An Empirical Study on the Impact of Aspect-oriented Model-driven Code Generation

André Menolli[1,2][a], Luan de Souza Melo[3][b], Maurício Massaru Arimoto[1][c] and Andreia Malucelli[3][d]

[1] *University of Northen Paraná (UENP), Brazil*

[2] *Postgraduate Program in Computer Science, State University of Londrina (UEL), Brazil*

[3] *Postgraduate Program in Computer Science, Pontificia Universidade Católica do Paraná (PUCPR) , Brazil*

Keywords: Model-driven Development, Aspect-oriented Software Development, Software Quality, Experimentation.

Abstract: Over the years innovative approaches in software development have been proposed. Among the main approaches, we can highlight aspect-oriented software development. However, applying aspect-oriented software development is not simple, but may be facilitated by the model-driven development, mainly because it is possible to build models to drive consolidated aspect solutions. In this context, we analyzed the impact of aspect-oriented solutions created from a model-driven approach. To this end, a model-driven approach to create aspect-oriented code was proposed and an experiment focusing on data persistence was conducted. From data gathering, we empirically discuss the impact of the generated solutions compared to oriented-object solutions. Some code metrics were analyzed using quantitative analysis and the results show that the approach may help to reuse aspect-oriented solutions and improve the code quality and productivity.

1 INTRODUCTION

The aspect-oriented software development (AOSD) is a software development paradigm designed to reduce the effort required to maintain software systems by replacing cross-cutting code with aspects. Many studies (Hoffman and Eugster, 2009; Katic et al., 2013; Kulesza et al., 2006) have been conducted showing that AOSD may be effective to improve the separation of concerns, and, consequently, improve the reusability and maintainability. However, recently, the study of (Przybyłek, 2018) indicates that AOSD decreases the code understandability while improving changeability. This explains why previous experiments measuring completion time generally showed the advantage of object-oriented development over AOSD (Mehmood, 2013).

Considering this, we believe it is possible to use model-driven development (MDD) to create aspect code to have less impact on understandability and, at the same time, have the advantages that the use of AOSD provides. MDD allows code development from high-level models and its function is essentially to transform a software model into executable code and to vary totally or partially according to needs (France and Rumpe, 2007).

While MDD allows domain specific abstraction, AOSD provides greater support for modularization concerns throughout the software development lifecycle (Groher and Voelter, 2009; Pinto et al., 2007).

Thereby, we proposed an MDD-based approach to assist in semi-automatic coding of aspect-oriented applications, and through a controlled experiment with 20 students, we evaluated our approach and compared it with OO development. The experiment was carried out in the context of data persistence. This domain was chosen because there are many methods and tools to implement data access in separate ways. Our approach aims to allow AOSD to be applied almost transparently for the end user. We followed the Goal-Question-Metric (GQM) approach (Basili et al., 1994) to define a measurement system for our research. Figure 1 presents an overview of the goal, questions, metrics, as well as their interrelations.

[a] https://orcid.org/0000-0002-4755-8031
[b] https://orcid.org/0000-0002-3085-3819
[c] https://orcid.org/0000-0002-3972-0764
[d] https://orcid.org/0000-0002-0929-1874

Menolli, A., Melo, L., Arimoto, M. and Malucelli, A.
An Empirical Study on the Impact of Aspect-oriented Model-driven Code Generation.
DOI: 10.5220/0010442802750282
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 2*, pages 275-282
ISBN: 978-989-758-509-8

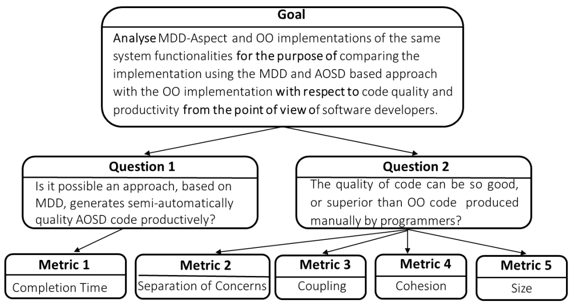
Goal
Analyse MDD-Aspect and OO implementations of the same system functionalities for the purpose of comparing the implementation using the MDD and AOSD based approach with the OO implementation with respect to code quality and productivity from the point of view of software developers.
Question 1
Is it possible an approach, based on MDD, generates semi-automatically quality AOSD code productively?
Question 2
The quality of code can be so good, or superior than OO code produced manually by programmers?
Metric 1
Completion Time
Metric 2
Separation of Concerns
Metric 3
Coupling
Metric 4
Cohesion
Metric 5
Size

Figure 1: GQM Measurement plan.

