Qualoss

Sponsored through Framework Programme Sixth (Call 5) by

European Commission

Information Society Technologies

Document Information

Version: 1.0
Date : Oct 24, 07
Pages : 74

Owning Partner:
IESE

Author(s):
Marcus Ciolkowski, Martín Soto, Jean-Christophe Deprez, Frédéric Fleurial Monfils, Flora Kamseu, Jose Ruiz, Alvaro del Castillo, Daniel Izquierdo

Reviewer(s):
Jean-Christophe Deprez

To:
CONSORTIUM

Purpose of distribution:

The QUALOSS Consortium consists of: CETIC (BE), Facultés Universitaires Notre Dame de la Paix à Namur (BE), Universidad Rey Juan Carlos (ES), Fraunhofer IESE (DE), ZEA Partners (BE), MERIT (NL), AdaCore (FR), PEPITe (BE)

Printed
on 10/22/07 at 04:00:34 PM

Status:

[ ] Draft
[ ] To be reviewed
[ ] Proposal
[X] Final/Released

Confidentiality:

[X] Public - Intended for public use
[ ] Restricted - Intended for QUALOSS consortium only
[ ] Confidential - Intended for individual partner only

Deliverable ID: **D1.5**

Title:

QualOSS D1.5

(Calibration of the Prototype QualOSS Model)

A deliverable of Task 1.5

Deliverable: D1.5

Title: QualOSS D1.5

Executive Summary:
This document describes the work done and results obtained in task 1.5 ("Calibration of Metrics Systems and Prototype Quality Models") of the QualOSS project.

This deliverable has been first and mainly created as a wiki, and then extracted into this document. The wiki content will continue to be evolved; in this view, this deliverable represents a snapshop of the QualOSS work.

The deliverable is structured as follows:

Section 1 presents the motivation of task 1.5, and explains how the tasks in workpackage 1 collaborate to produce the initial QualOSS model.

Section 2 introduces the concept of indicators we used to define the interpretation model, and to connect metrics to quality attributes.

Section 3 describes the indicators we defined for the prototype parts of the QualOSS model; that is, for the quality characteristics with basic metrics that were measured during D1.4.

Section 4 focuses on the results of applying the QualOSS indicators to the projects measured in D1.4. The results themselves are contained in spreadsheets, which are a second part of this deliverable.

Section 5 describes the data mining approach defined for QualOSS, which will be applied for the QualOSS advanced models, and validated in task 1.6.

Section 6 discusses lessons learned and implications for advanced models learned during definition and application of the indicators.

Finally, Section 7 summarizes the results of this deliverable and the task 1.5.

The Appendix contains the template we used to define indicators.

CHANGE LOG

Ver.	*Date*	*Author*	*Description*
0.1	15.08.07	Marcus Ciolkowski	Initial proposal for structure
0.2	07.10.07	Marcus Ciolkowski	Draft Deliverable, containing input from all partners
1.0	22.10.07	Marcus Ciolkowski, Martín Soto	Final version

APPLICABLE DOCUMENT LIST

Ref.	*Title, author, source, date, status*	*Deliverable Identification*

TABLE OF CONTENTS

1. Introduction

1.1 Motivation

The strategic objective of the QuaLOSS project is to enhance the competitive position of the European software industry by providing methodologies and tools for improving their productivity and the quality of their software products. To achieve this objective, QuaLOSS notes that many organizations integrate Free *libre* Open Source Software (F/OSS) in their systems hence QuaLOSS aims at facilitating the selection of the most adequate F/OSS . In particular, QuaLOSS focuses on assessing the evolvability and robustness of F/OSS.

This higher competitiveness is to be addressed by providing a reliable assessment method of open source software in order to integrate them into industrial software. This will ease the integration of high quality level open source components, and increase the productivity.

To achieve this goal, QualOSS proposes to build a high level methodology to benchmark the quality of open source software in order to ease the strategic decision of integrating adequate F/OSS components into software systems. Therefore, one of the main outcomes of the QuaLOSS project is to deliver an assessment methodology for gauging the evolvability and robustness of open source software.

This first workpackage (WP1) performs requirements analysis through prototyping while the other scientific workpackages (WP2-4) improve on the functional prototype build in WP1. The first three tasks of WP1 (T1.1, T1.2 and T1.3) perform requirements analysis while the remaining three tasks (T1.4, T1.5, and T1.6) build the functional prototype and validate the approach. The goal of this deliverable´, D1.5, is to calibrate the quality model defined in D1.3; that is, to define interpretations, where possible, on the basis of the measurements taken in task 1.4.

1.2 Goal

The key result of task 1.5 is to define the interpretation model and to calibrate the metrics for the QualOSS quality model.

More specifically, the goals of task 1.5 are to:

- Define indicators to interpret metrics for each quality attribute, as defined in D1.3.
- Apply the indicators to the metrics collected in D1.4
- Identify necessary adjustments of the QualOSS quality model
- Describe and test an approach for applying data mining techniques

1.3 Strategy For Workpackage 1

The main objective of WP1 is to perform requirement analysis through prototyping. Previous to task 1.5, there existed a quality model; that is, a set of quality characteristics with associated metrics and corresponding measuring tools.

The outcome of prototyping in WP1 serves in performing a thorough requirement analysis in order to test our approaches and to well formulate our requirements for the remainder of the project. It also helps identify promising metrics and tools to integrate in our final QUALOSS platform. A first prototype schema for the QUALOSS repository also emanates from WP1, in particular from task 1.4. If our prototype quality models constructed on basic metrics and the calibration exercise yield interesting results directly usable and transferable to our QUALOSS platform then that is extra benefit.

The tasks of workpackage 1 can be grouped as follows: (1) Definition of goals for the QualOSS method, (2) definition of quality models that support these goals, and (3) evaluation and calibration of the quality models.

(1) Definition of goals to be supported by the QualOSS method is addressed in task 1.2. Thereby, the approach is to first define and elicit usage scenarios for OSS components, and to define evolvability/ robustness based on these scenarios and on related work in quality modelling and assessment of OSS projects.

(2) Definition of QualOSS quality models is addressed in task 1.3. The definition was done top-down as well as bottom-up. The top-down part is addressed by selecting and defining models suitable to meet the previously defined goals, based on a survey on available models as well as of decision makers in industry. This includes existing assessment methods for F/OSS projects, relevant quality models (such as ISO 9126), and on insights from related projects on F/OSS evaluation, such as FlOSSmetrics. In addition, the definition also takes into account available data and tools, as elicited in task 1.1. Figure 1 shows the inputs for task 1.3. In particular, the usage scenarios' goals and requirements were partially elicited through stakehoder interviews.

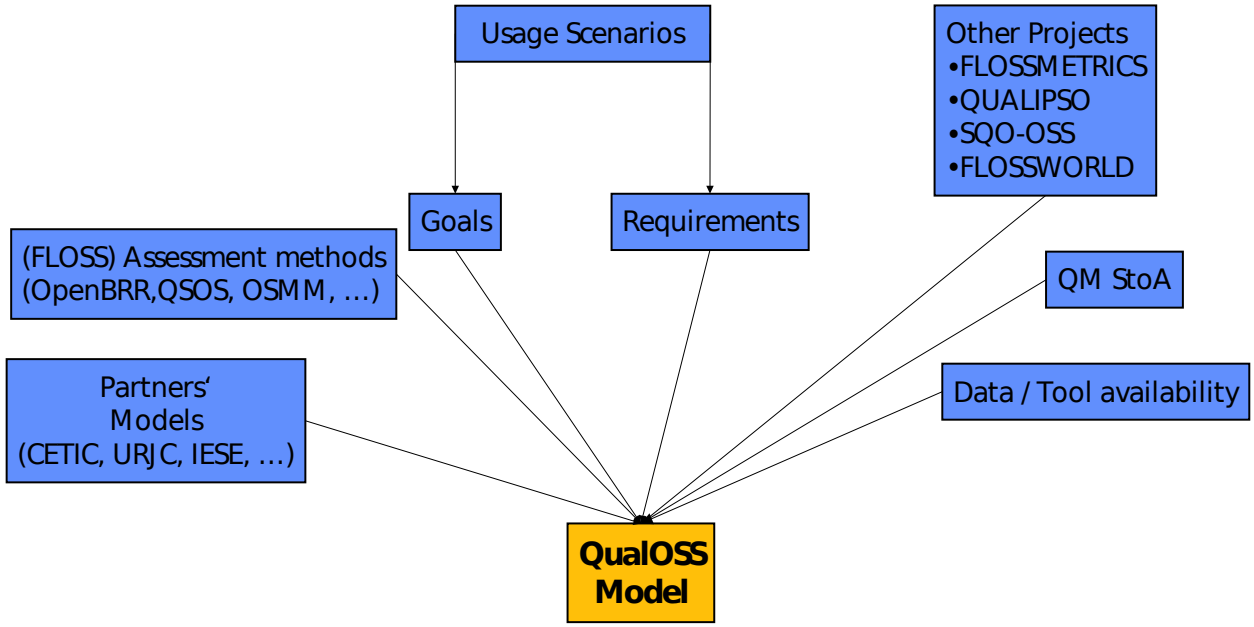


Figure 1: Input Sources for QualOSS model (D 1.3)

(3) Evaluation and calibration of the quality models are addressed in tasks 1.4 to 1.6. Thereby, task 1.4 implements a prototype and repository for data extraction, and uses this prototype to process a set of reference projects. Workpackage 2 will build an advanced set of tools based on the experience gathered in task 1.4. Definition of the interpretation and calibration of the quality models is addressed in task 1.5. More precisely, task 1.5 examines the applicability of the quality models resp. their metrics and tries to find patterns and dependencies (through data mining) in the data that can be used as input to improve the quality models. Task 1.6 validates the quality models through interviews and by measurement of additional projects. This includes, for example, evaluating the definition and prioritization of quality characteristics from stakeholders' viewpoints. Workpackages 4 and 5 pick up on the results of tasks 1.5 and 1.6 by creating advanced quality models and extensively evaluating them.

It is important to note that work in task 1.2 and 1.3 made it clear that we need to restrict D1.2 to definition of robustness and evolvability characteristics. In terms of the goal-question-metric (GQM) paradigm's terminology, these are the measurement goals and questions. The GQM metrics; that is, the definition of appropriate metrics and identification of measurement tools, is part of D1.3. In addition, as product and community aspects need to be considered, and as process maturity is intrinsic to assessing a community, we decided that part of task 1.3 will be to develop an assessment method. The vision of the QualOSS quality model is that all stakeholders use the same definition and metrics to measure robustness and evolvability. What may change depending of the stakeholder's situation, however, is the priority of the quality characteristics. For example, stability of a product is measured in the same way for all products; however, if it is to be used as desktop tool or as part of an external service the company offers, the stability is of different importance to the stakeholder. For this reason, we decided to elicit usage scenarios for F/OSS components. These usage scenarios will later be used to define an initial weighting of the different quality characteristics. The definition of quality characteristics will be independent of the scenario. The challenges that need to be addressed in the QualOSS quality model are missing or inconsistent data; for example.

1.4 Approach

This section describes the approach we took to achieve the goals of Deliverable 1.5.

The goals of D1.5 can be summarized as follows: Defining the interpretation model or user manual through indicators, and calibrating the QualOSS model.

The approach taken in task 1.5 was to define indicators for the different quality characteristics that combine and interpret the different metrics of a quality characteristic into a single metric value, as well as define initial aggregation and abstraction mechanisms. As the data from D1.4 was not sufficient to run statistical analyses for calibration, the indicators had to be defined by experts, and their calibration had to be done by validating the indicators by applying them to the D1.4 measurement. This approach is not optimal, but it is sufficient for the purposes of WP1, as the main goal is to construct a prototype and identify and prioritize issues to be tackled in the remainder of the project. These issues include feasibility of the measurement, and identification of parts of the QualOSS model that are important but have no metrics available.

For the interpretation scale, we have proposed a four-level scale, as described in Section 2. The indicators and the corresponding interpretation were defined on the QualOSS internal wiki system, using a template for defining interpretation rules as well as for systematically capturing experience. These indicators were applied to the projects measured in task 1.4, and the resulting experience added to the indicator discussion on the QualOSS wiki system. In addition, we defined the approach for data mining to be used to construct advanced models, which will be validated in task 1.6.

1.5 Structure of the Deliverable

This document presents initial interpretation and calibration of the QualOSS quality model.

The rest of the deliverable is structured as follows: Section 1 presents an introduction to the goals and approach taken to produce this deliverables. Section 2 introduces the concept of indicators we used to define the interpretation model. Section 3 describes the indicators we defined for the prototype parts of the QualOSS model; that is, the quality characteristics with basic metrics that were measured during D1.4. Section 4 focuses on the results of applying the QualOSS indicators to the projects measured in D1.4. Section 5 describes the data mining approach defined for QualOSS. Section 6 discusses lessons learned and implications for advanced models. Finally, Section 7 summarizes the results of this deliverable and the task 1.5.

Keywords: Free / Open Source Software, quality modelling, process assessment, project assessment, product assessment, evolvability, robustness

2. Interpretation guide: The Indicator concept

This section describes the means we used to define the the interpretation guide.

2.1 What is an indicator?

One of the main goals of the QualOSS project is to provide potential F/OSS users with a means to determine the quality of available products (and of the projects that develop them). Until now, we have defined a number of metrics that, when evaluated on F/OSS products and projects, provide raw data about them. In order to have a high-level idea of the quality of product, however, this raw, low-level data, must be analyzed and integrated to provide a clear view that is easy to understand.

Indicators interpret a set of metrics with respect to a specific quality attribute. An indicator takes the values of one or more metrics as input, and consolidates them into a single value that can be used by decision makers to assess a product's quality regarding the corresponding attribute. For this reason, indicator values should have a form that is easy for general decision makers and users (as opposed to software quality experts) to understand and use.

