D2.4 – Metamodels for OSS Project Metrics

Version 1.0
30 October 2014
Final

Public Distribution

University of L’Aquila

Project Partners: Centrum Wiskunde & Informatica, SOFTEAM, Tecnalia Research and Innovation, The Open Group, University of L’Aquila, UNINOVA, University of Manchester, University of York, Unparallel Innovation

Every effort has been made to ensure that all statements and information contained herein are accurate, however the OSSMETER Project Partners accept no liability for any error or omission in the same.

© 2014 Copyright in this document remains vested in the OSSMETER Project Partners.
### Project Partner Contact Information

<table>
<thead>
<tr>
<th>Centrum Wiskunde &amp; Informatica</th>
<th>SOFTEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul Klint</td>
<td>Alessandra Bagnato</td>
</tr>
<tr>
<td>Science Park 123 1098 XG Amsterdam, Netherlands</td>
<td>Avenue Victor Hugo 21</td>
</tr>
<tr>
<td>Tel: +31 20 592 4126</td>
<td>75016 Paris, France</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:paul.klint@cwi.nl">paul.klint@cwi.nl</a></td>
<td>Tel: +33 1 30 12 16 60</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:alessandra.bagnato@softeam.fr">alessandra.bagnato@softeam.fr</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tecnalia Research and Innovation</th>
<th>The Open Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jason Mansell</td>
<td>Scott Hansen</td>
</tr>
<tr>
<td>Parque Tecnologico de Bizkaia 202</td>
<td>Avenue du Parc de Woluwe 56</td>
</tr>
<tr>
<td>48170 Zamudio, Spain</td>
<td>1160 Brussels, Belgium</td>
</tr>
<tr>
<td>Tel: +34 946 440 400</td>
<td>Tel: +32 2 675 1136</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:jason.mansell@tecnalia.com">jason.mansell@tecnalia.com</a></td>
<td>E-mail: <a href="mailto:s.hansen@opengroup.org">s.hansen@opengroup.org</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University of L’Aquila</th>
<th>UNINOVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Davide Di Ruscio</td>
<td>Pedro Maló</td>
</tr>
<tr>
<td>Piazza Vincenzo Rivera 1</td>
<td>Campus da FCT/UNL, Monte de Caparica</td>
</tr>
<tr>
<td>67100 L’Aquila, Italy</td>
<td>2829-516 Caparica, Portugal</td>
</tr>
<tr>
<td>Tel: +39 0862 433735</td>
<td>Tel: +351 212 947883</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:davide.diruscio@univaq.it">davide.diruscio@univaq.it</a></td>
<td>E-mail: <a href="mailto:pmm@uninova.pt">pmm@uninova.pt</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University of Manchester</th>
<th>University of York</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophia Ananiadou</td>
<td>Dimitris Kolovos</td>
</tr>
<tr>
<td>Oxford Road</td>
<td>Deramore Lane</td>
</tr>
<tr>
<td>Manchester M13 9PL, United Kingdom</td>
<td>York YO10 5GH, United Kingdom</td>
</tr>
<tr>
<td>Tel: +44 161 3063098</td>
<td>Tel: +44 1904 325167</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:sophia.ananiadou@manchester.ac.uk">sophia.ananiadou@manchester.ac.uk</a></td>
<td>E-mail: <a href="mailto:dimitris.kolovos@york.ac.uk">dimitris.kolovos@york.ac.uk</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unparallel Innovation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuno Santana</td>
<td></td>
</tr>
<tr>
<td>Rua das Lendas Algarvias, Lote 123</td>
<td></td>
</tr>
<tr>
<td>8500-794 Portimão, Portugal</td>
<td></td>
</tr>
<tr>
<td>Tel: +351 282 485052</td>
<td></td>
</tr>
<tr>
<td>E-mail: <a href="mailto:nuno.santana@unparallel.pt">nuno.santana@unparallel.pt</a></td>
<td></td>
</tr>
</tbody>
</table>
Contents

1 Introduction .................................................................................................................. 2
   1.1 Purpose of the Deliverable ................................................................................... 2
   1.2 Structure of the deliverable ................................................................................. 2
   1.3 Relationship to other OSSMETER Deliverables .............................................. 3
   1.4 Contributors ......................................................................................................... 3

2 Software quality models ............................................................................................... 4
   2.1 Traditional software quality models .................................................................... 4
       2.1.1 McCall's Quality Model ................................................................................ 4
       2.1.2 Boehm's Quality Model .............................................................................. 6
       2.1.3 Dromey's Quality Model ............................................................................ 7
       2.1.4 ISO 9126 .................................................................................................... 9
       2.1.5 FURPS Model ........................................................................................... 12
       2.1.6 Summary .................................................................................................... 12
   2.2 OSS-specific quality models ............................................................................... 13
       2.2.1 OSMM: Capgemini Open Source Maturity Model .................................... 13
       2.2.2 OpenBRR: Open Business Readiness Rating ............................................ 15
       2.2.3 QSOS: Atos Qualification and Selection of Open Source Software .......... 18
       2.2.4 OpenBQBR: Open Business Quality Rating ............................................. 18
       2.2.5 OMM: QualiPSo OpenSource Maturity Model ......................................... 19
       2.2.6 SQO-OSS: Software Quality Observatory for Open Source Software ...... 20
       2.2.7 QualOSS .................................................................................................... 21
       2.2.8 Squale Software Quality Model ................................................................ 23
       2.2.9 Summary .................................................................................................... 24

3 OSSMETER quality model ............................................................................................ 25
   3.1 OSSMETER quality metamodel ......................................................................... 25
   3.2 The default OSSMETER quality model ............................................................. 26
       3.2.1 Quality attributes ....................................................................................... 26
       3.2.2 Measuring quality attributes by means of factoids ..................................... 29

4 Conclusions .................................................................................................................. 39
# Document Control

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Document outline</td>
<td>5 August 2014</td>
</tr>
<tr>
<td>0.2</td>
<td>First draft</td>
<td>12 September 2014</td>
</tr>
<tr>
<td>0.7</td>
<td>First full draft</td>
<td>19 September 2014</td>
</tr>
<tr>
<td>0.8</td>
<td>Second draft</td>
<td>29 September 2014</td>
</tr>
<tr>
<td>0.9</td>
<td>For internal review</td>
<td>18 October 2014</td>
</tr>
<tr>
<td>1.0</td>
<td>QA review</td>
<td>30 October 2014</td>
</tr>
</tbody>
</table>
Executive Summary

Deciding whether an open source software (OSS) meets the required standards for adoption in terms of quality, maturity, activity of development and user support is not a straightforward process as it involves exploring various sources of information, including the project’s source code repositories, communication channels, and bug tracking systems. To deal with such problems, OSSMETER is developing an extensible, scalable, cloud-based platform to monitor, analyse, and present the wealth of data available in OSS projects in a systematic and automated manner.

The analysis and the comparison of open source software requires the definition of a number of attributes that have to be calculated and evaluated in order to support the ranking of the compared software systems and to take the final decision about which of them has to be adopted. In other words the adoption of a quality model is mandatory. A quality model is typically hierarchically organized since they decompose quality into a hierarchy of criteria and attributes leading to specific metrics at their lowest level.

In this document we present the OSSMETER quality metamodel and an instance of it. In particular, after an overview of already developed quality models (both generic and OSS specific), a quality metamodel is defined with the aim of collecting the characteristics that necessarily have to be considered when defining a quality model and the ways to evaluate it. Then the default OSSMETER quality model is presented as an instance of the OSSMETER quality metamodel. Such a model is used to customize the front-end of the application developed in WP5. For each quality aspect of the proposed quality model we specify the factoids (and consequently the metrics providers) developed in WP3 and WP4 that are used for the measurements. Users can extend and refine such model by using the concepts included in the OSSMETER metamodel and the means provided by the OSSMETER platform.
1 Introduction

Deciding whether an open source software (OSS) meets the required standards for adoption in terms of quality, maturity, activity of development and user support is not a straightforward process as it involves exploring various sources of information, including the project’s source code repositories, communication channels, and bug tracking systems. To deal with such problems, OSSMETER is developing an extensible, scalable, cloud-based platform to monitor, analyse, and present the wealth of data available in OSS projects in a systematic and automated manner.

The analysis and the comparison of open source software requires the definition of a number of attributes that have to be calculated and evaluated in order to support the ranking of the compared software systems and to take the final decision about which of them has to be adopted. In other words the adoption of a quality model is mandatory. A quality model is typically hierarchically organized since they decompose quality into a hierarchy of criteria and attributes leading to specific metrics at their lowest level.

1.1 Purpose of the Deliverable

This document presents the work that has been done with respect to the following task as presented in the OSSMETER Description of Work:

Task 2.4: OSS Project Metrics Metamodel Development. Develop an EMF/Ecore based metrics metamodel that will make OSS projects measurable by enabling the different analysis provided by WP3 and WP4. In particular, the output of the analysis techniques and tools developed in WP3 and WP4 will be represented as models conforming to the metamodel developed in this task. Such models will be used also by the OSSMETER platform developed in WP5 (e.g., for comparing selected projects with respect to calculated metrics and preferred criteria).

