Qualoss

Sponsored through Framework Programme Sixth (Call 5) by

Document Information

Version: **1.0**
Date : Sep 15, 07
Pages : 18

Owning Partner:
CETIC

Author(s):
Jean-Christophe DEPREZ

Reviewer(s):
Frédéric Fleurial-Monfils (CETIC),
Martin Soto (IESE)

To:
European Commission

Purpose of distribution:
Final version ready for review

The QUALOSS Consortium consists of: CETIC (BE), Facultés 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)

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: **D1.4**

Title:

Reference F/OSS Project Report

(for WP1)

Deliverable: D1.4

Title: Reference F/OSS Project Report

Executive Summary:

This report explains the execution of task 1.4.

In particular, Section 2 first presents a set of criteria for selecting the F/OSS projects to measure during task 1.4. In the context of WP1, we find it important to test the limit of our prototype quality models and of the basic metrics, both described in D1.3. This desired is reflected in our selection criteria and in the F/OSS projects selected.

Section 3 describes the 5 F/OSS projects selected and Section 4 shows that these selected projects fulfil the selection criteria.

For WP1, it is more appropriate to collect the measurements in pre-formatted spreadsheet vs. a RDBMS. Spreadsheets gives the necessary structure while leaving the freedom to add easily additional information or comment. Nonetheless, we also want to use task 1.4 to elaborate a initial list of requirements for our future database schema (to be refined and implemented during WP2). In turn, Section 5 elicits all important requirements for the database schema that will store all our measurement results in later WPs.

Section 6 briefly describes the organization of the spreadsheet documents that contains our measurement results for the 5 F/OSS projects. Currently, one of the five projects is still being measured, namely, GNAT Pro Ada Compiler (front-end). Furthermore, the data and measurements results of the GNAT Pro Ada Compiler belong to AdaCore and we are still discussing privacy issues regarding these measurement results. In turn, they will be submitted separately as they become available.

Section 7 ends by enumerating all the lessons learned during task 1.4 and explains how we plan to address the different issues raised in the remaining WP's of QUALOSS.

CHANGE LOG

Ver.	Date	Author	Description
0.1	04/19/2007	Jean-Christophe DEPREZ	Initial Draft of D1.4
0.2	05/12/2007	Jean-Christophe DEPREZ	Initial proposition of F/OSS Projects
0.3	06/27/2007	Jean-Christophe DEPREZ	Final selection criteria, list of F/OSS projects changed + verification that it satifies our criteria.
0.4	09/11/2007	Jean-Christophe DEPREZ	Add Sections 5, 6, and 7
0.5	09/13/2007	Frédéric Fleurial Monfils	Review of D1.4
0.6	09/14/2007	Martin Soto	Review of D1.4
0.7	09/14/2007	Jean-Christophe DEPREZ	Corrections from reviews
0.8	09/14/2007	Jean-Christophe DEPREZ	Inserted the Executive summary and produce the final version of the deliverable
1.0	09/15/2007	Jean-Christophe DEPREZ	Sanity Check

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Deliverable Identification
	Deliverable D1.3 - Metrics System and Prototype QualOSS Models (Measurements taken during Task 1.4 are those defined as basic metric in D1.3)	D1.3

TABLE OF CONTENTS

1. Introduction 5
1.1 Objectives 5

2. Task Constraints and F/OSS Project Selection Criteria 6
2.1 Measurement Constraints 6
2.2 F/OSS Project Selection Criteria 6

3. Reference F/OSS Projects for WP1 9
3.1 Plone 9
3.2 GNAT Pro Ada Compiler 9
3.3 Hildon Application Framework of Maemo (HAF) 9
3.4 JavaCC 9
3.5 Swallow (DBE Servent) 9

4. Satisfaction of Constraints & Selection Criteria 10

5. Requirements of QUALOSS Database Schema 12

6. Perform measurements on selected F/OSS projects 13

7. Lessons Learned 14
7.1 Synchronization Issues 14
7.1.1 Synchronization on the scope of a whole F/OSS project 14
7.1.2 Synchronization on the scope of releases of a F/OSS project 15
7.2 Data and Data-Source Issues 15
7.3 Tool Issues 16
7.4 Metric Issues 16
7.5 Missed opportunities 17

1. Introduction

WP1 consists in the prototyping phase of QUALOSS. Besides proposing an initial solution for our measurements, its purpose is to highlight problems. It is therefore important to conduct task 1.4 with the intention to identify such problems. In other words, at this stage, we find it more important to identify problems than merely to select F/OSS projects for which we know our approach, tools and measurements will work. At the end of task 1.4, the ultimate outcome would be to identify all problems likely to arise during WP2, 3, and 4 early in the life cycle of these WP's so we have ample time to solve them.

