A Qualitative and Quantitative Analysis on Metadata-based Frameworks Usage

Eduardo Guerra[1], Clovis Fernandes[1]

[1] Aeronautical Institute of Technology (ITA) - Praça Marechal Eduardo Gomes, 50
CEP 12.228-900 - São José dos Campos – SP, Brazil
guerraem@gmail.com, clovistf@uol.com.br

Abstract. The usage of metadata-based frameworks are becoming popular for some kinds of software, such as web and enterprise applications. They use domain-specific metadata, usually defined as annotations or in XML documents, to adapt its behavior to each application class. Despite of their increasingly usage, there are not a study that evaluated the consequences of their usage to the application. The present work presents the result of an experiment which aimed to compare the development of similar applications created: (a) without frameworks; (b) with a traditional framework; (c) with a metadata-based framework. As a result, it uses metrics and a qualitative evaluation to assess the benefits and drawbacks in the use of this kind of framework.

Keywords: framework, metadata, metric, experiment, software design, software architecture.

1 Introduction

A framework is a set of classes that supports reuses at larger granularity. It defines an object-oriented abstract design for a particular kind of application which does not enable only source code reuse, but also design reuse [1]. Frameworks can enable functionality extension by providing abstract methods in its classes which should be implemented with application-specific behavior. Other alternative is to provide methods to configure instances for which part of the functionality is delegated. This instances can be application-specific or from framework's built-in classes [2]. In the present work, the frameworks that use those approaches based on inheritance or composition to enable its extension are called Traditional Frameworks.

The framework structures has evolved and recent ones make use of introspection [3] [4] to access at runtime the application classes metadata, like their superclasses, methods and attributes. As a result, it eliminates the need for the application classes to be coupled with the framework abstract classes and interfaces. The framework can, for instance, search in the class structure for the right method to invoke. The use of this technique provides more flexibility to the application, since the framework reads dynamically the classes structure allowing them to evolve more easily [5].

For some frameworks, however, once they need a domain-specific or application-specific metadata to customize their behavior, the information found in the class definition is not enough [6]. This kind of information can be represented and defined in code annotations [7], external sources, like XML files and databases, or implicitly by using naming conventions [8] [9]. In the present work this kind of framework is named Metadata-based Framework, which can be defined as the one that process their logic based on the metadata from the classes whose instances they are working with [10].

Before this study, not much information about the benefits of developing and using metadata-based frameworks were found in the literacture, but some development communities are increasingly adopting them as standards. Consistent with that, there are many recent frameworks developed and APIs defined using this approach, such as Hibernate [11] , EJB 3 [12], Struts 2 [13] and JAXB [14].

The main goal of this study is to evaluate the benefits and drawbacks in the usage of metadata-based framework. In order to do that, an experiment was conducted aiming to compare the uses of traditional and metadata-based frameworks to create the same functionality. The experiment carried out during an undergraduate course of advanced topics in object-orientation. The same application was developed by the students using three different approaches: (a) without frameworks; (b) with a framework that do not use metadata; (c) with a metadata-based framework. Students also answered a questionnaire to register their impressions on the experience.

Metrics and visualization techniques were applied to the source code of the tree applications in order to evaluate the design of each one. Issues like coupling, amount of code and complexity were considered in the analysis. Other more subjective issues like the facility to use the framework, easiness to evolve the application and the development time were addressed in the questionnaire and in observations during the implementation. The evaluation resulted in a set of consequences, both positives and negatives, concerning the use of metadata-based frameworks.

2 Metadata-based Frameworks

Metadata is an overloaded term in computer science and can be interpreted differently according to the context. In the context of object-oriented programming, metadata is information about the program structure itself such as classes, methods and attributes. A class, for example, has intrinsic metadata like its name, its superclass, its interfaces, its methods and its attributes. In metadata-based frameworks, the developer also must define some additional application-specific or domain-specific metadata.

The metadata consumed by the framework can be defined in different ways [9]. One althernative is to define them in external sources, like XML files and databases. Another possibility that is becoming popular in the software community is the use of code annotations, that is supported by some programming languages like Java [7] and C# [15]. Using this technique the developer can add custom metadata elements directly into the class source code. The use of code annotations is also called attribute-oriented programing [6] [16].

Metadata-based frameworks can be defined as frameworks that process their logic based on the metadata of the classes whose instances they are working with [10]. The use of metadata changes the way frameworks are build and how they are used by software developers. In metadata-based frameworks there are some variable points in the framework processing which are determined by class metadata. Reflective algorithms in some cases cannot be applied due to more specific variations for some classes. In this context, metadata can be used to configure specific behaviors when the framework is working with that class.

From developer's perspective in the use of this kind of framework, there is a stronger interaction with metadata configuration than with method invocation or class specialization. That makes the number of method invocations in framework classes smaller and localized.

