On Metamodel-Based Design of Software Metrics

Erki Eessaar

Department of Informatics, Tallinn University of Technology,
Raja 15, 12618 Tallinn, Estonia
eessaar@staff.ttu.ee

Abstract. Metric values can be used in order to compare and evaluate software entities, find defects, and predict quality. For some programming languages there are much more known metrics than for others. It would be helpful, if one could use existing metrics in order to find candidates for new metrics. A solution is based on an observation that it is possible to specify abstract syntax of a language by using a metamodel. In the paper a metrics development method is proposed that uses metamodel-based translation. In addition, a metamodel of a language helps us to find the extent of a set of metrics in terms of that language. That allows us to evaluate the extent of the core of a language and to detect possible quality problems of a set of metrics. The paper contains examples of some candidate metrics for object-relational database design, which have been derived from existing metrics.

Keywords: Metric, Measure, Metamodel, UML, Object-relational database, Data model, Reusability.

1 Introduction

Metrics, the values of which characterize software designs, can be used in order to compare designs, find defects, and predict quality. For example, Choinzon and Ueda [1] refer to 22 *object-oriented design* metrics that are presented in the research literature. In addition, they define 18 new design metrics. There are fewer metrics that allow us to evaluate database designs. For example, Piattini et al. [2] present three table oriented metrics for *relational* databases. Piattini et al. [3] present twelve metrics that help us to evaluate the design of an *object-relational* database. Muller [4] proposes to evaluate structural cohesion of tables based on their normal forms.

Metamodeling is a well-known activity in software engineering that allows us to specify abstract syntax of a language. Seidewitz [5] writes that a metamodel "*makes statements about what can be expressed in the valid models of a certain modeling language*." If we use UML as a metamodeling language, then language elements and their relationships are presented by using classes/attributes (properties) and attributes/relationships, respectively [6]. Is it possible to use metamodels in order to create and improve metrics? McQuillan and Power [7] write that definitions of metrics should be *reusable*. Researchers have used *metamodels* and *ontologies* in order to *present* object-oriented design metrics [8] and database design metrics [9], respectively, as precisely as possible. For example, SQL:2003 [10] is a large international standard that specifies the database programming language SQL. An SQL:2003 ontology [9] is presented by using UML. The ontology resembles a metamodel. A

B. Meyer, J.R. Nawrocki, and B. Walter (Eds.): CEE-SET 2007, LNCS 5082, pp. 40–54, 2008.

difference with a metamodel of SQL:2003 is that the ontology follows the *principle of minimal ontological commitment* [11] and therefore covers *only* the most important parts of SQL:2003 instead of specifying the entire language.

Baroni et al. [12] think that an SQL:2003 ontology, which is a step towards a complete SQL:2003 metamodel, helps us to prevent ambiguity in metrics specifications and automate the collection process of metrics values.

In this paper, we propose *additional* means for using metamodels in the development of metrics. We assume that metrics that belong to a set M help us to evaluate software entities that are created by using a language L. Models, patterns, and fragments of code are examples of software entities.

The *first goal* of the paper is to propose a *metamodel-based* method for creating *candidate* metrics. This novel method uses a metamodel-based translation and allows us to *reuse* existing metrics specifications. Such method could be used in case of any software development language if a metamodel of the language is available.

The *second goal* of the paper is to propose a *metamodel-based* method for calculating the *extent* of a set of metrics M in terms of a language L. This method allows us to find concrete numerical estimates of the size of the core of L as the designers of metrics see it. A small extent of M is a sign of *possible* quality problems of M because M may be incomplete. For example, McQuillan and Power [7] note that existing UML metrics deal only with a small part of all the possible UML diagram types. The existing metrics evaluation methods [3, 13, 14] do not take into account whether all the metrics, which belong to a set of related metrics, together help us to evaluate all (or at least most of the) parts of a software entity.

The data model, based on which a database system (DBMS) is implemented, is a kind of abstract language [15]. In this work, we investigate two *object-relational* data model approaches as the examples:

1. The underlying data model of SQL:2003 (OR_{SQL}) [10].
2. The underlying data model of The Third Manifesto (OR_{TTM}) [16].

We have found few metrics about OR_{SQL} database design and no metrics about OR_{TTM} database design.

The *third goal* of the paper is to use the proposed metamodel-based methods in order to evaluate the existing OR_{SQL} database design metrics and to show how to develop an OR_{TTM} database design metric based on an OR_{SQL} database design metric.

The rest of the paper is organized as follows. Section 2 analyzes how we can use metamodels of languages in order to create and evaluate metrics. In Section 3, we present examples. Firstly, we evaluate some OR_{SQL} database design metrics in terms of an OR_{SQL} metamodel. Secondly, we design some candidate OR_{TTM} database design metrics based on a set of OR_{SQL} database design metrics. Thirdly, we find the extent of some sets of metrics. Finally, Section 4 summarizes the paper.

2 On Using Metamodels in the Development of Metrics

Piattini et al. [3] and IEEE Standard for a Software Quality Metrics Methodology [17] describe frameworks of metrics development. They do not propose the *reuse* of existing

metrics as one possible method how to find *candidate metrics*. A candidate metric is a metric that has not yet been approved or rejected by experts.

