An Empirical Investigation of Perceived Reliability of Open Source Java Programs

Luigi Lavazza, Sandro Morasca, Davide Taibi, Davide Tosi
Università degli Studi dell'Insubria
Via Mazzini 5
I-21100 Varese, Italy
[luigi.lavazza, sandro.morasca, davide.taibi, davide.tosi]@uninsubria.it

ABSTRACT

Background: Open Source Software (OSS) is used by a continuously growing number of people, both end-users and developers. The quality of OSS is thus an issue of increasing interest. Specifically, OSS stakeholders need to trust OSS with respect to a number of qualities.
Objective: This paper focuses on the level of trust that OSS stakeholders have in OSS reliability, one of the most important software qualities. The goal of the work reported here is to investigate to what extent the perception of reliability by users depends on objectively measurable characteristics of software.
Method: We collected subjective user evaluations of the reliability of 22 Java OSS products, and then we measured their code characteristics that are generally believed to affect the quality of software. Finally, we carried out a correlational study to predict the perceived level of reliability of OSS based on the measured characteristics of the software code.
Result: We obtained a set of statistically significant quantitative models, collectively called MOSST\REL, which account for the dependence of the perceived reliability of OSS on objectively observable qualities of Java code.
Conclusions: The models we obtained can be used by: 1) end-users and developers that would like to reuse existing OSS products and components, to evaluate the perceived level of reliability of these OSS products that can be expected based on the characteristics of code; 2) the developers of OSS products, who can set code quality targets based on the level of perceived reliability they want to achieve.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *product metrics.*

General Terms
Measurement, Reliability, Experimentation.

Keywords
Open source software, reliability model, object-oriented measures.

SAC'12, March 25-29, 2012, Riva del Garda, Italy.

1. INTRODUCTION

Reliability is one of the most important qualities of any software product. As such, it has been included as one of the top-level characteristics of the international ISO9126 standard [18] in all of its versions. Reliability issues, like several quality issues, are even more important in Open Source Software (OSS) than in Closed Source Software (CSS), as some software stakeholders (i.e., end-users, developers, integrators, project managers, upper management, etc.) may still be somewhat concerned about the reliability (and more generally the quality) of OSS as compared to CSS. While often overestimated, these concerns show that at least some software stakeholders may have issues in trusting OSS and, more specifically, its reliability. This is certainly not surprising, since, like with any other product, stakeholders need to trust OSS and its specific qualities before they can use it and depend on it. Therefore, it is important that OSS stakeholders be able to evaluate and estimate the level of trust that they can have in the reliability of OSS products and components, so they can be confident when they choose OSS software products and components.

In this paper, we report on a study that we carried out in the QualiPSo project [4], funded by the European Union in the 6th Framework Program, whose general goal was to define and implement technologies, procedures, and policies to leverage the OSS development practices to sound and established industrial operations. Among the objectives of QualiPSo was the definition of a model of the trustworthiness of OSS products. To this end, we adopted a goal-oriented and empirical approach, i.e., based on analyzing OSS real-life projects and surveying extensive sets of OSS stakeholders.

One of the key results of QualiPSo was therefore MOSST (Model of Open Source Software Trustworthiness), which is a comprehensive set of statistical models that allow OSS stakeholders to estimate the trustworthiness of OSS products. OSS trustworthiness itself is a broad concept and has several "dimensions," of which trust in OSS reliability is one of the most important ones, along with trust in usability, portability, interoperability, security, documentation quality, etc. [10]. MOSST estimates the level of trust in each of these dimensions based on a number of product- and process-related factors, which play the role of the independent variables of the estimation models, while the trustworthiness dimensions play the role of the

dependent variables. Specifically, we obtained several estimation models for trustworthiness and all of its dimensions.