2.2 The QualOSS indicator scale

For QualOSS, we have decided that our indicators will produce one of only four values. For mnemonic purposes, these values are labeled with the colors in a traffic light-that is, green, yellow and red-with an additional “black” value related to a particularly undesirable state.

In order to define usable indicators, it is important to have a clear, generic definition of the meaning of these values that can be applied objectively to a variety of quality attributes and measurement objects. While the quality attributes encompass the whole QualOSS quality model, measurement objects include the product (code, documentation, etc.), the processes and infrastructure employed by the community, and the community itself. One possible way to do this generic definition is in terms of the effort necessary to improve the quality of the measured object. The basic idea is to define a desired *optimal quality*, that is considered completely satisfactory. Our indicator values are defined in terms of the estimated effort necessary to reach this optimal quality point:

- Green: *No or minor change of measurement object required*. Sizeable existing work or measurement object (e.g., code, or community processes, infrastructure) of sufficient quality is present. Nothing or very few issues need to be solved to achieve an optimal quality level. For classification in this category, required rework should concern at maximum 5% of the total existing work
- Yellow: *Significant rework needed*. Existing work / measurement object has flaws, but is basically useful. Problems can be remedied with (still) significant effort. For an object to be labeled yellow, required rework should concern at least 5% and less than 30% of the total existing work.
- Red: *Critical, needs serious rework*. Existing work / measurement object has serious flaws (i.e., cannot really be used; is beyond "threshold of pain"). However, some of it can be saved, with a still large amount of effort. Quantitatively, Required rework should concern more than 30% and less than 70% of the total existing work for an object to be placed in this category.
- Black: *“Discard” and start from scratch*. None or only a very small portion of the existing work / measurement object (if there is any) can be used in order to reach the desired optimal quality. That is, it is easier and less costly to start from scratch than to try rework existing work. For example, in quantitative terms, if required rework concerns at least 95% of the total existing work, the measurement object should be classified in this level.

Some notes about this definition:

- As already mentioned, the “existing work” can refer to products such as code or documentation,
- The number of levels and their quantitative limits are arbitrary. While deciding how many levels to use, it is important to reach a good trade-off between precision and ease of use. More levels may provide more information to decision makers, but the may also be more confusing. Also, classifying in more levels can be very difficult, specially when our understanding of the metrics is insufficient.

- What the definition calls "existing work" can vary depending on the quality attribute that is being evaluated. Possible measurement objects include products (such as code and documentation), as well as community processes and infrastructure, or the structure and composition of the community itself. Therefore, the existing and remaining work are generally related to the number and extent of the changes that have to made on the object to improve its quality.
- In some cases, indicators depend on context. For example, interoperability can be hardly measured in absolute terms but depends on the user profile, e.g., what are the required exchange formats for a particular user or application.

2.3 From metrics to indicators

As stated above, indicator values have to be derived from the values of the metrics associated to a particular quality attribute. This process is called interpretation. There are two basic approaches to deal with more than one associated metric (for more approaches, see Section 2.5):

1. **Weighted average**: The values of the metrics are normalized (2) (e.g., mapped into a [0, 100] interval), weighted according to their relevance for the evaluated quality attribute, and averaged. The final indicator value is then derived from this average value, by using a simple rule.

2. **Rule-based specification**: In many cases, it will be more natural to specify the interpretation in terms of rules, such as: "if at least one of the lower-level metrics is red, then the status of the higher-level characteristic is red"

The definition of interpretation rules or formulas can be based on expert estimation. If sufficient data are available (e.g, from QualOSS deliverable 1.4) experts should rely on analyzing it (e.g., mean values/quartiles) to define aggregation rules. In many cases, though, the amount of data will not suffice for quantitative analysis.

The first step will be to define indicators for metrics. Aggregation of these indicators towards higher-level characteristics will be done in a second step and is detailed in Section 2.5.

2.4 Defining QualOSS indicators

Based on the previous sections, we can outline a procedure for defining an indicator:

1. Define a clear rule of what "optimal quality" means for this attribute.

2. Provide a concrete interpretation of the four categories - Black, Red, Yellow, and Green - in the context of the concrete attribute and based on the defined optimal quality.

3. Define a rule or formula that interprets the metrics and provides a final value according to the definition made in the previous step.

The following subsections present two examples to illustrate how this can be done. Notice that we created these examples based on actual QualOSS indicators and data, and thus they show some of the problems an expert may run into while defining indicators. We consider that the fact that problems can (and most probably will) be found while defining most indicators is no reason to refrain from doing this work. Going through this effort will not only show us the limitations of our current approach, but will certainly suggest many of the improvements necessary to overcome them.

The indicator template we used to define indicators can be found in Appendix A.

2.4.1 Example 1: Actuality of code documentation

Defining optimal quality in this case is straightforward: quality is optimal with regard to actuality of code documentation, if all code documentation is accurate with respect to the current code version. Notice that you could have high actuality but still low coverage, in which case, actuality may be rather pointless.

We consider the metrics specified by D1.3 for this quality attribute:

- APIDocumentationDateProjectReleaseDateDifference (RelDiff)
- APIDocumentationDateSourceFilesDateDifference (SourceDiff)

- AvgSourceDiff: Weighted average: Sum (SourceDiff*LOC) / Sum(LOC)
- From coverage: SourceCodeCommentsPercentage

APIDocumentationDateSourceFilesDateDifference gives as the difference between file date and documentation date for all C files. For Java, the concepts will be somehow different, as documentation and code are basically stored in the same file. For the sake of convenience, we will call this metric SourceDiff from now on.

We define our quality categories as follows:

- Black:
  - At single-file level: Black if SourceDiff > 2 years.
  - At the system level: Compute the average AvgSourceDiff of the SourceDiff values for all files. Black if AvgSourceDiff > 2 years.
- Red:
  - At single-file level: SourceDiff >= 6 months and < 2 years.
  - At the system level: 6 months <= AvgSourceDiff < 2 years.
- Yellow:
  - At single-file level: SourceDiff > 2 months, < 6 months.
  - At the system level: 2 months <= AvgSourceDiff < 6 months.
- Green:
  - At single-file level: Green if SourceDiff <= 2 months.
  - At the system level: 6 months <= AvgSourceDiff < 2 years.

This definition is mainly limited by the fact that the metrics specified so far seem insufficient to answer the question of whether documentation is current. In particular, the question is not whether the documentation is old, but whether the interface has changed since documentation was last updated. For example, for Java it would be necessary to check whether interface changes were accompanied by changes of the related javadoc documentation. For the moment, we can use this definition as a first approximation, but we should refine this indicator in the future.

2.4.2 Example 2: Interoperability

We start by considering the defined metrics:

- ProductReleaseNumberOfRuntimeExchangeFormats
- ProductReleaseNumberOfStaticExchangeFormats
- ProductReleaseRuntimeExchangeFormatsRatio

The first Problem with these metrics is how to define what the ideal quality state is. In this case, the "ideal" number of exchange formats depends on the application's purpose, e.g., for word processors, a number of open exchange format exists. "Ideal" would be if all of these formats were supported, but this is hardly attainable in practice. How do we find out, for each application, what the "ideal" set of formats should be? The answer is that it depends much on the actual situation, usage context, and user profile (for example lawyers often work with WordPerfect and need support for it.) As a consequence, the interpretation of interoperability needs to be manual and relative to the needs of a particular user.

What we probably really need is list of exchange formats supported by an application together with the degree of support for each format (or rather, which parts of the format are not fully supported). Users then need to prioritize the formats and their support according to their own needs.

This leads us to the following category definition:

- Black: No required exchange formats are supported.
- Red: Required exchange formats exist, but are not sufficiently supported; code needs to be extended
- Yellow: Required exchange formats exist, but some important aspects are not supported; user is able to work around those (e.g., supportEvaluation is yellow for lower-priority formats)
- Green: All required formats are supported to a sufficient degree.

2.5 Interpretation And Aggregation

This section describes the interpretation and aggregation model for QualOSS. Interpretation is concerned with connecting metrics to quality attributes, while aggregation is the process of inferring higher-level interpretations from lower-level ones.

2.5.1 Overview

Interpretation is concerned with mapping a set of metrics to an interpretation scale, *I*. In the QualOSS prototype model, *I* contains four different interpretations (black/red/yellow/green, or short b/r/y/g; see Section 2.2), which are ordered (worst to best) as follows: black, red, yellow, green. The interpretation rule can thus be described as a function that maps n metrics to the interpretation scale:

$$InterpretationRule : M^n \rightarrow I \, , I = \{b \, , r \, , y \, , g\}$$

Where:

- M is a metric on any scale
- I is the set of possible interpretations. For the QualOSS prototype, we use four levels, where max (*I*) = green (g), and min (I) = black (b)

In contrast, aggregation is concerned with combining different interpretations to create a (usually more abstract) interpretation. That is, an aggregation rule can be described as a function that maps n interpretations to a single one:

$$AggregationRule : I^n \rightarrow I \, , I = \{b \, , r \, , y \, , g\}$$

The following Figure 2 shows the interpretation and aggregation model of QualOSS. Indicators comprise a set of metrics, interpretation and aggregation rules, together with an associated quality attribute. The task of interpretation rules is to map its associated metric values to an interpretation (i.e., to the defined indicator scale). The (optional) aggregation rule combines several interpretations through aggregation rules (i.e., in the most simple case, this rule is the identity function). For instance, example 1 in Section 2.4.1 makes use of such an aggregation by inferring the interpretation for actuality of documentation on system-level by the interpretation on file level. However, in the indicators applied in this deliverable, we have not used this option.

Each metric is associated with at least one quality attribute. Correspondingly, it needs at least one interpretation rule to generate an interpretation for the quality attribute. Metrics can aggregate other metrics in an hierarchical fashion by inferring complex metrics out of atomic ones (e.g., $AvgSourceDiff = \sum (SourceDiff * LOC) / \sum LOC$, where AvgSourceDiff aggregates SourceDiff and LOC). An interpretation rule takes n metrics as input and creates an interpretation.

A set of aggregation rules can follow the set of interpretation rules to create an interpretation for the quality attribute. In the most simple case, the aggregation function is the identity; that is, there is only one interpretation rule that combines all input metrics.

While interpretation and aggregation rules define interpretations for “leaf” quality attributes, the interpretation for higher-level quality attributes is derived through aggregation (or abstraction) of the lower-level attributes, or better: their interpretation. To this end, aggregation rules are used again; now, their purpose is to abstract interpretations. Consequently, each higher abstraction layer requires at least one abstraction rule to process one or many lower-level interpretations. As abstraction and aggregation rules have the same purpose (i.e., to combine different interpretations into one), we will refer to aggregation rules only for the remainder of the document.

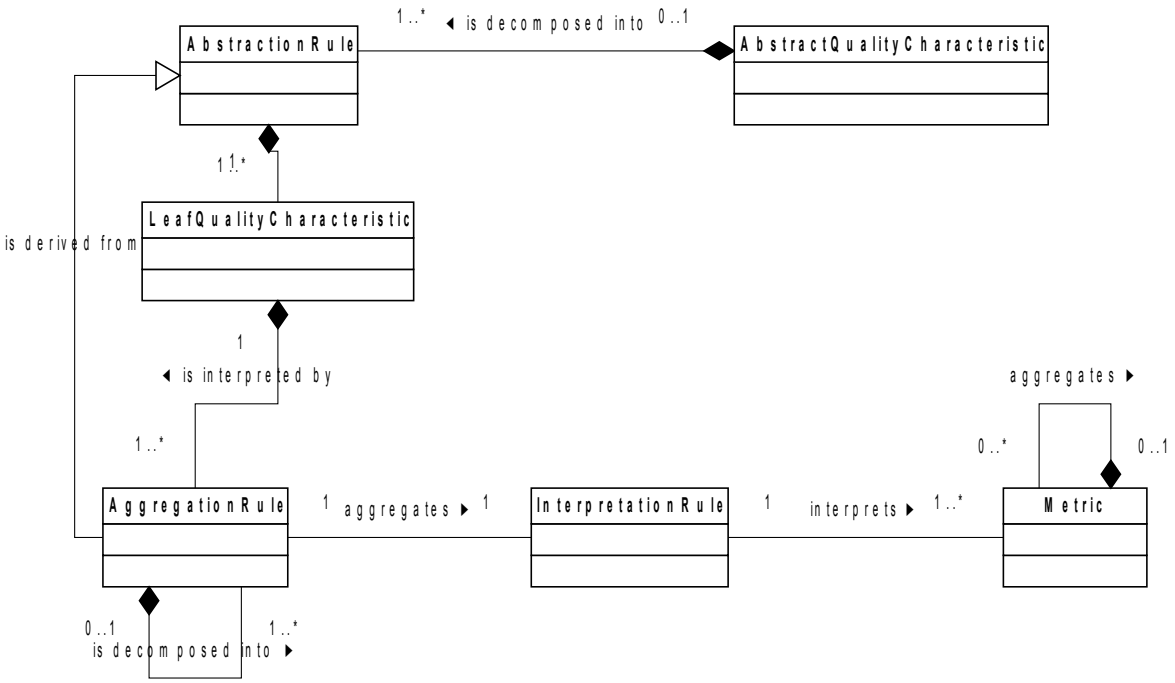
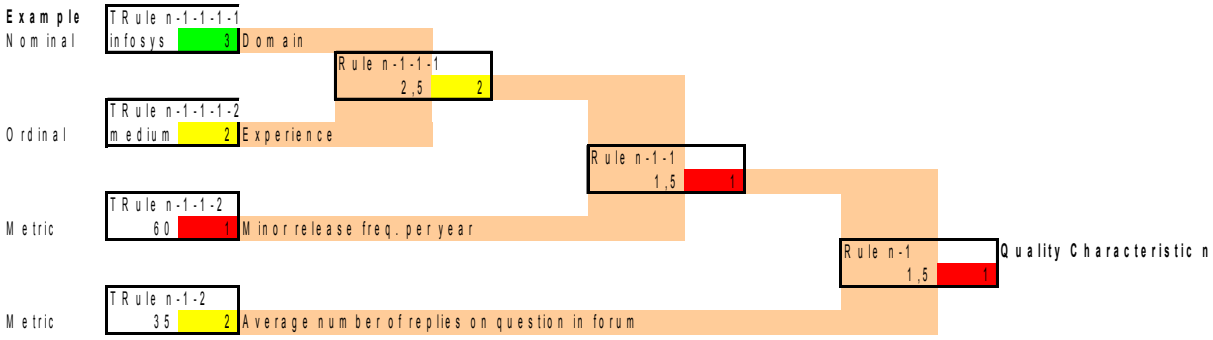


Figure 2: The Interpretation and Aggregation model of the QualOSS interpretation guide.