We think that it is not always necessary to start development of a metric from scratch. Instead, we could try to reuse existing metrics. The *motivation* of this approach is that it allows us to create quickly candidate metrics and experiment with them in order to improve our understanding of a domain and get new ideas. In addition, candidate metrics provide a communication basis for discussions among all groups that are involved in the development of a new set of metrics. It is possible that a candidate metric evolves and becomes accepted and validated metric or the candidate metric is rejected after evaluation. The proposed approach should *complement* existing methods of metrics development but not replace them. Figure 1 presents the concepts that are used in the proposed approach and their interconnections.

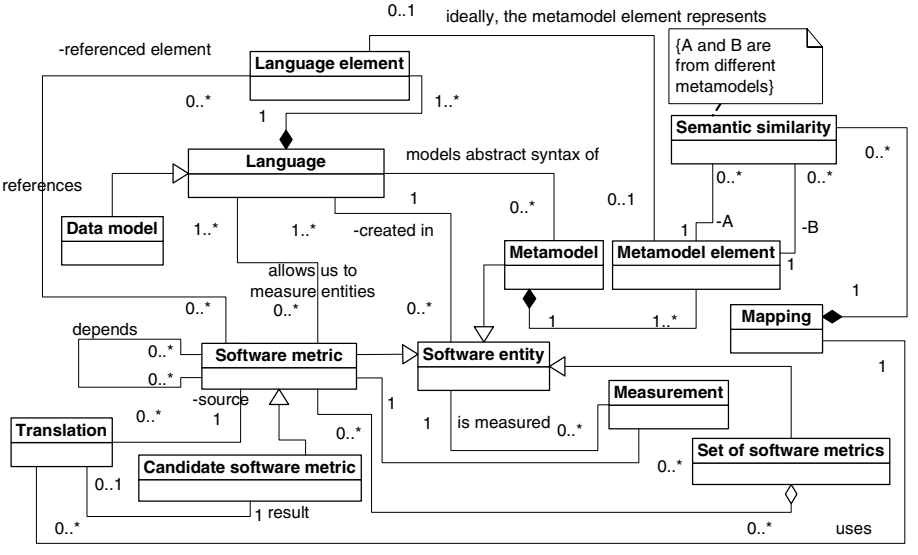

Fig. 1. A domain model of the proposed approach

Each language consists of one or more language elements. It is possible to represent abstract syntax of a language by using a metamodel. A language can have different metamodels, which are for instance created by different parties or are presented with the help of different languages. A metamodel, a software metric, and a set of software metrics are examples of software entities. Each software entity is created by using one or more languages. For example, a metamodel of UML [18] consists of UML diagrams, OCL expressions, and free-form English text. Another example is that OR_{SQL} metrics [9] are presented by using OCL expressions and free-form English text. A language can have associated metrics that can be used in order to measure properties of the software entities that are created by using this language. Each metamodel consists of one or more metamodel elements. There could exist mappings between elements of different metamodels that allow us to create candidate metrics by

using metamodel-based translation. A metric could be calculated based on values of other metrics.

Let us assume that the metrics that belong to a set M help us to evaluate software entities that are created by using a language L. Let us also assume that there is a language L', the corresponding metrics of which belong to a set M'. All the methods that are proposed in this paper require the existence of the *metamodels* of L and L' and also the existence of *mapping* of elements of L and L' metamodels. If UML is used in order to create these metamodels, then the elements that must participate in the mapping are *classes*. For example, a metamodel of UML [18] contains classes like "class", "action", and "actor" and a metamodel of OR_{SQL} [19] contains classes like "data type", "constructed data type", and "data type constructor". We follow the example of Opdahl and Henderson-Sellers [20], who evaluate a language based on classes of a metamodel and do not use a mapping between relationships and a mapping between attributes.

A pair of elements (that are from the different metamodels) exists in the mapping if the constructs behind these elements have exactly the same semantics or they are semantically quite similar. Designers of L and L' and users of both these languages are the experts who are the best suited to decide whether the *semantic similarity* of the underlying constructs of two elements is big enough in order to place a pair of these elements into the mapping or not. Ideally, these mappings should be *standardized*.

The use of mapping of elements of metamodels in order to evaluate languages or translate models is not a new idea. However, we use this approach in a *new context*. For example, ontological evaluation of a language is a comparison of the concrete classes of a language metamodel (language constructs) with the concepts of an ontology in order to find ontological discrepancies: construct overload, construct redundancy, construct excess, and construct deficit [20]. Opdahl and Henderson-Sellers [20] use UML metamodel in order to perform an ontological evaluation of UML by comparing it with Bunge–Wand–Weber (BWW) model of information systems. Researchers have proposed metamodel-based comparison of ontologies [21]. It is also possible to compare two languages by using their metamodels. For example, researchers have proposed metamodel-based comparison of data models [19, 22]. The work of Levendovszky et al. [23] is an example of study about metamodel-based model transformations from one language to another.