In addition to identify potential problems, task 1.4 also serves to get a first round of measurements results for basic metrics. These results are presented in the accompanying spreadsheet documents (one per F/OSS project).

1.1 Objectives

The two objectives of task 1.4 is first to build a repository of measurements for a few selected F/OSS projects and second, to identify measurement problems so WP2-4 can work on solving them.

In order to reach these objectives, we perform the activities below.

1. Specify task 1.4 constraints and selection criteria for F/OSS projects (in the context of WP1);
2. Select reference F/OSS projects for WP1;
3. Verify that F/OSS project selected satisfy the constraints enumerated;
4. Specify schema requirement for our future measurement database;
5. Perform measurements on selected F/OSS projects and store data in pre-formatted structured spreadsheets;
6. Highlight the lessons learned during our measurement activities including aspect related to quality of data, which specifically refers to Milestone 1.2 – Quality of Data from Reference F/OSS Projects)

2. Task Constraints and FLOSS Project Selection Criteria

The first activity consists in identifying the constraints imposed on this task and then verify that they are satisfied by the F/OSS projects selected.

Currently, we have identified two categories of constraints, one on measurements and the others on F/OSS projects. The first category of constraints specifies the type of measurements that must be applied while the second category imposes restriction on the F/OSS projects to select for measurements. Each constraint is assigned a unique identification to simplify referencing later in Section 4 when verifying that each constraint is satisfied. Constrains related to measurement (the first category) start with M while those related to F/OSS projects selection start with PS; except the first two metrics of this second category, which are some what different in scope so we have simply named using R (in reference to the generic term Required).

2.1 Measurement Constraints

In order to identify problems related to measurements, the following constraints must be met before starting task 1.4 :

- M1: Basic metrics taken on selected projects must cover the level of automation of measurements: fully automated, partially automated, manual.
  Rationale: During WP1, we want to verify if the level of automation of measurements raises problems. For example, We anticipate that manual measurement risk to be applied differently if taken by different people or even by the same person on different F/OSS projects. In turn, we want to make sure that different level of automation of measurements are tested during WP1.

- M2: Basic metrics must be performed on data sources outside the control of the selected F/OSS projects, for example, NVD, amazon, or slashdot.
  Rationale: Sources outside F/OSS projects control can bring interesting information for assessing projects. However, the impact of that data is unknown in turn, we want to have a first assessment during WP1.

2.2 FLOSS Project Selection Criteria

- R0: For WP1, at most 5 F/OSS project should be selected for measurement and assessment
  Rationale: Based on the Person-Month effort allocated to Task 1.4, the number of basic metrics inventoried, and assuming an average of 2 to 3 minutes effort per metric per project, we may at most perform measurements for 5 projects, actually 4 would also be acceptable.

- R1: Our assessment methodology should produce results for every basic metrics in the case of at least one of the selected F/OSS projects.
  Rationale: Although WP1 is the prototyping phase and is used to highlight problems, we also want WP1 to produce a few interesting results so as to raise the interest of people external to the QUALOSS consortium.

- PS1: Cover most usage scenarios in our two dimensional grid (see: platform vs end-user application and embedded vs desktop vs service vs development (See Figure 1). When selecting projects, the distinction between internal and external service does not make sense as that can only be distinguished while the service is in real operation. In turn, only eight usage scenarios are left: Platform/Embedded, Platform/Service, Platform/Desktop, Platform/Development, End-Application/Embedded, End-Application/Service, End-Application Desktop and End-Application/Development. It is unlikely that all eight scenarios can be covered with the 4 or 5 FlOSS projects selected in WP1 in turn, we only aim at covering each topic: end-application, platform, embedded, service, desktop, and development.
  Rationale: Different kinds of software are likely to be assessed slightly differently. It is the job of the prototype to find if such variations exists.