In this paper, we focus on the results we have obtained for the trust in OSS reliability, for which we have obtained a set of models, which we call MOSST\REL, which allow for the identification, quantification, and assessment of the factors related to the OSS products and processes that have a statistically significant impact on the level of trust in the reliability of OSS products. The models in MOSST\REL *quantitatively* estimate the impact that these factors, in isolation or combined, have on the level of trust in OSS reliability. We used Binary Logistic Regression to derive these models, as we explain in Section 4.

Data collection and analysis were supported by automated software tools, with the goal of making the evaluation of OSS trustworthiness as easy and seamless as possible both during the building of the MOSST model (based on 22 Java OSS products) and afterwards, when MOSST is used by OSS stakeholders. The tool set includes tools from external OSS providers as well as tools developed by the partners during the QualiPSo project. Here, we describe a tool of the MOSST tool set, namely MACXIM, which is a static code measurement and analysis tool. More information about the MOSST toolset can be found in [19].

The remainder of this paper is structured as follows. Section 2 describes the measures and the tool that we used for characterizing OSS code. Section 3 describes the subjective measures and reports about their collection. Section 4 describes our approach to the reported analysis, and explains the usage of Binary Logistic Regression. Section 5 reports and discusses the results of the analysis, and provides some indications on how these results may be used. Section 6 discusses threats to the validity of the reported study. Section 7 accounts for related work, while conclusions and an outline of future work are in Section 8.

2. CODE MEASURES

Currently, several OSS and CSS measurement tools are available to collect information about software code properties. However, most of the existing tools lack in usability and support only a subset of the possibly relevant code metrics. Here, we describe both the tool we used during the collection of the measures for the independent variables of MOSST\REL, and the measures themselves.

In the QualiPSo project [4], we developed MACXIM, a new generation measurement tool which provides static measures of source code via a meta representation of the code abstract syntax tree. MACXIM is released under the OSS LGPL license and can be downloaded from [6].

The latest version of MACXIM –obtained by enhancing an earlier version [5]– supports the most recent versions of Java, thanks to the incorporation of the parser contained in the Eclipse compiler, a component of the Eclipse Core Java Development Tools [9], and supports a wide set of measures.

MACXIM stores a simplified representation of the abstract syntax tree of the source code in a relational database (see Figure 1), thus allowing users to extract several metrics simply via SQL queries. The main advantage of this approach, when compared to similar approaches, consists in the separation of two tasks that are usually deeply interconnected:

- Source code analysis;
- Computation of measures.

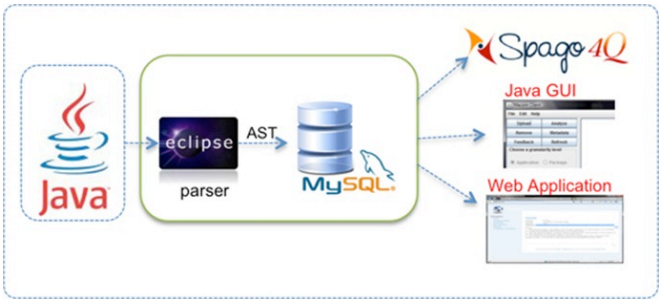

Figure 1. An architectural representation of MACXIM

The separation of these concerns is quite an advantage, since different measures can be implemented on top of the data provided by code analysis. Measurers can thus change or extend the set of supported metrics without necessarily having to modify the part of the program that performs code analysis. The current implementation of the code analyzer is powerful enough to support a wide range of measures, including all the most widely used object-oriented measures, like those by Chidamber and Kemerer [3].

Moreover, MACXIM is developed with a plug-in architecture that allows the integration of external measurement tools: in the current version, we integrated the measurement capabilities of PMD [7] and Checkstyle [8].

For each Java product, MACXIM provides measures aggregated at application, package and class levels, where appropriate. The current release of MACXIM [6] provides 130 measures (of which 58 yielded by the incorporated PMD and Checkstyle code), a selection of which is in Table 1. The complete list and explanation of measures supported by MACXIM can be found in [6].

