TOWARDS A SEMIOTIC QUALITY FRAMEWORK OF SOFTWARE MEASURES

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

Keywords: Metrics, Measures, Semiotics, Quality, Metamodel, Database design, SQL.

Abstract: Each software entity should have as high quality as possible in the context of limited resources. A software quality measure is a kind of software entity. Existing studies about the evaluation of software measures do not pay enough attention to the quality of specifications of measures. Semiotics has been used as a basis in order to evaluate the quality of different types of software entities. In this paper, we propose a multidimensional, semiotic quality framework of software quality measures. We apply this framework in order to evaluate the syntactic and semantic quality of two sets of database design measures. The evaluation shows that these measures have some quality problems.

1 INTRODUCTION

Values of software quality measures (software measures) allow developers to evaluate the quality of software entities and improve them if necessary. Measures themselves are also software entities and must have as high quality as possible.

A part of the development of each measure is *formal* and *empirical* evaluation of the measure (Piattini et al., 2001b). Existing evaluation methods of measures do not pay enough attention to the quality of *specifications of measures*. If the quality of a specification is low, then it is difficult to understand and apply the measure. Therefore, we need a method for evaluating the quality of *specifications of measures*. On the other hand, there is already quite a lot of studies about how to use *semiotics* (the theory of signs) in order to evaluate the quality of software entities. In this paper, we extend this research to the domain of measures.

The *first goal* of the paper is to introduce a *semiotic* quality framework for evaluating *specifications of software measures*. This framework is created based on the semiotic quality framework of conceptual modeling SEQUAL that was proposed by Lindland et al. (1994) and has been improved since then. The *second goal* of the paper is to show the *usefulness* of the proposed framework by presenting the results of a study about the syntactic and semantic quality of two sets of specifications of database design measures.

We follow the guidelines of García et al. (2006) and use the term "measure" instead of the term "metric". In this paper the word "measure" denotes "software measure", if not stated otherwise. We use *analogy* (Maiden & Sutcliffe, 1992) as the research method in order to work out the framework and new measures based on the results of existing research.

The rest of the paper is organised as follows. In Section 2, we specify a semiotic quality framework for evaluating specifications of measures. In Section 3, we use the framework in order to evaluate two sets of specifications of database design measures. Section 4 summarizes the paper and points to the future work with the current topic.

2 A SEMIOTIC QUALITY FRAMEWORK

Many authors have investigated how to evaluate measures and have proposed frameworks that involve empirical and formal validation of measures (Schneidewind, 1992; Kitchenham et al., 1995; IEEE Std. 1061-1998, 1998; Kaner & Bond, 2004).

Jacquet and Abner (1998) investigate the state of the art of validation of measures and describe a detailed model of measurement process. They claim, based on the literature review, that existing validation frameworks of measures do not pay enough attention to the validation of the design of a measurement method. McQuillan and Power (2006) write that many measures "are incomplete, ambiguous and open to a variety of different interpretations."

Some researchers have used *semiotics* as the basis in order to work out evaluation frameworks of different kinds of software entities. According to Merriam-Webster dictionary <http://www.m-w.com/> semiotics is "a general philosophical theory of signs and symbols that deals especially with their function in both artificially constructed and natural languages and comprises syntactics, semantics, and pragmatics." Belle (2006) writes that any informational object has a syntactic, semantic, and pragmatic aspect. Syntax, semantics, and pragmatics relate an informational object to specification language, specified domain, and audience of the object, respectively (Lindland et al., 1994).

Semiotics has been used as the basis in order to evaluate the quality of conceptual models (Lindland et al., 1994), specifications of requirements (Krogstie, 2001), ontologies (Burton-Jones et al., 2005), enterprise models (Belle, 2006), and process models (Krogstie et al., 2006). A software measure is a kind of software entity. In this paper, we propose that *semiotics* can be successfully used in order to evaluate specifications of measures.

2.1 Specification of the Framework

In this section, we present a multidimensional, semiotic evaluation framework of the quality of specifications of measures. A model is a kind of software entity. A measure is a kind of software entity. Each software entity can be characterized in terms of different quality levels (physical, empirical, syntactic etc.). Each quality level has one or more quality goals. Each quality goal has zero or more associated measures that allow us to measure the quality of a software entity in terms of the goal.

