QualiPSo

Quality Platform for Open Source Software

IST- FP6-IP-034763

Deliverable A5.D1.5.6

How the trustworthiness of OSS products and artifacts can be assessed and predicted (v3)

Luigi Lavazza
Sandro Morasca
Davide Taibi
Davide Tosi

Due date of deliverable: 31/10/2010

Actual submission date: 31/01/2011

This work is partially funded by EU under the grant of IST-FP6-034763.

Change History

Version	Date	Status	Author (Partner)	Description
2.0	31/10/2009	final	INS	Final deliverable. It summarizes the contents of wd 5.6.1 (v2) and wd 5.6.2 (v2).
3.0	31/01/2011	final	INS	Final deliverable. It summarizes the whole results of WP5.6.

What is new

This is the third and final release of this document. The previous version illustrated the results of the first round of experimentations carried out in WP 5.6, including the collection of data –both subjective evaluations and objective measures– and their analysis.

This release of the document differs from the previous releases, in that it reports a summary of the final and complete results of the analysis of data concerning OSS trustworthiness and measurable characteristics that have been collected throughout the project.

As usual, this report summarizes the results achieved over the entire duration of the QualiPSo project. The details are in the corresponding working documents [12] [13].

EXECUTIVE SUMMARY FOR DELIVERABLE A5.D1.5.6

Work organization

The work in workpackage WP5.6 is organized into two tasks:

- Task 5.6.1: Experimentation
- Task 5.3.2: Model building.

Both Experimentation and Model building have been performed in two rounds. This document accounts for the results obtained after the second and final round.

In Task 5.6.1, the techniques and tools defined in Activity A5 are used for experimentation purposes. Specifically, we used the trustworthiness factors identified in WP5.3, the test approaches, suites, and benchmarks identified in WP5.4, and the tools developed, customized, and integrated in WP5.5.

The main result generated by the experimentation of Task 5.6.1 consists of the data on the trustworthiness of the OSS products examined during the experimentations [12]. These data are the input to Task 5.6.2, which analyzed them to find out whether the factors identified are actually influential on the trustworthiness of the OSS products and artefacts. The goal of Task 5.6.2 was to build a quantitative model of software trustworthiness [13], to explain how the trustworthiness perceived by OSS users depends on the actual (mostly objectively assessable) qualities of the OSS products. Note that we consider different types of "users," namely all the professional figures that deal with an OSS product, including developers, integrators, system administrators, product managers, end users, etc.

The produced models will also allow users to estimate how trustworthy a given OSS product is likely to be, on the basis of its measurable characteristics.

Method

The main instrument for the experimentation is represented by empirical studies and measurement.

According to the indications from WP5.3, the experimentation addressed two aspects of trustworthiness: users' perception of trustworthiness and the contribution to trustworthiness from the intrinsic characteristics of the software products and projects. The former was assessed by collecting evaluations from users; the latter was measured.

Users evaluations were collected by means of questionnaires that the users filled out on paper while one of the authors was present to answer possible questions.

The measurements of the OSS product were performed by using the tools identified, produced, or customized in WP5.5. The collected information was stored in a specifically designed repository, from which it is retrieved for analysis purposes.

The collected data were analyzed to find out whether the factors identified are actually influential on the trustworthiness of the OSS products and artefacts, and to derive quantitative models of such dependencies.

A variety of different statistical techniques were used for data analysis, based on the specific nature of the independent and dependent variables involved and the objectives of the data analysis. In particular, logistic regression was largely used to correlate trustworthiness with objectively measured qualities of OSS products.

Results

We were able to find several statistically significant correlational models for the prediction of users' perceptions of a number of user-relevant qualities such as reliability, usability, portability, functionality, interoperability, security, performance, usefulness of the developer community, documentation quality, and overall trustworthiness. We call the ensemble of these correlational models MOSST (Model of Open Source Software Trustworthiness). Thus, we were able to find at least one quantitative model for predicting every subjective quality for which we collected data from users by means of questionnaires. The models in MOSST are built by using data on a number of objective measures on OSS products and projects, like modularity, defect density, size, number of downloads, to predict the user-relevant qualities. .

As we focused on Java and C++ products, we derived three classes of prediction models:

- models for Java programs' perceived qualities with the objective measures produced by a collection of QualiPSo tools, including MacXim (for both object-oriented measures and Elementary Code Assessment rule violations), StatSVN (for measures concerning software configurations and versioning activities) and OSLC (for measures concerning licensing);
- models for C++ programs, which were obtained considering the code measures produced by Kalibro;
- models for Java and C++ programs' perceived qualities with the objective measures produced by the objective measures produced by MacXim and Kalibro. Since java and C++ are subject to different measures, only the measures that are common to both languages were considered;

Novelty and use of the results

MOSST is a set of quantitative models that account for the dependence of the perceivable qualities of OSS on objectively observable characteristics of OSS products and projects. These models can be used by:

- end-users and developers that would like to (re)use existing OSS products and components, to evaluate the level of trustworthiness, reliability, usability and several other qualities of these OSS products that can be expected based on objectively observable characteristics of OSS product and projects

- the developers of OSS products, who can set code quality targets based on the level of trustworthiness, reliability, usability and several other qualities they want to achieve.

Unlike existing quality models for OSS, MOSST is built by means of a theoretically valid approach and solid statistical techniques that use evidence coming from OSS stakeholders and the analysis of actual OSS products and projects. We collected data from 694 OSS stakeholders and obtained 4101 evaluations on 22 Java and 22 C++ programs. This has allowed us to build statistically valid models to quantitatively predict the impact of objectively observable characteristics of OSS products and projects on qualities of practical interest, instead of collecting only data on objectively observable qualities of the code that may only be conjectured to influence qualities of practical interest.

Also, this is one of the few studies that address and build models for different languages, albeit both belonging to the category of Object-Oriented languages, to start building from the knowledge acquired on programs written in individual languages and find out common trends.

Project planning and control

According to the Description of Work, the deliverables and working documents due by WP5.6 - Experimentation and model building in the final part of the QualiPSo project are listed in Table 1 and Table 2, respectively.

Table 1. Due deliverables

Deliverable No	Deliverable title	Delivery date	Nature	Dissemination
A5.D1.5.6	How the trustworthiness of OSS products and artifacts can be assessed and predicted (v 3)	48	R	PU

Table 2. Due working documents

Working Doc. No	Working Document title	Delivery date	Nature	Dissemination
wd5.6.1	Experimentation on the trustworthiness of Open Source Software (version 3)	46	R	PU
wd5.6.2	Trustworthiness models for Open Source Software (version 3)	48	R	PU

Both the deliverable and the working documents were released at the end of month M51.

Document Information

IST Project Number	FP6 – 034763	**Acronym**	QualiPSo
Full title	Quality Platform for Open Source Software		
Project URL	http://www.qualipso.org		
Document URL			
EU Project officer	Michel Lacroix		

Deliverable	**Number**	A5.D1.5.6 QualiPSo	**Title**	A5.D1.5.6: Report: How the trustworthiness of OSS products and artifacts can be assessed and predicted (v3)
Work package	**Number**	5.6	**Title**	Experimentation and model building
Activity	**Number**	A5	**Title**	Trustworthy Results

Date of delivery	**Contractual**	31/10/2010	**Actual**	31/01/2011
Status			final ☑	
Nature	Report ☑ Demonstrator ☐ Other ☐			
Dissemination Level	Public ☑ Consortium ☑			
Abstract (for dissemination)	The quality of Open Source Software (OSS) is much debated, since OSS is used by a continuously growing number of people and organizations. However, the discussions on the quality of OSS are usually based on opinions, feelings, personal preferences, and sometimes even political ideas. This document reports on an analysis of the perceived quality of OSS and the objectively measurable factors that may influence it. Specifically, the users' and developers' evaluations of the trustworthiness of OSS products (and of related qualities, like reliability and functionality) were collected and correlated to objective code measures. The result is a set of quantitative models that account for the dependence of the perceivable qualities of OSS on objectively observable qualities of the code. The ensemble of these models is called MOSST (Model of Open Source Software Trustworthiness). MOSST can be used by: 1) end-users and developers that would like to reuse existing OSS products and components, to evaluate the level of trustworthiness, reliability, usability and several other qualities of these OSS products that can be expected based on objectively observable characteristics of OSS product and projects; 2) the developers of OSS products, who can set code quality targets based on the level of trustworthiness, reliability, usability and several other qualities they want to achieve.			
Keywords	Trustworthiness, quality model, GQM, trustworthiness evaluation, empirical studies, measurement.			
Authors (Partner)	Luigi Lavazza, Sandro Morasca, Davide Taibi, Davide Tosi (University of Insubria)			
Responsible Author	Luigi Lavazza		**Email**	luigi.lavazza@uninsubria.it
	Partner	INS	**Phone**	+39-0332-219830

TABLE OF CONTENTS

1 THE BIG PICTURE

Here, we summarize the work carried out in WP 5.6.

Figure 1 reports the conceptual model of the entities involved in the work. We start with a GQM measurement plan –defined in WP5.3– whose execution will lead to the construction of the QualiPSo model of trustworthiness. The execution of the GQM plan involves two phases: the actual measurement and the analysis of the collected data (details of these activities are reported in in the various versions of WD 5.6.1 and WD 5.6.2).

The GQM plan involves two types of measures: objective measures, which are meant to quantify the intrinsic, objective properties of the OSS products, and subjective measures (called "subjective trustworthiness evaluations" in Figure 1), which are meant to represent how users (subjectively) evaluate the trustworthiness of OSS products.

The actual measures corresponding to the GQM measures definitions are collected and stored in a repository.

There is a set of measures for every considered OSS product.

The analysis phase that is described in WD 5.6.2 aims at correlating the objective, measurable properties of OSS products (like modularity, defect density, size, etc.) with their properties (like reliability, security, etc.) that are relevant for the users. Trustworthiness is the ensemble of the subjective properties.

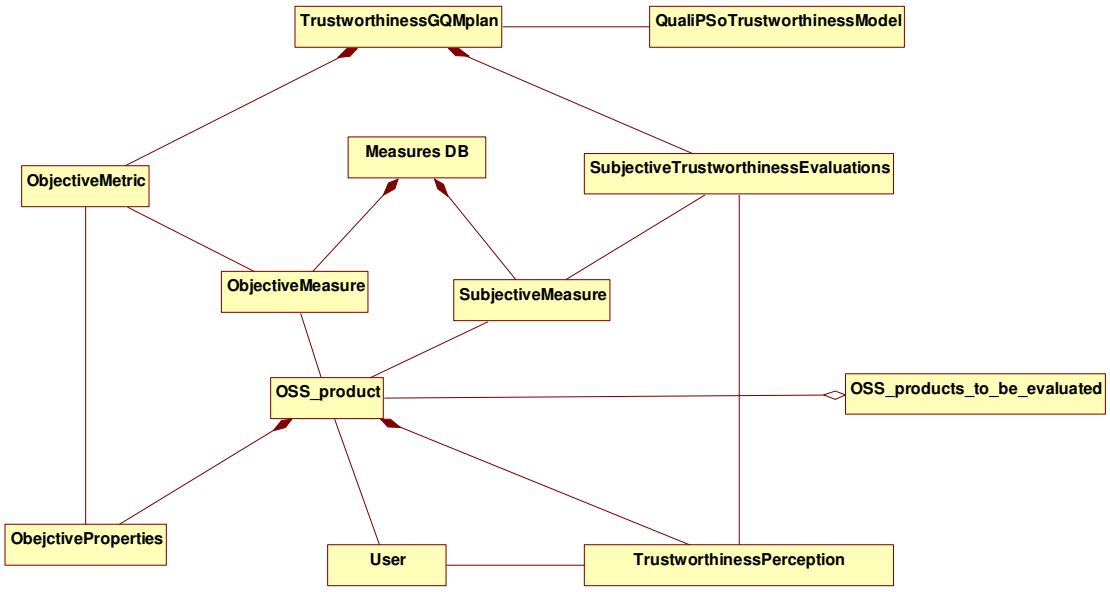

Figure 1. Conceptual model of the items involved in WP5.6.