2.5.2 Example



Aggregation and interpretation rules applied in this case are:

- 1. Interpretation of metrics: 0,1,2,3 for black, red, yellow, green according to individual rules for metric, ordinal, nominal data
- 2. Aggregation by simple averaging

This example uses a total of 7 rules.

2.5.3 Types of interpretation rules

The interpretation rule can be described as a function that maps n metrics to the interpretation scale:

$$InterpretationRule: M^n \rightarrow I, I = \{b, r, y, g\}$$

Where:

- M is a metric on any scale
- I is the set of possible interpretations. For the QualOSS prototype, we use four levels, where max (I) = green (g), and min (I) = black (b)

This section describes approaches for defining interpretation rules, while the next describes possible definitions of aggregation rules.

Interpretation of Metrics on Nominal Scale

In this case, a metric on a nominal scale has to be interpreted. That is, an N:M mapping (N,M=1..n), from the scale to an interpretation value has to be done. Such a mapping can be done in a matrix form.

Figure 3 shows an example of a nominal scaled application domain of open source software under evaluation

Nominal values / Color	b	r	y	g
AUTO			X	
MIL				X
SPACE				X
ACCTNG		X		
BILLING		X		
…				
MGT INFOSYS	X			

Figure 3: Interpretation of nominal scale metrics

Interpretation of Metrics on Ordinal Scale

In this case, a metric on an ordinal scale has to be interpreted. As before, an N:M mapping (N,M=1..n), from the scale to an interpretation value has to be done. However, the difference here is that the mapping is ordered; thus, intervals of ordinal values (e.g. „Seldom“-“Occasionally“) may be mapped to one interpretation value. As before, such a mapping can be done in a matrix form.

Figure 4 gives an example of an ordinal scaled measure of forum usage of persons in the open source community leading the development and evolution of the software

Ordinal Values / Color	b	r	y	g
Never	X			
Seldom		X		
Infrequently		X		
Occasionally		X		
Frequently			X	
Quite often				X
All the time				X

Figure 4: Interpretation of ordinal scale metrics

Interpretation of Metrics on Metric Scale

In this case, a metric on a nominal scale has to be interpreted. Metric scales are interval, rational, and absolute scales. That is, in this case, intervals for mapping the scale to interpretation values have to be defined. Such a mapping can be done in functional form.

Figure 5 gives an example of a metric-scaled (absolute scaled) measure of average response time of the open source community to new discovered problems / bugs with the open source software

$$InterpretationRule(RespTime) := \begin{cases} g \Leftrightarrow \text{RespTime} \in [0..3] \text{days} \\ y \Leftrightarrow \text{RespTime} \in]3..7] \text{days} \\ r \Leftrightarrow \text{RespTime} \in]7..12] \text{days} \\ b \Leftrightarrow \text{RespTime} \in]12..\infty] \text{days} \end{cases}$$

Figure 5: Interpretation of metric scales

2.5.4 Types of Aggregation Rules

Aggregation is concerned with combining different interpretations to create a (usually more abstract) interpretation. That is, an aggregation rule can be described as a function that maps n interpretations to a single one:

$$AggregationRule : I^n \rightarrow I,\ I = \{b, r, y, g\}$$

This section describes approaches to define aggregation rules. Basically, two approaches can be applied: arithmetical and statistical computations, and multi-criteria decision making, which uses rule-based criteria to combine inputs.

To allow arithmetical and statistical computations, the interpretation needs to be mapped to a rational scale. As the interpretation used by the QualOSS prototype uses a four-point ordinal scale, the interpretation values need to be mapped to integer values. After an aggregation function is used, the result needs to be interpreted again:

$$Assign : I \rightarrow \mathbb{R},\ Assign(i) := \begin{cases} 0 \Leftrightarrow i = b \\ 1 \Leftrightarrow i = r \\ 2 \Leftrightarrow i = y \\ 3 \Leftrightarrow i = g \end{cases},\ i \in I$$

$$Interpretation : \mathbb{R} \rightarrow I,\ Interpret(v) = Assign^{-1}(v) := \begin{cases} b \Leftrightarrow v \in [0 \ldots 0.75[\\ r \Leftrightarrow v \in [0.75 \ldots 1.75[\\ y \Leftrightarrow v \in [1.75 \ldots 2.75[\\ g \Leftrightarrow v \in [2.75 \ldots \infty[\end{cases},\ v \in \mathbb{R}$$

Note: the interval borders used in $Assign^{-1}$ may be changed to context-specific needs

Arithmetic & statistical rules

If there are more than two interpretations to combine the main arithmetic and statistical approaches to be applied are:

- Min or Max of values
- Mean of values
- Median of values
- Mode of values
- Weighted scoring method: $\sum_i w_i * v_i,\ where \sum_i w_i = 1$

$$AggregationRule(i_{1,}\dots,i_n)=Min\{i_{1,}\dots,i_n\}$$
$$AggregationRule(i_{1,}\dots,i_n)=Max\,i_{1,}\dots,i_n\}$$
$$AggregationRule(i_{1,}\dots,i_n)=Median\{i_{1,}\dots,i_n\}$$
$$AggregationRule(i_{1,}\dots,i_n)=Mode\{i_{1,}\dots,i_n\}$$

$$\text{Aggregation}(i_1,..,i_n)=Interpret\left(\frac{1}{n}\sum_{j=1}^{n}\text{Assign}(i_j)\right)$$

Weighted Scoring Method:

$$\text{Aggregation}(i_1,..,i_n)=Interpretation\left(\sum_{j=1}^{n}\text{weight}(i_j)\times Assign(i_j)\right),$$

$$\text{where}\sum_{i=1}^{n}\text{weight}(i_j)=1$$

In the case where there are two interpretations to aggregate, the (weighted) average can be expressed through a simple optimism/pessimism rule of the form $\alpha v+(1-\alpha)w$:

$$\text{Aggregation}(i_1,i_2)=Mapping^{-1}\left(\alpha\times Assign(i_1)+(1-\alpha)\times Assign(i_2)\right),\ \text{where}\ \alpha\in[0..1]$$

Multi-Criteria Decision Making (MCDM) rules

Multi-Criteria decision making rules aggregate their inputs through rule-based approaches. Different possibilities include the following basic approaches and combinations thereof:

- Conjunctive method: inputs are combined through logical AND
- Disjunctive method: inputs are combined through logical OR
- Relaxed/hybrid Conjunctive/disjunctive method
- Methods to identify & define weights: Eigenvector Method (from AHP) (Reference: [Saaty: „The Analytic Hierarchy Process"])
- Intervalling Method (Reference: [Eisenführ/Weber: „Rationales Entscheiden"])
- Delphi Method (Reference: [„The Annals of Operations Research"])

Examples for rule-based aggregation:

$$Aggregation(i_{1,}\dots,i_n):=\begin{cases}g\Leftrightarrow\wedge_{j=1}^{n}(i_j=g)\\ y\Leftrightarrow(\wedge_{j\in S}(i_j=g))\wedge(\wedge_{j\notin S}(i_j\notin\{r,b\}))\\ r\Leftrightarrow\left|(\cup_{j\in S}(i_j=g))\right|\geq T\wedge(\wedge_{j\notin S}(i_j\neq b\}))\\ b\Leftrightarrow\left|(\cup_{j\in S}(i_j=g))\right|<T\vee_{j=1}^{n}(i_j=b)\end{cases},\ where\ i_j\in I,S\subset\{1..n\},T\leq|S|$$

- Green: All input interpretations i_j are green
- Yellow: All i_j of a specific set of inputs (S) are green, the rest may be yellow but not red or black
- Red: At least T i_j of a specific set of inputs (S) are green, the rest may not be black
- Black: Less than T i_j of a specific set of inputs (S) are green, one input is black

3. Indicators for QualOSS model

This section contains the indicators defined for the QualOSS prototype model, as of Monday, October 22nd, 2007. The indicators were defined using the QualOSS internal trac system and its associated wiki pages. Trac is a light weight project management tool that associate a wiki and a ticketing system. The definitions will be evolved on the wiki pages; thus, this deliverable represents a snapshot of the QualOSS work. For the prototype model, the interpretation rules defined for the indicators are sometimes expressed as logic or pseudo-code. As the purpose was to evaluate the feasibility of the indicators, this heterogeneity is currently sufficient. However, for the advanced models, the interpretation rules will have to be formalized.

3.1 Actuality (Usefulness of Code Documentation)

3.1.1 Contact person

Martin (IESE)

3.1.2 Quality level definition

Defining optimal quality in this case is straightforward: quality is optimal with regard to actuality of code documentation, if all code documentation is accurate with respect to the current code version. Notice that you could have high actuality but still low coverage, in which case, actuality may be rather pointless.

We consider the metrics specified by D1.3 for this quality attribute: (4) APIDocumentationDateProjectReleaseDateDifference (RelDiff?) APIDocumentationDateSourceFilesDateDifference (SourceDiff?) AvgSourceDiff?: Weighted average: Sum (SourceDiff?*LOC) / Sum(LOC) From coverage: SourceCodeCommentsPercentage? APIDocumentationDateSourceFilesDateDifference gives as the difference between file date and documentation date for all C files. For Java, the concepts will be somehow different, as documentation and code are basically stored in the same file. For the sake of convenience, we will call this metric SourceDiff? from now on.

Black

Black means the documentation is completely out of date and its use will be hasardous.

Red

Red means the actuality of documentation is old and its use necessitate carefull verification with respect to its actuality

Yellow

Yellow means the documentation is no more actual but it can be used with certain caution.

Green

Green means the documentation is actual, only slight and limited changes could be undocumented.

3.1.3 Metrics

Metrics from D1.3

We consider the metrics specified by D1.3 for this quality attribute: (4) APIDocumentationDateProjectReleaseDateDifference (RelDiff?) APIDocumentationDateSourceFilesDateDifference (SourceDiff?) From coverage: SourceCodeCommentsPercentage? APIDocumentationDateSourceFilesDateDifference gives as the difference between file date and documentation date for all C files. For Java, the concepts will be somehow different, as documentation and code are basically stored in the same file. For the sake of convenience, we will call this metric SourceDiff? from now on.

New required metrics

AvgSourceDiff?: Weighted average: Sum (SourceDiff?*LOC) / Sum(LOC)

APIDocumentationDateSourceFilesDateDifference gives as the difference between file date and documentation date for all C files. For Java, the concepts will be somehow different, as documentation and code are basically stored in the same file. For the sake of convenience, we will call this metric SourceDiff? from now on.

Unused metrics from D1.3

Missing

3.1.4 Indicator evaluation

First, compute the average AvgSourceDiff? of the source diff values for all files.

Black

At single-file level: Black if SourceDiff? > 24 months.
At the system level: Black if AvgSourceDiff? > 24 months.

Red

At single-file level: 6 months <= SourceDiff? < 24 months.
At the system level: 6 months <= AvgSourceDiff? < 24 months.

Yellow

At single-file level: 2 months < SourceDiff? < 6 months.
At the system level: 2 months <= AvgSourceDiff? < 6 months.

Green

At single-file level: Green if SourceDiff? <= 2 months.
At the system level: 6 months <= AvgSourceDiff? < 24 months.

3.2 Coverage (Usefulness of Code Documentation)

3.2.1 Contact person

Flora Kamseu, Naji Habra (FUNDP)

3.2.2 Quality level definition

As defined by Deliverable 1.2, Coverage represents the ratio between size of documented code and general product code size. It is useful to know about the difference from the the size of the documented code and the size of the general product code size. From this, we can notice that we can have high coverage but a low actuality. Il will be important to have them both high. Then, we will also use metrics from Actuality. Therefore, here are metrics to consider: SourceCodeCommentsPercentage? (SourceComPer?) APIDocumentationDateSourceFilesDateDifference (SourceDiff?) AvgSourceDiff?: Weighted average: Sum (SourceDiff?*LOC) / Sum(LOC)

Black

At single-file level: Black if SourceDiff? > 2 years and SourceComPer? < 20%.

At the system level: Compute the average AvgSourceDiff? of the source diff values for all files. Black if AvgSourceDiff? > 2 years and SourceComPer? < 20% .

Red

* At single-file level: SourceDiff? >= 6 months and < 2 years and SourceComPer? >= 20% and < 40%.

* At the system level: 6 months <= AvgSourceDiff? < 2 years and SourceComPer? >= 20% and < 40%.

Yellow

* At single-file level: SourceDiff? > 2 months, < 6 months and SourceComPer? >= 40% and < 60%.

* At the system level: 2 months <= AvgSourceDiff? < 6 months and SourceComPer? >= 40% and < 60%.

