A Metamodel for Specifying Quality Models in Model-Driven Engineering

Parastoo Mohagheghi, Vegard Dehlen

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

Abstract. In the context of model-driven engineering, models act as the primary artifacts and are eventually transformed into other artifacts, such as code and documentation. Consequently, to evaluate and assure the quality of software systems, developers need tools and techniques that allow them to reflect upon the quality of the models themselves. This research paper discusses existing quality models before proposing a metamodel for specifying quality models in the context of model-driven engineering. A tool is being developed in Eclipse based on the proposed metamodel. For each project, developers can use the tool, consisting of predefined concepts relevant to quality, to select quality goals based on the context of their particular project. We will use the quality models to integrate previous work on the quality issues in model-driven engineering by relating the identified quality goals to quality-carrying properties, practices or means to achieve them, metrics for evaluation and collected empirical evidence.

Keywords: Quality model, model-driven engineering, metrics, metamodel.

1 Introduction

Model-Driven Software Development (MDSD) or Model-Driven Engineering (MDE) is an approach to software development that emphasizes using models when specifying, developing, analyzing, verifying and managing software systems[1]. MDE promises to provide better communication between stakeholders, increase portability of solutions to different platforms, provide traceability between artifacts, reduce error-prone and costly manual work and improve software quality. These promises cover several of the quality goals identified in various quality models and researchers have also started work on specific quality issues in MDE such as identifying characteristics of models that are required to achieve software quality.

The work described in this paper also aims at identifying quality attributes and approaches to improve the quality of software artifacts in MDE. These are affected by the quality of modeling languages (including Domain-Specific Languages or DSLs), models, transformations performed on models, tools, modeling processes, quality

[1] We use the term MDE in the remainder of this paper to cover these approaches.

assurance activities, and the knowledge of model developers on the domain, tools and languages they are using. Therefore we define these as targets in our quality model that are subject of improvement. Earlier work on software quality covers several quality models that include various quality attributes and different classifications of them. A detailed comparison of existing quality models is out of the scope of this paper, but we describe those quality models that our work is based on or related to. Furthermore, we apply the practice of metamodeling to identify and define the elements that are required to develop a quality model for MDE. A metamodel allows us to share a common language when discussing quality and adopt general models to special needs or domains. We also provide an example of applying the approach and of an early implementation of tool support.

The remainder of this paper is organized as follows. Section 2 discusses some relevant work on quality models while Section 3 discusses why we need a quality model for MDE. Section 4 presents the matamodel of the quality model while the supporting tool for modeling is presented in Section 5. Section 6 presents an instance model developed for Domain-Specific Languages (DSLs) by using the tool. The paper is concluded in Section 7.

2 Related Work on Quality Models

The term *quality model* is often used to refer to a set of quality attributes (also called as quality characteristics) and relations between them, with the goal of evaluating the quality of something. Research on quality models has been going on for decades and different quality models have emerged. Some of the best known quality models are:

- McCall's hierarchical quality model which focuses on product quality, dividing it into the *external view* as seen by users (quality factors to specify) and the *internal view* as seen by the developers (quality criteria to build) [8]. By answering "yes" and "no" to questions related to quality criteria, one may measure to what extent a quality criteria is achieved.
- Boehm's hierarchical quality model with three levels of quality characteristics: *high-level characteristics* from the users' perspective, *intermediate characteristics* which are software characteristics needed to achieve the high-level characteristics, and *primitive characteristics* which are foundation for evaluation and defining metrics [2].
- ISO standards, especially the ISO-9126 series [5] (recently updated in the SQuaRE series of standards [3]) with the hierarchical model of six quality factors and sub-characteristics related to each of them. The standard divides metrics into *internal*, *external* and *quality-in-use* as shown in Fig. 1.
- Dromey's model, which has three main principles: quality attributes, product properties that are important for achieving quality attributes, and links between product properties to quality attributes [4]. Dromey defines a five step process for building product-specific quality models:

    1. Identify a set of high-level quality attributes for the product;
    2. Identify the product components;

3. Identify and classify the most significant, tangible, quality-carrying properties for each component;
4. Propose a set of axioms for linking product properties to quality attributes;
5. Evaluate the model, identify its weaknesses and refine it.