The following are examples of how metadata-based frameworks and APIs can be used in different contexts: Hibernate [11] is a framework for object-relational mapping; SwingBean [19] is a framework that generates forms and tables in Java Swing based on class structure and metadata; EJB 3 [12] is an standard Java EE API for enterprise development that uses metadata to configure concerns such as access control and transaction management; and JColtrane [20] is a XML parsing framework based on SAX which uses annotations for conditions to define when handler's methods should be invoked.

3 Experiment Description

One of the great difficulties to evaluate the benefits and drawbacks of the use of a metadata-based framework is the nonexistence of comparison basis. In other words, it is hard to find two frameworks with the same purpose, one build using traditional methods and other based on metadata, that can both be used for comparison. Four different scenarios abstracted from existent frameworks were used as reference for the case studies in this experiment.

The experiment main goal can be defined as: *"To create traditional and metadata-based frameworks for the same purpose and applications with the same behavior using them, aiming to generate a comparison basis and identify the benefits and drawbacks of the metadata-based approach."*

According to [21] classification, a Controlled Experimentation Method is used in the experiment, that can also be classified as a Synthetic Environment Experiment, since it is performed on an academic setting and simulates the creation of a piece of functionality in an application. Based on the taxonomy presented by [22], the experiment is designed to present cause-effect results, to be performed by novices and on an in-vitro environment. A similar approach to evaluate implementation approaches can be found in [23] and [24].

The following are the requirements that were considered in the elaboration of the experiment to reach its objectives: (a) two frameworks for the same purpose must be created using the traditional and the metadata-based approach; (b) solutions with the same external specified behavior must be developed using both frameworks and also without their use; (c) solutions with the same specified behavior to be compared must

not be developed by the same persons; (d) neither the frameworks nor the solutions that implements the specified behavior must be developed by the present work's authors; (e) the development time of the solutions must be measured; (f) the design of the solutions must be assessed; and (g) the participants development experience to create the solutions must be assessed.

The experiment took place in *Advanced Topics in Object Orientation* discipline, which is an optional class in the fifth year of the *Computer Engineering* graduation coarse in the *Aeronautical Institute of Technology*. It was executed in the second semester of 2009, when twelve students attended the course. They were divided in four groups of tree students, one for each scenario.

3.1 Experiment Stages

The development of the frameworks and the implementations using them, were divided in five distinct stages. The class was divided in four groups of tree students, each responsible for the development of the first solution and both versions of the framework for one scenario. The other solutions using the frameworks were developed by other distinct groups. Fig. 1 illustrates graphically the experiment stages and the software products generated in each one.

In **Stage 1**, students received a specification with the solution requirements which must be implemented by them. This solution could be composed by more than one class and the specification also defined how an external class should interact with them to use its functionality. Based on this defined protocol, the students also had to create an automated test suite to verify if the solution implements the specified requirements. They must not use frameworks and they should measure the development time for the solution's and test suite's implementation. This stage was executed at the student's home, and was carried on the beginning of the coarse when only testing techniques and basic concepts had been taught.

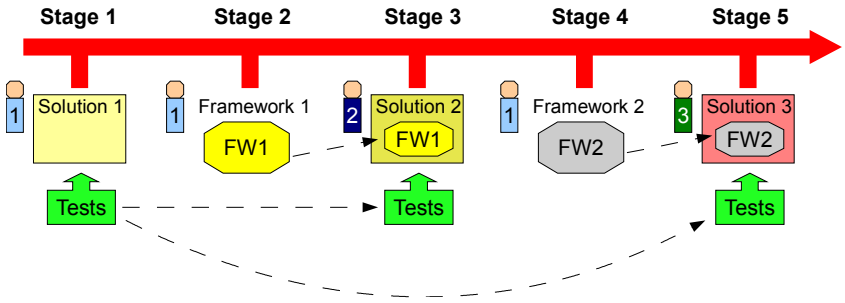

Fig. 1. Experiment stages representation.

In **Stage 2**, the same group should develop a framework using traditional and reflection techniques in order to make easier the creation of similar solutions of the one developed in Stage 1. The framework scope and functionalities were specified in

a document and used by the students as a reference. They also should provide documentation for the framework usage. The only restriction was that the framework must not use annotations or information defined externally. Nothing was said about reflection and code conventions in the specification, but their use were allowed. This stage was executed at their home as well, and happened at the middle of the coarse when reflection and object-oriented design techniques had already been taught.

