Qualoss

Sponsored through Framework Programme Sixth (Call 5) by

Document Information

Version: 1.0
Date : Jun 22, 07
Pages : 121

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):
Flora Kamseu
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 06/22/07 at 06:12:36 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: **D1.3**

Title:

QualOSS D1.3

Metrics System and Prototype QualOSS Models

A deliverable of Task 1.3

Deliverable: D1.3

Title: QualOSS D1.3

Executive Summary:
This document describes the work done and results obtained in task 1.3 (“Definition of Metrics Systems and Prototype Quality Models”) of the QualOSS project.

This document describes the work done and results obtained in task 1.2 (“Goal of measurements”) of the QualOSS project. We review relevant definitions of robustness and evolvability in F/OSS assessment approaches and in the state of the art and practice of quality models. Additionally, we take into account stakeholders' perceptions and requirements through a series of interviews.
Goals and requirements for the QualOSS model are defined in terms of (a) business and measurement goals, and (b) a consolidated definition of quality characteristics for evolvability and robustness from related work and from stakehoders' views, and (c) an initial plan for validation of the QualOSS model.

Further work is still required. In particular, the QualOSS model needs to be further refined into metrics; this is part of task 1.3. We foresee that part of task 1.3 will be to develop an assessment method for evaluating the community maturity, as approaches to evaluate associated processes have so far not been considered in F/OSS assessment methods.

Section 1 presents the motivation of task 1.3, and explains how the tasks in workpackage 1 collaborate to produce the initial QualOSS model.

Section 2 presents the revised definitions of evolvability and robustness characteristic, based on D1.2 and the insights gained during task 1.3

Sections 3 to 6 present the initial version of the prototype QualOSS model; that is, they describe how we intend to measure the quality characteristics defined in Section 2. Thereby, Sections 3 and 4 focus on the product and community aspects of robustness, respectively. Sections 5 and 6 describes product and community aspects of evolvability, respectively.

Section 7 presents the initial version of a process assessment framework for F/OSS projects. During task 1.2, we identified the need to better understand an F/OSS project's processes to assess its maturity. Process assessment aspects impact both evolvability (e.g., in terms of how a project deals with sustaining its community) as well as robustness (e.g., in terms of how a project deals with resolving reliability problems).

Section 8 presents the initial version of a documentation assessment framework. As identified during task 1.2, there are no readily available metrics to assess the quality of documentation available for an F/OSS product.

Section 9 lists issues identified so far that need to be addressed by the advanced models.

Section 10 contains the interpretation model that will allow to interpret metric values with respect to the quality characteristic they intend to measure. This also includes aggregation issues.

Finally, Section 11 presents conclusions and future steps.

The Appendix contains the detailed tables of identified metrics.

CHANGE LOG

Ver.	*Date*	*Author*	*Description*
0.1	09/15/06	Marcus Ciolkowski	Initial proposal for structure
0.2	06.06.07	Marcus Ciolkowski	Draft Deliverable, containing input from all partners
0.3	07/06/07	Flora Kamseu	First review of the deliverable; correction of few typos apart from Naji' comments
0.4	11/06/07	Jose Ruiz	Update tools to analyze Ada, C, and C++ code as well as some metrics
0.5	13/06/07	Martín Soto	Integration of contributions from all partners. Formatting.
0.6	15/06/07	Jean-Christophe Deprez	Review Section 8
0.7	20/06/07	Alvaro del Castillo	Update and rework of sections 4 and 6
0.8	20/06/07	Marcus Ciolkowski	Update of Sections 1, 3, 10 and integrationof contributions
0.9	21/06/07	Jean-Christophe	Review of all doc
0.10	21/06/07	Flora Kamseu	Second review of all D1.3
0.11	22/06/07	Marcus Ciolkowski	Rework of D1.3 according to review
0.12	22/06/07	Jean-Christophe Deprez	Sanity Check
1.0	22/06/07	--	Submission to EC

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 the deliverable D1.2 was to define the goals and requirements for this assessment method, the QUALOSS quality model. The goal of this deliverable, D1.3, is to operationalize the definitions from D1.2; that is, to assign metrics to the quality characteristics defined in D1.2, and to identify issues to be tackled in the advanced quality models.

1.2 Goal

The key result of task 1.3 is the definition of concrete metrics for the quality model (i.e., the definition of quality characteristics) in D1.2, and to identify issues that need to be resolved for the advanced quality models.

More specifically, the goals of task 1.3 are to:

- Identify metrics for the quality model in D1.2, taking into account findings from tasks 1.1 and 1.2., thus relating metrics to measurement and business goals.
- Partition the metrics in two sets i.e., a first set of “basic” metrics to use for building prototype quality models and a second set of “advanced” metrics to be used later during the second phase when augmenting our prototype. The constraint for including a metric in the “basic” set is that raw data and tools needed to measure the metrics should be available. Neither new research nor involved development need take place to measure basic metrics.
- Identify the information needed to create new “advanced” metrics, and issues to be addressed when building advanced quality models (e.g., constructed from advanced metrics)

Compared to the description of work (DoW), there are several changes to the structure of this document:

- The definition of underlying terms is moved to the appendix.
- We introduced a section that presents the refined definitions of the quality characteristics from D1.2
- According to the DoW, section 2 presents all metrics and catalogues them as either basic or advanced. We have split this section across sections 3-6 to avoid deeply nested hierarchies. Also, currently, we have not yet marked any metrics as “on hold” because they are too time consuming to be implemented in the timeframe of the project. We expect that this identification will occur in tasks 1.4 and 1.5.
- So far, we have only identified simple formulae to aggregate metrics towards quality characteristics (referred to as quality models in the DoW). For this reason, we have integrated their description into the user manual / interpretation guide section. Consequently, we have not yet marked any formulae as authoritative or alternative.
- Compared to the DoW, we have introduced two sections on process and documentation assessment, as insight gained during D1.2 suggested that these aspects are important for the QualOSS approach.
- The first draft of the user manual/ interpretation guide is contained in Section 10

- Issues for advanced quality models are described in Section 9.

1.3 Strategy For Workpackage 1

The main objective of WP1 is to perform requirement analysis through prototyping. Currently, there exists a set of metrics and corresponding measuring tools.

The outcome of prototyping in WP1 serves in performing a thorough requirement analysis in order to well formulate our requirements and eventually, 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 will be 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. 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 will also take into account available data and tools, as elicited in task 1.1. shows the inputs for task 1.3. In particular, this implies that, compared to the description of work, the definition of metrics for the QualOSS model will completely be shifted to task 1.3.

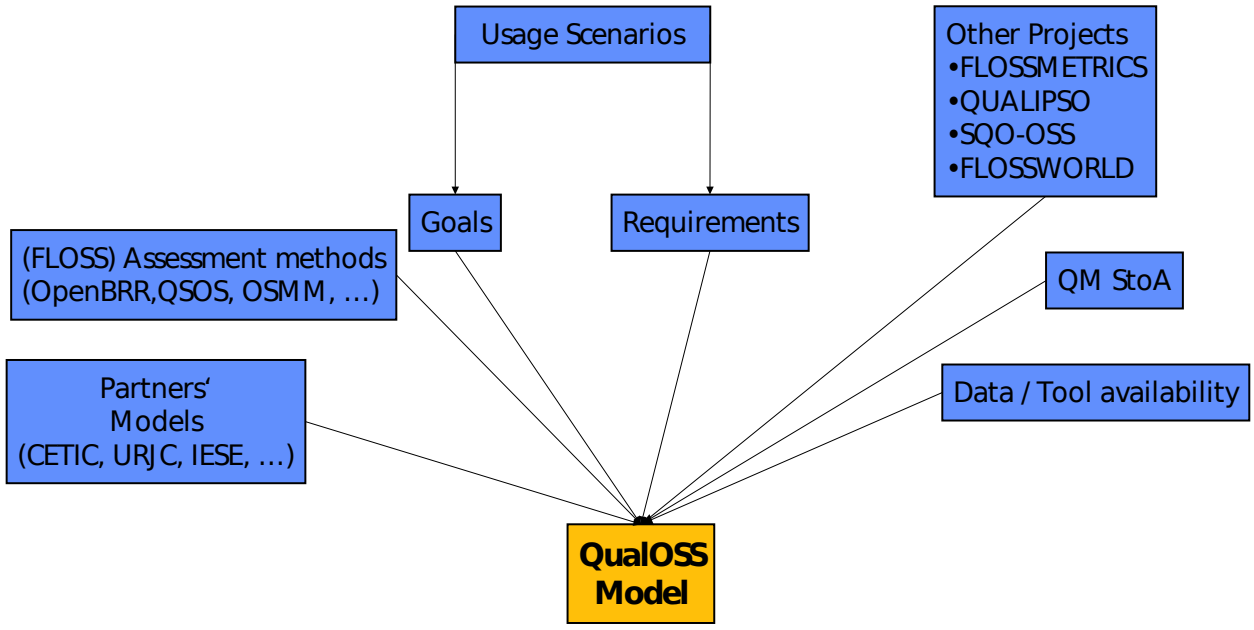


Figure 1: Input Sources for QualOSS model (D 1.3)

(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. Calibration of the quality models is addressed in task 1.5. More precisely, task 1.5 examines the usefulness and applicability of the quality models and tries to find patterns and dependencies in the data that can be used as input to improve the quality models. Task 1.6 validates the quality models on 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. Figure 1 illustrates the dependency between D 1.2 and D 1.3.

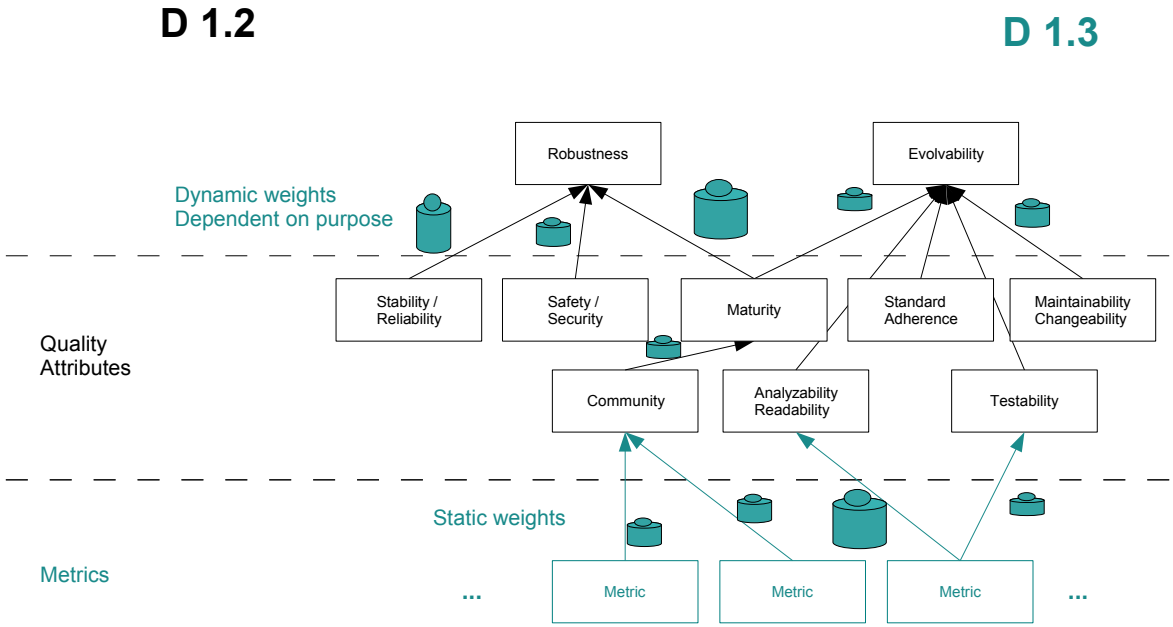


Figure 2: Relation between Deliverables 1.2 and 1.3: D 1.2 defines the quality characteristics that are relevant for evolvability and robustness, while D 1.3 contributes the definition of metrics, and D1.4 and 1.5 will propose initial weighting schemes to reflect different priorities between quality characteristics

1.4 Approach

This section describes the approach we took to achieve the goals of Deliverable 1.3.

The goals of D1.3 can be summarized as follows: Mapping metrics to the quality attributes defined in D1.2

The approach taken in task 1.3 is to identify metrics for the quality attributes defined in D1.2, and to update the quality characteristics definition from D1.2, where necessary. To this end, we used collection sheets to systematically collect and map input from the different partners (see Appendix C: Metric Collection Sheets). To do this, we defined and collected specific collection sheets that the partners filled in (see Appendix). In addition, we found that it is necessary to assess aspects of the processes that are used in the F/OSS projects, as well as to assess the documentation quality. To this end, we developed assessment frameworks (see Section 7 and Section 8) that enhance the metrics identified so far. Finally, task 1.3 defines issues for advanced models.

The next step in evolving the QualOSS quality model is 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 derive initial weights for the characteristics; this will be done in the remainder of WP1 (see Section 10).

1.5 Structure of the Deliverable

This document presents initial quality models related to Evolvability and Robustness. First, we reiterate the definition of each characteristic found in D1.2. Second, we present data sources and metrics that help assess leaf characteristic in the two trees defining evolvability and robustness. Furthermore, we also propose interpretations for aggregating several related measurements into a meaningful assessment of a characteristic. For example, how to aggregate coupling and cyclomatic complexity to propose a meaningful information that characterizes “Product Complexity”. At this moment, Aggregation methods are either based on intuition or on information found in the scientific literature.

The rest of the deliverable is structured as follows:

Section 1 presents the motivation of task 1.3, and explains how the tasks in workpackage 1 collaborate to produce the initial QualOSS model.

Section 2 presents the revised definitions of evolvability and robustness characteristic, based on D1.2 and the insights gained during task 1.3

Sections 3 to 6 present the initial version of the prototype QualOSS model; that is, they describe how we intend to measure the quality characteristics defined in Section 2. Thereby, Sections 3 and 4 focus on the product and community aspects of robustness, respectively. Sections 5 and 6 describes product and community aspects of evolvability, respectively.

Section 7 presents the initial version of a process assessment framework for F/OSS projects. During task 1.2, we identified the need to better understand an F/OSS project's processes to assess its maturity. Process assessment aspects impact both evolvability (e.g., in terms of how a project deals with sustaining its community) as well as robustness (e.g., in terms of how a project deals with resolving reliability problems).

Section 8 presents the initial version of a documentation assessment framework. As identified during task 1.2, there are no readily available metrics to assess the quality of documentation available for an F/OSS product.

Section 9 lists issues identified so far that need to be addressed by the advanced models.

Section 10 contains the interpretation model that will allow to interpret metric values with respect to the quality characteristic they intend to measure. This also includes aggregation issues.

Finally, Section 11 presents conclusions and future steps.

The Appendix contains the detailed tables of identified metrics.

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

2. QualOSS Prototype Model

This section presents the refined definition of each the quality characteristics found in D1.2, based on the insights we gained during task 1.3.

We note that compare to D1.2, the tree of characteristics slightly changed, for evolvability, the “Coverage” characteristic under “Usefulness of User Documentation” would be too hard to measure reliability hence it is replaced with “Understandability”. Understandability of Documentation influences the usability of a software product and and can therefore influence the rate of adoption of the product.

Furthermore, also in the hierarchy defining evolvability, the quality “Popularization Support Availability” is added under Support availability”. This new characteristic is less technical and includes the notion of having groups of facilitators that promote the product, search for donation, etc. A F/OSS foundation is a type of group that helps with popularization.

Beside pointing to interesting analyses on a single data occurrence such as source code analyses, it is also possible to propose historical analyses of data found in the list of data sources inventoried in D1.1. Historical analyses can be specified in relation to a single type of data, to several data types found in a single data source, or to data types found in several data sources. Examples of historical analysis possible for each Respective scenarios are lines of code evolution over several product distributions, the average time range and number of comments taken to resolve an issue, the average number of exchange over issue tracking system, mailing lists and version control to solve an issue.

In addition to product-centric metrics, we widened the scope of QualOSS due to our interactions with other E.C. projects, namely, FLOSSMETRICS and SQO-OSS. In particular, we started development on an assessment framework for software development processes of open source projects to better assess project maturity. In addition, we are creating an extensive framework of development processes and best practices to use in open source so as to have a broad framework against which to evaluate actual F/OSS projects. Furthermore, we studied the possibility to development evaluation method for non-trivial data such as user documentation. Initial results of these efforts are presented in Sections 7 and 8.

2.1 Evolvability

We define *evolvability* as the general ability of a F/OSS project to deliver useful products (or product updates) over an extended period of time. Also the ability of such products to remain useful for an extended period of time. In order to be able to decompose this wide notion into smaller criteria that can be studied separately, we consider products and their related F/OSS community independently from each other. Figure 3 shows the resulting structure.

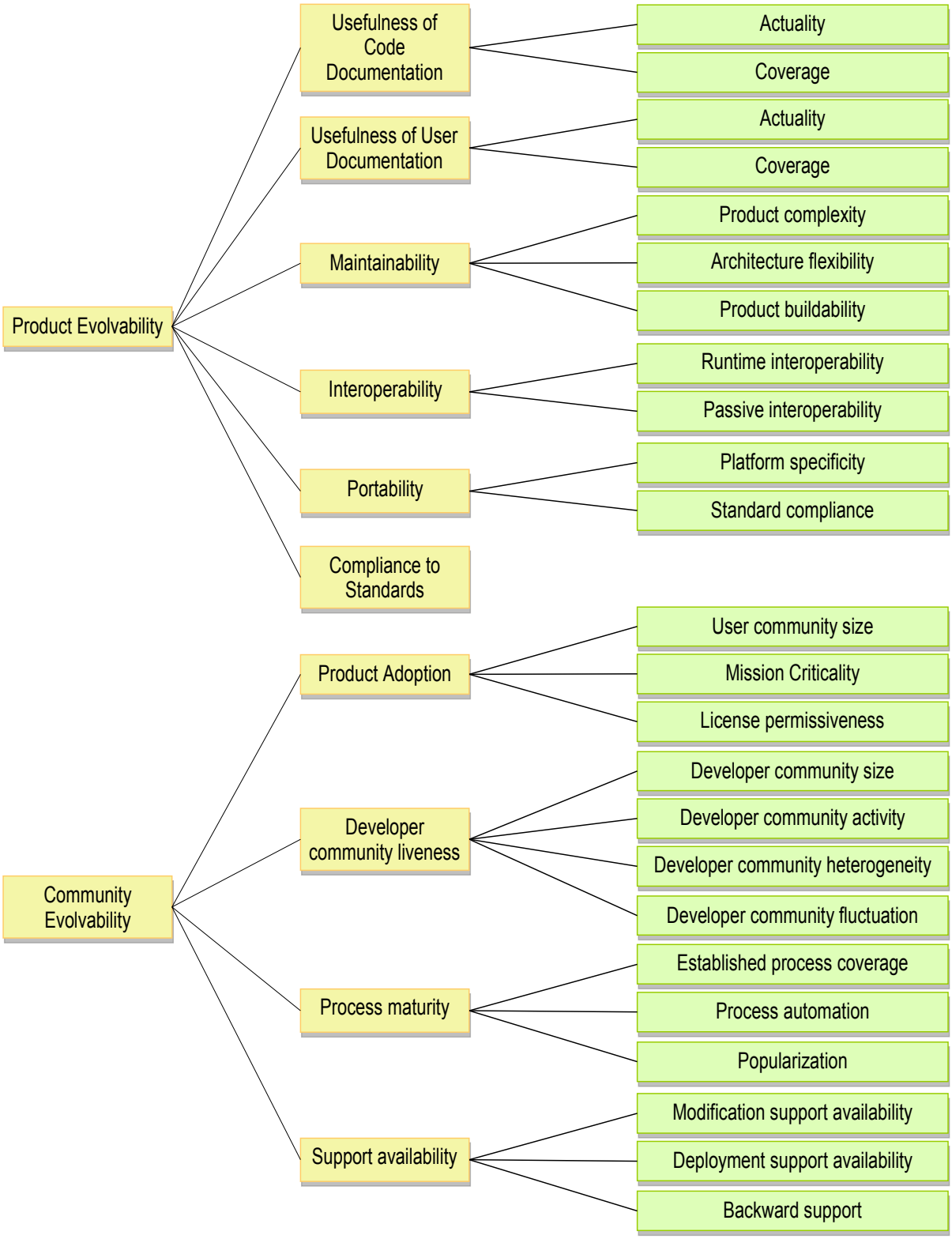


Figure 3: Prototype QualOSS evolvability model

- *Product evolvability:* The ability of a product to be corrected, adapted and extended over time, according to the needs of its users.

- *Usefulness of code documentation*: The extent to which the source code documentation (documentation explicitly describing the product's internals) is useful when performing corrections, adaptations or extensions to the product.
    - *Actuality*: The extent to which the code documentation describes the current version of the source code as opposite to describing older versions of it.
    - *Coverage*: The ratio between size of documented code and general product code size.
    - *Code documentation standard compliance:* The degree to which a product complies with published standards relevant to code documentation.
- *Usefulness of user documentation*: The extent to which the product's user/administrator oriented documentation is useful when deploying and using the product.
    - *Actuality*: The extent to which the user documentation describes the current version of the product functionality as opposite to describing outdated functionality.
    - *Coverage*: The ratio between the number of documented product features and the general number of features offered by the product.
    - *Internationalization*: Availability of the documentation in various natural languages.
    - *user documentation standard compliance:* The degree to which a product complies with published standards relevant to documentation.
- *Maintainability:* The amount of effort required by a programmer or team of programmers with no previous knowledge of the product, to understand its code to the point that successful modifications are possible. IEEE: *The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.*
    - *Product complexity: (IEEE) The degree to which a system or component has a design or implementation that is difficult to understand and verify.*
    - *Architecture flexibility:* The ability of the product's architecture of being applied to new problems. *(IEEE) 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. (*IEEE*) *The ease with which a system or component can be modified to increase its storage or functional capacity.*
    - *Product Buildability: (IEEE) The degree to which a system or component can be rebuild after modifications to the source.*
    - *Fixability: The ease with which a software product can be fixed.*
    - *Maintainability standard compliance:* The degree to which a product complies with published standards relevant to maintainability.
- *Interoperability:*The degree to which a software product can interoperate with other software product either live or based on input/output data.
    - *Runtime Interoperability:* Interoperability with other software products while in operation.
    - *Passive Interoperability*: Interoperability with other software products based on output data generated by the software product or based on the capacity of the software product to read various data types and formats.
- *Portability: (IEEE) The ease with which a system or component can be transferred from one hardware or software environment to another.*
    - *Platform specificity:* The degree to which a product's code is specific to a particular hardware or software environment.
    - *Portability Standard compliance:* The degree to which a product complies with published standards relevant to portability.
- *Compliance to standards*: The degree to which a product complies with published standards that are relevant to its functionality. Important note: for measurement purposes, this criterion is applied separately to various relevant software artefacts, i.e., source code, documentation, etc.

- *Community evolvability:* The likelihood that a F/OSS community remains able to maintain the product or products it develops over an extended period of time.
    - *Product adoption:* The extent to which a F/OSS product is actively used by individuals and organizations around the world.
        - *User community size:* The number of users (individuals and organizations) that use a F/OSS product worldwide.
        - *Strategic importance: (aka. Mission criticality)* 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.

    - *License permissiveness:* The amount of freedom allowed to product users by the product's licence.
- *Developer community liveness:* The amount of work put by a development community into the creation and further development of a software product over a certain period of time.
    - *Developer community size:* The number of individuals and organizations actively contributing to a product's development over a certain period of time.
    - *Developer community activity:* The general number and size of the contributions made to a product's development over a certain period of time.
    - *Developer community heterogeneity:* 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.
    - *Fluctuation:* The rate movement of people into, and out of a developer community over time
- *Process maturity:* The ability of a developer community to achieve development related goals by following established processes. Additionally, the level to which the processes followed by a development community are able to guarantee that certain desired product characteristics will be present in the product.
    - *Established process coverage:* 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.)
    - *Process automation:* 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.
    - *Popularization:* The availability of support related to popularize a software product. The assumption is that a mature project is attempting to popularize its product.
- *Support availability:* The ease with which a user can engage experienced individuals or organizations (on a for-profit or voluntary basis) to perform tasks that make it possible to use a product for a particular purpose.
    - *Modification support availability:* The availability of support related to performing specific modifications to a software product.
    - *Deployment support:*The availability of support related to solving problems arising from the deployment and use of a software product.
    - *Backward Support:* The availability of support related to older version of a software product still in use.

2.2 Robustness

In general, robustness may be studied in a priori or posteriori fashion. A priori analyses study a particular version of the product of interest by searching for weaknesses currently in it that may yield to poor robustness. Posteriori analyses study the software product history to check for old cases where the software exhibited poor robustness and if the community provide adequate solution to solve the causes in timely manner. Figure 4 shows the resulting structure.

