Eindhoven University of Technology

MASTER

Quality metrics for ASOME data models

Zhang, H.

Award date:
2018

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

EINDHOVEN UNIVERSITY OF TECHNOLOGY

MASTER'S THESIS

Quality metrics for ASOME data models

Author:
H. ZHANG

Supervisor:
Prof.dr.ir J.F. GROOTE
MSc. J. MARINCIC

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in Computer Science and Engineering

in the

Department of Mathematics and Computer Science

November 5, 2018

Contents

Basic Mathematical Concepts

Some basic mathematical concepts are presented, which help us to describe models and metrics in this thesis.

Definition 0.1 (#X). Let X be a finite set. $\#X$ denotes the total number of the elements of the set X.

Definition 0.2 (Transitive Closure). Let X be a finite set and $R \subseteq X \times X$ be a binary relation. The transitive closure R^* of R is inductively defined for all $x, y, z \in X$:

- $x\,R\,y \implies x\,R^*\,y$.
- $x\,R^*\,y \wedge y\,R^*\,z \implies x\,R^*\,z$.

Definition 0.3 (Data Types). The following table gives data types which are involved in the ASOME data models:

Data type	*notation*
Positive number $(1, 2, 3, \ldots)$	$\mathbb{N}^+$
Natural numbers $(0, 1, 2, \ldots)$	$\mathbb{N}$

TABLE 0: Data Sorts with *notations*

Chapter 1

Introduction

1.1 Problem Statement

Software engineering has gained increasing importance for all fields of society and individuals. The software development causes not only the complexity of software products to grow tremendously but also the maintenance costs to increase significantly [60, 1]. This increasing demand in software engineering promotes several new software development methodologies and MDE (Model Driven Engineering) is one of them. Two main benefits of MDE are productivity improvement and flexibility improvement [49]. However, there is a lack of methods, especially is the software metrics, to evaluate and guarantee the quality of the MDE products.

In this thesis, we will look how we can enhance the quality assessment of data models designed in ASML with software metrics.

1.2 Goal and Questions

To address the problem statement, we formulate an overall goal:

Design software metrics to assess the quality of data models designed in ASML.

To achieve the overall goal, we have to answer the following questions:

Q1: *How can the quality of data models be decomposed into quality attributes?*

Q2: *Can we reuse the existing software metrics to assess the quality of data models?*

Q3: *What metrics can be defined for data models?*

Q4: *What metrics are effective as quality attributes of data models?*

1.3 Thesis Organization

In this thesis, we first present the preliminaries in Chapter 2. Next, we discuss the quality problem related to our design in Chapter 3. Chapter 4 contains the metrics we define and formalized by using the literature study, interview study and document analysis. In Chapter 5, we evaluate these metrics and discuss their performance in our survey experiment. Finally, Chapter 6 presents a conclusion.

Chapter 2

Preliminaries

In this chapter, preliminary material is presented. Section 2.1 provides an overview of model-driven engineering. Section 2.2 gives an overview of domain-specific models in ASML and the definition of the models is given in the Section 2.3.

2.1 Model-Driven Engineering

The challenges in developing increasingly complex systems motivated the introduction of MDE (Model Driven Engineering), which uses models to represent the behaviors or other aspects of complex systems [55]. J.D. Haan uses an interesting metaphor to explain the differences between the traditional programming method and the MDE method [23]. In the Figure 2.1, traditional programming can be described as a way of building a house brick by brick while MDE purposes to use a model to *generate* a house.

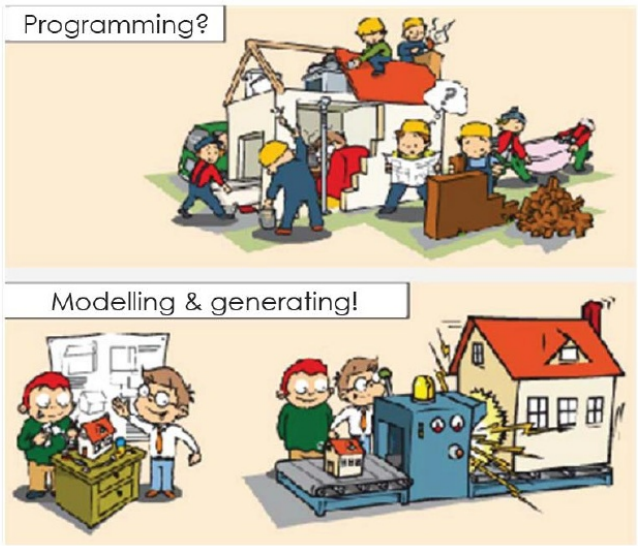


FIGURE 2.1: A metaphor for Model Driven Engineering (MDE)[23]

Many organizations such as IBM and Object Management Group (OMG) have been focusing on the development of MDE. The best known MDE approach is MDA (Model-Driven Architecture) provided by OMG [22].

2.1.1 Domain-specific model

In MDE, models are typically domain-specific, which means they are tailored to certain domains or specific goals. We call these models DSMs (domain-specific models).

A complex system is usually specified with multiple DSMs. These models refer to each other and can be combined to represent the whole system. They are executable which means that code can be generated from them. A DSM is constructed in a Domain-specific language, or DSL for short.

2.1.2 Domain-specific language

A. v. Deursen defines the DSL (Domain-specific language) as:

a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [11].

Similar to a DSM, a DSL is also tailored to a certain domain. Actually, using DSLs is a typical method to create or to modify DSMs. A sound DSL construction contains an abstract syntax, one or more concrete syntax descriptions, mappings between abstract and concrete syntax, and a description of the semantics [59], in which abstract syntax is the key connection between DSL code and a DSM. The designers of a DSL often use metamodels to define the abstract syntax. Models created by using DSLs have to conform to the corresponding metamodels.

2.1.3 Domain-specific model representation

There are two typical ways to represent a DSM:

- **Text:** A DSM can be represented as a textual format. Since using DSLs is one method to create DSMs, the corresponding DSL code can also be treated as the textual representations of the models.
- **Diagram:** In order to improve the readability of a model, some developers prefer to use graphical tool to design and view DSMs.

A simple example is given in Section 2.3.4.1, where Figure 2.3 is the textual representation of the model and Figure 2.2 is its visual representation.

Some researchers use DSVL (domain-specific visual language) and DSTL (domain-specific textual language) or textual DSL to differentiate the textual and visual representations [21, 52].

2.1.4 Quality of Model-Driven Engineering

Quality is always an important factor for all software engineering approaches. In general, MDE techniques and tools allow specifying constraints on the models and validating the model against them. This helps to find errors early in the design process [50]. These techniques are able to check the correctness of DSMs but cannot evaluate the quality of the models. In Chapter 3, we explain the quality problem further.

2.2 Domain-Specific Models at ASML

ASML has developed various model-based development environments. In this thesis we study one of them: ASOME (ASML SOftware Modeling Environment). ASOME is developed to provide an integrated modeling environment allowing software engineers to define specific models for ASML domains. A part of the ASOME environment is the ASOME data modeling language for modeling data aspects of the

system. The ASOME platform supports both textual and visual representations for an ASOME data model. A simple example of an ASOME data model is given in Section 2.3.4.1.

ASOME data models have been used in several projects. These models are reviewed for correctness and quality. There are modeling guidelines and some design constraints are preset in the ASOME platform [61]. However, there is no software metrics to guarantee and evaluate the quality of the ASOME data models.

2.3 ASOME Data Models

To study ASOME data models and find some methods to evaluate their quality, we should understand the basics of the ASOME data model language. In this section, the basic definitions of an ASOME data model is given. Please note that these definitions only contain the most important components. Less important parts are omitted.

2.3.1 Basic elements: type, enumeration, constant and multiplicity constant

In the following sections, we present the definitions of each basic element in detail.

2.3.1.1 Type

Types, also called primitive types, are used to specify the primitive data types such as integer, boolean and string. More precisely:

Definition 2.3.1.1. (type). A type $typ \in Typ$ where Typ is a set of type names. We use TYP to denote the set of all types.

2.3.1.2 Enumeration

Enumerations are used to specify a list of elements represented as enumeration literals. The definition of the enumerations is:

Definition 2.3.1.2. (enumeration). An enumeration enu is a two-tuple $enu = (n, EL)$ where

- n is the name of the enumeration.
- EL is a set of enumeration literals.

The set of all enumerations is denoted as ENU.

2.3.1.3 Constant and multiplicity constant

Constants are used to specify some values that cannot be altered.

Definition 2.3.1.3. (constant). Let $Typ \subseteq TYP$ be a set of types. A constant con is a three-tuple $con = (n, t, Val)$ where

- n is the name of the constant.
- $t \in Typ$ is the type of the constant.

- *Val* is the value of the constant, where *Val* has to conform to the type of the constant.

The set of all constants is denoted as *CON*.

A multiplicity constant is a special constant which is used to specify a multiplicity end. The concepts of multiplicity and multiplicity end are further explained in Section 2.3.2. The definition of a multiplicity constant is given below.

Definition 2.3.1.4. (multiplicity constant). A multiplicity constant *mc* is a two-tuple $mc = (n, Val)$ where

- n is the name of the multiplicity constant.
- $Val : \mathbb{N}^+$ is the value of the multiplicity constant.

The set of all multiplicity constants is denoted as *MC*.

2.3.1.4 The definition of basic element

Types, enumerations, constants and multiplicity constants are the basic elements of an ASOME data model. Formally:

Definition 2.3.1. (Basic Element). A basic element *be* can be a type, constant, multiplicity constant or an enumeration. The set of all basic elements is denoted as $\mathcal{BE}$, where $\mathcal{BE} = TYP \cup CON \cup MC \cup ENU$.

2.3.2 Class: entity and value object

An entity and a value object are two main concepts to describe the objects in an ASOME data model. Before we define the entity and the value object, we should first explain what an attribute is.

2.3.2.1 Attribute

An attribute defines objects that can be attached to instances of an entity or a value object in an ASOME data mode. An attribute needs a multiplicity property, which is an indication of how many objects may participate in the corresponding entity or value object. A multiplicity can be either ordered or unordered. More precisely:

Definition 2.3.2.1. (multiplicity). An ordered multiplicity is defined as $[a..b]$ and a unordered multiplicity is defined as $\{a..b\}$, where a is called the lower bound and b is called the upper bound. The lower and upper bounds a and b satisfy:

- $a \in \mathbb{N}$, and
- $b \in (\mathbb{N}^+ \cup \{*\})$, and
- if $b \neq *$, then $a \leq b$,

where the star $*$ represents the infinite upper bound. The set of all multiplicities is denoted as $\mathcal{M}$. For simplicity we write $a..a$ as a. Specially, since $*$ implies an upper bound value of more than 1, we can also simplify $0..*$ as $*$.

The definition of an attribute is as follows:

Definition 2.3.2.2. (attribute). Let $Typ \in TYP$ be the set of types in an ASOME data model and $Enu \in ENU$ be the set of enumerations. An attribute att is a three-tuple $att = (n, t, m)$ where

- n is the name of the attribute.
- $t \in Typ \cup Enu$ is the type of the attribute.
- $M \in \mathcal{M}$ is the multiplicity of the attribute.

The set of all attributes is denoted as ATT.

2.3.2.2 Entity

In an ASOME model, an entity is used to describe a category of objects that contains the same identity. The formal definition is a follows:

Definition 2.3.2.3. (entity). Let $Att \in ATT$ be the set of attributes of an ASOME data model. An entity ent is a three-tuple $ent = (n, A, M)$ where

- n is the name of the entity.
- $A \in Att$ is a set of attributes.
- $M \in \mathcal{M}$ is the multiplicity of the entity.

The set of all entities is denoted as ENT.

Let ent be an entity. The entity ent can be mutable or immutable. We use ent_{mut} to indicate ent is mutable and use ent_{immut} to indicate ent is immutable. An immutable entity can never be modified after creation while a mutable entity can be updated after creation.

Similarly, the entity ent can be data type, control type or algorithm type. We use ent_{dat}, ent_{ctr} and ent_{alg} to represent them.

2.3.2.3 Value object

Value objects are used to describe a category of objects that do not have identity. The definition is as follows.

Definition 2.3.2.4. (value object). Let $Att \in ATT$ be the set of attributes in an ASOME data model. A value object vo is a two-tuple $vo = (n, A)$ where

- n is the name of the value object.
- $A \in Att$ is a set of attributes.

The set of all value objects is denoted as VO.

2.3.2.4 The definition of class

Entities and value objects are the basic building blocks of an ASOME data model. To simplify the representation of elements in an ASOME model, we use a class to stand for either a value object or an entity. The definition is as follows.

Definition 2.3.2. (Class). The set of all classes is denoted as $\mathcal{C}$. Obviously, $\mathcal{C} = ENT \cup VO$.

Let $ent = (n, A, M)$ be an entity. We use $ent.A$ to represent the set of attributes of the entity ent and $ent.M$ to represent the multiplicity of the entity ent. Similarly, let c be a class. We use $c.A$ to represent the set of attributes of the class c.

