5
Effort on the Detection of Inconsistency

Modeling languages, such as UML (OMG, 2011) and its extensions, provide different types of models (e.g., class and sequence diagrams) to represent complementary views of a software system. These models define the system structure and behavior so that design decisions can be properly understood. Developers will implement these complementary models later. Examples of these complementary models would be UML sequence diagrams and UML class diagrams. It is well known that, in practice, these models are created and used by different developers in parallel and often suffer from the inconsistency problems (Lange, 2007a; Apel et al., 2011; Mens, 2002;). These inconsistencies are mainly caused by the mismatch between the overlapping parts of complementary models and by the lack of formal semantics to prevent these contradictions (Lange et al., 2006a; Lange et al., 2004). Consequently, developers must invest some effort to detect and properly deal with these inconsistencies (Farias et al., 2011); otherwise, misinterpretation caused by inconsistencies could be transformed into defects in code.

Different modeling languages support different forms of modular decomposition and may influence how developers detect or neglect inconsistencies (Farias et al., 2010a). This might be particularly the case with aspect-oriented modeling (AOM) (Clarke & Banaissad, 2005; Clarke, 2001) as it intends to improve design modularity of otherwise crosscutting concerns. Current research in AOM varies from UML extensions (Losavio et al., 2009; Chavez et al., 2002; Clarke & Banaissad, 2005) 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 and minimize misinterpretations arising between multiple design models.

PUC-Rio - Certificação Digital Nº 0821407/CA

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 to support a superior modularization lead to detrimental effects on design understanding. Second, it is still not clear if an aspect affecting multiple join points can 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 (Filman & Friedman, 2000; Clarke & Walker, 2001) 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 (Farias et al., 2010a).

In this context, the goal of this chapter is to investigate the effects of the design modeling languages on the following quality notions: detection, social, syntactic, and semantic ones. This Chapter, therefore, reports a controlled experiment aimed at investigating the impact of aspect-oriented (AO) modeling on: (1) the rate of inconsistency detection; (2) the developers' effort to detect these inconsistencies; and (3) developers' misinterpretation rate. The use of AO models was contrasted with the use of OO models in a particular context: the use and understanding of design models by developers needed to produce the corresponding implementation. The results supported by statistical tests and qualitative analysis, show that AO models alleviated the effort to detect inconsistencies. Nevertheless, it reduced neither inconsistency detection rate nor misinterpretation rate.

Other findings were also reported. For instance, we observed that the downsides of AO modeling were largely caused by the degree of aspect quantification (Filman & Friedman, 2000). 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 number of crosscut relationships influences the creation of the "intended model." To the best of our knowledge, our results are the first to pinpoint the potential (dis)advantages of AO modeling in imprecise multi-view modeling.

The remainder of this chapter is organized as follows. Section 5.1 presents background. Section 5.2 describes the study methodology. Section 5.3 and Section

PUC-Rio - Certificação Digital Nº 0821407/CA

5.4 are the main contributions — the experimental results and their discussion itself. Section 5.5 compares the study with the related work and, Section 5.6 discusses the threats to validity. Finally, Section 5.7 gives some conclusions.

5.1. Background

This background is complementary to the explanations described in Chapter 2. Inconsistency detection has been studied for many years in software engineering (Lange et al., 2006a; Lange et al., 2004) and in other related disciplines. In fact, developers often need to detect conflicting information between artifacts during the software development process. In the context of our study, we investigate if developers are more able to detect inconsistencies in AO models rather than OO models used to communicate design decisions.

5.1.1. Aspect-Oriented Modeling

As previously mentioned in Chapter 2, 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.

The modularization of such crosscutting concerns is achieved by the definition of a new model element, called aspect. In general, the notation enables to explicitly distinguish between aspects and classes. An aspect can crosscut several classes in a system. These relations between aspects and other modules are called crosscutting relationships.

This aim is achieved in different ways in the AOM techniques. The current proposed approaches e.g., (Klein et al., 2006) are mainly aimed at supporting innovative weaving process for base and aspect models. That is, they aim at expressing and simulating the weaving relations between the base model and aspectual model elements. Approaches that are more conservative propose UML profiles (Losavio et al., 2009; Chavez & Lucena, 2002; Stein et al., 2002) for supporting the modeling aspect-oriented design. These techniques are more aligned to AOP models, such as those realized by AspectJ (AspectJ, 2011) and dialects.

