Information Quality Management – a Model-Driven Approach

Anna Wingkvist, Morgan Ericsson, and Welf Löwe
Linnaeus University, Sweden
anna.wingkvist@lnu.se, morgan.ericsson@lnu.se, welf.lowe@lnu.se

Abstract. Large amounts of information are produced on a daily basis. We include any form of electronic data in the term information, for example e-books, web sites, software, databases, etc. Several studies have investigated the quality of this information, and often find it lacking. It is a complex process of producing said information but also to define and assess quality of the information, especially across the many different forms. We present a model-driven approach to information quality management, were models and indicators are used to define and assess quality. We rely on abstraction of models, which are described using meta model. We show how this model-driven approach is implemented by a software tool that (i) reads information, (ii) performs analyses on this information, and (iii) visualizes the results, to help stakeholders understand quality issues. The software tool has been used to evaluate the quality of real world software and documentations.

1 Introduction

Quality assessment and assurance is an important part of any information production process, such as technical documentation (Hargis et al. 2009) or software. A lack of quality reduces not only the value of the information, but also of the product (service and/or process) it is attached to. It can also reduce the perceived quality of the brand or company associated with it (Smart et al. 1996).

Quality depends not only on the information, but also the product and the context it is used within. A change to the product will reduce the quality of the information if it is not updated to reflect the change. Hence, it is important that quality assessment and assurance are continuous processes. In this paper, we present a model-driven approach to information quality management. We take a total perspective on information quality, and include software, source code, data, and information (processed data). We also present a software tool that implements our model-driven approach and uses visualization to communicate quality. The (software) tool, VizzAnalyzer, is available as Open Source Software and can be downloaded from `http://arisa.se`.

The paper is organized as follows. We begin by discussing how models can be used to describe quality concepts and capture abstractions of information. We then discuss how indicators exist in both types of models and the importance of defining the various models. We continue with the use of meta and meta meta modeling to describe models and how mappings are used to support

abstractions between (meta) models. We then provide an example of how models interact to assess the quality of software and documentation, as well a discussion of how our tool implements the model-driven approach. We end with related work and future directions for this research.

2 Abstraction, Models, and Quality

In this section we introduce the main concepts of our approach to quality assessment: models and indicators. We rely on models to describe quality and to represent (abstractions of) the information we want to assess. Indicators exist in both types of models, and are used to assess quality. Indicators and models are abstractions of concepts from the real world.

A model consists of entities and their relations, and is generally represented as a graph. We differ between two types of models, the quality model and the information model, for semantic reasons. The quality model is used to represent a view on quality, where quality concepts are decomposed until they can be measured. For example, consider the Goal-Question-Metric model (Basili et al. 1994), where a quality Goal is decomposed into one or more Questions, which in turn are decomposed into one or more Metrics. The Metrics are measurable, and contribute to answer the Questions and determine if the Goals are reached. We take a more general approach, and allow for arbitrary many levels of decomposition. The entities of our quality model are desired quality concepts, and the relations are *consists-of* relations. We consider any concept that is not further decomposed as something that is (or should be) measurable. Figure 1 depicts (part of) a quality model. In this model, "Sentence lenght" is not decomposed, and should hence be measurable.

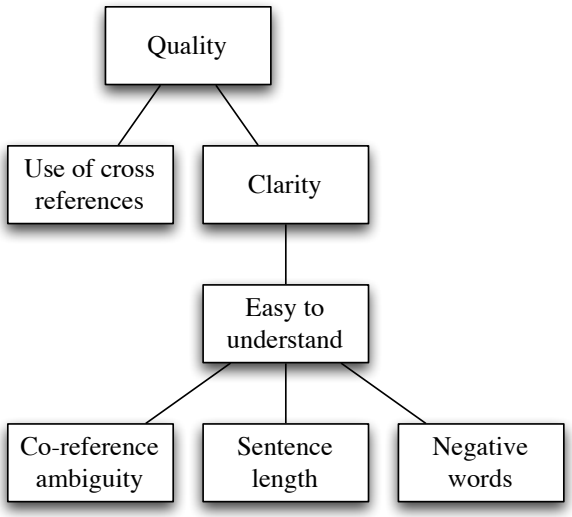

Figure 1: A quality model with concepts *Easy to understand* and *Use of cross references*. Easy to understand is decomposed to *Clarity*, which is decomposed to *Co-reference ambiguity*, *Sentence length* and *Negative words*.