A high level view of the process carried out in WP5.6 is reported in Figure 2. As already mentioned, the GQM plan is defined in WP 5.3. The GQM plan and the list of examined projects drove the collection of –subjective and objective– data. The collection of data was largely supported by tools (namely, those developed

in WP5.5) but not completely automated, since some of the required information can be safely retrieved only manually.

The collected data are analyzed and quantitative models of trustworthiness and other user-relevant properties are derived. The data analysis activity also provided suggestions about the refinement, extension or reduction of the GQM plan. In fact, the work described in Figure 2 was carried out in two subsequent phases.

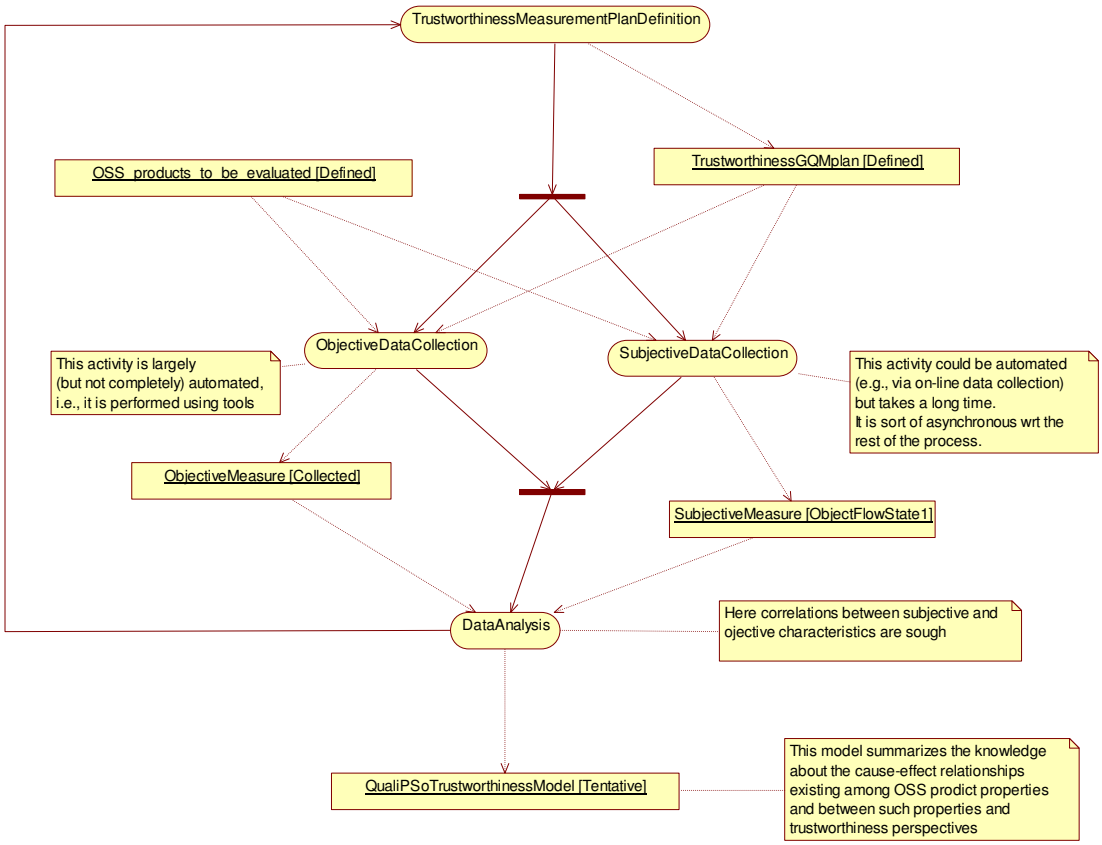

Figure 2. Workflow of activities in WP5.6.

2 THE DATA COLLECTION

2.1 The OSS products being analysed

The first round of experiments was performed on a set of 44 OSS products, of which 22 written in Java and 22 written in C++.

In the first round of experiments the subjective evaluations by users were collected about the whole set of 44 projects, while the analysis was performed only on the Java products, since the tools for analysing C++ code were still under development.

In the second round of experiments the analysis was performed on the whole set of products, and considering all the subjective evaluations collected.

The list of products and the selection criteria, as well as the questionnaire, are reported in [12].

2.2 Refinement of the measurement plans

This activity concerned the refinement of the GQM plan defined in WP5.3, to assure that the measures' definitions match the characteristics of the products to be evaluated.

Some of the measures defined in the GQM plan [8][9] had to be refined to clarify the details that were necessary for selecting the proper tools and defining the actual measurements performed in the experimentation.

The products to be evaluated were duly taken into account, since their characteristics can affect the precise (e.g., operational) definition of measures.

The quality factors affecting user-relevant qualities have been refined or otherwise reviewed.

The corresponding updated definitions are reported in the appendix of [10].

2.3 The data repository

The data collected by means of measurements, interviews, from other data sources, etc., are stored in a well-structured, persistent repository that supports the analysis activities to be performed in the context of Task 5.6.2.

The repository is integrated –at the data level– with the measurement and data collection tools. The repository receives the data from the various tools and makes them available to the analysis activities and to the reporting tool (Spago4Q), as shown in Figure 3.

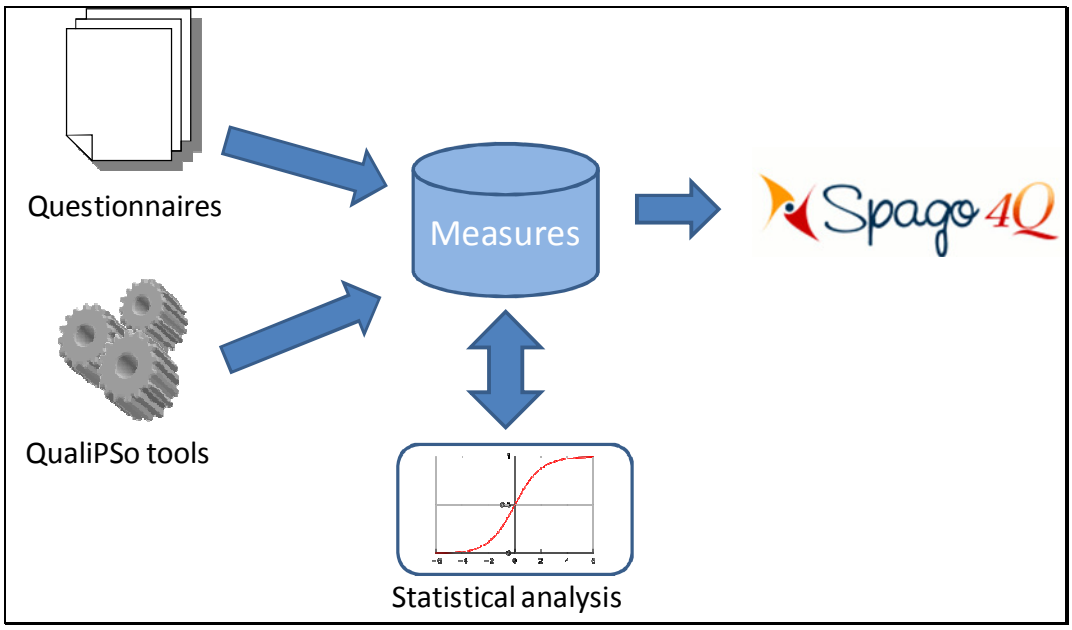

Figure 3. Role of the measures DB in WP5.6.

The repository is based on MySQL relational DBMS.

2.4 Instrumentation

This activity dealt with choosing the proper tools to perform the measurements and the analysis of data.

All the tools developed, customized or otherwise delivered by WP5.5 are used. However, we also use 'third party' OSS tools that match our measurement plan. The main tools used are:

- MacXim (incorporating also Checkstyle and PMD)
- Kalibro
- StatSVN/StatCVS
- OSLC

The collection of the subjective evaluations of the various aspects of trustworthiness by users was carried out mainly via a questionnaire that is reported in the appendix of [12].

For the analysis of data we use R [14], the statistical analysis tool that was already used in WP5.1. R is a GPL-licensed tool that uses a language and an environment for statistical computing and graphics that is reasonably easy to use and comes with a huge repository packages for analysis, database integration, etc. (see the Comprehensive R Archive Network at http://cran.r-project.org/). Thus, R is programmable. To carry out the analysis reported here, we wrote R code, which has been released as part of the QualiPSo open software tools developed within WP5.5.

2.5 Measurement and data collection

The objective of data collection activities is to fulfil the GQM plan, by collecting all the defined measures. Different methods have been used for the different types of measures. Among others, we used the following techniques:

- Data collection from users, concerning their evaluation of (the various aspects of) trustworthiness was performed by means of a questionnaire concerning 22 Java programs and 22 C++ programs. Up to the end of the data collection reported here (early October 2010), 694 questionnaires were collected. Overall, they account for 4101 evaluations (of which 1357 for Java projects and 2744 for C++ projects).
- The chosen projects were measured (i.e., their characteristics were objectively evaluated) using the tools developed in WP5.5.
- Dynamic measurement was performed along the lines defined in WP5.4.

The details of the measurements are available in [12].

3 DATA ANALYSIS

The analysis presented here was carried out on the data collected by Task 5.6.1 and stored in the measurement repository. The analyses reported in this document are based on the evaluations and measures collected up to October 2010.

To find relationships that link the trustworthiness of the OSS products and artefacts with the objectively measurable characteristics and qualities of software, we mainly used binary logistic regression. Binary logistic regression is a statistical technique that is used when the dependent variable is dichotomous (i.e., it takes two values: in our case the fact that a user is satisfied or unsatisfied with a product with respect to a specified quality) and the independent variables are of any type. Binary logistic regression estimates the probability that the dependent variable assumes one of the two values. In our case, therefore, we estimate the probability that a user is satisfied with a product as for a specified quality. This probability indicates the percentage of users that are satisfied with the product with respect to that quality.