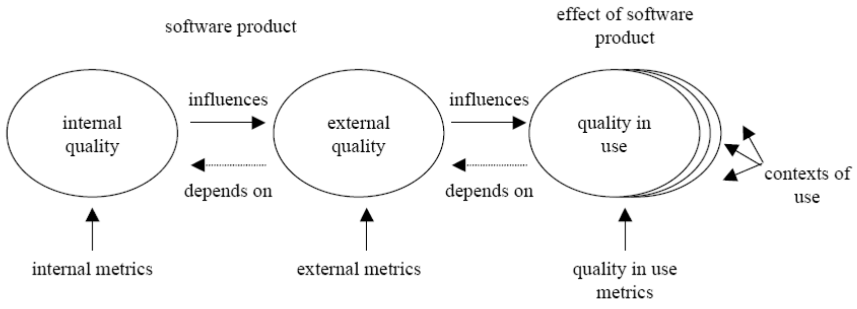


Fig. 1. Relationship between types of metrics in ISO-9126 [5].

The McCall, Boehm and ISO models share some quality attributes and differ in others, and all of them lack rationale behind the selected quality attributes. Another problem with hierarchical models is the lack of rationale behind relating sub-characteristics to quality characteristics (see [1] on problems with ISO standards). Dromey’s approach is more flexible since the quality attributes are advised to be defined based on the product. Other contributions of Dromey’s work are the emphasis on identifying tangible quality-carrying properties and establishing links between quality attributes and quality-carrying properties required to achieve them, something which is missing in the other models. For example, reliability of code can be achieved by expressions that are computable and free of side-effects. Establishing of links is done by identifying the impact of quality-carrying properties on four properties of software that impact software quality. These are defined as [4]:

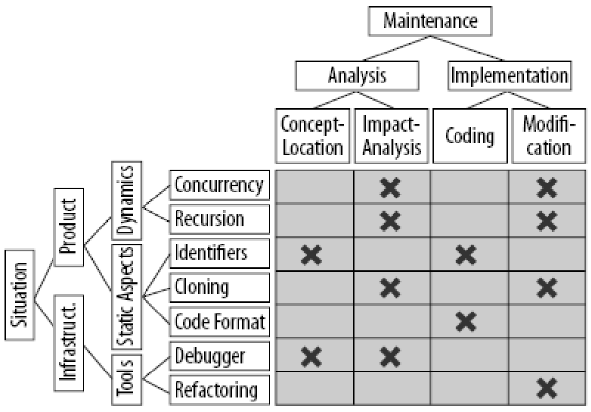
- Correctness properties; related to the deployment of components and that rules are not violated; either internally or associated with their use in the context;
- Internal properties; how well a component is deployed according to its intended use or requirements, covering both correctness and other properties;
- Contextual properties; how to compose components in a context;
- Descriptive properties; requirements, design, implementations and user interfaces must be easy to understand and use for their intended purpose.

Recognizing the overlaps and inconsistencies in different quality models, Wagner and Deissenboeck propose to define a metamodel that enables defining quality attributes in a so-called *base model*, which may be extended later to application-specific *purpose models* [15]. They have identified some elements of the metamodel to be:

- Purpose of the model; as being constructive, predictive or assessing;
- View; as being either product, user, manufacturing or value-based;
- Quality attribute such as defined in the ISO standards;

- Technique; if a quality model focuses on a specific technique, for example inspections;
- Abstractness, which is the detail of a model, for example being general or product-specific.

An example of an instance of the metamodel for quality attribute "maintainability" is shown in Fig. 2. On the left-hand side there is a tree of cost factors, i.e., the activities that impact the cost of development, while there is a tree of quality concepts above the matrix. The matrix itself shows which activities are affected by which characteristics. The paper has not presented the metamodel completely, and the example is not described well enough to understand the rationale behind developing it. However, the work emphasizes the necessity of building a metamodel and establishing relations between quality attributes and software activities.



	Concept-Location	Impact-Analysis	Coding	Modifi-cation
Concurrency		✗		✗
Recursion		✗		✗
Identifiers	✗		✗	
Cloning		✗		✗
Code Format			✗	
Debugger	✗	✗		
Refactoring				✗

Fig. 2. Maintainability matrix from [15].

Although several of the quality attributes included in the described quality models are relevant for software models as well, the focus of earlier quality models has mainly been on code. Recently, the MDE community has started research on the quality of models and modeling activities (see [9] and [10] for an overview). The working session in the 2nd workshop on Quality in Modeling (QiM'07 held in conjunction with MoDELS 2007) put three questions for participants to answer[2]:

- What qualities of models and modeling matter?
- How do they relate (similarity or dependence)?
- How can they be measured?

[2] See the workshop proceedings at http://www.ipd.bth.se/lku/Quality%2Din%2DModeling%2D2007/. The report of the working session was distributed to participants and the final version is still not available.

The qualities were further asked to be classified into (based on the model of ISO-9126 and with distinguishing also between process and project quality):

- Project quality: how well an organization executes the software process that involves modeling;
- Process quality: how well the software development process supports modeling;
- Product quality: technical properties of the model itself;
- Quality in use: how well users of models can achieve their goals.

