3
A Quality Model for Model Composition Effort

Software quality is defined as “conformance to requirements” (Boehm, 1978). Therefore, the quality of a software system can be seen as the characteristics that lead its comprising artifacts or its development activities to satisfy a set of requirements. A *software quality model* defines and organizes the concepts required to characterize or evaluate the quality of a software system (Lange & Chaudron, 2005b; Boehm et al., 1978). Certain quality models are intended to be general — i.e., they can be used to evaluate certain quality attributes in any software engineering context. However, in order to be useful in practice, each quality model should support the evaluation of a particular category of software artifacts and/or software development activities relevant to a certain software engineering context, such as model composition.

In this context, a quality model for model composition effort should: (i) define the conceptual elements required to characterize and evaluate model composition effort, and (ii) define and structure the quality notions (Lange, 2007; Boehm et al., 1978) that are relevant to model composition artifacts and activities. A quality model with these components is proposed in this thesis. The goal of this quality model is to fill the gap in the current literature that fails to provide adequate quality frameworks for model composition.

Therefore, the goal of this chapter is to define a quality model for model composition effort. This quality framework serves as a guideline for researchers and developers to carry out qualitative investigations considering model composition effort and to assess any quality achievements. The proposed quality model (Section 3.5) is a practical quality framework built from evidence-based knowledge acquired throughout the execution of a series of empirical studies (Table 1). The empirical studies range from controlled experiments, case studies, quasi-experiment, and observational study. These studies will be described in Chapters 4, 5, and 6. Additionally, this quality model is also based on (1) experience obtained from previous works performed over the past six years (Table

1), and (2) previous quality models such as (Marín et al., 2010; Lange, 2007a; Lindland et al., 1994; Boehm et al., 1979; McCall et al., 1977). Although the proposed quality model overcomes the limitations of related work (Section 3.2) and it can be applied to any design models, it does not aim to be a final and complete one. With this in mind, it has been designed to be extensible so that other researchers can tailor it for different purposes.

The creation of this quality model requires answering some open questions. First, what are the artifacts and activities involved in model composition? What do we expect from model composition? Developers do not know which tasks should be performed and what models participate in a model composition process (Section 3.3). Second, how can we evaluate the model composition effort? Researchers do not know which evaluation criteria should be used (Section 3.5), and how they can contribute to achieve the required quality (Fitzpatrick, 1999). Therefore, the proposed quality model addresses the first research question of this thesis (RQ1): *How can the evaluation of model composition be organized in terms of a comprehensive framework?*

The remainder of this chapter is organized as follows. First, Section 3.1 provides some additional motivation for our quality model. Then, Section 3.2 discusses the limitations of the related work. Section 3.3 defines how model composition effort can be evaluated. Section 3.4 defines composition conflicts and inconsistencies. Finally, Section 3.5 brings forward the quality model, which serves as the reference frame for the empirical studies conducted throughout this research.

3.1. Motivation

Although researchers and developers recognize the importance of evaluating model composition (France & Rumpe, 2009; Farias et al., 2010), the practice of this evaluation is not a trivial task (Basili & Lanubile, 1999; Basili et al., 1999). This can be explained by some reasons. First, the current quality models fail to define the concepts (and their relations) required to characterize and evaluate model composition. Examples of these concepts are conflicts, inconsistencies, types of modeling languages, and model composition techniques. These concepts

are not even mentioned in the current quality models. Hence, it is not possible to study the interplay of these concepts and model composition effort.

Second, because of the aforementioned problem, the use of prevailing quality models, discussed in Section 3.2, does not enable developers to distinguish between: (i) general quality notions that are typically associated with the design models in general, and (ii) quality notions that are specifically relevant to the evaluation of model composition effort. Rather, they only take into account well-known general concepts in software modeling. The imprecise specification of specific quality notions for composition effort causes misunderstanding about what should be evaluated in this context. Even worse, researchers cannot properly formulate and test hypotheses as well as replicate studies. If researchers cannot replicate studies, then the generalization of the results is hindered.

Third, the lack of a quality model jeopardizes the understanding about how conclusions can be drawn and related. According to (Basili and Lanubile, 1999; Wohlin et al., 2000), the degree of validity of any finding of empirical studies depends on how conclusions are drawn — i.e., the degree of confidence in a cause-effect relationship between the study variables and to what the extent the conclusions can be extrapolated to other contexts. A quality model guides researchers to investigate cause-effect relationships and promote the alignment between the results of empirical studies. Without a quality model, the conclusions across multiple studies are weakly connected, and a body of knowledge about model composition cannot be built.

Finally, the understanding of model composition is based on common wisdom, intuition, evangelist feedback, or even proofs of concepts. All these sources of information are not reliable sources of knowledge (France & Rumpe, 2007). Therefore, the lack of a quality model for model composition is a key factor for the empirical evaluation of effort on composing design models. In fact, without an adequate quality model the problem stated in Section 1.1 cannot be addressed. In the following section, we discuss the limitations of the related work.

3.2. Limitations of Related Work

Researchers recommend the use of quality models in empirical investigations (Runeson & Höst, 2009; Wohlin et al., 2000). In (Runeson & Höst, 2009), Runeson and Höst highlight the need for a reference frame (e.g., quality model or theory) to plan and execute case studies. The authors emphasize, for example, that quality models make the context of the empirical study clearer, and help researcher to conduct as well as review the results obtained. In (Wohlin et al., 2000), Wohlin and colleagues also confirm the importance of a quality model for empirical investigations.

To date, most approaches involving model composition rest on subjective assessment criteria. They depend on experts who build up an arsenal of mentally held indicators to evaluate the growing complexity of the produced design models (France & Rumpe, 2007). Consequently, developers ultimately rely on feedback from experts to determine "how good" the input models and their compositions can be. According to (France & Rumpe, 2007; Uhl, 2008), the state of the practice in assessing model quality provides evidence that modeling is still in the craftsmanship era and when we assess model composition this problem is accentuated. Finally, to the best of our knowledge, the need for methods for qualitative evaluation during a model composition process neither have been pointed out nor even proposed by current model composition techniques (Brun et al., 2011a; Maoz et al., 2011; Apel et al., 2011; Sarma et al., 2011; Dingel et al., 2008; Zito, 2006).

