Preserving Aspects via Automation: a Maintainability Study

Aram Hovsepyan, Riccardo Scandariato, Stefan Van Baelen, Wouter Joosen
IBBT-DistriNet, Katholieke Universiteit Leuven
Leuven, Belgium
Email: first.last@cs.kuleuven.be

Serge Demeyer
Universiteit Antwerpen
Antwerp, Belgium
Email: first.last@ua.ac.be

***Abstract*—This paper presents an empirical study comparing two alternatives for generating code from aspect-oriented models. In an aspect "disrupting" process, an object oriented implementation in Java is automatically generated from domain specific models, comprising a mix of UML (for core functionality) and DSLs (for qualities like security and performance). In an aspect "preserving" process, an aspect oriented implementation in AspectJ is automatically generated from the same models. In both alternatives, a number of subjects are asked to perform several maintenance tasks requiring the addition and improvement of functionality. The results show that, in most of the cases, the AO alternative provides for shorter maintenance cycles.**

***Keywords*-Experimental study; domain specific modeling; model driven engineering;**

I. Introduction

Separation of concerns is a key software engineering principle to master the increased complexity of today's software applications. This work focusses on aspect oriented techniques. In this context, separation of concerns can be achieved at several levels of abstraction, ranging from requirements (early aspects) to implementation code (aspect oriented programming). At design level, aspect oriented modeling (AOM) is a promising evolution towards a new dimension in the modularization and separation of concerns.

The aspect oriented modeling community, backed up by the advances of the model driven engineering community, advocates model composition as a way to reduce and even eliminate the need for aspect oriented implementation artifacts, such as pointcut specifications. However, in a previous work the authors have provided evidence that preserving the separation of concerns from design to code (by generating an aspect oriented implementation from the aspect oriented models) can lead to improved maintainability with respect to the case of generated object oriented code [1]. At the modeling level, the previous study used Theme/UML [2] as the uniform way of modeling all concerns. Would similar results be observed when a different AOM technique is used?

This paper accounts on a new experimental study that provides answers to the above question. In particular, we analyze the case of domain specific modeling (DSM) as a form of AOM. DSM is a design methodology involving the systematic use of domain specific languages (DSLs) to represent the various facets (i.e., concerns) of a system. DSLs are compact languages providing notations that is very close to the problem domain and quite intuitive for the domain experts and, as a consequence, they reduce and bridge the abstraction gap between the problem and the solution domains. Finally, note that a single DSL is rarely sufficient to describe all the system facets, hence a number of DSLs are typically used in combination to describe all the views of a complex system. In this respect, DSLs and the DSM paradigm provide an approach to separation of concerns and can be regarded as a form of AOM [3], [4].

This study compares two alternatives:

1) Aspect "disrupting" process (OO treatment): An object oriented implementation in Java is automatically generated from domain specific models, comprising a mix of UML (for core functionality) and DSLs (for qualities like security and performance).
2) Aspect "preserving" process (AO treatment): An aspect oriented implementation in AspectJ is automatically generated from the same models.

In both alternatives, a number of subjects are asked to perform several maintenance tasks that require the system to be modified at the modeling level (addition and improvement of functionality), a new version to be generated, and the generated code to be adapted. Ultimately, the experimental study will answer to the following research question: *Are maintenance cycles, on average, shorter when the AO alternative is used?*. The results show that, indeed, the AO alternative is more favorable in most of the cases.

The presented study observes the test subjects from two complementary perspectives. On one hand, the effort spent by the subjects is objectively measured and compared. On the other hand, subjects are administered a series of questionnaires that help understand the results of the experiment.

The rest of the paper is structured as follows. In Section II, we describe the problem statement in detail. In Sections III and IV, we present the design of the experimental user study followed by a statistical analysis of the obtained results. In Section V, we describe the questionnaires and their results. In Section VI, the threats to the validity of this study are discussed. Finally, Sections VII and VIII conclude the paper by presenting the related work and summarizing our findings.

II. Problem Statement Revisited

The two alternatives introduced in Section I and compared further on in this work are explained in detail here. This section also elaborates on the motivations behind the selection of the technologies used in this experiment.

A. Modeling Level

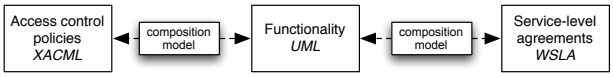


Figure 1. Modeling with DSLs

As described in Figure 1, this paper investigates the case of applications designed via multiple modeling languages. The core functionality of the application is modeled via a UML model, while the access control policies (security concern) and the service level agreements (performance concern) are specified using two DSLs. An integrated DSM approach should allow the developer to select either a suitable DSL for a given concern (if such candidate exists) or to fall back on the default setting of using, e.g., UML. Indeed, a general purpose modeling language like UML remains very suitable for the specification of certain system concerns, such as most functionality related issues.

For specifying the application-level access control policy, XACML is selected, while the service level agreement is modeled via WSLA. The selection of the security and performance concerns is motivated by their key relevance in virtually any realistic software project. The selection of the sub-concerns (access control policies and service level agreements) was mandated by the availability of existing DSLs.

The choice of XACML [5] was motivated by a number of factors. First, XACML is well known, widely accepted, and standardized. It is a declarative DSL that allows the specification of fine grained and context aware access control policies. Moreover, XACML features a reference architecture proposing a standard for the deployment of the necessary software modules within an infrastructure to allow efficient enforcement of all XACML policies. Finally, there is a solid open source implementation for Java that provides a complete support for all the mandatory features of XACML.