The contributions led to identifying several quality attributes that can be included in a quality model for MDE. However, the issues of relations and measurement were not answered to the same extent, and as we discussed in relation with ISO and similar models, defining quality attributes and classifying them per se is not enough without discussing how to achieve these attributes and who are the intended users. The model proposed to organize the contributions is shown in Fig. 3 and we identify the elements of Fig. 3 in the metamodel described in this paper as well.

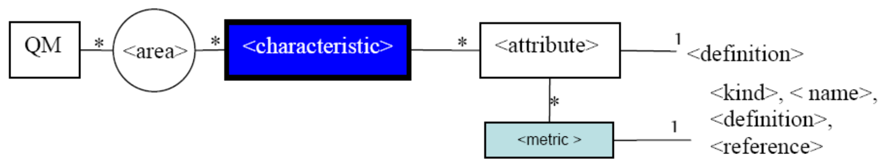


Fig. 3. Graphical form for quality attributes proposed in QiM'07 working session.

We should also mention the contribution of Lindland et al. on the quality of conceptual models that emphasizes distinguishing quality goals from means to achieve them [7], as depicted in Fig. 4. Model quality goals are divided into *syntactic*, *semantic* and *pragmatic* in their work.

The discussion on quality models verifies that some have focus on defining and measuring quality attributes, while others also emphasize identifying means and quality-carrying properties important to achieve quality attributes. Quality attributes are also defined differently in these frameworks and there is a need to identify those relevant for MDE.

3 The Need for a Quality Model in MDE

While several quality models exist, most are targeted towards evaluating source code or final products and not models or modeling activities as intermediate steps towards generating the final products. The quality model we are developing has several purposes:

- It can be viewed as a kind of research programme to facilitate the understanding of the meaning of quality in the MDE context;

- It provides a platform for collecting state-of-the-art and for classification and comparison of approaches to develop artifacts with higher quality in a MDE approach. Here we also include results of empirical studies;
- It provides a means to integrate earlier quality models.

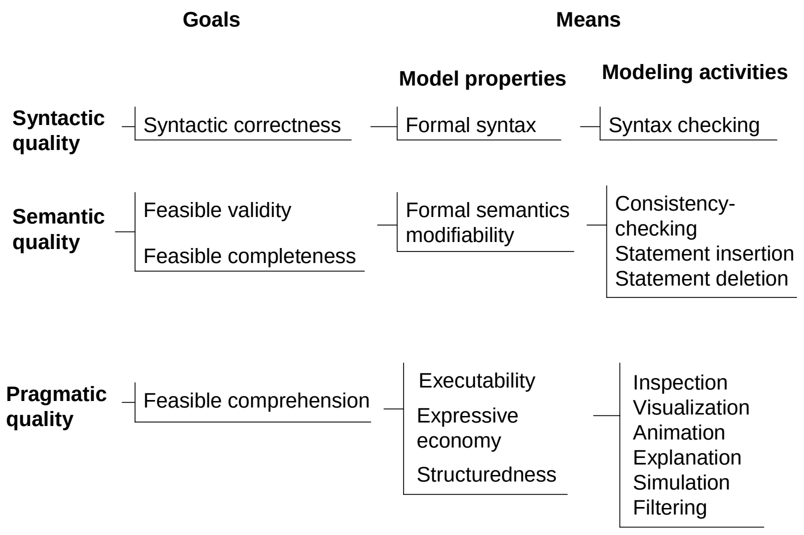


Fig. 4. Proposed framework by Lindland et al. for distinguishing quality goals and means to achieve them [7].

Our approach has focus on the quality attributes of software models and the development environment around modeling. The rationale behind this is the assumption that in an MDE development approach, several artifacts are generated from the models, and by improving the quality of the models we will consequently improve the quality of the final product. For models that are developed with the mere purpose of description or analysis, the impact on the quality of the final product is indirect and by improving the design or removing misunderstandings.

Furthermore, our quality model has a *constructive view* to quality; i.e., the purpose is to achieve the desired quality-carrying properties by implementing proper means or practices; especially MDE practices. It is also essential to evaluate the models to be sure that the desired quality has been achieved. Therefore our approach includes identifying proper methods of evaluation. Finally, the quality model provides concepts to establish links between elements of the model; i.e., between quality goals and quality-carrying properties, and between quality-carrying properties and practices. These concepts are described in the next section.