Green

* At single-file level: Green if SourceDiff? <= 2 months and SourceComPer? >= 60%.

* At the system level: 6 months <= AvgSourceDiff? < 2years and SourceComPer? >= 60%.

3.2.3 Metrics

Metrics from D1.3

* SourceCodeCommentsPercentage?

* From Actuality : APIDocumentationDateSourceFilesDateDifference (SourceDiff?)

New required metrics

* AvgSourceDiff?: Weighted average: Sum (SourceDiff?*LOC) / Sum(LOC)

Unused metrics from D1.3

Missing

3.2.4 Indicator evaluation

Black

* At single-file level: Black if SourceDiff? > 2 years and SourceComPer? < 20%.

* At the system level: Compute the average AvgSourceDiff? of the source diff values for all files. Black if AvgSourceDiff? > 2 years and SourceComPer? < 20% .

Red

* At single-file level: SourceDiff? >= 6 months and < 2 years and SourceComPer? >= 20% and < 40%.

* At the system level: 6 months <= AvgSourceDiff? < 2 years and SourceComPer? >= 20% and < 40%.

Yellow

* At single-file level: SourceDiff? > 2 months, < 6 months and SourceComPer? >= 40% and < 60%.

* At the system level: 2 months <= AvgSourceDiff? < 6 months and SourceComPer? >= 40% and < 60%.

Green

* At single-file level: Green if SourceDiff? <= 2 months and SourceComPer? >= 60%.

* At the system level: 6 months <= AvgSourceDiff? < 2years and SourceComPer? >= 60%.

3.3 Code Documentation Standard Compliance (Usefulness of Code Documentation)

3.3.1 Contact person

Flora Kamseu, Naji Habra (FUNDP)

3.3.2 Quality level definition

Quality is optimal in term of Code Documentation Standard Compliance if while writting the different type of documentation, authors take into account existing standard norms. Here is the metric used :

- APICommentsErrorsAverage (AComErrAvg)

Black
Black if AcomErrAvg? is high

Red
Red if AcomErrAvg? is medium-high

Yellow
Yellow if AcomErrAvg? is low

Green
Green AcomErrAvg? is very low

3.3.3 Metrics

Metrics from D1.3

- APICommentsErrorsAverage (AComErrAvg)

APICommentsErrorsAverage is *the Ratio between the number of errors encountered in documentation comments respecting the standards and the total number of documentation comments.*

New required metrics
Missing

Unused metrics from D1.3
Missing

3.3.4 Indicator evaluation

Black
Black if AcomErrAvg? is high

Red
Red if AcomErrAvg? is medium-high

Yellow
Yellow if AcomErrAvg? is low

Green
Green AcomErrAvg? is very low

3.4 Actuality (Usefulness of User Documentation)

3.4.1 Contact person

Flora Kamseu, Naji Habra (FUNDP)

3.4.2 Quality level definition

Quality is optimal regarding actuality of user documentation if documentation related to the different type of users is accurate with respect to the current code version. The main idea is to have an actual documentation of the software product. It will be useful to have also a real coverage of the functionalities of the software product. Therefore, we proposed to insert coverage of user documentation.

The used metrics are :

- UserDocumentationDateProjectReleaseDateDifference? (UDocRealDiff)
- From coverage (Usefulness of User documentation) :UserDocumentationAPIDocumentationCommonAbstractionsPercentage (UDocComAbsPer)

Black

Black if UDocRealDiff > 3 years and UDocComAbsPer is high.

Red

Red if UDocRealDiff > 2 years and <=3 and UDocComAbsPer is high

Yellow

Yellow if UDocRealDiff > 1 year and <= 2 years and UDocComAbsPer is low.

Green

Green if UDocRealDiff <= 1 year and UDocComAbsPer is low.

3.4.3 Metrics

Metrics from D1.3

- UserDocumentationDateProjectReleaseDateDifference?
- From coverage (Usefulness of User documentation) : UserDocumentationAPIDocumentationCommonAbstractionsPercentage

UserDocumentationDateProjectReleaseDateDifference? is the difference between the date of the user documentation and the date of the project release.
UserDocumentationAPIDocumentationCommonAbstractionsPercentage is the Ratio between the total number of abstractions found in the user documentation in common with the abstractions found in the technical API documentation and the total number of abstractions found in the technical API documentation of the product release.

New required metrics

Missing

Unused metrics from D1.3

Missing

3.4.4 Indicator evaluation

Black

Black if UDocRealDiff > 3 years and UDocComAbsPer is high.

Red

Red if UDocRealDiff > 2 years and <=3 and UDocComAbsPer is high

Yellow
Yellow if UDocRealDiff > 1 year and <= 2 years and UDocComAbsPer is low.

Green
Green if UDocRealDiff <= 1 year and UDocComAbsPer is low.

3.5 Coverage (Usefulness of User Documentation)

3.5.1 Contact person

Flora Kamseu, Naji Habra (FUNDP)

3.5.2 Quality level definition

Quality is optimal for the coverage regarding the usefulness of user documentation if the documentation related to user is complete and accurate in term of the documenation of the software product. For this, we have to consider if the documentation is upgrade and feets the user's needs. We then take into account the actuality part of the documenation.

We will used these metrics :

- UserDocumentationAPIDocumentationCommonAbstractionsPercentage (UDocComAbsPer)
- From Actuality : UserDocumentationDateProjectReleaseDateDifference? (UdocRealDiff?)

Black
Black if UDocRealDiff > 3 years and UdocComAbsPer? is high.

Red
Red if UdocRealDiff? > 2 years and <=3 and UdocComAbsPer? is high.

Yellow
Yellow if UdocRealDiff? > 1 year and <= 2 years and UdocComAbsPer? is low.

Green
Green if UdocRealDiff? <= 1 year and UdocComAbsPer? is low.

3.5.3 Metrics

Metrics from D1.3

- UserDocumentationAPIDocumentationCommonAbstractionsPercentage (UDocComAbsPer)
- From Actuality : UserDocumentationDateProjectReleaseDateDifference? (UdocRealDiff?)

UserDocumentationAPIDocumentationCommonAbstractionsPercentage is the Ratio between the total number of abstractions found in the user documentation in common with the abstractions found in the technical API documentation and the total number of abstractions found in the technical API documentation of the product release. UserDocumentationDateProjectReleaseDateDifference? is the difference between the date of the user documentation and the date of the project release.

New required metrics
Missing

Unused metrics from D1.3
Missing

3.5.4 Indicator evaluation

Black
Black if UDocRealDiff > 3 years and UdocComAbsPer? is high.

Red
Red if UdocRealDiff? > 2 years and <=3 and UdocComAbsPer? is high.

Yellow
Yellow if UdocRealDiff? > 1 year and <= 2 years and UdocComAbsPer? is low.

Green
Green if UdocRealDiff? <= 1 year and UdocComAbsPer? is low.

3.6 Internationalization (Usefulness of user documentation)

3.6.1 Contact person

Flora Kamseu, Naji Habra (FUNDP)

3.6.2 Quality level definition

Quality Internationalization is optimal if we have a high number of languages in which the User's documentation is correctly translated. The main idea is to have a high number and a diversity of users around the world. The basic metric used hee is :

- NumberOfUserDocumentationTranslations? (NumDocTrans?)

Black
Black if NumDocTrans? =0

Red
Red if NumDocTrans? = 1

Yellow
Yellow if NumDocTrans? = 2

Green
Green if NumDocTrans? = 3

3.6.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.6.4 Indicator evaluation

Black
Black if NumDocTrans? =0

Red
Red if NumDocTrans? = 1

Yellow
Yellow if NumDocTrans? = 2

Green
Green if NumDocTrans? = 3

3.7 User Documentation Standard Compliance (Usefulness of User Documentation)

No basic metrics identified so far.

3.7.1 Contact person

Flora Kamseu, Naji Habra (FUNDP)

3.7.2 Quality level definition

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.7.3 Metrics

No basic metrics identified so far; the documentation assessment framework addresses this question

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.7.4 Indicator evaluation

Missing

3.8 Product Complexity

3.8.1 Contact person

Martin Soto (IESE)

3.8.2 Quality level definition

The degree to which a system or component has a design or implementation that is difficult to understand and verify.

Compute Average Level Value (ALV) as the average of the level values computed for all considered metrics (see Indicator Evaluation below.)

Black

ALV < 1.5

Red

1.5 <= ALV < 2.8

Yellow

2.8 <= ALV < 3.5

Green

ALV >= 3.5

3.8.3 Metrics

Metrics from D1.3

- HotFilesPercentage?, HotPackagesPercentage?, HotClassesPercentage?, HotMethodsPercentage?
- MethodLinesOfCodeAverage?
- ClassNumberOfMethodsAverage?

New required metrics

None

Unused metrics from D1.3

- FileCyclomaticComplexityAverage?, PackageCyclomaticComplexityAverage?, ClassCyclomaticComplexityAverage?, MethodCyclomaticComplexityAverage?: The metrics based on counting "hot" elements also use cyclomatic complexity and are probably as significant as this set.
- MethodUnderstandabilityAverage?: The value of this metric is very hard to interpret intuitively. Unless experimental information is available, it is difficult to tell which value ranges are "good" or "bad".

Indicator evaluation

We define separate level values for each one of the metrics as follows:

For the Hot* set of metrics:

- Level value 1: `value > 50%`
- Level value 2: `20% < value <= 50%`
- Level value 3: `5% < value <= 20%`
- Level value 4: `value <= 5%`

For MethodLinesOfCodeAverage?:

- Level value 1: `value > 20`
- Level value 2: `12 < value <= 20`
- Level value 3: `8 < value <= 12`
- Level value 4: `value <= 8`

For ClassNumberOfMethodsAverage?:

- Level value 1: `value > 15`
- Level value 2: `9 < value <= 15`
- Level value 3: `6 < value <= 9`
- Level value 4: `value <= 6`

The final level is evaluated as explained above based on the average of the level values of these metrics.

3.9 Architecture Flexibility

3.9.1 Contact person

Martin Soto (IESE)

3.9.2 Quality level definition

The ability of the product's architecture of being applied to new problems. The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed. (Note: Architecture flexibility includes the notion of extensibility, which is defined as the possibility of extending the architecture through external code modules (add-ons, plug-ins) that do not require modifying the program's core. The ease with which a system or component can be modified to increase its storage or functional capacity.

Black

None of the four basic metrics (DeveloperDocumentationExistence?, APIDocumentationExistence, ThirdPartyPlugInPossibility?, ProductConfigurationFilePossibility?) is true.

Red

Only one or two of the four basic metrics (DeveloperDocumentationExistence?, APIDocumentationExistence, ThirdPartyPlugInPossibility?, ProductConfigurationFilePossibility?) are true.

Yellow

Three of the basic metrics (DeveloperDocumentationExistence?, APIDocumentationExistence, ThirdPartyPlugInPossibility?, ProductConfigurationFilePossibility?) are true.

Green

All of the basic metrics (DeveloperDocumentationExistence?, APIDocumentationExistence, ThirdPartyPlugInPossibility?, ProductConfigurationFilePossibility?) are true.

3.9.3 Metrics

Metrics from D1.3

- DeveloperDocumentationExistence?
- APIDocumentationExistence
- ThirdPartyPlugInPossibility?
- ProductConfigurationFilePossibility?

New required metrics

Missing

Unused metrics from D1.3

Missing

3.9.4 Indicator evaluation

Missing

3.10 Product Buildability

3.10.1 Contact person

Martin Soto (IESE)

3.10.2 Quality level definition

The degree to which a system or component can be rebuild after modifications to the source.

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.10.3 Metrics

Metrics from D1.3
No basic metrics are available for this indicator. For this reason, it cannot be computed during the prototype phase.

New required metrics
Missing

Unused metrics from D1.3
Missing

3.10.4 Indicator evaluation

Missing

3.11 Fixability

3.11.1 Contact person

Martin Soto (IESE)

3.11.2 Quality level definition

The ease with which a software product can be fixed.

Black
IssueOpenToCloseTimeAverage > 60 days

Red
25 days < IssueOpenToCloseTimeAverage <= 60 days

Yellow
10 days < IssueOpenToCloseTimeAverage <= 25 days

Green
IssueOpenToCloseTimeAverage <= 10 days

3.11.3 Metrics

Metrics from D1.3
IssueOpenToCloseTimeAverage?

New required metrics
Missing

Unused metrics from D1.3
Missing

3.11.4 Indicator evaluation

As explained in the levels above.

3.12 Maintainability standard compliance

3.12.1 Contact person

Martin Soto (IESE)

3.12.2 Quality level definition

The degree to which a product complies with published standards relevant to maintainability.

Black
ProductNamingConventionErrorsRate > 1/10

Red
1/20 < ProductNamingConventionErrorsRate <= 1/10

Yellow
1/100 < ProductNamingConventionErrorsRate <= 1/20

Green
ProductNamingConventionErrorsRate <= 1/100

3.12.3 Metrics

Metrics from D1.3
- ProductNamingConventionErrorsPercentage?

New required metrics
- ProductNamingConventionErrorsRate?: Ratio between the number of code style errors related to naming conventions and the total number of lines of code.

Unused metrics from D1.3
- ProductNamingConventionErrorsPercentage?: This metric relates errors breaking naming conventions to the total of style errors. It is difficult to interpret in order to evaluate maintainability.

3.12.4 Indicator evaluation

As explained in the levels above.

3.13 Runtime Interoperability

(should be better renamed to InUseInteroperability?)

The Interoperability is the degree to which a software product can interoperate/interact with other software product either live or based on input/output data.
The Runtime Interoperability is the interoperability with other software products while in operation.