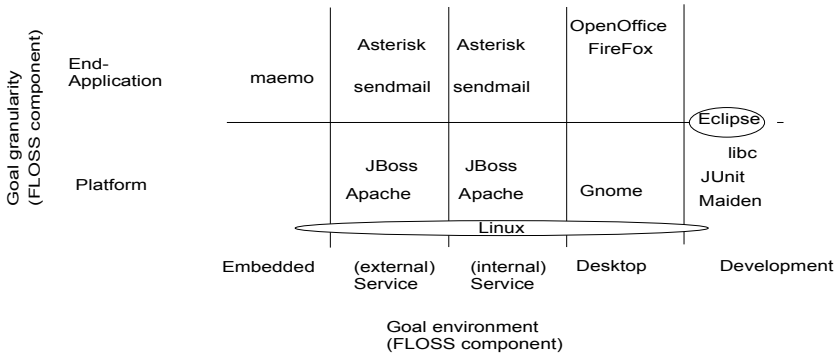

Figure 1: Usage scenarii identified in Deliverable D1.2

- PS2: One of the selected project should probably cover more than one usage scenario (for example, an end-user application with desktop and server parts such as Azureus)

  Rationale: Projects that overlap over several usage-scenario boundaries may highlight special cases for our assessment methodology. For example, Azureus implements peer-to-peer libraries which may be considered desktop/platform as well as server/platform and it also implements the end-user application part of the application, which can be considered desktop/application. So this raises the question how do we apply assessment methodology on Azureus, must we partition the project into different parts or can we assess it as a whole?

- PS3: Cover all programming languages addressed by QUALOSS:

  Ada, C, C++, Java & Python

  Rationale: QUALOSS targets the languages above. Although we have tested the code analysis tools selected, we want to test them even more thoroughly during WP1. This will highlight which tools must be improved so that we can treat each languages in a fairly equal manner.

- PS4: One F/OSS project should be implemented in several programming languages, for example, C++ and Java or C and Python.

  Rationale: Usually code analysis tools handle a single programming in turn, we already want to consider an application implemented using multiple programming language. This will help us address the question, do we treat each languages separately or can we develop unified model for several languages.

- PS5: One F/OSS project should be implemented in several programming languages, one of which is not supported by QUALOSS (that is a language other than Ada, C/C++, Java and Python)

  Rationale: Some F/OSS projects will be partly implemented in a language targeted by QUALOSS and partly in a language that was not to be treated during QUALOSS. It is therefore important to check the boundaries of the applicability of our assessment methodology.

- PS6: One selected F/OSS projects should be fairly young while exhibiting a certain level of professionalism by tracking and recording data in various sources such as version control, bug tracker, mailing list, website.

  Rationale: Younger projects with potential may not have all the infrastructure that mature projects, part of large foundation, have in place. However, QUALOSS's goal is not limited in assessing already successful projects. In fact the added value of a method such as QUALOSS will be in its ability to assess the potential of smaller less mature projects.

- PS7: One selected F/OSS project should be on the smaller side (small product, small community, not part of a large foundation)

  Rationale: Although a single F/OSS project may satisfy PS8 and PS9, we keep these criteria separate since it could be possible for an old project to have remained at a fairly small size (product and community). In turn, the two criteria PS8 and PS9 may also be satified by two distinct F/OSS projects. In fact, it would be interesting to test our assessment methodologies on a small project with a proven track record.

- PS8: the set of selected projects should have used various issue tracking systems, in particular, it would be interesting to have esoteric issue system not necessarily used by many F/OSS projects.
  Rationale: Currently, QUALOSS relies on other E.C. projects such as FLOSSMETRICS to deal with a programmatic access to issue tracking systems of projects. However, FLOSSMETRICS is likely to get an agreement of only a few F/OSS projects hence, during WP1, we want to assess how hard it will be to extract issue data manually for different systems encountered.

- PS9: Most project must have enough data for taking measurements of several basic metrics associated to each characteristic of our hierarchies.
  Rationale: A scientific methodology may only judge F/OSS projects based on the information that they provide. This becomes obvious when considering our mapping between metrics and characteristics. If all metrics related to a characteristic cannot be measured for a given project then we cannot assess the characteristic.

3. Reference FLOSS Projects for WP1

This section proposes a list of F/OSS projects to be measured in a next activity of task 1.4. Furthermore, it also verifies that all constraints enumerated in section 2 of this document are satisfied.

We also want to emphasis that two QUALOSS partners, namely AdaCore and ZEA Partners are closely affiliated to the first two projects. AdaCore is the workhorse behind GNAT Pro, while ZEA Partners is an association whose members develop and sell services around Plone.

3.1 Plone

Plone is a content management framework built on the top of the Zope application server. It was created in 2001. It is written in Python. Plone has gained a great popularity since then having developed a large community; user and developers included. Plone is a mature open source project, which has been recognized by winning awards at different occasions. Plone is managed by the Plone Foundation. Many small businesses proposing services based on Plone have been created.

