operations are included in framework classes to implement feature variations and constraints defined in F3 models. For instance, according to the *Variant Decomposition* F3 pattern, the `getResourceTypeClasses` operation was included in the code of the `Resource` class (Figure 3) so that the framework recognizes which classes implement the `ResourceType` feature in applications. Part of the `Resource` class code is presented as follows:

```
public abstract class Resource extends DObject {

  private int id;

  private Sting name;

  private List<ResourceType> types;

  public abstract Class<?>[] getResourceTypeClasses();
```

The DSML template invokes a set of templates that originate EMF/GMF models to the framework DSML. An example of these models is illustrated in Figure 5a, which was generated from the F3 model shown in Figure 3. Then, DSML source-code is be generated by EMF/GMF in three steps: 1) using the EMF generator from the *genmodel* file (Figure 5a); 2) using the GMF generator from the *gmfmap* file (Figure 5b); and 3) using the GMF generator from the *gmfgen* file (Figure 5c). After that, the DSML will be composed of 5 plug-in projects in the Eclipse IDE. The projects that contain the framework source-code and the DSML plug-ins for the domain of trade and rental transactions are shown in Figure 5d. The Java project in which the framework source-code was generated is identified by the domain name and the suffix ".framework". The others are DSML plug-ins.

3.3 Application module

Application Module (AM) has also been developed with the support of JET. It generates application source-code from an application model created from a framework DSML. The AM templates 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, 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 6. This application is intended for a small store that trades products. Therefore, the `ProductTrade` application class extends the `ResourceTrade` framework class. `TradeItem` represents the products in a trade transaction, so it extends `TransactionItem`. `Product` is the resource in this application. Each product in the store may be classified by `Category` and by `Manufacturer`. As this store keeps no register of its customers, the `DestinationParty` feature was not used in this application.

The application source-code is generated in the source folder of the project, in which the application model is. AM generates a class for each feature instantiated in the application model. Since the framework is a white box, the application classes extend the framework classes indicated by the stereotypes in the model. It is expected that most of the class attributes requested by the application requirements have already 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:

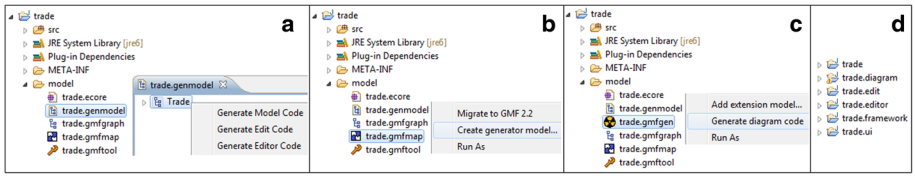

Figure 5 Generation of the DSML plugins. This figure shows how DSML plug-ins are generated by F3T: **a)** EMF/GMF models generated by F3T to create DSML plug-ins; **b)** Gmfmap model is used to create generator model; **c)** Generator model (gmfgen) is used to create the DSML graphical editor plug-in; **d)** all DSML plug-ins and the framework source-code project (trade.framework).

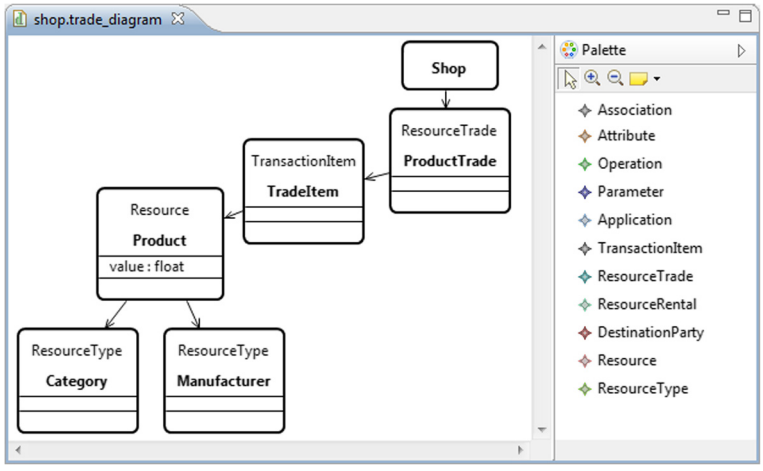

Figure 6 Application model created with the framework DSML. This Figure shows the Shop application model created with the DSML of trade and rental transactions.

```
public class Product extends Resource {

  private float value;

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

4 Evaluation

In this section, two experiments are presented: one to evaluate the advantages of using F3T to develop frameworks; and the other to compare F3T with Pure::variants (Pure::Variants 2013). These experiments were conducted by following all steps described by Wohlin et al. 2000.

4.1 Experiment 1

In the first experiment, the use of F3T for framework development has been evaluated, since framework reuse supported by DSML was evaluated in a previous paper (Viana et al. 2012). This experiment was defined as: (*i*) **analysis** of F3T, described in Section 3; (*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.

4.1.1 Planning

The experiment was planned to answer two research questions:

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

All of the subjects had to develop two frameworks applying the F3 approach. One of them should be done manually and the other by using F3T. In order to answer the first

question, the time spent to develop each framework was measured. To answer the second question, the frameworks developed by the subjects have been analyzed and the problems found in their source-code have been identified. The planning phase was divided into seven parts, as follows:

Context selection The subjects of this experiment were 26 MSc and PhD students of Computer Science. All of them had prior experience in software development, Java programming, patterns and framework reuse.