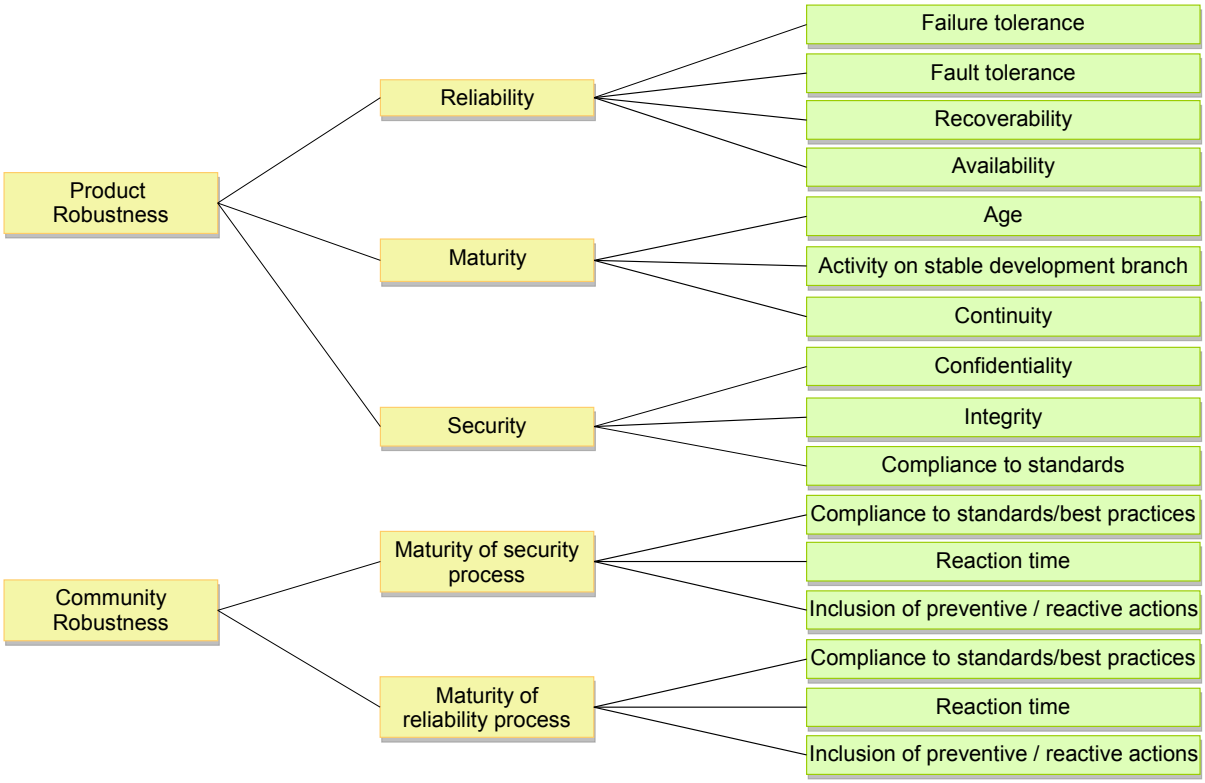


Figure 4: Prototype QualOSS robustness model

- *Product* robustness: (IEEE) The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.
  - *Reliability: (IEEE) The ability of a system or component to perform its required functions under stated conditions for a specified period of time.*
    - *Failure tolerance (ISO 9126: maturity): The capability of the software product to avoid failure as a result of faults in the software.*
    - *Fault tolerance (ISO 9126): 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.*
    - *Recoverability (ISO9126): 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.*
    - *Availability (IEEE): The degree to which a system or component is operational and accessible when required for use*.
  - *Maturity:*The degree to which the general, long-term objectives set for a product have been reached by the current implementation.
    - *Age:* The time span over which a product has been developed.
    - *Activity on stable development branch:* The number and size of the contributions made to a product's stable development branch over a certain period of time. High activity on a branch declared to be stable can be a sign of low product maturity.
    - *Continuity:* The regularity with which community contributions have been made to the a product or in relation to the product over its lifespan.
  - *Security* (ISO 12207): *The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.* This includes measures and controls that ensure confidentiality, integrity, and availability of IS assets including hardware, software, firmware, and information being processed, stored, and communicated (CNSS, 2006).
    - *Confidentiality:* The degree to which a system prevents unauthorized disclosure of information; that is, provides assurance that information is not disclosed to unauthorized individuals, processes, or devices. (CNSS, 2006)

        - *Integrity (ISO): The degree to which a system or component is able to protect the accuracy and completeness of information and processing methods.* This includes preventing unauthorised modification or destruction of information (CNSS, 2006).
        - Compliance to SECURITY standards: The degree to which a product complies with published security standards that are relevant to its functionality.
- *Community robustness: The ability of the established processes in a community to guarantee the delivery of robust products.*
    - *Maturity of security process:* The degree to which a development community has established processes dedicated to guarantee the security of delivered products. Also, the degree to which a community reacts effectively and timely when a security defect is found in a released product.
        - Compliance: The degree to which the processes and procedures dealing with security adhere to best practices and security standards
        - Reaction time: The amount of time that is typically required for resolving security-related issues
        - Inclusion of preventive/reactive actions: The degree to which the community commits to actions aimed at preventing security problems
    - *Maturity of reliability process:* The degree to which a development community has established processes dedicated to guarantee that delivered products are free of critical defects (defects that prevent the operation of the product under common operation conditions). Also, the degree to which a community reacts effectively and timely when a critical defect is found in a released product.
        - Compliance: The degree to which the processes and procedures dealing with reliability adhere to best practices and security standards
        - Reaction time: The amount of time that is typically required for resolving reliability-related issues
        - Inclusion of preventive/reactive actions: The degree to which the community commits to actions aimed at preventing reliability problems

3. Evolvability: Product Quality Model

This section details the metrics identified for measuring the product aspects of the quality model for evolvability. For details, please refer to the appendix. In this and in the following sections, metrics are described using the following format:

Level	*Measurement*	*Tool*
Basic or Advanced	(**UniqueName**) *All basic metrics receive a unique name Brief explanation of the metric* Artefact: *the artefacts / documents needed to compute the metric* Rationale: *A brief rationale why this metric influences the corresponding quality characteristic* Contact*: The project member or organization to contact for more details on the metric*	Either tool names for computing the metrics, or "Manual"

3.1 Usefulness of Code Documentation

Evaluation of quality of documentation is a focus of Section 8.

3.1.1 Actuality

Level	*Measurement*	*Tool*
Basic	**APIDocumentationDateSourceFilesDateDifference** **Difference between** the date of the generated technical API documentation from documentation comments ("javadoc", "docstring") **and** the date of the originating source files. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: API documentation Files, Source Files Rationale: *If the generated API documentation is older than the source files, the chance is high that this API documentation is no more in line with the source code.* Contact: FFM (CETIC)	Manual
Basic	**APIDocumentationDateProjectReleaseDateDifference** **Difference between** the date of the generated technical API documentation from documentation comments ("javadoc", "docstring") **and** the date of the product release. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: API Documentation Files Rationale: *If the generated API documentation is far older than the product release date, the chance is high that this API documentation is no more in line with the real content of the product release.* Contact: FFM (CETIC)	Manual

Level	*Measurement*	*Tool*
Advanced	**APICommentsParameterCompatibilityPercentage** **Ratio between** the number of incompatibilities between input/output declared in documentation comments ("javadoc", "docstring") and actual input/output declared in the code element declaration **and** the total number of input/output declared in the code element declarations Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files Rationale: *If there is a lot of incompatibilities, the chance is high that the documentation comments (and then the API Documentation Files) are no more in line with the actual source code.* Contact: FFM (CETIC)	Manual
Advanced	**OutdatedCodeDocumentationPercentage** **Ratio between** the number of lines of comments commenting outdated code **and** the total number of lines of comments Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files Rationale: *If there is a lot of lines of old code that are commented, this means that these lines of code should be removed, because these are outdated comments. The average ratio comment to code is then lower than originally calculated.* Contact: FFM (CETIC)	Manual
Advanced	**InadequateCodeDocumentationPercentage** **Ratio between** the number of lines of comments not related to the environing code **and** the total number of lines of comments Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files Rationale: *If there is a lot of comment lines that are not in line with the environing code, the comments are not commenting actual code.* Contact: FFM (CETIC)	Manual

3.1.2 Coverage

Level	*Measurement*	*Tool*
Basic	**SourceCodeCommentsPercentage** **Ratio between** the total number of lines of comments **and** the total number of lines of code in the package distribution list. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files Rationale: *If there is a high number of lines of comments, the coverage is higher.* Contact: FFM (CETIC)	Squal GNATmetric

Level	*Measurement*	*Tool*
Advanced	**SufficientlyCommentedFilesPercentage** **SufficientlyCommentedPackagesPercentage** **SufficientlyCommentedClassesPercentage** **SufficientlyCommentedMethodsPercentage** **Ratio between** the number of specific code elements (such as files, packages, classes or methods) whose ratio comment to code is above a threshold **and** the total number of those specific code elements. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages, Classes, Methods Rationale: *The more commented are the code elements, the higher is the code documentation coverage.* Contact: FFM (CETIC)	Squal GNATmetric
Advanced	**APICommentsForPublicClassesPercentage** **APICommentsForPublicMethodsPercentage** **Ratio between** the number of public classes and public methods (or functions) having documentation comments (“javadoc”, “docstring”) **and** the total number of public classes and public methods (or functions) Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages, Classes, Methods Rationale: *The more documentation comments there are, the higher is the code documentation coverage.* Contact: FFM (CETIC)	Squal GNATmetric

3.1.3 Code Documentation Standard Compliance

Level	*Measurement*	*Tool*
Basic	**APICommentsErrorsAverage** **Ratio between** the number of errors encountered in documentation comments respecting the standards **and** the total number of documentation comments. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Lines of Comments Rationale: *If the ratio is low, then the code documentation is more likely to comply with the standards of the documentation comments.* Contact: FFM (CETIC)	Checkstyle

Level	*Measurement*	*Tool*
Advanced	**APIDocumentationStandardCompliance** **Ratio between** the sum of the scores related to the quality of the documented API **and** the maximum score possible for this documented API. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: API Documentation Files Rationale: *See section 8* Contact: FFM (CETIC)	Manual

3.2 Usefulness of User Documentation

Evaluation of quality of documentation is a focus of Section 8.

3.2.1 Actuality

Level	*Measurement*	*Tool*
Basic	**UserDocumentationDateProjectReleaseDateDifference** **Difference between** the date of the user documentation **and** the date of the project release. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: User Documentation Files, Project Release Rationale: *If the user documentation is far older than the product release date, the chance is high that this user documentation is no more in line with the real content of the product release.* Contact: FFM (CETIC)	Manual

3.2.2 Coverage

Level	*Measurement*	*Tool*
Basic	**UserDocumentationAPIDocumentationCommonAbstractionsPercentage** **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. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: User Documentation Files, API Documentation Files Rationale: *If the number of abstractions found in the user documentation that are in common with the abstractions found in the API documentation is low, the chance is high that the user documentation covers a few features offered by the product release.* Contact: FFM (CETIC)	Squal GNATmetric

3.2.3 Internationalization

Level	*Measurement*	*Tool*
Basic	**NumberOfUserDocumentationTranslations** Number of languages in which the User's documentation is correctly translated. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: User Documentation Files *Rationale: If there is a high number of languages in which the User's documentation is translated, the product will have a wider audience.* Contact: FFM (CETIC)	Manual

3.2.4 User Documentation Standard Compliance

Level	*Measurement*	*Tool*
Basic	no basic metrics identified so far; the documentation assessment framework (Section 8) addresses this question	Manual
Advanced	**UserDocumentationStandardCompliance** **Ratio between** the sum of the scores related to the quality of the User Documentation Files **and** the maximum score possible for the User Documentation Files. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: User Documentation Files *Rationale: See section 8* Contact: FFM (CETIC)	Manual

3.3 Maintainability

3.3.1 Product Complexity

Level	*Measurement*	*Tool*
Basic	**FileCyclomaticComplexityAverage** **PackageCyclomaticComplexityAverage** **ClassCyclomaticComplexityAverage** **MethodCyclomaticComplexityAverage** **Ratio between** the sum of the cyclomatic complexity of code elements such as files, packages, classes and methods **and** the total number of lines of code of these code elements. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages, Classes, Methods Rationale: *If the ratio is low, the product is not so complex to understand, and hence to maintain.* Contact: FFM (CETIC)	Squal GNATmetric

Level	*Measurement*	*Tool*
Basic	**HotFilesPercentage** **HotPackagesPercentage** **HotClassesPercentage** **HotMethodsPercentage** **Ratio between** the number of specific code elements (such as files, packages, classes or methods) whose cyclomatic complexities are above a threshold **and** the total number of those specific code elements. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages, Classes, Methods Rationale: *If this ratio is low, this means that the number of code elements to focus on is low, the product release is then easier to maintain.* Contact: FFM (CETIC)	Squal GNATmetric
Basic	**MethodUnderstandabilityAverage** **Average ratio between** the cyclomatic complexity of methods **and** their percentage of comments. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Methods Rationale: *If this ratio is low, this means that complex methods are well documented, then the product release is not so hard to maintain.* Contact: FFM (CETIC)	Squal GNATmetric
Basic	**MethodLinesOfCodeAverage** **Average** number of lines of code per method. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Methods Rationale: *If the number of lines of code per method is high, this means that the methods are quite long, and hence less easy to grasp.* Contact: FFM (CETIC)	Squal GNATmetric
Basic	**ClassNumberOfMethodsAverage** **Average** number of methods per class Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Methods, Classes Rationale: *If the number of methods per class is high, this means that the interfaces of the classes are rather difficult to grasp.* Contact: FFM (CETIC)	Squal GNATmetric

Level	*Measurement*	*Tool*
Advanced	**ClassDepthOfInheritanceAverage** **Average** depth of inheritance tree for a class Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Classes Rationale: *If average depth of inheritance tree for a class is high, this means that the classes are deep in the inheritance tree. Thus to grasp the behaviour of a method in these classes, the whole ancestors needs to be mastered.* Contact: FFM (CETIC)	Squal GNATmetric
Advanced	**MethodComplexityCouplingAverage** **Average ratio between** the cyclomatic complexity of methods **and** their efferent coupling. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Methods Rationale: *If this ratio is high, this means that complex methods does not require the understanding of the inners of a lot of different concepts, then the product release is not so hard to maintain.* Contact: FFM (CETIC)	Squal GNATmetric

3.3.2 Architecture Flexibility

Level	*Measurement*	*Tool*
Basic	**APIDocumentationExistence** **DeveloperDocumentationExistence** Presence of API Documentation Files or Technical Programmer's Guide. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If there are API Documentation Files or Technical Programmers Guide, the application will be easier to extend.* Contact: FFM (CETIC)	Manual
Basic	**ThirdPartyPlugInPossibility** Existence of third-party plug-ins. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If there are third-party plug-ins, the architecture of the application is clearly extensible.* Contact: FFM (CETIC)	Manual

Level	*Measurement*	*Tool*
Basic	**ProductConfigurationFilePossibility** Existence or Use of configuration files and properties (look for usage patterns) to enable/disable functionalities. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If there are configuration files, the application has the ability to be easier tailored to new environments.*	Manual
Advanced	**PackagePrivacyAverage** **Average** privacy of the packages. The privacy being computed as the ratio between the number of public methods and the number of private methods of the packages. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages, Methods Rationale: *If the average privacy of the packages is high, the original architecture will be easier reused in new contexts because many functionalities will be publicly available.* Contact: FFM (CETIC)	Squal
Advanced	**PackageAbstractnessAverage** **Average** abstractness of the packages. The abstractness being computed as the ratio between the number of interfaces or abstract classes and the number of concrete classes. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the average abstractness is high, it will be easier to add new functionalities by extending the abstract classes found in the original packages of the product release.* Contact: FFM (CETIC)	Squal
Advanced	**PackageAfferentCouplingAverage** **Average** afferent coupling of the packages in the product release. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the average afferent coupling is high, the packages are heavily used, the more a package is relied on, the less likely it is to change. The flexibility of the architecture will then be low if this average is high.* Contact: FFM (CETIC)	Squal

Level	*Measurement*	*Tool*
Advanced	**PackageEfferentCouplingAverage** **Average** efferent coupling of the packages in the product release. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the average efferent coupling is high, the product depends on a lot of external libraries. The product release is then more difficult to extend.* Contact: FFM (CETIC)	Squal
Advanced	**PackageInstabilityAverage** **Average** instability of packages. The instability being computed as the ratio between the efferent coupling of package and the sum of its efferent and afferent couplings. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the average instability of the packages is high, the architecture will be hard to maintain, because a tailoring of a package will make the architecture unstable, and will generate many cascading changes.* Contact: FFM (CETIC)	Squal
Advanced	**PackageLackOfCohesionAverage** **Average** lack of cohesion of the packages Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the average cohesion of the packages is low, the product will be hard to maintain because the product depends on a lot of external libraries. Its flexibility will be low.* Contact: FFM (CETIC)	
Advanced	**PackageNumberOfCyclesAverage** **Average** the number of cycles in the packages in the product release. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the ratio is high, there is a lot of cycles per package. The architecture of the product release is not less flexible because the modification of one package in a cycle has a potential impact on all the packages involved in the cycle, hence.* Contact: FFM (CETIC)	Jdepend

Level	*Measurement*	*Tool*
Advanced	**PackageArchitecturalAntiPatternAverage** **Average** number of architectural anti-patterns found per package. Example of anti-patterns being “Blob”, “Envy”, ... Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If these anti-patterns are found, the architecture of the application suffers from drawbacks that reduce its extensibility.* Contact: FFM (CETIC)	Semi-Manual
Advanced	**APIDocumentationStandardCompliance** **DeveloperDocumentationStandardCompliance** Quality of the API Documentation Files or the Technical Programmer's Guide. Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *If the quality of the technical documentation is low, the application will be more difficult to extend.* Contact: FFM (CETIC)	Semi-Manual
Advanced	**PackageNumberOfGoodDesignPatternsAverage** Average number of used of good architecture design patterns per package Source: Package Distribution Lists, Version Control Repositories Artefact: Source Files: Packages Rationale: *In the same sense as the detection of anti-patterns, detecting good design patterns, eases the possible extension of the application.* Contact: FFM (CETIC)	Semi-Manual

3.3.3 Fixability

Level	*Measurement*	*Tool*
Basic	**IssueOpenToCloseTimeAverage** **Average** time needed to close an issue. That is, the average difference between the date of the creation of the issue and the first date where a status such as CLOSE, FIXED, WONTFIX or SOLVED is assigned to this issue. Source: Issue Tracking Systems Artefact: Issue Rationale: *If the average time is low, this means that the correction of a bug is easy. This product release seems then easier to maintain.* Contact: FFM (CETIC)	Semi-Manual

3.3.4 Maintainability Standard compliance

Level	*Measurement*	*Tool*
Basic	**ProductNamingConventionErrorsPercentage** **Ratio between** the number of code style errors related to naming conventions **and** the total number of code style errors Source: Package Distribution Lists Artefact: Source Files Rationale: *If the ratio is high, the product release is not standard compliant.* Contact: FFM (CETIC)	Semi-manual
Advanced	**ProductMaintainabilityStandardCompliance** **Ratio between** the number of design patterns effectively used **and** the total number of design patterns recommended Source: Package Distribution Lists Artefact: Source Files Rationale: *If the ratio is low, the product release is not standard compliant.* Contact: FFM (CETIC)	Semi-manual

3.4 Interoperability

3.4.1 Runtime Interoperability

Level	*Measurement*	*Tool*
Basic	**ProductReleaseNumberOfRuntimeExchangeFormats** Total number of formats exchanged at runtime by the application or listed in the TODO Lists Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If the number of formats is high, the application will be easier to work with, hence it is more likely to evolve easily as wanted by the user.* Contact: FFM (CETIC)	Manual
Basic	**ProductReleaseNumberOfStaticExchangeFormats** Total number of formats exchanged in a static way by the application or listed in the TODO Lists Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If the number of formats is high, the application will be more convenient to work with, hence it is more likely to evolve easily as wanted by the user.* Contact: FFM (CETIC)	Manual

Level	*Measurement*	*Tool*
Basic	**ProductReleaseRuntimeExchangeFormatsRatio** **Ratio between** the number of exchange formats dealt with at runtime or listed in the TODO Lists **and** the number of formats dealt with in a static way Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If the ratio is low, then the application is less like to evolve easily in the way wanted by the user.*	Manual
Advanced	**ProductReleaseRuntimeExchangeFormatsCompliance** **Ratio between** the weighted number of matching exchange formats actually dealt with by the application at runtime or listed in the TODO Lists **and** the weighted number of formats wished by the user at runtime Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If the ratio is low, then the application is less like to evolve easily in the way wanted by the user.*	Manual

3.4.2 Passive Interoperability

Level	*Measurement*	*Tool*
Basic	**DocumentationInteroperabilityPresence** User Documentation Files and/or Technical Documentation Files have sections about Interoperability. Source: Package Distribution Lists, Version Control Repositories, Websites Artefact: User Documentation Files, Technical Documentation Files Rationale: *If yes, the product is more likely to evolve easily.* Contact: FFM (CETIC)	Manual
Basic	**ProductReleaseNumberOfExchangedFormats** Total number of exchange formats that the application can deal with or that are listed in the TODO Lists. Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If there is a low number of exchange formats, the product is less likely to evolve easily.* Contact: FFM (CETIC)	Manual

Level	*Measurement*	*Tool*
Basic	**ProductReleaseNumberOfOpenExchangeFormats** Total number of open exchange formats that the application can deal with or that are listed in the TODO Lists. Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If there is a low number of open exchange formats, the product is less likely to evolve easily.* Contact: FFM (CETIC)	Manual
Basic	**ProductReleaseNumberOfCommunicatingApplications** Total number of applications that the application can communicate with or that are listed in the TODO Lists. Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If there are a few such applications, the product release is less likely to evolve easily.* Contact: FFM (CETIC)	Manual
Advanced	**ProductReleaseExchangeFormatsCompliance** **Ratio between** the weighted number of matching exchange formats actually dealt with by the application or listed in the TODO Lists **and** the weighted number of formats wished by the user Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If the ratio is low, then the application is less like to evolve easily in the way wanted by the user.* Contact: FFM (CETIC)	Manual
Advanced	**ProductReleaseCommunicatingApplicationsCompliance** **Ratio between** the weighted number of matching application with which the application actually communicates or that are listed in the TODO Lists **and** the weighted number of applications wished by the user Source: Package Distribution Lists, Version Control Repositories, Website Artefact: User Documentation Files, Technical Documentation Files Rationale: *If the ratio is low, then the application is less like to evolve easily in the way wanted by the user* Contact: FFM (CETIC)	Manual

3.5 Portability

3.5.1 Platform Specificity

Level	*Measurement*	*Tool*
Basic	**ProductReleaseHighlyPortableProgrammingLanguageUsed** Used programming language (Portable: Java > Python > Perl, Less protable: C++ > C) Source: Package Distribution Lists Artefact: Source Files Rationale: *It the programming language of the product release is Java, it is more evolvable than if it is C.* Contact: FFM (CETIC)	Manual
Advanced	**ProductReleaseUseOfStandardLibrariesPercentage** **Ratio between** the number of standard libraries used **and** the total number of libraries used in the source code of the project release. Source: Source files in package distribution list Rationale: *If the ratio is high, the product release is not reduced to a platform.* Contact: FFM (CETIC)	Semi-manual
Advanced	**ProductReleaseUseOfSpecificLibrariesPercentage** **Ratio between** the number of specific platform libraries used **and** the total number of libraries used. Ex: use of OLEDB, SQLServer instead of JDBC Source: Source files in package distribution list. Rationale: *If the ratio is high, the product release is highly reduced to a platform.* Contact: FFM (CETIC)	Semi-manual

3.5.2 Portability Standard compliance

Level	*Measurement*	*Tool*
Basic	**ProductReleaseCodeStyleErrorsAverage** **Ratio between** the total number of code style errors reported **and** the total number of lines of code Source: Source files in Package Distribution List Rationale: *If the ratio is high, the product release is not standard compliant.* Contact: FFM (CETIC)	Manual

Level	*Measurement*	*Tool*
Advanced	**ProductReleaseCodeNamingConventionErrorsPercentage** **Ratio between** the number of code style errors related to naming conventions **and** the total number of code style errors Source: Source files in Package Distribution List Rationale: *If the ratio is high, the product release is not standard compliant.* Contact: FFM (CETIC)	Semi-manual
Advanced	**ProductReleaseUsedDesignPatternsPercentage** **Ratio between** the number of design patterns effectively used **and** the total number of design patterns recommended Source: Source files in package distribution list Rationale: *If the ratio is low, the product release is not standard compliant.* Contact: FFM (CETIC)	Semi-manual

4. Evolvability: Community Quality Model

This section details the metrics identified for measuring the community aspects of the quality model for evolvability. In parts, this is addressed by the process and document assessment frameworks in Sections 7 and 8.For details on the metrics listed in this section, please refer to the appendix.