In this respect, this document presents the OSSMETER quality metamodel and an instance of it. In particular, after an overview of already developed quality models (both generic and OSS specific), a quality metamodel is defined with the aim of collecting the characteristics that necessarily have to be considered when defining a quality model and the ways to evaluate it. Then the default OSSMETER quality model is presented as an instance of the OSSMETER quality metamodel. Such a model is used to customize the front-end of the application developed in WP5. Moreover, the quality model permit to compose measured aspects allowing the user to get an early indication of the quality of the considered project. For each quality aspect of the proposed quality model we specify the factoids and consequently the metrics providers developed in WP3 and WP4 that are used for the measurements. Users can extend and refine such model by using the concepts included in the OSSMETER metamodel and the means provided by the OSSMETER platform.

1.2 Structure of the deliverable

The deliverable is structured as follows:
D2.4 – Metamodels for OSS Project Metrics

– in Section 2 we overview representative quality models that have been defined to assess the quality of software systems. We distinguish the quality models that have been defined for traditional software systems from those that are OSS specific;
– in Section 3 we present the OSSMETER quality model which is defined as an instance of the OSSMETER quality metamodel presented in the same section;
– in Section 4 we conclude this document.

1.3 Relationship to other OSSMETER Deliverables

This document presents the OSSMETER quality metamodel and an instance of it also called quality model hereafter. The quality model is used to support the evaluation and the comparison of open source projects by considering both their source code and the communities built around them. To this end, for each quality attribute, the quality model refers to factoids developed in D3.3 [2], D3.4 [3] and D4.6 [18]. A JSON representation of the quality model is used to configure the front-end of the application developed in WP5 and presented in D5.6 [20].

1.4 Contributors

The main contributor of this deliverable is University of L’Aquila. All project partners contributed to this document by providing system requirements, contents, as well as by providing feedback and suggestions for editing and refinements of this document.
2 Software quality models

When comparing software systems it is necessary to define a number of attributes that have to be calculated and evaluated in order to support the ranking of the compared software systems and to take the final decision about which of them has to be adopted. The identification of attributes to be considered for the evaluation represents a difficult task per se and it is related to the concept of software quality. Over the years many authors and researchers have provided answers to the question what is quality whose definition relies on two different paths [9]:

- Conformance to specification: quality is defined in terms of products and services whose measurable characteristics satisfy a fixed specification;
- Meeting customer needs: quality is identified independently of any measurable characteristics and is defined as the products or services capability to meet (explicit or not) customer expectations.

Software Quality Engineering [12] is a discipline that is concerned with improving the approach to software quality by defining appropriate quality models. A software quality model acts as a framework for the evaluation of attributes of an application that contribute to the software quality [22]. A common approach to formulating a model for software product quality is to first identify a small set of high-level quality attributes and then, in a top-down fashion decompose these attributes into sets of subordinate attributes. Over the last decades, the need for quality in software has increased. Several quality models have been proposed and they emphasize the need to have quality checks while developing a software system. However none of the proposed models has reached a significant acceptance.

In the next sections we make an overview of the most representative quality models that have been proposed by both academia and industry. We first present models that have been proposed to assess the quality of traditional software systems (Section 2.1). Then we focus on quality models that have specifically conceived to measure the quality of open source software systems (Section 2.2).

2.1 Traditional software quality models

In the following we outline the most widely accepted traditional software quality models that have been introduced by the software quality engineering area [14].

2.1.1 McCall’s Quality Model

In 1977 McCall [13] introduced the quality model whose quality attributes are shown in Fig.1. In particular, on the left-hand side of the figure there are quality aspects that are not directly measurable. Such quality aspects are evaluated by means of the measurable properties shown on the right-hand side of the figure and that are described in the following:

- Correctness: it represents at what extent a system functionality matches its specification.
• **Reliability**: it is the probability that a product, system or service will perform its intended function adequately for a specified period of time, operating in a defined operating environment without failure.
• **Efficiency**: it refers to the way system resources are used, e.g., processor time, and memory.
• **Integrity**: it represents the protection level of the considered system from unauthorized access.
• **Usability**: it represents the ease of using software.
• **Maintainability**: it is the effort required to locate and fix a fault in the program within its operating environment.
• **Testability**: it represents the ease with which a software system can be tested to ensure that it is error-free and meets its specification, i.e., validating the software requirements.
• **Flexibility**: it represents the ease of making changes required as dictated by business changes in the operating environment.
• **Portability**: it represents the effort required to transfer a system from one environment to another.
• **Re-usability**: it is the ease of reusing software in a different context.
• **Interoperability**: it refers to the effort required to couple the system to another system.

The quality attributes shown in Fig.1 are further organized in the quality model shown in Fig.2. In particular, the quality of a software product is assessed by considering:

• **Product Revision**: it identifies the quality factors that influence the ability to change the software product, i.e., maintainability, flexibility, and testability;
• **Product Transition**: it groups the quality factors that influence the ability to adapt the software to new environments and contexts, i.e., portability, re-usability, and interoperability;
• **Product Operations**: it identifies quality factors that influence the extent representative quality models that are specifically defined to assess the quality of OSS projects are summarized the software fulfils its specification, i.e., correctness, reliability, efficiency, integrity, and usability.

According to McCall, the grading scheme of the quality factors ranges from 0 (low) to 10 (high). However, many of the metrics defined by McCall et al. can be measured only subjectively and therefore it is difficult to use the McCall quality model to set precise and specific quality requirements [24].

### 2.1.2 Boehm’s Quality Model

Boehm’s quality model represents the second of the basic and founding predecessors of today’s quality models and it was proposed in 1978 [1]. Boehm’s model is similar to the McCall Quality Model in the sense that it also presents a hierarchical quality model structured around high-level, intermediate level, and primitive characteristics.

The **high-level characteristics** address three main questions that a buyer of software might have:

- **As-is utility**: How well (easily, reliably, efficiently) can I use the software as-is?
- **Maintainability**: How easy is the software to understand, modify and retest?
- **Portability**: Can I still use the software if I change my environment?

The **intermediate level characteristics** consist of the following 7 quality factors of software systems:

- **Portability**: it represents at what extent the software can be operated easily and well on computer configurations other than its current one;
• **Reliability**: it refers to the extent to which the software can be expected to perform its intended functions satisfactorily;
• **Efficiency**: it refers to the extent to which the software fulfils its purpose without waste of resources;
• **Usability**: it refers to the extent to which the software is reliable, efficient and human-engineered;
• **Testability**: it refers to the extent to which the software facilitates the establishment of verification criteria and supports evaluation of its performance;
• **Understandability**: it refers to the extent to which the software purpose is clear to the inspector.
• **Modifiability**: It refers to the extent to which the software facilitates the incorporation of changes, once the nature of the desired change has been determined.

The *primitive characteristics* provide the foundation for defining qualities metrics. However, the model provides only a diagram without any suggestion about measuring the quality characteristics. Figure 3 shows all the characteristics (hierarchically organized) outlined above.

### 2.1.3 Dromey’s Quality Model

In [6] Dromey proposes a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to apply for different systems. Dromey is focusing on the relationship between the quality attributes
and the sub-attributes, as well as attempting to connect software product properties with software quality attributes. In other words, each artifact produced in the software life-cycle can be associated with a quality evaluation model. Dromey gives the following examples of what he means by software components for each of the different models:

- Variables, functions, statements, etc. can be considered components of the implementation model;
- A requirement can be considered a component of the requirements model;
- A module can be considered a component of the design model;

According to Dromey, all these components have intrinsic properties that can be classified into four categories (see Fig. 4):

- **Correctness**: it evaluates if some basic principles are violated.
- **Internal**: it is a measure of how well a component has been deployed according to its intended use.
- **Contextual**: it deals with the external influences by and on the use of a component.
- **Descriptive**: it is a measure of the descriptiveness of a component (for example, does it have a meaningful name?)

Dromey focuses also on the relationship between the quality attributes and the sub-attributes, as well as attempts to connect software product properties with software quality attributes. In particular:

- Product properties that influence quality.
- High level quality attributes.
- Means of linking the product properties with the quality attributes.

Dromey’s Quality Model is further structured around a 5 step process:

1. Choose a set of high-level quality attributes necessary for the evaluation.
2. List components/modules in your system.

3. Identify quality-carrying properties for the components/modules (qualities of the component that have the most impact on the product properties from the list above).

4. Determine how each property effects the quality attributes.

5. Evaluate the model and identify weaknesses.

2.1.4 ISO 9126

With the main objective of dealing with the proliferation of quality models, in 1993 ISO made the attempt of specifying a quality model with the aim of having a single universal model. That would make easier to compare one product with another. Starting from the McCall and Boehm quality models, ISO released the ISO 9126: Software Product Evaluation: Quality Characteristics and Guidelines for their Use-standard [11].