The quality model is built around a metamodel. A *metamodel* is an explicit model of the constructs and rules needed to build specific models within a domain of interest; in our case quality of models and other artifacts in MDE. A meta-model can be viewed from three different perspectives[3]:

- As a set of building blocks and rules used to build models;
- As a model of a domain of interest;
- As an instance of another model.

The purpose of defining a quality metamodel is therefore to build quality models; either general ones or specific ones for a domain. Earlier approaches have focused on models with a set of quality attributes while [15] has identified the need for a metamodel, but proposing only a few constructs of it. We discuss the constructs of our metamodel in the next section.

4 The Metamodel of the Quality Model

The main constructs of the metamodel and their relations are shown in Fig. 5 and described in the remainder of this section.

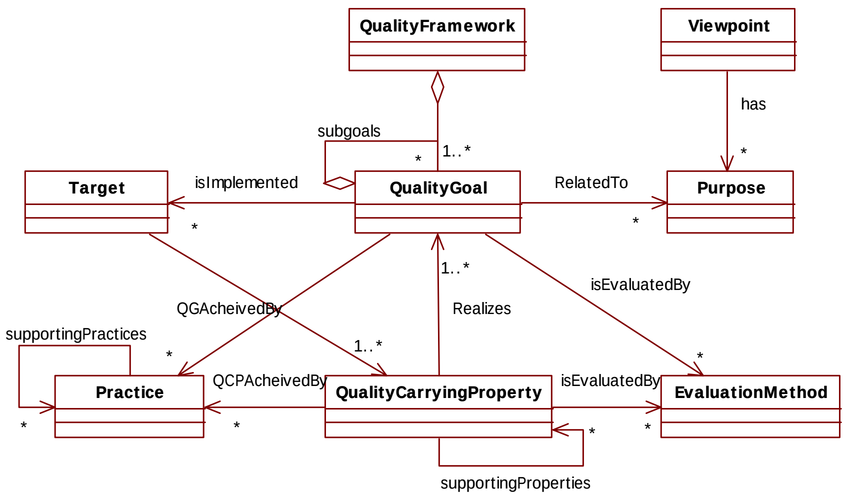


Fig. 5. Main constructs of the metamodel.

Not visible in the figure are attributes; all the metamodel elements have a “type”, “definition” and “evidence”. The use of the “type” property varies for different elements and is described in some of them, “definition” is a textual description of the element, while “evidence” is used to link evidence from empirical studies and theory to the element.

Quality framework

[3] http://www.metamodel.com/article.php?story=20030115211223271

A quality framework (or quality model as instances are called) is a collection of quality entities and their relations. It has a scope type which may be used to indicate whether it is generic or related to a specific domain. The concept of “domain” may be used to refer to application domains such as telecom or business systems, or a domain of improvement such as modeling or communication with stakeholders.

Quality Goal
We define a quality goal as a clear and understandable definition of what quality means to a stakeholder such as users of a model, developers or managers. As emphasized in [6], a good definition must let us measure quality in a meaningful way. However, we do not include “measurable” in the definition of a quality goal since many quality goals are not tangible and measurable, and stakeholders’ evaluation of quality goals such as “easy to use” is often an evaluation of a set of properties. Examples of quality goals are “consistent” for models or “easy to use” for a domain-specific modeling environment.

We have used the term “quality goal” and not “quality attribute” to avoid confusion with attributes of the constructs of the quality metamodel. One of the attributes of quality goals is “type”, which is used for classifying goals; for example by:

- Using the classification of Lindland et al. as described in Section 2;
- Classifying quality goals into “hard goals” that can be achieved by an activity or “soft goals” that can be positively or negatively affected by an activity or activities;
- Classifying into product, project or process quality goals.

Viewpoint
Viewpoint is used to indicate stakeholders of a quality goal such as model users, model developers, managers etc.

Purpose
Purpose describes what purpose a stakeholder, represented by a viewpoint, has in a given quality goal. For example, the purposes of modeling may be generation of code or documentation and these are related to specific quality goals.

Target
A target is the artifact or activity that contains the property required to achieve a quality goal. For example, the quality of artifacts developed in a MDE approach depends on the quality of models, metamodels, tools, languages, transformations, modeling process and the expertise of people involved [9]. Therefore these elements are defined as target elements in the MDE quality framework.

Target has an additional attribute called “phase” for indicating the software development phase where the target is used. For models, we can identify types such as CIM/PIM/PSM or specification/analysis/implementation/documentation or structure/behavior. These may be defined as subtypes of models.

QualityCarryingProperty
A quality-carrying property is some tangible property of an artifact or activity that is needed to achieve a quality goal. It has the same purpose as in the Dromey’s work described in Section 2; i.e., breaking down intangible quality goals into tangible

properties of targets that can be evaluated. For example, understandability of models depends on them being simple (not including too many elements) and well-organized. Thus "being minimal" and "well-organized" are quality-carrying properties relevant for understandability of models.

