Towards a Quality Model for Semantic Technologies

Filip Radulovic[1], Raúl García-Castro[2]

Ontology Engineering Group,
[1]Departamento de Inteligencia Artificial
[2]Departamento de Lenguajes y Sistemas Informáticos e Ingeniería Software
Facultad de Informática, Universidad Politécnica de Madrid, Spain
fradulovic@delicias.dia.fi.upm.es, rgarcia@fi.upm.es

Abstract. Semantic technologies have become widely adopted in the last years. However, in order to correctly evaluate them we need to ground evaluations in a common quality model. This paper presents some first steps towards the definition of such quality model for semantic technologies. First, some well-known software quality models are described, together with methods for extending them. Afterwards, a quality model for semantic technologies is defined by extending the ISO 9126 quality model.

1 Introduction

Software quality is acknowledged as a main need across domains (e.g., security, health) and technologies (e.g., operating systems, databases) and, in order to obtain high-quality software products, during the software development process the specification and evaluation of quality is of crucial importance [**?**].

One important component in software evaluation are software quality models, since they provide the basis for software evaluation and give a better insight of the software characteristics that influence its quality. Furthermore, quality models also ensure a consistent terminology for software product quality and provide guidance for its measurement.

In recent years, semantic technologies have started to gain importance and, as the field becomes more and more popular, the number of these technologies is increasing exponentially. Just as with any other software product, the quality of semantic technologies is an important concern. Multiple evaluations of semantic technologies have been performed, from general evaluation frameworks [**?**] to tool-specific evaluations [**?**,**?**] and even characteristic-specific evaluations [**?**]. However, the problem is that there is no consistent terminology for describing the quality of semantic technologies and it is difficult to compare them because of differences in the meaning of the evaluation characteristics used. Also, existing software quality models do not provide specification of quality characteristics that are specific to semantic technologies.

This paper describes a first step to build a quality model for semantic technologies, to provide a consistent framework to support the evaluation of such

technologies, that extends the ISO 9126 quality model. To build this quality model, we have used a bottom-up approach, starting from real semantic technology evaluations and extracting from them the elements of the model.

This paper is structured as follows. Section 2 gives an overview of existing software quality models. Section 3 describes already defined methods for extending software quality models. Sections 4, 5 and 6 describe how we have defined the quality model . Finally, Section 7 draws some conclusions and includes ideas for future work.

2 Review of Software Quality Models

Quality in general is a complex and multifaceted concept that can be described from different approaches depending on whether the focus is in the concept of quality, the product, the user of the product, how the product was manufactured, or the value the product provides [**?**].

The way we define software quality depends on the approach that we take [**?**]; software quality means different things to different people and therefore defining and measuring quality will depend on the viewpoint. Similarly, choosing one software quality model or another will also depend on the intended users and uses of such model in concrete evaluations.

Next, this section describes some well-known software quality models and identifies their elements.

2.1 McCall's Model

McCall's software quality model is represented as a hierarchy of *factors, criteria* and *metrics* [**?**]. Factors are at the highest level in the hierarchy and represent the characteristics of the software product. Criteria are the middle layer and are considered to be the attributes of the factors, so that for every factor a set of criteria is defined. At the bottom level, metrics provide measures for software attributes.

McCall's model predefines a set of eleven software quality factors that are classified according to the product life cycle in three different groups:

- Product transition: portability, reusability, and interoperability
- Product revision: maintainability, flexibility, and testability
- Product operations: correctness, reliability, efficiency, integrity, and usability

McCall's quality model also gives the relationships between quality factors and metrics in the form of linear equations based on regression analyses. This is considered one of the major contributions of this model; however, the omission of the functionality aspect is regarded as the main lack [**?**].

2.2 Boehm's Model

Like McCall's model, Boehm's one has a hierarchical structure. It consists of twenty four quality characteristics divided into three levels [**?**]. It also gives a set of metrics and, while McCall's model is more related to the product view, Boehm's model includes users' needs.

2.3 ISO 9126's Model

The International Organization for Standardization (ISO) identified the need for a unique and complete software quality standard and, therefore, produced the ISO 9126 standard for software quality [**?**].

The ISO 9126 standard defines three types of quality: internal quality, external quality, and quality in use.

Six main software quality characteristics for external and internal quality are specified: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability, which are further decomposed into sub-characteristics (see Fig.1) that are manifested externally when the software is used, and are the result of internal software attributes [**?**]. The standard also provides the internal and external measures for sub-characteristics (see Table 1 for an example for Accuracy and Fault tolerance).