Each of our dependent variables represents the subjective evaluations of users concerning a specific quality. Every quality evaluation is a dichotomy in the sense that users' evaluations are divided into two sets: the one containing positive evaluations and the one containing negative evaluations. Evaluations are classified as positive or negative with respect to a threshold, set as explained in Section 3.2.

3.1 Analysis procedures

As mentioned above, and described in the working documents and deliverables of WP5.6, the main goal of this task is to correlate subjective user evaluations with objective software measures.

All subjective evaluations are expressed by each user in an ordinal scale with grades from zero to six.

We interviewed several users about a given quality of a given OSS product, so we need to reduce this amount of data to a single number that can be effectively treated. To this end, we establish a threshold that represents an acceptable quality level and then partition the population of the respondents into two datasets: one containing the users that rated the product below the threshold, and one containing the users that rated the product above the threshold.

More formally, given an OSS product P and a quality Q, we start from the multiset[1] of evaluations $E = \{e_i\}$, where $i \in [1..N]$ indicates the i-th user, N is the number of interviewed users, and e_i is the rating of the quality Q of product P according to the i-th user.

[1] A multiset or bag is a set with repetitions. This clearly accounts for the fact that multiple users can assign a given quality of a given product the same grade.

By establishing a threshold T, we can partition E into E_s and E_u, the multisets of satisfied and unsatisfied users, respectively:

$E_s = \{x \mid x \in E \wedge x > T\}$

$E_u = \{x \mid x \in E \wedge x \leq T\}$

Now, we are not interested in distinguishing user identities; rather, we are interested in how many users are satisfied and how many are unsatisfied. To this end, we consider the pairs $< |E_s|, |E_u| >$ of the cardinalities of E_s and E_u.

Of course, we have a pair $< |E_s|, |E_u| >$ for every subjective quality defined in the GQM plan [8][9][10] and actually collected [12]. For every quality we have thus a pair, which can be interpreted as a percentage of satisfaction ($|E_s|/(|E_u|+|E_s|) = |E_s|/N$).

Since we performed the evaluation of several OSS products, we actually have a vector of pairs and percentages:

$V_e = <P_j>$, where P_j is the pair $< |E_s|, |E_u| >$ concerning the j-th OSS product.

Actually we have not just one vector, but several: one for each investigated quality. Similarly, we have a vector for each objective quality that has been measured.

The analysis consists in correlating a vector of subjective evaluations with one or more vectors of objective measures, in order to evaluate to what extent the qualities perceived by the users depend on the internal, objectively measurable qualities. For instance, we correlated Trustworthiness to the measures of size and complexity, as well as reliability to the measures of modularity.

The analysis was based on binary logistic regression. Binary (or binomial) logistic regression is a form of regression which is used when the dependent variable is a dichotomy and the independent variables are of any type.

Logistic regression has many analogies to linear regression. Unlike the latter, however, logistic regression does not assume linearity of relationship between the independent variables and the dependent, does not require normally distributed variables, does not assume homoscedasticity, and in general has less stringent requirements. It does, however, require that observations be independent and that the independent variables be linearly related to the logit of the dependent.

The logistic curve, illustrated in Figure 4, is better for modeling binary dependent variables coded 0 or 1 because it comes closer to hugging the y=0 and y=1 points on the y axis. Even more, the logistic function is bounded by 0 and 1, whereas the linear regression function may predict values above 1 and below 0.

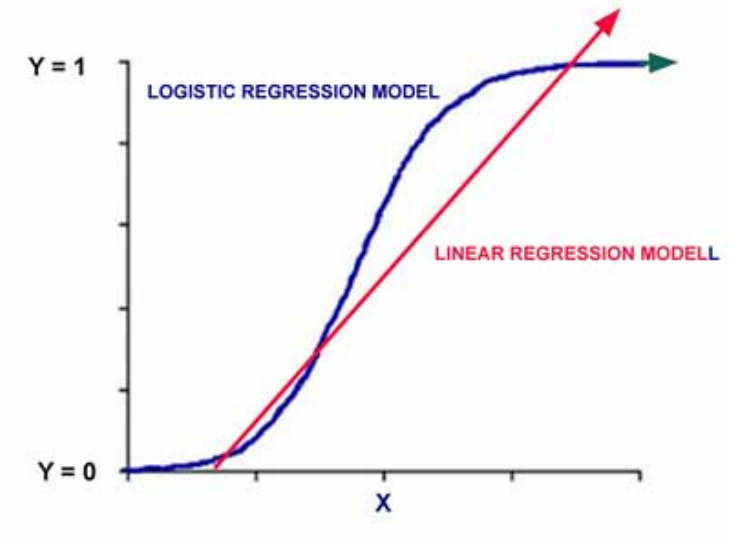

Figure 4. Logistic vs. linear regression curves.

Additional information on the logistic regression is reported in the appendix of wd 5.6.2.

We tested all the possible correlations, i.e., all the combinations y=f(X), where y is a subjective evaluation and X is an array of measures. Of these potential correlations, only the statistically significant ones were selected as valid models.

It must be noted that using multiple independent variables leads to the danger of "overfitting." Since the number of OSS products we analyzed is somewhat limited, we used up to three independent variables in every correlation. However, when more projects are analyzed, it will be possible to study models with a larger number of independent variables (even though this does not appear necessary, as several statistically valid and accurate models are currently available for each of the considered qualities).

3.2 The dataset

The analysis reported here is based on the subjective evaluations about OSS collected throughout the QualiPSo project.

For every subjective evaluation, we used the numbers of satisfied and not satisfied users. The threshold is 4, i.e., users who ranked a product > 4 were counted as "satisfied," while those who ranked it ≤ 4 were counted as "not satisfied." We chose to set the threshold to 4 because in this way we are able to distinguish really satisfied users from other users and also because in the set of responses really few users gave a 1 or 2 score to a product for some quality. So, using a lower threshold would have implied an increased level of general satisfaction for all products, thus making it hard to derive models that are practically useful to say how good a product is with respect to a given quality.

For each product we have a variable number of users' evaluations, since most popular products like Eclipse or MySQL tend to be evaluated by more users

than products –like Weka or Tapestry– that are or interest to a smaller, often specialized, set of users.

Moreover, some users reported a low familiarity with the products in the questionnaires.

Accordingly, we had to select the data to be used for the analysis: only products for which no less than six subjective evaluations expressed by users having a good familiarity with product were considered in the analysis. As a consequence, every analysis involved 16 to 19 products, depending on the specific quality being considered. As a matter of fact, users were invited to provide their own evaluations only for those qualities of a product about which they felt confident in their own judgment. Thus, we had an uneven number of evaluations per quality and per product.

3.3 Outcomes from analysis

The complete and detailed results of the performed analysis are reported in [13]. In that document, every correlation found is illustrated by means of a set of results from the statistical analysis as illustrated in Figure 5.

```
 Reliability  vs.  LCOM ,  Num. interfaces
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  1.3718709280 0.3043314130  4.507819 6.549741e-06
x1          -0.0007125015 0.0002546989 -2.797426 5.151154e-03
x2          -0.0014663310 0.0005899464 -2.485533 1.293577e-02
R2log =  0.9215926
Excluded as outliers:  Eclipse HttpUnit Ant Struts (4/16)
MMRE =  19.73220
Pred(25) =  75
Error range = [-15.67529 .. 131.7843]
```

Figure 5. Data about a correlation.

The first line indicates the statistical correlation being reported: the correlation reported in Figure 5 concerns reliability vs. LCOM (the Lack of COhesion between Methods measure) and the number of interfaces.

The following lines reports in the first column the values of the coefficients of the correlation (where x1 and x2 indicate the independent variables as reported in the title, thus x1 = LCOM and x2 = Num. interfaces). Therefore, $z = 1.371870928 - 0.0007125015\,x_1 - 0.001466331 x_2$ and $Reliability(z) = \frac{1}{1+e^{-z}}$.

The column 'Pr(>|z|)' indicates the significance of the coefficients: all the values, except the one concerning the intercept, should be < 0.05. In fact, we adopt 0.05 as a threshold, as usually done in empirical software engineering.

Note that the sign of the coefficient also provides the indication of whether the dependent variable increases or decreases when an independent variable increases. For instance, Figure 5 shows that LCOM has a negative coefficient, so the probability of a product to be considered reliable decreases when LCOM increases. The same applies to Num. Interfaces. In the next section, a Logistic Regression model is shown to predict Portability based on NOC (the Number of

Children of a class in a inheritance hierarchy). The sign of the coefficient of NOC in the Logistic Regression model is positive so the probability that a product is believed to be portable increases when NOC increases, as **Figure 6** shows.

R2log is the value of R^2_{log}, a measure of goodness of fit defined in [18] that ranges between 0 and 1: the higher R^2_{log}, the higher the effect of the model's explanatory variables, the more accurate the model.

The next line reports the products that were excluded from the analysis, having been considered outliers. In our example, 4 products out of 17 (namely, Eclipse, HttpUnit, Ant and Struts) were excluded as outliers. The presence of outliers is a common problem that arises when building correlational models. An outlier is a data point that lies far from the bulk of the data points and which may overly and unduly influence the regression model. We identified and excluded outliers based on their Cook's distance [19].

The last three lines give some indication on the precision of the fitting. MMRE Mean Magnitude of Relative Error) indicates what is the average absolute percent error: values below 25% are generally considered good. Pred(25) indicates how many products are within ±25% error with respect to the regression line. Finally, the error range indicates the minimum and maximum distance between observed values and estimated ones (always in percentage terms).

3.4 Results and how to use them

MOSST is the main result of the activity reported in this document. MOSST does not only show that relationships exist between trustworthiness (and other perceived qualities) and objectively measurable characteristics of the OSS. A really important point is that MOSST *quantifies* the nature of these relationships.

The quantitative knowledge of the relationships can be beneficial to both the users and the developers of OSS:

- The users can rely on the measures of the software in order to estimate to what extent a given OSS product can be expected to satisfy a given quality aspect (e.g., reliability). In this way, the potential users can get a rough evaluation of OSS without the need to even try the product.
- Developers can derive from their client satisfaction targets (i.e. to what extent users will be satisfied with a given quality of their OSS product) into threshold of quality measures that must be met by their code.

The procedure for using the quantitative knowledge of relations is exemplified below, considering the dependency of Portability on the average number of children of classes in Java products. The equation that describes the correlation found is the following:

$$Portability = \frac{1}{1 + e^{-(0.0061183 + 0.44062\ NOC)}}$$

For a user that would like to evaluate the portability of a Java product based on other users' perceptions, the procedure is simple: if the average number of children per class of the Java product is 0.8 then the user can expect that the product's portability will be satisfactory with probability around 60%, i.e., 60% of the users will be satisfied with the product's portability. In fact $\frac{1}{1+e^{-(0.0061185+0.440062 \cdot 0.8)}} = 0.5887$.

Instead, if the average number of children per class is 1.4, then the user can expect that the product's portability will be satisfactory with probability around 65%. In fact, $\frac{1}{1+e^{-(0.0061185+0.440062 \cdot 1.4)}} = 0.651$.

