# 4
# Effort on the Application of Composition Techniques

The goal of this Chapter is to evaluate the effects of model composition techniques on the developers' effort. To this end, two studies are performed. The first study investigates the effort that developers invest to compose design models based on a controlled experiment. The second study evaluates the effort to compose design models from industrial case studies.

## 4.1.
## Effects of Composition Techniques on the Composition Effort

Model composition techniques can be classified in two categories (Chapter 2): (i) specification-based techniques, such as Epsilon (Epsilon, 2011) and MATA (Whittle & Jayaraman, 2010), and (ii) heuristic-based techniques, such as merge and override (Clarke, 2001; Clarke & Walker, 2001) and the three-way merge algorithm (Mens, 2002). The manual model composition is expected to be error-prone and time consuming. Then, developers apply model composition techniques with the aims of reducing the composition effort and improving the correctness of the composed model. The techniques in the first category primarily aim at producing correctly composed models, but it is questionable if they necessarily reduce composition effort. On the other hand, the second category aims at alleviating the developers' effort. However, its positive impact on the correctness of the composed models is expected to be worse than the first category.

By using the specification-based techniques, developers explicitly specify the correspondence and composition relations between the input model elements $M_A$ and $M_B$ to produce $M_{AB}$ (Section 2.4). On the other hand, by using the heuristic-based techniques developers apply a set of predefined heuristics, which "guess" the relations between model elements before composing $M_A$ and $M_B$. Specification-based techniques provide a systematic way to specify the relations between the input model elements, instead of trying to "guess" them. It is expected that these techniques not only alleviate the composition effort, but also

assure a higher rate of correctly composed models when compared to the heuristic techniques (Epsilon, 2011; Whittle & Jayaraman, 2010).

To date, however, there is little empirical evidence to confirm (or not) if these expectations hold; mainly, when developers try to: (1) select and apply the model composition techniques; (2) detect syntactic and semantic inconsistencies; and (3) resolve the identified inconsistencies in realistic settings. As described in Chapter 3, these three composition activities are required to obtain the intended model $M_{AB}$. Empirical studies in model composition are lacking, mainly ones considering the impact of the composition techniques on the following quality notions described in our quality model: effort, application, detection, resolution, syntactic, and semantic notions. In fact, the literature fails to provide such empirical evidence to software developers. As a result, developers are left without any practical knowledge to answer questions such as "*what are the effects of specification-based and heuristic-based techniques on the developers' effort and the correctness of the composed models*?" It is important to answer this question because, before adopting any composition technique in realistic settings, it is necessary to have practical knowledge about the effects of model composition techniques.

In fact, to date, both specification-based and heuristic-based techniques have been used without any empirical evidence. Currently developers rely on diverging feedbacks (Norris & Letkeman, 2011) from evangelists to evaluate how good techniques can be, rather than on practical, evidence-based knowledge derived from experimental studies. The practical knowledge about these effects (or even a trade-off analysis) can be viewed as the main impairment to the wide application of composition techniques in practice where resources and time are tight. Note that if a composition technique reduces effort but does not favor model correctness (or vice-versa), it is quite questionable whether it can be applied in industry. On the other hand, if the composition effort is high, the potential benefits of using composition techniques (e.g., gains in productivity) can be compromised.

The literature in model composition fails to provide assessments of model composition techniques (Apel et al., 2011; Sarma et al., 2011; Shao et al., 2011; Brun et al., 2011; Whittle et al., 2009; Klein et al., 2006). Apel (Apel et al., 2011). Mens (Mens, 2002) also reinforces the need for more empirical and experimental research. Burn and colleagues (Brun et al., 2011b) evaluate the composition of

code in the context of a retrospective, quantitative study of the evolution of nine open-source systems. They concluded that inconsistencies in code are the norm rather than the exception, and that 16% of all merges required human effort to resolve them. However, even this kind of primary empirical analysis is lacking in the context of model composition.

With this in mind, this Chapter reports a controlled experiment performed with 24 subjects, which used Epsilon, IBM RSA and traditional composition algorithms to evolve design models. The techniques are investigated in 144 evolution scenarios and by about 2304 compositions of model elements (such as classes and relationships). The main results, supported by a complete statistical and qualitative analysis, are: (1) the IBM RSA and traditional composition algorithms require less effort to produce the intended model than Epsilon, and (2) there is no significant difference in the correctness of the output composed models generated by these techniques. Additionally, in some cases, the number of inconsistencies produced by Epsilon was significantly higher than one generated by IBM RSA and traditional composition algorithms. The techniques investigated are robust and representative and there are reasons to believe the results will generalize to broader scenarios. However, we do not claim generalization beyond these techniques and their use on other types of design models, in particular class and sequence diagrams.

The remainder of the chapter is organized as follows. Section 4.1.1 presents the experiment planning. Section 4.1.2 analyzes the results. Section 4.1.3 contrasts our work with related work. Section 4.1.4 presents the threats to validity. Finally, Section 4.1.5 describes some concluding remarks.



### 4.1.1. Experiment Planning

This section presents the experiment planning followed to carry out a controlled experiment. This planning is based on practical and conventional guidelines of empirical studies such as (Wohlin et al., 2000; Kitchenham et al., 2008; Shadish et al., 2002; Sjober et al., 2002). We have opted to conduct a controlled experiment to investigate the hypotheses formulated in Section 4.1.1.2 due to a number of reasons (Basili et al., 2007). First, it allows us to conduct well-

defined, focused studies, with the potential for gathering statistically significant results, which is not possible with non-controlled case studies. Moreover, it helps to formulate hypotheses by forcing us to clearly state the question being studied and allow us to maximize the number of questions being asked.

Second, as controlled experiments require well-formulated dependent and independent variables as well as null and alternative hypotheses, it also allows us to understand the relations of specific variables and measures.

Third, by running a controlled experiment, we are forced to state clearly what questions the investigation is intended to address and how we will address them, even if the study is exploratory (Basili, 2007). Moreover, we can create a study design in such a way that maximizes the chance for replication of the study in order to test the hypotheses in different contexts and by independent researchers.

Fourth, controlled empirical studies can better investigate the cause-effect relationships between variables, allowing us to understand, for example, the effects of the independent variables on the dependent variables. Additionally, a controlled study provides insight into why relationships and results do and do not occur. It also forces us to analyze the threats to validity, leading to the identification of where replications or alternate studies are needed and where variations might show different effects. It also allows us to build a body of knowledge in model composition that helps researchers to build theories supported by clear empirical evidence.



#### 4.1.1.1. Experiment Definition

This study aims at evaluating the effects of model composition techniques on six quality notions, namely syntactic, semantic, effort, application, detection, and resolution ones. For this, we control two variables: the *effort* to compose design models and the *correctness* of the output models. Correctness is also controlled, as the evaluation of effort needs to be put in the perspective of the quality of the produced models. Otherwise, the cost-effective analysis cannot be fully drawn. These effects are investigated through a controlled experiment in which developers use specification-based and heuristic-based techniques to evolve

design models. With this in mind, the objective of this study is stated based on the GQM template (Basili et al., 1994) as follows:

*Analyze* composition techniques
*for the purpose of* investigating their effects
*with respect to* effort and correctness
*from the perspective of* developers
*in the context of* the evolution of design models.

Therefore, this controlled experiment addresses the research question RQ2, as stated in Section 1.3.

- **RQ2:** What is the relative effort of composing design models with specification-based composition techniques and heuristic-based composition techniques?

Based on this, we further decompose the RQ2 into two research subquestions described below:

- **RQ2.1:** What is the relative effort of composing two input models by using specification-based composition techniques with respect to heuristic-based composition techniques?
- **RQ2.2:** Is the number of correctly composed models higher with specification-based techniques than with heuristic techniques?



#### 4.1.1.2. Hypothesis Formulation

Table 8 describes the hypotheses for testing the effects of composition techniques on effort and correctness. These hypotheses are elaborated throughout this section.

*Hypothesis 1*. The first hypothesis of this section is that, although the specification-based composition technique provides a more systematic way to compose the input models, it does not reduce the composition effort. Our expectation is that developers invest more effort to write down the specifications rather than using the heuristic-based composition techniques. This can be explained based on the expectation that they are not intuitive or flexible enough to express the change requests. Moreover, the presence of inconsistencies in the output composed model may have a detrimental effect on the composition effort.

As developers should examine all points in the input models (affected by the specifications) or even "guess" which input model elements are incorrectly combined. Consequently, this additional effort may increase the composition effort rather than minimize it. However, it is by no means obvious that this hypothesis holds. It may be, for example, that they help developers to match and then compose the input models more quickly. With this in mind, the null hypothesis states that the specification-based technique requires less (or equal) effort to compose the input models than the heuristic-base technique. On the other hand, the alternative hypothesis states that the effort is significantly higher. These hypotheses are summarized as follows. Note that our expectation has a specific direction, which leads, in turn, to the definition of one-tailed hypotheses.

> **Null Hypothesis 1, $H_{1\text{-}0}$:** The specification-based composition techniques require less (or equal) effort than the heuristic-based composition techniques to produce $M_{AB}$ from $M_A$ and $M_B$.
>
> $H_{1\text{-}0}$: $Effort(M_A,M_B)_{Specification} \leq Effort(M_A,M_B)_{Heuristic}$
>
> **Alternative Hypothesis 1, $H_{1\text{-}1}$:** The specification-based composition techniques require more effort than the heuristic-based composition techniques to produce $M_{AB}$ from $M_A$ and $M_B$.
>
> $H_{1\text{-}1}$: $Effort(M_A,M_B)_{Specification} > Effort(M_A,M_B)_{Heuristic}$

For a more detailed investigation, we break this hypothesis in three subhypotheses ($H1_2$, $H1_3$, and $H1_4$). The goal is to evaluate the relative efforts (f, diff, and g) defined in the composition effort equation (see Figure 3). A complete formulation of these hypotheses can be seen in Table 8.

*Hypothesis 2.* The second hypothesis is that the use of specification-based composition techniques increases the number of correctly composed models. This is because developers can explicitly specify the composition relations between the input models. However, it is not clear whether this manner of realizing model composition promotes higher correctness of the output model. The need to explicitly take into consideration each of the models' properties (such as *isAbstract*), when specifying the relations, may cause difficulties to properly write down the specifications. If this is confirmed, then inconsistencies are inserted into the output composed model, compromising its correctness (i.e., $M_{CM} \neq M_{AB}$). With this in mind, the null hypothesis assumes that the specification-based

composition technique produces a lower (or equal) number of correctly composed models than the heuristic-based composition technique. On the other hand, the alternative hypothesis states that the specification-based technique produces a higher number of correctly composed models than the heuristic-based technique. In other words, the correctness (Cor) of the output composed models is usually assured when they are produced by the specification-based techniques. These hypotheses are presented as follows:

**Null Hypothesis 2, $H_{2\text{-}0}$:** Specification-based techniques produce a lower (or equal) number of correctly composed models than the heuristic-based techniques.

**$H_{2\text{-}0}$:** $Cor(M_{CM})_{Specification} \leq Cor(M_{CM})_{Heuristic}$

**Alternative Hypothesis 2, $H_{2\text{-}1}$:** Specification-based techniques produce a higher number of correctly composed models than heuristic-based techniques.

**$H_{2\text{-}1}$:** $Cor(M_{CM})_{Specification} > Cor(M_{CM})_{Heuristics}$

| Null Hypothesis | Alternative Hypothesis |
|---|---|
| $H1_{1\text{-}0}$: $Effort(M_A,M_B)_S \leq Effort(M_A,M_B)_H$ | $H1_{1\text{-}1}$: $f(M_A,M_B)_S > f(M_A,M_B)_H$ |
| $H1_{2\text{-}0}$: $f(M_A,M_B)_S \leq f(M_A,M_B)_H$ | $H1_{2\text{-}1}$: $f(M_A,M_B)_S > f(M_A,M_B)_H$ |
| $H1_{3\text{-}0}$: $diff(M_{CM},M_{AB})_S \leq diff(M_{CM},M_{AB})_H$ | $H1_{3\text{-}1}$: $diff(M_{CM},M_{AB})_S > diff(M_{CM},M_{AB})_H$ |
| $H1_{4\text{-}0}$: $g(M_{CM})_S \leq g(M_{CM})_H$ | $H1_{4\text{-}1}$: $g(M_{CM})_S > g(M_{CM})_H$ |
| $H2_{1\text{-}0}$: $Cor(M_{CM})_S \leq Cor(M_{CM})_H$ | $H2_{1\text{-}1}$: $Cor(M_{CM})_S > Cor(M_{CM})_H$ |
| $H2_{2\text{-}0}$: $Rate(M_{CM})_S \geq Rate(M_{CM})_H$ | $H2_{2\text{-}1}$: $Rate(M_{CM})_S < Rate(M_{CM})_H$ |
| **Dependent Variables** | |
| *Effort*: Effort to compose the input models (RQ3.1) | |
| *f*: Effort to apply the composition techniques (RQ3.1) | |
| *diff*: Effort to detect inconsistencies (RQ3.1) | |
| *g*: Effort to resolve the inconsistencies (RQ3.1) | |
| *Cor*: Correcteness of the composition (RQ3.2) | |
| *Rate*: Inconsistency rate of the composed model (RQ3.2) | |

Table 8: Tested hypotheses

The correctness of the model compositions is influenced by the presence (or not) of inconsistencies in the output composed model. Thus, we attempt to investigate if the specification-based technique also produces a lower inconsistency rate than the heuristic-based techniques. The new elaborated hypotheses are stated in Table 8.

#### 4.1.1.3. Context and Subject Selection

The subjects used the the traditional algorithms (Section 2.4.1), the IBM RSA (Section 2.4.2), and Epsilon (Section 2.4.3) to realize six evolution scenarios (Table 9). They had no previous knowledge about the design models or the changes. Thus, the evolution scenarios were typical tasks where developers were not the initial designers of the models. The design models used were fragments of industrial models captured from different application domains, such as financial applications and simulation of petrol extraction.

The experiment was conducted with 24 subjects (being 8 students) from Brazilian companies. All professionals held a Master's degree, Bachelor's degree, or equivalent, and had the required knowledge on software modeling and programming to participate in the experiment. Students were also invited to

| Task | Models | Required Changes to the Base Model |
|---|---|---|
| 1 | Oil Extraction | ***Add*** one class, one method, and one relationship. ***Modify*** one class from concrete to abstract. |
| 2 | Car System | ***Remove*** two methods and ***modify*** the direction of a relationship. |
| 3 | ATM | ***Add*** two classes and ***refine*** two classes from one. ***Remove*** this last class. |
| 4 | Supply Chain | ***Add*** two classes and one relationship. |
| 5 | Finance | ***Remove*** one class and ***add*** two methods to a particular class. ***Refine*** two classes from one and remove the last one. ***Remove*** one relationship. |
| 6 | Simulation of extraction | ***Modify*** the direction of five relationships. ***Modify*** the name of two methods. |

Table 9: The tasks of the evolution scenarios

participate in the experiment because of the recognized importance of students in empirical studies (Host et al., 2000); they are important to enable us to have subjects with different levels of experience in the study. They are from two Master and Doctoral programs in Computer Science at two Brazilian universities: Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and Federal University of Bahia (UFBA). These students attended to two courses: "empirical studies in software engineering" (PUC-Rio) and "software evolution" (UFBA). The experiments were part of the courses and were performed as practical laboratory exercises. In all cases, we had to ensure that every participant would undergo the same learning experiences and had previous experience with software evolution.

#### 4.1.1.4. Experimental Design

The experimental design of this study is characterized as a randomized complete block design with three treatments (Wohlin et al., 2000). The study had a set of activities that are organized in three phases (Figure 7). The subjects were randomly assigned and equally distributed to the treatments. The distribution follows a within-subjects design in which all subjects serve in the three treatments. This allowed us to compare the data collected. In each treatment, the subjects used a composition technique to carry out two experimental tasks. As three composition techniques were used, then six tasks were performed. Therefore, the experimental design was, by definition, a balanced design.

#### 4.1.1.5. Operation and Material

**Operation**. Figure 7 shows through an experimental process how the three phases were organized. The subjects individually performed all activities to avoid any threat to the experimental process. The activities are further described as follows.

*Training*. All subjects received training to be sure of their familiarity with both software modeling and model composition techniques. It is important to

highlight that the subjects were not aware about the research questions (and hypotheses) of the study in order to avoid biased results.

*Apply the techniques*. The participants were encouraged to compose $M_A$ and $M_B$ based on a document with the evolution descriptions, which define how the model elements should be changed. This document describes (in a more elaborated way) the experimental tasks shown in Table 9. The measure of application effort, video and audio records, and a composed model represent the results of this activity. Each subject performed it six times (for each task presented in Table 9). The video and audio records were later used during the qualitative studies. It is important to point out that a participant (subject x) produced $M_{CM}$ and in the second phase other (subject n-x) detected and resolved the inconsistencies in $M_{CM}$ in order to produce $M_{AB}$.

*Detect inconsistencies*. Subjects reviewed $M_{CM}$ in order to detect inconsistencies. For this, they checked if $M_{CM}$ had the changes described in the evolution descriptions and if the contradicting changes between $M_A$ and $M_B$ were correctly solved. As a result, we have the measure of the detection effort, video and audio records, and a list of inconsistencies identified.