3.13.1 Contact person

FFM (CETIC)

3.13.2 Quality level definition

- This indicator qualifies the quality of the interactions with other softwares.
- The level should be based on the list of formats provided by the user to be somewhat useful.
- Indeed, the fact that the assessed application uses proprietary exchange formats could be the feature expected by the usser.

Black
The product runs standalone, is not meant to communicate with any other application or does not use any known open standard exchange format

Red
The product can communicate with other applications through proprietary exchange formats and no open standard exchange formats are not used.

Yellow
The product can communicate with other applications through at least one open standard exchange format.

Green
The product can communicate with other applications through various known open standard exchange formats.

3.13.3 Metrics

Metrics from D1.3

- ProductReleaseNumberOfOpenExchangeFormats
- ProductReleaseNumberOfCommunicatingApplications

New required metrics
We need to define the list of OpenExchangeFormats? and compute the metric WeightedExchangedFormatsCompliance? as the number of exchange formats effectively matching the required exchanged formats expressed by the user

Unused metrics from D1.3

- ProductReleaseNumberOfRuntimeExchangeFormats
- ProductReleaseNumberOfStaticExchangeFormats

3.13.4 Indicator evaluation

Black
ProductReleaseNumberOfCommunicatingApplications == 0
and ProductReleaseNumberOfOpenExchangeFormats == 0

Red

ProductReleaseNumberOfCommunicatingApplications > 0
and ProductReleaseNumberOfOpenExchangeFormats == 0

Yellow

ProductReleaseNumberOfCommunicatingApplications > 0
and ProductReleaseNumberOfOpenExchangeFormats > 0

Green

Yellow and ProductReleaseNumberOfOpenExchangeFormats > 2

3.14 Passive Interoperability

(better called titled Interoperability)

3.14.1 Contact person

FFM (CETIC)

3.14.2 Quality level definition

Missing

Black

The product is standalone and not supposed to communicate with any other application or does not declare the use of any known open standard exchange format

Red

The product is planned to communicate with other applications in some future version and to use any known open standard exchange format.

Yellow

The product declares its ability to communicate with other applications but does not mention any standard open exchange format.

Green

The product declares its ability to communicate with other applications and the interfaces for the open exchange formats are clearly described.

3.14.3 Metrics

Metrics from D1.3

- DocumentationInteroperabilityPresence
- ProductReleaseNumberOfExchangedFormats
- ProductReleaseNumberOfOpenExchangeFormats
- ProductReleaseNumberOfCommunicatingApplications

New required metrics

Missing

Unused metrics from D1.3

Missing

3.14.4 Indicator evaluation

Black

DocumentationInteroperabilityPresence == 0
and ProductReleaseNumberOfOpenExchangeFormats == 0
and ProductReleaseNumberOfExchangedFormats == 0
and ProductReleaseNumberOfCommunicatingApplications == 0

Red

not Black and DocumentationInteroperabilityPresence != 0

Yellow

Red and ProductReleaseNumberOfCommunicatingApplications != 0

Green

Yellow and ProductReleaseNumberOfOpenExchangeFormats != 0

3.15 Platform specificity

The degree to which a product's code is specific to a particular hardware or software environment

3.15.1 Contact person

FFM (CETIC)

3.15.2 Quality level definition

Black

The application is targeted to be used on a single platform and uses specific libraries.

Red

The application can be used on different platforms but the source code uses only platform-dependent libraries/components on each platform.

Yellow

The application is designed to run on different platforms and the source code uses a number of plaftorm-dependent libraries/components.

Green

The application is designed to run on different platforms and the source code uses only standard libraries/components.

3.15.3 Metrics

Metrics from D1.3

- ProductReleaseHighlyPortableProgrammingLanguageUsed
- ProductReleaseUseOfStandardLibrariesPercentage*
- ProductReleaseUseOfSpecificLibrariesPercentage* (better renamed ProductReleaseUseOfPlatformDependentLibrariesPercentage)

New required metrics

NumberOfPlatformsSupported: Number of platforms on which the application can run

Unused metrics from D1.3
ProductReleaseHighlyPortableProgrammingLanguageUsed

3.15.4 Indicator evaluation

Black
NumberOfPlatformsSupported == 1
and ProductReleaseUseOfSpecificLibrariesPercentage== 0

Red
NumberOfPlatformsSupported > 1
and ProductReleaseUseOfSpecificLibrariesPercentage== 100%

Yellow
NumberOfPlatformsSupported > 1

Green
NumberOfPlatformsSupported > 1
and ProductReleaseUseOfStandardLibrariesPercentage == 100%

3.16 Portability Standard Compliance

3.16.1 Contact person

FFM (CETIC)

3.16.2 Quality level definition

Black
The source code of the application is above a VERY HIGH rate of code style errors per line of code

Red
The source code of the application has a HIGH rate of code style errors per line of code

Yellow
The source code of the application has a MIDDLE rate of code style errors per line of code

Green
The source code of the application is below a LOW rate of code style errors per line of code

3.16.3 Metrics

Metrics from D1.3
ProductReleaseCodeStyleErrorsAverage

New required metrics
Missing

Unused metrics from D1.3
Missing

3.16.4 Indicator evaluation

Black
ProductReleaseCodeStyleErrorsAverage >= VERYHIGH

Red
HIGH <= ProductReleaseCodeStyleErrorsAverage < VERYHIGH

Yellow
LOW <= ProductReleaseCodeStyleErrorsAverage < HIGH

Green
ProductReleaseCodeStyleErrorsAverage < LOW

3.17 UserCommunitySize

3.17.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.17.2 Quality level definition

D1.3 definition: The number of users (individuals and organizations) that use a F/OSS product worldwide.

Black
<3 (NumOfDevelopers)

<6 (NumOfPostersMailingLists)

Red
<10 (NumOfDevelopers)

<20 (NumOfPostersMailingLists)

Yellow
<30 (NumOfDevelopers)

<60 (NumOfPostersMailingLists)

Green
>30 (NumOfDevelopers)

>60 (NumOfPostersMailingLists)

3.17.3 Metrics

NumOfDevelopers? NumOfPostersMailingLists?

Metrics from D1.3
NumOfDevelopers? NumOfPostersMailingLists?

New required metrics
Missing

Unused metrics from D1.3
Missing

3.17.4 Indicator evaluation

This indicator could be more useful if measured over time. This is, if size is X and it was X last year as well, it shows an stagnation of the community.

3.18 StrategicImportance/Mission? criticality

3.18.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.18.2 Quality level definition

D1.3 definition: The extent to which users of a product apply it to mission-critical tasks. Alternatively, the degree to which users of a product depend on the product for reaching their business goals.

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.18.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.18.4 Indicator evaluation

3.19 LicensePermissiveness

3.19.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.19.2 Quality level definition

D1.3 definition: The amount of freedom allowed to product users by the product's licence.

Black
Non free software

Red
Restrictive clauses

Yellow
GPL Family LGPL Family

Green
BSD Family

3.19.3 Metrics

Metrics from D1.3
LicenseUsedSourceCode? LicenseUsedDocumentation?

New required metrics
Missing

Unused metrics from D1.3
Missing

3.19.4 Indicator evaluation

Missing

3.20 DeveloperCommunitySize

3.20.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.20.2 Quality level definition

D1.3 definition: The number of individuals and organizations actively contributing to a product's development over a certain period of time.

Black
>5 (TotalNumOfDevelopers)

<3 (TotalNumOfActiveDevelopers)

Decreasing function (EvolutionNumOfDevelopers and EvolutionNumOfActiveDevelopers)

Red
>20 (TotalNumOfDevelopers)

>10 (TotalNumOfActiveDevelopers)

Constant function (EvolutionNumOfDevelopers and EvolutionNumOfActiveDevelopers)

Yellow
>75 (TotalNumOfDevelopers)

>20 (TotalNumOfActiveDevelopers)

Growing function (EvolutionNumOfDevelopers and EvolutionNumOfActiveDevelopers)

Green
>100 (TotalNumOfDevelopers)

>30 (TotalNumOfActiveDevelopers)

More than linear function (EvolutionNumOfDevelopers and EvolutionNumOfActiveDevelopers)

3.20.3 Metrics

Metrics from D1.3

TotalNumOfDevelopers? EvolutionNumOfDevelopers? TotalNumOfActiveDevelopers? EvolutionNumOfActiveDevelopers?

New required metrics

Missing

Unused metrics from D1.3

PastNumOfDevelopers? TotalNumOfNonActiveDevelopersInPast?

3.20.4 Indicator evaluation

3.21 DeveloperCommunityActivity

3.21.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.21.2 Quality level definition

D1.3 definition: The general number and size of the contributions made to a product's development over a certain period of time.

Black

< 10 changes to source in a period of time AND < 5% of developers replying

Red

> 30 changes to source in a period of time AND 15% of developers replying

Yellow

> 150 changes to source in a period of time AND > 40% of developers replying

Green

> 300 changes to source (including not only source code) in a period of time

AND

> 50% of developers replying questions in mailing lists.

3.21.3 Metrics

Metrics from D1.3

NumOfChangesToSource?, EvolutionOfChangesToSource?, NumOfMessagesOfDevelopers?, EvolutionMessagesOfDevelopers?

New required metrics

Missing

Unused metrics from D1.3

TotalNumOfDevelopers?

3.21.4 Indicator evaluation

Missing

3.22 DeveloperCommunityHeterogeneity

3.22.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.22.2 Quality level definition

D1.3 definition: The degree to which different types of developers (e.g., individuals vs. organizations, for-profit vs. non-for-profit organizations, hobbyists vs. paid professionals) are present in a developer community.

Black

Documentation = >90%, Images = >90%, Translation = >90%, UI = >90%, Multimedia = >90%, Code = >90%, Build = >90%, Devel-doc = >97%, Unknown = >90% (*)

Red

Documentation = >75%, Images = >75%, Translation = >75%, UI = >75%, Multimedia = >75%, Code = >75%, Build = >75%, Devel-doc = >93%, Unknown = >75% (**)

Yellow

Documentation = >30%, Images = >22%, Translation = >80%, UI = 30>%, Multimedia = >50%, Code = >60%, Build = >65%, Devel-doc = 90%, Unknown = >60% (***)

Green

Documentation = 15,5%, Images = 11%, Translation = 63,8%, UI = 17,4%, Multimedia = 0,2%, Code = 37,5%, Build = 50,9%, Devel-doc = 88,2%, Unknown = 30,1% (****)

3.22.3 Metrics

PeopleOnFiles?, PeopleOnGroupsOfFiles?

Metrics from D1.3

PeopleOnFiles?, PeopleOnGroupsOfFiles?

New required metrics

Missing

Unused metrics from D1.3

Missing

3.22.4 Indicator evaluation

Missing

3.23 Fluctuation

3.23.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.23.2 Quality level definition

D1.3 definition: The rate movement of people into, and out of a developer community over time

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.23.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.23.4 Indicator evaluation

Missing

3.24 Established process coverage

3.24.1 Contact person

Martin Soto (IESE)

3.24.2 Quality level definition

The degree to which the development activities a community performs are covered by established, repeatable processes that are widely known and accepted by community members. Development processes that have been observed to be well established in existing development communities include project management (i.e., milestone and roadmap definition, release management including coherence numbering schemes for releases), quality assurance (i.e., bug tracking, different forms of code and code change inspections) and requirements engineering (i.e., product improvement proposals.).

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.24.3 Metrics

Metrics from D1.3

No basic metrics are available for this indicator. For this reason, it cannot be computed during the prototype phase.

New required metrics

Missing

Unused metrics from D1.3

Missing

3.24.4 Indicator evaluation

Missing

3.25 Process automation

3.25.1 Contact person

Martin Soto (IESE)

3.25.2 Quality level definition

The degree to which established processes are partially or completely automated though the use of software tools. Examples of software tools commonly used by development communities to automate software processes include bug tracking systems, build farms and build daemons, and automated test suites.

Black

No process is automated in a established, repeatable way.

Red

The project has established tools for supporting at least one from code control, bug/issue tracking, release package creation and automated testing.

Yellow

The project has established tools for supporting at least three from code control, bug/issue tracking, release package creation and automated testing.

Green

The project has established tools for supporting source code control, bug/issue tracking, release package creation and automated testing.

3.25.3 Metrics

Metrics from D1.3

ToolSupport?

New required metrics

Missing

Unused metrics from D1.3

3.25.4 Indicator evaluation

As explained in the level definition.

3.26 Modification support availability

3.26.1 Contact person

Martin Soto (IESE)

3.26.2 Quality level definition

The availability of support related to performing specific modifications to a software product.

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.26.3 Metrics

Metrics from D1.3
No basic metrics are available for this indicator. For this reason, it cannot be computed during the prototype phase.

New required metrics
Missing

Unused metrics from D1.3
Missing

3.26.4 Indicator evaluation

Missing

3.27 Deployment support

3.27.1 Contact person

Martin Soto (IESE)

3.27.2 Quality level definition

The availability of support related to solving problems arising from the deployment and use of a software product.

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.27.3 Metrics

Metrics from D1.3
No basic metrics are available for this indicator. For this reason, it cannot be computed during the prototype phase.

New required metrics
Missing

Unused metrics from D1.3
Missing

3.27.4 Indicator evaluation

Missing

3.28 Backward support

3.28.1 Contact person

Martin Soto (IESE)

3.28.2 Quality level definition

The availability of support related to older version of a software product still in use.

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.28.3 Metrics

Metrics from D1.3
No basic metrics are available for this indicator. For this reason, it cannot be computed during the prototype phase.

New required metrics
Missing

Unused metrics from D1.3
Missing

3.28.4 Indicator evaluation

Missing

3.29 Failure Tolerance

The capability of the software product to avoid failure as a result of faults in the software

3.29.1 Contact person

FFM (CETIC), JCD (CETIC)

3.29.2 Quality level definition

Note: The application has a decent number of lines of code, a decent number of users and a certain maturity

Black

The application has a high number of critical issues in all or a subset of its releases. It has also a high number of vulnerabilities reported.