Figure 6 shows the relation between portability and NOC, highlighting the values of portability for NOS = 0.8 and NOC = 1.4.

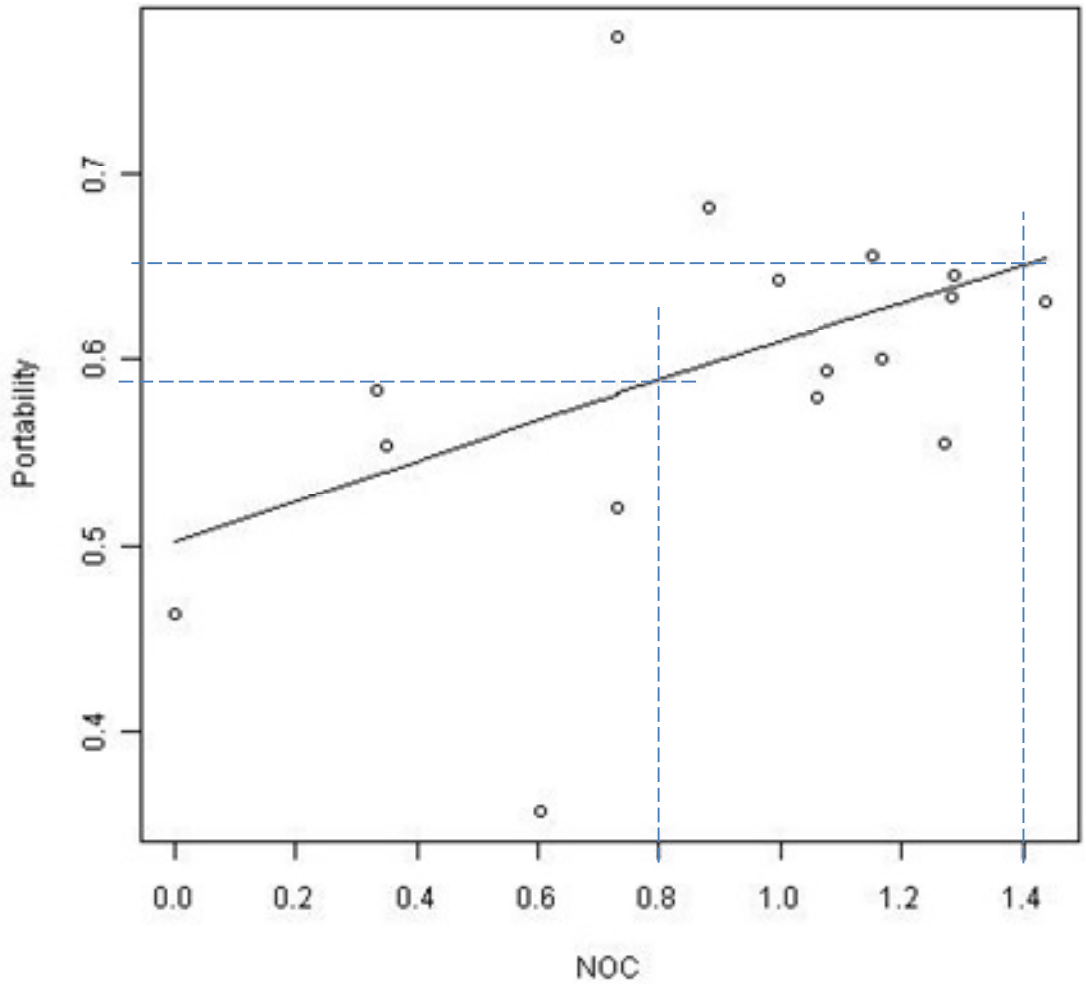

Figure 6. Portability vs. NOC (number of children) for Java products.

For developers, the procedure is reversed, since they have to establish what is the minimum value of NOC to achieve a target portability probability. By solving for NOC the equation that describes the relation between portability and NOC we get:

$$NOC = 0.44062^{-1}\left(-0.0061183 + \log\left(\frac{Portability_{target}}{1 - Portability_{target}}\right)\right)$$

If the goal of the developer is that the product is considered trustworthy by over 55% of the users, he/she must aim at an average NOC greater than 0.44. In fact, $0.44062^{-1}\left(-0.0061183 + \log\left(\frac{0.55}{1-0.55}\right)\right) = 0.44$.

In this sense, NOC=0.44 can be seen as a "threshold" below which developers should not take their code.

Other thresholds may be established too, by users and developers alike. For instance, one may decide that a product's quality is

- "good" if the product's portability is expected to be rated satisfactory by at least 75% of users
- "acceptable" if the product's portability is expected to be rated satisfactory by 25% to 75% of users
- "poor" if the product's portability is expected to be rated satisfactory by at most 25% of users.

By solving the logistic regression formula for the value of the objectively measurable characteristic and substituting 0.75 and 0.25, one obtains the threshold values for the objectively measurable characteristic that correspond to the 75% and 25% values of satisfied users, so one finds the thresholds on the range of the objectively measurable characteristic. For instance, these thresholds can be displayed by Spago4Q, which is used for result visualization in our tool set.

In conclusion, our analyses let users and developers perform the needed evaluations on the basis of clear quantitative data.

3.5 Improvements over the state of the art

We also would like to point out that we used the theoretically sound way to build models belonging to MOSST, which are relevant to developers and users (e.g., reliability, usability, portability, etc. as listed in Section 0). All of these characteristics are classified as "external" software qualities, as opposed to "internal" software quality in the experimental software engineering literature [20]:

- an "internal" quality of a software product can be quantified based on the knowledge of the software product alone. For instance, one can measure software size based only on the source code. An "internal" software quality has no practical interest *per se*, but it can be used to predict some interesting product (e.g., fault proneness) or process (e.g., effort) quality. "Internal" software qualities are usually easy to measure, possibly with the aid of automated tools.
- an "external" quality of a software product can only be quantified based on the knowledge of the software product and additional information. For

instance, one cannot measure software usability based only on the source code or its user interface, as usability also depends on the skills and knowledge of specific users and the way they interact with the product. “External” software qualities are practically relevant, though it is commonly said that they are usually very hard to measure (and even define).

However, the distinction between “internal” and “external” qualities is entirely peculiar to the software measurement literature. No such distinction exists in the authoritative general references on measurement [21][22][23]. So, this distinction between “internal” and “external” qualities in software engineering literature can be only accepted as an illustration convention used to explain the differences in the nature of software qualities and somehow organize them in categories.

At any rate, the authoritative, foundational work on measurement already shows how these “external” qualities can be measured. Technically, this is carried out via so-called “Probability Representations.” From a practical point of view, this amounts to measuring “external” qualities via prediction models based on “internal” qualities, as we have shown elsewhere [24]and done in our work here, as MOSST is the ensemble of a set of prediction models.

Thus, our approach is different from other approaches that have been used in the quantification of qualities of OSS, like OpenBRR [26], QSOS [25], OSMM [27], OpenBQR [28], which are typically based on weighted sums of directly measurable characteristics. However, these models are not theoretically valid, nor are they validatable, as they provide a *definition* of a quality. From a practical point of view, these models do not provide any reliable indication on whether the directly measurable characteristics they use actually influence the qualities of interest, nor on the values for the weights of the directly measurable characteristics (a discussion on the use of weighted sums in the definition of measures is in [29]). So, the choice of directly measurable characteristics and the values used for their weights are fairly subjective, and, as a result, so is the definition of the quality. Instead, the solid statistical analysis used in MOSST shows which directly measurable characteristics are truly influential on the OSS qualities of interest and which weights should be used, based on an extensive set of data coming from the field and not on some fairly subjective analysis.

So, unlike existing quality models for OSS, MOSST is built by means of a theoretically valid approach and solid statistical techniques that use evidence coming from OSS stakeholders and the analysis of actual OSS products and projects. We collected data from 694 OSS stakeholders and obtained 4101 evaluations on 22 Java and 22 C++ programs. This has allowed us to build statistically valid models to quantitatively predict the impact of objectively observable characteristics of OSS products and projects on qualities of practical interest, instead of collecting only data on objectively observable qualities of the code that may only be conjectured to influence qualities of practical interest.

4 SAMPLE MODELS

In this section we report a representative model belonging to MOSST for each of the subjectively evaluates qualities that have been considered in the project. All the models reported here are from [13], where the complete set of models – over four hundred– is reported.

The quality level used in this section is the one obtained with threshold = 4, i.e., projects are considered satisfactory if they have been graded 5 (i.e., very satisfactory) or 6 (completely satisfactory). Setting the threshold to such a high level brings to the identification of really good products, and sets a challenging target for the objectively measurable quality of code.

4.1 Trustworthiness

The model states that trustworthiness increases for well modularized products (here modularization at the class level is indicated by the coupling between objects: the lower the coupling, the better the modularization).

```
=====================================================================
Trustworthiness  vs.  CBO
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  1.0898005  0.2500808  4.357793 1.313805e-05
x1          -0.1206116  0.0562384 -2.144648 3.198103e-02
R2log =  0.9198285
Excluded as outliers:  Eclipse Perl Saxon  ( 3 / 19 )
MMRE =  17.77963
Pred(25) =  73.68421
Error range = [ -99.39152 .. 52.6094 ]
=====================================================================
```

After removing data coming from projects Eclipse, Perl and Saxon, which appeared as outliers, we obtained a statistically significant logistic regression model, which is represented in **Figure 7**.

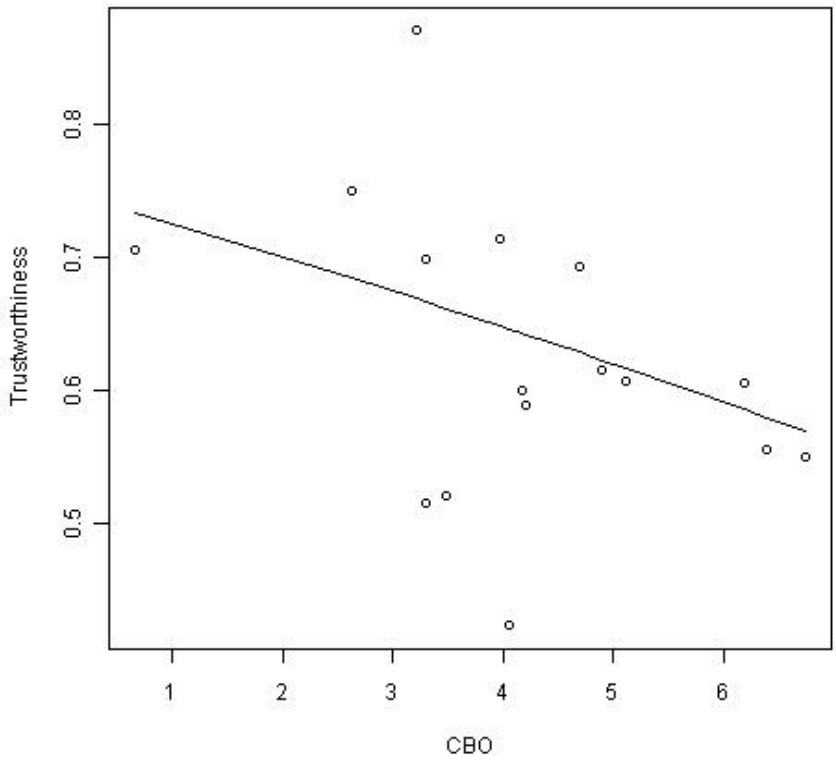

Figure 7. Trustworthiness (threshold = 4) as a function of CBO: Logistic regression.