PUC-Rio - Certificação Digital Nº 0821407/CA

Given the goal of our work (Section 5.2.1), we opt for evaluating the impact of aspect-oriented UML profiles on inconsistency detection processes. This choice can be explained by some reasons. First, real developers use UML profiles for AO modeling instead of any other AO modeling technique. Second, these profiles have the advantage of supporting classical AOP concepts at a more abstract level (Losavio et al., 2009; Aldawud et al., 2003; Chavez & Lucena, 2002). This means that AO key concepts are usually represented via conventional extension mechanisms of the Unified Modeling Language (UML), such as stereotypes. This alternative avoids classical side effects related to the learning curve in a controlled experiment like ours. Otherwise, it would not be possible to investigate the causal relationships between the dependent and independent variables (Section 5.2.6) without any high overhead to the subjects involved.

Another reason is that UML is the standard for designing software systems. The use of stereotypes reduces the gap between subjects with low experience and ones with more experience (Ricca et al., 2010). The other consequence of using UML profiles for AOM is that the model reading technique used by the subjects would not be more influenced by new notation issues. As UML profiles are supported by academic and commercial modeling tools, such as IBM Rational Software Architect and Borland Together, developers are familiar with stereotype notations. Moreover, the learning curve of the current state-of-the-art of AOM is not a trivial task for developers in early adoption of aspect-oriented programming.

Finally, UML profiles for aspect-oriented design is the approach more common for structural and behavioral diagrams. Therefore, the interpretation of the models is exclusively influenced by the use of the concepts in object-oriented and aspect-oriented modeling. Based on these reasons, the AOM language used in our study is a UML profile (Losavio et al., 2009; Aldawud et al., 2003; Chavez & Lucena, 2002). **Erro! Fonte de referência não encontrada.**Figure 16 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: (A) and (B) represent the conflicting structural diagrams, while (C) and (D) represent the structural and sequence diagrams without inconsistencies. 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

PUC-Rio - Certificação Digital Nº 0821407/CA

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 (Clarke & Walker, 2005; Clarke & Walker, 2001). 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 particular pointcut is applied.

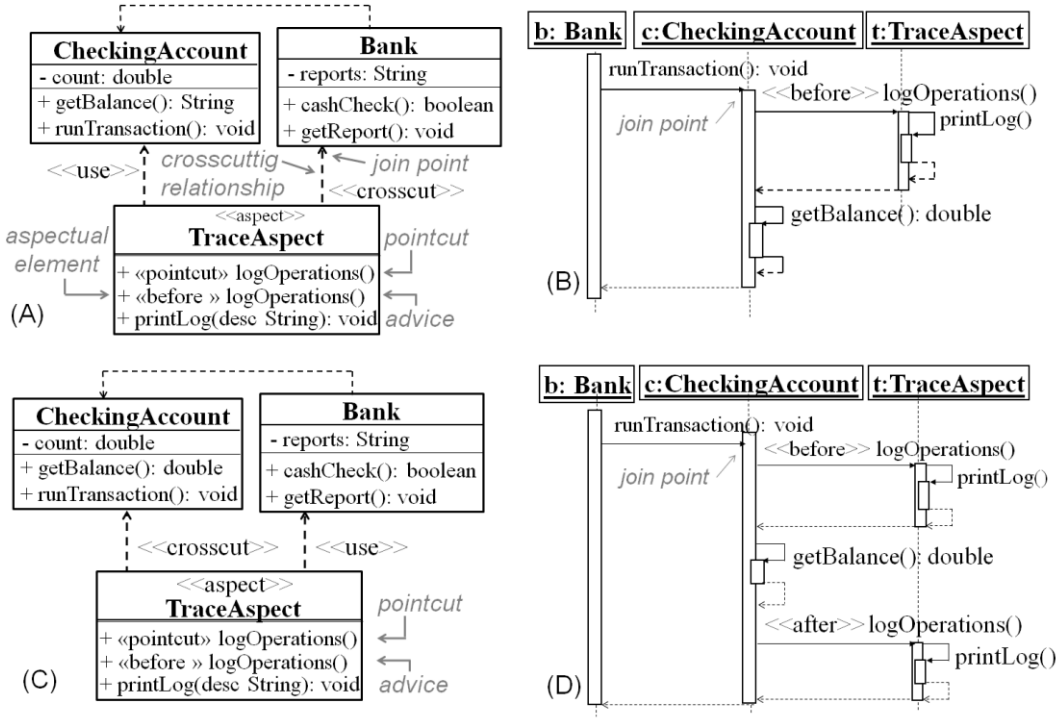

Figure 16: An illustrative example of aspect-oriented models used

5.1.2. Model Inconsistency

Model inconsistency was previously discussed in Chapter 2. However, it is discussed again due to the need for further details to investigate the research questions addressed in this Chapter. Additionally, it is only discussed here due to readability issues.