Some quality models in the area of modeling have been proposed through the last decades, such as (Marín et al., 2010; Lange, 2007; Lindland et al., 1994; Boehm et al., 1979; McCall et al., 1977). In (Boehm et al., 1979) and (McCall et al., 1977), the authors present quality models for conceptual modeling. However, both of them do not convey any concept related to model composition, such as conflicts and inconsistencies. In (Lange, 2007), Lange aims at proposing an extension of (Boehm et al., 1979) and (McCall et al., 1977) in the context of software modeling; they provide guidelines for selecting metrics and rules to quantify the quality of UML models. The purpose of this quality model is to support a broad quality evaluation of UML models. Although the Lange's quality

model has been created based on a literature review and on experiences from industrial case studies, it is not suitable to evaluate model composition effort due to the reasons described in the previous section.

Moreover, we have also observed that previous works have been structuring and specifying the quality model in different ways. Although Boehm (Boehm et al., 1978), McCabe (McCabe, 1976), and Lange (Lange, 2007a) structure their proposed quality models following a hierarchical approach, they differ as to the manners of the hierarchical levels are defined. Each level defines a different set of concepts of the quality model. For example, McCall defines the quality framework in three hierarchical levels containing *Uses*, *Factors*, and *Criteria*, respectively. Boehm uses a different vocabulary but similar meaning for these levels. On the other hand, Lange proposes his quality model with four hierarchical levels containing *Use*, *Purpose*, *Characteristics*, and *Indicators*. Our proposed quality model adopts these four levels as the relation between quality notions and the indicators can be better specified and understood.

As mentioned in Section 3.1, the current quality models fail to specify the relations between the concepts found in software modeling and the ones defined in model composition. Hence, it is not trivial to grasp how developers' effort can be quantified only considering the concepts defined by Lange (Lange, 2007a). They are *User*, *Modeling Language*, *Domain*, and *Design Model*. It is not possible to answer whether, in fact, there are (or not) relations between those concepts and those found in the realm of model composition. For example, the related works do not discuss how the above concepts would relate to concepts such as *Conflict*, *Inconsistency*, and *Model Composition Techniques*. Understanding if these relations are possible, or even how it would occur, is important when studying model composition effort.

In 2010, Marín proposes a quality model based on the metamodeling standard (Marín et al., 2010). This type of specification offers some advantages concerning the previous ones. First, the elements of a quality model are defined by a description, syntax abstract, and semantics constraints. Second, the UML metamodel is also defined following a metamodeling approach. This means that the use of metamodeling can favor the comprehension of the quality model as developers are often familiarized with the UML specification. More specifically, the purpose of the quality model is to formalize the elements involved in the

identification of the different types of defects relevant to Model-Driven Development (MDD). This not only encapsulates common defect types usually found in MDD, but also takes advantage of current standards in order to automate defect detection in MDD environments (Marín et al., 2010).

According to Boehm (Boehm et al., 1978), McCall (McCabe, 1976), and Lange (Lange, 2007a), researchers can evaluate software systems by relating metrics to quality attributes. Today, there are many works defining metrics in order to measure source code and design models such as (Fenton et al., 1996; Chidamber & Kemerer, 1994; McCabe, 1976; Martin, 2003). However, none of them explores the relation of metrics and quality notions in the context of model composition assessment. For example, in (Chidamber & Kemerer, 1994), the authors define a set of canonical metrics for OO designs, such as coupling between object classes (CBO) and the lack of cohesion in methods (LCOM).

Martin in (Martin, 2003) proposes another metrics and discusses design characteristics, such as stability. Although these works are effective to assess quality attributes of both source code and design models, they are inadequate to assess the model composition effort. For example, these quality models do not consider important elements in model composition, such as conflicts, inconsistencies, and composition techniques. That is, the current quality models are unable to guide researchers during the planning of empirical studies about model composition effort. This thesis, therefore, extends the previous quality models so that researchers and developers are able to characterize and evaluate model composition tasks. We structure the proposed quality model by using a four-level framework following a metamodeling standard, as in Marín's work (Marín et al., 2010). The proposed extensions are described in the next sections. Nevertheless, the main differences are (1) an abstract syntax is defined to represent the concepts that are the basis of the quality model, (2) new concepts are included in the model (such as conflict, inconsistency, composition technique, and design characteristic), and (3) four quality notions are added (such as effort, application, detection, and resolution notions).

3.3. A Quality Model for Model Composition Effort

After motivating the quality model (Section 3.1) and contrasting the related works (Section 3.2), this section describes the quality model for model composition effort, which is based on previous works (Lange, 2007; Krogstie, 1995; Lindland et al., 1994; Marín et al., 2010).

3.3.1. Model Composition Effort and Change Categories

In this section, we define model composition effort and the types of changes that are applied to the design models during the empirical studies. Moreover, this section answers some questions that have motivated the creation of the quality model (Section 3.1).

To begin with, we identify the different types of effort that developers can invest to produce an output intended model. Model composition effort can refer to the time invested (or the number of activities required) to produce the output intended model. In Figure 3, an effort equation summarizes three complementary facets of model composition effort. The equation makes explicit that developers invest effort to realize three activities to compose the base model, $M_{A,}$ i.e. the model to-be changed, and the delta model, M_B, so that the intended model, M_{AB}, can be produced. However, some additional effort may be invested to solve inconsistencies in the composed model, M_{CM}:

1. $f(M_A,M_B)$: effort to apply composition technique to produce M_{CM} from M_A and M_B.
2. $diff(M_{CM},M_{AB})$: effort to detect inconsistencies in M_{CM}.
3. $g(M_{CM})$: the effort to resolve inconsistencies i.e., the effort to transform the composed model (M_{CM}) into the intended model (M_{AB}). Note that if M_{CM} is equal to M_{AB}, then $diff(M_{CM},M_{AB}) = 0$ and $g(M_{CM}) = 0$. Otherwise, $diff(M_{CM},M_{AB}) > 0$ and $g(M_{CM}) > 0$.

