Developing a Quality Framework for Model-Driven Engineering

Parastoo Mohagheghi, Vegard Dehlen

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

Abstract. This paper presents some related work on quality frameworks and requirements for evaluating them. It also discusses characteristics of model-driven engineering that are important when building a quality framework, such as its use of models in several stages of development and maintenance, generation of other artifacts from models and its multi-abstraction level approach that requires consistency and traceability. We present a 7-step process on how to define a quality framework that is adapted to model-driven engineering, and which integrates quality engineering with quality evaluation. As an example, the framework is applied on transformation quality. We maintain that the transformation process and transformation mapping should be discussed separately, as they require different approaches, and suggest quality goals, quality-carrying properties to achieve the quality goals and methods for evaluating these properties.

Keywords: Model-driven engineering, quality, transformation, metrics

1 Introduction

More attention is paid to the quality aspects in Model-Driven Engineering (MDE) along with the growing importance of modeling in software development. Some challenging issues (especially for complex or large systems and special domains) are the increasing complexity that we need to understand and handle, the need for reliable systems and approaches that can verify and preserve quality requirements, as well as the dynamic adaptation and management of systems using transformations at runtime. Our research on the "Quality in MDE" project in SINTEF (http://quality-mde.org/) focuses on developing a quality framework applicable for MDE that includes quality goals, means or quality-carrying properties to achieve them, and evaluation methods. The research questions include:

1. *What quality aspects are important in MDE? Are there any differences in quality goals and activities when using MDE compared to other approaches?*
2. *How can quality goals be achieved and evaluated?*
3. *How can MDE improve the quality of developed software?*

This paper gives some answers to the above questions and defines an initial framework for defining and evaluating quality in MDE. It further discusses the quality of transformations as an example of applying the framework. This paper is a revised and shortened version of a paper presented at the 2nd workshop on Quality in Modeling co-located with MODELS 2007 and we refer to the workshop version for more discussions on the requirements of quality frameworks.

The paper is organized as follows. Section 2 presents some definitions of software quality, the different purposes of modeling, work on quality frameworks and characteristics of MDE that are important when defining a quality framework. Section 3 presents our quality framework and Section 4 applies it on the transformation quality. The paper is concluded in Section 5.

2 Background

2.1 Definitions of Quality and Relation to Modeling Purposes

According to IEEE, software quality as an attribute is (1) the degree to which a system, component, or process meets specified requirements, and (2) the degree to which a system, component, or process meets customer or user needs or expectations [10]. ISO 9126-1 defines quality as a set of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs [11]. Evaluating quality based on the goals or needs is also emphasized by Claxton and McDougal who write that assessing the quality of anything – models included – has two parts. One comes from measuring the right things, in the right way, with the right yardsticks. But the heart of quality comes from the second aspect; judging something based on its intended function and purpose [2]. So the search for quality (in modeling) starts by asking, "What's the purpose of a model?" as models are in fact developed for various purposes.

Kühne classifies models as being either *descriptive* (capture some knowledge; e.g. requirements or domain analysis) or *prescriptive* (aka specification models; used as blueprints of a possible or imaginary system) [13]. In other words, a model can exist later or earlier than its original. Hesse thinks that in the software engineering field, a model often plays a double role: describing a part of an application domain and prescribing a piece of software for that domain [8]. Daniels defines three kinds of models based on their purposes [4]:

- *Conceptual models* describe a situation of interest in the world, such as a business operation or factory process.
- *Specification models* define what a software system must do, the information it must hold, and the behavior it must exhibit. They assume an ideal computing platform.
- *Implementation models* describe how the software is implemented, considering all the computing environment's constraints and limitations.