The variable Trustworthiness, which appears on the y axis, is defined as

$$Trustworthiness = \frac{TrustOK}{TrustOK + TrustKO}$$

i.e., it is the percentage of users that considered the product as satisfactory and rated it above the threshold.

The curve has equation

$$Trustworthiness = \frac{1}{1 + e^{-(1.0898 - 0.12 \times CBO)}}$$

The model is statistically significant with threshold 0.05, since its p-value is 0.03198 (see the value of Pr(>|z|) above).

The precision of the model is documented in **Table 3**.

Table 3. Precision of the fitting of the regression line: indicators

Indicator	Value
MMRE	17.8%
Error range	-99% .. 53%
Pred(25)	74%

More precisely, the distribution of relative residuals (i.e., differences between the estimated values and the real ones) for the various considered projects is reported in **Figure 8**. It can be observed that only a couple of products do not fit very well in the model.

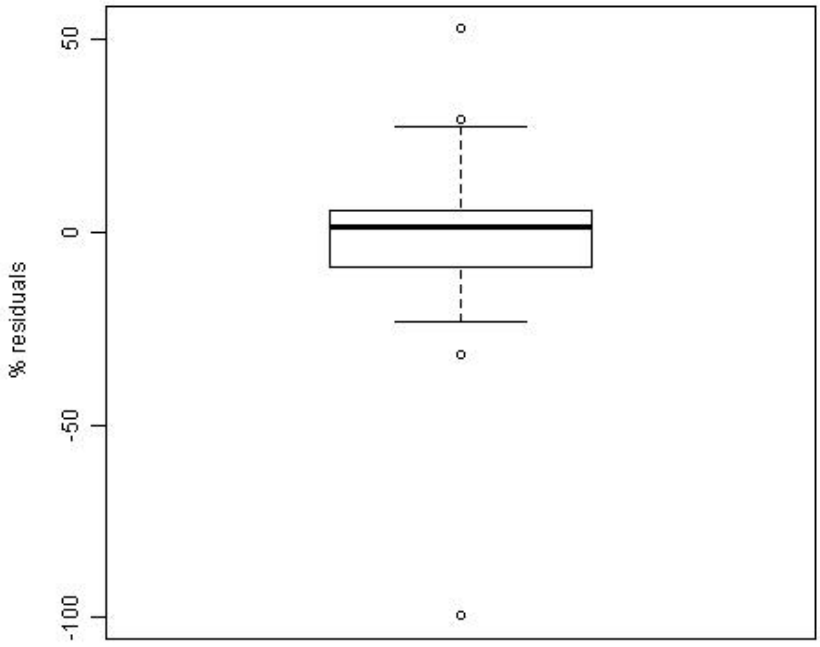

Figure 8. Boxplot reporting the distribution of residuals for the Trustworthiness vs. CBO model.

4.2 Reliability

A very good model correlates reliability with two characteristics of the versioning/reviewing process of OSS. The model indicates that reliability is higher for actively maintained products (high number of commits) that are fairly stable (most files do not need revisions).

```
============================================================
Reliability  vs.  number_of_commits,  avg_number_of_revisions_per_file
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  0.8920003252 2.535470e-01  3.518087 0.000434670
x1           0.0001288062 5.530034e-05  2.329212 0.019847837
x2          -0.1339535226 4.821182e-02 -2.778437 0.005462102
```

```
R2log =  0.9204644
Excluded as outliers:  Eclipse (1/13)
MMRE =  10.67684
Pred(25) =  92.3077
Error range = [-14.19207 .. 33.96322]
============================================================
```

The distribution of relative residuals for this model is reported in

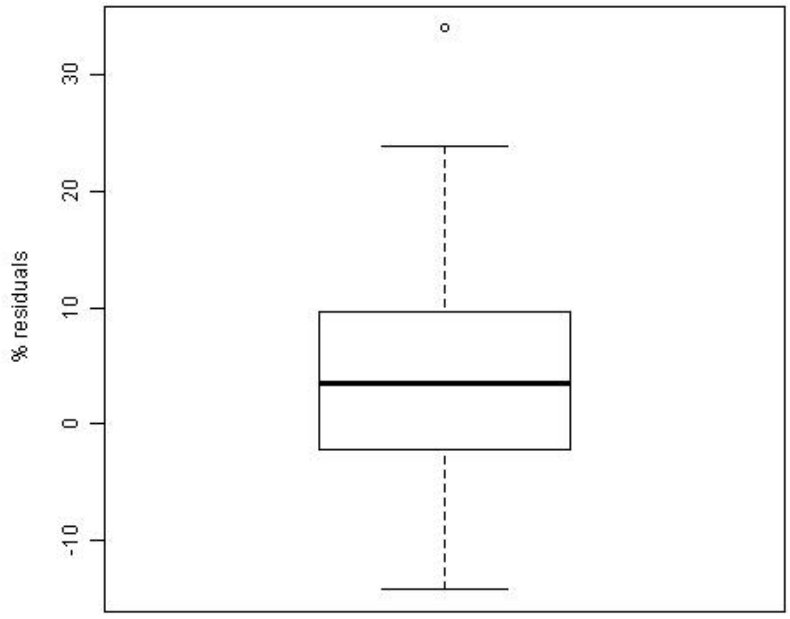

Figure 9. Boxplot reporting the distribution of relative residuals for the model of Reliability vs. Number of commits and average number of revisions per file.

4.3 Usability

An interesting model of usability indicates that users consider more usable the OSS products that are maintained by many developers. This is reasonable, since the more developers, the more likely it is that the user usability needs are taken care in effectively.

```
============================================================================
Usability  vs.  number_of_developers
               Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.46917258 0.27110062 -1.730622 0.08351928
x1           0.02933536 0.01256109  2.335416 0.01952172
R2log =  0.9096772
Excluded as outliers:  Log4J Saxon JBoss Eclipse  ( 4 / 13 )
MMRE =  15.42022
Pred(25) =  84.61538
Error range = [ -52.42674 .. 19.70674 ]
============================================================================
```

The regression line is illustrated in **Figure 10**. The distribution of relative residuals is reported in **Figure 11**.

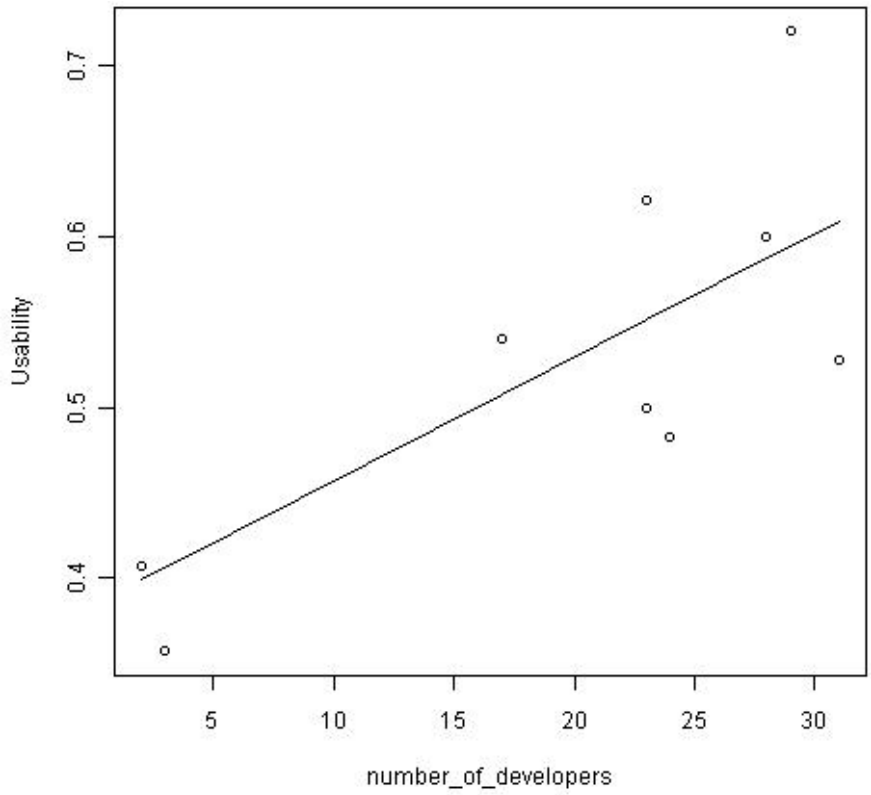

Figure 10. Regression line for the model of Usability vs. Number of developers.

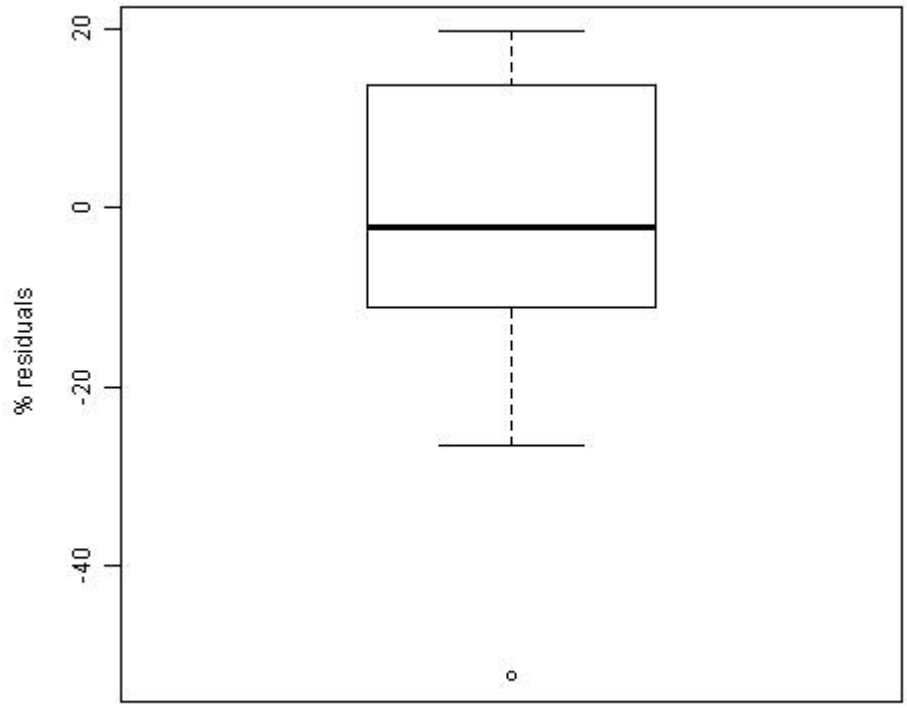

Figure 11. Boxplot reporting the distribution of relative residuals for model of Usability vs. Number of developers.