Table 1. A selection of the measures supported by MACXIM

Size/structure measures	
ELOC (Effective Lines Of Code)	ELOC per Class
ELOC per Interface	Number of Packages
Number of Classes	Number Of Classes Out Of Packages
Number of Abstract Classes	Number of Interfaces
Number of Methods	Number of Public Methods
Number of Private Methods	Number of Protected Methods
Number of Methods per Class	Number of Methods per Interface
Number of Parameters per Method	Number of Attributes per Class
Number of Public Attributes per Class	Number of Classes With Defined Attributes
Number of Classes With Defined Methods	Number Implemented Interfaces per Class
Number of Not Implemented Interfaces	Cyclomatic Complexity (Mc Cabe Index)
Chidamber & Kemerer measures	
CBO (Coupling Between Objects)	LCOM (Lack Of Cohesion Of Methods)
DIT (Depth of Inheritance Tree)	NOC (Number Of Children)
RFC (Response For A Class)	

Figure 2. An example of project analysis

Figure 3. An example of graphical representation of the MOSST\REL model

MACXIM can be used by means of a Web application or a desktop client.

The Web application is the main Graphical User Interface (GUI) that provides direct access to the MACXIM engine. The results of the analysis of a project can be accessed at different granularity levels, like the application, package, class and method levels: for instance, the number of effective LOC is provided for the entire application, for each package, for each class and for each method. Figure 2 illustrates the measures of three releases of a Java project, while Figure 3 reports the graphical representation of a MOSST\REL model provided by the Spago4Q framework [22].

The desktop client provides the same functionalities of the Web application, but communicates with MACXIM via Web services. A detailed description of this interface can be found at [6].

3. STAKEHOLDERS' MEASUREMENT OF TRUST IN RELIABILITY

We carried out a survey in the QualiPSo project [4] to collect OSS stakeholders' evaluations of 22 Java products according to the dimensions of the trustworthiness of OSS products we mentioned in the introduction. These dimensions are believed to be the ones that contribute to trustworthiness the most, based on a previous survey that we carried out among OSS stakeholders [10]. The list of products appears in [11]. In addition, we asked a few questions for profiling and characterizing the OSS stakeholders, including questions on how familiar they were with the products. We used a 1 to 6 ordinal scale, where 1 was the worst and 6 the best evaluation for a specific quality of a product. As for reliability, we asked OSS stakeholders "How reliable is the product?" with the following possible answers:

1 = absolutely not; 2 = little; 3 = just enough; 4 = more than enough; 5 = very/a lot; 6 = completely.

Up to the end of October 2010, we collected 694 questionnaires, which included 1357 evaluations of the following 22 OSS Java products: Ant, Checkstyle, Eclipse, Findbugs, Hibernate, HttpUnit, Jack.CommonsIO, JasperReports, JBoss, JFreeChart, JMeter, Log4J, PMD, Saxon, Servicemix, SpringFramework, Struts, Tapestry, Weka, Xalan, Xerces.

Our sample of respondents included all sorts of OSS stakeholders (i.e., end users, developers, integrators, managers, etc.). The questionnaires were collected at major international events –not necessarily strictly dealing with OSS topics– including: The Apache Conference 2009, The OW2 Conference 2009, XP 2009, OSS 2009, ICSE 2009, CONFSL 2009, QualiPSo Meeting June 2009, ESC 2009, XML Conf 2010, Microsoft Real Code Conf 2010, CONFSL 2010, OSCON 2010, Debian Conf 2010, Open World Forum 2010, Open Opportunity 2010 and fOSSa 2010.

The questionnaire can be found on line at http://qualipso.dscpi.uninsubria.it/limesurvey (QualiPSo survey 2).

In the analysis, we used only products for which we obtained at least 6 subjective evaluations by stakeholders that indicated that they had good familiarity with the product, i.e., for which there were enough people in the sample that could provide us with data for the product to be included in the analysis. Seventeen products turned out to satisfy the selection criteria.

