7
Conclusions

This thesis addresses several limitations of the current literature with respect to empirical evaluation of model composition effort. An overall research question has been formulated to specify the scope of this thesis: How can the composition of design models be evaluated with respect to developers' effort? This overall question was further decomposed into four specific research questions (Section 1.3); the goal was to explicitly investigate specific dimensions of model composition effort. Even though many contributions have been presented in the previous chapters, overall conclusions need to be drawn and much work remains to be done. Therefore, this chapter: (i) summarizes the main topics studied (Section 7.1) to address our research questions, (ii) refines the contributions previously discussed (Section 7.2), and (iii) gives directions for future work (Section 7.3).

7.1. Summary

Model composition plays a pivotal role in many software engineering activities. Moreover, software modeling is increasingly becoming a collaborative work. However, a clear understanding of the effort required for composing design models is still a challenging task. Developers need to know how to quantify this effort and grasp the possible factors that influence it. To address these issues, a systematic evaluation approach for model composition effort and a range of empirical studies are crucial.

Most existing work on model composition proposes new composition techniques (Sarma et al., 2011; Epsilon, 2011; Whittle et al., 2009). In addition, as far as the assessment of such techniques is concerned, nothing has been done so that an evaluation framework for model composition can be proposed. Even worse, there is no empirical study aimed at understanding how certain software modeling factors affect model composition effort in practice. As a result, developers are left without any evaluation framework and practical knowledge about how to identify model composition problems and alleviate the developers' effort.

We believe that without practical knowledge derived from empirical investigations (rather than conflicting advice of evangelists (Norris & Letkeman, 2011)), it is not possible to realize well-informed improvements on techniques and strategies for model composition. It would be not possible, for example, to tame the side effects of the influential factors - such as the composition technique, the design decomposition, and model stability - more effectively. With this in mind, we investigate four research questions (Section 1.3) and confront the results collected from them. Thus, developers can be aware of the overall cost of composing design models and identify means to ameliorate this cost.

In this context, this thesis proposes a quality model (RQ1) derived from our experience of conducting a series of empirical studies. This quality model identifies three relevant factors: the model composition techniques, the design decomposition technique, and model stability. More importantly, the quality model identifies a series of quality notions, including semantic, syntactic, social, and so on. This framework for evaluating model composition has guided all empirical investigations performed in this thesis. We believe that this quality model also serves as a guideline for other researchers to select procedures and metrics while evaluating how the same or different influential factors affect the model composition. Given the unifying terminology of our quality model, it also enables to map, contrast, and bring together findings from different empirical studies on model composition effort.

After defining the quality model (RQ1), we started investigating the effects of specific model composition techniques on the developers' effort (RQ2). More specifically, we evaluate the effects of some specification-based and heuristic-based composition techniques on the developers' effort and the correctness of the output composed models. This evaluation is performed based on a set of empirical studies including one controlled experiment, five industrial case studies, observational studies, and interviews. The combination of these studies allows to build a body of knowledge about the effort that developers invest to compose design models. The results, supported by statistical analyses, contradict the intuition by disclosing that specification-based techniques neither reduce the developers' effort nor assure the correctness of the compositions when compared to the heuristic-based techniques.

Following the studies of the four research questions, we investigate the effects of alternative design decompositions (e.g., OOM and AOM) on the effort to detect inconsistencies (RQ3). We performed one controlled experiment, five industrial case studies, observational studies, and interviews to understand these effects. This allowed us to study RQ3 from different perspectives. The results, also supported by a complete statistical analysis, show that aspect-oriented modeling neither increased the inconsistency detection rate nor improve the interpretation of the models. However, developers invested less effort to detect inconsistencies in AO models than in OO models.