3.2 GNAT Pro Ada Compiler

The GNAT Ada compiler includes the Ada 83/Ada 95/Ada 2005 front-end, the GCC code generator, the binder, linker, and run-time library. All of these components, except for the code generator, are written in Ada, and are completely target-independent. the GNAT Pro Tool suite is managed by AdaCore, an international SME with offices in Paris and New York. Founded in 1994, AdaCore has provided most of the development of the Ada Compiler, which is by now a very mature free software endeavour.

3.3 Hildon Application Framework of Maemo (HAF)

The Hildon Application Framework is a project of the Maemo endeavour. Maemo aims at providing a complete environment for the Nokia 770 Tablet. Measuring the complete Maemo endeavour would have been to heavy for all product characteristics so we decided to focus our measurement effort on the Hildon Application Framework (HAF). HAF implements the Gnome Gtk widgets for the Nokia 770 Tablet. HAF is currently in the process of moving to under the Gnome project so has to benefit from its whole infrastructure and support tools such as live.gnome.org.

3.4 JavaCC

Java Compiler Compiler (JavaCC) is the most popular parser generator for use with Java applications. A parser generator is a tool that reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. In addition to the parser generator itself, JavaCC provides other standard capabilities related to parser generation such as tree building (via a tool called JJTree included with JavaCC), actions, debugging, etc. JavaCC was initially developed as a proprietary tool by Metamata, which later was acquired by WebGain. WebGain stopped its operation in 2002 and since then the source code of JavaCC have been hosted by Sun on their java.net site, where Java open-source projects are available.

3.5 Swallow (DBE Servent)

Swallow is the new name of DBE Servent. DBE stands for Digital Business Ecosystem, an FP6 project sponsored by the European Commission. It ran from November 2003 to January 2007. DBE is composed of three main components: (1) Swallow, a server platform for hosting decentralized software services, (2) DBE Studio, a development environment for creating and publishing services on Swallow, and (3) a evolutionary components for service composition based on genetic algorithms. In the study performed during WP1, we only focused on Swallow, which is hosted on SourceForge.

4. Satisfaction of Constraints & Selection Criteria

This section verifies that all constraints enumerated in Section 2 are satisfied by our measurements and by our project selection.

R0: Only 5 projects have been selected so this constraint is satisfied
R1: GNAT Pro Ada Compiler and Plone are two mature F/OSS projects for which we have access to a lot of data hence we expect that they will provide interesting results for assessing the basic QUALOSS quality models and indicators proposed in WP1.

Based on D1.3, we find that the next two constraints M1 and M2 are satisfied:

M1: Some basic metrics such as SourceCodeCommentPercentage can be compute automatically once the actual source files to measure have been provided; others are partially automated, for example, VulSubsetReleases where the advanced search facility provided by NVD partially helps; and yet some are fully manual such as APIDocumentationExistence

M2: A metric such as VulSubsetReleases is to be computed on Security Databases such as NVD, which is not owned by FlOSS project

PS1: All alternatives of usage scenarios and coverage of selected projects

	Pltf/Emb	Pltf/Serv	Pltf/Dktp	Pltf/Dev	EApp/Emb	EApp/Serv	EApp/Dktp	EApp/Dev
Plone		X				X		
GNATPro Compiler				X				
Maemo-HAF	X							
JavaCC				X				
DBE swallow		X						

It is virtually impossible to covered all 8 usage scenarios with only 5 projects. However, if we look at the two dimensions in a linear fashion, we then find that all but the Desktop scenario are covered. The two dimension are (Dim1 = {end user app, platform} and Dim2 = {embedded, service, desktop, development})

	End Application	Platform	Embedded	Service	Desktop	Development
Plone	X	X		X		
GNATPro Compiler	X					X
Maemo		X	X			
JavaCC		X				X
DBE swallow		X		X		

We find that in the context of WP1, this coverage is sufficiently diverse. Moreover, in general, quality models are often created with lambda end-user in mind so it is more crucial to find project that cover usage scenarios other than desktop.

PS2: Plone covers two usage scenarios: End-Application/Service (when used by a user lambda) and Platform/Service (when a Plone programmer must tailor the workflow to match that of the client company).

PS3: Ada for GNAT Pro, C/C++ for Maemo, Java for JavaCC and swallow (DBE Servent)

PS4: Maemo is implemented in C/C++ and Python

PS5: Plone is implemented in Python but also heavily relies on JavaScript. QUALOSS has no plan to support JavaScript, during the lifetime of the project at least.

PS6: Swallow (DBE Servent) was registered in May 2005 on SourceForge. This is young enough to verify if our assessment methodology is also adapted for project with a relatively young age and it is old enough so data about the project have been generated (mainly by the committers)

PS7: JavaCC was started by Sun as Jack 12 year old, which is considered very old for F/OSS project. Moreover, concerning PS7, JavaCC is on the smaller size: between 30K and 40K lines of code, the community is made of 9 committers from which 3 are contributing most changes.