The information model represents abstractions of artifacts (e.g., computer programs or technical documentations). The entities represent structural parts of the artifact, such as classes, chapters, methods, etc. We differ between different types of relations between entities, such as the structural containment relations or reference relations (such as method calls or cross references in text). Figure 2 depicts (part of) the information model of an XML document. If we consider the XML document and the information model, both are abstractions. We extend this idea and consider any information we want to represent as a series of models.

An indicator is a part of a model that is used to assess quality. These exist in both quality and information models. An indicator is a combination of metrics, analyses and/or thresholds (or levels). A metric is a quantitative measurement of one or more properties of an artifact. An analysis breaks down complex relations and/or properties, into something smaller and manageable. Thresholds and levels are used to interpret analyses and metrics, from a quality perspective. A quality model can be considered a hierarchy of indicators that are mapped to indicators in the information model.

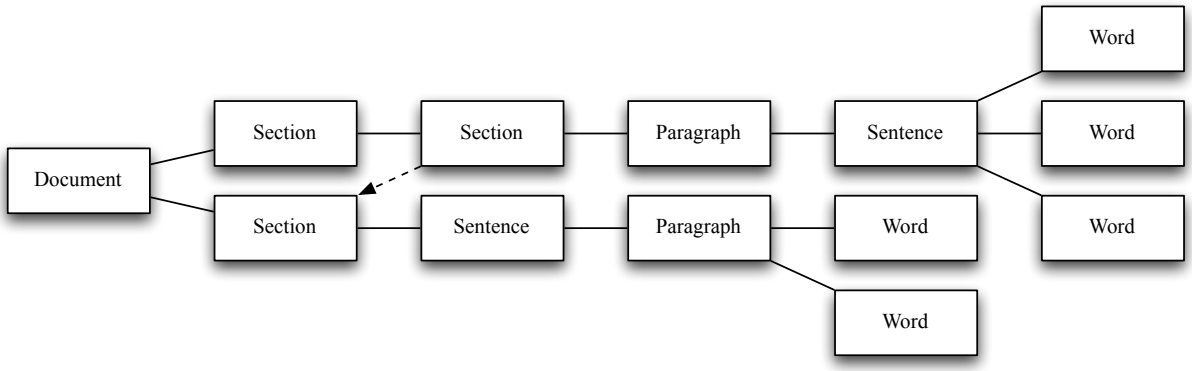

Figure 2: The information model of an XML document that consists of sections and paragraphs. Note that sections can reference other sections.

Consider the quality model depicted by Figure 1. The quality concepts *Easy to understand* is decomposed to *Clarity* and *Use of cross references*. Clarity is in turn decomposed to *Co-reference ambiguity*, *Sentence length* and a number of word use analyses such as *Negative words*. The information model depicted by Figure 2 defines a *Document* that consists of *Section*s, which in turn consists of *paragraph*s. Each paragraph consists of *Sentence*s, that each has a number of *Word*s. Sections can reference other sections.

The quality model is defined in terms of indicators, with analyses, metrics and thresholds, that are provided by the information model. For example, Co-reference ambiguity uses co-reference resolution to determine what object each noun and pronoun refers to and marks those that can refer to more than one object as ambiguous. Co-reference resolution is an analysis, the count is a metric, and one is the threshold. The indicator operates on words, which are provided by the information model.

There are three types of indicators in the quality model depicted by Figure 1: language, structure, and both. In general, top level quality concepts (such as Easy to understand) are weighted combinations of the concepts they are decomposed into, hence, top level indicators are of type both, while lower level indicators concern either language or structure. The Co-reference ambiguity illustrates another important property; indicators are often defined over a small (local) subset of entities and relations from the information model. The Co-reference analysis is only concerned with Word and Sentence entities. The reason that indicators are defined over a small subset is due to the hierarchical nature of the quality models.

The clear division of the quality model into three indicator types suggests that it is actually three related models, where the total quality model can be decomposed into structure and language quality models. This allows for more focused, and less complex quality models. However, while certain quality indicators are local to a specific model, it also shows the need for combined models. Apart from the decomposition illustrated by this example, we also find that models can be mirrored and transformed. A mirrored model contains indicators that are relevant for indicators in other models (cf. Section 4 for an example), for example a text that references variable names in a source code. A transformation model considers the quality of transformation processes, such as converting an XML document to PDF.