Lastly, we investigate the effort that developers spend to resolve inconsistencies (RQ4). For this, we study the influence of modeling languages and model stability on the inconsistency rate and on the effort to resolve these inconsistencies. From two quasi-experiments in the context of the evolution of design models, the results revealed that aspect-oriented design models had a higher inconsistency rate than non-AO ones. However, the inconsistency resolution effort required by AO models was lower than the OO models. The model stability has shown to be a good indicator of high density of inconsistency and resolution effort. That is, unstable models tended to present a higher inconsistency rate and require a higher effort to transform the output composed model into an output intended model. All results were supported by statistical tests.

7.2. Contributions

We claim that evaluation of model composition must not only be based on conventional design attributes. Model composition evaluation must be oriented by the effort that developers should invest to produce an output intended model. This research work defined an evaluation approach that promotes effort as an explicit measurement unit, thereby filling the gap between experimental investigations and the influential factors that affect the composition effort. Additionally, we applied this new evaluation approach in a series of empirical studies in order to evaluate the effects of the influential factors on: (i) the effort to apply composition techniques, (ii) the effort to detect inconsistencies, and (iii) the effort to resolve inconsistencies.

After investigating the four research questions in the previous Chapters, we refine the contributions of this work stated in Chapter 1.

1. *A quality model for model composition effort* (RQ1). As previously mentioned in Chapter 1, the central topic of this thesis is the empirical evaluation of effort on composing design models. Therefore, we first define quality notions for model composition effort to be applied in this thesis (Section 3.5.2). We selected and extended existing quality models for software modeling in the context of model composition. In total, seven quality notions were introduced in the proposed quality model, namely syntactic, semantic, social, effort, application, detection, and resolution. The syntactic, semantic, and social quality notions were tailored from the previous studies, while the effort, application, detection, and resolution quality notions were proposed in this thesis. We believe that these quality notions together are effective to comprise a basic quality model for model composition effort. The quality model was defined in four levels following a metamodeling approach. Its main practical contribution is to guide researchers and developers in two main contexts: (i) the adoption of a unifying terminology related to the evaluation of model composition effort – this adoption enables the comparison of different studies and their findings, and (ii) the selection of metrics for structuring empirical studies on model composition (Section 3.5.3). In fact, this model has driven all studies in Chapters 4, 5, and 6; we observed that this model was effective to support our evaluation of different facets of model composition effort through the empirical studies. For instance, the quality model was instantiated to select metrics as well as structuring the procedures required to evaluate how the influential factors affect model composition effort.
2. *Practical knowledge on model composition effort (RQ2,3,4).* To address RQ2, RQ3, and RQ4, we apply the quality model to assess the effects of the composition factors on the model composition effort. Empirical knowledge was reported from a series of experimental studies including: two controlled experiments, five industrial case studies, three quasi-experiments, more than

fifty interviews, and observational studies. The chief contributions were practical knowledge about the impact of the influential factors on: (i) the effort to apply model composition techniques (Chapter 4), (ii) the effort to detect inconsistencies (Chapter 4 and 5), and (iii) the effort to resolve inconsistencies (Chapters 4, 5, and 6). Moreover, practical knowledge about how to: (i) evaluate the developers' effort, (ii) reduce the likelihood of emerging inconsistencies, and (iii) tame the side effects of the influential factors are defined in the previous Chapters 4, 5, and 6. An overview of the generated knowledge is emphasized as follows:

Model Composition Techniques

a) Developers tend to spend less effort by using the heuristic-based techniques rather than the specification-based techniques. In fact, the heuristic-based techniques required less effort to apply them, detect inconsistencies, and resolve inconsistencies. Consequently, the general composition effort invested by developers was lower. The traditional algorithms required less effort than the IBM RSA, which in turn required less than the Epsilon.
b) The specification-based technique did not reduce the inconsistence rate whereas also got higher measures than the heuristic-based techniques. Developers were not more effective to produce the output intended model by using the specification-based composition techniques. This finding did not confirm the claims reported in the current literature that such techniques significantly reduce the number of inconsistencies compared to the heuristic-based composition techniques (Epsilon, 2011; Kolovos et al., 2011; Kompose, 2011; Whittle et al., 2009). This finding indicates that developers should more carefully use specification-based techniques.
c) The specification-based techniques added undesired difficulties to specify the similarity between the input model elements. In particular, it was challenging for developers to proactively write down match and merge rules, which were able to produce an output intended model. Severe compositions dominated by relations of the type many-to-many (N:N) between the input model elements

characterized the most effort-consuming scenarios. In short, the specification-based technique demonstrated to be a highly intensive manual task and more prone to errors. This leads to the insight that developers should be equipped with heuristics that, for instance, automatically recommend relations between elements of the input models.

