Evaluating the Impact of Aspects on Inconsistency Detection Effort: A Controlled Experiment

Kleinner Farias, Alessandro Garcia, and Carlos Lucena

OPUS Research Group/LES, Informatics Department, PUC-Rio
Rio de Janeiro - RJ - Brazil
{kfarias,afgarcia,lucena}@inf.puc-rio.br

Abstract. Design models represent modular realizations of stakeholders' concerns and communicate the design decisions to be implemented by developers. Unfortunately, they often suffer from inconsistency problems. Aspect-oriented modeling (AOM) aims at promoting better modularity. However, there is no empirical knowledge about its impact on the inconsistency detection effort. To address this gap, this work investigates the effects of AOM on: (1) the developers' effort to detect inconsistencies; (2) the inconsistency detection rate; and (3) the interpretation of design models in the presence of inconsistencies. A controlled experiment was conducted with 26 subjects and involved the analysis of 520 models. The results, supported by statistical tests, show that the effort of detecting inconsistencies is 20 percent lower in AO models than in their OO counterparts. On the other hand, the inconsistency detection rate and the number of misinterpretations are 43 and 37 percent higher in AO models than in OO models, respectively.

Keywords: Aspect-Oriented Modeling, Model Composition, Inconsistency, Developer Effort, Empirical Studies.

1 Introduction

Modeling languages (e.g., UML [11] and its extensions) provide different types of models, such as class and sequence diagrams, to represent the structure and behavior of software systems. These complementary models represent the design decisions that developers will implement later. In practice, these models often suffer from the inconsistency problems [16]. These inconsistencies are mainly caused by the mismatch between the overlapping parts of complementary models and the lack of formal semantics to prevent these contradictions [2][3]. Consequently, developers must invest some effort to detect and properly deal with these inconsistencies [6]; otherwise, emerging misinterpretations of the design models can compromise the resulting implementation.

Different modeling languages support different forms of modular decomposition and may influence how developers detect or even neglect inconsistencies [3]. This might be particularly the case with aspect-oriented modeling (AOM) [7][17] as it intends to improve design modularity of otherwise crosscutting concerns. Current

R.B. France et al. (Eds.): MODELS 2012, LNCS 7590, pp. 219–234, 2012.
© Springer-Verlag Berlin Heidelberg 2012

research in AOM varies from UML extensions [7][17][19][20] to alternative strategies for model weaving. Unfortunately, nothing has been done to investigate whether aspect-oriented models can alleviate the burden of dealing with model inconsistencies. Someone might hypothesize that they might help developers to understand the design before implementing it. Others could also postulate that the improved modularization would reduce the effort to detect inconsistencies or even reduce misinterpretations arising between complementary design models.

Unfortunately, it is by no means obvious whether these assumptions hold (or not). First, it may be the case that additional constructs in AO models lead to detrimental effects on design understanding. Second, it is still not clear if an aspect affecting multiple join points may increase the inconsistency detection and improve the model interpretation. Third, developers might get "distracted" by the global reasoning motivated by the presence of crosscutting relations [10] between classes and aspects. At last, developers might even invest more effort using AO models while examining all points that are crosscut by the aspects [6].

In this context, this paper reports a controlled experiment (Section 3) aimed at investigating the impact of AOM on: (1) the rate of inconsistency detection; (2) the developers' effort to detect these inconsistencies; and (3) developers' misinterpretation rate. We compare the use of AO models to OO models in a particular context: the use and understanding of design models by developers needed to produce the corresponding implementation. The results (Section 4) supported by statistical tests and qualitative analysis, show that AO models alleviate the effort to detect inconsistencies. But, it neither reduces inconsistency detection rate nor misinterpretation rate.

Moreover, we also discuss some additional findings (Section 4.4). For instance, we observed that the downsides of AOM were, to a large extent, caused by the degree of quantification [10] of the aspects. That is, the higher the number of modules affected by an aspect, the lower the inconsistency detection rate and the higher the misinterpretation rate. Moreover, we observed that developers tended to detect inconsistencies more quickly in AO models when the scope of aspect pointcuts was narrow. Equally relevant was the finding that the required mental model is directly influenced by the number of crosscut relationships.

To the best of our knowledge, our results are the first to pinpoint the potential (dis)advantages of AOM in imprecise multi-view modeling. After presenting how we tried to mitigate the possible threats to validity (Section 5), we make it clear the contributions of our experiment in the light of the related work (Section 6) and present final remarks (Section 7).

2 Background

2.1 Aspect-Oriented Modeling

Aspect-oriented modeling (AOM) languages aim at improving the modularity of design models by supporting the modular representation of concerns that cut across multiple software modules. This superior modularization of crosscutting concerns is achieved by the definition of a new model element, called *aspect*. An aspect can

crosscut several modules within a system. These relations between aspects and other modules are called *crosscut* relationships. These basic concepts and other aspect-oriented modeling elements are usually represented as classic UML stereotypes in AOM languages [7][17]. The AOM language used throughout our study is a UML profile [17][19][20]. The choice of the UML profile for AOM is based on some reasons. First, the Unified Modeling Language [11] is the standard for designing software systems. Second, the use of stereotypes reduces the gap between subjects with low skill (or experience) and highly skilled (or experienced) subjects. Third, the model reading technique used by the subjects would not be influenced by new notation issues; therefore, the interpretation of the models is exclusively influenced by the use of the concepts in object-oriented and aspect-oriented modeling. Finally, UML profile for AO programming is the approach more common for structural and behavioral diagrams [11].