4. ANALYSIS APPROACH

Here, we explain how we built the models that correlate subjective OSS stakeholder evaluations concerning reliability with objective measures of code, to obtain the estimation models which compose MOSST\REL.

To summarize the different evaluations in such a way that meaningful models could be built, we decided to divide the evaluations in positive ones (grade 5 or 6) and negative ones (grade 4 or less). We chose to set the dividing threshold between 4 and 5 because in this way we are able to distinguish really satisfied stakeholders from other stakeholders and also because in the set of responses really few stakeholders gave a 1 or 2 score to a product for reliability. Using a lower threshold (e.g., between 3 and 4) would have blurred the distinction between satisfied and unsatisfied stakeholders.

If we denote the number of satisfied OSS stakeholders by S and the number of unsatisfied OSS stakeholders by U, we can compute the proportion S/(S+U) of satisfied stakeholders for the product, which provides a better indication of the level of trust in the reliability of the product than the absolute number of positive responses. As the absolute numbers of respondents (and therefore of positive and negative responses) for a product depends on a number of different factors like the operating system of the product, the application domain and many others, it makes more sense to use proportions to assess the level of trust in the reliability of a product. This proportion, based on the sample, quantifies the probability that the reliability of that product will be rated positively.

Suppose that we need to assess the reliability of an OSS product that is not included in the set of OSS products of our survey. We do not have a direct evaluation from the field; accordingly, we need to *estimate* the proportion of satisfied users, i.e., the

probability that the reliability of that product will be rated positively. Thus, we need to build a model that estimates the probability that an OSS product will be positively rated.

To this end, we used Binary Logistic Regression (BLR), a kind of regression used to estimate the probability that the dependent variable assumes one out of two possible values and the independent variables are of any type, i.e., discrete or continuous. In our case, the dependent variable values are "satisfied stakeholder" (which is coded with the numerical value 1 in our analysis) and "unsatisfied stakeholder" (which is coded with the numerical value 0 in our analysis). The independent variables are the measures of OSS project described in Section 2. More formally, BLR is based on the following formula

$$Prob(\mathrm{Y} = 1 \mid \mathrm{X}_1 = \mathrm{x}_1, \ldots, \mathrm{X}_\mathrm{k} = \mathrm{x}_\mathrm{k}) = \frac{1}{1 + e^{-z}}$$

which computes the conditional probability that the dependent variable Y assumes value 1, conditioned on the fact that the k independent random variables X_1, …, X_k respectively assume the values x_1, …, x_k. The exponent $z = b_0 + b_1x_1 + b_2x_2 + \ldots + b_kx_k$ is usually called the *logit*. The Logistic Regression formula always provides a value between 0 and 1, which can be interpreted as a probability. The univariate (i.e., with one independent variable) BLR curve –illustrated in Figure 4– is an S-shaped curve that asymptotically tends to hug the y=0 and y=1 points on the y-axis. Also, it can be shown that, when assessing the impact of an independent variable x_i alone (i.e., by keeping all other independent variables fixed), the value estimated by BLR increases when x_i increases if b_i, the coefficient associated with x_i, is positive and decreases if b_i is negative. If b_i is null, then variable x_i has no impact on the estimated probability. The values of the coefficients are estimated with Maximum Likelihood Estimation, based on the data contained in the data set, and statistical significance tests can be used to assess the evidence that b_i is not null, i.e., variable x_i does have an impact on the estimated probability.

BLR has many analogies to linear regression. Unlike the latter, however, BLR does not assume linearity of relationship between the independent variables and the dependent 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 variable. At any rate, the logistic curve, as illustrated in Figure 4, is better for estimating probability values, as it is bounded by 0 and 1, whereas the linear regression function may predict values below 0 or above 1.

We automated the building of the models in MOSST\REL by means of R [1] scripts. Specifically, we built BLR models with up to 3 independent variables. We limited the number of independent variables to a maximum of 3 to avoid possible "overfitting" problems, since the number of OSS products we analyzed is not very large.