Model inconsistency is often the case that complementary diagrams of a software system, such as class and sequence diagrams, inevitably have conflicting information (Langes & Chaudron, 2004). If software developers do not detect and properly deal with these inconsistencies the potential benefits of using design

PUC-Rio - Certificação Digital Nº 0821407/CA

models can be compromised. This means that, for instance, gains in productivity and design understandability will be hindered. Consequently, developers must invest some considerable effort to detect these inconsistencies. Two broad categories of the most common inconsistencies are: (1) syntactic inconsistencies, which arise when the models do not conform to the modeling language's metamodel; and (2) semantic inconsistencies, in which the meaning of one or more model elements does not match with that of the actual design model. Our study focused on semantic inconsistencies because they cannot be automatically identified with tool support (Lange & Chaudron, 2006a). Moreover, they are usually the main cause of design misinterpretation (Wohlin et al., 2000).

Occurrences of semantic inconsistencies are particularly very common when class and sequence diagrams are used in conjunction with a system (Lange & Chaudron, 2006a; Lange & Chaudron, 2004). This is probably due to the fact they are the most used UML models in practice (Doring & Parsons, 2006) and represent the same concepts under different perspectives. These are the key reasons governing the selection of these diagrams in our experimental study. Moreover, we have particularly selected semantic inconsistencies that are: (i) detectable by developers (Lange & Chaudron, 2004), and (ii) difficult or impossible to detect automatically. The reason for the latter is that the semantics of model elements are rarely expressed in a formal manner. Semantic inconsistencies are even more difficult to detect in multi-view modeling (Kitchenham et al., 2008). Semantic inconsistencies arise in multi-view models when they have overlapping parts. For instance, objects exchange messages in sequence diagrams, while these messages represent methods in the class diagram. In addition, a message from one object to another means that the first object calls a method that is provided by the second object. Other forms of overlapping elements occur in aspect-oriented models. There are several forms of multi-view inconsistencies and we discuss below how they can manifest in both OO and AO models. This thesis aims at inconsistencies that have been documented elsewhere (Lange et al. 2004) and used in a previous empirical study (Lange et al. 2006). The inconsistencies used in this study are described as follows:

1) *Conflicting relationships*: this inconsistency occurs when the presence or the nature of a relationship diverge in structural and behavioral models. For instance, according to the sequence diagram, the advice of an *aspect A*

PUC-Rio - Certificação Digital Nº 0821407/CA

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, Figure 16 presents this kind of inconsistency. The aspect *t:TraceAspect* crosscuts the *c:CheckingAccount* objects (Figure 16.B). In this case, the relationship between *TraceAspect* and *CheckingAccount* should be <<*crosscut*>> instead of <<*use*>> (Figure 16.C) given the logging semantics of the advice *logOperations().* In the structural diagram (Figure 16.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 an around advice (Losavio et al., 2009; Aldawud et al., 2003; Chavez & Lucena, 2002) and the return type from a method execution being advised by the latter.
3) *Object without class/aspect*: an object in a sequence diagram does not have a corresponding class or aspect in the class diagram.
4) *Weaving in a wrong element*: an aspect *A* weaves advice into model element *B* in the sequence diagram, but in the class diagram does not exist any crosscutting relationship from *A* to *B*.
5) *Message without name*: a message between objects in the sequence diagram does not have a name.
6) *Message without method*: a message from an object of class *A* to an object of class *B* does not correspond to any method of the class *B* in the class diagram.
7) *Message with wrong return type*: the return type of a message *X* from an object of class *A* to an object of class *B* does not match with the return type of the method *X* of the class *B* in the class diagram.
8) *Message in the wrong direction*: there is a message from an object of class *A* to an object of class *B*, but the method corresponding to the message is a member of class *A* instead of class *B*.

PUC-Rio - Certificação Digital Nº 0821407/CA

9) *Class without meaning*: a class does not have any semantic value in the class diagram.
10) *Instance of abstract class*: an abstract class is used in the sequence diagram as object.

Although the behavioral and structural diagrams are syntactically correct, the contradicting information makes the models semantically incorrect. Note that if developers do not detect these inconsistencies, they will likely transform them into defects in code due to the misinterpretation. For example, a developer might take in consideration the specification of the method *CheckingAccount.getBalance* in the structural diagram (i.e., string as return type), whereas other developer might consider the specification in the sequence diagram (double as return type). Consequently, this can give rise to unexpected behavior in the code as a method can expect a string as return type instead of double (Mens, 2002). This contradicting information between the models may lead to static and behavioral inconsistencies in code.

PUC-Rio - Certificação Digital Nº 0821407/CA

5.1.3. Inconsistency Detection Effort

Developers detect inconsistencies when they identify conflicting information in the models and, then, possibly report that the models cannot be implemented. This decision often relies on "guessing" the semantics of model elements. To reach this conclusion, developers need to invest some effort: the time to go through the model and infer that the models suffer from inconsistencies. There is currently very limited knowledge regarding the amount of effort required to detect inconsistencies. Anecdotal evidence from companies suggests that the effort is significant (Farias et al., 2011), but nothing can be conjectured considering AO models in comparison to OO models.