In **Stage 3**, the group that worked in a different scenario should implement a solution similar to the one developed on Stage 1, but using the framework developed in Stage 2. The students received the same requirements specification used in Stage 1 and the framework documentation developed in Stage 2 to be read just before the implementation beginning. Students also received a project configured with: the tests; the framework in the classpath; and empty classes needed for the test to compile. The solution was considered implemented when the test suite executed successfully. This stage was executed in the class lab and the implementation time was measured by the present work's author.

In **Stage 4**, the same group that implemented stages 1 and 2 should develop a metadata-based framework for the same purpose of the one developed in Stage 2. The framework scope, functionalities and the role of metadata were specified in a document and used by the students as a reference. They also should provide documentation for the framework usage as they did in Stage 2. This stage was executed at their home, and was accomplish at the end of the coarse when annotations and techniques to develop frameworks with metadata had already been taught.

In **Stage 5**, a group that has not been worked already in the scenario should implement a solution similar to the ones developed in stages 1 and 3, but using the framework developed in Stage 4. The conditions were similar to Stage 3. This stage was executed in the class lab and the implementation time was measured by the present work's author.

The solutions developed by the students are not complete applications, but pieces of code that could potentially be a part of an architectural layer. They focused on a single concern, which is the domain aimed by the frameworks to be developed. Therefore, the specifications define simple problems to be implemented nevertheless with a lot of constraints to simulate the requirements of a real application.

The groups were free to use any strategy learned in the classes for the first framework implementation, since it fulfills the objective to make easier the development of that kind of solution. For the second framework, the specification defined more clearly for which purpose the metadata would be used and students did not have much freedom on their choices.

Each case study aimed to use respectively the following scenarios for the metadata usage: (a) mapping between command-line parameters and a class that represents them; (b) validation of method parameters and constraints; (c) stock market event handling; and (d) automatic generation of an HTML form.. Each scenario focus on a different architectural pattern documented for this kind of framework [25]. This scenario diversity is important to enable the assess of not only the general characteristics, but also the specific ones from each distinct metadata usage.

After the implementation, the design of each solution was measured and evaluated using the metrics and the visualization techniques, such as polymetric views [26] and class blueprints [27]. The development experience was assessed through time

measurements, the present work's author's observations and the students answers to a questionnaire whose questions are presented in the next subsection.

3.2 Questionnaire

The students answered a questionnaire at the end of Stage 5 to evaluate their experience and impressions on the development of each solution.

The students filled a table answering for each solution development the following three questions: (1) how easy was the development of the application's source code; (2) how easy was the use of the framework; and (3) how easy would be to change the code to add new features. Each question could be answered as one of the following alternatives: (a) very easy; (b) easy; (c) average; (d) hard; and (e) very hard. The students also wrote a free text about their experience to justify his answers, which was also considered in the analysis.

To compare quantitatively the characteristics of each solution, the answers were turned into numbers using numeric scale from one to five respectively from very easy to very hard. This quantitative analysis was complemented with a qualitative one, using the author's observations and the answers to the open question.

3.3 Limitations

Despite the fact that the experiment achieved all the requirements, it still has some inherent limitations that can influence the implementations, which are used for the measurements and the conclusions. The following are the identified limitations that can have influence in the implementations: (a) the students learned about object-oriented design and frameworks from the beginning to the end of the Advanced Topics in Object Orientation discipline, which might have some influence in the source code quality; (b) the students did not have a wide experience in framework development and the difficulties in its use could have been from problems in the framework; (c) the solutions developed are not entire applications and the creation of only a functionality piece might not simulate the usage of a framework in a real system; and (d) the requirements in the specifications were not taken from real applications and were created to match the metadata usage scenarios, which might not be a precise simulation of a real development.

To deal with the two first limitations, student's source code was examined carefully by the present work's author, who also observed the implementations. Whenever mistakes that can compromise the analysis were found, they were considered and referenced in the analysis in order to not invalidate the conclusions.

The two last limitations are related to the specifications and requirements used for each scenario. The requirements are based on concerns that might appear in real applications. The clear specification of how a class should interact with the solution simulates the framework usage encapsulation in order to minimize the effect of the implementation to cover only a piece of functionality.

The time measurements and the questionnaires can also suffer variations due to the following experiment characteristics: (a) the solution developed is a small piece of

software and any unexpected fact, such as a bug, can increase greatly the relative development time; (b) students might have unconsciously evaluated the difficulty to develop each solution in comparison to the solutions of the other case studies developed; and (c) student could have more difficulty in software programming then others and this could interfer in the comparison between their answers.

To avoid the influence of those factors in the conclusions, the analysis of solutions developed was not strictly quantitative, but also qualitative. The development of stages 3 and 5 were observed by the present work's author, who took notes about students difficulties and other events that could interfere with the results. The students also had an opportunity in the questionnaire to write their impressions about the development and justify their answers. This information was considered to the conclusions.