The framework comprises physical, empirical, syntactic, semantic, perceived semantic, pragmatic, and social quality. We adapt the semiotic quality framework SEQUAL in order to use it in a new context – the evaluation of measures. The framework has to *enhance* the existing validation frameworks of measures. In addition, we present three *candidate* measures for evaluating the *syntactic* and *semantic* quality of specifications of measures. A candidate measure is a measure that has not yet been accepted or rejected by experts. We demonstrate the use of these measures in Section 3. These measures do not form a *complete* suite for evaluating measures. Future studies must work out a suite of measures that covers all the aspects of the framework.

We propose to use *metamodels*, mapping of *elements of models,* and *model-management operations* in order to check the quality of some aspects of a specification of a measure. The *novelty* is in the combined use of them.

The use of metamodels and ontologies in order to specify and evaluate measures is not a new method. Baroni et al. (2005) define some database design measures in terms of SQL:2003 ontology and use Object Constraint Language (OCL) in order to specify measures as precisely as possible. McQuillan and Power (2006) propose to extend the metamodel of Unified Modeling Language (UML) with a separate package that contains specifications of measures as OCL queries. It allows us to find measurement results based on a software entity *e* that is created by using a language L. The precondition of the use of the method is the existence of a metamodel of L and the existence of a UML model that represents e.

The use of a mapping of model elements has been used, for instance, in order to evaluate UML metamodel (Opdahl & Henderson-Sellers, 2002) in terms of Bunge–Wand–Weber (BWW) model of information systems. In the proposed method and examples we assume that the relevant models are UML class models.

2.1.1 Syntactic Quality

Syntactic correctness is the only syntactic goal (Krogstie et al., 2006). The syntactic correctness has two subgoals in the context of measures because we have to use two different types of languages in order to specify measures.

Firstly, the *content* of each specification of a measure is written by using one more languages. For instance, these languages could be natural languages like English, generic formal textual languages like OCL, domain-specific formal textual languages like Performance Metrics Specification Language (Wismüller et al., 2004), or generic visual languages like UML. For example, Baroni et al. (2005) specify database design measures by using English and

OCL. Therefore, *the first subgoal of the syntactic correctness* is to ensure that all specifications of measures follow the syntax rules of languages that are used in order to write the content of these specifications.

Next, we use an analogy with the *database* domain in order to illustrate additional aspects of the syntactic quality of specifications of measures. Each specification of a measure consists of one or more user-visible components. The Third Manifesto (Date & Darwen, 2006) is a specification of future database systems. According to the manifest each appearance of a value of a scalar type T has exactly one physical representation and one or more possible representations. Specification of each possible representation for values of type T is part of the specification of T. We could *conceptually* think about measures as values that belong to the scalar type *Measure*. In this case, each measure has one or more possible representations of its specification.

Therefore, *the second subgoal of the syntactic correctness* is to ensure that all appearances of specifications of measures conform to the rules of one the possible representations of type *Measure*.

There is more than one specification that can be used as a basis in order to work out a possible representation of a measure. IEEE Standard for a Software Quality Metrics Methodology ("IEEE," 1998) prescribes how to document software metrics (measures) and Common Information Model ("DMTF CIM Metrics schema," 2006) provides specification of metrics (measures) schema.

Each possible representation has one or more associated constraints that a correctly structured specification of a measure must follow. A problem with the IEEE Standard for a Software Quality Metrics Methodology is that it does not clearly describe *constraints* that must be present in the possible representation of a measure. For example, if we want to specify this possible representation by using UML class model, then we do not have precise information in order to specify *minimum* and *maximum* cardinality at the ends of associations.

If we want to check whether a specification of a measure m conforms to the second subgoal, then we have to do the following. Firstly, we have to create a model of the structure of *m*. After that we have to create a mapping between the model of the structure of *m* and the model that specifies a possible representation of the type *Measure*. There is a pair of model elements in the mapping if the constructs behind these elements are semantically similar or equivalent.

Let us assume that we create these models as UML class models. The elements of these models are classes, properties, and relationships. If X is the set of all the elements of the model of the structure of *m* and Y is the set of all the elements of the model of possible representation, then ideally there must be a *bijective* function f: X→Y. The amount of discrepancies between the models characterizes the amount of syntactic problems of m.