4.1 Product Adoption

4.1.1 User Community Size

Level	*Measurement*	*Tool*
Basic	(***NumOfDevelopers***) Number of developers who have made commits. Source: Version Control Repositories Artefact: Author Rationale: Commiters in a project are (generally speaking) also users of this product. Developers can be identified by their commits. Contact: DI/ACS (URJC)	CVSAnaly
Basic	(***NumOfPostersMailingLists***) Number of people writing in the several lists of the project. Source: Mailing List Archive Artefact: Original Message ID, Date, Subject Rationale: Users can be identified because they write messages in Mailing Lists. Contact: DI/ACS (URJC)	MLStats
Advanced	Number of people writing in forums. Source: Discussion Forum (accessible via a Web or News server) Artefact: Name, email of poster Rationale: Users can be identified because they write messages in forums Contact: DI/ACS (URJC)	Manual
Advanced	Number of people participating in the IRC Source: IRC log Artefact: Name, nickname. Rationale: Users can be identified because they use the IRC. Contact: DI/ACS (URJC)	Manual

4.1.2 Mission Criticality

Level	*Measurement*	*Tool*
Basic	No metrics identified so far.	
Advanced		

4.1.3 License permissiveness

Level	*Measurement*	*Tool*
Basic	(***LicenseUsedSourceCode***) Type of licenses used by the project source code Source: Product Distribution List Artefact: Source Code Rationale: Source code files, usually, contain information regarding to license	Manual

	used in the project. Contact: DI/ACS (URJC)	
Basic	(***LicenseUsedDocumentation***) Type of licenses used by the project documentation Source: Documentation files, information in web site and even Mailing List Archive, Discussion Forum, Website or IRC Logs. Rationale: Documentation has its own license. Contact: DI/ACS (URJC)	Manual
Advanced	No Advanced Metrics identified so far	

4.2 Developer Community Liveliness

4.2.1 Developer Community Size

Level	*Measurement*	*Tool*
Basic	(***TotalNumOfDevelopers***) Number of developers who have made changes in the project in the whole life of the project. Source: Version Control Repositories Artefact: Author Rationale: Total number of developers is necessary to measure how big is the community. Contact: DI/ACS (URJC)	CVSAnaly
Basic	(***PastNumOfDevelopers***) Number of developers who had made commits in the project in an exact date in the past. Source: Version Control Repositories Artefact: Author Rationale: Number of developers working in the past is useful to know how big was the community in the past. Contact: DI/ACS (URJC)	
Basic	(***EvolutionOfNumOfDevelopers***) Evolution of number of developers along the life of the project. Source: Version Control Repositories Artefact: Author Rationale: Evolution of num of developers is extremely useful to know the evolution of the community of developers. Contact: DI/ACS (URJC)	
Basic	(***TotalNumOfNonActiveDevelopers***) Number of non-active developers Source: Version Control Repositories Artefact: Author Rationale: Knowing number of non-active developers (we can consider as non-active developer a person who has not made commits in more than six months) nowadays for a project is useful to guess how big is this group of non-active developers regarding the total. Contact: DI/ACS (URJC)	CVSAnaly
Basic	(***TotalNumOfNonActiveDevelopersInPast***) Number of non-active developers in an exact date in the past. Source: Version Control Repositories Artefact: Author	

	Rationale: How large the non-active developers group was in an exact date in the past. Contact: DI/ACS (URJC)
Basic	(***EvolutionNumOfNonActiveDevelopers***) Evolution of number of non-active developers. Source: Version Control Repositories Artefact: Author Rationale: Its own name is auto-explicative. Contact: DI/ACS (URJC)
Basic	(***TotalNumOfActiveDevelopers***) Number of active developers at the present time. Source: Version Control Repositories Artefact: Author Rationale: How big the group of active developers is. Contact: DI/ACS (URJC)
Basic	(***TotalNumOfActiveDevelopersInPast***) Number of active developers in an exact date in the past. Source: Version Control Repositories Artefact: Author Rationale: How big the group of active developers was. Contact: DI/ACS (URJC)
Basic	(***EvolutionNumOfActiveDevelopers***) Evolution of active developers Source: Version Control Repositories Artefact: Author Rationale: Its own name is auto-explicative. Contact: DI/ACS (URJC)
Advanced	None

4.2.2 Developer Community Activity

Level	*Measurement*	*Tool*
Basic	(***NumOfChangesToSource***) Number of changes (commits) made in the source code until now. Artefact: Comment Log Source: Version Control Repositories Rationale: This activity is basic for developers. Contact: DI/ACS (URJC)	CVSAnaly
Basic	(***EvolutionOfChangesToSource***) Activity of version control systems (number of commits per month or year, it depends on how big the project is) Source: Version Control Repositories Artefact: Historical Analyses specific to Version Control Data Rationale: Its own name is auto-explicative. Contact: DI/ACS (URJC)	CVSAnaly

Level	Measurement	Tool
Basic	(***NumOfMessagesOfDevelopers)*** Developers Mailing list activity. Number of messages. Source: Mailing List Archive Artefact: Original Message ID, Date, Subject Rationale: Discussions, generally speaking in big projects, are taken in public mailing lists. For instance, developers take design decisions there. Contact: DI/ACS (URJC)	MLStats
Basic	(***EvolutionMessagesOfDevelopers***) Evolution in the number of messages from developers in a mailing list. (Number of messages per month or year, it depends on how large the mailing lists are.) Source: Mailing List Archive Artefact: Historical Analyses specific to Mailing List Archive Rationale: Its own name is auto-explicative. Contact: DI/ACS (URJC)	MLStats
Advanced	Developers Forum activity. Number of reply posts. Source: Discussion Forum Artefact: Thread of answers Rationale: Instead of using mailing list, developers can use forums. Contact: DI/ACS (URJC)	No tool
Advanced	Evolution in the number of reply posts in developers forums Source: Discussion Forum Artefact: Historical Analyses specific to Discussion Forum. Rationale: Its own name is auto-explicative. Contact: DI/ACS (URJC)	No tool
Advanced	Number of participants in all IRC logs Source: IRC logs Artefact: Participants Rationale: IRC is another place where developers take decisions Contact: DI/ACS (URJC)	No tool

4.2.3 Developer Community Heterogeneity

Level	Measurement	Tool
Basic	(***PeopleOnFiles***) Number of people working in the same group of files. Source: Version Control Repository Artefact: Set of Files Rationale: People can work on the same files or in different files. It is better if a developer is specialized in their own files. If there is a bug, it is likely he/she will be able to solve it before than others. Contact: DI/ACS (URJC)	CVSAnaly
Basic	(***PeopleOnGroupsOfFiles***) Number of people working in several group of files. Source: Version Control Repository Artefact: Set of Sets of Files Rationale: People more specialized is better for the group. CVSAnalY is able to detect if people are working in a group of files or another group of files. Thus, it is better to have people specialized, for instance in translation activities.	CVSAnaly

	Contact: DI/ACS (URJC)	
Advanced	Betweeness: identification of people in the project with a high knowledge of it. Source: Version Control Repository Artefact: Author Rationale: Betweeness is a social network metric. It measures if a person is between two big social network groups. In this case, it measures if a person is working on files where people, usually, are working only in one of them. It means, this persons is important for the project because he/she has a knowledge of the two groups of files and he can modify them. Contact: DI/ACS (URJC)	CVSAnaly + scripts + Conan
Advanced	Community composition: Number of developers, number of companies working in the project, contributions (in terms of lines of code) by companies .... Source: Version Control Repository, Mailing List Rationale: These are several metrics which must be analysed by several tools and it is not clear if they are useful for measuring quality. However they are quite interesting for other companies and individual authors. Contact: DI/ACS (URJC)	CVSAnaly, Pyternity and others

4.2.4 Developer Community Fluctuation

Level	*Measurement*	*Tool*
Basic	(***DevRegeneration***) Developer regeneration Source: Version Control Repository Artefact: Comment Log Rationale: It is necessary to know if there are new developers working on the project. Developers in FLOSS projects work as a volunteers (generally), thus because of several causes they decide to leave the project. Hence, new developers are basic for the project. Regeneration measures if this being made in a good way, or on the other hand, there are some periods without activity in the project. (at least in the core group). Contact: DI/ACS (URJC)	CVSAnaly
Advanced	Number of people who are not developers but they provide ideas in developers mailing lists. Perhaps this is an idea near of regeneration of developers because these persons in the future could be developers. Source: Mailing List Archive Artefact: Name, email of poster Rationale: There is a process of regeneration in all the projects, some people leave the project and some others enter as new people. Some of the last one can be detected and a study of their work in the project, from mailing lists to commits is useful. Contact: DI/ACS (URJC)	MLStats + CVSAnaly

4.3 Process Maturity

4.3.1 Established Process Coverage

Level	*Measurement*	*Tool*
Advanced	The degree to which best practices of F/OSS projects are implemented. This	Manual

	metric uses the **process assessment framework** (see Section 7) Artefact: User Documentation Files, Project Release, project website Rationale: *Maturity of the community/project is usually perceived in terms of how well the project implements best practice processes.* *Contact:* MC/MS (IESE)	
Advanced	Idea for an advanced metric is to study the political structure of the community, a great example here is Apache, Mozilla or OpenOffice community. Artefact: User Documentation Files, Project Release, project website Rationale: *Maturity of the community/project also determined by its management structure.* *Contact:* MC/MS (IESE)	Manual

4.3.2 Process Automation

Level	*Measurement*	*Tool*
Basic	(***ToolSupport***) Tools used within the project (Result: List of tools) Artefact: User Documentation Files, Project Release, project website, community composition Rationale: To a certain degree, tool support is necessary for effective collaboration in F/OSS projects. Contact: DI/ACS (URJC)	Manual
Advanced	Manual methods (looking for administrative structure and tools like BTS, CVS or similar. Contact: DI/ACS (URJC)	Manual

4.3.3 Popularization

Level	*Measurement*	*Tool*
Advanced	The question of whether organisations and processes exist to foster popularization of the projects can be assessed using the **process assessment framework** (see Section 7) Artefact: User Documentation Files, Project Release, project website, community composition Rationale: *Maturity of the community/project is usually perceived in terms of how well the project implements best practice processes.* *Contact:* MC/MS (IESE)	Manual

4.4 Support Availabilty

4.4.1 Modification Support Availability

Level	*Measurement*	*Tool*
Advanced	Functionality suggested by users are implemented by developers? How long	CVSAnalY +

	does it takes? Source: Version Control Repository Artefact: Comment Log Contact: DI/ACS (URJC)	Mailing list stats
Advanced	How many developers reply messages in mailing lists oriented to users? Number of developers replying in mailing lists regarding total number of developers accessing CVS. Source: Mailing List Archives Artefact: Thread of answers Rationale: Measure activity in mailing lists and we can try to look for developers who participate in these mailing lists in order to provide great information to normal users. Contact: DI/ACS (URJC)	MLStats
Advanced	Evolution of number of answers in Mailing List Source: Mailing List Archives Artefact: Historical Analyses specific to Mailing List Data Contact: DI/ACS (URJC)	MLStats
Advanced	How many developers reply posts in forum to users? Number of developers replying in forum regarding total number of developers accessing CVS. Source: Discussion Forum Artefact: Thread of answers Contact: DI/ACS (URJC)	No tool
Advanced	Evolution of number of answers in Discussion forum Source: Discussion Forum Artefact: Historical Analyses specific to Discussion Forum Contact: DI/ACS (URJC)	No tool

4.4.2 Deployment Support Availability

Level	*Measurement*	*Tool*
Advanced	Deployment functionality suggested by users are implemented by developers? How long does it takes? Source: Version Control Repository Artefact: Comment Log Contact: DI/ACS (URJC)	CVSAnalY + Mailing list stats
Advanced	How many developers reply messages in mailing lists oriented to users for deployment issues?	MLStats

	Source: Mailing List Archives Artefact: Thread of answers Contact: DI/ACS (URJC)	
Advanced	Evolution of number of answers in Mailing List for deployment issues. Source: Mailing List Archives Artefact: Historical Analyses specific to Mailing List Data Contact: DI/ACS (URJC)	MLStats
Advanced	How many developers reply posts in forum to users asking for deployment problems? Source: Discussion Forum Artefact: Thread of answers Contact: DI/ACS (URJC)	No tool
Advanced	Evolution of number of answers in Discussion forum for deployment issues. Source: Discussion Forum Artefact: Historical Analyses specific to Discussion Forum Contact: DI/ACS (URJC)	No tool

4.4.3 Backward Support

Level	*Measurement*	*Tool*
Basic	No basic Metrics Identified.	
Advanced	No basic Metrics Identified so far. To be done in D1.5	

5. Robustness: Product Quality Model

This section details the metrics identified for measuring the product aspects of the quality model for robustness. For details, please refer to the appendix.

5.1 Reliability

5.1.1 Failure Tolerance

Definition: The capability of the software product to avoid failure and to maintain a specified level of performance when software faults are executed.

Level	*Metrics/Analyses*	*Tool*
Basic	• (***TotalIssuesAllReleases***) Total Number of issues for all releases • (***ResolvedIssuesAllReleases***) Number of issues for all releases whose resolution flag has been assigned a value • (***RatioResolvedIssuesAllReleases*** = *ResolvedIssuesAllReleases* / *TotalIssuesAllReleases*) For all releases, ratio of number of issues whose resolution flag has been assigned a value over total of all issues for all releases • (***TotalIssuesSubsetReleases***) Total number of issues for a specific set of releases (e.g., a singleton of a single release or a subset of releases within the same major release number) • (***ResolvedIssuesSubsetReleases***) Number of issues for a specific set of releases whose resolution flag has been assigned a value • (***RatioIssuesSubsetReleases*** = *ResolvedIssuesSubsetReleases* / *TotalIssuesSubsetReleases*) For a specific set of releases, ratio of number of issues whose resolution flag has been assigned a value over total number of issues. • (***CrashIssuesAllReleases***) Number of issues for all releases whose title, description or additional comments containing the word “CRASH” • (***CrashIssuesSubsetReleases***) Number of issues for a specific set of releases whose title, description or additional comments containing the word “CRASH” Source: Issue Tracking Database Artefact: Set of Issues (set may vary depending on the query used, e.g., set of issues related to a single specific release or to all releases.) Rationale: The number of issues and their ratio between solved and still open issues definitely indicate the level of robustness of the product. Contact: JCD – CETIC	Manual: Advanced Search in Issue Tracking System
Basic	• (***VulAllReleases***) Number of all exposures and vulnerabilities for all releases of a software product • (***SeverVulAllReleases***) Number of sever exposures and vulnerabilities for all releases of a software product (sever = vulnerabilities that could yield to system crash or control being taken by an outsider) • (***VulSubsetReleases***) Number of all exposures and vulnerabilities for a specific subset of releases of a software product (can be a singleton or more) • (SeverVulSubsetReleases) Number of sever exposures and vulnerabilities for a specific subset of releases of a software product Source: Security databases Artefact: Set of Vulnerabilities and Exposures	Manual: Advanced Search of NVD

Level	*Metrics/Analyses*	*Tool*
	Rationale: Vulnerabilities and exposures identified by users identify clearly failures observed under real usage scenarios Contact: JCD – CETIC	
Basic	• (***CrashMessage***) Number of messages whose subject line contains the word CRASH. Source: Discussion Archive Artefact: Set of Mails Rationale: Some FOSS projects do not use ITS and instead use mailing lists as a mean of communication between community members. In such cases, failures are reported by emails. Contact: JCD – CETIC	Manual: Search of pages with discussion archive
Advanced	Analysis to detect potential runtime failures: deadlocks, memory leaks, illegal memory accesses (array out of bound, dangling pointers, double free), ... Source: Distributions List Artefact: • Executable files of a single specific release or version OR • Source files of a single specific release or version Rationale: Potential runtime errors lead to an unstable state which often leads to failure. Contact: JCD – CETIC	Valgrind, GNATstack, GNATmem, GNATcheck, Jlint, ...
Advanced	Analysis of error handling in source code Source: Distributions List Artefact: Source files of a single specific release or version Rationale: if exception are not handle properly in the code, it may lead to failure Contact: JCD – CETIC	Augmented JavaAnalyzer
Advanced	List of Environments on which a FOSS distribution release was tested as described in documentation Source: Distributions List Artefact: Documentation Files Rationale: Argument: The more environments the product was tested on, the more reliable it is likely to be on this environment) Contact: JCD – CETIC	Manual
Advanced	Test coverage: percentage of classes, methods, basic block covered by tests Source: Distributions List Artefact: Executable files of a single specific release or version OR	Emma, GCOV

Level	Metrics/Analyses	Tool
	Source files of a single specific release or version Rationale: The more tests cover the code, the more are failures likely to be identified during testing and therefore addressed before release. Contact: JCD – CETIC	
Advanced	Run full test suite on executables and collect test log then determine the number of software failures in test log (Analysis for crash or unexpected test failures evidences in log). Possibility to directly analyse logs of nightly builts if available. Source: Distributions List Artefact: Test Suite AND/OR Nightly-built test logs of a single release Rationale: If tests show to much of an evidence of lack of robustness then that is bad; however, a certain number of failure shows that the test suite is good. Contact: JCD – CETIC	Manual (automation of test script execution)
Advanced	Historical variation of number of empty catch bloc, bad use of throws (and of potential runtime error) in software product within the same major release. Source: Historical Analyses specific to Product Distribution Artefact: Source Files of a set of releases Rationale: Argument: steady improvement in error checking over time show dedication by developers to improve robustness Contact: JCD – CETIC	To build
Advanced	Historical variation of code coverage obtained by testing. Source: Historical Analyses specific to Product Distribution Artefact: Source Files + Test Scripts for a set of releases (most likely children of a major release) Rationale: A steady improvement and then a maintained high code coverage by tests shows high level of dedication by testers to maintain and improve robustness. Contact: JCD – CETIC	Test Coverage Tools + Historical Analysis of test coverage results
Advanced	• (IssuesResolvedNotReopenAllReleases) Number of issues for all releases whose resolution flag has been assigned a value and where Status has not been set to REOPEN afterward • (IssuesResolvedNotReopenSubsetReleases) Number of issues for a specific set of releases whose resolution flag has been assigned a value and where Status has not been set to REOPEN afterward Source: Issue Tracking Database Artefact: Set of Issues (set may vary depending on the query used, e.g., set of issues related to a single specific release or to all releases.) Rationale: The number of issues and their ratio between solved and still open issues definitely indicate the level of robustness of the product. Contact: JCD – CETIC	Manual: Advanced Search in Issue Tracking System

Level	*Metrics/Analyses*	*Tool*
Advanced	Historical variation of issues (alternatively only those containing the word CRASH.) This curve should fit with the logarithmic function. Source: Issue Tracking System Artefact: Historical data on set of issues Rationale: Logarithmic decrease in number of failures reported indicate a good trend in relation to improving robustness. Contact: JCD – CETIC	Historical analysis of issue tracking database
Advanced	Historical variation of issues (alternatively only those containing the word CRASH.) This curve should fit with the logarithmic function. Number of undetected defects left in specific version as predicted by statistical analysis of full history of issues Source: Issue Tracking System Artefact: Historical data on set of issues Rationale: statistical prediction based on the past often yield a good indicator for the future. Contact: JCD – CETIC	Manual: Advanced Search of Issue Tracking System + Manual
Advanced	Number of messages whose *content body* contains the word CRASH Source: Discussion Archive Artefact: Set of Mails Rationale: Some FOSS projects do not use ITS and instead use mailing lists as a mean of communication for the community members hence failures are reported by emails. The equivalent basic metric only searches for CRASH in title however that may not be sufficient and the body of the email may also need to be searched. (unfortunately, when done by hand, this requires searching message by message) Contact: JCD – CETIC	To build
Advanced	Number of defect discovered during code review (This metrics assumes that a source code review process is explicitly described as part of the verification process and that recording of review results is required) Source: Textual Documents Artefact: Project Website Rationale: Code review has shown to be one of the most efficient mechanism to discover defects in code) Contact: JCD – CETIC	Manual
Advanced	• Number of vulnerabilities with patches proposed as compared to all vulnerabilities	Manual: Advanced

Level	*Metrics/Analyses*	*Tool*
	• Average time between vulnerabilities posted and patch provided Source: Security databases Artefact: Set of Vulnerabilities and Exposures Rationale: Risk of vulnerabilities alleviated if patch exists. Moreover this risk decreases as patches are made available in a timely manner. Contact: JCD – CETIC	Search of NVD
Advanced	(The analysis below is maybe more related to Community Robustness Process) Verify that test and other QA tools e.g., styles checkers, are ran automatically when committing changes to the version control repository. Source: Version Control ~~Repository~~ Configuration Artefact: Version Control Configuration Rationale: Applying QA checks before granting a commit improves the quality of the code and will help avoid failures in the long run. Contact: JCD – CETIC	To build or Manual

5.1.2 Fault / Error Tolerance

Definition: The capability of the software product to avoid failures and to maintain a specified level of performance in cases of infringement of its specified interface.

Level	*Measurement*	*Tool*
Advanced	Check for Code/SQL injection (Additional code check related to fault tolerance will be identified) Source: Distributions List Artefact: Source Files Rationale: Injection are cases of malicious infringement to the specified interface Contact: JCD – CETIC	LAPSE, JavaAnalyzer
Advanced	Methods/Classes that catch generic exceptions. Source: Distributions List Artefact: Source or Executable Files Rationale: catching generic exception make it much harder to react appropriately to go from unstable back to a stable state of execution Contact: JCD – CETIC	LAPSE, JavaAnalyzer
Advanced	Check User-Interface libraries used Source: Distributions List Artefact: Source or Executable Files Rationale: Some UI are more fault tolerant than other, e.g., such as java GUI is fault tolerant, in cases where exceptions are raised, the UI usually stays up and running only dumping stack trace on background console Contact: JCD – CETIC	LAPSE, JavaAnalyzer

Level	Measurement	Tool
Advanced	Check that test for Code and SQL injection exist Source: Distributions List Artefact: Test Files Rationale: If test cases exist for injection, it is likely to be identified before releases) Contact: JCD – CETIC	Manual
Advanced	Installation, Administration, and or User documentation explain the environment and scope in which the product was tested and remains functional even in case of some failures. Source: Distributions List Artefact: Documentation Files Rationale: If the user is informed of the framework to stay in, he is likely to be less frustrated when crashes occur due to uses outside the foreseen scope. Contact: JCD – CETIC	Manual

5.1.3 Recoverability

Level	Measurement	Tool
Basic	• (***TotalIssuesAllReleases***) Total Number of issues for all releases • (***RecoverIssuesAllReleases***) Number of issues for all releases containing RECOVER in the title, description or additional comments • (***RatioRecoverIssuesAllReleases*** = *RecoverIssuesAllReleases / TotalIssuesAllReleases*) For all releases, ratio of number of issues containing RECOVER in the title, description or additional comments over total number of all issues • (***TotalIssuesSubsetReleases***) Total number of issues for a specific set of releases (e.g., a singleton of a single release or a subset of releases within the same major release number) • (***RecoverIssuesSubsetReleases***) Number of issues for a specific set of releases containing RECOVER in the title, description or additional comments • (***RatioRecoverIssuesSubsetReleases*** = *RecoverIssuesSubsetReleases / TotalIssuesSubsetReleases*) For a specific set of releases, ratio of number of issues containing RECOVER in the title, description or additional comments over total number of issues. Source: Issue tracking database Artefact: Set of Issues Rationale: Recoverability issues reported in bug tracking system are highly likely to use the words recovered, recoverable, unrecoverable, ... Contact: JCD – CETIC	Manual: Advanced Search of Issue Tracking System
Advanced	Software patterns related to recoverability are present in the source code, for example, presence of a thread that wakes up periodically to auto-save data Source: Distributions List Artefact: Source Files	To build

Level	Measurement	Tool
	Rationale: If patterns related to recoverability are found in the code, it is likely that the product has some degree of recoverability. Contact: JCD – CETIC	
Advanced	Verify that test script or test procedure tests the software product for recoverability, for example, by bringing down the software application and then starting it again to verify that data could be recovered Source: Distributions List or Version Control Repository Artefact: Test Files Rationale: If some tests actually check for recoverability, it is more likely that the product has the intend to provide a recoverability feature) Contact: JCD – CETIC	Manual
Advanced	User Documentation has content dedicated to Recoverability (for example, section on recoverability mentions how to activate and customize data recoverability for the software product or how to install it to improve recoverability) Source: Distribution List Artefact: Documentation Files Rationale: If the documentation addresses recoverability, it is more likely that the product has the intend to provide a recoverability feature) Contact: JCD – CETIC	Manual

5.1.4 Availability

Availability (IEEE): The degree to which a system or component is operational and accessible when required for use.

Level	Measurement	Tool
Basic	• (***TotalIssuesAllReleases***) Total Number of issues for all releases • (***AvailIssuesAllReleases***) Number of issues for all releases containing AVAILABILITY or ACCESS in the title, description or additional comments • (***RatioAvailIssuesAllReleases*** = *AvailIssuesAllReleases* / *TotalIssuesAllReleases*) For all releases, ratio of number of issues containing AVAILABILITY or ACCESS in the title, description or additional comments over total number of all issues • (***TotalIssuesSubsetReleases***) Total number of issues for a specific set of releases (e.g., a singleton of a single release or a subset of releases within the same major release number) • (***AvailIssuesSubsetReleases***) Number of issues for a specific set of releases containing AVAILABILITY or ACCESS in the title, description or additional comments • (***RatioAvailIssuesSubsetReleases*** = *AvailIssuesSubsetReleases* / *TotalIssuesSubsetReleases*) For a specific set of releases, ratio of number of issues containing AVAILABILITY or ACCESS in the title, description or additional comments over total number of issues. Source: Issue Tracking Database Artefact: Set of Issues	Advanced Search Issue tracking database

Level	Measurement	Tool
	Rationale: Availability issues reported in bug tracking system are highly likely to use the words availability, access (note: some seach tools perform matching on word root hence available is a match however, based on some tests too many hits are returned when matching available so they must be ignored) Contact: JCD – CETIC	
Advanced	Product is build on libraries that have proven track record regarding availability Source: Distributions List Artefact: Executable Files Rationale: Software using existing libraries that have a trac record for their way of managing high availability are also likely to have a high level of availability. Contact: JCD – CETIC	To build
Advanced	Patterns showing the ability to handle and manage multiple client connections or multi tasking, for example, proper use of multi threading or processing, Potential use of a queue scheduling and management system for system with high demand, Presence of a separate subcomponent dedicated to the scheduling of processes (or threads). Source: Distributions List Artefact: Source Files Rationale: High availability should be dealt with explicitly and clearly in the code. Contact: JCD – CETIC	To build
Advanced	Test script contains stress tests to assess the availability of the software product before releases. Source: Distributions List Artefact: Test Files Rationale: If the product is put under stress, load testing it is more likely to address availability. Contact: JCD – CETIC	To build or Manual

5.2 Maturity

5.2.1 Age

Definition: The time span over which a product has been developed.

Level	Measurement	Tool
Basic	(***FirstFOSSAge***) Age of the first stable distribution release in FOSS (as compared to present time) Source: Distributions List OR Website Artefact: Single Packaged Release OR Webpages (listing releases) Rationale: Directly linked not need for rational Contact: JCD – CETIC	Manual

Level	*Measurement*	*Tool*
Basic	(***FirstSourceAge***) Age of the oldest source file of the first stable release in the Version Control Repository (as compare to present time) Source: Distributions List Artefact: Source Files Rationale: Alternate mean to compute age. Contact: JCD – CETIC	Manual
Basic	(***NonFOSSAge***) Age of the software product from its first closed source version (as compare to present time) (In case, the software product existed in closed source prior to its FOSS release) Source: Website Artefact: Webpages Rationale: Some product existed under a proprietary licence and close source prior to being released. It is an important information to take into account when determining the age of a product. Contact: JCD – CETIC	Manual

5.2.2 Continuity

Definition: The regularity and intensity with which the product or information related to the product was created or modified over the product's lifespan.