d) The aforementioned results also lead to three lessons: (1) the model composition techniques should be more 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 as soon as they arise; and (3) new composition techniques could be a mixture of specification-based and heuristic-based techniques.

a) **Design Decomposition Techniques**The technique used for design decomposition, such as object-orientation and aspect-orientation, definitely has a profound impact on model composition effort. For instance, developers tend to detect more inconsistencies in OO models than in their AO counterparts. Therefore, AO models explicitly representing crosscutting modularity do not necessarily imply on more effective inconsistency detection. This contradicts somehow the intuition that the improved modularity of AO models would help developers to localize inconsistencies. Therefore, developers of AO designs should be more conscious that the increased number of abstractions in AO models requires more attention from them while revising the output composed models.

b) Developers tend to invest more effort to detect inconsistencies in OO models than in AO models. In fact, developers tend to report more often the presence of inconsistency in AO models (compared to OO models) instead of trying to find any other solution. On the other hand, by using OO models, developers try to provide more often the corresponding implementation even observing the presence of inconsistencies. That is, the superior modularity of AO models accelerates inconsistency detection. Therefore, this implies that although developers detect fewer inconsistencies in aspect-oriented models, they spend less effort to localize them.

c) Developers localized more quickly inconsistencies in AO models when the scope of aspect pointcuts is narrow, thereby confronting structural and behavioral information about the crosscutting relations. This faster localization happened because the similarity between advices represented in structural and behavioral diagram allowed an “easy transition” between the two diagrams. This leads to the insight that developers should, whenever it is possible, avoid wildcards in their pointcuts and break them down in more explicit pointcut expressions. This strategy seems to improve the readability and consistency detection in AO models.

d) AO models with inconsistencies tend to cause a higher number of misinterpretations compared to the OO counterparts. The presence of the inconsistencies cause a detrimental effect due to the nature of the AO constructs. In fact, the need to scan all join points affected by the aspects increased the likelihood of different interpretations by developers. Therefore, we confirmed our initial expectation that by using contradicting AO design models would lead to a higher number of diverging interpretations of the participants. Therefore, developers working on parallel on aspect-oriented design should be more conscious about the increased likelihood of different design interpretations by the team members.

e) Developers tend to consider the sequence diagrams as the basis for the design implementation, as it is closer to the final implementation of the method (or advice) bodies; hence, developers become confident that the information present in the sequence diagram is the correct one compared to the class diagram. That is, the lower level of abstraction of this diagram leads the software developers to be more confident into the behavioral diagrams than the structural ones. Therefore, inconsistencies in behavioral diagrams tend to have a superior detrimental effect than those in class diagrams.

Design Characteristics

a) A number of design characteristics, such as coupling and size, play a role in the stability characteristic of an evolving design. We have observed that the inconsistency rate and the inconsistency resolution effort in stable design models are significantly lower than in unstable design models. The model stability has demonstrated to be a good indicator of inconsistency rate and inconsistency resolution effort. This also leads to the insight that developers should also invest upfront on applying well-known design principles to improve the stability of each new delta model to be composed. This is going to save cost involved in resolving critical inconsistencies later.
b) The location where the inconsistencies emerge is important. For instance, inconsistencies are more harmful when they take place in design model elements realizing mandatory features of software product lines. Because inconsistency propagation is often higher in model elements implementing mandatory features than in alternative or optional features. When inconsistencies emerge in elements realizing optional and alternative features they also tend to naturally propagate to elements realizing mandatory features. Consequently, the mandatory features end up being the target of inconsistency propagation. This observation further confirms the importance of structuring well key modules of a system in order to avoid instability and critical inconsistencies later.
c) Developers must structure product-line architectures in such a way that inconsistencies can keep precisely “confined” in the model elements where they appear. Otherwise, the quality of the products extracted from the SPL can be compromised; as the core elements of the SPL can suffer from problems caused by incorrect feature compositions. The higher the number of inconsistencies, the higher the chance of them to continue in the same output model, even after an inspection process performed by a designer. Consequently, the extraction of certain products can become error-prone or even prohibitive.