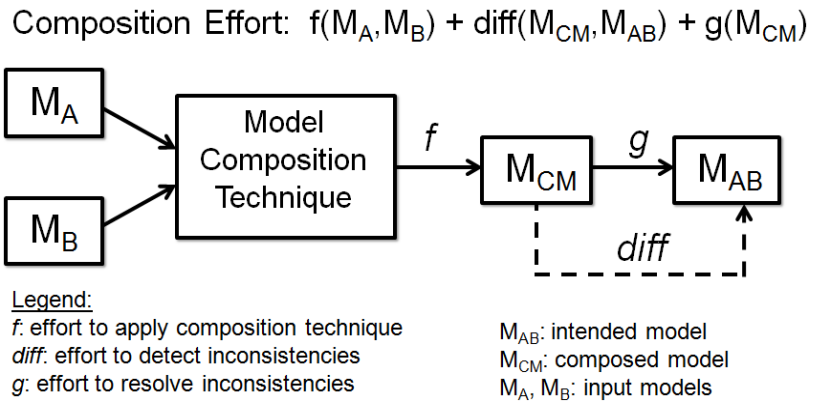

Figure 3: Overview of model composition effort: an equation

Developers spend effort to accommodate changes from the M_B to the M_A. We have identified four types of changes that usually happen during this composition, which are widely accepted by researchers (Mens, 2002). Note that the quality model is not limited to be used to these changes. The changes are described as follows:

- *Addition*: new model elements from some delta model are inserted into the base model; for instance, the new attribute – *name: String* is inserted into the class Researcher (Figure 4).
- *Removal*: a model element in the base model is removed; for example, the attribute, *+salary: int* is removed from the class *Researcher*.
- *Modification*: a model element has some properties modified; for instance, the class Researcher in the base model has its property *isAbstract = false* modified to true in the delta model (name in italic style).
- *Derivation*: model elements are refined and/or moved to accommodate the changes (Mens, 2002); for example, the class *Researcher* in the intended model (Figure 4) has the attributes *name* and *salary* moved to the classes *Assistant* and *Professor*.

When developers accommodate these different types of changes into the base model (M_A) some conflicts between the properties of the design models can arise. We present the concept of conflicts and inconsistencies in the next section.

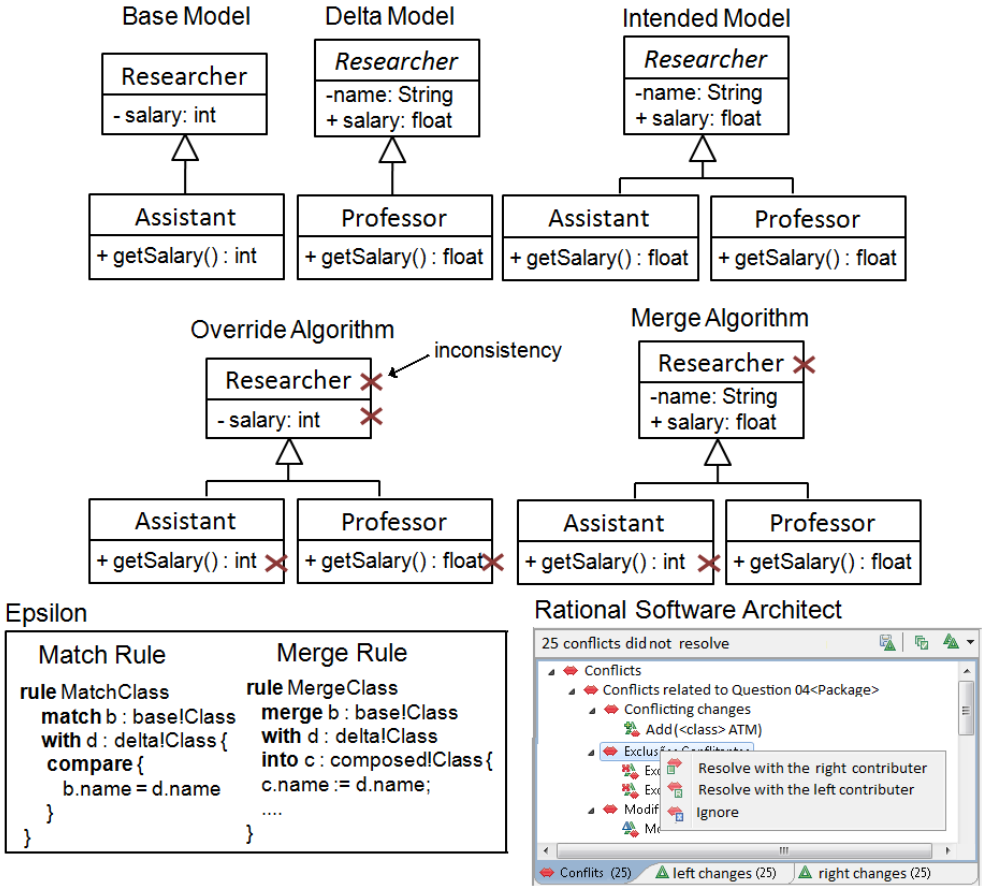

Figure 4: Illustrative example

3.3.2. Composition Conflicts and Inconsistencies

Composition conflicts consist of contradictions between the values assigned to the properties of the design models (Mens, 2002). They emerge when the input models M_A and M_B need to be composed and their overlapping parts have contradicting values. Figure 4 shows a practical example of conflicting changes when we try to compose the classes *Researcher* of the base and delta model.

In the base model, the UML class *Researcher* is defined as a concrete class (i.e., *Researcher.isAbstract* = false) whereas in the delta model class *Researcher* is set as an abstract class (i.e., *Researcher.isAbstract* = true). That is, we have contradicting values assigned to the same class. Then, the developers need to properly answer the question: should class *Researcher* be abstract or not? In this particular case, the correct answer is that the *Researcher* is abstract — i.e., *Researcher.isAbstract* = true. This can be observed in the intended model in Figure 4.