PS8: Plone uses track, GNAT Pro uses it own proprietary system (might become F/OSS in the future), Maemo uses bugzilla but requires a registration, JavaCC uses the issues tracking system used by the java.net forge, Swallow (DBE Servent) uses the on from SourceForge.

PS9: After a quick verification, all selected projects have abundant data of different type and located in various data sources, at least those controlled by the F/OSS projects themselves.

5. Requirements of QUALOSS Database Schema

We realize that forcing a database in the context of WP1 would impose a heavy burden to the various partners. Furthermore, database enforces a rigorous structure to data and in the context of our prototyping WP1, we believe that it is important to allow every partner to perform measurements freely and eventually include measurement result in unexpected format, for example, graphs or additional comments. In turn, during WP1, we decide to record all measurements in spreadsheets. The next section explains where our measurement results are found. We are also aware that allowing every partner to perform measurements freely without constraints is likely to cause a bit of synchronization problems and indeed, such issues happened and we discuss them in more details in section 7.1.

Although a RDBMS is not used to record our results, we still want to use WP1 to identify important requirements for the database schema to implement later on in WP2. Eliciting DB schema requirements in WP1 may help identify potential problems early on and leave us more time to solve them in the remaining part of the QUALOSS project.

We emphasize that the database envisaged here will only contain measurement data. In other words, this is not a database of raw data on which measurements will be performed but the database where measurement results will be stored. We leave it to measurement tools to define their own schema requirements.

Req1. Every measurement must be keep track of the time when it was taken.

Req2. The time when every measurement was entered in the database must be recorded.

Req3. Every measurement must be associated to the artefact on which it was taken.

Req4. Every measurement must be associated to a metric. (where a metric = a measurement unit)

Req5. Every metric must be associated to an artefact type.

Req6. The type of an artefact on which measurement may be taken must be known and must match with the type associated to metric. For example, an artefact to measure has the type *source-code* file and the metric *cyclomatic complexity* is also associated to the type source-code file hence the measurement of the artefact for its cyclomatic complexity can take place. On the other hand, the metric *number of publications*, which is associated to the type *publication list*, cannot take place on the given artefact of type source-code file.

Req7. Every artefact belong to a single F/OSS project and when possible, to a single releases of a F/OSS project.

Req8. The universal location of every artefact must be recorded in our database. This information can eventually be decomposed in several fields if needed. The important aspect of this requirement is to have the capacity to recover every artefact measured from its original storage.

Req9. In order to enforce our constraints on data, we plan on centralizing the DB storage.

Req10. Record the measurement tool and its version used for computing a measurement

Req11. Record the person who performed the measurement

Req12. Record the description of the measurement environment and how the measurement tool were used, for example, the command line used and the content of configuration files for the measurement tools.

6. Perform measurements on selected FLOSS projects

To remain synchronized with the DoW, we proposed an initial DB schema. However, no application was implemented to insert data in the DB hence forcing people to deal with SQL. In turn, we preferred the solution of recording measurement results in spreadsheets.

Each selected project has its measurements recorded in its own spreadsheet documents in the open document format (.ods).

- WP1_Task1_4_HAFMaemoMeasurements.ods
- WP1_Task1_4_JavaCCMeasurements.ods
- WP1_Task1_4_PloneMeasurements.ods
- WP1_Task1_4_SwallowDBEMeasurements.ods

Currently, we are still working on measurements of the GNAT Pro Compiler. The resulting spreadsheet WP1_Task1_4_GNATProMeasurements.ods will be submitted as it becomes available in late September.

Furthermore, the QUALOSS consortium is still solving privacy issue with AdaCore's measurements in turn, they cannot submitted at the moment. In the future, we still expect that these measurement remain strictly confidential with in the consortium and the European Commission.

All WP1_Task1_4_xxxMeasurements.ods are sumbitted in an accompanying compressed archive (.tar.gz)

In every measurements files, a cover sheet summaries the content of all the other sheets. In short, measurements of the spreadsheets are organized following our four quality models on Product Evolvability (1 sheet), Community Evolvability (3 sheets), Product Robustness (between 1-3 sheets), and Community Robustness (1 sheet).