*Resolve inconsistencies*. The subjects resolved the inconsistencies previously localized to produce $M_{AB}$. The resolution effort was also measured and the video and audios were registered.

*Make interview.* Subjects reflected on their experiences on model composition using an in-depth semi-structured interviews. These interviews enriched the qualitative data collected. For example, it was possible to observe, for example, some non-verbal communication issues that help us to infer the study's findings.

*Answer questionnaire*. The subjects filled out a questionnaire. This allows us to collect their background (i.e., their academic background and work experience) and apply some inquisitive questions.

**Material**. The subjects used UML class diagrams in the experiment because they are the most used design models in practice. Each model had approximately eight classes and seven relationships. We have avoided using large models due to some reasons. First and more importantly, proper modeling practices determine that each model should not have much more than seven modular units. Second, experimental guidelines recommend that artifacts used in experiments should be

simple; otherwise, the size and complexity may affect the results in undesirable ways (Wohlin et al., 2000).

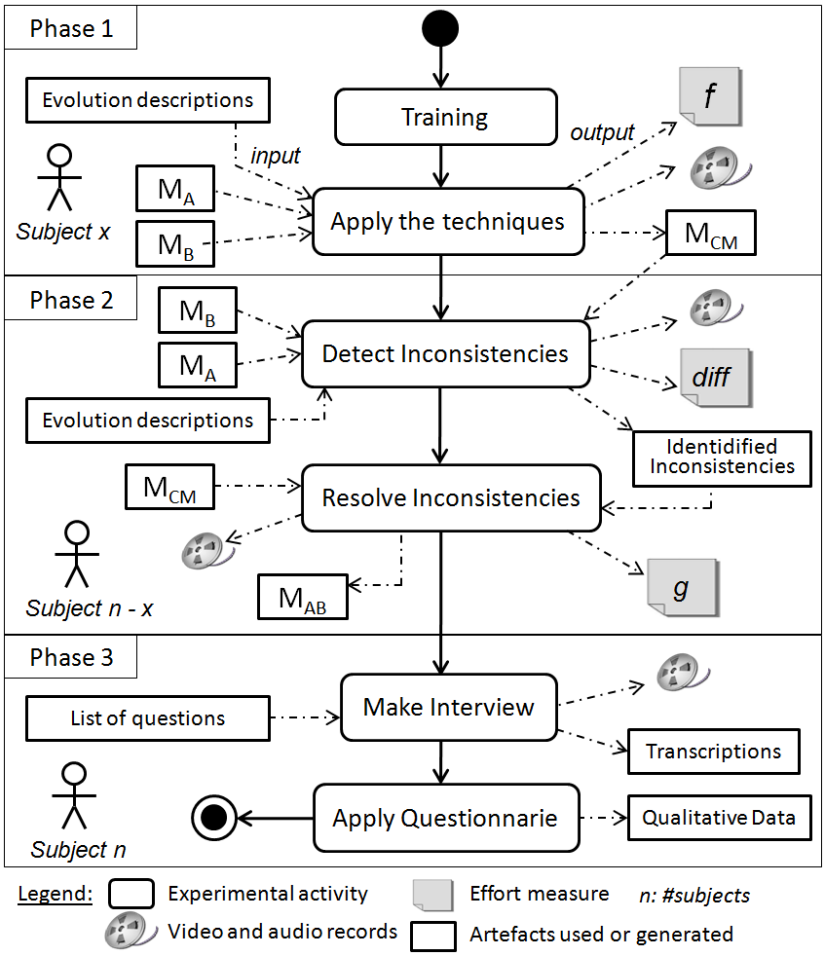



Figure 7: The Experimental process

Third, the delta model should be as small as possible; otherwise, the subjects would have conflict management problems (Mens, 2002). In (Asklund, 1994), Asklund recommends that software changes should be relatively small so that the number of conflicts is not very high. In (Perry et al., 1998), Perry confirms this idea from a statistical basis in a large-scale industrial case. As previously mentioned, the subjects used another material named evolution description. This file describes the changes that should be performed in $M_A$ to transform it into $M_{AB}$. Table 9 illustrates the changes.

#### 4.1.1.6. Instrumentation and Measurement

The independent variable of this study is the choice of composition techniques. This variable is nominal and assumes two values: specification-based technique and heuristic technique. We investigate the impact of these independent variables on the following dependent variables.

- *Effort*. This variable measures the overall time (in minutes) invested by subjects to compose the input models ($H_{1\text{-}1}$). This measure is required by three other variables: effort to apply model compositions ($H_{1\text{-}2}$), effort to detect inconsistencies ($H_{1\text{-}3}$), and effort to resolve inconsistency ($H_{1\text{-}4}$).
- *Correctness*. The correctness of a composition ($H_{2\text{-}1}$) is asserted when the output composed model produced is correct with respect to intended model that fully satisfies the evolution description (i.e., $M_{CM} = M_{AB}$). The composed model produced may be rated as either correct or incorrect. Note that each composition performed by a subject produces a dichotomous data (correct or incorrect) defined from the comparison between $M_{CM}$ and $M_{AB}$. Therefore, this variable is a categorical one. Note that a composed model with one of the previously described inconsistencies (Section 3.3.4) would be deemed as incorrect. To promote a deeper understanding, we also investigate the inconsistency rate of the incorrectly composed model. It represents the ratio of the number of inconsistencies of a composed model divided by its number of model elements ($H_{2\text{-}2}$). The inconsistencies considered were previously described in Chapter 3.

#### 4.1.1.7. Analysis Procedures

*Quantitative Analysis.* We performed descriptive statistics to analyze its normal distribution (Kitchenham et al., 2008) and statistical inference to test the hypotheses. The level of significance of the hypothesis tests was $\alpha = 0.05$. The analyses were carried out to test the hypotheses both individually for each experiment task and across all experiment tasks. To test $H_{1\text{-}1}$ (and its

subhypotheses) we applied the non-parametric Wilcoxon signed-rank test (Wohlin et al., 2010) for the six tasks. This test is similar to the t-test, but does not require two separate sets of independent and identically distributed samples. Note that we have a same subject design. As a result, our samples are dependent. Moreover, the non-parametric Friedman ANOVA test (Conover, 1999) was also applied to reduce some potential threats to the validity of statistical conclusions. To test $H_{2\text{-}1}$ we applied the McNemar's test for marginal homogeneity (Wohlin et al., 2010; Devore etal., 1999). To test $H_{2\text{-}2}$ we consider the inconsistency rate produced during the evolution scenarios. As in $H_1$, we also applied the Wilcoxon signed-rank test and Friedman test.

*Qualitative Analysis.* Qualitative data were collected from some sources: questionnaire, audio/video records, and transcriptions, think aloud comments and interviews. This helped us to potentially obtain some complementary evidence to explain the quantitative results and then derive the conclusions from a chain of evidences (Jorgensen, 2005), which are formed from the systematic alignment of the quantitative and qualitative data.



### 4.1.2. Experimental Results

In this section, we present and interpret the experimental results about the RQ2.1 and RQ2.2. For this, a complete statistical analysis is presented, including descriptive statistics and statistical inference.

#### 4.1.2.1. RQ2.1: Effort and Composition Techniques

*Descriptive Statistics.* The collected data indicate that the developers tend to spend less effort by using heuristic-based techniques rather than the specification-based techniques. In fact, they required less effort to-be applied (*f*), detect inconsistencies (*diff*), and resolve inconsistencies (*g*). Consequently, the general composition effort was also smaller. The traditional algorithms required less effort than the IBM RSA, which in turn required less than the Epsilon. This is a very interesting finding because the common sense would be otherwise i.e., developers

would invest less effort by using the Epsilon and IBM RSA. Table 10 shows pieces of evidence through descriptive statistics of the collected data.

| | Effort | | | f | | | diff | | | g | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **TRA** | **RSA** | **EPS** | **TRA** | **RSA** | **EPS** | **TRA** | **RSA** | **EPS** | **TRA** | **RSA** | **EPS** |
| N | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 |
| Min | *5* | 5 | 9 | 2 | 3 | 4 | 1 | 1 | 1 | 0 | 0 | 0 |
| 25th | 7 | 11 | 14 | 4 | 6 | 8.7 | 2 | 2 | 3 | 0 | *0* | *0.5* |
| Med | 11 | 14 | 21 | 6 | 8 | 12 | 3 | 4 | 4.5 | 0.5 | 2 | 3 |
| 75th | 18 | 24 | 34 | 9 | 11 | 17 | 5.2 | 8 | 8.7 | 4 | 7 | 9 |
| Max | 31 | 66 | 114 | 25 | 22 | 39 | 11 | 22 | 38 | 9 | 22 | 38 |
| Mean | 13.3 | 18.2 | 29.1 | 7.2 | 9.0 | 14.8 | 3.9 | 5.3 | 7.7 | 2.1 | 3.8 | 6.6 |
| St D. | 6.9 | 11.0 | 23.3 | 4.4 | 4.2 | 8.8 | 2.4 | 4.4 | 8.2 | 2.9 | 5.1 | 9.1 |

N: #compositions, Min: minimum, Med: median, Max: maximum,
StD: Standard Deviation, TRA: traditional, RSA: Rational Software Architect, EPS: Epsilon

Table 10: Descriptive statistic for the composition effort

Regarding the median of the general effort, it grew significantly from 11 to 14 and 21 by using RSA and Epsilon, respectively. This superior effort represents an increase by about 27.27 and 90.90 percent. This upward trend was not only observed in the measure of the general effort, but also in the *f*, *diff*, and *g*. Considering the mean of effort computed, this evidence was still clearer. The general effort increased from just over 13 minutes in the Traditional algorithms to 18.26 minutes in the IBM RSA, reaching almost 30 minutes in the Epsilon. This represents a rise of 36.88 and 118.66 percent, respectively. This evidence, therefore, demonstrate that the developers in fact tend to invest less effort with heuristic-based techniques than specification-based one. The next step it is to scrutinize whether this evidence are statistically significant to reject the null hypotheses ($H_{1\text{-}1}$, $H_{1\text{-}2}$, $H_{1\text{-}3}$ and $H_{1\text{-}4}$) stated in Section 4.1.1.2.

*Hypothesis Testing.* Since the Shapiro-Wilk test (Sheskin, 2007) indicated deviations from normality, the Wilcoxon signed-rank test and Friedman test were applied. While the Wilcoxon test allowed us to realize a pairwise comparison of the distributions, Friedman test allowed checking if there exist significant differences among the three techniques under investigation. We test $H_1$ (and its subhypotheses) to evaluate the RQ2.1 in the six experimental tasks (Table 11).

Table 11 and Table 12 show the p-values for the pairwise comparison. Bold p-values highlight statistically significant results (i.e., p-value < 0.05). They indicate the rejection of the respective null hypothesis. The main feature is that the

general composition effort (*f*, *diff* and g) using heuristic-based techniques were significantly lower than using automated techniques in all cases. Still, by using the traditional algorithms this significance is higher. Thus, we can reject the H1 null hypotheses (and its $H1_{1\text{-}0}$, $H1_{2\text{-}0}$, $H1_{3\text{-}0}$ e $H1_{4\text{-}0}$). For example, in row 2 of Table 12, for measure Effort, between RSA and EPS, the W is negative (-544) and p-value is less than 0.05 (p = 0.0015) our selected significance level). This means that the composition effort by using the IBM RSA is significantly lower than one using Epsilon. Still in row 2 just a null hypothesis was not rejected in just one case: the effort to detect inconsistencies considering the IBM RSA and Epsilon (p-value = 0.0891). This means that the subjects did not spend substantially different effort to detect inconsistencies in IBM RSA and Epsilon. Therefore, our initial intuition that the specification-based technique would not reduce the composition effort is confirmed.

Given this surprising result, we were encouraged to apply the Friedman's test to eliminate threats to statistical conclusion validity. This test also confirmed the above conclusions. The results are shown in Table 13. Again bold p-value (<0.05) means that there is a significant difference between the mean ranks in repeated measures of effort. Hence, there is sufficient evidence to reject the null hypothesis, and conclude that there is a difference between the composition efforts at the 0.05 level of significance. For example, in row 1, a chi-Square ($\chi^2$)

| | | **f($M_A$,$M_B$)** | | | **diff($M_{CM}$,$M_{AB}$)** | | |
|---|---|---|---|---|---|---|---|
| task | statistics | TRA *vs* RSA | TRA *vs* EPS | RSA *vs* EPS | TRA *vs* RSA | TRA *vs* EPS | RSA *vs* EPS |
| All | p-value | **0.0269** | **0.0001** | **0.0003** | **0.0337** | **0.0003** | 0.0891 |
| | W | -77 | -834 | -588 | -233 | -533 | -186 |
| 1 | p-value | 0.4294 | 0.4062 | 0.3628 | 0.1438 | 0.5 | 0.3981 |
| | W | -4 | 5 | 6 | 16 | -1 | 4 |
| 2 | p-value | 0.2305 | **0.0078** | **0.0342** | **0.0178** | 0.2284 | 0.2303 |
| | W | -12 | -34 | -27 | -21 | -8 | 8 |
| 3 | p-value | 0.3762 | **0.0171** | 0.1548 | 0.2731 | 0.0526 | 0.1250 |
| | W | -4 | -26 | -16 | -8 | -20 | 8 |
| 4 | p-value | 0.2931 | **0.0111** | **0.0171** | 0.2931 | 0.0634 | **0.0369** |
| | W | -3 | -28 | -26 | 3 | -19 | -22 |
| 5 | p-value | 0.0747 | **0.0039** | **0.0177** | **0.0207** | .0.848 | 0.1982 |
| | W | -18 | -36 | -31 | -11 | -25 | -11 |
| 6 | p-value | 0.2188 | 0.0750 | 0.1094 | 0.0672 | **0.0111** | 0.1163 |
| | W | -9 | -18 | -13 | -12 | -28 | 15 |

W: sum of signed ranks, f: effort to apply the composition technique,
Diff: inconsistency detection effort, RSA: IBM rational software architect, EPS: Epsilon, TRA: traditional algorithm

Table 11: Wilcoxon test results for application and detection effort

value of 26.21 and p = 0.001 (with p<0.05) indicates a statistically significant difference in the effort measures associated with the three techniques.

| | | General Effort | | | $g(M_{CM})$ | | |
|---|---|---|---|---|---|---|---|
| task | statistics | TRA *vs* RSA | TRA *vs* EPS | RSA *vs* EPS | TRA *vs* RSA | TRA *vs* EPS | RSA *vs* EPS |
| All | p-value | **0.0056** | **0.0001** | **0.0015** | **0.0164** | **0.0003** | **0.0422** |
| | W | -420 | -900 | -544 | -261 | -423 | -248 |
| 1 | p-value | 0.3349 | 0.5 | 0.5 | 0.4661 | 0.3989 | 0.3054 |
| | W | 6 | 0 | 0 | -2 | -4 | -7 |
| 2 | p-value | **0.0149** | **0.0039** | 0.1462 | 0.0828 | 0.0528 | 0.2226 |
| | W | -32 | -36 | -16 | -14 | -24 | -10 |
| 3 | p-value | 0.2891 | **0.0156** | 0.1355 | 0.2303 | 0.0625 | 0.1238 |
| | W | -8 | -21 | -14 | -8 | -10 | 12 |
| 4 | p-value | 0.5 | **0.0111** | **0.0156** | 0.5 | **0.0178** | **0.0445** |
| | W | -1 | -28 | -26 | 0 | -21 | -17 |
| 5 | p-value | **0.0167** | **0.0071** | 0.977 | 0.2763 | 0.4326 | 0.5 |
| | W | -26 | -36 | -20 | -8 | -3 | -1 |
| 6 | p-value | **0.0452** | **0.0313** | 0.4228 | **0.0463** | 0.1250 | 0.4219 |
| | W | -21 | -23 | 3 | -17 | -28 | 28 |

W: sum of signed ranks, g: resolution effort, RSA: IBM rational software architect, EPS: Epsilon, TRA: traditional algorithm

Table 12: Wilcoxon test results for the resolution and general effort

| **Task** | **Statistics** | **Effort** | **$f(M_A,M_B)$** | **$diff(M_{CM},M_{AB})$** | **$g(M_{CM})$** |
|---|---|---|---|---|---|
| all | p-value | **0.0001** | **0.0001** | **0.0048** | **0.0017** |
| | $\chi^2$ | 26.21 | 26.64 | 10.66 | 12.76 |
| 1 | p-value | 0.7682 | 0.8135 | 0.5690 | 0.3977 |
| | $\chi^2$ | 0.8571 | 0.4 | 1.1515 | 1.931 |
| 2 | p-value | **0.0048** | 0.0789 | 0.0789 | 0.1495 |
| | $\chi^2$ | 9.75 | 5.25 | 5.12 | 3.931 |
| 3 | p-value | 0.1916 | 0.1916 | 0.4861 | 0.3046 |
| | $\chi^2$ | 3.630 | 3.630 | 1.68 | 2.5454 |
| 4 | p-value | **0.0084** | **0.0036** | **0.0272** | **0.0207** |
| | $\chi^2$ | 8.615 | 9.333 | 6.333 | 7.5238 |
| 5 | p-value | **0.0099** | 0.0024 | 0.0024 | 1 |
| | $\chi^2$ | 8.968 | 10.516 | 10.51 | 0 |
| 6 | p-value | 0.0854 | **0.0272** | **0.0207** | **0.0003** |
| | $\chi^2$ | 5.429 | 6.231 | 7.6923 | 12.074 |

$\chi^2$: Friedman's Chi-Square, $\alpha = 0.05$

Table 13: Statistical test for the Friedman Test

### 4.1.2.2. RQ2.2: Correctness and Composition Techniques

*Descriptive Statistics*. Figure 8 shows the correctness of the compositions generated by using the three techniques: traditional algorithms, Epsilon, and IBM RSA during the six experimental tasks. The axis-y represents the proportions of numbers of $M_{AB}$ (the intended model) achieved by the number of compositions realized in each task using each composition technique, while the axis-x consists of the experimental tasks. Recall that the composition of $M_A$ and $M_B$ often $M_{CM}$ instead of $M_{AB}$. In this case, we calculate the rate of $M_{AB}$ produced in 46 compositions. Thus, the histogram shows how the correctly composed model happened throughout the experimental tasks.