There are some tools to support the visualization of crosscutting relation effects in class diagrams (Clarke & Walker, 2005). There are also tools to generate a woven sequence diagram (Klein et al., 2006) or even integrating or simulating the effects of composing state machines. The use of these tools was not included in our study for several reasons. First, the nature of the investigated conflicts would require that developers undergo model inspection anyway. In fact,

the focus of our study is to investigate whether developers can pinpoint inconsistencies and understand the design decisions when producing the corresponding implementation. Second, even though the use of these tools might reduce or exacerbate the generation of specific categories of inconsistencies in AO models, it was not our goal to evaluate particular tools. More importantly, these tools are not used in practice yet; either because they are not robust enough to be applied in real-world settings, or because they are not intuitive to be used in practice. Hence, their use would impose severe threats the validity of our experimental results.

5.2. Study Methodology

This section presents the main decisions underlying the experimental design of the controlled experiment, which adheres to guidelines of empirical studies (Kitchenham et al. 2008; Wohlin et al. 2000). We chose controlled experiment due to the same reasons discussed in Section 4.1.1.

5.2.1. Experiment Definition

We formulate the goal of this study using the GQM template (Wohlin et al. 2000) 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.

Therefore, this is related to research question RQ3, as stated in Chapter 1:

- **RQ3:** What is the effect of design decomposition techniques in particular with respect to misinterpretation, inconsistency rate, inconsistency detection effort, and inconsistency resolution effort?

Regarding the quality notions defined in Chapter 3, we study how design modeling languages affect six quality notions, namely: syntactic, semantic, pragmatic, social, effort, and detection ones. Based on this, we refine the research

PUC-Rio - Certificação Digital Nº 0821407/CA

question into three more specific research questions. Thus, we focus on the following research questions:

RQ3.1: Does AO model affect the efficiency of developers to detect multi-view model inconsistencies?

RQ3.2: Does AO model influence the effort invested by developers to detect model inconsistencies?

RQ3.3: 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 Federal University of Bahia (UFBA). In both courses, AO modeling and OO modeling were 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.

PUC-Rio - Certificação Digital Nº 0821407/CA

5.2.2. Hypothesis Formulation

First Hypothesis. 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 (Lange et. al., 2006). 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, developer may localize more inconsistencies than in OO models. Theses 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) ≥ 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. The second research question investigates whether developers invest less (or more) effort to detect inconsistencies in AO models than in 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 (Farias et al., 2010a). 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 and 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)

PUC-Rio - Certificação Digital Nº 0821407/CA

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. The third research question investigates whether developers' misinterpretation rate (MisR) is higher (or lower) in AO models than in OO models. The chief reason of the disagreement between developers' interpretations 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 (Lange & Chaudron, 2006a; Farias et al., 2010a). 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 can 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 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)

PUC-Rio - Certificação Digital Nº 0821407/CA

5.2.3.
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.

PUC-Rio - Certificação Digital Nº 0821407/CA

5.2.4.
Experiment Design

The design of this study was a *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. The subjects were randomly assigned and equally distributed to the treatments. The distribution followed a within-subjects design in which all subjects served in the two treatments. Each treatment had a printed questionnaire with five multiple-choice questions. That is, the subjects did not make use of modeling tools to understand and answer the questions. Although it was generally accepted nowadays that the current state-of-the-art of AOM (such as (Klein et al., 2006)) should be always used with a tool, the use of any kind of tool would certainly add some bias to the collected data: the subjects would be influenced by the different maturity and usability degrees of AO and OO modeling tools. Hence, we would end up comparing the tools instead of modeling languages. Moreover, we emphasize that the focus of this work is on the current state-of-the-practice of AOM instead of the state of the art of AOM, as briefly justified in Section 5.1.1. By doing so, 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 was by definition a balanced design.

To minimize the "gain in information" from one treatment to another one, the models used in the study were fragments of real class and sequence diagrams. Hence, the subjects had no prior information and no accumulated knowledge about the semantics of the model elements. In addition, each pair of structural and behavioral models had different kinds of inconsistencies, 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.

5.2.5. Operation and Material

Operation. In both treatments, the subjects received a pair of corresponding class (structural) and sequence (behavioral) diagrams similar to the models presented in Figure 16. 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). Our goal is to identify how developers deal with contradicting information between complementary models 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, although the subjects were responsible for registering the time invested in each question ("start time" and "end time"), they were properly managed to avoid bias in the collected data. They were also stimulated to justify their answers on the answer sheet, but this part of the time was not counted. In total, ten questions were answered. After the experiment, the subjects were also interviewed to clarify the answers and results.