Concerning service level agreement (SLA) languages, we have surveyed the literature [6], [7], [8], [9] and selected WSLA [7] based on its maturity, academic status, tool support, available documentation, and feature set. Further, WSLA has been used as the starting point for the design of the upcoming standardized SLA language [10]. Interestingly, WSLA features a reference deployment architecture, as XACML. A complete implementation was not available, and the authors had to implement additional infrastructure to support the enforcement of WSLA agreements. In this respect, the reference architecture was a solid guideline for authors in order to avoid misinterpretations.

Finally, two composition models are used as glue. There is a growing number of approaches that address the problem of DSL composition [11], [12]. However, they require that the input DSLs belong to the same technical space, i.e., share a common metamodel. In this work, we leverage an approach developed by the authors in previous work and that doesn't suffer from the previously mentioned limitations [4]. It is based on the concept of a concern interface, which bridges between concerns expressed in different modeling languages.

B. Implementation Level

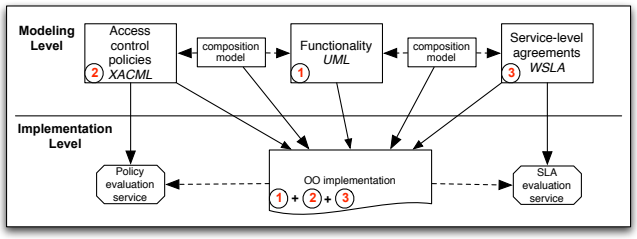
Starting from the same set of models, two different implementations are generated via two tool chains. The code generation is semi-automatic, as behavior is not defined at the modeling level and should be manually implemented in the code. Note that the code generation uses the concept of *protected regions* [13] in order to make sure that subsequent code generation steps do not overwrite any any manually added modifications. Details of the code generation specifics are further described in [4].

The first alternative is depicted in Figure 2(a). The concerns are merged at the modeling level by means of model composition and object-oriented source code is produced. Even though model composition does not necessarily imply the use of object-oriented programming languages, it is the closest match in terms of abstractions used. Traditional AOM approaches (e.g., [14], [15], [16]) implicitly suggest to perform the composition at the modeling level that produces an integrated system view. This eliminates the need for aspect composition at the implementation level.

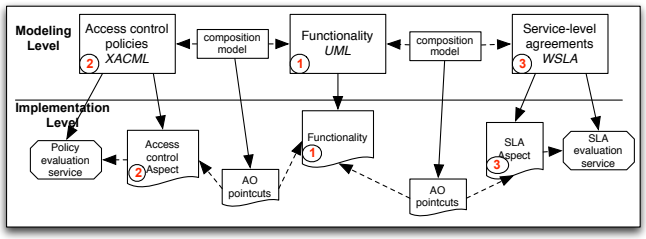
The second alternative is depicted in Figure 2(b). The concerns are kept modularized all the way down to the source code. A model-to-code transformation tool is used to transform each modularized concern to the code level. The composition models are reflected at the code level by a set of aspect-oriented pointcuts. The composition itself is performed by the aspect code weaver that produces a composed byte-level code (not shown on the figure). Several AOM approaches follow this alternative (e.g., see [2], [17]).

Note that in addition to the common modeling approach, both alternatives leverage on reusable components (i.e., a policy evaluation service and a SLA evaluation service) in order to evaluate the access control policies and SLA descriptions. Such components are suggested by the corresponding reference architectures and are preserved in the two alternatives as such.

The choice between the two alternatives is not neutral. Rather, it has a direct impact on several dimensions, such as the productivity of developers, the quality of the final product, the ease of maintenance of the artifacts, and several other matters. In the remainder of this paper we focus



(a) OO treatment



(b) AO treatment

Figure 2. Alternative treatments

on maintainability and we propose a detailed empirical evaluation strategy to compare the two alternatives.

Note that both alternatives start from the same specification at the modeling level. We are interested in investigating the potentially beneficial effect of preserving the separation of concerns from models to code. In this respect, the effort that is necessary at the modeling level in order to carry out the maintenance tasks is (i) common to both alternatives and (ii) a confounding factor. Furthermore, the effort to generate the two alternatives is the same as it is fully automated. Therefore, we have factored out the model modification step in our experiment design.

III. Design of Experimental Study

In order to measure which of the two processes (OO or AO) provides better support over the maintenance cycles, we have mapped the notion of maintainability info an effort metric. The time necessary to execute the maintenance tasks is used to estimate the maintenance effort. We refer to this measure as the EFFORT in the rest of the paper. This metric is measured in a series of controlled experiments executed by a number of experimental subjects.

Note that the EFFORT measure includes both the understanding time and the change execution time. Indeed, the most common approach when performing a maintenance task is to switch back and forth between understanding and changing, thus it is quite difficult to identify (and hence, measure) the two activities apart.

In this section we present the application, the selection of the maintenance tasks, the selection of the subjects and the experiment execution details.