Different types of models have of course different quality goals, where a "quality goal" is defined as a clear definition of what quality means to a stakeholder and that can be measured in a meaningful way. For example, conceptual models should be understandable for external stakeholders but not necessarily detailed. However, it is not often straightforward to define quality goals for each purpose of modeling or aspect, because:

- Some quality goals are in conflict with one another. For example using the same modeling language for different models reduces the need for learning new languages. On the other hand, we want to use different modeling features in each model (for example, the implementation model has to take the programming environment into account [4]) and using the same modeling language might therefore not be appropriate. Paige et al. believe that users may profit from using different languages for different purposes and combining them [21].
- Some quality goals crosscut models or activities. For example, if our conceptual model contains the concept of *customer*, our software will contain direct representations of customers, and our software customers will have similar attributes to their real-world counterparts. We want this correspondence because it improves traceability between requirements and code, and because it makes the software easier to understand [4].

Thus any research on quality in MDE should take into account the various modeling purposes, relations of purposes to quality goals and the dependencies or conflicts between them. In MDE, models are refined progressively and transformed to new models or code. In [19], we discussed that the quality of models depends on the quality of modeling language(s) used, the quality of tools used for modeling, the knowledge of developers of the problem in hand and their experience of modeling languages and tools, the quality of the modeling processes and the quality assurance techniques applied to discover faults or weaknesses. We also add the quality of activities performed on the models such as transformations to the above list, and discuss it in more details throughout this paper.

2.2 What Characterizes Model-Driven Engineering?

The characteristics of MDE that are important when defining a quality framework are:

- *Use of models in several stages of software development*: Models are used from early development phases to testing, simulation and code generation. Models are often incomplete, imprecise and inconsistent early in the software development life-cycle and get gradually more precise and complete. Models can be non-executable or executable (even early analysis models can be executable).
- *Models on different levels of abstraction and from different viewpoints*: An example is the OMG MDA's viewpoints of Computational Independent Models (CIM), versus Platform Independent Models (PIM) and Platform Specific Models (PSM) [20]. Relations between these models are important when evaluating them for some quality characteristics. For example, refined models have additional classes and methods that can increase complexity metrics. Another example is

structural models vs. behavioral models. This is a characteristic of e.g. UML and not necessarily all modeling languages. The multi-view and multi-abstraction level development approach means that each of the diagrams and abstraction levels might require specific quality goals and metrics. Lange describes this for the model size metrics that varies on various diagrams and abstraction levels [15]. Mellor and Balcer refer to several challenging issues that inevitably arise from the multi-view and multi-notational approach of UML in MDE [18]:

  - *Consistency*: The models of various views need to be syntactically and semantically compatible with each other (i.e., horizontal consistency).
  - *Transformation and evolution*: a model must be semantically consistent with its refinements (i.e., vertical consistency).
  - *Traceability*: A change in the model of a particular view should lead to corresponding consistent changes in the models of other views.
  - *Integration*: Models of different views may need to be seamlessly integrated before software production.

- *Activities are performed on models by tools*: Models undergo transformations and refinements. Many activities have models as input, output, or both. The quality of such activities can preserve, improve or reduce the quality of models. Model transformation is applied by tools, and during a transformation output models are supplied with information not present in the input model. Examples are domain-specific information or the platform concept during the PIM to PSM transformation. Models should therefore be complete and precise but not include unnecessary or redundant information [23].
- *Generation of code and other artifacts from models*: This means that evaluating the quality of models is more important in MDE than in traditional software development, where the code is mostly evaluated for quality.
- *Developing Domain Specific Languages (DSLs) and models*: DSLs have existed for a while and Domain Specific Modeling Languages are also getting more popular as a means to increase productivity and tailor the development environment to a domain. Selecting any approach for developing a DSL such as defining a metamodel or a UML profile needs knowledge of language and tool design and appropriate quality guidelines.