However, if this question is not properly answered, inconsistencies are inserted into the output composed model. Inconsistencies are unexpected values assigned to the properties (or characteristics) of the design models. For example, *Researcher.isAbstract* = false represents an inconsistency as the expected value is true. Note that when the conflicts are incorrectly resolved they are converted into inconsistencies in the output composed model. Figure 4 shows the class *Researcher* produced by the override and merge algorithms (Section 2.4.1) as a concrete class (*isAbstract* = false) instead of abstract (*isAbstract* = true) as would be expected. Note that these inconsistencies lead the model to-be considered not compliant with the intended model. Two categories of inconsistencies can emerge as follows:

- *Syntactic inconsistency* emerges when any output composed model elements do not conform to the rules defined in the modeling language's metamodel. For example, a class must have attributes with different names.
- *Semantic inconsistency* arises when the meaning of the elements of a composed model does not match with the elements of the intended model. For instance, a class in M_{CM} has an unexpected method or it requires functionality from another class that no longer exists.

We consider both categories of inconsistencies throughout this thesis. The composition techniques, such as IBM RSA (Section 2.4.2), are able to automatically detect syntactic inconsistencies while the semantic inconsistencies can be only detected manually. The composition techniques are unable to detect semantic inconsistencies because semantic information about the model elements is rarely represented in a formal way.

Metric	Description
NFCon	The number of inconsistent functionalities
NCCon	The number of model elements that are not compliant with the intended model
NDRCOn	The number of dangling reference inconsistencies
NASCon	The number of abstract syntax inconsistencies
NUMECon	The number of meaningless model elements
NBFCon	The number of behavioral feature inconsistencies

Table 2: Metrics of semantic inconsistencies (Farias et al., 2008)

Hence, the composition techniques cannot proactively localize such inconsistencies. With this in mind, six metrics are proposed. Table 2 briefly presents these metrics. These inconsistencies were chosen because we have observed from empirical studies that they are the most common types of inconsistencies faced by developers in practice (Farias et al., 2008; Mens, 2002).

3.3.3. Abstract Syntax of the Quality Model

The goal of the abstract syntax is to define the quality model more precisely, thereby identifying the main concepts and their relationships. As this quality model is based on previous works (Lindland et al., 1994; Krogstie, 1995; Lange, 2007), the extensions are based on the creation of four new model elements, and six relationships, which are discussed as follows.

Figure 5 shows the abstract syntax of the proposed quality model, which relies on the metamodeling pattern used in the UML metamodel (OMG, 2011). Note that the numbers in Figure 5 correspond to the numbers in brackets of the quality notions to be discussed in Section 3.5.2. We adopted the UML metamodel as a reference because the UML is in fact the standard modeling language in both

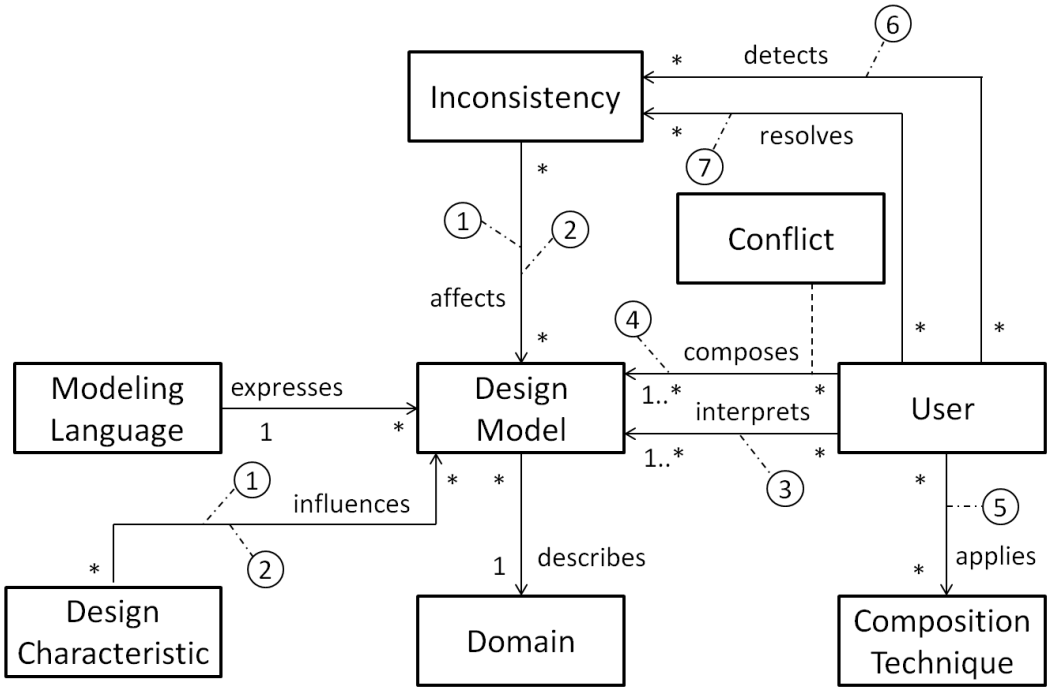

Figure 5: Abstract syntax of the quality model for model composition (based on (Lange, 2007))

academia and industry (Dobing & Parsons, 2006). It is important to highlight that each association represents some effort that developers should invest. With this in mind, the elements of the abstract syntax (Figure 5) are presented as follows.

a. Domain

The first element to be discussed is the concept of *domain*. This concept represents an area of expertise or application that needs to be examined to solve a problem. The solution of the problem is represented in a design model. In other words, a domain consists of a reality to be represented by using a modeling language. Supply chain, finance, and telecommunications are three examples of domains. Typically, it can be stated as a conceptual model where a set of concepts and relations are represented.

Association

- Without a directed relationship

b. Modeling Language

Modeling language is the concept that represents the language used to design a software system. Object-oriented modeling languages and aspect-oriented modeling languages are two examples of typical categories of languages used to represent significantly different forms of design decompositions. Modeling languages are commonly used in practice to improve the communication between development teams and provide alternative means for achieving design modularity. Different modeling languages – such as object-oriented and aspect-oriented ones – may influence the structure of a design. Software engineers use these languages to communicate design decisions and check the feasibility of implementing the envisaged design. Example of a premier software modeling tool is the IBM Rational Software Architect (IBM RSA, 2011). The modeling languages define a set of constructs that are used to create instances of the design models.

Association

- expresses: Design Model[*]

  Each *expresses* represents the statement of design models. An *expresses* means that the constructs of the design modeling language are instantiated to create a *Design Model* concerning some *Domain*.