For this element, the "type" attribute is used mainly for linking properties to quality goals (establishing links is based on experience or proven patterns and theories),

Practice
While Dromey's quality model emphasizes identifying tangible quality-carrying properties, it does not include means to achieve these properties, as identified in the Lindland et al. work [7]. We define a practice as the means to achieve a required property. For example, using modeling conventions is a practice that can lead to developing correct and consistent models.

Evaluation method
Every property should be evaluated either quantitatively or qualitatively as defined in the evaluation method. Evaluation method includes metrics and other appropriate ways of evaluation such as expert evaluation, testing or surveys. For example, including domain knowledge in a domain specific code generator (a practice in our model) will result in less error-prone code (a property) that can be evaluated by the reduction in the number of defects (a metrics). From a value-based viewpoint with emphasize on costs and savings, it is also important to estimate how much effort is saved (less defects require less detection and correction activities) compared to effort spent on developing the code generator.

The above constructs are the main constructs of the quality metamodel and other elements inherit from them or extend them. For example, models and metamodels inherit from target, and metrics inherit from evaluation method. In addition, there is a group of relations that are not included in the model, used for indicating whether a goal, property or practice helps, breaks, depends on or realizes other elements.

As emphasized by Dromey, it is important to link properties to quality goals and from our viewpoint it is also important to link practices to properties. Dromey proposes using a classification of properties to provide such linking by classifying them into correctness, internal, contextual and descriptive (see Section 2). We have included the attributes "type" and "evidence" to establish such links. We provide an example based on the metamodel in Section 6.

5 Tool Support

We started modeling our quality models with StarUML[4]. The problem with using a general modeling tool is that models get complicated and difficult to comprehend with the increasing number of elements and links, and the possibility for extension of elements and generation of other artifacts from models is limited. To support

[4] http://staruml.sourceforge.net/en/

developers in specifying quality models for different domains or purposes and gaining experience with our metamodel, we provide tool support for the metamodel described in Section 4. An early version of this tool has been implemented on the Eclipse platform using the Graphical Model Framework (GMF). Eclipse is widely used as a tool and development platform within academia and consequently provides several benefits; (1) people are experienced in using the environment, (2) using it promotes interoperability and allows our models to be used by other EMF-based tools, and (3) many plug-ins exist for possible reuse. The GMF plug-in, for example, allows one to rather quickly create a concrete syntax in a graphical editor. To support our concrete syntax, the metamodel from Fig. 5 has been extended and detailed with additional concepts.

Currently, our concrete syntax uses a UML profile-like syntax and consists of; a box, the concept name inside guillemots, and the name of the instanced concept itself. We also support showing the semantics of connectors by differentiating them either through graphics or tags. This syntax is considered temporary, and our intention is to increase the use of graphics to differentiate concepts. The flexibility of GMF in defining graphics and icons is also a key reason for choosing GMF over UML-profiling for our tool solution. However, having an early implementation of the tool, our plan is to test it in a use case in order to gain experience with its usability and ability to model quality frameworks. These experiences will also be the basis of the following tool iterations. We show an example model developed with the tool in the next section.

6 Example: A Quality Model for Domain-Specific Languages

One of the promises of MDE has been automatic generation of artifacts from models which apparently leads to increased productivity and cost savings. However, it is often impossible to express enough detail in transformations as required for automatic generation of code for a specific domain in standard modeling languages like UML, so they have to be customized [13]. This customization is either done by developing profiles of standard modeling languages or developing Domain-Specific Languages (DSLs). Other promises of using DSLs or profiles are bridging the communication gap between domain experts who are familiar with the domain concepts and technical experts. These benefits are widely discussed by industry as motivations to use DSLs, such as in [12] and [14]. A survey of industry experience reports showed that in fact most of them prefer using DSLs or profiles over standard modeling languages [11]. However, there is a risk that developers of DSLs do not have enough expertise in language engineering and develop languages with poor quality, or the developed languages are not maintained and updated as development goes by, and thus may be abandoned, as also discussed in [12]. The risk is higher if there is no domain organization behind standardizing the domain concepts and languages.

The European IST project MODELPLEX[5] aims at developing modeling solutions for complex software systems and is a consortium of research organizations,

[5] See http://www.modelplex.org

academia and industry partners. One of the tasks of the project focuses on evaluating DSLs that are to be developed by different partners of the project. Some of them are using the existing domain standards for developing languages; for example the Common Information Model (CIM)[6] for managing services, networks and devices which will be used as the metamodel of a language developed by a telecom company. For others, the developed language will be a step towards developing a domain-specific ontology. There will be also languages with focus on specific aspects such as security which will be integrated with other languages. Since the languages will have graphical syntax, they are in fact Domain-Specific Modeling Languages.