2 RELATED WORKS

The idea of combining model-driven techniques and aspect-oriented development is not new. Over the years, approaches have been proposed to solve different problems in the research. For example, a long-standing initiative in this direction is the paper (Simmonds et al., 2005) that presents an Aspect Oriented Model Driven Framework (AOMDF) that facilitates separation of pervasive features and supports their transformation through different levels of abstraction. Another work (Singh and Sood, 2009) proposes to incorporate the merits of AOSD such as modularization, reuse and complexity in the Model Driven Architecture (MDA) software development strategy.

Many papers focus on the integration of these two technologies, proposing high-level solutions using model-driven engineering (MDE) or MDA. For example, (Tekinerdogan et al., 2007) considers MDA and the aspect-oriented approach as complementary techniques for the separation of concerns and develops a systematic analysis of cross-cutting concerns in the MDA context. Another study on the same line of work (Groher and Voelter, 2009) presents an approach that facilitates variability implementation, management, and tracing by integrating model-driven and aspect-oriented software development, where features are separated into models and composed of aspect-oriented composition techniques at model level.

However, the combination of these technologies is not limited for the aforementioned areas. Different approaches may be proposed for several areas to address specific problems. For example, (Pérez et al., 2013) propose a methodology that should allow code generation from models that specify functional and non-functional requirements. In addition, the work of (Mehmood, 2013) presents a systematic mapping study of aspect-oriented model-driven code generation. They conclude that aspect-oriented model-driven code generation is indicated as a relatively immature area.

The papers presented works focus on generating the final AO code, and unlike the other presented, this work focus on generating the final aspect code in a transparent way for developers, for very specific consolidated solutions, such as patterns and to empirically compare the generated code with the OO code produced for the same solution.

3 PROPOSAL APPROACH

The proposed approach tries to help developers to apply the AOSD paradigm, helping them to define where and how to use the AOSD, using a model-driven approach. Figure 2 shows how our approach works. The proposed approach presents two roles: (a) the Framework Developer - responsible for creating PIM and Model Transformations (Class and Aspect Transformations); and (b) the Application Developer - responsible for creating a PSM based on PIM.

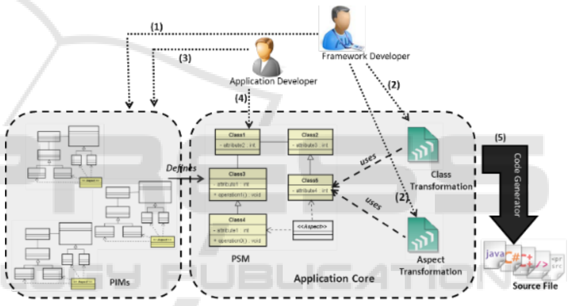

Figure 2: The proposed approach.

The first step (1) is the creation of the PIM. In our approach, PIM is a metamodel, that is, a generic model that describes a reusable solution to a problem that usually occurs within a given context and can be used in different situations. Thus, the PIM is a generic model that describes a domain solution to guide the creation of applications and is created by the Framework Developer.

Therefore, any application developed by the Application Developer must be created from a PIM. Specifying the aspect and where (which classes) it may act on the PIM is the critical point of the approach. The Framework Developer is responsible for defining the aspects that a model should contain, helping to create an organizational standard, since any application in a given domain will share the same aspects and implementation.

The second step (2) is to create the transformations, which is also the responsibility of the Framework Developer, and will be used to generate low-level source code. We have chosen to use Acceleo to define the Model Transformations (Koch, 2007). Acceleo (https://www.eclipse.org/acceleo/) is an implementation of the Object Management Group (OMG) Model to Text Language (MTL). This choice is due to the possibility of automatically transforming platform-independent models (PIMs) to low-level source code (Mtsweni, 2012).