The creator of a UML class model can often choose whether to model something as a class or as a property (attribute) of a class. Larman (2002) suggests about the construction of conceptual class model: "If in doubt, define something as a separate conceptual class rather than as an attribute." Based on this suggestion, we can *simplify* the use of the method by considering only *classes* and not considering properties/relationships that are present in the class models (see Figure 1). It is in line with the example that is provided by Opdahl and Henderson-Sellers (2002). They evaluate a language based on classes of a metamodel (and not based on properties or relationships). We note that Figure 1 *illustrates* bijective functions and Y does not contain all the possible model elements.

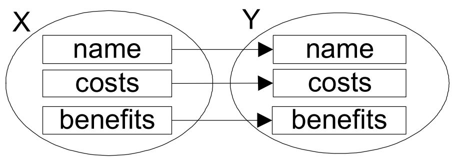

Figure 1: A bijective function.

Next, we present a *candidate* measure for evaluating the syntactic richness of a specification of a measure m.

SR(m): Let Y be the set of all the classes in a model of possible representation of measures. Let y be the cardinality of Y. Let X be the set of all the classes in a model of the structure of a specification of a measure m. Let Z be the set of all the classes in Y that have a corresponding class in X. There exists a pair of (corresponding) classes if the constructs behind these classes are semantically similar or equivalent. Let z be the cardinality of Z. Then SR(m) = z/y.

The possible value of SR(m) is between 0 and 1. 0 and 1 denote minimal and maximal syntactic richness, respectively.

2.1.2 Semantic Quality

Each measure has one or more associated domains. For instances, Choinzon and Ueda (2006) present 40 measures that belong to the domain of *object-oriented design*. Piattini et al. (2001b) present twelve measures that belong to the domain of *object-relational database design*.

Let us assume that we have a specification of a measure *m* that is created in order to measure a domain *d*. The *feasible validity* and *feasible completeness* are the only two semantic goals according to SEQUAL framework (Krogstie et al., 2006). Validity means that each statement about *d* that is made by *m* must be correct and relevant. Completeness means that *m* must contain all the statements about *d* that are correct and relevant. On the other hand, it is often impossible to achieve the highest possible semantic quality due to limited resources. Therefore, the goal is to achieve *feasible validity* and *feasible completeness*. In this case, there *does not exist* an improvement of the semantic quality that satisfies the rule: its additional benefit to m exceeds the drawbacks of using it.

Each measure considers only some aspect of the domain and not the entire domain. Therefore, we have to consider completeness in terms of sets of related measures. Measures, which belong to a set of measures about some domain, must together contain all the statements about the domain that are correct and relevant.

How can we evaluate the validity and completeness of measures? Krogstie et al. (2006) writes about models that it is only possible to objectively measure the *syntactic* quality of models. Krogstie et al. (2006) think that objective measurement of other quality levels (including semantic quality) of models is not possible because "both the problem domain and the minds of the stakeholders are unavailable for formal inspection." We claim that the situation is partially different in case of measures. The minds of the stakeholders are still unavailable for formal inspection. On the other hand, each measure can be used in order to measure the quality of one or more software entities. Each software entity is created by using one or more languages. Many of these languages are formal languages. Examples of these languages are UML and the underlying data model of SQL:2003. The abstract syntax of a formal language can be specified by using a metamodel (Greenfield et al., 2004). In the context of measures, the metamodels of these languages are specifications of the domains. We can use the metamodels as a basis in order to evaluate the semantic quality of specifications of measures.

Let us assume that we use UML *class models* for creating metamodels. In this case *classes* specify language elements and *properties/ relationships* specify relationships between the language elements (Greenfield et al., 2004). Let us assume that we want to evaluate the *validity* of a specification of a measure m that is used for evaluating software entities that are created by using a language L. The procedure:

1. Identification of L-specific concepts from m. For instance, Piattini et al. (2001b) specify the measure "Referential Degree of a table T" as "the number of foreign keys in the table T." In this case, L is SQL and L-specific concepts are *foreign key* and *table*.
2. Construction of a UML class model based on the concepts that are found during step 1.
3. If X is the set of all the model elements from step 2 and Y is the set of all the elements of a metamodel of L, then ideally there must exist a *total injective* function f: X→Y.

We can *simplify* the evaluation of *validity* by considering only classes (see Figure 2) and not considering properties/relationships that are present in the class models (see previous section). Model elements in Y in Figure 2 are from a metamodel of the underlying data model of SQL:2003 (Melton, 2003). We note than Figure 2 *illustrates* total injective functions and Y does not contain all the possible model elements.