2.3.3 Relation: association, composition and specialization

A relation is a general term to describe a specific connection between different classes in an ASOME data model. Each relation has two relation ends attached to one of the classes. In an ASOME data model, relations are all unidirectional. We use an arrow to present the direction of a relation in an ASOME diagram. To simplify the representation of a relation with unidirection, we use

- a pair $\langle a, b \rangle$ to represent an element of a unidirectional relation from the class a to the class b.

Some relations also have multiplicity properties, which indicates how many entities or value objects may participate in a given relation end. For relations with the multiplicity property, we use

- a pair $\langle (a, m_a), (b, m_b) \rangle$ to denote an element of a unidirectional relation, where $a, b \in \mathcal{C}$ and $m_a, m_b \in \mathcal{M}$ are the multiplicity symbols at the ends of the classes a and b respectively.

Relations in ASOME data models include three sorts, which are *Association*, *Composition* and *Specialization*. The following are the definitions of these three relations:

1. *association* An association is a unidirectional relation from a source entity to a target entity. Note that the source and target ends of an association are always entities in an ASOME data model. The definition is as follows:

   Definition 2.3.3.1. (association relation). Let $Ent \subseteq ENT$ be a set of entities. An association relation $r_{ass} \subseteq \langle (Ent \times \mathcal{M}) \times (Ent \times \mathcal{M}) \rangle$ is a unidirectional relation among entities. If $\langle (e_1, m_1), (e_2, m_2) \rangle \in r_{ass}$, then e_1 is called the source entity, e_2 is called the target entity and m_1, m_2 are called the source multiplicity and target multiplicity respectively. The association relation satisfies:

   - if $m_1 = \{a..b\}$ or $m_1 = [a..b]$, then $a = 0$.

   The source multiplicity m_1 cannot have a lower bound more than zero to keep the consistency of the repository.

2. *composition* A composition is a relation from a source class to a target value object. The composition relation is used to describe a containment of a value object in an entity or in another value object. Hence, the target of a composition is always a value object in an ASOME model.

   Definition 2.3.3.2. (composition relation). Let $C \subseteq \mathcal{C}$ be a set of classes and $Vo \subseteq VO$ be a set of value objects. A composition relation $r_{com} \subseteq \langle (C \times \mathcal{M}) \times Vo \rangle$ is a unidirectional relation among entities.

3. *specialization* A specialization refers to one entity that inherits (or extends) the identical attributes or methods of another entity.

   Definition 2.3.3.3. (specialization relation). Let $Ent \in ENT$ be a set of entities. A specialization relation $r_{spe} \subseteq Ent \times Ent$ is a unidirectional relation among classes. If $\langle e_1, e_2 \rangle \in r_{spe}$, then the entities e_1 and e_2 satisfy:

   - $e_1 \neq e_2$

2.3.4 The definition of an ASOME Data Model

Basic elements, classes and relations form the basic structure of ASOME data models. More formally:

Definition 2.3.3. (ASOME Data Model). An ASOME data model is a three tuple $ADM = (BE, C, R)$ where:

- $BE \in \mathcal{BE}$ is a finite set of basic elements.
- $C \in \mathcal{C}$ is a finite set of classes.
- R is the following set of relations: $R = \{r_{ass}, r_{com}, r_{spe}\}$.

As we stated before, an ASOME data model can be presented as a visual representation. We call the ASOME data model a diagram. An ASOME model diagram shows the model elements in a graphical notation. Table 2.1 gives a summary of the model elements and their icons in an ASOME model diagram.

Element name	Icon
type	T
enumeration	E
constant	C
multiplicity constant	M
entity	E
value object	VO
association	→
composition	◆→
specialization	—▷

TABLE 2.1: Model icons in an ASOME model diagram

Note that the ASOME platform allows model designers to hide some elements in an ASOME model or add some other elements from multiple external models.

2.3.4.1 An example of an ASOME data model

The following ASOME data model (Figure 2.2) is an example.

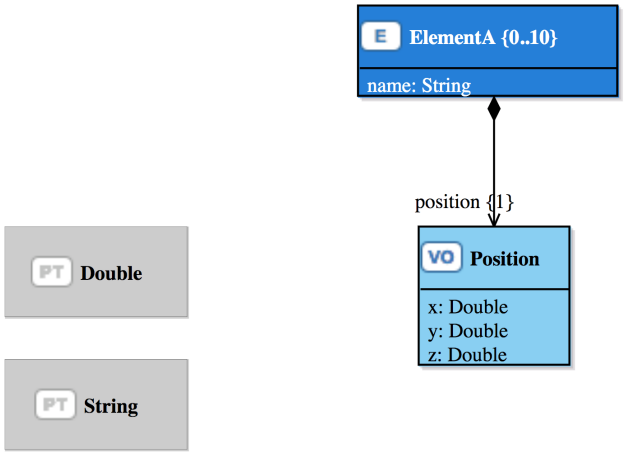


FIGURE 2.2: ASOME data model diagram example

```
Model BasicModel{

DomainInterface iDomain1DM {
        Entity ElementA [0, 10] {
            lifecycle : Constructable Immutable Undeletable Volatile
            attributes :
                name : BasicModel.iDomain1DM.String [1, 1] unordered;
                position : BasicModel.iDomain1DM.Position [1, 1] unordered;
        }
        Type String;
        Type Double;
        ValueObject Position {
            attributes :
                x : BasicModel.iDomain1DM.Double [1, 1] unordered;
                y : BasicModel.iDomain1DM.Double [1, 1] unordered;
                z : BasicModel.iDomain1DM.Double [1, 1] unordered;
        }
    }
}
```

FIGURE 2.3: ASOME data model code example

2.3.4.2 Superclasses and subclasses

Let $AMD = (BE, C, R)$ be an ASOME model and $c \in C$ be a class. Classes which specialize the attributes from the class c, are called specialization superclasses, and the classes which create the specialized attributes, are called specialization subclasses. More precisely:

1. the set of specialization superclasses of the class c is

$$Sup_{spe}(c, AMD) = \{c' \in C \mid c' \; r^*_{spe} \; c\}$$

2. the set of subclasses of the class c is

$$Sub_{spe}(c, AMD) = \{c' \in C \mid c \; r^*_{spe} \; c'\}.$$

Similiarly, the set of association superclasses of the class c is denoted as $Sup_{ass}(c, AMD)$ and the set of association subclasses of the class c is denoted as $Sub_{ass}(c, AMD)$.

Chapter 3

Quality

Software quality is a multidimensional concept. In this chapter, we provide an overview of the work relevant to the quality of ASOME models. We first analyse the quality in the MDA field in Section 3.1. Then we explain two ISO standards in Section 3.2. In Section 3.4, we analyse three quality attributes which are relevant for the quality of ASOME models. In Section 3.5, an overview of software metrics is presented.

3.1 Quality in MDA

Since models are the first-class citizens in MDE [55], developing high-quality systems depends on developing high quality models and performing model transformations or code generations that preserve quality and even improve it. P. Mohagheghi claimed that the quality of a model in MDE depends on four aspects [38]. They are:

1. the quality of the modeling platform which includes the quality of the corresponding DSL(s) and tools for model transformations or code generations;
2. the quality of the model itself and its modeling process;
3. the knowledge of the model designers which includes the problem they face and the modeling experience they have;
4. the quality assurance techniques applied to discover the design faults or weaknesses.

In this thesis, we mainly focus on the fourth method: using quality assurance techniques to discover the design faults or weaknesses. There are several common methods to check and ensure the quality in MDE [19] and here we briefly introduce two of them:

- **Model checking:** Model checking is a formal verification technique based on state exploration. When we give a state transition model and a requirement property, model checking algorithms will exhaustively explore the state space to determine whether the model satisfies the property.
- **Model validation:** In MDE, model validation checks the syntactic and semantic correctness of a model. More precisely, this approach checks whether the model conforms to its metamodel and semantic description. The model validation technique is usually integrated in the modeling platform.

Most quality assurance approaches in MDE are formal verification techniques and they are able to check the correctness of a model. However, when we want to evaluate the quality attributes like complexity and modifiability of a model, these approaches are not that useful. There are still some other techniques that can assess the quality of models such as model-based testing [65], auditing and software metrics. In this thesis, we mainly focus on using software metrics for assessing the quality of an ASOME model.

3.2 The ISO Quality Standard

In order to evaluate the quality of a software project, many quality models have been introduced [36, 3]. Among them, ISO 9126 and ISO 25010, provided by the international standard for the evaluation of software quality (ISO/IEC), have been widely used in the software industry [25, 24].

3.2.1 ISO 9126

In 1991, ISO 9126 was introduced to provide an explicit quality model which contains six quality attributes (also called quality characteristics) [25]. Every quality attribute can be further decomposed into several sub-attributes (also called factors or sub-characteristics) and every sub-attribute can be measured by several software metrics. The hierarchy structure concept [41] of the ISO 9126 standard is shown in Figure 3.1 and the corresponding quality attributes are summarized in Figure 3.2(a).

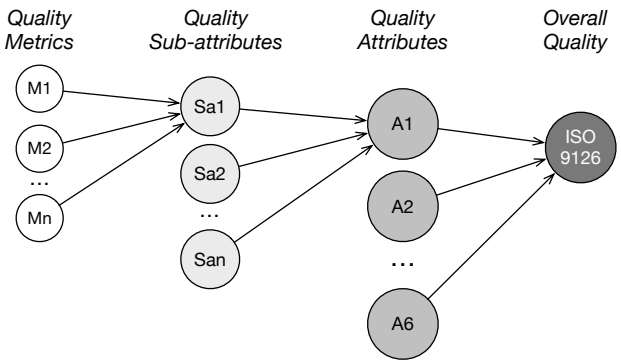


FIGURE 3.1: Hierarchy structure of ISO 9126 quality model

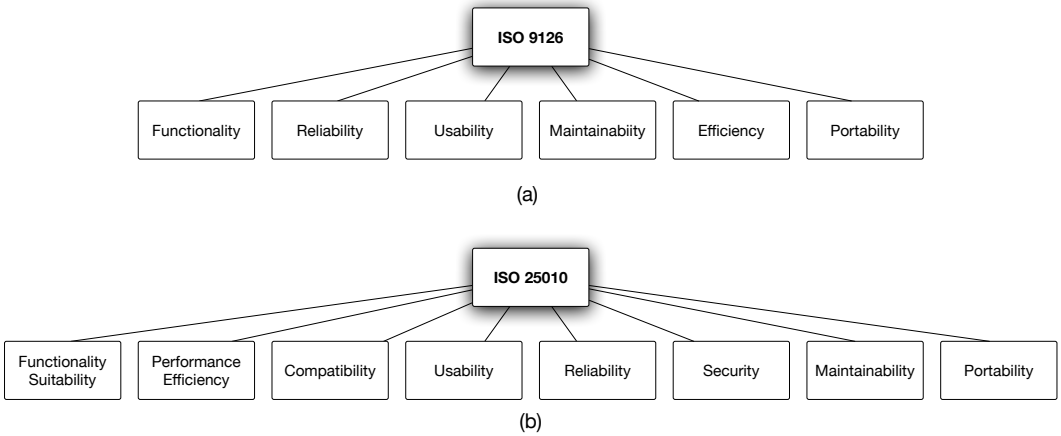


FIGURE 3.2: Quality attributes of ISO 9126 (a), Quality attributes of ISO 25010 (b)

3.2.2 ISO 25010

In 2007, ISO 25010 replaced the old standard ISO 9126 [24]. ISO 25010 is called a product quality model. The model re-defines the quality attributes. Figure 3.2(b) provides the new attributes of the new standard. ISO 25010 is also referred to as Software engineering- Software product Quality Requirements and Evaluation (SQuaRE).

3.2.3 Summary of the ISO quality models

ISO 9126 and 25010 have several things in common. Firstly, they are both hierarchical in structure since they decompose the concept of quality into a set of lower quality attributes [2]. Secondly, they are general-purpose quality models since they apply to any type of software product [37].

3.3 Other Quality Models

There are still some other quality models [57, 12, 4, 20]. We summarize these quality models and their quality attributes in tables in Appendix A.

3.4 Quality Attributes

In the previous section, we observed several quality attributes from different quality models. Since all these quality attributes have been defined in a general approach, we should analyze which attributes are truly useful for an ASOME data model and why these attributes are relevant for an ASOME data model.

3.4.1 Complexity

Complexity, as one quality attribute, has been studied extensively in the software development process. Many researchers studied this topic in different approaches [44, 46, 56]. Pippener suggests that the complexity of a system is based on the number of elements and the number of relations between the elements [44]. Similarly, we generally define the complexity of an ASOME data model as the degree of the effort

to understand the classes and relationships between these classes and the domain itself.

3.4.2 Maintainability

Software maintainability is important since maintenance takes approximately 75% of the cost related to a project [48]. ISO 25010 defines the maintainability as

the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.

In an ASOME data model, maintainability reflects the effort to modify the model. It is important to understand that the maintainability of an ASOME data model does not cover the maintainability of the code written around the data repositories.