Level	*Measurement*	*Tool*
Basic	• (***TotalCommitAllRelease***) Number of Commits for all releases • (***TotalCommitSubsetRelease***) Number of Commits for a specific set of releases (e.g. all minor releases under a specific major release or even just a singleton set with a single release) • (***TotalLOCCommitAllRelease***) Number of lines of code committed for all releases. • (***TotalLOCCommitSubsetRelease***) Number of lines of code committed for a specific set of releases Source: Version Control Repository Artefact: Set of Commits and Set of Change set Rationale: The number of commit and the quantity of changes in general as well as targeted to specific releases can be an indirect approximation of the continuity of effort put in the product. Contact: JCD – CETIC	CVSAnalY / Version Control Repository (VCR)
Basic	• (***TotalMajorRelease***) Number of Major Releases • (***TotalAllReleases***) Number of stable releases (all, major and minor) • (***TotalMajorOverAge*** = *TotalMajorRelease / FOSSAge*) Number of Major Releases over age of existence in FOSS • (***TotalAllReleasesOverAge*** = *TotalAllReleases / FOSSAge*) Number of stable releases (all, major and minor) over age of existence in FOSS Source: Version Control Repository or Website Artefact: Tagged Snapshot in Version Control Repository OR Webpages listing	CVSAnalY / OR Manual: Search Website

Level	*Measurement*	*Tool*
	the Packaged Releases. Rationale: All these measurements approximate the continuity in the development effort. Contact: JCD – CETIC	
Basic	• (***TotalMajorPerYear***) Number of Major Releases per year • (***TotalAllReleasesPerYear***) Number of stable releases (all major and minor) per year (or per trimester) Source: Version Control Repository or Website Artefact: Tagged Snapshot in Version Control Repository OR Webpages listing the Packaged Releases. Rationale: The historical analysis of number of releases over a period of time is the best indicator of continuity related to effort development. Contact: JCD – CETIC	Manual + CVSAnaly
Basic	• (***TotalBook***) Number of books published about the software product • (***TotalBookPerYear***) Number of books on the software product published per year Source: Publication Database Artefact: Set of Books Rationale: The presence of books shows high degree of commitment from the community especially when studied over a time period. Contact: JCD – CETIC	Search Book on Amazon.com
Basic	(COULD BE MOVE ELSEWHERE) • (***TotalSciArt***) Number of scientific article published related to the FOSS software product (not limited to article studying the project but also research innivation implemented in the product, e.g. gcc research) • (***TotalSciArtPerYear***)Number of scientific article published related to the FOSS software product per year Source: Publication Database Artefact: Set of Articles OR Set of Bibliographies Rationale: The presence of research article indicates that innovation is likely to be implemented in the project and to continuously generate improvement effort in the product. Contact: JCD – CETIC	Tool: Search on (http://liinwww.ira.uka.de/bibliography/)
Advanced	For the first 6 metrics below, the acceptable variation must be defined. • Historical variation of commits per month for all releases • Historical variation of commits for a specific release per month • Historical variation of commits for a specific set of releases (e.g. all minor releases under a specific major release) per month • Historical variation of lines of code committed for all releases per month • Historical variation of lines of code committed for a specific release per month	CVSAnaly + historical statistical analysis of Version Control Repository

Level	Measurement	Tool
	• Historical variation of lines of code committed for a specific set of releases (e.g. all minor releases under a specific major release) per month Source: Version Control Repository Artefact: Set of Commits (per month) Rationale: Historical study of community contribution shows a continuous effort in improving the product. Contact: JCD – CETIC	

5.2.3 Activity on stable development branch

Definition: The number and size of the contributions made to a product's stable development branch over a certain period of time. High activity on a branch declared to be stable can be a sign of low product maturity.

Level	Measurement	Tool
Basic	(***TotalSubsetReleases***) Number of children releases within a selected release/version number. Source: Website Artefact: Webpages listing Set of Distributions Rationale: The number of sub releases is directly related to the number of activity within a stable branch Contact: JCD – CETIC	Manual: Search Website
Basic	(***TotalSubsetReleasesPerYear***) Number of children releases under a selected release/version number per year Source: Website Artefact: Webpages listing Set of Distributions (per year) Rationale: Release activity over time period is a good indicator of the level of stability of a release Contact: JCD – CETIC	Manual: Search Website
Basic	(***TotalCommitAllReleases***) Number of commits performed for all releases containing a given prefix in their tag (assuming that children releases share a common prefix with their parent release) Source: Version Control Repository Artefact: Tags in VCR Rationale: Analysis of activity in the version control is a direct indication of product stability. Contact: JCD – CETIC	CVSAnaly
Basic	(***TotalVulAllReleases***) Number of vulnerabilities and exposures in NVD for all releases of a software product Source: Security Database Artefact: Set of Vulnerabilities and Exposures Rationale: Identification of vulnerabilities or exposures in a stable branch requires actions hence directly participate to the level of activities in relation to the branch.	Manual: Advanced Search of NVD

Level	*Measurement*	*Tool*
	Contact: JCD – CETIC	
Advanced	• Number of issues reported for all children releases of a selected release • Number of issues whose resolution flag show an action took place vs. all issues reported for a single selected release • Number of issues whose resolution flag show an action took place vs. all issues reported for all children releases of a selected release. NOTE: A resolution flag that indicates is, for example, FIX where as WONTFIX or INVAL shows that no real action on the product took place in response to the issue report. Source: Issue Tracking Database Artefact: Set of Issues Rationale: Issue reporting is also a indirect indicator regarding activity on stable branches Contact: JCD – CETIC	Manual: Advanced Search of Issue Tracking System
Advanced	• Number of issues reported for all children releases under a selected release per year • Number of issues per year whose resolution flag show an action took place vs. all issues reported for a single selected release • Number of issues per year whose resolution flag show an action took place vs. all issues reported for all children releases under a selected release NOTE: A resolution flag that indicates is, for example, FIX where as WONTFIX or INVAL shows that no real action on the product took place in response to the issue report. Source: Issue Tracking Database Artefact: Set of Issues (per year) Rationale: More advanced studies of data provided in issue tracking system can help identify finer level of activities in a specific branch . Contact: JCD – CETIC	Manual: Advanced Search of Issue Tracking System
Advanced	Historical variation, on a monthly basis, of the number of commits performed for all releases containing a given prefix in their tag (assuming that children releases share a common prefix with their parent release). This curve should fit with the logarithmic function. Source: Version Control Repository Artefact: Tags in VCR Rationale: It is expected that a specific release has a decreasing level of activity over time. Contact: JCD – CETIC	CVSAnaly + Manual
Advanced	Number of vulnerabilities and exposures in NVD for a specific subset of releases of a software product (for example all children release under 1.*) Source: Security databases Artefact: Set of Vulnerabilities and Exposures Rationale: More advanced studies of data provided in security databases can help identify finer level of activities in a specific branch.	Manual: Advanced Search of NVD

Level	Measurement	Tool
	Contact: JCD – CETIC	
Advanced	Historical variation of number of vulnerabilities and exposures in NVD for a specific proper subset of releases of a software product. (This curve should fit with the logarithmic function) Source: Security databases Artefact: Set of Vulnerabilities and Exposures (over time) Rationale: It is expected that a specific release has a decreasing level of vulnerabilities reported over time. Contact: JCD – CETIC	Manual: Advanced Search of NVD

5.3 Security

5.3.1 Confidentiality

Definition: The degree to which a system prevents unauthorized disclosure of information; that is, provides assurance that information is not disclosed to unauthorized individuals, processes, or devices. (CNSS, 2006)

Level	Measurement	Tool
Basic	• (***TotalIssuesAllReleases***) Total Number of issues for all releases • (***ConfidIssuesAllReleases***) Number of issues for all releases containing AUTHENTICATION AUTHORIZATION or ACCESS CONTROL in the title, description or additional comments • (***RatioConfidIssuesAllReleases*** = *ConfidIssuesAllReleases* / *TotalIssuesAllReleases*) For all releases, ratio of number of issues containing AUTHENTICATION AUTHORIZATION or ACCESS CONTROL in the title, description or additional comments over total number of all issues • (***TotalIssuesSubsetReleases***) Total number of issues for a specific set of releases (e.g., a singleton of a single release or a subset of releases within the same major release number) • (***ConfidIssuesSubsetReleases***) Number of issues for a specific set of releases containing AUTHENTICATION AUTHORIZATION or ACCESS CONTROL in the title, description or additional comments • (***RatioConfidIssuesSubsetReleases*** = *ConfidIssuesSubsetReleases* / *TotalIssuesSubsetReleases*) For a specific set of releases, ratio of number of issues containing AUTHENTICATION AUTHORIZATION or ACCESS CONTROL in the title, description or additional comments over total number of issues. Source: Issue Tracking Database Artefact: Set of Issue Titles and Textual Description Rationale: Confidentiality issues reported in bug tracking system are highly likely to use these words. Contact: JCD – CETIC	Manual: Advanced Search of Issue Tracking System
Basic	• (***SeverVulAllReleases***) Number of sever vulnerabilities or exposures for all releases of a product Source: Security Databases Artefact: Set of Vulnerabilities and Exposures Rationale:	Manual : Advanced Search of NVD

Level	*Measurement*	*Tool*
	• raw data can provide overall information as to the expected level of confidentiality one may expect from the product. • Argument: Users want to make sure the specific version in use (or to be integrated) contains literally no known vulnerabilities and exposures Contact: JCD – CETIC	
Advanced	Use of renown libraries/framework for authentication, authorization and access control. Source: Distributions List Artefact: Executable Files Rationale: A know access control framework simplify the integration the appropriate confidentiality in a product Contact: JCD – CETIC	To build
Advanced	• Use of appropriate code and pattern to interact with an authentication, authorization and access control framework • Use of appropriate code and pattern to encrypt data before streaming it out of the application. (Argument: On the top of using a framework, the code must use it properly to avoid confidentiality leakage, for example, information streamed out of the application are encrypted prior, ...) Source: Distributions List Artefact: Source Files Rationale: Using existing libraries is not sufficient, they must be used correctly hence the proper use can be seen in the source code. Contact: JCD – CETIC	To build
Advanced	Test suite contains tests that attempt to gain access to the application or its data without appropriate rights) (Argument: such test scripts would show that confidentiality is tested for) Source: Distributions List Artefact: Test Files Rationale: If there are tests regarding confidentiality then the product is more likely to provide confidentiality to its users. Contact: JCD – CETIC	Manual
Advanced	• User Documentation explains how the software product handle authentication, authorization, access control, and encryption. • Installation Manual explain how to setup the product to guarantee a high level of confidentiality Source: Distributions List Artefact: Documentation Files Rationale: Documentation that contains information related to confidentiality shows that confidentiality is addressed by the software product. Contact: JCD – CETIC	Manual

Level	Measurement	Tool
Advanced	• Ratio of sever vulnerabilities with patch vs all for a specific release of the product Source: Security Databases Artefact: Set of Vulnerabilities and Exposures Rationale: • raw data can provide overall information as to the expected level of confidentiality one may expect from the product. • Argument: Users want to make sure the specific version in use (or to be integrated) contains literally no known vulnerabilities and exposures Contact: JCD – CETIC	Manual : Advanced Search of NVD
Advanced	Verify that patches are provided quickly after the initial report (for example, no later than a week after initial report) Source: Security Databases Artefact: Set of Vulnerabilities and Exposures Rationale: Patch created promptly maintain a high confidence from users regarding low risk of potential intrusion and confidentiality leaks) Contact: JCD – CETIC	To be build or Manual

5.3.2 Integrity

Integrity (ISO): The degree to which a system or component is able to protect the accuracy and completeness of information and processing methods. This includes preventing unauthorised modification or destruction of information (CNSS, 2006).

Level	Measurement	Tool
Basic	• (***TotalIssuesAllReleases***) Total Number of issues for all releases • (***IntegrIssuesAllReleases***) Number of issues for all releases containing CORRUPTED or CHECKSUM in the title, description or additional comments • (***RatioIntegrIssuesAllReleases*** = *ConfidIssuesAllReleases* / *TotalIssuesAllReleases*) For all releases, ratio of number of issues containing CORRUPTED or CHECKSUM in the title, description or additional comments over total number of all issues • (***TotalIssuesSubsetReleases***) Total number of issues for a specific set of releases (e.g., a singleton of a single release or a subset of releases within the same major release number) • (***IntegrIssuesSubsetReleases***) Number of issues for a specific set of releases containing CORRUPTED or CHECKSUM in the title, description or additional comments • (***RatioIntegrIssuesSubsetReleases*** = *ConfidIssuesSubsetReleases* / *TotalIssuesSubsetReleases*) For a specific set of releases, ratio of number of issues containing CORRUPTED or CHECKSUM in the title, description or additional comments over total number of issues. Source: Issue Tracking Database Artefact: Set of Issues title or description text Rationale: Confidentiality issues reported in bug tracking system are highly likely to use these words. Contact: JCD – CETIC	Manual: Advanced Search of Issue Tracking System

Level	*Measurement*	*Tool*
Advanced	Use of renown libraries/framework for encryption and digital signature Source: Distributions List Artefact: Executable Files Rationale: A know access control framework simplify the integration the appropriate code to implement integrity in a product. Contact: JCD – CETIC	To build
Advanced	Use of appropriate code and pattern to saving and checking data integrity before streaming it in and out of the application. Source: Distributions List Artefact: Source Files Rationale: On the top of using a framework, the code must use it properly to guarantee integrity Contact: JCD – CETIC	To build
Advanced	Test suite contains tests that attempt to corrupt data and to process corrupted data) Source: Distributions List Artefact: Test Files Rationale: Such test scripts would show that integrity is tested for hence has more chances to be address adequately Contact: JCD – CETIC	Manual
Advanced	• Installation Manual explain how to setup the product to guarantee a high level of integrity • User Manuel explain to the user how to sign data (Argument: helping the user sign data show a concern for integrity) Source: Distributions List Artefact: Documentation Files Rationale: Documentation that contains information related to integrity shows that integrity is addressed by the software product. For example, how to setup the product to use digital signature Contact: JCD – CETIC	Manual
Advanced	(not integrity of the product but integrity of the product download) Product Distribution download packages provide their MD5 or other checksum. Source: Website Artefact: Webpages listing distribution releases Rationale: Providing a checksum show that the project consider integrity to be a priority Contact: JCD – CETIC	Manual

5.3.3 Security Standard compliance

Compliance to security standards: The degree to which a product complies with published security standards that are relevant to its functionality.

We were unable to identify simple basic metrics to be used in tasks 1.4-1.6. That is, we will not be able to measure this quality attribute yet.

Level	*Measurement*	*Tool*
Basic	No metrics identified	
Advanced	Are the security framework used to guarantee confidentiality and integrity renown for following X.509 standard, Kerberos, ... Source: Distributions List Artefact: Executable Files Rationale: This renown standards have been implemented in libraries. If these libraries are used it shows a desire to refer to existing security standard rather than have an ad-hoc solution. Contact: JCD – CETIC	To build
Advanced	(For Server Application) Are the security specifications (confidentiality, integrity) found in product configuration files written in a renown language for specifying security policies such as XACML Source: Distributions List Artefact: Source Files Rationale: The use of a standard security policy configuration indicate a certain level of standard compliance Contact: JCD – CETIC	To build or Manual
Advanced	Are documents mentioning that the software product has been used in systems that are now certified Common Criteria level X (where is must be certified) Source: Distributions List Artefact: Documentation Files Rationale: If the documentation refers to existing security standard, it show commitment to include such standard in the product. Contact: JCD – CETIC	Manual

6. Robustness: Community Quality Model

This section details the metrics identified for measuring the community aspects of the quality model for evolvability. In parts, this is addressed by the process and document assessment frameworks in Sections 7 and 8.For details on the metrics listed in this section, please refer to the appendix.

6.1 Maturity of Security Processes

6.1.1 Compliance

Level	*Measurement*	*Tool*
Basic	No Basis metrics identified	
Advanced	The degree to which best practices in security processes of F/OSS projects are implemented. This metric uses the **process assessment framework** (see Section 7) Artefact: User and developer documentation Files, project website, process guidelines, ... Rationale: *Maturity of the community/project is usually perceived in terms of how well the project implements best practice processes.*	Doceval and others
Advanced	Number of messages in the mailing list about security bug issues. Source: Mailing List Archive Artefact: Text content and attachments. Thread of answers. Contact: DI/ACS (URJC)	MLStats
Advanced	Evolution in the number of messages in the mailing list about security bug issues. Source: Mailing List Archive Artefact: Historical Analyses specific to Mailing List Archive Contact: DI/ACS (URJC)	MLStats
Advanced	Developers Forum activity. Number of posts about security issues. Source: Discussion Forum Artefact: Text content and attachments. Thread of answers Contact: DI/ACS (URJC)	To build or Manual
Advanced	Evolution in the number of posts about security issues. Source: Discussion Forum Artefact: Historical Analyses specific to Discussion Forum. Contact: DI/ACS (URJC)	To build or Manual
Advanced	How many messages are in the IRC naming a bug for security issues. Source: IRC logs Artefact: Static text content. Contact: DI/ACS (URJC)	Basic tools like “grep” and others

6.1.2 Reaction Time

Level	*Measurement*	*Tool*
Basic	No Basic metrics identified	
Advanced	Time between a security bug is accepted to be solved and the final commit with the bug solved to CVS repository. Sources: Version Control Repository, Issue Tracking Database Rationale: We need to cross results from Bug Tracking System for security issues and CVSAnaly (At URJC we do not have yet a tool for measuring BTS). And not in all cases we will be able to cross these results. It depends on the use of the BTS and how much information it has. Contact: DI/ACS (URJC)	To build or Manual
Advanced	We could cross results from Bug Tracking Systems (BTS) security bugs about security issues and try to guess how long it takes appearing a mailing list thread about it using the message date and bug date. Source: Mailing List Archive, Issue Tracking Database Contact: DI/ACS (URJC)	To build or Manual
Advanced	We could cross results from Bug Tracking Systems (BTS) security bugs and try to guess how long it takes appearing a forum thread about it using the post date and bug date. Source: Discussion Forum, Issue Tracking Database Contact: DI/ACS (URJC)	To build or Manual
Advanced	We could cross results from Bug Tracking Systems (BTS) security bugs and try to guess how long it takes appearing a message in the IRC about it using the irc message date and bug date. Source: IRC logs, Issue Tracking Database Contact: DI/ACS (URJC)	To build or Manual

6.1.3 Inclusion of Preventive/Reactive Actions

Level	*Measurement*	*Tool*
Advanced	The degree to which best practices in security processes of F/OSS projects are implemented. This metric uses the **process assessment framework** (see Section 7) Artefact: User and developer documentation Files, project website, process guidelines, ... Rationale: *Maturity of the community/project is usually perceived in terms of how well the project implements best practice processes.*	Manual

Level	*Measurement*	*Tool*
Advanced	Number / Percentage of commits related to security problems. With CVSAnalY we know commits and the files used. Source: Version Control Repository Rationale: Perhaps, an idea could be: we need to identify critical files for security issues and then we can measure commits over these files, but we need at the beginning to identify these files. Contact: DI/ACS (URJC)	CVSAnaly and Manual

6.2 Maturity of Reliability Processes

6.2.1 Compliance

Level	*Measurement*	*Tool*
Basic	(***NumOfBugMessages***) Number of messages in the mailing list about bugs. A basic metric can be obtained crossing number of bugs with the body of messages. (The problem here is that it is necessary a list of bugs (its names)). Source: Mailing List Archive Artefact: Text content and attachments. Thread of answers. Rationale: If people and developers use mailing list to solve this problems, everybody can observe the evolution of this bug and its solution. Contact: DI/ACS (URJC)	MLStats
Basic	(***EvolutionHistoryBugMessages***) Evolution in the number of messages in the mailing list about bugs. A basic metric can be obtain crossing number of bugs with the body of messages and then, observe the evolution in Mailing List. (The problem here is that it is necessary a list of bugs (its names)). Source: Mailing List Archive Artefact: Historical Analyses specific to Mailing List Archive Rationale: Its name is auto-explicative. Contact: DI/ACS (URJC)	MLStats
Advanced	Developers Forum activity. Number of posts about bugs. Source: Discussion Forum Artefact: Text content and attachments. Thread of answers Rationale: Same as above Contact: DI/ACS (URJC)	To build or Manual
Advanced	Evolution in the number of posts about bugs. Source: Discussion Forum Artefact: Historical Analyses specific to Discussion Forum. Rationale: Same as above Contact: DI/ACS (URJC)	To build or Manual
Advanced	How many messages are in the IRC naming a bug. Source: IRC logs Artefact: Static text content. Rationale: Same as above Contact: DI/ACS (URJC)	Basic tools like “grep” and others

6.2.2 Reaction Time

Level	*Measurement*	*Tool*
Advanced	Time between a bug is accepted to be solved and the final commit with the bug solved to CVS repository. Sources: Version Control Repository, Issue Tracking Database Rationale: We need to cross results from Bug Tracking System and CVSAnaly (At URJC we do not have yet a tool for measuring BTS). And not in all cases we will be able to cross these results. It depends on the use of the BTS and how much information it has. Contact: DI/ACS (URJC)	To build or Manual
Advanced	We could cross results from Bug Tracking Systems (BTS) bugs and try to guess how long it takes appearing a mailing list thread about it using the message date and bug date. Source: Mailing List Archive, Issue Tracking Database Contact: DI/ACS (URJC)	To build or Manual
Advanced	We could cross results from Bug Tracking Systems (BTS) bugs and try to guess how long it takes appearing a forum thread about it using the post date and bug date. Source: Discussion Forum, Issue Tracking Database Contact: DI/ACS (URJC)	To build or Manual
Advanced	We could cross results from Bug Tracking Systems (BTS) bugs and try to guess how long it takes appearing a message in the IRC about it using the irc message date and bug date. Source: IRC logs, Issue Tracking Database Contact: DI/ACS (URJC)	To build or Manual

6.2.3 Inclusion of Preventive/Reactive Actions

Level	*Measurement*	*Tool*
Advanced	The degree to which best practices in reliability processes of F/OSS projects are implemented. This metric uses the **process assessment framework** (see Section 7) Artefact: User and developer documentation Files, project website, process guidelines, ... Rationale: *Maturity of the community/project is usually perceived in terms of how well the project implements best practice processes.*	Manual
Advanced	Number / Percentage of commits related to reliability problems. With CVSAnalY we know commits and the files used. Source: Version Control Repository Contact: DI/ACS (URJC)	CVSAnaly and Manual

7. Process Assessment Aspects

The QualOSS quality model was explicitly designed to cover not only product quality but *community quality*, i.e., the general ability of a F/OSS community to deliver quality products over an extended period of time. Although, per definition, community quality is reflected in the resulting product quality, high product quality alone cannot really guarantee that a community is generally capable, since it is well known that the efforts of capable and motivated individuals may lead to high product quality for particular instances of a product. So, for example, the fact that a particular version of a product is especially good does not guarantee that future versions will be as good, because the degree of commitment of some developers may wane over time.

Experience with standard software development shows that the capability of an organization to *consistently and predictably* deliver high quality software is connected with the maturity of their software processes, i.e., the actual practices used to develop or maintain a software product. During the last 20 years, a number of models and standard have been created that allow to assess the maturity of an organization's software processes and provide guidelines to improve such maturity over time.

Although these models have been development with traditional, hierarchically controlled software organizations in mind, we believe that many of the concepts they use can be applied to F/OSS as well. The main rationale is that many F/OSS projects actually deliver consistently high quality products over time, and that this fact must be linked to good practices that are enforced inside the community. For this reason, we decided to study the possibility of applying existing process maturity models to F/OSS projects. Due to the obvious differences between traditional software organizations and F/OSS communities, it must be expected that a number of the practices required by established process maturity models simply do not apply to F/OSS. On the other hand, many good practices certainly apply and can be found in one form or another in actual F/OSS projects.

Our final aim is to identify this latter set of practices in order to produce a process maturity model for F/OSS that makes it possible to comprehensively assess the process maturity of a given F/OSS community. Although this maturity model is not yet complete, we present here the first set of results related to it. These results are discussed in more detail in the following subsections.

7.1 Status and Next Steps

Currently, we have defined a reference framework for the process assessment that covers all CMMi process areas relevant to QualOSS, as well as identified assessment levels. However, we have currently not defined an interpretation model for the assessment framework; that is, a procedure to assign assessment levels to a specific F/OSS project.

Therefore, the next steps in tasks 1.4 and 1.5 will include to calibrate the process assessment; that is, to define which practices are required on which level to be able to assign a metric value to each process area.

7.2 A Model for Process Maturity: CMMI

The Capability Maturity Model for Software, published in the early 1990's by the Software Engineering Institute (SEI) at Carnegie Melon University was one of the first, and has been one of the most influential process maturity models until now. Over the years, the CMM evolved into the so-called CMMI-DEV (Capability Maturity Mode Integration for Development), which can be considered the current version of the model. From its preface:

> *CMMI® (Capability Maturity Model® Integration) is a process improvement maturity model for the development of products and services. It consists of best practices that address development and maintenance activities that cover the product lifecycle from conception through delivery and maintenance.*
>
> *[...]*
>
> *The purpose of CMMI for Development is to help organizations improve their development and maintenance processes for both products and services. CMMI for Development is a collection of best practices that is generated from the CMMI Framework. 1 The CMMI Framework supports the CMMI*

Product Suite by allowing multiple models, training courses, and appraisal methods to be generated that support specific areas of interest.