In the third step (3), the Application Developer chooses a PIM to start developing an application, and in the fourth step (4) he/she creates the PSM. In this step, the Application Developer must create a model based on the PIM defined by the Framework Developer. In addition, Application Developer must define all parameters to configure the classes and aspects being generated. PSM, along with the transformations, comprises the Application Core, from which the source code may be generated (5).

4 RESEARCH STRUCTURE

To evaluate our approach, we defined the research structure following the research decision-making structure presented by (Wohlin and Aurum, 2015), which shows decision points that are grouped into three phases: strategic, tactical and operational. Once the research questions issues had been defined, the decision points were selected.

4.1 Definition of Hypotheses

To guide our experiment, two hypotheses were formally stated:

- **Ha** μ**T** $< \mu$**A**: where, the approach based on MDD and AOSD is superior in productivity compared to an OO implementation produced manually by programmers.
- **Hb** μ**T** $< \mu$**A**: where, the approach based on MDD and AOSD generates a higher code quality than an OO implementation produced manually by programmers.

The next step in the planning activity was to determine variables (independent variables (persistence technology, the approach used) and the dependent variables (development time and code quality)).

4.2 Participants

The participants who collaborated in this experiment were undergraduate and graduate students of the Computer Science course. The participants were specially selected individuals from the population of interest for the experiment, since the experiment wants to simulate the behavior of individuals within a software development company. We applied a placement test to select participants who had the same level of coding skills and knowledge about all the technologies involved in the experiment (in general participants have little knowledge about AOSD), so that they can perform the proposed functions at a similar time. Therefore, a non-probabilistic sampling was used in the selection of participants, considering a convenience sample, which the closest and most convenient people are selected. Furthermore, the experiment was complex, requiring knowledge of design patterns, and distinct technologies (such as Hibernate, JDBC and JPA), in addition to AOSD. Thus, the population had been limited by these restrictions.

4.3 Description of the Experiment and Procedure

In this experiment, the participants were invited to perform a software implementation in library loan management software, developed in the Java platform. The proposed implementation was based on the Data Access Object (DAO) pattern using GenericDAO. Each participant implemented the data persistence code for a given functionality (borrow book), using one of the technologies: JPA, Hibernate or JDBC.

The experiment was divided into two parts: **A** – using the approach and **B** – not using the approach. The tasks requested to be implemented in the experiment were: 1A – JPA using the approach; 1B – JPA not using the approach (OO implementation); 2A – Hibernate using the approach; 2B – Hibernate not using the approach (OO implementation); 3A – JDBC using the approach; 3B – JDBC not using the approach (OO implementation).

For **A** implementations, the participants followed the flow presented in Figure 3, which includes four steps: (1) Access the documentation and source code of the lending functionality (domain layer). The participants, in this step, had access to the requirement and project artifacts (Book Borrow Requirements - Use Case; Class Diagram; Sequence Diagram; Code of domain layer (.java files) and Script to generate the database). (2) Create the PSM (based on the PIM) for the requested technology. (3) Generate the initial source code, according to the PSM created and (4) Finalize and test the source code.

On the other hand, for **B** implementations, participants followed two steps: (1) Access the documentation and source code of the borrow functionality (domain layer) and (2) Create OO source code using the requested technology.

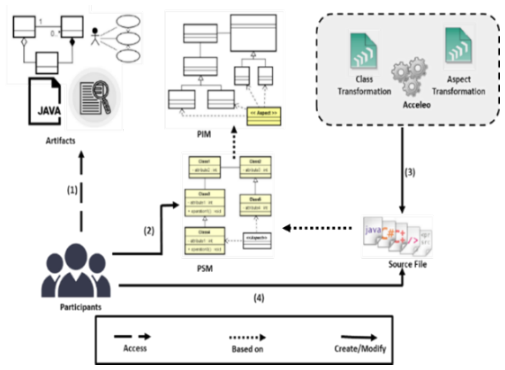

Figure 3: Flow of experiment to implement solution using the proposed approach.

In this experiment, a quantitative analysis was used. The analysis was performed using productivity metrics and a set of separation, coupling, cohesion, and size metrics (Sant'Anna et al., 2003). Moreover, an expert analysed the solutions created by participants in order to understand whether the solutions corresponded to what was required, and whether the solutions were acceptable for the proposed problem.