3.4.3 Understandability

Understandability refers to the amount of effort that is requires to understand the purpose of an ASOME data model. As stated in Section 3.4.2, software maintenance is an expensive and highly demanding process. Activities which cause high maintenance cost are mainly coming from understanding the program, generating changes and testing the program. Among these, understanding an existing program is a major factor [7] and takes around 66% of the time [45]. What's more, understandability of a model also directly affect the model's complexity. Therefore, we pick understandability as the quality attribute we want to study.

3.5 Software Metrics

Software metrics have been studied extensively and have been proposed for measuring various kinds of software artifacts [14]. In MDA, there are many metrics which can be applied to model-based software artifact such as a UML class diagram and an ER (entity-relation) diagram. Some of these metrics did evolve from classical metrics, for example, the size metrics in Li and Henry's metrics [29] while some of them are modified by the metrics in OO (Object-Oriented) design such as CK metrics [9].

In order to design the proper quality metrics for ASOME data models, we use three approaches: literature study, interview study and document analysis. In the next chapter (Chapter 4), we explain these three approaches and corresponding metrics further.

Chapter 4

Metrics

4.1 Metrics - Literature Study

Since ASOME models are domain-specific, it is not possible to directly search for the metrics for an ASOME data model. However, we can learn from the metrics which are designed for some similar data modeling technique namely UML diagrams [34, 5], ER (entity relation) diagrams [31, 26], data models [33] and multidimensional schemas [8]. A selection of the metrics that could be used for ASOME data models and corresponding metrics design, is given in Table 4.1.

Reference	Overview
M. Genero (2008) [18]	A suite of objective metrics is provided to be used as indicators of the understandability of the ER diagrams.
M. Piattini (2001) [32]	The author provides a state of the art of measures for conceptual data models, where five different metrics proposals have been summarized and analyzed.
M. Genero (2001) [17]	A metrics suite is introduced for the structural complexity and maintainability of conceptual data models. An experiment is given as the empirical validation for the metrics suite.
M.Genero (2003) [5]	The paper introduces a metrics suite to predict the maintainability and structural complexity of UML class diagrams. A controlled experiment is conducted to gather the empirical data for the metrics validation.
Childamber and Kemerer (1994) [9]	Six design metrics are introduced for object-oriented design. The authors claim that using several of their metrics collectively helps managers and designers to make better design decision.
M.Marchesi (1998) [34]	A new metrics suite is introduced for UML use cases diagrams and class diagrams. An experiment is given to evaluate the metrics suite which includes three real projects.
S. Kesh (1995) [27]	This paper discusses a quality model and a methodology of metrics for evaluating the quality of ER models.
D.L.Moody (2005) [41]	This paper discusses several existing quality frameworks for conceptual data models and some software metrics in these frameworks.
D.L.Moody (1999) [40]	This paper describes a methods for clustering the quality of large data models. A set of principles and metrics has been defined for evaluating the quality of the models.
M. Serrano (2007) [51]	The paper proposes a set of metrics in order to predict the understandability of the conceptual schema used in the early stages of a DW (data warehouses) design. The second provides a empirical validation of the proposed metrics set.

A.M. Fernández-Sáez (2016) [15]	This paper focuses on how the LoD (level of detail) metrics of multiple UML diagrams influences the understandability and modifiability of source code.
Y.Lu (2016) [30]	This paper describes a case study for assessing software maintainability based on UML class diagram design and evaluate several metrics suites.
A. Nugroho (2008) [43]	This paper proposes a novel approach to measure LoD (level of detail) of a UML class diagram. LoD can be treated as the indicator of the defect density in the empirical analysis.
H. Eichelberger (2005) [13]	This paper introduces two layout metrics to reflect the magnitude or complexity which influences the size of elements in UML class diagrams.
P. Mohagheghi (2009) [39]	This paper presents on-going work on quality models and discuss the use of metrics for assessing the quality based on these quality models.
S. Stevanetic (2015) [53]	This article provides a systematic study of software metrics that measure the understandability of the higher-level architectural structures. The article selects sets of metrics based on different types of metrics.
T.J. McCabe (1976) [35]	This paper describes a graph-theoretic complexity calculation approach, which can be used as a software metric to measure program complexity.
F. Wu (2007) [62]	This paper presents a metrics suite to assess the structural complexity of components.
Zhou (2003) [63]	This paper proposes a metric, namely entropy distance based structure complexity metric. The metric is designed to predict structural complexity of UML class diagrams.

TABLE 4.1: A selection of the software metrics in literature study

In these articles, metrics have been designed for different types of models and for different different quality attributes. Also, they have different validation approaches. For example, M. Genero [18] provides a metrics set for ER diagrams and discusses its usefulness for understandability prediction. D.L. Moody [40] starts with the discussion of the existing quality models and then further analyzes some metrics related to them.

The summary of the quality attributes, scopes and validation approaches is given in Table 4.2. Note that we skip some articles that have not proposed their own metrics.

Reference	**Quality Attributes**	**Scope**	**Validation**
M. Genero (2008) [18]	understandability, structural complexity	ER diagram	experiment, survey
M. Genero (2001) [17]	maintainability, structural complexity	ER diagram	experiment
S. Kesh (1995) [27]	quality	ER diagram	
M. Serrano (2007) [51]	understandability	UML diagram	experiment, survey
A.M. Fernández-Sáez (2016) (2016) [15]	understandability, modifiability	UML diagram	experiment
A. Nugroho (2008) [43]	software defect density, maintainability	UML diagram	experiment

T.J. McCabe (1976) [35]	testability, maintainability		experiment, survey, case study
Childamber and Kemerer (1994) [9]	complexity	OO design, C++, Smalltalk	experiment, case study
M.Marchesi (1998) [34]	complexity	OO design, UML diagram, Smalltalk	experiment
M.Genero (2003) [5]	understandability, maintainability	ER diagram	experiment, survey
Zhou (2003) [63], DaKung	structural complexity	OO design, UML class diagram	case study

TABLE 4.2: A comparison of the software metrics in literature study

As already stated above, since the metrics in our literature study cannot be used directly, we make a selection of these metrics and re-interpret them for ASOME data models in Section 4.1.1 to 4.1.4. In Section 4.1.5, a summary of the metrics in the literature is given.

4.1.1 Chidamber and Kemerer's Metrics (1994)

Let $ADM = (BC, C, R)$ be an ASOME data model. The Chidamber and Kemerer's metrics is the following.

1. Max_{DST} is the maximum value of the depth of the specialization tree [1]. More precisely:
$$Max_{DST} = Max\{DST(c) \mid c \in C\}$$
$$\text{where } DST(c) = \begin{cases} 1 + DST(c'), & \text{if } c\ r_{spe}\ c' \text{, and} \\ 0, & \text{otherwise.} \end{cases}$$

2. $NOC(c)$ is the number of immediate classes (also called children) of c, where $c \in C$ and we call it as base class. The definition of $NOC(c)$ is:
$$NOC(c) = \#\{c' \in C \mid c\ r_{spe}\ c'\}.$$

The set of Chidamber and Kemerer metrics is a widely accepted standard for measuring object-oriented software systems [54]. The metrics set originally consists of 6 metrics. However, only Max_{DST} and $NOC(c)$ can be directly used since the ASOME data models lack the concept of methods. Some researchers believed that $NOC(c)$ measures the breadth of a model, and Max_{DST} measures the depth of a model. They found higher $NOC(c)$ may cause fewer faults while higher Max_{DST} may increase faults [10].

4.1.2 Marchesi's Metrics (1998)

Let $ADM = (BC, C, R)$ be an ASOME data model. The Marchesi's metrics for the ASOME data model ADM consist of the metrics 1 until 7 below:

1. $\#C$ is the total number of classes.

[1] DST is called DIT (depth of inheritance tree) in original metrics

2. #*ROOTS* is the total number of roots, where *ROOTS* is the set of classes satisfying:
$$ROOTS = \{c \in C \mid \neg\exists c' \in C : \langle c, c' \rangle \in r_{spe}\}.$$

3. $\overline{RES}$ is the value of the average weighted responsibilities of the classes. The definition of $\overline{RES}$ is:
$$\overline{RES} = \frac{RES}{\#C}$$
$$\text{where } RES = \sum_{c \in C} Res(c).$$
RES is the value of total weighted responsibilities of the classes.
Marchesi thought that the responsibilities of a class are related to the information it contains or the computation that must be performed for this class [42]. For simplicity we directly use the number of attributes to stand for the responsibilities of a class. Thus, the definition of $Res(c)$ is the following. Let $c \in C$ be a class where $c.A$ is the set of attributes of the class c.
$$Res(c) = \#c.A + K_a \cdot \sum_{i \in Sub(c,ADM)} (\#i.A) + K_r \cdot \sum_{j \in Sup(c,ADM)} (\#j.A).$$
The values K_a and K_r are some weighted coefficients that satisfy $0 < K_r < K_a < 1$.

4. $\|RES\|$ is the standard deviation of the weighted responsibilities of the classes, where
$$\|RES\| = \sqrt{\frac{1}{\#C} \sum_{c \in C} (Res(c) - \overline{RES})^2}.$$

5. $\overline{DEP}$ is the value of the average direct dependencies of the classes. The direct dependencies include all the relations except specialization. The definition is the following:
$$\overline{DEP} = \frac{DEP}{\#C}$$
$$\text{where } DEP = \#r_{ass} + \#r_{com}.$$

6. $\|DEP\|$ is the standard deviation of the direct dependencies of classes, defined as
$$\|DEP\| = \sqrt{\frac{1}{\#C} \sum_{c \in C} (Dep(c) - \overline{DEP})^2}.$$
Let $c \in C$ be a class. $Dep(c)$ is the value of the direct dependencies of the class c. The definition of $Dep(c)$ is as follows:
$$Dep(c) = \sum_{c' \in C} d_{c,c'}$$
where $d_{c,c'}$ is the number of the direct dependencies between the class c and c', defined as

$$d_{c,c'} = d_{c,c'}(r_{ass}) + d_{c,c'}(r_{com})$$

$$\text{where } d_{c,c'}(r) = \begin{cases} 1, & \text{if } c \; r \; c' \text{ , and} \\ 0, & \text{otherwise.} \end{cases}$$

7. *PSR* is the percentage of specialized responsibilities of classes. The definition is the following:

$$PSR = \frac{\sum\limits_{c \in C} SR(c)}{\sum\limits_{c \in C} TR(c)}$$

where $SR(c)$ is the specialized responsibilities of the class c and $TR(c)$ is the total number of responsibilities of the class c defined as:

$$SR(c) = \sum_{j \in Sup(c,CD)} \#j.A$$

$$TR(c) = \#c.A + IR(c).$$

Marchesi applied these metrics for three real projects based on UML 1.0, all developed in Smalltalk [42]. By analyzing the value of the metrics related to the man-months needed to develop the systems, Marchesi concluded that a man-month seems to be able to develop between 14 to 20.5 responsibilities. She also believed that the productivity of Smalltalk is very high compared with other programming languages for small or medium-sized projects.

4.1.3 Genero's Metrics (2000)

Let $ADM = (BC, C, R)$ be an ASOME data model. The Genero's metrics are defined as:

1. $\#C$ is the total number of classes.
2. $\#A$ is the total number of attributes where

$$\#A = \sum_{c \in C} \#c.A.$$

3. $\#r_{ass}$ is the total number of elements in the association relation.
4. $\#r_{com}$ is the total number of elements in the composition relation.
5. $\#r_{spe}$ is the total number of elements in the specialization relation.
6. *NCH* is the total number of composition hierarchies, defined as:

$$NCH = \#\{c \in C \mid \exists c' \in C \,.\, c' \; r_{com} \; c\}.$$

7. *NSH* is the total number of specialization hierarchies, defined as:

$$NSH = \#\{c \in C \mid \exists c' \in C \,.\, c' \; r_{spe} \; c\}.$$

8. Max_{DST} is the maximum value of the depth of the specialization tree, which is same as the first metric in Chidamber and Kemerer's Metrics (Section 4.1.1)

   The definition of Max_{DST} is same with the definition in

   the following 1:

$$Max_{DST} = Max\{DST(c) \mid c \in C\}$$

$$\text{where } DST(c) = \begin{cases} 1 + DST(c'), & \text{if } c\ r_{spe}\ c' \text{, and} \\ 0, & \text{otherwise.} \end{cases}$$

M. Genero's metrics set is designed to evaluate the structural complexity of a UML class diagram [6]. These metrics are related to the usage of the relations in a UML class diagram, such as associations, generalisations, aggregations and dependencies. Genero interpreted the above metrics into two types in a general way [6]:

- The metrics 1 and 2 are the size metrics for measuring the size or capacity of an ASOME data model.
- The metrics 3-8 are the complexity metrics for evaluating the system complexity or maintainability of an ASOME data model.

Compared with Marchesi's metrics, Genero's metrics is a compensation for the relations measurements in a class diagram because Marchesi considered all types of relations except inheritance as dependencies, without distinguishing between them [16]. For empirical validation of the metrics, Genero also applied these metrics to a real experiment, carried out by students of the Department of Computer Science at the University of Castilla-La Mancha, in Spain [47].