7. Lessons Learned

Each WP1_Task1_4_xxxMeasurements.ods enumerates problems encountered during the measurement exercises. These problems relay either an ambiguity in the definition of a particular metric or weaknesses of tools used to perform measurement. For each of these specific problems we refer the reader to the spreadsheet of each project measured. In this section, we want to raise the discourse to a high level of abstraction and summarize the issues encountered so as to clearly highlight the general problems to solve in the remaining time of the QUALOSS projects. Nonetheless to explain the issues encountered, we refer to specific a example that happened in our measurements.

7.1 Synchronization Issues

We have clearly identified two kind of synchronizations that must occurs when measuring a F/OSS project:

1. Synchronization on the scope of a F/OSS project to measure
2. Synchronization on the releases of a F/OSS project to measure

Section 7.1.1 explains the first point above while Section 7.1.2 addresses the second point.

Prior to discussing these two synchronization issues, we point out that the only way to solve such synchronization issues is to be aware of them and to keep a regular contact between the various people performing measurements. More importantly, everyone taking measurements must be aware that she or he must always record and keep track of the data measured so that if there are discrepancies between data scope, we notice it. Furthermore, a large part of synchronisation problems will be solved when using a RDBMS with well implemented constraints on data and data association.

7.1.1 Synchronization on the scope of a whole F/OSS project

Point 1 was encountered when measuring Hildon Application Framework (HAF), a project under the umbrella of the Maemo endeavour. The main problem of synchronizing the scope of data only related to HAF is due to the structure of the different data sources, which did not scope their data in a similar fashion. Below, we explain how data was scoped for each type of data source.

The issue tracking system used by Maemo, namely, Bugzilla presented HAF in its list of project. Hence, we initially planned to limit our dataset of issues to that with the HAF project in Bugzilla. However, in mid-August HAF was removed from the list of project in Bugzilla. In turn, we changed the measurement strategy. We noticed that some hildon components previously categorized under HAF were now under the SDK project. So, we decided to scope the data to all issues in SDK filtered on the terms hildon or HAF.

Concerning mailing lists, we had faced two problems. First, Maemo does not create a mailing list for each projects but it only has four mailing lists for all its projects, one for users, one for developers, one for announcement and one for commits. So the mailing list data have a scope broader than that of HAF. However, the current tool used to analyse mailing list did not allow filtering mail message to be filtered based on content. Due to this limitation, mailing list data were scope to the entire Maemo project and not just HAF.

The version control repository contains data for several Maemo projects and not just HAF. Although the repository contains a HAF directory (or module), it includes more that just HAF related code, for example, libcairo. The person responsible for measuring code therefore decided to fix the scope to the HAF code implementing the HAF widgets (at URL https://stage.maemo.org/svn/maemo/projects/haf/tags/hildon-libs/0.15.1-1/). However, the other person responsible for community measurements scoped data in the version control repository to the complete Maemo project. Hence, there are discrepancies between these measurements.

The synchronization problem highlighted above is present for HAF because it is a project that takes place under another wider F/OSS project, namely, Maemo. In turn, for the future, we must be careful when dealing with a F/OSS endeavour made of several sub components. Although we could limit our assessment methodology to apply only to complete F/OSS project, this seems unacceptable for a stakeholder only interested in evaluating a subcomponent of a F/OSS project and not the full project.

7.1.2 Synchronization on the scope of releases of a F/OSS project

The synchronisation problem presented in 2 points above is related to our quality model on product robustness (see D1.3 for the definition of our quality models). In particular, some metrics used to assess product-robustness quality charateristics requires to scope data on a selected subset of releases. These metrics let the person performing the measurement decide on the scoping of data that seems most appropriate to the circumstances. It is therefore important to fix the subset prior to starting the measurement exercise. This issue was well identified prior to taking the measurement in Task 1.4 so we only encountered two small problems:

(1) when considering a scope that includes all releases, we noticed that some of the people performing the measurements also included data related to the latest development not yet released, i.e., information available in the data sources but not yet tagged to a particular release.

(2) Certain metrics consider only stable releases. However, certain projects such as Swallow DBE, have not releases a stable version of their product yet. In that case, we decided to take measurements on unstable releases rather than not being able to perform measurements.

7.2 Data and Data-Source Issues

The two main issues regarding data are:

1. Data is not available in the expected source but in another one
2. Data quality varies tremendously among F/OSS projects (related to Milestone 1.2 – Quality of Data from Reference F/OSS Projects)

In some cases, we found that data for taking a measurement is available in another data source than that identified in the metric definition. For example, the issue tracker of JavaCC only contains issues for 4.0 but not for 3.2. This is due to the fact that JavaCC was initially released under a proprietary software license and its release 3.2 was then subsequently open sourced. At that time, the JavaCC issues were not managed by an issue tracking system however, a newsgroup existed were people reported and discussed errors encountered. In turn for version 4.0 and up, issue information can be found in an issue tracking system while for version 3.2, bug information can only be found in a newsgroup. Initially, we assumed that all data to be measured would reside in the same repository. However, from the above example, we observe that it is not always the case.