Formulation of hypotheses The experiment questions have been formalized as follows:

- **RQ_1, Null Hypothesis, $H1_0$:** Considering the F3 approach, there is no significant time difference when developing frameworks with the support of F3T or manually. Thus, F3T does not reduce the time spent to develop frameworks. This hypothesis may be formalized as: **$H1_0$:** $\tau_{F3T} = \tau_{manual}$
- **RQ_1, Alternative Hypothesis 1, $H1_1$:** The time spent to develop frameworks by applying the F3 approach with the support of F3T is significantly lower than when applying the F3 approach manually. Thus, F3T reduces the time spent to develop frameworks. This hypothesis is formalized as: **$H1_1$:** $\tau_{F3T} < \tau_{manual}$
- **RQ_1, Alternative Hypothesis 2, $H1_2$:** The time spent to develop frameworks by applying the F3 approach with the support of F3T is significantly greater than when applying the F3 approach manually. Thus, F3T does not reduce the time spent to develop frameworks. This hypothesis is formalized as: **$H1_1$:** $\tau_{F3T} > \tau_{manual}$
- **RQ_2, Null Hypothesis, $H2_0$:** There is no significant difference in the number of problems found in the frameworks developed manually or using F3T. Thus, F3T does not reduce the number of mistakes made by the subjects during framework development. This hypothesis is formalized as: **$H2_0$:** $\rho_{F3T} = \rho_{manual}$
- **RQ_2, Alternative Hypothesis 1, $H2_1$:** The number of problems found in the frameworks developed with the support of F3T is significantly lower than when applying the F3 approach manually. Thus, F3T reduces the number of mistakes made by the subjects during framework development. This hypothesis is formalized as: **$H2_1$:** $\rho_{F3T} < \rho_{manual}$
- **RQ_2, Alternative Hypothesis 2, $H2_2$:** The number of problems found in the frameworks developed by with the support of F3T is significantly greater than when applying the F3 approach manually. Thus, F3T increases the number of mistakes made by the subjects during framework development. This hypothesis is formalized as: **$H2_1$:** $\rho_{F3T} > \rho_{manual}$

Variable 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:

- **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.

Selection of subjects The subjects were selected through a non probabilist approach by convenience. Therefore, the probability of all population elements belong to the same sample is unknown.

Experiment design The subjects were grouped in two groups of 13 subjects:

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

We have chosen to use groups in order to reduce the effects of the subjects experience. In order to measure this, the subjects answered a form 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.

To use the F3 approach manually, the subjects had to model the domain by using the class diagram of Astah and apply the F3 patterns on this model to implement framework source-code. On the other hand, to develop the frameworks by using F3T, the subjects had to create the F3 model of the domain using the editor provided by the tool and generate the framework source-code.

Design types The design type of this experiment was **one factor with two treatments paired** (Wohlin et al. 2000). The **factor** in this experiment is the way how the F3 approach was used to develop a framework and the **treatments** are the support of F3T in contrast with the manual development.

Instrumentation All the necessary materials used during the execution of this experiment were given to the subjects beforehand. These materials consisted of forms for collecting experiment data, domain requirements, F3 approach documentation and test units code. In the end of the experiment, all subjects received a questionnaire, in which they should report about the F3 approach and F3T.

4.2 Operation

The operation phase was divided into two parts, Preparation and Execution, as described in the subsections Preparation and Execution.

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

Execution Initially, the subjects signed a consent form and then answered the characterization form. After this, they watched a presentation about frameworks, which included

the description of some popular examples and their hot spots. The subjects were also trained on how to develop frameworks using the F3 approach with and without the support of F3T.

After the training, the pilot experiment was executed. The subjects were split into two groups considering the results of the characterization forms. The 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 one, except that the applications were different, however equivalent. Beforehand, all subjects were given ample time to read the approach and to ask questions on the experimental process. This could affect the experiment validation, therefore, this data was only used to balance the groups.

During the experiment execution, the subjects who had to develop the framework manually used a class diagram of the Astah Community to create the F3 model of the domain. Then, they used the documentation of the F3 patterns to apply them and implement the framework based on the features they defined in the F3 model. On the other hand, the subjects who had access to F3T had to create the F3 model and generate the framework source-code by using the resources of this tool. In both cases, after finishing the implementation, the subjects had to pause the chronometer and ran the test units to verify whether their code was correct or not. In case of success, their task was done. Otherwise, they had to continue measuring the time and fix the problems of the framework.

4.3 Analysis of data

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

Descriptive statistics The time each subject spent to develop a framework and the number of problems found in the outcome frameworks are shown in Table 2. From this table, one may notice that the subjects spent more time to develop the frameworks when they were doing it manually (M) then when using F3T, 72.5% against 27.5%, respectively. This result was expected, since 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 the framework implementation and the effort to fix the problems found in the frameworks, while in the framework development supported by F3T it was due to domain modeling. The dispersion of time spent by the subjects are also represented graphically in a boxplot on the left side of Figure 7.

In Table 2, the four types of problems that were analyzed in the outcome frameworks are presented: (*i*) incoherence, (*ii*) structure, (*iii*) bad smells, (*iv*) interface.

The incoherence problem indicates that, during the experiment, the subjects did not model the domain of the framework as expected. In other words, the subjects did not develop the frameworks with the correct domain features and constraints (mandatory, optional, and alternative features). As the ability 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 F3T.