Thus a quality framework in MDE should take into account the role of models, languages, tools, transformations and their appropriateness for the domain and modeling purposes. *Model-driven Quality Assurance (MDQA)* is often defined as the automatic quality assurance that is based on models such as using system models for testing and verification (see e.g., http://www.mdqa.org/). In this paper, we suggest the notion of *Model-Driven Quality Engineering (MDQE)* meaning taking advantage of MDE to prevent and discover quality defects as early as possible in the software development lifecycle. MDE lends itself to quality engineering because of two reasons. First, models are primary software artifacts in MDE and several other artifacts are generated from models. Thus developing high quality models improves the quality of e.g., test cases and code that may be fully or partly generated from models. Second, quality engineering is enhanced by the extensive use of tools in transforming models to other models or code. Tools can analyze and monitor models for various characteristics. An example is discussed by Haesen and Snoeck in relation

with consistency checking which can be done *by analysis* (an algorithm detects inconsistencies between deliverables), *by monitoring* (meaning that a tool has a monitoring facility that checks every new specification), and *by construction or by generation* (meaning that a tool generates one deliverable from another and guarantees semantic consistency) [7]. Another example is using tools for checking rules or constraints during modeling or transformations as proposed in [1]. Rules and constraints can also be defined on metamodels.

2.3 Related Work on Quality Frameworks

In this section, we present some work on quality frameworks that either address the quality of models or quality in MDE, or may be used in building such a framework for MDE.

ISO/IEC 14598 International Standard (Standard for Information technology - Software product evaluation - Part 1: General overview) defines the term *quality model* as "the set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality". ISO 9126 is an example of a widely used software quality model [11]. We use the term *quality framework* in our work to avoid any confusion between quality model and model quality.

Dromey proposes a five step approach in constructing a quality model [5]:

1. Identify a set of high-level quality attributes for the product like reliability or maintainability.
2. Identify the product components. Examples are modules, requirements or relations.
3. Identify and classify the most significant, tangible, quality-carrying properties for each component. These are properties that result in manifestation of the high-level quality attributes.
4. Propose a set of axioms for linking product properties to quality attributes. This is not an easy task and the links cannot always be empirically verified.
5. Evaluate the model, identify its weaknesses and refine it.

To identify high-level quality attributes, one may ask:

- What are the most important usages of this product?
- What kind of defects we want to avoid for these usages?

Trendowicz and Punter discuss quality models for software product lines [25]. The activities during development of a quality model or framework are shown in Figure 1. The definition of goals, characteristics and sub-characteristics should be done iteratively and involve the stakeholders. This procedure goes on for as long as there is a set of measurable sub-characteristics defined. A sub-characteristic is measurable when it is possible to attach it to a particular component of a product line and define one or more corresponding metrics (which can be quantitative, qualitative evaluation or a combination of both); thus similar to the tangible quality-carrying properties in the Dromey's process. Reviewing should guarantee that the quality model is feasible and not too complex. The final step is actually execution. They further write that quality models should be *flexible* (to be tailored to a specific organization and

project), *reusable* and *transparent* (clear insight into their rationale as well as the meaning of the characteristics and relations among them).

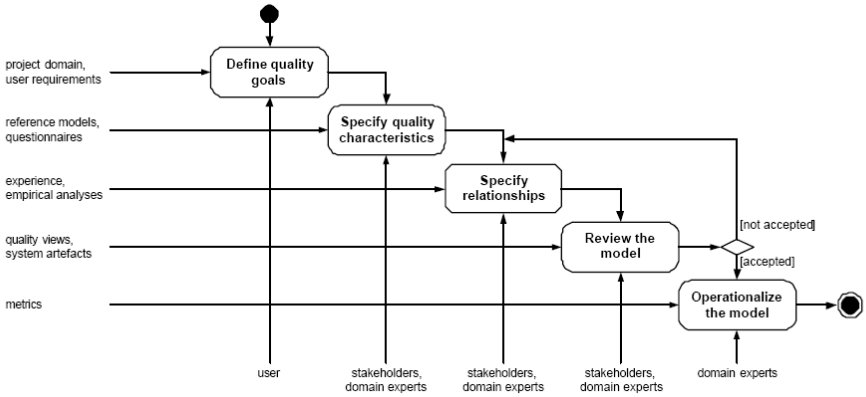

Fig. 1. Activities during development of quality models as defined in [25].