The experimental result indicated that *NAH* (the total number of aggregation hierarchies), *NIH* (the total number of inheritance hierarchies) and #*A* (the total number of attributes) are related to the complexity and maintainability of a UML class diagram. Genero argued that, in a class diagram, lower *NAH* and *NIH* may benefits the whole system because the aggregation and inheritance relations are highly related to the understandability time and modifiability correctness and completeness based on the experiment data [16].

4.1.4 Zhou's Metrics (2003)

The metrics we mentioned above all use multiple indicators to measure the complexity of class diagrams. In 2003, Y. Zhou argued that metrics with multiple indicators might cause disorganized data results and he proposed a new metric with merely one indicator, namely the entropy distance based on the structural complexity metric, or Zhou's metrics for short [64]. Let $ADM = (BC, C, R)$ be an ASOME data model. The entropy distance metric is built by the following three steps:

4.1.4.1 Weights of relations

Firstly, we give an order of the relation types. Each of the relation type is given an individual weight, as indicated by the following table:

Type of relations	$Weights(r)$
association	w_1
composition	w_2
specialization	w_3

TABLE 4.3: Weights of Relations

Based on the features of those relations, the relation weights $Weights(r)$ satisfy:

$$w_1 \leq w_2 < w_3$$

.

Note that Zhou [64] distinguished the relations into ten different types. However, based on our relations in ASOME data model, we merely have three.

4.1.4.2 Weighted class dependency graphs (*WCDG*)

The second step is to transform the ASOME data model *ADM* to a weighted class dependency graph *WCDG*. The definition of a *WCDG* is the following.

Definition 4.1.1. (Weighted Class Dependency Graph). A weighted class dependency graph is a two tuple $WCDG = (N, E)$ where

- N is a set of nodes.
- $E : N \times N \times H$ is a set of unidirectional edges where H is the set of all weight values.

If $\langle n_1, n_2, h\rangle \in E$, then n_1 is called the source node, n_2 is called the target node and h is called the edge weight.

The transformation rules from *ADM* to *WCDG* are defined as:

1. $N = C$.
2. Let $c_1, c_2 \in C$ be two classes of the class diagram and $n_1, n_2 \in N$ be the corresponding nodes of the *WCDG*. We build an edge $\langle n_1, n_2, h\rangle \in E$ between the nodes n_1 and n_2 iff there is at least one relation between the classes c_1 and c_2 and the edge weight value h equals to the sum of all the relation weights between c_1 and c_2. More precisely:

$$h_{c_1,c_2} = h_{c_1,c_2}(r_{ass}) + h_{c_1,c_2}(r_{com}) + h_{c_1,c_2}(r_{spe})$$

$$\text{where } h_{c_1,c_2}(r) = \begin{cases} Weights(r), & \text{if } c_1\ r\ c_2 \quad \text{and } r = r_{ass} \vee r_{com} \vee r_{spe} \\ 0, & \text{otherwise.} \end{cases}$$

Note that if there is no relation between the classes c_1 and c_2 in the *ADM*, the weight value h equals to zero. In other words, $h = 0$ means that no edge from the node n_1 to n_2 in the *WCDG*.

A basic transformation rule from *ADM* to *WCDG* are given in Figure 4.1.

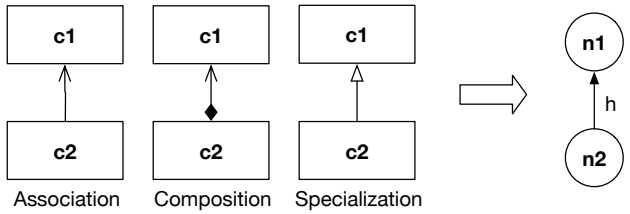


FIGURE 4.1: Transformation rule from *ADM* to *WCDG*

4.1.4.3 Entropy distance based metric

After transforming the class diagram $ADM = (C, R)$ to the weighted class dependency graph $WCDG = (N, E)$, we use a matrix $M(n, n')$ to represent the weights of the edges in the *WCDG* graph, where

$$M(n, n') = h, \quad \text{if } \langle n, n', h \rangle \in E$$

We use two discrete random variables X and Y to denote the outgoing and incoming edges weight of each note, where A_x and A_y represent the sets of the variables X and Y separately.

Let $A_x = A_y = N$, for each $x_i \in A_x$ and $y_j \in A_y$, we have the following probability equations:

$$p(x_i) = \sum_{n' \in N} M(x_i, n') / \sum_{n, n' \in N} M(n, n') \tag{4.1}$$

$$p(y_j) = \sum_{n \in N} M(n, y_j) / \sum_{n, n' \in N} M(n, n') \tag{4.2}$$

$$p(x_i, y_j) = M(x_i, y_j) / \sum_{n, n' \in N} M(n, n') \tag{4.3}$$

$$p(x_i \mid y_j) = M(x_i, y_j) / \sum_{n \in N} M(n, y_j) \tag{4.4}$$

Based on the equations above, the structure complexity of the class diagram *ADM* can be defined as the entropy distance between X and Y. More precisely:

$$\boxed{DH(X, Y) = \begin{cases} H(X, Y) - I(X; Y), & \text{if } r_{ass} \cup r_{com} \cup r_{spe} \neq \emptyset \\ 0, & \text{if } r_{ass} \cup r_{com} \cup r_{spe} = \emptyset \end{cases}}$$

where $H(X, Y)$ is called joint distribution defined as:

$$H(X, Y) = - \sum_{x_i \in A_x \wedge y_j \in A_y} p(x_i, y_j) \log p(x_i, y_j)$$

and $I(x; y)$ is the mutual-information defined as:

$$\begin{aligned} I(X; Y) &= H(X) - H(X \mid Y) \\ &= - \sum_{x_i \in A_x} p(x_i) \log p(x_i) - \sum_{x_i \in A_x \wedge y_j \in A_y} p(x_i, y_j) \log \frac{1}{p(x_i \mid y_i)}. \end{aligned}$$

Note that log means the common logarithm, where $\log x$ represents the logarithm of x to base 10.

Through the above three steps, we can get the value of the entropy distance based metric $DH(X, Y)$ as the indicator to evaluate the structural complexity of a UML

class diagram. Compared with previous metrics, $DH(X, Y)$ only use one indicator. Zhou [64] thought that the complexity of a UML class diagram is depended on the complexity of relations in the class diagram in most cases. Furthermore, he also pointed out that the weights of class relations can be reorded according to the real conditions in a project.

4.1.5 Main problems of the metrics in literature study

We summarize some main problems in our literature study.

- Converting the metrics from their original scopes to ASOME data models requires that we give our own interpretations.
- Several metrics cannot be used in ASOME data models since their scopes are different, which causes some metrics sets not to be complete compared with the original versions. CK metrics in Section 4.1.1 is an example.

4.2 Metrics - Interview Study

Interviews are useful methods to obtain information for personal experiences, perceptions and opinions. To prepare the interviews, we need to answer the following questions:

1. *What information do we need to get from the interviews?*
   We need to know the insights of software engineers and architects about what makes an ASOME model easy or difficult to comprehend and what makes it complex.
2. *What kind of interview we will conduct?*
   An interview can be structured, semi-structured or unstructured [28]. A structured interview is typically quite formal and well-organized while an unstructured interview is informal without any discussion limitation. We conduct semi-structured interviews since semi-structured interviews need an 'interview guide', which will give a main topic and a list of questions that need to be covered during the conversation. The question list is built with both open-ended and specific questions, which allows an interviewer to gather unanticipated and specific information.
3. *How to choose respondents?*
   Senior architects and experienced model designers are the respondents to the interview because they have useful insights related to data models in the ASML domain.

According to the above questions, we designed an interview guide in Appendix B.1. In Section 4.2.1, we formalize our findings from interviews into metrics. In Section 4.2.2, a summary is presented.

4.2.1 Metrics in interview study

Let $ADM = (BC, C, R)$ be an ASOME data model, the metrics defined in our interview study are the following:

- **Size Metrics - basic element**

1. $\#Typ$ is the total number of primitive types.
2. $\#Enu$ is the total number of enumerations.
3. $\#Con$ is the total number of constants.
4. $\#Mc$ is the total number of multiplicity constants.

- **Size Metrics - class**

1. $\#Ent$ is the total number of entities.
2. $\#Ov$ is the total number of value objects.

- **Size Metrics - relation**

1. $\#r_{ass}$ is the total number of elements in the association relation.
2. $\#r_{com}$ is the total number of elements in the composition relation.
3. $\#r_{spe}$ is the total number of elements in the specialization relation.

- **Size Metrics - association relation**

1. $\#r_{ass}(e)$ is the total number of elements in the association relation, where the source entity is e. The definition is the following:

$$\#r_{ass}(e) = \{\langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass} \mid e_1 = e\}.$$

2. $\#r_{ass} \mid_{1-1}$ is the number of 1-1 associations, where

$$\#r_{ass} \mid_{1-1} = \{\langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass} \mid (m_1 = [0..1] \vee m_1 = \{0..1\}) \wedge (m_2 = [a..b] \vee m_2 = \{a..b\}) \wedge (b = 1)\}.$$

3. $\#r_{ass} \mid_{1-N}$ is the number of 1-N associations, where

$$\#r_{ass} \mid_{1-N} = \{\langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass} \mid (m_1 = [0..1] \vee m_1 = \{0..1\}) \wedge (m_2 = [a..b] \vee m_2 = \{a..b\}) \wedge (b > 1 \wedge b \neq *)\}.$$

4. $\#r_{ass} \mid_{1-*}$ is the number of 1-Many associations, where

$$\#r_{ass} \mid_{1-*} = \{\langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass} \mid (m_1 = [0..1] \vee m_1 = \{0..1\}) \wedge (m_2 = [a..b] \vee m_2 = \{a..b\}) \wedge (b = *)\}.$$

- **Multiplicity Metrics**

1. $\#MULT$ is the number of multiplicities. The definition is the following:

$$\#MULT = \#Ent + 2 \cdot \#r_{ass} + \#r_{com}.$$

2. $\#|MULT|$ is the number of different multiplicities. The definition is the following:

$$\begin{aligned} |MULT| = {} & \{m \in \mathcal{M} \mid m = e.M \wedge e \in Ent\} \\ & \cup \{m \in \mathcal{M} \mid m = m_1 \wedge \langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass}\} \\ & \cup \{m \in \mathcal{M} \mid m = m_1 \wedge \langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass}\} \\ & \cup \{m \in \mathcal{M} \mid \langle (c_1, m), ov\rangle \in r_{com}\}. \end{aligned}$$

- **Mutable Entity and Control Entity Metrics**

  1. $\#Ent_{mut}$ is the number of mutable entities, where Ent_{mut} is the set of the mutable entities.

  $$\#Ent_{mut} = \{ent \in Ent \mid ent = ent_{mut}\}.$$

  2. $?Ent_{mut}$ is the existence of mutable entities,

  $$\text{where } ?Ent_{mut} = \begin{cases} 1, & \text{if } \#Ent_{mut} \geq 1, \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

  3. $\#Ent_{ctr}$ is the number of control entities. where Ent_{ctr} is the set of the control entities.

  $$\#Ent_{ctr} = \{ent \in Ent \mid ent = ent_{ctr}\}.$$

Note that metric number of cross reference is not defined in this section since this metric is based on the scope of multiple ASOME data models. All the above metrics are selected from the interview summary report B.2.

4.2.2 Summary of the metrics in interview study

Several metrics are derived through the interview study. Among these metrics, **Size Metrics - class** might be more important than the others since two interviewers have mentioned these metrics. It is also noteworthy that some metrics such as $\#E_{mut}$ and $\#E_{ctr}$ are apparently domain specific and we cannot find them in literature study.

4.3 Metrics - Document Analysis

In order to provide some general rules or suggestions to the model designers, ASML prescribes 57 guidelines and 55 standards for data models [61]. We analyze these guidelines and standards to find whether some metrics may relate to them.

In Section 4.3.1, we formalize our findings into metrics and in Section 4.3.2 we present a summary.

4.3.1 Metrics in document analysis

Let $ADM = (BC, C, R)$ be an ASOME data model, the metrics defined in document analysis are the following

1. $\#r_{ass} + \#A$ is the total number of elements in the association relation combined with the total number of the attributes, where it satisfies

$$\#r_{ass} + \#A \leq 9$$

2. $\#r_{ass} \mid_{NZT}$ is the number of the associations without zero target multiplicity. The definition is the following:

$$\#r_{ass} \mid_{NZT} = \{\langle (e_1, m_1), (e_2, m_2) \rangle \in r_{ass} \mid (m_2 = [a..b] \vee m_2 = \{a..b\}) \wedge (a \neq 0)\}.$$

3. $\#r_{ass}\ |_{FT}$ is the number of the associations with fixed target multiplicity, where

$$\#r_{ass}\ |_{FT} = \{\langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass}\ | \\ (m_2 = [a..b] \vee m_2 = \{a..b\})\ \wedge\ (a = b)\}.$$

4. $\#|r_{ass}|$ is the number of the different kinds of associations, where

$$|r_{ass}| = \{(m_1, m_2)\ |\ \langle (e_1, m_1), (e_2, m_2)\rangle \in r_{ass}\}.$$

5. $\#Ent_{ctr}$ is the number of the control entities.
6. $\#Ent_{alg}$ is the number of the algorithm entities.
7. $\#Ent_{mut}$ is the number of the mutable entities.
8. $\#r_{spe}$ is the number the elements in specialization relation.
9. $\#r_{com}$ is the number the elements in composition relation.

