Existing Model Metrics and Relations to Model Quality

Parastoo Mohagheghi, Vegard Dehlen
SINTEF- P.O.Box 124- Blindern, N-0314 Oslo, Norway
{parastoo.mohagheghi, vegard.dehlen}@sintef.no

Abstract

This paper presents quality goals for models and provides a state-of-the-art analysis regarding model metrics. While model-based software development often requires assessing the quality of models at different abstraction and precision levels and developed for multiple purposes, existing work on model metrics do not reflect this need. Model size metrics are descriptive and may be used for comparing models but their relation to model quality is not well-defined. Code metrics are proposed to be applied on models for evaluating design quality while metrics related to other quality goals are few. Models often consist of a significant amount of elements, which allows a large amount of metrics to be defined on them. However, identifying useful model metrics, linking them to model quality goals, providing some baseline for interpretation of data, and combining metrics with other evaluation models such as inspections requires more theoretical and empirical work.

1. Introduction

Software metrics has a long history and a wide range of metrics are defined. Several studies have shown that software metrics can help in improving the software development process, understanding complexity of software, discovering and predicting faults and fault-prone software units, allocating resources and estimating development and maintenance efforts [11]. During the end of the 1990's and the beginning of the new millennium, models and modeling – especially driven by the development of UML (Unified Modeling Language) and, later, MDA (the OMG's Model Driven Architecture[1]), Mode-Driven software Development (MDD) and MDE (Model Driven Engineering) – became increasingly important in the software engineering community.

However, the role of models varies a lot in software development approaches applied in companies. Fowler has for example identified three modes of UML use[2]: UMLAsSketch, UMLAsBlueprint and UMLAsProgrammingLanguage. The emphasis of sketches is on selective communication rather than complete specification. Blueprints are developed by a designer whose job is to build a detailed design for a programmer to code up and thus models are required to correct and complete. In the third mode, semantics is added to UML models to make them executable. Here models should have the quality required for the purpose of execution. Brown has also discussed the spectrum of modeling as presented in [29]; i.e., from code-centric approaches where models are used to visualize the design, to when models are used as sketches of software to be developed (basic modeling), when the code and the model coexist and one is synchronized once the other is updated (round-trip engineering), and to approaches that are model-centric and most (or all, if possible) of the code is generated from models or models are executable. We call the approaches where models are widely used in software development and for more than visualizing the design for *model-based software development*, which also covers MDE. In MDE models are primary software artifacts and are subjects of transformations. Besides, a system is often modeled at several different abstraction levels and from multiple viewpoints[3].

There has been some research on the quality of models as sketches but the main focus has been on the communication perspective. Since in model-based software development models play a central role in most or all development phases and several artifacts may be generated from models, researchers have started working on other quality aspects of models as

[1] http://www.omg.org/mda/

[2] See his blog http://martinfowler.com/bliki/

[3] We use the term MDE in the remainder of this paper to cover also MDA and MDD.

well; encapsulated often in so-called quality models as discussed in [25]. Model metrics are consequently useful as they allow developers to predict and assess the characteristics of models as representations of the software system, and the quality of software systems themselves at an earlier phase of development.

In the QiM[4] (Quality in Model-driven engineering) project at SINTEF we have developed a language and tool for defining quality models in model-based software development [24]. A quality model in this context is a set of quality goals and relations between them defined by some stakeholders based on the purposes of modeling, and other elements such as practices (or means) to achieve quality and evaluation methods. These quality models can have models, modeling languages, modeling tools, modeling process or even transformations as targets for quality assessment and improvement.

This paper presents our findings from a literature review whose aim has been to summarize state of the art related to model metrics and relate them to model quality goals[5]. Section 2 presents the goals of measurement at model level while Section 3 discusses differences between collecting metrics from models and source code. Section 4 gives an overview of model metrics detected so far in the reviewed literature. Finally, Section 5 is conclusion and discussion of gaps and ideas for future work in order to get feedback from the workshop participants.