A framework that is applied on conceptual models is first presented by Lindland et al. [16] and later extended by Krogstie et al. and applied for evaluating the quality of modeling languages (see for example [12]). Lindland et al. divided quality goals into *syntactic* (adherence to the language rules or syntax), *semantic* (relevance to the problem domain and containing statements that are correct and relevant) and *pragmatic* (understandability of a model by stakeholders).The framework separates quality goals from means to achieve them. For example having formal syntax in a language is a means to achieve syntactic quality. Means are similar to quality-carrying properties in the Dromey's process. Solheim and Neple have simplified and adapted this framework to MDE [23]. They further identify *transformability* and *maintainability* as two quality goals that are important in MDE, which are in turn decomposed into several characteristics.

Lange and Chaudron identify two primary use of models; either development or maintenance [14]. They further define some purposes of modeling for each phase (e.g., analysis and prediction are done in the development phase) and relate some quality characteristics to each purpose. These characteristics are further related to metrics that are mainly on the detailed design level. Of other work on the quality of models we can mention [2] on the quality of data models (a data model is a model describing parts of business) and [26] on the quality of UML 2.0 models.

In addition to models, modeling languages has been subject of research, as in [6, 12 and 21]. The three works have some language quality requirements in common such as having minimal set of concepts that are precisely defined, uniqueness of concepts and understandability, while they complement each other in other aspects. Another difference is when they are applied. Paige et al. [21] recommend their principles for designing modeling languages, while Krogstie et al. [12] and Grossman [6] have defined criteria for evaluating modeling languages.

Putting all the related work together provides requirements for quality frameworks and a list of quality goals for some aspects such as models and languages, while other

aspects such as processes, activities and tools are less studied. There is also a need for more empirical studies and evaluation of the frameworks.

3 Defining a Quality Framework for MDE

In the previous section we presented some work on quality frameworks, MDE characteristics, and on the quality of models and modeling languages. In this section, we present a process for defining a quality framework in MDE which is illustrated in Figure 2.

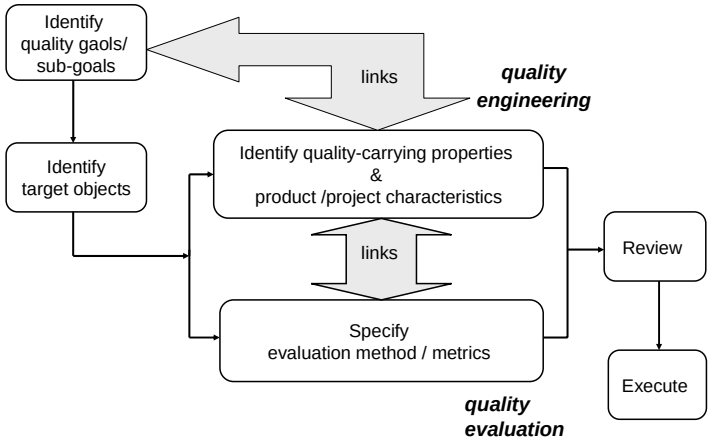

Fig. 2. Steps in developing a quality framework for MDE; specifying both quality engineering and quality evaluation.

We define the steps as:

1. Identify *quality goals*. Examples of quality goals are maintainability, reuse or increased productivity. Identifying quality goals should involve all stakeholders and reflect the purposes of modeling and the priorities of the project.
2. Identify *target objects* that can impact the quality goals. Proper target objects can be the software development approach or process, models, metamodels, languages, tools, transformations or the quality assurance techniques.
3. Identify the *quality-carrying properties* of the target objects and the *product or project characteristics* that they help to achieve. For example the possibility to generate code from models is a quality-carrying property of the modeling tool that reduces the amount of manual coding and provides more consistent code. Identifying the quality-carrying properties is based on several aspects such as:
   - Purpose of the target object.
   - Lifecycle phase (stages of development, maintenance or run-time).