4.3.2 Summary of the metrics in document analysis

The summary of the metrics based on the analysis of the modeling guidelines is given in the Appendix C.

4.4 Tool Design

We developed an automated tool to calculate the metrics. This tool is based on the ASOME development environment and it refers to the metamodels and textual syntax of the ASOME language. Figure 4.2 describes a basic workflow of the prototype. We we run the prototype:

1. It will first search all the ASOME data models (the files end with *.asome*) in the preset directory.
2. And then for each ASOME data model, the prototype will resolve it and get the resource set (also called a tree of related resource objects) and we can get specific objects from this set.
3. Finally, the prototype will calculate the corresponding metrics for each resource set of the ASOME model and the results will be stored in an SQL database.

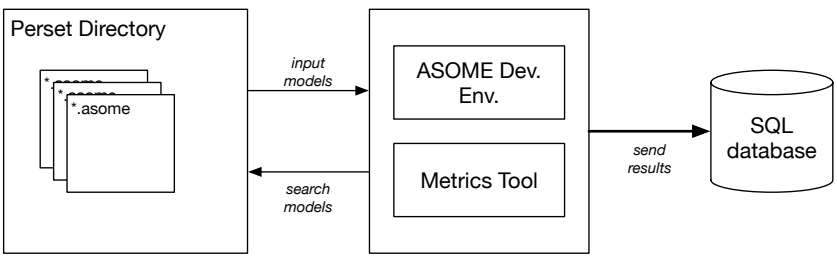


FIGURE 4.2: The workflow of the Metrics Tool

Figure 4.3 gives an example of the metrics results in SQL database.

id	project	modelname ▲ 1	metricset	metric	value	createtime	comment
176	/KDDA_models	Alignment_DI	MetricsGeneral	numVO	5	2018-07-18 03:52:19	General metrics for some basic metrics
177	/KDDA_models	Alignment_DI	MetricsGeneral	numEnt	0	2018-07-18 03:52:19	General metrics for some basic metrics
178	/KDDA_models	Alignment_DI	MetricsGeneral	numAssoc	0	2018-07-18 03:52:19	General metrics for some basic metrics
179	/KDDA_models	Alignment_DI	MetricsGeneral	numTyp	0	2018-07-18 03:52:19	General metrics for some basic metrics
180	/KDDA_models	Alignment_DI	MetricsGeneral	numAttri	13	2018-07-18 03:52:19	General metrics for some basic metrics

FIGURE 4.3: An example of the metrics results

4.4.1 Advantages of our metrics tool

There are several advantages of our metrics tool:

- The tool supports processing the calculation of multiple ASOME models and multiple metrics. The database also supports storing historical data by adding a time stamp to each result.
- The tool is scalable for metrics. It allows you to write your own metrics by adding teh code for a new metrics.
- Since the tool is based on the ASOME development environment, it can be directly updated if the environment evolves.

4.4.2 Disadvantages of our metrics tool prototype

There are several disadvantages of the metrics tool prototype:

- The tool is only able to resolve the textual ASOME models (end with *.asome*). Note that the data models in old versions (end with *.hddd*) should be converted to the ASOME models first.
- The metrics prototype is developed for a single ASOME model scope. Thus, it is not possible to calculate metrics at system level with multiple models.

Chapter 5

Evaluation

In this chapter, we present our evaluation for the metrics provided in the previous chapter. First we review several existing evaluation approaches for software metrics. Then we define our evaluation settings and data collection. Finally, we analyse and interpret the results.

5.1 Evaluation approaches

In general, there are two main methodologies to evaluate metrics [32]:

- **Theoretical validation**: The main goal of theoretical validation is to check if the intuitive or formal idea of the data being measured is reflected in the measurement [32]. Although several researchers have attempted to establish their validation frameworks, there is no satisfactory standard to be able to validate metrics theoretically [58].
- **Empirical validation**: Empirical validation is another methodology to evaluate the quality of metrics. The goal of empirical validation is to prove the practical utility of the proposed metrics [32]. M. Piattini divides the empirical validation into experiments and case studies [32].

We carried out an empirical validation with experiment to evaluate these metrics.

5.2 Experiment Settings and Data Collection

The participants in our experiment study were 8 staff members of ASML who were using ASOME data models. The participants were asked to complete a survey we specifically designed. The survey includes:

- **Twenty six ASOME data models**: we selected 26 ASOME models from four different projects in the ASML repository. In this survey experiment, we provided the model diagrams to the participants and we used two questions for each ASOME model.
- **Two questions for each ASOME model**: we designed two questions for each ASOME model, where
  - Question 1 is a 'Yes / No' question to check whether the participant really understood the information conveyed by the model diagram. Question 1 is an objective question with a correct answer. There are several aspects which question 1 covers:
    1. the relations between entities and value objects,

2. the multiplicity of association and composition relations,
3. the dependency between entities and value objects,
4. the multiplicity of an entity.

– Question 2 is a ranking question which asks the participants to rate the complexity and maintainability of the model. Both complexity and maintainability use the scale which consists of five linguistic labels in the following table (Table 5.1). Question 2 is a subjective question.

0	1	2	3	4
very low	low	middle	high	very high

TABLE 5.1: complexity and maintainability linguistic labels

- **Time consumption for each ASOME data model:** when a participant completes the two questions of an ASOME data model, we record the starting time and ending time to get his time consumption on this model.

Since the 26 models which ASML were using were real and not public, we used Microsoft Forms [1]. An example of the survey questions is given in the Appendix D.1.

5.3 Results analysis

5.3.1 Survey data results

In our survey, each ASOME data model has two questions. For the response of a model to be considered as complete, it is a prerequisite that both question 1 and question 2 of this model are answered. If only one of the questions or both questions are not answered, the response is considered incomplete and will not be added to our data results.

A general survey result is given in the following table (Table 5.2), where the Completeness is $\frac{\text{Number of responses}}{\text{Number of ideal responses}} = \frac{193}{26*8} = 93\%$.

Number of models	26
Number of participants	8
Number of complete responses	193
Completeness	93%

TABLE 5.2: A summary of survey results

The survey results can be downloaded from the MS forms. Since the raw survey results are separated into different models, we need to process the survey results and combine these results into one data set.

We first process the correctness (according to question 1) of the survey results. There are 158 correct responses and the general correctness ratio is $158/193 = 82\%$. We exclude the responses with wrong answers and keep the remaining 157 responses.

Then we consider the time consumption. The time consumption of the survey is depicted in box-plot (see Figure 5.1). The horizontal axis represents the model id and

[1] Microsoft Forms: https://forms.office.com/

the vertical axis shows the time consumption for each model. We found 13 outliers in the box-plot of Figure 5.1. These 13 points were also excluded in the further analysis. Figure 5.2 gives the box-plot of the time consumption without outliers.

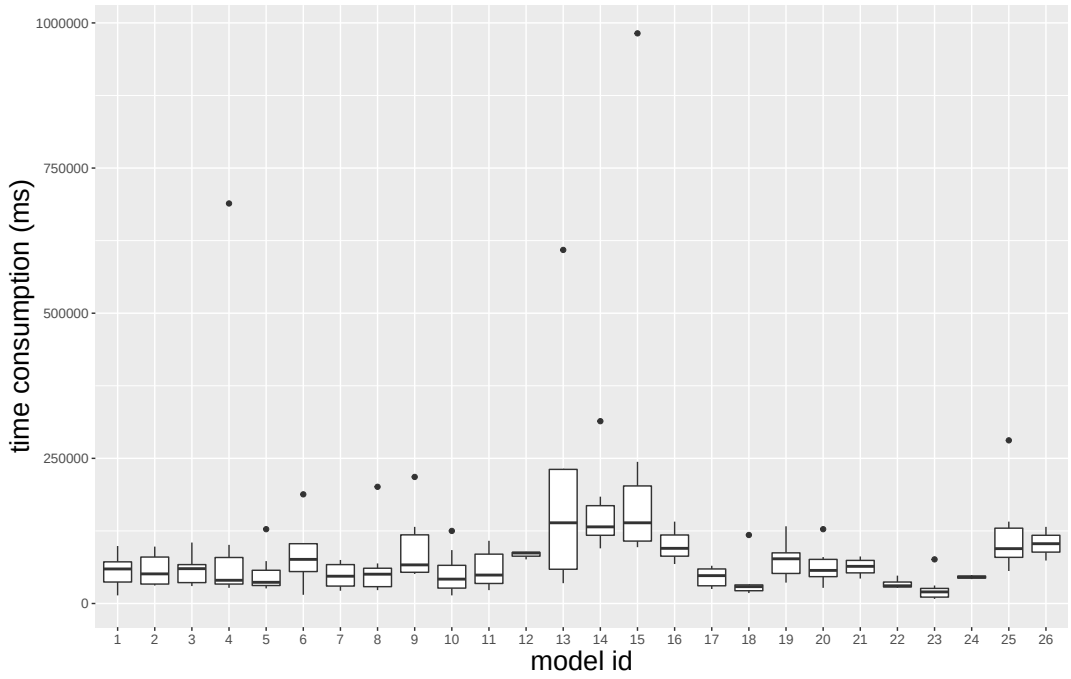


FIGURE 5.1: Time consumption for different models

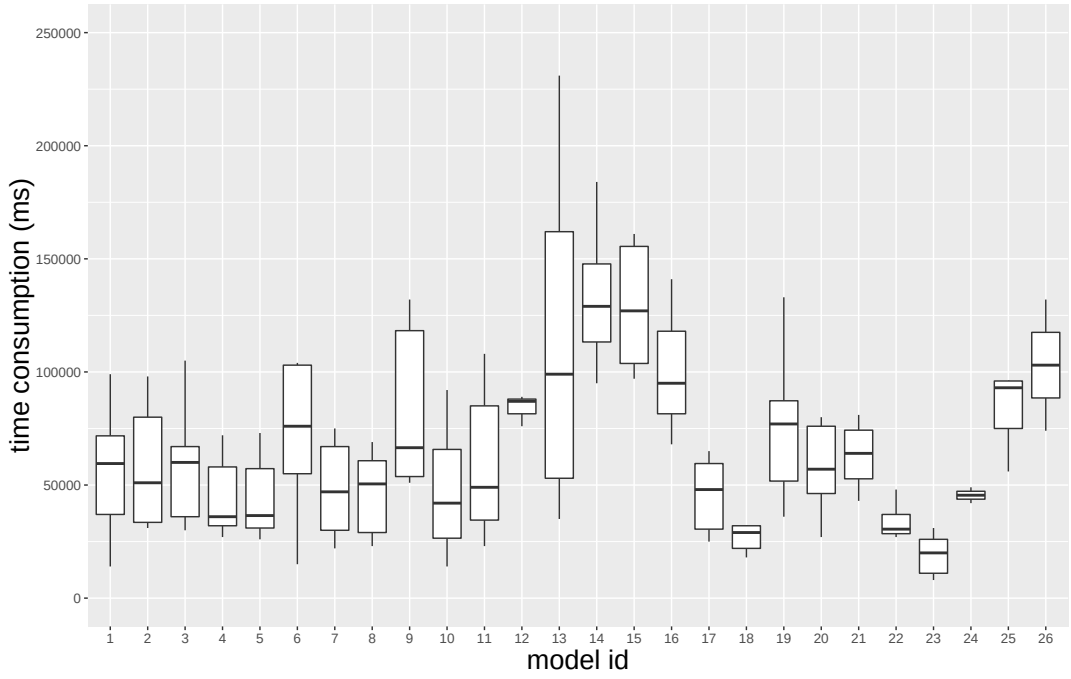


FIGURE 5.2: Time consumption for different models (outliers removed)

We also process the results of complexity and maintainability for different models. Figure 5.3 and Figure 5.4 show the results of complexity and maintainability

respectively, which gives the means and medians of complexity or maintainability for different models.