The solution was developed in class by the group, which helped to eliminate the influence of personal difficulties in programming, since the students helped each other to finalize the implementation. It was also important in the equality of each group's development capacity, to make the comparison of development time measurements more reliable.

4 Experiment Experience

The objective of this section is to present the questionnaire answers and the development time measurements. These data are used in the analysis performed in the next sections.Table 1 presents a summary of the the questionnaire answers.

Table 1. Questionnaire answers and development time

Scenario	Questions	Without Frameworks	Traditional Framework	Metadata-based Framework
A	Difficulty to Develop	5	11	10
	Difficulty to Use	-	8	11
	Difficulty to Modify	9	10	6
	Development Time	180 min	97 min	120 min
B	Difficulty to Develop	12	9	6
	Difficulty to Use	-	12	6
	Difficulty to Modify	12	11	5
	Development Time	300 min	144 min	43 min
C	Difficulty to Develop	8	9	7
	Difficulty to Use	-	9	12
	Difficulty to Modify	7	12	9
	Development Time	150 min	71 min	83 min
D	Difficulty to Develop	10	7	6
	Difficulty to Use	-	9	6
	Difficulty to Modify	14	8	9
	Development Time	360 min	128 min	68 min

The first column presents the number of the case study group with the development time for each case study in each phase. The questions are presented in a simplified way, but they represent the three questions described in the section 3.2. The three students answers in each experiment stage were summed and are presented at the table. It is important to highlight that for the implementation without frameworks, the

time was measured by the students, but in the other phases, were measured by the present work's author.

5 Case Studies Metrics and Analysis

This section presents the metrics taken from the solutions and an analysis from the results of each case study. The metrics were based on the one from the overview pyramid [28], which is a metrics-based mean that both describe and characterize the structure of an object-oriented system by quantifying its complexity, coupling and usage of inheritance. The measured values for each version of each scenario are presented on Table 2.

Table 2. Metrics values for the three versions of each scenario. The metrics are Number of Classes (NOC), Number of Methods (NOM), Cyclomatic Number (CYCLO), Lines of Code (LOC), Number of Operation Calls (CALLS) and Number of Called Classes (FANOUT).

	Scenario a			Scenario b			Scenario c			Scenario d		
	1	2	3	1	2	3	1	2	3	1	2	3
Simple Metrics												
NOC	4	4	4	5	5	5	14	11	14	2	2	2
NOM	26	29	23	18	16	16	58	48	71	35	21	21
LOC	163	172	119	82	61	28	271	204	249	287	108	76
CYCLO	41	48	30	26	9	9	80	65	83	74	21	21
CALLS	27	29	17	23	16	2	43	20	14	44	5	1
FANOUT	16	24	13	21	5	1	30	17	11	15	2	1
Computed Proportions												
NOM/NOC	6.5	7.25	5.75	3.6	3.2	3.2	4.14	4.36	5.07	17.5	10.5	10.5
LOC/NOM	6.26	5.93	5.17	4.55	3.81	1.75	4.67	4.25	3.50	8.2	5.14	3.61
CYCLO/LOC	0.25	0.27	0.25	0.31	0.14	0.32	0.29	0.31	0.33	0.25	0.19	0.27
CALLS/NOM	1.03	1.0	0.73	1.27	1.	0.12	0.74	0.41	0.19	1.25	0.23	0.04
FANOUT/CALLS	0.59	0.82	0.76	0.91	0.31	0.5	0.69	0.85	0.78	0.34	0.4	1.0

The analysis takes in consideration the metrics, a qualitative code analysis, questionnaire answers, students observations, development time and the author's observations during the development. The following subsections present a detailed analysis of each case study.

5.1 Scenario 1 - Command-line Parameters Mapping

Analyzing the absolute number of lines of code on Table 2, it is possible to verify that the lines of code increased a little comparing the solution without frameworks with the solution using the traditional framework. The solution using the metadata-based framework has the lower number of lines of code, even if seven additional ones used for annotations were considered.

The Intrinsic Operation Complexity (CYCLO/LOC) do not change much among the implementations, but considering the reduction in the lines of code, it is not the best metric to evaluate the solution's complexity. Since the quantity of methods

remains more stable, the complexity per method is probably a better indicator. Calculating the Cyclomatic Complexity per Method of each solution, it is possible to observe that the solution with metadata has less value.

According to the development times presented in Table 1, the solution using the traditional framework was the fastest to implement followed by the one using the metadata-based framework. From the students notes and from author's observations, the following factors slowed down the development in Stage 5: (a) the framework was hard to understand and did not support the mapping of most of the situations; (b) the exceptions did not point out where were the problems; and (c) some framework exceptions had the same name of application exceptions in the test, which was a fact that took some time for the students to perceive.