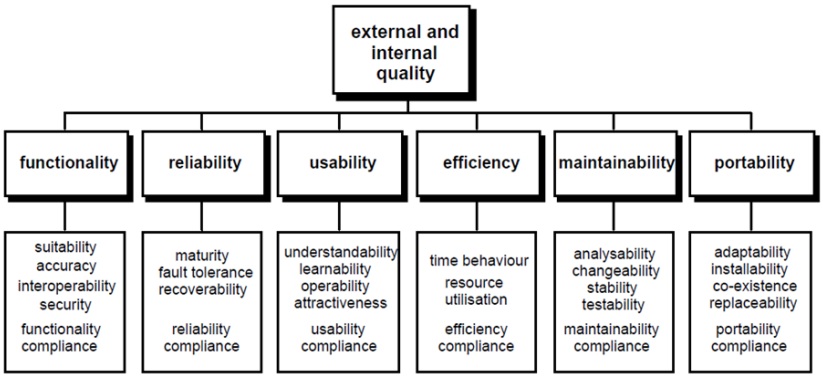

Fig. 1: ISO 9126 internal and external quality characteristics and sub-characteristics.

Regarding quality in use, the model proposes four characteristics: Effectiveness, Productivity, Safety, and Satisfaction.

The ISO 9126 standard gives the complete view of software quality with evaluation criteria and precise definitions for all software characteristics and sub-characteristics. Some authors also suggests that according to the nature of the product itself some new sub-characteristics can be added, the definitions of

existing ones can be changed, or some sub-characteristics can be eliminated from the model [**?**].

Table 1: ISO 9126 internal and external measures for Accuracy and Fault tolerance.

Quality Characteristics	**Quality Sub-Characteristics**	**External Measures**	**Internal Measures**
Functionality	Accuracy	Computational accuracy	Computational accuracy
		Precision	Precision
Reliability	Fault tolerance	Failure avoidance	Failure avoidance
		Incorrect operation avoidance	Failure avoidance
		Breakdown avoidance	

2.4 SQuaRE's Model

Although the ISO 9126 standard has been accepted and used successfully, some problems and issues for its further use and improvement have been identified. They eventually arise mainly because of advances in technologies and changes of users needs. As pointed out by Azuma [**?**], the main problems were due to issues on metrics and lack of a quality requirement standard.

In order to address those issues, the existing standard is being redesigned and has been named SQuaRE. By the time of writing this paper, the parts of the SQuaRE standard related to the quality model and evaluations are still under development (ISO 25010, ISO 25040) and their final versions will be published in 2011.

As a summary of this section, Table 2 presents the elements of the described quality models. In our work, we have decided to adopt terminology from ISO 9126 standard.

Table 2: Elements of the described quality models.

Structure/Model	**McCall**	**Boehm**	**ISO 9126**
First level	Factor	High level characteristic	Characteristic
Second level	Criteria	Primitive characteristic	Sub-characteristic
Third level	Metrics	Metrics	Measures
Relationships between entities	Factor-Metric	/	Measure-Measure

3 Approaches for Extending Software Quality Models

One existing software quality model (such as the ISO 9126) can be a good starting point for building a suitable quality model for a specific software product or domain. Its already defined characteristics and sub-characteristics should be considered, and some of them could be excluded or defined in a different manner, according to the nature of the domain. Also, the model can be extended by introducing new sub-characteristics if needed.

Software quality model extensions can be performed following two main approaches [**?**]:

- A **top-down** approach that starts from the quality characteristics and continues towards the quality measures.
- A **bottom-up** approach that starts from the quality measures and defines the quality sub-characteristics that are related to each specific measure.

In their work, Franch and Carvallo proposed a method based on a top-down approach for customizing the ISO 9126 quality model [**?**]. After defining and analyzing the domain, the method proposes six steps:

1. *Determining quality sub-characteristics.* In the first step, according to the domain, some new quality sub-characteristics are added while other are excluded or their definitions are changed.
2. *Defining a hierarchy of sub-characteristics.* If it is needed, sub-characteristics are further decomposed according to some criteria.
3. *Decomposing sub-characteristics into attributes.* In this step abstract sub-characteristics are decomposed into more concrete concepts which refer to some particular software attribute (i.e., observable feature).
4. *Decomposing derived attributes into basic ones.* Attributes that are not directly measurable are further decomposed into basic ones.
5. *Stating relationships between quality entities.* Relationships between quality entities are explicitly defined. Three possible types of relationships are identified:
   - *Collaboration.* When increasing the value of one entity implies increasing of the value of another entity.
   - *Damage.* When increasing the value of one entity implies decreasing the value of another entity.
   - *Dependency.* When some values of one entity require that another entity fulfills some conditions.
6. *Determining metrics for attributes.* To be able to compare and evaluate quality, it is necessary to define metrics for all attributes in the model.