We chose CMMI-DEV as a basis for our F/OSS process maturity model. Reasons for this choice include the fact that CMMI-DEV is well established and widely used, that it has excellent and freely available documentation, and that it has a flexible, modular structure that makes it possible to cherry pick interesting elements (that is, we can use the elements that apply to F/OSS and ignore those that do not apply.) Although we are using CMMI-DEV as our main process maturity model, the possibility of including elements from other models such as SPICE [] remains open.

CMMI-DEV is organized in 22 so-called *process areas* each one covering a different area of the general software process. Process areas contain goals, which, in turn, are divided into *practices*. Both goals and practices are classified in two categories: *specific* and *generic*. From the 22 process areas, we have identified those that may apply to F/OSS projects. Afterwards, we have gone through the goals and practices looking for evidence of their use in actual F/OSS projects. The rationale of doing this is that if there exists at least one F/OSS project/community that actually uses a particular practice, it is sensible to ask for the presence of this practice in other F/OSS projects. Section 7.3 presents the results of this analysis for five different process areas.

Section 7.4 presents an example of how the results of an assessment may look like, namely, the Python project was analysed with respect to requirements management.

7.3 Evidence of Process Maturity in F/OSS Projects

As explained above, finding evidence of the use of a particular practice in a F/OSS project tells us that it is reasonable to look for that same practice in other F/OSS projects. For this reason, we chose a number of CMMI-DEV process areas and looked for such evidence in well known F/OSS projects. This serves as a form of validation of the maturity model.

The following sections present the relevant CMMI-DEV process areas, goals (SG or GG for *specific* or *generic goal*) and practices (SP or GP for *specific* or *generic* practice). The text in boxes is taken verbatim from CMMI-DEV, and briefly explains each goal or practice (the reader is invited to refer to CMMI-DEV for more detailed explanations and examples.) Finally, the text following the boxes presents our collected evidence of the use of practices in actual F/OSS projects.

7.3.1 Configuration Management

SG 1 Establish Baselines

Baselines of identified work products are established.

SP 1.1 Identify Configuration Items

Identify the configuration items, components, and related work products that will be placed under configuration management.

Most OSS projects are opened for the whole community, so there isn't much of a selection which the community can have access. Most of the project's files are managed under configuration management.

SP 1.2 Establish a Configuration Management System

Establish and maintain a configuration management and change management system for controlling work products.

OSS projects use a version control system to maintain a configuration management and change management. The tool CVS (Concurrent Version System) is used for version control in most projects as it satisfies all basic requirements. All versions are kept in a repository and mechanisms are used to minimize the space consumption. It can handle branches, information about versions can be given in the change log, symbolic tags can be assigned to versions. Usually write access to the CVS repository is generously granted such that several

hundred developers can add new versions to it. Regular patches can also be added to the repository by the moderators (a person who owns one or more modules). Patches have to be sent to the moderator, who alone decides whether to apply the change or to reject it. Moderators are used in Linux. In the OSS projects, versions are almost never used to revert to an older version. Instead, versions are used as a history trail, describing how a file has developed by reading the log comments and by comparing versions using the comparison functionality (which can show the differences between two versions of a file). Most projects are targeted towards several platforms. OSS projects seem to handle variants by either separating code into different files or directories, or by using conditional compilation, so all variants can exist in the same branch. Changes apply either to the whole project, or platform-specific code.

SP 1.3 Create or Release Baselines

Create or release baselines for internal use and for delivery to the customer.

Most OSS projects do not release software in the traditional sense, wrapping up code, documentation, help files, install scripts and more, turning it into a software packet. They rely on users themselves or on commercial companies, like Red Hat and others. Few OSS projects use fixed release dates, and labeling and timing of releases is mostly arbitrary. They do, however, carry out what can be called internal releases, which are points in time where they freeze the source code and for which there is a process. When an internal release is getting nearer, the development branch enters a freeze stage: initially a soft freeze stage means that new features that break compatibility are discouraged but not forbidden, then a hard freeze stage, in which any contribution that will change an interface is forbidden. Only bug fixes are allowed. Finally, when the internal release is made, the code is copied to a new branch, called stable, where maintenance can be performed if needed. Further development continues on the development branch, heading for the next internal release. An internal releases of sufficient quality can be used when creating a traditional release. In the Mozilla project they use a time-based release schedule. This means that development proceeds until a certain date, when a release is labeled (called a milestone in Mozilla terms). The milestone is then used to see what has been achieved. Features and achievements are not planned into milestones; they only work as a feedback tool.

SG 2 Track and Control Changes

Changes to the work products under configuration management are tracked and controlled.

SP 2.1 Track Change Requests

Track change requests for the configuration items.

In OSS the review of change proposals is not explicit, if there at all. Anyone can propose a change and often changes are not even proposed before a change implementation is submitted directly. Change proposals might be prioritized implicitly or explicitly, but an OSS project cannot assign tasks to developers – everyone works on what he chooses.

Two slightly different processes exist depending on whether contributions have to be sent to a moderator or if the developer can apply his changes directly to the repository through his write access. In both cases, however, the overall process is the same: an idea for a change is conceived, implemented and tested, submitted as a patch or applied directly on the repository, and then the implementation (and sometimes the change idea itself) is evaluated through testing, review and discussion.

SP 2.2 Control Configuration Items

Control changes to the configuration items.

The final evaluation may result in the patch being rejected by a moderator or a change to the repository being reverted by a coordinator. Usually write access to the repository is given only to trusted developers, so reversion of a change to the repository is rare

Most change management problems seem to be caught during review. Contributions are often tested via code reviewing and special run time tests, formal testing is not always used. When a change has been made, developers sometimes just use the new code. Developers who have submitted many good patches are more trusted, and their contributions make their way into the repository more quickly. Accepted contributions show up immediately in the repository.

Even though wish lists and lists of bugs are kept, bugs and change proposals seem to be fixed somewhat arbitrarily. Changes are kept track of using detailed lists, so that willing users can test new features. Mail and newsgroups are used to communicate wish lists, bugs, and changes and to discuss the general development of the project.

7.3.2 Requirements Management

Specific Practices by Goal

SG 1 Manage Requirements

Requirements are managed and inconsistencies with project plans and work products are identified.

SP 1.1 Obtain an Understanding of Requirements

Develop an understanding with the requirements providers on the meaning of the requirements.

It appears that open software requirements are articulated in a number of ways that are ultimately expressed, represented, or depicted on the Web. On closer examination, requirements for open software can appear or be implied within an email message or within a discussion thread that is captured and/or posted on a project's Web site message board for open review, elaboration, refutation, or refinement.

These requirements are simply asserted without reference to other documents, sources, standards, or JAD focus groups--they are requirements because some developers wanted these capabilities.

SP 1.2 Obtain Commitment to Requirements

Obtain commitment to the requirements from the project participants.

Developing open software requirements is a community building process that must be institutionalized both within a community and its software informalisms to flourish. In this regard, the development of requirements for open software is usually not a traditional requirements engineering process, at least, not yet. It is instead socio-technical process that entails the development of constructive social relationships, informally negotiated social agreements, and a commitment to participate through sustained contribution of software discourse and shared representations. So the Author and/or the Core Group must motivate and explain why the volunteers should stick to the requirements.

SP 1.3 Manage Requirements Changes

Manage changes to the requirements as they evolve during the project.

Asserted system capabilities are post-hoc requirements characterizing a functional capability that has already been implemented. Concerned developers justify their requirements through their provision of the required coding effort to make these capabilities operational. Senior members or core developers in the community then vote or agree through discussion to include the asserted capability into the system's distribution. The historical record may be there, within the email or message board discussion archive, to document who required what, where, when, why, and how. However, once asserted, there is generally no further effort apparent to document, formalize, or substantiate such a capability as a system requirement.

Asserted capabilities then become invisible or transparent, taken-for-granted requirements that can be labeled or treated as obvious (i.e., a shared awareness) to those familiar with the system's development.

SP 1.4 Maintain Bidirectional Traceability of Requirements

Maintain bidirectional traceability among the requirements and work products.

The traceability of the requirements is done using the archives from the mailing list or bulletin boards. It isn't common to find evidence or data to indicate the occurrence or documentation of a requirements elicitation effort arising in an open software development project. However, finding such evidence would not invalidate the other observations; instead, it would point to a need to broaden the scope of how software requirements are captured or recorded.

SP 1.5 Identify Inconsistencies Between Project Work and Requirements

Identify inconsistencies between the project plans and work products and the requirements.

Software requirements are validated with respect to the software's implementation. But the open software requirements are generally not recorded in a formal SRS document, nor are these requirements typically cast in a mathematical logic, algebraic, or state transition-based notational scheme. So the requirements for open software are co-mingled with design, implementation, and testing descriptions and software artifacts, as well as with user manuals and usage artifacts (e.g., input data, program invocation scripts). The inconsistencies found are posted in the mailing list or message board so it can be reviewed and fixed.

Generic Practices by Goal

GG 2 Institutionalize a Managed Process

The process is institutionalized as a managed process.

GP 2.1 Establish an Organizational Policy

Establish and maintain an organizational policy for planning and performing the requirements management process.

Open software requirements can emerge from the experiences of community participants through their email and message board discussion forums. These communication messages in turn give rise to the development of narrative descriptions that more succinctly specify and condense into a web of discourse about the functional and non-functional requirements of an open software system. This discourse is rendered in descriptions that can be found in email and discussion forum archives, on Web pages that populate community Web sites, and in other informal software descriptions that are posted, hyperlinked, or passively referenced through the assumed common knowledge that community participants expect their cohorts to possess.

GP 2.2 Plan the Process

Establish and maintain the plan for performing the requirements management process.

This plan for performing the requirements management process can be part of (or referenced by) the project plan as described in the Project Planning process area.

GP 2.3 Provide Resources

Provide adequate resources for performing the requirements management process, developing the work products, and providing the services of the process.

Tools to support the requirements management process are available in web portals like SourceForge.org. The most used tool is The Open Source Requirements Management Tool (OSRMT) and Shore.

GP 2.4 Assign Responsibility

Assign responsibility and authority for performing the process, developing the work products, and providing the services of the requirements management process.

The tool OSRMT is commonly used to provide services of the requirements management process.

GP 2.5 Train People

Train the people performing or supporting the requirements management process as needed.

The archives from the mailing lists and bulletin boards are used as manuals to trains the new incomers.

GP 2.6 Manage Configurations

Place designated work products of the requirements management process under appropriate levels of control.

Not found in the OSS Projects

GP 2.7 Identify and Involve Relevant Stakeholders

Identify and involve the relevant stakeholders of the requirements management process as planned.

OSS deals with Human Resources with a Joining Policy, some explicitly make sure to admit only contributors from whom high quality submissions can be expected while other projects are more liberal. The Author and Core Team are always involved motivating and selecting the ones that might contribute for the project.

GP 2.8 Monitor and Control the Process

Monitor and control the requirements management process against the plan for performing the process and take appropriate corrective action.

This practice is usually made with the support of OSRMT

GP 2.9 Objectively Evaluate Adherence

Objectively evaluate adherence of the requirements management process against its process description, standards, and procedures, and address noncompliance.

Not found in the OSS Projects

GP 2.10 Review Status with Higher Level Management

Review the activities, status, and results of the requirements management process with higher level management and resolve issues.

Not found in the OSS Projects

7.3.3 Project Planning

The term project means different things in CMMI and F/OSS. In the case of CMMI, a project, in its simplest form, is an endeavour with a specific start and end date to accomplish specific objectives. However, in the case

of F/OSS, the term project is much loser as we commonly refer to project to mean an existing endeavour on a software product without knowing when this endeavour ends and in fact, hoping that is will live as long as possible. On the other hand, many F/OSS project use the concept of releases, which resemble much more to project as defined by CMMI. In turn, in this interpretation of the CMMI project planning process is equated to F/OSS release planning process.

SG 1 Establish Estimates

Estimates of project planning parameters are established and maintained.

Although most method for estimating the release planning of F/OSS project are not explicitly shared, many successful F/OSS projects display their plans. In order to come up with such plans, they must clearly follow a logic historically, it is possible to check whether previous plans held correct. One usually finds that serious, successful FOSS projects satisfy their plan estimation. However there is little documented evidence to support that F/OSS projects "establish estimates" for their releases.

SP 1.1 Estimate the Scope of the Project

Establish a top-level work breakdown structure (WBS) to estimate the scope of the project.

Yes/Satisfied. There are distinct cases: 1. initial release of F/OSS project, 2. a major release between two version and 3. a minor release (within the same major release). Release Management includes a phase for selecting what request will be implemented in the given release cycle. (AdaCore has such a procedure in place, in larger F/OSS project, the scope of a release is also specified in plans. (see the Eclipse projects but also ASF, SFS)

SP 1.2 Establish Estimates of Work Product and Task Attributes

Establish and maintain estimates of the attributes of the work products and tasks.

Partial. In some F/OSS project, the plan specifies the work products or components to be implemented along with milestones and dates

SP 1.3 Define Project Life cycle

Define the project lifecycle phases on which to scope the planning effort.

Yes/Satisfied. Some project specifically state that they use an Agile Development Methodology while in other cases it is implied by the fact that they release early and release often.

SP 1.4 Determine Estimates of Effort and Cost

Estimate the project effort and cost for the work products and tasks based on estimation rationale.

Likely. F/OSS Projects that perform SP1.1 likely perform SP1.4 but within the private context of companies working on the an F/OSS project (hence hard to measure since data will not be render public by companies)

SG 2 Develop a Project Plan

A project plan is established and maintained as the basis for managing the project.

SP 2.1 Establish the Budget and Schedule

Establish and maintain the project's budget and schedule.

Likely. F/OSS Projects that perform SP1.1 likely perform SP1.4 but within the private context of companies working on the an F/OSS project (hence hard to measure since data will not be render public by companies)

SP 2.2 Identify Project Risks

Identify and analyze project risks.

No.

SP 2.3 Plan for Data Management

Plan for the management of project data.

Yes. Data are often managed in a forge (SourceForge, Apache, GNU, ...)

SP 2.4 Plan for Project Resources

Plan for necessary resources to perform the project.

No.

SP 2.5 Plan for Needed Knowledge and Skills

Plan for knowledge and skills needed to perform the project.

Often satisfied. Successful F/OSS project attract talented developers who like the challenge to fix bugs or develop new functionalities

SP 2.6 Plan Stakeholder Involvement

Plan the involvement of identified stakeholders.

Yes. Presence of alpha release, release candidate then stable release (where stakeholders are involved)

SP 2.7 Establish the Project Plan

Establish and maintain the overall project plan content.

Yes. Milestones are established and published for the next release or new few releases

SG 3 Obtain Commitment to the Plan

Commitments to the project plan are established and maintained.

SP 3.1 Review Plans That Affect the Project

Review all plans that affect the project to understand project commitments.

Likely. Milestones for releases must probably be approved by vote of important community members before publication

SP 3.2 Reconcile Work and Resource Levels

Reconcile the project plan to reflect available and estimated resources.

No.

SP 3.3 Obtain Plan Commitment

Obtain commitment from relevant stakeholders responsible for performing and supporting plan execution.

No.

7.3.4 Validation

In many cases, a project starts from a developer's need hence the solution he develops matches its need. Later on successful projects gather communities. Then, validation is performed by early adopters and also during the scope of release management

SG 1 Prepare for Validation

Preparation for validation is conducted.

SP 1.1 Select Products for Validation

Select products and product components to be validated and the validation methods that will be used for each.

Partial Software validation takes place but what about validation of other work products (different doc.)

SP 1.2 Establish the Validation Environment

Establish and maintain the environment needed to support validation.

Yes. Specifies the distribution or environment on which product works or has been tested

SP 1.3 Establish Validation Procedures and Criteria

Establish and maintain procedures and criteria for validation.

Likely Partial. Test script for automatic integration testing (however nothing for non-software products including maintenance, support and training). Larger projects like Apache may create a sub-project in charge of testing and validation. In some cases, standard test suite may be used (for example, J2EE test suite, WebSPEC99, ...)

SG 2 Validate Product or Product Components

The product or product components are validated to ensure that they are suitable for use in their intended operating environment.

SP 2.1 Perform Validation

Perform validation on the selected products and product components.

Little. Overall test results (pass/failed) + Test Logs saved

SP 2.2 Analyze Validation Results

Analyze the results of the validation activities.

No.

7.3.5 Technical Solution

SG 1 Select Product Component Solutions

Product or product component solutions are selected from alternative solutions.

This subgoal does not apply to all release management cycles. In many cases, all product and component selection have be done initially. However, it is possible to identify certain release cycles (between certain major releases for example) where alternative must be developed. At the foundation

or F/OSS project level, it is possible to identify discussion regarding technical issues or find justification for the selection of certain exiting F/OSS products/libraries to include in a new F/OSS project.

SP 1.1 Develop Alternative Solutions and Selection Criteria

Develop alternative solutions and selection criteria.

Satisfied. At the foundation level, decision to integrate existing project or to reimplement are found and such decisions are quite strategic, for example, Geronimo was reimplemented to have a Java AS under the Apache license + for IBM to be involved in a F/OSS AS project without open sourcing WebSphere. At the level of a single project, the decision to reuse can also be present, for example, Jakarta Commons for sharing reusable libraries across Jakarta Sub-projects.

SP 1.2 Select Product Component Solutions

Select the product component solutions that best satisfy the criteria established.

Satisfied. At the initiation of a F/OSS project, the project initiators select technologies and a language to implement their solution.

SG 2 Develop the Design

Product or product component designs are developed.

The outcome is rarely a single design document. However, during thorough refactoring, re-engineering, discussion threads on specialize mailing list can usually be identified

SP 2.1 Design the Product or Product Component

Develop a design for the product or product component.

Partial. Complete reimplementation of a F/OSS product sometimes take place due to discussion among important community members

SP 2.2 Establish a Technical Data Package

Establish and maintain a technical data package.

No.

SP 2.3 Design Interfaces Using Criteria

Design product component interfaces using established criteria.

Yes. Criteria are based on feedback from users

SP 2.4 Perform Make, Buy, or Reuse Analyses

Evaluate whether the product components should be developed, purchased, or reused based on established criteria.

F/OSS project often use other F/OSS product as part of their implementation. Or they make sure data exchange with other F/OSS project can be achieved, for example, by implementing input/output functionality based on open, published standards

SG 3 Implement the Product Design

Product components, and associated support documentation, are implemented from their designs.

SP 3.1 Implement the Design

Implement the designs of the product components.

Yes. All F/OSS project must provide their code. Per se, code does not necessarily implement an explicit design since most F/OSS do not create design

SP 3.2 Develop Product Support Documentation

Develop and maintain the end-use documentation.

Yes. All popular F/OSS product have growing set of documentation styles along with tutorials

7.4 Sample Analysis

This section presents a detailed assessment of the project developing the well-known F/OSS Python programming language, with respect to process area “Requirements Management”.

Specific Practices by Goal

SG 1 Manage Requirements

Requirements are managed and inconsistencies with project plans and work products are identified.

SP 1.1 Obtain an Understanding of Requirements

Develop an understanding with the requirements providers on the meaning of the requirements.

The Python Programming Language Project has the PEP system to understand and manage their requirements. PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment. The PEP should provide a concise technical specification of the feature and a rationale for the feature.

PEPs are intended to be the primary mechanisms for proposing new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

The PEPs are well structured and meet the following criteria:

- Clearly and properly stated
- Complete
- Consistent with each other
- Uniquely identified
- Traceable

SP 1.2 Obtain Commitment to Requirements

Obtain commitment to the requirements from the project participants.

All contributions to the PEP need one or more maintainers. This can be an individual, but it is frequently a group of people such as the XML-SIG. Groups may subdivide maintenance tasks among themselves. Head maintainers are convenient people the integrators can address if

they want to resolve specific issues, such as the ones detailed later in this document. They are responsible for the contributors to stick to the requirements and negotiate commitments.

SP 1.3 Manage Requirements Changes

Manage changes to the requirements as they evolve during the project.

To submit a PEP update depends on several factors, such as the maturity of the PEP, the preferences of the PEP author, and the nature of the comments. For the early draft stages of the PEP, it's probably best to send the comments and changes directly to the PEP author. For more mature, or finished PEPs it is best to submit corrections to the SourceForge bug manager or better yet, the SourceForge patch manager so that the changes don't get lost. If the PEP author is a SourceForge developer, the bug/patch is assigned to him, otherwise assigned to the PEP editor.

PEP authors who are also Python/SourceForge committers can update the PEPs themselves by using “svn commit” to commit their changes.

SP 1.4 Maintain Bidirectional Traceability of Requirements

Maintain bidirectional traceability among the requirements and work products.

The requirements are well organized in PEP numbers, a little description, the owner and the status (accepted, final, etc…). Every change or bug fix release has to be linked with the related PEP number.

SP 1.5 Identify Inconsistencies Between Project Work and Requirements

Identify inconsistencies between the project plans and work products and the requirements.

PEPs consist of two parts, a design document and a reference implementation. The PEP should be reviewed and accepted before a reference implementation is begun, unless a reference implementation will aid people in studying the PEP.

PEP authors are responsible for collecting community feedback on a PEP before submitting it for review. A PEP that has not been discussed on python-list@python.org and/or python-dev@python.org will not be accepted.

Once the authors have completed a PEP, they must inform the PEP editor that it is ready for review. PEPs are reviewed by the BDFL and his chosen consultants, who may accept or reject a PEP or send it back to the author(s) for revision. For a PEP that is pre-determined to be acceptable the BDFL may also initiate a PEP review, first notifying the PEP author(s) and giving them a chance to make revisions.

For a PEP to be accepted it must meet certain minimum criteria. It must be a clear and complete description of the proposed enhancement. The enhancement must represent a net improvement. The proposed implementation, if applicable, must be solid and must not complicate the interpreter unduly. Finally, a proposed enhancement must be "pythonic" in order to be accepted by the BDFL. (However, "pythonic" is an imprecise term; it may be defined as whatever is acceptable to the BDFL. This logic is intentionally circular.).

Generic Practices by Goal

GG 2 Institutionalize a Managed Process

The process is institutionalized as a managed process.

GP 2.1 Establish an Organizational Policy

Establish and maintain an organizational policy for planning and performing the requirements management process.

The BDFL (Benevolent Dictator for Life) is responsible for the whole PEP process and requirement management.

GP 2.2 Plan the Process

Establish and maintain the plan for performing the requirements management process.

The Python project has a well-structured and detailed on-line documentation to explain how the PEP process works and how it should be done. The PEP process is also opened for new improvements by the community, but it has to follow the structured PEP Work Flow and be accepted by the BDFL.

GP 2.3 Provide Resources

Provide adequate resources for performing the requirements management process, developing the work products, and providing the services of the process.

PEPs are kept in text-based files in Python Project's website, and because the PEPs are maintained as text files in a versioned repository, their revision history is the historical record of the feature proposal.

Python uses Sourceforge.org tools to facilitate the requirement management: Bug Tracking and Patch Tracking.

GP 2.4 Assign Responsibility

Assign responsibility and authority for performing the process, developing the work products, and providing the services of the requirements management process.

The PEP's owner is the one to assign responsibilities related to his requirement.

GP 2.5 Train People

Train the people performing or supporting the requirements management process as needed.

The Python Project's webpage has the manual for new incomers so they can follow the rules and train on their own. The new people can also communicate with other members in case of questions and doubts.

GP 2.6 Manage Configurations

Place designated work products of the requirements management process under appropriate levels of control.

The PEP's owner controls its requirements or/and change request, and the BDFL controls if the PEP should be accepted or rejected.

GP 2.7 Identify and Involve Relevant Stakeholders

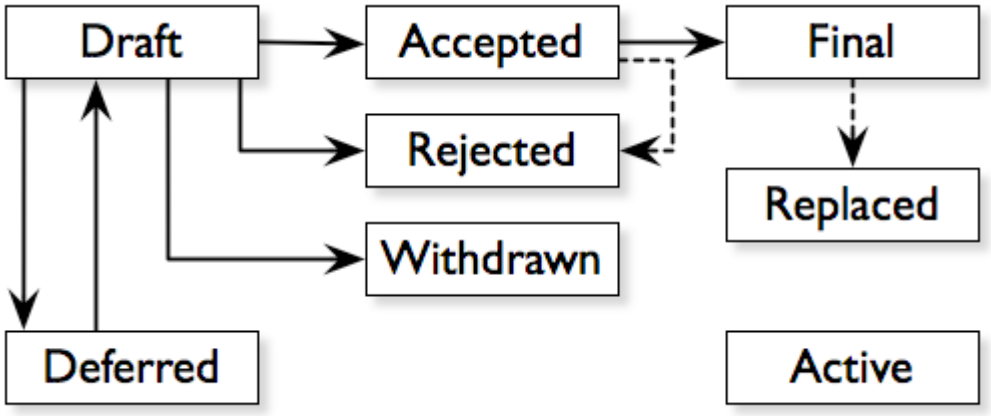
Identify and involve the relevant stakeholders of the requirements management process as planned.

The stakeholders involved in the requirements management are: PEP's owners, BDFL and the community.

GP 2.8 Monitor and Control the Process

Monitor and control the requirements management process against the plan for performing the process and take appropriate corrective action.

The PEP's author submits his PEP for approval by the BDFL, who will review and decide if the PEP is consistent or not, if it is not, the PEP is taken back to its author for corrective action so it can be submitted again and try a new approval.



GP 2.9 Objectively Evaluate Adherence

Objectively evaluate adherence of the requirements management process against its process description, standards, and procedures, and address noncompliance.

Not found in the Python Project.

GP 2.10 Review Status with Higher Level Management

Review the activities, status, and results of the requirements management process with higher level management and resolve issues.

Not found in the Python Project.

8. Evaluation of documentation in Open Source Software

This section describes an initial version of the proposed documentation assessment framework. As identified during task 1.2, there are no readily available metrics to assess the quality of documentation available for an F/OSS product.