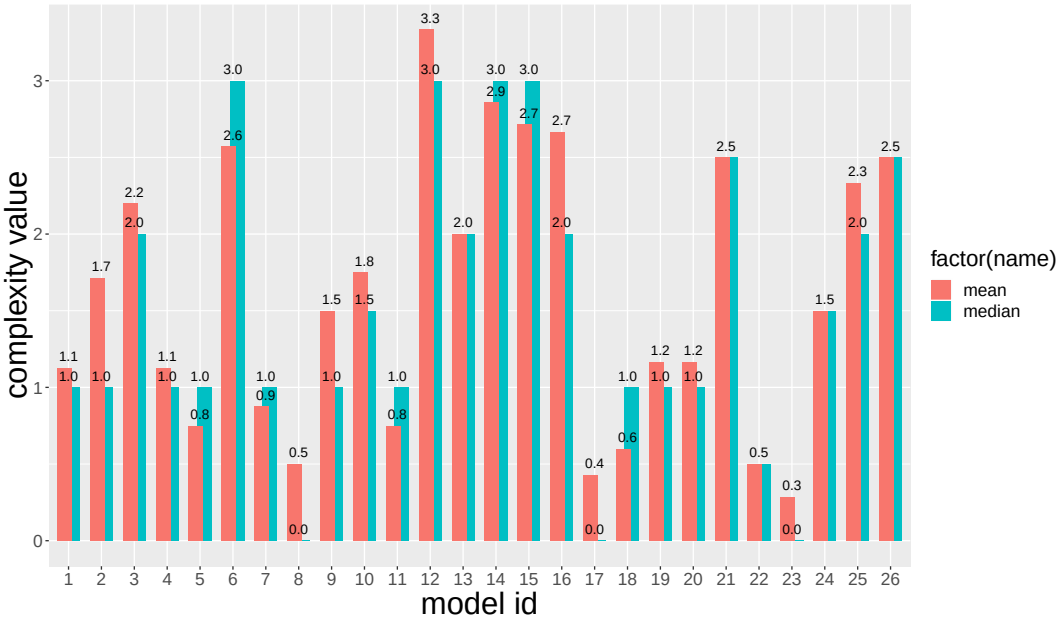


FIGURE 5.3: Complexity for different models

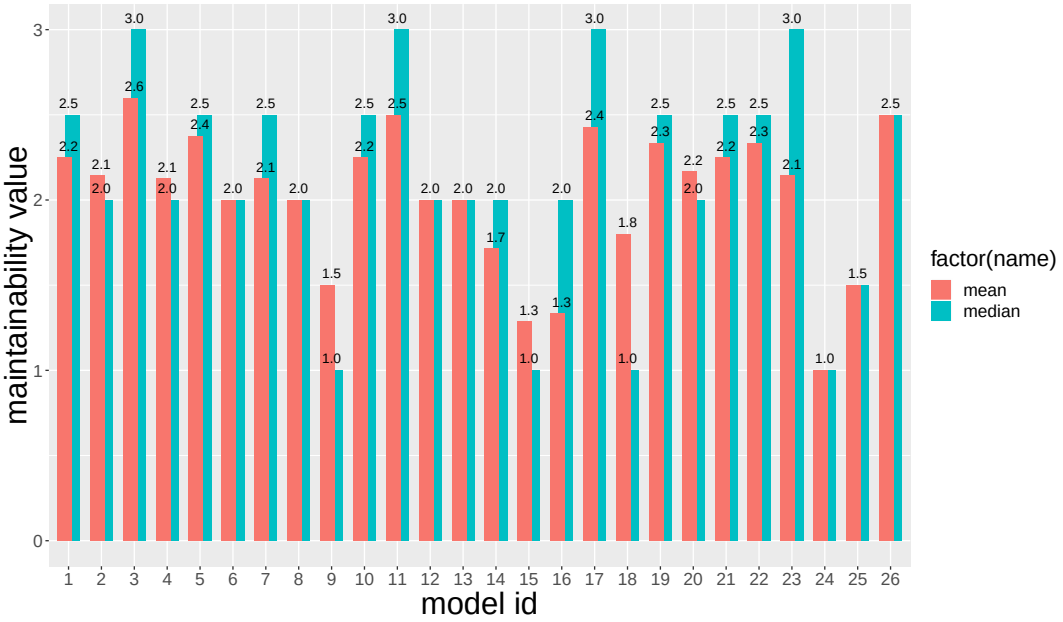


FIGURE 5.4: Maintainability for different models

After excluding both the incorrect answers and the time consumption with outliers, we consider the rest of the 145 responses as our final data of survey results. The processed data of survey results is given in Appendix E.1.

This experiment did not have a number of responses large enough to conduct statistic analysis. Also to be able to be validate every metric, we would need to conduct multiple experiments in which we would change one variable and have everything else the same. This is very difficult to organize in an industrial setting in a limited time scope. To really assess which metric is useful and which not requires

more extensive analysis. The analysis in the following section is a first step, the analysis we would perform in a larger experiment.

5.3.2 Survey data and metrics data

As stated in pervious section, we consider the 145 responses as our survey data results. In this section, we use the survey data and software metrics data to analyze the performance of the metrics we defined in Chapter 4.

We start with the Kolmogorov–Smirnov test, which can indicate whether the data distributions are normal or not. Then we use Spearman's correlation coefficient as a non-parametric test statistic to analyze the survey data and metrics data. The obtained Spearman's correlation coefficients are partially provided in Table 5.3. The full version of correlation results can be checked in Appendix.

	time consumption	complexity	maintainability
$\#Ent$	0.099	0.211	-0.073
$\#Vo$	0.462	0.437	-0.117
$\#C$	0.251	0.355	-0.111
$\#r_{ass}$	0.018	0.190	-0.056
$\#r_{com}$	0.553	0.549	-0.175
$\#r_{spe}$	-0.214	-0.353	0.027
$\#r$	0.018	0.190	-0.056

TABLE 5.3: Spearman's correlation coefficients between the metrics data and survey data

Since we have multiple types of metrics, to pick the metrics which have higher correlation with the time consumption, complexity or maintainability, we compared these correlation values and selected the values larger than 0.5. There metrics and corresponding correlation coefficients are given in the following table.

Metrics	Correction type	Value
$\#r_{com}$	time consumption	0.553
$\#r_{com}$	complexity	0.549
$\#ROOTS$	complexity	0.531
$\#A$	complexity	0.605

TABLE 5.4: Spearman's correlation coefficients larger than 0.5

The findings that Table 5.4 reveals are:

- The highest correction value is the metric of the number of attributes
- The metric of the number of composition has high correlation with both time consumption and complexity.
- Unfortunately, we did not find any metrics which has correlation with maintainability larger than 0.5. The highest one is the number of composition with value of -0.175.

Chapter 6

Conclusion

In this chapter, the main findings with regard to the design goal and research questions are summarized. We first study the quality attributes of the ASOME data models and select understandability, maintainability and complexity as the goals of our metrics design. Then we we discover three categories of metrics defined for models so far: metrics in literature study, metrics in interview study and metrics in document analysis. In total 57 metrics are defined and formalized although some of them are repetitive. These metrics measure the the syntactic properties of ASOME data models, which means they can be calculated automatically, without human intervention. We also evaluate these metrics by using a survey experiment. The correlation analysis shows that some metrics could be validated as indicators of the quality of ASOME data models.

There are still some limitations in this thesis. The first limitation is that not all the metrics we formalized have been evaluated in our survey experiment since some metrics are not supported in our tool. A second limitation is our research scope is based on a single ASOME data model while some may claim that the scope of a system level (with multiple ASOME data models) is more reasonable.

Appendix A

Quality Attributes in Different Quality Models

In this chapter, six software quality models are summarised in the following tables, including their quality attributes and attributes descriptions. Note that we omit the descriptions of some attributes if the author did not give the clear definitions for them.

Quality attributes	**Descriptions**
Functional Suitability	The degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions.
Performance efficiency	The performance relative to the amount of resources used under stated conditions.
Compatibility	The degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment.
Usability	The degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.
Reliability	The degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.
Security	The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.
Maintainability	The degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.
Portability	The degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.

TABLE A.1: Quality attributes in ISO 25010 quality model [24]

Quality attributes	Descriptions
Functionality	A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
Reliability	A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
Usability	A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
Efficiency	A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
Maintainability	A set of attributes that bear on the effort needed to make specified modifications
Portability	A set of attributes that bear on the ability of software to be transferred from one environment to another.

TABLE A.2: Quality attributes in ISO 9126 quality model [25]

Quality attributes	Descriptions
Functionality	The ability of a component to provide the required services and functions, when used under the specified conditions.
Reliability	The capability of a component to maintain a specified level of performance when used in stated conditions in a stated period of time.
Usability	
Efficiency	The capability of a component to provide appropriate performance, relative to the amount of resources used under stated conditions.
Maintainability	The effort required to replace a COTS (commercial off-the-shelf) component with the corrected version and to migrate an existing, software component from a current component based software system to a new version of the system.
Portability	The ability of a component to be transferred from one environment to another with little or no modification.
Reusablity	
Traceability	The extent of a component's built in capacity of tracking the status of component attributes and component behavior.

TABLE A.3: Quality attributes in SCQM quality model [57]

Quality attributes	**Descriptions**
Correctness Properties	Correctness properties fall broadly into three categories that deal with computability, completeness and consistency.
Structural Properties	The structural properties we have used focus upon the way individual statements and statement components are im- plemented and the way statements and statement blocks are composed, related to one another and utilized.
Modularity Properties	The modularity properties employed largely address the high-level design issues associated with modules and how they interface with the rest of a system.
Descriptive Properties	
Refinement Properties	

TABLE A.4: Quality attributes in Dromey's quality model [12]

Quality attributes	**Descriptions**
Portability	The software can be operated easily and well on computer configurations other than its current one.
Reliability	The software can be expected to perform its intended functions satisfactorily.
Efficiency	The software fulfills its purpose without waste of resources.
Usability	The software is reliable, efficient and human-engineered.
Testability	The software facilitates the establishment of verification criteria and supports evaluation of its performance.
Understandability	The software purpose is clear to the inspector.
Flexibility	The software facilitates the incorporation of changes, once the nature of the desired change has been determined.

TABLE A.5: Quality attributes in Boehm's quality model [4]

Quality attributes	**Descriptions**
Functionality	it may include feature sets, capabilities, and security.
Usability	it may include human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and agents, user documentation, and training materials.
Reliability	it may include frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF).
Performance	it imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage.
Supportability	it may include testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability.

TABLE A.6: Quality attributes in FURPS quality model [20]

Appendix B

Interview Guide and Results

B.1 Interview Guide

- **Pre-interview information:** Some pre-interview questions are asked to get a basic information about the respondent. The information includes his/her name and job background (relevant with ASOME data models).
- **Main question:**

  "Understandability (ISO 25010) is the degree to which data has attributes that enable it to be read and interpreted by users, and are expressed in appropriate languages, symbols and units in a specific context of use."

  Note that we use the ISO definition of understandability (ISO 25010) as an introduction to start our main questions. The questions include the following three parts:

  1. general feeling about the term understandability,
  2. habit when reading an ASOME model, and
  3. specific cases.

Here we give our question lists of the interview, where the question 1-4 cover the pre-interview information and the question 5-8 are the main questions.

1. What is your role (or position) in ASML?
2. How long have you had it?
3. How would you rate your knowledge about the ASML domain?
   - Expert
   - Medium
   - Beginner
4. In what way do you interact with ASOME data modeling tool and data models? (multiple choices)
   - Tool co-designer
   - Designer of the models (how many projects and how many models)
   - Reviewer of the models (how many projects and roughly how many models)
   - Designing software for accessing data repositories or interacting or depending them

5. How would you define understandability of an ASOME data model?
6. What criteria do you use to evaluate understandability of an ASOME data model? Number of entities, attributes, relations?

   What kind of relations? (internal, external relations; understanding of the domain described with the model, how 'elegant' the model is; diagram layout, naming, etc.)
7. When you review a model, how do you read it, what is the process?

   And what is difficult?
8. Do you have an example of a difficult to understand model?

B.2 Summary of Interview Results

Goal of the interviews

The goal of the interview is to determine the quality attributes of ASOME data models most relevant for their understandability, maintainability and complexity

B.2.1 Summary - Interview 1

Interview date	13 June 2018
Duration	around half an hour
Role	senior software architect, ASOME tool co-designer
Experience level	between the expert and medium for ASML domain knowledge, expert for tools for model-driven engineering

- Insights about model quality that **can** be formalized into metrics:

  1. If we look at an ASOME data model as a graph structure, in which entities and value objects represent nodes and their relation edges, then increasing number of nodes in the graph may increase model's complexity; it will take more time to think about these nodes (entities and value objects).
  2. Consider the ASOME data model as a tree-structured graph, the structural complexity of the graph will influence the understandability of the model, where the structural complexity covers the maximum hierarchies, number of roots, number of leaves and cross reference.
  3. Specialization (inheritance) relationship increases the complexity of an ASOME data model.

No.	Related metrics
1	number of entities, number of value objects, number of primitive types, number of enumerations, number of constants, number of multiplicity constants
2	number of cross references
3	number of Specializations (inheritance)

TABLE B.2: Summary of the related metrics of intervew 1

- Insights about quality that **cannot** be formalized int metrics but are useful to design guidelines:

    1. Some complexity on the system cannot be changed (or improved) since we should keep a basic functionality of the system, we describe this behavior as necessary or inherent complexity. One should not try to reduce this complexity, as the model will not be correct any more.
    2. Some models cannot be refactored since it may change the internal behavior of the system. E.g. a model with 50 entities => 5 models with 10 entities for each.
    3. Consistent naming scheme helps the quality of data models.
    4. A good documentation is important for viewers to understand the models since it describes the models in detail, through a well-written story.