According to Table 1, the solution without framework was considered easier to develop but the students recognized that it demanded a lot of manual work. Despite the second solution had been the fastest to implement, the students had the feeling that the framework did not help and increased the development complexity. The metadata-based framework was considered even more complex to understand and with a development difficulty similar to the second solution, but it was considered easier to maintain.

From the first to the second solution, the development time was reduced despite the framework being considered hard to understand and the solution having more lines of code and cyclomatic complexity. This fact can be assigned to the guidance that the framework usage provided for the developers to design the solution's structure.

In the metadata-based framework, the lean and less complex source code did not offset difficulty to understand and use the framework. The framework made difficult the development since the mapping functionalities did not support the application needs. In the case study, for seven properties to be mapped, only tree could be mapped using only the metadata.

Observing the implementation, it is possible to notice that using the metadata-based framework, the implementation of some mapping methods was not necessary since the metadata was enough for the framework to execute the translation. This reduction of effort can be perceived in the metrics by the reduction of complexity and lines of code, but it was not enough to reduce the development time. Following this logic, if the framework supported more mapping functionalities only through metadata configuration, those benefits probably would be higher, consequently reducing the development time.

Another difficulty highlighted in the student's comments was the unclear messages in the exceptions threw by the framework. Those messages did not pointed out what was wrong in the metadata configuration, which hindered in the debug, taking a considerable piece of the development time away.

5.2 Scenario 2 - Method Invocation Constraints

Following the implementation's lines of code evolution on Table 2, it is possible to notice that using the frameworks they got reduced. Even considering the lines of code with annotations, that sums 21, the metadata-based framework is the shortest solution.

The complexity was reduced using both frameworks, since they work with configurations and eliminate the implementation of rules of the application code.

In all the solutions, a proxy was used to implement the validation. In the first solution the proxy was implemented manually and used a lot of operation calls to implement the required validations in each method, which explains the coupling metrics. The number of calls in the solution that used the traditional framework were concentrated in the method that configured the proxy, which invoke a great number of operations on the framework classes. In the solution that uses annotations, only one call to a framework class was needed since it configured the validations based on annotations on the interface. Due to those annotations, this interface had with the framework a semantic coupling, which is not addressed by the metrics.

A large difference in the development time between the implementations can be found in Table 1. According to the students, the code creation was hard-working in the Stage 1 due to a lot of specifications for each method validation that did not allow code reuse. The creation of the method context validation was pointed by the students as specifically hard.

In the second solution, developed with the traditional framework, the students pointed the framework out as one great difficulty. This fact can be verified in Table 1. The lack of documentation for some features made necessary the consultation of the framework authors during the development. The framework also did not implemented correctly the functionalities for method context validation, and consequently five unit tests were unable to execute successfully. The solution was not flexible to allow an extension to workaround this problem.

The students considered the implementation using the metadata-based framework easier and indeed the development time was significantly smaller. According to the group, the annotation names made them intuitive to use. They also felt that the code became a little polluted with the annotations, but they recognize that it was worth for the other benefits.

The development strategies used in Stage 1 and in Stage 3 were completely different. In the first solution the proxy implemented the validation rules in each method, using conditional rules to identify the invalid invocations, which explains its higher cyclomatic complexity. In the second solution, the proxy was created by the framework and configured by the application invoking methods to set the constraints in the proxy class. This configuration did not demand conditional logic, which reduced the solution complexity.

The solution implementation using the traditional framework had some problems that impacted in the development time, such as the framework lack of documentation and missing functionalities. The present work's author, which followed the implementation, judges that without those problems, the team probably would not had reached closer to the last development time.

The coupling had a remarkable difference between the second solution and the one that uses metadata. The source code that created the proxy using the traditional framework were completely dependent on that application class. Contrasting to this, in the third solution this source code was independent from the application class. For instance, in an application whose method invocations should be validated, using the metadata-based framework it would be possible to reuse the code for proxy creation

for all classes. Notwithstanding, that would not be true using the traditional framework.

5.3 Scenario 3 - Stock Exchange Events

The size and complexity metrics do not change much among the implementations according to Table 2. The solution with the traditional framework has the lowest lines of code number, but amounts of unused code were found in the third solution. It was a student's attempt to implement the event representation that was not cleaned when another alternative was chosen.

The coupling is a characteristic that observing the metrics clearly changes among the implementations. The use of interfaces provided by the framework reduced the coupling between the event generator and event handlers. The metadata usage reduced even more this coupling, making easier changes in both sides. Fig. 2 presents the blueprint complexity [27] for the three developed solutions. The dark blue edges, that represent method invocations among classes, are clearly reduced following the implementations.