7.3. Future Works

This section categorizes the areas where future work is still required such as composition technologies, additional quality notions and heuristics, formal foundations, and additional empirical investigations. These areas are discussed below.

Improvement of Model Composition Technologies

We can highlight two main areas in which supporting tools would be pivotal to improve model composition in the context of real-world projects: support for improved awareness in collaborative model composition activities; and automated detection and resolution of inconsistencies.

First, it would be useful to investigate and develop model composition tools that support developers with awareness about model composition activities being performed in parallel. These tools should be able to make developers conscious about relevant changes in the design model elements. This improvement is important because developers should be able to identify conflicting changes earlier than the model composition time. Therefore, future work in this area will be focused on including support for “awareness” in model composition tools, such as IBM RSA and Kompose (Kompose, 2011).

Second, the current software modeling tools should support the anticipation, detection, and resolution of the most critical inconsistencies. Since, it is particularly challenging for developers to detect and resolve severe inconsistencies without any guidance (or recommendations) supported by tools. Therefore, as a future work in this direction, the model composition tools might incorporate, for instance, the use of model stability as an indicator of severe inconsistencies emerging in the output composed models. After the detection of inconsistencies, a recommendation system should assist the developers to resolve the inconsistencies.

Additional Quality Notions

The proposed quality model for model composition effort was defined based on the limitations of existing quality models and from empirical studies. A possible direction for future research related to the quality model is to go further in its application in different contexts. By doing so, new empirical studies might be planned and carried out to evaluate the quality model considering the different purposes of using model composition. In this thesis, the quality model was mainly evaluated in the context of changing and reconciling of deign models (Section 3.5.3), but the model may be applied to support the analysis of overlapping design models. In this context, quality notions such as social and effort quality should be investigated.

Formal Foundations

The specification of the metrics and the quality model in this thesis is informal. Therefore, we cannot state that their definitions are, for instance, mathematically sound and fully free of ambiguities. We believe that a formal foundation for the metrics and the quality model is a useful additional step in the future. For example, the metrics could be formalized using set theory and theoretically evaluated using systematically criteria from the measurement theory.

Additional Empirical Investigations

We can highlight at least two requirements for replications of the studies performed in this thesis.

First, even though the results of the studies ($RQ_{2,3,4}$) were statistically significant, the studies were limited with respect to the types of design models and inconsistencies analyzed. More types of inconsistencies and models should be analyzed in replications of our studies. This would allow us to confront the collected data with the new data. Another proper way to go is to investigate the effects of inconsistency propagation on the inconsistency detection rate, detection effort, and the degree of misinterpretation of the design model. In this study, we have observed that inconsistencies in AO models led to a superior misinterpretation compared to OO models. However, further studies should be performed to evaluate, for example, whether the inconsistencies are in fact converted into a higher number of implementation defects in AO programming rather than OO programming. That is, we are going to investigate if inconsistencies in design level are converted into defects in code. Moreover, it would be great to investigate the effects of key properties in AO modeling such as obliviousness and quantification on the inconsistency detection rate, detection effort, and misinterpretation.

Second, although the results (RQ2) were also statistically significant, the study considered small design models and a low number of subjects. Thus, the results may have been threatened by the size of the design models or by level of experience of the subjects. Therefore, future works might replicate the study by considering more experienced subjects and more complex design models.