B.2.2 Summary - Interview 2

Interview date	18 June 2018
Duration	around half an hour
Role	data architect for the Twinscan machine software. ASOME co-designer, designer of the models and reviewer of the models
Experience level	expert for everything relates to data for the Twinscan machine (data inside the machine, external communication with the customer data)

- Insights about model quality that **can** be formalized into metrics:

    1. Hierarchy (including association, composition and specialization) may influence the understandability. In an ASOME data diagram, the classes (entities or VOs) with higher hierarchies are more important than the those with lower hierarchies.
    2. The total number of the model elements will influence the understandability of an ASOME data model. The higher number of the elements, the worse understandability. The elements include the entities, value objects, types, enumerations and also the three relations (associations, compositions and specializations).
    3. Many-to-many associations are more difficult than the normal associations (Many-to-one, many-to-N).

No.	Related metrics
1	maximum association hierarchy, maximum composition hierarchy, maximum specialization hierarchy
2	number of entities, number of value objects, number of primitive types, number of associations, number of composition, number of specializations
3	number of associations of an entity, Number of different multiplicities 1-N associations, number of 1-Many associations, number of 1-1 associations

TABLE B.4: Summary of the related metrics of intervew 2

- Insights about quality that **cannot** be formalized int metrics but are useful to design guidelines:

  1. The classes (entities and value objects) with the same hierarchy should be grouped in the diagram. For example, the classes with the same hierarchy is good to place in the same horizontal position.
  2. A logic layout of an ASOME data model (in the graphical representation) will be easy to read but hard to implement because line crossing and elements covering may occur sometimes.
  3. Some elements (such as composition relation) or features (such as immutability of an entity) in an ASOME data model are allowed to be hidden. It is important to have a balance of these elements or features are hidden or not hidden in the model.
  4. Some external elements (especially some elements in the core models) will improve the understandability since they give some relevant context.
  5. If there is an association to a core data model, then the association should be present in the diagram.

B.2.3 Summary - Interview 3

Interview date	18 June 2018
Duration	around one hour
Role	senior design engineer (focus on data models), designer of the models, reviewer of the models
Experience level	between the expert and the medium for ASML domain knowledge (around 4 year experience on several projects, expert on Leveling part), expert on data models

- Insights about model quality that **can** be formalized into metrics:

  1. Multiplicities, especially the source multiplicities of an ASOME data model will require more attention of the reviewer of the model.
  2. Often, f there are no control entities in an ASOME data model, it is difficult to understand the model. Control entities in an ASOME data model will help the viewers to understand the runtime behaviors of the model.
  3. The mutability of the ASOME data model will influence the correctness checking and modifiability of the model.

No.	Related metrics
1	number of multiplicities, number of different multiplicities
2	number of control entities
3	number of mutable entities, existence of mutable entities

TABLE B.6: Summary of the related metrics of interview 3

- Insights about quality that **cannot** be formalized int metrics but are useful to design guidelines:

1. Naming and elements grouping will influence the understandability of an ASOME data model.
2. While designing an ASOME data model, it is important to identify what data is an input and what data is an output for different software actions.
3. If an entity's lifetime should not be linked to the lifetime of another entity, but still be associated to that entity, create an optional association, with a target multiplicity including zero.
4. Run-time behavior understanding will influence one's understandability of the corresponding data model.

Appendix C

Summary of document analysis

In this Appendix, a summary of the analysis of data model guidelines is present.

C.1 Attributes and association

- *"Due to code generator and compiler limitations, the number of associations combined with the number of attributes may not exceed 9. "*

Related metrics: the number of associations and attributes should not exceed 9.

  Related metrics: the number of associations and attributes should not exceed 9.

C.2 Entity lifetime

- *"If an entity's lifetime should be linked to the lifetime of another entity, create an association to the other entity with a target multiplicity of at least 1"*

  "If an entity's lifetime should not be linked to the lifetime of another entity, but still be associated to that entity, create an optional association, with a target multiplicity including 0."

  "An entity with a fixed multiplicity may only have non-optional associations in case the target is also an entity with a fixed multiplicity."

Related metrics: number of associations without zero target multiplicity, number of associations with fixed target multiplicity, number of different kinds of associations.

  Rationale: The multiplicity of an association relation not only give the entity number constraints but also may influence the lifetime and cascade deletion.

C.3 Multiplicities and ordering

- Related metrics: number of unordered multiplicities, number of order multiplicities

C.4 Control and algorithm entities

Related metrics: number of control entities, number of algorithmic entities

C.5 Mutability

- *"Immutable entities are easier to reason about and do not have locking or race condition issues. In addition the execution architecture impact of immutable entities is lower as some additional optimizations are possible. "*

  Related metrics: number of mutable entities.

C.6 Commonality between entities

- *"Use inheritance in case entities need to refer to a generic concept and are not interested in the differences between various specializations of the generic concept."*

  "Inheritance is the only way to model this, without having to use additional constraints."

  Related metrics: number of specializations

- *"Using composition prevents the complications of inheritance. "*

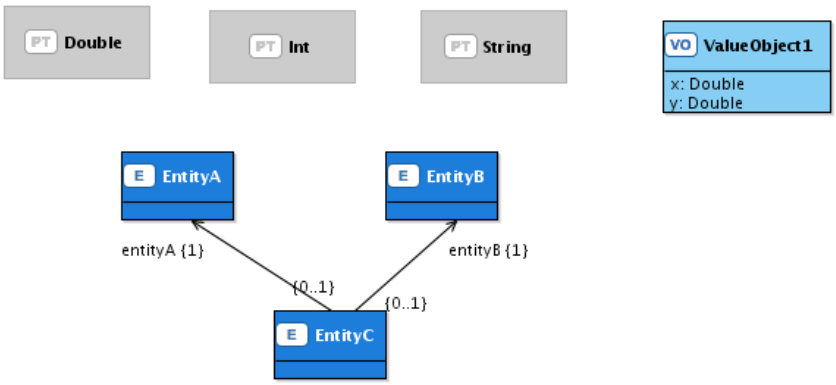
  Related metrics: number of compositions.

Appendix D

Survey material

D.1 An example of the survey

This section presents an example of the survey, including an ASOME data model and two questions.



1. At runtime, if **EntityA** is deleted, will any of the instance of **EntityB** be changed?

   ☐ Yes

   ☐ No

2. How would you rate the complexity and maintainability of the above ASOME data model?

	very low	low	middle	high	very high
complexity	◯	◯	◯	◯	◯
maintainability	◯	◯	◯	◯	◯

D.2 Survey questions of 26 ASOME data models

This section gives all the survey questions and the corresponding data models. The number in box refers to the corresponding model id in Appendix E. Note that we put the answers for all question 1 and we omit the question 2 for each model after model 2 because they are the same.

Appendix E

Survey Data and Metrics Data

E.1 Survey data results - proceeded

[1] model id
[2] participant name (we replaced the real names to alphabets A to H)
[3] time consumption

id[1]	part. name[2]	t.c.[3] (ms)	maintainability	complexity
1	A	59000	1	1
1	B	34000	1	3
1	C	60000	1	3
1	D	14000	1	1
1	E	99000	2	2
1	F	77000	2	1
1	G	70000	1	3
1	H	38000	0	4
2	A	31000	1	1
2	B	35000	1	3
2	C	62000	3	2
2	D	98000	3	3
2	E	98000	1	1
2	G	32000	1	3
2	H	51000	2	2
3	B	30000	2	2
3	C	105000	2	3
3	D	67000	3	3
3	G	36000	2	2
3	H	60000	2	3
4	A	36000	1	1
4	B	29000	1	3
4	D	44000	1	1
4	E	101000	2	1
4	F	72000	2	1
4	G	35000	1	3
4	H	27000	0	4
5	A	26000	1	1
5	B	31000	0	4
5	C	38000	0	1
5	D	73000	2	2
5	F	52000	1	3
5	G	31000	1	3
5	H	35000	0	4
6	A	15000	2	2
6	B	40000	2	2
6	C	70000	3	2
6	D	102000	3	3

– continued from previous page

id[1]	part. name[2]	t.c.[3] (ms)	maintainability	complexity
6	F	104000	3	1
6	H	76000	3	1
7	A	27000	1	1
7	B	22000	0	3
7	C	65000	1	4
7	D	73000	1	1
7	E	63000	0	0
7	F	75000	1	3
7	G	31000	1	3
7	H	31000	2	2
8	A	23000	1	1
8	B	23000	0	4
8	D	69000	0	0
8	E	52000	0	0
8	F	58000	0	3
8	G	31000	1	4
8	H	49000	2	1
9	B	53000	1	2
9	D	77000	1	1
9	E	132000	1	1
9	G	51000	1	3
9	H	56000	2	1
10	A	14000	1	2
10	B	25000	1	3
10	C	49000	3	3
10	D	57000	3	2
10	F	92000	2	1
10	G	27000	1	3
10	H	35000	2	3
11	A	85000	2	1
11	B	23000	0	4
11	C	85000	1	3
11	D	108000	1	1
11	E	38000	0	0
11	F	50000	1	3
11	G	48000	1	4
11	H	24000	0	4
12	D	87000	3	3
12	F	89000	4	2
12	H	76000	3	1
13	B	35000	2	2
13	C	231000	3	3
13	F	139000	3	1
13	H	59000	1	2
14	A	153000	2	2
14	B	109000	2	2
14	D	95000	3	3
14	E	184000	2	1
14	G	126000	3	2
14	H	132000	4	0
15	B	100000	2	2
15	D	139000	3	3
15	E	244000	1	1
15	F	115000	3	1
15	G	161000	3	1

– continued from previous page

id[1]	part. name[2]	t.c.[3] (ms)	maintainability	complexity
15	H	97000	4	0
16	A	68000	2	2
16	B	95000	2	2
16	H	141000	4	0
17	A	29000	1	1
17	B	25000	0	4
17	D	65000	1	1
17	E	54000	0	0
17	F	65000	1	3
17	G	32000	0	4
17	H	48000	0	4
18	A	18000	1	1
18	B	22000	1	3
18	D	32000	1	1
18	E	118000	0	0
18	H	29000	0	4
19	B	46000	1	3
19	D	85000	1	1
19	E	133000	2	2
19	F	88000	2	1
19	G	69000	1	3
19	H	36000	0	4
20	B	27000	1	3
20	D	50000	1	1
20	F	80000	2	1
20	G	64000	1	3
20	H	45000	1	4
21	B	56000	2	3
21	D	81000	3	3
21	G	43000	2	2
21	H	72000	3	1
22	A	27000	1	2
22	B	28000	0	4
22	D	39000	1	1
22	E	48000	0	0
22	G	30000	1	3
22	H	31000	0	4
23	A	9000	1	1
23	B	13000	0	4
23	D	8000	0	0
23	E	31000	0	0
23	G	20000	1	3
23	H	21000	0	4
24	D	42000	2	1
24	F	49000	1	1
25	A	75000	2	2
25	B	56000	2	2
25	E	141000	1	1
25	G	93000	3	1
25	H	96000	2	2
26	B	74000	2	2
26	D	132000	3	3

TABLE E.1: Proceeded Survey data results

Bibliography

[1] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. "Software Complexity and Maintenance Costs". In: *Commun. ACM* 36.11 (Nov. 1993), pp. 81–94. DOI: 10.1145/163359.163375. URL: http://doi.acm.org/10.1145/163359.163375.

[2] J. Bansiya and C. G. Davis. "A hierarchical model for object-oriented design quality assessment". In: *IEEE Transactions on Software Engineering* 28.1 (Jan. 2002), pp. 4–17. DOI: 10.1109/32.979986.

[3] B. W. Boehm, J. R. Brown, and M. Lipow. "Quantitative Evaluation of Software Quality". In: *Proceedings of the 2nd International Conference on Software Engineering*. ICSE '76. San Francisco, California, USA: IEEE Computer Society Press, 1976, pp. 592–605. URL: http://dl.acm.org/citation.cfm?id=800253.807736.

[4] B. W. Boehm, J. R. Brown, and M. Lipow. "Quantitative Evaluation of Software Quality". In: *Proceedings of the 2Nd International Conference on Software Engineering*. ICSE '76. San Francisco, California, USA: IEEE Computer Society Press, 1976, pp. 592–605. URL: http://dl.acm.org/citation.cfm?id=800253.807736.

[5] M. Genero and M. Piattini and C. Calero. "Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics". In: 9^{th} *International Symposium on Software Metrics (Metrics 2003)*. Proceedings IEEE Computer Society, 2003, pp. 263–275.

[6] M. Genero and M. Piattini and C. Calero. "Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics". In: 9^{th} *International Symposium on Software Metrics (Metrics 2003)*. Proceedings IEEE Computer Society, 2003, pp. 263–275.

[7] G. Canfora. "Software Maintenance". In: *Software Engineering and Knowledge Engineering*. World Scientific Publishing Co.Pte.Ltd, 2001, pp. 91–93.

[8] Samira Si-Said Cherfi and Nicolas Prat. "Multidimensional Schemas Quality: Assessing and Balancing Analyzability and Simplicity". In: *Conceptual Modeling for Novel Application Domains*. Ed. by Manfred A. Jeusfeld and Óscar Pastor. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 140–151.