2. Goals of measurement at the model level

Models often consist of a significant amount of elements, which allows a large amount of metrics to be defined on these [19]. E.g. UML contains elements such as use case, classes, associations, messages, actions, methods and states. In addition, models can have different representations, like diagrams, XMI, or mappings to other models. The challenging task, however, is to define useful metrics. To select appropriate metrics on this huge amount of information, we need to define the goals of measurement.

A framework that can be used to derive metrics that serve specific goals is the Goal-Question-Metric paradigm developed by Basili et al. [6]. GQM starts by expressing the overall goals of the measurement. Then questions are generated whose answers must be known to determine if the goals are met. Finally, each question is analyzed in terms of what measurements are needed to answer the question. However, GQM is a generic approach and should be related to specific quality goals for models. Others have therefore defined quality models with quality goals for models in mind, as discussed in [25]. One example is the framework presented by Lange and Chaudron in [18] which relates modeling purposes to model characteristics (or model quality goals as we call them), and to some metrics and rules, as depicted in Figure 1 and Table 1.

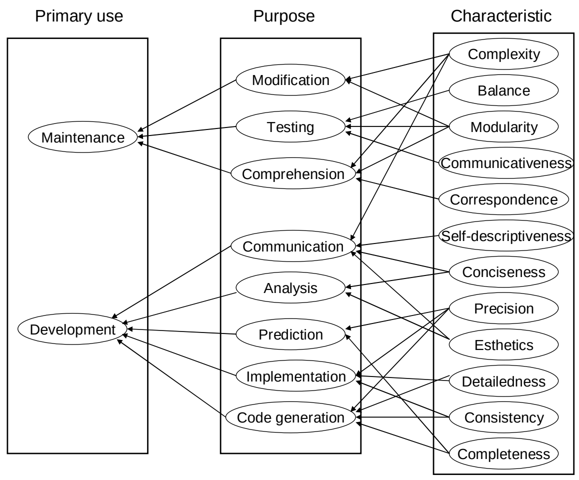

Figure 1. Modeling purposes and model characteristics presented in [18]

In Table 1 we have only shown four out of the 18 metrics and rules proposed in [18]. Here "DIT" stands for "Depth of Inheritance Tree", "Coupling" is the number of other classes a class is related to, "NCU" is the number of classes per use case and "NUC" is the number of use cases per class.

Table 1. Examples of relations between metrics and model characteristics from [18]

	Modularity	Complexity	Communitveness	Detailedness	Balance	Conciseness
DIT	√	√	√	√	√	√
Coupling	√				√	
NCU	√	√			√	√
NUC	√	√			√	√

A metric or rule can be relevant for several characteristics, a characteristic can be relevant for several purposes, and so on. Metrics in this framework

[4] http://quality-mde.org/
[5] The details of this review regarding publication channels and results will be published in future.

are mainly size metrics (such as NUC) or on the detailed design level and do not cover all the purposes of modeling. Some characteristics such as self-descriptiveness are not related to any metrics but to naming conventions and modeling rules and are therefore best assessed by inspections.

Model quality goals are included in several other papers as well and various definitions of them are given. We have done a literature review to find what model quality means [25] and have identified six classes of model quality goals collectively called as the *6C* (*model quality) goals*, which are depicted in Figure 2 and defined as:

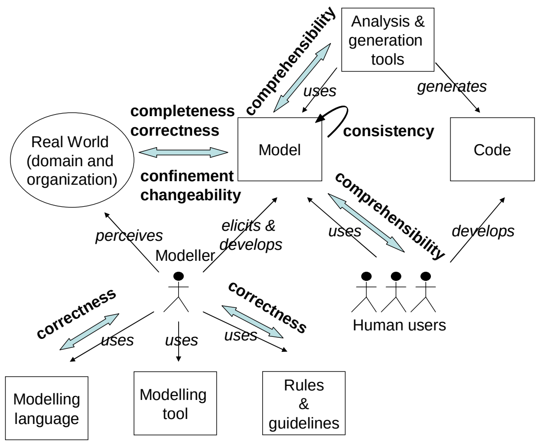

Figure 2. The *6C* goals in model-based software development with transformation of real world to running software