- Isolated or in relation with other objects: it may be a need to integrate models / languages/ tools/ activities with other models / languages/ tools / activities, or they may need to exchange data. Integration may require consistency, portability, traceability, compatibility etc.
- Scale of the project.
- Domain-specific or general.
- Lifetime (long-living or not): lifetime has impact on the need for training, documentation, or maintainability.

As discussed by Trendowicz and Punter [25], relations should also be identified.

4. Specify how to evaluate the quality-carrying properties and characteristics; e.g., measuring quantitatively by metrics or subjective evaluation, inspections using checklists or interviewing the users. Specify links that validates that the right thing is measured.
5. Specify association links between the quality-carrying properties and the quality goals. For example, including domain knowledge in a domain specific code generator may reduce the number of certain defect types and thus improve software quality. This should be validated by analyzing the number and the type of defects.
6. Review and evaluate the framework in practice for characteristics such as completeness, orthogonality, parsimony, reusability, flexibility, transparency, relevance and possibility to be adopted.
7. Execute: Execution covers the implementation of quality-carrying properties and evaluation.

The process can support a hierarchical model of goals and quality-carrying properties as well. For example, transformations as a target object may be decomposed to the transformation process and the transformation rules as discussed in the next section.

The differences of the process in Figure 2 and the Dromey's process described in Section 2.3 are introducing target objects in the MDE context, adding the evaluation step and the requirements for evaluating the quality framework. We also work on identifying the quality-carrying properties and the product / project characteristics that MDE can support; i.e., MDQE.

4 Quality of Transformations

4.1 Motivation

A key point in MDE is the transformation of models. This approach has been proven useful both during the development and the maintenance of software systems, allowing refinements, new views or system code to be generated from models. Transformations automate tasks that are either too tedious or complex for most developers to consistently and reliably implement [9]. One can – and should – therefore engineer and evaluate the quality of the transformation itself. For instance, it is important that the output model maintains the properties of the input model, e.g. the transformation produces consistent models [17 and 24].

Other reasons for considering the quality of transformations are due to reuse and runtime concerns. Just like software components and services should be reused when building new systems, so should transformations be reused when developing new transformations. A relevant example of a transformation repository is the ATL Transformation zoo, which is a part of the Eclipse project[1]. Having access to quality criteria for transformations would allow meaningful comparison of transformation quality according to a set of chosen quality metrics. When using transformation at runtime, additional quality attributes come into play. In some systems, e.g. safety-critical ones, response times are usually important and, thus, the transformations have to adhere to constraints on timeliness. Also, during runtime adaptation it is even more important that the transformations maintain consistency and reliability among system configurations.

4.2 Applying the Quality Framework

The quality framework for MDE presented in Figure 2 suggests starting with identifying quality goals and target objects. Improving software quality and increasing the productivity of software developers are the high-level quality goals which may be achieved by the transformation activity in MDE as the high-level target object. Further, one may discuss the quality of transformations itself. This section suggests target objects for transformation quality.