3 Meta Modeling and Mappings

In this section we introduce meta modeling, a technique that we use to define models. In the previous section, we discussed how we think of models as entities and their relations. We also discussed how there exist different types of entities and relations. These are all defined by meta models. We also discussed how we rely on abstractions of models to represent information, and these abstractions are enabled by meta models.

Consider a documentation represented using XML. The XML representation abstracts certain aspects, such as layout, font size, font style, etc. There is a need to define what elements are valid in the model. We rely on meta modeling, and use higher level models, meta models, to define models on lower levels. For example, an XML document is often defined by a specific XML Schema or Document Type Definition (DTD). The XML document can be considered a model of a documentation, and the XML Schema or DTD is the meta model that defines what elements the model can contain.

We use abstraction to simplify models, e.g., by reducing the number of elements, and to reduce the number of different models, e.g., by defining general elements. For example, XHTML has six levels of headings. We can use abstraction to define a general Heading element, and map the six levels to this single element. We reduce the number of elements in the model, and rules that applies to headings only needs to be defined once. We can also use abstraction to define a Section element and map all XHTML headings to this Section element. The Section element is more general than a Heading element, and is used by many different documentations, so this abstraction allows us to represent a wide range of XML documents using a single model.

We extend the idea of abstraction and models, and consider any information we want to rep-

resent as a series of models. For example, an XHTML documentation is a model of the actual documentation, and an abstraction where different level headings are represented using a Section element is another model of the same documentation. Each of the models are defined by meta models, which allow us to define mappings between different models.

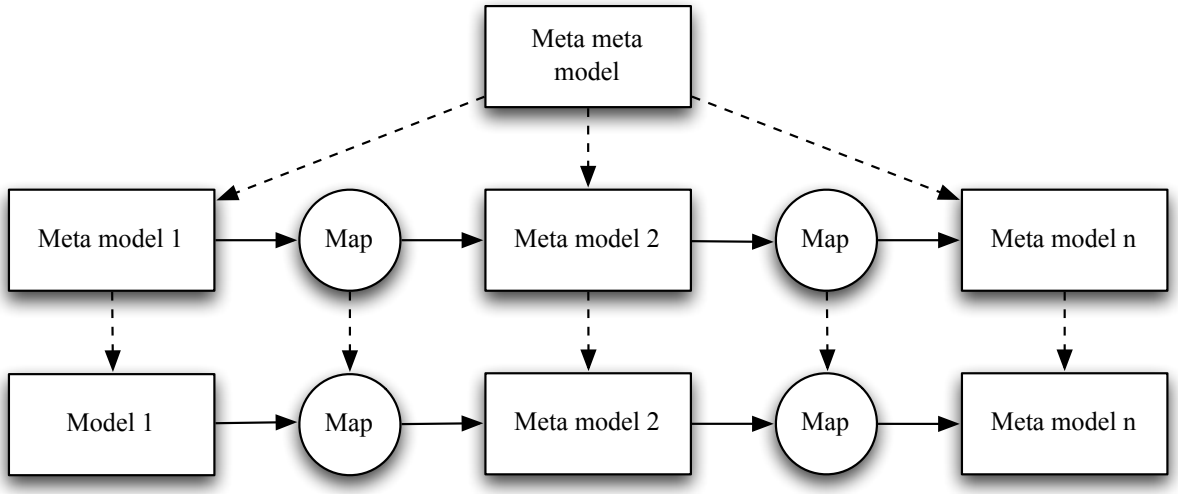

Figure 3: The three layers of models (meta modeling) and three abstraction levels of the model and meta model layers. Models are depicted by rectangles and mappings are depicted by circles.

We define abstractions of meta models as well as models, so there is a need for mapping between meta models. Consequently, there is a need to define the valid elements of a meta model, i.e., there is a need for a meta meta model. This meta meta model defines what can be expressed by a meta model. So, for each model, there exist a specific meta model. All of these specific meta models are described by a single, meta meta model. This relationship is depicted by Figure 3.

In Section 2 we considered models as entities and their relations. Each meta model is defined by a specific Entity/Relationship (E/R)-diagram, that defines what types of entities and relations that can be used by the model. A model is an instance of an E/R-diagram, and the meta meta model is defined by the rules of E/R modeling. Any E/R-diagram is a possible meta model, and anything that can be modeled by an E/R-diagram is a possible model. Figure 4 depicts an E/R-diagram that represents the meta model used in Section 2. Indicators are generally defined by relations and attributes of the entities.