Fig. 1 presents an illustrative example of the models used in our study: a class and a sequence diagram of the AOM language used in our study. The notation supports the visual representation of aspects, crosscutting relationships and other AOM concepts. The stereotype <<aspect>> represents an aspect, while the dashed arrow decorated with the stereotype <<crosscut>> represents a crosscutting relationship. Inner elements of an aspect are also represented, such as *pointcut* (<<pointcut>>) and advice. An advice adds behavior before, after, or around the selected join points [7]. The stereotype associated with an advice indicates when (<<before>>, <<after>> or <<around>>) a join point is affected by the aspect. The join point is a point in the base element where the advice specified in a specific pointcut is applied.

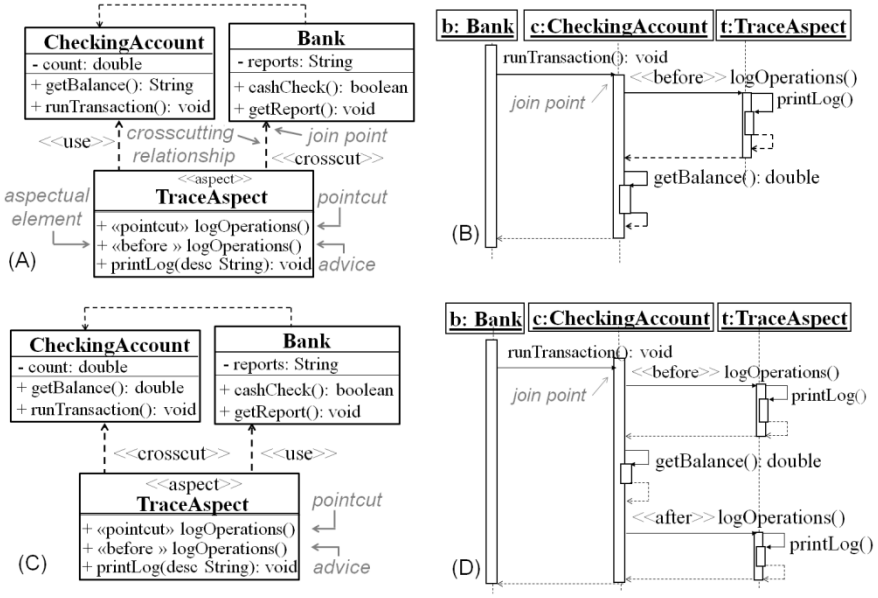

Fig. 1. An illustrative example of aspect-oriented models used in our study. (A) and (B) represent the conflicting structural diagrams. (C) and (D) represent the structural and sequence diagrams without inconsistencies.

2.2 Model Inconsistency and Detection Effort

The multiple views of a software system inevitably have conflicting information [2]. If software developers do not detect and properly deal with these inconsistencies the potential benefits of the use of the models (e.g., gain in productivity) can be compromised. Developers must invest some considerable effort (time) to detect these inconsistencies; otherwise, the potential benefits of the use of models such as specification of the implementation of a system can be compromised. Two broad categories of inconsistencies were used in this study: (1) *syntactic inconsistencies*, which arise when the models not conforming to the modeling language's metamodel; and (2) *semantic inconsistencies*, where the meaning of the model element does not match that of the actual design model. We have particularly selected semantic inconsistencies that are: (i) detectable by developers [2], and (ii) difficult or impossible to detect automatically. We focused on inconsistencies that have been documented elsewhere [3] and used in a previous empirical study [2]. A complete description is also available at our complementary website [9], and two representative examples are presented below:

1) *Conflicting relationships*: the nature of a relationship diverges in structural and behavioral models. For instance, according to the sequence diagram, the advice of an aspect A crosscuts the behavior of class B; however, the semantics of the advice in A dictates when the class diagram should have either a <<crosscut>> or a <<use>> relationship between A and B. For example, Fig. 1 presents this kind of inconsistency. The aspect *t:TraceAspect* crosscuts the *c:CheckingAccount* objects (Fig. 1.B). In this case, the relationship between *TraceAspect* and *CheckingAccount* should be <<crosscut>> instead of <<use>> (see Fig. 1.C) given the logging semantics of the advice *logOperations()*. In the structural diagram (Fig. 1.A), the aspect *TraceAspect* has a <<use>> relationship with the class *CheckingAccount* instead of <<crosscut>> relationship.

2) *Messages with different return types*: the return type of a message *m* from an object A to an object B does not match with the return type of the method M in the corresponding class B in the class diagram. For instance, the method *CheckingAccount.getBalance* has conflicting return types: *string* in the class diagram and *double* in the sequence diagram. A similar conflict can occur with the return type of a around advice [17] and the return type from a method execution being advised by the latter.

Developers detect inconsistencies when they identify conflicting information in the models and, then, report that the models cannot be implemented. This decision often relies on "guessing" the semantics of the model elements. To reach this conclusion, developers need to invest some effort: the time (in minutes) to go through the model and infer that the models suffer from inconsistencies.