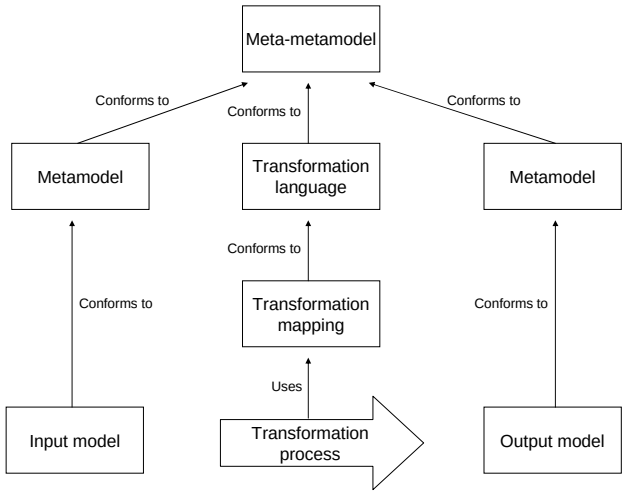

Fig. 3. A transformation is described by its transformation mapping. It takes a model as input and produces a different model as output (we can also view code as a model). Each model – the transformation mapping included – conforms to a metamodel.

[1] http://www.eclipse.org/m2m/atl/atlTransformations/

Kühne writes that a transformation is information on a mapping from one model to another, created by a transformation engineer, for the transformation engine, in order to automate a transformation process [13]. So a transformation can be regarded as a model that describes a transformation function. Hesse, on the other hand, writes that although a transformation can be modeled if one wants to do so, the static model of a transformation should not be confused with its dynamic original [8]. In his view, transformations are processes and not models. These views show how transformations have both a dynamic and a static part. To us, a *transformation* denotes the process, while a *transformation specification*, *model* or *mapping* refers to the description of this process. In our opinion, these parts are equally relevant when considering quality, and they require different approaches.

Figure 3 depicts the transformation process which can be regarded as an operator; i.e., output model=Trans(input model), with numerous properties. As can be seen, there are also additional elements involved in a transformation, which are all candidates for target objects. Our main focus, however, is on the transformation process and mapping. In Table 1 we view these two dimensions of a transformation as the target objects and suggest lower level quality goals with quality-carrying properties and evaluation techniques. These suggestions are not considered as exhaustive.

Table 1. Applying the quality framework on the quality of transformations.

Target Object	Quality Goal	Quality-carrying Property	Evaluation
Transformation process	*High performance*	Effective transformation engine [3]	Measure performance
		Select appropriate transformation approach [3]	Measure performance
Transformation model / rules	*Preservation of consistency*	Enforce consistency by tools [7]	Consistency analysis tool, measuring consistency before and after transformation
	Reusability	Modularization, i.e. specialize and chain transformations, and rule inheritance	Inspection
	Simplicity	Few number of rules, i.e. modularization	Measure complexity in the number or size of rules
		Appropriate algorithm	Measure the complexity of algorithms
		Simple output models	Measure complexity and size of the output model [22]

Target Object	Quality Goal	Quality-carrying Property	Evaluation
	Compactness	Generic transformations [27]. They contain rules where types of some object types are variables, allowing a single generic rule to handle several situations.	Inspection

5 Conclusion and Future Work

The MDE approach allows us to automate many activities in software development. Since models in MDE are expected to get progressively more complete, precise and executable during development, they can be used to evaluate and verify the quality of design, fix errors and eliminate unwanted complexity, preferably at the early stages of software development. We defined a process for defining a quality framework and based on existing literature, we provided some initial observations on transformation quality related to MDQE.

However, much work is still needed in all the stages defined in Figure 2. We will build further on the quality framework presented here to identify quality goals, quality-carrying properties and evaluation methods for aspects that affect the quality of models and are relevant for our partners in the MODELPLEX project (www.modelplex.org). One of such aspects is identifying quality criteria for Domain-Specific Languages (DSLs) appropriate for modeling large and complex systems. Suggestions for future work on transformations are further analysis of what affects the quality of transformations and to gather empirical evidence on associations between the proposed quality-carrying properties and the quality of generated software. Especially important is the development of tool support for quality engineering, as tools are such an important part of MDE. This would support the execution part of the MDE quality framework.