The structure problem indicates that the subjects did not implement the frameworks properly during the experiment. For instance, either they implemented classes with no constructor and with incorrect relationships or they forgot to declare the classes as

Table 2 Data obtained from framework development applying two approaches: Manual(M) and F3T

Subject	Time spent		Number of problems									
			Incoherence		Structure		Bad smells		Interface		Total	
	M	F3T	M	F3T	M	F3T	M	F3T	M	F3T	M	F3T
S1	72	26	1	1	0	0	1	0	0	0	2	1
S2	74	32	1	2	1	0	1	0	1	0	4	2
S3	83	31	3	3	1	1	1	0	1	0	6	4
S4	78	29	1	1	2	0	1	0	2	0	6	1
S5	67	26	0	1	0	0	0	0	0	0	0	1
S6	81	32	4	3	3	1	2	0	3	0	12	4
S7	79	24	3	3	1	0	1	0	0	1	5	4
S8	73	23	1	0	0	0	0	0	0	0	1	0
S9	79	26	1	2	2	0	1	0	1	0	5	2
S10	69	27	2	1	0	0	1	0	1	0	4	1
S11	71	29	1	2	1	0	0	0	1	0	3	2
S12	83	31	3	2	1	0	3	0	1	0	8	2
S13	74	26	1	0	1	0	1	0	0	0	3	0
S14	72	29	1	2	0	0	0	0	1	0	2	2
S15	76	31	3	4	2	0	1	0	2	0	8	4
S16	68	26	1	1	0	0	0	0	0	0	1	1
S17	80	33	5	4	4	1	3	0	4	1	16	6
S18	75	27	1	1	2	0	2	0	2	0	7	1
S19	73	29	0	1	1	0	1	0	0	0	2	1
S20	81	32	2	1	3	0	2	0	1	0	8	1
S21	86	35	3	4	3	0	3	0	2	0	11	4
S22	76	28	2	1	1	0	1	0	1	0	5	1
S23	83	31	4	3	3	1	2	0	3	1	12	5
S24	79	28	2	2	1	0	1	0	1	0	5	2
S25	77	29	3	2	1	0	1	0	1	0	6	2
S26	78	33	2	3	2	0	1	0	3	0	8	3
AVG	76.42	28.96	1.96	1.92	1.38	0.15	1.19	0	1.23	0.12	5.77	2.19
%	72.52	27.48	50.50	49.50	90	10	100	0	91.43	8.57	72.46	27.54

abstract. This kind of problem occurred when the subjects did not properly follow the instructions provided by the F3 patterns. From Table 2, one may observe that F3T helped the subjects to develop frameworks with less structure problems, i.e., 10% in opposition to 90%.

The bad smell problem indicates design weaknesses that do not affect functionality, however it makes the frameworks harder to maintain. In the experiment, this kind of problem occurred when the subjects forgot to apply some of the F3 patterns related to the organization of the framework classes, such as the *Modular Hierarchy* F3 pattern. In Table 2, one can notice F3T made a design with higher quality than the manual approach, i.e, 0% against 100%, because F3T automatically identified which patterns should be applied from the F3 models.

The interface problem indicates absence of getter/setter operations and also of operations that allow the framework to access the application-specific classes. Frequently, this kind of problem is a consequence of structure problems, hence the results of these two problems are quite similar. As shown in Table 2, the subjects designed a better framework interface when using F3T, i.e., 8.6% against 91.4%.

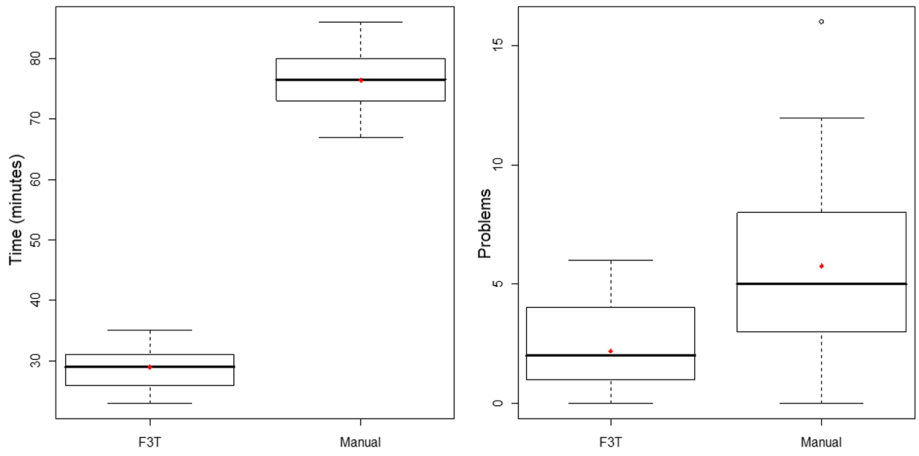

Figure 7 Dispersion of the total time and number of problems. This figure show a box plot created from experiment 1 data.

In the last two columns of Table 2, one may observe that F3T reduced the total number of problems found in the frameworks developed by the subjects. This is also graphically represented in the boxplot on the right side of Figure 7.

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. All the tests were applied in the experiment data by using the software Action (Portal Action 2015). Since some statistical tests are applicable only in case 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. When the p-value of this Shaphiro-Wilk test is greater than 0.05, it means that data is normally distributed and we can apply the Paired T-Test to verify which hypothesis is valid. Otherwise, the Paired Wilcoxon Signed Rank test is used. More details about these tests are found in the site of Action (Portal Action 2015). The tests have been carried out as follows:

- **Time**: The Shapiro-Wilk test has been applied to the experiment data that represents the time spent by each subject to develop a framework manually or using F3T, as shown in Table 2. Considering that the p-values were 0.8780 (Manual) and 0.6002 (F3T), the Shapiro-Wilk test confirmed that the time spent in framework development is normally distributed, as illustrated in the Q-Q charts (a) and (b) in Figure 8. Then, the Paired T-Test was applied to verify which hypothesis is accepted for RQ_1. The Paired T-Test resulted in a p-value 1,11E-28. It means that the chance of $H1_0$ to be accepted is lesser than 5% and the average values in columns "Time Spent" in manual development and F3T in Table 2 are valid. Therefore, when the F3 approach is applied, one spends less time developing a framework by using F3T than doing it manually.