5 THE FRAMEWORK

To implement a framework using the proposed approach, we chose the data persistence domain for two main reasons: previous studies, such as (Oliveira et al., 2008), confirm that the use of AOSP in this type of problem could be a good strategy and data persistence is common to several types of applications, thus facilitating data reuse.

The framework was built to help create data persistence applications using DAO and AOSD. Moreover, it supports the following technologies: Java Database Connectivity (JDBC), Java Persistence API (JPA) or Hibernate. The following subsections describe how each step of the proposed approach was implemented.

5.1 The PIM

In order to support the creation of data persistence applications using three different technologies, the Framework Developer defined two PIMs (Hibernate and JPA use the same PIM). They were created through the Eclipse Modeling Framework (EMF), and define the essential elements of the submitted domain. The EMF is a modeling framework and code generation facility for the development of applications based on a structured data model. We used the EMF, since it has become undoubtedly has become standard in the MDE community, as well as in the most MDE tools.

As an example of PIM, Figure 4 presents a diagram containing the entities of the Hibernate and JPA PIM as well as their relationships. The PIM is composed of the following entities:

Init: has the role of starting point when building the PSM. Therefore, it is possible to create AspectDAO, Control and Persistence entities. **Control**: associates the Object to DaoClass. In addition, Control contains a set of interfaces equivalent to the set of interfaces brought by DaoClass, which may be used by Init to trigger the data persistence functionality. **GenericDAO**: A skeleton of DaoClass. GenericDAO is an abstract entity that contains declarations of the required interfaces of a DaoClass. Furthermore, it contains the functionality that the AspectDAO will use as a pointcut. **DaoClass**: an extension of the GenericDao class. As DaoClass is associated with Object, it may contain specific queries performed on it. **Object**: a POJO which determines an entity of the domain as to its attributes and behaviors. In other words, in the data persistence domain, the Object could also be described as an element of the model layer. **Attribute**: responsible for defining the Object class attributes. **Persistence**: responsible for the information needed to connect to the database. textbfAspectDAO: defines a pointcut and an advice that is triggered when data persistence functions from GenericDAO are executed or called.

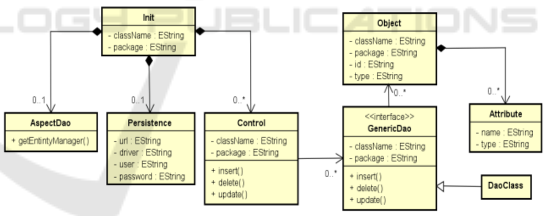

Figure 4: JPA and Hibernate PIM.

5.2 The Model Transformation

As previously described, the model transformations performed on our framework run on Acceleo. In Acceleo, the model transformation is designed through standard templates, which are able to load model data and manipulate it in order to build the resulting artifacts, namely, low-level source code.

The transformations were built upon the entities present in the PIMs. Therefore, every entity will present a distinct data treatment and all of them will be supported in the model transformation process. The creation of model transformation is responsibility of the Framework Developer (the Framework Developer has created seven transformation classes and one aspect transformation for each technology JPA, Hibernate and JDBC). These transformations are templates, which will be used to generate low-level source code. The approach supports two types of transformations: class and aspect transformations. Class transformations contain the Acceleo content to create the class structure and the main constraints. Aspect transformations contain the Acceleo content to define the whole aspect.

5.3 The PSM

Once the PIMs and Transformation are created, the Framework Developer is able to generate a plugin containing all metadata of the PIMs, namely, an Eclipse build that supports the creation of the PSM based on the entities of a PIM. To create the PSM, the Application Developer creates a model based on a PIM, using a tree-view feature to build the hierarchy and attribute entities' values. Then, the Application Developer must define the attributes of the entities. Once the PSM is built, the Application Developer must run the Acceleo, so the low-level source code is generated from the PSM and Transformations (which act on the PSM).

6 DATA ANALYSIS

To evaluate our approach and compare it with the OO implementation produced manually by the programmers, we used the Framework presented. The experiment was conducted as described in Section 4, and once all participants had executed the requested tasks, the data were collected and analyzed. To analyze the code metrics we used 10 software metrics frequently used in software projects presentd in (Varela et al., 2017) .

6.1 Productivity

To define whether the productivity to generate code is higher using our approach than manually by programmers, we used two types of statistical analysis. The first analysis sought to understand: given an implementation using the approach, what is the probability that the time to complete this implementation will be less than an OO implementation produced manually by programmers?

In our experiment, we believe that the time to develop a solution using the proposed approach is lesser than to develop manually the solution OO. Thus, from the hypothesis Ha, we proposed a null Ha_0 hypothesis, which states that there is no difference in the time of development using the approach or not using it. That is, the null hypothesis can be formulated as: Ha_0: $\mu T = \mu A$; where, there is no productivity difference in relation to the use of the approach based on MDD and AOSD with respect to the OO implementation produced manually by programmers.

In order to reject the null hypothesis, we used a non-parametric test (binomial test). In the experiment, twelve solutions were produced to compare the two types of development. In 11 times, the time to develop was shorter using the proposed approach. Only once was the time the same or shorter not using the approach. To calculate the binomial test, we defined that α should be less than 0.05 (the critical area, C is $\alpha \geq 0.05$). To verify the hypothesis we used the binomial formula:

$$P\ (0\text{-}1\ \mu T) = \sum_{i=0}^{1} \left(\frac{12}{i}\right) \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{12-i} = \frac{1}{2^{12}} \sum_{0}^{1} \left(\frac{12}{i}\right) = 0.003 \tag{1}$$

In this case, we rejected Ha_0 because $0.003 < 0.05$. As a result, the Ha hypothesis is discussed. We already knew that this is the probability of developing a solution using the proposed approach in a shorter time. However, we want to analyze this difference.

We compared if there is a statistically significant difference in the development time between the two categories (using the approach and developing manually using OOP). First, we checked whether the sample is normally distributed using Shapiro–Wilk test, which presents a null hypothesis that a sample comes from a normally distributed population if p-value $\geq$ x (for all tests we considered p-value as 0.05). The result showed that the sample is normally distributed. So, as the samples are independent, we use t-test to compare them. The t-test null hypothesis is that the two samples are equivalent. The results of these tests are presented in Table 1. Comparing all solutions (first line) there is a difference. However, looking for each solution, there is a difference in time only for the first solution (JPA implementation). So, we applied the t-test (greater) to identify if the side A is greater than B, in cases where there was a difference. In both cases, the implementation time when the approach was used was statistically better.

6.2 Code Quality

To evaluate the code quality, we used the metrics summarized in Table 2. First, we used the binomial test to see if there is a higher probability of having a better-quality code if the proposed approach is used instead of implementing it manually using OOP. From the Hb hypothesis, we proposed a null H_0 hypothesis, which states that there is no difference in the code quality

Table 1: Statistical difference between development time.

Comparison (tasks)	t-test (p-value)	t-test (greater) (p-value)
A-B (total)	0.0206	0.989
1A-1B	0.0237	0.988
2A-2B	0.1338	–
3A-3B	0.4024	–
A - Task executed using the approach B - Task executed manually by programmers		

using the approach or not using it. That is, the null hypothesis can be formulated as: Hb_0 $\mu Tl = \mu A$; where, there is no quality difference in relation to the use of the approach based on MDD and AOSD with respect to the OO implementation produced manually by programmers.

We compared ten metrics, considering the twelve solutions of two kinds of development. Thus, there are 120 metrics to compare. At 70 times the metric was higher using the approach. At 50 times the metric was the same or higher not using the approach.

We defined that α should be below 0.05 (the critical area, C is $\alpha \geq 0.05$). Hence, to verify the hypothesis, we used the formula (2):

$$P(0\text{-}50\ \mu T) = \sum_{i=0}^{50} \left(\frac{120}{i}\right)\left(\frac{1}{2}\right)^{i}\left(\frac{1}{2}\right)^{120-i} = \frac{1}{2^{120}}\sum_{0}^{50}\left(\frac{120}{i}\right) = 0.041 \qquad (2)$$

In this case, we rejected Hb_0 because $0.041 < 0.05$. As a result, the Hb hypothesis is discussed. We already know that there is a higher probability of having a better-quality code using the proposed approach if compared to not using the approach. However, we want to analyze each metric and apply statistical techniques. Table 2 presents the median and standard deviation for each technology used. Moreover, we compare with which metrics there are statistically significant differences between the two categories of implementation.

The results are presented in Table 3 that presents the general evaluation (comparing all implementations using the approach with the implementations that do not use it) and in Table 4 that compares each task.

The results based on metrics was superior when using the proposed approach. Other studies have already shown that aspects support the improvement of the separation of concern, such as (Garcia et al., 2005), and (Oliveira et al., 2008), which explores the separation of concerns specifically for data persistence applications. Therefore, as shown in Table 2, the Concern Diffusion over Components (CDC) was improved by 83%. The Concern Diffusion over Operations (CDO) was improved by 86%. The last metric, Concern Diffusions over LOC (CDLOC), was improved by 92%. Besides the superior results when the approach was used, it is important to realize that none of the metrics presented a difference in the standard deviation when the approach was used.

Table 2: Metrics and their statics.

Metrics		Using Approach			Not Using Approach		
		1A	2A	3A	1B	2B	3B
CDC	Median	1,0	1,0	1,0	6,0	6,0	6,0
	Stand.Dev.	0,0	0,0	0,0	0,0	0,0	0,0
CDO	Median	6,0	6,0	4,0	38,0	40,0	39,0
	Stand. Dev.	0,0	0,0	0,0	2,1	2,8	1,0
CDLOC	Median	2,0	2,0	2,0	22,5	25,5	25,0
	Stand. Dev.	0,0	0,0	0,0	2,4	6,2	6,1
CE	Median	1,9	2,3	2,2	2,3	2,5	2,3
	Stand. Dev.	0,3	0,5	0,6	0,6	0,6	0,7
CA	Median	3,1	3,2	3,1	3,0	3,1	3,1
	Stand. Dev.	0,3	0,4	0,5	0,3	0,3	0,6
DIT	Median	1,0	1,0	1,0	1,0	1,0	1,0
	Stand. Dev.	0,0	0,0	0,0	0,0	0,0	0,0
LCOO	Median	0,4	0,4	0,4	0,4	0,3	0,3
	Stand. Dev.	0,0	0,1	0,0	0,0	0,0	0,0
LOC	Median	581,0	569,0	686,0	643,0	613,0	647,5
	Stand. Dev.	59,4	58,2	135,6	68,7	76,2	120,9
NOA	Median	30,0	29,5	35,5	30,0	33,5	32,0
	Stand. Dev.	2,7	2,6	1,0	2,7	5,7	3,3
WOC	Median	95,5	102,5	118,5	102,5	115,0	112,5
	Stand. Dev.	4,9	8,8	17,6	13,3	5,5	18,1

Table 3: Statistical difference between Metrics in general.

		Parametric Samples		Nonparametric Samples	
Metrics	SW*	t-test	t-test (greater)	MW**	MW**(greater)
DC	1.762e-06	--	--	1.912e-06	1
CDO	1.219e-05	--	--	2.342e-05	1
CDLOC	5.513e-05	--	--	9.513e-06	1
CE	0.07471	0.03527	0.9824		--
CA	0.1579	0.5501	--		--
LCOO	0.06497	0.2094	--		--
LOC	0.01593	--	--	0.3186	
NOA	0.02325	--	--	0.9531	--
WOC	0.0186	--	--	0.3855	--
*SW – Shapiro-Wilk Test **MW-Mann–Whitney U					

Table 4: Statistical difference between Metrics in each task.

		Compared Experiments		
Metrics	Statistical tests	1A – 1B	2A-2B	3A-3B
CDC	MW*	0.0131	0.0131	0.0131
	MW* greater	0.997	0.997	0.997
CDO	MW*	0.02021	0.02021	0.01771
	MW* greater	0.9958	0.9958	0.9964
CDLOC	MW*	0.02021	0.02021	0.02021
	MW* greater	0.9958	0.9958	0.9958
CE	t-test	0.2108	0.3351	0.3325
CA	t-test	1	0.9462	0.325
LCOO	t-test	0.9462	0.7241	0.191
LOC	MW*	0.3429	0.3429	0.6857
NOA	MW*	1	0.3065	0.1441
WOC	MW*	0.6612	0.1143	0.4857
*MW - Mann–Whitney U				

Considering coupling metrics,the metric epth Inheritance Tree (DIT) showed no difference from the two approaches. However, the results obtained in the Coupling Efferent (CE) and Coupling Afferent (CA) metrics show that the implementations using the proposed approach had a reduction in coupling of 9% and 3% respectively. Regarding Lack of Cohesion in Operations (LCOO), it was the only metric that in general presented a worse result using the proposed approach, although the difference was small, only 4%.

In the last metrics, regarding code size, the following data are analyzed: Lines of Code (LOC), Number of Attributes (NOA) and Weighted Operations per Component (WOC). Regarding LOC metric, the proposed approach was 4% more efficient, while in the NOA metric it was 2% better. In the last WOC metric the proposed approach was 3% more efficient. Moreover, observing Table 2, it is possible to see that in general the size metrics presented a lower standard deviation when the approach was used.

7 DISCUSSIONS

To answer the question a "Is it possible to productively generate semi-automatically quality AOSD code based on MDD?", we analyzed the experiment and its results. Based on the experiment, we conclude that our approach generates semi-automatic quality AOSD productivity. In the experiment, all participants were able to develop three different persistence AOSD codes. Thus, based on experiment, the approach helps to create different solutions for a generic domain semi-automatically. Yet considering the experiments, analyzing the development time, and the code quality metrics that are considered to generate quality code productivity. Thus, we consider that the first main question is true, and that it is possible an approach, based on MDD, productively generates semi-automatically quality AOSD code.

However, to analyze productivity and quality compared to a OO code developed manually, it is necessary to analyze question b: "Can the quality of the code and productivity be as good, or better than the OO code produced manually by the programmers?". Based on the results of the experiment, we may state that, using the proposed approach, there is a greater probability of having a shorter development time compared to OO manual development. Furthermore, the development time was statistically significantly shorter compared to manual OO development.

Regarding code quality, using the proposed approach there is a greater probability of having higher quality code compared to the manual OO development. Statistically, we may state that development using the approach helps to improve the Separation of Concerns and Coupling. Therefore, we consider that the second question is also true, since the productivity and code quality produced using the approach was superior to the OO code produced manually.

Based on the experiments, there are other aspects that may be discussed, considering the expert's perception. In the approach, the use of aspects should be as transparent as possible, i.e., the developer only creates an aspect in the model, and sets some parameters for it works properly. Therefore, the developers do not need to understand this concept, and one advantage is that all implementations are equals, helping the maintenance and reuse.

7.1 Threats to Validity

The main threats to the validity of this study were identified as its limitations are discussed below. **Construct validity**. Participants in the experiments were only students. Data were collected from a single data source. We tried to mitigate this threat by applying a placement test to ensure all students have had similar knowledge levels and all were able to perform the requested tasks. **Design Threats**. The use of poorly designed experimental artifacts may be a threat. To mitigate this issue, we designed our experiment to work as software maintenance. **Inappropriate selection of subjects**. The scope of domain (database) is a threat since for other domains could present different results. However, we believe the application of the approach to other domains has a great chance of success, since the domain enables aspect solutions, and a well-defined PIM is created. **Conclusion validity**. The use of inappropriate measure occurring the wrong understanding of the results. We defined clear hypotheses and chose the appropriate statistical tests for each hypothesis. Furthermore, we tested all data, verifying the data characteristics to use the correct statistical test and we applied the power test.

8 CONCLUSIONS

We have carried out an experiment to report an empirical evaluation of our approach and to compare the AO code generated by using the approach with the OO code generated. The final code produced by the framework based on the proposed approach, considering all the data obtained in the analysis, was considered of better quality than that OO code produced manually by programmers. This corroborates with other studies that have already presented comparisons between AO and OO code, as instance (Hoffman and Eugster, 2009);(Katic et al., 2013);(Kulesza et al., 2006). However, although AOSD is effective in improving the separation of concerns, (Przybyłek, 2018) pointed out that AOSD decreases understandability, and this explains why object-oriented development is advantageous over AOSD at completion time. However, using our approach, the completion time to produce AO code was shorter than the time to produce the OO code manually.

As main advantage, we can highlight that our approach brought the benefits already discussed in the literature by the use of ASOD, and on the other hand avoid the problems pointed out by (Przybyłek, 2018), since the approach may help to improve the software development (both quality and productivity).

Finally, the AOSD appeared more than two decades, and so far its use remains a challenge in the industry. Therefore, a fundamental novelty of our research is that the application of AOSD should be rethought, given the difficulty of its use in practice; however this study shows that there are alternatives, such as the use of higher-level solutions.

ACKNOWLEDGEMENTS

This study was financed in part by the Fundação Araucária and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), protocol n. 9908272120340724

REFERENCES

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). *Goal Question Metric Approach". In: Encyclopedia of Software Engineering*. Wiley.

France, R. and Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, United States. IEEE Computer Society.

Garcia, A., Sant'anna, C., Figueiredo, E., and Kulesza, U. (2005). Modularizing design patterns with aspects: A quantitative study. In *International Conference on Aspect-Oriented Software Development*.

Groher, I. and Voelter, M. (2009). Aspect-oriented model-driven software product line engineerings. *Trans. Asp. Softw. Dev*, 5560:111–152.

Hoffman, K. and Eugster, P. (2009). Cooperative aspect-oriented programming. *Sci Comput Program*, 74(5–6):333–354.

Katic, M., Boticki, I., and Fertalj, K. (2013). Impact of aspect-oriented programming on the quality of novices' programs: A comparative study. *Journal of Information and Organizational Sciences*, 37(1).

Koch, N. (2007). Classification of model transformation techniques used in uml-based web engineering. *Software, IET*, 1:98 – 111.

Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., von Staa, A., and Lucena, C. (2006). Quantifying the effects of aspect oriented programming: A maintenance study. In *22nd IEEE Intl. Conf. on Software Maintenance*.

Mehmood, A. (2013). Aspect-oriented model-driven code generation: A systematic mapping study. *Information and Software Technology*, 55(2):395 – 411.

Mtsweni, J. (2012). Exploiting uml and acceleo for developing semantic web service. In *The 7th International Conference for Internet Technology and Secured Transactions (ICITST-2012)*.

Oliveira, A. L., Menolli, A., and Coelho, R. (2008). Separating data access crosscuting concerns using aspectj, a quantitative assessment. In *International Conference on Software Engineering*.

Pérez, J., Ramos, I., Carsí, J. A., and Costa-Soria, C. (2013). Model-driven development of aspect-oriented software architectures. *J. Univers. Comput. Sci.*, 19:1433–1473.

Pinto, M., Fuentes, L., Fernández, L., and Valenzuela, J. (2007). Using aosd and mdd to enhance the architectural design phase. *Springer-Verlag*, 5872:360 – 369. Second International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems.

Przybyłek, A. (2018). An empirical study on the impact of aspectj on software evolvability. *Empir. Software Eng.*, 23:2018–2050.

Sant'Anna, C., Garcia, A., Chavez, C., Chavez, G., Lucena, C., and von Staa, A. (2003). On the reuse and maintenance of aspect-oriented software: An assessment framework. *Brazilian S. on Software Engineering*.

Simmonds, D. M., Reddy, Y., France, R., Ghosh, S., and Solberg, A. (2005). An aspect oriented model driven framework. *Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, pages 119–130.

Singh, Y. and Sood, M. (2009). Models and transformations in mda. In *2009 First International Conference on Computational Intelligence, Communication Systems and Networks*, pages 253–258.

Tekinerdogan, B., Aksit, M., and Henninger, F. (2007). Impact of evolution of concerns in the model-driven architecture design approach. *Electronic Notes in Theoretical Computer Science*, 163(2):45 – 64. Second International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems.

Varela, A., Pérez-González, H., Martínez-Pérez, F. E., and Soubervielle-Montalvo, C. (2017). Source code metrics: A systematic mapping study. *J. Syst. Softw.*, 128:164–197.

Wohlin, C. and Aurum, A. (2015). Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, 20:1427–1455.