The main outstanding feature is the lack of a distribution pattern of the proportions of correctly composed model in the tasks. For example, in task 1, TRA produced a lower proportion of correctly composed model than RSA and EPS. That is, the intended model was generated in 42.86 percent of the cases in TRA, whereas 57.14 percent of the cases in RSA and EPS. On the other hand, in task 2, TRA outnumbers RSA and EPS. It produced the intended model in 71.43 percent of the cases, while EPS and RSA produced 28.57 and 57.14 percent of the cases, respectively. Although TRA has obtained low measures in task 3 in comparison to task 2 (a decrease from 71.43 to 42.86 percent), it still got a superior value compared to EPS and RSA — i.e., value by about three times higher than the measure of EPS and RSA, comparing 42.86 and 14.29 percent.

Moreover, TRA and EPS had an equal proportion of correctly composed model in task 4, presenting an increase of around 20 percent considering RSA. On the other hand, in task 6, this superiority was reversed. RSA got double the value than TRA and EPS, comparing 28.57 and 57.14 percent. In task 5, the superiority of TRA and RSA considering EPS was evident. Still, subjects obtained the intended model by using TRA and RSA in all composition cases, while less than half of the cases in EPS. We have observed that TRA got a higher number of intended models than RSA and EPS. The subjects produced the intended model in 61.90 percent of the compositions using TRA against 59.52 and 42.86 percent using the RSA and Epsilon technique, respectively. Two interesting insight were that (1) the composition techniques require different effort in front of the

categories of evolution changes, and (2) the specification-based technique does not guarantee superiority in terms of correctness in comparison with the heuristic-based techniques.

Table 14 shows the descriptive statistics of the inconsistency rate of the composed models. Our initial expectation was that the specification-based technique would minimize the inconsistence rate whereas also get lower measures than the heuristic-based techniques. However, this expectation was not confirmed. We have observed that, in most cases, the inconsistency rate was similar using specification-based and heuristic-based techniques. This means that developers will not produce correctly composed model by using a technique based on composition specifications. Rather, the output models will have equal (or even more) inconsistency rate. For example, on average, EPS produced a higher inconsistency rate than TRA and RSA. Table 14 shows evidence of the superiority of EPS compared to the TRA. In general, the mean of the inconsistency rate in Epsilon is two times higher than one TRA and RSA, increasing by about 123 and 176 percent, respectively. Still note that the inconsistence rate in RSA is also higher than in TRA. In short, the inconsistency rate in EPS is higher than RSA, which outnumber TRA. This suggests that the inconsistency rate have favored TRA in comparison with RSA and EPS in most cases. This implies that to some extent the number of inconsistencies is decreased whenever the composed model is produced by TRA and RSA. In the next section, we test H5 and H6 to check if whether the differences observed are substantially significant.

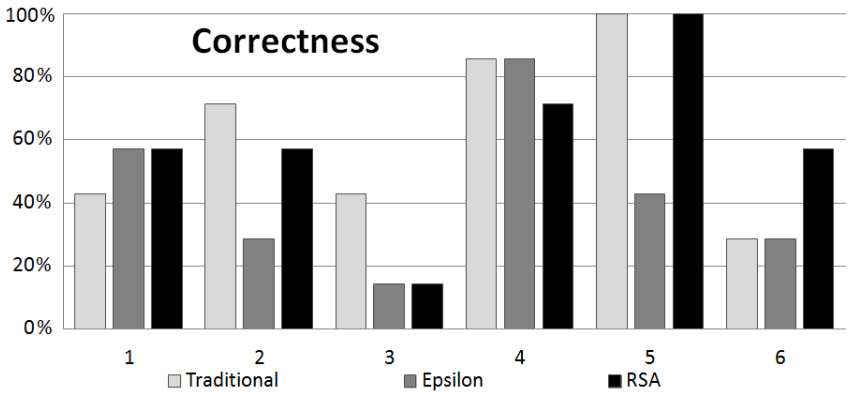



Figure 8: The correctness of the output composed model

| | N | Min | 25th | Med | 75th | Max | Mean | St D. |
|---|---|---|---|---|---|---|---|---|
| TRA | 46 | 0 | 0 | 0 | 0.31 | 1.63 | 0.26 | 0.45 |
| RSA | 46 | 0 | 0 | 0 | 0.425 | 1.22 | 0.21 | 0.29 |
| EPS | 46 | 0 | 0 | 0.47 | 0.78 | 5.22 | 0.58 | 0.88 |

N: #compositions, Min: minimum, Med: median, Max: maximum, StD: Standard Deviation,

Table 14: Descriptive statistic for the inconsistency rate

*Hypothesis Testing.* RQ2.2 evaluates if the specification-based techniques assure a higher number of correctly composed model than the heuristic-based techniques. We test $H2_1$ (and its sub hypothesis $H2_2$) to investigate RQ2.1. For this, we apply the McNemar test. Table 15 shows the chi-square statistic ($\chi^2$) and p-values for the pairwise comparisons. In all cases, the p-value is large ($p > 0.05$), so the null hypothesis of $H2_{1\text{-}0}$ cannot be rejected. Although the p-value to the six tasks is not shown in the table, the p-value took values greater than 0.05 in the six tasks. This implies that there is no significant difference between the proportions of correctly composed model of the composition techniques.

| Task | Comparison | $\chi^2$ | p-value |
|---|---|---|---|
| all | TRA vs RSA | 0.27 | 0.606 |
| | TRA vs EPS | 0.75 | 0.387 |
| | RSA vs EPS | 0 | 1 |

$\chi^2$: Friedman's Chi-Square, $\alpha = 0.05$

Table 15: McNemar test results for correctness

We test $H2_2$ by applying the Wilcoxon test. Table 16 depicts the pairwise p-values for each measure. Bold p-values point out statistically significant results. They also indicate the rejection of the null hypothesis. Note that the sum of signed ranks (W) shows the direction in which the result is significant. For example, in row 2, W is negative (-250) and p-value is lower than 0.05 ($p = 0.0301$) for the measure between TRA *vs* EPS. This means that the inconsistency rate for TRA is significantly lower than in EPS. RSA also obtained an inconsistence rate significantly lower ($p = 0.001$) than EPS. For instance, in row 1, the W is negative (-5) and p-value is higher than 0.05 for the inconsistency rate between TRA vs. RSA. This means that the inconsistency rate for TRA is lower, but no significantly lower than RSA.

| Task | Statistics | Rate |
|---|---|---|
| all | p-value | **0.0258** |
| | $\chi^2$ | 7.314 |
| 1 | p-value | 0.7682 |
| | $\chi^2$ | 0.4210 |
| 2 | p-value | 0.0854 |
| | $\chi^2$ | 4.666 |
| 3 | p-value | 0.4861 |
| | $\chi^2$ | 1.407 |
| 4 | p-value | 0.7682 |
| | $\chi^2$ | 0.666 |
| 5 | p-value | 0.4861 |
| | $\chi^2$ | 2 |
| 6 | p-value | 0.2366 |
| | $\chi^2$ | 3.3076 |

$\chi^2$: Friedman's Chi-Square, $\alpha = 0.05$

Table 16: Friedman test result for inconsistencv rate

These results also encouraged us to apply the Friedman test (Table 17). We obtained a chi-square value ($\chi 2$) of 7.314 with p-value = 0.0258, which is lower than 0.05 hence is significant. This means that there exists a significant difference between the inconsistency rate by using TRA, RSA, and EPS. However, considering each experimental task, the results did not take significance (i.e., p > 0.05). This means that a technique did not significantly outperform the other two ones. For example, in task 1, the chi-square value ($\chi 2$) of 0.4210 with a p-value = 0.7682 indicates that there exist no significant difference between the three techniques in terms of inconsistency rate.

| Task | statistic | Inconsistency Rate | | |
|---|---|---|---|---|
| | | **TRA *vs* RSA** | **TRA *vs* EPS** | **EPS *vs* RSA** |
| All | p-value | 0.4851 | **0.0301** | **0.0011** |
| | W | -5 | -250 | 344 |
| 1 | p-value | 0.2188 | 0.2188 | 0.5000 |
| | W | 7 | 7 | -1 |
| 2 | p-value | 0.3750 | 0.2188 | 0.0781 |
| | W | 2 | -9 | 15 |
| 3 | p-value | 0.2002 | 0.1094 | 0.1355 |
| | W | -9 | -16 | 14 |
| 4 | p-value | 0.5000 | 0.5000 | 0.2071 |
| | W | -1 | 1 | -4 |
| 5 | p-value | 0.5000 | 0.1875 | 0.1250 |
| | W | 1 | -6 | 8 |
| 6 | p-value | 0.1982 | 0.1094 | **0.0469** |
| | W | 9 | -16 | 17 |

W: sum of signed ranks, g: resolution effort, RSA: IBM rational software architect, EPS: Epsilon, TRA: traditional algorithm

Table 17: Wilcoxon test results for the corretness

This finding can be explained based on two reasons captured during the interviews and analysis of the qualitative data (i.e., video records and audio). First, the specification-based technique adds a difficulty undesired to match and compose the input model elements, as it was not particularly challenging for the subjects write down the compositions. In particular, this was more often observed in compositions dominated by relations of the type one-to-many (1:N) or many-to-many (N:N) between the input model elements. The specification-based technique proved to be a highly intensive manual task and more prone to errors. Second, the IBM RSA shows the commonalities and differences between the input models in multiple views. This jeopardizes the subjects create a "big picture view" of the output intended model. Finally, we summarized three lesson learned as follows: (1) the model composition techniques should be more intuitive and flexible to express different categories of changes; (2) the techniques should represent the conflicts between the input models in more innovative views and report them soon after they arise; (3) new composition technique should be a mixture of specification-based and heuristic-based techniques; and (4) the heuristic-based techniques consumed less effort and were more effective than the specification-based technique. This suggests that the tools for specification-based techniques may be very rigid and need more flexibility so that, for example, developers can adjust the composition specification considering their experience.



### 4.1.3. Limitations of Related Work

Model composition techniques have been studied in many research areas such as merging of state charts (Whittle et al., 2010), composition of software product lines (Thaker et al., 2007; Jayaraman et al., 2007), composition of aspect-oriented models (Klein et al., 2006), and mainly composition of UML design models (Clarke, 2001; Dingel et al., 2008). Such research initiatives focus on proposing model composition techniques or even creating innovative modeling languages. However, the evaluation of the developers' effort on composing design models using the proposed techniques is still incipient. The lack of quantitative and qualitative indicators on composition effort hinders mainly the understanding of side effects peculiar to certain composition techniques.

Current works have notably aimed at evaluating modeling languages, such as UML in terms of some quality attributes such as comprehensibility (Ricca et al., 2010) and completeness (Lange et al., 2004). Although UML has been adopted as the industry standard modeling language, it is just a point of investigation in empirical studies considering model composition. In general, most of the research on the interplay of effort and composition techniques rest on subjective assessment criteria (France & Rumpe, 2007; Mens, 2002; Uhl, 2008; Farias, 2010a). Mens points out the need for studies aimed at investigating the effort to integrate software artifacts such as the source code. Uhl also highlights the superior difficulty of composing models compared to code and reinforce the need for studies reporting the effort required to compose design models.

Even worse, this has led to depend on feedback's experts, who have built up an arsenal of mentally held indicators to analyze the growing complexity of models and then evaluate the effort on composing them (Farias et al., 2010). Consequently, developers ultimately rely on feedback from experts to determine "how well" the compositions were performed. There are many examples of model composition techniques in the literature such as MATA (Whittle & Jayaraman, 2010), Kompose (Kompose, 2011), Epsilon (Epsilon, 2011), and IBM RSA (IBM RSA, 2011). Nevertheless, they will only be useful if the quality of the output composed models (e.g., correctness) is assured and the composition effort required is low. Unfortunately, these approaches do not offer any insight or empirical evidence about the effort required to compose design models. As a matter of fact, the current literature about composition techniques points out the absence of empirical studies and do highlight the importance of studies reporting empirical evidence (Farias et al., 2010; France et al, 2007; Whittle et al., 2010; Apel et al., 2011, Sarma et al., 2011; Mens, 2002; Nejati et al., 2007). To the best of our knowledge, our results are the first to empirically investigate the topics of the research questions in a systematic and controlled way.

### 4.1.4. Threats to Validity

This section discusses how the internal, statistical conclusion, construct, and external threats were mitigated.

*Internal Validity.* The inferences between the independent and dependent variables are internally valid if a causal relation is demonstrated (Wohlin et al., 2000; Kitchenham et al., 2008). 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.

*Statistical Conclusion Validity.* We checked if the independent and dependent variables were submitted to suitable statistical methods. For this, two points were analyzed. First, whether the presumed cause and effect covaries. The study of the normal distribution of the data collected reduced this threat; as it was possible to verify if parametric or non-parametric statistical methods might be used (or not). In doing so, the Shapiro-Wilk test (Sheskin, 2007) was used to determine how likely the collected sample was normally distributed. As the dataset did not assume a normal distribution, non-parametric statistics were used. Hence, the assumptions of the test statistics were not violated. Second, how strongly the inferences covary. The hypotheses were tested at significance level of 0.05 level (p-value $\leq 0.05$). In addition, some guidelines (Wohlin et al., 2000; Shadish et al., 2002; Sjoberg et al., 2002) were followed so that the assumptions of the statistical test were not violated and the homogeneity of the subjects' background was assured.

*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. That is, it answers the question: "Are we actually measuring what we think we are measuring?" All variables of this study were quantified based on previous studies (Farias et al., 2010). Thus, they were defined and independently validated. Moreover, the concept of effort used in our study is well known in the literature (Jorsengen, 2005). Therefore, we are sure that the quantification method used is correct, and the quantification was accurately done.

*External Validity.* We analyzed whether the causal relationships investigated in this study could be held over variations in people, treatments, composition techniques, and the design models. There are reasons to believe the results generalize beyond the three techniques used, but leave it to further work to fully test this.

### 4.1.5. Concluding Remarks of the First Study

The previous section represents a first controlled experiment to assess and compare the specification-based and heuristic-based techniques in terms of effort and correctness. By controlling these variables, we investigated the effects of model composition techniques on six quality notions, namely syntactic, semantic, effort, application, detection, and resolution ones. From the quantitative and qualitative analyses, we observed some findings.

First, developers tend to have an additional difficulty to match and compose the input model elements by using specification-based composition techniques, such as Epsilon. The main reason was that the creation of composition specifications has often been an effort-consuming task. Developers invested so much effort to define how the properties of the model elements should be related. This additional difficulty was converted into a superior effort to compose the design models. On the other hand, developers invested less effort to compose the design model by using the heuristic-based composition techniques, such as IBM RSA. The techniques did not require an extra effort to define the similarity between the model elements and realize the compositions.

Second, the composition techniques required different amount of effort in specific composition scenarios. That is, the type of change found in the delta model affected the composition effort. The compositions whose goal were to only accommodate new model elements from the delta model into the base model required similar effort between the heuristic-based and specification-based composition techniques. On the other hand, composition scenarios in which were not dominated by additions, the effort invested to compose the models were different. In particular, this was more often observed in compositions dominated by relations of the type one-to-many (1:N) or many-to-many (N:N) between the input model elements. The specification-based technique proved to be a highly intensive manual task and more prone to errors.

Moreover, we summarized three lessons learned as follows: (i) all the model composition techniques should be more flexible to express different categories of changes (Section 4.1.2.1); (ii) the techniques should report conflicts as soon as

they arise (Section 4.1.2.1); such conflicts between the input models should be represented in more intuitive views; (iii) new composition technique should be a mixture of specification-based and heuristic-based techniques as if a set of adequate composition rules are defined and reused, the specification-based techniques can present better results compared to the heuristic-based techniques; and (iv) the heuristic-based techniques consumed less effort and were more effective than the specification-based technique. The latter finding suggests that the tools for specification-based techniques are hard to perform model composition, mainly due to the additional difficulty of manually specifying how the input models should be composed, given the problem at hand.