The mappings defined between abstractions of models and meta models is an important concept of our approach to quality management. In Section 2 we addressed how often we find that quality concepts and indicators are local to specific parts of the information, for example, language indicators are only concerned with the text, and not the structure. So, even if the meta model defines a large set of entities and relations, most indicators use a small subset of these. Abstractions and mappings allow us to create specific subset models that precisely captures the exact entities and relations that are used by specific quality concepts and indicators. For example, if we are

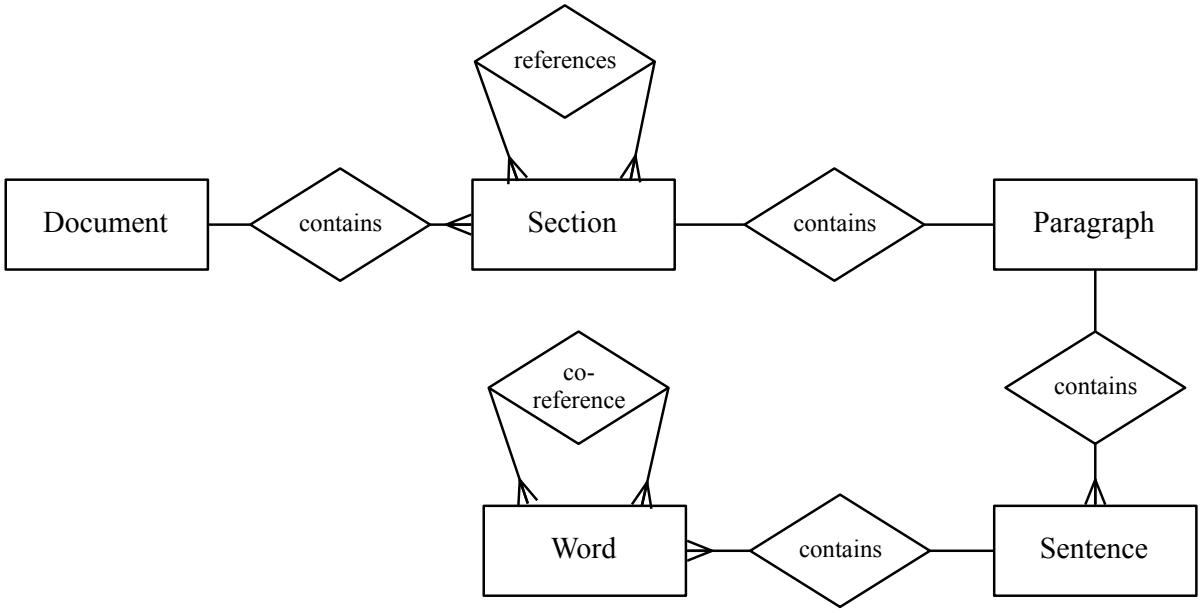

Figure 4: The information meta model for the model depicted by Figure 2 described as an E/R-diagram.

interested in the quality of the (natural) language, we create a specific model that only contains words, sentences and paragraphs. Mappings allow us to map to and from the larger model, which also contains structure information, to this more precise model.

The use of indicators and precise models that are mapped from a more general model allows for two major benefits. First, the indicators used by the more precise models are reusable to a larger degree. Second, we can express the general model using a set of more precise models, which makes the more general model redundant. As discussed in Section 2 indicators are abstract concepts defined between the quality and information models, and as such, many of these can be quite general.

For example, cross references exist in XML documents and in source code. If we use abstraction and precise models, we can disregard the context that these abstractions exist in, and thus reuse the same cross reference analysis for both XML documents and source code. To be able to replace part of the general model by very precise models is an important optimization and implementation technique, since we virtualize the model. Some of the documentations we analyze contain hundreds of thousands of words. In the quality model discussed in Section 2, we are only interested in properties of single words, the length of sentences and ambiguous co-references. Each of these analyses can be performed without constructing a complete model of paragraphs, sentences, and words. The use of precise models allows us to perform this optimization, and only construct the parts of the model we actually require.

4 Assessment of Source Code and Documentation

In this section we consider a larger example where we use all the concepts defined in the previous two sections. We want to assess the quality of software by assessing the source code and the documentation (i.e., the comments in the source code). We will not go into detail regarding the specific indicators and quality concepts, but focus on how the models interact.