An overview of the quality (sub-)factors identified by the ISO9126 Quality Model is shown in Fig. 6 and summarized in the following:

- **Functionality**: A set of attributes that relate to the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
Figure 6: ISO9126 Quality Factors and Sub-factors [14]

- **Suitability**: Attribute of software that relates to the presence and appropriateness of a set of functions for specified tasks.
- **Accuracy**: Attributes of software that bear on the provision of right or agreed results or effects.
- **Security**: Attributes of software that relate to its ability to prevent unauthorized access, whether accidental or deliberate, to programs and data.
- **Interoperability**: Attributes of software that relate to its ability to interact with specified systems.
- **Compliance**: Attributes of software that make the software adhere to related standards, conventions, regulations in laws, and other prescriptions.

**Reliability**: A set of attributes that relate to the capability of software to maintain its level of performance under stated conditions for a given period of time.

- **Maturity**: Attributes of software that relate to the frequency of failure by faults in the software.
- **Fault tolerance**: Attributes of software that relate to its ability to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
- **Recoverability**: Attributes of software that relate to the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.
- **Compliance**: See above.
• **Usability**: A set of attributes that relate to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
  
  – Understandability: Attributes of software that relate to the users’ effort for recognizing the logical concept and its applicability.
  – Learnability: Attributes of software that relate to the users’ effort for learning its application (for example, operation control, input, output).
  – Operability: Attributes of software that relate to the users’ effort for operation and operation control.
  – Attractiveness: Attributes of software that relate to user interface aesthetics.
  – Compliance: Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions.

• **Efficiency**: A set of attributes that relate to the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
  
  – Time behaviour: Attributes of software that relate to response and processing times and on throughput rates in performing its function.
  – Resource behaviour: Attributes of software that relate to the amount of resources used and the duration of such use in performing its function.
  – Compliance: See above.

• **Maintainability**: A set of attributes that relate to the effort needed to make specified modifications.
  
  – Analyzability: Attributes of software that relate to the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
  – Changeability: Attributes of software that relate to the effort needed for modification, fault removal or for environmental change.
  – Stability: Attributes of software that relate to the risk of unexpected effect of modifications.
  – Testability: Attributes of software that relate to the effort needed for validating the modified software.
  – Compliance: See above.

• **Portability**: A set of attributes that relate to the ability of software to be transferred from one environment to another.
  
  – Adaptability: Attributes of software that relate to on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered.
  – Installability: Attributes of software that relate to the effort needed to install the software in a specified environment.
  – Conformance: Attributes of software that make the software adhere to standards or conventions relating to portability.
  – Replaceability: Attributes of software that relate to the opportunity and effort of using it in the place of specified other software in the environment of that software.
2.1.5 FURPS Model

The FURPS model was originally presented in 1987 by by Robert Grady et al. [8] and the quality characteristics that are taken into account are functionality, usability, reliability, performance and supportability as explained in the following:

- Functionality: it includes feature sets, capabilities and security;
- Usability: it includes human factors, consistency in the user interface, online and context-sensitive help, wizards, user documentation, and training materials;
- Reliability: it includes frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failure (MTBF);
- Performance: it prescribes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage;
- Supportability: it includes testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability/internationalization.

2.1.6 Summary

The quality characteristics that are considered by the quality models summarized above are all shown in Table 1.

<table>
<thead>
<tr>
<th>Factors/Attributes/Characteristics</th>
<th>McCall</th>
<th>Boehm</th>
<th>Dromey</th>
<th>ISO 9126</th>
<th>FURPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintainability</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Flexibility</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reusability</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human Engineering</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understandability</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modifiability</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supportability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

Table 1: Summary of the traditional software quality models

Due to the nature of OSS development and use, the aforementioned quality models might not be sufficient. OSS project characteristics include shared artefact repositories, big and distributed communities, asynchronous distributed development, and lack of formal support. Such characteristics are...
not taken into account by traditional quality models. To address these limitations, OSS-specific quality models have been proposed in the literature and the most representative ones are summarised in the next section.

2.2 OSS-specific quality models

Open source quality models emerged due to the inability of traditional quality models to measure unique features (such as community) of open source software. In this section representative quality models that are specifically defined to assess the quality of OSS projects are summarized.

2.2.1 OSMM: Capgemini Open Source Maturity Model

The OSMM model \[7\] assumes that the quality of an OSS project is directly related to its maturity, and to the environment in which it has to be adopted. To this end OSMM introduces two sets of indicators, namely product and application that are summarized in the following.

Product indicators are used to assess the maturity of Open Source products. Product indicators receive a score valued between one (poor) and five (excellent). All the scores are summed to produce a product score. The resulting value presents a measure about the general maturity of the measured product. Product indicators are organized in four categories as summarized in the following:

Product: it focuses on the internals of the product, i.e., things like the development and stability of the developer group or the purpose of the product. The indicators of the group are:

- **Age**: It represents the age of the product, the longer a product remains under active development, the smaller the chance becomes that the developers suddenly stop.
- **Licensing**: It represents the legal license used to release the product. In other words, this indicator gives information about they in which the intended users are approached.
- **Human hierarchies**: it refers to the way the product is managed ranging from situations with one leader controlling all the aspects of the product, to active members hierarchically organized each of them devoted to the management of separate areas of attention.
- **Selling points**: it refers to the overall maturity of the product and consequently to the interest that it can attract from the market.
- **Developer community**: it refers to the way people involved in the development of the product are organized and work together. Group organisation is one of the driving forces behind an effective community.

Integration: it measures the possibility to link the product to other products or infrastructure, and the modularity level of the considered product.

- **Modularity**: It refers to at what extent the product’s functionality can be split in modules that can be separately adopted/exploited.
- **Collaboration with other products**: it refers to at what extent the product can be integrated and put in synergy with other products.

Use: it refers to the way users are supported in everyday use of the product.
Standards: it refers to the extent to which the product uses and exploits open standards. For Open Source products adhering to widespread and accepted standards is extremely important. By only supporting standards that are common in just a couple of environments can adversely influence the acceptance of a product.

Support: it refers to the kind of support provided to the users of the product. For instance, for some products the is obtained by mailing the single developer. Others maintain a discussion group (or even groups), but only a couple of regulars respond to request. A few maintain very active discussion groups in which a large number of members will offer support. Some products will even guarantee support if you pay them. The manner in which support is given or offered says a lot about the way the development group takes its users seriously.

Acceptance: it contains indicators related to the way the product is available to the user community and permits to assess the product’s ability to grow and to penetrate the market.

Ease of deployment: it is related to the availability of documents, documentations, HOWTO’s, or training courses (even provided by independent parties) that permit new users to adopt the product.

User community: it is related to the liveness of the community of users that are in interested or are adopting the product. When an Open Source product is well received it is common to witness an increasing numbers of user requests, suggestions and problem reports. The discussion group quickly fills up with large numbers of messages, so the developers must expand and start to manage this huge flow of feedback.

Market penetration: it refers to the acceptance of the product. In fact, the number of installation of the considered product gives an indication about the importance of the product within the intended users.

Application indicators are related to the context in which the product being evaluated has to be adopted. The rational of application indicators is that a given open source product cannot be introduced into an existing working environment based only on a measurement of it strengths and weaknesses. Such indicators are evaluated by Capgemini consultants and users. Like in the case of product indicators, the scores of application indicators range from one (poor) to five (excellent). The application indicators defined by OSMM are the following:

Usability: it refers to the experience of the intended user audience.

Interfacing: it is related to the extent the product fit into the organization by considering e.g., the standards that are used and their compatibility with those used in the organisation.

Performance: it refer to the expected load and processing capability of the product being adopted has to meet.

Reliability: it refers to the level of availability that the product should deliver. What level of availability should the product deliver?

Security: it refers to the security levels that are required and the restrictions that are imposed onto the product to work in the considered environment.

Proven technology: it states if the product being evaluated relies on mature technologies that have been proven in daily production or not.
Vendor independence: it refers to the level of commitment between the supplier of the product and the user. This is a relevant indicator since expresses the vendor lock-in level that might affect in the future migrations towards other products.

Platform independence: it states if the product is available for particular ICT environments, or if it can work on any platform.

Support: it refers to the availability (and to the kind) of user support.

Reporting: it refers to the kind of reporting facilities that are required by the considered working environment.

Administration: it refers to the possibility to use existing maintenance tools, or if other operational management tools are required.

Training: it refers to the training sessions and facilities that are required to start profitably using the product.

Staffing: it refers to people that will be employed to use the product being evaluated, i.e., if they will be bought, taught or hired experts.

The indicators previously summarized are used to support the evaluation process shown in Fig. 7. The main limitation of the OSMM is that it does not take into consideration the analysis of source code and the collection of the data required to apply the model has to be performed by humans.