2.1 New Means of Using Metamodels in Metrics Development

In this section, we present some new means of using metamodels of languages L and L' in order to create and improve metrics that belong to the sets M and M', respectively. We will present examples of the use of these means in Section 3.

1. A metamodel of L helps us to find shortcomings in the specification of individual metrics that belong to M. We have to make sure that all the language elements that are referenced in a specification of a metric (the set of these language elements is X) have a corresponding element in a metamodel of L (the set of all the metamodel elements is Y; there is a total injective function f: $X \rightarrow Y$) and X is the same in all the different specifications of the same metric. If these conditions are not fulfilled, then

it shows us that the wording of a metric may not be precise enough. The result of this investigation could be improved wording of the specifications of metrics or the creation of new candidate metrics.
2. It is possible to develop candidate metrics for L' (that belong to M') by translating metrics that belong to M. This translation is based on a mapping of elements of metamodels of languages L and L'.
3. A metamodel of L helps us to evaluate the extent of M and find the elements of L that are not covered by M. This *may* lead to the creation of new candidate metrics.

The quality of the results of the use of these means depends on the quality of a metamodel. For example, if a metric refers to a language element *l* (that belongs to L) but a metamodel of L has no element that represents *l*, then we will erroneously conclude that the metric is imprecise (see the first mean) because an element of X has no corresponding element in Y. This example stresses an importance of evaluation and standardization of metamodels.

2.1.1 Metamodel-Based Creation of a Candidate Metric

Let us assume that we want to translate a metric *m* from a set M in order to use it in case of software entities that are created by using L'. Next, we propose a method that allows us to develop metrics by using a *metamodel-based translation*:

1. Extract *nouns* from the text of a specification of *m*.
2. Find all the elements of L metamodel that correspond to the nouns that are found during step 1. It allows us to find language elements, based on which a value of *m* is calculated.
3. For each element of L metamodel that is found during step 2, find a corresponding element of L' metamodel. *Discrepancies* of the metamodels will cause some problems:
   - If an element of L metamodel has more than one corresponding element of L' metamodel, then it is not possible to perform *automatic* translation and a human expert has to choose *one* corresponding element of L' metamodel.
   - If at least one of the found elements (see step 2) of L metamodel does not have a corresponding metamodel element of L' (there is a *construct deficit* in L'), then it is not possible to perform *automatic* translation. A human expert has to investigate whether it is possible to use any metamodel element of L'. If it is not possible, then the process finishes. As you can see, the bigger are the discrepancies between two languages, the harder it is to translate a metric.
4. In case of each element of L' metamodel (each class in case of UML) that is found during step 3, check whether it is part of a specialization hierarchy.
   - If a metamodel element *e'* is part of a specialization hierarchy, then a human user has to evaluate whether it is *instead* possible to use some direct or indirect *supertype* of *e'* in order to construct a metric for L'. If the use of a supertype is reasonable, then a metrics designer has to use this supertype *instead* of *e'* in order to construct a new metric. It ensures that this new metric can be used in as many cases as possible.
5. Use the names of all the selected metamodel elements of L' (from step 3, 4) in order to construct a candidate metric *m'*.
   - "Initialism is an abbreviations formed from initial letters" [24]. If *m* has an initialism, then create an initialism of *m'* based on *m*. Firstly, we have to identify a

phrase or name based on which an initialism of *m* is created. Secondly, we have to find the corresponding name or phrase in *m'*. Finally, we have to use initial letters of words in this phrase or name in *m'* in order to construct an initialism to *m'*.

6. Validate the new candidate metric *m'* formally and empirically in order to accept or reject it. The validation procedure is not the subject of this paper. However, there are already a lot of studies about evaluation of metrics [3, 13, 14].

If a metric *m* is a derived metric, the value of which is calculated based on the values of a set of metrics, then we firstly have to translate metrics that belong to this set before we can translate *m*.

For instance, the proposed method could be used in order to translate metrics of UML models [25] or OR_{SQL} database designs to metrics that could be used in case of Object-Process models [26] or OR_{TTM} database design, respectively. This would allow us to quickly find some metrics and to start their evaluation.

A poblem is that if the quality of an initial metric is low, then the quality of a resulting metric will also be low. If an initial metric has associated tresholds of undesirable values [1], then we cannot use them in case of a new metric, without extensive testing. It is also possible that a new metric will become less important than the original, because languages L and L' could pay attention to different things and hence different parts of these languages are important to the designers. A new metric might be about relatively unimportant part.

The existence of this method makes it possible to at least partially automate translation of metrics. It is not possible to fully automate it because sometimes a human expert has to make decisions (see steps 3, 4, 6).

2.1.2 Metamodel-Based Calculation of the Extent of a Set of Metrics

It is possible that some elements of a language L are not taken into account by *any* metric in M. The percentage of the metamodel elements that are covered by at least one metric in M shows us the *extent* of M in terms of L. The extent of M (we denote it E(M)) is a candidate *metric* that helps us to evaluate M in terms of *completeness*. E(M) value is a percentage. The bigger the value is, the more complete is M.

More precisely, let us assume that we use UML in order to create metamodels. If we calculate the value of E(M), then we have to take into account a mapping MA between metrics that belong to M and classes in a metamodel of L. MA contains a pair of a metric *m* and a class *c*, if the calculation formula of *m* takes into account a language element that is presented by *c*.