While some quality concepts might overlap between the source code and the comments (natural language), most will not. Source code concepts generally relate to size, complexity, dependence, etc, while comments concepts generally relate to size, clarity, etc. Our use of models, abstractions and mappings allows us to create separate quality and information models for the source code and comments, resulting in four models. We can in turn create more precise models of each of these, as discussed in Section 3. The use of a virtualized general model is particularly interesting in this case, since there exist a number of source code and language analysis tools and toolkits that we can reuse to populate the precise models, without the need of constructing the general model (and implementing all indicators in our tool).

For example, to support the indicator "Co-reference Ambiguity" (c.f. Figure 1), we need an information model that contains sentences and words, as well as a set of analyses to determine linguistic properties of the sentences and words, such as part-of-speech tags. An information model that considers individual words will have a large number of entities for non-trivial documentations and source codes, which in turn makes analyses computationally expensive (i.e., graphs with millions of nodes). The require analyses to determine linguistic properties, as well as the co-reference resolution are complex, multi-stage analyses that require significant effort to implement. If we consider the indicator, we are only interested in the number of words that can refer to more than one object, so the information model with words, and all the determined linguistic properties are only used to support the single analysis. If we instead use a virtualized general model, we can use an existing implementation of co-reference resolution (Lee et al. 2011), check if there are any ambiguous references, and attach these to a suitable text unit (such as comment or paragraph). This results in a virtualized general model, where we only have the value for the indicator, not the supporting models and analyses. We support all quality concepts, but gain implementation and execution speed.

We can now determine the quality of the software by determining the quality of the source code and the comments in isolation, and then compute a weighted value from the two. However, there are quality concepts that are defined over both models, that use indicators that co-exist in the two models. For example, consider the accuracy of the comments and the completeness of the program. Accuracy in terms of comments can be defined as every source code element that appears in the comment should appear in the source code. Completeness can be defined as every parameter of a method should be documented by the comment. In order to determine this, we for example need to extract the parameters of a method, the parameters mentioned in the comment and compare these two sets. To assess accuracy and completeness, we need to define a new information model that pairs indicators from each of the models. We refer to such as model as a mirrored model.

In total, we define four "major" information and quality models to assess the source code. We

define models for the source code, the comments, and a mirrored model, and we define a combined model that weights concepts from these three models. We have not discussed (information) meta models, but each of the source code, comments and mirrored models need meta models. The meta models might overlap, and can be combined to a single, unified meta model. Depending on what indicators we use in the combined quality model, we may or may not need to define an information model or a meta model for it.

5 Tool Support

In this section we describe how we use model, meta models, and mappings to implement a software tool to assess quality. The tool has been successfully applied to both software and information quality assessment. We begin by describing the specific types of models that we use, and then we define the main components of the tool. The tool is described in more detail in Ericsson et al. (2011) and applications in Wingkvist et al. (2010*b*). Note that we use the term document to refer to any contained information, such as a source code file or an XML document.

We rely on three abstractions, Document-Specific, Common, and Analysis-Specific. First, there is an actual, Document-Specific model that corresponds to the actual format/schema used. This model is mapped to a more abstract, Common model, that can represent (abstractions of) any Document-Specific model that we need to support. Finally, we use an Analysis-Specific model to keep analyses reusable, even if the Common model is changed to support other Document-Specific models. The Analysis-Specific models are implementations of the precise models discussed in Section 3. Each of the three meta models are defined by E/R-diagrams, and we use E/R-diagramming as the meta meta model. Figure 5 depicts the model layers and the levels of abstractions within the model and meta model layers.

The three abstractions is a simplification of the more general modeling described in Section 2. We can use Analysis-Specific models to achieve a large degree of the generality, but we still require a single, Common meta model to describe all entities and relations, that the Analysis-Specific models abstract from.

The tool consists of three major components: (i) the *Common Information Repository* that captures models, (ii) *readers* that map to the repository, and (iii) *analyses* that assess and modify the repository and *visualizations* that present interactive views of the repository to stakeholders. The analyses are an implementation of the indicators.

The Common Information Repository is the center of the tool. It captures models, and any analysis or visualization uses data stored in it. As discussed in Section 2 there exist several different models. The Common Information Repository captures models according to the Common Meta Model. Models that are described by various Analysis-Specific Meta Models are computed from the repository.