2.2.2 OpenBRR: Open Business Readiness Rating

The OpenBRR [28] defines a model and a process for evaluating OSS with particular emphasis on attributes interesting to businesses. This method focuses on the constraints faced by firms, which want to integrate OSS into their business processes. Moreover, this model focuses on the Total Cost of Ownership (TCO) perceived for OSS. The assessment process proposed by OpenBRR relies on the definition of a reference application, for identifying desirable characteristics and their weights. These identified characteristics are then used to assess the quality of the evaluated application. The sponsors for this methodology are Carnegie Mellon West Center for Open Source Investigation, CodeZoo, SpikeSource and Intel.

The evaluation process proposed by OpenBRR consists of four phases: i) quick assessment to rule in or rule out software packages and create a shortlist of possible candidates, ii) ranking the importance of the application indicators.

---

2Total cost of ownership (TCO) is a financial estimate intended to help buyers and owners determine the direct and indirect costs of a product or system (http://en.wikipedia.org/wiki/Total_cost_of_ownership)
of categories or metrics, iii) data processing, and iv) translating the data into the Business Readiness Rating that can range from 1 (unacceptable) to 5 (excellent).

During the initial assessment phase a number of viable software are considered for the definition of the short list of projects to be further analyzed and compared. The list is defined by considering several viability indicators i.e., licensing, standard compliance, referenceable adopters, availability of support, implementation language(s), third party reviews, books, and review by industry analysts such as Gartner. To rank the products that have been selected in the previous step, metrics and categories are used for supporting the in-depth assessment phases. In particular, the following 12 assessment categories are considered:

- **Functionality**: it refers to the extent the software meets the average user’s requirements.
- **Usability**: it refers to the quality of the user interface, how easy the software for end-users is, to the effort required to install, configure, deploy, and maintain the software.
- **Quality**: it refers to the quality of the design, code, and tests of the considered software.
- **Security**: it refers to the extent to which software is able to handle security issues.
- **Performance**: it refers to how the software performs.
- **Scalability**: it refers to how the software scales to large environments.
- **Architecture**: it refers to the modularity of the software, if it is portable, extensible and even if it can be integrated with other components.
- **Support**: it refers to the kind of support that users adopting the software may benefit.
- **Documentation**: it refers to the availability of technical and user documentation, and even its quality.
- **Adoption**: it is a measure of how well the measured product is adopted the community of even by industry.
- **Community**: it refers to quality of the community supporting the product and contributing to its development and improvement.
- **Professionalism**: it refers to the development process adopted to implement the considered software, and to the professionalism of the whole project organization.

Given a project to be ranked, all the previous assessment categories are scored by considering and measuring corresponding metrics. Depending on the software’s functional orientation, the relevance of certain categories may not be significant enough. In this respect, the evaluation is performed by ranking both metrics and categories according to their importance, which is decided by users.

Once all the measurements within a category are performed, they are normalized (and weighted) with respect to a given scale. The suggested one ranges from 1 (unacceptable) to 5 (excellent).

According to the information available on the official link of the quality model[^1] for some reason the project has not created a thriving community. However, in order to move OpenBRR forward, the following actions have been considered peculiar to be undertaken:

- the creation of automated tools to extract project data from repositories;
- user-friendly mechanisms for entering and searching project reviews;
- identification of functional evaluation criteria for the software taxonomy;

[^1]: [http://openbrr.org/]
expanding the framework to facilitate comparison between FOSS and other software, and;
expanding project leadership.

It seems that the promoter of the project have worked on the two first points listed below even though no links are provided to download or even play with the developed tools. However, they claim that they will be updating the Web site with new content reflecting their plans to revitalize the community.
2.2.3 QSOS: Atos Qualification and Selection of Open Source Software

The QSOS model \[21\] defines a four-step process for assessing OSS quality. In particular,

1. similarly to OpenBRR, the first step of the evaluation process consists of defining a list of projects that seem to fit with the overall requirements defined by the product user of integrator,
2. by considering a list of evaluation criteria given by QSOS, the projects previously selected are evaluated by assigning an absolute score to each evaluation criterion,
3. the importance of each criterion can be adjusted by assigning a weight and threshold to each criterion according to the particular considered context,
4. the scores obtained in ii) are weighted based on iii), and finally
5. QSOS suggests trying the first 2-5 projects with the highest score.

Similarly to other quality models, the proposed criteria used to evaluate open source software are hierarchically organized and QSOS provides also a procedure to score each leaf criterion. Criteria are organized in two groups, namely generic and specific. The former includes criteria that apply to all software products, whereas the latter includes references to functionalities that are specific to the application domain of the software being evaluated, e.g., CMS, groupware, DBMS, etc. The general criteria of QSOS are shown in Fig. 8.

One aspect neglected by this model is the quality of the community around an OSS project. In fact, this model focuses on four main aspects of an OSS product, namely its intrinsic durability, the industrialization level of the solution, its technical adaptability to existing contexts, and the applied strategy in terms of licensing, sponsorships, etc.

Although in principle the method is effectively applicable to most OSS, the QSOS approach does not represent a relevant step forward with respect to other evaluation methods. Its main contribution is probably the explicitation of the set of characteristics that compose the Identity Card (IC) of products (reporting their general information, available services, functional and technical specifications, etc.) and the provision of a guideline for the consistent evaluation of these characteristics.

2.2.4 OpenBQBR: Open Business Quality Rating

OpenBPQR \[27\] is an extension and integration of OpenBRR and QSOS. The model focuses on four aspects of an OSS product, namely the functional adequacy to requirements, the availability of maintenance support, the quality in terms of absence of defects or time to fix, and cost of non OSS modules or necessary development tools.

In line with OpenBRR the evaluation process is composed of three phases, i.e., i) quick assessment filter, ii) data collection & processing, and iii) data translation.

It is important to note that differently from OpenBRR, the quick assessment filter step is performed after having weighted the evaluation characteristics in order to avoid data collection for irrelevant characteristics or with little importance. In the overall evaluation process, OpenBPQR consider a number of quality indicators that are organized into five different areas:
• **functional requirement analysis**, it is related to the functional characteristics of the product and they are considered in order to assess their adequacy with respect to the requirements identified by the user;

• **target usage assessment**, it considers different product characteristics, i.e., license, compliance with standards, implementation language, internationalization support, availability of books about the product, and the interest to the product by well known industry and marker analysts and consultants (like Gartner or IDC);

• **internal quality**, it refers to quality aspects that are not disclosed for commercial software. For the purpose of evaluating the internal qualities there is the availability of many metrics proposed in literature and effectively supported by tools, like McCabe Cyclomatic Complexity, Chidamber and Kemerer’s object-oriented metrics suite, etc.

• **external quality**, it is related to the defect density of the product, the number and severity of bugs over time, as well as defect removal speed.

• **probability of support in the future**, it is related to the probability that the product will be dismissed in the future or that its support will be maintained in the future.

The main limitation of the OpenBQR is that it does not take into consideration the analysis of source code and the collection of the data required to apply the model has to be performed by humans by means of web-based tool specifically developed for carrying out the evaluations according to the criteria defined by the OpenBQR.

### 2.2.5 OMM: QualiPSo OpenSource Maturity Model

The QualiPSo [4] model evaluates the trustworthiness of an OSS project in terms of the cost of adoption and deployment, customer satisfaction and developer quality. OMM focuses on process quality and improvement, and only indirectly on the product quality. It is a CMM-like process model for FLOSS development and aims to help in building trust in development processes of companies using or producing FLOSS products.

![OMM Overview](http://qualipso.icmc.usp.br/OMM/)

OMM is also in levels, each level building on and including trustworthy elements (TWEs) at the lower level (see Fig. 9). Each maturity level (Basic, Intermediate or Advanced) represents a certain
level of FLOSS process maturity. A maturity level is said to be achieved only when all mandatory TWE’s included in that level are fulfilled. Movement to a higher maturity level requires that lower level mandatory practices are fully satisfied.

The TWEs that are considered in OMM are from CMMI and from a survey performed during the QualiPSo project that led to the definition of the following FLOSS specific TWEs: Product Documentation (PDOC), Popularity of the SW Product (REP), Use of Established and Widespread Standards (STD), Availability and Use of a (product) Roadmap (RDMP) Quality of Test Plan (QTP), Relationship between Stakeholders (Users, Developers etc) (STK), Licenses (LCS), Technical Environment (ENV) Number of Commits and Bug Reports (DFCT), Maintainability and Stability (MST), Contribution to FLOSS Product from SW Companies (CONT), and Results of Assessment of the Product by 3rd Party Companies (RASM).

OMM has been defined to be adopted by both FLOSS development teams and FLOSS system integrators. The former may use the model as a set of guidelines for setting up and continually improving their development processes. The latter may use OMM to find guidance on what to look for when considering to integrate a FLOSS project to their solution.