Material. Table 24 describes some design characteristics for the OO and AO models used in the study. For example, in the first task, the AO model had seven classes and one aspect, seven relationships between the classes and aspect, and six crosscutting relationships. Additionally, it is important to highlight three points: (1) every pair of OO or AO class and sequence diagrams had two kinds of

PUC-Rio - Certificação Digital Nº 0821407/CA

inconsistencies, (2) research questions were investigated in all tasks of the experiment, and (3) the AO models vary with respect to the number of crosscutting relationships. The reason for the latter decision is that we suspect that these relationships might affect the variables (i.e., inconsistency detection rate) and detection effort) of this study (Section 5.2.6). 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 5.1.1.

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 in concordance with 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, but those careful explanations are not part of the time required to solve the task.

PUC-Rio - Certificação Digital Nº 0821407/CA

Task	Treatment	Class Diagram				Sequence Diagram	
		#CA	#RC	#AT	#OP	#O	#M
1	OO	7	6	18	27	6	7
	AO	8	11(6)	5	16	7	13
2	OO	8	6	16	23	6	6
	AO	6	5(1)	9	19	5	10
3	OO	4	4	4	16	4	7
	AO	5	4(1)	6	14	5	10
4	OO	4	4	6	12	5	10
	AO	6	7(2)	7	20	6	11
5	OO	4	4	11	13	5	7
	AO	5	5(2)	7	14	5	8

#CA: the number of classes or /and aspects;
#RC: the number of UML relationships or crosscutting relationships
#AT: the number of attributes. #OP: number of operations.
#O: the number of objects or instance of aspects. (n): number of aspects.
#M: the number of messages between the classes and aspects.

Table 24: Measures of the diagram used in the study

5.2.6. Variables and Quantification Method

The independent variable of this study is the choice of the modeling language. It is nominal and can assume two values: AO modeling and OO modeling. We investigate the effects of this independent variable on following dependent variables.

Inconsistency detection rate (Rate). This variable is intended to measure the overall rate of inconsistencies detected by all subjects (RQ4.1). 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. Note that subjects detect inconsistencies when they explicitly indicate that they are unable to achieve a suitable implementation from the conflicting diagrams.

Inconsistency detection effort (Effort). It represents the mean of time (minutes) spent by the subjects to detect inconsistencies in a question (RQ4.2).

Misinterpretation rate (MisR). This variable represents the degree of variation of the answers (RQ4.3). 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 (Lange & Chaudron, 2006a) 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

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

Where:
K: the number of alternatives for a question
k_i: the number of times alternative i was selected,
where $0 \le i < K$ and (for all i : $0 \le i < K - 1 : k_i \ge k_{i+1}$)
N: the sum of answers over all alternatives: $N = \sum_{0 \le i < K} k_i$

PUC-Rio - Certificação Digital Nº 0821407/CA

inconsistencies cause serious misinterpretations (MisR = 0). That is, the misinterpretation rate is 0 (zero) if the answers are distributed equally over all options, and 1 (one) if the answers are concentrated only one answer option. This variable can be measured as follows (Lange et al., 2004).

5.2.7. Operation

Preparation phase. The subjects (students and professionals) were not aware about 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 grades. The results obtained in the questionnaire had no effect on their grades. 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.

5.2.8. Analysis Procedures

Quantitative Analysis. The normal distribution of the collected data was checked using the Shapiro-Wilk and Kolmogorov-Smirnov test (Devore et al., 1999; Wohlin et al., 2000). The three hypotheses were tested using the parametric paired t-test and the non-parametric Wilcoxon test. Both methods compare two related samples or repeated measurements on a single sample to assess whether

PUC-Rio - Certificação Digital Nº 0821407/CA

their population means differ (Devore et al., 1999). All hypotheses were tested considering a significance level of 0.05 (p-value < 0.05). The null hypotheses were rejected when the p-value was lower than 0.05.

Qualitative Analysis. Qualitative data were collected from two sources: think aloud answer sheet comments and interviews. The comments were expressed in a free-text field in which the subjects could report anything to explain their answer. In addition, some questions were prepared and asked to developers in interview sessions. Interview guidance with relatively open questions was prepared and all sessions were audio recorded with the permission of the subjects

5.2.9. Qualitative Data

Interviews. A semi-structured interview approach (Wohlin et al., 2000) was chosen following a funnel model, in which one initial open question is told and then directed towards to more specific one. It was organized in topics with open and closed questions. They were organized in such a way that research questions could be exploited. 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 the interviews. It was assured that only anonymous data would be presented externally. Each interview lasted from 30 to 55 minutes, depending on how talkative the subjects were.