Column "logit" contains the logit of the model: so, for instance, the second row indicates that perceived reliability is a function of the average RFC according to the equation

$$Reliability(RFC\_AVG) = \frac{1}{1 + e^{-(1.67 - 0.0557\ \mathrm{RFC}_{\mathrm{avg}})}}$$

which represents the probability that a given product is given a reliability grade ≥ 5.

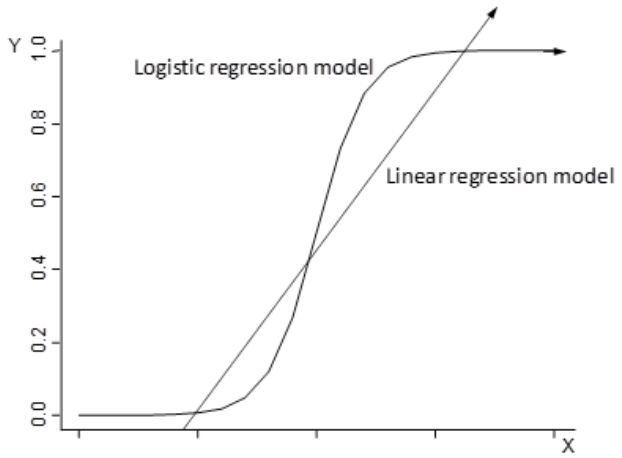

Figure 4. Logistic vs. linear regression models

All the models reported in this paper are significant at the 0.05 level, which is the threshold usually used in Empirical Software Engineering studies.

Table 2 also reports some characteristics of the models:

- R^2_{log} is a measure of goodness-of-fit [2] 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 fourth column in the table reports the number of products that were excluded from the analysis, having been considered outliers (we used Cook's distance to identify outliers).
- MMRE (Mean Magnitude Relative Error), which we used because it is a *de facto* goodness-of-fit indicators typically used in Empirical Software Engineering, indicates the average absolute percent error: the lower, the better.
- Pred(25), another *de facto* goodness-of-fit indicator used in Empirical Software Engineering, shows how many products are within ±25% error with respect to the regression line.
- The % error range indicates the minimum and maximum distance between observed values and estimated ones (always in percentage terms).

5. ANALYSIS RESULTS: MOSST/REL

5.1 Type of results

Table 2 summarizes the statistically significant models found. In this table, each row represents a statistically significant reliability model.

5.2 Discussion of results

A first finding is that perceived reliability appears to be negatively related to the mean number of methods per class. This is a result that could be expected, since a low number of methods per class usually indicates a quite "focused" class, and the fewer the methods, the fewer the opportunities to insert defects in the code. Interestingly, no significant models were found relating perceived reliability to the size of classes in LOC or the number of attributes, etc. This seems to indicate that the perceived reliability of a class does not depend on how big the class is, or how much data it manages, but on "how much the class does."

Table 2. Models of Reliability

ID	logit	R^2_{log}	Outliers	MMRE	Pred25	% error
1	1.67 – 0.1114 num_methods_per_class_avg	0.884	1	16%	76%	-30%..44%
2	1.67 –0.0557 RFC_avg	0.884	1	16%	76%	-30%..44%
3	1.7–0.015 CBO_avg –0.109 num_methods_per_class_avg	0.919	0	14%	82%	-29%..44%
4	1.7-0.015 CBO_avg –0.056 RFC_avg	0.919	0	14%	82%	-29%..44%
5	0.36 –0.0126 e_loc_per_class_avg +0.533 McCabe_index_avg	0.885	1	14%	82%	-25%..50%

Models 2, 3 and 4 in Table 2, involve Chidamber&Kemerer measures [3]. They confirm expectations: large values of CBO and RFC tend to decrease perceived reliability. Again, it is interesting to note the *absence* of models that could have been expected: we found no models involving measures of inheritance (like NOC and DIT); similarly, LCOM does not appear to affect reliability in our sample.