UML and its profiles are examples of design modeling language used in practice. This is an ordered association from *Modeling Language* to *Design Model*.

c. Design Model

Design model refers to the diagram used to represent static and dynamic aspects of a software system. UML class and sequence diagrams are examples of these design models. Developers commonly use these two diagrams, for example, to design structural and dynamic aspects of an application. Moreover, a design model represents the concepts (and their relations) from a domain. This representation helps to describe this domain.

Association

- describes: Domain[1]

  Each *describes* represents a particular domain. This representation defines that every design model should describe a particular domain. This is an ordered association from *Design Model* to *Domain*. *Design Models* can describe just a domain.

d. User

User is a person who interprets design models to get an understanding of the domain (Lange, 2007a). A user can interpret one (or more) design model and compose design models for any particular purpose. Additionally, the user detects and resolves inconsistencies that arise from the compositions. Typical categories of users are software developers and researchers.

Association.

- composes: Design Model[2..*]

  Each *composes* represents the instance of a composition that is realized by *User*. A *composes* declares that there may be composition between instances of two (or more) design models. A composition is a tuple with two (or more) design models for each end of the association, where each design model is an instance of the type of the end (i.e., *Design Model*). This is an ordered association from *User* to *Design Model*. Users can compose tow (or more) design models.

- detects: Inconsistency[*]

  Each *detects* represents the detection of inconsistencies by the *User*. A *detects* specifies that there can be detection of inconsistencies when a *User* realizes composition of design models. This is an ordered association from *User* to *Inconsistency*. *User* can detect anything to many inconsistencies.

- resolves: Inconsistency[*]

  Each *resolves* represents the resolution of inconsistencies by *User*. A *resolves* specifies that there can be resolution of inconsistencies when a *User* realizes composition of design models. This is an ordered association from *User* to *Inconsistency*. *User* can resolve from none to many inconsistencies.

- interprets: Design Model[1..*]

  Each *interprets* represents the interpretation of design models by *User*. A *resolves* specifies that there can be resolution of inconsistencies when a *User* realizes composition of design models. This is an ordered association from *User* to *Inconsistency*. *User* can interpret no or many inconsistencies.

- applies: Composition Technique[*]

  Each *applies* represents the application of model composition technique to compose design models by *User*. A *applies* specifies that there can be the use of composition technique when a *User* realizes composition of design models. This is an ordered association from *User* to *Composition Technique*. *User* can apply no or many composition techniques.

e. Conflict

Conflict is the concept that represents the contradictions between different design models to be composed. Since *User* tends to assign contradicting values to the properties of the *Design Models* (Section 3.4). Conflicts arise why the design models receive conflicting changes. These contradictions happen when the ordered association *composes*: Design Model [2..*] from *User* to *Design Model* is instantiated. Thus, conflict is a derived concept from the association *composes*.

For example, a developer defines that a class is abstract (i.e., *isAbstract* = true) while another developer specifies that the same class is concrete (i.e., *isAbstract* = false). *User* should grasp and tame these conflicts in order to able to produce an intended design model.

Association

- Without a directed relationship

f. Inconsistency

Inconsistency is the concept that represents the defects found in the output composed model (Section 3.4). It usually arises because *User* tends to incorrectly resolve the *Conflicts*. For example, developers can incorrectly tame the conflict whether a class should be abstract or not.

Association

- affects: Design Model[*]
  Each *affects* consists of problems jeopardizing quality notions of the *Design Model*. When the *affects* takes place implies to say that an output composed model and the output intended model do not match ($M_{CM} \neq M_{AB}$). This is an ordered association from *Inconsistency* to *Design Model*.

g. Design Characteristic

A design characteristic is the concept that illustrates the strategies used by developers to structure design models such as coupling and cohesion. Design characteristics are used to improve, for example, the capability of design models to be (more straightforwardly) composed. The design characteristics are also used as indicators (Martin, 2003) of prone to problems. An example of this design characteristic is model stability (Section 2.6).

Association

- influences: Design Model[*]
  Each *influences* represents that the design characteristics modify the manner of the design model is created or can act as an indicator such as stability. This is an ordered association from *Design Characteristic* to *Design Model*.

h. Composition Technique

Composition technique is the concept that represents the technique used by developers to compose the design models. Examples of these techniques are Epsilon and IBM Rational Software Architect. A model composition technique defines a set of operators that are used to manipulate the input model elements. More detail about this concept can be found in Section 2.4.

Association

- Without a directed relationship.

3.3.4. Quality Notions

After presenting the basic elements of the quality model, we discuss the quality notions associated somehow with each one of them. In our study, quality notions can be seen as non-functional requirements used to evaluate the effort of a composition. Our quality model focuses on seven quality notions, namely syntactic, semantic, social, effort, application, detection, and resolution notions. We propose four quality notions effort, application, detection, and resolution notions. Each of them captures a fundamental dimension of quality related to model composition activities. The other quality notions are tailored from previous works (Lindland et al., 1994; Krogstie, 1995; Lange, 2007a). Lindland (Lindland et al., 1994) proposed three quality notions — i.e., syntactic, semantic, and pragmatic ones. Krogstie (Krogstie, 1995) and Lange (Lange, 2007) add the social and communicative quality notion to the Lindland's quality notions, respectively. All these notions were tailored to the context of evaluation on model composition effort. These extensions are discussed as follows:

- **Syntactic Quality (1).** Krogstie originally proposed this quality notion (Krogstie, 1995) to represent the correctness of design models produced by a design modeling language (Lange, 2007a). If a design modeling language is not properly used, then some syntactic inconsistencies may emerge. This quality notion is relevant to our quality model as syntactic inconsistencies can also arise during model compositions (Mens, 2002). Developers need to be concerned with checking the syntactic consistency of the output composed model. The degree of correctness should be evaluated in terms of

the presence or absence of inconsistencies of the composed model. In other words, syntactic quality is computed by measuring the inconsistencies resulting from conflicts between the input models. For this, inconsistency metrics (Farias et al., 2008a) are used. This notion helps developers to identify the number of deviations in the output composed model with respect to the language specification. This quality notion is studied in empirical studies presented in Chapters 5, 6, and 7.