In addition, we found that the specification-based techniques neither reduce the developers' effort nor guarantee the correctness of the compositions. Even worse, the traditional composition algorithms outperformed the specification-based technique to some extent. Given that little is known about the real effort that developers invest to compose design models, this study might be seen as a first exploratory study that investigates the effects of the composition techniques on the effort in a systematic and controlled manner. However, further empirical studies are still required to better understand if these findings are confirmed or not in other contexts, considering other design models, having different evolution scenarios, and evaluating new composition techniques.



## 4.2. Analyzing the Effort of Composing Design Models of Large-Scale Software

As previously mentioned, there has been a significant body of research into defining model composition techniques in the area of governance and management of enterprise design models (Norris & Letkeman, 2011), software configuration management (Perry et al., 2001), composition of software product lines (Jayaraman et al., 2007; Thaker et al., 2007), aspect-oriented modeling (Whittle et al., 2009; Klein et al., 2006), and integration of state charts (Whittle & Jayaraman, 2010).

Unfortunately, both commercial and academic model composition techniques suffer from the *composition conflict problem*. That is, models to-be composed conflict with each other and developers are usually unable to deal with

the conflicting changes. Hence, these conflicts are transformed into inconsistencies in the output composed model (Diskin et al., 2010). For example, two developers concurrently work on a same class diagram, which has two abstract classes *A* and *B*. The first developer creates an inheritance relationship between the abstract class *A* and *B* (i.e., *B.superclass = A*), while the second developer modifies the class *A* from abstract to concrete (i.e., *A.isAbstarct = false*). Although these are simple changes, usually the developers are not aware of these conflicting changes performed in parallel. Hence, the composition of the partial models produces an inconsistent class diagram i.e., an inheritance relationship between an abstract class *B* and a concrete class A. The current composition techniques cannot automatically resolve these inconsistencies (Egyed, 2010; Egyed, 2007); because inconsistency resolution relies on an understanding of what the models actually mean. This semantic information is typically not included in any formal way in the design models. Consequently, developers must invest some effort to manually detect and resolve these inconsistencies. The problem is that high effort compromises the potential benefits of using model composition techniques, such as gains in productivity.

To date, however, nothing has been done to (1) *quantify* the composition effort in key software development activities, including software evolution, and (2) *characterize* the influential factors that can affect the developers' effort in practice. Hence, developers cannot adopt or assess model composition based on practical, evidence-based knowledge from experimental studies. Rather, they rely on diverging feedbacks from evangelists; these feedbacks often diverge.

The goal of this second study, therefore, is to report on five industrial exploratory case studies that aimed at (1) providing empirical evidence about model composition effort, and (2) describing the influential factors that affected the developers' effort. These studies were performed in the context of the evolution of design models of five large-scale software systems. During 56 weeks, 297 evolution scenarios were performed, leading to 2.288.393 compositions between modules, classes, interfaces, and relationships. We draw the conclusions from quantitative and qualitative investigations including the use of metrics, interviews, and observational studies. We investigate the composition phenomena in their context, stressing the use of multiple sources of evidence, and making clear the boundary between the identified phenomenon and its context. While we

believe this study is representative of the broader issues, we make no claims about the generality of our results beyond the composition of UML class and sequence diagrams of large-scale software.

The following subsections are organized as follows. Section 4.2.1 introduces the main concepts and knowledge that are going to be used and discussed throughout the thesis. Section 4.2.2 elaborates the composition scenario that will be used as a frame of reference. Section 4.2.3 describes the research methodology followed. Section 4.2.4 presents the analysis of composition effort. Section 4.2.5 contrasts our work with related work. Finally, Section 4.2.6 discusses some concluding remarks and future work.

### 4.2.1. Background

Three-way merge algorithm (Mens, 2002) is a well-known method to merge software artifacts. This method has increasingly been incorporated into the most popular and robust industrial modeling tools, such as IBM RSA (IBM RSA, 2011). This algorithm refines the specification of model composition cited previously. Instead of taking into consideration only two input models $M_A$, the local design model version, and $M_B$, the last design model release located in the enterprise repository, it also considers $M_P$, the parent of $M_B$. This means that it takes into account not only the differences between the two input design models $M_A$ and $M_B$ to conduct the composition, but also the contrast between them and $M_P$. For example, in Figure 10(A), the developer, Steve, produces a composed model, V3, merging the local version, S3, with its parent, V1, and with the last version of the repository, V2. Note that the more precise the match processes between the $M_P$, $M_A$, and $M_B$, the better the "best-guess" analysis to generate the resulting compositions.

Model composition following this algorithm can be represented as Merge($M_P$, $M_A$, $M_B$), where $M_P$ is the model version from which $M_A$ is descent, MA is the base model, and $M_B$ is the delta model. $M_P$ is used to better track the changes between $M_A$ and $M_B$. For example, revisiting the example in Section 4.2, the decision if the class *A* should be (i.e., *A.isAbstract* = false) or abstract (i.e., *A.isAbstract* = true) may be supported by considering a previous version, $M_P$. This

ancestral version will provide some addition information about how the class was previously. Based on this, developers can make decisions more effectively.

The merging session between $M_P$, $M_A$, and $M_B$ is typically executed as soon as an automated difference analysis between them is done. After identifying the commonalities and differences between the input models, they are merged so that a new release can be produced, $M_{AB}$. This type of composition is applied to collaborative working environment in order to enable more effective team collaboration. It is expected that this effectiveness can be transformed into gain of productivity, and sometimes this is possible because a couple of reasons (Mens, 2002). For example, it requires less user intervention, and in many cases, requires no intervention at all (depending upon the complexity of the composition). Hence, the expectation is that developers' effort invested in parallel increase their productivity proportionally. On the other hand, even though it has reached a high level of precision to compose UML design models, the three-way merge still remains one of the more taxing tasks of any collaborative software development team . This is due to the prior knowledge that developers should accumulate about the initial design model, $M_P$, the current version, $M_A$, and the intended changes, $M_B$.



### 4.2.2. Composition Scenario

After describing the main concepts used in our study, we describe the context where our study was carried out. In the absence of a theory about model composition (Sjøberg et al., 2008), this description is used as a frame of reference (Runeson & Höst, 2009) for our study. The goal is to illustrate the real-world settings in which the case studied happened. To this end, a motivating composition scenario is presented to carefully highlight the problems faced.

#### 4.2.2.1. Collaborative Model Evolution

Figure 9 represents an ever-present collaborative software modeling scenario in our study. We explain three points about this scenario. First, developers work in parallel to increase productivity. They take part of the system

functionalities represented in use cases, and then create UML classes, and sequence diagrams from them. The system functionalities described in these use cases overlap with each other; hence, the design models become to have some *critical overlapping points*. That is, diagrams that share model elements. This is a critical because if a model element is inconsistent, then all diagrams are affected. These points are a source of inconsistency propagation and developers are unable to trace the side effects of all propagations. For example, Peter, Steve and Bill produce UML class diagrams, named P1, S2 and B3, related to the first, second and third use case specification, respectively. However, it is by no means obvious (if not impossible) for the developers to foresee these overlapping points, detect the possible conflicts, and measure their consequences *at modeling time*. Steve cannot predict that changes performed in his model, S2, may give rise to conflicting changes into the Peter's model, P1, and Bill's model, B3. Similarly, it is an effort-consuming task for Peter to identify and grasp that conflicting changes between his model and the Steve's model may propagate into the Bill's model, B3, given the problem at hand. Consequently, the developers inevitably end up creating inconsistent models, since they are unable to effectively deal with a set of conflicting changes.

Second, to overcome this problem, the developers need to invest effort to localize and resolve the inconsistencies. For this, developers must understand the system functionalities and the reasons why the changes happened. For example, Steve would need to understand the semantics of the system functionalities described in the first and third use case specifications. This understanding is required to properly identify and resolve all composition inconsistencies present in his design models (S2). Finally, given the inherent complexity of composing design models it is particularly challenging for developers to: (1) objectively localize these critical overlapping points, (2) quantify the effort variables (*f*, *diff*, and *g*), (3) overcome the emerging inconsistencies, and (4) grasp which influential factors affects the effort variables.

### 4.2.2.2. Motivating Example

Given the need to evolve enterprise design models (e.g., UML class diagrams) and the time constraint (only three days), three developers (Peter, Steve, and Bill) work concurrently to increase the productivity. Firstly, developers check out the last version of the design model (V1) from the repository (Figure 10(A)). V1 is the base model represented in Figure 11(A). After that, they perform a set of modifications over their local versions (i.e., P1, S1, and V1) to evolve them. Figure 10(B) shows a timeline of the modifications and Figure 11(B) represents the delta model that brings together the changes. The developers perform four types of modifications:

(1) Add the stereotype <<MainClass>> to indicate that a class starts up a use case.

(2) Modify the color of a class from white to gray (and vice-versa) to indicate that is part of a framework (or not).

(3) Add the stereotypes <<use>> and <<instance>> to relationships to indicate that a class use and instantiate the other one, respectively.

(4) Add methods to represent that a class implements a new (part of) functionality.

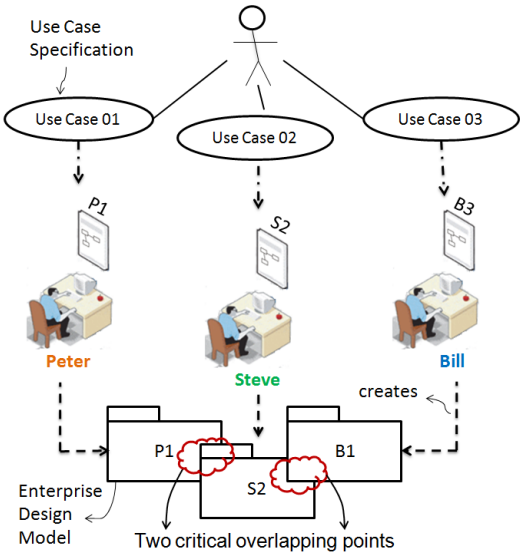



Figure 9: A real-world collaborative model composition leading to two critical overlapping points

(5) Delete some model element.

However, some composition conflicts between the V2 and S3 emerge when Steve submits its last local version, S3, to the repository. This composition session can be briefly represented by Merge(V1,V2,S3). These conflicting changes between the Peter and Steve versions are described as follows:

1) Peter sets correctly the color of the class *ApplicationType* to *gray* (step 1), while Steve sets the color to *white* (step 2).

2) Peter sets incorrectly the color of the class *Application* to *white* (step 2), while Steve updates the color of it to *gray* (step 3).

3) Peter adds the stereotype <<use>> to the relationship between the class *MarlimCore* and *EditPSDiagOptionsAction*, while Steve removes this relationship.

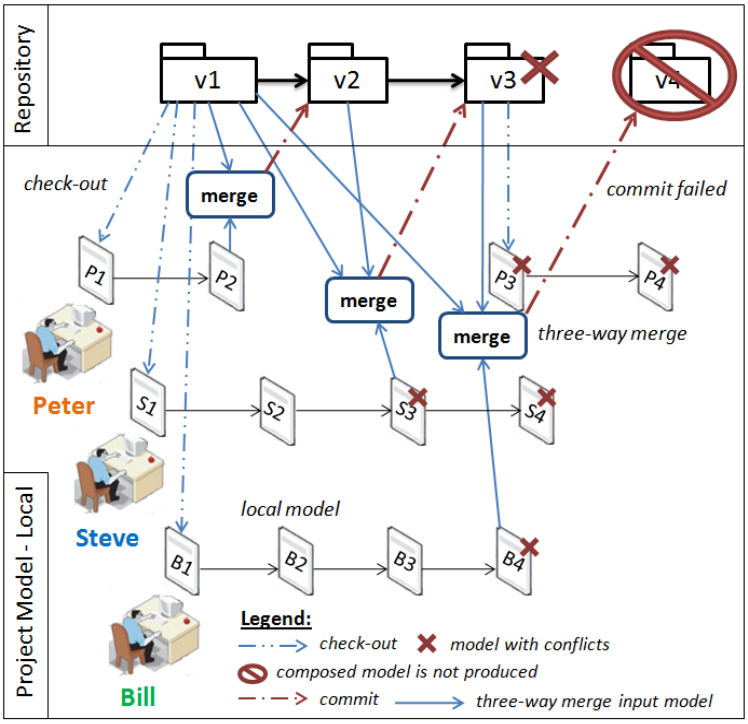
4) Peter removes the class *PSElementGroup*, while Steve creates an inheritance relationship between the class *PSElementGroup* and Production.

5) Peter creates a relationship of association between *PSDiagramOptionsDialog* and *MarlimInputData*, while Peter removes the attribute status: *StatusPanel* from the class *PSDiagramOptionsDialog* and transform it into a new class, and creates a relationship of aggregation between the new class *StatusPanel* and *MarlimInputData*.

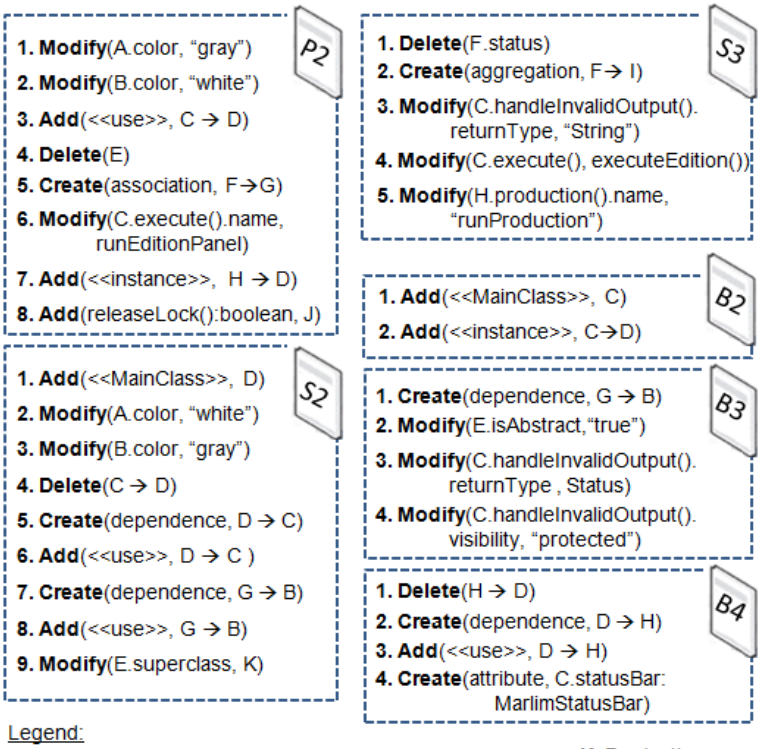
6) Peter modifies the method *execute():void* to *runEditionPanel*, while Steve modifies the method's name to *executeEdition*().

To submit his changes, Steve should know to deal with these contradicting modifications so that the new model version, V3, can be produced. The problem is that, in general, the developers are not always able to understand the emerging conflicts or properly solve them. As a consequence, they realize (or let pass) some incoherent modifications over the input models.

To illustrate these incoherent actions, let us regard the conflicting change number one. If Steve does not accept Peter's changes, then the output composed model is going to have an unexpected change. That is, the class *AppliactionType* of the enterprise framework will have erroneously the color *white* instead of *gray*.




(A)




(B)

Figure 10: A real-world use scenario of model composition (A). The change descriptions performed by the developers (B).

Another example would be the conflicting change five. Peter and Steve propose two ambiguous modifications to allow the class *PSDiagramOptionsDialog* to access objects of the *MarlimInputData*. However, usually these ambiguities are neither properly localized nor understood. This leads the output composed model to have both changes. The result is, therefore, an output composed model with inconsistencies, which is produced from the local project to the enterprise repository V3. Even though, these inconsistencies are usually propagated downward to the developers' local projects. Peter's P3 and P4 local version in Figure 10(A), and the Steve's S4 local version represent this propagation. Bill follows the same submission procedures performed by Peter and Steve; then, he produces the composition session (see Figure 10(A)) represented briefly by Merge(V1,V3, B4) (see Figure 12). The problem is that, in this case, the output composed model, V4, could not be generated. The chief reasons were: the size of the delta model, once Peter's and Steve's changes are also considered during the composition session; and the amount and complexity of the conflicting changes that should be analyzed, since to produce V4 correctly, many semantic and syntactical issues need to be considered. That is, Bill inevitably needs to grasp the meanings of each modification accomplished previously by Peter and Steve. Even worse, this understanding cannot be always acquired. This problematic evolution scenario is described as follows:

1) Bill assigns correctly the stereotype <<MainClass>> to the class *MarlimCore* (B2.step 1), while Peter attaches this stereotype to the class *EditPSDiagOptionsAction* (step 1).