References

1. Berenbach, B.: Evaluation of Large, Complex UML Analysis and Design Models. In: 26th Int'l Conference on Software Engineering (ICSE'04), pp. 232--241 (2004)
2. Claxton, J.C., McDougall, P.A.: Measuring the Quality of Models. In: The Data Administration Newsletter (TDAN.com), http://www.tdan.com/i014ht03.htm, visited on June 22 (2007)
3. Cuadrado, J.S., Molina, J.G.: Building Domain-Specific Languages for Model-Driven Development. IEEE Softw. 24 (5), pp. 48--55 (2007)
4. Daniels, J.: Modeling with a Sense of Purpose. IEEE Softw. 19 (1), pp. 8--10 (2002)
5. Dromey, R.G.: Concerning the Chimera. IEEE Software 13 (1), pp. 33--43 (1996)
6. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML Make the Grade? Insights from the Software Development Community. Info and Softw Tech. 47, pp. 383--397 (2005)
7. Haesen, R., Snoeck, M.: Implementing Consistency Management Techniques for Conceptual Modeling. In: Third International Workshop, Consistency Problems in UML-

based Software Development III – Understanding and Usage of Dependency Relationships, pp. 99--113 (2004)
8. Hesse, W.: More Matters on (Meta-) Modeling: Remarks on Thomas Kühne's "Matters". J. Softw Syst Model, 5, pp. 387--394 (2006)
9. IBM, http://www.ibm.com/developerworks/rational/library/apr05/brown/index.html
10. IEEE 610.12 IEEE Standard Glossary of Software Engineering Terminology
11. ISO- International Organization for Standardization, ISO/IEC 9126-1, URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=2274
12. Krogstie, J.: Evaluating UML Using a Generic Quality Framework. Chapter in UML and the Unified Process, Idea Group Publishing, pp. 1--22 (2003)
13. Kühne, T.: Matters of (Meta-) Modeling. J. Softw Syst Model 5, pp. 369--385 (2006)
14. Lange, C.F.J., Chaudron, M.R.V.: Managing Model Quality in UML-based Software Development. In: 13th Int'l Workshop on Software Technology and Engineering Practice (STEP'05), pp. 7--16 (2005)
15. Lange, C.F.J.: Model Size Matters. In: Workshop on Model Size Metrics at MoDELS'06, 5 p. (2006)
16. Lindland. O,I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software 11(2), pp. 42--49 (1994)
17. Liu, Z., Jifeng, H., Li, X., Chen, Y.: Consistency and Refinement of UML Models. In: Third International Workshop, Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, pp. 23--40 (2004)
18. Mellor S.J., Balcer, M.J.: Executable UML: a Foundation for Model-Driven Architecture. Addison-Wesley (2002)
19. Mohagheghi, P., Aagedal, J.Ø.: Evaluating Quality in Model-Driven Engineering. In: Workshop on Modeling in Software Engineering (MISE'07), In: Proc. of ICSE'07, 6. p (2007)
20. Object Management Group's Model Driven Architecture: http://www.omg.org/mda/
21. Paige, R.F., Ostroff, J.S., Brooke, P.J.: Principles for Modeling Language Design. Info and Softw Tech. 42, pp. 665–675 (2000)
22. Saeki, M., Kaiya, H.: Model Metrics and Metrics of Model Transformations. In: The First Workshop on Quality in Modeling, pp. 31--45 (2006)
23. Solheim, I., Neple, T.: Model Quality in the Context of Model-Driven Development. In: 2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS'06), pp. 27--35 (2006)
24. Straeten, R.: Formalizing Behaviour Preserving Dependencies in UML. In: Third International Workshop, Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, pp. 71--82 (2004)
25. Trendowicz, A., Punter, T.: Quality Modeling for Software Product Lines. In: 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'03), 7 p. (2003)
26. Unhelkar, B.: Verification and Validation for Quality of UML 2.0 Models, Wiley (2005)
27. Verró, D., Pataricza, A.: Generic and Meta-Transformations for Model Transformation Engineering, In UML 2004, pp. 290-304 (2004)