We have had several sessions discussing project partners' goals with DSLs. The main goals are identified to be:

- Improved communication between stakeholders, some of them being non-technical experts such as business domain experts, by sharing the same language between domain and IT experts;
- Reduced effort on manual work and faster development;
- Ease of learning for developers;
- Suitability of the language for complex system development;
- Improved software quality.

The above list shows the quality goals identified by the partners, while the targets for achieving the goals are DSLs and modeling and metamodeling tools such as the Eclipse GMF environment[7] or MetaEdit+[8]. We discuss the quality model related to DSLs in this section.

The stakeholders are identified to be developers who want to use the language for development (modeling and generation of other artifacts from models) and communication with domain experts, and project managers who take decisions on new technologies and follow project costs. From their viewpoints and purposes, a set of quality goals are identified as depicted in Fig. 6. We have also related these goals to properties and practices that support achieving them based on the existing literature as depicted in Fig. 7 and Fig. 8. A practice may also depend on implementing other practices. Some of the practices may be evaluated by yes/no questions or performing surveys while other such as the extent of generation of artifacts may be evaluated by defining proper metrics. Metrics and evidence from literature (for example those identified by our survey on experiences with MDE in industry [11]) will be inserted in the model.

[6] http://www.dmtf.org/standards/cim/
[7] http://www.eclipse.org/modeling/gmf/
[8] http://www.metacase.com/

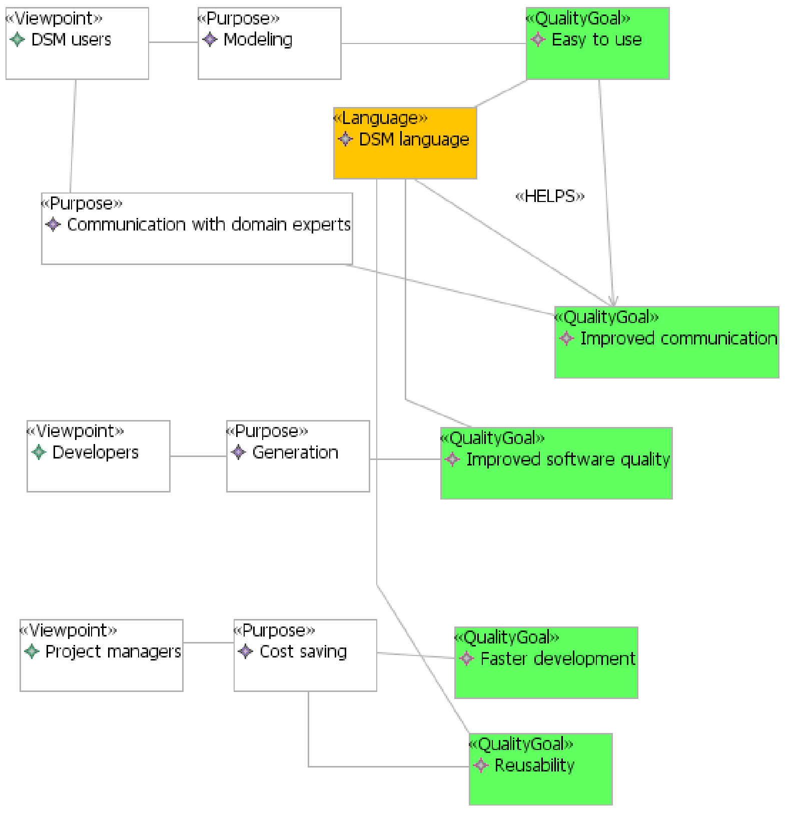

«Viewpoint» DSM users
«Purpose» Modeling
«QualityGoal» Easy to use
«Language» DSM language
«HELPS»
«Purpose» Communication with domain experts
«QualityGoal» Improved communication
«Viewpoint» Developers
«Purpose» Generation
«QualityGoal» Improved software quality
«Viewpoint» Project managers
«Purpose» Cost saving
«QualityGoal» Faster development
«QualityGoal» Reusability

Fig. 6. Quality goals defined by different stakeholders for their purposes of using DSM

We do not have space to describe the model in details but some of the elements are described below:

- To improve understandability of models, we need practices for information hiding such as aggregation and composition, developing hierarchical models, including several viewpoints or aspects in the models, and designing the concrete graphical syntax in a way that similar concepts look alike while conceptually different concepts look different (Concept Distance). Understandability is also improved if models are executable. A DSM language that is easy to use, helps communication as well. A model developed by a DSL should be at the right abstraction level; i.e., hide details of implementation that can be included in the code generator to improve understandability..
- One of the goals with DSLs is to be able to focus on one aspect at a time such as the functional model or the security model. Models that are focused on non-functional aspects such as security should be reusable in several systems and therefore be language independent. This practice will improve reusability of models.