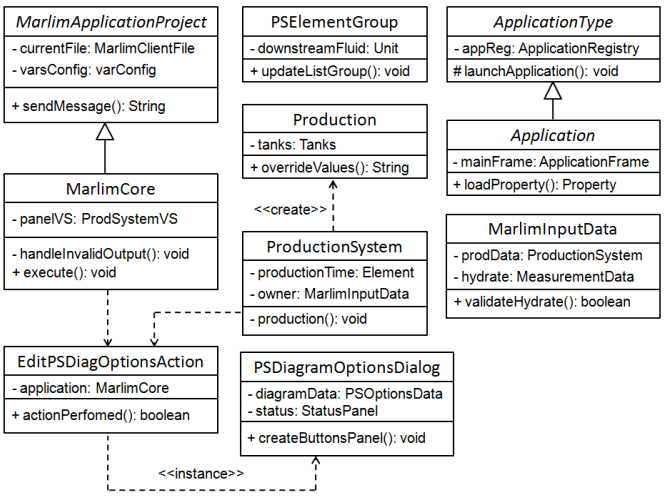
2) Bill attaches the stereotype <<instance>> to the dependence relationship (B2.step 2), while Peter attaches the stereotype <<use>> to this relationship (step 3) and Steve deletes this relationship (S2.step 4).

3) Bill just creates the dependence relationship between the class *MarlimCore* and *EditPSDiagOptionsAction* (B3.step 1), while Steve correctly creates this relationship and attaches it to the stereotype <<use>> (S2.steps 7 and 8).

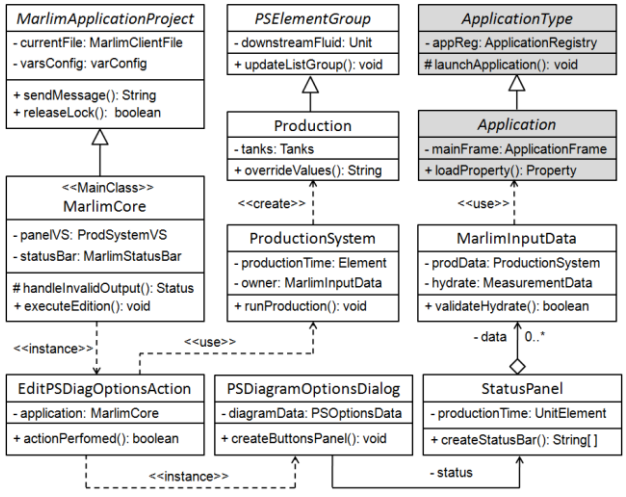
4) Bill correctly transforms the concrete class *PSElemenGroup* to an abstract class (B3.step 3), while Peter removes this class (P2.step 4) and Steve creates an inheritance relationship between the classes *PSElemenGroup* and *Production*. This implies that if the change of Bill is accepted, then the

change of Steve should be rethought, otherwise we will have a syntactically incorrect inheritance relationship between the now abstract class *PSElemenGroup* and the concrete class *Production.*

5) Bill modifies correctly the return type of the method *MarlimCore.handleInvalidOutput()* from *void* to *Status* (B3.step 4), while Steve modifies it wrongly to *String*.




(A)




(B)

Figure 11: The Base Model (A) and the Intended model (B)

6) Peter attaches the stereotype <<instance>> to the dependence relationship between the classes *ProductionSystem* and *EditPSDiagOptionsAction* (P2.step 7), while Bill removes this relationship improperly (B4.step 1) (see Figure 13).

To resolve properly such conflicts, sometimes the developers must engage to seek solutions for conflicts that come from different sources. For example, the

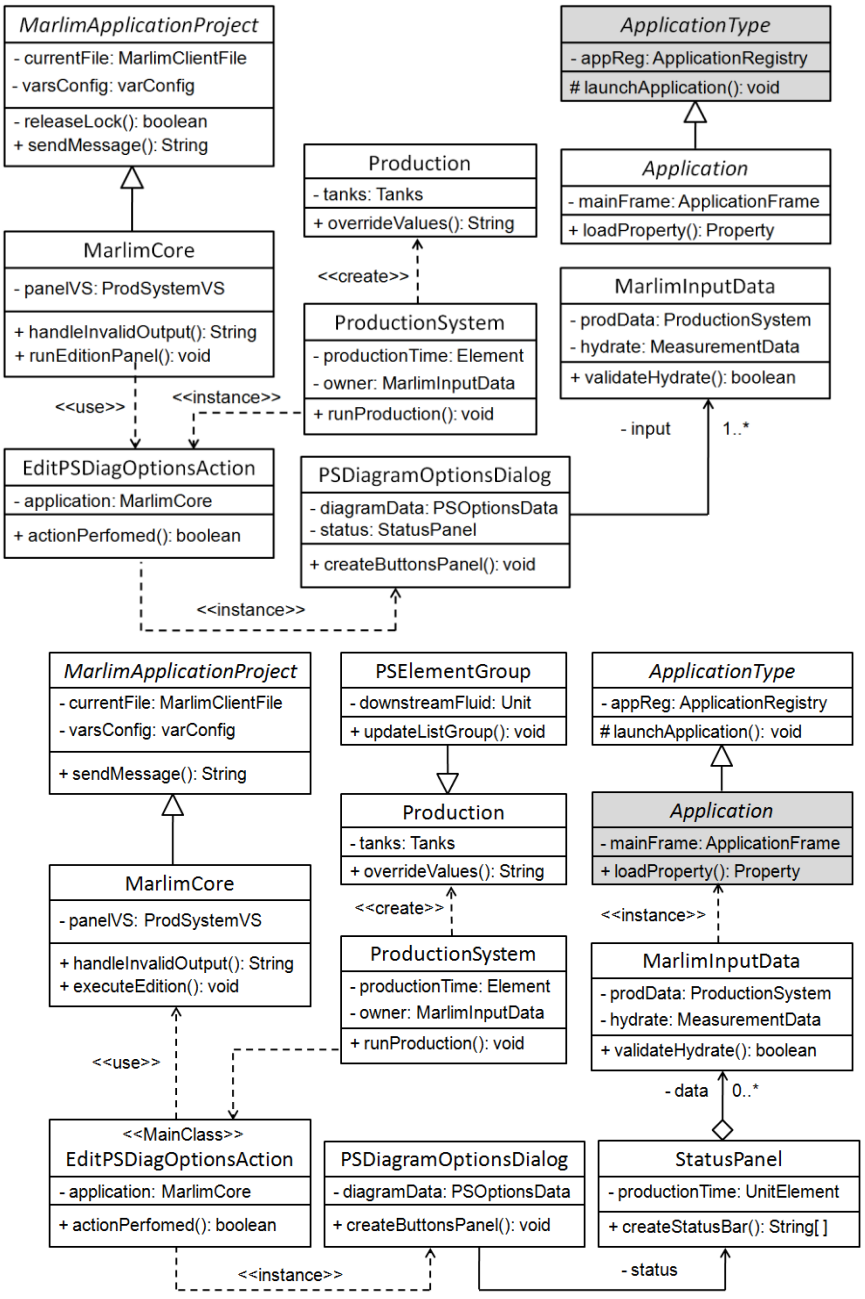



Figure 12: The model versions created by Peter (P2) (above) and Steve (S3) (below).

resolution of the second conflicting changes requires handling systematically the contradicting modifications created by not just one developer (Peter's changes), but by two developers (Peters' and Steve's changes). Moreover, this manipulation must necessarily involve the three developers so that semantic and syntactical issues can be carefully understood.

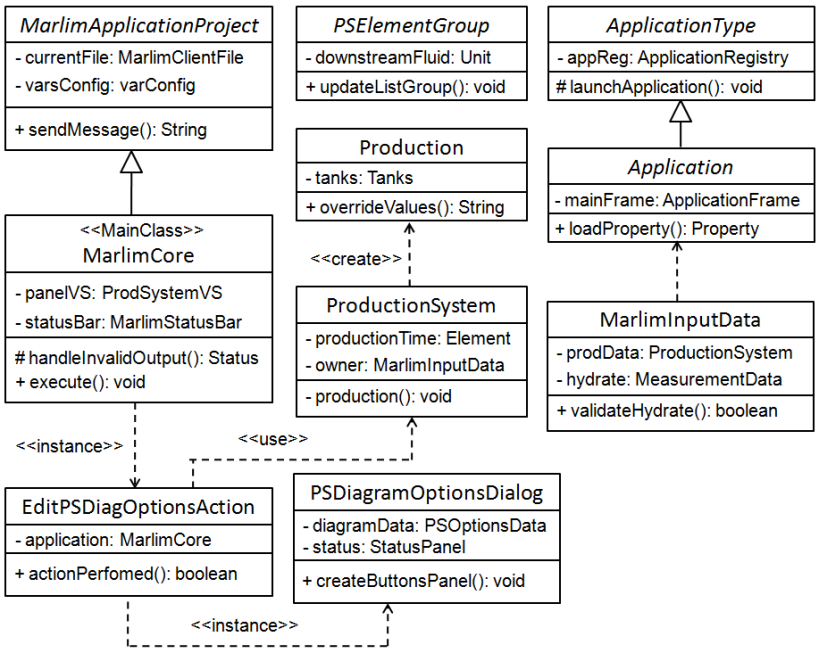



Figure 13: The model versions created by Bill (B4).

### 4.2.3. Study Methodology

This section presents the study methodology based on practical guidelines of empirical studies (Runeson & M. Höst, 2009; Wohlin et la., 2000; Kitchenham et al., 2008).

#### 4.2.3.1. Objective and Research Questions

This study aims at evaluating the effects of model composition techniques on six quality notions, namely syntactic, semantic, effort, application, detection, and resolution ones. In particular, this Chapter focuses on generating practical knowledge about the values that the composition effort's variables assume in real-

world settings. To this end, the research question (RQ2) defined in Section 1.3 is evaluated in this second study. As these variables may be affected by some influential factors, this work also attempts to understand and characterize these factors. With this in mind, we formulate two research questions:

- **RQ2.3:** What is the *effort* to compose design models?
- **RQ2.4:** What are the factors that affect composition effort?

#### 4.2.3.2. Context and Case Studies

We performed five case studies to investigate RQ2.3 and RQ2.4 The context of the studies was collaborative modeling in industrial projects. Developers used model composition to evolve and reconcile design models. Table 18 presents a suite of metrics to characterize the models involved in the studies. Table 19 shows the collected measures for these metrics. As previously mentioned, during 56 weeks, 297 evolution scenarios were performed leading to 2.288.393 compositions between modules, classes, interfaces, and relationships.

All five cases differ in terms of their size, number of participants, and application domain. These cases are characterized as *holistic case studies* (Runeson & M. Höst, 2009; Wohlin et la., 2000; Kitchenham et al., 2008), where contemporary phenomena of model composition are studied as a whole in their real-life context. We present a brief description of the systems used as follows:

- *Alope***:** a system that controls and manages the import and export of Petroleum (and its derived products).
- *Bandeira*: a logistics system is responsible for the complement management of the flow of goods.
- *GeoRisco*: a system that supports forecast and controls of environmental catastrophes.
- *Marlim*: a system that simulates the design and extraction of Petroleum from deep ocean areas.
- *PlanRef*: a system that provides decision making support for logistics and planning processes in Petroleum refineries.

| Type | Metric | Description |
|---|---|---|
| Size | NumClass | #classes |
| | NumAttr | #attributes |
| | NumOps | #operations |
| | NumInter | #interfaces |
| Inheritance | DIT | the sum of depth of the class in the inheritance hierarchy. |
| | OpsInh | #inherited operations. |
| | AttrInh | #inherited attributes. |
| Coupling | Dep_Out | #dependencies where the package is the client. |
| | Dep_In | #dependencies where the package is the supplier. |
| | NumPack | #packages |
| | R | #relationships between classes and interfaces. |
| | H | relational cohesion |
| | Ca | #afferent coupling of the packages |
| | Ce | #efferent coupling of the packages |
| | A | # abstractness (or generality) of the packages. |
| Project | NumWeeks | # weeks |
| | NumDev | # developers |
| | NumEvol | # evolutions scenarios |

#: the number or degree of all

Table 18: Metrics used

| Metrics | Alope | Bandeira | GeoRisco | Marlim | PlanRef |
|---|---|---|---|---|---|
| NumClass | 316 | 892 | 1394 | 2828 | 1173 |
| NumAttr | 1732 | 3349 | 8424 | 9689 | 3808 |
| NumOps | 3479 | 7590 | 10608 | 23722 | 9111 |
| NumInter | 18 | 83 | 143 | 223 | 93 |
| DIT | 140 | 216 | 1109 | 2528 | 871 |
| OpsInh | 3414 | 6620 | 12482 | 38181 | 16369 |
| AttrInh | 1507 | 1766 | 9003 | 9147 | 4406 |
| Dep_Out | 72 | 464 | 61 | 453 | 330 |
| Dep_In | 65 | 423 | 58 | 418 | 322 |
| NumPack | 34 | 166 | 175 | 345 | 187 |
| R | 1285 | 1360 | 3008 | 4493 | 2251 |
| H | 47.5 | 216.8 | 261.9 | 448.6 | 282.5 |
| Ca | 278 | 1147 | 1632 | 4044 | 2329 |
| Ce | 235 | 996 | 1278 | 2723 | 1451 |
| A | 9.58 | 50.45 | 36.9 | 66.5 | 51.9 |
| NumWeeks | 6 | 15 | 8 | 17 | 10 |
| NumDev | 3 | 7 | 2 | 7 | 4 |
| NumEvol | 6 | 95 | 55 | 64 | 77 |

Table 19: The collected measures of the case studies

These systems are featured as scientific software (Kelly, 2006) because they require knowledge from multiple application domains, and encompass a broad class of concepts of physical phenomena, including oil pressure, fluid density, logistic, temperature scale, dilatation of fluids, temperature, fluid pressure, geologic risk, and supply chain. They were chosen based on some reasons presented in the following. First, the cases used robust modeling tool (IBM Rational Software Architect) allowing developers to merge design models, work in parallel, and validate the design models. The IBM RSA was used due to: (1) the implementation robustness of its composition algorithms; (2) the tight integration with the Eclipse IDE; and (3) the tool had been already adopted in previous successful projects. In addition, we also required the UML CASE tools to have an XMI export facility, which will allow us to analyze the design models using metrics tool. Additionally, all cases used a bug tracking system, i.e., JIRA, with which it was possible to coordinate the developers' tasks, specifically during the creation of the design models and review of the models.

Finally, on average, four professional developers have participated in each case study, totaling more than 10 developers in all case studies. The advantage of using experienced professional developers is to avoid one of the main criticisms of most case studies in software engineering, in especial software modeling, regarding the degree of realism of the studies. Thus, we believe that the collected data are representative of developers with industrial skills.



#### 4.2.3.3. Subjects

The background of the subjects was an ever-present concern in the experimental design. As the case studies were performed in vivo in a Brazilian company, the subject selection was based on *convenience* (Wohlin et al., 2000). In total, 12 subjects were recruited. Table 20 describes the subjects' background. We analyzed the level of theoretical knowledge and practical experience of these subjects.

Regarding the *theoretical knowledge* issues, we checked the quality of the education system that the subjects come from. We observed that this system, where the subjects were students, is a system that places a high value on

| Variables | Mean | SD | Min | 25th | Med | 75th | Max |
|---|---|---|---|---|---|---|---|
| Age | 25.3 | 4.47 | 21 | 22 | 24.5 | 27 | 38 |
| Degree | 2.16 | 1.06 | 1 | 1 | 2 | 3 | 4 |
| Graduation year | 2006.4 | 4.8 | 1992 | 2005.25 | 2006.5 | 2010 | 2010 |
| Years of study at university | 5.75 | 2.8 | 3 | 3 | 5 | 7.5 | 12 |
| YOEW UML | 1 | 1.4 | 1 | 1.25 | 3 | 4.75 | 5 |
| YOEW Java | 4.5 | 1.84 | 2 | 2.5 | 4 | 6.75 | 7 |
| Used IBM RSA (1 or 0) | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| YOEW soft. development | 5 | 3.6 | 2 | 2.25 | 4.5 | 5.75 | 16 |
| Hours of software modeling | 98.33 | 40.38 | 60 | 60 | 90 | 120 | 180 |
| Hours of OO programming | 156.66 | 89 | 80 | 80 | 130 | 225 | 360 |
| Hours of software design | 130 | 53.85 | 80 | 80 | 120 | 190 | 220 |

*Degree: 1 = Student, 2 = Bachelors, 3 = Masters, 4 = Ph.D.*
*YOEW = Year of Experience with, Med: Median*
*SD = Standard Deviation, $25^{th}$ = lower quartile, $75^{th}$ = upper quartile*

Table 20: Descriptive Statistics: Subjects' Background

theoretical issues about the foundational principles of software engineering and software modeling. Moreover, this educational system provides an academic formation with much more than 120 hours of courses (lecture and laboratory) exclusively dedicated to software engineering, object-oriented programming, and software modeling. This can be seen, in part, as an intensive UML-specific training. Furthermore, other important courses present in their formation are operating systems, databases, computer architecture, requirement engineering, and so on. Therefore, the subjects fulfilled the level of theoretical knowledge required.