Finally, model 5 deserves some comments, since it accounts for McCabe complexity, but appears somewhat strange, as the coefficient for McCabe complexity is *positive* (i.e., perceived reliability seems to increase with complexity). Actually, complexity appears together with the size of classes in effective LOC: accordingly, the model says that products characterized by small but complex classes are perceived as more reliable. This is not that surprising, and is actually coherent with model 1. For instance, small complex classes are probably better (and more easily) tested, thus increasing perceived reliability.

5.3 Use and novelty of results

MOSST\REL is a set of quantitative models that account for the dependence of OSS perceived reliability 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 reliability of these OSS products that can be expected based on objectively observable characteristics of OSS product and projects;
- developers of OSS products, who can set code quality targets based on the level of reliability they want to achieve.

Unlike existing quality models for OSS, MOSST\REL is built by means of an approach that is valid from a theoretical point of view [20][21] and solid statistical techniques that use evidence coming from OSS stakeholders and the analysis of actual OSS products and projects. Unlike existing software quality models, the models in MOSST\REL are not based on weighted sums of independent variables, in which there is no guarantee that [19]: 1) the independent variables have a provably significant effect on the dependent variable, but the effect can only be conjectured, at best; 2) the weights are not given values through a solid statistical analysis, but may be given largely arbitrary values.

6. THREATS TO VALIDITY

A number of threats may exist to the external validity of a correlational study like ours. We used a so-called "convenience sample," composed of respondents who agreed to answer our questions. The sample of our respondents may not be fully "balanced," however it was not possible to interview several additional people that could have made our sample more "balanced," because they were not available and –most important– because no reliable demographic information about the overall population of OSS "users" is available, so it would be impossible to know if a sample is "balanced" in any way.

In any case, we dealt with motivated interviewees, so this ensured a good level for the quality of responses, and there is no researcher's bias in our survey.

7. RELATED WORK

In [13], seven OSS developers were asked their opinions on a set of OSS projects such as Apache, GNOME, Debian and others. The investigation resulted in the identification of a set of development and quality related problems, including lack of good support infrastructures, lack of development documentation and of adequate configuration management, unsatisfactory management of security updates, and problems in communication and coordination.

An empirical study on correlating the subjective evaluation of "code smells" with objective measures [12] showed that the evaluations given by different roles tend to be different, and that developers' evaluations of code smells do not correlate with the source code objective measures.

We analyzed the correlation between the subjective evaluations of OSS Java products mentioned in Section 3 with the number of rule violations (i.e., potential problems) discovered by PMD [17]. The results showed that the perceived trustworthiness of OSS programs can be correlated to the number of rule violations concerning only *critical* code characteristics.

Stamelos et al. [14] empirically studied the relationship between the size of application components and the delivered quality, measured through stakeholder satisfaction. The results indicated that applications with relatively small average component size seem to work better than applications that are composed of components of larger average size.

Midha [15] analyzed 450 OSS projects from SourceForge and verified that high values of McCabe's Cyclomatic Complexity and Halstead's effort [16] are positively correlated with the number of bugs and with the time needed to fix bugs.

To the best of our knowledge, no other studies have investigated the perception of OSS reliability. So, our work breaks new ground with respect to the studies mentioned above, as it involves the

subjective evaluation of OSS reliability and correlates it with measurable code properties.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated whether it is possible to estimate the level of trust that OSS stakeholders may have in OSS products based on objectively measurable characteristics of OSS projects. We used an empirical approach, in which we interviewed a number of OSS stakeholders and collected data about the characteristics of 22 Java OSS projects. We found a number of statistically significant models (which we collectively call MOSST\REL) that correlate the level of trust in the reliability of OSS products with a number of measurable characteristics of OSS products. The models have fairly high goodness-of-fit and also are consistent with each other in the way in which measurable characteristics influence perceived reliability.