The implementation of the Common Information Repository (classes and methods in Java) is automatically generated from a meta model, and can be adapted to the information that we want to analyze. We generally use the Common Meta Model as a configuration parameter to instantiation of the tool.

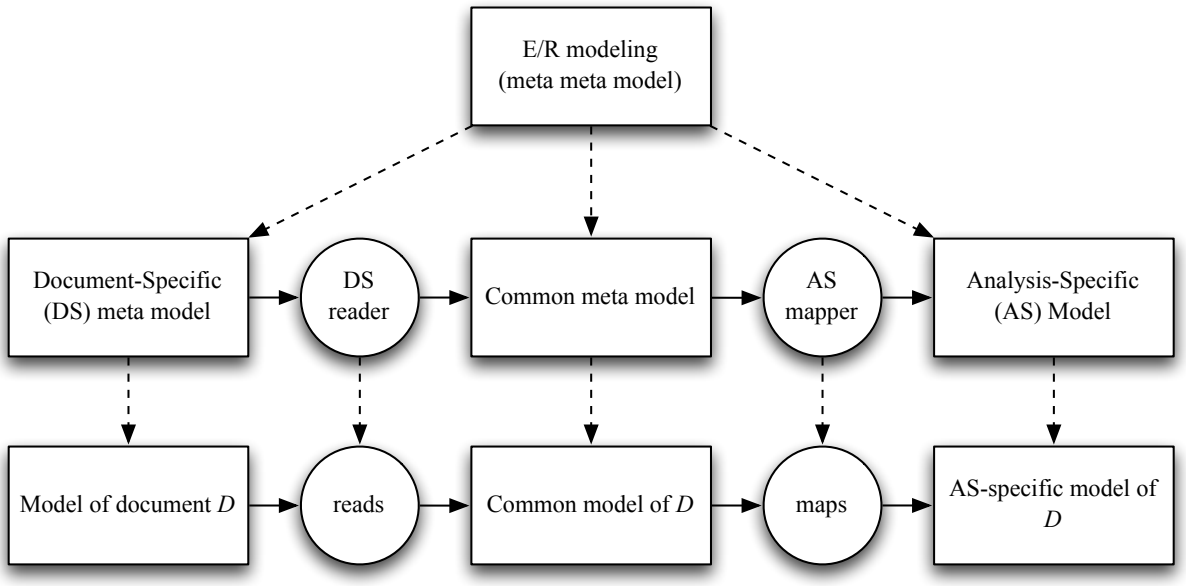

Figure 5: The simplified model structure used by our implementation.

To apply predefined analyses and visualizations, there is a need to map information that is captured in a specific format, such as XML, to the Common Meta Model. This process can be considered a two stage process, where we first create a model of the documentation according to the Document-Specific Meta Model. We then apply a mapping to map this model to entities and relationships that exist in the Common Meta Model. The mapping of the containment trees is defined recursively. Starting at the root, we traverse the document-specific containment tree in Depth-First-Search (DFS) order. We create new Common Meta Model entities for Document-Specific Meta Model entities of types that have a mapping defined. These are referred to as the relevant entities. The other, irrelevant document-specific entities are ignored. Note that the current implementation of the tool does not support a virtualized Common model. We allow for certain, expensive indicators to be determined by the Readers, but we do not support Analysis-Specific Models that do not abstract from information that is present in the Common model.

Analyses such as cross reference analysis or co-reference analysis read from and write to the repository. Since the repository and the models of the information it contains are independent of the document format, analyses can easily be reused for several different types of documents.

We use visualizations to communicate information to stakeholders about the quality. A visualization defines a mapping from the information contained in the repository to a visual domain. The visual domain contains objects with various attributes, such as shapes, colors, textures, and positions. The mapping between the two brings meaning and context to the visual objects in terms of quality.

A visualization is a specific kind of analysis that operates on specific views of the repository. The mappings created by these analyses can be configured. We map repository entities (of different

types) to visual objects (of different shape or color) and map their quality attributes to visual attributes of these objects.

6 Related Work

Data and Information Quality are commonly considered multi-dimensional (Klein 2001) and context-dependent concepts. A common definition of (information) quality provided by Juran (Juran 1998) is "fitness for use".

Hargis et al. (2009) discuss quality of information and define nine quality characteristics, for example, accuracy and clarity. These nine quality characteristics are divided into three categories: easy to use, understand, and find. The characteristics are quite broad, e.g., clarity does include concepts as conciseness and consistency.