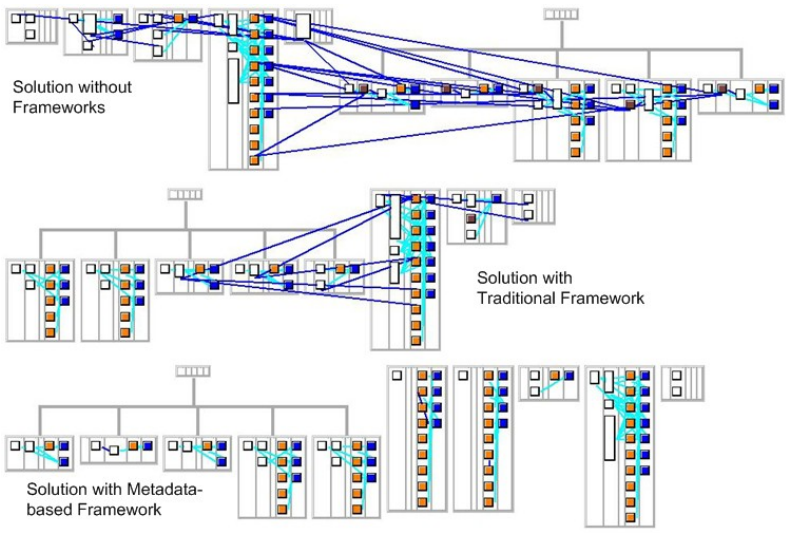

Fig. 2. Class Blueprint from the three implementations.

It was expected that the complexity was reduced using the metadata-based framework, due to rules configuration using annotations. The students that implemented that solution misuse the framework and did not use the annotations to receive only the events with the desired property values.

Observing Table 1, the solution with the traditional framework was the one with the littlest development time. According to the student's observations, the framework

guided the implementation using its interfaces in the application classes and the solution programming was simple.

In opposition to this, the implementation using the metadata-based framework was interfered by many problems that occurred. One of the problems was related to the environment configuration. The metadata-based framework version used an external library to create the dynamic proxies and that library was not included in the path of the project template used by the students. The exception thrown by the framework was not clear about this and it took some time for the students to perceive that a library was missing. The time used to copy the files and configure the project was not considered in the development time.

According to the framework documentation, for attributes in event classes, wrapper classes must be used instead of primitive types. For instance, Integer should be used instead of int. The students that implemented the solution did not attend to this and used primitive types. The framework did not throw an error and simply did not populate that attributes in the event. The group expended a long time finding out what was wrong.

The consequences of this fact can be observed in Table 1. For the metadata-based framework, the framework usage was considered more difficult than the application development itself. For the second implementation no exceptional fact was observed or reported by students, and even so they felt some difficulty in it. The implementation itself was not complicated, so the solution without frameworks, despite taking more time, was considered easy to create.

In this case study, the use of metadata did not reduce the amount of source code which should be developed. The reduction in complexity could not be evaluated since the framework functionalities that could impact on this were not used on the solution. Contrasting this, the reduction on the coupling between the event generator and its handler could be clearly observed by the metrics and the views.

The problems that occurred showed that using a metadata-based framework the developers lose even more control over the processing flow. Unexpected situations that happen inside the framework classes, even due an application class misconfiguration, are hard to be identified and understood by the developers. It highlights the importance of a good error handling strategy implemented by the framework, to validate the class structure and metadata.

5.4 Scenario 4 – HTML Form Generator

According to Table 2, the size metrics reduced through the implementations, especially from the solution without framework to the solution using a traditional framework. If the 24 lines of code with annotations were considered in the last solution, the difference comparing to the second implementation is not so significant.

The solution without frameworks used many methods defined in the same class to generate the HTML form. The inFusion tool, used to generate the overview pyramid, pointed this class out as a God Class and found two Feature Envy inside it [28].

Using the traditional framework, the functionality implementation was reduced to one method which invokes the framework many times to configure the form specific characteristics. The last solution was similar, but those configurations are in

annotations on the target class, reducing even more the method size. This also explains the reduction in the coupling metrics.

For this case study, the development time using metadata-based frameworks was almost the half of the time to create the solution using the traditional framework, as presented in Fig. 2. Without using the frameworks, the development time was really longer, which confirms that developing a graphical interface can be a time consuming task.

The only exceptional situation observed was in the use of the traditional framework, when the students did not observe in the documentation that the class attributes should be public and then they took some time to find out what was wrong. By the observations during the development, without that setback the development time would not be largely reduced.

According to the students evaluation, presented in Table II, the solution without framework was hard to create and could be considered even harder to modify. A student noted that the solution was not flexible and impossible to be reused to generate another form. Comparing the answers in the table, it is possible to affirm that the implementations had a technical draw in difficulty to develop and to change. The major difference was in the framework understanding, which was reinforced by some student's observations. According to them, the traditional framework use configurations in imperative code which is not much intuitive. The opposite was stated for the metadata-based framework.