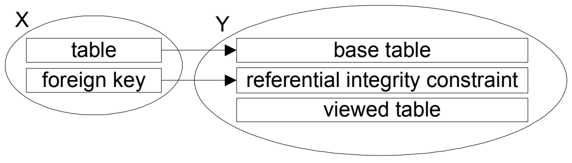

Figure 2: A total injective function.

One of the object-relational database design measures (Piattini et al., 2001b) is "Percentage of complex columns of a table T." The SQL standard (Melton, 2003) does not specify the concept "complex column". Therefore, in this case the function f is a *partial injective* function. Next, we present a *candidate* measure EV(m) for evaluating the validity of a specification of a measure m.

EV(m): Let X be the set of all the classes in a class model that is constructed based on the L-specific concepts that are present in a specification of a measure *m*. Let x be the cardinality of X. Let Y be the set of all the classes in a metamodel of a language L. Let Z be the set of all the classes in X that have a corresponding class in Y. There exists a pair of (corresponding) classes if the constructs behind these classes are semantically similar or equivalent. Let z be the cardinality of Z. Then EV(m) = z/x.

The possible value of EV(m) is between 0 and 1. 0 and 1 denote minimal and maximal semantic validity, respectively. For instance, x=2, z=2, and z/x=1 in case of the example in Figure 2.

Next, we present a *candidate* measure EC(M) for evaluating the *completeness* of a set of specifications of measures (we denote this set as M). We assume that all these measures allow us to evaluate software entities that are created by using a language L. For simplicity, the calculation procedure considers only *classes* and does not consider properties and relationships. The calculation of EC(M) starts with the *preparative phase* that contains three steps:

1. For each specification in M perform step 1 from the validity evaluation procedure.
2. For each specification in M, construct a simplified class model that specifies only classes (based on the result of step 1).
3. *Merge* all the models that are constructed during the step 2 by using the generic model management operator *merge* (Bernstein, 2003).

EC(M): Let X be the set of all the classes in the merged model that is produced as the result of step 3. Let Y be the set of all the classes in a metamodel of a language L. Let y be the cardinality of Y. Let Z be the set of all the classes in Y that have a corresponding class in X. There exists a pair of (corresponding) classes if the constructs behind these classes are semantically similar or equivalent. Let Z' be the set that contains all classes from Z together with all their direct and indirect *subclasses*. Let z' be the cardinality of Z'. Then EC(M) = z'/y.

The possible value of EC(M) is between 0 and 1. 0 and 1 denote minimal and maximal semantic completeness, respectively.

Why we have to construct the set Z'? *Value substitutability* in case of a parameter of a read only operator (that has the declared type T) means that "wherever a value of type T is permitted, a value of any subtype of T shall also be permitted" (Date & Darwen, 2006). Similarly, for instance, base table is a kind of table. In a metamodel of SQL, base table can be specified as a subclass of table. If we have a measure that allows us to measure tables in general, then it is possible to use this measure in order to measure base tables in particular.

For example, if X = {table} and Y = {table, base table}, then Z = {table}, Z' = {table, base table}, y = 2, z' = 2, and z'/y = 1.

2.1.3 Other Quality Levels

We use the works of Krogstie et al. (2001; 2006) as the basis in order to *introduce* the other quality levels.

Physical quality has the goals: *externalisation* and *internalisability* (Krogstie et al., 2006). Externalisation means that each measure must be available as a physical artefact that uses statements of one or more languages. Each measure must represent the knowledge of one or more software development specialists. Internalisability means that each measure must be accessible so that interested parties can make sense of it.

Minimal error frequency is the only empirical quality goal (Krogsie et al., 2001). Each externalised measure has one or more possible specifications that a human user can read and use. The layout and readability of each specification must allow users to correctly interpret the measure.

Feasible perceived validity and *feasible perceived completeness* are the only two perceived semantic quality goals (Krogstie et al., 2001). The perceived semantic quality of measures considers how the audience of measures interprets measures and their domains. For instance, if we want to evaluate the perceived validity of a specification of a measure, then we have to construct a model that specifies how some interested parties *understand* the specification. We also have to construct a model that specifies how the parties *understand* the domain of the measure. After that we have to compare these models (see Section 2.1.2).

Comprehension is the only pragmatic quality goal (Krogstie et al., 2006). Each specification of a measure must be understandable to its audience. For instance, Kaner and Bond (2004) present ten evaluation questions about measures. If a specification of a measure has high pragmatic quality, then an interested party should be able to answer these questions based on the specification.