Arthur & Stevens (1992) present a framework to assess the "adequacy" of software maintenance documentation. Their framework defines quality as four Document Quality Indicators (DQIs): accuracy, completeness, usability, and extendibility. The DQIs are decomposed into Factors that refine a Quality, and Factors are decomposed into Quantifiers that are used to measure a Factor. A Factor of accuracy is consistency, which includes the Quantifiers: conceptual and factual consistency. Wingkvist et al. (2010*b*) present a similar model for technical documentation, which is based on the idea of Key Performance Indicators that are considered an application of the more general Goal-Question-Metric (Basili et al. 1994) approach from (Software) Quality assessment.

There exist several quality frameworks for data and information quality, for example Wang & Strong (1996), Stvilia et al. (2007), Naumann (2002), Chidamber & Kemerer (1994). Knight & Burn (2005) provides an overview of some of these, and Ge & Helfert (2007) provide a review of the research in information quality (including frameworks). Most of the frameworks are hierarchical, and group information quality dimensions. For example, the framework by Wang & Strong (1996) groups 16 quality dimensions, such as accuracy and relevance, into four dimensions (i.e., Intrinsic, Accessibility, Contextual, and Representational IQ). Many of the quality dimensions are common to several frameworks. Knight & Burn (2005), for example, discovered that out of 12 surveyed frameworks, eight included accuracy and seven included consistency.

In many cases, information quality is assessed using surveys, e.g., Huang et al. (1998) and Lee et al. (2002). The framework discussed by Arthur & Stevens (1992) is assessed using a checklist approach, which is also used by Stvilia et al. (2007). Hargis et al. (2009) suggests a combination of surveys and checklists. Wingkvist et al. (2011) suggest to complement surveys and checklists with quality metrics (derived from software quality metrics) to automate the quality assessment. They acknowledge that the process cannot be fully automated, due to qualities such as ease of understanding, and suggests information testing as a complement (Wingkvist et al. 2010*a*).

The tool set and meta models discussed in this paper are inspired by research on software quality assessment. It relies on high-level abstractions, i.e., meta models, of software that contain enough information to support analyses. Meta models relevant in the software quality assessment community include the object-oriented FAMIX meta model developed in the European Esprit Project FAMOOS (Bär et al. 1999), and the Dagstuhl Middle Meta (DMM) model (Leth-

bridge et al. 2004). Strein et al. (2007) presented the ideas of software meta model definition and evolution. It provides a sound foundation that information quality assessment can relate to.

7 Conclusions and Future Work

This paper presents a model-driven approach to information quality management. We discuss indicators that are present in different models, then how we describe the models using meta modeling, and further how we can abstract (map) between models. We introduce a number of important concepts, such as how quality can be expressed in terms of models and relations between models, such as mirrored models. The use of precise models allows us to define more general indicators that can be reused between models as well as virtualized models to allow us to optimize expensive computations (to asses indicators) and/or reuse external tools.

We also present an implementation of our approach. The implementation has been used in a number of real world scenarios to assess quality of both software and information, such as the documentation of mobile phones and warships. We are currently collaborating with Sigma-Kudos and Ericsson to develop quality models and implement these in our tool to assess the quality of their technical documentation.

We are currently working on extending the types of indicators that we support, based on research results from natural language processing, information retrieval, etc. We work close with our industry partners to validate how well the indicators capture quality and how well we can determine the indicators. We are conducting evaluations of how accurately our analyses can be used to detect quality defects, and if a continuous quality assessment and assurance based on our infrastructure has any impact on the perceived quality of technical documentation.

Some of the concepts discussed in this paper are currently not supported by our tool, or supported in a limited way. For example, the mirrored models discussed in Section 4 require a single Common model that captures language, source code, and any indicators of them. Virtualized models are partially supported by Readers. We are currently investigating how to implement these features, and if they are required. Part of this investigation is to implement the analysis described in Section 4.

Acknowledgment

We would like to extend our gratitude to Applied Research in System Analysis AB (ARiSA AB, `http://www.arisa.se`) for providing us with the VizzAnalyzer tool, to Sigma Kudos AB, (`http://www.sigmakudos.com`) for providing us with their Content Management System (DocFactory), and to Ericsson (`http://www.ericsson.com`) for providing us with data. The research presented in this paper was in part funded by the Knowledge Foundation (KK) and the Swedish Governmental Agency for Innovation Systems (VINNOVA).