### 2.2.6 SQO-OSS: Software Quality Observatory for Open Source Software

The SQO-OSS [25] is constructed with the focus on automation, and consequently it does not evaluate the functionality of OSS products. In fact, functionality assessment cannot be automated and it always requires user inference. SQO-OSS focuses mainly on the source code of a project. It con-

![SQO-OSS quality model](image.png)

Figure 10: The SQO-OSS quality model [25]

siders some aspects of the community as well, but community characteristics constitute only a small part of the quality model. The quality characteristics that are considered by SQO-OSS to assess the
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzability</td>
<td>Cyclomatic Number</td>
</tr>
<tr>
<td></td>
<td>Cyclomatic Number</td>
</tr>
<tr>
<td></td>
<td>Comments frequency</td>
</tr>
<tr>
<td>Changeability</td>
<td>Average size of statements</td>
</tr>
<tr>
<td></td>
<td>Vocabulary frequency</td>
</tr>
<tr>
<td></td>
<td>Number of unconditional jumps</td>
</tr>
<tr>
<td>Stability</td>
<td>Number of unconditional jumps</td>
</tr>
<tr>
<td></td>
<td>Number of entry nodes</td>
</tr>
<tr>
<td></td>
<td>Number of exit nodes</td>
</tr>
<tr>
<td>Testability</td>
<td>Number of exits of conditional structs</td>
</tr>
<tr>
<td></td>
<td>Cyclomatic number</td>
</tr>
<tr>
<td></td>
<td>Number of nested levels</td>
</tr>
<tr>
<td>Maturity</td>
<td>Number of open critical bugs in the last 6 months</td>
</tr>
<tr>
<td></td>
<td>Number of open bugs in the last six months</td>
</tr>
<tr>
<td>Effectiveness</td>
<td>Number of critical bugs fixed in the last 6 Months</td>
</tr>
<tr>
<td></td>
<td>Number of critical bugs fixed in the last 6 Months</td>
</tr>
<tr>
<td></td>
<td>Number of exit nodes</td>
</tr>
<tr>
<td>Security</td>
<td>Null dereferences</td>
</tr>
<tr>
<td></td>
<td>Undefined values</td>
</tr>
<tr>
<td>Mailing list</td>
<td>Number of unique subscribers</td>
</tr>
<tr>
<td></td>
<td>Number of messages in user/support list per month</td>
</tr>
<tr>
<td></td>
<td>Number of messages in developers list per month</td>
</tr>
<tr>
<td>Documentation</td>
<td>Available documentation documents</td>
</tr>
<tr>
<td></td>
<td>Update frequency</td>
</tr>
<tr>
<td>Developer base</td>
<td>Rate of developer intake</td>
</tr>
<tr>
<td></td>
<td>Rate of developer turnover</td>
</tr>
</tbody>
</table>

Table 2: Subset of the metrics used to evaluate Product (Code) Quality and Community Quality in SQO-OSS \[25\]

The quality characteristics included in the SQO-OSS model are measured by means of specific metrics (for explanatory purposes some of them are reported in Table 2). In order to score each quality attribute, SQO-OSS defines also an aggregation approach able to aggregate the values of the calculated metrics for each quality attribute.

### 2.2.7 QualOSS

In the context of the EU project QualOSS a comprehensive measurement framework for OSS project has been defined \[10\]. The approach relies on the QualOSS (quality) Model which has been specifically defined to cover both robustness of the evaluated product, as well as its evolvability, that is its ability to thrive and keep growing over time \[26\]. Open source projects are analysed by considering different information sources, such as source code, mailing lists, bug tracking systems, and version-
The QualOSS model is composed of three interrelated elements namely, quality characteristics, metrics, and indicators. According to [26], quality characteristics correspond to the attributes of a product or community that we consider relevant for evaluation. Metrics correspond to concrete aspects that can be measured and that are correlated with targeted quality characteristics. Indicators define how to aggregate and evaluate the measurement values in order to obtain a consolidated value that can be readily used by decision makers when performing an evaluation. The quality characteristics of the QualOSS model are shown in Fig. 11.

The metrics and indicators required to support the quality assessment according to the QualOSS quality model have been defined by following a process defined in [10] that lead to a set of Open-Document spreadsheet documents. Each document corresponds to a particular area of the model. Inside the document, at least two spreadsheets are available, namely, one for metric definitions and one for indicator definitions.

As previously mentioned, calculated metrics are aggregated by means of indicators that permit to interpret their underlying metric values on the following scale:

- **Green** meaning no or minor risks for the measurement object in relation to a given characteristic
- **Yellow** meaning significant risk for the measurement object in relation to a given characteristic
- **Red** meaning critical risks for the measurement object in relation to a given characteristic

---

5 https://www.dropbox.com/s/akfo20iqj4jemmj/WP4_Deliverable4_2.StdQualOSSAssessment-1.1.zip
Black meaning prohibitive risks for the measurement object in relation to a given characteristic.

To enable mathematical aggregations, such a scale is mapped onto an interval ranging from -100 to +100.

QualOSS stresses the importance of considering community characteristics when evaluating the overall quality of open source products. Some of the community measures can be automatically computed, and their collection requires only setting up measurement tools. Other measures, for example the ones related to process maturity, require more manual effort.

2.2.8 Squale Software Quality Model

The Squale software quality model is an open-source quality model developed by several companies and pushed further in the context of the Squale research project [15]. It uses measurements to assess software quality. These measurements cover a number of different aspects of a software, including specification accuracy, programming rules, and test coverage. The Squale model is tool supported by MoQam (Moose Quality Assessment Model) that is the implementation of the model in the Moose open-source reengineering environment.

The Squale model is inspired by the factors-criteria-metrics (FCS) model of McCall [13]. However, while McCall defined a top-down model to express the quality of a system, the Squale model promotes a bottom-up approach, aggregating low-level measures into more abstract quality elements. In addition to the FCS model, Squale introduces the new level between criteria and metrics as shown in Fig. [12].

At the lower level of abstraction, a measure is a raw piece of information extracted from the project data. It comes from human expertise (manual measures) or from different tools (raw metrics, e.g., code metrics, rule checking metrics, or test metrics).

Higher level marks are practice, criterion, and factor. A practice assesses whether a technical principle in the project is respected. Composition and aggregation of low-level marks occurs at this level. A criterion assesses one main component of software quality (e.g., the criterion Simplicity assesses...
D2.4 – Metamodels for OSS Project Metrics

<table>
<thead>
<tr>
<th>Quality Model</th>
<th>Code analysis</th>
<th>Community analysis</th>
<th>Tool support</th>
<th>Automatic metric collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSMM</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OpenBRR</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QSOS</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>OpenBQR</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>OMM</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>SQO-OSS</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>QualOSS</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>partial</td>
</tr>
<tr>
<td>Squale</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 3: Summary of the OSS specific quality models

the source code readability and the ease to diagnostic regardless documentation). A factor represents the highest-level quality assessment, used to provide an overview of a project’s health. It is addressed to non-technical persons. Factors correspond roughly to the characteristics of the ISO 9126.

Each computed low-level mark gives a result in its own range, while high-level marks are all normalized to $[0, 3]$. In particular, $[0, 1]$ maps to “goal not achieved”, $[1, 2]$ maps to “goal mostly achieved”, and $[2, 3]$ maps to “goal achieved”.

Squale has been applied on several industrial projects by monitoring millions of lines of code. It uses connectors to third-party tools to collect raw data from source configuration management, to apply source code metrics, to perform test coverage, rule-checking, security-checking, etc. However, communication channels and in general communities built around projects are not taken into account for the evaluations.

2.2.9 Summary

As shown in Table 3, most of the overviewed quality models lack some dimensions when evaluating FLOSS projects, most notably the community. Moreover, most of the models are based on manual work, supported by evaluation forms. SQO-OSS considers both code and community aspects, and provides user with a tool support both for collecting metric values and to aggregate them. However, none of the considered approaches provide the means to customize the quality models in terms of quality attributes to be considered and metrics to measure for evaluating them.

In the next section the OSSMETER quality model is presented. It is implemented on top of the OSSMETER platform that permits to address all the characteristics shown in Table 3 and additionally provides the means to customize the quality model according to user needs.
3 OSSMETER quality model

In this section we present the OSSMETER quality model. It is important to remark that we are not aiming at proposing a "yet another quality model". As shown in the previous section, over the last decades many quality models have been proposed and none of them seem to have gained the status of standard de facto. A general explanation of that is the fact that depending on the application domain and user needs, the required quality characteristics supporting the evaluation of a given open source product might change. In this respect, the OSSMETER project proposes an extensible platform that let users to define its own quality model and even extend the set of available metrics providers that are required to measure specific quality attributes that are considered to relevant by users. Another important concept supported by the OSSMETER platform is that of factoids [19], i.e., textual summaries of measurement results including a star rating. Most source metrics from WP3 and WP4 will serve as input parameters for one or more factoids.