- **Correctness**; as including correct elements and correct relations between them and not violating rules and conventions, for example adhering to language syntax, style rules or naming guidelines.
- **Completeness**; as having all the necessary information and being detailed enough; according to the purpose of modeling.
- **Consistency**; as no contradictions in the model. It covers consistency between views or diagrams that belong to the same level of abstraction or development phase (*horizontal consistency*), and between models or diagrams that represent the same aspect, but at different levels of abstraction or in different development phases (*vertical consistency*). It also covers semantic consistency between models; i.e., the same element does not have multiple meanings in different models.
- **Comprehensibility**; as being understandable by the intended users; either human users or tools. For human users, several aspects impact comprehensibility such as aesthetics of diagrams, organization of a model, model simplicity (or complexity), conciseness (expressing much with little), and using concepts familiar for users.
- **Confinement**; as being in agreement with the purpose of modeling and the type of system; such as including relevant views and being at the correct abstraction level.
- **Changeability**; as supporting changes so that models can be improved and evolved rapidly and continuously.

Note that the above quality goals focus on the quality of models describing the system and not the quality of system design and implementation.

Some model quality goals may be measurable by metrics while others may not. In the next section we discuss differences between metrics at the model and code level.

3. Differences between metrics at model and source code level

What we measure and why we measure it differs between models and source code.

Firstly, models and source code often differ in abstraction level, precision, completeness, consistency and correspondence to the ultimate system [18]. Thus metrics from models cannot be directly transferred to code or vice versa. Even in MDE approaches where models are the primary artifacts of software development and source code is generated from them, some details are added during transformations.

Secondly, the goals of measurement are also often different: If modeling is performed for capturing, abstracting and communicating domain knowledge, requirements and main characteristics of the system under development, model metrics should focus on characteristics important earlier in the development life cycle. Examples of such characteristics are the quality of requirements, the correspondence between a model and the problem domain or the system it presents, and the usefulness of models for communicating requirements and design. On the other hand, design metrics are often collected to measure the quality of the design and implementation at the late stage of development.

Thirdly, even same quality goals mean differently for different models. For example completeness of a domain model means including all the necessary elements of the domain while at the design level it means including all the details necessary for code generation.

Fourthly, often a system is modeled in several diagrams from multiple viewpoints and it is necessary to define which diagram contains the right information for evaluating a quality goal. Sometimes one should also evaluate the relation between these diagrams as well such as their relative size, detailedness and consistency.

Finally, metrics collected from source code are often language dependent while models offer the possibility of evaluating some characteristics independent of the implementation language. We also have the possibility to evaluate some characteristics both before and after adding implementation details, such as dependencies between the elements of a model.

Despite these differences, little work has been done to define specific model metrics while the existing work has so far mainly focused on transferring source code metrics to models. In the next section we present an overview of existing work on model metrics.

4. Existing model metrics

Because of the differences between models and source code as discussed in the previous section, not all the metrics defined for source code can easily be transferred to models. Even so, recent literature on model metrics shows that a significant amount can be reused. In addition, since UML is almost the de facto modeling language in industry, researchers have defined several metrics suites targeting UML models directly. Some of these metrics are applicable to other modeling languages as well while approaches such as Domain-specific Modeling (DSM) may require additional metrics. In the following we provide an initial classification of model metrics detected in literature so far.

4.1. Model size metrics

Metrics targeting models are characterized by mostly being size metrics, that is, they count the number of classes, use case, associations and so on. These size measures are proposed to be used to measure several characteristics of models. Note that model size metrics here does not include characteristics inside elements (such as the number of methods of a class) which are related to structural complexity and are covered in the next section.

Lange defines the goals of measuring model size as [19]:

- **Comparing models**. Comparing the size of models, e.g. different versions of the same model, different models for the same system or models for different systems.
- **Measuring progress**. Answering questions like 'How fast is our model growing?'.
- **Prediction**. Predicting for example the effort needed for a project or the size of the implementation of the system. Note that size is the main driver in most effort estimation models such as in COCOMO [8].
- **Description**. Describing the characteristics of a model. For example, in empirical studies it is necessary to describe carefully characteristics of the model under study.