The proposed solution is to define a set of data-source specific metrics and then a conceptual metric independent of data source, which is computed from the data-source specific metrics. So in our example, above, the number of issues for all releases of JavaCC would include issues found in the issue tracking system and in the newgroups. Defining a conceptual metric in term of data-source specific metrics forces a systematic consideration of all data sources with regard to a specific king of data. For example, we must therefore wonder whether each data source could include information about issues. Furthermore, it also forces us to think about a composition strategy when aggregating the results from several data-source specific metrics. Indeed, in our example above, an issue in 4.0 could be accessible from several data sources, in particular, the issue tracking system and the newsgroup. In turn, we need to verify whether this situation is anecdotal or not. In any case, the definition of such a composition strategy will need to be addressed on a metric-per-metric basis and this decision is delayed to later WP's.

Regarding the second point on data quality, we found during our measurement exercise that F/OSS projects have a large variation in their data quality, in particular, GNAT Pro and Plone have good data quality while Maemo and JavaCC fall in the medium of the spectrum, and Swallow only has sparse and incomplete data available. For example, Swallow does not have bugs related to crash, availability, etc. This is likely due to the fact that no stable release has come out yet and also that the tool has just come out of research and is not currently used broadly. However when assessing Swallow, it would likely be incorrect to assume that no crash or availability issues exist in the software component.

The solution already under way consists in performing an assessment of data quality. The results of that assessment can then be used to join a confidence factor next to every measurement. This confidence factor could later be used compute the error factor in our overall assessment methodology. For example, the lack of data regarding crash and availability issues for Swallow would lead us to assign a low confidence to our resulting measurement. If a project contains only a few low confidence measurements then the assessment will be more accurate while too many low confidence measurements would make our assessment much less precise.

7.3 Tool Issues

We identified the following issues regarding tools:

- Issue/bug tracking databases are not directly accessible i.e, they can only be accessed through their common graphical user interface. This restriction prevents us from elaborating more advanced analysis on issues, which could be performed if database access were granted.
- Mailing lists analysis tools: currently the tools used does not allow filtering of data based on keywords
- Version control analysis tools: currently the tools can only compute the number of lines added and deleted on per-commit basis when dealing with CVS repositories but not with Subversion repositories.
- Code analysis tools: The main issue regarding code measurement happened with C/C++ code analysis. In particular, Sissy, the tool currently selected (in D1.1) does not compute metrics but rather provides a database with information from which it is possible to compute some metrics. However, writing the needed queries for computing basic metrics was not simple. In turn, for WP1, the analysis for C and C++ code were performed with Source Monitor. Source Monitor is not open source hence some approval by the QUALOSS consortium shall be needed to use it for our later measurement efforts. This kind of decision

Except for the first issue in the list above, the resolution are to improve the tools currently used with the functionalities needed. These new requirements are fairly simple and shall not introduce excessive overhead for WP2. With regards to the last point, the decision to use tools not open-sourced for our measurements needs to be discussed with the consortium.

Concerning the first point on issue database access, we believe that FLOSSMetrics may help us gain access to certain issue database. However, this will be for a fairly small amount of projects and QUALOSS does not want to have to take its decision regarding F/OSS project selection based on whether or not issue tracking database are directly accessible or not. So, rather than trying to solve this issue, it is view as a constraint that we must live with. This could however influence the metrics taken on issues or the precision of measurements taken on issue. We do however want to help those who give access to their issue databases. In turn, once we have a clear list of projects that are giving access, we will identify the interesting metrics for those particular cases. This effort will have to be coordinated with FLOSSMetrics.

It is worth mentioning that AdaCore is part of the QUALOSS consortium hence, we have a direct access to their data (although due to privacy issues these results may have to remain confidential). We also hope that through ZEA partners we may gain access more directly to Plone data. Moreover, we also want to emphasize that other F/OSS projects will have a tendency to give us access to their data if they clearly see an advantage to it. In turn, we have already started to establish contacts with F/OSS communities during FOSDEM in February 2007 in order to explain the purpose of QUALOSS. We will continue interacting with the F/OSS community. This will be an important step for selecting the long list of 50 F/OSS projects to analyse as part of WP3. A second action undertaken to obtain data access will be gain FlOSS communities trust by letting them give feedback on our quality models and then, take their concerns into account when processing their project data.