Observational Study. In order to investigate how the tasks in the experiment were performed, extensive observations were conducted through two different approaches. First, the authors run the experiment. This allowed a more effective observation and monitoring of the tasks of the subjects. Second, to obtain an additional feedback from the subjects, they were encouraged to write down the rationale used to answer the questions.

5.3. Experimental Results

This section discusses the experimental results related to the research questions RQ4.1, RQ4.2, and RQ4.3 (Section 5.2.1). All hypotheses were tested at

PUC-Rio - Certificação Digital Nº 0821407/CA

the significance level of α = 0.05 and the findings were derived from both descriptive statistics and statistical inference.

5.3.1. RQ4.1: 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. Contradicting the expected AOM superiority, the collected data indicate that developers tend to detect more inconsistencies in OO models than in their AO counterparts. Table 25 provides evidence for this observation through descriptive statistics of the collected data. The superior detection rate in OO models manifests in terms of both means and medians. As far as the latter in concerned, the median of the detection rate is 0.35 in AO models and 0.5 in OO models. This difference represents a superiority of 42.85 percent in favor of OO models. This observation is reinforced by analyzing the means of the detection rate. Developers detected, on average, 43.24 percent more inconsistencies in OO models (0.53) than AO models (0.37). These results suggest that OO models enable developers to identify more inconsistencies than AO models. As a consequence, classical UML-based modeling for crosscutting modularity (Section 5.1.1) do not necessarily imply on more effective inconsistency detection according our observations. This contradicts somehow the intuition that the improved modularity of AOM helps developers to localize inconsistencies (Section 5.1.2).

Variable	Treatment	Mean	St Dev	Min.	25th	Med.	75th	Max	%diff
Detection	AO	0.37	0.09	0.23	0.29	0.35	0.46	0.54	43.24
	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
	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
	OO	0.7	0.07	0.62	0.64	0.69	0.77	0.81	

St Dev: standard deviation, diff: difference

Table 25: Descriptive statistics

Hypothesis Testing. We check whether this result is statistically significant by trying to reject the first null hypothesis $H_{1\text{-}0}$ in the five experimental tasks (Table 26). Since the Shapiro-Wilk and Kolmogorov-Smirnov normality tests

PUC-Rio - Certificação Digital Nº 0821407/CA

(Devore et al., 1999) suggest that the data are normally distributed, the paired t-test was applied to test H_1. This strategy allowed us to realize a pairwise comparison of the distributions and check if there exists a significant difference between AO and OO models with respect to detection rate. Pairwise p-values and mean differences across pairs for each measure are reported in (Table 26). The mean differences between 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); in addition, the p-value (0.015) is less than 0.05, our selected level of significance. This implies that the detection rate in AO models was statistically lower than in OO models. Given this unexpected result, we were encouraged to apply the non-parametric Wilcoxon test to eliminate any threats to statistical conclusion validity. The low value of the p-value collected (0.031) also confirmed the aforementioned conclusion. Hence, there is sufficient evidence to reject the null hypothesis, and conclude that there is a difference between the detection rates in AO and OO models at the 0.05 level of significance.

PUC-Rio - Certificação Digital Nº 0821407/CA

Variables	**Treatment**	**Paired t-test**			**Wilcoxon**
		t	***p-value***	***Mean Difference***	***p-value***
Detection	AO	4.03	0.015	- 0.16	0.031
	OO				
Effort	AO	3.1	0.036	- 1.48	0.033
	OO				
MisR	AO	2.94	0.042	- 0.192	0.029
	OO				

*with 4 degree of freedom, a significance level of $\alpha = 0.05$

Table 26: Hypotheses testing

5.3.2. RQ4.2: Detection Effort in AO and OO models

Descriptive Statistics. The second research question investigates the effort that developers must invest to detect inconsistencies in AO and OO models. The gathered data in Table 25 indicate that 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 on 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 difference suggests that users of AOM tend to realize faster that: (i) a particular multi-view conflict exists, and (ii) such a conflict will compromise the implementation of the intended design. This phenomenon would confirm our initial intuition that the superior modularity of AO models accelerates inconsistency detection. In fact, during the interviews, the subjects (18) reported that the manifestation of inconsistencies in crosscutting relations is an influential factor on the conflict detection. According to them, such inconsistencies are perceived more quickly than other non-AOM inconsistencies. They 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. We also check if the finding above is statistically significant as follows. The Shapiro-Wilk and Kolmogorov-Smirnov certified the normal distribution of the measure (Devore et al., 1999). Therefore, the paired t-test was also applied to test H2 and evaluate RQ4.2. Table 26 shows the pairwise p-values and mean differences across pairs for each measure. Recall that the mean differences between pairs of AO and OO models indicate the direction in which the result is significant. The detection effort in AO and OO groups presented a negative value for the mean difference (-1.48), while p-value (0.036) is less than 0.05. The non-parametric Wilcoxon was also applied, which confirmed the above results given the p-value equal to 0.033. This enables us to infer that the average difference for detection effort between AO and OO models is not zero and that there is significant evidence that AO models required lower detection effort than in the OO counterparts.