In OSSMETER the quality model groups logically-related metrics and factoids together, and as such guides the presentation of these metrics. As explained in the deliverable D5.6 - End User Web Application [20] single alphabetical list of all metrics would be difficult to consume, whereas metrics grouped by quality attribute would aid the user in interpreting the metrics and therefore aiding their understanding of the quality of the project. The quality model permits also to define the overall quality score as a personalised aggregation of the metrics/factoids that are important to them. The implementation of this is presented in the deliverable D5.6.

Quality models that can be defined in OSSMETER conform to the OSSMETER quality metamodel presented in Section 3.1. In OSSMETER a default quality model has been conceived even though it can be refined if it does not completely satisfy user requirements 3.2.

3.1 OSSMETER quality metamodel

According to the literature survey that we have summarized in the previous section, the concepts we consider peculiar to define a QualityModel are shown in the metamodel shown in Fig. 13 and are explained below.

Figure 13: OSSMETER quality metamodel
QualityAttribute: it represents a quality aspect that is considered to be relevant for contributing to the assessment of the quality of a considered product. Each quality attribute has associated a number of facts that are obtained by corresponding factoids. A quality attribute, like code quality can be an aggregation of other attributes, like maintainability, maturity, and testability. Each quality attribute specifies how to aggregate the contained attributes in order to provide an overall quality score for the considered attribute. It is important to remark that such a score is very challenging to produce and the user should not put too much emphasis on this value - it is just an indicator.

Factoid: it represents a fact about the considered product. Such a fact is obtained by considering the values of related metrics. In other words a factoid is an aggregation unit depending on a number of related metrics. As explained in D5.4 [19], since metric aggregation and metric combination is a very active area of research, OSSMETER does not commit to a specific approach. Thus the factoid mechanism is proposed as a means to aggregate heterogeneous metric providers into a 1-4 star system and a piece of text with an interpretation of the numerical value.

Metric provider: it represents the software component that implements a metric by using the functionality provided by the OSSMETER platform. Lines of Code (LOC) is a very simple example of metric that can be implemented with a corresponding metric provider.

Next section presents the proposed quality model conforming to the metamodel in Fig. 13 and that is included in the default installation of the OSSMETER platform.

### 3.2 The default OSSMETER quality model

The quality model that is already implemented and configured in the OSSMETER platform is shown in Fig. [14] It is an instance of the OSSMETER quality metamodel, and consequently it can be customised according to user needs. The proposed quality model consists of attributes hierarchically organized. They are distinguished into those related to the quality of the product code and those related to the community built around the considered product. In order to enable the automated manipulation of the quality model especially to configure the front-end of the OSSMETER platform (see deliverable D5.6), a JSON representation of it is required. In Appendix A.1 a fragment of the JSON representation of the quality model shown in Fig.14 is given.

In the next section the quality attributes of the defined quality model are singularly described. The factoids that are used to measure such quality attributes are properly organized as presented in Section 3.2.2 For a complete description of such factoids their implementation in terms of metrics, we refer to the appropriate deliverables i.e., D3.3 [2], D3.4 [3], and D4.6 [18].

#### 3.2.1 Quality attributes

Product Code quality: it identifies the quality factors that influence the ability to understand software product, change and test it, as well as it permits to assess the maturity of the considered product by considering the way it is developed. Product code quality is assessed by considering the following attributes:

- Maintainability: it is an important attribute that refers to how easy is the software to understand and modify. Thus it is further decomposed into the following attributes:
Figure 14: The default OSSMETER quality model

- **Analyzeability**: it refers at what extent the source code of the considered product is properly designed and implemented. Adherence to coding style and usage of comments in the code are only some examples of facts that can affect the analyzability of the product code.
- **Changeability**: it refers to the effort required to change the product e.g., to address unforeseen requirements. The size of code, the used programming languages, the amount of cloned code, or even characteristics like coupling and cohesion are examples of facts that can affect the changeability property of a given product.
  
  - **Maturity**: it refers to the possibility of having software failures. The amount of error prone or even inefficient code can reduce the overall maturity of the considered product.
  - **Testability**: software systems are evolving entities and as such a sign of quality is also the effort needed to test the system when for some reason it has been changed. The availability of unit tests is an example of fact that can affect the testability factor of a given product.

*Community quality*: Considering characteristics of the the community strongly influences the overall product quality, especially when they observed over an extended period of time. Moreover, the ability of a FOSS community to remain active over time is obviously very important for product survival, and thus very relevant when considering sustainability, i.e., the capability of a product to sustain its self. In the following, we present factors that are related to the quality of product communities.
• Development team quality: An important quality factor of products is represented by some characteristics of the development teams working and supporting them. The decision of adopting an open source product takes in consideration for sure if the project will be maintained in the long-run, or even the commitment of the people developing and working on it. In this respect the following quality attributes are considered:
  – Stability: it refers to the availability of a stable number of developers working on the project. The number of commits over time per developer, is an example of measure that can contribute to understand if the team working on a product is growing or shrinking.
  – Professionalism: it refers to the experience of the developers working on the product. This can be measured by considering their activity e.g., the number of commits per developer, or the number of commits per day, etc.
  – Commitment: if a project is developed and maintained by a representative number of developers with a high level of commitment to the project is of course a good sign. The commitment of a developer can be assessed by analysing his/her activity e.g., if there are commits done over the weekend, or if the amount of code committed by the developer is increasing.

• Communication channel quality: further than the development team, the community of a project consists also of people interacting with the bug tracking systems and the newsgroups of the product. In this respect, the following quality attributes have been considered:
  • Bug Tracking System Quality: it refers to quality aspects that can be assessed by considering information retrieved from the bug tracking systems of a product. By considering such systems it is possible to assess the following quality attributes:
    – Sustainability: the responsiveness to bug reports and to feature requests is an important measure that permits to assess if there is a strong community maintaining the product and its development over time.
    – Maintenance capacity: it refers to at what extent potential users might take advantage of the information available from bug tracking systems and thus being supported during the adoption of the product. The number of available bug tracking systems and the number of comments therein are only some of the facts related to the maintenance capacity of a product.
    – Maturity: this attribute permits to assess if a considered bug tracking system is mature with respect to the ways bugs are replied (e.g., properly or inadequately), to number of bugs closed, fixed, and even duplicated.
    – Polarity: sentimental and emotional polarities are important indicators about the overall consideration of the users that are adopting the considered product. For instance, if the average sentimental and emotional polarities in all bug tracking systems associated with the project are negative then it is possible to conclude that people that are using the project and are trying to ask for support are somehow unhappy and consequently the overall quality of the product is affected.
  • Newsgroup Quality: it refers to quality aspects that can be assessed by considering information retrieved from the newsgroups of a product. The quality attributes related to this aspect are the same of those defined for bug tracking systems, i.e., sustainability, maintenance capacity, maturity and polarity.
In the next section we present how the quality attributes previously described are measured by means of factoids.

### 3.2.2 Measuring quality attributes by means of factoids

As said above, the quality attributes defined in the OSSMETER quality model are measured by means of a set of factoids developed in WP3 and WP4. More precisely, the factoids measuring quality attributes related to product code and to development teams are presented in the deliverables D3.3 and D3.4. The factoids measuring quality attributes related to bug tracking systems and news groups of a product are presented in the deliverable D4.6. Table 4, 5, 6 show the developed factoids in order to convey the information about what factoids are used to measure what quality attribute. The second column of such tables refer also to the corresponding deliverables where the mentioned factoids are properly presented also in terms of metrics providers implementing them. The third column of the tables shows sample (template) strings that the corresponding factoids emit when executed.

It is important to recall that each project, which is monitored by the OSSMETER platform by means of the proposed quality model has a 1-4 stars score related. Currently, such a score is obtained by simply calculating the arithmetic mean of all the factoid scores. This solution is not intended to be optimal and aims at giving just an indicator about the project quality. Of course, such a behaviour can be changed by extending the platform and/or refining the quality model. Details about how to do that are presented in the deliverable D5.6.