- **Semantic Quality (2).** This notion deals with the degree of correspondence between the design model and the problem domain (Lange, 2007a). If the semantics of the model elements are affected, the main purpose of use of the design models — i.e., communication between the team members can be damaged. Thus, developers and designers need to be concerned with checking the meaning of the model elements in the output composed model. In a similar way to the syntactic notion, the degree of correctness should be evaluated in terms of the presence or absence of inconsistencies. That is, semantic quality is calculated by measuring the conflicting correspondence between the design model and the problem domain (Chapter 2). This inadequate representation may occur by two reasons (but not limited to): (i) the inability of the developers to represent the concepts and the relationship of the domain, and (ii) the inaccuracy of the composition techniques that inadequately manipulate the semantics of the model elements (Mens, 2002). To quantify these semantic inconsistencies, some metrics defined in (Farias et al., 2008a) are used. This quality notion is studied in Chapters 5, 6, and 7.
- **Social Quality (3).** Design models are essentially used to communicate design decisions between the software development teams (Larman, 2004; Dobing et al., 2006). If there is a disagreement between the interpretations of the design models, the communication between the developers is severely harmed. With this in mind, researchers should elaborate studies in order to understand the effects of the misinterpretations on the implementation. For example, if the degree of misinterpretations is high, the diverging understanding may be converted into defects in code. These two reasons can in fact damage the interpretation of the output composed models. The social quality notion, therefore, matches the interpretations of the developers and checks the degree of disagreement between them. Therefore, the focus of

such social notion is to evaluate the threats to the agreement of interpretations of the design models by the developers. The evaluation aims at comprehending how the misinterpretation may be motivated by (but not limited to): (1) the inadequate layout of the model elements caused by the incorrect positioning of the model elements, and (2) the representations of the constructs of the current modeling languages are not friendly. The method described in (Lange, 2007a) to measure the degree of the misinterpretations is used. This quality notion is studied in Chapter 6.

- **Effort Quality (4).** This quality notion addresses the effort of producing an output intended model. It is expected that the practices of applying a composition technique, detecting, and resolving inconsistencies are not effort-consuming tasks. However, they will inevitably require extensive effort to produce an indented model in several cases. Therefore, this quality notion deals with the cost of obtaining an expected output model. This quality notion is studied in Chapters 5, 6, and 7. The next three quality notions refine this quality notion by addressing the easiness (or difficulty) in the tasks of applying composition techniques, detecting, and resolving composition inconsistencies.
- **Application Quality (5).** This notion represents the applicability of a particular model composition technique. In other words, it addresses the ease of producing an output composed model by applying a model composition technique. Ideally, developers expect to be able to effortlessly compose design models by using either heuristic-based or specification-based composition techniques. However, two difficulties make the practice of applying composition techniques not trivial. The first difficulty arises from the inherent challenge of making use of different categories of model composition techniques. Each of them imposes different burdens on software designers. For instance, developers need to manually specify rules in order to define the equivalence and composition relations between the input model elements. On the other hand, they may also compose the models using heuristic-based composition techniques. The second difficulty consists of the accidental problems that emerge from the practice of bringing design models together. Usually developers need to resolve

conflicting changes performed in parallel. This quality notion is studied in Chapter 5.

- **Detection Quality (6).** After producing an output composed model, developers should review it to assure its correctness. That is, developers should check if some inconsistency was produced as the result of the composition. When inconsistencies arise, developers should be able to quickly localize them. If the detection of inconsistencies is hard, then the assurance of the correctness of the models may also be hard. Unfortunately, the localization of inconsistencies is not always a trivial task. This can be explained by at least two reasons (but not limited to): (i) the composition techniques cannot often help developers to automatically detect all kinds of inconsistencies. Since, the meanings of the model elements are rarely represented in a formal way; and (ii) developers cannot understand specific inconsistencies, mainly semantic inconsistencies, given the problem at hand and their knowledge about the meaning of the model elements. With this in mind, researchers should study the degree of difficulty that developers face to localize inconsistency so that the consistency of the output composed model can be assured. In particular, it is expected that researchers provide a clear understanding about the effort to detect inconsistencies in practice. Therefore, the focus of this quality notion is on evaluating the cost to localize inconsistencies in the output composed model. This evaluation is important because it allows researchers to understand, for example, if design modeling languages such as UML and aspect-oriented modeling can significantly affect the detection effort, or if alternative composition techniques such specification-based or heuristic-based ones can influence the detection. This quality notion is studied in Chapters 5 and 6.
- **Resolution Quality (7).** After detecting inconsistencies, developers should resolve them in order to transform the output composed model into the output intended model. That is, developers should invest some additional effort (apart from producing the output composed model) trying to find some solution to the inconsistencies already localized. Otherwise, the practice of composing design model can become prone to inconsistencies or even require more effort than it would be expected. This additional effort can make the practice of assuring the consistency of the composed models

difficult and costly. Unfortunately, the resolution of inconsistencies is not always an easy task. This can be explained by the lack of accuracy of the composition techniques to understand the meaning of the model elements and the incapability of the developers to find an adequate solution to the inconsistencies (Mens, 2002). This notion, therefore, addresses the degree of difficulty to resolve inconsistencies. This difficulty of resolving inconsistency can be calculated considering the time invested to resolve them or even the number of activities that developers should perform. Moreover, it copes with the inherent and accidental difficulties of solving composition anomalies e.g., syntactic and semantic inconsistencies. The first complexity arises from the need to reason and then make decision about how to tame inconsistencies. The accidental difficulty is caused by the modeling technique such as OO or AO modeling used to represent the design models and by the manner as they are structured i.e., more modularized or not. This quality notion helps understanding the difference between how the developers think about inconsistency resolution and how in fact they resolve inconsistencies. This quality notion is studied in Chapters 5 and 7.

Table 3 describes how the quality notions that are addressed through the empirical studies presented in the next chapters.

Chapter	Quality Notion	Description
3	all quality notions	Definition of the quality model for model composition effort
4	effort, application, detection, resolution, syntactic, semantic	Empirical studies address the quality notions in practice
5	effort, detection, social, syntactic, semantic	A controlled experiment is performed to investigate the five quality notions
6	effort, resolution, syntactic, semantic	Quasi-experiments were realized to study the four quality notions
7	all quality notions	All quality notions are discussed based on the series of empirical studies performed

Table 3: Definition of chapters where quality notions are investigated

3.3.5. Levels of the Quality Model

The quality model is organized following a 4-level specification pattern. To define the quality model with levels, we need to consider: (1) when model composition is used i.e., in which phase of the development process it is used; (2) why model composition is applied i.e., the purpose of using the model composition; (3) what can be used to characterize model composition i.e., the characteristics that are directly related to model composition; and (4) how such characteristics can be quantified i.e., the definitions of rules and metrics used to measure the characteristics. These four levels are hierarchically organized and this fine-grained partitioning allows separating concerns across layers of abstractions, and providing flexibility to future studies so that they may extend the quantity model.