3 Study Methodology

3.1 Goal, Research Questions, and Context

We formulate the goal of this study using the GQM template [5] as follows:

Analyze AO and OO modeling techniques *for the purpose of* investigating the impact *with respect to* detection effort and misinterpretation *from the perspective of* developers *in the context of* multi-view design models.

Based on this, we focus on the three research questions:

RQ1: Does AOM affect the efficiency of developers to detect multi-view model inconsistencies?
RQ2: Does AOM influence effort invested by developers to detect model inconsistencies?
RQ3: Do AO models lead to a different misinterpretation rate as compared to OO models?

The context selection is representative of situations where developers implement classes (or aspects) based on design models. The experiment was conducted within two postgraduate courses at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and the Federal University of Bahia (UFBA). Both courses are taught in the first year of Master and Doctoral programs in Computer Science. Therefore, all the subjects (18) hold a Master's or Bachelor's degree, or equivalent. In addition, eight (8) professionals from three companies also participated in the experiment. Most of the professionals held a Master's or Bachelor's degree.

3.2 Hypothesis Formulation

First Hypothesis (H_1). The first research question investigates whether developers by using AO models produce a lower (or higher) inconsistency detection rate than by using OO models. Usually developers do not indicate the presence of existing inconsistencies in multi-view models [3]. The main reason is that they can make implicit assumptions about the correct design decisions based on previous experience. Moreover, they might feel forced to produce an implementation even in the presence of inconsistency. Thus, our intuition is that developers identify fewer inconsistencies in AO models than OO models because they might get distracted by the global reasoning motivated by the presence of additional crosscutting relations in the models. Consequently, they may have a higher number of implicit assumptions to assemble the "big picture" of a system. However, it is by no means obvious that this hypothesis hold. Perhaps, the increased modularity of AOM models may help developers to switch more quickly between the behavioral and structural views while implementing their aspects. Consequently, the software developer may localize more inconsistencies than in OO models. These hypotheses are summarized as follows:

***Null Hypothesis 1, $H_{1\text{-}0}$*:** The inconsistency detection rate in AO models is equal or higher than in OO models.
$H_{1\text{-}0}$: DetectionRate (AO) $\geq$ DetectionRate (OO)
Alternative Hypothesis 1, $H_{1\text{-}1}$: The inconsistency detection rate in AO models is lower than in OO models.
$H_{1\text{-}1}$: DetectionRate (AO) < DetectionRate (OO)

Second hypothesis (H_2). The second research question investigates whether developers invest less (or more) effort to detect inconsistencies in AO models than OO models. The superior modularity of AO models may help developers to better match and contrast the structural and behavioral information about the crosscutting relations. In

this case, developers may switch more quickly between the behavioral and structural views while systematically implementing their aspects. Thus, our expectation is that the higher the number of crosscutting relationships (an aspect crosscutting a wider scope) in the model, the lower the effort to detect inconsistencies. This assumption is based on the superior ripple effects of inconsistencies observed in AO models when model composition techniques are applied [6]. This propagation can directly affect the effort in detecting inconsistencies, since developers, facing the complexity of the propagations, avoid doing any implementation. That is, by using AOM developers tend to get more quickly convinced about the severity of multi-view inconsistencies. This means that they are more likely to report them and not going forward on the design implementation. However, it is not clear whether this intuition holds because, at first, developers may examine all model elements affected (or not) by the inconsistencies, or even the inconsistencies, to some extent, may even be confined in the aspectual elements. This leads to the second null hypothesis and an alternative hypothesis as follows:

Null Hypothesis 2, $H_{2\text{-}0}$: The effort to detect inconsistencies in AO models is equal or higher than in OO models.
$H_{2\text{-}0}$: EffortToDetect (AO) ≥ EffortToDetect (OO)
Alternative Hypothesis 2, $H_{2\text{-}1}$: The effort to detect inconsistencies in AO models is lower than in OO models.
$H_{2\text{-}1}$: EffortToDetect (AO) < EffortToDetect (OO)

Third hypothesis (H_3). The third research question investigates whether the misinterpretation rate (MisR) of the developers is higher (or lower) in AO models than in OO models. The chief reason of the disagreement between developers' interpretation is the contradicting understanding of the design models. They are often caused by inconsistencies emerging from the mismatches between the diagrams specifying the multiple, complementary views of the software system [3]. Contradicting design models make it difficult for developers to think alike and, hence, producing code with the same semantics. The key reason is that software implementation widely depends on cognitive factors. Someone could consider that additional AOM concepts, such as crosscutting relationships or aspects, may negatively interfere in a common understanding of design models by different developers. For instance, developers need to precisely grasp the actual meaning of the crosscutting relations (in addition to all other relations), and when they are actually established during the system execution. Then, as developers have to examine all join points affected by the aspects, their extra analyses can increase the opportunities of diverging interpretations. However, this expectation might not hold because the crosscutting modularity may improve the overall understanding of the design a when compared to pure OO models. This would lead to the following null and alternative hypotheses:

Null Hypothesis 3, $H_{3\text{-}0}$: The misinterpretation rate (MisR) in AO models is equal or higher in AO models than in OO models.
$H_{3\text{-}0}$: MisR(AO) ≥ MisR(OO)
Alternative Hypothesis 3, $H_{3\text{-}1}$: The misinterpretation rate in AO models is lower than in OO models.
$H_{3\text{-}1}$: MisR(AO) < MisR(OO)

3.3 Experiment Design

Selection of subjects. Subjects (18 students and 8 professionals) were selected based on two key criteria: the level of theoretical knowledge and practical experience related to software modeling and programming. The subjects studied in educational systems that place a high value on key principles of software modeling and programming. In addition, the subjects were exposed to more than 120 hours of courses (lectures and laboratory) exclusively dedicated to software design, software modeling, OO programming, and AO software development. It can be considered they underwent an intensive modeling-specific and programming training. As far as practical knowledge is concerned, the main selection criterion was that subjects had, at least, 2 years of experience with software modeling and programming acquired from real-world project settings.

Paired comparison design. All subjects were submitted to two treatments (AO and OO modeling) to allow us to compare the matched pairs of experimental material. Each treatment had a questionnaire with five multiple-choice questions. The first treatment had only questions with AO models while the second one had only questions with OO models. The subjects were assigned *randomly* and *equally* distributed to these treatments so that the effects of the order could be discarded. Therefore, the experimental design of this study is by definition a *balanced design*.

As the subjects were submitted to two treatments, an ever-present concern was the information that the subject could gain from the first treatment to perform the experiment with the second treatment. To minimize the "gain in information," some experimental strategies [4][5] were followed. First, the models used in the study were fragments of class and sequence diagrams from realistic, industrial design models of different application domains. Hence, the subjects had no prior information and no accumulated knowledge about the semantics of the model elements. Second, each question had a class and sequence diagram representing different functionalities of a software system. Third, each pair of structural and behavioral models had different kinds of inconsistencies (Section 2.2), and the meanings of their elements were completely different. Therefore, we can assume that the performance of subjects was not influenced by the treatments of previous questions.

*Tasks***.** In both treatments, the subjects received a pair of corresponding class (structural) and sequence (behavioral) diagrams. They were asked how they would implement particular classes (or aspects) based on these diagrams. That is, rather than stimulated to review or inspect the diagrams, the subjects were encouraged to implement particular model elements (classes or aspects). The goal is to identify how developers would deal with inconsistencies in the context of concrete software engineering tasks. The subjects should choose, then, the most appropriated implementations between the five possible answer options. In each question, the subjects were required to register the time invested to answer the question ("start time" and "end time"). They were also stimulated to justify their answers on the answer sheet. In total, ten questions were answered. After the experiment, the subjects were also interviewed to clarify the results.

*Objects***.** In the questions of the first treatment, the OO class diagram had, on average, 7 classes and 8 relationships, while in the second treatment the questions had an AO class diagram with, on average, 5 classes and 2 aspects, and 8 relationships. The cor-

responding AO and OO sequence diagrams had, on average, 5 objects and 15 messages between the objects (and/or aspects). Each pair of OO or AO diagrams had two kinds of inconsistencies. The inconsistencies were always related to contradictions between the class and sequence diagrams. That is, there was conflicting information between those diagrams, as the examples given in Section 2.2. Considering the answer options in each question, they were planned according to the following schema. The first answer option is according to the class diagram while the second one is just according to the sequence diagram. The third answer option is based on the combination of the information presented in both diagrams. The fourth one is incorrect considering all two diagrams. All questions had a fifth answer option where the subjects could indicate that an inconsistency was detected in the models. The subjects were encouraged to carefully explain their answers. Further details of the experimental design can be found in [9].

3.4 Variables and Quantification Method

The independent variable of this study is the choice of the modeling language. It is nominal and two values can be assumed: AO modeling and OO modeling. These variables describe the treatments, and we investigate their impacts on following dependent variables.

Inconsistency detection rate (Rate) and Inconsistency detection effort (Effort). The *Rate* variable is intended to measure the overall rate of inconsistencies detected by all subjects (RQ1). It represents the ratio of the number of subjects that detect inconsistencies in a question divided by the number of subjects that answer the question without notifying the presence of inconsistency. The *Effort* variable represents the mean of time (minutes) spent by the subjects to detect inconsistencies in a question (RQ2). Note that subjects detect inconsistencies when they explicitly indicate that they are unable to achieve a suitable implementation from the contracting diagrams.