[9] S. R. Chidamber and C. F. Kemerer. "A metrics suite for object oriented design". In: *IEEE Transactions on Software Engineering* 20.6 (June 1994), pp. 476–493. DOI: 10.1109/32.295895.

[10] Munkhnasan Choinzon and Yoshikazu Ueda. "Detecting Defects in Object Oriented Designs Using Design Metrics". In: *Proceedings of the 2006 Conference on Knowledge-Based Software Engineering: Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2006, pp. 61–72. URL: http://dl.acm.org/citation.cfm?id=1565098.1565107.

[11] Arie van Deursen, Paul Klint, and Joost Visser. "Domain-specific Languages: An Annotated Bibliography". In: *SIGPLAN Not.* 35.6 (June 2000), pp. 26–36. DOI: 10.1145/352029.352035. URL: http://doi.acm.org/10.1145/352029.352035.

[12] R. Geoff Dromey. "A Model for Software Product Quality". In: *IEEE Trans. Softw. Eng.* 21.2 (Feb. 1995), pp. 146–162. DOI: 10.1109/32.345830. URL: http://dx.doi.org/10.1109/32.345830.

[13] Holger Eichelberger. "On Class Diagrams, Crossings and Metrics". In: *Graph Drawing*. 2005.

[14] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. 3nd. Boston, MA, USA: PWS Publishing Co., 2014.

[15] Ana M. Fernández-Sáez, Marcela Genero, Danilo Caivano, and Michel R. V. Chaudron. "Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments". In: *Empirical Software Engineering* 21.1 (Feb. 2016), pp. 212–259. DOI: 10.1007/s10664-014-9354-4. URL: https://doi.org/10.1007/s10664-014-9354-4.

[16] M. Genero. "Early Measures for UML Class Diagrams". In: Hermes Science Publications, 2003, pp. 263–275.

[17] Marcela Genero, Jose A. Olivas, Mario Piattini, and Francisco Romero. "Knowledge Discovery For Predicting Entity Relationship Diagram Maintainability". In: (2001). URL: https://www.researchgate.net/publication/221389885_Knowledge_Discovery_For_Predicting_Entity_Relationship_Diagram_Maintainability.

[18] Marcela Genero, Geert Poels, and Mario Piattini. "Defining and validating metrics for assessing the understandability of entity-relationship diagrams". In: *Data & Knowledge Engineering* 64.3 (2008), pp. 534–557. DOI: https://doi.org/10.1016/j.datak.2007.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0169023X07001796.

[19] Carlos A. González and Jordi Cabot. "Formal verification of static software models in MDE: A systematic review". In: *Information and Software Technology* 56.8 (2014), pp. 821–838. DOI: https://doi.org/10.1016/j.infsof.2014.03.003. URL: http://www.sciencedirect.com/science/article/pii/S0950584914000627.

[20] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.

[21] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. "MontiCore: A Framework for the Development of Textual Domain Specific Languages". In: *Companion of the 30th International Conference on Software Engineering*. ICSE Companion '08. Leipzig, Germany: ACM, 2008, pp. 925–926. DOI: 10.1145/1370175.1370190. URL: http://doi.acm.org/10.1145/1370175.1370190.

[22] Object Management Group. *Model Driven Architecture (MDA) MDA Guide rev. 2.0*. OMG Document Number formal/2014-06-01 (https://www.omg.org/mda/). 2014.

[23] J.D. Haan. *A metaphor for Model Driven Engineering*. http://www.theenterprisearchitect.eu/blog/2009/08/05/a-metaphor-for-model-driven-engineering/. 2009.

[24] ISO/IEC. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Tech. rep. 2010.

[25] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[26] S. Kesh. "Evaluating the quality of entity relationship models". In: *Information and Software Technology* 37.12 (1995), pp. 681–689. DOI: https://doi.org/10.1016/0950-5849(96)81745-9. URL: http://www.sciencedirect.com/science/article/pii/0950584996817459.

[27] Someswar Kesh. "Evaluating the quality of entity relationship models". In: *Information and Software Technology* 37.12 (1995), pp. 681–689. DOI: https://doi.org/10.1016/0950-5849(96)81745-9. URL: http://www.sciencedirect.com/science/article/pii/0950584996817459.

[28] N. King. "The qualitative research interview". In: *Qualitative methods in organizational research: A practical guide*. 1994.

[29] W. Li and S. Henry. "Maintenance metrics for the object oriented paradigm". In: *Proceedings First International Software Metrics Symposium*. May 1993, pp. 52–60. DOI: 10.1109/METRIC.1993.263801.

[30] Yao Lu, XinJun Mao, and Zude Li. "Maintainability Based on Class Diagram Design : A Preliminary Case Study". In: 2016.

[31] M.Piattini M. Genero G. Poels. "Defining and validating metrics for assessing the understandability of entity–relationship diagrams". In: *Data and Knowledge Engineering* 64.3 (2008), pp. 534–557. DOI: https://doi.org/10.1016/j.datak.2007.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0169023X07001796.

[32] C. Calero M. Piattini M. Genero. "Data Model Metrics". In: (2001).

[33] S. Marche. "Measuring the stability of data models". In: *European Journal of Information Systems* 2.1 (Jan. 1993), pp. 37–47. DOI: 10.1057/ejis.1993.5. URL: https://doi.org/10.1057/ejis.1993.5.

[34] M. Marchesi. "OOA Metrics for the Unified Modeling Language". In: *Proceedings of second Euromicro Conference on Software Maintenance and Reengineering*. Palazzo degli Affari, 1998, pp. 67–73.

[35] T. J. McCabe. "A Complexity Measure". In: *IEEE Transactions on Software Engineering* SE-2.4 (Dec. 1976), pp. 308–320. DOI: 10.1109/TSE.1976.233837.

[36] J McCall. *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*. Vol. 1-3. ADA049055. General Electric, Nov. 1977. URL: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049055.

[37] Jose P. Miguel, David Mauricio, and Glen Rodriguez. "A Review of Software Quality Models for the Evaluation of Software Products". In: *CoRR* abs/1412.2977 (2014). arXiv: 1412.2977. URL: http://arxiv.org/abs/1412.2977.

[38] P. Mohagheghi and J. Aagedal. "Evaluating Quality in Model-Driven Engineering". In: *International Workshop on Modeling in Software Engineering (MISE'07: ICSE Workshop 2007)*. May 2007, pp. 6–6. DOI: 10.1109/MISE.2007.6.

[39] P. Mohagheghi and V. Dehlen. "Existing model metrics and relations to model quality". In: *2009 ICSE Workshop on Software Quality*. May 2009, pp. 39–45. DOI: 10.1109/WOSQ.2009.5071555.

[40] Daniel L. Moody and Andrew Flitman. "A Methodology for Clustering Entity Relationship Models — A Human Information Processing Approach". In: *Conceptual Modeling — ER '99*. Ed. by Jacky Akoka, Mokrane Bouzeghoub, Isabelle Comyn-Wattiau, and Elisabeth Métais. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 114–130.

[41] D.L. Moody. "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions". In: *Data & Knowledge Engineering* 55.3 (2005). Quality in conceptual modeling, pp. 243–276. DOI: https://doi.org/10.1016/j.datak.2004.12.005. URL: http://www.sciencedirect.com/science/article/pii/S0169023X04002307.

[42] M. Mrerearchesi. "OOA Metrics for the Unified Modeling Language". In: *Proceedings of second Euromicro Conference on Software Maintenance and Reengineering*. Palazzo degli Affari, 1998, pp. 67–73.

[43] Ariadi Nugroho, Bas Flaton, and Michel R. V. Chaudron. "Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density". In: *Model Driven Engineering Languages and Systems: 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings*. Ed. by Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 600–614. DOI: 10.1007/978-3-540-87875-9_42. URL: https://doi.org/10.1007/978-3-540-87875-9_42.

[44] Nicholas Pippenger. "Complexity Theory". In: *Scientific American* 238.6 (1978), 114–125B. URL: http://www.jstor.org/stable/24955758.

[45] V. Rajlich. "Program Reading and Comprehension". In: *Proc. of summer school on Engineering of Existing Software*. Giuseppe Laterza Editore, 1994, pp. 161–178.

[46] Fabrizio Riguzzi and Fabrizio Riguzzi. *A Survey of Software Metrics*.

[47] M. Genero and M. Piattini and J. Olivas and F. Romero. "A controlled experiment for validating class diagram structural complexity metrics". In: 8^{th} *International Conference on object-oriented Information Systems (OOIS 2002)*. Montpellier, 2002, pp. 372–383.

[48] A. F. Rosene, J. E. Connolly, and K. M. Bracy. "Software Maintainability - What It Means and How to Achieve It". In: *IEEE Transactions on Reliability* R-30.3 (Aug. 1981), pp. 240–245. DOI: 10.1109/TR.1981.5221065.

[49] D. C. Schmidt. "Guest Editor's Introduction: Model-Driven Engineering". In: *Computer* 39.2 (Feb. 2006), pp. 25–31. DOI: 10.1109/MC.2006.58.

[50] D. C. Schmidt. "Guest Editor's Introduction: Model-Driven Engineering". In: *Computer* 39.2 (Feb. 2006), pp. 25–31. DOI: 10.1109/MC.2006.58.

[51] Manuel Serrano, Juan Trujillo, Coral Calero, and Mario Piattini. "Metrics for data warehouse conceptual models understandability". In: *Information and Software Technology* 49.8 (2007), pp. 851–870. DOI: https://doi.org/10.1016/j.infsof.2006.09.008. URL: http://www.sciencedirect.com/science/article/pii/S0950584906001327.

[52] Jonathan Sprinkle and Gabor Karsai. "A domain-specific visual language for domain model evolution". In: *Journal of Visual Languages & Computing* 15.3 (2004). Domain-Specific Modeling with Visual Languages, pp. 291–307. DOI: https://doi.org/10.1016/j.jvlc.2004.01.006. URL: http://www.sciencedirect.com/science/article/pii/S1045926X0400014X.

[53] Srdjan Stevanetic and Uwe Zdun. "Software Metrics for Measuring the Understandability of Architectural Structures: A Systematic Mapping Study". In: *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. EASE '15. Nanjing, China: ACM, 2015, 21:1–21:14. DOI: 10.1145/2745802.2745822. URL: http://doi.acm.org/10.1145/2745802.2745822.

[54] Giancarlo Succi, Witold Pedrycz, Snezana Djokic, Paolo Zuliani, and Barbara Russo. "An Empirical Exploration of the Distributions of the Chidamber and Kemerer Object-Oriented Metrics Suite". In: *Empirical Software Engineering* 10.1 (Jan. 2005), pp. 81–104. DOI: 10.1023/B:EMSE.0000048324.12188.a2. URL: https://doi.org/10.1023/B:EMSE.0000048324.12188.a2.

[55] M. Voelter T. Stahi. "MDSD – Basic Ideas and Terminology". In: *Model-Driven Software Development*. John Wiley & Sons Ltd, 2006, pp. 11–27.

[56] Douglas A. Troy and Stuart H. Zweben. "Measuring the Quality of Structured Designs". In: *J. Syst. Softw.* 2.2 (June 1981), pp. 113–120. DOI: 10.1016/0164-1212(81)90031-5. URL: http://dx.doi.org/10.1016/0164-1212(81)90031-5.

[57] Nitin Upadhyay, Bharat M. Despande, and Vishnu P. Agrawal. "Towards a Software Component Quality Model". In: *Advances in Computer Science and Information Technology*. Ed. by Natarajan Meghanathan, Brajesh Kumar Kaushik, and Dhinaharan Nagamalai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 398–412.

[58] Klaas van den Berg and P.M. van den Broek. "Validation in the Software Metric Development Process". Undefined. In: *Memoranda informatica* 95-10 (Feb. 1995),

[59] M. Voelter. "Modeling and Model-Driven Development". In: *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. 2013, pp. 31–34.

[60] E. J. Weyuker. "Evaluating software complexity measures". In: *IEEE Transactions on Software Engineering* 14.9 (Sept. 1988), pp. 1357–1365. DOI: 10.1109/32.6178.

[61] ASML tech wiki. *Guidelines for data models*. (https://techwiki.asml.com/index.php/Guidelines_for_datamodels).

[62] F. Wu and T. Yi. "A Structural Complexity Metric for Software Components". In: *The First International Symposium on Data, Privacy, and E-Commerce (ISDPE 2007)*. Nov. 2007, pp. 161–163. DOI: 10.1109/ISDPE.2007.127.

[63] Y. Zhou and B. Xu. "Measuring Structure Complexity of UML Class Diagrams". In: *Journal of Electronics (China)*. Vol. 20(3). 2003, pp. 227–231.

[64] Y. Zhou and B. Xu. "Measuring Structure Complexity of UML Class Diagrams". In: *Journal of Electronics (China)*. Vol. 20(3). 2003, pp. 227–231.

[65] Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman. *Model-Based Testing for Embedded Systems*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 2011.