4.4 Portability

A quite precise model of portability indicates that OSS applications that make larger use of generalization tend to be more portable.

```
=========================================================================
Portability  vs.  NOC
               Estimate Std. Error    z value   Pr(>|z|)
(Intercept) 0.006118287  0.1886794 0.03242690 0.97413161
x1          0.440620371  0.1784647 2.46894945 0.01355104
R2log =  0.9353108
Excluded as outliers:   ( 0 / 16 )
MMRE =  10.71177
Pred(25) =  93.75
Error range = [ -24.75138 .. 58.92724 ]
=========================================================================
```

The regression line is illustrated in **Figure 12** The distribution of relative residuals is reported in **Figure 13**.

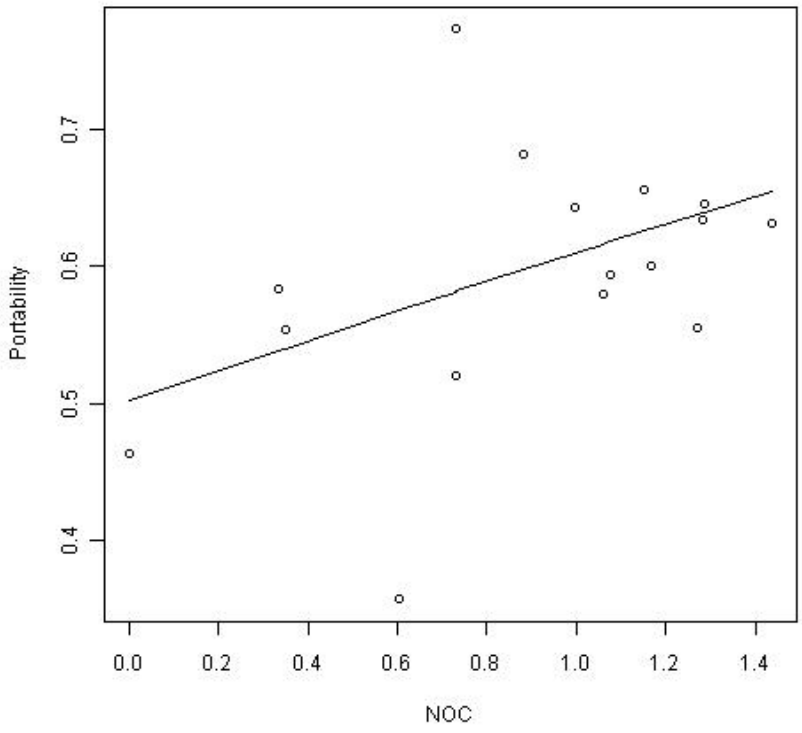

Figure 12. Regression line for the model of Portability vs. NOC (number of children).

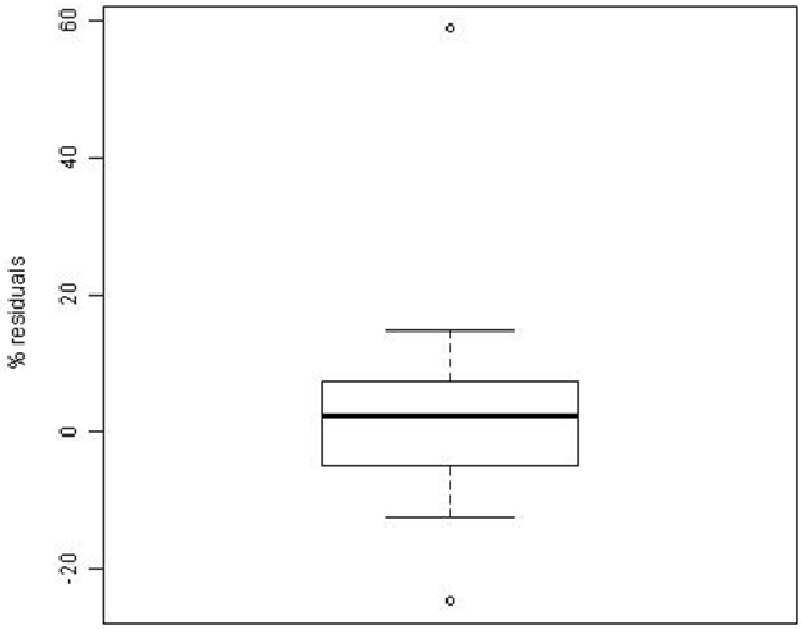

Figure 13. Boxplot reporting the distribution of relative residuals for model of Portability vs. NOC (number of children).

4.5 How well functional requirements are satisfied

An interesting model indicates that the users' requirements are more easily satisfied by products that are made of many packages and that feature a higher than average complexity.

```
=============================================================================
Functionality  vs.  McCabe ,  Num. packages
               Estimate  Std. Error   z value    Pr(>|z|)
(Intercept) -1.757822813 0.607124101 -2.895327 0.003787637
x1           0.825670508 0.260955347  3.164030 0.001556008
x2           0.001780181 0.000643463  2.766563 0.005665057
R2log =  0.9403595
Excluded as outliers:  Log4J Xerces  ( 2 / 16 )
MMRE =  17.37332
Pred(25) =  75
Error range = [ -26.99862 .. 89.81255 ]
=============================================================================
```

The distribution of relative residuals is reported in **Figure 14**.

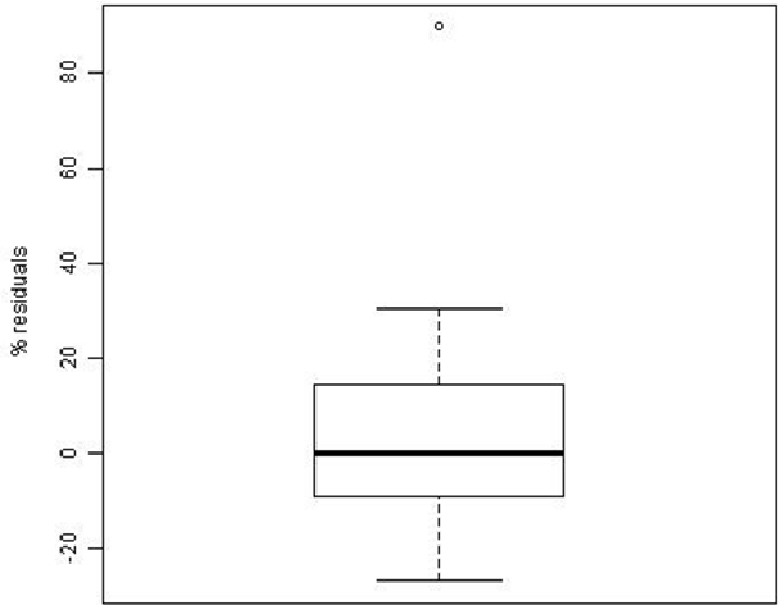

Figure 14. Boxplot reporting the distribution of relative residuals for model of Functionality vs. McCabe and number of packages.

4.6 Interoperability

A quite interesting and fairly precise model indicates that interoperability of OSS products improves with the usage of generalization and the number of methods per interface.

```
=========================================================================
Interoperability  vs.  NOC ,  Num. methods per interface
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -1.2769657  0.5250736 -2.431975 0.01501675
x1           0.8443261  0.3777582  2.235097 0.02541099
x2           0.2734487  0.1135832  2.407474 0.01606331
R2log =  0.9128017
Excluded as outliers:  Eclipse Log4J  ( 2 / 14 )
MMRE =  16.93735
Pred(25) =  78.57143
Error range = [ -29.80316 .. 73.30486 ]
=========================================================================
```

The distribution of relative residuals is reported in **Figure 15**.

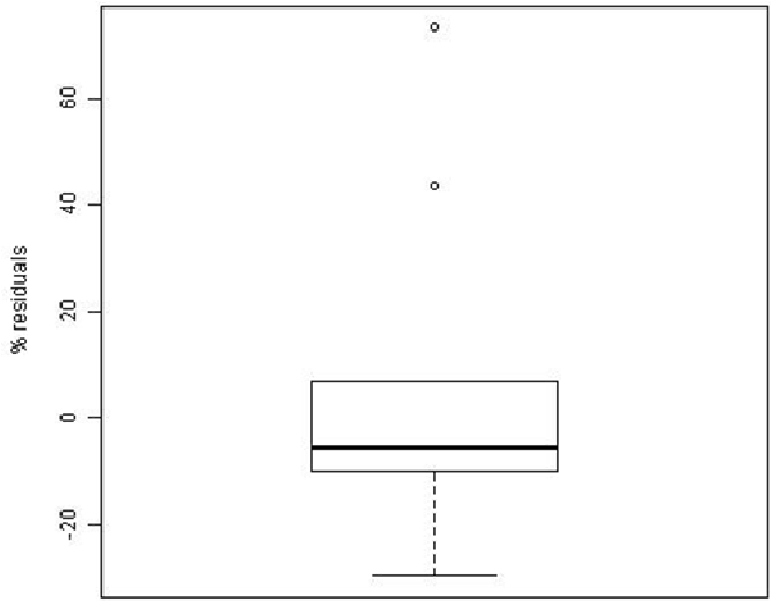

Figure 15. Boxplot reporting the distribution of relative residuals for model of Interoperability vs. NOC and number of methods per interface.

4.7 Security

The model below says that the applications that are considered more secure by users have methods that have been well specified as far as parameters are concerned. For instance, methods with many parameters are expected to have less side-effects and less I/O points, which are usually considered threats to security. This is quite credible.

```
=================================================================
Security  vs.  Num. parameters per method
             Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.9851316  0.4394675 -2.241648 0.02498410
x1           1.1150867  0.5064968  2.201567 0.02769589
R2log =  0.9332337
Excluded as outliers:   ( 0 / 16 )
MMRE =  12.6%
Pred(25) =  93.75
Error range = [ -20.41925 .. 21.6 ]
=================================================================
```

The regression line is illustrated in **Figure 16**. The distribution of relative residuals is reported in **Figure 17**.

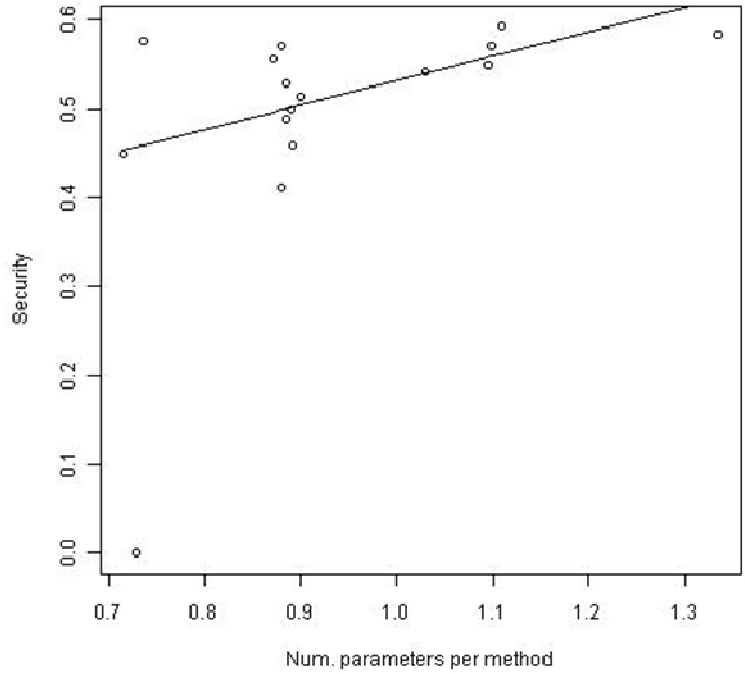

Figure 16. Regression line of the model of Security vs. number of parameters per method.