Red

The application has a high number of critical issues and a few number of vulnerabilities reported.

Yellow

The application has a few number of critical issues and a few number of vulnerabilities reported.

Green

The application has no critical issues and no vulnerabilities reported.

3.29.3 Metrics

Metrics from D1.3

The metrics from D1.3 are no more used but the result of some of them are combined to produce new metrics

New required metrics

- TotalLinesOfCodeSubsetOpenSourceReleases : Sum of the lines of code for all releases of the product
- CriticalIssuesSubsetOpenSourceReleases : Number of critical issues for a subset of the releases since the application has been liberated. Typically, this supersedes CrashIssuesSubsetReleases, CrashMessage, VulSubsetReleases and ServerVulSubsetReleases?
- CriticalIssuesSubsetOpenSourceReleasesAverage : Number of critical issues per line of code. CriticalIssuesSubsetOpenSourceReleases / TotalLinesOfCodeSubsetOpenSourceReleases
- RemainingIssuesSubsetOpenSourceReleases : Number of issues still open for a subset of the releases since the application has been liberated. Typically this supersedes and combines the metrics TotalIssuesSubsetReleases and ResolvedIssuesSubsetReleases

Unused metrics from D1.3

- TotalIssuesAllReleases
- ResolvedIssuesAllReleases
- RatioResolvedIssuesAllReleases
- TotalIssuesSubsetReleases

- ResolvedIssuesSubsetReleases
- RatioIssuesSubsetReleases
- CrashIssuesAllReleases
- CrashIssuesSubsetReleases
- VulAllReleases
- SeverVulAllReleases
- VulSubsetReleases
- SeverVulSubsetReleases
- CrashMessage

3.29.4 Indicator evaluation

Black:
CriticalIssuesSubsetOpenSourceReleasesAverage >= VERYHIGH
and RemainingIssuesSubsetOpenSourceReleases >= VH

Red:
HIGH <= CriticalIssuesSubsetOpenSourceReleasesAverage < VERYHIGH
and H <= RemainingIssuesSubsetOpenSourceReleases < VH

Yellow:
LOW <= CriticalIssuesSubsetOpenSourceReleasesAverage < HIGH
and L <= RemainingIssuesSubsetOpenSourceReleases < H

Green:
CriticalIssuesSubsetOpenSourceReleasesAverage < LOW
and RemainingIssuesSubsetOpenSourceReleases < L

3.30 Fault Tolerance

The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface
This characteristic has no basic metric defined

3.30.1 Contact person

FFM (CETIC), JCD (CETIC)

3.30.2 Quality level definition

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.30.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.30.4 Indicator evaluation

Missing

3.31 Recoverability

The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure.

3.31.1 Contact person

FFM (CETIC), JCD (CETIC)

3.31.2 Quality level definition

Black
The application cannot recover from a failure.

Red
The application cannot recover from all situations of failure and requires a large amount of manual tune.

Yellow
The application can recover from a failure but requires a certain amount of manual time

Green
The application can recover from a failure automatically without the user's intervention.

3.31.3 Metrics

Metrics from D1.3
The current definitions of the metrics from D1.3 does not say anything about the ability of the software product to recover from a failure. A new definition is needed for the metric RatioRecoverIssuesSubsetReleases

- RecoverIssuesSubsetReleases

New required metrics

- ResolvedRecoverIssuesSubsetReleases: Number of closed issues related to "Recoverability"
- RatioRecoverabilityIssuesSubsetReleases: Ratio between the number of resolved issues and all the issues related to "Recoverability".

Unused metrics from D1.3

- TotalIssuesAllReleases
- RecoverIssuesAllReleases
- RatioRecoverIssuesAllReleases
- TotalIssuesSubsetReleases
- RecoverIssuesSubsetReleases
- RatioRecoverIssuesSubsetReleases

3.31.4 Indicator evaluation

Black

RatioRecoverabilityIssuesSubsetReleases < LOW

Red

LOW <= RatioRecoverabilityIssuesSubsetReleases < HIGH

Yellow

HIGH <= RatioRecoverabilityIssuesSubsetReleases < VERYHIGH

Green

RatioRecoverabilityIssuesSubsetReleases >= VERYHIGH

3.32 Availability

The degree to which a system or component is operational and accessible when required for use.

3.32.1 Contact person

FFM (CETIC), JCD (CETIC)

3.32.2 Quality level definition

Black

The application has a VERY HIGH number of issues reporting its unavailability

Red

The application has a HIGH number of issues reporting its unavailability

Yellow

The application has a MIDDLE number of issues reporting its unavailability

Green

The application has a LOW number of issues reporting its unavailability

3.32.3 Metrics

Metrics from D1.3

RatioAvailIssuesSubsetReleases

New required metrics
Missing

Unused metrics from D1.3

- TotalIssuesAllReleases
- AvailIssuesAllReleases
- RatioAvailIssuesAllReleases
- TotalIssuesSubsetReleases
- AvailIssuesSubsetReleases

3.32.4 Indicator evaluation

Black
RatioAvailIssuesSubsetReleases >= VERYHIGH

Red
HIGH <= RatioAvailIssuesSubsetReleases < VERYHIGH

Yellow
LOW <= RatioAvailIssuesSubsetReleases < HIGH

Green
RatioAvailIssuesSubsetReleases < LOW

3.33 Age

3.33.1 Contact person

Missing

3.33.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.33.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.33.4 Indicator evaluation

Missing

3.34 Activity on stable development branch

3.34.1 Contact person

Missing

3.34.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.34.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.34.4 Indicator evaluation

Missing

3.35 Continuity

3.35.1 Contact person

Missing

3.35.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.35.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.35.4 Indicator evaluation

Missing

3.36 Confidentiality

3.36.1 Contact person

Missing

3.36.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.36.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.36.4 Indicator evaluation

Missing

3.37 Integrity (ISO)

3.37.1 Contact person

Missing

3.37.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.37.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.37.4 Indicator evaluation

Missing

3.38 Security Availability (ISO)

3.38.1 Contact person

Missing

3.38.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.38.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.38.4 Indicator evaluation

Missing

3.39 Compliance to standards

3.39.1 Contact person

Missing

3.39.2 Quality level definition

Missing

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.39.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.39.4 Indicator evaluation

Missing

3.40 MaturityOfSecurityProcessCompliance

3.40.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.40.2 Quality level definition

D1.3 definition: The degree to which the processes and procedures dealing with security adhere to best practices and security standards

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.40.3 Metrics

Metrics from D1.3
Missing

New required metricsMissing
Missing

Unused metrics from D1.3
Missing

3.40.4 Indicator evaluation

Missing

3.41 MaturityOfSecurityProcessReactionTime

3.41.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.41.2 Quality level definition

D1.3 definition: The amount of time that is typically required for resolving security-related issuesv

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.41.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.41.4 Indicator evaluation

Missing

3.42 MaturityOfSecurityProcessInclusionOfPreventive?/reactiveAction

3.42.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.42.2 Quality level definition

D1.3 definition: The degree to which the community commits to actions aimed at preventing security problems

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.42.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.42.4 Indicator evaluation

Missing

3.43 MaturityOfReliabilityProcessCompliance

3.43.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.43.2 Quality level definition

D1.3 definition: The degree to which the processes and procedures dealing with reliability adhere to best practices and security standards

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.43.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.43.4 Indicator evaluation

Missing

3.44 MaturityOfReliabilityProcessReactionTime

3.44.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.44.2 Quality level definition

D1.3 definition: The amount of time that is typically required for resolving reliability-related issues

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.44.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.44.4 Indicator evaluation

Missing

3.45 MaturityOfReliabilityProcessInclusionOfPreventive?/reactiveActions

3.45.1 Contact person

Álvaro del Castillo (URJC), Israel Herraiz (URJC), Daniel Izquierdo (URJC)

3.45.2 Quality level definition

D1.3 definition: The degree to which the community commits to actions aimed at preventing reliability problems

Black
Missing

Red
Missing

Yellow
Missing

Green
Missing

3.45.3 Metrics

Metrics from D1.3
Missing

New required metrics
Missing

Unused metrics from D1.3
Missing

3.45.4 Indicator evaluation

Missing

4. Results of Indicator Application to 1.4 measurement

We applied the indicators as defined in Section 3 to the measurement data gained in task 1.4. The result of this application is documented in a separate spreadsheets. The experience and issues identified during the application are documented as part of the indicator definition in the QualOSS internal wiki pages, and additionally, they are included in this deliverable in Section 6.

Table 1 lists indicator definitions that are missing for the prototype phase, either because no basic metrics were defined, or because the basic metrics turned out not to be available in the projects we measured in task 1.4.

Product evolvability	
Maintainability	
Product Buildability	No basic metrics defined for this characteristic
Fixability	No measurement data available
Maintainability standard compliance	No measurement data available
Community evolvability	
Product adoption	
Strategic importance	No basic metrics defined for this characteristic
Developer community liveness	
Fluctuation	Not a basic metric; interpretation based on 3D-graphic
Process maturity	
Established process coverage	No basic metrics defined for this characteristic
Process automation	No info on release package creation
Support availability	
Modification support availability	No basic metrics defined for this characteristic
Deployment support	No basic metrics defined for this characteristic
Backward Support	No basic metrics defined for this characteristic
Product robustness	
Reliability	
Fault tolerance (ISO 9126)	Only partial indicator provided based on basic metrics provided
Security (ISO 12207)	
Confidentiality	Only partial indicator provided based on basic metrics provided
Integrity (ISO)	Only partial indicator provided based on basic metrics provided
Security Availability (ISO)	Only partial indicator provided based on basic metrics provided
Compliance to standards	No basic metrics defined for this characteristic
Community robustness	
Maturity of security process	
Compliance	No basic metrics defined for this characteristic
Reaction time	No basic metrics defined for this characteristic
Inclusion of preventive/reactive actions	No basic metrics defined for this characteristic
Maturity of reliability process	
Compliance	No measurement data available
Reaction time	No basic metrics defined for this characteristic
Inclusion of preventive/reactive actions	No basic metrics defined for this characteristic

Table 1: List of missing indicators in the prototype

5. Data Mining Approach

5.1 Introduction

For reminder, the aim of the first work package WP1, taking place in the first year of the QUALOSS project, is to build a functional prototype of the quality models (namely, the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality).

As a result of the activities related to tasks T1.1-4, a set of metrics data files (formatted in Open Office spreadsheets) relating to 4 F/OSS projects were built. More precisely, these 4 F/OSS projects of interest for the prototyping phase are:

- JavaCC
- Plone
- HAF Maemo
- SwallowDBE

For each of these F/OSS projects, the associated metrics data files comprise several tabs since the aim of the QUALOSS project is to discover and establish some relationships between particular metrics (measurements of quality characteristics) and 4 distinct quality goals relative to either OSS product component or the OSS community itself.

Namely, the 4 quality goals we would like to assess based on quality metrics are (see also Deliverable D1.3):

- **Product Evolvability**, that this the ability of a product to be corrected, adapted and extended over time, according to the needs of its users.
- **Product Robustness**, that this to say the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.
- **Community Evolvability**, that this the likelihood that a F/OSS community remains able to maintain the product or products it develops over an extended period of time.
- **Community Robustness**, that this to say the ability of the established processes in a community to guarantee the delivery of robust products.

Considering that the number of presently retained F/OSS projects is quite small (only 4) and that the quantity of metrics data made available for these projects is relatively short, it makes no sense to start now applying advanced Data Mining techniques on such data. Consequently, the activities carried out within the task 1.5 were focused on some suggested recommendations in order to help structure and organize the received raw metrics data files into convenient databases being well suited to carry out later deeper analyses using statistical tools but also more advanced Data Mining methods.

Moreover, the directions we recommend to follow and the advices we give herein will be favourably used to realize the validation work during the task T1.6.

5.2 Terminology used for the Data Mining approach

In our Data Mining jargon, a database is seen as a collection of "objects" (most ofen corresponding to the rows of a spreadsheet). These objects are characterised by a set of "attributes" (most ofen corresponding to the columns of a spreadsheet). This way, a particular object consists of a vector of values (i.e. the values taken by the attributes for that given object). An example of such organization of data (a so-called "flat database") is illustrated in Figure 6.

Object Nb	M1	M2	M3	M4	...	kpi-1	kpi-2
o-1	51	35	Bad	2.51	...	High	Low
o-2	49	30	Good	2.89	...	Medium	High
o-3	47		Good	1.67	...	High	
o-4	46	31		2.09	...	Low	Medium
o-5	50	36	Bad	2.56	...		Low
o-6		39	Bad	4.71	...	High	Medium
o-7	46	34	Good	3.98	...	Medium	Medium
o-8	50		Bad	2.34	...		High
o-9	44	29		2.16	...	Low	Low
o-10	49	31	Good	1.09	...	Low	
...	...	...	...	...	...	...	...
o-998	48	30	Bad	1.14	...	Medium	Medium
o-999	43	30	Good	1.46	...	High	High
o-1000	58	40	Good	2.68	...	Medium	Medium

Figure 6: Typical template of a database sample.

The attributes can be of two types, either symbolic or numerical. For instance (see Figure 1), attributes *M1*, *M2* and *M4* are numerical since they take numerical values (integer or floating point, it doesn't matter) whereas attributes *M3*, *kpi-1* and *kpi-2* are symbolic considering that they take qualitative values (namely, the symbols *Bad*/*Good* for the 2-class *M3* attribute or the symbols *Low*/*Medium*/*High* for the 3-class *kpi-1* and *kpi-2* attributes).