- **Problems**: Similarly, the Shapiro-Wilk test has been applied to the experiment data shown in the last two columns of Table 2, which represent the total number of problems found in the outcome frameworks. The resulting p-values were 0.1522 in

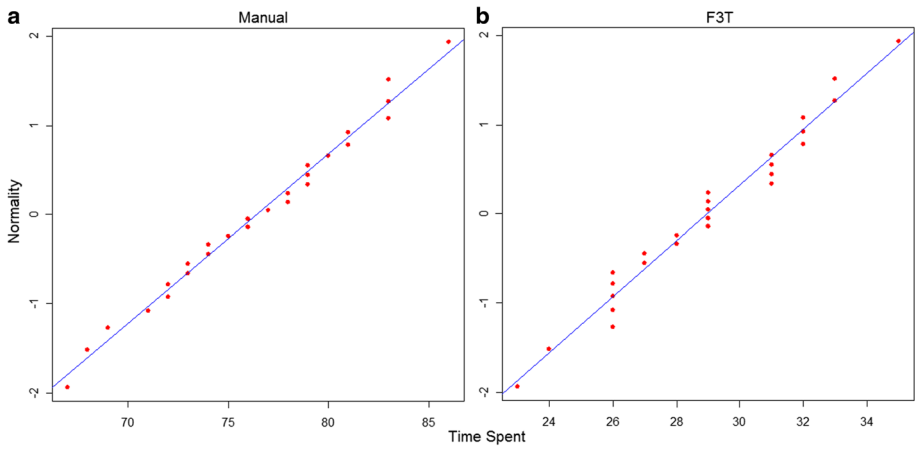

Figure 8 QQ-Plots of the normality tests applied to the time spent in framework development when **(a)** the manual approach was applied and when **(b)** F3T was used. This figure shows 2 QQ Plots illustrating the result of normality tests applied to the time spent on framework development.

manual development and 0.0075 with F3T, as represented by the QQ-Plots in Figure 9. Thus, the data related to manual development is normally distributed, but the same does not happen with the data related to F3T. Therefore, the Wilcoxon Signed Rank test was applied to verify whether the null hypothesis of RQ_2 may be accepted or not. As a result, the p-value was 2.87E-05, which means that the chance of $H2_0$ to be accepted is lesser than 5% and the average values in column "Number of Problems - Total" in Table 2 are valid. It reinforces that F3T reduces the number of problems found in the outcome frameworks.

Opinion of the subjects The opinion of the subjects has been in order to evaluate the impact of using F3T. After the experiment operation, all subjects received a questionnaire, in which they could report their perception about applying the F3 approach manually or supported by F3T. As shown in Figure 10, when asked if they encountered difficulties in

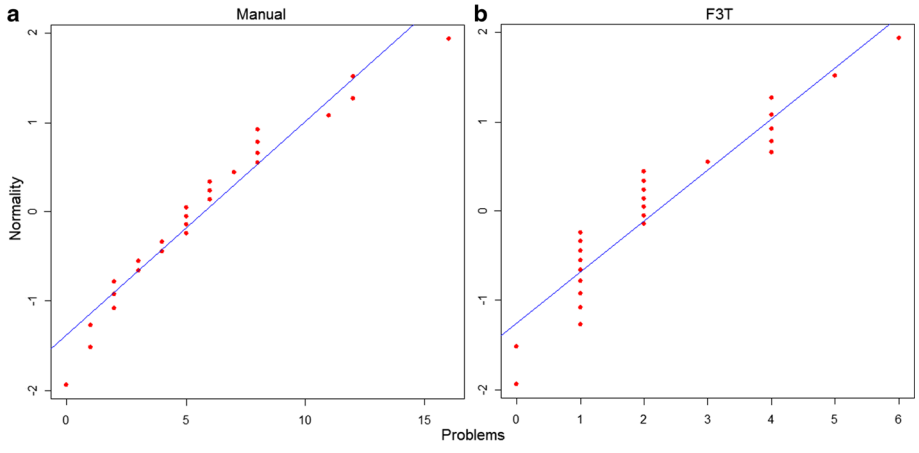

Figure 9 QQ-Plots of the normality tests applied to the number of problems on the outcome frameworks when **(a)** the manual approach was applied and when **(b)** F3T was used. This figure shows 2 QQ-Plots illustrating the result of normality tests applied to the number of problems on the outcome frameworks.

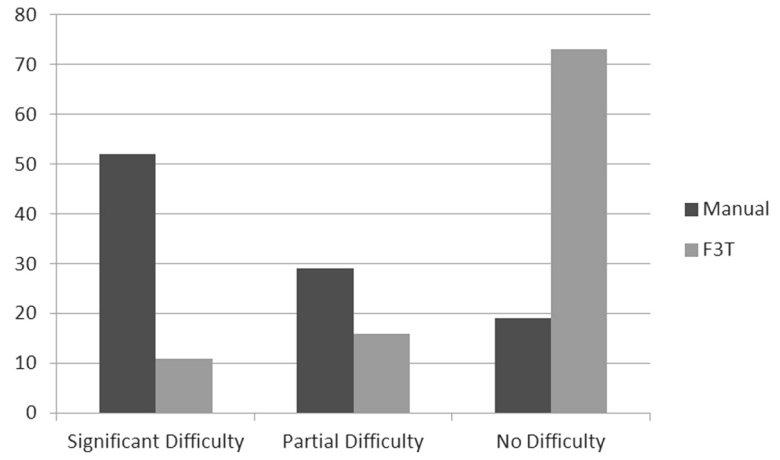

Figure 10 Level of difficulty of the subjects. This figure shows a chart illustrating the difficulties of the students while developing the frameworks of the experiment.

framework development 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 concerning the use of F3T, 73% subjects reported having no difficulty, 16% mentioned partial difficulty and only 11% had significant difficulty.

The subjects also specified the kind of difficulties they faced during framework development. The most common difficulties pointed out in the manual task 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 F3 models. In contrast, the most common difficulties faced by the use of F3T were: 1) lack of practice with the tool; and 2) some actions in the tool interface, for instance, there are many steps in order to open the F3 model editor. The subjects said that the F3 patterns helped them to identify the necessary structures to implement the frameworks manually. They also said F3T automatized the tasks of identifying which F3 patterns should be used as well as of implementing the framework source-code. Thus, they could focus on domain modeling.

4.4 Experiment 2

In the second experiment, F3T has been compared to Pure::variants in a software product line environment. Pure::variants (Pure::Variants 2013) is a tool that supports the development of application variants. From the application source-code, Pure::variants generates a feature model, that specifies the features found in the application, as well as a family model, which defines the components that implement these features. Then, applications, variants of the base one, may be generated by selecting a subset of features of the feature model. Although these tools are based in different approaches, they have been chosen since both of them can be used to generate several applications in a domain. However, due to the differences of the tools, only the time to perform Domain and Application Engineering steps was taken into consideration in this experiment.