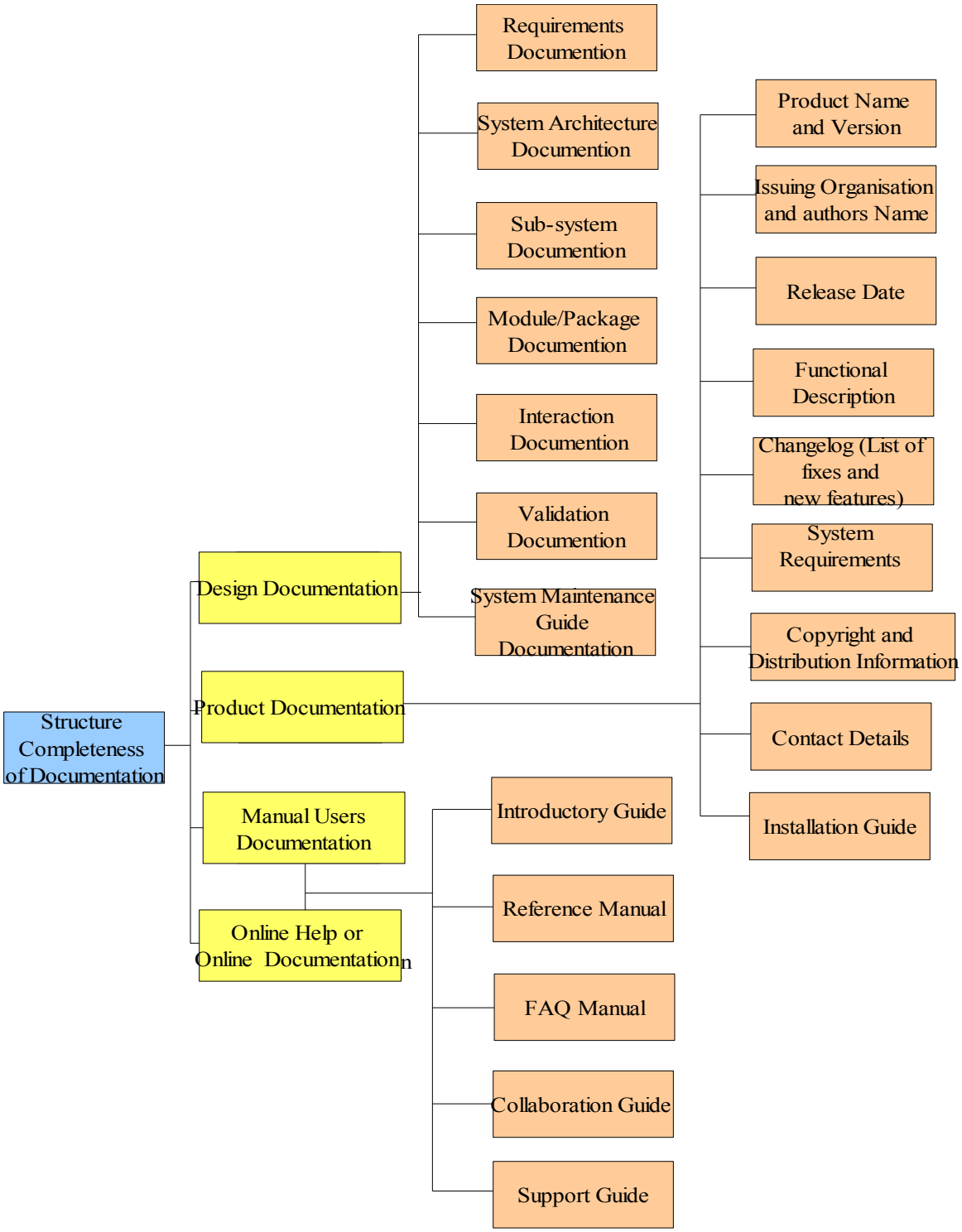


Figure 5: Part of an Open Source Software Documentation for a completeness documentation.

In section 8.1, we inventory the different type of documentation and describe them. Later, we will use this list to propose metrics for measure the completeness of documentation provided by a F/OSS product. In section 8.2, we present a definition of a document quality based on the IEEE 1063 standard for software user documentation. This IEEE standard highlights the adequate structure for documentation documents. Finally, in Section 8.3, we propose a list of criteria for evaluating functional description documents that we have elaborated.

8.1 Different Type of Documentation: Documentation Completeness

The completeness of documentation and the adequacy of its content directly contributes to product evolvability since it will facilitate use by a larger population size. Furthermore, documentation documents are themselves an integral part of the product and therefore, their own degree of evolvability impacts the evolvability of the product as a whole. In turn, regarding documentation, we must produce metrics that measure documentation completeness, documentation adequacy, compliance to documentation standards and documentation evolvability.

As shown in Figure 3, FOSS present various kinds of documentation. FOSS projects with a high quality documentation are expected to have different documentation parts listed below. The existence of each part contributes to the « completeness » of documentation. We will later define more formal metrics to assess the completeness of documentation, currently, we simply inventory and describe each documentation type below.

8.1.1 Design Documentation

Documents describing the design, implementation and testing of a system are essential if the program is to be understood and maintained. A Design Document is a description of a software product that a software designer writes in order to give a software development team an overall guidance of the architecture of the software project. It usually accompanies an architecture diagram and has pointers to the detailed feature specifications of smaller pieces of the design. A design document is practically required to coordinate a large team under a single vision.

Ideally, the Design documentation should include the items, as show on Figure 5 :

- A requirements documentation and an associated rationale. The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process. This could be the functional and non functional requirements, users requirements or system requirements.
- A document describing the architecture system : The architecture design uses information flow characteristics, and maps them into the program structure. Transformation mapping method is applied to exhibit distinct boundaries between incoming and outgoing data. The Data Flow diagrams allocate control input, processing, and output along three separate modules.
- For each program (sub-system...) of the system, a description of the architecture of the program
- For each component of the system (module, package...), a description of its functionality and interface
- A document describing Interaction between users and the system. It describes internal and external program interfaces as well as the design of human interface.
- A validation document describing how each program is validated and how the validation information relates the requirements.
- A system maintenance guide which describes known problems with the system, describes which parts of the system are hardware or software dependent and also describes how evolution of the system has been taken into account in its design.

8.1.2 Product Documentation

The software product should have a file in a text format « Read Me » or other kind of text file that includes the following:

- Product Name and Version
- Issuing organisation (and Authors Name)
- Release date
- Functional description – Description of services provided
- A change log : this is the list of fixes and new features
- System requirements (CPU, RAM, disk space, operating systems supported)
- Copyright and distribution information (rules for people who want to distribute your product)
- Contact details (email, phone, fax website and postal address)
- Installation guide & System requirements
- System Interoperability

The «Read Me » file is important in open source software because everybody who might be interested in an open source software product is expecting it, including reviewers, users, or people who want to use it or to contribute to it.

8.1.3 Manual Users documentation / Online Help or online documentation

There are many kinds of documentation of interest to various kind of users. In FOSS, we commonly find these types of user documentation:

- Introductory Guide (for beginner) or Tutorial Guide– Getting started with the system
- Reference Manual – Details of all system functionalities
- FAQ Manual
- Collaboration guide (developer info, contributor, ...) (this could be online or manual)
- Support Guide (which is sometimes in the form of a Wiki where users can contribute new information)

8.2 Document quality

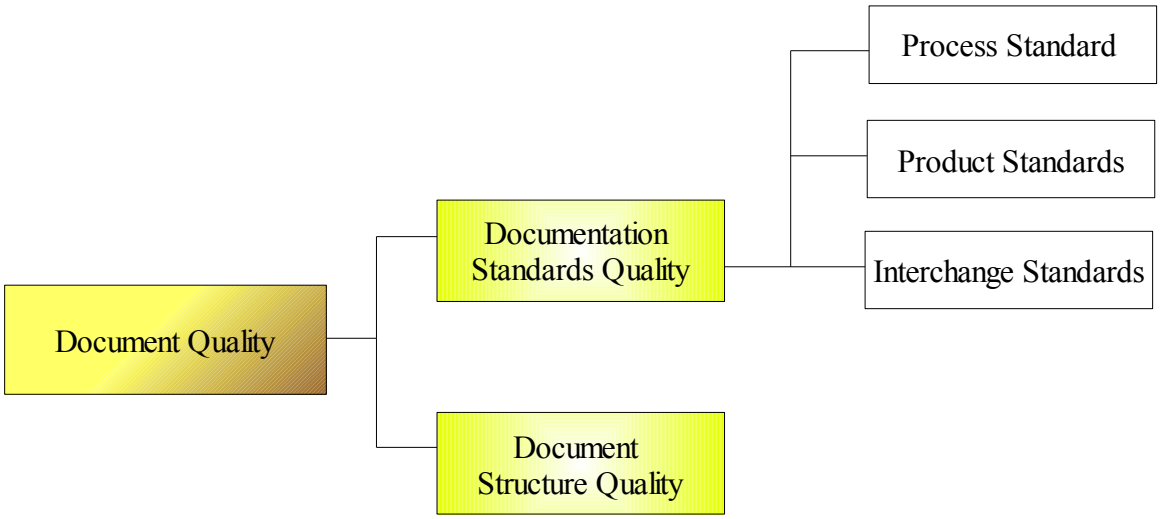


Figure 6: Description of a document quality

Document quality is as important as program quality for Open Source Software development. Without information on how to use a system or how to understand it, the utility of that system is degraded. The document structure and documentation standards has a major impact on readability and usability and it is important to design this carefully when creating documentation. Figure 6 below present how we structure documentation quality. Each criteria is detailed in sub-sections below.

8.2.1 Document Structure

As with software systems, it is important to design document structure so that the different parts are as independent as possible. The IEEE 1063 standard for software user documentation proposes that the structure of a document should include the components as show below.

Component	Description
Identification data (documentation title, documentation version and date published, software product and version, issuing organisation, ...)	Data such as title and identifier that uniquely identifies the document
Table of contents	Chapter/Section names and page number
List of illustrations	Figure number and title
Introduction	Defines the purpose of the document and a brief summary of the contents
Information for use of the documentation	Suggestions for different readers on how to use the documentation effectively
Concepts of operations	An explanation of the conceptual background to use of the software
Procedures	Directions on how to use the software to complete the tasks that it is designed to support
Information on software commands	A description of each of the commands supported by the software
Errors messages and problem resolution	A description of the errors that can be reported and how to recover from these errors
Glossary	Definitions of specialized terms used
Related information sources	References or links to other documents that provide additional information
Navigational features	Features that allow readers to find their current location and move around the document
Index	A list of key terms and the pages where these terms are referenced
Search capability	In electronic documentation, a way of finding specific terms in the document

The structure provided by IEEE1063 is a good starting point however, we must still develop a complete model suited to the evaluation of the different types of FOSS documentation. All in all, it will not be expected that every documentation document produced respects all the criteria enumerated in the table above.

8.2.2 Documentation Standards

Documentation standards is important and useful for document quality assurance. Documents produced according to appropriate standards have a consistent appearance, structure and quality. There are many others standards that may be used in the documentation process; In particular when writing documentation in open source software projects.

Process standards: they define the process which should be follow for high-quality document production. One possible process (iterative process) will be drafting, checking, revising and re-drafting which should be

continued until a document of acceptable quality is produced. This part of the evaluation is more related to the quality characteristics “Established Process Coverage” however, we just mention it here.

Documentation Product standards

Some of the product standard which should be developed are:

- Document identification standards : each document must be uniquely identified. If possible, with a formal identifier
- Document structure standards: as presented above, a document structure standards should have those information.
- Document presentation standards: document presentation standards define a « house style » for documents and they contribute significantly to document consistency
- Document update standards: as a document is changed to reflect changes in the system, a consistent way of indicating these changes should be used.

Interchange standards

FOSS projects produce many electronic documents. Document interchange standard is therefore important for robustness and evolvabiliy of the product. The use of interchange standards allow documents to be transferred electronically and re-created in their original form.

8.3 Study for the case of Functional Description Documentation

Beside assessing the quality of documentation generically as presented in the two previous sections 8.1 and 8.2, it is also possible to qualify the quality of each particular type of documentation. In this section, we present our initial effort for producing a list of criteria for evaluating functional description documents (FDD). This is still work in progress and we expect to develop advanced analysis to be use in the later part of the QUALOSS project. Currently, the list of criteria is quite exhaustive and we must select those that are unavoidable to produce a high quality documentation while keeping FOSS in mind.

We have structured the criteria based on the main topic expected to be found in documents describing the software product functions. All such documents is therefore providing the Functional Description Documentation (FDD)

FDD1 (Introduction)

It describes the:

- **FDD1-1 (Purpose)**
- **FDD1-2 (Scope)** and
- **FDD1-3 (Organization)**

of the Functional Description Documentation.

FDD2 (Software Overview)

- **FDD2-1 (Product description)**: Describes briefly why the software (or upgrade) is being developed, and lists the most important features and capabilities.
- **FDD2-2 (Product functional capabilities)**: Presents a list of the functions that the software will be required to perform. Where a product feature comprises several functional capabilities, a table may be developed to illustrate these relationships.

- **FDD2-3 (User characteristics)**: Describes the intended users of the software in terms of job function, specialized knowledge, or skill levels. Considers various user classes or profiles such as managers, engineers, equipment operators, IT support staff, and network or database administrators.
- **FDD2-4 (User operations and practices)**: Describes how persons will normally use the software, and the tasks they will most frequently perform. Also covers how users might use the software on an occasional basis, such as creating data backups or importing data from another program.
- **FDD2-5 (General constraints)**: Algorithm limitations, user interface limitations, and data limitations.
- **FDD2-6 (Assumptions)**: Lists any assumptions that were made in specifying the functional requirements.
- **FDD2-7 (Other software)**: How does the program interact with other software, for example, spreadsheet or database systems.

FDD3 (Specific Function Descriptions)

This section is repeated for each function of the software (some examples of functions are: engineering calculations, sorting or sequencing, other operations relating inputs to outputs, validity checks on inputs, error handling and recovery).

- **FDD3-1 (Description)**: Describes the function and its role in the software.
- **FDD3-2 (Inputs)**: Describes the inputs to the function. Where user interface (UI) elements are present, these are described (some examples of UI elements are check boxes, dropdown lists, and alphanumeric fields). Input validation strategy, allowed data types and value ranges are specified for each input.
- **FDD3-3 (Processing)**: Describes what is done by the function. Where algorithms, equations, or other logic are used, they are eventually described here. If calculations are done utilizing the methods of specific standards or references, these are cited.
- **FDD3-3 (Outputs)**: Describes the outputs of the function. Where a user interface description is relevant, it is included. Reports generated are also defined.

FDD4 (External Interfaces)

The interfaces in this section are specified by documenting: the name and description of each item, source or input, destination or output, ranges, accuracy and tolerances, units of measure, timing, display formats and organization, and data formats.

- **FDD4-1 (User Interfaces)**: Describes all major forms, screens, or web pages, including any complex dialog boxes. This is usually best done via non-functioning screen shots simulating usage scenarios, and it may be presented in a separate document. Specific items are described for each screen such as input fields, control buttons, sizing options, and menus. The navigation flow of the windows, menus, and options is described, along with the expected content of each window.
- **FDD4-2 (Hardware Interfaces)**: Describes the equipment needed to run the software, and also other output or input devices such as printers or handheld devices.
- **FDD4-3 (Software Interfaces)**: Describes any software that will be required in order for the product to operate fully. Also describes any software that the software product will interact with such as operating system platforms supported, file import and export, networking, automation, or scripting. Specifies whether the users must provide the interfacing software themselves, and any special licensing requirements.
- **FDD4-4 (Communication Interfaces)**: Describes how the software product will communicate with itself (for multi-platform applications) or other software applications, including items such as networking, email, intranet, and Internet communications.

FDD5 (Performance)

Discusses items such as response times, throughput requirements, data volume requirements, maximum data file size or problem complexity, maximum number of concurrent uses, and peak load requirements (for web-based applications). Includes expected response times for entering information, querying data files and databases, performing calculations of various complexities, and importing/exporting data.

FDD6 (Design Constraints)

Discusses items constraints that affect software design choices such as memory constraints involving minimum and maximum RAM and hard disk space, and limitations arising from hardware, software or communications standards.

FDD7 (Attributes)

- **FDD7-1 (Security)**: Describes any password-protected access levels such as operator, engineer/modeler, manager, database administrator-and which functionality will be accessible to each access level. If relevant, describes the planned approach to locking the software.
- **FDD7-2 (Reliability, Availability, Maintainability)**: Describes requirements items such as days or weeks of continuous operation, strategy for data recovery, code structuring for ease of future modification.
- **FDD7-3 (Configuration and Compatibility)**: Describes requirements such as those connected with individual customization or operation in specific computing environments.
- **FDD7-4 (Installation)**: Describes the planned method for installation: done by the user independently, done by customer company internal IT services, done by an external contractor.
- **FDD7-5 (Usability)**: Describes items that will ensure the user-friendliness of the software.

  These are some criteria on software usability elements.
- **FDD7-5-1** Same screen appears each time application is launched
- **FDD7-5-2** Consistent and logical navigation flow
- **FDD7-5-3** Uses standard GUI features (e.g., pull-down menus, dialog boxes, toolbar buttons)
- **FDD7-5-4** Application windows have consistent look and feel
- **FDD7-5-5** Data formats are consistent throughout application windows
- **FDD7-5-6** Menu options can be accessed via keyboard commands and/or arrow keys. Mouse-only access to options should be avoided.
- **FDD7-5-7** Controls on page must respond properly to Tab order and hot-keys (alt-keys). A user should be able to use the Tab key to move from one control to the next. This should work in a logical manner such as from left to right, or top to bottom.
- **FDD7-5-8** Interface recovers gracefully from anticipated user errors (e.g., invalid input)
- **FDD7-5-9** Information and error messages are useful, accurate, and correctly spelled
- **FDD7-5-10** Unnecessary warnings do not appear
- **FDD7-5-11** Technical support information identical to that stated in documentation
- **FDD7-5-12** The organization Copyright, Ordering Information, and Disclaimer Notice elements for software appear as required.

FDD8 (Additional Requirements)

Describes other characteristics the software must have, that were not covered in the prior sections.

- **FDD8-1 (Database)**: Describes any specific requirements relating to the database, such as database type (e.g. relational), capability to handle large text fields, real-time capability (e.g. handling an incoming data stream, as from instruments), multi-user capability, special requirements relating to queries and forms.
- **FDD8-2 (Administration)**: Includes any periodic updating or data management needed.
- **FDD8-3 (User documentation)**: Describes the user documentation to be delivered with the software, including both hard copy and online requirements.
- **FDD8-4 (Other requirements):** Describes any other requirements not already covered above that need to be considered during the design of the software.

9. Issues for the QualOSS Advanced Quality Model

Open issues and information that need to be refined to create the “advanced” QualOSS quality model; that is, these issues are relevant for the QualOSS model but cannot be addressed in the first iteration. This can include complete quality models, parts of models that need to be further refined, or metrics that cannot be collected in the first iteration. The advanced model workpackage (WP 4) will prioritize and potentially enhance the list of open issues for the second iteration.

In the sections 3-6, we have already highlighted metrics that will be investigated as advanced quality model.

In addition, there are the following main problems with initial model: Currently, the model assumes that (a) all data are available as requested, and (b) the metric values are comparable between different measurements

However, in practice, these assumptions are often not met.

Missing data: It is to be expected that some projects will miss some required data. Typical strategies to address this are: (a) Ignore the project, (b) ignore the missing quality characteristic, (c) use alternate metrics to interpret the quality characteristic, or (d) try to predict/interpolate the missing data (e.g., by using Bayesian analysis).

The question of missing data is further complicated by the fact that the fact that data are missing in itself can be an important statement, such as in a repository of vulnerability.

The advance models will have to investigate these strategies to identify the best suited one for QualOSS.

Comparability of metrics: Even when using the same metrics, results of measurements may not be directly comparable. For example, even when computing the same metrics with different tools, the result may be different. Computing similar metrics, for example for different programming languages will lead to different results. And for manual measurements, it is obvious that there is a bias introduced by the measurer. To address these comparability issues, we introduced the notion of indicators (LEVEL OF ABSRACTION) that interpret their underlying metrics into a common scale. However, this will have to be further examined when constructing advanced models; in particular, our insights gained so far suggests that the notion of looking at trends from historical data might prove useful.

10. Interpretation Guide / QualOSS User Manual

The previous sections described which metrics we intend to use to measure the QualOSS quality model. However, to be able to interpret the model's results for a specific quality characteristic, the user needs to be presented with an aggregated result, which comprises and summarizes all individual metrics and characteristics it is composed of. This section describes how we intend to aggregate metrics and characteristics, thus making the model interpretable.

For this purpose, we introduce indicators that wrap metrics and abstract from their concrete value, thus providing an interpretation. Thus, for each quality characteristic, the QualOSS quality model will assign a comparable indicator value, which can be aggregated. The individual metric may be concerned with measuring an attribute or its trend, while the indicator is supposed to summarize a metric's value (or a set of metrics) in a traffic-light manner. For example, an indicator for product complexity may summarize several metrics and offer a single interpretable value (such as complexity acceptable, risky, unacceptable). In other words, the indicators contain the formulae to condense underlying metrics into a single value that enables to assess the corresponding quality characteristic; that is, to answer its underlying question. Currently, we have not yet identified authoritative and alternate formulae. Thereby, alternate formulae are supposed to be used when the authoritative formulae are not applicable (e.g. for reasons of missing data, or if the data quality is too low). As described in the previous section, such questions are to be tackled with the advanced quality models; hence, authoritative and alternate formulae will be used to represent and implement these findings.

Two aspects need to be addressed:

- Normalization of metric output: Indicators need to use a common scale to interpret the underlying values. For example, this can be a range of 0 to 100, with values from 0 to 34 indicating “critical” ranges of metrics, from 35 to 69 indicating some problems, and values from 70-100 indicating that the underlying metrics are acceptable.
- Weighting of metrics and characteristics: Quality characteristics are of different importance; this will be represented through weights. Although the user will have to be able to adjust these weights, we will propose weights for different usage scenarios, which will be elicited from stakeholders.

Aggregation in this view basically consists of calculating weighted means of sub-characteristics. However, we are aware that there are limitations to this approach. For example, some characteristics are “show-stoppers”; that is, if one metric exceeds thresholds, it has over-proportional influence on the results. For example, for one particular user, a F/OSS product may be unacceptable if it exceeds a certain threshold in reliability; as long as the value is not exceeded, reliability is not important at all. In that case, a simple weighting will not reflect the user's intention, as the weight would have to be equal to zero if the metric does not exceed the critical value, and equal to infinity if it does. Therefore, the QualOSS model resp. the interpretation guide will have to be able to address such issues.

11. Summary and Conclusions

This report defines the the prototype QualOSS quality model in terms of a revised definition of the quality characteristics and identification of basic and advanced metrics to operationalize these definitions. It defines aspects to be addressed in the advanced quality models. These aspects include metrics and quality characteristics that we are not able to measure yet, as well as strategies for dealing with incomplete or incompatible data.

The prototype QualOSS quality model separates product-specific and community-specific aspects of robustness and evolvability. In particular, measurement of community aspects is enhanced by introducing process and documentation assessment frameworks, as the ability of a project to *consistently and predictably* deliver high quality software is connected with the maturity of their software processes. We have found that successful F/OSS projects use a range of practices and mature processes, so we believe that a process assessment approach is able to deliver useful results for QualOSS.

Further work is still required. The interpretation of the QualOSS prototype model needs to be defined; that is, for each quality characteristic, we need to define indicators that summarize and aggregate the underlying metric values. Task 1.4 will target this through the calibration of the quality models. In addition, the assessment frameworks need to be operationalized; that is, they need to be refined in a way that allows to assign a metric value to a project.

Another issue to be addressed in the next steps is weighting the different quality characteristics. In the remainder of WP1, we will elicit initial weightings from stakeholders

12. Appendix A: Product Metrics Tables

The table below help to systematically ask whether the content of a data type, data sources and historical evolution of data can be studied to assess the quality characteristics defining evolvability and robustness found at the leaves of our hierarchies.

IMPORTANT – This comment regards the analyses or metrics that can be specified in each cell. In particular, the table below contains an ambiguity. An analysis can be specified on a single instance of a data type or on a list of instances of that data type. For instance, an analysis can either work on an a single issue in a issue tracking system or on a set of issues, that is, searching data fields of a single issue vs. searching certain data fields of an set of or all the issues in the database. In addition, each data source also has a row to specify historical analysis of its data content. For example, variation in numbers of bugs reported over time.

In addition to data sources of D1.1, we have added Discussion Forum (accessible via a Web or News server)

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List	Executable/Library Files	
	Source Files	
	Test Files (input + scripts)	
	Documentation Files	
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database	Request ID, Title, description	
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	
	Historical Analyses specific to Issue Tracking	
Mailing List Archive, discussion forum (accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website	Static webpages	
	Dynamic webpages (wiki or others)	
	Historical Analyses specific to Website	
IRC logs	Discussion topic	
	Participants	
	Static text content	

	Historical Analyses specific to IRC Data	
Security Databases	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
FLOSSMETRICS, SQO-OSS and FLOSSMole	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

12.1 Metrics

12.1.1 Simple Analysis

Metric	Tools

12.1.2 Advanced Analysis

List of advanced analyses whose results could be transformed into interesting metrics for assessing evolvability and robustness and tools potentially useful to build on to compute these advanced analyses.

13. Appendix B: Product Robustness

13.1 Reliability – Fault Tolerance – Failure Tolerance

Definition: The capability of the software product to avoid failure and to maintain a specified level of performance when software faults are executed.