Misinterpretation rate (MisR). This variable represents the degree of variation of the answers (RQ3). That is, it measures the concentration of the answers over the four possible alternatives (the fifth alternative represents the detection of inconsistency). Our concern is if the differences in (un)detected inconsistency affects the design interpretation of the subjects. An undetected inconsistency is not necessarily problematic [3] if all subjects have the same interpretation. For example, if the 26 subjects have the same answer (e.g., the alternative "A") for a question, then the inconsistencies in the diagrams did not lead to misinterpretations (MisR = 1). On the other hand, if the developers' answers spread equally over the four alternatives, then the inconsistencies cause serious misinterpretations (MisR = 0). That is, the misinterpretation rate is 0 if answers are distributed equally over all options, and 1 if the answers are concentrated only one answer option. According to [3], this variable can be measured as follows.

$$MisR(k_0, \ldots, k_{K-1}) = 1 - 2\,\frac{\sum_{0 \le i < K} k_i i}{N(K-1)}$$

Where:
K: The number of alternatives for a question

k_i: The number of times alternative i was selected, where $0 \leq i < K$ and
(for all i : $0 \leq i < K - 1 : k_i \geq k_{i+1}$)
N: The sum of answers over all alternatives: $N = \sum_{0 \leq i < K} k_i$

3.5 Operation

Preparation phase. The subjects (students and professionals) were not aware of the research questions (and hypotheses) of our study in order to avoid biased results. The motivation of the students was to gain extra points for their grade. The results obtained by the students had no effect on their grade; instead, their dedication and quality of the justifications of the sheet and interviews. The professionals received the same questions as a printable questionnaire. All subjects received a refresher training to be sure of their familiarity with the modeling concepts used in the study.

Execution phase. The experiment tasks were run within two courses at two different Brazilian universities (PUC-Rio and UFBA). Both runs were carried out in a classroom following typical exam-like settings. However, because of time constraints and location, the professionals run the experiment in their work environment. However, the experiment was carefully controlled. All subjects received 10 questions and the answer sheets. It is important to point out that there was no time pressure for the subjects, but they were rigorously supervised to correctly register the time. Therefore, we are confident that the time was recorded properly. For clarification reasons, the subjects were encouraged to justify their answers. After finishing the experiment, the subjects filled out a questionnaire to collect their background, i.e. their academic background and work experience. Observational studies were conducted to improve understanding how the tasks in the experiment were performed,. This allowed a more effective observation and monitoring of the tasks of the subjects. To obtain an additional feedback from the subjects, they were also encouraged to write down the rationale used to answer the questions.

Interview phase. Additionally, a semi-structured interview approach [5] was performed, which followed a funnel model, i.e. one initial open question was presented and followed by more specific ones. It was organized in topics with open and close questions in such a way that qualitative evidence on the research questions could be gathered. An interview guide was created based on the authors' experience and the study design. The interviews were recorded and transcribed into text. All subjects were selected for interviews. Each interview lasted from 30 to 55 minutes, depending on how talkative the subjects were.

4 Experimental Results

4.1 RQ1: Detection Rate in AO and OO Models

Descriptive Statistics. The first research question investigates if developers detect more (or less) inconsistencies in AO models or OO models. Developers detected more inconsistencies in OO models than AO models. The superior detection rate in OO models can be explained comparing means and medians (Table 1). Developers detect, on

average, by about 43.24 percent more inconsistencies in OO models than AO models, i.e. a mean of 0.37 (AO) compared with a mean of 0.53 (OO). The difference observed between the medians also favors the OO models. This comprises a superiority of 42.85 percent in the number of the cases in which developers reported to be unable to provide an implementation. The results suggest that OO models enable developers to identify more inconsistencies than AO models. This contradicts somehow the intuition that the improved modularity of AOM helps developers to localize inconsistencies.

Hypothesis Testing. Since the Shapiro-Wilk and Kolmogorov-Smirnov normality tests [1] indicates that the data are normally distributed, the paired t-test is applied to test H_1. The collected *t-statistic* is 4.03 with the *p-value* = 0.01 (Table 1). This small *p-value* (< 0.05) indicates the first null hypothesis ($H_{1\text{-}0}$) can be rejected. This implies that the average difference of the detection rate in AO and OO models is not zero. Therefore, there is strong evidence (at the 0.05 level significant) that developers detect more inconsistencies in OO models than in AO models. The mean differences between the pairs of AO and OO models indicate the direction in which the result is significant. For example, considering the varying detection rate for AO and OO models, the mean difference is negative (-0.16). This implies that the detection rate in AO models was statistically lower than in OO models. Moreover, the non-parametric Wilcoxon test is applied to eliminate any threat related to statistical conclusion validity. The low value of the p-value = 0.031 collected (< 0.05) also confirmed the aforementioned conclusion. Therefore, we can reject the null hypothesis $H_{1\text{-}0}$.

Table 1. Descriptive statistics and Stastical tests for measures

Variables	Treat.	Mean	St Dev	Min.	25th	Med.	75th	Max	%diff	Wilcoxon	Paired t-test		
										p-value	*t*	*p*	*MD*
Detection	AO	0.37	0.09	0.23	0.29	0.35	0.46	0.54	43.24	0.031	4.03	0.01	-0.16
	OO	0.53	0.11	0.38	0.42	0.5	0.67	0.69					
Effort	AO	5.28	1.67	4	4.08	4.22	7	7.8	19.69	0.033	3.1	0.03	-1.48
	OO	6.32	1.57	4.33	5.06	6.08	7.71	8.65					
MisR	AO	0.51	0.07	0.38	0.45	0.52	0.57	0.58	37.25	0.029	2.94	0.04	-0.19
	OO	0.7	0.07	0.62	0.64	0.69	0.77	0.81					

*with 4 degree of freedom, a significance level of α = 0.05, MD: mean difference, p: p-value, St Dev: standard deviation