Therefore, this second experiment was defined as: (*i*) **analyse** F3T; (*ii*) **for the purpose of** evaluation; (*iii*) **with respect to** time spent; (*iv*) **from the point of view of** the developer; and (*v*) **in the context of** MSc and undergraduate Computer Science students.

4.4.1 Planning

The experiment aimed to answer the following research question:

- **RQ_1: Which tool allows a more efficient development of Domain and Application Engineering steps in terms of time?**

In this experiment, all the subjects had to carry out Domain Engineering (DE) and Application Engineering (AE) steps of software product lines by using F3T and Pure::variants. In order to answer the research question, the time spent in carrying out DE and AE steps with each tool were measured. The planning phase was divided into seven parts, which are described as follows.

Context selection The subjects of this experiment were 32 MSc and undergraduate students of Computer Science. All of them had prior experience in software development, Java programming, patterns and framework reuse.

Formulation of hypotheses The experiment questions have been defined as follows:

- **RQ_1, Null Hypothesis, $H1_0$:** There is no significant difference in the time spent carrying out DE and AE steps using F3T or Pure::variants. Thus, using F3T is not more efficient than Pure::variants. This hypothesis may be formalized as: $\mathbf{H_0}$**:** $\tau_{F3T} = \tau_{pure}$
- **RQ_1, Alternative Hypothesis 1, $H1_1$:** The time spent to carry out DE and AE steps by using F3T is significantly lower than by using Pure::variants. Thus, it is more efficient to use F3T than Pure::variants. This hypothesis may be formalized as: $\mathbf{H_1}$**:** $\tau_{F3T} < \tau_{pure}$
- **RQ_1, Alternative Hypothesis 2, $H1_2$:** The time spent to carry out DE and AE steps by using F3T is significantly greater than by using Pure::variants. Thus, it is more efficient to use Pure::variants than F3T. This hypothesis may be formalized as: $\mathbf{H_1}$**:** $\tau_{F3T} > \tau_{pure}$

Variable selection The dependent variables of this experiment were:

- **time spent to carry out DE and AE steps**;
- **usability, related to the opinion of the subjects**.

The independent variables were as follows:

- **Domain**: Each subject had to develop two software product lines: in the first, they had to develop the artifacts for the domain of trade and rental transactions (DE1) and a library application (AE1); and in the second, they had to develop the artifacts for the domain of medical care (DE2) and a veterinary clinic application (AE2). These domains had 10 features each and the applications presented a similar complexity level.
- **Development Environment**: Eclipse 4.2.1 with F3T, Pure::variants evaluation version 3.2.
- **Technologies**: Java version 6.

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

Experiment design The subjects were grouped in two groups of 16 subjects:

- **Group 1**, development of DE1/AE1 using F3T and development of DE2/AE2 using Pure::variants;
- **Group 2**, development of DE1/AE1 using Pure::variants and development of DE2/AE2 using F3T;

As in Experiment 1, the subjects were grouped in groups according to the level of experience, which was measured from a form in which they had to inform the software development techniques they had already used. This form was given to the subjects one week before the pilot experiment. The goal of this pilot experiment was to ensure that the experiment environment and materials were adequate and that the tasks could be properly executed.

While using F3T in DE, the subjects should create an F3 model based on the domain requirements, generate the framework source-code and DSML plug-ins from this model and install the DSML plug-ins in the Eclipse IDE. In AE, the subjects should create an application model by using the framework DSML according to the application requirements.

Pure::variants creates software product line artifacts from the source-code of a base application. Therefore, while using this tool, the first thing the subjects had to do was to implement this base application. Then, in DE they should generate the feature and the architectural models of the domain and define its variant environment. Finally, in AE the subjects should select the domain featured according to the application requirements and generate this application source-code.

Design types The design type of this experiment was **one factor with two treatments paired** (Wohlin et al. 2000). The **factor** was the tool used to carry out DE and AE steps and the **treatments** were the tool used in this experiment: F3T and Pure::variants.

Instrumentation All the necessary materials to assist the subjects during the execution of this experiment were given to the subjects beforehand, including tool manuals and domain/application requirements and models. They also received a form for collecting experiment data, in which the subjects have to report the time spent to carry out DE and AE steps and their opinion about the tools. All the subjects were also trained in the use of F3T and Pure::variants.

4.5 Operation

The operation phase was divided into two parts, Preparation and Execution, as described in the subsections Preparation and Execution.

Preparation Firstly, the subjects received a characterization form, containing questions on their knowledge about Java programming, Eclipse IDE, patterns, frameworks, F3T and

Pure::variants. A pilot experiment had also been previously performed so that the subjects could get more used to the experiment activities.

Execution In the first activity, the subjects should carry out DE for trade and rental transactions domain and develop an application for a library in AE. The subjects in Group 1 used F3T while the ones in Group 2 used Pure::variants. Each subject measured the time spent using the tools to carry out DE and AE steps. The subjects who were using Pure::Variants also measured the time spent on the implementation of the base application. The second activity was carried out in a similar way. However, in this activity, the subjects of the two groups should work with the domain of medical care in DE and with the veterinary clinic application in AE. Besides, the subjects in Group 1 were using Pure::variants and those in Group 2 were using F3T.

4.6 Analysis of data

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

Descriptive statistics The time spent by each subject to develop the base Application Implementation (AI), Domain Engineering (DE) and Application Engineering (AE) is shown in Table 3. Considering DE and AE, one can observe that the subjects spent 45.27% of the total time using Pure::variants and 54.73% using F3T. The main reason for this is that most of the models in Pure::variants are generated from the base application code. Only in EA, the subjects had to decide which domain features should be included in the outcome application. While using F3T, they had to interpret domain requirements in order to create an F3 model and interpret the application requirements to create its model by using the framework DSML. The time spent by the subjects is also represented graphically in the boxplot in Figure 11.