We can calculate E(M) based on the formula (1) where:

- a is the *total number* of different classes of a metamodel of L, which participate in at least one pair in MA, and their direct or indirect subclasses. We should not count any class more than once. For example, if two classes in the mapping have the same subclass, then we have to count this subclass only once.
- b is the *total number* of all classes in a metamodel of L.

$$E(M) = a*100/b \; . \tag{1}$$

All the elements of a metamodel of L that do not participate in any pair in MA represent the parts of L that are not covered by the metrics in M.

We try to measure *completness* of a set of metrics by using this metric. We have to use *matching* of metrics and metamodel elements and *counting* of matches and metamodel elements in order to calculate this metric. This metric can be used within and across projects and workgroups that deal with the development of metrics or decide the use of particular metrics in a particular project.

Firstly, if we assume that metrics should pay attention only to the most important elements of L, then E(M) shows us the extent of the *core* of L as the designers of metrics see it. If this core is small, then it raises a question whether L containts unnecessary elements. If a set of metrics M has small E(M) value, then it does not *necessarily* mean that this set has quality problems. Different parts of a language could contribute differently to the overall quality of a software entity, that is created by using L. However, a small E(M) value points to the *possible* quality problems of M, because M might be incomplete and therefore additional investigation is needed.

A language could have more than one metamodel. For example, they could be created by different parties or by using different languages. It is possible, that:

1. Different metamodels specify different sets of language elements. For instance, CIM (Common Information Model) is a conceptual information model that specifies different areas of information technology management. Part of CIM Database Model [27] is a model of SQL Schema. It presents only eight classes that correspond to the constructs that are specified in the SQL standard [10]. On the other hand, the OR_{SQL} metamodel [19] contains 110 classes.
2. In one metamodel a relationship between language elements is presented with the help of an association class but in another metamodel by using an association. For instance, Baroni et al. [12] use associations in order to model relationships between classes *Referential constraint* and *Column*. On the other hand, the OR_{SQL} metamodel [22] contains association classes *Referencing column* and *Referenced column* in order to specify these relationships.
3. In one metamodel a language element is presented with the help of an attribute but in another metamodel by using a class. For instance, CIM Database Model [27] contains class *SqlDomain* that has attribute *DataType*. There is no separate class *DataType* in CIM Database Model. On the other hand, *Data type* is a separate class in the OR_{SQL} metamodel [22].

There could also be similar differences between different versions of the same metamodel. Therefore, we can find different E(M) value for the same set of metrics if we use different metamodels. It means that each E(M) value should always be accompanied with the information about the metamodel (including its version) based on which it is calculated. If a language has more than one set of metrics and we want to compare these sets in terms of E(M), then we have to use the same metamodel version in order to calculate E(M) values.

A possible negative side efect of the use of this metric is the creation of simplistic and unuseful metrics in order to increase the value of E(M).

Empirical validation of a metric should involve case studies [3]. The next section contains a case study about the use of E(M).

3 Case Study: Object-Relational Database Design Metrics

In this section, we demonstrate and analyze the use of metamodel-based methods that allow us to develop and analyze metrics. We introduced them in Section 2.

The concept "data model" has different meanings in different contexts. In this paper a *data model* is an abstract, self-contained, implementation-independent definition of elements of a set of sets {T, S, O, C} that together make up the abstract machine with which database users interact. In this case: T is a set of data types and types of data types; S is a set of data structures and types of data structures; O is a set of operators and types of operators; C is a set of constraints and types of constraints. This is a revised version of the definition that is presented by Date [15] and our previous definition [19]. Relational and object-relational data model are examples of this kind of data models. These data models are abstract languages [15] and we can use the methods that were presented in Section 2 in order to create and improve their corresponding metrics.

In this section, we investigate the object-relational (OR) data model. This model should combine the best properties of the relational data model and object-oriented programming languages. Currently there is no common OR data model yet. The work of Seshadri [28], 3rd- generation DBMS manifesto [29], The Third Manifesto (OR_{TTM}) [16], the work of Stonebraker et al. [30, 4], and SQL: 2003 (OR_{SQL}) [10] are all examples of different OR data model approaches. However, they have significant differences. For example, all the approaches from the set of previously mentioned approaches support the idea of an abstract data type system that allows designers to construct new types. However, there are different opinions about the exact nature of this system. For example, only 3rd- generation DBMS manifesto [29] and SQL: 2003 [10] propose the use of array type constructor. On the other hand, only Stonebraker et al. [30, 4] and SQL: 2003 [10] propose the use of reference type constructors. Eessaar [22] presents metamodels of OR_{SQL} and OR_{TTM} and their metamodel-based comparison.

More precisely, in this section we investigate OR_{SQL} and OR_{TTM} database design metrics. Piattini et al. [3] propose twelve metrics in order to evaluate OR_{SQL} database designs. We denote the set of these metrics as M_{ORSQL}. We are not aware of database design metrics, the specification of which uses OR_{TTM} terminology and which are created specifically for OR_{TTM}. Therefore, a task of this section is to investigate, how to create candidate OR_{TTM} database design metrics.