Based on the four dimensions of source code size defined by Fenton and Pfleeger [11] – length, complexity, functionality and reuse – Lange proposes five dimensions for UML2 model size [19]:

1. **Absolute size**. Metrics that measure a model's absolute size is the number of elements, like use cases, sequence diagrams or classes in a diagram.
2. **Relative size**. This dimension represents ratios between absolute size metrics, such as the number of sequence diagrams divided by the number of use cases, and can be used to compare proportions of different models with each other and to give an indication about the completeness of the models.
3. **Complexity**. Complexity of the describing model (as opposed to the described system) is suggested to be measured as a subset of the absolute and relative size metrics, although no specific metrics is proposed. The system complexity on the other hand should be measured by commonly accepted complexity metrics such as the metrics suite of Chidamber and Kemerer [10].
4. **Functionality**. Lange suspects that there exist relations between functionality and model size metrics that say something about a model's completeness or the model's level of abstraction. Established metrics for functionality are Function Points [2] and Object Points [8]. Specific Use Case Points are also proposed by Karner, who employs use cases as a representation of system functionality and uses them for estimating effort in a project [16].
5. **Reuse**. According to Lange, "A reuse metric can only be applied to a UML model that makes use of a profile to denote reuse (such as OMG's Reusable Asset Specification [26])." A simple metric could be the percentage of reuse.

Lange et al. have developed a tool called MetricViewEvolution that collects some size metrics from UML models and visualizes them in order to help analysis [20].

Kim and Boldyreff also propose size metrics on classes, messages, use cases and on models as a whole

[17], such as the number of objects in a model. These metrics are used to measure various characteristics at an early phase of development, without being tied to any specific characteristics or design principles.

4.2. Metrics on design and implementation models

This class covers metrics proposed to measure the quality of detailed design and implementation which is often discussed to be important for later maintenance. For example Genero et al. relate size and structural complexity metrics (which are measured based on the number of relations) to two main sub-characteristics of the maintainability of class diagrams: understandability and modifiability [14].

Here it is most reuse of source code metrics, for example metrics for Object-Oriented (OO) systems. Most existing metrics for OO systems were defined during the 90's and were targeted towards source code. These metrics are often linked to established design patterns and best practices and provide measures for analyzing whether or not a software system satisfies these patterns and practices. An example of one such best practice is to avoid heavy coupling between classes, as it makes the source code difficult to understand and maintain. Therefore coupling is included in most OO metrics suites [4, 9, 10, 15 and 21]. Also receiving notable focus is inheritance [1, 4, 10, and 22], cohesion [4 and 10], polymorphism [1], information hiding [1 and 4] and complexity [10 and 21]. OO metrics suites have been suggested as applicable for UML models in [5, 13, 23 and 30].

Finally, Reijers describes a cohesion metric that can be used to evaluate operations in a workflow and take decisions on whether to split or combine them, which is related to the quality of design [27].

4.3. Other metrics

In this section we include metrics proposed for models other than size and design metrics.

Some work cover detecting defects in UML models related to consistency between models or the degree of completeness. For example Berenbach has developed a tool called DesignAdviser that reports defects such as "missing associations" and "class not instantiated" [7].

Since one of the main purposes of modeling is improved communication between stakeholders, metrics related to comprehensibility and aesthetics of models should receive attention while few of them are defined so far. As examples we can mention counting the number of elements or crossing lines in a diagram as proposed in [18]. In [12] the authors propose measuring state diagram complexity by counting the number of entry and exit actions, number of activities, states and transitions. They discuss that these metrics correlate to the understandability aspect of state diagrams.

For the purpose of generating artifacts from models, Solheim and Neple have proposed "transformability" (including attributes such as completeness and precision) and "maintainability" (including traceability and well-designedness) as model quality goals [28], but their work do not include metrics.

4.4. Summary regarding existing metrics

We discovered three categories of metrics defined for models so far: size metrics, design metrics, and a few model-specific metrics related to comprehensibility of models. From model size metrics, "relative size metrics" are specific to UML or modeling languages that include several diagrams.