Most of the time, certain attributes within the database can be considered as inputs of a modelisation problem whereas other attributes can be seen as outputs. Indeed, one general task of advanced statistical tools and machine learning techniques is to discover the underlying relationships between variables (dependence and correlation analysis) but also be able to predict the value of output variables from the information conveyed by other input variables.

In our example of database sample, the variables *M1*, *M2*, *M3*, etc. could play the role of input attributes and the variables *kpi-1* and *kpi-2* could play the role of output attributes. In practice, we use the terminology KPI (which stands for Key Performance Indicator) in order to designate in the database particular attributes that can clearly be considered as the goal variables we would like to understand and explain from the other attributes.

5.3 Consolidation and organization of the raw data

Adapting these Data Mining concepts specifically for the QUALOSS project context, we can propose the following recommendations in order to consolidate, merge and organize the available data relative to the 4 retained F/OSS projects:

- The 4 spreadsheet files we received gather together several sets of values corresponding to QUALOSS-defined quality metrics (a metric being a way to measure a particular quality characteristic).

  → Each quality metric will be an attribute within the database (for example, the metric *ProductReleaseNumberOfFiles* could play the role of attribute *M1* of Figure 1).
- The QUALOSS project aims at assessing 4 distinct quality goals, namely Poduct Evolvability, Product Robustness, Community Evolvability and Community Robustness.

  → Each quality goal can be considered as an output variable within the database (for example, the quality goal *ProductEvolvability* could play the role of attribute *kpi-1* of Figure 1).

  Note that, unfortunately, the value of these quality goals are unknown (missing) given that the fundamental objective of the QUALOSS project is precisely to provide, by means of quality models, an estimation of the quality goals (for a given F/OSS product or product component)!
- The quality metrics retained for a given quality goal are not the same that the ones used for the three other quality metrics. Consequently, it makes no sense to merge within a unique huge database all the metrics contained in the spreadsheets and all the 4 quality goals (i.e. *kpi-1*, *kpi-2*, *kpi-3*, *kpi-4*).

  → We will accord a specific database to each quality goal (KPI). Thus, we will carry out our Data Mining approach on 4 distinct sub-projects, each of them being characterised by its own set of quality metrics and their corresponding quality goal.

- The same quality metrics (for a given quality goal) have been used for the 4 retained F/OSS projects. Indeed, in each spreadsheet file, we can find a tab relative to each quality goal and the metrics of these tabs are the same for all the F/OSS projects.

  → For a given quality goal, the corresponding database will gather together several objects (rows in the database), one object per F/OSS project. Note that a specific attribute conveying the name of the F/OSS projects will be added to the database in order to be able to filter and put the focus on a given project if necessary.
- Some quality metrics values are missing in the spreadsheets. Most of the time, they are represented by an empty cell, but sometimes, the '?' character was used too.

  → It is important to point out missing values in an unambiguous way and to be able to track them all along the Data Mining analysis. To do so, we recommend to represent missing values by letting empty cells (by way of example, see again the database sample depicted in Figure 1).
- Depending on the F/OSS project and on the quality goal considered, no plausible value was associated with particular metrics.

  → In such cases, the corresponding cells within the databases will be indicated by the NA value (where the acronym NA stands for Not Applicable).
- For the two quality goals regarding the product (namely, Product Evolvability and Product Robustness), the 4 spreadsheets include several tabs relative to different subsets of releases of the product considered (for instance, three sets of metrics are available for the Productivity Robustness quality goal of the F/OSS JavaCC product: a set for all the releases on the whole, another for the sole release JavaCC32 and a third one for the sole release JavaCC40).

  → For these two quality goals, the corresponding databases will gather together several objects (rows in the database) for the same F/OSS project, that is to say one object per distinct subset of releases (a subset can of course represent a unique release). Note also that a specific attribute conveying the name of the release will be added to the database in order to be able to filter and put the focus on a given project release if necessary.
- For the evolvability quality goal regarding the community (namely, Community Evolvability), each of the 4 spreadsheets include a tab containing complementary information. Indeed, these tabs contain additional metrics values obtained for consecutive quarter time periods (for instance, the number of active/non active developers, the number of commits recorded, the number of bugs recorded, etc.). These data may be significant for the building of quality models of this quality goal, so we have to include them within the corresponding database.

  → For the Community Evolvability quality goal, the database will gather together several objects (rows in the database) for the same F/OSS project, that is to say one object per distinct quarter time period. With this end in view, a specific attribute conveying the time information relative to the community activity will be added to the database in order to be able to filter and put the focus on a given period of time if necessary.
- We got very few metrics values for the Community Robustness quality goal (this remark is valid for the 4 F/OSS projects studied). Indeed, on one hand, the basic metrics used didn't provide values and on the other hand, no advanced metrics were used.

  → There is no need to consolidate into a specific database the metrics data concerning the Community Robustness quality goal. Consequently, the future Data Mining analyses will be carried out only on the 3 databases associated with the 3 quality goals Product Evolvability, Product Robustness and Community Evolvability.

5.4 Future activities proposed for the next tasks

On the basis of the previous recommendations suggested with regard to the consolidation and the organization of the metrics data, here are some future orientations we could take in our Data Mining analyses (that is to say for the validation task T1.6 and during achievement of the whole work package WP4):

– The spreadsheets that we received contain additional information indicating for each metric value the source(s) from which it comes. The possible sources are:
  - Issue Tracking
  - Security DB
  - Version Control Repository

- Publication DB
- Package Distribution List
- Website
- Mailing List

This information is presently not used. To a certain extent, that means that we make the assumption that the reliability of these different sources are equivalent. Nevertheless, we do know that their reliabilities are not the same. For instance, data coming from mailing lists are less significant and less reliable than data coming from bugs tracking or version control repositories.
Thus, we could associate the different data sources with weights according to their assessed reliability in order to take into account this complementary information for building the advanced quality models using Data Mining techniques.

– It seems relatively difficult to merge the data relative to PeopleOnFiles metrics with the proposed databases described above. In fact, these data are quite specific to a particular F/OSS project and we don't see the real advantage to unfold the databases (i.e. repeating several times the same object and changing only the value corresponding to the PeopleOnFiles metric) to take into account these data. Therefore, if QUALOSS partners wish to exploit this information all the same, we will have to define and set a new structure of data to carry out appropriate Data Mining analyses on these complementary data.
– It's likely that the retained F/OSS projects are so different that it makes no sense to try to presently discover and to identify similarities between them (a given metric having most probably not exactly the same meaning for a Web Server F/OSS project and for a Application F/OSS project). Consequently, we may want to realize statistical analyses on a per project basis (without taking into account the metrics data available for the other F/OSS projects made available in the databases). This kind of per project study is of course possible since that the proposed structure of the databases plans to include a specific attribute conveying the project name information. Through this attribute, we can filter and put the focus on any desired project.
– For the moment, we have no quality goals data at the disposal of advanced statistical tools and machine learning techniques. Consequently, we cannot use at present the supervised learning methods in order to build predictors of the quality goals from data metrics. However, we can get prepared to do it if we judge that certain metrics, whose data are available, are worth to be considered themselves as goal metrics. That is to say, we can define available metrics as being typical goal metrics we would like also to explain and predict. For instance, we can consider that "the number of bugs" metric is an important metric and that we would like to discover what are the other most significant metrics connected to it.
– Most of the metrics data are numerical and consequently it will be possible to make use of all families of Data Mining techniques (clustering, regression, dependence analysis,...). Moreover, it is not a problem to transform numerical metrics into symbolic metrics (subdividing the range of the numerical variable into the desired number of classes). In that way, an other important family of Data Mining techniques can also be used, namely the classification algorithms (for instance, Decision Trees, Neural Networks, K-Nearest Neighbours, etc.).

6. Discussion

This section summarizes lessons learned, experiences with measurements, and adaptations/changes to QualOSS model.

Main issues for the advanced models are

- The current indicator scale has four levels. It may make sense to define intermediate levels, e.g., between green/yellow, and red/black, if it turns out in task 1.6 that the current interpretation is not fine-grained enough. More than 7 levels, however, are usually not helpful, so six or seven levels of interpretation should be the maximum number for the interpretation scale. Alternatively, the interpretation could be done on a rational scale (e.g., from 0 to 100); the problem there is that it is usually difficult to assign interpretation values to nominal metrics in that case.
- We need to identify a reliable way to aggregate indicator value up the tree of characteristics. In this deliverable, we have defined several potential approaches that need to be further evaluated.
- Currently, we have one interpretation rule for an indicator. We may need to have different interpretation rules for an indicator dependent on product (e.g., programming language), and type of intended usage.
- There are several areas of the QualOSS model where either not basic metrics were defined, or they were defined but proved not be measurable in practice. For some quality attributes, the metrics are only available for a few F/OSS projects. For the remainder of this project, we need to focus on some areas to define advanced metrics, as it will not be possible to address all of them in detail. In addition, we need to identify a reliable way to deal with missing data, and to assess the data quality.
- Defining indicators and corresponding interpretation rules helps documenting the rationale for metrics, and identify missing information for interpreting measurements.

The remainder of this section lists the issues and lessons learned for each indicator.

6.1 Usefulness of code documentation – Actuality

6.1.1 Issues for advanced metrics

This definition is mainly limited by the fact that the metrics specified so far are insufficient to answer the question of whether documentation is current or not. In particular, the issue is not to determine if the documentation is old, but whether the interface has changed since documentation was last updated. For example, for Java it would be necessary to check whether interface changes were accompanied by changes of the related javadoc documentation.

For the moment, we use this definition as a very first approximation, but we should refine this indicator in the future. In particular, we should find a way (through which metrics ?) to correlate the "age" of documentation and the changes being done.

6.1.2 Additional Comments / Problems

FFM: The basic metric only captured when the code documentation has been generated with respect to the source code. One way to do this is to compare the date of the API Files compared to their originated source files. So this metric is OK. In the basic metrics we did not designed a solution on which we can base ourselves to assess the quality of the commented lines of code with respect to their surrounding lines of code.

6.2 Usefulness of code documentation – Coverage

6.2.1 Issues for advanced metrics

None

6.2.2 Additional Comments / Problems

We want to notice that it will be also important to discuss about the code coverage in term of measure used in software testing. It describes the degree to which the source code of a program has been tested. For the

moment, we propose a first approach regarding of the definition of the QualOSS model and it will be interesting to refine it.

FFM: About the general definition: I'm not quite OK that we should mix here information about the Actuality of the code documentation since it is already captured. So we should have stuck to the real code coverage part here. The fact that we want to have a high coverage and a high actuality will be captured in the QualOSS Model, but not here in the indicators.

FFM: About the definition of the levels: We should provide here a text explaining the level, not a "logic" or "mathematic" definition. For example, Black if there is less than 1 line of comment per 5 lines of code. This is the same as SourceComPer? < 20% but more "user friendly"

6.3 Usefulness of code documentation - Code documentation standard compliance

6.3.1 Issues for advanced metrics

None

6.3.2 Additional Comments / Problems

FFM: About the definition of the levels: We should provide here a text explaining the level, not a "logic" or "mathematic" definition.

For example, Black if there is a high rate of errors in the API documentation compared to the guidelines of the code documentation.

6.4 Usefulness of user documentation – Actuality

6.4.1 Issues for advanced metrics

None

6.4.2 Additional Comments / Problems

It is important to notice that metrics used here is limited by the fact that we do not consider the standardization of the way to write a user documentation. It will be important to consider this other aspect of the user documentation.

FFM: UserDocumentationAPIDocumentationCommonAbstractionsPercentage has been promoted to Advanced Metric

FFM: We should not mix coverage and actuality. This is the purpose of the Qualoss Model to capture this.

6.5 Usefulness of user documentation – Coverage

6.5.1 Issues for advanced metrics

None

6.5.2 Additional Comments / Problems

FFM: Definition of the levels in the Quality Level Definition should be expressed in a textual form, not a mathematical one For example, Black if the difference between the date of the user documentation is far earlier than the project release.

FFM: UserDocumentationAPIDocumentationCommonAbstractionsPercentage has been promoted to "advanced metrics"

FFM: Same remark about the removal of Actuality from this characteristic.

6.6 Usefulness of user documentation – Internationalization

6.6.1 Issues for advanced metrics

None

6.6.2 Additional Comments / Problems

FFM: The metrics are not enclosing all cases. What if an application has 5 translations?

6.7 Usefulness of user documentation – Code documentation standard compliance

6.7.1 Issues for advanced metrics

None

6.7.2 Additional Comments / Problems

None

6.8 Maintainability – ProductComplexity

6.8.1 Issues for advanced metrics

Our current metrics are mainly based on cyclomatic complexity, which, in turn, relates to the structural complexity of the involved algorithms. Often, the difficulty in understanding a program lies rather on the involved data flow. Further research on metrics related to data flow complexity may be necessary.

6.8.2 Additional Comments / Problems

None

6.9 Maintainability – Architecture flexibility

6.9.1 Issues for advanced metrics

None

6.9.2 Additional Comments / Problems

None

6.10 Maintainability – Product Buildability

6.10.1 Issues for advanced metrics

None

6.10.2 Additional Comments / Problems

None

6.11 Maintainability – Fixability

6.11.1 Issues for advanced metrics

IssueOpenToCloseTimeAverage? may also be high due to low community participation. It would be appropriate to also relate this indicator to metrics that evaluate the product´s intrinsic complexity.

6.11.2 Additional Comments / Problems

This indicator is closely related to other maintainability indicators. We should consider merging it with them.

No data available to evaluate this indicator from any of our test projects.

6.12 Maintainability – Maintainability standard compliance

6.12.1 Issues for advanced metrics

Automatically evaluating standard compliance can be very difficult in certain cases. Manual code inspection may be necessary in many cases.