Taking into consideration the *practical experience* of the subjects, we also observed that there are some even more compelling evidences about the level of practical experience of them. This knowledge was acquired from previous software development projects. This was confirmed by the analysis in which provides background data on the subjects that participated in the case studies. The data show that the subjects fulfill the requirements in terms of age, education, and experience. A benefit of the presence of a considerable theoretical and practical knowledge is that the members of the teamwork can learn from each other in terms of theoretical and practical issues. The main consequence of this knowledge

sharing between team members is that the emerging problems can be solved more quickly and properly. If, for example, well-formedness rules of the design models are challenged, the subjects can work together to get it solved. Another point that is essential to emphasize is that, in all cases, the subjects were familiar with the software modeling tool they had to use, IBM RSA, and all subjects received training about merging design models. Lastly, based on this information (summarized in Table 20), we deemed that the subjects had the required training, theoretical knowledge and practical experience to perform the software modeling and merging tasks properly.

#### 4.2.3.4. Study Design and Evaluation Procedures

Having presented the context of our studies and subjects, the next step is to describe precisely how the case studies were conducted.

##### 4.2.3.4.1. Operation

The procedures of the study can be grouped into two phases: *creation* and *review*. In the first phase, the developers collaboratively created the design models. In the latter, they detected and resolved inconsistencies in the output composed models. Note that the intended model was produced after executing these two phases. Moreover, it is also important to emphasize that the effort variables (*f*, *diff*, and *g*) are incrementally measured as the phases are performed.

Figure 14 summarizes the procedures associated with both the production of the intended models and the measurement of the effort variables. Activities are represented using rounded rectangles, and the arrows indicate transitions between the activities. The diamonds are decisions (conditional branch), and the arrows connected to them are marked with the conditions. The initial state in an activity diagram is indicated by the black circle, while the final state is the encircled black circle. Following the simplest path of the procedure, issues are first submitted and examined (issue refers to general activities registered during the modeling project). Each issue is assigned to a developer. After opening the issue, the developer may execute three possible activities: creation of the design model, detection of inconsistencies, and resolution of inconsistencies. As these activities

were carried out, the effort variables were quantified. Developers closed the issue after it has been validated.

*Creation of the Design Models.* First, the developers created a UML class diagram for each use case specification. In addition, sequence diagrams were created for the most important use cases, which represent around 30 percent of the full system specification. This percentage and the choice of the use cases were not made in an arbitrary manner, but based on the policies of the company. After that, the developers made use of the model composition technique to submit the created model to the repository. It is important to emphasize that developers created sequence diagrams only after its corresponding class diagram had been created and validated. To calculate the developer's effort to compose the local model with the repository version, the members of the team were stimulated to make a record of all composition sessions by using the software Camtasia Studio Pro (Camtasia, 2011). The generated videos were essentials to further analyses.

*Detection of Inconsistencies.* The developers reviewed the composed models in order to detect syntactic and semantic inconsistencies. For this, they performed a double checked model reviews by using the IBM RSA's model validation mechanism and by manually inspecting the models. During each review, the developers could read the use case specifications to check whether (or not) the generated models fulfill the requirements described in the specification. It is important to point out that a developer reviewed the models created by other developers, never the model created by him. Since the IBM RSA's validation mechanism can report false positive and false negative inconsistencies (Altmanninger et al., 2009), the teamwork members were encouraged to check if the reported inconsistencies were posing, in fact, a problem.

*Inconsistency Resolution.* Having identified the inconsistencies, the developers invested some effort to revolve them. In practical terms, they added, removed, or modified some existing model elements to solve them. After addressing the model inconsistencies, the developers submitted the intended model to the repository. Thus, the compositions were executed in two moments: after the original creation of the models and after the inconsistency resolutions. All model versions were registered in a version controlling control system, thereby allowing a systematic analysis of the history of the generated model versions.

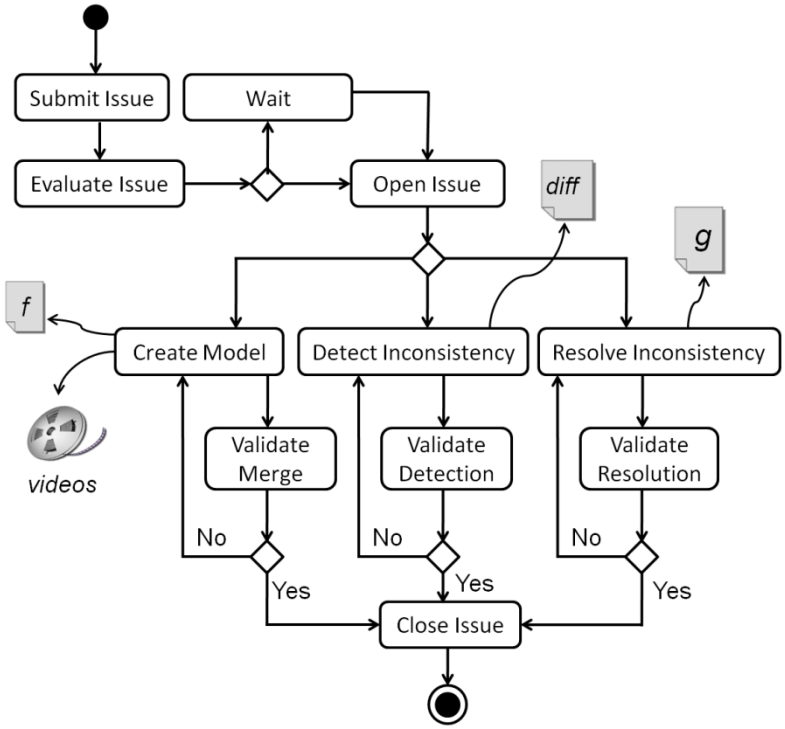



Figure 14: The flow of activities during the studies



#### 4.2.3.4.2. Design Model Versions and Releases

The design models are semantically rich, have been evolving over the long term, can be checked for consistency. These features were carefully analyzed and elected as pre-requirements to perform the case studies. We feel, therefore, confident that the model releases are going to promote (1) more reliability and accuracy of our results, and (2) chiefly suitable conditions for yielding lessons about driving composition effort variables. Consequently, this enables us to grasp as the composition effort variables (*f*, *diff* and *g*) turn up in real-world settings, and identify and understand the factors that affect the production of the desired releases during the composition session.

*Deriving the Design Model Releases.* Given the collaborative environment work, the subjects incrementally created the releases using the IBM RSA's composition technique throughout the evolution scenarios. The creation steps are presented as follows. First, from a reverse engineering process, the team leader generates a set of elementary model elements, which will be used by other

developers to create the design models. Note that this derivation of the model elements is indispensable in real-world settings; since the size of systems is considerably large (see Table 20).

Next, the developers make use of these elements to manually generate the design models. For example, the developers define which model elements should be inserted into the UML class diagram and what their relationships are. This decision is made from the information collected from the use case specification and the code. This creation process of the models is not only marked by intensive discussion among the members of the development team, but also by the constant submission of new model release increments to the repository so that the changes can be broadcasted to the other developers. To control the changes of the models and to facilitate collaboration, the version control system was intensively used during all case studies.

*Model Releases and Composition Specification.* For each evolution scenario, a new release was created. For each new release, the previous release was modified in order to incrementally accommodate the changes. To implement a new evolution scenario, a model composition specification can remove, add, derive, or modify the entities present in the previous release. During the design of all releases, a main concern was to follow the best practices of modeling and carefully realize the requirements described in the use case specifications.



#### 4.2.3.4.3. Variables and Quantification Method

This section defines as the three effort variables (*f*, *diff*, and *g*) were quantified and their unit of measurement (time in minutes). Our analysis and quantification, therefore, rely on three effort measures described as follows.

*Application Effort Measure (f).* This measure represents the required time (in minutes) to match the input model element, resolve the conflicting changes, and submit the evolving changes to the repository. That is, the effort invested by developers to apply the model composition technique. This measurement only quantifies the effort to produce the composed model ($f(M_A,M_B)$) rather than the effort to detect ($diff(M_{CM},M_{AB})$) and resolve inconsistencies ($g(M_{CM})$). This effort was calculated from recorded movies created by own developers, which were

stimulated to record these videos throughout the case studies.

*Detection Effort Measure (diff).* The detection effort consists of the time needed to localize inconsistencies in the composed model for a given output composed model. Subjects were responsible for registering the time. This detection can be characterized as a semi-automated process; as developers make use of the IBM RSA's model validation mechanisms and manually go through the model to identify semantic problems. We consider all syntactic inconsistencies can be automatically detected. On the other hand, given that it is impossible to count all semantics inconsistencies automatically, we count only semantic inconsistencies that can be manually spotted. For example, relationships (e.g., association and inheritance) between model elements that no longer exist or a stereotype attached improperly. Usually these inconsistencies are not detected by tools upfront, but are visually by developers.

*Resolution Effort Measure (g).* It represents the time required to perform a set of activities (creations, removals, and modifications) needed to transform $M_{CM}$ into $M_{AB}$. Again, subjects were the responsible for registering the time.



#### 4.2.3.4.4. Analysis Procedures

The analysis of the collected data was conducted with quantitative and qualitative methods. While the quantitative data concerns the measurements involving the study variables, objects, and units of the analysis, the qualitative data deals with the diagrams (pictures), descriptions, transcripts from interviews, and annotations. The goal of using a combination of qualitative and quantitative data is to exclusively provide a better understanding of the studied phenomena in their context.

a. *Quantitative Analysis*

The descriptive statistic is used so that the outstanding trends might be pinpointed. Box-plot graphically illustrates these trends. The presence of patterns in the data distribution, and lack thereof acted as a driver for further investigation allowing a deeper understanding. Note that we are not concerned with any correlation analysis or probabilistic formulation. Rather, our focus is only to

describe and graphically present interesting aspects of the data. Further, these statistics were important to analyze and possibly remove outliers from the data. Outliers are extreme values of the measured variables that may influence the study's conclusions. To analyze the outliers we made use of box-plot. According to Wohlin (Wohlin et al, 2000), we should verify whether "the outliers are caused by extraordinary exceptions (unlikely to happen again), or whether their cause can be expected to happen again. For the first case, we should remove the outliers, and for the latter we should not remove the outliers." In our study, some outliers were identified. However, they did not represent any extraordinary exceptions, since they were expected to happen again. Consequently, they were not removed, as they did not compromise the results.

### b. *Qualitative Analysis*

The qualitative analyses were concentrated on interviews, observational study, and archival data. Hence, the RQs were investigated from different viewpoints, subjects, artifacts, and projects.

*Interviews.* A *semi-structured* interview approach was performed following a *funnel model* (Runeson & Host, 2009), 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 (Runeson & Host, 2009). They were organized in such a way that research questions (*f*, *diff*, and *g*) could be exploited. An interview guide was created based on the authors' experience in model composition and on previous studies, together with the research questions of the study. The author of this thesis conducted the interviews. The interviews were recorded and transcribed into text; this was done by one else than the authors. Experienced subjects were selected for the interviews from the involved company and other Brazilian companies. That is, the interviewees (8) were not only developers that participated in the case studies, but also with other developers with different experiences of other companies. The selection was based on the interviewees' different experience in terms of model composition rather than their similarities. It was also 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 model composition was performed in practice extensive observations were conducted through three

different approaches. First, one of the authors worked in the modeling projects during the case studies taking part in everyday activities. This allowed a more effective observation. Secondly, the model composition tasks were recorded, and after analyzed. This allowed monitoring the task of the subjects. Thirdly, to obtain a feedback of the subjects about the task performed, they encouraged to "think aloud" by asking questions like "What is the key difficult to resolve the inconsistencies?", "What is your strategy to deal with conflicting changes?", and "What do you do to reduce composition effort?". In summary, data collected consisted of field notes, audio recordings of interviews and their transcriptions, videos, screenshots, and copies of artifacts.

*Archival Data.* The company's repository was an important source of data, since it enables us to access the different versions (specifically the evolution track) of the design models. The developers were encouraged to describe the evolution changes performed before executing the compositions. This description helped us to understand how the compositions were performed and reasons why the inconsistencies arose. For example, in the motivating example (Section 4.2.2.2), the developers, Peter, Steve, and Bill, should necessarily describe the changes performed by them. In total, more than 240 descriptions were created and the information stored in the repository. The comments were expressed in a free-text field, in which the subjects could report anything they thought might be relevant in explaining the changes that were being done. In addition, the developers were well aware the importance of these descriptions to understand the evolutions and the results obtained on each evolution scenario. For example, the comments helped us to identify when the composition had success (i.e., $M_{CM} = M_{AB}$) or failed (i.e., $M_{CM} \neq M_{AB}$), and grasp the rational what the developers thought at the time of composition session.

### 4.2.4. Study Results

In this section, we interpret the results about the RQ2.3 and RQ2.4. For this, we present and analyze quantitatively and qualitatively the collected data about the composition effort variables (Section 4.2.4.1) and explains the factors that influence these variables in practice (Section 4.2.4.2).

### 4.2.4.1. RQ2.3: Composition Effort Analysis

The composition effort analysis involves the examination across cases of a single variable, focusing on three characteristics: the distribution, the central tendency, and the dispersion.

**Application Effort (f)**

This section investigates the variable concerning the effort to apply the composition technique. Table 21 shows a descriptive statistic about the application effort. These statistics will help us to pinpoint the central tendency and spread of values around it. A tally of 40 and 69 (N) compositions was registered in the Marlim and Bandeira project, respectively. The central tendency was calculated using the two most-used statistics: the mean and the median. The most interesting feature was that the composition of the large-scale industrial models used in our study required by about 4 minutes.

More specifically, the results indicate that effort to compose models was, on average, 3.17 minutes and 4.43 minutes in Bandeira and Marlim projects, respectively. Given the complexity and the size of the design models in question (Table 19), these central tendency measures are in fact low values. For example, a developer spent just around 4 minutes to submit the most complex evolving changes to the repository in the Marlim project. In addition, the median measures accompany these measures: 3 minutes and 3.12 minutes in the Bandeira and Marlim project, respectively. Thus, this implies that the required effort to apply the semi-automated model composition technique is low. Consequently, it is possible to advocate it as appropriate to collaborative software modeling in which resources and time are usually tight.

| Cases | N | Mean | SD | Min | 25th | Med | 75th | Max |
|---|---|---|---|---|---|---|---|---|
| Marlim | 40 | 4.73 | 4.52 | 0.25 | 2 | 3.2 | 6.79 | 22 |
| Bandeira | 69 | 3.29 | 1.93 | 0.83 | 2 | 3 | 4 | 14.25 |

*N = number of compositions, SD = standard deviation, Min = minimum, 25th = first quartile; Med = median, 75th: third quartile, Max: maximum.*

Table 21: Descriptive statistics for application effort

To understand the dispersion of the data around this tendency, not only the standard deviation, 25th and 75th percentiles were computed, but also the minimal and maximum values. Developers' effort tends to concentrate by around the central tendency rather than spreading out over a large range of values. Indeed, with 1.55 and 1.58 minutes, the standard deviation measures indicate that in the majority of the composition sessions the developers spend an effort near 3.17 minutes or 4.43 minutes. This information can help modeling mangers to: (1) systematically propose the effort estimation rather than essentially based on their judgment; and (2) check if the effort spent by developers is an expected value (or not), since it falls inside (or outside), these ranges of statistics that is expected to occur. Consequently, it is possible to improve the effort estimation, and hence a typical UML-based development, for example. Finally, this measure can be seen as the first step to overcome the lack of empirical evidence about the impact of model composition techniques on developers' effort in real-world settings.

To deepen our understanding about the application effort, Figure 15 distributes the collected sample in six effort ranges. These ranges in the histogram systematically group the application effort cases. The y-axis of the histogram represents the counts of merging, while the x-axis consists of the ranges of effort. The main outstanding feature is that: the presence of a distribution pattern of the application effort through the ranges of effort. The low-effort categories (i.e., $t < 2$, $2 \leq t < 4$, and $4 \leq t < 6$) represents the most likely range of effort that developers invest to compose the input models. The number of cases is equal to 29 (in Marlim) and 64 (in Bandeira), representing 72.5 percent and 92.75 percent of the composition cases, respectively. On the other hand, the number of cases in the high-effort categories (i.e., $6 \leq t < 8$, $8 \leq t < 10$ and $10 \leq t$) is equal to 12 (in Marlin) and 5 (in Bandeira), comprising 17.39 percent and 12.5 percent of the cases respectively. Thus, the number of composition cases in the low-effort categories outnumbers the amount of cases in the high-effort categories, comprising more than 70 percent and 90 percent of the cases in the Marlim and Bandeira project, respectively. On the other hand, the number of cases in the high-effort categories was by around 30 percent (in Marlin) and 7.25 percent (in Bandeira). In practice, this means that developers spent less than 6 minutes in 85.32 percent of the whole composition cases, and just 14.68 percent of the cases required more than 6 minutes.

Another even more compelling feature is that: there is a changing pattern among the effort categories. Although the changing pattern of the measures from a category to another one happens in different forms, it comes about with the same type of change in the most of the cases.

There are five changes in the number of counts of merging from one category to another being three of them similar as follows. From the first to the second category, the count of compositions had a gradual rise from nine to 13 (in Marlim) and from 10 to 33 (Bandeira). This means a growth of 44 percent and 230 percent, respectively. On the other hand, observing the third category, the count had a significant drop compared to the previous category.