Interpretation of metrics is a subject that is less discussed. It should be covered by linking metrics to applicable theories or best practices (as done for source code metrics), empirical studies and finding some company baseline. Counts of elements and relations do not often reveal substantial information by themselves unless analyzed, and often linked to other attributes. As an example, counting the number of interactions in a class, as in [15], can say something about coupling, or some metrics may be related to complexity of design as discussed in [14]. Berenbach writes that metrics such as inheritance depth, arguments in a class method or methods in a class model, have been understood for quite some time. What is still not well understood is what constitutes a "bad" lower or upper limit to a metric [7].

Some quality goals are most important for models developed and used early in the development life cycle (conceptual models, requirement models and so on); such as comprehensibility (being self-descriptive, nice to see, conciseness etc.) and covering the domain or requirements. For these models we detected a few metrics related to comprehensibility and some size metrics which are applicable for evaluating completeness and high-level architecture quality. Other evaluation methods such as inspections and involving users or domain experts are proposed in literature for evaluating the quality of such models as well, although not covered here.

Models developed and used later for detailed design and implementation should be correct, complete, consistent and changeable, especially if they are used for generating source code. Again size metrics can be used for evaluating completeness and tools can detect or prevent inconsistencies and some violation of rules (related to correctness). Design metrics such as the OO ones are proposed to be applied on models to evaluate the quality of design. Inspecting models is also an effective method for detecting defects.

From analysis of literature, we have also identified the need for maintenance of models covered in "changeability" [25] or "maintainability" [28] quality goals. Related metrics may be those regarding organization of a model such as the number of use cases or packages.

Visualizing metrics can help in analysis [20], and tools detecting and reporting defects can improve the quality of models [7]. However, we do not see wide industrial usage of such tools yet.

5. Conclusions and future work

The purpose of this work has been to present on-going work on model quality and discuss the use of metrics for assessing quality. We presented an overview of proposed metrics in literature and some examples of usage. Some identified research gaps are defining metrics for early models, linking model metrics to quality goals by using theories or best practices (why and to what degree a metrics can measure if a given model fulfills a quality goal), collecting empirical data that helps in interpreting metrics, and providing some baseline data. There are also guidelines for developing high-quality UML models such as in [3, 7 and 31] which may provide a base for specifying new metrics.

Some quality goals and metrics presented in this paper are integrated in our work on the MODELPLEX project (MODELing solution for comPLEX software systems)[6] where a set of research questions and evaluation criteria are defined for evaluating the impact of MDE tools and technologies on project and product characteristics in four industrial case studies, including the quality of models. We will also take advantage of other evaluation methods and integrate them all in a quality model for models as part of the QiM project.

Acknowledgments. This work has been co-funded by the Quality in Model-driven engineering (QiM) project and the European Commission within the 6th Framework Programme project MODELPLEX contract number 034081.

[6] https://www.modelplex.org/

6. References

[1] B.F. Abreu, and W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality", *Proc. 3rd International Metric Symposium*, 1996, pp. 90-99.

[2] A.J. Albrecht, "Measuring Application Development Productivity", In Tutorial Programming Productivity: Issues for the Eighties, *IEEE Computer Society*, 1986, pp. 35-44.

[3] S.W. Ambler, *The Elements of UML 2.0 Style*. Cambridge University Press, 2005.

[4] J. Bansiya, and Davis C., "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering* 28(1), 2002, pp. 4-17.

[5] A.L. Baroni, S. Braz, and B.F. Abreu, "Using OCL to Formalize Object-Oriented Design Metrics Definitions", *Proc. 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02)*, 2002, 11 p.

[6] V.R. Basili, G. Caldiera, and H.D. Rombach, "The Goal Question Metric Paradigm", In *Encyclopedia of Software Engineering*, volume 2, John Wiley and Sons, 1994, pp. 528-532.

[7] B. Berenbach, "Evaluation of Large, Complex UML Analysis and Design Models", *Proc. 26th Int'l Conference on Software Engineering*, 2004, pp. 232-241.