4.2 RQ2: Detection Effort in AO and OO Models

Descriptive Statistics. The second research question investigates the effort that developers should invest to detect inconsistencies in AO and OO models. Developers spend more effort to detect inconsistencies in OO models than AO models. The mean of detection effort is 5.28 (minutes) in AO models and 6.32 in OO models. This comprises a representative increase of 19.69 percent against plain UML models. This lower effort in the use of AOM is also observed comparing the medians. The detection effort ranges from 4.22 (minutes) in AO models to 6.08 in OO models, which

represents an increase of 44.07 percent in the latter case. This phenomenon confirmed our initial intuition that the superior modularity of AO models would accelerate the inconsistency detection. In fact, during the interviews, the subjects (18) reported that the manifestation of inconsistencies in crosscutting relations made the implementation to be prohibitive. Hence, the subjects reported more quickly in the AO model than in OO models. We noticed they were keener to match and contrast the structural and behavioral information governing the crosscut relations. Therefore, developers often report conflicting crosscutting relations as the reason for not progressing towards the implementation. This implies that although developers detect fewer inconsistencies in AO models, they spend less effort to localize them.

Hypothesis Testing. Since the Shapiro-Wilk and Kolmogorov-Smirnov normality tests [1] indicate that the data are normally distributed, the paired t-test is applied to test H_2. The collected *t-statistic* is 3.1 with the *p-value* = 0.03 (Table 1). This small *p-value* (< 0.05) indicates the second null hypothesis ($H_{2\text{-}0}$) can be rejected. This suggests that the average difference of the inconsistency detection effort in AO and OO models is not zero. Thus, there is strong evidence (at the 0.05 level significant) that developers invest more effort to detect inconsistencies in OO models than in AO models. The detection effort in AO and OO groups assumes a negative value for the mean difference (-1.48), while the p-value (0.03) is less than 0.05. This implies that detection effort in OO models is statistically higher than in AO models. Moreover, the non-parametric Wilcoxon test is applied to eliminate any threat related to statistical conclusion validity. The low value of the p-value collected (0.033) also confirmed the previous conclusion. Therefore, we can reject the null hypothesis $H_{2\text{-}0}$.

4.3 RQ3: Misinterpretation Rate in AO and OO Models

Descriptive Statistics. The third research question investigates whether AO models lead to a higher or lower misinterpretation rate than OO models. Table 1 shows the descriptive statistics to the misinterpretation measures of AO and OO models. Recall that MisR varies between 0 and 1, and that MisR = 1 indicates that developers do not have misinterpretation. On the other hand, MisR = 0 indicates that the developers' answers spread equally over the four different alternatives, which represent the most serious misinterpretations. OO models cause less misinterpretation (higher MisR value) than AO models. The misinterpretation rate is 37.25 percent lower in OO models; the mean is 0.51 in AO groups against 0.7 in OO groups. This upward trend is also observed in the medians: 0.52 in AO models against 0.68 in OO models, comprising an increase of 32.69 percent. The results suggest that the presence of inconsistencies in AO models entails a higher detrimental impact on model interpretation by developers than in OO models. Our initial expectation that by using contradicting AO design models would increase the number of diverging interpretations was confirmed. During the interviews and examining the answer sheets, the subjects (22) reported that the need to scan all join points affected by the aspects increased the likelihood of different interpretations.

Hypothesis Testing. We analyze the strength of the aforementioned result testing H_3 as follows. As in the previous analysis, the paired t-test is applied to test H_3 as the

measures assume a normal distribution. Table 1 shows the pairwise p-values and mean differences across pairs for each measure. As the mean difference is negative (-0.19) and p-value (0.04) is less than 0.05, we can conjecture that there is significant evidence that the number of diverging interpretations in AO models is statistically higher than in OO models. We also applied the non-parametric Wilcoxon test to check this conclusion. The p-value (0.029) also assumed a low value ($p < 0.05$). Therefore, as the p-value is less than 0.05, and the mean difference is negative, we can conclude that: there is evidence that the MisR in AO models is significantly lower than in OO models. Therefore, we reject the null hypothesis $H_{3\text{-}0}$.

4.4 Discussion

We have identified five outstanding findings from the answer sheets, interviews, and observational study.

1) Higher Aspect Quantification and Inconsistency Detection. First, aspects with higher quantification [10] harmed inconsistency detection (RQ1) and the model interpretation (RQ3) by developers. We observed that when an aspect had six crosscutting relationships and, therefore, affected multiple join points (11, in this case), the subjects spend more time on performing global reasoning. The analysis of several aspect effects in the structural diagrams made developers often to neglect the analysis of behavioral interactions at each specific join point in the behavioral diagrams. According to the interviewees, this effect distracts away developers from observing possible inconsistencies between the structural and behavioral views. We observed, for example, that the inconsistency detection rate in OO models was 71 percent higher than in AO models when the latter were composed of aspects with high quantification; in these circumstances, the mean in OO models was 0.65 compared to 0.38 in AO models. We noticed that 20 subjects explicitly reported that they felt distracted by the presence of high density of crosscutting relationships among the model elements.

2) Overlapping Information about Crosscutting Relationships. Conversely, we observed that the subjects tended to detect more quickly inconsistencies in AO models when the scope of aspect pointcuts was narrow. In these cases, developers invested effort in only confronting structural and behavioral information about the crosscutting relations. According to the subjects, they could observe inconsistencies more quickly in AO models because structural diagrams often express the type of an advice (i.e. before, after or around), which is also a behavioral information that is present in the sequence diagram. Then, they could easily identify inconsistencies between: (i) the types of advices in the class diagram, and (ii) when a particular message was being advised by the aspect in the sequence diagram.

3) Crosscutting Relationships and Diverging Mental Models of the "Big Picture." Data analysis suggests that uniform interpretation of AO models by different developers is harder to achieve than in OO models. The subjects had difficulties to create a "big picture" view from the conflicting class and sequence diagrams. This view represents a "mental model" reflecting how software developers perceive the problem, think about it, and solve it by producing the expected code from the diagrams. This understanding shapes the actions of the developers and defines the approach to guide the design realization in the code. In particular, the developers apparently had

diverging mental models when the model inconsistencies were sourced in the crosscutting relationships. In these cases, developers came up with very different solutions for realizing crosscutting relationships in the code. They provided different answers on which and when the advice should affect the base model elements. Consequently, the communication from designers to programmers seems to be more sensitive to inconsistencies in aspect-oriented models.

4) The Level of Model Detail Matters. Developers usually consider the sequence diagrams as the basis for the design implementation. Note that in this case, the developers do not report the presence of inconsistency. This can be explained for some reasons. First, sequence diagrams are less abstract than class diagrams. This leads developers to rely on the behavioral diagrams than structural diagrams. Second, sequence diagrams are closer to the final implementation; hence, developers become confident that the information present on it is the correct one compared with the class diagram. As a result, it means that when models are used to guide the implementation of design decisions, inconsistencies in behavioral diagrams have a superior detrimental effect than those in class diagrams.

5) Identifying Fewer Inconsistencies in Less Time. Developers identify fewer inconsistencies in AO models than in OO models. However, they spend less effort to detect it in AO models. During the interviews, it was possible to observe that the main reason why developers stop in AOM and go ahead in OOM is that inconsistencies in AOM cause more severe doubts than in OOM. Hence, developers do not feel comfortable with using their experience to overcome the inconsistency problems given the problem at hand. Note that the subjects identify fewer inconsistencies in AOM not because they spent less time, but because it is seen as a "wicked problem." Thus, the developers may be more afraid of dealing with problems in AO models rather than OO models. Finally, the results suggest that developers might insert more defects into code by using AO models. This can be motivated for two reasons: (1) low inconsistency detection, and (2) high disagreement on design interpretations.

5 Threats to Validity

Internal Validity. Inferences between our independent variable and the dependent variables are internally valid if a causal relation involving these two variables is demonstrated [5]. Our study met the internal validity because: (1) the temporal precedence criterion was met; (2) the covariation was observed, i.e. the dependent variables varied accordingly when the independent changed; and (3) there is no clear extra cause for the detected covariation. Our study satisfied all these three requirements for internal validity.

External Validity. It refers to the validity of the obtained results in other broader contexts [5]. Thus, we analyzed whether the causal relationships investigated during this study could be held over variations in people, treatments, and other settings. Some characteristics that strongly contributed to this were identified. First, the subjects used: (1) a practical AOM technique to perform the tasks; and (2) the design models were fragments of real-world models. Second, the reported controlled experiment was rigorously performed, in particular, when compared with controlled experiments previously reported [3].

Construct Validity. It concerns the degree to which inferences are warranted from the observed cause and effect operations included in our study to the constructs that these instances might represent. All variables of this study were quantified using a suite of effort metrics or indicators that were previously defined and independently validated in experiments of inconsistency detection [2][3]. Moreover, the concept of effort used throughout our study is well known in the literature [8] and its quantification method was reused from previous work [2][3]. Therefore, we are confident that the quantification method used is correct, and the quantification was accurately performed.

Statistical Conclusion Validity. Experimental guidelines were followed to eliminate this threat [5]: (1) the assumptions of the statistical tests (paired t-test and Wilcoxon) were not violated; (2) collected datasets were normally distributed; (3) the homogeneity of the subjects' background was assured; (4) the quantification method was properly applied; and (5) statistic methods were used. The Kolmogorov-Smirnov and Shapiro-Wilk tests [1] were used to check how likely the collected sample was normally distributed.

6 Related Work

Aspect-oriented modeling is a very active research field [7][17]. However, there is little related work focusing on the quantitative and qualitative assessment of AOM. The current AOM literature does highlight the importance of performing empirical studies [8]. However, none of them empirically investigate the research topics addressed in our research questions. Research has been mainly carried out in two areas: (1) defining new AOM techniques [7][17], and (2) proposing new weaving mechanisms [13]. Several authors have proposed new modeling languages, focusing on the definition of constructs, such as <<aspect>> and <<crosscut>>. These constructs represent concepts of aspect-orientation as UML-based extensions [7][17][18][19][20]. For example, Clarke and Baniassad [7] make use of UML templates to specify aspect models. On the other hand, the chief motivation of some works is to provide a systematic method for weaving aspect and base models (e.g. [12][13]). For example, Klein and colleagues [13] present a semantic-based aspect weaving algorithm for hierarchical message sequence charts (HMSC). They use a set of transformations to weave an initial HMSC and a behavioral aspect expressed with scenarios. Moreover, the algorithm takes into account the compositional semantics of HMSCs.