Testing the hypotheses As in Experiment 1, the software Action has been used to verify whether it was possible to reject the null hypotheses in favor of the alternative hypotheses (Portal Action 2015). The Experiment 2 tests have been carried out as follows:

- **Time**: The Shapiro-Wilk test has been applied to the data that represents the time spent with each tool, as shown in columns “DE+AE" of Pure::variants and F3T in Table 3. Considering that the p-values were 0.6133 for Pure::variants and 0.4990 for F3T, the test confirmed that the time spent to carry out DE and AE steps was normally distributed, as shown in the Q-Q charts (a) and (b) in Figure 12. Then, the Paired T-Test was applied to verify which hypothesis would be accepted for RQ1. The Paired T-Test resulted in a p-value 1.19E-3. It means that the chance of H1_0 to be accepted is lower than 5% and the average values in columns “DE+AE" of Pure::variants and F3T in Table 3 are valid. Therefore, the time spent to carry out a software product line is lower with Pure:variants than with F3T.

Positive and negative characteristics of each tool After each experiment, the subjects had to write their opinion on Pure::variants and F3T and highlight positive and negatives

Table 3 Data obtained from using Pure::variants and F3T

Domain	Time spent (min.)								
	Pure::variants					F3T			
	Sub.	AI	DE	AE	DAE	Sub.	DE	AE	DAE
Medical	S1	35	19	10	29	S17	39	19	58
Care	S2	13	32	16	48	S18	20	8	28
	S3	31	30	8	38	S19	26	21	47
	S4	32	18	14	32	S20	18	10	28
	S5	15	15	9	24	S21	31	19	50
	S6	18	20	12	32	S22	36	26	62
	S7	28	31	11	42	S23	23	14	37
	S8	30	27	14	41	S24	35	22	57
	S9	17	26	18	44	S25	36	24	60
	S10	14	30	9	39	S26	32	18	50
	S11	17	23	13	36	S27	28	12	40
	S12	10	18	11	29	S28	23	16	39
	S13	19	22	10	32	S29	26	13	39
	S14	26	25	11	36	S30	31	22	53
	S15	22	21	15	36	S31	21	15	36
	S16	19	29	13	42	S32	24	19	43
Trans.	S17	21	30	10	40	S1	33	18	51
	S18	9	14	11	25	S2	25	20	45
	S19	14	24	12	36	S3	38	18	56
	S20	6	33	10	43	S4	30	12	42
	S21	15	20	8	28	S5	30	9	39
	S22	25	28	15	43	S6	26	13	39
	S23	16	24	12	36	S7	31	20	51
	S24	22	33	16	49	S8	28	17	45
	S25	30	18	10	28	S9	20	11	31
	S26	31	32	17	49	S10	21	10	31
	S27	16	30	9	39	S11	24	13	37
	S28	32	29	15	44	S12	24	14	38
	S29	18	23	10	33	S13	27	16	43
	S30	19	25	12	37	S14	35	22	57
	S31	13	19	13	32	S15	29	17	46
	S32	17	22	14	36	S16	32	14	46
AVG		20.31	24.69	12.13	36.81		28.19	16.31	44.50
%		35.56	43.22	21.23	45.27		63.34	36.66	54.73

characteristics of each tool. Most of the subjects mentioned that the models in both Pure::variants and F3T demand too many steps to be created. The reason is that both tools are based on the Eclipse IDE, in which every file/model is created through a set menu items and wizard forms.

The subjects also mentioned that the main positive characteristic of Pure: variants is that all DE models are generated from the base application source-code, whereas in F3T, an F3 model is created manually. However, each of the Pure::variants models is generated in a sequence of 4-5 steps. Therefore, the subjects argued that they needed to consult the manual tool very often to know how to proceed.

About F3T, the subjects mentioned the following positive characteristics: an application source-code is not needed to develop ED models; the number of models to be

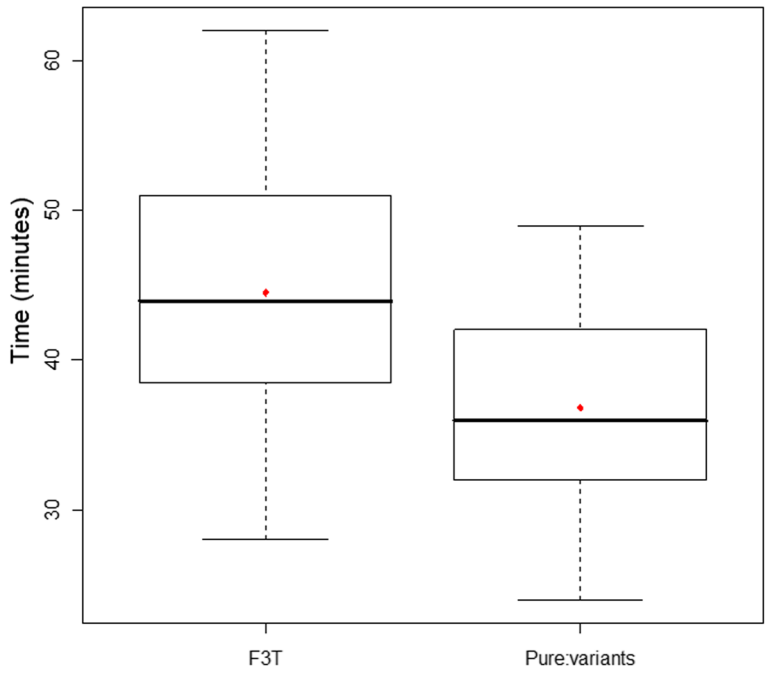

Figure 11 Dispersion of sum the time spent in DE and AE. This figure shows a box plot created from sum the time spent in DE and AE in experiment 2 data.

created in F3T ED (1 model) is smaller if compared with Pure::variants (4 models); the DE step in F3T results in a framework that may be reused in any Java environment besides the Eclipse IDE; and it is easier to customize the outcome applications in AE. The subjects also mentioned that they had some difficulties in creating F3 models. However, this may be attributed to two factors: 1) their lack of experience with domain modeling; and 2) the difficulty in interpreting the domain requirements used in the experiment.