3.1 On Evaluating the Wording of Existing OR_{SQL} Database Design Metrics

A metamodel of a language (a data model in this case) allows us to find shortcomings in the specifications of metrics. A metamodel, is in this case a kind of aiding tool.

A metrics designer has to check, whether all the language elements that are referred in various specifications of a metric have exactly one corresponding element in a metamodel of the language or whether there are inconsistencies. For example, some specifications of the metrics that belong to M_{ORSQL} refer to "complex columns". The OR_{SQL} metamodel [22] does not have a class "complex column" and OR_{SQL} specification [10] does not refer to this concept. In addition, Piattini et al. [3] do not give exact definition of "complex column". Baroni et al. [9] write that a complex column has a structured type. However, a user-defined type is a structured type or a distinct type in

OR_{SQL}. In addition, OR_{SQL} allows us to use *constructed types* (multiset type, array type, row type) as declared types of columns. Both base and viewed tables can have columns, the declared type of which is not a predefined data type.

A metrics designer has also to check, whether all specifications of the same metric refer to exactly the same set of metamodel elements. For example, informally, a value of metric PCC(T) is "percentage of complex columns of a table T" [3]. Based on a metamodel of OR_{SQL} [22], we can see that a table is a base table, a transient table or a derived table (these classes form a specialization hierarchy). A viewed table (view) is a derived table. However, Piattini et al. [3] do not indicate, whether PCC(T) considers only base tables or also viewed tables. They are both schema objects. Baroni et al. [9] presents PCC(T) more formally by using OCL and shows that a PCC(T) value is calculated only based on *base tables*.

These examples illustrate that (1) informal specifications metrics should be more precise and (2) we need additional metrics that would take into account viewed tables, distinct types and constructed types.

3.2 On Designing OR_{TTM} Database Design Metrics Based on Existing Metrics

Table 1 presents mapping of some *classes* of the metamodels of OR_{SQL} and OR_{TTM}.

Table 1. Mapping of some classes of the metamodels of OR_{SQL} and OR_{TTM}

Class in the metamodel of OR_{SQL} [22]	*Class in the metamodel of OR_{TTM} [22]*
Base table	Real relvar, Relation
Typed base table	-
Structured type	User-defined scalar type
Base table column	Relvar attribute
Predefined data type	Built-in scalar type
Attribute	Attribute
SQL-invoked method	Read-only operator, Update operator
SQL-schema	-
Referential constraint	Referential constraint
Referencing column, Referenced column	-

Column "*Class in the metamodel of OR_{SQL}*" contains names of classes from the OR_{SQL} metamodel [22]. Name of a class exists in this column, if specification of at least one metric from the set M_{ORSQL} refers to a language element that has this corresponding class in a metamodel of OR_{SQL}. Column "*Class in the metamodel of OR_{TTM}*" contains names of the corresponding classes in the OR_{TTM} metamodel [22]. A pair of classes from the metamodels of OR_{SQL} and OR_{TTM} exists in the mapping, if these classes represent language elements that are semantically equivalent or significantly similar.

Next, we present examples of manual resolution of *construct deficit* problem that was described in step 3 of the algorithm in Section 2.1.1. The Third Manifesto argues explicitly against pointers at the *logical* database level and typed tables (including *typed base tables*) in the section "OO Prescriptions" [16]. Therefore, we cannot completely translate metric *Table size of a table T* that belongs to M_{ORSQL}.

Schema Size is a metric from M_{ORSQL}. A database is a named container of database relational variables (relvars) in OR_{TTM} [16]. OR_{SQL}, on the other hand, does not use the concept "Database". Instead it uses concepts "SQL-schema", "Catalog" and "Cluster", which are all collections of objects. An object is a cluster, a catalog, a SQL-schema, or a schema object. The OR_{SQL} metamodel class "SQL-schema" has no corresponding class in the OR_{TTM} metamodel. We think that in case of OR_{TTM} we could instead calculate *Database Size* (DS) instead of Schema Size. DS is sum of the size of every relvar in a database (a metric for estimating the size of a relvar must also be translated from OR_{SQL}).

Depth of relational tree of a table T DRT(T) is a metric from M_{ORSQL} that shows us "the longest path between a table and the remaining tables in the schema database" [9]. We have created classes *Referencing Column* and *Referenced Column* in the OR_{SQL} metamodel in order to model associations between *Base table column* and *Referential constraint*. Classes *Referencing Column* and *Referenced Column* are necessary in the OR_{SQL} metamodel because OR_{SQL} pays attention to the order of column names in a referential constraint specification and we need a place for the attribute *ordinal_position*. It is possible (but not necessary) to create corresponding classes in the OR_{TTM} metamodel. However, these classes would not have any attributes (including *ordinal_position*, because OR_{TTM} does not pay attention to the order of attribute names in a referential constraint specification). In addition, metrics in M_{ORSQL} do not take into account the ordinal position and therefore we conclude that it is possible to find corresponding metrics for DRT(T) in OR_{TTM} despite the construct deficit.