This case study illustrates how the solutions with similar size metrics, considering the annotation's lines of code, could have a great difference in the development time. This was the only case study that did not have great issues that could interfere in the development time in the implementation with both frameworks. The reason can be found in the following student's observations: it is more intuitive to define metadata declaratively close to the code element which it is referring to, than using imperative code and referencing the code elements using strings.

The coupling in this case study also reduced through the implementations. Using the metadata-based framework, the same method could be used to generate the HTML forms for different classes since the difference between them can be found in their defined metadata. Contrarily, using the traditional framework different methods must be used for different classes, since the configurations should be made inside the methods.

6 General Analysis

A first conclusion that can be draw based on the metrics and development time is that the frameworks, traditional or metadata-based, bring benefits in the application design and can increase the productivity in those scenarios. However, its usage is inadvisable when it does not fulfill the application's needs and it is not flexible enough to be adapted to them. The frameworks provide an easy way to reuse functionality among features of the same application and even among different ones. Besides, they guide the development providing a ready-to-use design structure to the application, which

can reduce the development time even when the lines of code are almost the same comparing to a solution without their use.

A metadata-based framework can potentially provide a solution in which the developer can add metadata to the existent classes intuitively increasing productivity, as it happened in the groups 2 and 4. In contrast to this, as evidenced by groups 1 and 3, the use of a framework based on metadata do not guaranties a high productivity. Consistent with this, in group 1 the solution that used the metadata-based framework took more time even having less lines of code.

According to [29], the lack of an explicit control flow in applications which uses frameworks can difficult the developer's understanding of it. In frameworks that use the metadata-based approach, where the adaptations are based on the class metadata, this problem is even worst since the flow of control is even more implicit. Because of that, it is difficult to find errors related to their usage in applications. For instance, metadata configuration errors, such as a missing property or a misspelled string, are pretty hard to detect. This difficulty to find errors can be a bottleneck in the team productivity. Those facts can be observed in the implementations with the metadata-based framework in groups 1 and 3.

This evidence makes the error handling and metadata validation important features for a metadata-based framework. The error or warning messages should be designed to help the developer to find a misconfiguration. Those frameworks were not automatically good just for using metadata. Best practices valid for every piece of software, such as good naming and clear documentation, are also important in this context. Specific best practices, such as those presented in [10], are also important to make the framework more flexible enabling it to be adapted to the application needs.

An interesting fact that happened in the traditional framework's implementations was that three of them used a programmatic approach to set additional information about the application classes into the framework, in other words, metadata. In Group 1, the application class had to implement an interface which had methods to return additional metadata about the class. In groups 2 and 4, the framework main class provides methods to set information referencing the application class elements directly in the framework. If inexperienced students in framework development had chosen a solution based on metadata definition even without its knowledge, that might evidence that defining metadata in those scenarios is an intuitive approach.

Despite all other facts, a constant characteristic of the solutions that used the metadata-based frameworks is the coupling reduction, which can be confirmed in all case studies. The use of this kind of framework decouples the application classes from framework since the need for them to implement interfaces or extend a superclass from the framework is eliminated. Its use also decouples the client class that invokes the framework functionalities from the application class that is processed from the framework. However, it is important to notice that it still exist an indirectly or semantic coupling between the framework metadata definition and the application class, which was not addressed by the metrics [30]. The use of an external metadata strategy or domain annotations mapped to framework annotations [31] can help to reduce this semantic coupling.

Other benefits also can be achieved by the use of metadata-based frameworks, which depends on the framework's functionality and domain. When the framework manages to encapsulate features that must be implemented by the application using a

traditional approach, it probably would reduce the complexity and the lines of code number in the application where it is applied.

7 Conclusion

This paper presents an evaluation of metadata-based frameworks usage based on an experiment. The experiment created a comparison basis for applications without frameworks, using traditional frameworks and using metadata-based frameworks for distinct scenarios. As a result, it was possible to assess benefits and drawbacks in the use of this approach. The analysis used object-oriented metrics, questionnaire answers, observations, source code analysis and development time measurements to reach the conclusions.

Further studies can explore the use of metadata-based frameworks with more features for more complete applications. In these scenarios, it would be possible to explore other issues, such as the reuse provided among different functionalities. Other future works could aim on solutions to common needs of this kind of framework, such as exception handling on metadata reading.