In building their quality model for B2B applications, Behkamal et al. proposed a method to customize the ISO 9126 quality model in five steps [**?**]. The main difference with the previous method is that in Behkamal's approach the quality characteristics are ranked by experts; the experts should provide weights for all quality characteristics and sub-characteristics, and these weights are later

used to establish their importance. Besides, Behkamal's approach does not contemplate defining relationships between quality entities.

These previous approaches follow the top-down approach, and we have not found any example of a bottom-up approach in the literature. Because of this, and because of the existence of plenty of evaluations in our domain (i.e., semantic technologies), we have decided to follow a bottom-up approach for extending the ISO 9126 quality model to cover semantic technologies. Evaluation results are used as the starting point, from which the quality measures, sub-characteristics and characteristics are specified.

Some authors have proposed software quality models for various types of applications: B2B [**?**], mail servers [**?**], web-based applications [**?**], e-learning systems [**?**], model-driven web engineering methodologies [**?**], and ERP systems [**?**]. All those authors have used the ISO 9126 standard as the basis software quality model, and have extended it to fit their particular domain.

Furthermore, some authors proposed introducing quality models for various steps of the product development, like in the case of web engineering [**?**]. The authors analyze the influence of every step in web engineering process on a final product, and suggest that evaluation process should be performed after each step, with respect to quality models defined for each of them.

Since ISO 9126 is a widely adopted and used standard, we have also adopted it for constructing the quality model for semantic technologies. For building such quality model, we have used a bottom-up approach.

The following sections describe in detail how we have extracted the quality model for semantic technologies from existing evaluations.

4 Defining Quality Measures

The starting point for defining software quality measures has been the set of evaluation results obtained in the SEALS European project[1] , which provides evaluation results for different types of semantic technologies (ontology engineering tools [**?**], reasoning systems [**?**], ontology matching tools [**?**], semantic search tools [**?**], and semantic web service tools [**?**]).

Based on the evaluation results for a particular software product and the analysis of these results, a set of quality measures was defined. Then, some primitive measures were combined to obtain derived ones. For any derived quality measure defined,we specified the function (or set of functions) that allows obtaining such derived measure from the primitive ones.

Different evaluation scenarios were defined for each type of technology and in each of them different types of test data were used as input. Evaluation raw results represent the data obtained as the output of the evaluation when using some test data, and they are considered to be primitive measures. These test data and raw results provide us with enough information for defining the hierarchy of software quality measures.

[1] http://www.seals-project.eu

For instance, for evaluating the conformance of ontology engineering tools, the experiment consisted in importing the file containing an ontology (O_i) into the tool and then exporting the imported ontology to another file (O_i^{II}) (Fig. 2 shows the steps of the conformance evaluation).

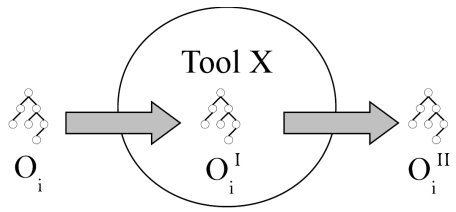

Fig. 2: Steps of a conformance test execution.

In this evaluation test suites with ontologies modeled in different ontology languages were used. Each test in a test suite contains:

- *Origin ontology*. The ontology to be used as input.

The raw results of one test execution are:

- *Final ontology*. The ontology that is produced by the tool when importing and exporting the origin ontology.
- *Execution Problem*. Whether there were any execution problems in the tool when importing and exporting the origin ontology. Possible values are *true*, and *false*.

Based on the test data and the raw results of one test execution, the following interpretations and their functions were made:

- *Information added*. The information added to the origin ontology after importing and exporting it.

$$\text{final ontology} - \text{origin ontology}$$

- *Information lost*. The information lost from the origin ontology after importing and exporting it.

$$\text{origin ontology} - \text{final ontology}$$

- *Structurally equivalent*. Whether the origin ontology and the final one are structurally equivalent. Possible values are *true*, and *false*.

$$(\text{information added} = \text{null}) \wedge (\text{information lost} = \text{null})$$

- *Semantically equivalent*. Whether the origin ontology and the final one are semantically equivalent. Possible values are *true*, and *false*.

$$\text{final ontology} \equiv \text{origin ontology}$$

– *Conformance*. Whether the ontology has been imported and exported correctly with no addition or loss of information. Possible values are *true*, and *false*.