6.12.2 Additional Comments / Problems

Data available for only one project.

6.13 Interoperability – Runtime Interoperability

6.13.1 Issues for advanced metrics

The computation of this metrics is based on the documentation (hence needs heuristics) and the use of the software, this is thus not easily automated.

6.13.2 Additional Comments / Problems

The definition of "Open Standard Exchange Format" needs to be unambiguous and satisfy a large audience

The case: ProductReleaseNumberOfCommunicatingApplications == 0 and ProductReleaseNumberOfOpenExchangeFormats > 0 is not considered. Should it be Red?

6.14 Interoperability – Passive Interoperability

6.14.1 Issues for advanced metrics

None

6.14.2 Additional Comments / Problems

None

6.15 Portability – Platform specificity

6.15.1 Issues for advanced metrics

- In order to decide whether a library is standard or not, a list of standard libraries should be created for each domain and programming language.
- A new metric WeightedWantedPlatformsCompliance should be defined as the weighted sum of the declared platforms found in the list of platforms expressed by the user.

6.15.2 Additional Comments / Problems

- The notion of "Highly portable programming language" is not well defined and also raised discussion. Hence it has been removed.
- A new metric related to the number of platforms on which the application can run is introduced.

6.16 Portability – Portability standard compliance

6.16.1 Issues for advanced metrics

The user should define the rates VERYHIGH, HIGH, LOW

6.16.2 Additional Comments / Problems

The thresholds should be defined based on the analysis of a decent number of applications.

6.17 Product adoption – User community size

6.17.1 Issues for advanced metrics

It could be more meaningful to measure the size during the last month, the last three months, the last year and the overall history of the project.

6.17.2 Additional Comments / Problems

It is necessary to say that a community of developers is a set of people with several roles, from documentators, to developers and translators.

Number of posters (NumOfPostersMailingLists?) missing in 4 out of 5 cases (from D1.4)

6.18 Product adoption – Strategic importance

6.18.1 Issues for advanced metrics

None

6.18.2 Additional Comments / Problems

None

6.19 Product adoption – License permissiveness

6.19.1 Issues for advanced metrics

None

6.19.2 Additional Comments / Problems

Is it better to have a non restrictive license? It depends on the point of view. There are some cases where the business model improves if the license is BSD based instead of GPL based and so on.

For D1.4, the license type information was missing in 4 out of 5 cases; time to manually extract approx. 15 min. comment in measurement sheets is that where there are multiple license files, it is not possible to analyze this manually nor automatically.

6.20 Developer community liveness – Developer community size

6.20.1 Issues for advanced metrics

This indicator could be measured over size for the last month, last three months, overall history, etc., to see how the size of the community has changed in the near history of the project.

6.20.2 Additional Comments / Problems

PastNumOfDevelopers? -> No sense to have a metric here. The TotalNumOfDevelopers? can be measured in several periods of time. Thus, at the beginning a community could have the black label and in the end it could have the green label because it has evolved (same applies to PastNumOfActiveDevelopers?).

Carlos: It will be difficult to find any projects with >100 active developers. The boundaries may have to be changed for the advanced models.

The rules are ambiguous. It is not clear that a project falls into exactly one category. In particular, growing/constant function of developers: How do we judge whether the function is growing? What if you have

120 developers, but you lose 20; will the project then be black because the number of developers is decreasing?

HAF-Example: With 26 active developers, this project is "red". Should the indicator levels be redefined?

6.21 Developer community liveness – Developer community activity

6.21.1 Issues for advanced metrics

We need "backup-rules" that work in cases where not all information is available, e.g., if #replies cannot be measured

The "green" rule states: ">300 changes to source (including not only source code) "; this cannot be measured right now

6.21.2 Additional Comments / Problems

It is not clear how the listed metrics and the descriptions in the rules refer to each other. Eg., is NumOfChangesToSource the same as "number of changes... in a period of time"? How are "developers replying" related to the NumOfMessagesOfDevelopers?

For HAF, #posts are not measured

Evolution-Measures are not used; are they obsolete?

Number of postings (NumOfMessagesOfDevelopers) were not available for all five projects in D1.4

6.22 Developer community liveness – Developer community heterogeneity

6.22.1 Issues for advanced metrics

None

6.22.2 Additional Comments / Problems

Results must be divided It must be divided in several kind of files (depending on CVSAnalY results). For example, files used for the translation of the project. There is, sometimes, a file per language and the number of translators is not usual to grow. (See Sheet 2 for more information).

(****)Those results have been obtained working on Gimp, Evolution, Evince, DogTail?, Ekiga and Planner (Analysis made April/June 2007)

(*)(**)(***) Most of committers working on each type of file (near 90% en each type of file), if there are higher percentages, it means that there are low heterogeneity. Thus, yellow, red and black indicators are steps from green indicator (****). They are not based in estimations from projects like (****).

It is not clear how the rules have to be interpreted; are they connected by "and", "or"? After some thought, it seems that if all file types are above their threshold, the project falls within a certain category. Unclear, and needs to be clarified: When does a project fall into a specific category?

The percentages add up to more than 100% (because developers contribute more than once). To simplify interpretation, the measurement sheet should already contain percentages (or at least specify what 100% is; assumption: TotalNumDevelopers?). Currently, the data from 1.4 does not match indicator definition but contains absolute numbers of developer ids. Time to extract information from measurement ca. 3 minutes

6.23 Developer community liveness – Fluctuation

6.23.1 Issues for advanced metrics

None

6.23.2 Additional Comments / Problems

It must be a metric with just a value from 0 to 1. 1 means good regeneration, and 0, bad regeneration. A plug-in in CVSAnalY must be implemented to obtain this requirement. In general, it is better to talk in percentage results instead of absolute numbers. Results are based on the study of six projects (dogtail, ekiga, evince, evolution, gimp and planner)

We need to develop criteria to evaluate fluctuation; currently, we used the 3D graph of fluctuation for the indicator. The graph is missing for GNAT, and in 3 out of 4 cases, there are too few developers for good evaluation. That is, only for one project (Plone), the evaluation yielded good results. --> consequence: Need to redesign this indicator

6.24 Process maturity – Established process coverage

6.24.1 Issues for advanced metrics

None

6.24.2 Additional Comments / Problems

None

6.25 Process maturity – Process automation

6.25.1 Issues for advanced metrics

New metrics may be necessary to determine to what extent a particular tool is being used. For example, a bug tracking system may be in place, but many of the bug reports may actually flow through other channels, such as the mailing lists, IRC channels, or personal communications.

6.25.2 Additional Comments / Problems

Determining which tools are actually used by a project and their acceptance among the developer community may require deep knowledge of the project, that may be hard for an external observer to acquire.

6.26 Support availability – Modification support availability

6.26.1 Issues for advanced metrics

None

6.26.2 Additional Comments / Problems

None

6.27 Support availability – Deployment support

6.27.1 Additional Comments / Problems

None

6.28 Support availability – Backward Support

6.28.1 Issues for advanced metrics

None

6.28.2 Additional Comments / Problems

None

6.29 Reliability – Failure tolerance (ISO 9126: maturity)

6.29.1 Issues for advanced metrics

- The definition of a "Critical Issue" needs to be done carefully
- The user has to define the rate VERYHIGH, VH, HIGH, H, LOW and L

6.29.2 Additional Comments / Problems

- The current metrics (D1.3) takes into account the number of resolved issues, but this seems to be more related to the Evolvability than to the Robustness. The robustness should compute only the remaining issues (issues still open) at the time of the assessment is performed.
- There is a need to clearly identify the Open Source Releases
- Should we compare absolute or relative values? [should we normalize the metrics]?

6.30 Reliability – Fault tolerance (ISO 9126)

6.30.1 Issues for advanced metrics

None

6.30.2 Additional Comments / Problems

None

6.31 Reliability – Recoverability (ISO9126)

6.31.1 Issues for advanced metrics

The user has to define the rates VERYHIGH, HIGH and LOW

6.31.2 Additional Comments / Problems

None

6.32 Reliability – Availability (IEEE)

6.32.1 Issues for advanced metrics

The user has to define the rates VERYHIGH, HIGH and LOW

6.32.2 Additional Comments / Problems

By "Subset Releases" we mean "Subset of Open Source Releases"

6.33 Maturity – Age

6.33.1 Issues for advanced metrics

None

6.33.2 Additional Comments / Problems

None

6.34 Maturity –Activity on stable development branch

6.34.1 Issues for advanced metrics

None

6.34.2 Additional Comments / Problems

None

6.35 Maturity – Continuity

6.35.1 Issues for advanced metrics

None

6.35.2 Additional Comments / Problems

None

6.36 Security (ISO 12207) – Confidentiality

6.36.1 Issues for advanced metrics

None

6.36.2 Additional Comments / Problems

None

6.37 Security (ISO 12207) – Integrity (ISO)

6.37.1 Issues for advanced metrics

None

6.37.2 Additional Comments / Problems

None

6.38 Security (ISO 12207) – Security Availability (ISO)

6.38.1 Issues for advanced metrics

None

6.38.2 Additional Comments / Problems

None

6.39 Security (ISO 12207) – Compliance to standards

6.39.1 Issues for advanced metrics

None

6.39.2 Additional Comments / Problems

None

6.40 Maturity of security process – Compliance

6.40.1 Issues for advanced metrics

None

6.40.2 Additional Comments / Problems

None

6.41 Maturity of security process – Reaction time

6.41.1 Issues for advanced metrics

None

6.41.2 Additional Comments / Problems

None

6.42 Maturity of security process – Inclusion of preventive/reactive actions

6.42.1 Issues for advanced metrics

None

6.42.2 Additional Comments / Problems

None

6.43 Maturity of reliability process – Compliance

6.43.1 Issues for advanced metrics

None

6.43.2 Additional Comments / Problems

None

6.44 Maturity of reliability process – Reaction time

6.44.1 Issues for advanced metrics

None

6.44.2 Additional Comments / Problems

None

6.45 Maturity of reliability process – Inclusion of preventive/reactive actions

6.45.1 Issues for advanced metrics

None

6.45.2 Additional Comments / Problems

None

7. Summary and Conclusions

This deliverable defines the interpretation model (or user manual) through indicators, and calibrates the QualOSS model. The main goal of this task was to construct a prototype interpretation guide and calibration, and to identify issues to be tackled in the remainder of the project. These issues include feasibility of the measurement, and identification of parts of the QualOSS model that are important but have no metrics available.

The approach taken in task 1.5 was to define indicators for the different quality characteristics that combine and interpret the different metrics associated to a quality characteristic into a single metric value, as well as define initial aggregation and abstraction mechanisms. For the interpretation scale, we have proposed a four-level scale (black, red, yellow, and green). To validate and calibrate the indicators and the corresponding interpretation, they were applied to the projects measured in task 1.4, and the resulting experience was documented. In addition, we defined the approach for data mining to be used to construct advanced models, which will be validated in task 1.6 before being applied in a task of WP4.

Further work is still required. In the QualOSS prototype quality models, there are some characteristics that cannot be measured at the moment, such as community robustness. Reasons include that no basic metrics could be defined, or that the metrics we defined turned out not to be measurable with an appropriate amount of effort, or that no indicators were completely defined yet for aggregating several metrics into a single value.

Appendix A: Indicator Definition Template

Appendix B: The purpose of this template is to document a QualOSS indicator. While using this template, please keep the following points in mind:

- *A detailed explanation of indicators can be found in* IndicatorGuidelines. *You should read it prior to using this template.*
- *For consistency, it is important that all indicator definitions follow the same structure, so, please do not alter the structure of the template. Particularly, do not add or remove headings, or introduce fancy structure (such as tables). If you absolutely feel you have to do that, please discuss it with us (Martín Soto, Marcus Ciolkowski) before you proceed.*
- *Both this introduction, and the explanations for each section are intended for indicator authors, and not for indicator users. Please delete them before creating the final version of your definition.*

Quality Attribute

<Quality attribute this indicator applies to. Use the name from D1.3>

Contact person

<Name and institution of the person responsible for this indicator definition>

Quality level definition

This section defines the meaning of the four quality levels, Black, Red, Yellow, and Green in the context of this particular indicator. They should be specific to the particular quality attribute.

Black

<Definition of the Black level>

Red

<Definition of the Red level>

Yellow

<Definition of the Yellow level>

Green

<Definition of the Green level>

Metrics

This section is used to refer to metrics relevant to the indicator.

Metrics from D1.3

List here the metrics defined in D1.3 that are necessary to calculate the indicator. Please use the exact names from D1.3.

- *<MetricOne>*
- *<MetricTwo>*

New required metrics

Describe here any metrics that you consider necessary in order to evaluate the indicator, but that are not defined in D1.3. Use a description list as follows:

<MetricThree>

<Explanation of the third metric.>

<MetricFour>

<Explanation of the fourth metric.>

Unused metrics from D1.3

List here the basic metrics that were defined in D1.3 for the quality attribute, but that you decided not to use. Add a brief explanation telling why they are unsuitable. Use a description list as follows:

<UnusedMetricOne>

<Why UnusedMetricOne was not used>

<UnusedMetricTwo>

<Why UnusedMetricTwo was not used>

Indicator evaluation

Describe here the rule or formula used to evaluate the indicator. The inputs are the metrics listed in the previous section. The output is a value from the set {Black, Red, Green, Blue}.

Changes to QualOSS model

Describe the changes to the QualOSS model you did during measurement or indicator definition, or changes that you think would be necessary in the future.

Issues for advanced metrics

Discuss any issues you may have observed regarding the introduction of advanced metrics in later phases of QualOSS. This includes problems that may arise, or opportunities we may have.

Additional Comments / Problems

Add here any additional comments you may have. This includes but is not limited to problems that may occur while evaluating the indicator, and issues related to data availability and reliability.