This section, therefore, brings forward the levels of the quality model and the concepts that belong to the levels. Recall that this thesis attempts to investigate the effort that developers invest to use model composition in the context of design model evolution; however, that does not mean that the model cannot be tailored to other contexts. The model has four levels (based on (Lange, 2007a)), which are described as follows:

a. Level 1: Use of Composition

The top level of our quality model describes the high-level use of model composition in practice. These uses are:

- **Development:** developers use model composition to incrementally create the design models before the implementation phase. This use combines quality characteristics that concern the composition before the design model of a system has been completely finished.
- **Evolution:** developers make use of composition techniques to evolve design models. This use combines quality characteristics that concern the product when it is changed.

b. Level 2: Purposes of Composition

The second level defines the purposes of using that model composition is applied. These purposes are directly related to the purposes discussed in Section 2.1. In practical terms, it specifies why developers use composition. Thus, we identify three purposes of using that are described as follows:

- **Analysis:** Users identify overlapping parts between the model to-be composed. This allows them to analyze possible conflicting changes that are strong candidate to become inconsistencies.
- **Change:** Users essentially use composition techniques to add, modify, remove, or even refine model elements of some existing design model.
- **Reconciliation:** Users use the resource of model composition techniques to reconcile contradicting changes (Clarke, 2001).

c. Level 3: Characteristics of Composition

The third level of our quality model contains the inherent characteristics of the design model and model composition technique. The characteristics are described in Table 4. According to the distinction between the characteristics of

Characteristic	M	T	Description
Effort		X	The effort to execute f, diff, and g.
Complexity	X		The degree of difficulty to understand a model (Lange, 2007; Feton et al., 1994).
Modularity	X		The manner by which a software system can be systematically structured and separated such that it can be understood in isolation (Parnas, 1972).
Stability	X		The degree of changes that a module suffers given a need of change i.e. a module is stable if its design characteristics have a low variation (Kelly, 2006).
Size	X		The number of model elements in a design model
Correctness	X		The extent to which a design model is complaint with a reference design model.
Consistency	X		The extent to which no inconsistency is contained (Easterbrook et al., 1996)
Communicativeness	X		The degree of facility to communicate and assimilate content (Boehm et al., 1978; Lange, 2007).

Table 4: Characteristics of design models

the design model and the characteristics of the model composition technique, we indicate for each characteristic whether it is a characteristic of the design model (column M) or a characteristic of the model composition technique (column T). Some characteristics are defined for both design model and composition technique.

The composition effort that is applied to exclusively to the model composition is characterized by the effort to apply the composition techniques ($f(M_A,M_B)$), to detect ($diff(M_{CM}, M_{AB})$) and resolve inconsistencies ($g(M_{CM})$). With this in mind, the characteristics (in Table 4) describe the design models and the composition technique.

d. Level 4: Metrics and Rules

The fourth level defines how the aforementioned characteristics are quantified. To allow the quantification of these characteristics, a suite of metrics and rules were used. Rules are special cases of metrics; being usually mappings of some observations from the empirical domain to a binary value: true or false (Wust, 2011; Lange, 2007a). These rules evaluate and measure design models, mainly checking well-formed rules and design rules. Two practical examples of well-formed rules would be "Abstract class must not be instantiated" and "Abstract class must not have a concrete class as superclass." Note that the consistency of the design model is affected if these two rules are not assured.

In our empirical studies, several elements appear in the models, depending on the types of diagrams used. Class, interface, and component and examples of elements in component diagrams, which were used in several studies of this thesis. Metrics can be defined to quantify these elements. In order to illustrate these specific metrics: (i) Table 5 describes the metrics for classes, (ii) Table 6 shows the metrics for interfaces, and Table 7 describes the metrics for components. These tables also describe the relations between the characteristics (level 3) and the metrics and rules (level 4) are specified.

The metrics and rules are defined in previous work (Chidamber & Kemerer, 1994; Lorenz & Kidd, 1994; Lee et al., 1995; Martin, 2003; Lorenz, 1994; Chidamber et al., 1998; McCabe; 1976). Although these metrics are often used in previous research, we do not claim that this list of metrics and rules is complete. These metrics were chosen because they are well-known indicators to quantify

model characteristics, and are often supported by robust measurement tools, such as SDMetrics (Wust, 2011).

After presenting the concepts and describing the three levels, Figure 6 describes the three top levels of the quality model: *Use*, *Purpose*, and *Characteristic*. The fourth level Metrics and Rules and the relations to level three are depicted in Table 5, Table 6, and Table 7. Note that a checkmark indicates which characteristic of level three is related to the metric or rule in level four. In Figure 6, the arrows indicate relations between two concepts of different levels. The arrows can be interpreted as follows: a lower level concept is part of all higher-level concepts to which it is related by an arrow, and a higher-level concept contains the related lower level concepts. The interpretation of the relations is that a concept in a lower level in the quality model contributes to the related concepts of the higher level.

Metric	Characteristic	Description
NAttr	SI	The number of attributes in the class.
NOps	SI	The number of operations in a class.
IFImpl	CO, MO	The number of interfaces the class implements.
NOC	CO, CM	The number of children of the class.
NDesc	CO	The number of descendents of the class.
NAnc	CO	The number of ancestors of the class.
DIT	CO, CM	The depth of the class in the inheritance hierarchy.
OpsInh	CO	The number of inherited operations.
AttrInh	CO	The number of inherited attributes.
DepOut	CO, MO, CM	The number of elements on which this class depends.
DepIn	CO, MO, CM	The number of elements that depend on this class.
ECAttr	MO	The number of times the class is externally used as attribute type.
ICAttr	MO	The number of attributes in the class having another class or interface as their type.

SI: size, CO: complexity, MO: modularity, and CM: communicativeness

Table 5: Metrics for class

Metric	Characteristic	Description
NOps	SI	The number of operations in the interface.
Assoc	CO	The number of elements the interface has an association with.
NAnc	CO	The number of ancestors of the interface.
NDesc	CO	The number of descendents of the interface.
NOps	SI	The number of operations in the interface.
ECAttr	CO	The number of times the interface is used as attribute type.
ECPar	CO	The number of times the interface is used as parameter type.
Assoc	CO	The number of elements the interface has an association with.
NDirClients	CO	The number of elements directly implementing the interface.
NIndClients	CO	The number of elements implementing a descendent of the interface.
NAnc	CO, MO	The number of ancestors of the interface.
NDesc	CO, MO	The number of descendents of the interface.

SI: size, CO: complexity, MO: modularity, CM: communicativeness

Table 6: Metrics for interface

Metric	Characteristic	Description
NOps	SI	The number of operations of the component.
NComp	SI	The number of subcomponents of the component.
NPack	SI	The number of packages of the component.
NCCmp	SI	The number of classes of the component.
NIntCmp	SI	The number of interfaces of the component.
Connectors	CO	The number of connectors owned by the component.
ProvIF	CO, MO	The number of interfaces the component provides.
ReqIF	CO, MO	The number of interfaces the component requires.
DepOut	CO, MO, CM	The number of outgoing dependencies.
DepIn	CO, MO, CM	The number of incoming dependencies.
AssocOut	CO, CM	The number of associated elements via outgoing associations.
AssocIn	CO, CM	The number of associated elements via incoming associations.

SI: size, CO: complexity, MO: modularity, CM: communicativeness

Table 7: Metrics for components

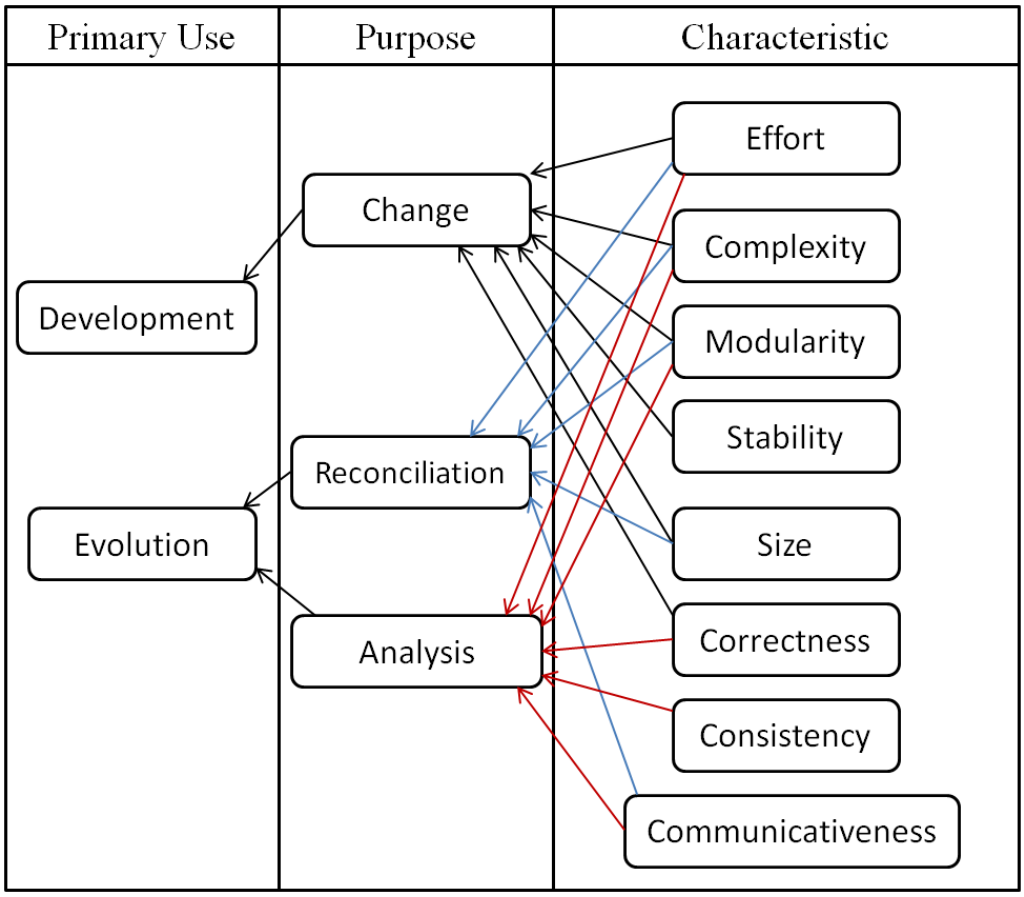

Figure 6: The purposed quality model (based on (Lange, 2007a))

3.4. Concluding Remarks

Developers need to evaluate model composition effort. However, the evaluation without any quality model is not a trivial task (Basili & Lanubile, 1999) as usually developers have no previous knowledge or experience about empirical evaluations of model composition. This chapter, therefore, presents a quality model for model composition effort. It is intended to help researchers and developers to carry out empirical studies of model composition.

The proposed model extends three previous quality frameworks for conceptual models proposed by Lindland (Lindland et al., 1994), Krogstie (Krogstie, 1995), and Lange (Lange, 2007a). The model is organized in a four-level structure. The first level defines the context where model composition is

used in practice, being development and evolution the two usage scenarios proposed and investigated. The second level refers the purposes of using model composition. We identify and evaluate model composition for three purposes of using: change, analysis, and reconciliation. The third level refers to the characterization of the elements involved in model composition: the models and model composition techniques. That is, it considers the artefacts and the techniques responsible for manipulating them. The fourth level aims at quantifying the elements identified in the third level. To this end, metrics and rules are used.

By defining this quality model, we can solve the problems presented in Section 4.1 First, researchers and developers can make use of a unifying framework for the evaluation of model composition. As a result, the findings resulting from multiple studies can be compared, or even checked whether they are valid in a specific context or not. Finally, the use of the quality model serves as a reference frame for structuring empirical studies of model composition. In this context, the quality model guides all empirical studies performed throughout the thesis.