The distribution of merging fell back from 13 to 6 and from 33 to 21 in the Marlim and Bandeira project, respectively. This implies into a significant drop of 53.84 and 36.3 percent. Following this same drop pattern, in the fifth category, the number of cases decreased abruptly from 7 to 1 (Marlim) and 3 to 1 (Bandeira), comprising a fall of 85.71 percent and 66.67 percent, respectively. However, the transitions from the third category to the fourth one as well as from the fifth category to the sixth one had different changing pattern. In the fourth category, the count kept stable (seven cases) in Marlim project and a decrease of 85.71 percent in Bandeira project was observed, from 21 to 3. In the sixth category, the count did not change, stagnating in 1 (Marlim), and, however, quadruplicated its value from 1 to 4 in the Bandeira project. This implies, therefore, that there is to some extent a particular behavior of change between the ranges of effort.

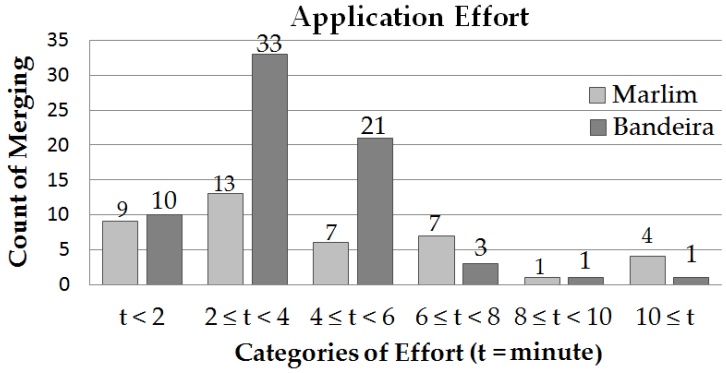



Figure 15: Histogram of the application effort measures

With these two previous features in mind, an important finding was observed: the application effort tends to reduce as developers become more familiar with technical issues rather than application domain issues. This finding is supported by the fact that developers invested more effort in Marlim project than in Bandeira project. After a careful analysis, the main reason was that the developers were more familiar with composition issues. That is, 30 percent of the cases had effort higher than 6 minutes, rather than the 7.24 percent ones in the Bandeira project. It is important to point out that: (1) both projects had a similar level of complexity; (2) the members of the development team had a similar level of knowledge about the meaning of application domain elements; and (3) the teamwork was the same throughout the both projects. Therefore, the application effort tends to decrease as the developers gained experience with the activities considering key steps to apply the composition technique, i.e., match the input models, resolve the conflicting changes, and then combine the input model elements.

**Detection Effort (diff)**

This section investigates the variable concerning the effort to detect the inconsistencies of the output composed model. Table 22 shows a descriptive statistic about the effort spent to detect inconsistencies. A careful analysis indicated that some interesting features were happing. First, the more experienced developers in both modeling and IBM RSA spend 23.2 percent less effort to detect inconsistencies than less experienced developers. This observation was derived from the comparison of the medians in the Marlim and Bandeira cases. This finding was possible to reach because the same development team firstly worked in the Marlim project and after this in the Bandeira. Observing the values of the mean computed this affirmation is still reasserted. In this case, the more experienced developers invested 38.57 percent less effort to detect inconsistency than less experienced developers, compared 7.57 and 4.65.

Second, the higher the number of teamwork members, the higher the effort to localize inconsistencies. This outstanding finding is supported by the comparison of the medians of the projects with high versus low number of developers. Comparing the number of teamwork members of the projects, we could observe that the developers of the Marlim and Bandeira project, both with 7 developers, invested a higher amount of effort to detect inconsistencies than the developers of the GeoRisc and PlanRef (with 2 and 4 developers, respectively). For example, the developers spent 49.46 percent more effort to detect inconsistencies in the Marlim project than in GeoRisc project, compared the medians 6.55 and 3.31, respectively. This striking observation was also reinforced when we compared the Marlim and PlanRef. That is, Marlim's developers spent 64.27 percent more effort to localize the inconsistencies, compared the medians 6.55 and 2.34, respectively. Therefore, the projects with a higher number of developers had to invest the double of effort to localize the inconsistencies.

Third, a remarkable finding is that the higher the number of inconsistencies in behavioral models, the higher the effort to detect inconsistencies. Even though, the Alope project had a low number of developers, a considerable number of inconsistencies were concentrated in behavioral models like sequence diagrams. The chief problem highlighted by developers was that the *behavioral models* require an additional effort to go through the flows of execution. For example, an association in a structural model (e.g., class diagram) represents essentially one relationship between two classes. On the other hand, in a behavioral model (e.g., sequence diagram) that represents the interaction between the instances of these classes; this simple association may be represented by *n* interactions (i.e., messages

| Cases | N | Mean | SD | Min | 25th | Med | 75th | Max |
|---|---|---|---|---|---|---|---|---|
| Marlim | 63 | 7.57 | 5.1 | 0.54 | 2.45 | 6.55 | 12.49 | 16.54 |
| Bandeira | 86 | 4.65 | 2.39 | 0.36 | 2.37 | 5.03 | 6.38 | 9.21 |
| GeoRisc | 24 | 3.66 | 1.52 | 1.32 | 2.67 | 3.31 | 4.16 | 7.39 |
| PlanRef | 44 | 2.91 | 1.75 | 1.04 | 1.39 | 2.34 | 4.12 | 7.15 |
| Alope | 6 | 12.37 | 4.2 | 5.26 | 8.25 | 13.15 | 16.36 | 17.37 |

N = number of compositions, SD = standard deviation, Min = minimum, 25th = first quartile; Med = median, 75th: third quartile, Max: maximum.

Table 22: Descriptive statistics for detection effort

exchanged between the objects). The problem is that developers must check each interaction. This problem is enlarged with the need to check the consistencies between the class diagram and the sequence diagram. For example, there is a message from an object A to an object B in the sequence diagram, but there is no relationship between the class A and B in the class diagram. Even worse, sometimes the method corresponding to such message does not even exist in the class B. Another typical inconsistency is that a concrete class A becomes abstract, however, its instance remains represented in the sequence diagram. Thus, developers had an additional effort to examine the consistency between the structural and behavioral model.

Another observation is that the higher the distribution of inconsistencies in different modules, the higher the effort to identify them. In the case studies, the systems were strongly decomposed in conceptual areas. This unit of modularization brings together application domain concepts in a same space. The problem arises when the inconsistencies in a conceptual area give rise to an abundance of inconsistencies, and hence affecting many other model elements located in other conceptual areas as a ripple effect. This propagation is inevitable as there are usually some relationships between these units of modularization. Hence, developers must be able to identify inconsistencies in model elements of conceptual areas that they do not know. Note that during the case studies the developers created diagrams related to a specific functionality of the system (specified in case uses), and these diagrams were grouped in a conceptual are (something like a package). Thus, the lack of knowledge about the model elements in unknown conceptual area led developers to invest an extra effort to pinpoint the inconsistencies.

**Resolution Effort (g)**

This section investigates the variable concerning the effort to resolve the inconsistencies in the output composed model. Table 23 shows a descriptive statistic of the inconsistency resolution effort. The main outstanding feature is that the developers invest more effort to resolve inconsistencies rather than to both apply the model composition technique and detect the inconsistencies. This can be explained based on some evidences.

First, in Marlim project, for example, the teamwork members spent 64.91 percent more effort resolving inconsistencies than applying the model

composition technique. This difference comprises the comparison between the medians 3.2 (application) and 9.12 (resolution). This difference becomes more explicit when we consider the values of the mean. This evidence is reinforced in Bandeira project. The resolution of inconsistencies consumes 80.31 percent more effort than the application of the composition technique, compared the medians 3.2 (application) and 9.12 (resolution). The difference between the application and resolution effort becomes stronger when we consider the value of the mean i.e., jumping significantly their values from 64.91 percent to 88.40 percent (in Marlim) and from 80.31 percent to 88.35 percent (in Bandeira).

Second, in Marlim project, the inconsistency resolution consumed 28.17 percent more effort than the inconsistency detection. This comprises the difference between the medians 6.55 and 9.12. The results in Bandeira project followed the same trend. Developers spent 66.99 percent more effort with inconsistency resolution than with inconsistency detection, compared the medians 5.03 and 15.24. Considering the mean, this difference of effort becomes more evident, leaping abruptly from 28.17 percent to 81.44 percent (in Marlim) and from 66.99 percent to 83.42 percent (in Bandeira). Analyzing the collected data from the GeoRisc and Alope project, this observation is confirmed. For example, the resolution effort is 82.98 percent and 54.96 percent higher than the detection effort in GeoRisc and Alope, respectively. On the other hand, in Alope project, the resolution and detection effort were practically equal. Therefore, the collected data suggest that teamwork members tend to spend more effort resolving inconsistency rather than applying the model composition technique and detecting inconsistencies.

Another striking feature is that the experience acquired by the developers did not help to minimize the inconsistency resolution effort. Although more experienced developers have invested less effort to compose the input models and detect inconsistencies, their additional experience did not help significantly to minimize the inconsistency resolution effort. For example, in Bandeira project, more experienced developers spent 40.15 percent more effort to resolve inconsistency than less experienced developers from Marlim project, compared the medians 9.12 and 15.24. The main reason is that more experienced developers tend to be more cautious than less experienced ones, and hence they tend to invest more time analyzing the impact of the resolution of each inconsistency.

| Cases | N | Mean | SD | Min | 25th | Med | 75th | Max |
|---|---|---|---|---|---|---|---|---|
| Marlim | 31 | 40.79 | 74.79 | 3.09 | 4.13 | 9.12 | 11.33 | 246.25 |
| Bandeira | 8 | 28.06 | 28.04 | 5.55 | 8.17 | 15.24 | 41.44 | 95.44 |
| GeoRisc | 16 | 25.86 | 13.75 | 5.12 | 17.70 | 19.45 | 42.5 | 53.33 |
| PlanRef | 44 | 2.86 | 1.92 | 1.2 | 2.03 | 2.33 | 2.52 | 10.41 |
| Alope | 5 | 31.04 | 12.75 | 16.21 | 16.21 | 29.20 | 46.8 | 55.4 |

*N = number of compositions, SD = standard deviation, Min = minimum, 25th = first quartile; Med = median, 75th: third quartile, Max: maximum.*

Table 23: Descriptive statistics for resolution effort

#### 4.2.4.2. RQ2.4: Influential Factors on Composition Effort

Some factors influence the effort of composing large-scale design models in real-world settings. This section analyzes the side effects of these factors on the composition effort variables.

##### 4.2.4.2.1. The Effects of Conflicting Changes

A careful analysis of the results pointed out that the production of the intended model is affected by the presence of different types of change categories in the delta model. These changes would be the addition, removal, modification, and derivation of model elements. The current composition algorithms are not able to effectively accommodate these into a base model; mainly, when these changes occur simultaneously. We described the most common categories of changes identified throughout the study and after analyzing their effects:

- *Addition:* model elements are inserted into base model; for example, a stereotype <<instance>> was added to the directed relationship between the *ProductionSystem* and *EditPSDialogOptionsAction*.
- *Removal*: a model element in the base model is removed; for example, the class *PSElementGroup* is removed;
- *Modification*: a model element has some properties modified; for instance, the class *PSElementGroup* becomes abstract. For this, the property *isAbstract* has its value modified from *false* to *true*.

*Derivation*: model elements are refined to accommodate new changes and/or moved to other ones. For example, the class *ProductionSystem* is refined into two new classes: *ProductionAction* and *ProductionPanel*. The method *ProductionSystem.runProduction()* is inserted into *ProductionAction.* The attribute *ProductionSystem.productionTime* is inserted into *ProductionPanel*. This type of modification can be seen as a 1:N modification.

Developers and researchers recognize that evaluable software should adhere to the Open-Closed principle (Meyer, 1997) as evolutions become easier. This principle states "software should be open for extensions, but closed for modifications." However, this observation did not occur in all the cases as modifications and derivations of model elements happened as well. In our study, the open-closed principle was more closely adhered by the evolutions dominated by additions rather than any other one. In this case, developers invested low effort compared to other cases. This suggests that the closer to the Open-Closed principle the change is, the lower the composition effort.

On the other hand, evolution scenarios that do not follow the Open-Closed principle required more effort to produce the intended model, $M_{AB}$. This finding was identified when the change categories simultaneously occur in the delta model; hence, compromising the composition for some extent. This extra effort was due to the incapability of the matching algorithm to identify the similarities between the input model elements given the presence of widely scoped changes. In the Marlim project, for example, the composition techniques were not able to execute the compositions by about 17 percent (11/64) of the evolution scenarios. This required developers to recreate the models manually. In the Bandeira project, by about 10 percent (10/95) of the composition cases did not produce an output model as well, or the composed model produced had to be thrown away due to the high amount of inconsistencies.

In particular, we also observed that *the refinement (1:N) of model elements in the delta model caused severe problems*. A practical example of this refinement encompassed the direct relationship between *PSDiagramOptionsDialog* and *MarlimInputData*, named as *input*. This relationship was decomposed into (1) a direct relationship between *PSDiagramOptionsDialog* and *StatusPanel*, (2) the class *StatusPanel*; and (3) the aggregation between *StatusPanel* and *MarlimInputData*. In this case, the relationship (1:3) was not identified. This

problematic scenario was also noticed during the refinement of some classes belonging to the MVC (Model-View-Controller) architecture style into a set of more specialized ones. In both cases, the name-based, structural model comparison was unable to recognize the 1:N composition relations between the input model elements. However, we have observed these conflicts do not only happen when developers perform modifications, removals, or refinements in parallel, but also when developers insert new model elements. This finding was noted from the fact that although evolutions following the Open-Closed principle had reduced the developers' effort, they still caused too frequent undetected inconsistencies.

Developers were often unable to localize inconsistencies that did not affect the model elements created by them. Even worse, the composition algorithms were unable to identify that overlapping changes might cause "cross-semantic inconsistency." That is, the semantic attributed to a model element conflict with another one assigned to the same (other) element. A very concrete example of semantic inconsistencies in our case studies was when UML stereotypes used to attribute new semantic to the model elements conflict with each other. The illustrative example shows two typical inconsistencies in our studies. For example, Steve attaches the stereotype <<MainClass>> to the class *EditPSDiagOptionAction*, while Bill attaches this attribute to *MarlimCore*. Hence, the algorithm does not detect that only one class can be defined as the main class.

We have noted that *these problems are more challenging to be detected when they occur in multi-valued properties defined in the UML metamodel* such as *Class.ownedOperation: Operation [*]*, which defines the methods of a class, or *Class.extension: Extension [*]*, which specifies the stereotypes applied to a class. For example, Bill attaches the stereotype <<instance>> to the directed relationship (B2.step 2) from *MarlimCore* to *EditPSDiagOptionsAction*, while Peter attaches the stereotype <<use>> to this relationship (P2.step 3). As these stereotypes are not present in ancestor version (*V1*), the algorithm incorrectly brings both to the new version (*V4*). One of the reasons for this is that the meaning of the stereotypes are often not taken into account during compositions—either because the semantics of these stereotypes are rarely represented or either because the composition algorithms are unable to infer that the stereotypes <<instance>> and <<use>> are semantically contradicting. However, developers must tame this

problem.

Still considering the conflicting changes between Bill and Peter, whatever the change accepted — if the class *PSElemenGroup* is transformed into an abstract class, or if it is removed — inconsistencies will emerge when the Steve's changes are applied to *PSElemenGroup*. For example, Steve creates an inheritance relationship between the classes *PSElemenGroup* and *Production* (a concrete class). If the class *PSElemenGroup* is abstract, then a semantic inconsistency emerges because *PSElemenGroup* has an inheritance relationship with a concrete class *Production*. Note that this inconsistency is not related to the modeling language as the UML metamodel hinder inheritance relationship from the abstract class to concrete one. This inconsistency is because object-oriented programming like Java does not permit this type of relationships. On the other hand, if the class *PSElemenGroup* is removed, then a static semantic inconsistency arises because the inheritance relationship refers to a class that no longer exists.

Thus, we have observed that the current state-of-the practice composition techniques superficially support the evolution categories. For accuracy reasons, this implies that developers need innovative techniques supporting restructuring changes and identifying the ripple effects of the semantic added to the model elements. Moreover, developers know that these problems (from structural to semantic inconsistencies) may happen in practice. However, they neither know their side effects nor grasp the meaning of the changes. To demonstrate this distinct side effect more clearly, let us take a closer look at the illustrative example in Figure 11, Figure 12, and Figure 13. As a prerequisite to produce the composed model, it is necessary to match the input model elements, which are suffering the effects of the changes performed by Peter, Steve, and Bill. For this, the composition technique identifies the similarities between the model elements. With addition based evolutions, the conflicting changes are identified because of the superimposition of changes: the composition algorithm detects that two contradicting values were attributed to a particular property defined in the language metamodel (e.g., *isAbstract* or *isDerived*). For example, Bill modifies the value of the property *return type* of the method *MarlimCore.handleInvalidOutput()* from *void* to *Status* (B3.step 4), while Steve modifies it to *String*. Similarly, Bill transforms the concrete class *PSElemenGroup* into an abstract class (B3.step 3), while Peter removes this class (P2.step 4).