Empirical studies of AOM (such as [6]) have not been conducted, in particular, in the context of modeling inconsistencies (or defects). Only the literature on OO modeling does highlight that empirical studies have been done on identifying defects in design models [2][3]. Lange and Chadron [3] investigate the effects of defects in UML models. The two central contributions were: (1) the description of the effects of undetected defects in the interpretation of UML models, and (2) the finding that developers usually detect more certain kinds of defects than others. In conclusion, there are two critical gaps in the current understanding about AOM: (1) the lack of practical knowledge about the developers' effort to localize inconsistencies, and (2) the lack of empirical evidence about the detection rate and misinterpretations when understanding AO models.

7 Concluding Remarks

This paper reports an empirical investigation about the impact of AOM on the inconsistency detection rate, the effort to detect inconsistencies, and the misinterpretation rate. We observed that developers detected fewer inconsistencies in AO models than OO models. The reason is that they got more distracted by the global reasoning motivated by the presence of crosscut relations and overlooked the negative effects of existing model inconsistencies. According to the subjects, a complex crosscutting collaboration between modules led developers to unconsciously make more *implicit assumptions* about the correct design decisions. As a consequence, aspects with higher quantification were the cause of a lower detection rate of inconsistencies.

Second, developers spent less effort using AO models to detect each inconsistency than in OO models. This was mainly due to a higher degree of overlapping information in structural and behavioral views of AOM. Third, the software developers presented a superior rate of misinterpretation in AO models, mostly thanks to the additional number of modeling concepts (e.g., crosscut relationships and aspects). They also had to examine all join points affected by the aspects. This extra analysis increased the degree of disagreement by developers while interpreting AO models and producing the code. It is important to highlight that all the aforementioned findings were independent of inconsistencies being assessed.

References

1. Levine, D., Ramsey, P., Smidt, R.: Applied Statistics for Engineers and Scientists. Duxbury (1999)
2. Lange, C., Chaudron, M.: An Empirical Assessment of Completeness in UML Designs. In: 8th Empirical Assessment in Software Engineering 2004, pp. 111–121 (2004)
3. Lange, C., Chaudron, M.: Effects of Defects in UML Models – An Experimental Investigation. In: International Conference on Software Engineering 2006, Shangai, China, pp. 401–410 (May 2006)
4. Kitchenham, B., et al.: Evaluating Guidelines for Reporting Empirical Software Engineering Studies. Empirical Software Engineering 13(1), 97–112 (2008)
5. Wohlin, et al.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Norwell (2000)
6. Farias, K., Garcia, A., Whittle, J.: Assessing the Impact of Aspects on Model Composition Effort. In: Aspect-Oriented Software Development 2010, Saint Malo, France, pp. 73–84 (2010)
7. Clarke, S., Banaissad, E.: Aspect-Oriented Analysis and Design the Theme Approach. Addison-Wesley, Upper Saddle River (2005)
8. France, R., Rumpe, B.: Model-Driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering at ICSE 2007, pp. 37–54 (2007)
9. Evaluating the Impact of Aspects on Inconsistency Detection Effort: a Controlled Experiment (2012), `http://www.les.inf.puc-rio.br/opus/models2012-aom`
10. Filman, R., Friedman, D.: Aspect-Oriented Programming is Quantification and Obliviousness. In: RIACS (2000)

11. OMG, Unified Modeling Language: Infrastructure, v2.2, Object Management Group (February 2010)
12. Whittle, J., Jayaraman, P.: Synthesizing Hierarchical State Machines from Expressive Scenario Descriptions. ACM TOSEM 19(3) (January 2010)
13. Klein, J., Hélouët, L., Jézéquel, J.: Semantic-based Weaving of Scenarios. In: 5th Aspect-Oriented Software Development, Bonn, Germany (March 2006)
14. AspectJ (2011), http://www.eclipse.org/aspectj
15. Dobing, B., Parsons, J.: How UML is Used. Communications of the ACM 49(5), 109–113 (2006)
16. Brun, Y., Holmes, R., Ernst, M., Notkin, D.: Proactive Detection of Collaboration Conflicts. In: 8th SIGSOFT ESEC/FSE, Szeged, Hungary, pp. 168–178 (2011)
17. Losavio, F., Matteo, A., Morantes, P.: UML Extensions for Aspect Oriented Software Development. Journal of Object Technology 8(5), 85–104 (2009)
18. Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., Kapsammer, E.: A survey on UML-based aspect-oriented design modeling. ACM Computing Survey 43(4), 1–33 (2012)
19. Aldawud, O., Elrad, T., Bader, A.: A UML Profile for Aspect- Oriented Software Development. In: Workshop on Aspect-Oriented Modeling at AOSD (2003)
20. Chavez, C., Lucena, C.: A Metamodel for Aspect-Oriented Modeling. In: Workshop on Aspect-Oriented Modeling with the UML, at AOSD 2002, Netherlands (April 2002)