#### Table 4: Code quality attributes and corresponding factoids

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Factoids</th>
<th>Example of output strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysability</td>
<td>percentageCommentedOutCode [2]</td>
<td>The percentage of commented out code over all measured languages is totalPercentage. The percentages per language are commaSeparatedTable.</td>
</tr>
<tr>
<td></td>
<td>commentPercentage [2]</td>
<td>The percentage of lines containing comments over all measured languages is totalPercentage.</td>
</tr>
<tr>
<td></td>
<td>understandabilityFactoid [3]</td>
<td>Currently, there is hardly any understandable code in this project and this situation is stable in the last six months. Currently, hard to understand code practices are widely spread throughout the project, but the situation has been improving over the last six months.</td>
</tr>
</tbody>
</table>
### Changeability

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>codesize</td>
<td>The total size of the code base is totalSize physical lines of code.</td>
</tr>
<tr>
<td></td>
<td>The main development language of the project is mainLanguage, with locFor-</td>
</tr>
<tr>
<td></td>
<td>MainLanguage physical lines of code. The following (size(sorted) - 1) oth-</td>
</tr>
<tr>
<td></td>
<td>er languages were recognized: tabelWithOtherSizes.</td>
</tr>
<tr>
<td>coupling</td>
<td>The percentage of language classes with problematic coupling is badPercentage.</td>
</tr>
<tr>
<td>cohesion</td>
<td>The percentage of language classes with problematic cohesion is badPercentage.</td>
</tr>
<tr>
<td>cloneCode</td>
<td>The percentages of clone code per language are tableWithPercentages.</td>
</tr>
<tr>
<td>cyclomaticComplexity</td>
<td>The cyclomatic complexity footprint of the system’s language code shows a</td>
</tr>
<tr>
<td></td>
<td>(very high</td>
</tr>
<tr>
<td></td>
<td>CC are respectively perc1, perc2 and perc3.</td>
</tr>
<tr>
<td>commitLocality</td>
<td>Commits are usually local to a single file. Commits usually include between</td>
</tr>
<tr>
<td></td>
<td>5 and 10 files.</td>
</tr>
</tbody>
</table>

### Maturity

<table>
<thead>
<tr>
<th>Factoid</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorProneFactoid</td>
<td>Currently, there is no error prone code in this project and this situation</td>
</tr>
<tr>
<td></td>
<td>is stable in the last six months.</td>
</tr>
<tr>
<td></td>
<td>Currently, error prone code practices are widely spread throughout the proj-</td>
</tr>
<tr>
<td></td>
<td>ect, but the situation has been improving over the last six months.</td>
</tr>
<tr>
<td>inefficienciesFactoid</td>
<td>Currently, there is no obviously inefficient code in this project and this</td>
</tr>
<tr>
<td></td>
<td>situation is stable in the last six months.</td>
</tr>
<tr>
<td></td>
<td>Currently, obviously inefficient code practices are widely spread throughout</td>
</tr>
<tr>
<td></td>
<td>the project, but the situation has been improving over the last six months.</td>
</tr>
</tbody>
</table>
The percentage of files with a header is \textit{headerPercentage}.
The largest group of similar headers spans \textit{highestSimilarity} of the files.
Only \textit{headerPercentage} of the files contain a header.
No headers found.

| Testability                  | JavaUnitTestCoverage [3] | The percentage of methods covered by unit tests is estimated at \textit{testCoverage}.
This situation is \textit{(stable|getting worse|improving)} over the last six months.
The estimated coverage of public methods is \textit{testOverPublicMethods}. |

Table 5: Development team quality attributes and corresponding factoids

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Attribute</th>
<th>Factoids</th>
<th>Example of output strings</th>
</tr>
</thead>
</table>
| Stability        | developmentTeamStability [2] | In the last half year the development team was stable and active.
The project has seen hardly anybody developing in the last half year.
People have been leaving the development team in the last half year. |
|                   | developmentTeamExperience-Spread [2] | Developers are spreading out over the entire project and it can be expected that most files have more than one contributor. |
Professionalism

There were no experienced committers working for the project in the last 6 months. The was only one experienced committer working for the project in the last 6 months. Overall, he/she contributed **this-Much** commits. The number of experienced committers working for the project in the last 6 months is `numExperiencedCommitters`. Their average overall number of commits is `thisMuch`.

Commitment

In the last two weeks the average size of a commit was `ratio`, this is **(more|less)** than the normal `ratioMedian` in the last six months. The trend is **(going down| stable | going up)**.

<table>
<thead>
<tr>
<th>Professionalism</th>
<th>developmentTeamExperience</th>
<th>There were no experienced committers working for the project in the last 6 months. The was only one experienced committer working for the project in the last 6 months. Overall, he/she contributed <strong>this-Much</strong> commits. The number of experienced committers working for the project in the last 6 months is <code>numExperiencedCommitters</code>. Their average overall number of commits is <code>thisMuch</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitSize</td>
<td>In the last two weeks the average size of a commit was <code>ratio</code>, this is **(more</td>
<td>less)** than the normal <code>ratioMedian</code> in the last six months. The trend is **(going down</td>
</tr>
<tr>
<td>Commitment</td>
<td>weekendProject</td>
<td>Over the entire lifetime of this project, commits have been done usually over the weekend. Over the entire lifetime of this project, commits are done usually during the week, and hardly during the weekend.</td>
</tr>
<tr>
<td>churnVolume</td>
<td>In the last two weeks the churn was <code>churn</code>, this is **(more</td>
<td>less)** than the normal <code>churn-Median</code> in the last six months. The trend is **(going down</td>
</tr>
</tbody>
</table>

Table 6: Communication channels quality attributes and corresponding factoids

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Factoids</th>
<th>Example of output strings</th>
</tr>
</thead>
</table>
## BTS Sustainability

| org.ossmeter.factoid.bugs.channelusage | The project is associated with $X$ bug tracking systems. In the last year, they have (received high | received much | received some | not received much) attention. $A$ new bugs, $B$ new comments and $C$ new patches have been posted, in total. $D$ new bugs, $E$ new comments and $F$ new patches have been posted to bug tracking system $G$. In the last month, $H$ new bugs, $I$ new comments and $J$ new patches have been posted to the bug trackers of the project. $K$ new bugs, $L$ new comments and $M$ new patches have been posted to bug tracking system $N$. |
| --- | --- |

## BTS Maintainance Capacity

| org.ossmeter.factoid.bugs.size | The project is associated with $X$ bug tracking systems. Bug tracking system $A$ is of (very large | large | medium | small) size. It contains $B$ bug, $C$ comments and $D$ patches. |
| org.ossmeter.factoid.bugs.responsetime | Lately, requests receive a first response (very | fairly | not so) quickly. Response speed is lately (better | approximately equal | worse) than the overall average for the project. |

| org.ossmeter.factoid.bugs.users | The community of the project contains $X$ users, of which currently $A$ are active ($A/X$%). There are approximately $B$ new users per day in the last month, while approximately $C$ users are active. Each user has contributed approximately $D$ messages, $E$ requests and $F$ replies. In the last year, there are $G$ new users per day in the last month, $H$ of which are active. Each user has contributed approximately $J$ comments. Bug tracking system $M$ hosts $N$ users, of which $O$ are currently active ($O/N$%) |
**Threaded discussions tend to be** (long | of medium length | short), approximately $X$.

**The number of comments**, (largely depends | does not depend) on the day of the week. There is (no) significant difference between the number of comments during working days and weekends.

**The number of comments**, (largely depends | does not depend) on the hour of the day. There is (no) significant difference between the number of comments within as opposed to out of working hours.

**The project is associated with** $X$ bug tracking systems.

Bugs on bug tracking system $A$ are on average replied (excellent | well | inadequately).

In a total of $D$ bugs, $E$ are resolved ($E/D\%$) and $F$ are non-resolved ($F/D\%$).

$G$ bugs are closed as fixed ($G/D\%$), $H$ as non-reproducible ($H/D\%$), $I$ as won’t fix ($I/D\%$), $J$ as invalid ($J/D\%$) and $K$ as duplicates of other bugs ($K/D\%$).

**The average sentimental polarity measured on all comments is** $A$. The measurement lies in the range $[-1,1]$, where -1 designates entirely negative sentiment while 1 designates entirely positive sentiment.

The middle of the range, designates neutral sentiment.

The average sentimental polarity in all bug tracking systems associated with the project is close to (negative | neutral | positive).

In the beginning of threads, the average sentiment score is $B$, while at the end of threads it is $C$, showing that users are (happier | equally happy | unhappier) at the end of a discussion (than | as) in the beginning of it.
There are (many | not so many | few | very few) comments that express positive emotions, while there are (many | not so many | few | very few) comments that express negative emotions. The most and least common negative emotions are $A (B\%)$ and $C (D\%)$, while the most and least common positive emotions are $E (F\%)$ and $G (H\%)$.

The project is associated with $X$ newsgroups. In the last year, they have (received high | received much | received some | not received much) attention. $A$ new threads, and $B$ new articles have been posted, in total. $C$ new threads and $D$ new articles have been posted to newsgroup $E$. In the last month, $F$ new threads and $G$ new articles have been posted to the newsgroups of the project. $H$ new threads and $I$ new articles have been posted to newsgroup $J$.