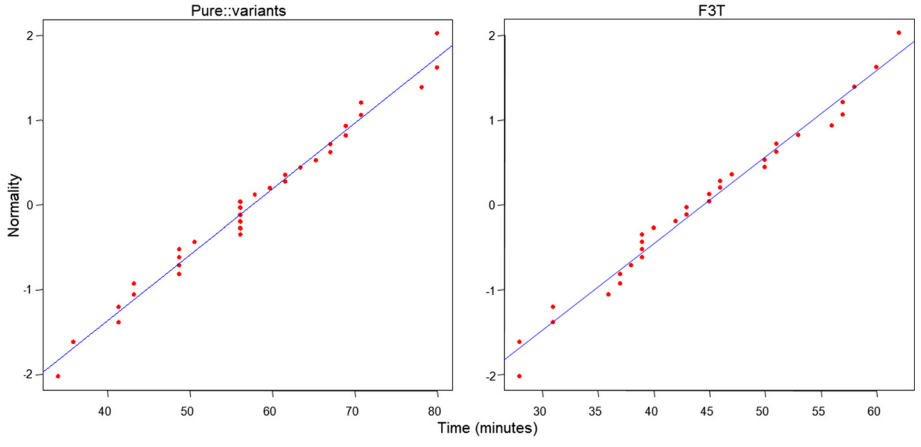

Figure 12 QQ-Plots of the normality tests applied to the time spent in DE and AE. This figure shows a QQ-Plot illustrating the result of normality tests applied to the time spent in DE+AE.

4.7 Threats to the validity of the experiments

Internal validity:

- Experience level of the subjects: the subjects had different levels of knowledge and it could affect the data collected. To mitigate this threat, the subjects were divided into two balanced groups considering their level of knowledge and the groups were rebalanced considering the preliminary results. Moreover, all the subjects had prior experience in application development by reusing frameworks, but no experience in developing frameworks. Thus, the subjects were trained with common framework implementation techniques and how to use the F3 approach and F3T.
- Productivity under evaluation: this might have influenced the experiment results since subjects tend to think they are being evaluated by experiment results. In order to mitigate this, the subjects were told 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 that F3T was supposed to ease framework development, which reflects one of our hypothesis. These issues could affect the data collected and cause the experiment to be less impartial. In order to keep impartiality, the participants were asked to keep a steady pace during the whole study.

External validity:

- Interaction between configuration and treatment: there is a chance 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 real world.

Conclusion validity:

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

5 Related work

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

Amatriain and Arumi 2011 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 may also be applied in iterative and incremental activities, starting from a small domain and then adding features. Applications may also be used to facilitate the identification of the features of the framework domain. However, the advantage of the F3 approach is that the design and the implementation of the frameworks are supported by the F3 patterns and it is automatized by F3T.

Oliveira et al. 2011 presented the ReuseTool, which assists framework reuse by manipulating UML diagrams. The ReuseTool is based on the Reuse Description Language (RDL), a language created by these authors to facilitate the description of framework instantiation processes. Framework hot spots may be registered in the ReuseTool with the use of RDL. In order to instantiate the framework, application models may be created based on the framework description. Application source-code is generated from these models. Thus, RDL works as a meta language that registers framework hot spots and the ReuseTool provides a more friendly interface to develop applications by reusing the frameworks. In comparison, F3T supports framework development through domain modeling and application development through framework DSML.

Common Variability Language (CVL) is an Object Manager Group (OMG) standardization used for specifying and resolving domain variability (Rouille et al. 2012). Like F3 models in F3T, CVL is an extended feature model. However, CVL uses a mechanism similar to OCL to implement domain constraints. In comparison, F3 models define domain constraints through relationships and properties. Moreover, since F3 models and F3T focus on framework development, the features in this kind of model may contain attributes and operations.

6 Conclusions

F3T supported framework development and reuse through the generation of code from models. This tool provided an F3 model editor for developers to define the features of the framework domain. Then, framework source-code and DSML may be generated from F3 models. Framework DSML may be installed in the F3T to allow developers to model and to generate the source-code of applications that reuse the framework.

F3T has been created to semi-automatize the application of the F3 approach. Here, domain features are defined in F3 models in order to separate the framework elements from the complexities involved when developing 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 propose solutions to design and implement domain features defined in F3 models. A DSML is generated along with the framework source-code and it includes all domain features. Developers may create models by mapping application requirements to these features to configure framework hot spots. Thus, F3T supports both Domain Engineering and Application Engineering, which improves productivity and the quality of the outcome frameworks and applications. Apart from this, F3T may be used to help the construction of software production lines. It provides an environment to model domains as well as to create frameworks that may be used as core assets for application development.

In addition to the advantages of the F3 approach, F3T improves the framework efficiency and the application development, since the implementation steps of the approach

are executed through code generation. It also results in better quality artifacts, due to the model validations provided by the tool and to the fact that code generated is less likely to contain defects.

The first experiment has shown that, besides the gain of efficiency, F3T reduces the complexities surrounding framework development, since, by using this tool, developers are more concerned about defining framework features in a graphical model. F3T generates classes that provide flexibility to the framework and allows it to be instantiated in several applications.

In the second experiment, F3T was compared to Pure::variants. Each tool applies different approaches and artifacts to carry out DE and AE and both tools present pros and cons. In conclusion, F3T is more useful when there is no previous artifact and when the domain architecture is needed as a software artifact, such as a framework. Pure::variants is more useful when variations of an existing application need to be developed.

The current version of F3T only generates the model and persistent layers of frameworks and applications. As future work, it is intended to include the generation of a complete multi-portable Model-View-Controller architecture.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
MV developed the F3 approach and F3T and wrote most of the manuscript and carried out the experiments. RD participated in the planning of the experiments and performed the statistical analysis. RD was also responsible to write about the experiments in the manuscript. RP helped to execute the experiments. RP and AP helped in the design of the manuscript. All authors read and approved the final manuscript.

Acknowledgments
We would like to thank CAPES and FAPESP for sponsoring our research. We also want to thank all students who participated in the experiments presented in this article.

Author details
[1]Federal Institute of Sao Paulo, Campus Sao Carlos, Rod. Washington Luis, km 235, Block AT6, 13565-905 Sao Carlos, Brazil. [2]Department of Computing, Federal University of Sao Carlos, Rod. Washington Luis, km 235, 13565-905 Sao Carlos, Brazil. [3]Institute of Mathematical and Computer Sciences, University of Sao Paulo, Av. Trabalhador Sao Carlense, 400, 13566-590, Sao Carlos, Brazil.

Received: 3 June 2014 Accepted: 28 March 2015
Published online: 22 April 2015

References
Abi-Antoun M (2007) Making Frameworks Work: a Project Retrospective. In: ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications. ACM, New York, NY, USA. pp 1004–1018
Amatriain X, Arumi P (2011) Frameworks Generate Domain-Specific Languages: A Case Study in the Multimedia Domain. IEEE Trans Softw Eng 37(4):544–558
Antkiewicz M, Czarnecki K, Stephan M (2009) Engineering of Framework-Specific Modeling Languages. IEEE Trans Softw Eng 35(6):795–824
Bayer J, Flege O, Knauber P, Laqua R, Muthig D, Schmid K, Widen T, DeBaud J (1999) PuLSE: a Methodology to Develop Software Product Lines. In: Symposium on Software Reusability. ACM, New York, NY, USA. pp 122–131
Fowler M (2003) Patterns. IEEE Software 20(2):56–57
Frakes W, Kang K (2005) Software Reuse Research: Status and Future. IEEE Trans Softw Eng 31(7):529–536
Gomaa H (2004) Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, Boston, MA, USA. p 736
Gronback RC (2009) Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley, New York. p 736
Hibernate. http://www.hibernate.org
Jezequel JM (2012) Model-Driven Engineering for Software Product Lines. ISRN Softw Eng 2012:1–24
Johnson RE (1997) Frameworks = (Components + Patterns). Commun ACM 40(10):39–42
Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS (1990) Feature-Oriented Domain Analysis (FODA): Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, Pittsburgh, Pennsylvania, USA
Kim SD, Chang SH, Chang CW (2004) A Systematic Method to Instantiate Core Assets in Product Line Engineering. In: 11th Asia-Pacific Conference on Software Engineering. IEEE Computer Society, Los Alamitos, CA, USA. pp 92–98

Kirk D, Roper M, Wood M (2007) Identifying and Addressing Problems in Object-Oriented Framework Reuse. Empir Softw Eng 12(3):243–274
Lee K, Kang KC, Lee J (2002) Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: 7th International Conference on Software Reuse: Methods, Techniques and Tools. Springer, London, UK. pp 62–77
Liem I, Nugroho Y (2008) An Application Generator Framelet. In: 9th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'08). pp 794–799
Lolong S, Kistijantoro AI (2011) Domain Specific Language (DSL) Development for Desktop-Based Database Application Generator. In: International Conference on Electrical Engineering and Informatics (ICEEI). IEEE Computer Society, Los Alamitos, CA, USA. pp 1–6
Oliveira TC, Alencar P, Cowan D (2011) Design Patterns in Object-Oriented Frameworks. ReuseTool: An Extensible Tool Support for Object-Oriented Framework Reuse 84(12):2234–2252
OMG's MetaObject Facility. http://www.omg.org/mof
Parsons D, Rashid A, Speck A, Telea A (1999) IEEE Computer Society. In: Technology of Object-Oriented Languages and Systems, Los Alamitos, CA, USA. pp 141–151
Portal Action. http://www.portalaction.com.br/en
Pressman RS (2009) Software Engineering: A Practitioner's Approach, 7th edn. McGraw-Hill Science, New York. p 928
Pure::Variants. http://www.pure-systems.com/pure_variants.49.0.html
Rouille E, Combemale B, Barais O, Touzet D, Jezequel J-M (2012) Leveraging CVL to Manage Variability in Software Process Lines. In: 19th Asia-Pacific Software Engineering Conference (APSEC) Vol. 1. pp 148–157
Sarasa-Cabezuelo A, Temprado-Battad B, Rodríguez-Cerezo D, Sierra JL (2012) Building XML-Driven Application Generators with Compiler Construction. Comput Sci Inform Syst 9(2):485–504
Shiva SG, Shala LA (2007) IEEE Computer Society. In: Fourth International Conference on Information Technology, Los Alamitos, CA, USA. pp 603–609
Spring Framework. http://www.springsource.org/spring-framework
Srinivasan S (1999) Design patterns in object-oriented frameworks. ACM Comput 32(2):24–32
Stanojevic V, Vlajic S, Milic M, Ognjanovic M (2011) Guidelines for Framework Development Process. In: 7th Central and Eastern European Software Engineering Conference. IEEE Computer Society, Los Alamitos, CA, USA. pp 1–9
The Eclipse Foundation Eclipse Modeling Project. http://www.eclipse.org/modeling/
Viana M, Penteado R, do Prado A (2012) Generating Applications: Framework Reuse Supported by Domain-Specific Modeling Languages. In: 14th International Conference on Enterprise Information Systems. doi:10.5220/0003990000050014
Viana M, Durelli R, Penteado R, do Prado A (2013) F3: From Features to Frameworks. In: 15th International Conference on Enterprise Information Systems. doi:10.5220/000441770110011
Weiss DM, Lai CTR (1999) Software Product Line Engineering: A Family-Based Software Development Process. Addison-Wesley, New York. p 448
Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Norwell, MA, USA
Xu L, Butler G (2006) Cascaded Refactoring for Framework Development and Evolution. ASWEC, Australian Software Engineering Conference. pp 319-330, doi:10.1109/ASWEC.2006.19