13.2 Mapping Data sources to Metrics of interest

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	ADVANCED – Single Instance (For jar or pyc) analysis to detect potential runtime failures: deadlocks, memory leaks, illegal memory accesses (array out of bound, dangling pointers, double free), ... (Argument: potential runtime errors lead to an unstable state which often leads to failure)
	Source Files	ADVANCED – Single Instance 1. Analysis of error handling (Argument: if exception are not handle properly in the code, it may lead to failure) 2. Analysis to detect potential runtime exceptions: deadlock detection, memory leaks, illegal memory access (array out of bound, dangling pointers, double free), ... (Argument: potential runtime errors lead to an unstable state which often leads to failure)
	Test Files (input + scripts)	ADVANCED – Single Instance 1. Test coverage: percentage of class, methods, basic block covered by testing (Argument: the more tests cover the code, the more are failures likely to be identified during testing and therefore addressed before release) 2. Run full test suite on executables and collect test log then determine the number of software crashes in test log (Analysis for crash evidences in log) (Argument: if tests show to much of an evidence of lack of robustness then that is bad; however, a certain number of failure shows that the test suite is good)
	Documentation Files	ADVANCED – Single Instance 1. Number of test environments as described in documentation (Argument: The more environments the product was tested on, the more reliable it is likely to be on this environment)
	Build Files	
	Historical Analyses specific to Product Distribution	1. Historical variation of number of empty catch bloc, bad use of throws (and of potential runtime error) in software product within the same major release. (Argument: steady improvement in error checking over time show dedication by developers to improve robustness) 2. Historical variation of code coverage obtained by testing. (Argument: a constant high code coverage by test shows high level of dedication by testers to maintain and improve robustness)
Version Control Repository (VCR)	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database (ITD	Request ID, Title, description	BASIC – Set of Instances 1. Numbers of issues in the ITD (possible variations based on the text containing the word “CRASH”, issues related to a single, a set of or all releases and based on the status and resolution flag of issues) (Argument: the number of issues and their ratio between solved and open issues definitely indicate the level of robustness of the product)
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	BASIC – Set of Instances 1. Numbers of issues in the ITD (possible variations based on issues related to all versions vs. single and based on closed vs open issues.) 2. Number of issues whose additional content contains the word CRASH (possible variations based on issues related to all versions vs. single and based on closed vs open issues.)
	Historical Analyses specific to Issue	ADVANCED – Set of Instances

	Tracking	1. Historical variation of issues (alternatively only those containing the word CRASH) 2. Number of undetected defects left in specific version as predicted by statistical analysis of full history of issues (Argument: statistical prediction based on the past often yield a good indicator for the future)
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	BASIC – Set of Instances ?? Number of messages whose subject line contains the word CRASH
	Target email and list of recipients	
	Text Content and Attachments	ADVANCED – Set of Instances ?? Number of messages whose content body contains the word CRASH
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	ADVANCED – Set of Instances Source Code Review process are explicitly described as part of the verification process and require the recording of reviews so issues highlighted are addressed before committing code Number of defect discovered during code review (Argument: Code review has shown to be one of the most efficient mechanism to discover defects in code)
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	BASIC – Set of Instances 1. Number of all exposures and vulnerabilities for a software product 2. Number of all exposures and vulnerabilities for a single software product release 3. Number of all exposures and vulnerabilities for all minor releases of a software product under the same majors release. 4. The 3 analysis above for sever vulnerabilities only, that is, vulnerabilities that could yield to system crash or external control been taken)
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	ADVANCED – Set of Instances 1. Number of vulnerabilities with patches proposed as compared to all vulnerabilities (Argument: Risk of vulnerabilities alleviated if patch exists) 2. Average time between vulnerabilities posted and patch provided.
	Historical Analyses specific to Security Data	
Publication Database (PD)	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites (GN)	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.3 Details on Metrics

13.3.1 Basic Metrics

ALL means bugs of all status

OPEN means bug of status open

Metric	Tools
Number of issues of ANY status for all product releases (same with "CRASH" in title and/or details, description and/or comments) (Argument all basic metrics except the last one: the number of issues and their ratio between solved and open issues definitely indicate the level of robustness of the product)	Any tools with an advanced search functionality enabling search of the title, the body and comments of an issue tracking systems Example of Existing Tools: • Bugzilla Advanced Search (bugzilla is used by Eclipse, Apache), • SourceForge Tracker Advanced Search, • Jira Search New (used by Apache Jakarta projects) What about? • FSF repository whose strategy is to use mailing lists to report and comment on issues? • Unstructured repositories such as Trac (could search tickets)?
Number of issues of status OPEN for all product releases (same with "CRASH" in title and/or details, description and/or comments)	same tool as above
Ratio of OPEN vs. ANY bugs for all product releases (same with "CRASH" in in title and/or details, description and/or comments)	same tool as above
Number of issues of ANY status for a specific product release (same with "CRASH" in in title and/or details, description and/or comments)	same tool as above
Number of issues of status OPEN for a specific product release (same with "CRASH" in title and/or details, description and/or comments)	same tool as above
Ratio of OPEN vs. ANY bugs for a specific product release (same with "CRASH" in title and/or details, description and/or comments)	same tool as above
Number of subjects in mailing list or news forum containing "CRASH"	Existing • SourceForge Mailing List Advanced Search What about? • Eclipse Mailing List Subject Search (inconvenient

	must search one page at a time) • FSF must subscribe to mailing list
Checks related to user exception handling (PMD for Java, PyLint for Python, CETIC analyzer for Java, what about for C/C++ and Ada?) (Argument: if error handling is not handle properly in the code, it may lead to failure)	Existing Tools: • PMD, SQUALAnalyzer for Java • PyLint for Python What about? • for C/C++ and Ada

13.3.2 Advanced Metrics

- Check for potential run time exception in executable or source files: number of potential deadlocks, memory allocation accesses, memory leaks (Argument: potential runtime errors lead to an unstable state which often leads to failure)
- Check for more complex user define exception and their proper handling (Argument: if error handling is not handle properly in the code, it may lead to failure)
- Percentage of code covered by testing (classes, methods, blocks) (Argument: the more tests cover the code, the more are failures likely to be identified during testing and therefore addressed before releases)
- Historical variation of test coverage over several product releases (Must improve or stabilize at certain thresholds: classes: 100%, methods: 80%, basic blocks: 60% for example. (Argument: a constant high code coverage by test shows high level of dedication by testers to maintain and improve robustness)
- Number of issues of ANY status for a specific set of product releases (e.g. all release under the same major release) (same with “CRASH” in title and/or details, description and/or comments) (Argument for this and two metrics below: the number of issues and their ratio between solved and open issues definitely indicate the level of robustness of the product)
- Number of issues of status OPEN for a specific set of product release (e.g. all release under the same major release) (same with “CRASH” in in title and/or details, description and/or comments)
- Ratio of OPEN vs. ANY bugs for a specific set of product release (e.g. all release under the same major release) (same with “CRASH” in title and/or details, description and/or comments)
- Fitness with a logarithmic function of the function defined by the number of issues (Y-axis) over time (X-axis) for all minor releases of a same major release. (time period for the X-axis must be set to a particular value, for example, 1 month period. This metric can also be defined over an inverse logarithmic function of the delta of issues reported over each period + also a variant on issues status and resolution flag and also with “CRASH” in title, description body or comments bodies) (Argument: statistical prediction based on the past often yield a good indicator for the future)
- Number of undetected defects left in specific version as predicted by statistical analysis of full history (Argument: statistical prediction based on the past often yield a good indicator for the future)
- Number of crash evidences in test log (after executing the full test suite provided as part of the F/OSS product or by stress tests executed by the quality assessor. (Argument: if tests show a great evidence of lack of robustness then that is bad; however, the exhibition of a certain number of failures shows that the test suite is good)
- Number of environments on which the software product was tested as described in the documentation (Argument: The more environments the product was tested on, the more reliable it is likely to be on this environment)

13.4 Product Robustness – Reliability – Fault Tolerance – Error Tolerance

Definition: The capability of the software product to avoid failures and to maintain a specified level of performance in cases of infringement of its specified interface.

In the context of identifying metrics for this deliverable, the insight we gained showed that it is hard to separate the different subconstructs of failure/fault tolerance

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List	Executable/Library Files	
	Source Files	ADVANCED • Check for Code/SQL injection (Argument: injection arevcases of malicious infringement to the specified interface) (Additional code check related to fault tolerance will be identified) • Methods/Classes that catch generic exceptions (Argument: catching generic exception make it much harder to react appropriately to go from unstable back to a stable state of execution) • ??Check UI libraries used?? (Argument: a UI such as java GUI is fault tolerant, in cases where exceptions are raised, the UI usually stays up and running only dumping stack trace on background console)
	Test Files (input + scripts)	ADVANCED Check that test for Code and SQL injection exist (Argument: If test cases exist for injection, it is likely to be identified before releases)
	Documentation Files	ADVANCED Installation, Administration, and or User documentation explain the environment and scope in which the product was tested and remains functional even in case of some failures. (Argument: if the user is informed of the framework to stay in, he is likely to be less frustrated when crashes occur due to uses outside the foreseen scope)
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out process programs (VC metadata)	ADVANCED (The analysis below is maybe more related to Community Robustness Process) Checkin process programs automatically runs all tests and the checkin is only granted if a smaller number of failures are generated then prior to the change set. Further, process programs can run additional checks on code convention respect, ...
	Historical Analyses specific to Version Control Data	
Issue Tracking Database	Request ID, Title, description	
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website	Static Pages	
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	

Security Databases	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
FLOSSMETRICS, SQO-OSS and FLOSSMole	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.5 Product Robustness – Reliability – Recoverability

Definition: 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.

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List	Executable/Library Files	
	Source Files	ADVANCED – Single Instances 1. Software patterns related to recoverability are present in the source code, for example, presence of a thread that wakes up periodically to auto-save data (Argument: if patterns related to recoverability are found in the code, it is likely that the product has some degree of recoverability.
	Test Files (input + scripts)	ADVANCED – Single Instances 1. Test script or test procedure tests the software product for recoverability, for example, by bringing down the software application and then starting it again to verify that data could be recovered (Argument: if some tests actually check for recoverability, it is more likely that the product has the intend to provide a recoverability feature)
	Documentation Files	ADVANCED – Single Instances 1. User Documentation has content dedicated to Recoverability (for example, section on recoverability mentions how to activate and customize data recoverability for the software product or how to install it to improve recoverability) (Argument: if the documentation addresses recoverability, it is more likely that the product has the intend to provide a recoverability feature)
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out process programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database	Request ID, Title, description	BASIC – Set of Instances

		1. Numbers of issues whose title or description body contains the word RECOVER (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Env. Specs related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	BASIC – Set of Instances 1. Numbers of issues whose title or description body contains the word RECOVER (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags) (Argument: recoverability issues reported in bug tracking system are highly likely to use the words recovered, recoverable, unrecoverable, ...)
	Historical Analyses specific to Issue Tracking	ADVANCED – Set of Instances 1. Historical variation of issues whose content contains the word RECOVER
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website	Static Pages	
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
FLOSSMETRICS, SQO-OSS and FLOSSMole	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.5.1 Basic Metrics

Metric	Tools
Number of issues of ANY status for all product releases with string “RECOVER” in title and/or details, description and/or comments)	Same search tools as for Failure tolerance
Number of issues of status OPEN for all product releases with string “RECOVER” in title and/or details, description and/or comments	Same search tools as for Failure tolerance
Ratio of OPEN vs. ANY bugs for all product releases with “RECOVER” in in title and/or details, description and/or comments)	Same search tools as for Failure tolerance
Number of issues of ANY status for a specific product release with string “RECOVER” in in title and/or details, description and/or comments	Same search tools as for Failure tolerance
Number of issues of status OPEN for a specific product release with string “RECOVER” in title and/or details, description and/or comments	Same search tools as for Failure tolerance
Ratio of OPEN vs. ANY bugs for a specific product release with string “RECOVER” in title and/or details, description and/or comments	Same search tools as for Failure tolerance

13.5.2 Advanced Metrics

- Software patterns related to recoverability are present in the source code, for example, presence of a thread that wakes up periodically to auto-save data
- Test script or test procedure tests the software product for recoverability, for example, by bringing down the software application and then starting it again to verify that data could be recovered
- User Documentation has content dedicated to Recoverability (for example, section on recoverability mentions how to activate and customize data recoverability for the software product or how to install it to improve recoverability)
- Historical variation of issues whose content contains the word RECOVER and deviation from logarithmic curve (or at least a “steep” linear decrease)

13.6 Product Robustness – Reliability – Availability

Availability (IEEE): The degree to which a system or component is operational and accessible when required for use.

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List	Executable/Library Files	ADVANCED - Single Instances 1. Product is build on libraries that have proven track record regarding availability
	Source Files	ADVANCED – Single Instances 1. Patterns showing the ability to handle and manage multiple client connections or multi tasking, for example, proper use of multi threading or processing, Potential use of a queue scheduling and management system for system with high demand.
	Test Files (input + scripts)	ADVANCED – Single Instances

		1. Test script contains stress test to assess the availability of the software product before releases.
	Documentation Files	ADVANCED – Single Instances For server product: 1. User doc has sections on redundancy, load balancing 2. Installation manual mentions how to set the product to increase availability
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out process programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database	Request ID, Title, description	BASIC – Set of Instances 1. Numbers of issues whose title or description body contains the word AVAILABILITY or ACCESS (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Env. Specs related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	BASIC – Set of Instances 1. Numbers of issues whose additional comment contains the word AVAILABILITY or ACCESS (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website	Static Pages	
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	

General News sites	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
FLOSSMETRICS, SQO-OSS and FLOSSMole	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.6.1 Basic Metrics

The metrics below are approximative at best

Metric	Tools
Number of issues of ANY status for all product releases with string “AVAILABILITY” in title and/or details, description and/or comments)	Same issue search tools as for Failure tolerance
Number of issues of status OPEN for all product releases with string “AVAILABILITY” in title and/or details, description and/or comments	Same issue search tools as for Failure tolerance
Ratio of OPEN vs. ANY bugs for all product releases with “AVAILABILITY” in in title and/or details, description and/or comments)	Same issue search tools as for Failure tolerance
Number of issues of ANY status for a specific product release with string “AVAILABILITY” in in title and/or details, description and/or comments	Same issue search tools as for Failure tolerance
Number of issues of status OPEN for a specific product release with string “AVAILABILITY” in title and/or details, description and/or comments	Same issue search tools as for Failure tolerance
Ratio of OPEN vs. ANY bugs for a specific product release with string “AVAILABILITY” in title and/or details, description and/or comments	Same issue search tools as for Failure tolerance

13.6.2 Advanced Metrics

- Product is build on libraries that have proven track record regarding high availability
- Software Patterns in source code showing the ability to handle and manage multiple client connections or multi tasking, for example, proper use of multi threading or processing, Potential use of a messaging system for product with high demand.
- Test script contains stress tests to assess the availability of the software product before releases.
- (For Server Product) User doc has sections on redundancy, load balancing

- (For Server Product) Installation manual mentions how to set the product to increase availability

13.7 Product Robustness – Security – Confidentiality

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	ADVANCED – Single Instance 1. Use of renown libraries/framework for authentication, authorization and access control (Argument: a know access control framework simplify the integration the appropriate confidentiality in a product)
	Source Files	ADVANCED – Single Instance 1. Use of appropriate code and pattern to interact with an authentication, authorization and access control framework 2. Use of appropriate code and pattern to encrypt data before streaming it out of the application. (Argument: On the top of using a framework, the code must use it properly to avoid confidentiality leakage, for example, information streamed out of the application are encrypted prior, ...)
	Test Files (input + scripts)	ADVANCED – Single Instance 1. Test suite contains tests that attempt to gain access to the application or its data without appropriate rights) (Argument: such test scripts would show that confidentiality is tested for)
	Documentation Files	ADVANCED – Single Instance 1. User Documentation explains how the software product handle authentication, authorization, access control, and encryption. 2. Installation Manual explain how to setup the product to guarantee a high level of confidentiality (Argument: documentation that contains information related to confidentiality shows that confidentiality is addressed by the software product.
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository (VCR)	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database (ITD)	Request ID, Title, description	BASIC – Set of Instances (not a very reliable metric) 1. Numbers of issues whose title or description body contains one of the words AUTHENTICATION AUTHORIZATION or ACCESS CONTROL (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	BASIC – Set of Instances 1. Numbers of issues whose additional comment contains one of the words AUTHENTICATION AUTHORIZATION or ACCESS CONTROL (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	
	Wiki Pages	
	Historical Analyses specific to Website Data	

IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	BASIC – Set of Instances 1. Number of sever vulnerabilities or exposures for all releases of a product (Argument: raw data can provide overall information as to the expected level of confidentiality one may expect from the product.) 2. Ratio of sever vulnerabilities with patch vs all for a specific release of the product (Argument: Users want to make sure the specific version in use (or to be integrated) contains literally no known vulnerabilities and exposures
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	ADVANCED – Set of Instances 1. Verify that patches are provided quickly after the initial report (for example, no later than a week after initial report) (Argument: patch created promptly maintain a high confidence from users regarding low risk of potential intrusion and confidentiality leaks)
	Historical Analyses specific to Security Data	
Publication Database (PD)	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites (GN)	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.7.1 Basic Metrics

ALL means bugs of all status

OPEN means bug of status open

Metric	Tools
Numbers of issues whose title or description body contains one of the words AUTHENTICATION AUTHORIZATION or ACCESS CONTROL (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags) (Not a very reliable metric)	same issue tracking advanced search as for Failure tolerance
Numbers of issues whose additional comment contains one of the words AUTHENTICATION AUTHORIZATION or ACCESS CONTROL (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)	
Number of sever vulnerabilities or exposures for all	Advanced search of the National Vulnerability

releases of a product (Argument: raw data can provide overall information as to the expected level of confidentiality one may expect from the product.)	Database (http://nvd.nist.gov/nvd.cfm?advancedsearch)
Ratio of sever vulnerabilities with patch vs all for a specific release of the product (Argument: Users want to make sure the specific version in use (or to be integrated) contains literally no known vulnerabilities and exposures)	Advanced search of the National Vulnerability Database (http://nvd.nist.gov/nvd.cfm?advancedsearch)

13.7.2 Advanced Metrics

- Use of renown libraries/framework for authentication, authorization and access control (Argument: a know access control framework simplify the integration the appropriate confidentiality in a product)
- Use of appropriate code and pattern to interact with an authentication, authorization and access control framework (Argument for this analysis and the next one: On the top of using a framework, the code must use it properly to avoid confidentiality leakage, for example, information streamed out of the application are encrypted prior, ...)
- Use of appropriate code and pattern to encrypt data before streaming it out of the application.
- Test suite contains tests that attempt to gain access to the application or its data without appropriate rights) (Argument: such test scripts would show that confidentiality is tested for)
- User Documentation explains how the software product handle authentication, authorization, access control, and encryption. (Argument for this analysis and the next one: documentation that contains information related to confidentiality shows that confidentiality is addressed by the software product.)
- Installation Manual explain how to setup the product to guarantee a high level of confidentiality
- Verify that patches are provided quickly after the initial report (for example, no later than a week after initial report) (Argument: patch created promptly maintain a high confidence from users regarding low risk of potential intrusion and confidentiality leaks)

13.8 Product Robustess – Security – Integrity

Integrity (ISO): The degree to which a system or component is able to protect the accuracy and completeness of information and processing methods. This includes preventing unauthorised modification or destruction of information (CNSS, 2006).

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	ADVANCED – Single Instance 1. Use of renown libraries/framework for encryption and digital signature (Argument: a know access control framework simplify the integration the appropriate confidentiality in a product)
	Source Files	ADVANCED – Single Instance 1. Use of appropriate code and pattern to saving and checking data integrity before streaming it in and out of the application. (Argument: On the top of using a framework, the code must use it properly to guarantee integrity)
	Test Files (input + scripts)	ADVANCED – Single Instance 1. Test suite contains tests that attempt to corrupt data and to process corrupted data) (Argument: such test scripts would show that integrity is tested for)
	Documentation Files	ADVANCED – Single Instance 1. Installation Manual explain how to setup the product to guarantee a high level of integrity (Argument: documentation that contains information related to integrity shows that integrity is addressed by the software product. For example, how to setup the product to use digital signature) 2. User Manuel explain to the user how to sign data (Argument: helping the user sign data show a concern for integrity) 3. (not integrity of the product but integrity of the product download) Product Distribution download packages provide their MD5 or other checksum. (Argument: Providing a checksum show that the project consider integrity with high priority)
	Build Files	
	Historical Analyses specific to Product	

	Distribution	
Version Control Repository (VCR)	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database (ITD)	Request ID, Title, description	BASIC – Set of Instances (not a very reliable metric) 1. Numbers of issues whose title or description body contains one of the words CORRUPTED CHECKSUM (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	BASIC – Set of Instances 1. Numbers of issues whose additional comment contains one of the words CORRUPTED CHECKSUM (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database (PD)	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites (GN)	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	

F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.8.1 Basic Metrics

ALL means bugs of all status

OPEN means bug of status open

Metric	Tools
Numbers of issues whose title or description body contains one of the words CORRUPTED CHECKSUM (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)	same issue tracking advanced search as for Failure tolerance
Numbers of issues whose additional comment contains one of the words CORRUPTED CHECKSUM (possible variations based on issues related to product, single release, set of releases and based on status and resolution flags)	same issue tracking advanced search as for Failure tolerance

13.8.2 Advanced Metrics

- Use of renown libraries/framework for encryption and digital signature (Argument: a know access control framework simplify the integration the appropriate confidentiality in a product)
- Use of appropriate code and pattern to encrypt data before streaming it out of the application.
- Test suite contains tests that attempt to corrupt data and to process corrupted data) (Argument: such test scripts would show that integrity is tested for)
- Installation Manual explain how to setup the product to guarantee a high level of integrity (Argument: documentation that contains information related to integrity shows that integrity is addressed by the software product. For example, how to setup the product to use digital signature)
- User Manuel explain to the user how to sign data (Argument: helping the user sign data show a concern for integrity)
- (not integrity of the product but integrity of the product download) Product Distribution download packages provide their MD5 or other checksum. (Argument: Providing a checksum show that the project consider integrity with high priority)

13.9 Product Robustness – Security – Compliance to Standards

Compliance to security standards: The degree to which a product complies with published security standards that are relevant to its functionality.

13.10 Mapping Data sources to Metrics of interest

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	ADVANCED – Single Instance 1. Are the security framework used to guarantee confidentiality and integrity

		renown for following X.509 standard, Kerberos, ...
	Source Files	ADVANCED – Single Instance (For Server Application) 1. Are the security specifications (confidentiality, integrity) found in product configuration files written in a renown language for specifying security policies such as XACML
	Test Files (input + scripts)	
	Documentation Files	ADVANCED – Single Instance 1. Are documents mentioning that the software product has been used in systems that are now certified Common Criteria level X (where is must be certified)
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository (VCR)	Time stamp (VC metadata)	
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database (ITD)	Request ID, Title, description	
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	ADVANCED – Set of Instances 1. Can we find reference to security standards on the webpage (for example, that the product follow X.509 standard regarding digital certificates.
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database (PD)	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	

General News sites (GN)	News Source, Article title and date	ADVANCED – Set of Instances 1. Are they any announcement regarding the common criteria certification of a system that uses or integrated the software product.
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.11 Product Robustness – Maturity – Age

Definition: The time span over which a product has been developed.

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	
	Source Files	
	Test Files (input + scripts)	
	Documentation Files	
	Build Files	
	Historical Analyses specific to Product Distribution	BASIC – Single Instance 1. Age of the first stable distribution release (as compare to present time)
Version Control Repository (VCR)	Time stamp (VC metadata)	BASIC – Single Instance 1. Age of the oldest source file of the first stable release in the Version Control Repository (as compare to present time)
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	
Issue Tracking Database (ITD)	Request ID, Title, description	
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	ADVANCED 1. Age of the software product from its first closed source version (as compare to

		present time) (In case, the software product existed in closed source prior to its FOSS release)
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database (PD)	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites (GN)	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.11.1 Basic Metrics

Metric	Tools
Age of the first stable distribution release (as compare to present time)	None (visit website)
Age of the oldest source file of the first stable release in the Version Control Repository (as compare to present time)	(CVSAnaly)

13.11.2 Advanced Metrics

Age of the software product from its first closed source version (as compare to present time) (In case, the software product existed in closed source prior to its FOSS release)	None (visit website)

13.12 Product Robustness – Maturity – Continuity

Definition: The regularity and intensity with which the product or information related to the product was created or modified over the product's lifespan.

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	
	Source Files	
	Test Files (input + scripts)	
	Documentation Files	
	Build Files	
	Historical Analyses specific to Product Distribution	
Version Control Repository (VCR)	Time stamp (VC metadata)	BASIC – Set of Instances Tool: CVSAnaly 1. Number of Commits for all releases 2. Number of Commits for a specific release 3. Number of Commits for a specific set of releases (e.g. all minor releases under a specific major release)
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	BASIC – Set of Instances Tool: CVSAnaly 1. Number of lines of code committed for a specific release 2. Number of lines of code committed for all releases 3. Number of lines of code committed for a specific set of releases (e.g. all minor releases under a specific major release)
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	ADVANCED – Set of Instances Tool: CVSAnaly + statistical Analysis (for all 7 metrics) For the first 6 metrics below, the acceptable variation must be defined. 1. Historical variation of commits per month for all releases 2. Historical variation of commits for a specific release per month 3. Historical variation of commits for a specific set of releases (e.g. all minor releases under a specific major release) per month 4. Historical variation of lines of code committed for all releases per month 5. Historical variation of lines of code committed for a specific release per month 6. Historical variation of lines of code committed for a specific set of releases (e.g. all minor releases under a specific major release) per month 7. Prediction of lines of code growth as performed by SQO-OSS?
Issue Tracking Database (ITD)	Request ID, Title, description	
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	
	Historical Analyses specific to Issue Tracking	
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	BASIC (No Tools, simply visit the website) 1. Number of Major Releases 2. Number of stable releases (all, major and minor)
	Wiki Pages	
	Historical Analyses specific to Website Data	BASIC (No Tools, simply visit the website and collect releases names numbers and dates) 1. Number of Major Releases per year 2. Number of stable releases (all major and minor) per year

IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	
	Date entered	
	Authors	
	Severity	
	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	
Publication Database (PD)	Publication	BASIC – Set of Instances Tool: Search on Amazon.com 1. Number of books published about the software product Tool: Search on (http://liinwww.ira.uka.de/bibliography/) 2. Number of scientific article published related to the FOSS software product (not just the project)
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	ADVANCED – Set of Instances Tool: Search on Amazon.com + collect publication dates for each book edition 1. Number of book on the software product published per year
General News sites (GN)	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

13.12.1 Interesting Question

It may be interesting to find whether levels of Activity (in community) and Continuity (of product) have a relationship. Hence use one to predict the other. This could then be use to determine the health of a community based on code contribution, for example, if there enough community interaction for the size of committed code.