3.2.1 An Example

Next, we demonstrate how to create candidate OR_{TTM} database design metrics based on the OR_{SQL} metrics by using the algorithm that was introduced in Section 2.1.1. We investigate metrics NFK(T) and RD(T) that belong to the set M_{ORSQL} [3]. Baroni et al. [9] present specifications of NFK(T) and RD(T) in the following way:

- "NFK (Number of Foreign Keys): Number of foreign keys defined in a table.

```
BaseTable:: NFK(): Integer= self.foreignKeyNumber()
```

- RD (Referential Degree): Number of foreign keys in a table divided by the number of attributes of the same table.

```
BaseTable::RD(): Real= self.NFK() / (self.allColumns()
-> size())"
```

These specifications consist of a natural language part and are also presented by using OCL, which arguably makes them more formal and understandable. Unfortunately, Baroni et al. [9] do not specify functions that are used in the OCL specification. We note that tables have *columns* and structured types have *attributes* according to the OR_{SQL} metamodel [22]. As you can see, analysis with the help of a metamodel may help us to improve the existing wording of metrics.

RD is an example of a metric that depends on another metric (NFK) and therefore we have to firstly translate NFK. We also note that metric *Referential Degree of a table T* (RD(T)) has different semantics in the studies of Piattini et al. [3] and Baroni et al. [9] and it causes confusion. Piattini et al. [3] defines RD(T) metric as "as the

number of foreign keys in the table T". The corresponding metric in the work of Baroni et al. [9] is named *Number of Foreign Keys*.

Steps 1, 2: Relevant classes of the OR_{SQL} metamodel [22] are: *Base table*, *Base table column*, *Referential constrain* (see Table 1). We can find them by investigating nouns in the existing specifications of metrics.

Step 3: Table 1 presents classes of the OR_{TTM} metamodel that correspond to some classes of the OR_{SQL} metamodel. Firstly, some elements of the OR_{SQL} metamodel have more than one corresponding element in the OR_{TTM} metamodel. *Base table* has two corresponding classes in the OR_{TTM} metamodel – *Real relational variable (Real relvar)* and *Relational value (Relation).* OR_{TTM} clearly distinguishes the concepts "value" and "variable". A variable has at any moment one value, but it is possible to change this value. In OR_{SQL}, the concept "table" means "table value" as well as "table variable". The next definition is an example of that: "*A table is a collection of rows having one or more columns*" [10]. It is an example of *construct overload* [20] in OR_{SQL} because a construct in OR_{SQL} corresponds to several not-overlapping constructs in OR_{TTM}. Date and Darwen [16] write that referential constraints apply to *relvars*. Therefore, we decide that the corresponding class to *Base table* is in this case *Real relvar*.

Step 4: We identified the concept "real relvar" during the step 2. Date and Darwen [16] write: "*Referential constraints are usually thought of as applying to real relvars only. In the Manifesto, by contrast, we regard them as applying to virtual relvars as well.*" *Real relvar* and *Virtual relvar* are subclasses of *Relvar* in the OR_{TTM} metamodel. Therefore, in this case we can use class *Relvar* instead of class *Real relvar*.

Step 5: Now we can create specifications of two candidate metrics for OR_{TTM} by replacing OR_{SQL} concepts in the specifications with OR_{TTM} concepts. The specification consists of an informal natural language specification and a specification that is written in OCL. The level of precision of the specifications is analogous to [9].

- NRC (Number of Referential Constraints): Number of referential constraints where a relvar is the referencing relvar.

```
Relvar:: NRC(): Integer=
self.referentialConstraintNumber()
```

- RD (Referential Degree): Number of referential constraints where a relvar is the referencing relvar divided by the number of attributes of the same relvar.

```
Relvar:: RD(): Real= self.NRC() /(self.allAttributes()
-> size())
```

We created initialisms NRC and RD based on the names "Number of Referential Constraints" and "Referential Degree", respectively.

We also note that we can translate some metrics that are not intended to database design, in order to find candidate database design metrics. For example, Habela [31] presents a *metamodel* of an object-oriented database system. Date [15] explains that classes in object-oriented systems correspond to scalar data types in OR_{TTM} databases. An attribute in a class corresponds to a component of a possible representation of a scalar type. A method of a class corresponds to an operator that has been defined in an OR_{TTM} database. Therefore, it is possible to translate some OO design metrics [1] to candidate

OR_{TTM} database design metrics. For example, *Number of Attributes (NOA)* [1] becomes to *Number of components in a possible representation of a given type* and *Number of Methods in a Class (NOM)* becomes to *Number of read-only operators, the return value of which has a given type*.

3.3 On Evaluating the Extent of Sets of Database Design Metrics

We could create a set of metrics for OR_{TTM} by translating all the metrics in M_{ORSQL}. We denote this set as M_{ORTTM}. In this section, we evaluate the extent of the metrics in M_{ORSQL} and M_{ORTTM} based on the formula (1) (see Section 2.1.2).

The OR_{SQL} metamodel contains 110 classes [19]. Table 1 refers directly to 11 classes of the metamodel. These classes have additional 20 *different* subclasses. Therefore, the extent of M_{ORSQL} is: $E(M_{ORSQL})=((11+20)*100)/110=28.2\%$. As you can see, more than two thirds of OR_{SQL} constructs are not covered by these metrics.

UML [18] allows us to use packages in order to group model elements and manage complexity. According to *definition* (see Section 3), a data model has *four* components. Eessaar [19, 22] proposes to create four corresponding packages in order to manage the complexity of a metamodel of a data model that is presented by using UML: *Data types*, *Data structures*, *Data operators,* and *Data integrity*.

Ideally, each metamodel element should belong to exactly one of these packages. However, Eessaar [19, 22] has found 3 classes of the OR_{SQL} metamodel that cannot be classified to any of these packages. Table 2 presents the extent of M_{ORSQL} in terms of each of these packages. It shows us, how much metrics in M_{ORSQL} pay attention to the *different aspects* of OR_{SQL} data model.

Table 2. The extent of M_{ORSQL} in terms of the different data model components

Data model component	*Amt. of classes and their subclasses in the mapping (a)*	*Total amt. of classes in a package (b) [19]*	*$E(M_{ORSQL})$ (a*100)/b*
Data types	11	38	28.9%
Data structures	14	26	53.8%
Data integrity	3	16	18.8%
Data operators	3	27	11.1%

Table 2 shows us that metrics in M_{ORSQL} pay attention mostly to the *structural* part of OR_{SQL}. This is in line with the claims of the authors of metrics in M_{ORSQL}, who see these metrics as *structural* metrics. The biggest advantage of OR data models is possibility to create new types and operators [15]. However, existing OR_{SQL} metrics should pay more attention to types and operators. We can say this because Table 1 does not refer to classes of the OR_{SQL} metamodel that specify language elements like constructed data types, distinct types, and regular SQL-invoked functions. Table CHECK constraints, viewed tables, and user-defined functions / stored procedures with no overloading are examples of *mandatory* SQL features [10] that are not covered by the existing metrics according to Table 1. Type constructors, domains, triggers, and sequence generators are examples of *optional* SQL features [10] that are not covered by the existing metrics according to Table 1. On the other hand, a metric in

M_{ORSQL} takes into account typed tables and structured types that are *optional* SQL features [10]. As you can see, there is not one-to-one correspondence between the core of SQL and the existing metrics that belong to M_{ORSQL}.

Next, we calculate the extent of M_{ORTTM} based on the OR_{TTM} metamodel in order to evaluate M_{ORTTM}. We assume that M_{ORTTM} covers the following classes (and their subclasses): *Relvar*, *User-defined scalar type*, *Relvar attribute*, *Built-in scalar type*, *Attribute*, *Read-only operator*, *Update operator*, *Referential constraint*, *Database*. These 9 classes have 29 subclasses. The OR_{TTM} metamodel contains 95 classes [19].

Therefore, the extent of M_{ORTTM} is: $E(M_{ORTTM})=((9+29)*100)/95=40\%$. This extent is bigger compared to the extent of M_{ORSQL}.

A possible reason could be that OR_{SQL} violates the *orthogonality* principle more than OR_{TTM} [16, 19, 22]. Date and Darwen [16] write that the orthogonality principle means that a deliberate attempt has been made to *avoid arbitrary restrictions* in combinations of different language constructs. For example, OR_{SQL} permits foreign key constraints *only* in base tables but OR_{TTM} in *all* relvars (including virtual). Therefore, M_{ORTTM} metrics are calculated based on bigger amount of *different* types of database objects compared to M_{ORSQL}.

It could be argued that some constructs of a data model cannot be used or misused in a way that affects the overall quality of database design and therefore corresponding metrics are not needed. However, why when to develop standards and systems that specify and allow us to create entities that are unnecessary and not very useful? Most of the current database design metrics that are proposed by researchers are simple counts that are not very precisely described. It rather seems that small E(M) values point to the need to continue development of OR_{SQL} and OR_{TTM} metrics.

4 Conclusions

In the paper, we investigated how to use metamodels of languages in order to evaluate and improve specifications of existing software metrics and to design candidate metrics.

We proposed a metamodel-based derivation method of candidate metrics and new candidate metric E(M) that allows us to evaluate completeness of sets of metrics. The metamodel-based derivation method allows us to reuse existing metrics by translating them so that they are possibly usable in a new context. Actual usefulness of these new candidate metrics must be found out based on careful evaluation. The evaluation procedure was not in the scope of the paper. The proposed method is not intended to replace existing methods of metrics development but should complement them. Currently it is too early to say whether its use will become common practice.

We demonstrated the usefulness of the proposed method based on database design metrics. The paper considered two object-relational data model approaches – SQL:2003 (OR_{SQL}) and The Third Manifesto (OR_{TTM}) as the examples. The analysis of some existing OR_{SQL} design metrics revealed problems in the wording of them. We demonstrated how to translate some existing OR_{SQL} metrics in order to create candidate metrics for evaluating OR_{TTM} database design. In the proposed case study the languages (data models) are relatively similar to each other. There would be more

discrepancies between metamodels if the languages are more different. It will allow us to translate fewer metrics and will reduce possibility of automatic metric translation.