References

Arthur, J. D. & Stevens, K. T. (1992), 'Document quality indicators: a framework for assessing documentation adequacy', *Journal of Software Maintenance* **4**, 129–142.

Bär, H., Bauer, M., Ciupke, O., Demeyer, S., Ducasse, S., Lanza, M., Marinescu, R., Nebbe, R., Nierstrasz, O., Przybilski, M., Richner, T., Rieger, M., Riva, C., Sassen, A., Schulz, B., Steyaert, P., Tichelaar, S. & Weisbrod, J. (1999), 'The famoos object-oriented reengineering handbook', http://www.iam.unibe.ch/~famoos/handbook/.

Basili, V. R., Caldiera, G. & Rombach, H. D. (1994), The goal question metric approach, *in* 'Encyclopedia of Software Engineering', Wiley.

Chidamber, S. R. & Kemerer, C. F. (1994), 'A metrics suite for object-oriented design', *IEEE Transactions on Software Engineering* **20**(6), 476–493.

Ericsson, M., Wingkvist, A. & Löwe, W. (2011), A software infrastructure for information quality assessment, *in* 'Proceedings of the 16th International Conference on Information Quality'.

Ge, M. & Helfert, M. (2007), A review of information quality research — develop a research agenda, *in* 'Proceedings of the 12th International Conference on Information Quality'.

Hargis, G., Carey, M., Hernandez, A. K., Hughes, P., Longo, D. & Rouiller, S. (2009), *Developing quality technical information?A handbook for writers and editors*, Upper Saddle River, NJ: Pearson Education.

Huang, K.-T., Wang, Y. R. & Lee, W. Y. (1998), *Quality Information and Knowledge*, Prentice Hall, Upper Saddle River, NJ.

Juran, J. (1998), *Juran's Quality Control Handbook*, 5th edn, McGraw-Hill.

Klein, B. D. (2001), 'user perceptions of data quality: Internet and traditional text sources', *Journal of Computer Information Systems* **41**(4), 5–15.

Knight, S. A. & Burn, J. (2005), 'Developing a framework for assessing information quality on the World Wide Web', *Informing Science* **8**, 159–172.

Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M. & Jurafsky, D. (2011), Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task, *in* 'Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task', CONLL Shared Task '11, pp. 28–34.

Lee, Y. W., Strong, D. M., Kahn, B. K. & Wang, R. Y. (2002), 'Aimq: a methodology for information quality assessment.', *Information & Management* **40**(2), 133–146.

Lethbridge, T. C., Tichelaar, S. & Ploedereder, E. (2004), The dagstuhl middle metamodel, *in* 'Proceedings of the International Workshop on Meta-Models and Schemas for Reverse Engineering (ateM 2003)', Vol. 94, pp. 7–18.

Naumann, F. (2002), *Quality-driven query answering for integrated information systems*, Springer-Verlag, Berlin, Heidelberg.

Smart, K., Madrigal, J. & Seawright, K. (1996), 'The effect of documentation on customer perception of product quality', *IEEE Transactions on Professional Communication* **39**(3), 157–162.

Strein, D., Lincke, R., Lundberg, J. & Löwe, W. (2007), 'An extensible meta-model for program analysis', *IEEE Trans. Software Eng.* **33**(9), 592–607.

Stvilia, B., Gasser, L., Twidale, M. B. & Smith, L. C. (2007), 'A framework for information quality assessment', *JASIST* **58**(12), 1720–1733.

Wang, R. Y. & Strong, D. M. (1996), 'Beyond accuracy: what data quality means to data consumers', *J. Manage. Inf. Syst.* **12**(4), 5–33.

Wingkvist, A., Ericsson, M. & Löwe, W. (2011), 'Making sense of technical information quality — a software-based approach', *Journal of Software Technology* **14**(3), 12–18.

Wingkvist, A., Ericsson, M., Löwe, W. & Lincke, R. (2010*a*), 'Information quality testing', *Lecture Notes in Information Processing (LNBIP)* **64**, 14–26.

Wingkvist, A., Ericsson, M., Löwe, W. & Lincke, R. (2010*b*), A metrics-based approach to technical documentation quality, *in* 'Proceedings of the 7th International Conference on the Quality of Information and Communications Technology', pp. 476–481.