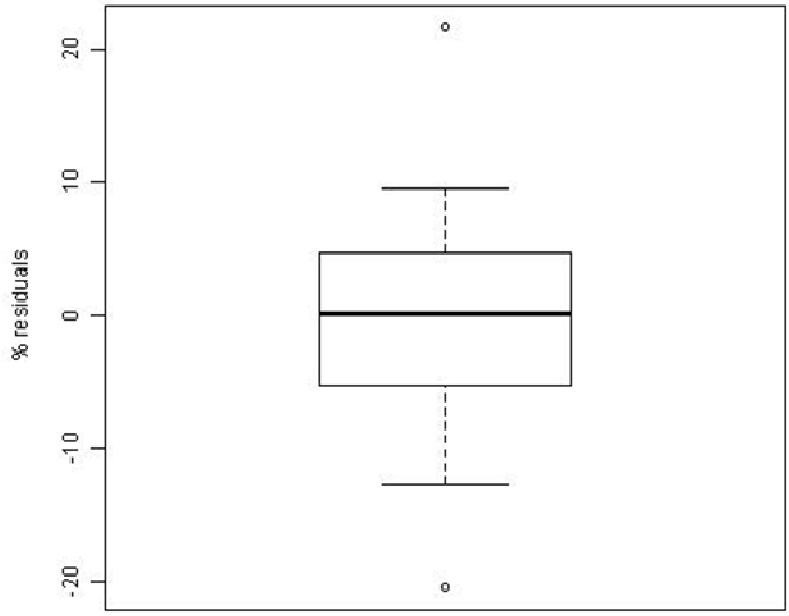

Figure 17. Boxplot reporting the distribution of relative residuals for model of Security vs. number of parameters per method.

4.8 Performance (in terms of speed)

A quite credible model of speed indicates that the efficiency of OSS products decreases with the lack of cohesion between methods and the number of abstract classes. The model may be explained by the fact that higher values for

LCOM imply that most of the needed information is not local, thus requiring indirect or non-optimally efficient access, while a higher number of abstract classes suggests that inefficient late-binding is used.

```
=============================================================================
Speed  vs.  LCOM ,  Num. abstract classes
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  0.2571669343 0.2020958999  1.272500 0.20319570
x1          -0.0004935335 0.0002169290 -2.275092 0.02290042
x2          -0.0039756765 0.0015553041 -2.556205 0.01058207
R2log =  0.929076
Excluded as outliers:  Eclipse PMD  ( 2 / 15 )
MMRE =  17.16227
Pred(25) =  86.66667
Error range = [ -38.8406 .. 105.9486 ]
=============================================================================
```

The distribution of relative residuals is reported in **Figure 18**.

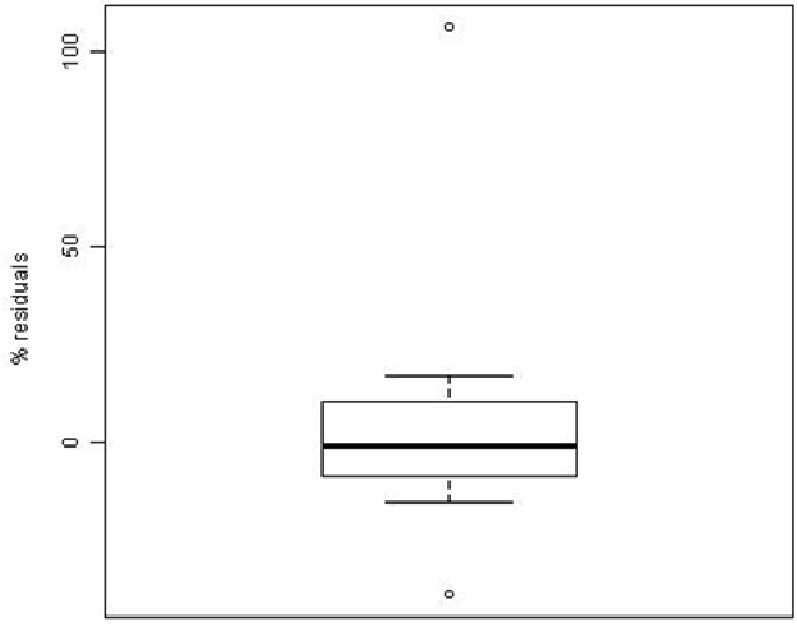

Figure 18. Boxplot reporting the distribution of relative residuals for model of Speed vs. LCOM and the number of abstract classes.

4.9 Usefulness of the product developer community

The model reported below suggests that the usefulness of the community is greater for simpler products that are maintained by a large number of developers.

```
============================================================
Community  vs.  McCabe ,  number_of_developers
               Estimate Std. Error    z value    Pr(>|z|)
(Intercept) -0.14068354 0.37999532 -0.3702244 0.711215296
x1          -0.66074402 0.27967275 -2.3625613 0.018149139
x2           0.06185318 0.02329124  2.6556411 0.007915781
R2log =  0.9497033
Excluded as outliers:  JBoss Saxon JFreeChart  ( 3 / 13 )
```

```
MMRE =  20.63438
Pred(25) =  69.23077
Error range = [ -72.15354 .. 47.90988 ]
============================================================
```

The distribution of relative residuals is reported in **Figure 19**.

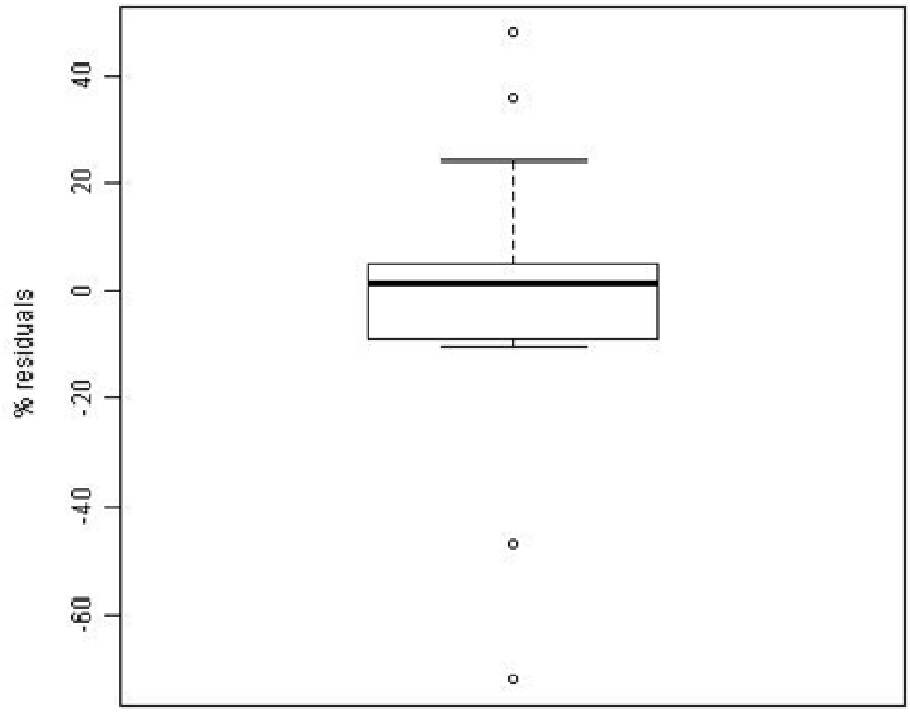

Figure 19. Boxplot reporting the distribution of relative residuals for model of Community utility vs. McCabe and number of developers.

4.10 Documentation Quality

The model reported below states that the quality of documentation improves for OSS products that feature many public methods and are frequently maintained.

```
=====================================================================
DocQuality  vs.  Num. public methods ,  avg_major_release_per_year
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.737036e+00 4.983675e-01 -3.485452 0.0004913072
x1           7.329941e-05 3.067189e-05  2.389791 0.0168579763
x2           8.063410e-01 3.833795e-01  2.103245 0.0354443690
R2log =  0.9045322
Excluded as outliers:  Eclipse Log4J Hibernate  ( 3 / 12 )
MMRE =  19.49775
Pred(25) =  75
Error range = [ -40.20908 .. 69.13603 ]
=====================================================================
```

The distribution of relative residuals is reported in **Figure 19**.

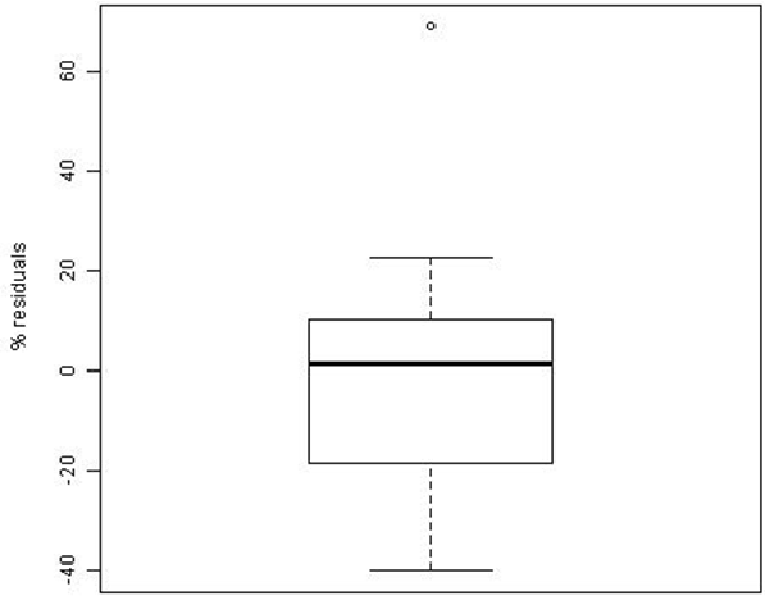

Figure 20. Boxplot reporting the distribution of relative residuals for model of Documentation quality vs. Number of public methods and average major releases pe year.

4.11 Trustworthiness with respect to non Open Source (closed source) products

Building this model is very hard, because it aims at establishing a connection between product objective characteristics and a quality that not only is subjective and external, but is also relative to other products.