- Some practices such as automatic generation of code from models impact several quality-carrying properties positively such as improved traceability and increased productivity by reducing manual work.

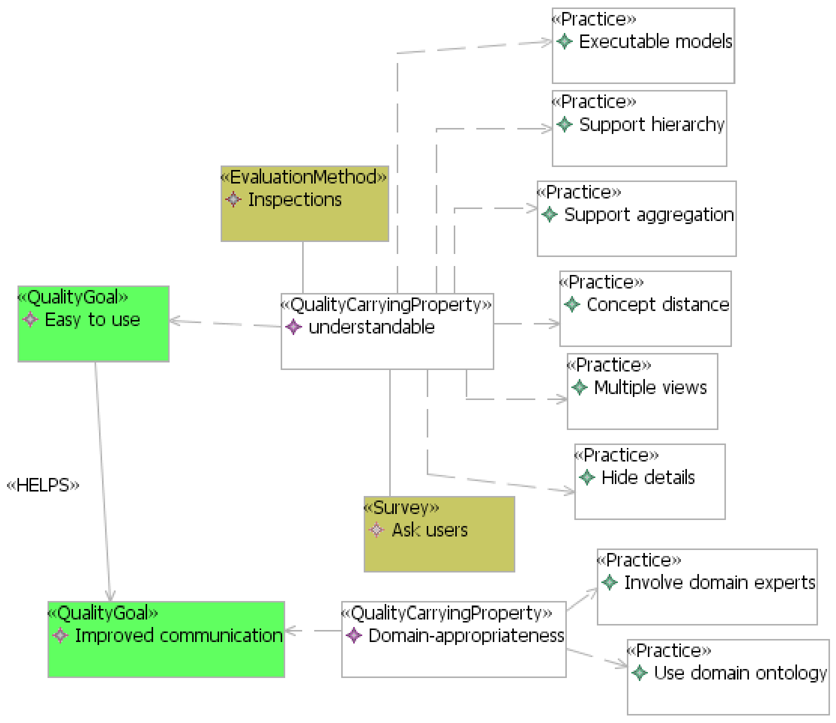

«Practice» Executable models
«Practice» Support hierarchy
«EvaluationMethod» Inspections
«Practice» Support aggregation
«QualityGoal» Easy to use
«QualityCarryingProperty» understandable
«Practice» Concept distance
«Practice» Multiple views
«HELPS»
«Practice» Hide details
«Survey» Ask users
«Practice» Involve domain experts
«QualityGoal» Improved communication
«QualityCarryingProperty» Domain-appropriateness
«Practice» Use domain ontology

Fig. 7. The detailed model for improving ease of use and communication by using DSM language

To perform an initial evaluation of the developed languages for the above properties, we will develop the smallest system that is practical (and includes all the concepts of the DSL) and perform evaluation to evaluate whether the required practices are implemented and how they impact quality goals. The final goal is to include the results of the state-of-the-art analysis to the model, add proper metrics and detail the model to a level that it can be used as a guideline for developing DSLs in the MODELPLEX project and in future work.

The tool is under development and we experience with various models such as the ones showed here. For example, we consider whether it is best to define all the evaluation methods in the metamodel (for example metrics in Fig. 8) or not.

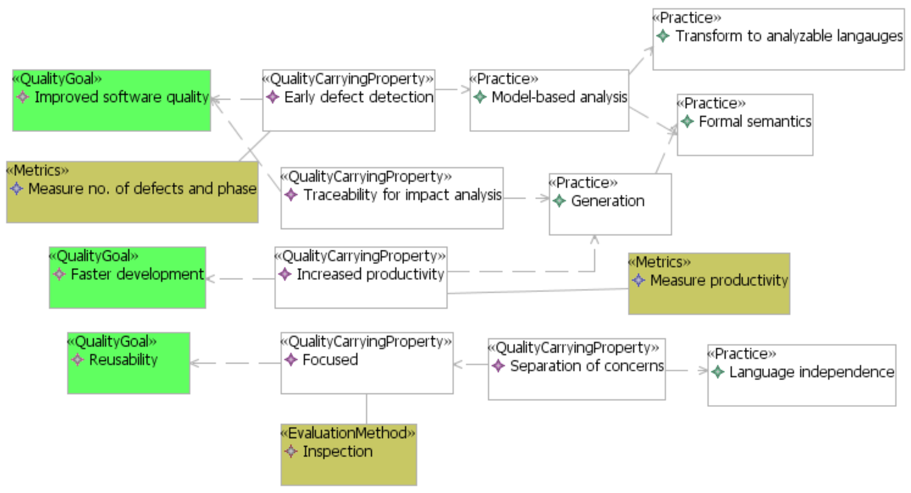


Fig. 8. The detailed model for improving software quality, achieve faster development and reusability by using DSM language