We also found that the completeness of an existing set of OR_{SQL} metrics is small (E(M)≈28%). These metrics together cover only small part of all possible OR_{SQL} constructs. Closer investigation showed that these metrics do not pay enough attention to different kinds of data types and routines and therefore design of new metrics must continue.

Future work will include development of more OR_{TTM} database design metrics and further evaluation of E(M).

References

1. Choinzon, M., Ueda, Y.: Design Defects in Object Oriented Designs Using Design Metrics. In: 7th Joint Conference on Knowledge-Based Software Engineering, pp. 61–72. IOS Press, Amsterdam (2006)
2. Piattini, M., Calero, C., Genero, M.: Table Oriented Metrics for Relational Databases. Software Quality Journal 9(2) (June 2001)
3. Piattini, M., Calero, C., Sahraoui, H., Lounis, H.: Object-Relational Database Metrics. L'Object (March 2001)
4. Muller, R.J.: Database Design for Smarties. Morgan Kaufmann, San Francisco (1999)
5. Seidewitz, E.: What models mean. IEEE Software 20(5), 26–31 (2003)
6. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons, USA (2004)
7. McQuillan, J.A., Power, J.F.: On the application of software metrics to UML models. In: Model Size Metrics Workshop of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (2006)
8. Reißing, R.: Towards a Model for Object-Oriented Design Measurement. In: 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, pp. 71–84 (2001)
9. Baroni, A.L., Calero, C., Piattini, M., Abreu, F.B.: A Formal Definition for Object-Relational Database Metrics. In: 7th International Conference on Enterprise Information Systems (2005)
10. Melton, J.: ISO/IEC 9075-2:2003 (E) Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation) (August 2003)
11. Gruber, T.R.: Towards principles for the design of ontologies used for knowledge sharing. International Journal of Human Computer Studies 43(5/6), 907–928 (1995)
12. Baroni, A.L., Abreu, F.B., Calero, C.: Finding Where to Apply Object-Relational Database Schema Refactorings: An Ontology-Guided Approach. In: X Jornadas Sobre Inginiería del Software y Bases de Datos (2005)
13. Schneidewind, N.F.: Methodology for Validating Software Metrics. IEEE Trans. Softw. Eng. 18(5), 410–422 (1992)
14. Kaner, C., Bond, P.: Software Engineering Metrics: What Do They Measure and How Do We Know? In: 10th International Software Metrics Symposium (2004)
15. Date, C.J.: An Introduction to Database Systems, 8th edn. Pearson/Addison-Wesley, Boston (2003)
16. Date, C.J., Darwen, H.: Types and the Relational Model. The Third Manifesto, 3rd edn. Addison-Wesley, Reading (2006)

17. IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology. IEEE Standards Dept. (1998)
18. OMG UML 2.0 Superstructure Specification, formal/05-07-04
19. Eessaar, E.: On Specification and Evaluation of Object-Relational Data Models. WSEAS Transactions on Computer Research 2(2), 163–170 (2007)
20. Opdahl, A.L., Henderson-Sellers, B.: Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. Software and Systems Modeling 1(1), 43–67 (2002)
21. Davies, I., Green, P., Milton, S., Rosemann, M.: Using Meta Models for the Comparison of Ontologies. In: Evaluation of Modeling Methods in Systems Analysis and Design Workshop (2003)
22. Eessaar, E.: Relational and Object-Relational Database Management Systems as Platforms for Managing Software Engineering Artifacts. Ph.D. Thesis. Tallinn University of Technology, Estonia (2006)
23. Levendovszky, T., Karsai, G., Maroti, M., Ledeczi, A., Charaf, H.: Model Reuse with Metamodel-Based Transformations. In: Gacek, C. (ed.) ICSR 2002. LNCS, vol. 2319, pp. 166–178. Springer, Heidelberg (2002)
24. Merriam-Webster, Inc. Merriam-webster's online dictionary, `http://www.m-w.com/`
25. Kim, H., Boldyreff, C.: Developing software metrics applicable to UML Models. In: 6th International Workshop on Quantitative Approaches in Object–Oriented Software Engineering, pp. 147–153 (2002)
26. Dori, D., Reinhartz-Berger, I.: An OPM-Based Metamodel of System Development Process. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 105–117. Springer, Heidelberg (2003)
27. DMTF Common Information Model (CIM) Standards. CIM Schema Ver. 2.15. Database specification
28. Seshadri, P.: Enhanced abstract data types in object-relational databases. The VLDB Journal 7(3), 130–140 (1998)
29. Stonebraker, M., Rowe, L.A., Lindsay, B., Gray, J., Carey, M., Brodie, M., Bernstein, P., Beech, D.: Third-generation database system manifesto. Computer Standards & Interfaces 13(1–3), 41–54 (1991)
30. Stonebraker, M., Brown, P., Moore, D.: Object-Relational DBMSs: Tracking the Next Great Wave, 2nd edn. Morgan Kaufmann, San Francisco (1999)
31. Habela, P.: Metamodel for Object-Oriented Database Management Systems. PhD Thesis. Polish Academy of Sciences, Warsaw, Poland (2002)