A. Screening Lab Application

For the purposes of our study we have selected an application developed within the context of an e-health research project. The project focuses on a large-scale platform to share clinical information among multiple healthcare providers. The application represents a screening lab system and is used to carry out mammographic images of patients. Screening lab was selected because it is a realistic application from the domain of enterprise applications requiring complex and dynamic access control policies as well as compliance with service-level agreements. Figure 3 illustrates the overall architecture of the application. Initially three web services (*ReadingService, ScreeningService* and *ScreeningSubjectService*) are deployed on the server. Each web service consists of a number of web operations that manipulate data stored in a MySQL database. The service consumers are three controllers (*ReadingController, ScreeningController* and *ScreeningSubjectController*) that create HTML views and send them back to a web-based client. As these web services are accessible via the HTTP, each request is first processed by the authorization engine (*XACML Engine*). The authorization engine parses the access control policies specified in XACML and decides whether a request permission should be granted or denied. Certain operations within the web services should guarantee a certain level of service. *ScreeningSubjectService* should comply with the throughput SLA (i.e., support a minimum number of completed operations per second). *ScreeningService* should comply with both the throughput SLA as well as the response time SLA (i.e., the measured response time should be less than a certain threshold). Hence, relevant SLA parameters should be measured and persisted. Each SLA-compliant web service should carry out the necessary measurements itself. Two internal SLA-related web services (*ThroughputSLAService* and *ResponseTimeSLAService*) are used in order to persist the collected measurements.

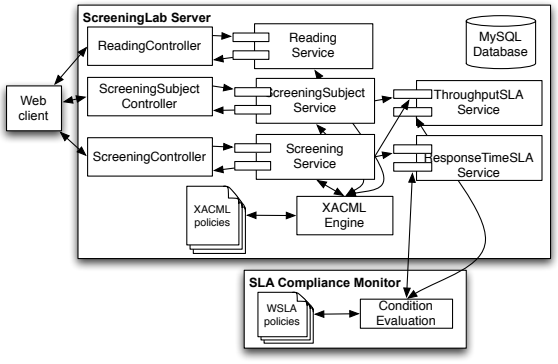


Figure 3. Screening Lab Application: Overall Architecture

Two reference implementations (OO and AO) have been produced starting from the models. First, the two skeletons

Table I
INTERNAL ATTRIBUTES OF THE TWO IMPLEMENTATIONS

Metric	OO	AO
Lines of Code (LOC)	2127	2153
Coupling between components (CBC)	71	75
Concern diffusion over components (CDC)	11	8
Concern diffusion over operations (CDO)	27	24
Concern diffusion over lines of code (CDLOC)	24	12

Table II
SUMMARY OF THE MAINTENANCE TASKS

Task number and description	**Impact**
T1. Introduce a conclusion service that allows radiologists to create conclusions on readings. Introduce a double reading performed by two radiologists. Once the second reading is created, a conclusion should be automatically created.	Modification to the functionality with impact on the access control concern.
T2. Introduce an image postprocessing service that must comply with both response time and throughput SLAs. Modify the implementation of the screening operation so that once created an image is postprocessed before added to the database.	Modification of the functionality with impact on the SLA concern.
T3. Modify the system so that radiographers are granted a fair-use access to the patient data. Each access to patient data should be logged and once the number of accesses exceeds a certain threshold, access should be denied.	Modification to the access control concern with impact on functionality.
T4. The throughput SLA specification is updated where the throughput calculation formula is modified to account for current CPU load. Adjust the system so that CPU load measures are persisted.	Modification to the SLA concern with impact on functionality.

have been generated via the tool chains and, subsequently, the behavior has been manually implemented. As shown in Table I, the two implementations have about the same size (cfr. LOC).

We have assessed the two implementations w.r.t. the internal attributes like coupling [18] and concern diffusion [19]. These are indicators of code quality and are often argued to be predictors of maintainability, although no solid evidence has been provided by studies so far. As shown in Table I, the software metrics are very similar (with the exception of CDLOC) and it would be difficult to promote one alternative on these grounds.

B. Maintenance Tasks

For the experimental study we have used the screening lab application introduced in the previous section. We have designed four maintenance tasks so that the tasks are of comparable (although not equal) difficulty. For the selection of each task the rationale was to have tasks that involve changes to a single concern, however with impact on at least one other concern. As we are dealing with an experiment as the choice of our empirical research strategy each task is relatively confined and requires a limited amount of time to a successful completion. Table II presents each maintenance task in detail and summarizes the impact of the required modifications.

Table III
BACKGROUND OF THE EXPERIMENTAL SUBJECTS

	Subject ID	Years seniority	Experience				
			Java	AspectJ	UML	XACML	WSLA
Group OO	2	2	3	1	3	1	1
	4	5	3	2	2	2	2
	6	2	3	2	2	1	1
	8	4	3	2	3	1	1
	10	2	2	1	2	1	2
	12	4	3	1	2	1	1
	14	0.5	2	1	2	1	1
	16	1	3	2	2	1	1
	17	2	2	1	1	1	1
Group AO	1	1	3	2	3	1	2
	3	5.5	3	2	3	2	1
	5	6	3	3	2	1	1
	7	4	3	2	3	2	1
	9	1	3	3	2	1	1
	11	3	3	3	2	1	1
	13	5	2	2	3	2	1
	15	3	3	3	3	1	1

C. Subjects

For the selection of subjects we have resorted to the *convenience sampling* scheme. We have enrolled 17 volunteering participants, fifteen of whom were post-graduate researchers at our lab and two more post-graduate researchers from the LORE lab at the University of Antwerp. We have screened the subjects using a questionnaire in order to assess their skills over the technology stack that was used throughout the application. Based on the results of this questionnaire the subjects were assigned to two groups, following the OO and the AO treatments respectively. In theory, the learning process of the involved technologies (e.g., WSLA) is not a confounding factor. However, in practice, even if it is, the experiment design makes sure that this confounding factor is evened out over the two groups. Table III presents the subjects' group assignments and their detailed profiles in terms of their knowledge of each technology. The numbers 1, 2 and 3 stand for limited, competent and expert knowledge respectively.