Work remains to be done in this area. Specifically, we plan to

- carry out more interviews, to keep up with the evolution of the field;
- study the evolution of the field;
- include more OSS products in the evaluation;
- include in the dataset products written in multiple languages;
- study the additional dimensions of OSS trustworthiness.

9. ACKNOWLEDGMENTS

The research presented in this paper has been partially funded by the IST project QualiPSo (http://www.qualipso.eu/), sponsored by the EU in the 6th FP (IST-034763); the FIRB project ARTDECO, funded by the Italian Ministry of Education and University; and the project "Metodi e tecniche per l'analisi, l'implementazione e la valutazione di sistemi software" funded by the Università degli Studi dell'Insubria.

10. REFERENCES

[1] 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.

[2] Briand, L. C., Morasca, S., and Basili, V. R. Defining and Validating Measures for Object-Based High-Level Design, *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, September/October 1999.

[3] Chidamber, S. R. and Kemerer, C. F. A metrics suite for object oriented design, *IEEE Transactions on Software Engineering*, vol.20, no.6, pp.476-493, Jun 1994.

[4] The QualiPSo project web site. http://www.qualipso.org

[5] Crisà, A. F., Del Bianco, V., Lavazza, L. A tool for the measurement, storage, and pre-elaboration of data supporting the release of public datasets, *Workshop on Public Data about Software Development – WoPDaSD 2006*, Como, June 10th, 2006.

[6] MACXIM, http://qualipso.dscpi.uninsubria.it/macxim/.

[7] PMD, http://pmd.sourceforge.net/

[8] Checkstyle, http://checkstyle.sourceforge.net/

[9] Eclipse Core Java Development Tools, http://www.eclipse.org/jdt/

[10] Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D. A Survey on OSS Product Trustworthiness, *IEEE Software, September/October 2011 (vol. 28 no. 5),* pp. 67-75.

[11] Del Bianco, V., Lavazza, L., Morasca, S., Taibi, Tosi, D. An investigation of the users' perception of OSS quality, *Int. Conf. on Open Source Systems – OSS 2010*, Notre Dame, IN, USA, May-June 2010.

[12] Mantyla, M. V. and Lassenius, C. Subjective Evaluation of Software Evolvability Using Code Smells: An Empirical Study. *Empirical Software Engineering (2006)* 11: 395-431

[13] Michlmayr, M., Hunt, F., Probert, D. Quality Practices and Problems in Free Software Projects, *First International Conference on Open Source Systems,* pp. 24-28, 2005.

[14] Stamelos, I., Angelis, L., Oikonomou, A., Bleris, G. L. Code Quality Analysis in Open Source Software Development. *Information Systems Journal*, 12:43–60, 2002.

[15] Midha, V. Does Complexity Matter? The Impact of Change in Structural Complexity On Software Maintenance and New Developers' Contributions in Open Source Software. *ICIS* 2008.

[16] Halstead, "Elements of Software Science", New York, Elsevier North-Holland, 1977.

[17] Lavazza, L., Morasca, S., Taibi, D., Tosi, D. Predicting OSS Trustworthiness on the Basis of Elementary Code Assessment, *Int. Symposium on Empirical Software Engineering and Measurement – ESEM 2010.* Bozen, September 16-17, 2010.

[18] International Organization for Standardization, ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model.

[19] MOSST: model of Open Source Software trustworthiness. http://www.qualipso.org/mosst-champion

[20] Morasca, S. On the use of weighted sums in the definition of measures, *ICSE Workshop on Emerging Trends in Software Metrics – WETSoM 2010*, Cape Town, South Africa, May 4, 2010.

[21] Morasca, S. A probability-based approach for measuring external attributes of software artifacts, *Int. Symposium on Empirical Software Engineering and Measurement –ESEM 2009*, Lake Buena Vista, Florida, USA, October 15-16, 2009, pp. 44-55.

[22] SPAGO4Q, http://www.spagoworld.org