References

1. Johnson, R., Foote, B.: Designing reusable classes. In Journal Of Object-Oriented Programming. v.1, n. 2, Jun./Jul., 22-35 (1988)
2. Pree, W.: Hot-spot-driven development. In Building application frameworks: object-oriented foundations of frameworks design. New York: Wiley, Chap. 16, 379-393 (1999)
3. Doucet, F., Shukla, S., Gupta, R.: Introspection in system-level language frameworks: meta-level vs. Integrated. In Source Design, Automation, and Test in Europe, 382-387 (2003)
4. Forman, I., Forman, N.: Java reflection in action. Greenwich, Manning Publications (2005)
5. Foote, B., Yoder, J.: Evolution, architecture, and metamorphosis, In Pattern Languages of Program Design 2. Boston: Addison-Wesley Longman, Chap. 13, 295-314 (1996)
6. Schwarz, D.: Peeking inside the box: attribute-oriented programming with Java 1.5, http://missingmanuals.com/pub/a/onjava/2004/06/30/insidebox1.html (2004)
7. JSR 175: a metadata facility for the java programming language, http://www.jcp.org/en/jsr/detail?id=175 (2003)
8. Chen, N.: Convention over configuration, http://softwareengineering.vazexqi.com/files/pattern.html (2006)
9. Fernandes, C., Ribeiro, D., Guerra, E., Nakao, E.: XML, Annotations and Database: a Comparative Study of Metadata Definition Strategies for Frameworks. In: XML: Aplicações e Tecnologias Associadas, Vila do Conde, Portugal (2010)
10. Guerra, E., Souza, J., Fernandes, C.: A pattern language for metadata-based frameworks, In Conference on Pattern Languages of Programs, 16, Chicago (2009)
11. Bauer, C., King, G.: Java persistence with hibernate. Greenwich, Manning Publ. (2006)
12. JSR 220: Enterprise JavaBeans 3.0, http://www.jcp.org/en/jsr/detail?id=220 (2006)
13. Brown, D., Davis, C., Stanlick, S.: Struts 2 in action. Greenwich, Manning Publ. (2008)

14. JSR 222: Java Architecture for XML Binding (JAXB) 2.0, http://jcp.org/en/jsr/detail?id=222 (2006)
15. Miller, J.: Common language infrastructure annotated standard. Boston, Addison-Wesley (2003)
16. Rouvoy, R., Pessemier, N., Pawlak, R., Merle, P.: Using attribute-oriented programming to leverage fractal-based developments. In International ECOOP Workshop on Fractal Component Model, 5, Nantes (2006)
17. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
18. O'Brien, L.: Design patterns 15 years later: an interview with Erich Gamma, Richard Helm and Ralph Johnson. http://www.informit.com/articles/article.aspx?p=1404056 (2009)
19. Swingbean: aplicações Swing a Jato!, http://swingbean.sourceforge.net
20. Nucitelli, R., Guerra, E., Fernandes, C.: Parsing XML Documents in Java Using Annotations. In XML: Aplicações e Tecnologias Associadas, Vila do Conde, Portugal (2010)
21. Zelkowitz, M., Wallace, D.: Experimental validation in software engineering. In Information and Software Technology, V. 39, 735-743 (1997)
22. Basili, V.: The role of experimentation in software engineering: past, current and future. In Proceedings of the 18th international conference on Software engineering. IEEE Computer Society, Washington, DC, USA, 442-449 (1996)
23. Soares, S., Borba, P.: Towards progressive and non-progressive implementation approaches evaluation, In Proceedings of Experimental Software Engineering Latin American Workshop (2004)
24. Noël, R.: Evaluating Design Approaches in Extreme Programming. In Proceedings of Experimental Software Engineering Latin American Workshop (2005)
25. Guerra, E., Fernandes, C., Silveira, F.: Architectural Patterns for Metadata-based Frameworks Usage. In Proceedings of Conference on Pattern Languages of Programs, 17, Reno (2010)
26. Lanza, M., Ducasse, S.: Polymetric views: a lightweight visual approach to reverse engineering. In IEEE Transactions on Software Engineering. V.29, n.9, 782-795 (2003)
27. Ducasse, S., Lanza, M.: The Class Blueprint: visually supporting the understanding of classes export. In IEEE Transactions on Software Engineering. V.31, n. 1., 75-90 (2005)
28. Lanza, M., Marinesco, R.: Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer (2006)
29. Fayad, M., Schmidt, D., Johnson, R.: Application frameworks. In Building Application Frameworks: Object-oriented Foundations of Frameworks Design. New York: Wiley, Chap 1, 3-27 (1999)
30. Guerra, E., Silveira, F., Fernandes, C.: Questioning traditional metrics for applications which uses metadata-based frameworks. In Workshop on Assessment of Contemporary Modularization Techniques. V. 3, Orlando (2009)
31. Perillo, J., Guerra, E., Silva, J., Silveira, F., Fernandes, C.: Metadata Modularization Using Domain Annotations. In Workshop on Assessment of Contemporary Modularization Techniques. V. 3, Orlando (2009)