$$\text{semantically equivalent} \wedge \neg(\text{execution problem})$$

Those interpretations are actually derived measures obtained from one particular test, and are further combined in order to obtain measures for the whole test suite. From the derived measures in the conformance scenario, the following test suite measures were obtained:

– *Ontology language component support*. Whether the tool fully supports an ontology language component.

$$\frac{\#\text{ tests that contain the component where conformance} = \text{true}}{\#\text{ tests that contain the component}} = 1$$

– *Ontology language component coverage*. The ratio of ontology components that are shared by a tool internal model and an ontology language model.

$$\frac{\#\text{ components in the ontology language where component support} = \text{true}}{\#\text{ components in the ontology language}} \times 100$$

– *Ontology information persistence*. The ratio of information additions or loss when importing and exporting ontologies.

$$\frac{\#\text{ tests where information added} \neq \text{null or information lost} \neq \text{null}}{\#\text{ tests}} \times 100$$

– *Execution errors*. The ratio of tool execution errors when importing and exporting ontologies.

$$\frac{\#\text{ tests where execution problem} = \text{true}}{\#\text{ tests}} \times 100$$

Similarly to the example of the conformance evaluation for ontology engineering tools presented above, we have defined measures for the other types of tools. Table 3 summarizes the obtained results.

Table 3: Total number of measures obtained for semantic technologies.

Tool/Measures	**Raw results**	**Interpretations**	**Measures**
Ontology engineering tools	7	20	8
Ontology matching tools	1	3	4
Reasoning systems	7	0	7
Semantic search tools	13	16	18
Semantic web service tools	2	3	3

5 Defining Software Quality Sub-characteristics

Every software product from a particular domain has some sub-characteristics that are different from other software systems and those sub-characteristics, together with more generic ones, should be identified and precisely defined. Every quality measure provides some information about one or several software sub-characteristics; therefore, based on the software quality measures that we defined, we specified a set of software quality sub-characteristics. In some cases, a quality sub-characteristic does not have only one measure that determines it, but a set of measures. Finally, some quality sub-characteristics were combined into more general ones.

In the example of the conformance scenario for ontology engineering tools, based on the measures and analysis presented in previous section, we have identified two quality sub-characteristics that are specific to this type of software, which are the following:

- *Ontology language model conformance*. The degree to which the knowledge representation model of the software product adheres to the knowledge representation model of an ontology language. It can be measured using two different measures, which are obtained in the conformance evaluation:
  - *Ontology language component coverage*
  - *Ontology language component support*
- *Ontology I/E accuracy*. The accuracy of the process of importing and exporting ontologies. It can be measured using:
  - *Ontology information persistence*

Furthermore, we have identified one quality sub-characteristic which is general and can be used for different kinds of software:

- *Robustness*. The ability of the software product to function correctly in the presence of invalid inputs or stressful environmental conditions. It can be measured using:
  - *Execution errors*

Figure 3 presents the raw results, interpretations, quality measures and quality characteristics of the conformance evaluation for ontology engineering tools.

In total, we have identified twelve semantic quality sub-characteristics:

- *Ontology Language Model Conformance*. The degree to which the knowledge representation model of the software product adheres to the knowledge representation model of an ontology language.
- *Ontology Language Interoperability*. The degree to which the software product can interchange ontologies and use the ontologies that have been interchanged.
- *Reasoning Accuracy*. The accuracy of the reasoning process.
- *Ontology Alignment Accuracy*. The accuracy of the matching process.
- *Semantic Search Accuracy*. The accuracy of the semantic search process.

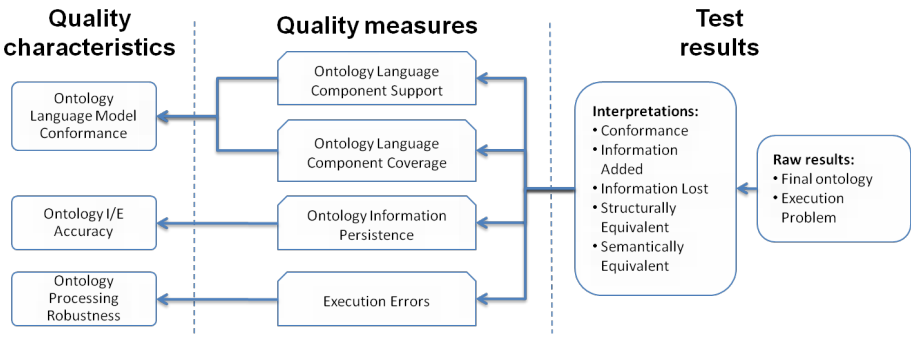

Fig. 3: Entities in the conformance scenario for ontology engineering tools.