[8] B.W. Boehm, E. Horowitz, R. Madachy, D. Reifer, B.K. Clark, B. Steece, A.W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[9] L. Briand, W. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++", *Proc. 19th International Conference on Software Engineering,* 1997, pp. 412-421.

[10] S.R. Chidamber, and C.F. Kemerer, "A Metrics Suite for Object-Oriented Design", *IEEE Trans. on Software Engineering* 20(6), 1994, pp. 476–493.

[11] N.E. Fenton, and S.L. Pfleeger, *Software Metrics; a Rigorous and Practical Approach*, Thomson Computer Press, 2nd Edition, 1996.

[12] M. Genero, D. Miranda, and M. Piattini, "Defining and Validating Metrics for UML Statechart Diagrams", *Proc. QAOOSE'02*. 2002, 18 p.

[13] M. Genero, M. Piattini, and C. Calero, "A Survey of Metrics for UML Class Diagrams", *Journal of Object Technology* 4 (9), 2005, pp. 59-92.

[14] M. Genero, E. Manso, A. Visaggio, G. Canfora, and M. Piattini, "Building Measure-Based Prediction Models for UML Class Diagram Maintainability", *Empirical Software Engineering Journal* 12(5), 2007, pp. 517-549.

[15] R. Harrison, S. Counsell, and R. Nithi, "Coupling Metrics for Object-Oriented Design", *Proc. 5th International Software Metrics Symposium Metrics*, 1998, pp. 150-156.

[16] G. Karner, *Metrics for Objectory*, Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21, 1993.

[17] H. Kim, and C. Boldyreff, "Developing Software Metrics Applicable to UML Models", *Proc. QAOOSE'02*, 2002.

[18] C.F.J. Lange, and M.R.V. Chaudron, "Managing Model Quality in UML-based Software Development", *Proc. 13th Int'l Workshop on Software Technology and Engineering Practice (STEP'05)*, 2005, pp. 7-16.

[19] C.F.J. Lange, "Model Size Matters", *Proc. Model Size Metrics Workshop*, 2006. http://www.win.tue.nl/~clange/papers/Lange_ModelSizeMatters.pdf

[20] C.F.J. Lange, A.M.W. Martijn, and M.R.V. Chaudron, "MetricViewEvolution: UML-based Views for Monitoring Model Evolution and Quality", *Proc. 11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, 2007, pp. 327-328.

[21] W. Li, and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software* 23(2), 1993, pp. 111-122.

[22] M. Lorenz, and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[23] M. Marchesi, "OOA Metrics for the Unified Modeling Language", *Proc. 2nd Euromicro Conference on Software Maintenance and Reengineering*, 1998, pp. 67-73.

[24] P. Mohagheghi, and V. Dehlen, "A Metamodel for Specifying Quality Models in Model-Driven Engineering", *Proc. Nordic Workshop on Model Driven Engineering (NW-MoDE '08)*, 2008, 15 p. http://quality-mde.org/publications.html

[25] P. Mohagheghi, V. Dehlen, and T. Neple, "Towards a Tool-Supported Quality Model for Model-Driven Engineering", *Proc. 3rd Workshop on Quality in Modeling (QiM'08)*, 2008, 15 p. http://quality-mde.org/publications.html

[26] *Object Management Group. Reusable Asset Specification*, version 2.2, formal 05-11-02 edition, November 2005.

[27] H.A. Reijers, "A Cohesion Metric for the Definition of Activities in a Workflow Process", *Proc. EMMSAD'03*, 2003, http://www.emmsad.org/2003/Final%20Copy/02.pdf

[28] I. Solheim, and T. Neple, "Model Quality in the Context of Model-Driven Development", *Proc. 2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS'06)*, 2006, pp. 27-35.

[29] M. Staron, "Adopting Model Driven Software Development in Industry – A Case Study at Two Companies", *Proc. MoDELS 2006*, LNCS 4199, 2006, pp. 57-72.

[30] M.H. Tang, and M. Chen, "Measuring OO Design Metrics from UML", *Proc. UML'02*, 2002, pp. 368-382.

[31] B. Unhelkar, *Verification and Validation for Quality of UML 2.0 Models*, Wiley-Interscience, 2005.