Feasible agreement is the only goal of social quality (Krogstie et al., 2006). The social quality considers how well different parties have accepted a measure (how widely a measure is used), how much they agree on interpretation of a measure, and how well they resolve the conflicts that arise from different interpretations.

2.2 Discussion

Next, we discuss the advantages and possible problems of the proposed approach.

2.2.1 Advantages

The use of the semiotic framework has already been tested in case of different types of software entities. The proposed framework allows us to organize the knowledge about the evaluation of specifications of measures. We can use the existing studies about semiotic frameworks in order to find new means of improving the quality of specifications of measures and *candidate* measures for evaluating the quality of these specifications. For instance, Burton-Jones et al. (2005) present a suite of measures for evaluating *ontologies*. The suite consists of ten measures that allow us to measure the syntactic, semantic, pragmatic, and social quality.

The measure SR(m) (see Section 2.1.1) is *analogous* to the measure for evaluating syntactic richness of an ontology. The measure EV(m) is *similar* to the measure EI for evaluating semantic interpretability of an ontology: "Let C be the total number of terms used to define classes and properties in ontology. Let W be the number of terms that have a sense listed in WordNet. Then EI = W/C" (Burton-Jones et al., 2005). Instead of WordNet, the measure EV(m) uses a metamodel of the language that is the domain of m. The measure EC(M) does not have a corresponding measure in the suite of measures for evaluating ontologies.

2.2.2 Challenges

Firstly, the construction of a model based on a specification of a measure, and the creation of a mapping between different models requires somewhat *subjective* decisions. Therefore, it is possible that two different parties who use the same measure in case of the same set of specifications of measures will get different results.

For instance, in our view Piattini et al. (2001a; 2001b) use the concept *table* in order to denote *base tables*. Base table is not the only possible type of tables. A human user can find this kind of inconsistent use of terminology by studying the context of specification. On the other hand, it makes the *automation* of the evaluation process more difficult. Another example is that if we simplify the calculation of syntactic richness, validity, and completeness by considering only classes, then the result depends on whether the designers of models prefer to use attributes or classes in UML class models.

Secondly, the use of EV(m) and EC(M) requires the existence of metamodels of languages. If the required metamodels do not exist, then the use of the measures will be time consuming because a developer has firstly to acquire the metamodels.

Thirdly, there could exist more than one specification of the same measure. These specifications could refer to different language elements. For instance, informal specification of the measure "Referential Degree of a table T" that is proposed by Baroni et al. (2005) refers to the language (SQL) elements *foreign key* and *table*. On the other hand, formal specification of the same measure in OCL (Baroni et al., 2005) refers to the language (SQL) elements *foreign key* and *base table*. Therefore, each evaluation must be accompanied with the information about the specification of the measure that is used as the basis of this evaluation.

Finally, it is possible that a language has more than one metamodel. These metamodels could be created by different parties. For instance, DMTF Common Information Model database specification of SQL Schema ("DMTF CIM Database," 2006), relational package of OMG Common Warehouse Metamodel ("OMG," 2003), and the ontology of SQL:2003 (Baroni et al., 2005) are variants of metamodel of SQL. These models contain 8, 24, and 38 classes, respectively. It is also possible that there are differences between the different versions of the same metamodel. The values that characterize the quality of a specification of a SQL-database design measure will be different depending on the used metamodel (see Section 3). Therefore, each metamodel-based evaluation of a specification of a measure must be accompanied with the information about the version of the metamodel that is used in the evaluation. If we want to compare two sets of measures based on the values of the proposed measures, then these values must be calculated based on the same metamodel version.

3 EVALUATION OF DATABASE DESIGN MEASURES

Next, we illustrate the use of the proposed framework. In this paper, we investigate the quality of specifications of *database design measures*. The work of Blaha (1997) shows us that many databases do not have the highest possible quality. Blaha (1997) writes that about 50% of databases, which his team has reverse engineered, have major design errors. Therefore, it is clearly necessary to evaluate and improve the design of databases. We can use database design measures for this purpose.

Unfortunately there exist few database design measures. Piattini et al. (2001a) present three table oriented measures for relational databases. Piattini et al. (2001b) present twelve measures that help us to evaluate the design of object-relational databases. The measures allow us to evaluate databases that are created by using SQL. We call the set of *informal* specifications of these measures as M_{SQL} and M_{ORSQL}, respectively. We investigated M_{SQL} and M_{ORSQL} by using the proposed measures (see Section 2). For recording the evaluation results and performing the calculations, we constructed a software system (based on the database system MS Access).