7 Conclusions and Future Work

This paper started by giving an overview over selected quality models for software engineering, which uncovered both weaknesses and strengths of existing approaches. Quality models should give a sound rationale for the selected set of quality characteristics and explain the relationships between concepts. One way of avoiding a static set of characteristics, which should rather be selected dynamically depending on the stakeholders' needs, is to have a dynamic and flexible framework that allows developers to define a quality model based on the context; whether it is a general purpose quality model or one tailored for a specific context. A way of achieving this is to provide a metamodel and supporting tool. Our metamodel includes a set of important concepts that needs to be considered when defining quality models, and takes into consideration the previous work done in the field of quality in software engineering. Metamodeling is one of the main practices of MDE and we take advantage of it to develop quality models for MDE as well. Although we focus on the quality of models and MDE practices, the metamodel is generic and may be used for defining quality models in any area. We also have a first implementation of a tool that offers graphical modeling of quality models, allowing quick development and testing.

For future work, we plan to use our framework and tool to implement the quality model for DSLs outlined in Section 5, and detail it with more metrics and evidence from earlier studies. In addition, we have analyzed literature on approaches to improve the quality of models and will insert the work in a base model that can be extended by users. In parallel and based on experiences from these cases, we also plan to iterate on our tool support, in order to provide an expressive and easy-to-use language for modeling quality models.

Acknowledgments. This work has been co-funded by the Quality in Model-Driven Engineering project (cf. http://quality-mde.org/) and the European Commission within the 6th Framework Programme project MODELPLEX contract number 034081 (cf. http://www.modelplex.org).

References

1. Al-Kilidar, H., Cox, K., Kitchenham, B.: The Use and Usefulness of the ISO/IEC 9126 Quality Standard. International Symposium on Empirical Software Engineering, 7 p. (2005)
2. Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M.: Characteristics of Software Quality. North Holland (1978)
3. Bøegh, J.: A New Standard for Quality Requirements. IEEE Software 25(2), 57--63 (2008)
4. Dromey, R.G.: Concerning the Chimera. IEEE Software 13 (1), pp. 33--43 (1996)
5. ISO, International Organization for Standardization: ISO 9126-1:2001, Software Engineering – Product Quality, Part 1: Quality model (2001)
6. Kitchenham, B., PFleeger, S.L.: Software Quality: The Elusive Target. IEEE Software, 13(1), pp. 12--21 (1996)
7. Lindland, O.I., Sindre, G., Solvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software 11(2), pp. 42--49 (1994)
8. McCall, J. A., Richards, P. K., Walters, G. F.: Factors in Software Quality. Nat'l Tech. Information Service, Vol. 1, 2 and 3 (1977)
9. 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)
10. Mohagheghi, P., Dehlen, V.: Developing a Quality Framework for Model-Driven Engineering. 2nd Workshop on Quality in Modeling at MoDELS 2007, 15 p., URL: http://www.ipd.bth.se/lku/Quality%2Din%2DModeling%2D2007/ (2007)
11. Mohagheghi, P., Dehlen, V.: Where is the Proof? – A Review of Experiences from Applying MDE in Industry. Proc. 4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA'08), LNCS 5095, p. 432—443 (2008)
12. Safa, L. The practice of deploying DSM, report from a Japanese appliance maker trenches. In Proceedings of the 6th OOPSLA Workshop on Domain Specific Modeling (DSM'06), 2006, 12p.
13. Staron, M., Kuzniarz, L. and Wallin, L. Case Study on a Process of Industrial MDA Realization: Determinants of Effectiveness. Nordic Journal of Computing 11(3), 2004, 254 – 278.
14. Trask, B., Paniscotti, D., Roman, A. and Bhanot, V. Using model-driven engineering to complement software product line engineering in developing software defined radio components and applications. In Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'06), 2006, 846 – 853.
15. Wagner, S., Deissenboeck, F.: An Integrated Approach to Quality Modeling. Fifth International Workshop on Software Quality, In: Proc. of ICSE'07, 6 p. (2007)