Therefore, although the composition algorithm is effective to detect the changes, it is unable to identify whether the differences are caused by a simple (or multiple) modification, removal, or even refinement of model elements. Having more semantically richer information about the type of the changes, developers might detect and earlier resolve the conflicts. This would increase the number of correctly composed models as this semantic information aided those developers in making better-informed decisions.

With this in mind, to alleviate these problems would be necessary to grasp the actual meaning of the model elements (in the base model and delta model) and the impact of the change categories on their quality issues (e.g., comprehensibility and correctness). However, the current name-based, structural model comparison strategy has demonstrated to be ineffective to recognize intricate equivalence relationships between the model elements. The meaning of the model elements is rarely represented in a formal way. Hence, the definition of the correspondence between the input model elements is essentially based on a signature-based approach (Reddy et al., 2005). In doing so, the developers have to address some false positives and false-negative definitions of correspondence between the input model elements. However, the problem is rarely resolved without causing any negative effects on the developers' effort and expected characteristics of the design models e.g., correctness (Table 4).

Consequently, it was particularly challenging for developers to perform the compositions, or even for modeling managers, authorize the execution of the compositions. The developers are reluctant to compose the input models, and hence all potential benefits (e.g., gains in productivity) of the use composition in collaborative software modeling are compromised. In these cases, the current composition techniques are not effective to compose design models in collaborative model evolution.

#### 4.2.4.2.2. Conflict Management

The detection of all possible semantic conflicts between two versions of a model is an undecidable problem (Mens, 2002), as many false positive conflicts can appear. To reduce this problem, some previous works have recommended

reducing the size of the delta model in order to reduce the number conflicts (Perry et al., 2001). However, this approach does not ameliorate in fact the complexity of the changes. That is, the problem is not essentially the number of conflicts that the size of the delta can cause, but the complexity of the conflicts. To alleviate the effort to resolve the conflicts, we narrowed down the scope of the conflicts. For this, the delta model became to represent one or two functionalities of a use case in particular. Hence, the conflicts became more manageable and reasonable. Following this strategy, we were able to reduce the number and complexity of the conflicts. In practical terms, this complexity was minimized by reducing the number of functionalities implemented in the delta model. That is, the compositions had a smaller scope.

On the other hand, sometimes the changes with broader scope were inevitable in the delta model. This was, for example, the case when the models (e.g., class and sequence diagrams) were reviewed and meliorated for reasons of quality assurance. Unfortunately, this results in a decreased precision of the compositions due to the presence of non-trivial compositions. It is known that the domain independent composition algorithms cannot rely on the detailed semantics of the models being composed or on the meaning of changes. Instead of being able to identify all possible conflicts, the algorithms detect as many conflicts as possible, assuming an approximate approach. Consequently, developers need to deal with many false positive conflicts.

In practice, we noted that if the composition generates many conflicts, developers prefer throwing the models away (and investing more effort to recreate it after) to resolving all conflicts. Although the composition algorithm detects the conflicting changes created by developers in parallel, developers are unable to understand and proactively resolve these conflicts generated from non-trivial compositions. This can be explained by two reasons. First, the complexity of the conflicts affected the model elements. Second, the difficulty of understanding the meaning of the changes performed by other developers. More importantly, developers were unable to foresee the ripple effects of their actions.

This is linked to two very interesting findings. First, developers have a tacit assumption that the models to-be-composed will *not* conflict with each other, and a common expectation is that little effort must be spent to integrate models. Hence, developers tend to invest low effort to check whether the composition

produced inconsistencies or not. Therefore, we can conclude that the need to throw the model away in order to recreate it after demonstrates the complexity of the problem.

We have observed that the developers spend more effort when inconsistency propagation occurs. Although it is well known that the spread of the inconsistencies lead developers to spend some additional time to detect and resolve them, we have observed that this extra effort is due to, in part, the developers produce the inconsistencies are not the same to detect and resolve them. Note that in general inconsistencies are produced from the conflict resolution process performed incorrectly. This can be explained based on some reasons.

First, it is not always clear for developers that any inconsistency was produced. This perception is only realized along the project when the inconsistencies have already been resolved. Second, the inconsistencies tend to "keep alive" during the project because developers do not always detect and resolve the inconsistencies when they appear—either because they do not know which models are affected by the inconsistencies or either because the inconsistencies do not affect the use purpose of the models created by them.

In the first case, developers are concerned with the models under their responsibility i.e., models that they must produce. However, they feel comfortable to resolve inconsistencies localized in models that they are not under their responsibility. The main reason is that developers need to understand use cases (or scenarios) describing the functionalities represented in the diagrams. For a perfect understanding, developers should often grasp *business rules* and *design rules*, which define the domain elements and their constraints. That is, developers should know about the company business before resolving the inconsistencies. This represents one of the impairments to resolve the inconsistencies when they are detected. Another finding is that to resolve the inconsistencies, developers need sometimes to grasp the reasons why a composition was realized in one way and not in an expected manner.

In the second case, developers obligatorily spend effort to resolve inconsistencies that compromise the main purpose of use of the design models e.g., communication, but rarely to solve the inconsistencies that damage secondary purpose e.g., prediction. Developers do not solve all inconsistencies

due to time constraint. Consequently, they live with inconsistencies in practice. In our case studies, the models were used for improving the communication between the developers. Although other inconsistencies might be resolved, only the inconsistencies that jeopardize the comprehensibility of the models were necessarily solved. For example, the layout of the models was an ever-present concern during the modeling. This means that developers invested time to arrange the elements in the model to ensure a good understanding of the features. Therefore, all inconsistencies that affect this layout must be resolved; otherwise, the purpose of use of the model is compromised. We can conclude that, although it is desired to keep models without inconsistencies only the inconsistencies that affect the purpose of use of the models are resolved.

#### 4.2.4.2.3. Social Factors

The reputation of the developers influences the resolution of conflicting changes. We observed this finding during the observational study, interviews, and analyzing the change history in the repository. Recall that a developer can accept and reject a change of a second developer. This situation can be illustrated in turns of our motivating example. The developers Peter and Bill have distinct levels of experience. Peter is less experienced than Steve. Thus, if Peter performs a change that conflicts with another carried out by Steve (and he is not sure about how to resolve them) then he accepts the changes performed by Steve. That is, given that Peter is indecisive, he relies on the Steve's reputation.

Reputation can be seen as the opinion (or a social evaluation) of a member of the development team toward other developer. We have identified two types of reputation: *technical* and *social*.

Technical reputation refers to the level of knowledge considering issues related to the technology and tools used in the company such as the composition tool, IDEs, CASE tools, and version control systems. This type of reputation is mainly acquired solving daily problems. Social reputation refers to the position held by the members of the development team (e.g., senior developer). More experienced teamwork members (e.g., senior ones) influence less experienced members (e.g., novice ones). This happens mainly because the experienced ones

are the human face of the development projects, making important project decisions, and coordinating teams.

Knowing that the reputation of the developers might affect the conflict resolution, we investigated which reputation would cause more influence. For this, eight developers were interviewed. The data collected suggests that technical knowledge causes more influence on decision making than social reputation. More specifically, 75 percent of the developers (6/8) reported that the technical reputation would influence more developers' decisions than social one.

### 4.2.5. Limitations of Related Work

We contrast this work with previous studies considering empirical studies, development effort, composition techniques, and modeling language as follows.

*Empirical Studies.* It is well known that empirical studies in model composition are severely lacking (Uhl, 2008; France & Rumpe, 2007). Some authors have contributed toward clarifying how conflicts emerge and how they are tamed in artificial scenarios. For the most part, these works have considered limited composition scenarios compared to the scenarios evaluated in this work. Still, the most of them do not consider effort as the investigation variable.

The observational study in (Perry et al., 2001), for example, investigates the change history of a legacy system to delineate the boundaries of (and to understand the nature of) the problems considering the software development in parallel. The authors considered only one observational study and all work was concentrated in level of code. Another example would be the experimental report in (Altmanninger et al., 2009). That study analyzes the challenges in merging different versions of one model, proposes an initial categorization of typical changes, and identifies resulting conflicts from the compositions. Although interesting, the current empirical studies do not evaluate composition effort. Still, the findings are normally collected from artificial and limited case tests rather from realistic composition scenarios. Finally, some previous works (Mens, 2002; Whittle & Jayaraman, 2010; Dingel et al., 2008) reinforce the need for empirical studies in model composition.

Considering two empirical studies in model-driven development (Hutchinson et al., 2011a; Hutchinson et al., 2011b), Hutchinson and colleagues presents some initial results from a twelve-month empirical research study of model driven engineering (MDE). More specifically, they document a set of technical, organizational, and social factors that apparently influence organizational responses to MDE (Hutchinson et al., 2011a). In (Hutchinson et al., 2011b), they describe the practices of three commercial organizations concerning MDE approach to their software development. The main contribution is a range of lessons learned, reporting the importance of social factors instead of technical factors on the relative success, or failure, of the adoption of MDE in practice. The authors do not mention any problem concerning model composition during these qualitative studies. This does not mean it is not a problem in practice since they take a much broader view and ask questions that are more general about the role and effectiveness of MDE.

On the other hand, in (Uhl, 2008), Uhl points out that composition of enterprise artefacts is not a trivial issue. Most because it requires the composition of graphical views, forms, dialogs, and depends on "friendly" views to tame all conflicts between the multiple models. Hence, developers end up avoiding model composition and adopting pessimistic locking of design models. Therefore, our results can be seen as the first to empirically investigate RQ2.3 and RQ2.4 using the state-of-the-practice composition technique in industry.

*Development Effort.* A major contribution of our work is the investigation of composition effort as a critical factor for the acceptance of the composition techniques in practice. Some previous works have also demonstrated that the effort is a critical factor during the software development (Jorgensen, 2005). Usually the effort is based on *ad hoc* estimation (Farias et al., 2011; Jorgensen, 2005). Jorgensen (Jorgensen, 2005) highlights that effort estimation is still a real, open problem due to the lack of empirical evidences about the effort required to perform development tasks. In fact, estimating effort based on the expert judgment is the most common approach today. Even worse, these feedbacks are often diverging or overoptimistic. When we consider this problematic in the context of composition, the problem is aggravated. However, little has been done to investigate this problem.

*Composition Techniques*. Model composition is a very active research field in many research areas such as synthesis of state charts (Ellis & Gibbs, 1989), weaving of aspect-oriented models (Whittle et al., 2009; Klein et al., 2006; Whittle & Jayaraman, 2010), governance and management of enterprise design models (Norris & Letkeman, 2011), software configuration management (Whitehead, 2007), composition of software product lines (Jayaraman et al., 2007), and composition of design models (Nejati et al., 2007; Epsilon, 2011). For this reason, several academic and industrial composition techniques have been proposed such as MATA (Whittle et al., 2009), Kompose (Kompose, 2011), Epsilon (Epsilon, 2011), IBM RSA (IBM, 2011), and so on. With this in mind, some observations can be done.

First, these initiatives focus only on proposing the techniques instead of also demonstrate their effectiveness. Consequently, qualitative and quantitative indicators considering these techniques are still incipient. In addition, the situation is accentuated considering effort indicators. This lack hinders mainly the understanding of their side effects. Second, their chief motivation is to provide a systematic algorithm. Unfortunately, these approaches do not offer any insights or empirical evidences whether developers might reach the potential benefits claimed by using composition techniques in practice. Although some techniques are interesting approaches, they are fundamentally flawed because of the large number of false positives that will be produced for large-scale systems. Nevertheless, the effort required for the user to understand and correct composition inconsistencies will ultimately prove to be too great. The current study takes a different approach. It aims to provide a precise assessment of composition effort in real life context, quantifying effort and identifying the influential effort.

Next, current works tend to investigate on the proactive detection and earlier resolution of conflicts. Most recently, Brun (Brun et al., 2011a) proposes an approach, namely Crystal, to help developers identify and resolve conflicts early. The key contributions are that conflicts are more common than would be expected, appearing overlapping textual edits but also as subsequent build, and test failures. In a similar way, Sarma (Sarma et al., 2011) proposes a new approach, named Palantír, based on the perception of workspace awareness, on the detection and earlier resolution of a larger number of conflicts. Based on two

laboratory experiments, the authors confirmed that the use of the Palantír reduced of the number of unresolved conflicts. Although these two approaches are interesting studies, the earlier detection does alleviate the problem of model composition. The problem is the same, but is only reported more quickly. In addition, they appear to be overly restrictive to the code, not leading to broader generalizations at modeling level. Lastly, they neither make consideration about the effort to compose the artefacts used nor investigate the research questions in vivo case studies.

*Modeling Language.* There has been more research on evaluating the use of UML models (and its extensions) rather than the effort of composing them. These studies notably aimed at evaluating modeling languages in terms of some quality attributes such as comprehensibility (Lange & Chaudron, 2006), interpretation (Nugroho et al., 2008), and maintainability (Dzidek et al., 2008) rather than the composition effort. Additionally, most existing works have focused attention on exploring different quality issues considering UML models and understanding its appropriateness in mainly artificial scenarios. However, none of them attempt to understand how these quality issues may be affected during compositions and to examine a set of wider issues about the effort on composing these models in real-life scenarios. Some these issues include: are these quality issues of the UML models affected during the composition? In which composition tasks should the developer invest more effort? What is the trade-off between the composition tasks in practice? What are the characteristics of the UML models that help developers to compose them?

To sum up, there has been very limited empirical research evaluating the effort of composing large-scale design models in literature. Even worse, nothing has been done to both understand and describe the influential factors that can jeopardize the potential benefits of using composition techniques in industry. In particular, there are four critical gaps in current understanding. Firstly, the lack of practical knowledge on the effort of applying composition techniques, detecting and resolving inconsistencies in practice. More importantly, the lack of a trade-off analysis about three effort variables (Section 4.2.3.4.3). Secondly, a precise understanding about the influential factors of composition effort is lacking. Next, the lack of understanding of how technical and social factors can affect composition effort. Last, the absence of evaluation of important aspects in model

composition beyond modeling languages and composition techniques. Some of these aspects would be such as the potential benefits of good practice of software modeling, merging in pair (two or more developers work together to compose the input models), inconsistency management, and strategies to allocate tasks to minimize the composition effort.

### 4.2.6. Concluding Remarks of the Second Study

Model composition is a key mechanism to support the evolution of design models in large-scale software projects. In particular, this mechanism is essential to promote collaborative work of separate development teams whereas increasing their productivity. Thus, developers naturally become concerned about the quality of the software evolutions produced (i.e., the composed models) and the effort invested by the teamwork members. However, there is a lack of empirical studies evaluating model composition effort in practice. This means that little empirical findings can be converted into practical knowledge to the industry. Developers have no guidance on how to reduce model composition effort and the number of emerging model inconsistencies.

This study represents the first in vivo exploratory study to evaluate the *effort* that developers invest to compose design models (RQ2.3) and to identify and analyze the *factors* that affect developers' effort (RQ2.4). In our study, a best-of-breed model composition technique was applied to evolve industrial design models along 297 evolution scenarios. Developers conducted the work during 56 weeks, which resulted in more than 2 million compositions of model elements. We investigated the composition effort in this sample, and analyzed the side effects of key factors that affected the effort of applying the composition technique as well as detecting and resolving inconsistencies. All conclusions from RQ2.3 and RQ2.4 were drawn from quantitative and qualitative analyses based on the use of metrics, interviews, and observational studies.

We summarize the findings related to RQ2.3 as follows: (1) the application effort measures do not follow an *ad hoc* distribution and, rather, it assumed a distribution pattern; (2) the application effort tends to reduce as developers become more familiar with technical issues rather than application domain issues;

(3) the more experienced developers spend 23.2 percent less effort to detect inconsistencies than less experienced developers; and (4) the higher the number of inconsistencies in behavioral models, the higher the effort to detect inconsistencies. Additionally, we also present four findings with respect to RQ2.4 as follows: (1) the production of the intended model is strictly affected by the presence of different types of change categories in the delta model; (2) the closer to the Open-Closed principle the change is, the lower is the composition effort; (3) evolution scenarios that do not follow the Open-Closed principle required more effort to produce the intended model; and (4) the refinement (1:N) of model elements in the delta model caused severe composition problems and hence increased the composition effort.

Although there is a significant amount of quantitative and qualitative evidence supporting our findings previously mentioned, further empirical studies are still required to check whether they are observed in other contexts with different subjects. For example, we need to better understand if the composition effort is alleviated when developers compose well-modularized input models. There is some expectation that design models with an improved modularization can aid the composition techniques to accommodate the changes in the base model. Another two interesting investigation points would be: (1) Do developers invest more effort to compose behavioral models (e.g., sequence diagrams) than structural models (e.g., component diagrams)? (2) Do developers invest more effort to resolve semantic inconsistencies than syntactic ones? It is by no means obvious that, for example, developers invest less effort to resolve inconsistencies related to the well-formedness rules of the language metamodel than to resolve inconsistencies considering the meaning of the model elements.

Finally, we hope that the issues outlined throughout the thesis encourage other researchers to replicate our study in the future under different circumstances. Moreover, we also hope that this work represents a first step in a more ambitious agenda on better supporting the model composition tasks.