For each specification of a measure, we calculated the value of SR(m) based on the specification of possible representation of measures that is proposed in IEEE Std. 1061-1998 ("IEEE," 1998). We assumed that all the components of the possible representation are modelled as separate classes. In Table 1, we summarize the results. For each set of specifications (M), we present the lowest value, the mean value, and the highest value of SR(m) among all the specifications that belong to M.

Table 1: Syntactic richness of measures.

	lowest	mean	highest
M_{SQL}	0.31	0.36	0.38
M_{ORSQL}	0.19	0.24	0.31

The only components that are *in our view* present in all the evaluated specifications are *name, data items,* and *computation*.

For each specification of a measure, we calculated the values of EV(m) based on the following specifications of the domain (SQL): Relational package of OMG Common Warehouse Metamodel (v1.1), DMTF CIM database specification (v2.16), and the ontology of SQL:2003 (Baroni et al., 2005). In Table 2, we summarize the results. For each pair of a set of specifications (M) and a specification of the domain, we present the lowest value, the mean value, and the highest value of EV(m) among all the specifications in M.

We also calculated EC(M_{SQL}) and EC(M_{ORSQL}) based on the same specifications that we used in case of calculating EV(m). Table 3 summarizes the results. For each pair of a set of specifications (M) and a specification of the domain (d), we present the value of EC(M) that is calculated in terms of d.

The results in Table 2 and Table 3 demonstrate that the values of measures EV(m) and EC(M) depend on the metamodel that is used in the calculation. The CIM database specification specifies fewer classes (8) compared to the CWM (24) and the SQL:2003 otnology (38). Therefore, EC(M_{SQL}) has relatively high value in case of the CIM database specification.

The specifications that belong to M_{SQL} have bigger completeness problems compared to the specifications that belong to M_{ORSQL}. However, M_{ORSQL} is also not complete. For instance, the measures in M_{ORSQL} do not consider type constructors, domains, triggers, SQL-invoked procedures, and sequence generators.

Table 2: Validity of measures.

	lowest	mean	highest
OMG Common Warehouse Metamodel (v1.1)			
M_{SQL}	0.33	0.61	1
M_{ORSQL}	0.12	0.63	1
DMTF CIM database specification (v2.16)			
M_{SQL}	0.33	0.44	0.50
M_{ORSQL}	0.12	0.54	1
The ontology of SQL:2003			
M_{SQL}	0.33	0.61	1
M_{ORSQL}	0.25	0.64	1

Table 3: Completeness of sets of measures.

	CWM	CIM	SQL:2003
M_{SQL}	0.08	0.12	0.05
M_{ORSQL}	0.21	0.38	0.18

On the other hand, the specifications of measures refer to elements that *in our view* do not have *a* corresponding element in the used metamodels: aggregation, arc, attribute of a table, class, complex attribute, complex column, generalization, hierarchy, involved class, referential path, shared class, simple attribute, simple column, and type of complex column.

4 CONCLUSIONS

In this paper, we proposed a new framework for evaluating the quality of specifications of software measures (*measures* in short). The *novelty* of this framework (in the context of development of measures) is that it is based on *semiotics* – the theory of signs. We developed this framework by adapting an existing semiotic framework. The existing framework is used in order to investigate the quality of different kinds of software entities. We proposed how to use this framework in order to evaluate specifications of measures. We proposed three candidate measures for evaluating the syntactic and semantic quality of specifications of measures.

The proposed evaluation framework has to *enhance* the existing evaluation methods of measures, which do not pay enough attention to the quality of specifications of measures.

We also investigated two sets of specifications of database design measures in terms of the proposed framework as an example. These measures allow designers to measure the design of relational and object-relational databases that are created by using SQL language. We evaluated the semantic quality of these specifications in terms of different metamodels that specify the domain of the measures (SQL). The results demonstrate that the selection of a metamodel affects the results of the evaluation. We found that the syntactic and semantic quality of the specifications is quite low.

The future work must include improvement of the quality of measures that were proposed in the paper. We also have to improve of the quality of existing database design measures, develop more database design measures, and evaluate these measures in terms of the proposed framework.

REFERENCES

Baroni, A.L, Calero, C., Piattini, M., & Abreu, F.B., 2005. A Formal Definition for Object-Relational Database Metrics. In *7th International Conference on Enterprise Information Systems*.