PUC-Rio - Certificação Digital Nº 0821407/CA

5.3.3.
RQ4.3: 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 25 shows the descriptive statistics to the misinterpretation measures of AO and OO models. Recall that MisR varies between zero and one and that MisR = 1 indicates that developers did 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. The data revealed that the use of in OO models led to less misinterpretation (higher MisR value) than AO models. The misinterpretation rate was 37.25 percent lower in OO models; the mean was 0.51 in AO groups against 0.7 in OO groups. This upward trend was 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 lead the number of diverging interpretations of the participants 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 by developers.

Hypothesis Testing. We analyze the strength of the result testing H3 as follows. As in the previous analysis, the paired t-test was applied to test H3 as the measures assumed a normal distribution. Table 26 shows the pairwise p-values and mean differences across pairs for each measure. As the mean difference is negative (-0.192) and p-value (0.042) 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 (Devore et al., 1999) 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 $H_{3\text{-}0}$.

PUC-Rio - Certificação Digital Nº 0821407/CA

5.4. Discussion

This section highlights particular characteristics of the design modeling languages that more influenced the dependent variables. The answer sheets, interviews, and observational study were instrumental in this investigation. We have identified four main outstanding findings, which are described as follows.

Higher Aspect Quantification and Lower Inconsistency Detection. First, aspects with higher quantification (Filman & Friedman, 2000) harmed inconsistency detection (RQ4.1) and the model interpretation (RQ4.3) by developers. We observed that when an aspect had six crosscutting relationships (see Table 24) and, therefore, affected multiple join points (11, in this case), the subjects spend more time 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. This finding is also confirmed by complementary data analyses. 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. An explanation for this phenomenon can be derived from the interviews and the observational study. We noticed that 20 subjects explicitly reported that they felt distracted by the presence of high density of crosscutting relationships in the models.

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

PUC-Rio - Certificação Digital Nº 0821407/CA

diagram, and (ii) when a particular message was being advised by the aspect in the sequence diagram.

Crosscutting Relationships and Diverging Mental Models of the "Big Picture." Data analysis seems to suggest that uniform interpretation of AO models by different developers is harder to achieve than in OO models. According to the comments from the subjects, they often faced 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.

The Level of Model Detail Matters. Given the presence of inconsistencies in the diagrams, 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 phenomenon can be explained based on some reasons observed during the interviews and the observational study. First, sequence diagrams often present a higher number of details than the class diagrams. Thus, the lower level of abstraction leads the software developers to be more confident to the behavioral diagrams than structural diagrams. Next, sequence diagrams are closer to the final implementation; hence, developers become confident that the information present in the sequence diagram is the correct one compared to 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.

This finding is useful for improving quality assurance procedures in some activities in model-driven software development as, for example, model review.

PUC-Rio - Certificação Digital Nº 0821407/CA

Model review is a well-known, effective way to minimize defects in code. Nevertheless, it is not clear for developers what diagram should be reviewed at first. By using this finding, developers can put the focus on the behavioral diagrams rather than the structural diagrams. In practice, this information is important because the preference of the behavioral diagrams can result in action that is more effective. Since model review requires some considerable effort to examine and define the focus of the analysis, it usually receives some criticism. By using this finding, developers can also tame or improve this problem.

Identifying Fewer Inconsistencies in Less Time. Developers identify fewer inconsistencies in AOM than in OOM. However, they spend less effort to detect it in AOM. Note that when developers identify an inconsistency, they have two options: they report that they detected an inconsistency or try to overcome the problem based on their experience, but will give a wrong answer at the end. Based on this, we have observed that developers report more often the presence of inconsistency in AO models (compared to OO models) than try finding any other solution. On the other hand, by using OO models developers try answering the question even observing the presence of inconsistency.

During the interviews, it was possible to observe the main reason why developers stop in AOM and go ahead in OOM: inconsistencies in AOM cause more severe doubts to developers than in OOM. Hence, developers do not feel comfortable using their experience to overcome the inconsistency problems given the problem at hand. It is important to point out that the subjects identify fewer inconsistencies in AOM not because they spent less time but because it is seen as a "wicked problem." In doing so, we observed that the subjects are more afraid of dealing with problems in AO models rather than OO models. Finally, given that multi-view design models usually have inconsistencies (Lange et al., 2004), this can mean that classical UML extensions for AOM (Section 5.1.1) need to be carefully employed. The observed results of our study suggest that developers might insert more defects into code. This can be motivated for two reasons: (1) low inconsistency detection (Section 5.3.1), and (2) high disagreement on design interpretations (Section 5.3.3).