One of the best models we found is reported below. It says that OSS products that are considered more competitive are the ones that grow faster (in term of files added per year) on a stable base (there is little need to remove code).

```
=====================================================================
CssCompetitors  vs.  avg_loc_del_per_year ,  avg_files_added_per_year
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  3.691809e-01 1.596831e-01  2.311960 0.02077991
x1          -8.480383e-06 3.593819e-06 -2.359714 0.01828904
x2           1.092654e-03 4.454859e-04  2.452724 0.01417790
R2log =  0.943119
Excluded as outliers:  Struts  ( 1 / 13 )
MMRE =  9.655532
Pred(25) =  92.3077
Error range = [ -13.12781 .. 37.83713 ]
=====================================================================
```

The distribution of relative residuals is reported in **Figure 21**.

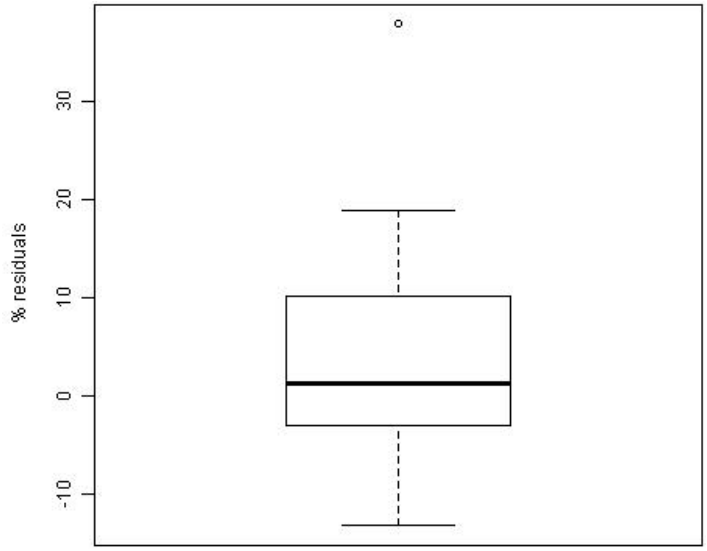

Figure 21. Boxplot reporting the distribution of relative residuals for model of Trustworthiness wrt CSS competitors vs. average LOC deleted per year and average files added per year.

4.12 Trustworthiness with respect to Open Source products

An interesting model indicates that a product is preferable to OSS alternatives if it is better commented and faster growing. This seems to indicate that OSS users are greedy of new functionality (which has to be supported by extensive documentation).

```
=====================================================================
OssCompetitors vs. Comment lines per class,  avg_files_added_per_year
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -0.3065353449 0.3412067533 -0.898386 0.36897984
x1           0.0134078953 0.0065998498  2.031546 0.04219969
x2           0.0009208207 0.0003732333  2.467145 0.01361951
R2log =  0.9174157
Excluded as outliers:  Eclipse JFreeChart Hibernate  ( 3 / 13 )
MMRE =  10.50353
Pred(25) =  84.61538
Error range = [ -7.918586 .. 32.89196 ]
=====================================================================
```

The distribution of relative residuals is reported in **Figure 22**.

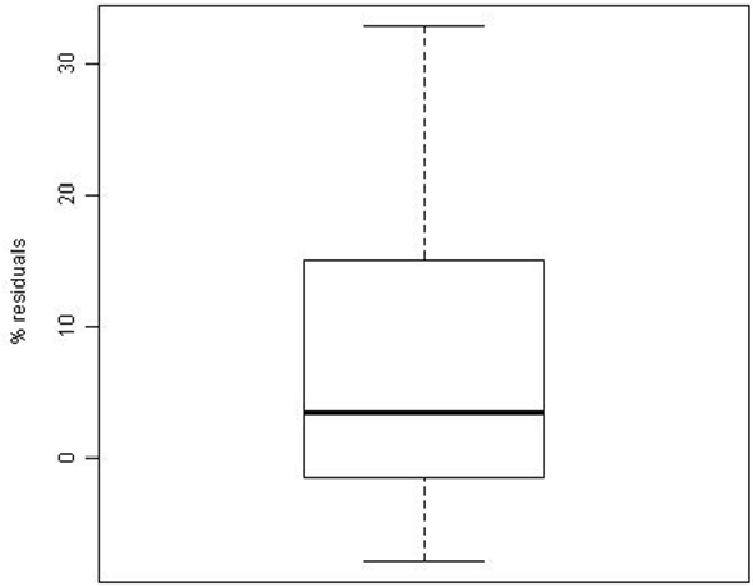

Figure 22. Boxplot reporting the distribution of relative residuals for model of Trustworthiness wrt OSS competitors vs. comment lines per class and average files added per year.

5 PUBLICATIONS

The work reported here (often together with the results of other workpackages) was the base for several publications. Here follows the list.

[1] Vieri del Bianco, Luigi Lavazza, Sandro Morasca and Davide Taibi, "A Survey on OSS Product Trustworthiness", *IEEE Software*. Accepted for publication.

[2] D. Taibi, V. del Bianco, D. Dalle Carbonare, L. Lavazza, S. Morasca, "Towards the evaluation of OSS trustworthiness: lessons learned from the observation of relevant OSS projects", *the 4th International Conference on Open Source Systems*, 7-10 September 2008, Milano.

[3] Vieri del Bianco, Luigi Lavazza, Sandro Morasca and Davide Taibi, "Quality of Open Source Software: the QualiPSo Trustworthiness Model", *The 5th International Conference on Open Source Systems OSS09*, 3-6 June 2009, Skövde, Sweden.

[4] Vieri del Bianco, Luigi Lavazza, Sandro Morasca, Davide Taibi and Davide Tosi, "The QualiPSo approach to OSS product quality evaluation.", *3rd Workshop on Emerging Trends in FLOSS Research and Development (FLOSS-3)*, 8 May 2010, Cape Town, South Africa.

[5] Vieri del Bianco, Luigi Lavazza, Sandro Morasca and Davide Taibi, Davide Tosi, "A Survey on the Importance of Some Economic Factors in the Adoption of Open Source Software", *8th ACIS conference on Software Engineering Research, Management and Applications – SERA 2010, Montreal*, 24-26 May 2010. In Studies in Computational Intelligence, 2010, Volume 296/2010, Springer.

[6] Vieri del Bianco, Luigi Lavazza, Sandro Morasca, Davide Taibi and Davide Tosi, "An investigation of the users' perception of OSS quality", *the 6th International Conference on Open Source Systems, OSS 2010*, 30 May - 2 June 2010, Notre Dame, IN, USA.

[7] Luigi Lavazza, Sandro Morasca, Davide Taibi and Davide Tosi, "Predicting OSS Trustworthiness on the Basis of Elementary Code Assessment", *4th Int. Symposium on Empirical Software Engineering and Measurement – ESEM 2010.* 16-17 September 2010, Bolzano.

6 CONCLUSIONS

The techniques and tools developed in WP5.3, WP5.4 and WP5.5 were successfully used to measure and characterize a set of 44 OSS products.

The opinions of users about these projects were assessed via a questionnaire. 4101 product evaluations were collected.

Statistical techniques –mainly logistic regression– were employed to analyse the collected data.

The result is a set of statistically significant quantitative models that represent the dependence of trustworthiness user-perceivable OSS product qualities on measurable characteristics of the code.

The models found can be beneficial to both users and developers of OSS. Users can estimate the likely trustworthiness of OSS products without going through a thorough evaluation, but just on the basis of the measurable characteristics of the products. Developers have clear indications of what characteristics their products must have in order to increase their probability of being considered trustworthy by users.

7 REFERENCES

[8] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, "Definition of trustworthiness of software products and artefacts, Version 2.0", QualiPSo Working document wd 5.3.1, 31/10/2008.

[9] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, "Identification of factors that influence the trustworthiness of software products and artefacts", Working document wd 5.3.2, Version 2.0 - QualiPSo report, October 2008.

[10] L. Lavazza, S. Morasca, D. Taibi, D. Tosi, "Definition and identification of trustworthiness of software products and artefacts and its influencing factors, version 4", QualiPSo Working document wd 5.3.1, 31/01/2011.

[11] L. Lavazza, S. Morasca, D. Taibi, D. Tosi, "Experimentation on the trustworthiness of Open Source Software, version 2.0", QualiPSo Working document wd 5.6.1, 30/06/2009.

[12] L. Lavazza, S. Morasca, D. Taibi, D. Tosi, "Experimentation on the trustworthiness of Open Source Software, version 3.0", QualiPSo Working document wd 5.6.1, 31/01/2011.

[13] L. Lavazza, S. Morasca, D. Taibi, D. Tosi, "Trustworthiness models for Open Source Software", Working document wd 5.6.2, Version 3.0 - QualiPSo report, January 2011.

[14] The R Development Core Team, "R: A Language and Environment for Statistical Computing - Reference Index, Version 2.9.0 (2009-04-17)", R Foundation for Statistical Computing, 2009.

[15] Chidamber, S. R. and Kemerer, C. F. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.* 20, 6 (Jun. 1994), 476-493.

[16] T. J. McCabe, "A complexity measure", IEEE Transactions on Software Engineering, December 1976.

[17] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, "How the trustworthiness of OSS products and artifacts can be assessed and predicted (v1)", QualiPSo report A5.D1.5.6 version 1.0 – October 2008.

[18] Lionel C. Briand, Sandro Morasca, and Victor R. Basili, "Defining and Validating Measures for Object-Based High-Level Design", IEEE Transactions on Software Engineering, Vol. 25, No. 5, September/October 1999.

[19] Cook, R. D. and Weisberg, S. Residuals and Influence in Regression. Chapman and Hall, London, 1982.

[20] Norman Fenton and Shari Lawrence Pfleeger. Software Metrics (2nd Ed.): A Rigorous and Practical Approach. PWS Pub. Co., Boston, MA, USA. 1997.

[21] Roberts F.S., Measurement Theory: With Applications to Decision Making, Utility, and the Social Sciences, Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1979.

[22] Tversky A., Luce R.D., Suppes P., Krantz D., Foundations of Measurement vol. I: Additive and Polynomial Representations, Academic Press, 1971.

[23] Tversky A., Luce R.D., Suppes P., Krantz D., Foundations of Measurement vol. II: Geometrical, Threshold, and Probabilistic Representations, Academic Press, 1989.

[24] Sandro Morasca. A probability-based approach for measuring external attributes of software artifacts. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09). IEEE Computer Society, Washington, DC, USA, 2009.

[25] Atos Origin, "Method for Qualification and Selection of Open Source software (QSOS), version 1.6", http://www.qsos.org/download/qsos-1.6-en.pdf

[26] "Business Readiness Rating for Open Source - A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software", BRR 2005 - RFC 1, http://www.openbrr.org.

[27] "Making Open Source Ready for the Enterprise: The Open Source Maturity Model", from "Succeeding with Open Source" by Bernard Golden, Addison-Wesley, 2005, available form http://www.navicasoft.com

[28] D. Taibi, L. Lavazza, S. Morasca, "OpenBQR: a framework for the assessment of OSS", Open Source Software 2007, Limerick, June 2007.

[29] Sandro Morasca, "On the use of weighted sums in the definition of measures", In Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics, WETSoM '10, Cape Town, South Africa, May 4, 2010, pages 8-15, New York, NY, USA, 2010. ACM.