Belle, J. P., 2006. A Framework for the Evaluation of Business Models and its Empirical Validation. *Electronic journal of information systems evaluation*, Vol 9, Issue 1, 31-44.

Bernstein, A. P., 2003. Applying Model Management to Classical Meta Data Problems. In *Conf. on Innovative Database Research (CIDR)*.

Blaha, M., 1997. Dimensions of Database Reverse Engineering. In *Fourth Working Conference on Reverse Engineering*, 176-183.

Burton-Jones, A., Storey, V.C., Sugumaran, V. & Ahluwalia, P., 2005. A Semiotic Metrics Suite for Assessing the Quality of Ontologies. *Data & Knowledge Engineering*, Vol. 55, No. 1, 84-102

Choinzon, M.,& Ueda, Y., 2006. Design Defects in Object Oriented Designs Using Design Metrics. In *7th Joint Conference on Knowledge-Based Software Engineering*. IOS Press, 61-72.

Date, C. J. & Darwen, H., 2006. *Databases, Types and the Relational Model*, Addison Wesley. USA, 3rd edn.

DMTF Common Information Model Standards, 2006. CIM Schema Ver. 2.16. Database specification.

DMTF Common Information Model Standards, 2006. CIM Schema Ver. 2.15. Metrics schema.

García, F., Bertoa, M.F., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M., & Genero, M., 2006. Towards a Consistent Terminology for Software Measurement. *Information & Software Technology*, Vol. 48, 631-644.

Greenfield, J., Short, K., Cook, S., & Kent, S., 2004. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley Publishing, Inc. Indianapolis.

IEEE Standards Dept., 1998. IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology.

Jacquet, J. & Abran, A., 1998. Metrics Validation Proposals: A Structured Analysis. In *Proceedings of Eighth International Workshop of Software Measurement*.

Kaner, C., & Bond, P., 2004. Software Engineering Metrics: What Do They Measure and How Do We Know? In *10th International Software Metrics Symposium*.

Kitchenham, B., Pfleeger, S. & Fenton, N., 1995. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, Vol. 21, Issue 12, pp 929-944.

Krogstie, J., 2001. A Semiotic Approach to Quality in Requirements Specifications. In *IFIP 8.1 Working Conference on Organizational Semiotics, eds. Stamper et al., Montreal, Canada*, 231-249.

Krogstie, J., Sindre, G., & Jorgensen, H., 2006. Process models representing knowledge for action: a revised quality framework. *European Journal of Information Systems,* Vol. 15, No. 1, 91–102.

Larman, C., 2002. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall. USA, 2nd edn.

Lindland, O.I., Sindre, G., & Solvberg, A., 1994. Understanding quality in conceptual modeling. *IEEE Software*, Mar. 1994, Vol. 11, Issue 2, 42-49.

Maiden, N., & Sutcliffe, A., 1992. Exploiting reusable specifications through analogy. *Communications of ACM*, Vol. 35, No. 4, 55-64.

McQuillan, J. A. & Power, J. F., 2006. Towards re-usable measure definitions at the meta-level. In *PhD Workshop of the 20th European Conference on Object-Oriented Programming*.

Melton, J., ISO/IEC 9075-2:2003 (E) Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation). August, 2003.

Merriam-Webster, Inc. Merriam-webster's online dictionary, viewed 25 November, 2007, <http://www.m-w.com/>.

OMG Common Warehouse Metamodel Specification formal/03-03-02. March 2003. Version 1.1.

Opdahl, A.L., & Henderson-Sellers, B., 2002. Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. *Software and Systems Modeling,* Vol 1, No. 1, 43 – 67.

Piattini, M., Calero, C., & Genero, M., 2001a. Table Oriented Metrics for Relational Databases. *Software Quality Journal*, Vol. 9, No. 2, 79-97.

Piattini, M., Calero, C., Sahraoui, H., & Lounis, H., 2001b. Object-Relational Database Metrics. *L'Object*, vol. March 2001.

Schneidewind, N.F., 1992. Methodology for Validating Software Metrics. *IEEE Transactions Software Engineering*, Vol. 18, No. 5 (May 1992), 410-422.

Wismüller, R., Bubak, M., Funika, W., Arodz, T., & Kurdziel, M., 2004. Support for User-Defined Metrics in the Online Performance Analysis Tool G-PM. In *AxGrids 2004*, LNCS Vol. 3165, 159-168.