- *Semantic Web Service Discovery Accuracy*. The accuracy of the process of finding services that can be used to fulfil a given requirement from the service requester.
- *Ontology I/E Accuracy*. The accuracy of the process of importing and exporting ontologies by the tool.
- *Ontology Interchange Accuracy*. The accuracy of the interchange of ontologies between tools.
- *Query Language Suitability*. The ability of the tool to provide appropriate queries for a given questions.
- *Ontology Processing Time Behaviour*. The capability of the software product to provide appropriate response and processing times when working with ontologies.
- *Reasoning Time Behaviour*. The capability of the software product to provide appropriate response and processing times when performing reasoning tasks.
- *Semantic Search Time Behaviour*. The capability of the software product to provide appropriate response and processing times when performing the search task.

6 Aligning Quality Sub-characteristics to a Quality Model

In the previous step we have identified a set of quality sub-characteristics that are specific for semantic technologies. After that, the alignment with the ISO 9126 quality model was established; i.e., all the identified sub-characteristics were properly assigned to those already specified in the ISO 9126 quality model.

For instance, *Ontology language model conformance* is defined as a sub-characteristic of *Functionality compliance* (the capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality).

Fig. 4 shows the proposed quality model for semantic technologies, while Fig. 5 shows quality in use.

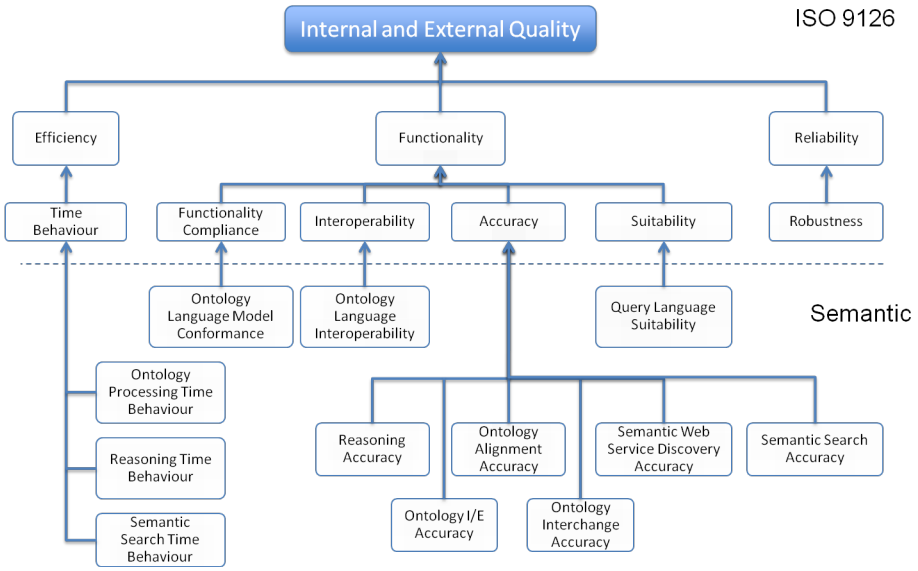

Fig. 4: External and internal quality characteristics for semantic technologies.

Fig. 5: Quality in use quality characteristics for semantic technologies.

7 Conclusions and future work

This paper presents a first step towards a quality model for semantic technologies, which extends the ISO 9126 software quality model. Such quality model can provide a framework for the evaluation and comparison of semantic technologies.

For building the quality model, we have used a bottom-up approach. It starts from the evaluations that have already been performed and continues towards

defining quality measures. When a set of quality measures is defined, quality sub-characteristics are specified together with their hierarchy. At the end, sub-characteristics are aligned to an existing quality model.

The model presented in this paper is not complete. It only includes quality entities that are based on some existing evaluations. In the future, the model will be completed either by extending the evaluations, or by applying a top-down approach and specifying additional quality entities that are currently not defined in the model.

Some authors include in their software quality models relationships between quality entities [**?**,**?**]. The inclusion of these entities into our model will be addressed in a future iteration.

Furthermore, the ISO 9126 standard is to be replaced by the SQuaRE one, and when the SQuaRE software quality model becomes available, the proposed quality model for semantic technologies should be adapted to it.

One future use of the quality model presented in this paper, based on the evaluation results that are being obtained in the SEALS project, is to build a recommendation system for semantic technologies that will allow extracting semantic technology roadmaps. This will provide users with guidance and recommendation of the semantic technologies that better suite their needs.

Acknowledgments

This work is supported by the SEALS European project (FP7-238975) and by the EspOnt project (CCG10-UPM/TIC-5794) co-funded by the Universidad Politécnica de Madrid and the Comunidad de Madrid.