The community of the project contains $X$ users, of which currently $A$ are active ($A/X\%$). There are approximately $B$ new users per day in the last month, while approximately $C$ users are active. Each user has contributed approximately $D$ messages, $E$ requests and $F$ replies. In the last year, there are $G$ new users per day in the last month, $H$ of which are active. Each user has contributed approximately $J$ messages, $K$ requests and $L$ replies. Newsgroup $M$ hosts $N$ users, of which $O$ are currently active ($O/N\%$).
### Newsgroup maintenance capacity

| org.ossmeter.factoid .newsgroups.size | The project is associated with $X$ newsgroups. Newsgroup $A$ is of \( \text{very large} \mid \text{large} \mid \text{medium} \mid \text{small} \) size. It contains $B$ threads and $C$ articles. |
| org.ossmeter.factoid .newsgroups.responsetime | Lately, requests receive a first response \( \text{very} \mid \text{fairly} \mid \text{not so} \) quickly. Response speed is lately \( \text{better} \mid \text{approximately equal} \mid \text{worse} \) than the overall average for the project. |
| org.ossmeter.factoid .newsgroups.threadlength | Threaded discussions tend to be \( \text{long} \mid \text{of medium length} \mid \text{short} \), approximately $X$. In particular, a thread contains approximately $Y$ requests and $Z$ replies. |
| org.ossmeter.factoid .newsgroups.weekly | The number of articles, requests or replies, \( \text{largely depends} \mid \text{does not depend} \) on the day of the week. There is \( \mid \text{no} \) significant difference between the number of articles during working days and weekends. |
| org.ossmeter.factoid .newsgroups.daily | The number of articles, requests or replies, \( \text{largely depends} \mid \text{does not depend} \) on the hour of the day. There is \( \mid \text{no} \) significant difference between the number of articles within as opposed to out of working hours |

### Newsgroup Maturity

| org.ossmeter.factoid .newsgroups.status | The project is associated with $X$ newsgroups. Newsgroup $A$ hosts $B$ requests \( \text{(percentage } \%) \) and $C$ replies \( \text{(percentage } \%) \), showing that requests are on average replied \( \text{excellent} \mid \text{well} \mid \text{inadequately} \). In total $D$ threads are unanswered. There are approximately $E$ requests \( \text{($E/(E+F)$ } \%) \) and $F$ replies \( \text{($F/(E+F)$ } \%) \) per day. |
The average sentimental polarity measured on all articles is $A$. The measurement lies in the range $[-1,1]$, where -1 designates entirely negative sentiment while 1 designates entirely positive sentiment. The middle of the range, designates neutral sentiment.

The average sentimental polarity in all newsgroups associated with the project is close to (negative \ neutral \ positive).

In the beginning of threads, the average sentiment score is $B$, while at the end of threads it is $C$, showing that users are (happier \ equally happy \ unhappier) at the end of a discussion (than \ as) in the beginning of it.

There are (many \ not so many \ few \ very few) articles that express positive emotions, while there are (many \ not so many \ few \ very few) articles that express negative emotions. The most and least common negative emotions are $A$ ($B\%$) and $C$ ($D\%$), while the most and least common positive emotions are $E$ ($F\%$) and $G$ ($H\%$).
There are (many | not so many | few | very few) newsgroup threads that report serious (i.e. major, critical and blocker) software problems. The response time for these threads is (very quick | quick | not so quick | quite slow).

On average threads about serious issues are addressed (more | less | equally) quickly than threads about less important issues.

On average, users express (more positive | equally positive | more negative) sentiments about how serious issues are being resolved than how all other issues are being resolved.

There are (many | not so many | few | very few) newsgroup threads that propose or ask for new features.
4 Conclusions

In this document we presented the OSSMETER quality metamodel and one specific instance of it. Their definition has been performed after having reviewed representative quality models, that have been distinguished as generic and OSS specific. The default quality model defined in this document permits to support the evaluation and the comparison of open source projects by considering both their source code and the communities built around them. To this end, for each quality attribute, the quality model refers to factoids developed in WP3 and WP4. A JSON representation of the quality model is used to configure the front-end of the application developed in WP5.

It is important to recall that the OSSMETER quality model will be continuously refined to address unforeseen requirements and to extend its expressiveness with respect to the outcomes of the evaluation phase that will be done by the use case providers during the last six months of the project. The final quality model and the description of how it is used to tie all the developed pieces together will be part of the deliverable D5.7: OSSMETER Platform - Final Version.

Compliance to the OSSMETER WP2 Requirements  In the deliverable D1.1 [29] the requirements of the OSSMETER project are listed. Such requirements were defined by both the Use Case providers and the research partners of the project. All the requirements presented in D1.1 are identified in terms of the modalities SHALL, SHOULD, and MAY defined as follows:

- SHALL is used to denote an essential requirement. A typical system could not be used, would not work, or cannot be validated if this requirement is not fulfilled. SHALL requirements are of highest priority for validation of the platform.
- SHOULD is used to denote a requirement that would help a typical system be easier to use, or to work better, even if it is not essential; in that case a trade-off can be achieved between development costs on the technology side and user benefit on the system side.
- MAY is used to denote a requirement that can lead to a benefit in order to fulfil an additional evaluation criterion or increase the usefulness of the technology. The fulfilment of the requirement is interesting but only in view of available resources and research and development partner interests.

In the deliverable D2.2 [16] we have already discussed how the work done by WP2 about the developed metamodels has addressed all the requirements but two of them since they are most related to

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
<th>Originating WP</th>
</tr>
</thead>
<tbody>
<tr>
<td>R7</td>
<td>Metric metamodels shall be defined by properly organizing the different metamodels supporting the metric providers that will be developed by all the technical workpackages.</td>
<td>SHALL</td>
<td>WP2</td>
</tr>
<tr>
<td>R8</td>
<td>Metric metamodels shall be used to defined the quality and activity metrics that the OSSMETER platform is able to calculate</td>
<td>SHALL</td>
<td>WP2</td>
</tr>
</tbody>
</table>

Table 7: Requirements from research partners not already addressed in D2.2 [16] and D2.3 [17]
the work presented in this document. In particular, in deliverable D2.2 the requirements shown in Table 7 are specified as not addressed since related to the work that has been done after then. With the work presented in this document, both R7 and R8 are now fully addressed. In fact the quality metamodel presented in Section 3 permit to define the quality attributes that are considered relevant for analysis and comparison purposes, and how to measure them by means of factoids (currently developed by WP3 and WP4).
References


[18] University of Manchester. D4.6 - Clustering System for Threads, October 2014.


[29] The Open Group with contributions from all partners. D1.1 - Project Requirements, June 2013.
A Appendix

A.1 JSON representation of the Default OSSMETER Quality Model

The following listing shows a fragment of the quality model represented in JSON that are used by any installation of the OSSMETER platform based on the default quality model presented in Section 3.

```json
{
  "qualityModel": {
    "name": "Default_OSSMETER_quality_model",
    "qualityAspects": [
      {
        "name": "code",
        "qualityAspects": [],
        "attributes": [
          {
            "name": "Maintainability",
            "description": "...",
            "factoids": ["percentageCommentedOutCode","commentPercentage","understandabilityFactoid"],
            "metrics": ["locPerLanguage","commentLinesPerLanguage","FileLength"]
          },
          {
            "name": "Changeability",
            "description": "...",
            "factoids": ["codesize","coupling","cohesion","cloneCode","cyclomaticComplexity"],
            "metrics": ["locPerLanguage","CBO-Java","lcom4-java","locPerLanguage","CC-Java"]
          }
        ],
      },
      {
        "name": "Maturity",
        "description": "...",
        "factoids": [...] ,
        "metrics": [...] 
      },
      {
        "name": "Testability",
        "description": "...",
        "factoids": [...] ,
        "metrics": [...] 
      }
    ],
    "attributes": {
      "name": "developerBaseQuality",
      "attributes": [
        {
          "name": "Stability",
          "description": "...",
          "factoids": [...] ,
          "metrics": [...] 
        },
        {
          "name": "Professionalism",
          "description": "...",
          "factoids": [...] ,
          "metrics": [...] 
        }
      ]
    }
  }
}
```

"metrics": [...]}
]
}

"qualityAspects":{}
}

"name": "communicationChannel",
"attributes": [
"qualityAspects": [

"name": "bugTrackingSystem",
"attributes": [

"name": "Sustainability",
"description": "...",
"factoids": [...],
"metrics": [...]
],

"name": "Maintenance_capacity",
"description": "...",
"factoids": [...],
"metrics": [...]
],

"name": "Maturity",
"description": "???",
"factoids": [...],
"metrics": [...]
],

"name": "Polarity",
"description": "...",
"factoids": [...],
"metrics": [...]
],

"name": "newsGroup",
"attributes": [

"name": "Sustainability",
"description": "...",
"factoids": [...],
"metrics": [ ]
],

"name": "Maintenance_capacity",
"description": "...",
"factoids": [...],
"metrics": [...]
],

"name": "Maturity",
"description": "...",
"factoids": [...],
"metrics": [...]
],

"name": "Polarity",
"description": "...",
"factoids": [...],
"metrics": [...]
],
Listing 1: Fragment of the JSON representation of the default OSSMETER quality model

```json
  "description": "...",
  "factoids": [...],
  "metrics": [...]
}
```