PUC-Rio - Certificação Digital Nº 0821407/CA

5.5. Limitations of Related Work

Aspect-oriented modeling supports early separation of otherwise crosscutting concerns in software design. Concerns are separated to improve, for example, the interpretation of design decisions governing crosscutting concerns by developers before the implementation is accomplished. In practice, AOM will be considered useful compared to traditional modeling techniques if the claimed improved modularity actually leads to practical benefits, such as reduction of inconsistency detection effort and misinterpretations. Unfortunately, it is well known, as previously mentioned, that empirical studies of AOM are rare in the current literature, which confirms that it is still in the craftsmanship era (France & Rumpe, 2007).

Research has been mainly carried out in two areas: (1) defining new AOM techniques, and (2) proposing new weaving mechanisms. First, 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 (Clarke & Walker, 2005; Chavez & Lucena, 2002; Aldawud et al., 2003; Stein & Hanenberg, 2002). In addition, (Clarke and Baniassad, 2005) make use of UML templates to specify aspect models. The chief motivation of some works is to provide a systematic method for weaving aspect and base models e.g., (Whittle et al., 2010; Klein et al., 2006; Jézéquel, 2008). Klein (Klein et, al, 2006) presents 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.

Most of empirical studies on aspect-orientation are performed at the code level. For example, Hanenberg (Hanenberg et al., 2009) compares the time invested by developers to implement crosscutting concerns using object-oriented and aspect-oriented programming techniques. Other studies focus on the assessment of aspect-oriented programming under different perspectives, such as stability (Ferrari et al., 2010; Greenwood et al., 2007) and fault-proneness (Lasavio et al., 2009; Burrows et al., 2010). However, empirical studies of AOM

PUC-Rio - Certificação Digital Nº 0821407/CA

(such as (Farias et al., 2010a)) 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 (Lange & Chaudron, 2004). Lange (Lange & Chaudron, 2006a) investigates 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 do.

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.

5.6. 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 (Wohlin et al., 2000). 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 (Wohlin et al., 2000). Thus, we analyzed whether the causal relationships investigated in this study could be held over variations in people, treatments, and other settings. Some characteristics were identified that strongly contributed for this purpose. 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 to previously reported controlled experiments (Lange et al., 2006; Ricca et al., 2010).

Construct Validity. It concerns the degree to which inferences are warranted from the observed cause and effect operations included in our study to the

PUC-Rio - Certificação Digital Nº 0821407/CA

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 (Lange, 2007). Moreover, the concept of effort used in our study is well known in the literature (Jorgensen, 2005; Menzies et al., 2006; Grimstad & Jorgensen, 2007; Jorgensen et al., 2008) and its quantification method was reused from previous work (Lange & Chaudron, 2006a). Therefore, we are confident that the quantification method used is correct, and the quantification was accurately performed.

Statistical Conclusion Validity. We evaluated the statistical conclusion validity checking if the independent and dependent variables were submitted to suitable statistical methods. Experimental guidelines were followed to eliminate this threat (Wohlin et al., 2000): (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) statistical methods were used. The Kolmogorov-Smirnov and Shapiro-Wilk tests (Devore et al., 1999) were used to check how likely the collected sample was normally distributed.

5.7. Concluding Remarks

This study reports an empirical investigation about the impact of alternative design decompositions on the inconsistency detection rate, the effort to detect inconsistencies, and the misinterpretation rate. We observed that developers detected fewer inconsistencies in AO decompositions than OO decompositions. The reason is that they got more distracted by the global reasoning motivated by the presence of crosscutting relations and overlooked the negative effects of existing model inconsistencies. According to the subjects, complex-crosscutting collaborations between modules led developers to unconsciously make assumptions that are more implicit about the correct design decisions. Consequently, aspects with higher quantification were the cause of the low 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

PUC-Rio - Certificação Digital Nº 0821407/CA

the higher degree of overlapping information in structural and behavioral views of AOM. Third, the 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.

We should point out that empirical studies in AOM are in its initial stage and there is very little practical knowledge that can be used to determine the effectiveness of the current AOM approaches on improving design understanding. This study represents the first controlled experiment that addresses this issue. Although we are confident that the collected results are very concrete, significant results, further empirical studies are still required to test the hypotheses in other contexts. This is essential to better understand whether the results of this study hold (or not) in a broader context. In further studies, some questions should be investigated: what will it be the impact of quantification on the misinterpretation rate? Which will inconsistencies cause a higher misinterpretation rate? What is the effort to repair AO models with elevated quantification rate? Will we collect the same results by using larger design models? Finally, we hope that the issues outlined throughout the Chapter encourage researchers to replicate our study in the future under different circumstances.

PUC-Rio - Certificação Digital Nº 0821407/CA