7.4 Metric Issues

During the measurement efforts of Task 1.4, we encountered the following issues with regards to metrics

- Imprecise or ambiguous definition of metrics.
- Very basic metrics for assessing our quality model on Product Evolvability should be added.
- Currently, the metric related to publication DB are to restrictive in their data source selection.
- Currently, some data grid related to Community Robustness characteristics are not yet metrics
- Many measurements involved more manual actions than expected.

Each of the point above is discussed in the paragraphs below.

Some metric definitions are ambiguously or imprecisely stated. We identify two kind of imprecision. The first kind is voluntary. In other words, our assessment methodology cannot specify a precise definition of a term in the risk of being to restrictive. There is a second kind of imprecision where we must simply improve the current metric definition and give precise definition for the ambiguous words use used for defining a metric.

For example, the term release and stable release are left imprecise and what constitute a release and a stable release should be defined on a per-F/OSS project basis. This imprecision is due to the lack of standards in the

naming scheme of releases. At best, we may expect that a project respects its own conventions however such internal rules were not stated clearly for any of the 5 F/OSS projects measured.

Discussions on definitions are starting to take place on the QUALOSS Trac website (http://qualoss-partners.libresoft.es/); this is the site where discussions and project management activities are taking place. The discussions address how to define certain terms and also what terms can stay ambiguous in the general case and only be given precise meanings in the context of assessing a selected F/OSS project.

Concerning the second point, we notice that the elementary metrics such as number of line of code (LOC), number of files, number of classes, number of functions/methods, etc. were not required. We believe that they should definitely be present in our later measurement effort in the remaining WP.

Regarding the point on publication databases, we observe that metrics on publications ask the person who performs the measurement to only search Amazon and http://liinwww.ira.uka.de/bibliography/ when taking measurements. This could seem restrictive and we may need to consider additional sites. In turn, instead of limiting the publication databases to search to a fix list of repositories, it may be more judicious to specify certain constraints on the publications in order to determine whether they can be counted or not. For example, book shall have an ISBN, reseach publications must appear in a scientific journal or a conference proceeding, etc.

Our next point notes that some measurement data grids proposed as metrics for measuring community evolvability characteristics are not true metrics; they are still in the form of data grids and graphs. We will address this issue during task 1.5 where these data grids and graphs will be used as basis to proposed true metrics.

The last point simply points out that the requirement Required_0 in Section 2.2, i.e., measurement should only take an average of 2-3 minutes, was hard to fulfil for many metrics because many tools only delivered part of the needed answer and a person still had to spend a few minutes to aggregate, sort or count these results to compute the final desired measurements. In the remaining part of the project, we must either update our tools so they directly compute the desired results or we must increase the expected time of measurements per metrics from 2-3 to 5-6 minutes. Furthermore, WP4 will also introduce advanced metrics. They will automatically take longer to compute than our basic metrics. Instead of estimating the time of measurement solely based on intuition, as it was the case for WP1, we will take a sample measurement for every advanced metric proposed. In turn, we expect to have a much better time estimate for all our advanced metrics. For our basic metrics, we will make sure that the final results are directly produced by our tools.

Incidentally, the next wave of measurements is planned for WP3-Task3.2. The current effort allocated is only 9 person-months. This now seems underestimated; especially for measuring ten times as many projects as in WP1. Although time will be taken into consideration when selecting the final set of metrics to measure, we still expect to need more time than the anticipated effort. In turn to reduce the associated risks, we will start with Task 3.2 as soon as possible, in particular, by November 2007. This will leave us more than a year to perform all measurements. Also, if needed, some shift in effort could also take place from WP4 to WP3. We also expect that the measurement tools integrated in the QUALOSS platform will deliver most results in their final form and only advanced metrics with high added value will require a manual measurement effort.

7.5 Missed opportunities

Beside the issues mentioned in the above section, we also want to highlight two missed opportunities.

- No metrics related to Community Robustness could be taken
- No metrics were finalized on time for evaluating user documentation (actuality and adequacy) nor for assessing the establish processes related characteristics (coverage and automation)

In the first case, most metrics specified in D1.3 are advanced but even the two basic metrics listed could not be taken with existing tools. In the second case, D1.3 only describes our initial work of inventorying important types of documentation and important software development processes to have in place. However, we have not yet elaborate evaluation methods for measuring documentation nor development processes.

In order to address these two shortcomings, we will derive evaluation methods for both documentation and development processes and we will prototype them on our 5 projects as an early step of WP4. This will allow identifying problems regarding these two evaluation methods for early 2008 and therefore give one year to address the problems identified.