D. Experiment Execution

The design of the experiment described in details in this section has been tested in a pilot phase, prior to the actual experiment execution. During the pilot experiment, the complete experiment setup, including the guiding materials, tutorials and toolset, was validated and fine-tuned. This guaranteed a smooth execution with the actual test subjects. For instance, we have modified the description of task T2, we have corrected the documentation of the database API, etc.

Before the actual experiment started, each subject was given a tutorial on the application architecture, the DSM approach, web services, access control (XACML) and service-level agreements (WSLA). Participants who were assigned to the AO treatment were also given a short recap on AspectJ

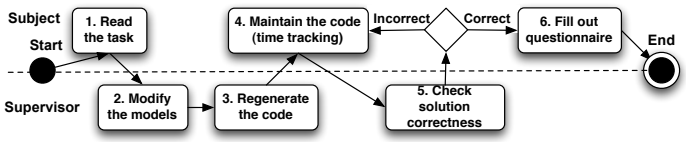


Figure 4. Test workflow

Table IV
DESCRIPTIVE STATISTICS FOR EFFORT (IN SECONDS)

	OO		AO		
Task	Mean	St. dev.	Mean	St. dev.	Reduction of EFFORT
T1	1024.38	352.077	706.29	110.521	31.1 %
T2	645.22	158.680	1239.75	399.351	-92.1 %
T3	1036.11	551.750	459.57	87.753	55.7 %
T4	362.75	93.744	230.63	66.350	36.4 %

syntax.

Prior to the execution of the measured maintenance tasks, each subject was asked to perform two warmup tasks. The first task required the addition of a new role, i.e., a supervisor role who is allowed to perform any action on any resource. For the second task the participants had to make an existing web operation compliant with the response time SLA. The goal of the warmup tasks, which covered a balanced mix of modifications, was to give the participants more hands-on experience with the application and to provide them with sufficient insight into the functionality, access control and SLA concerns.

The subjects participated one by one in supervised experimental tests. The maintenance tasks were executed on a workstation using MagicDraw for modeling and Eclipse for model transformations, code generation and coding. Accordingly with the best practices in empirical software engineering, we have used randomization of the task order in order to improve the precision of the results.

Figure 4 illustrates the workflow of each test. In order to complete a maintenance task assigned in Step 1, one needs to firstly modify the models (Step 2), regenerate the code (Step 3) and then manually complete the produced implementation (Step 4). This study focusses on the effect of preserving separation of concerns from models to code. In that respect, the time necessary to modify the models is not interesting, especially in light of the consideration that both groups work on the same models and the code regeneration steps are identical in both treatments. In fact, the modeling time represents a confounding factor. In order to factor out the differences in EFFORT that might emerge due to subjective modification of the models, in our experimental setup the modifications of all UML, XACML and WSLA models were done by the supervisor. Per each task, the supervisor would perform and explain the modeling adjustments to the subject and generate the code (cfr. lower part of the figure). Afterwards the participant would be asked to complete the maintenance task.

After the completion of each task, automated unit testing was launched by the supervisor (Step 5) in order to ensure the correctness of the produced solution. In case any errors were found, the subject was asked to fix them until the test was passed.

In order to measure the EFFORT, we have developed a web-based time recorder tool that was used by the experiment supervisor to start/pause/stop the time tracking per maintenance task. The timer was started right after the supervisor had regenerated the code and stopped when the subject reported to be ready. At that point the test suite was run. In case of failure, the additional time spent by the subjects in fixing their solution due to a failed test was accounted as well. In total, only 4 errors (2 for each group) have been observed out of 68 executed maintenance tasks and no task was failed more then once by the same subject. Note, also, that the time spent by the supervisor in modifying the model, explaining the changes and regenerating the code was not accounted in the measurements.

We have administered a total of six questionnaires at different points in time. Participants were asked to fill out an initial questionnaire immediately after the warmup tasks. Each task was followed by a questionnaire (Step 6). A final questionnaire was required at the end of the experiment. The purpose and details of all questionnaires is described in section V.

E. Experimental Hypotheses

Per each task, we want to study the correlation between the treatment used (AO or OO) and the EFFORT (time). Specifically, we are interested in understanding whether the AO treatment gives a competitive edge.

For each task, we test the (effective) null hypothesis H_0: $\mu_{OO} = \mu_{AO}$, with alternative hypothesis H_1: $\mu_{OO} > \mu_{AO}$, where μ_{OO} and μ_{AO} represent the average EFFORT for each treatment. Note that the tested null hypothesis is the worst case for the true null hypothesis H_0: $\mu_{OO} \leq \mu_{AO}$.

IV. ANALYSIS OF THE RESULTS

In this section we present the results of the experiment execution followed by their interpretation and a discussion of the threats to the validity of our experiment.

A. Outlier Analysis and Descriptive Statistics

First of all, we have performed an outlier analysis in order to determine the data points that deviate markedly from the rest of the data set. Figure 5 illustrates the outliers by means of box plots. The performance of subject 17 (OO treatment) is an outlier in Tasks 1 and 4. Two more outliers are found for Tasks 1 and 3 (AO treatment). Outliers are removed from the following statistical analysis.

Table IV presents the descriptive statistics for the EFFORT spent per task, in seconds. The results show that the

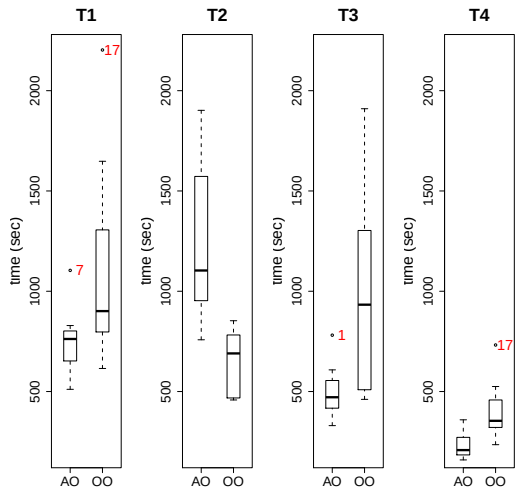


Figure 5. Box plots

Table V
NORMALITY TEST

	Task 1		Task 2		Task 3		Task 4	
	OO	AO	OO	AO	OO	AO	OO	AO
p-values	0.318	0.397	0.161	0.552	0.200	0.918	0.845	0.355

AO treatment performed better in three out of four tasks (T1, T3 and T4) and the effort reduction is substantial (>31%). However, the AO treatment performed much worse in the case of Task 2.

B. Statistical Analysis

The first step towards the selection of an appropriate statistical method is the normality test. Table V presents the results of the Shapiro-Wilk test for normality, which is the most suitable test given the relatively small data set. The test results indicate that the dependent variables are indeed normally distributed ($p > 0.05$). Hence, we can use parametric tests, which are known to have better power.

Table VI presents the results of the "two independent samples" t-test (2-tailed). According to these p-values, we can assert that the above-mentioned conclusions are statistically significant for all tasks. Hence, we can reject the null hypothesis for all the tasks. However, given the descriptive statistics, it is obvious that for task T2 the correlation is in the opposite direction than expected.

Finally, we have also retrospectively investigated the power of the statistical test performed, i.e., the probability that the test rejects a false null hypothesis. Table VII

Table VI
2-TAILED T-TEST

	Task 1	Task 2	Task 3	Task 4
p-values	0.040	0.003	0.014	0.006

Table VII
POST-HOC POWER ANALYSIS

	Task 1	Task 2	Task 3	Task 4
Effect size	1.18	-2.01	1.37	1.63
N_{OO}	8	9	9	8
N_{AO}	7	8	7	8
$1-\beta$	0.56	0.97	0.71	0.86

illustrates the results of the post-hoc power analysis, per task. The commonly accepted value ($1-\beta > 0.8$) is clearly reached for the tasks T2 and T4. The relatively smaller size of the two compared populations (N_{OO} and N_{AO}) is compensated by a very large effect size. For T3 the value is close to the threshold. For T1 the threshold is not met. In this case, the effect size is somewhat smaller (but still ample enough). Hence, a larger population size would increase the power.

C. Interpretation of the Results

The nature of the tasks had an impact on the results and in this section we provide some possible explanations of the obtained results.

The first task (T1) required modifications to system functionality with an impact on the crosscutting access control concern. In particular, the task required that access control check is skipped whenever the web operation is called internally from another web service. This requires a relatively complex conditional statement over the interception by the access control engine. The AO treatment results in lower EFFORT values as AspectJ offers more powerful abstractions to express such conditional statements related to the program execution flow (i.e., the *cflow* construct).

The second task (T2) required modifications to system functionality with an impact on the service-level agreement concern. Although the AO treatment required fewer modifications to the SLA concern, this treatment has resulted substantially higher EFFORT values. The main cause is that the calculation of the response time for one of the methods should have excluded the execution time of certain operations called within the method. Hence, participants needed a very fine-grained control over the method execution. While the Java implementation was very straightforward, the AspectJ code was clearly a challenge to all participants due to the lack of language features for this case. Most of the subjects in the AO group have actually removed the aspect and resorted to the non-modularized Java solution.

Finally, both the third (T3) and the fourth (T4) tasks required modifications of crosscutting nature. The AO treatment required fewer modifications and they were also more modular and "cleaner" compared to the OO treatment. This factor has affected the measurements in favor of the AO treatment for both tasks. This perception comes from the observations of the experiment supervisors and is also confirmed by the subjects in their questionnaires (especially

for T3), as described in Section V-C.

V. Questionnaires

The purpose of administering the questionnaires was twofold. First, we wanted to validate a number of assumptions regarding the experiment setup. If not confirmed, they could result in additional threats to the validity of the experiment. Second, we were looking for explanations supporting the quantitative data collected during the experiment. In this respect, we gathered the opinions of the participants to see whether they confirmed our intuitive understanding about the cause-effect relationship between the treatment and the dependent variable, i.e., the EFFORT.

In total, six questionnaires were administered, namely after the tutorial, after each task and at the end of the whole experiment. Each questionnaire consisted of a series of closed (multiple-choice) questions. Each closed question was formulated as a statement (positive or negative) and the participants had to indicate the degree to which they agreed with it. The available values ranged from 1 to 5, where 1 stands for "strongly disagree" and 5 for "strongly agree". We have deliberately chosen a mix of both affirmative and negative connotations for the statements in order to avoid introducing bias. Further, critical questions (e.g., about the assumptions on the experiment design) have been asked twice, using different statements. Finally, in case the subject was not in agreement with the proposed statement (values 1, 2, or 3 for the positive connotations and 3, 4 or 5 for the negative ones), she was asked to elaborate on the motivations via an open question.

A. Questions

Table VIII presents the questions from the six questionnaires, and the median value of the answers, which separates the higher half of the population from the lower half. Q17 and Q18 ask the participants to indicate which tasks were the most and least difficult to perform. Tables IX and X present for each task how many times the subjects have indicated that a given task is the most and least difficult, respectively.

Table VIII
Questions and median results

	Question	Median OO	Median AO
Initial	Q1. The provided guidance materials (explanation, tutorials and the tool) were clear and sufficient to grasp the overall approach.	5	4
	Q2. The warmup exercises were sufficient to understand the application architecture and design.	4	4
	Q3. The provided explanation was sufficient to obtain a basic understanding of how XACML works within the application.	4	4
	Q4. The provided explanation was sufficient to obtain a basic understanding of how WSLA works within the application.	4	4
After Task1	Q5. I could easily understand the required modifications.	4	4
	Q6. I could easily pinpoint the exact places where changes should occur.	4	4
	Q7. I would revisit my solutions if I had the chance (e.g., extra time).	3	2
After Task2	Q8. I could easily understand the required modifications.	5	5
	Q9. I could easily pinpoint the exact places where changes should occur.	5	3
	Q10. I would revisit my solutions if I had the chance (e.g., extra time).	2	4
After Task3	Q11. I could easily understand the required modifications.	4	4
	Q12. I could easily pinpoint the exact places where changes should occur.	4	4
	Q13. I would revisit my solutions if I had the chance (e.g., extra time).	3	2
After Task4	Q14. I could easily understand the required modifications.	5	5
	Q15. I could easily pinpoint the exact places where changes should occur.	5	5
	Q16.. I would revisit my solutions if I had the chance (e.g., extra time).	2	1
Final	Q17. Which tasks were the most difficult to perform.	see table IX	
	Q18. Which tasks were the easiest to perform.	see table X	
	Q19. After the warm up, I wasn't sure about the overall working of the application.	2	2

Table IX
Reported most difficult task. Frequencies.

Task	OO	AO
T1	6	2
T2	0	8
T3	5	1
T4	0	0

Table X
Reported least difficult task. Frequencies.

Task	OO	AO
T1	1	2
T2	7	0
T3	2	4
T4	9	8

B. Validation of the Experiment Design

This section analyzes the results of the questionnaire responses in the light of the assumptions that we have made concerning the experiment design.

Assumption 1: the provided tutorials and warmup tasks were sufficient to acquire a fair understanding of the overall DSM approach.: The median of the obtained answers to Q1 confirm this assumption. Only one participant from the AO group has given the score of 3 and stated that deeper explanation of the code structure would be useful.

Assumption 2: the provided tutorials and warmup tasks were sufficient to grasp the working of the Screening Lab application in order to perform the maintenance tasks.: In order to backup this assumptions we have used two similar questions. Q2 is affirmative and has been asked after the tutorials and before the execution of the maintenance tasks. Q19 is negative and has been asked at the end of the experiment. The median for both questions is on the positive side and supports the assumption. In more detail, eight participants gave a score of 3 (neutral) to either Q2 or Q19, but only two to both. These participants suggested to increase both the difficulty and the number of the warmup tasks. All participants indicated that after the first experimental task

the working of the application was clear. Hence, any replica of this experiment should consider adding an extra warm up task.

Assumption 3: the provided tutorials and warmup tasks were sufficient to understand how XACML and WSLA work in the application.: This assumption is backed up by the answers to questions Q3 and Q4. One subjects actually suggested to lessen the explanation about the access control concern as it is straightforward.

Assumption 4: the description of all tasks was clear and understandable for every participant.: The median value for Q5, Q8, Q11 and Q14 clearly indicate that the tasks were well understood by all participants.

Assumption 5: all tasks were of comparable difficulty.: Tables IX and X provide answers to question Q17 and Q18 regarding the task difficulty. All subjects in both treatments have indicated that T4 was one of the easiest tasks. In retrospect, we have realized that the most difficult part of this task were the modifications at the modeling level, which were actually performed by the supervisor.
Surprisingly, the rest of the answers seemed to depend on the treatment. The participants using the OO treatment have said T1 and T3 to be difficult. All participants from the AO group have reported T2 to be difficult, which is reported as easy by the participants using the OO treatment.

C. Validation of the Results

Questions Q6, Q7, Q9, Q10, Q12, Q13, Q15 and Q16 indicate the ease of finding the places where changes should occur and the perceived quality of the solution by the participants. Overall, the subjects could easily find the places where modifications were needed (except for Task 2, AO group). This suggests that understandability plays a lesser role in the total effort. The subjects also confident about the quality of their solution except for: (i) Task 1, OO group, (ii) Task 2, AO group, and (iii) Task 3, OO group. The reported quality of the executed tasks is to be used as an extra layer of assurance that goes beyond the checks performed by the automated test suite. The following paragraph elaborate on the motivations provided by the subjects in the diverging cases mentioned above.

Task 1.: The median metrics for the OO group indicate that the subjects were not sure about the quality of their solutions. Their doubts were substantiated as many of them had introduced a security threat. Indeed, i.e., they were using the 'null' subject as a guard condition for adding an automated conclusion object.

Task 2.: Five of the eight subjects from the AO group were unsure where the modifications should occur at the code. Further, six out of eight were unhappy about the quality of their solutions. This is due to the fact that the AspectJ does not offer simple means to deal with very fine-grained execution control flow, which are necessary for this task. Hence, the subjects started looking for a more complex AspectJ constructs and most of them have eventually decided to break up the aspect.

Task 3.: The median for the OO group indicates that the subjects were unsure regarding the quality of their solution. The reported reason lies on the amount of code they had to duplicate.

Task 4.: None of the participants from neither of the groups had problems.

VI. Threats to Validity

We discern between two types of threats to validity applicable to our experiment and we describe them separately.

1) Internal Validity:

- One of the possible threats to the internal validity is the familiarity of each subject with the set of technologies that were used throughout the experiment. However, the average level of knowledge across the treatment groups, both self-reported by the subjects (see Table III) and observed by the experiment supervisor, was approximately the same. Moreover, we have reduced the impact of the required skills to mainly Java and AspectJ, as the modifications at the modeling level were executed by the supervisor.
- Another threat to the internal validity is the subjects' increasing understanding of the the screening lab application, i.e., the learning effect. Although the questionnaires have indicated that the warmup tasks were sufficient to acquire a good understanding of the application architecture and design, eight participants have mentioned that the warmup tasks were possibly insufficient. However, by opting for a randomized order of execution of the tasks we have minimized this threat.
- During the execution of the experiment, the subjects may react differently over time. E.g., subjects may become bored or tired, or they may become more or less positive to one or another treatment. However, we did not notice any serious effects and this threat can be discarded.
- The subjects' knowledge of the purpose of the evaluation is also a threat to the internal validity. Hence, participants may not be impartial to the outcome of the experiment. Given the constant supervision, this threat has been minimized.

2) External Validity:

- For starters, a potential threat is the inclusion of non-professional programmers in the experiment. However, studies have hinted that the differences between students and professionals can be less than one could imagine. For instance, in [20] Runeson states that the differences are significant when comparing undergraduate and graduate students. However, the differences are small between graduate students and professionals.
- For the purposes of internal validity we have eliminated the need for performing any modifications at

the modeling level. Obviously, this choice could affect the external validity of the results. That is, subjects have less options to gain further insight into the code structure while thinking in terms of models. We have tried to minimize this threat by performing the modeling modifications immediately before each maintenance task and by explaining the modifications.

- An important threat to the external validity of the study is the relative small size of the applications and of the maintenance operations. Provided the time-frame we had (1 afternoon per subject) we had to make a trade-off in order to keep the tasks doable. Real-life maintenance operations may take several days or even longer. In addition, longer maintenance operations may require an iterative switching between models and code.
- Another threat is the limited scope of the maintenance operation types (e.g., no refactoring). Also, the chosen tasks might not be representative of real-word situations. However, we have tried to simulate "typical" and "diverse" maintenance operations given the scope of the study (addition and improvement of functionality).
- The results could be specific for the selected technology stack, although as described in section II each technology is a good representatives for the involved discipline. The same reasoning applies to the adoption of the specific AOM approach of this study, but to a smaller extent, as this is the second study we perform with two different modeling techniques.
- Also, the results could be specific to the chosen domains or applications. As illustrated for Task 2, there are examples where AO code does not pay off and, thus, the OO treatment is favored.
- Finally, it could also be difficult to generalize the results to other kinds of concerns not used in this study.

VII. Related Work

The results of this paper complement our previous study [1] where Theme/UML is used as an AOM approach. To the best of our knowledge, there are no other existing empirical studies that evaluate alternative development processes, which transform a modularized design into an implementation as described in section II. In this section we present the existing empirical efforts in the field of AO vs. OO comparisons that are complementary to our work. Two groups of empirical studies involving the AO and OO paradigms are known in the literature.

The first group explores and compares the use of AO against OO programming languages purely from the code perspective. In [21] Tonella and Ceccato assess the effect of the migration of the so-called aspectizable interfaces in Java programs to AspectJ. Their study focuses on maintainability and understandability and concludes that the migration to an AO platform provides externally observable benefits. Hannemann and Kiczales in [22] and Garcia et al. in [23] investigate the improvements of the AspectJ implementation compared to the Java implementation of GoF patterns focusing on code modularity and code reuse. In [24] Cacho et al. study the aspectization of design patterns in the context of existing software systems and evaluate the interactions between pattern implementations. They focus mainly on the scalability of AOP for composing GoF patterns. In [25] the authors present a quantitative and qualitative study that assesses the positive and negative impacts of using an AO language in developing software product lines. The study focuses on design stability in terms of modularity, change propagation and feature dependency. An evaluation of how AO mechanisms support enhanced incremental development and avoid early design degradation is presented in [26]. Hanenberg et al. [27] investigate the development times of crosscutting concerns using OO and AO techniques.

The second group performs a comparison of AO and OO paradigms at the modeling level only. In [28] the authors assess how suitable each paradigm is in the context of distributed real-time and embedded systems in terms of reusability of model elements. In [29] the authors analyze the impact of aspect on conflict resolution in the context of model composition. All these studies conclude that AOM offers benefits compared to the traditional OO modeling with respect to understandability, maintainability and composition conflict resolution.

Our work is situated across the two groups of studies and complements them by investigating which software development process alternative is likely to be more beneficial when it comes to maintainability.

VIII. Conclusions

The aspect oriented modeling community has been implicitly advocating the use of model driven techniques as an implementation strategy for concern composition that reduces the need for aspect oriented artifacts at the implementation level. In this paper we have investigated whether an aspect "preserving" process relying on an AO implementation offers benefits in the context of maintainability compared to a "disrupting" process that generates a streamlined OO code. Based on an DSM approach that leverages on the use of UML for the specification of the core functionality and DSLs for the specification of non-functional concerns (such as, access control and service level agreements) we have discovered that in three out of four cases the AO alternative shortens the maintenance cycles.

Obviously, we cannot draw too-broad conclusions from this experiment. However, these results are in line with the findings of our previous empirical study based on the use of Theme/UML. This initial, combined evidence clearly calls for the inception of a research path in this area, e.g., to further investigate and characterize the maintenance patterns that influence the results.

ACKNOWLEDGMENT

This work is partially funded by the Flemish government institution IWT, by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

REFERENCES

[1] A. Hovsepyan, R. Scandariato, S. V. Baelen, Y. Berbers, and W. Joosen, "From aspect-oriented models to aspect-oriented code? the maintenance perspective," in *9th International Conference on Aspect-Oriented Software Development*, 2010.

[2] E. Baniassad and S. Clarke, *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, 2005.

[3] S. Cook, "Separating concerns with domain specific languages," in *Modular Programming Languages*, ser. Lecture Notes in Computer Science, 2006, vol. 4228.

[4] A. Hovsepyan, S. Van Baelen, Y. Berbers, and W. Joosen, "Specifying and composing concerns expressed in domain-specific modeling languages," in *47th International Conference Objects, Models, Components, Patterns*, 2009.

[5] OASIS, "Core specification: Extensible access control markup language (XACML) v2.0."

[6] S. Frølund and J. Koistinen, "Quality of services specification in distributed object systems design," in *4th USENIX Conference on Object-Oriented Technologies and Systems*, 1998.

[7] IBM Corporation, "Web service level agreement (WSLA) language specification," http://www.research.ibm.com/wsla/.

[8] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A language for defining service level agreements," in *9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, 2003.

[9] V. Tosic, H. Lutfiyya, and Y. Tang, "Web service offerings language (WSOL) support for context management of mobile/embedded XML web services," in *International Conference on Internet and Web Applications and Services*, 2006.

[10] Grid Resource Allocation Agreement Protocol Working Group, "WS-Agreement specification," http://www.ogf.org.

[11] M. Emerson and J. Sztipanovits, "Techniques for metamodel composition," in *DSM Workshop*, 2006.

[12] A. Zito, Z. Diskin, and J. Dingel, "Package merge in uml 2: Practice vs. theory?" in *9th International Conference on Model Driven Engineering Languages and Systems*, 2006.

[13] SINTEF, "MOFScript," http://modelbased.net/mofscript/.

[14] O. Barais, J. Klein, B. Baudry, A. Jackson, and S. Clarke, "Composing multi-view aspect models," in *7th International Conference on Composition-Based Software Systems*, 2008.

[15] A. Hovsepyan, S. Van Baelen, Y. Berbers, and W. Joosen, "Generic reusable concern compositions," in *4th European Conference on Model Driven Architecture Foundations and Applications*, 2008.

[16] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg, "Directives for composing aspect-oriented design class models," *Transactions on AOSD*, vol. 3880, 2006.

[17] S. Hanenberg, D. Stein, and R. Unland, "From aspect-oriented design to aspect-oriented programs: tool-supported translation of JPDDs into code," in *International Conference on Aspect-Oriented Software Development*, 2007.

[18] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[19] C. Sant'anna, A. Garcia, C. Chavez, C. Lucena, and A. von Staa, "On the reuse and maintenance of aspect-oriented software: An assessment framework," in *Brazilian Symposium on Software Engineering*, 2003.

[20] P. Runeson, "Using students as experiment subjects - an analysis on graduate and freshmen student data," 2003.

[21] P. Tonella and M. Ceccato, "Refactoring the aspectizable interfaces: An empirical assessment," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 819–832, 2005.

[22] J. Hannemann and G. Kiczales, "Design pattern implementation in Java and AspectJ," in *Conference on Object-Oriented Programming, Systems, Languages and Applications*, 2002.

[23] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. von Staa, "Modularizing design patterns with aspects: A quantitative study," in *International Conference on Aspect-Oriented Software Development*, 2005.

[24] N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista, and C. Lucena, "Composing design patterns: a scalability study of aspect-oriented programming," in *5th International Conference on Aspect-Oriented Software Development*, 2006.

[25] E. Figueiredo, N. Cacho, and C. Sant'Anna, et al., "Evolving software product lines with aspects: an empirical study on design stability," in *30th International Conference on Software Engineering*, 2008.

[26] P. Greenwood, T. Bartolomei, E. Figueiredo, and A. Garcia, et al., "On the impact of aspectual decompositions on design stability: An empirical study," in *21st European Conference on Object-Oriented Programming*, 2007.

[27] S. Hanenberg, S. Kleinschmager, and M. Josupeit-Walter, "Does aspect-oriented programming increase the development speed for crosscutting code? an empirical study," in *International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 156–167.

[28] M. Wehrmeister, E. P. Freitas, D. Orfanus, C. E. Pereira, and F. J. Rammig, "Evaluating aspect and object-oriented concepts to model distributed embedded real-time systems using RT-UML," in *17th IFAC World Congress*, 2008.

[29] K. Farias, A. Garcia, and J. Whittle, "Assessing the impact of aspects on model composition effort," in *9th International Conference on Aspect-Oriented Software Development*, 2010.