13.13 Product Robustness – Maturity – Activity on Stable development branch

Definition: The number and size of the contributions made to a product's stable development branch over a certain period of time. High activity on a branch declared to be stable can be a sign of low product maturity.

Data Source	Data Type (= artifacts or = contains artifacts)	Useful Analysis and tools
Product Distribution List (PDL)	Executable/Library Files	BASIC No tools (visit website) 1. Number of children releases under a selected release/version number
	Source Files	
	Test Files (input + scripts)	
	Documentation Files	
	Build Files	
	Historical Analyses specific to Product Distribution	BASIC No tools (visit website for links and dates) 1. Number of children releases under a selected release/version number per year
Version Control Repository (VCR)	Time stamp (VC metadata)	BASIC: Set of Instances Tool: CVSAnaly 1. Number of commits performed for all releases containing a given prefix in their

		tag (assuming that children releases share a common prefix with their parent release)
	Author (VC metadata)	
	Change Set (= file diff) (VC metadata)	
	Comment Log (VC metadata)	
	Check in/out programs (VC metadata)	
	Historical Analyses specific to Version Control Data	BASIC: Set of Instances Tool: CVSAnaly + Statistical analysis 1. Historical variation, on a monthly basis, of the number of commits performed for all releases containing a given prefix in their tag (assuming that children releases share a common prefix with their parent release). This curve should fit with the logarithmic function.
Issue Tracking Database (ITD)	Request ID, Title, description	BASIC: Set of Instances Tool: Advanced Search of Issue Tracking system 1. Number of issues reported for all children releases under a selected release 2. Number of issues whose resolution flag show an action took place vs. all issues reported for a single selected release 3. Number of issues whose resolution flag show an action took place vs. all issues reported for all children releases under a selected release. NOTE: A resolution flag that indicates is, for example, FIX where as WONTFIX or INVAL shows that no real action on the product took place in response to the issue report.
	Env. Spec related to Request (software product, version, hardware, OS, etc.)	
	Request Reporter's name and email	
	Request Status, priority, severity	
	Request Assignee	
	Target Milestones	
	Attachments such as test cases for bugs, scenario for enhancement.	
	Additional Participants' Comments	
	Historical Analyses specific to Issue Tracking	ADVANCED – Set of Instances Tool: Advanced Search of Issue Tracking system 1. Number of issues reported for all children releases under a selected release per year 2. Number of issues whose resolution flag show an action took place vs. all issues reported for a single selected release 3. Number of issues whose resolution flag show an action took place vs. all issues reported for all children releases under a selected release NOTE: A resolution flag that indicates is, for example, FIX where as WONTFIX or INVAL shows that no real action on the product took place in response to the issue report.
Discussion Archive (DA) (that is, Mailing List Archive, Discussion Forum accessible via a Web or News server)	Name, email of poster	
	Original Message ID, Date, Subject	
	Target email and list of recipients	
	Text Content and Attachments	
	Thread of answers (fields above for following up on original message: who, when, what, where)	
	Historical Analyses specific to Mailing List Data	
Website Pages (WP)	Static Pages	
	Wiki Pages	
	Historical Analyses specific to Website Data	
IRC logs (ICR)	Discussion topic	
	Participants	
	Static text content	
	Historical Analyses specific to IRC Data	
Security Databases (SD)	Vulnerability and Exposure ID, description	BASIC – Set of Instances Tool: NVD advanced search webpage 1. Number of vulnerabilities and exposures in NVD for all releases of a software product ADVANCED – Set of Instances Tool: NVD advanced search webpage 2. Number of vulnerabilities and exposures in NVD for a specific subset of releases of a software product (for example all, 1.x... releases)
	Date entered	
	Authors	
	Severity	

	Patch info (who, when, how large, how impactful)	
	Historical Analyses specific to Security Data	ADVANCED – Set of Instances Tool: NVD advanced search webpage + history analysis 1. Historical variation of number of vulnerabilities and exposures in NVD for a specific proper subset of releases of a software product. (This curve should fit with the logarithmic function)
Publication Database (PD)	Publication	
	Type of publication	
	Popularity of publication	
	Historical Analyses specific to Publication Data	
General News sites (GN)	News Source, Article title and date	
	Visibility (Distribution Size)	
	Authors	
	Author recognition/credibility	
	Historical Analyses specific to General News data	
F/OSS Research Databases (FRD) (that is FLOSSMETRICS, SQO-OSS and FLOSSMole)	Sample of data above plus eventual additional measurements already performed and saved	
	Historical Analyses specific to F/OSS data repositories	
Historical ANALYSIS on Aggregated Data from Various Data Sources		

14. Appendix C: Metric Collection Sheets

14.1 Product Complexity / Analyzability - URJC

Metrics	• Complexity of the product delivered • The cyclomatic complexity for entire sub tree (The metrics) (Resource standard metrics C, C++, Java and C#) • Detecting abusive # includes (DEPS) • Depth of inheritance tree (AOPMetrics JAVA) (CCCC) • Cyclomatic complexity (not in sub classes/functions) (The metrics) • Overal complexity (Resource standard metrics C, C++, Java and C#) • Package dependencies (AOPMetrics JAVA) (Java-MetricsAnalyzer?) • Lack of Cohesion in Methods, Chidamber-Kemerer (The Metrics) (DEelphi-code-analyzer) • Lack of Cohesion in Methods, Henderson-Sellers (The Metrics) • Lack of Cohesion in operations (AOPMetrics)
Rationale	If there are more complex programs, deficiencies will be more complicated to diagnose, so we need clear code and bad programming practices must be detected or prevented.

Metrics	• Stylistic verification (pylint) • Coding standard enforcements (pylint)
Rationale	If we need to change code, it will be easier if there is a clean code and structured (modularized,...) program. it needs to be guided by styling guides, standards and similar ideas.

Metrics	• Processing of source code vulnerability scanners (Audit-Perl)
	• Detecting potential security problems in C (C-Code Analyzer)
	  • array out of bound accesses
	  • potential bufferoverflow detection
	• Others
	  • Number of the encoutered problems (pylint)
	• Severity of the encountered problems (pylint)

Rationale	Errors vulnerability scanners.

Metrics	• Total files.(for multiple file metrics). • Total lines. (SLOCCount) (CodeAnalyzer?) (VB.Net, Perl and C++ - CodeMetrics?) (Pythius) (perl-metrics) • Code lines. (CodeAnalyzer?) (Java-MetricsAnalyzer?) (VB.Net, Perl and C++ - CodeMetrics?) • Comment lines. (CodeAnalyzer?) (Java-MetricsAnalyzer?) (VB.Net, Perl and C++ - CodeMetrics?) (Pythius) (perl-metrics) • Whitespace lines. (CodeAnalyzer?) (Java-MetricsAnalyzer?) (VB.Net, Perl and C++ - CodeMetrics?) (Pythius) (perl-metrics) • Functions (Pythius) • Classes (Java-MetricsAnalyzer?) • Cycles (Java-MetricsAnalyzer?) Dependencies to and from (Java-MetricsAnalyzer?)

	• Packages (Java-MetricsAnalyzer?) • Average line length. (CodeAnalyzer?) • Code/comments ratio. (CodeAnalyzer?) • Code/whitespace ratio. (CodeAnalyzer?) • Code/(comments + whitespace) ratio. (CodeAnalyzer?) • subroutine_lines (perl-metrics) • subroutines (perl-metrics) • comment_lines (perl-metrics) • pure_code (perl-metrics) • non-subroutine lines (perl-metrics) • code-to-comment ratio (perl-metrics) • avg lines per subroutine (perl-metrics) • longest subroutine (perl-metrics) • Other basic metrics • Other tools measuring the same or similar basic metrics: • PyMetrics? • The-Metrics • Resource standard metrics (for C, C++, Java and C#) (in a more detailed way) • Delphi code analyzer • Cstor
Rationale	Basic metrics in order to obtain general information from the code. Complex code is related to long functions, large classes, files with numerous lines of code, and other similar ideas.

Addressed quality attributes	Analyzability/Readability: The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	Different indentation styles (whitespace) affect the readability of source code. (Wikipedia)

14.2 Community Maturity - URJC

Metrics	• Number of changes made in the project. (CVSAnalY) (Wholine) • Number of files changed in the project. (CVSAnalY) (Wholine) • Number of developers making changes in the project. (CVSAnalY) • Number of non-active developers. (CVSAnalY) • How much effort software may suppose to maintain it in the future. (Carnarvon) • Daily colaboration. (bloof) • Number of lines added, removed or changed (counted as both added and removed). (CVSAnalY) (Wholine) • Number of commits. (CVSAnalY) • Number of bugs, differentiating by status. (There is no tool for this metric at the moment). • Number of not yet fixed or closed bugs.(There is no tool for this metric at the moment). • Mean time elapsed to fix or to close a bug. Standard deviation. (There is no tool for this metric at the moment). • Activity (number of messages in a mail list). (MailingListStats?) • Participation (number of people participating in a mail list). (MailingListStats?)

	• Number of people writing in the list over time. (MailingListStats?) • The list of keywords could be obtained in a monthly basis. (MailingListStats?) • With a list of keyword for each month, we can find out which topics were the most discussed and if the topics have evolved. (MailingListStats?)
Rationale	This idea is similar to third idea (continuity) but this is refered to community. With actual tools we can not obtain really good metrics, but we can measure basic metrics as activiy in the community (if the community has great activity it can mean that the community has some future).

Metric	• Regeneration of developers.
Rationale	In short in time projects, regeneration of developers is impossible to measure, however, in projects as Evolution of something like that, it is normal to have a difference in the core group during the life of a project. A project and its community is robuster if there is a good regeneration of developers (referring to core group)

Metric	• Betweeness
Rationale	In each project, during the life of the project, there are some specific people which have important connections among different members of the project. It is a social network metric and a vertex has a high betweeness if it has loads of neighbours, so it mens this edge is among big networks of people. It is based on CVSAnalY results and using a script it converts BBDD from CVSAnalY to Conan format. Conan is a tool which obtains different metrics from a network, as a betweeness.

Addressed quality attributes	Community Maturity: The capability of the project's community to support the project over a specified amount of time. This includes, for example, answering questions of users, delivering bug fixes, and evolving the product.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	

14.3 Standard Adherence - URJC

Metrics	• Stylistic verification (pylint) • Coding standard enforcements (pylint) • Patterns in contributions from developers
Rationale	Companies usually create new standards and new specifications in order to have easier ways to build software, thus tools are needed to check code and results must be related to standards, laws, protocols and others. For instance, in Java program language, there are some standards to make comments, style specification, etc.

Addressed quality attributes	Compliance / Standard adherence: The capability of the software product to adhere to relevant standards, conventions or law regulations and similar prescriptions, such as network protocols, standard file formats, or design or architecture conventions.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	http://en.wikipedia.org/wiki/Compliance_%28regulation%29 The link above is related to law environment, but there are some useful ideas.

14.4 Continuity - URJC

Metrics	Regeneration of developers
Rationale	In short in time projects, regeneration of developers is impossible to measure, however, in projects as Evolution of something like that, it is normal to have a difference in the core group during the life of a project. A project and its community is robuster if there is a good regeneration of developers (referring to core group)

Metrics	Activity in mailing lists, control version systems.
Rationale	It can be measured the activity during the life of a project. Concretely, if last years of a project it is detected a lack of messages in mailing lists or commits, it can mean that there is less activity in a project. Of course it does not mean the project is dying, but it can say the project is not going in a good way because in general there is less activity in tools related to it. It must be noticed that, for instance, there is less activity in a mailing list because there are new mailing lists or there is less activity in a cvs because the project is migrating to svn.

Addressed quality attributes	Continuity: The prospect of the project to continue to be supported and evolved in future by a dedicated community (or company, for proprietary products).
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	

14.5 Generic Metrics (Advanced issues) - URJC

Metrics	• Stylistic verification (pylint) • Coding standard enforcements (pylint)
Rationale	We likely need clear code, it must be focused to respect standards, modularity, unit tests and others.

Addressed quality attributes	Testability: (1) According to ISO 9126, the capability of the software product to enable modified software to be validated. (2) According to IEEE 610.12, the degree to which the module facilitates the establishment of test criteria and the performance of tests to determine whether those

	criteria have been met. For QualOSS, we need to consider both aspects
Metrics / Computation	
Link to extended information	http://en.wikipedia.org/wiki/Testable (Scientific ideas)

14.6 Interoperability - URJC

Metrics	• The cyclomatic complexity for entire sub tree (The metrics) (Resource standard metrics C, C++, Java and C#) • Detecting abusive # includes (DEPS) • Depth of inheritance tree (AOPMetrics JAVA) (CCCC) • Cyclomatic complexity (not in sub classes/functions) (The metrics) • Overal complexity (Resource standard metrics C, C++, Java and C#) • Package dependencies (AOPMetrics JAVA) (Java-MetricsAnalyzer?) • Lack of Cohesion in Methods, Chidamber-Kemerer (The Metrics) (DEelphi-code-analyzer) • Lack of Cohesion in Methods, Henderson-Sellers (The Metrics) • Lack of Cohesion in operations (AOPMetrics)
Rationale	Some systems support a high level of interoperability with others. It means that there are numerous interchange data operations between them. It is normal to find that kind of interactions in complex systems with loads of modules, thousand of lines of code and networks actions. A developer needs a clear API, clear documentation and other similar ideas in order to work faster, better and definitively in a good way to interact with other tools. Thus, we measure ideas like "clean code", low complexity, use of standards, not many includes between files.

Metrics	• Stylistic verification (pylint) • Coding standard enforcements (pylint)
Rationale	In order to work easier, developers need special things as standards.

Addressed quality attributes	Interoperability: The ability of two or more systems or components to exchange information and to use the information that has been exchanged. Related to compatibility.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	http://en.wikipedia.org/wiki/Interoperability "Interoperability can be achieved in four ways: through product engineering, industry/community partnership, access to technology and IP, and implementation of standards."

14.7 Maintainability – Changeability - URJC

Metrics	`Complexity of the product delivered` • The cyclomatic complexity for entire sub tree (The metrics) (Resource standard metrics C, C++, Java and C#) • Detecting abusive # includes (DEPS) • Depth of inheritance tree (AOPMetrics JAVA) (CCCC) • Cyclomatic complexity (not in sub classes/functions) (The

	metrics) • Overal complexity (Resource standard metrics C, C++, Java and C#) • Package dependencies (AOPMetrics JAVA) (Java-MetricsAnalyzer?) • Lack of Cohesion in Methods, Chidamber-Kemerer (The Metrics) (DEelphi-code-analyzer) • Lack of Cohesion in Methods, Henderson-Sellers (The Metrics) • Lack of Cohesion in operations (AOPMetrics) • Stylistic verification (pylint) • Coding standard enforcements (pylint)

Rationale	Again, we need to choose tools which measure complexity, guide styles and standards in order to say that a software is easy to maintain. For this purpose it is necessary clear documentation, clear code and it is important to have a good development environment, it will help you to make easier some common tasks, but we can not measure which development environment was used, thus, we, again, must obtain metrics from code.
Addressed quality attributes	Maintainability/Changeability: The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. IEEE 610.12: Maintainability is (1) The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. See also: extendability; flexibility. (2) The ease with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	http://en.wikipedia.org/wiki/Maintainability

14.8 Performance - URJC

Metrics	• memory leak detection (C-code-analyzer) • multiple/dangling free detection (C-code-analyzer)

Rationale	There are some problems related to performance which are really important in specific systems, as a real time systems, critical systems, embedded systems (memory leak detection is very important and free detection as well).
Addressed quality attributes	Performance: The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	http://en.wikipedia.org/wiki/Performance_testing http://en.wikipedia.org/wiki/Performance_analysis http://en.wikipedia.org/wiki/Performance_tuning http://en.wikipedia.org/wiki/Performance_Engineering

14.9 Performance- Resource behaviour - URJC

Metrics	No metrics
Rationale	

Addressed quality attributes	Performance/Resource behaviour: The capability of the software product to use appropriate amounts and types of resource when the software performs its function under stated conditions
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	

14.10 Performance – Time behaviour - URJC

Metrics	No metrics
Rationale	

Addressed quality attributes	Performance/Time behaviour: The capability of the software product to provide appropriate response and processing times and throughput rates when performances its function, under stated conditions
Metrics / Computation	
Link to extended information	

14.11 Project Maturity - URJC

Metrics	• Errors vulnerability scanners.
	• Processing of source code vulnerability scanners (Audit-Perl)
	• Detecting potential security problems in C (C-Code Analyzer)
	  • array out of bound accesses
	• potential bufferoverflow detection
	• Others
	  • Number of the encoutered problems (pylint)
	  • Severity of the encountered problems (pylint)
	• Others
	  • How old the software is (Carnarvon)
	  • How much it has been maintained (Carnarvon)
	  • File age (bloof)

Rationale	Maturity must be specified in maturity of code, or maturity of a community. Here it refers to maturity of code. Generally speaking a project will be good enough when it has had a long life with loads of proofs and it has been running in production machines. However, it, sometimes, is complicated to determine if a young project is mature enough or not. There is a list of metrics and using them we will be able to say if a project is mature or not, but we will need to compare with other products in order to have as a “scale of maturity”.

Addressed quality attributes	Product/Project maturity: The software product has been used for a long

	time by many users, and as a consequence, (most) faults have been removed. According to ISO 9126, the capability of the software product to avoid failure as a result of faults in the software. However, this definition is counter-intuitive.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	

14.12 Safety/Security - URJC

Metrics	• Errors vulnerability scanners. • Processing of source code vulnerability scanners (Audit-Perl) • Detecting potential security problems in C (C-Code Analyzer) • Array out of bound accesses o potential bufferoverflow detection o Others • Number of the encoutered problems (pylint) • Severity of the encountered problems (pylint)
Rationale	If there are not vulnerabilities, software is safer. In this quality attribute we are not considering problems related to social engineering or similar problems. Only problems in code.

Addressed quality attributes	Safety/Security: Security is the capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them. Safety (according to ISO 9126 a subcharacteristic of quality in use) is the capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	http://en.wikipedia.org/wiki/Computer_security http://en.wikipedia.org/wiki/Computer_security#Secure_Coding http://en.wikipedia.org/wiki/Vulnerability_%28computer_science%29 http://en.wikipedia.org/wiki/Buffer_overflows http://en.wikipedia.org/wiki/Format_string_vulnerabilities http://en.wikipedia.org/wiki/Code_injection http://en.wikipedia.org/wiki/Integer_overflow

14.13 Stability / Reliability - URJC

Metrics	• Errors vulnerability scanners. • Processing of source code vulnerability scanners (Audit-Perl) • Detecting potential security problems in C (C-Code Analyzer) • array out of bound accesses • potential bufferoverflow detection • Others

	• Number of the encoutered problems (pylint) • Severity of the encountered problems (pylint) • Coding standard enforcements (pylint)
Rationale	Again, we need clear code because developers must change code, and with a clear code it will be easier to change code and to obtain stability. Of course, tools related to error detection are very useful in this context.

Addressed quality attributes	Stability/Reliability: Stability is defined (ISO 9126) as the capability of the software product to avoid unexpected effects from modifications of the software (--> subconstruct of maintainability in ISO 9126). Reliability is defined (ISO 9126) as the capability of the software product to maintain a specified level of performance when used under specific conditions; this includes fault tolerance and recoverability.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	http://en.wikipedia.org/wiki/Fault-tolerant_system (Looks at "See Also" section)

14.14 Suitability - URJC

Metrics	no metrics found.
Rationale	We need metrics which were able to recognize requirements from user. There are some environments which provide this feature, as Eiffel language programming, however there are not metrics associated to that kind of characteristics. Probably we will have to look for tools related to Eiffel or those which provide Eiffel functionality.

Addressed quality attributes	Suitability: The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	

14.15 Testability - URJC

Metrics	• Stylistic verification (pylint) • Coding standard enforcements (pylint)
Rationale	We need probably clear code, it must be focused to respect standards, modularity, unit tests and other

Addressed quality attributes	Testability: (1) According to ISO 9126, the capability of the software product to enable modified software to be validated. (2) According to IEEE 610.12, the degree to which the module facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. For QualOSS, we need to consider both aspects
Metrics / Computation	Look at metrics for each related tool.
Link to extended information	

14.16 Business structure and productivity model (BSPM) - MERIT

Name	Business structure and productivity model (BSPM)
Brief description of model	There is no existing model for evaluating the quality of OSS projects based on economic indicators such as business structure and productivity. Therefore, we propose the development of a model based on a number of indicators such as the productivity of developers, the governance structure of the project, the nature and extent of involvement of formal organisational structures such as foundations, companies, universities and public sector organisations. Sustainability is indicated by these factors, as well as the project's success at skills improvement among developers to ensure continuity in leadership roles.
Addressed quality attributes	Examples of quality attributes we plan to address are as follows: *Productivity:* extent of code output, indicating improvements in software, as a function of input in terms of developer time and effort. *Sustainability:* likely sustainability of governance structures of project, indicating the degree to which a project can adapt to non-technical risks and changes, as well as set a roadmap for predictable growth. *Responsiveness:* timely responsiveness to user needs (such as rapid bug fixing, structural involvement of users in the software design process).
Constructs / Rationale	This approach outlined above seems to be the most promising and fruitful contribution by MERIT to Qualoss. It addresses issues not yet tackled by the literature. Below some references are listed, eventhough they are not directly on the issue, the FLOSSIMPACT report has a section on productivity that we can build further, and the other papers touches upon different community aspects, as well as firm invovlement. These are just some starting points.
Link to extended information	UNU-MERIT report (2006) on the Economic impact of F/OSS on innovation and competitiveness of the EU ICT sector O'Mahony, Siobhan & Fabrizio Ferraro (2003) Managing the Boundary of an ?Open? Project Mockus, Audris & Roy T. Fielding & James Herbsleb (2000) A Case Study of Open Source Software Development: The Apache Server Rossi, Cristina and Andrea Bonaccorsi (2005) Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach? Gregorio Robles and Jesús M. González-Barahona Contributor Turnover in Libre Software Projects Rishab Aiyer Ghosh and Paul A. David (2003) Analysis of authorship clusters in the Linux Kernel Developer community http://dxm.org/papers/licks1/licksresults.pdf

15. Appendix D: Glossary

Defect: (Definition used in QUALOSS)

- **IN QUALOSS, a defect is a product anomaly**
- IEEE 982-1988 - A product anomaly. Examples include such things as (1) omissions and imperfections found during early life cycle phases and (2) faults contained in software sufficiently mature for test or operation. See also fault.

Error:

- **IN QUALOSS, an error is the discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. (As defined by ISO)**
- IEEE 610.12 (IMPORTANT see following note for redefinition) - (1) The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. For example, a difference of 30 meters between a computed result and the correct result. (2) An incorrect step, process, or data definition. For example, an incorrect instruction in a computer program. (3) An incorrect result. For example, a computed result of 12 when the correct result is 10. (4) A human action that produces an incorrect result. For example, an incorrect action on the part of a programmer or operator. Note: While all four definitions are commonly used, one distinction assigns definition 1 to the word “error,” definition 2 to the word “fault,” definition 3 to the word “failure,” and definition 4 to the word “mistake.” See also: dynamic error; fatal error; indigenous error; semantic error; syntactic error; static error; transient error
- IEEE 982-1988 - Human action that results in software containing a fault. Examples include omission or misinterpretation of user requirements in a software specification, incorrect translation, or omission of a requirement in the design specification.
- ISO - A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. See: anomaly, bug, defect, exception, and fault

Fault:

- **IN QUALOSS: A fault is An incorrect step, process, or data definition in a computer program. In other word, a fault = a defect.**
- IEEE 610.12 - (1) A defect in a hardware device or component; for example, a short circuit or broken wire. (2) **An incorrect step, process, or data definition in a computer program.** Note: This definition is used primarily by the fault tolerance discipline. In common usage, the terms “error” and “bug” are used to express this meaning. See also: data-sensitive fault; program-sensitive fault; equivalent faults; fault masking; intermittent fault.
- *(Alternate Non-preferred definition for Failure) IEEE 982-1988 - (1) An accidental condition that causes a functional unit to fail to perform its required function). (2) A manifestation of an error in software. A fault, if encountered, may cause a failure. Synonymous with bug.*

Failure:

- IEEE 610.12 - The inability of a system or component to perform its required functions within specified performance requirements. Note: The fault tolerance discipline distinguishes between a human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error). See also: crash; exception; failure mode; failure rate; hard failure; incipient failure; random failure; soft failure.
- **IN QUALOSS: a failure is the inability of a system or component to perform its required functions within specified performance requirements. In many cases, a failure is due to the execution of a fault.**

- *(Alternate Non-preferred definition for Failure) IEEE 982-1988 - (1) The termination of the ability of a functional unit to perform its required function.* (definition 1 is similar to the one given in IEEE 729-1983) *(2) An event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered*