Università degli Studi dell'Insubria

DIPARTIMENTO DI INFORMATICA E COMUNICAZIONE

PhD dissertation

TOWARDS A TRUSTWORTHINESS MODEL FOR OPEN SOURCE SOFTWARE

Student:
Davide Taibi

Supervisor:
Prof. Gaetano A. Lanzarone

Advisor:
Prof. Sandro Morasca

Reviewer:
Prof. Alberto Sillitti

2010

Executive Summary

Trustworthiness is one of the main aspects that contribute to the adoption/rejection of a software product. This is actually true for any product in general, but it is especially true for Open Source Software (OSS), whose trustworthiness is sometimes still regarded as not as guaranteed as that of closed source products. Only recently, several industrial software organizations have started investigating the potential of OSS products as users or even producers. As they are now getting more and more involved in the OSS world, these software organizations are clearly interested in ways to assess the trustworthiness of OSS products, so as to choose OSS products that are adequate for their goals and needs.

Trustworthiness is a major issue when people and organizations are faced with the selection and the adoption of new software. Although some *ad-hoc* methods have been proposed, there is not yet general agreement about which software characteristics contribute to trustworthiness. Such methods –like the OpenBQR [30] and other similar approaches [58][59]– assess the trustworthiness of a software product by means of a weighted sum of specific quality evaluations. None of the existing methods based on weighted sums has been widely adopted. In fact, these methods are limited in that they typically leave the user with two hard problems, which are common to models built by means of weighted sums: identify the factors that should be taken into account, and assign to each of these factors the "correct" weight to adequately quantify its relative importance.

Therefore, this work focuses on defining an adequate notion of trustworthiness of Open Source products and artifacts and identifying a number of factors that influence it to help and guide both developers and users when deciding whether a given program (or library or other piece of software) is "good enough" and can be trusted in order to be used in an industrial or professional context.

The result of this work is a set of estimation models for the perceived trustworthiness of OSS.

This work has been carried out in the context of the IST project QualiPSo (http://www.qualipso.eu/), funded by the EU in the 6th FP (IST-034763).

The first step focuses on defining an adequate notion of trustworthiness of software products and artifacts and identifying a number of factors that influence it.
The definition of the trustworthiness factors is driven by specific business goals for each organization. So, we carried out a survey to elicit these goals and factors directly from industrial players, trying to derive the factors from the real user needs instead of deriving them from our own personal beliefs and/or only by reading the available literature.

The questions in the questionnaire were mainly classified in three different categories: 1) Organization, project, and role. 2) Actual problems, actual trustworthiness evaluation processes, and factors. 3) Wishes. These questions are needed to understand what information should be available but is not, and what indicators should be provided for an OSS product to help its adoption.

To test the applicability of the trustworthiness factors identified by means of the questionnaires, we selected a set of OSS projects, widely adopted and generally considered trustable, to be used as references. Afterwards, a first quick analysis was carried out, to check which factors were readily available on each project's web site. The idea was to emulate the search for information carried out by a potential user, who browses the project's web sites, but is not willing to spend too much effort and time in carrying out a complete analysis.
By analyzing the results of this investigation, we discovered that most of the trustworthiness factors are not generally available with information that is enough to make an objective assessment, although some factors have been ranked as very important by the respondents of our survey. To fill this gap, we defined a set of different proxy-measures to use whenever a factor cannot be directly assessed on the basis of readily available information. Moreover, some factors are not measurable if developers do not explicitly provide essential information. For instance, this happens for all factors that refer to countable data (e.g., the number of downloads cannot be evaluated in a reliable way if the development community does not publish it).

Then, by taking into account the trustworthiness factors and the experience gained through the project analysis, we defined a Goal/Question/Metric (GQM[29]) model for

trustworthiness, to identify the qualities and metrics that determine the perception of trustworthiness by users.

In order to measure the metrics identified in the GQM model, we identified a set of tools. When possible, tools were obtained by adapting, extending, and integrating existing tools. Considering that most of metrics were not available via the selected tools, we developed MacXim, a static code analysis tool.

The selected tools integrate a number of OSS tools that support the creation of a measurement plan, starting from the main actors' and stakeholders' objectives and goals (developer community, user community, business needs, specific users, etc.), down to the specific static and dynamic metrics that will need to be collected to fulfill the goals.

To validate the GQM model and build quantitative models of perceived trustworthiness and reliability, we collected both subjective evaluations and objective measures on a sample of 22 Java and 22 C/C++ OSS products.

Objective measures were collected by means of MacXim and the other identified tools while subjective evaluations were collected by means of more than 500 questionnaires. Specifically, the subjective evaluations concerned how users evaluate the trustworthiness, reliability and other qualities of OSS; objective measures concerned software attributes like size, complexity, modularity, and cohesion.

Finally, we correlated the objective code measures to users' and developers' evaluations of OSS products.

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. Unlike the models based on weighted sums usually available in the literature, we have obtained estimation models [87], so the relevant factors and their specific weights are identified via statistical analysis, and not in a somewhat more subjective way, as usually happens.

Qualitatively, our results may not be totally surprising. For instance, it may be generally expected that bigger and more complex products are less trustworthy than smaller and simpler products; likewise, it is expected that well modularized products are more reliable.

For instance, our analyses indicate that the OSS products are most likely to be trustworthy if:

- Their size is not greater than 100,000 effective LOC;
- The number of java packages is lower than 228.

These models derived in our work can be used by end-users and developers that would like to evaluate the level of trustworthiness and reliability of existing OSS products and components they would like to use or reuse, based on measurable OSS code characteristics. These models can also be used by the developers of OSS products themselves, when setting code quality targets based on the level of trustworthiness and reliability they want to achieve. So, the information obtained via our models can be used as an additional piece of information that can be used when making informed decisions.

Thus, unlike several discussions that are based on –sometimes interested– opinions about the quality of OSS, this study aims at deriving statistically significant models that are based on repeatable measures and user evaluations provided by a reasonably large sample of OSS users.

The detailed results are reported in the next sections as follows:

- Chapter 1 reports the introduction to this work
- Chapter 2 reports the related literature review
- Chapter 3 reports the identified trustworthiness factors
- Chapter 4 describe how we built the trustworthiness model
- Chapter 5 shows the tools we developed for this activity
- Chapter 6 reports on the experimentation phase
- Chapter 7 shows the results of the experimentation
- Chapter 8 draws conclusions and highlights future works
- Chapter 9 lists the publication made during the PhD

Table of contents

Chapter 1

Introduction

Open Source Software (OSS) is a continuously growing movement. To give an idea of the size of the phenomenon, note that according to a report published in August 2006 by the market research group IDC, OSS is used by over 70% of all developers worldwide, and in production at 54% of organizations. Another IDC report, published in December 2009, shows that more than 85% of companies are using OSS. OSS can also boast several success stories: programs like the Apache projects, Netscape/Firefox, Eclipse, Linux, MySQL, and several others are well known and used by a huge number of people worldwide. Nevertheless, there are several areas where OSS was not adopted, at least not as widely as it could be expected. An example is given by so-called desktop environments and office applications. In fact, even in the areas where OSS has been successful, there are several potential users that have not yet adopted OSS.

Reluctance in adopting OSS may be due to several causes. A first reason is that the very concept of OSS is hardly understood [25][26]. People tend to confuse OSS with free software (i.e., software that can be used without paying any fee) and open standards with proprietary disclosed software (like PDF)[25]. Another reason is that it is not obvious how to carry out the cost/benefit analysis, given that the acquisition cost of OSS is usually null. Recently, the concept of Total Cost of Ownership (TCO) has been proposed as a means to evaluate the cost of adapting, managing and maintaining OSS. Nevertheless, the concept of TCO is not widely used, partly because it is not well understood (there are several, often inconsistent, definitions) and partly because there is the suspect that most published TCO evaluations are driven by software vendors who want to convince customers that the commercial option is economically profitable. Finally, deciding the adoption of OSS requires the evaluation of the qualities of candidate OS programs, and their comparison with commercial programs. However,

assessing the qualities of OSS is still a not well consolidated practice. Organizations facing the problem of deciding about the adoption of OSS have hardly any guide for carrying out a well structured comprehensive evaluation.
On the other hand, the producers of OSS cannot rely on clear indication concerning the factors that could determine the success of their products.

1.1 Trust in OSS

Modern society depends on large-scale software systems of astonishing complexity. As the consequences of their possible failure are so high, it is vital that software systems should exhibit a trustworthy behavior.
Trustworthiness is a major issue when people and organization are faced with the selection and the adoption of new software. Although some *ad-hoc* methods have been proposed (see for instance [30]), there is not yet general agreement about software characteristics contributing to trustworthiness.
In general, trustworthiness is a holistic property that encompasses security, safety, and reliability. To define trustworthy software, we can draw upon conventional notions of trust in other contexts. Trust is the reliance by one party on the behavior of another party. Trustworthiness is not a quality that can be claimed without being proved. Trust is a matter of perception and implies finding answer to non-technical questions like "why should people have confidence in my software?" or even "how can I make users confident in my software?" Trust is a relationship that involves two parties, the actual and the expected behavior of software. It is always conditional on the context and operational environment.
People may want to know useful key information about any software before making any commitment to use it and so, when users want to adopt new software, they have to trust it. Usually, during the selection of new software, users start by checking if the selected program does exactly what they want and they collect information about the products from other users. In this respect, the web is clearly an extremely valuable and easily accessible source of information. Many websites record a wide range of users' opinions and comments about several different kinds of products.

Clearly, there are other quality-related factors that should be verified. Measuring trustworthiness is possible only if there are specific attributes to measure. For example, in measuring reliability, there are many useful attributes (such as mean time to failure of hardware or software).
The problem surfaces both in Closed and in OSS, but, while in OSS we can measure the code quality, in Closed Source Software we can only trust the producer company.

In this work, we define a model for OSS trustworthiness and identify, quantify, and assess the quality factors that affect trust in OSS products. This methodology encompasses measure definitions, measurement practices, data analysis, and the actual computation of indicators. The approach is based and takes into account the needs of both OSS developers and users. Therefore, this work focuses on defining an adequate notion of trustworthiness of OSS products and artifacts and identifying a number of factors that influence it to provide both developers and users with an instrument that guides them when deciding whether a given program (or library or other piece of software) is “good enough” and can be trusted in order to be used in an industrial or professional context. In addition, as there are several quality factors that are believed to b related to trustworthiness, such as reliability, interoperability, this research also provides estimation models for them.

1.2 The Approach

Organizations perceive software trustworthiness on the basis of the role that software plays with respect to the organization itself. For instance, an organization may be a software producer, customizer, value adder, etc. To deal with the various aspects of software trustworthiness, we used an organized approach, whose steps are described in Figure 1.

Figure 1. The approach description

The first step focuses on defining an adequate notion of trustworthiness of software products and artifacts and identifying a number of factors that influence it. To this end, we carried out a survey to elicit these goals and factors directly from industrial players.
The survey was conducted via interviews supported by a questionnaire, partially derived from the existing literature. We interviewed several people with various professional roles, to derive the factors from the real user needs instead of deriving them from our own personal beliefs and/or only by reading the available literature.
To test the feasibility of deriving a correct, complete, and reliable trustworthiness model on the basis of these factors, a set of well-known OSS projects have been chosen. Then, we verified the possibility to assess the proposed factors on each project.
Next, we developed the trustworthiness models by using a number of factors as its independent variables and an assessment of trustworthiness by OSS practitioners and users as its dependent variable. Therefore, it was necessary to collect data from OSS practitioners and users about the trustworthiness of existing OSS products.
Finally, the information collected was analyzed to find out whether the factors influence the trustworthiness of the OSS products and artifacts.
The analysis were carried out via a variety of different statistical (e.g., Ordinary Least Squares, Logistic Regression) techniques were used for data analysis, based on the

specific independent and dependent variables involved and the objectives of the data analysis.

1.3 Acknowlegments

The research presented in this dissertation has been partially funded by the IST project Qualipso (http://www.qualipso.eu/), funded by the EU in the 6th FP (IST-034763).

Chapter 2

Literature review on OSS trustworthiness

Although there is a good deal of research work on software trustworthiness, the traditional software trustworthiness assurance mechanisms mainly focus on security and dependability properties of software behavior.

In this section, we discuss the literature we took into account during this work.

2.1 Free software

The term "free software" was coined by Richard Stallman in 1983 when he launched the free software movement [45]. This term define software which user can use for any purpose, study the source code of, adapt to their needs and redistribute. To avoid the ambiguity of the English word "free", and to avoid talking about the impact on freedom of non-free software, people have suggested alternative names. "open-source software" was proposed in 1998 as "a replacement label" for "free software"[46]. Later that same year, Open Source Initiative (OSI) was founded to promote the term as part of "a marketing program for free software"[44]. "Software Libre" was first used publicly in 2000, by the European Commission [47]. The word "libre" means having liberty. This avoids the freedom/cost ambiguity of the word "free". "FOSS" has since been used by others with the meaning of Free/Open-Source Software and was first introduced by the U.S. Department of Defense. "FLOSS" was used in 2001 as a project acronym by Rishab Aiyer Ghosh as an acronym for Free/Libre/Open-Source Software. Later that year, the European Commission (EC) used the phrase when they funded a study on the topic[48]. The L for "libre" was included in the hope that it would clarify that the word

"free" was referred to freedom, not to the price. OSS is software released under a license conforming to the Open Source Definition (OSD), as articulated by the OSI. OSS is computer software whose source code is available under a license (or arrangement such as the public domain) that permits users to use, change, and improve the software, and to redistribute it in modified or unmodified form. It is often developed in a public, collaborative manner. It is the most prominent example of OSS development and often compared to user generated content. "Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in"[44]

2.2 Openness, free software, and open source

Cerri and Fuggetta[49] analyze some topical aspects around the openness concept. They started by arguing that there is a real problem for governments, users, and organizations to understand correctly the meaning of “open” when it is used in three different contexts: Open Standards, Open Formats and Open Source.

Open standards define standard interfaces and requirements of ICT systems and services. They make it possible to have different interchangeable and interoperable products, developed by different software houses, companies, and communities. Unfortunately, the definition of standard has different meanings. Open formats are open standards to store and transmit documents, information, and in general knowledge. Since open formats are open standards applied to data, information, documents, there is the same problem seen before for the word standard. Open source is an approach to manage the development and the distribution of software. Open source means that the user of a software program is able (free) to access the source code of the program, study it, change it, and redistribute it. This is true under the conditions and the limitation expressed by software licenses as we said before.

Cerri and Fuggetta then present a survey of the different meanings of ”Open standards” they found on the Internet (e.g., from Wikipedia, Europa, Bruce Perens’ website). Almost all of the results are compatible with the definition given by Wikipedia, but are not coherent with the version provided by Bruce Perens. After an analysis about the

four different levels of openness (disclosed, concerted, open concerted, open de jure), they face with some surrounding aspects like the correlation between open source and open standards. They also focus on the protection of users' rights in open domains. The main claims they have identified to support the adoption of FLOSS (for all that concerns trustability) are: users can inspect the source code; users can modify the software; different systems can interoperate. They identify as extreme the exclusive adoption of FLOSS as working platform especially for governments, Public Administrations (PAs) and Small and Medium Enterprises (SMEs).

Among other conclusions, on protection of users' rights, they suggest to keep the source code of custom software owned by the producer or maintainer, at least in non exclusive form. The procurer is not bound to the original supplier, and can use, modify, and redistribute the software in the most appropriate way. Again, with respect to software packages, it is reasonable to request that the source code is made available to the customer for inspection and recompilation.

This work comes after a previous one [50], in which Fuggetta tries to analyze some propositions (some of which can also be found in[49]) about FLOSS in order to prove that there are some misleading or false claims. He compares FLOSS and proprietary software for each claim and he concluded that FLOSS is not so revolutionary.

Fuggetta position is probably a bit more leaning towards proprietary software and a classical software engineering model instead of a FLOSS model.

2.3 What does trust mean?

Trust is a complex phenomenon that has been the object of interest in various disciplines. Depending on the approach, trust has been defined in many ways. As an example, trust can be defined as "have confidence or faith in,"[89] "something (as property) held by one party (the trustee) for the benefit of another (the beneficiary),"[90].

Trust is a relationship between people, It involves the suspension of disbelief that one person will have towards another person or idea. Trust is a relationship of reliance, "A trusted party is presumed to seek to fulfill policies, ethical codes, law and their previous promises."[91] Also, in security engineering, a trusted system is a system that is relied upon to a specified extent to enforce a specified security policy.

All these quotes to underline the confusion that exists on defining what "trust" means if applied to FLOSS software and products. Since it is fairly difficult to define trust without a context, then defining trust in a particular topic like FLOSS is a real issue.
However, some assumptions can be done on trust. Cooperative relationships, for example, need to be built on the foundation of trust. Antikainen reports a distinction between affective and cognitive trust. Affective trust derives from an emotional attachment between a trustor and a trustee, while cognitive trust relies on the rational assessment of the target by the trustor [51].

2.4 Trust in communities

Antikainen [51] argues about the correlation between communities' sentiments and trust. She starts assuming that into communities discussions, trust is a key factor, because someone may have an opportunistic behavior and so it may manipulate the public opinion about an OSS product positively or negatively, to damage or to promote it. Also, Antikainen does not forget how trust is a very important factor when organizations and companies are making decision about whether they choose an OSS or not.
She defines trust as "the extent to which a person is confident in and willing to act on the basis of, the words, actions, and decisions of another." Trust requires a relationship between a trustor and a trust target. She analyzes one of the more active communities on the FLOSS world: the Linux Kernel community. She found eight factors which seem to affect trust in the community, ordered by their importance: skills (the most important one), practices, reputation, common goals, sharing information, culture and values, possibility to influence, familiarity.
Nearer to Antikainen work, Hertzum aims to explain the trust value of the relationships between colleagues [34]. Hertzum noticed how it is important and cheap for employees to ask information to colleagues rather than to external sources. Thus, this implies a problem about trustworthiness of received information. The quality and credibility of an object, a person, or a piece of information are not properties inherent in the object, person, or information. Rather, quality and credibility are perceived as properties.

Thus in looking for information of high quality, engineers are looking for information that has the following characteristics:

1. accessible in a way that enables the engineer to form a perception of its quality;
2. perceived to be of high quality.

In relation to human interaction, trust is defined as an emotive issue where the trusted party has a moral responsibility toward the trusting party. To the trusting party, trust involves an assessment of whether the other person possesses the required knowledge and skills and is likely to give a truthful and unbiased account of what he knows. People place trust in each other to varying degrees, depending on numerous situational factors.

It is possible to distinguish four types of trust by means of the evidence on which the trust is founded and with respect to the amount of evidence involved:

- first-hand experience;
- reputation: what third parties have reported;
- simple inspection of surface attributes;
- general assumptions and stereotypes.

Thus, knowing an information source first-hand, or knowing someone who knows it first-hand, provides people with a more solid basis for assessing the trustworthiness of the source.

Assuming that trust may govern cooperative relationships, it is possible to adopt that such a trusted relation needs to exist also between different applications. German explains [52] that almost every OSS application depends upon some other external application to be executed. Thus, if there is a need to evaluate how trustworthy a product is, the assessment should be extended to all of its external dependences. As a matter of fact, one single product may be evaluated as trustworthy, but it can depend upon an external library which is not trusted.

2.4.1 Dependencies in trust analysis

German [52] identifies four reasons because a software package (the minimum unit that can be selected and installed in a system) can depend upon others: build dependency, library dependency, main middle dependency, application dependency. This distinction

affects the weight an un-trusted package can have in determining the trust of the product that has to be evaluated. German also lists three main categories of issues affecting trust on a software A which are related to its dependencies on another package B:

- Package B is not present.
- Package B is present, and performs its expected duties, but its interface is not the one expected by package A.

Possible reason:

- a new version of B is released while there is not a corresponding version of package A.
- Package B is present, but it does not perform as expected.

2.4.2 Trustworthiness, security, and privacy

Hansen defines trustworthiness meaning in respect to security and privacy meanings [53]. More in details, he argues that security and privacy principally can be objectively stated, while trustworthiness strongly depends on the subjective experience and feelings of the user.

The trustworthiness enhanced by disclosure of the source code particularly affects privacy. While other qualities such as integrity or availability can be formulated as "do's" and be validated to some degree by practical experience, privacy requirements are very often "don'ts". The main security goal of privacy is confidentiality, which is clearly a "don’t" (expose information). Such requirements as well as formal proof of "don’ts" can only be validated by disclosure of sources. So this is a great advantage of FLOSS in respect to closed source software.

Hasselbring and Reussner [54] aim to provide a holistic view of trustworthiness in software in an interdisciplinary setting. They identified some attributes trustworthiness consists of:

- correctness: the absence of the improper system states;
- safety: the absence of catastrophic environmental consequences;
- quality of service: that includes three attributes: availability, reliability, performance;

- security: the absence of unauthorized access to a system;
- privacy: the absence of unauthorized disclosure of information.

2.4.3 Dependability and Trustworthiness

In [55] Lawrieand and Gacek present issues raised by the articles, presentations, and discussions concerning OSS, trustworthiness, and dependability at the “Open Source Development Workshop” held in Newcastle upon Tyne, UK, on the 25th & 26th of February 2002. Among other contributions, they underline some key concepts about OSS and trustworthiness. They also note that the terms Trustworthiness and Dependability are equivalent. Trustworthiness is a U.S. term and Dependability is a European term. Some other hints are noticeable. They report some given definitions of trust and trustworthiness, and summarize as follows. Trust and trustworthiness can be different: trust may exist where there is no evidence to justify the reliance placed in a certain system, whereas trustworthiness suggests that there are assurance criteria to justify our confidence in a system. Finally, Lawrie and and Gacek conclude that, to be a dependable and trustworthy system, a computer system needs to include certain attributes such as security, reliability, availability. One interesting point that becomes clear is that the stereotypical view of the FLOSS "Bazaar" model is not as chaotic and ad-hoc as it first appears.

At last, Bernstein in [36] analyzes how rarely trustworthiness is considered by software designers. They more often consider schedules, costs, performances, requirements. He does not distinguish FLOSS from closed source. Simply, his attention is focused on the trustworthiness issue. He complains with the lack of interest around trustworthiness and he would be pleased if there were laws that require every product to have a named Software Architect and a Software Project Manager, which control the trustworthiness of the product and of the development process. Trustworthiness is a holistic property, encompassing security, safety and reliability. It is not sufficient to address only one or two of these diverse dimensions, nor is it sufficient to simply assemble components that are themselves trustworthy. Integrating the components and understanding how the trustworthiness dimensions interact is a challenge. Because of the increasing complexity and scope of software, its trustworthiness will become a dominant issue.

Software fault tolerance is at the heart of the building trustworthy software. Trustworthy software is stable software. It is sufficiently fault-tolerant that it does not crash at minor flaws and will shut down in an orderly way in the face of major trauma. Trustworthy software does what it is supposed to do and can repeat that action time after time, always producing the same kind of output from the same kind of input.

Finally, the National Institute of Standards and Technology (NIST) defines trustworthiness as "software that can and must be trusted to work dependably in some critical function, and failure to do so may have catastrophic results, such as serious injury, lost of life or property, business failure or breach of security."

2.5 The OSS adoption

In this section we introduce the literature on the OSS acceptance in the world.

2.5.1 The OSS adoption in Australia

Goode's survey [56] reports an in-depth analysis of a surprising en mass rejection of OSS by Australia's top firms. The survey was made on a sample of 500 companies. The study found that managers rejected OSS because they could not see that it had any relevance to their operations, perceived a lack of reliable ongoing technical support of it, and also seemed to foresee substantial learning costs or had adopted other software that they believed to be incompatible with OSS.

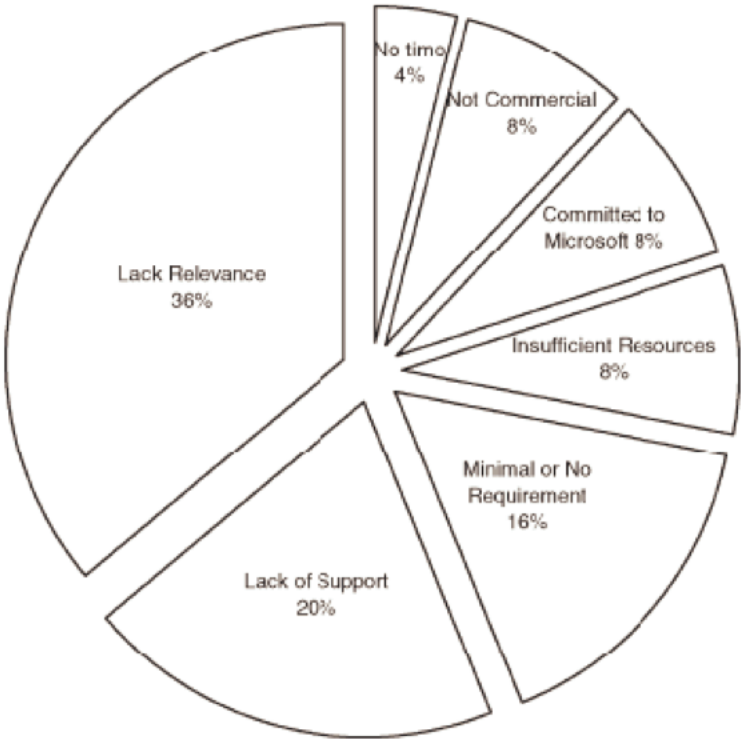

Figure 2: Reasons for rejecting OSS

Figure 2 reports the percentages breakdown of reasons for OSS rejection. The main reasons are the following:

Lack of Relevance. Most respondents had perceived only little relevance of OSS to their business, and could not see any benefits to use it. Some respondents argue that they might be open to adopt it in the future. "One firm argued that they had not adopted

OSS because other nearby firms had rejected OSS. This suggests that, for at least some managers, peer information networks are significant." This also confirms the high relevance that trustworthiness has in peer communications, as already indicated by Hertzum [34].

Lack of Support. The second largest segment cited a lack of conventional and ongoing support as a critical factor in their decision not to adopt OSS products. Here are some quotes from the interviewees. "We think there's a real lack of tangible support."; "We're not interested because it's not a commercial offering."; "We really don't know anything about them and don't want to know. We want someone we can sue when things go to the wall".

Requirement. The next group had evaluated OSS technology but had determined no business requirement for it: "at the moment it's just not feasible - we have no requirement for it". This suggests that managers might be poorly exploring existing software models. Although a huge variety of OSS is proposed to companies, managers would rather stay with their closed source offerings.

Resources. A number of respondents noted a lack of time and resource (i.e., companies and managers do not have enough time and/or resources to invest in OSS) as the barriers to OSS. Summarizing in one sentence, quoted by[43]: "open source software is only free if your time has no value".

• Committed to Microsoft. This is an interesting percentage (8%). The interviewees assert that the committing to Microsoft precludes them for making use of OSS.

2.5.2 The Canadian Collaborative Fact Finding Study

The main aim of the Canadian Collaborative Fact Finding Study [59] was to raise the level of understanding of why and how the OSS paradigm and its products, services and communities are important to Canada, both domestically and internationally. The report tries to fill a lack of information on OSS awareness, initiatives, opinions and attitudes in Canada. The study includes (quoting from the text):

- A scan and review of commercial and non-commercial OSS business models for software, applications and services delivery, to identify recent trends in

Canada, the United States and other major markets, and the most credible forecasts of future trends.

- Industry profiles of key ICT suppliers in Canada who support or supply OSS, applications and/or services.
- An assessment of the engagement of business, government, academia and civil society organizations in Canada toward OSS products, in order to better understand awareness, concerns about support and liability and conditions for acceptance.
- Assessment of the business advantages of alternative open OSS licenses and marketing strategies, from the standpoint of both suppliers and users.
- A synthesis of the issues, opportunities and constraints for Canadian industry and government decision-makers.

The e-Cology Corporation organized the methodology which this study was delivered with. First, they exhaustively surveyed all the Canadian and international literature published on OSS. Subsequently, a workshop on the future of software and OSS in Canada was held in Ottawa. After the workshop, Canadians were invited to answer an online questionnaire. The Corporation obtained more than 180 responses to be analyzed. Finally, 17 Canadian companies active in OSS business had been profiled to produce fact sheets on their products and services. The diagram in Figure 3 presents a composite view (depicted from a technology diffusion model developed by Industry Canada and here adapted and applied to facilitate an high level interpretation of the study results) of the state of OSS in Canada based on the primary research findings.

OSS adoption is framed in the context of its Political, Market and Infrastructure Environmental factors, which determine the starting conditions, and ongoing forces, which influence adoption of open source. Among other results, the study reveals how trust and collaboration are the DNA of OSS. In fact, OSS requires a very deep understanding of the dynamics, conditions and beliefs in the power of collaboration.

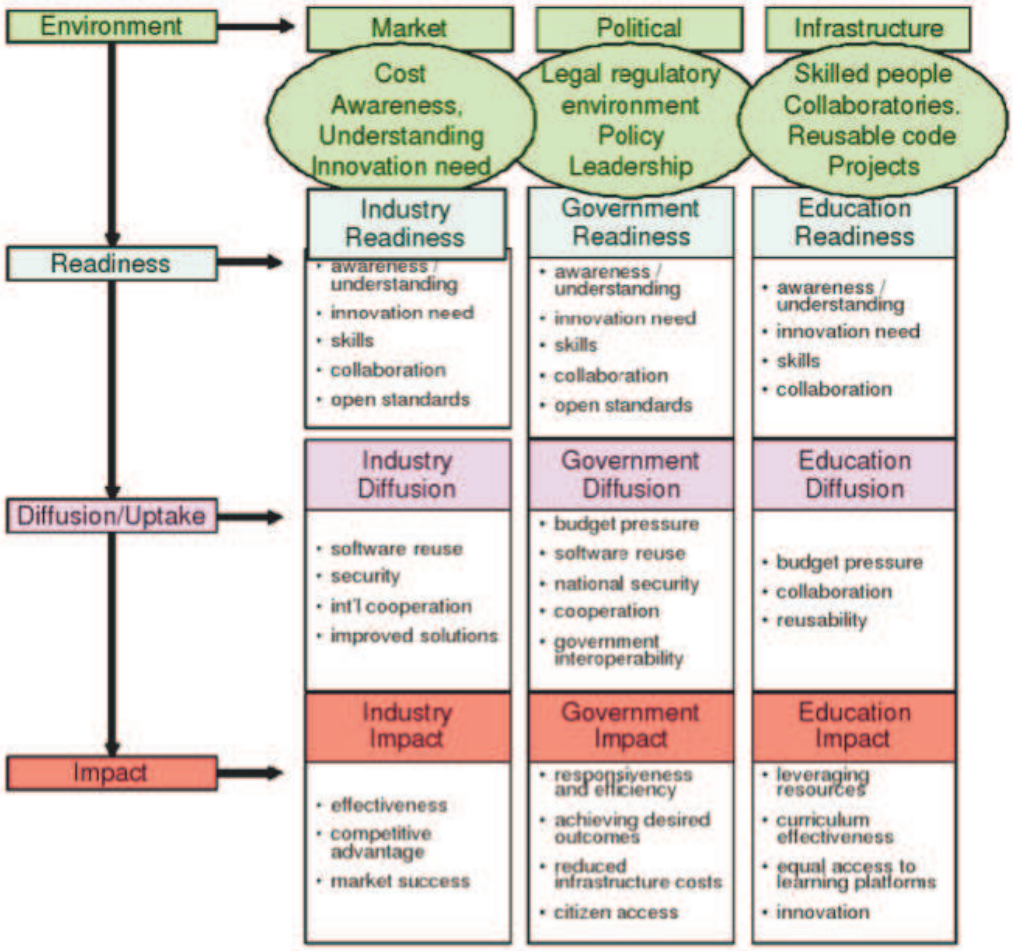

Figure 3: An Overview of OSS with respect to Readiness, Uptake, Impact

2.5.3 The Italian Public Administrations and OSS

The sentiment of Italian PAs on OSS is contrasting. On the one hand, many offices use multiple hardware/software platforms (Windows XP but also MacOS, Ubuntu, SuSe, RedHat or AIX), as desktop, servers, data management, front- end systems. But there is still distrust from PAs towards OSS alternatives. On the other hand, in June 2007, the Ministero per l'Innovazione e le Risorse nella PA has founded the "OpenSource" Commission, composed of several of the main Italian experts. At the same time the Open Source Observatory was started128, hosted by CNIPA [69](National Centre for Informatics in the Public Administration); one of the first objective was to shed light over reuse aspects of software products [75][76]. There is also an initiative fulfilled by the Roma Linux User Group. The project OpenPA [73] aims to spread the OSS knowledge toward PAs and schools. The Regione Piemonte has built the Consorzio per

il Sistema Informativo [73] to promote innovation in PAs using the most recent ITC technologies. This Consortium has eight local offices and 54 members. The Consortium trusts in OSS and it has used OSS for 10 years. During 2006 it has launched an OSS middleware platform, named OASI (Open Available Secure Integrated)[75], to develop and provide services to PAs and users. Ancitel S.p.A. has renewed its platform investing in OSS projects. Ancitel provides technological services to Italian municipalities, having as technological partners ACI IT division and Telecom Italia S.p.A.. ACI itself is supporting six different projects for PAs.

Regione Piemonte is still one of the more active subjects in the adoption of OSS software. There are two remarkable projects. Strategie Digitali S.r.l. has chosen to use only OSS for its services and products [76]. They aim to reach a more extended ROI, to have a social feedback, to reduce the "digital divide". Companies and PAs can use spared money thanks to the non-existent cost for OSS licenses investing them towards education, personalization, information updating, and evolution. The other project, named OSS Piemonte and funded by Regione Piemonte, gathers a set of companies which collaborate to achieve the objective of using OSS solutions to provide services and products to their customers.

2.6 FLOSS evaluation models and tools

There is a general uneasiness with FLOSS because of misleading, misunderstandings or completely false opinions around FLOSS and FLOSS world. In this section, we analyze the most important FLOSS evaluation models and tools.

2.6.1 OpenBRR

OpenBRR.org proposes a model named Business Readiness Rating for OSS as an open standard to facilitate the assessment and the adoption of OSS [58].

They point out how, in practice, many software evaluation projects are done *ad-hoc*, without a formal assessment methodology. *Ad-hoc* methods may be incorrect or incomplete in their assessment, and it is extremely difficult to validate the correctness of the evaluation. They suggest that using an open (to promote trust in the assessment process) and standard (to allow common understanding of the assessment ratings)

model to assess software will increase the ease and correctness of evaluation, and accelerate the adoption of OSS. Additionally, FLOSS users can share their assessment result with FLOSS communities.

2.6.2 QSOS

QSOS (Qualification and Selection of Open-Source software) is a free method developed by Atos Origin to allow software qualification by integrating the open source characteristics and software comparisons according to formalized needs requirements of weighted criteria, in order to make a final choice [59].

The general process of QSOS is made up of four interdependent steps, and can be applied iteratively with different granularity to refine each of the four steps. The four steps are the following:

1. Definition. Constitution and enrichment of frames of reference used in the following steps.
2. Evaluation. Evaluation of software made on three axes of criteria: functional coverage, risks for the user and risk for the service provider.
3. Qualification. Weighting of the criteria split up on the three axes, modeling the context.
4. Selection. Application of the filter set up in Step 3 of data provided by the first two Steps.

Atos Origin also developed a tool (named O3S, as for Open Source Selection Software) to apply the QSOS method in a coherent way.

2.6.3 OSMM, Navica

The OSMM (Open Source Maturity Model) is designed to enable organizations to evaluate FLOSS products and understand whether a product can fulfill the organization's requirements [60].

Enterprises, as well as PAs and organizations, are often wondering whether an open source product will satisfy their needs. The OSMM method evaluates a FLOSS product assessing its support, training, documentation, integration and offered services. There are the main requirements an enterprise has to have satisfied in order to adopt a software product. OSMM comes with a recommended minimum maturity scores to give a context to compare to the new evaluations.

2.6.4 OSMM, Capgemini

Capgemini developed an Open Source Maturity Model in seven steps to allow organizations, PAs and enterprises to determine if or which FLOSS product is suitable[61]. The Capgemini OSMM describes how an OSS should be assessed to ensure that the product meets the IT challenges companies face today. Twenty seven FLOSS indicators has been found, either for products and applications.

2.6.5 OpenBQR

OpenBQR (Open Business Quality Rating) is a model developed by Davide Taibi as his thesis project at Univeristà degli Studi dell'Insubria[30]. This model comes as the extension of and, at the same time, the join between OpenBRR and QSOS. It introduces new evaluation criteria and overturns the steps of selecting and of weighting products, starting from the weighting of elements and then, basing on the weight, evaluating which elements have to be scored. OpenBQR aims to be an open, standard, adaptable, complete, simple model.

Unlike other models, OpenBQR firstly assigns a weight for every element considering five indicators areas:

- Product use target
- Internal qualities analysis
- External qualities analysis

- Support availability in time
- Evaluation of functional requisites

With the model a tool named Open BQR Tool is also provided to support users in their comparative analysis.

2.6.6 The Balanced Scorecards

The Balanced Scorecards (BSC) technique was proposed in the early ’90s by Kaplan and Norton [83] as a reaction to the growing awareness that companies could no longer be managed on the basis of financial measures alone. Kaplan and Norton believed that traditional financial measures needed to be supplemented with the measures of the key factors which determined financial success. Therefore, they devised a set of operational measures that could create a balance of emphasis on the desired outcomes and the means of achieving them.

A few years ago, the BSC approach was adapted to IT, in order to provide the IT departments of large companies with a tool to measure in a complete and balanced way the contribution of IT to the main business of the company, thus overcoming the traditional view of IT as a cost.

Here we are concerned with the application of BSC to OSS.

The Balanced Scorecards (BSCs) are a measurement-supported strategic management method.

BSCs were proposed by Robert Kaplan and David Norton for general purpose organizations (i.e., not specifically for Information and Communication Technology organizations). They observed that traditional management-oriented metrics (like the Return On Investment, for instance) were too much centred on a financial view of the productive organizations. In particular, they observed that ROI-like techniques were limited with respect to scope (in that they provided an all-internal view of a company situation) and time (they concerned only the past performance of the company).

In order to get a more complete and effective view of the state of an organization, they proposed to measure, in addition to financial issues:

- The performance with respect to the outside world. Under this respect, customer satisfaction was considered the most representative indicator.

- How well is the company (or organization) equipped to be successful in the future. The ability to innovate, learn and grow is thus considered a fundamental domain of the BSC method.

Finally, Kaplan and Norton identified the need to assess the performance of the internal process, which is directly linked to customer satisfaction and to financial results, and that is where the learning and growth take place.

These additional metrics were meant to provide indications concerning the future financial results, the strategic objectives to address, and to maintain a healthy balance among the various relevant perspectives.

Figure 4 schematically illustrates the four perspectives addressed by the BSCs.

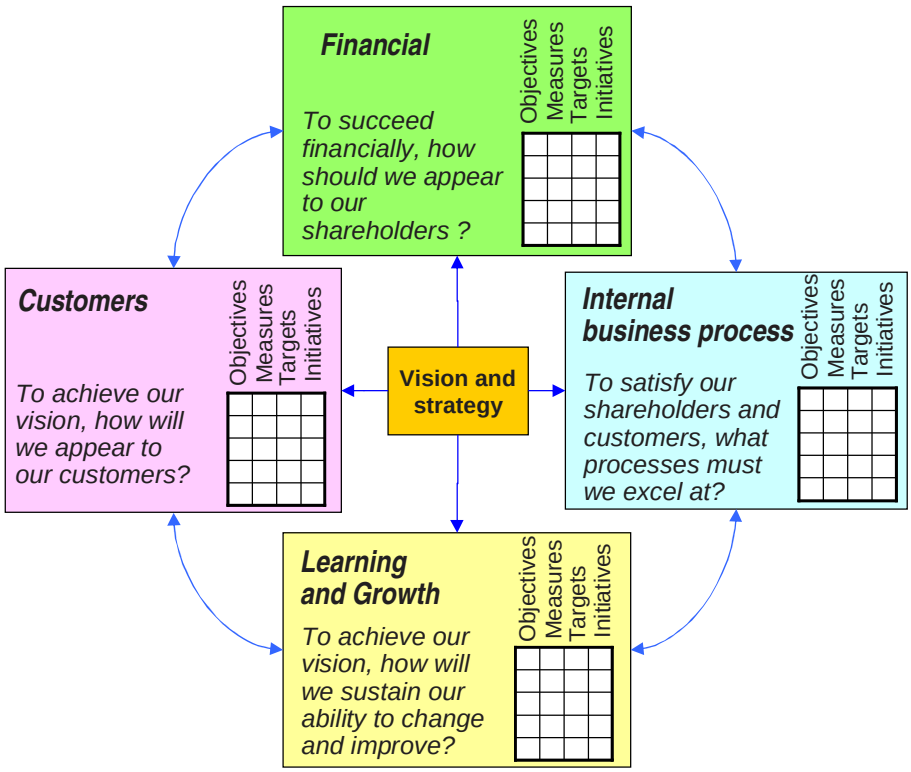

Figure 4. The domains addressed by the BSC.

Considering again the TCO and Open BQR techniques in the framework of the BSC it is quite clear that they do not provide a complete and balanced evaluation: the contribution of TCO is entirely contained in the financial perspective. The Open BQR contribution spans the internal process and growth perspectives, but with a partial coverage. In fact, the Open BQR addresses only the software aspects of the process, since non-software issues, e.g., concerning organization, training, etc., are not taken into consideration.

In practice, the Balanced Scorecards technique suggests that when defining a method for evaluating the trustworthiness of an OSS product, we consider different aspects:

- How well does the OSS product contribute to the business process of the user.
- How well does the OSS product support the user organization in addressing changes and new challenges. Conversely, how well and timely does the evolution of the OSS product match the new requirements of the users.
- What are the costs and benefits of using the OSS product.
- What is the contribution that the usage of the OSS product provides to the perception of the organization from outside (e.g., by customers).

The fact that applying BSC to OSS evaluation is a good idea is demonstrated by the following example.

An organization decides to adopt an OSS product instead of buying the licenses for using an equivalent commercial product.

A first effect of this decision is – quite obviously– that the license costs disappear and, at the same time, the commercial software becomes unavailable.

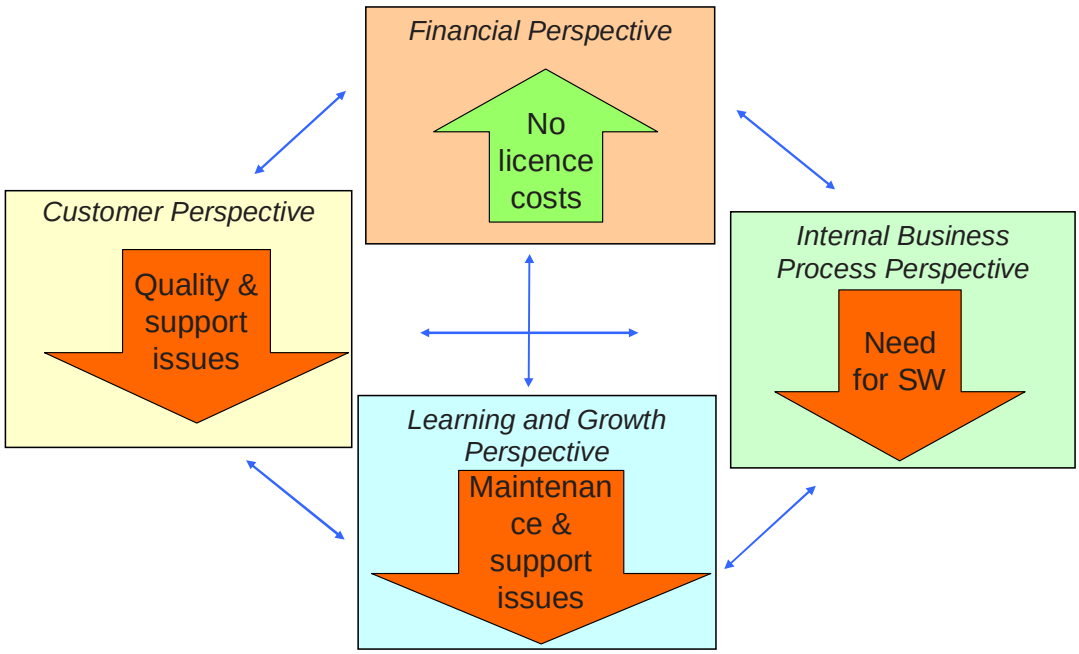

Figure 5. Example: effects of not buying commercial software.

These effects can be precisely classified in the BSC framework (see Figure 5). In practice the beneficial effect of not paying the licence for the software is accompanied by negative effects in all the other sectors.

Note that here we indicate only the qualitative effects of the decision, but according to the BSC we should define proper metrics to perform a quantitative evaluation. The measurement addresses different issues: from costs of licences to the efficiency of the process, to the quality of the products, to the satisfaction of the customers.

The second part of the decision is that OSS is used. Also these effects can be precisely classified in the BSC framework (see Figure 6). The evaluation shows that:

- From the financial point of view OSS is not for free: the organization will have to adapt it, to configure it, and possibly to perform maintenance activities.
- From the point of view of the process the OSS is suitable, and with respect to some issues even better. This is quite common with OSS: having the possibility to instrument the code means better testing of functionality and security.
- From the learning and growth perspective we have a negative effect (the cost of learning) and a positive effect (the knowledge of the software allows faster and better responses to new requirements).
- From the customer perspective, being recognized by the OSS community as a qualified user and/or developer of the OSS increases the reputation of the organization.

Finally, we have to combine the effects illustrated in Figure 5 and Figure 6 to get the complete picture. The measurement of the various aspects will be able to prove that the effects of the decision are balanced, and that the global consequences of the decision match the organization's goal. In this case we would find that the license savings are partially compensated by the need to adapt and configure the software, and that the lack of the (supposedly high-quality) commercial software is compensated by the ability to configure and adapt the OSS in a more timely and effective manner.

Although the situation described above is only an example that cannot be generalized indiscriminately, it illustrates quite clearly the advantages provided by the evaluations based on the BSC.

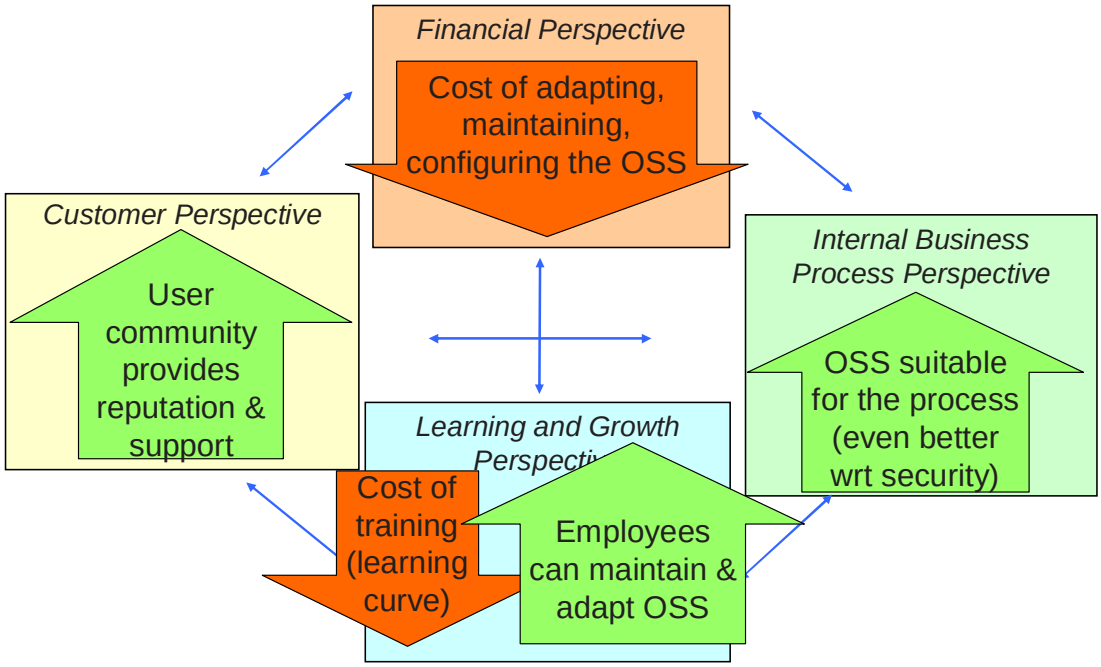

Figure 6. Example: effects of adopting Open Source software.

2.7 Software Quality Models

In this section, we introduce the software quality models taken into account in this work.

2.7.1 The ISO 9126 standard

The first of the ISO 9126 standards[68], namely ISO 9126-1, defines a set of quality characteristics and sub-characteristics that constitute its Quality Model, as shown in Figure 7.

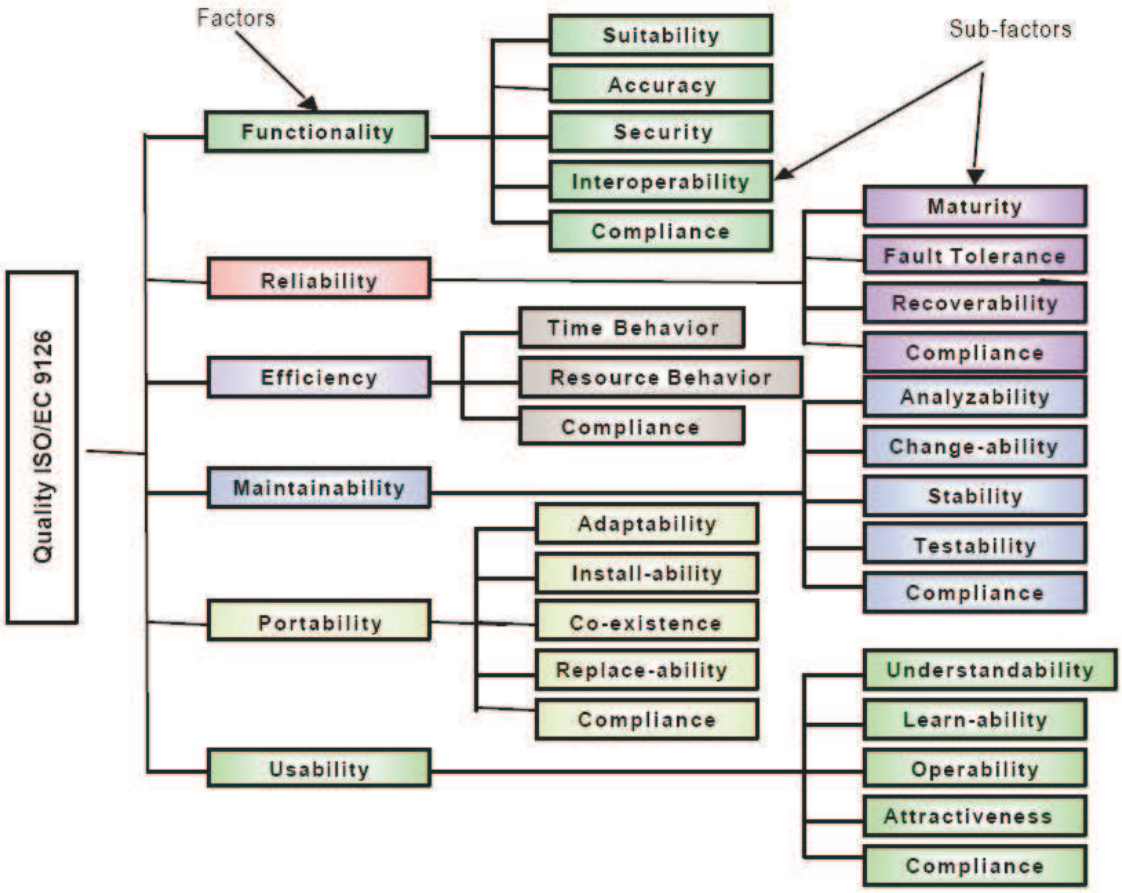

Figure 7. ISO 9126 quality model.

The qualities defined in the ISO 9126-1 standard are the ones that were believed to be the most relevant when the standard was defined.

Recently, there is the tendency to add security and interoperability to the set of ISO 9126 qualities. These qualities are recognized in the new set of ISO 25000 standards. In fact, it should be noted that security and interoperability are already present in the ISO 9126 standard, but only as 'sub-factors' of functionality.

2.7.2 Other models

The second of the basic and founding predecessors of today's quality models is the quality model presented by Barry W. Boehm [79][80]. His models attempts to qualitatively define software quality by a given set of attributes and metrics. Boehm's model is similar to the McCall Quality Model [81] in that it also presents a hierarchical quality model structured around high-level characteristics, intermediate level

characteristics, primitive characteristics - each of which contributes to the overall quality level.

The high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put – the general utility of software. The high-level characteristics address three main questions that a buyer of software has:

- As-is utility: How well (easily, reliably, efficiently) can I use it as-is?
- Maintainability: How easy is it to understand, modify and retest?
- Portability: Can I still use it if I change my environment?

As-is Utility, Maintainability, and Portability are necessary (but not sufficient) conditions for General Utility. As-is Utility requires a program to be Reliable and adequately Efficient and Human-Engineered. Maintainability requires that the user be able to understand, modify, and test the program, and is aided by good Human-engineering. Note that qualities overlap: e.g., communicativeness is part of both Human-engineered and Testability (and hence of Maintainability).

It must be noted that the comparison is made according to the quality hierarchy defined in the models (i.e., the qualities at the highest level in model A are compared to the qualities of model B at the same level). This can be misleading, since the way the hierarchy is defined depends on the aims and points of view of the models' authors. For instance, though Boehm's and McCall's models might appear very similar, McCall's model primarily focuses on the precise measurement of the high-level characteristics "As-is utility", whereas Boehm's quality mode model is based on a wider range of characteristics with an extended and detailed focus on primarily maintainability.

As a consequence, interesting qualities of a software product can be located in different places in the hierarchies. Consider for instance the Understandability: it is present among the qualities of Boehm's model [79] but not in the ISO 9126-1 model. However, it is quite clear that Analyzability and Changeability (which are sub-factors in the ISO 9126 model) depend on the understandability of the product being analyzed or changed. In practice while Understandability is considered among the main qualities by Boehm, it is considered 'only' functional to Maintainability (through the Analyzability and Changeability sub-qualities) by the ISO 9126-1 model. Similar considerations apply to qualities like Documentation, Clarity, etc.

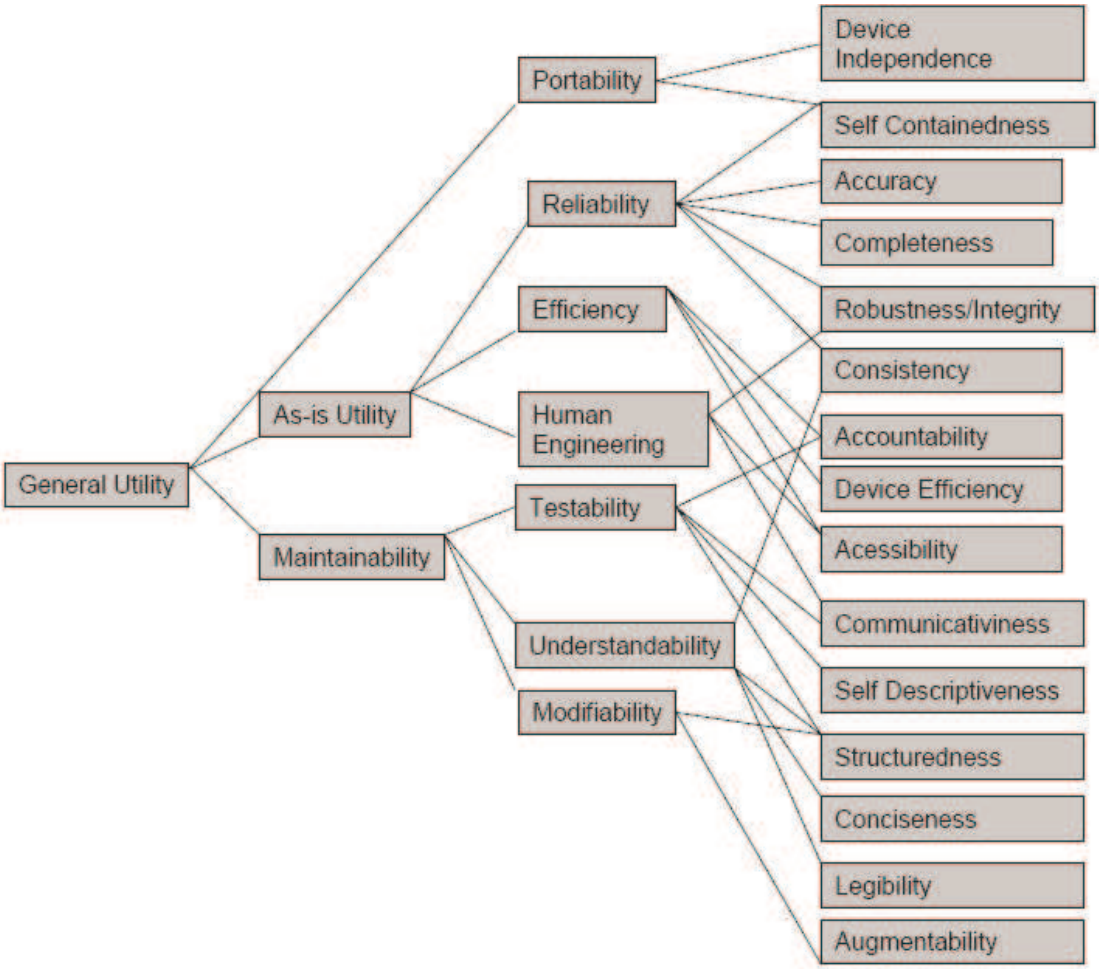

Figure 8. Boehm's quality model.

Chapter 3

The Identification of Trustworthiness Factors

Defining a method for evaluating the trustworthiness of OSS products requires the understanding of the trustworthiness goals of software organizations when they deal with OSS, and the factors to be taken into account when deciding whether a given OSS application (or library, or any other piece of software) is trustworthy enough to be used in an industrial or professional context.

To collect information about these goals and factors, we have used an empirical approach, by surveying 151 OSS users. By "users," we mean all the figures that deal with an OSS product, including developers, integrators, system administrators, product managers, end users, etc. Our survey has primarily focused on investigating goals and factors about the trustworthiness of OSS products in the context of industrial environments.

The objective of the survey was twofold:

- To understand the reasons and motivations that lead software companies to adopt or reject OSS, and, symmetrically, software developers to develop OSS;
- To understand which specific trust factors are taken into account when selecting an OSS product.

3.1 The questionnaire and the survey

The survey was carried out by using a questionnaire, which was developed jointly with the researchers that studied the trustworthiness of OSS processes in the QualiPSo project. Here we only report on the parts of the questionnaire that address the factors affecting the trustworthiness of OSS products.

The questionnaire was developed by taking into account the literature on OSS product trustworthiness and software quality evaluation (see [30][58][59][60]). The questionnaire is a general-purpose one, since:

- It addresses OSS "as-is" usage , as well as its development and its maintenance;
- It is applicable to companies of any size;
- It targets multiple organizational roles (from the inexperienced developer to upper management levels);
- It can be used in all application domains.

The questions in the questionnaire address two main purposes:

- Assessing the current situation, i.e., understanding the current trustworthiness problems, evaluation processes, and factors. The idea is to take a snapshot of the state-of-the-art in OSS trustworthiness according to our interviewees.
- Collecting "wishes," i.e., understanding what kind of information our interviewees would like to have about OSS, even though this information may not be commonly available.

The objective is to understand which additional important trustworthiness factors and measures should be available for an OSS product to help its adoption. This may help OSS developers enhance the quality and type of information provided in OSS repositories, so that users have a better way for assessing the trustworthiness of OSS products.

The questions in the questionnaire are organized according to the types of information we sought to collect, as described below:

- *Personal Information*: used to profile the interviewee, and the company and organizational unit the interviewee belongs to.
- *Role of the Organization in Relation to OSS*: used to understand the specific use of OSS in an organization.
- *Selection Process*: used to understand the process followed when selecting a specific OSS product, even in cases in which the process is completely informal.
- *Economic*: used to understand the main economic drivers behind the choice of a specific OSS product over other OSS products or closed source software.

- *License*: used to identify the most widely used licenses, the problems that can occur when using the available licenses and the characteristics that a good or ideal license should have.
- *Development Process*: even though we investigate the trustworthiness of OSS products, development process aspects need to be taken into account as well, as they may very well influence the trustworthiness of the product.
- *Product Quality Issues*: used to understand the product quality attributes that OSS users take into account when selecting OSS products.
- *Customer Requirements*: used to understand the extent to which customer requirements are influential when choosing an OSS product.

We wanted to collect information in a structured fashion by means of closed-answer questions, as well as additional, less structured information by talking with the interviewees. Thus, each section in the questionnaire also contained open-answer questions, to prompt the interviewees to provide us with additional information. As mentioned above, we also wanted to know the interviewees' wishes, i.e., which additional pieces of information the interviewees would like to have about OSS, even though these pieces of information may not be commonly available.

One of the most significant objectives was to investigate which factors are deemed to be more important during the assessment of OSS products. Thus, we asked the interviewees to provide us with an "importance" value for those factors on a 0 to 10 scale, with 0 meaning totally irrelevant and 10 meaning absolutely fundamental. We clarified that the single ranks have their real meaning only in comparison to other ranks. For instance, giving a value of 6 to interoperability and 4 to size indicates that interoperability is believed to be more important than size, but the individual values 6 and 4 have no meaning in themselves. So, the scale we used is a truly ordinal one.

We carried out the vast majority of the interviews in person and some by phone. We believe this is the most effective way to elicit information and establish an effective communication channel with the interviewees. All the interviews we carried out were individual ones, since we believe that it is important that the interviewees provide their own viewpoint without any sort of conscious or even unconscious interference due to the presence of other people, especially if belonging to the same organization. While

conducting the interviews, the feedback received from the first interviewees allowed us to revise and improve the questionnaire.

The complete questionnaire is listed in Appendix A

3.2 Qualitative and Quantitative Analysis of Factors

We have conducted 151 interviews. The interviewees' nationalities comprise several countries (Brazil, China, France, Germany, Italy, Poland, Spain, United Kingdom and USA). The sample contains interviewees that differ for:

- The organizational roles of the interviewees in their companies;
- The type of the organization of the interviewees;
- How OSS is used by the interviewees.

The sample of interviewees was not determined in advance. A pre-planned sample would have allowed for a more controlled result analysis, but it would also have limited the possibility to add interviewees to the set in an unanticipated manner. We are fully aware that this may have somewhat influenced our results, but:

- It was not possible to interview several additional people that could have made our sample more “balanced,” because they were not available or had no or little interest in answering our questionnaire;
- 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
- We dealt with motivated interviewees, so this ensured a good level for the quality of responses;
- There is no researcher’s bias in our survey, since we simply wanted to collect and analyze data from the field, and not provide evidence supporting or refuting some theory;
- We also investigated the influence of the interviewees' characteristics on the results, to check whether a more “balanced” sample would provide different results.

During the interviews, we kept track of the role of the interviewees. An interviewee could play multiple roles (for example, an interviewee could be both a Developer and a

Project Manager). The roles are fairly equally distributed among Upper managers, Project managers and Developers (see Table 1). OSS experts are clearly underrepresented.

Role	Percentage
Upper manager	30.8 %
Project manager	20.5%
Developer	39.7%
OSS expert	6.4%

Table 1: The interviewees role distribution

Here, we provide a concise analysis of the responses we obtained, with insights gained by statistical analysis. To carry out a sound statistical analysis on the importance ordering of the factors, we used three non- parametric tests that are appropriate with ordinal scales: the Sign Test, the Mann-Whitney Test, and the Wilcoxon Test [40]. We used 0.05 as the statistical significance threshold, as is customarily done in empirical software engineering studies.

Strictly speaking, the ordering of factors according to importance cannot be obtained directly using the arithmetic means of the preference values found on the sample, because:

- The importance of the single factors is measured by an ordinal scale and not by an interval or ratio scale, so the mean may not be used as a sensible central tendency indicator for comparison purposes;
- We wanted to assess the statistical significance of the ordering, that is, we need to know how "reliable" the ordering between two factors actually is.

At any rate, the arithmetic means of the importance rating of factors give an interesting and expressive piece of information, so we provide them alongside the factors ordering for illustrative purposes.

The summary of the results is shown in Table 2, in which the factors are organized according to the sections of the questionnaire. The statistical analysis has allowed us to partition the factors in 8 importance groups from 1-Negligible to 8-Fundamental, and has provided evidence for the existence of an ordering between factors belonging to different groups.

Specifically, it is not possible to find a statistically significance importance ranking among the factors in the same group, while at least one statistically significant importance ranking exists between factors belonging to different groups. For instance, no statistically significant ordering can be found between factors performance and usability, which are both at level 5. However, both factors are believed, in a statistically significant way, to be more important than complexity, which is in group 2.

Factor	Upper management	Project manager	Developer	All respondents	Group
TCO	2%	-6%	-26%	-15%	2
ROI	4%	-1%	-10%	-14%	2
Types of licenses used	6%	10%	5%	1%	4
Availability of tools for developing modifying customizing OSS products	-8%	-6%	8%	1%	4
Availability of best practices on the specific OSS products	-2%	-3%	-12%	-6%	3
Availability of technical documentation / user manual	8%	28%	20%	19%	7
Environmental issues	7%	12%	-4%	4%	4
Availability of training, guidelines, etc.	-13%	-16%	-24%	-16%	2
Mid- / long- term existence of a user community	16%	24%	7%	13%	6
Mid- / long- term existence of a maintainer organization / sponsor	-7%	-2%	-22%	-12%	2
Short-term support	6%	25%	4%	9%	5
Reputation of the OSS vendor	-20%	-4%	-19%	-14%	2
Distribution channel	-85%	-85%	-41%	-49%	1
Programming language uniformity	-19%	-8%	-4%	-9%	3
Existence of a sufficiently large community of users that can witness its quality	10%	16%	2%	12%	5
Existence of benchmarks / test suites that witness for the quality of OSS	-14%	-8%	-20%	-14%	2
Degree to which an OSS product satisfies / covers functional requirements	27%	28%	29%	28%	8
Reliability	21%	23%	27%	25%	8
Performance	4%	12%	9%	10%	5
Usability	14%	18%	9%	13%	5
Maintainability	12%	23%	21%	17%	6
Portability	-6%	1%	3%	-1%	4
Reusability	-4%	8%	9%	5%	4
Size	-51%	-48%	-38%	-39%	1
Complexity	-23%	-22%	-13%	-16%	2
Modularity	1%	8%	10%	8%	4
Standard architecture	4%	-7%	5%	6%	4
Usage od design patterns	-20%	-18%	-11%	-14%	2
Security	1%	27%	9%	13%	5

Standard compliance	-2%	14%	19%	12%	6
Self containedness	-10%	-22%	-4%	-9%	2
Interoperability	21%	14%	21%	19%	7
Localization and human interface	-5%	8%	-9%	-5%	3
Customer satisfaction	24%	33%	10%	16%	7
Interoperability	11%	32%	19%	18%	7
Law conformance	-9%	0%	5%	-1%	4
Standard imposed	-30%	-1%	-9%	-13%	2

Table 2: Trustworthiness Factors (importance for users and group)

The factors reported in Table 2 are analyzed in detail and discussed below.

OSS Selection Process

The majority of interviewees (74.3%) answered that they do not use a formal OSS selection process but, when they were asked further, they admitted that they actually do use an informal selection process, roughly followed in their organizational unit.

None of the interviewees mentioned the use of the existing OSS product evaluation methods that are available in the literature (See Section 2.6). This result shows that, even though some of the methods originated in software companies, there is still a gap to be bridged between these methods and the practice.

Economic Factors

In general, both Return On Investment (ROI) and Total Cost of Ownership (TCO) were expected to be considered very important, but the results do not support this intuition (both TCO and ROI are relegated in group 2!). This result is quite surprising, especially for TCO, since it is usually considered a relevant and direct indicator when comparing costs of OSS products to closed source proprietary products.

Other social and economical factors and issues have been mentioned as important by the respondents. OSS ethics is among the most important ones; OSS supporters consider ethic values at least as important as economic profit. Another important factor is related to integration, since integration cost and effort have been reported to be high, if there is the need to integrate proprietary software. Control of code is also considered a big advantage when choosing OSS products, since unwanted economic dependencies (vendor lock-ins) can be avoided. Ease of acquisition was also mentioned: this is a subtle topic but nevertheless important, since in many organizations spending money to buy software can be a lengthy and complex process. Since there is usually no need to

spend money at the moment of OSS acquisition, OSS is regarded as a faster and easier way to acquire the needed software.

License

Some interviewees identified a large number of licenses that are used in their organization, while the vast majority only named a few. Overall, GPL is considered as the standard license. Most of the interviewees considered licenses and legal issues of medium importance when incorporating an external OSS product in their own products: the factors type of licenses and the factor law are in group 4.

Sometimes, OSS products come with licenses that are not explicitly mentioned. Clarity in the licenses is a common requirement, since it is often difficult to understand what a license allows or forbids. The large number of existing licenses further complicates this issue, since some of the licenses appear to be similar, but turn out not to be fully compatible. This appears to be a relevant hindrance to the adoption of OSS.

Development Process

In this section, we deal with the factors that concern the usage of OSS in SW development. Some interviewees check the quality of an OSS product by testing it thoroughly, even if the factor benchmarks / test suites is regarded as a very low importance factor (it lies in group 2). This low ranking may be partially due to the fact that OSS users do not expect such benchmarks and test suites to be available for OSS products. In some cases, OSS could not be used because the available OSS components were not certified, while the applicable regulations mandated that software be certified.

The availability of documentation is considered very important in the process of selecting OSS: documentation lies in group 7. The environment and the context play significant roles in the selection of OSS, and this is confirmed by the factor environment being in group 4. The analysis indicate that interviewees do pay high attention to the vitality of the user community, in terms of how long it has been existing and, to a lesser degree, to the number of people involved: user community lies in group 6, user community that witnesses quality and short term support (the possibility to have bugs fixed in a short period of time) lies in group 5. Finally, the availability of tools lies in group 4.

Interviewees do not seem to be very interested in the existence of a sponsor organization behind an OSS product: the corresponding factors reputation of vendor and the mid / long term existence of a maintainer organization / sponsor both lie in group 2. The interviewees who are less interested in such an organization are usually willing to carry out the required modifications to the chosen OSS by themselves. Best practices is not believed to be an important factor, even though this factor is somewhat similar to the documentation factors: this factor lies in group 3. Again, this may partly be due to the absence of available best practices for OSS products. Other factors that are not considered important are language uniformity and training / guidelines: both are considered of low importance: language uniformity is in group 3 and training / guidelines is in group 2.

The answers to the open questions revealed some additional facts that are considered relevant at least by some of the interviewees. These indications are qualitative in nature, hence it was not possible to analyze them by means of statistical techniques. The most interesting findings are reported below, since they contribute to complete the picture of the users' perception of the importance of factors that affect OSS trustworthiness.

Some interviewees pointed out that they would like to have more information about the development process (most of the mentioned information is hardly ever available). Part of the requested additional information focuses on the reasons that led to make particular choices, that is, the rationale behind developing the OSS product. The future of a project is considered very important: this is confirmed by some of the additional information requested by the interviewees, such as a detailed roadmap (the planned milestones and releases), a detailed release history (the past milestones and releases), the expected lifetime of the project, and the project's active developers. Information on the actual perception and usage of the project is a piece of information usually not available; a list of real users and data on the popularity of the product are also believed to be worth collecting. Accurate information on the development process is considered useful to assess the quality of the OSS product.

Some interviewees mentioned a heterogeneous set of factors and measures, including: development approach visibility (the best practices, methodologies, tools, etc. used in development), bug lists, quality review process (how quality is taken care of),

benchmarks and certifications (the official certifications obtained by the product, or parts of the product).

Finally it has been suggested that if there is a well defined relationship with a sponsor, such relationship should be expressed clearly and made publicly available.

Product Quality

The top level ISO 9126 qualities [26] were adopted in the questionnaire as a reference for quality factors along with other commonly used qualities.

Quite expectedly, functionality was almost unanimously indicated as one of the most important quality. In fact, factor functional requirements lies in group 8, which is the most relevant factors group. The group contains only another factor: reliability. This is somehow intuitive and reasonable: a product should do what is expected and should do it reliably. There is another quality that is believed very important: maintainability, it lies in group 6. Some other qualities are also considered fairly important, since they all belong to group 5 (performance, usability) and 4 (portability). In conclusion, the ISO 9126 qualities are considered quite important.

The situation is quite different when dealing with code and design quality attributes: in general these are considered of lesser importance. The use of a standard architecture, the production of reusable code and a good modularization are the factors considered most important: modularity, reusability and standard architecture lie in group 4. Note that reusability was not one of the factors originally listed in the questionnaire, but it was frequently mentioned and ranked by our interviewees. The remaining code and design related qualities are considered not important at all: complexity and patterns are in group 2 and size is in group 1. Surprisingly, size is generally believed unimportant by the interviewees, while in the scientific literature [38] size is reported as the most important driver for a number of qualities of industrial interest, such as the development effort and time, and the number of faults.

Interoperability is believed to be very important (it lies in group 7): OSS products are supposed to interact heavily with several other pieces of software. Another factor associated with the issue of interaction is the standard compliance, which lies in group 6.

Security management is a factor that is believed to be very important, it lies in group 6. In addition, security was not one of the factors originally listed in the questionnaire, but it was often explicitly mentioned in answers to open questions of the questionnaire as a very important factor.

Also self containedness is believed to be fairly important in the literature [34], but the answers collected show a different opinion: the factor is in group 2 only. This result can be explained considering that one of the principles in OSS communities is to reuse as much code as possible, even if this creates complexities in the build process and in the management of component dependencies.

Finally, localization and human interface are believed to be of low importance as well: the factor lies in group 3.

Concerning the information on OSS product quality that is usually not available and the interviewees would like to have, it was stated several times that product and design documentation is a major issue: not only it should be available, but it is also required that it is of high quality and accurate. In addition, more attention should be given to several quality aspects: ease of use and ease of installation, certifications, accurate documentation on stability. Stability in particular requires special attention since several OSS products are released when they are not yet stable: this is a common practice in the OSS community that is not suitable for business users.

Customer Requirements

The factors related to customer requirements are believed to be important, because they are mandated by the customers or by the law.

Factor customer satisfaction lies in group 7, showing that it is considered very important (not surprisingly, since it is supposedly directly related to functional requirements and reliability factors). Another factor considered of very high importance is customer's interoperability issues, which lies in group 7 too (again, this factor is somewhat related to standard compliance and directly related to interoperability).

The factor law, i.e., the compliance of OSS with law regulations, lies in group 4, hence it is considered fairly important. One possible explanation for the fact that law does not belong to a higher importance group may be that OSS products are not always subject

to law regulations. The only factor related to customer requirements considered of a lesser importance is standard imposed, which lies in group 2.

Influence of Profiles

Our interviewees' responses are by their very nature subjective, and they may very well depend on the interviewees' roles, responsibilities, type of organization, etc. So, we also investigated whether it was possible to identify commonalities in the responses of interviewees sharing common characteristics. For instance, it may be sensible to expect that interviewees with managerial roles are more interested in economic aspects such as ROI and TCO than interviewees with technical roles. So, we analyzed our data to check for statistically significant associations between our interviewees' profiles and the responses they provided. This also helps overcome the possible lack of "balance" in our sample, since we can check to what degree the interviewees' characteristics may have influenced our results.

Much to our surprise, few such associations seem to exist. Among the most remarkable ones, there seems to be an association between the type of organization (for-profit vs. no-profit) and the importance given to ROI, but not the importance given to TCO. As another example, there seems to be an association between the fact that an interviewee is a project manager and the importance given to the existence of short term support. Other statistically significant associations seem to be somewhat surprising, like, for instance, the fact that developers are not all that interested in the existence of a user community that witnesses the product's quality.

Even though more data may lead to finding additional statistically significant associations, we would have expected that more of them would be detectable even with our sample, if such associations were strong.

3.3 Applying the identified factors to real projects

We identified a set of OSS projects, widely adopted and generally considered trustable, to be used as references. Afterwards, a first quick analysis was carried out, we checked which factors were readily available on each project's web site. The idea was to

emulate the search for information carried out by a potential user, who browses the project's web sites, but is not willing to spend too much effort and time in carrying out a complete analysis. Since the view of trustworthiness factors emerging from the analysis seemed too subjective, it was decided to precisely define measures specifying how to evaluate the OSS characteristics, and how to collect data that could be effectively used in the analysis phase, to be performed according to some statistical methods.

3.3.1 Project selection

The selection of projects addressed different types of software applications, generally considered stable and mature. The complete set of projects comprises 32 products, different with respect to age, implementation language, size of developers and users communities, etc.

Here the criteria used to select a representative set of OSS projects are reported. Projects have a set of characterizing attributes. The selection criteria aimed at:

- Including a reasonably small set of projects.
- Including at least a couple of projects for every possible value of any attribute.

For instance, an attribute is the size of the development team. Four possible values were defined: 0 (inactive project), no more than ten people, up to 50 people, more than 50 people. Therefore, we took care to include at least two projects for each of the four mentioned classes. The complete set of attributes is reported in Table 3.

Attribute	Possible values
Repository	SourceForge, Apache, Java.net, FreshMeat, Rubyforge, ObjectWeb, Free Software Foundation, SourceKibitzer, other
Standalone	Yes / No (Part of a Project family)
Type	Web Server, Operating System, ERP, CSM, …
Developer organization type	Sponsored/foundation, spontaneous, other
Cost	Free; pay for services and features; pay for everything
Size of the development team	0 (abandoned/closed project), 1–10, 11–50, >50
User community size	Small (<51), Medium (51–250), Large (>250)
Programming language	Java, C#, C/C++, scripting languages, Visual Basic, other
Tool support(*)	little use of tools (0-4 tools used); extensive use of tools (5-7)
Innovation	Traditional application (existing before 2003); Emerging application (only proprietary solutions before 2003)
Age	Project started before 1998; between 1998 and 2003; after 2003

Table 3: The projects' attributes

3.3.2 Project Selection Process

The project list was drawn up in three rounds:

- In the first round, we indicated the projects that we considered most important by giving the project name and some useful information easily retrievable from the project's website. That first collection was useful in order to create a project Identity Card. The first set of projects comprised 96 projects (set 3), having different characteristics;
- Once the first set was defined, we restricted our analysis to a subset of 32 projects selected as the most representative ones among the complete set of projects (set 2);
- On this reduced project set, we carried out a quick analysis in order to determine how long a complete analysis would take. Based on the effort required by the previous quick analysis, it was decided to analyze at first 11 projects (set 1) and then proceed with the analysis of the 32 projects of set 2.

Each subset is homogeneous and there are at least two projects for every selection criteria we identified. For instance, the first subset contains 15 projects that are written in C/C++, 15 in Java, and 2 in php; 18 projects have a large community of users, 7 a medium one, and 7 a small community.

Table 4 reports the complete list of 96 projects including some useful information:

- Project Name
- Homepage
- Programming language

Project name	Homepage	Prog. language	Set
Ant	ant.apache.org		1
Apache Httpd	www.apache.org/		2
Apache JMeter	jakarta.apache.org/jmeter	JAVA	3
Apache POI	jakarta.apache.org/poi/index.html	JAVA	1
Asterisk	www.asterisk.org/		1
Axis	ws.apache.org/axis/		1

Boost	www.boost.org/	C/C++	1
Bouncycastle	www.bouncycastle.org/	C#, Java	1
Bugzilla	www.bugzilla.org	Perl	1
BusyBox	www.busybox.net/	C?	3
Canoo WebTest	Webtest.canoo.com	JAVA	1
Centos Linux	www.centos.org	C	2
Checkstyle	checkstyle.sourceforge.net/		1
Cimero	incubator.apache.org/servicemix/cimero-editor.html	Java	2
CruiseControl	cruisecontrol.sourceforge.net	JAVA	1
CUP Parser generator	www2.cs.tum.edu/projects/cup/		1
CVS	cvs.nongnu.org	C/C++	1
Cygwin	cygwin.com/		1
DDD	www.gnu.org/software/ddd/	C/C++	2
Debian	www.debian.org/index.en.html	C/C++	2
drupal	www.drupal.org	php	3
Eclipse Platform	www.eclipse.org/platform/	Java	1
eXo platform	www.exoplatform.org	Java	1
Findbugs	Findbugs.sourceforge.net/		1
GDB	www.gnu.org/software/gdb/gdb.html	C/C++	2
GNU C library	www.gnu.org/software/libc/	C/C++	2
GNU gcc	gcc.gnu.org/	C/C++	2
GNU GRUB	www.gnu.org/software/grub/	C?	1
GNUPlot	www.gnuplot.info/		1
Hibernate	www.hibernate.org/		1
HttpUnit	httpunit.sourceforge.net/		1
JacORB	www.jacorb.org/	Java	1
Jade	jade.tilab.com/	Java	1
Jakarta	jakarta.apache.org/		1
Jakarta commons	jakarta.apache.org/commons/	Java	1
Jakarta Oro	jakarta.apache.org/oro/	Java	1
Jasper	jasperforge.org	Java	2
JasperReports	jasperforge.org/sf/projects/jasperreports	Java	1
JAVA	www.sun.com/software/opensource/java	JAVA	1
Jboss	www.jboss.com	Java	2
Jetspeed	portals.apache.org/jetspeed-1/		1
JfreeChart	www.jfree.org/jfreechart/	Java	1
joomla	www.joomla.org	php	3
JxPath	jakarta.apache.org/commons/jxpath/		1
libxml	xmlsoft.org/	C	1
Linux kernel	www.kernel.org/	C/C++	3
log4j	logging.apache.org/log4j/docs/		1
maxdev	www.maxdev.com	php	1
MediaWiki	www.mediawiki.org/wiki/MediaWiki		1
Mondrian	mondrian.pentaho.org/	Java	2

Mono	www.mono-project.com/Main_Page		1
myFaces	Myfaces.apache.org/		1
MySQL	www.mysql.org	C/C++	3
ncurses	www.gnu.org/software/ncurses/ncurses.html		1
NeuClear	sourceforge.net/projects/neuclear/		1
NeuDist	sourceforge.net/projects/neudist		1
Open Solaris	www.opensolaris.org	C	3
OpenLDAP	www.openldap.org/	C, Bourne Shell Programming	1
OpenPegasus	www.openpegasus.org/		1
OpenSSL	www.openssl.org/	C	2
Pentaho	www.pentaho.com	Java	2
Perl	www.perl.com/		1
phpnuke	www.phpnuke.org	php	1
PMD	pmd.sourceforge.net/		2
PostgreSQL	www.postgresql.org	C/C++	1
Quartz	www.opensymphony.com/quartz/		1
Red Hat Linux	www.redhat.com	C	1
Saxon	saxon.sourceforge.net/		1
ServiceMix	incubator.apache.org/servicemix	Java	3
Spago	spago.eng.it	Java	2
SpagoBI	spagobi.org	Java	3
Speex	www.speex.org/		1
SpiderMonkey	www.mozilla.org/js/spidermonkey/		1
Spring Framework	www.springframework.org/		1
SQLite	www.sqlite.org/		1
Struts	struts.apache.org/		1
Subversion	subversion.tigris.org	C/C++	2
Suse Linux	www.novell.com/linux/	C	1
Talend	www.talend.com	Perl/Java	3
Tapestry	tapestry.apache.org	JAVA	1
TCL/Tk	www.tcl.tk/		1
Termcap	gnuwin32.sourceforge.net/packages/termcap.htm		1
Tomcat	tomcat.apache.org/	Java	2
TPTP	www.eclipse.org/tptp/	Java	2
U-Boot	www.denx.de/wiki/UBoot/WebHome	C, Assembler	2
uClibc	www.uclibc.org/	C?	1
Velocity	velocity.apache.org	JAVA	1
Weka	www.cs.waikato.ac.nz/~ml/index.html	Java	3
Xalan	xalan.apache.org/		1
Xenomai	www.xenomai.org/index.php/Main_Page		1
Xerces	xerces.apache.org/	Java	2
Xml Pull Parser	www.extreme.indiana.edu/xgws/xso		1

	ap/xpp/		
XMLUnit	xmlunit.sourceforge.net/		1
xoops	www.xoops.org	php	1
ZFS	www.opensolaris.org/os/community/zfs/	C	2
zlib	www.zlib.net/	C	1

Table 4: The projects

3.3.3 Project analysis

The first round of our analysis was carried out by looking for the factor information that was readily available by surfing the project sites.

We discovered that most trustworthiness factors are not directly available and they need some specific measures to be specified.

In our experience, the only available information that can be obtained for any project is the number of downloads. Eight factors can be partially evaluated on the basis of the information available on the web sites: the availability of documentation, the type of license used, the long term existence of a maintainer/sponsor, the short term support, the availability of training and guidelines, the programming language uniformity, the distribution channel, and finally the knowledge about the organization that develops the software. Almost every project provides documentation on the website, but most projects do not supply technical and architectural documentation. Moreover, most sites do not provide up to date documentation.

Some factors can be partially evaluated only by carefully digging into the depths of the websites. They are: the availability of tools for developing and modify the software, the existence of benchmarks and test suites, the distribution channel, the self containedness, the interface localization, the availability of a roadmap, and finally the frequency of new product releases.

Other factors could not be evaluated, in some cases because of their subjectivity, in other cases because of lack of information.

Some factors seem to be easy to evaluate, but often the retrieved information is incomplete. For instance, most projects explicitly assert that they adopt a given license,

but one cannot in general be sure that all the sub-projects, components and libraries adopt licenses that are compatible with the license of the main project.

The main areas that could not be covered in that first analysis were those related to the internal quality of the product and related to the user community. In our experience, no website provides data about the user community size, the internal software quality and complexity, or the vitality of the project.

Unfortunately, some pieces of information that are important for our analysis are never highlighted into the project websites. Therefore, we recommend that the leaders of OSS projects who want to publicize the trustworthiness of their products also publish all the useful data. In any case, there are some factors that are inherently difficult to evaluate. For instance, it is quite hard to evaluate the quality of the user manuals. This task may be simplified, for example by collecting feedback from users, but this demands users being aware about the importance of feedback collectors. However, relying on the data provided by users may harden or bias the evaluation task if few users provide feedbacks or if only the satisfied users provide evaluations of the project, respectively. Therefore, web sentiment tools may be used to collect the opinions reported in the websites, blogs and forums, and to analyze the sentiment of the community, thus providing a good approximation of the users' opinion about a given project.

Table 5 summarizes, for each factor, how many project web sites –out of the 32 considered ones (see Table 4)– provided some information to evaluate the factor in their official web sites. We intentionally did not consider some factors like ROI (Return on Investment) and TCO (Total Cost of Ownership) because it was impossible to measure them.

Factor	N° of projects supporting the evaluation of the factor
Type of licenses used	32
Number of downloads	32
Distribution channel	31
Availability of user manual	30
Programming language uniformity	25
Availability of training, guidelines	20
Modularity	18
Availability of best practices on the specific OSS products	17
Human interface language/localization	15
Portability	15
Self containedness	15
Functional requirements satisfaction	11
Standard architecture	11
Availability of tools for developing modifying customizing OSS products.	10
Interoperability	10
Standard compliance	10
Usability	9
Performances	8
Reliability	8
Maintainability	7
Usage of patterns	6
Existence of benchmarks/test suites that witness for the quality of OSS	5
Availability of technical documentation	2
Complexity	0
Customer satisfaction	0
Existence of a sufficiently large community of users that can witness its quality	0
Mid/long term existence of a maintainer organization / sponsor	0
Mid/long term existence of a user community	0
Short term support	0
Size	0

Table 5: Number of projects that provide data about the considered factors

3.3.4 Trustworthiness Factor refinement

The experience gained via the first round of analysis (Section 3.3.1) showed that several factors need to be made more precise and ad-hoc measures need to be specified. The factors that have been identified in Table 2 are too general and unspecific so that it was hard to directly measure them. Therefore, we need new measures in order to assess the factors.

Accordingly, whenever a factor cannot be directly assessed on the basis of the web site information, a new set of proxy-measures needs to be defined. Some factors can be assessed in a simple and direct manner, while others call for specific tools.

In the next subsection, we present the checklist of the refined factors we identified and we discuss the results of the analysis based on the second subset of 11 projects.

3.3.5 Checklist definition

Due to the problems illustrated above, it is necessary to define proxy-measures for some factors in order to simplify the analysis and to obtain results as objective as possible.

In Table 6 both the new measures and also the original ones, defined for OSS product trustworthiness, are reported.

Each project has been evaluated according to the definitions of the measures shown in Table 6. These measures directly refer to the possibility of evaluating a factor by looking into the project website.

Some factors cannot be measured in an objective way, so the evaluation has to be done by ranking the measure coverage by using an ordinal scale. For example, taking into account the Feature List availability, the difference between the availability of a poor free text description (where you can find the features) and a comprehensive feature list will be measured with a subjective scale. Hence, users will be asked to assess whether a description is comprehensive or not.

Other factors are not measurable, unless the developers provide essential information. For instance, the number of downloads cannot be evaluated in a reliable way if the development community does not publish it.

An important output of this work was a as set of recommendations that were given as input to the OSS community. These recommendations are useful both for developers to highlight the trustworthiness factors into their project websites, and also for final users to simply evaluate these factors.

Factor	Measures
Functional requirements satisfaction degree	Availability of: feature list, free text description, release notes, product example/demo
Customer Satisfaction	List of organizations, testimonials and other projects using this software, case studies, usage histories User community satisfaction (according to forums, blogs, mailing lists, newsgroups, magazine/scientific articles)
Interoperability	Communication with other systems supported by suitable mechanisms (SOAP, Web services, protocols, public interfaces, …); Ease of integration with other products and possibility to migrate to other product with little effort
Reliability	Development status, frequency of patches, average bug time solving
Maintainability	Existence of a guide to extend/adapt the OSS product, maintenance releases and architectural documentation Coherent usage of coding guidelines/standard, source code quality and programming language uniformity
Modularity	The product provides plug-in interface(s)
Standard Architecture	Availability of architectural documentation and usage of architectural standard/pattern
Mid Long Term Existence of a User Community	Project Age; Trend of the number of users; Number of patches/releases in the last 6 month; Number of developers involved; Average bug solving time
Availability of technical and user documentation	Availability of: up to date technical/user manual, getting started guide, installation guide, Technical/User related F.A.Q., Technical/user forum and mailing list
Standard Compliance	Any information about standard implemented (like HTTP 1.0, SQL 97...) and coding standards
Existence of a sufficiently large community of users	Number of posts available on forums/blogs/newsgroup and related activity
Performance	Existence of performance tests and/or scenarios, specific performance-related documentation Implementation -Any best practices, concerning design and product construction, aimed at boosting performance.
Type of License	Main and sub license used
Short Term Support	Bug number, bug removal rate, availability of professional services
Availability of tools for developing, modifying, and customizing OSS products	General purpose build tools applicable to the product, build script, built-in customization facility (configuration API, …)
Usability	Detailed feature description and user manual Ease of installation/configuration, ease of use.
Portability	Supported environments, usage of a portable language (like Java), environment-dependent implementation (e.g., usage of hardware/software dependent libraries)
OSS Provider Reputation	Opinion and feedback from other users
Best Practices	Availability of best practices, code examples/tutorials
Programming language uniformity	Number of languages used in the project
Complexity	McCabe complexity number or any related information available on the web site
Human Interface Language Localization	Localization support availability (e.g., are language files provided?)
Self Containedness	Can the product be installed and executed “out of the box” or does it require other software? Are dependencies documented?
Existence of benchmarks/test suites	Availability of test suites/benchmarks, Usage of a test framework (JUnit, DejaGNU,…), results of tests published (on the project

that witness for the quality	site), existence of initiatives to encourage the community to contribute to quality efforts
Mid/long term existence of a maintainer / sponsor	Active maintainer organization / sponsor
Availability of training, guidelines, use cases, tutorial etc.	Up to date training materials, manuals and guidelines available free of charge Availability of official training courses
The distribution media	Source code download; Binaries download Access to the project repository; CD/DVD distribution
Size	Number of Lines of code, source files and functions (or classes and methods, for object oriented code)
Popularity of the product	Number of downloads

Table 6: New criteria for the evaluation of trustworthiness factors

3.4 Trustworthiness factors analysis

The main goal of the analysis is to obtain the information that is quickly available through a project's website.

Some factors have been analyzed, while others need some tools to be developed.

In this section, we analyze all the factors and their measures reported in Table 6 referring to the second set of 32 projects. We correlate users and developers requirements (Table 2) with the actual availability of the trustworthiness factors into web portals. Finally, we provide some guidelines useful to developers of OSS products in order to better highlight trustworthiness factors into their web portals.

Quite noticeably, most of the expected indications involve technical issues. Most of factors are assessed directly or indirectly via the identified measures, others need some tools to be developed.

Taking into account the development related factors, there are some problems in retrieving the majority of the factors. Only around half of the projects have technical documentation, forums and mailing list available, while only less than half of the projects have updated F.A.Q. (Frequently Asked Questions) and technical forum. The same problems appear when we have to check for the availability of best practices and the programming language uniformity. Some factors are often (but not always) available: the availability of training, the availability of tools for modifying, customizing OSS products and the distribution channel. Taking in account the community activity the situation is fairly negative. The dimension of their user community is not measurable unless the websites do not provide the number of partecipant. In the set of projects that has been analyzed, only 2 projects from 32 provide information over the size of their community, and not all projects clearly show

patches and releases; some projects inform only of the number of patches/releases in the last 6 months, others only the of total number and finally a last group shows both.

An interesting result is the availability of several community groups identified via different mailing lists (technical related, user related, translator related…). Unexpectedly, the situation about documentation is quite good from the user's side: almost every project has updated documentation (user manuals, getting started guide and installation guides) and there is a good level of communication between users and developers through forums and mailing lists.

Considering the product quality, there are no factors completely measurable, in some case because tools should be developed for this goal, in other because of the lack of information provided in the project websites. Almost no project provide any information about their performances, maintainability, reliability and complexity. On the other side, half projects show the usage of standard architectures, the availability of interfaces and plug-ins and its interoperability, the possibility to run without any other tools or library and their standard compliance. No project gives any information on code complexity. Some factors are not analyzed because of their subjectivity: all economic and customer related factors, the environmental issue and the reputation of the vendor.

The complete project analysis results are listed in Appendix B.

Chapter 4

Model Building

This section is aimed to defining a set of metrics to capture the trustworthiness of OSS products, a set of metrics to capture the factors that may influence trustworthiness, and a set of models that link these influencing factors to trustworthiness.

The identification and characterization of the qualities reported here are based on the results of Section 3.1, as well as a set of previous relevant contributions to the notion of quality desrcribed in Section 2.6, including:

- ISO 9126 [68];
- the quality models by Barry Boehm [79][80] and McCall [81];
- the balanced scorecards [83];
- the Open BQR [30];
- the TotalCost of Ownership [82].

4.1 A note on the terminology

Unfortunately, different quality models use different terms to indicate qualities and sub-qualities.

For instance, ISO 9126-1 identifies quality factors and sub-factors, while other models talk about (quality) goals. McCall uses the term “quality factor” to indicate top level qualities and “criteria” to indicate the properties that affect the quality factors. In other cases quality criteria indicate the top level qualities.

In this report we use the terminology illustrated in the (meta-)model of trustworthiness given in Figure 9 (using UML as the modeling notation).

Throughout the document the term “quality” is sometimes also used to indicate characteristics/properties of software products. The fact that the term is used in this sense should be clear from the context.

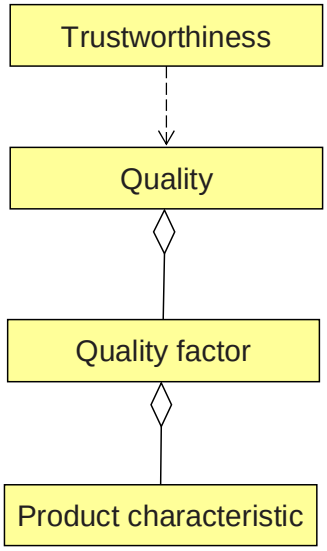

Figure 9: The meta model of trustworthiness

Figure 9 is our meta-model of trustworthiness. It is actually a meta-model since the actual model –which is the final goal of the work reported here– will be defined in the following sections, identifying the actual qualities that affect a product's trustworthiness. Note that this meta-model is not conceptually different from the "Factor-Criteria-Metric" approaches proposed by McCall and Boehm.

4.2 The GQM approach

The Goal/Question/Metric paradigm [29] has been proposed and applied as a systematic technique for developing a measurement programme for software processes and products. GQM is based on the idea that measurement should be goal-oriented, i.e., all data collection in a measurement programme should be based on a rationale which is explicitly documented.

Here we briefly introduce the GQM approach. Readers interested in a more detailed presentation of the GQM can find interesting documentation on the net: [86] is a short paper, while [85] is a complete book (a non-printable version can be found at http://www.gqm.nl/).

The most important concept/product of the GQM paradigm is the GQM plan, produced to define a set of metrics used to reach the organizational goals.

The GQM plan is produced through hierarchical refinements. The goals selected in Step 2 of the GQM process constitute the top level of the GQM plan. Goals are defined in terms of the following entities:

- Object of study: the part of reality that is being observed and studied.
- Purpose: the motivations for studying the object.
- Quality focus: the object characteristics that are considered in the study.
- Viewpoint: the person or group of people interested in studying the object.
- Environment: the application context where the study is carried out.

Each goal is associated with an Abstraction Sheet (Level 2 of the GQM plan) which is composed of four parts:

- Quality focus: it provides additional details on the object characteristics to study.
- Variation factors: this part specifies process and product characteristics that may affect the quality focus.
- Baseline hypotheses: they characterize the current status of the object of study with respect to the quality focus. They describe the initial beliefs of the observer concerning the quality focus described above.
- Impact on baseline hypotheses: this part describes how the variation factors are expected to affect the current state of the object of study.

From the abstraction sheet, a set of questions is derived (Level 3 of the GQM plan). These questions must be answered in order to understand if and how goals have been reached. Questions are a more detailed view of the abstraction sheet.

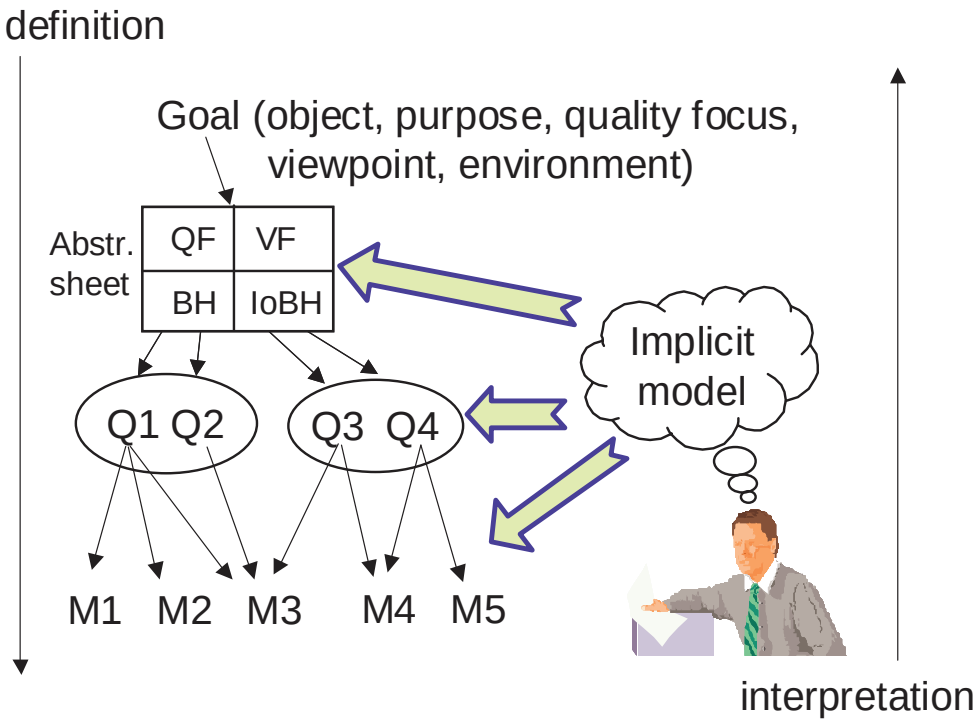

Figure 10. The GQM process: a schematic view.

Finally, from each question a set of metrics is derived (Level 4 of the GQM plan). These metrics are used to collect data, which will be used to answer the questions that have been raised. The process is schematically described in Figure 10.

4.3 Towards the definition of the goal

The GQM plan to be defined addresses the evaluation of trustworthiness of OSS products. However, in order to be able to define the goal, we have to have a closer look at the investigation framework: a first observation is that we need to take into account several variables:

- There are a huge number of OSS products. Of course, their trustworthiness varies from very low to extremely high. We have to take into account that an evaluation must mix up different products. However, we want to create a GQM plan that can be applied to several OSS products, in order to get a model (or a set of models) that represent correctly the trustworthiness of (almost) any OSS product, including the ones not yet released.
- There are many different types of users. Each type has its requirements and needs. These differences generate a number of different points of view. The

perception of a product's trustworthiness depends on the point of view of the user, which depends on the type usage. This is a particularly important issue the same product can be perceived as more or less trustworthy by different users.

- Trustworthiness is a very high-level, abstract quality. As discussed in section 2.7.1 and 2.7.2 it is convenient to identify the different "dimensions" of trustworthiness. In fact, the different perceptions of trustworthiness are determined by the qualities that users seek in the product. By identifying these qualities we will be able to define a flexible model, which can be adapted to different users and uses.

In practice, we have to deal with the situation represented in Figure 11. The figure represents the relations among the elements of the evaluations: every OSS product (the target of the investigation) has one developer (which can be an organization or a community) several users, and several qualities (the "dimensions" of trustworthiness).

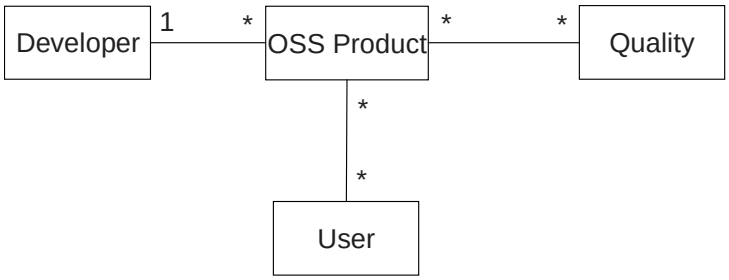

Figure 11. Relations among OSS products, developers, users, and qualities.

Actually, it could be observed that the representation in Figure 11 is a bit abstract; we can get a more detailed representation of relations if we consider how the OSS product is used. According to Section 0 there are two broad types of usage of OSS: the product is used directly (as a development platform, to provide services, etc.) or it is used in a development activity (e.g., it is customized or it is used as part of another software product).

Figure 12 shows the relations among OSS products, developers, users, and qualities when the OSS product is directly used. In this case the user perceives only the external qualities1 of the OSS product. On the contrary, the internal qualities are perceived by the developers.

[1] Qualities are classified as "internal" or "external" as in the ISO 9126 standards.

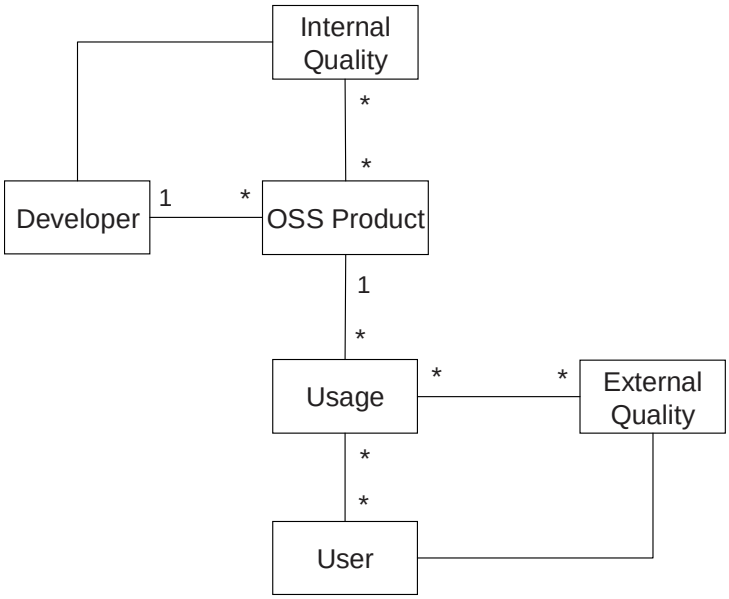

Figure 12. Relations when the OSS product is directly used.

Figure 13 shows the relations among OSS products, developers, users, and qualities when the OSS product is used as part of the development process. In this case the user perceives also (some of) the internal qualities of the OSS product, since the user's development process involves modifying (or examining) the source code. Quite interestingly, the external qualities of the product can be perceived only partially, often only through the result of the development. For instance, when an OSS product has been modified or integrated into another software product, only the performance of the resulting application is perceivable and relevant.

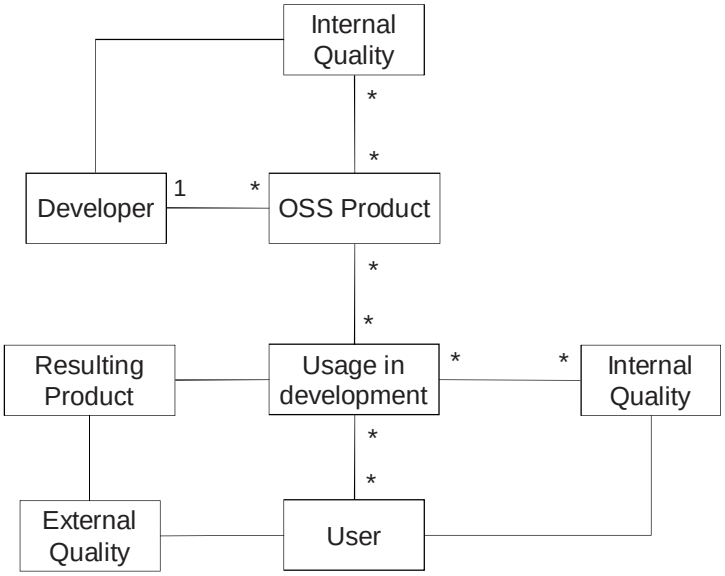

Figure 13. Relations when the OSS product is used as part of development.

For our purpose, we do not need to go into excessive details about the mechanisms that relate the usage of OSS to the perceived qualities. We need just that the usage of the OSS product is clearly related with the perceived quality. To this end, we can merge Figure 12 and Figure 13, thus obtaining the model reported in Figure 14.

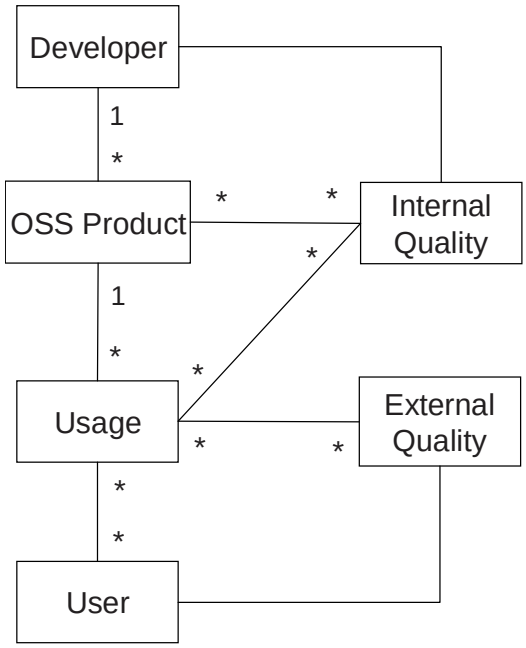

Figure 14. Relations among OSS products, developers, users, usage, and qualities.

According to the observation reported above, the GQM plan has to include questions concerning the OSS product, the users and uses, the developer and the perceived qualities.

Now we have to address an important question: how shall we get the evaluations of the qualities that are relevant to trustworthiness? There are two main options:

- Qualities are evaluated subjectively by users.
- Qualities are evaluated by means of measurement. Since measurement applies to relatively low level properties of the code, for each quality we need to identify its sub-properties, and the associated code properties. The measures of the code are then composed to rate sub-qualities and qualities.

If we chose just one of these options we would not make a big step forward in the evaluation of OSS trustworthiness. In fact, resorting to the subjective evaluation of qualities by users would just replicate –on a single product base– the work already performed in Section 3. On the contrary, just measuring the properties of the code would result in applying a model similar to the one proposed by the ISO 9126 standard.

In order to get the most complete and reliable model of trustworthiness, we intend to perform both subjective (i.e., user dependent) and objective (i.e., measurement-based) evaluations. Then we shall correlate the results of the subjective evaluations with the objective measures, in order to create a model that can provide qualitative indications on the basis of precise and objective quantitative data.

The structure of the GQM goals with respect to the subject of the investigation is described in Figure 15. The Product properties questions concern the measurement of the product as discussed above.

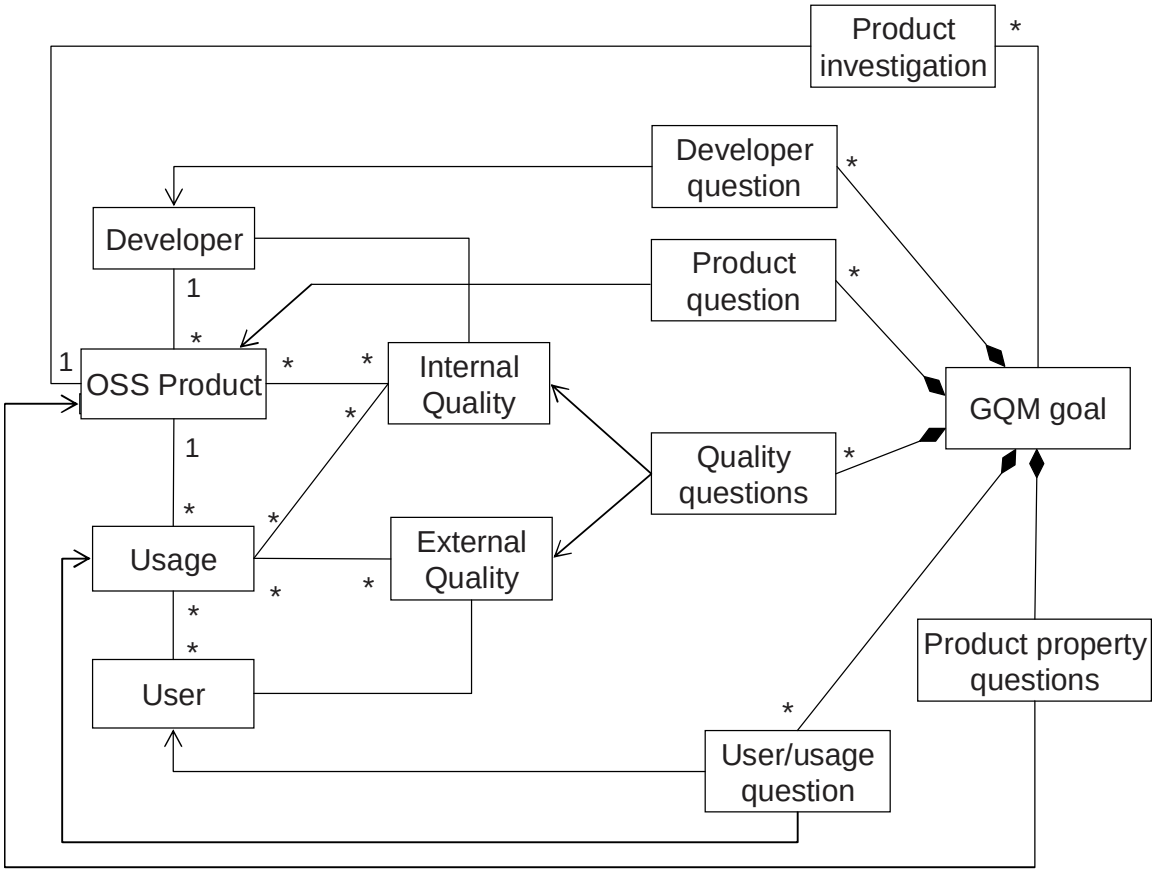

Figure 15. High-level model of the GQM plan and product investigation activities.

Since we are interested in modeling the trustworthiness of OSS products in general (not just of a specific product), the GQM investigation will be applied to several OSS products. This is shown in Figure 15 by making explicit that a single instance of the GQM goal definition is associated with multiple instances of product investigations.

Each product investigation will address:

- One product;

- Its developer (considering that the developer could be an organization involving several individuals);
- Its users. For each user, the following issues will be investigated:
  - Identity and characteristics of the user;
  - How is the OSS product used;
- For each quality, how good is the product, according to the user.

The measurement of the properties of the product.

In order to simplify the structure of the plan and limit the proliferation of roles, the developers of OSS will be treated as users. In fact, the possibility that a user modifies the OSS product makes the difference between developers and this type of users marginal (at least as far as the perception of internal qualities is concerned).

The questions concerning the product and the producer are included in the plan for the purpose of classifying the data.

The rest of the investigation involves an objective and a subjective part:

- The product property questions mentioned in Figure 15 will be carried out mainly through measurement. This evaluation will address features of the product and developer that can be evaluated in a fairly objective way, on the basis of well established Software Engineering knowledge. For instance, features like the complexity of a SW product will be evaluated according to well defined and commonly accepted metrics. Therefore, the results will be fairly objective and independent from who actually performed the measurement and analysis. In other words, we will not need to have product and developers evaluated by different independent teams, as they would provide very similar results.
- For every OSS product there are many users, with different culture, environments, means, and needs. It is thus quite clear that we cannot rely on interviewing a single user (or even a small number of users) in order to understand how users perceive the product trustworthiness. For each product several users will be involved in the evaluation. The indications provided by the users will be inherently subjective. This is perfectly acceptable, or even desirable, since we are building a model that will be usable, for instance, in the process of deciding about the adoption of OSS. Being such decision always

based on partly subjective criteria, it is quite reasonable that the underlying model is itself partly subjective. In this respect it will be necessary to characterize the users, so that in a decision process one can use the data provided by other users having similar characteristics.

4.4 The GQM plan

Before proceeding to the definition of the plan, it is necessary to observe that we are going to use the GQM approach in a slightly unconventional way. In fact, the GQM is usually used to pursue specific goals: e.g., analyzing the testing process in the context of a given organization, or evaluating a specific quality in a specific product. Here we are going to use the GQM to evaluate a whole class of products (OOS products) with respect to a complex quality (trustworthiness), which is determined by several characteristics, according to different users.

We are therefore facing the problem to accommodate these multiple dimensions in a single GQM plan. It is quite clear that the traditional way of using the GQM, i.e., defining a specific goal for every triple <product, quality, user type> is not applicable, since it would lead to an unmanageable number of goals. Actually, it is easy to estimate that in this case we would need no less than one thousand goals, which would require a total of about 100,000 data points for the analysis.

A different strategy has to be adopted, that allows us to limit the number of goals and data, while preserving the effectiveness of the plan.

The GQM plan presented here consists of a single goal. In fact, this is a most general goal that does not strive to focus on specific aspects or situations, at the cost of including a large number of questions and metrics.

For instance, the proposed goal adopts a single generic point of view, which includes both the developers and the different types of users. The characteristics of the developers and users are captured explicitly by means of quality foci within the goal. Similarly, another quality focus will represent the characteristics of the product being analyzed. Finally, we define a quality focus for every quality that contributes to determine the trustworthiness of the product.

Accordingly, the factors studied in Section 3 appear in the GQM plan as quality foci when they are considered to correspond to Qualities in the meta-model of Section 3. Instead, when trustworthiness factors are considered as Sub-qualities or Product characteristics, they are represented as questions. More rarely they appear as variation factors.

4.4.1 The goal

Goal: Analyze OSS for the purpose of evaluating/estimating the trustworthiness from the point of view of OSS users and developers in "business" organizations.

Note that the goal mentions business organizations. In fact, we are interested in the adoption of OSS in environments (like industry and the Public Administration) where the usage of OSS can have a financial/economic impact.

Object:	OSS
Purpose:	evaluate/estimate
Quality:	trustworthiness
Viewpoint:	OSS users and developers
Environment:	"business" organizations (e.g., industry and P.A.)

Table 7. Goal: GeneralTrustworthinessGoal

4.4.2 The dimensions of trustworthiness

In this section the conceptual model of trustworthiness is defined.

According to the findings of Section 3 and to the indications of the literature and the standards, it seems reasonable to define trustworthiness according to the following qualities.

- *As-is utility (quality in use).* This is the quality that the users seek when they want to use the OSS product "as-is", i.e., without changing the code.
- *Exploitability in development.* This quality indicates how easy, efficient, effective, etc. it is to change, maintain, develop the product, possibly to include it into another product.
- *Functionality.* This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the degree to which the considered *OSS product satisfies / covers functional requirements.*

Note that it is in the nature of OSS products that the 'requirements' are expressed by a (potentially heterogeneous) community of users. It is therefore rather difficult to evaluate to what extent the product actually satisfies the requirements, since different users have generally different requirements. This situation induced us to separate from 'functionality' as many qualities as possible, provided that they can be evaluated in a reasonably objective manner. For instance, to some extent interoperability could be considered a functionality, but not all users could be interested in this feature: it is therefore preferable to treat interoperability separately from functionality.

- *Interoperability*. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates how well the OSS product operates in conjunction with (i.e., exchanging data or control information with) other software products.
- *Reliability*. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software not to fail, i.e., to perform its function satisfactorily.
- *Performance*. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to perform its function within given constraints concerning the consumption of resources and time.
- *Security*. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to prevent unauthorized access to program or data.
- *Economy*. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to contribute positively to the financial balance.
- *Customer satisfaction*. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to contribute positively to satisfying the customer (i.e., the final beneficiary of the process in which the OSS product is involved).
- *Developer quality (reliability).* This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates to what extent the developer of the OSS product is reliable. This quality indicates (indirectly) that

we can expect a reasonably good quality of the current version of the product, and regular maintenance and evolution of the product.

Table 8 summarizes the differences among the top-level trustworthiness qualities defined in this Section and the corresponding factors considered in Section 3 and in the ISO 9126 standard.

It is possible to see that trustworthiness qualities match quite closely the trustworthiness factors, with the difference that, while Section 3 was a flat list of quality factors, here we have tried to structure the model of trustworthiness around the qualities that the users are presumably more interested into. For this reasons, we have highlighted the two typical types of usage of OSS products: as-is use and modification/development based on OSS products. These two types of use give rise to specific quality perspectives: As-is utility and Exploitability in development. Since these qualities are specific of OSS products, quite naturally they match only partially the ISO 9126 qualities. As-is utility and Exploitability in development are useful to highlight what qualities are a real concern for users, while others are only accessories.

Trustworthiness Quality	Trustworthiness Factors	ISO 9126
As-is utility	Only sub-qualities present	✓ (Quality-in-use)
Exploitability in development	Only sub-qualities present	Only sub-qualities present
Functionality	✓	✓
Interoperability	✓	✓ (sub-factor of functionality)
Reliability	✓	✓
Performance	(implicitly addressed as part of functionality)	✓ (efficiency)
Security	✓	✓ (sub-factor of functionality)
Economy	✓	(addressed only partly by productivity)
Customer satisfaction	✓	(addressed only indirectly by user satisfaction)
Developer quality	✓	×

Table 8. Trustworthiness qualities and ISO 9126

It is now interesting to evaluate whether any of the factors considered in Section 3 or by ISO 9126 have been neglected in the trustworthiness model. By looking at Table 8 it is possible to see that a large part of the trustworthiness factors do not appear in the

trustworthiness qualities model. Similarly, ISO 9126 Maintainability, Portability and Usability are not present in the models.

Actually, all these qualities have been included in the trustworthiness model, but not at the topmost level: Maintainability and Portability are considered sub-qualities of Exploitability in development, while Usability is considered a sub-quality of the as-is utility quality (alias quality in use). As to the trustworthiness factors, they have all been taken into consideration, at various levels of the model.

Of a few qualities (such as the degree to which an OSS product satisfies/covers functional requirements and the Security) it is possible to provide objective evaluations with respect to a "typical" or "average" usage.

The other qualities can be evaluated both subjectively and objectively.

Our definition of trustworthiness is largely based on the trustworthiness factors, complemented with a few other factors (like Performance) from the ISO 9126. Our definition of trustworthiness appears both sufficiently complete and balanced.

The structure above does not need to be reflected very faithfully in the GQM plan. It is more of a guideline for assuring the completeness of the plan for guiding the data interpretation process. By the way, the GQM supports only three levels (the Quality Focus/Variation Factor, the Question and the Metrics level). Instead our model has several quality/sub-quality levels, a product characteristic level (corresponding to the question level in the GQM) and a measurement level (corresponding to the metrics level in the GQM). Therefore, we will have to flatten the quality/sub-quality levels onto the unique GQM Quality Focus/Variation Factor.

4.4.3 The abstraction sheet of the GQM goal

The abstraction sheet of the GQM goal is illustrated in Table 9. Here we adopted the following naming convention:

- Names initiating by 'ID_' indicate elements concerning the identity of the product, the developer, the users, etc.
- Names initiating by 'Q_' indicate qualities or quality factors.
- Names initiating by 'Q_User' indicate qualities as evaluated by users.

- Names initiating by 'Q_Actual' indicate qualities (or quality factors) as evaluated via objective observations and measurements.

Object	Purpose	Quality Focus	Viewpoint	Environment
OSS	evaluate/ estimate	trustworthiness	OSS users and developers	"business" organizations
Quality Focus ID_OSSproduct ID_User_Info ID_Developer User_Trustworthiness Q_User_As-is utility (quality in use) Q_User_Exploitability_in_development Q_User_Functionality Q_User_Interoperability Q_User_Reliability Q_User_Performance_Resources Q_User_Performance_Time Q_User_Security Q_User_Customer_Satisfaction Q_User_Cost_Effectiveness Q_User_Developer_Quality(reliability) Q_Actual_As-is utility (quality in use) Q_Actual_Exploitability_in_development Q_Actual_Functionality Q_Actual_Interoperability Q_Actual_Reliability Q_Actual_Performance_resources Q_Actual_Performance_Time Q_Actual_Security Q_Actual_Developer_Quality(reliability) Q_Actual_As-is_Usability_Learnability Q_Actual_As-is_Usability_Operability Q_Actual_As-is_Usability_Attractiveness Q_Actual_As-is_Usability_Understandability Q_Actual_As-is_Usability_Compliance Q_Actual_Exploit_in_dev_Modifiability Q_Actual_Exploit_in_dev_Maintainability Q_Actual_Exploit_in_dev_Portability Q_Actual_Functionality_Suitability Q_Actual_Functionality_Accuracy Q_Actual_Cost_Effectiveness Q_Actual_Customer_Satisfaction			**Variation Factors** CodeCharacteristics	
Baseline Hypotheses Baseline hypotheses are given by the results of Section 0			**Impact on Baseline Hypotheses** Not specified. The consequences of variations on the B.H. are as documented in the literature.	

Table 9 The abstraction sheet for the GQM plan.

It is possible to see that the names of several quality foci in the abstraction sheet above start with "Q_user". These quality foci represent the user's perception of trustworthiness. These quality foci are fully expanded into questions and metrics in Appendix C

4.5 Refining the trustworthiness model

The conceptual model of trustworthiness defined in Section 4.4 is schematically represented in Figure 16. It is possible to see that the qualities that determine the trustworthiness of the product are defined only at a rather abstract level.

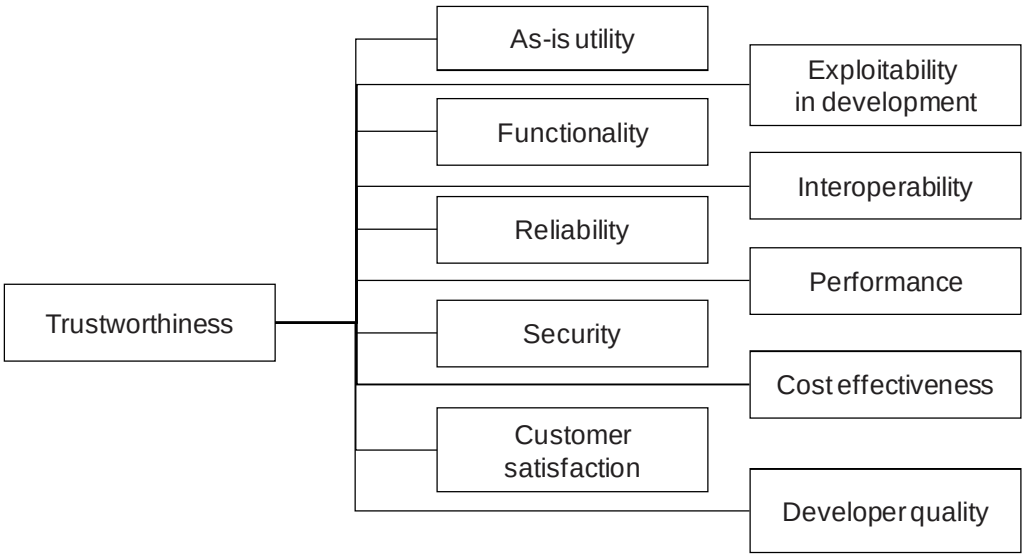

Figure 16. The model of the perceived trustworthiness.

The GQM plan addressing the evaluation of subjective qualities is quite straightforward: for each quality such as as-is utility, reliability, performance, etc. we just ask the users about their own level of satisfaction. On the contrary, the objective evaluation of qualities that affect trustworthiness requires that measurable elementary characteristics are identified. The qualities reported in Figure 6 were therefore refined into a set of observable SW characteristics.

The refined conceptual model of trustworthiness is defined as follows. The complete GQM Plan is described in Appendix C.

As-is utility (quality in use). This is the quality that the users seek when they want to use the OSS product "as-is," i.e., without changing the code. In practice, the quality indicates how well (easily, reliably, efficiently) can the software be used as-is. Accordingly, the quality in use is evaluated on the base of the following sub-qualities:

- Usability: This quality indicates the effort required to use the software, i.e., how easy it is to use the software. It depends on a set of sub-qualities:
  - Understandability: it indicates the users' effort for recognizing the logic of the software and its applicability.
  - Learnability: it indicates the users' effort for learning how to use the application.
  - Operability: it indicates the users' effort for using the application, i.e., to operate and control the software.
  - Attractiveness: it indicates how much the software is attractive for the user. This quality is related to the pleasantness of using the software.
  - Compliance: it indicates to what extent the software adheres to related standards or conventions or regulations in laws and similar prescriptions. For instance the usability of Web applications (e.g., the layout of pages) is subject to regulations.
- Reliability, performance, security, etc. are described below. In general, these qualities apply to both the usage as-is, and to the exploitation in development. Therefore, they are described separately.

Exploitability in development. This quality indicates how easy, efficient, effective, etc. it is to change, maintain, develop the product, possibly to include it into another product. The exploitability of the considered application in the development (possibly of another application) is defined by the following sub- qualities:

- Maintainability: A quality of the software that relates to the effort needed to make specified modifications. According to the traditional classification of maintenance activities, the required changes can be aimed at removing defects, extend the product functionality, or adapt it to environmental changes.

- Modifiability: A quality of the software that relates to the effort needed for modification. Modifiability is very similar to maintainability: we talk about modifiability when the OSS is (re)used in the context of the development of a larger product. You can consider it a sort of adaptation; however, since the OSS is often used as building material, we considered useful to distinguish this type of changes from regular maintenance.

Both maintainability and modifiability are rather complex to evaluate: accordingly, they are characterized by a set of sub-sub-qualities:

- Analyzability: The quality of software that relates to the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified. Analyzability depends largely on how easy it is to understand the program; hence, you can see analyzability as the sum of readability, modularization, documentation, etc.
- Stability: A quality that indicates to what extent software modifications can cause unexpected effects.
- Testability: The quality of software that indicates the effort needed for validating the modified software.

Portability: It indicates how easy it is to transfer software from one environment to another.

- Adaptability: Attributes of software that relate to its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered.
- Installability: Attributes of software that relate to the effort needed to install the software in a specified environment.
- Replaceability: Attributes of software that relate to the opportunity and effort of using it in the place of specified other software in the environment of that software.

Functionality. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the degree to which the considered OSS product satisfies / covers functional requirements. Functionality has two sub- qualities:

- Suitability: It indicates to what extent the software provides an appropriate a set of functions supporting the (stated or implied) user requirements.

- Accuracy: It indicates to what extent the software provides correct results and effects.

Suitability and accuracy complement each other: the software does what it is required to do (suitability), and does it well (accuracy).

Interoperability. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates how well the OSS product operates in conjunction with (i.e., exchanging data or control information with) other software products.

Interoperability can be defined as "The ability of two or more systems or components to exchange information and to use the information that has been exchanged." There are several issues that have to be considered in order to evaluate the interoperability, especially when it is referred to one application (i.e., we are dealing with the potential interoperability of the product with an unspecified other piece of software). We propose to evaluate interoperability according to the following sub-qualities:

- Data exchangeability: It evaluates how easy it is for the considered application to exchange data with other applications. It takes into consideration common/compatible data communication protocols, data representation models and standards.
- Control exchangeability: It evaluates how easy it is for the considered application to exchange control data with other applications. By control data we mean information that can affect the behavior of the involved applications. It takes into consideration issues like communication standards.
- Location independence: It evaluates to what extent the location of the interoperating applications needs to be taken into account and dealt with. It takes into consideration issues like the usage of middleware systems for language and location independence.

Reliability. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software not to fail, i.e., to perform its function satisfactorily. The reliability of the considered application is defined by the following sub-qualities:

- Maturity: It indicates the presence of failures by faults in the software. The term "maturity" was chosen because usually a mature application, i.e. an application that has been used and maintained for a long time, is expected to fail very seldom. In practice the maturity indicates how often an internal fault results in a user observable failure. In the definition of metrics for the software maturity, it should be considered that the frequency of failures depends on how the software is used.
- Fault tolerance: It indicates the ability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
- Recoverability: It indicates the capability of the software to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.

Performance. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to perform its function within given constraints concerning the consumption of resources and time (under stated conditions). The performance of the considered application is defined by the following sub-qualities:

- Time behaviour: This quality relates to the ability of the software to perform the required functionality according to the given time constraints. There are several issues related to the time behaviour that can be taken into consideration: the response time, the processing time, the throughput, etc.
- Resource behaviour: This quality indicates the ability of the software to perform the required function within constraints concerning the amount of resources used and the duration of such use. Among the considered resources there are: the CPU time, the amount of RAM, the amount of disk space and of communication bandwidth, and in general the usage of peripherals of different types.

Note: scalability is also important, and can be seen as an aspect of Performance. However, we do not treat it as a separate quality; rather, when evaluating the behaviour of the OSS with respect to time and resource consumption, we shall take into

consideration how this behaviour varies with respect to the size of the problem/data/computation to be performed.

Security. This quality is desirable in general, i.e., both if the product is used as- is, or if it is changed. It indicates the ability of the software to prevent unauthorized access to program or data.

- Access right enforcement. This quality relates to the ability of the software to provide to any potential user only the type of access privilege that he/she is entitled to (including no access at all for unauthorized users).
- Protection. This quality indicates the ability of the software to protect the programs and data from corruption due to malicious actions.
- Service level. This quality indicates the ability of the software to preserve the service level (no denial of service)

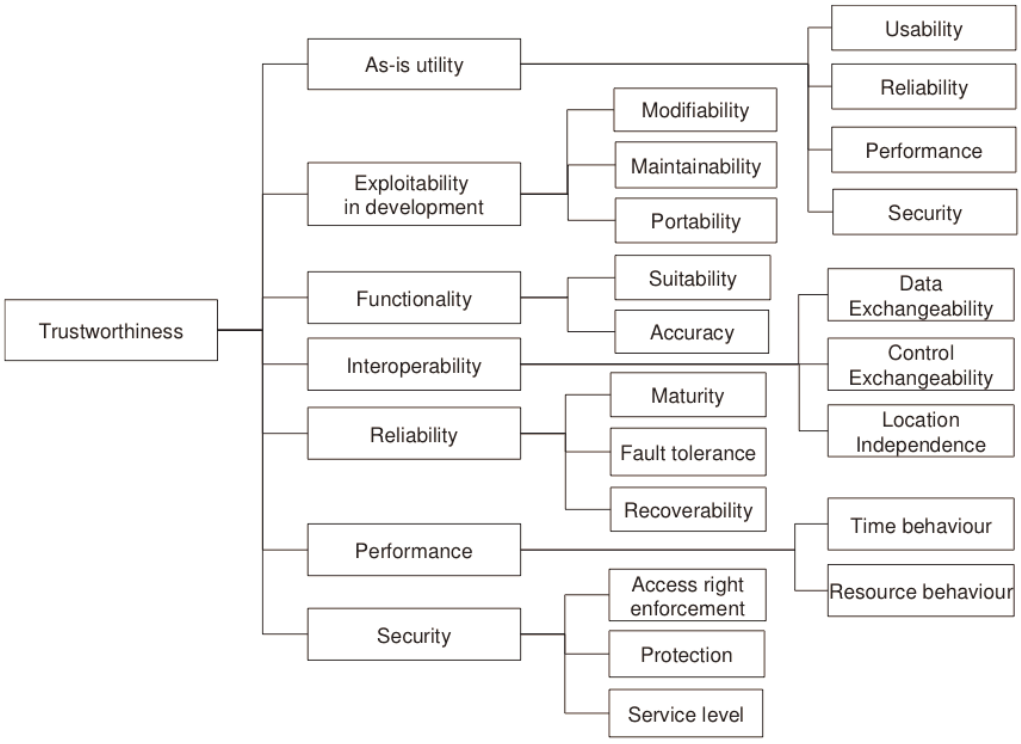

Figure 17: The conceptual model of trustworthiness (first part).

There are several sub-qualities to be considered. For instance, the usability depends on learnability, which depends on the qualities of the user manual. Since these qualities are at a rather low level of detail, they are shown only in the GQM plan.

Cost_Effectiveness. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to contribute positively to the financial balance.

This quality is evaluated objectively by means of the Total Cost of Ownership (TCO), i.e., by evaluating all the components of the TCO:

- Acquisition cost;
- Adaptation cost;
- Deployment cost;
- Maintenance cost;
- Operation cost;
- Training cost;

Components of the cost that depend on specific conditions are evaluated in an "average" context.

Actually, the evaluation of the cost effectiveness of a software product should include also the evaluation of the benefits. However, the benefits depend from the usage of the product: it is not even possible to identify a "typical" representative case. Therefore, we decided to limit the evaluation of the cost effectiveness to the aspects concerning costs.

Customer satisfaction. This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software to contribute positively to satisfying the customer (i.e., the final beneficiary of the process in which the OSS product is involved).

This quality is practically the same as the As-is utility, as far as the involved qualities help achieving external objectives, i.e., user perceivable properties.

Developer quality (reliability). This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates to what extent the developer of the OSS product is reliable. This quality indicates (indirectly) that we can expect a

reasonably good behaviour of the developer, e.g., regular maintenance and evolution of the product.

The sub-qualities that are considered to provide an evaluation of the developer quality are:

- The size and quality of the user community;
- The reputation of the developer;
- The efficiency in removing defects;
- The market share.

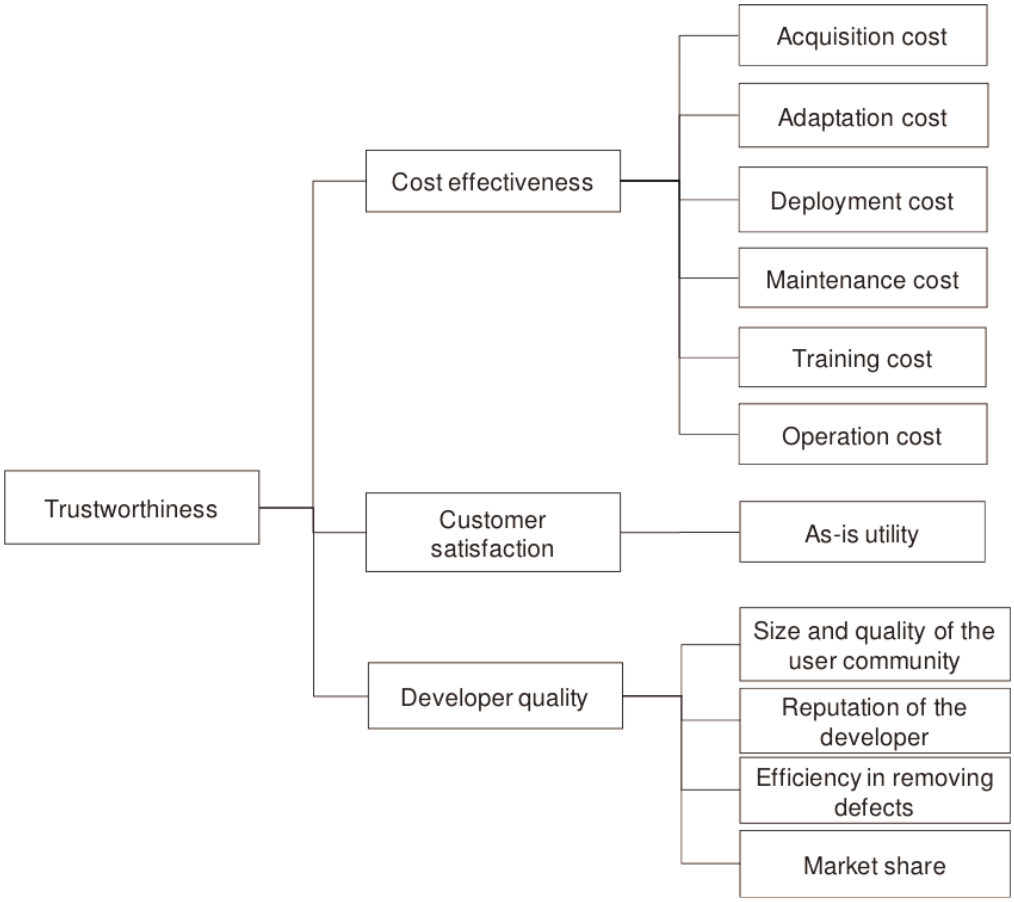

Figure 18: The conceptual model of trustworthiness (second part).

Chaprter 5

The measurement toolset

In order to execute a representative portion of the GQM plan, we need to identify a set of tools. In this section, we describe the tools we selected and MacXim, an OSS measurement tools we developed to measure a set of relevant trustworthiness factors.
Our toolset is composed of four tools developed ad hoc (Spago4Q, MacXim, JaBUTi, and the GQM tool) and seven tools integrated in the platform (StatSVN, StatCVS, PMD, FOSSology and JUnit, PMD and Checkstyle).

5.1 Spago4Q and the integration framework

Spago4Q is an OSS platform that supports the assessment and the quality inspection of software products: these goals are achieved by evaluating data and measures collected from various project management and development tools with non-invasive techniques.
Spago4Q is used to visually represent metrics for product evaluation. These metrics are part of the GQM plan that is instantiated inside the platform. Measurement tools are integrated with Spago4Q by means of a set of 'extractors' that interact with the tools to start an analysis process or to retrieve the results of previously performed analyses. All the collected data are merged into a unique report that visually summarizes the quality and trustworthiness level of the target OSS project. The main features provided by Spago4Q are: data aggregation, quality indicator computation, and dashboards rendering.
Spago4Q v2.0 has been released as FLOSS and can be freely downloaded from http://www.spago4q.org.

5.2 MacXim

MacXim (Model And Code XML-based Integrated Meter) is our tools for measuring the static properties of the source code. The QualiPSo version of MacXim was obtained

enhancing an earlier version. The new release 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 [www.eclipse.org/jdt/], and features a wide set of metrics.

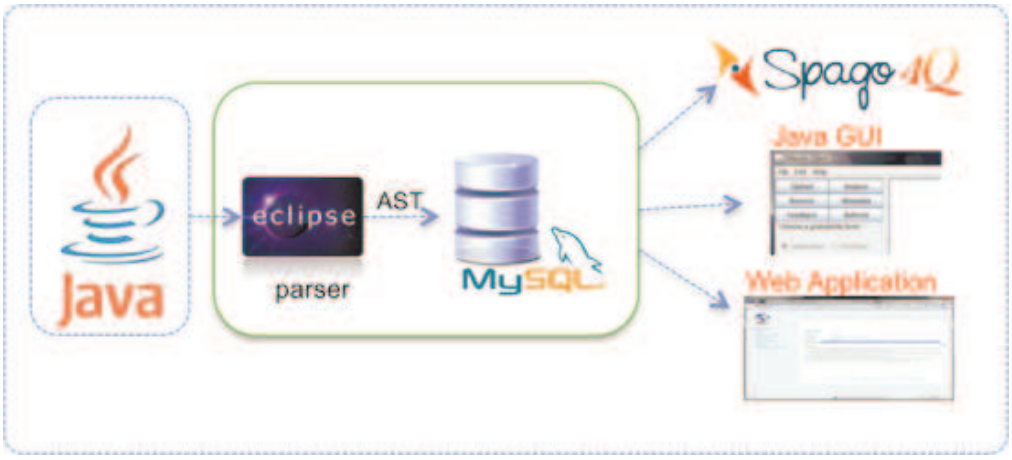

Figure 19: A schematic view of the architecture of MacXim.

While most code measurement tools perform code parsing and measure computation in an integrated way, so that changing the set of computed metrics is relatively complex (as it requires operating on the results of the parsing, often involving the parsing procedure itself), MacXim achieves a much higher flexibility, by clearly separating the parsing phase and the measurement computation phase. MacXim is organized as described in Figure 19. The Java code is parsed by means of the Eclipse parser: the resulting abstract data type is saved –after some elaboration– as a SQL database. Measures are computed by suitable queries (written in SQL or directly implemented in Java) on the contents of the DB. The latter database can also be loaded with measures computed by other tools.

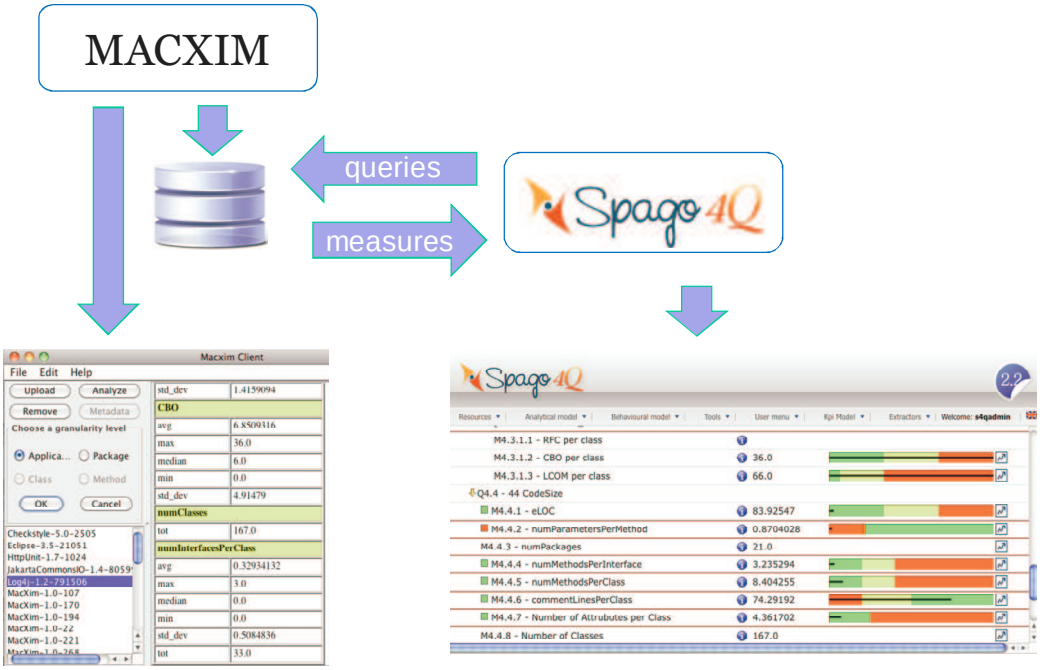

Figure 20. Visualization of quality measures.

As shown in Figure 20, the results of the measurements performed by MacXim can be visualized directly by means of the tool's own interface, or they can be loaded in Spago4Q (see section 5.1) for a visualization in the context of the whole GQM plan.

Currently, MacXim is fully integrated in Spago4Q and computes 70 metrics (including size, complexity, modularity, and various types of object-oriented metrics) at different abstraction levels: application, package, method, and class level.

MacXim has been released as OSS/FLOSS and can be downloaded from http://qualipso.dscpi.uninsubria.it/macxim.

Currently MacXim can measure only Java code. The extension to C++ is planned. In order to perform such extension we shall have to integrate a C++ parser in the MacXim engine. Since C++ and Java –though sharing several constructs and concepts– have also some relevant differences, we shall have also to enhance the schema of the XML database in which the XML representations of the codes are stored.

5.3 JaBUTi

JaBUTi (Java Bytecode Understanding and Testing) is a structural testing tool that implements intra-method control-flow and data-flow testing criteria for the Java byte-code language. JaBUTi implements four intra-method control-flow based testing criteria and four intra-method data-flow based testing criteria. It evaluates the coverage

of a given test set against these testing criteria, reporting the coverage obtained with respect to each one. It operates at unit level and provides aggregated testing reports by method, class, packages or project (composed by a set of packages and classes under test).

The tool works at byte-code level and no source code is required to compute the testing requirements and the coverage. If the source code is available, the tool is able to map back the results computed from the byte code to its corresponding source code. In the current version, JaBUTi is able to import JUnit test sets so that their quality can be evaluated against the different supported structural testing criteria.

The main advantage of JaBUTi, when compared to similar coverage testing tools, consists in the ability of supporting the application of both control and data-flow based testing criteria, while other tools just include control flow testing. Moreover, with JaBUTi, it is possible to evaluate the coverage of different combinations of test cases just by enabling and disabling some of them. Once a testing requirement is identified as infeasible, JaBUTi enables the tester to eliminate it from the coverage computation.

JaBUTi can be used through its GUI or via command line to start the process analysis. JaBUTi is partially integrated in Spago4Q for reporting the results of the analysis. Currently, JaBUTi is able to compute six metrics and contributes to evaluating the actual reliability and correctness of OSS products.

JaBUTi has been released in V1.0 and can be freely downloaded as OSS/FLOSS from http://incubadora.fapesp.br/projects/jabuti.

5.4 The GQM Tool

The GQM Tool implements the homonymous methodology. It is a graphical tool that simplifies the definition of a measurement model on three levels: a conceptual level (goal), an operational level (questions about the goal) and a quantitative level (metrics associated with questions in order to answer them). The GQM Tool makes it easier to define and implement a GQM plan: in the definition phase the GQM plan is modeled and defined (i.e. the associations between goals, quality foci, variation factors, hypotheses, impacts, questions and metrics are specified); in the implementation phase the GQM plan is applied to a concrete case and the values are interpreted through the metrics defined to answer one or more questions.

The GQM Tool saves the GQM plans in XML format: a specific XML Schema has been defined to support this. A GQM plan can be reused in multiple projects. The GQM Tool connects to a RDBMS to extract the values needed in the implementation phase. The extraction is straightforward, since the GQM Tool associates an SQL query with every metric in a GQM plan. The queries are executed during the implementation phase, on the project's repository.

Currently, the GQM Tool has not yet been integrated with Spago4Q and it has not yet publicly available.

5.5 StatSVN and StatCVS

StatSVN [http://www.statsvn.org] and StatCVS [http://statcvs.sourceforge.net/] are third-party OSS tools that were integrated in Spago4Q to generate statistics on the basis of information retrieved from SVN or CVS repositories. Such data are provided both for overall project characteristics/features and also with respect to individual authors, giving insight into their development activities. Beside activity history, these tools collect also data regarding the size of projects (such as the number of files and lines of code added/removed/changed from one revision to another). StatSVN and StatCVS were chosen because of to their usefulness in gathering repository statistics. With these applications it is possible to collect data from two of the most popular version control systems. In addition, these tools have similar capabilities and are easy to use. Moreover they use the same data model and report generators.

StatSVN and StatCVS have been fully integrated in Spago4Q, thus providing both the ability of starting a new project analysis or a data extraction directly via the Spago4Q interface. Hence, StatSVN and StatCVS are transparent to final users that only interact with the Spago4Q platform to perform the analysis and extract the results.

Currently, StatSVN and StatCVS are used to compute eleven metrics (such as the number of developers and commits, the mean number of LOC added per year, etc.). The modified versions of StatSVN and StatCVS are not yet publicly available.

5.6 PMD and Checkstyle

PMD and Checkstyle [http://sourceforge.net] are tools for code analysis that scan Java source code and look for potential problems, such as possible bugs (for example empty

try/catch/finally/switch statements), dead code (for example unused local variables, unused parameters, unused private methods), and suboptimal code (for example wasteful String concatenation usage instead of StringBuffer/StringBuilder). Other interesting features are the capability to detect cut and paste portions of source code. They can be used against the source code coming from two or more software systems, to detect plagiarism; or they can be used against the source code coming from a single software system, to detect cut and pasted portions of code that could lead to maintenance and quality problems.

PMD and Checkstyle have been chosen because they are widely used automated code review software tools for the Java language. PMD and Checkstyle have been fully integrated as is inside MacXim to compute 32 additional code quality metrics.

5.7 GQM metrics mapping

The tools identified and developed in this section are aimed to measure as much GQM metrics as possible. In Table 11 we can find the mapping between the available tools and the GQM metrics.

			COBERTURA	JABUTI	CHECKSTYLE	PMD	MACXIM	STATSVN	STATCVS
QF	**QUESTIONS**	**METRIC**							
Actual_Functionality_Suitability	LicenseCharacteristics	DistributionAgreement							
		FeeAllowed							
		ModificationOfCodeAllowed							
		NumberOfLicences							
		SourceCodeAccessible							
		copyrightedMaterial							
	RequirementsSatisfaction	RequirementsSatisfactionDegree							
Actual_Interoperability	EaseOfDataExchange	AutomaticalMagaementOfOtherSoftwareData							
		EaseOfDataParsing/Unparsing							
		SemanticallyWellDefinedDataFormat							
		StandardDataFormatSupported							
		UserDefinedData							
	EaseOfIntegration	StandardApplicationInterface							
		StandardInterfaceConformityEvidence							
	LocationIndependence	LocationIndependenceSupport							
	ProtocolBasedDataExchange	StandardProtocolSupportEvidence							
		StandardProtocolSupported							
Actual_Reliability	Correctness	CorrectnessWrtTests							
		TestConditionCoverage		**X**					
		TestInstructionCoverage	**X**	**X**					
		TestPathCoverage		**X**					
	Dependability	DependabilityEvidence							
	FailureFrequency	FailuresFrequency							
	HowProbableAreProblems WrtCodeConstruction	UnexpectedSituationHandlingIndex			**p**	**p**			
	ProductMaturity	BugTrend							
		ProductMaturityLevel							
		ReleasesTrend							**X**
	Robustness	Robustness (same as correctness, outside specs)							
Actual_Exploit_in_dev_Maintainability	AnalyzabilityForMaintenance	CodeDocumentation					**X**		
		CodeModularity					**X**		
		CodingStandardEnforcement			**X**	**X**			
		CodingStandards							
		MaintainabilityOrientedArchitecture							
		RunTimeModularity							
	BugRemovalEfficiency	BugClosedPercentageRate							

		BugClosureRatePerDeveloper							
		BugRemovalRate							
	CodeQuality	%OfClassesRespectingMaxSLOC					**X**		
		%OfMethodsRespectingMaxSLOC					**X**		
		ECA rules			**p**	**p**	**X**		
		McCabeIndex					**X**		
	CodeSize	CodeSizeinLOC					**X**	**X**	**X**
		OOCodeSize					**X**		
		SLOC					**X**		
	MaintenanceStability	DesignPatternUsage							
		NumberOFailuresDueToMaintenance							
	MaintenanceTestability	AvgNumOfTestPerMethods							
		AvgNumOfSLOCPerTestCase							
		AvailabilityOfTestDocumentation							
		McCabeCyclomaticNumber					**X**		
		NumberOfTestCases							
		TestResultsAvailability							
	NewReleaseRate	BugsReportingRateperKLOC							
		ChangedLOCSperYear						**X**	**X**
		MajorReleasesPerYear							**X**
		MinorReleasesPerYear							**X**
	StandardArchitecture	StandardArchitecture							
	SupportingToolAvailability	ToolSupport							

Table 10. The tools to be used for collecting the metrics in the GQM plan

Legend: x = tool available to support metric; p = tool available to partly support metric.

Chapter 6

Data Collection

The goal of this section is to collect data concerning 22 Java and 22 C/C++ OSS products. For each of these products we aimed to have:

- Evaluations concerning the qualities of the products that are perceived subjectively by the users. These evaluations are expressed in an ordinal scale (form 0 = totally unsatisfactory to 6 = excellent).
- Measures that capture in an objective way the characteristics of the software.

6.1 The OSS products being analyzed

6.1.1 Objectives: defining the set of product to be evaluated

The first round of experiments was performed on a small set of projects, in order to verify that the whole set of techniques, tools, models and methods defined in the previous section are effective with respect to the overall goal, i.e., that they are suitable for deriving the required information concerning the trustworthiness of OSS.

The set of OSS products to be evaluated during the first round of experiments was chosen by means of a careful procedure. The products must support the specific goal of the first round of experiments, i.e., proving that the techniques, tools, methods and models defined in this work are effective for the purpose of evaluating OSS trustworthiness. Moreover, the set of products must be numerous enough to support the subsequent data analysis and derivation of the trustworthiness model

6.1.2 Method: selection criteria

The choice of the products was carried out according to multiple criteria.

Coherence with the project selected in Section 6.1

The most obvious way of choosing the projects to evaluate in the first round is to reuse the consideration performed in Section 3.3.1. In Section 3.3.1, a set of relevant OSS products and artifacts have been selected, by taking into account different types of software applications, generally considered stable and mature (Table 4).

Comparability

Another interesting consideration is that by selecting products for which quality metrics have already been published, it would be possible to compare our overall quality results with the published values, and/or exploit the published evaluations as part of our data set. To this end, we considered as candidates for our OSS evaluation the projects evaluated by the Eclipse Enerjy plugin and by the commercial tool Structure101. Table 11 reports the list of projects that are evaluated in this task and also by Eclipse Enerjy or Structure101.

OSS product	QualiPSo	Structure101	Enerjy
Ant	✓	✓	
Eclipse	✓	✓	✓
Findbugs	✓	✓	✓
Hibernate	✓		✓
JasperReport	✓		✓
JBoss	✓	✓	
JFreeChart	✓	✓	✓
JMeter	✓	✓	✓
PMD	✓	✓	✓
Saxon	✓		✓
Struts	✓	✓	✓
Tapestry	✓	✓	✓
Velocity	✓		✓
Weka	✓		✓

Table 11. The list of OSS products being evaluated also by other initiatives.

Additional selection criteria

In order to validate the set of techniques and methods at the current stage, we took into consideration also the following points:

- The products should be different in the kind of “user interface” they offer, i.e., select a full-fledged GUI-based OSS (like Eclipse or JMeter) compared to a “library” product (like Jakarta commons).

- The products should allow for applying the measuring and testing capabilities provided by the tools identified in Section 0. For instance, since currently our code measuring tools deal only with source code written in Java, we made sure to include in the set of tools to be evaluated in the first round of experiments a statistically relevant number of Java programs.

6.1.3 Results: the list of products evaluated during the first round of experiments

During the first round of experiments we evaluated 22 java projects and 22 C/C++ projects reported in Table 12.

Product	
Checkstyle	Ant
Eclipse	Axis
Findbugs	BusyBox
Hibernate	CVS
HttpUnit	CygWin
Jakarta CommonsIO	DDD
JasperReport	GDB
JBoss	Gnu C Library
JFreeChart	Gnu GCC
JMeter	Lib XML
Log4J	Linux Kernel
PMDV	Mono
Saxon	MySQL
Spring-FW	OpeLDAP
ServiceMix	Open Pegasus
Struts	Open SSL
Tapestry	Perl
TPTPV	PosgreSQL
Velocity	SpiderMonkey
Weka	SQLite
Xalan	Subversion
Xerces	TCL/Tk

Table 12. The list of OSS products being evaluated during the first round of experiments

6.2 Preparation of the data repository

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

The repository should also integrate nicely with the measurement and data collection tools.

Such repository has to collect the data from the various tools and make them available to the analysis activities and to the reporting tool (Spago4Q), as shown in Figure 21.

The construction of the repository proceeded through the usual phases of database design and implementation. The repository is based on MySQL relational DBMS. MySQL was chosen because it is a reliable OS product and because it had already been used in conjunction with Spago4Q. It is also expected to support seamless integration with the analysis tools.

The main result of the database design activity is illustrated in Figure 22 and Figure 23.

In particular, Figure 22 accounts for the Tables that are dedicated to storing the user perception of the trustworthiness of the OSS products. Table OSS_Product stores the data concerning the OSS products (name, version, licence, etc.); table User stores a set of data that characterize the users who provided the trustworthiness evaluations; table PerceivedTrustworthiness has an attribute for every quality aspect (reliability, safety, usability, etc.) that is relevant to characterize the trustworthiness of OSS products.

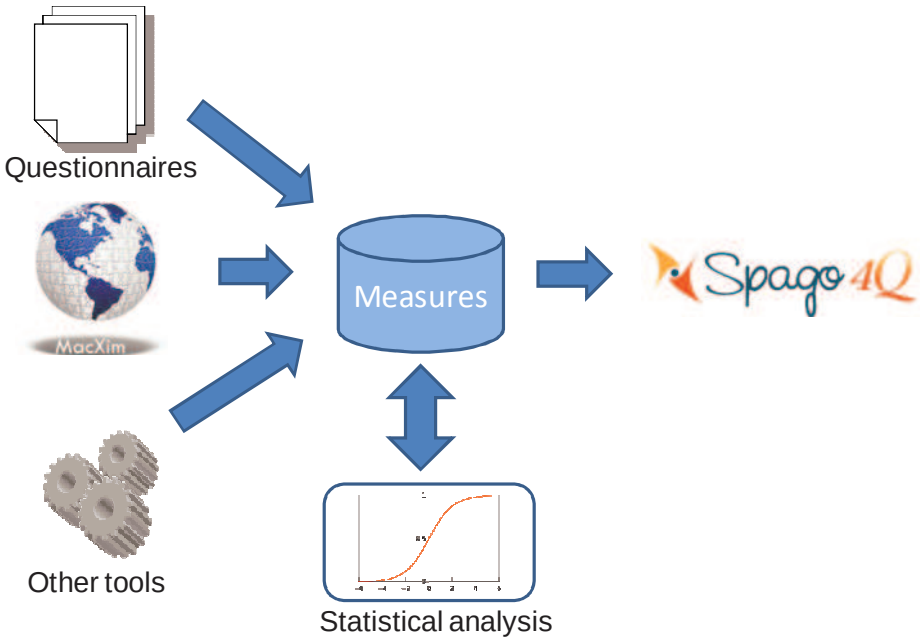

Figure 21. Role of the measures DB

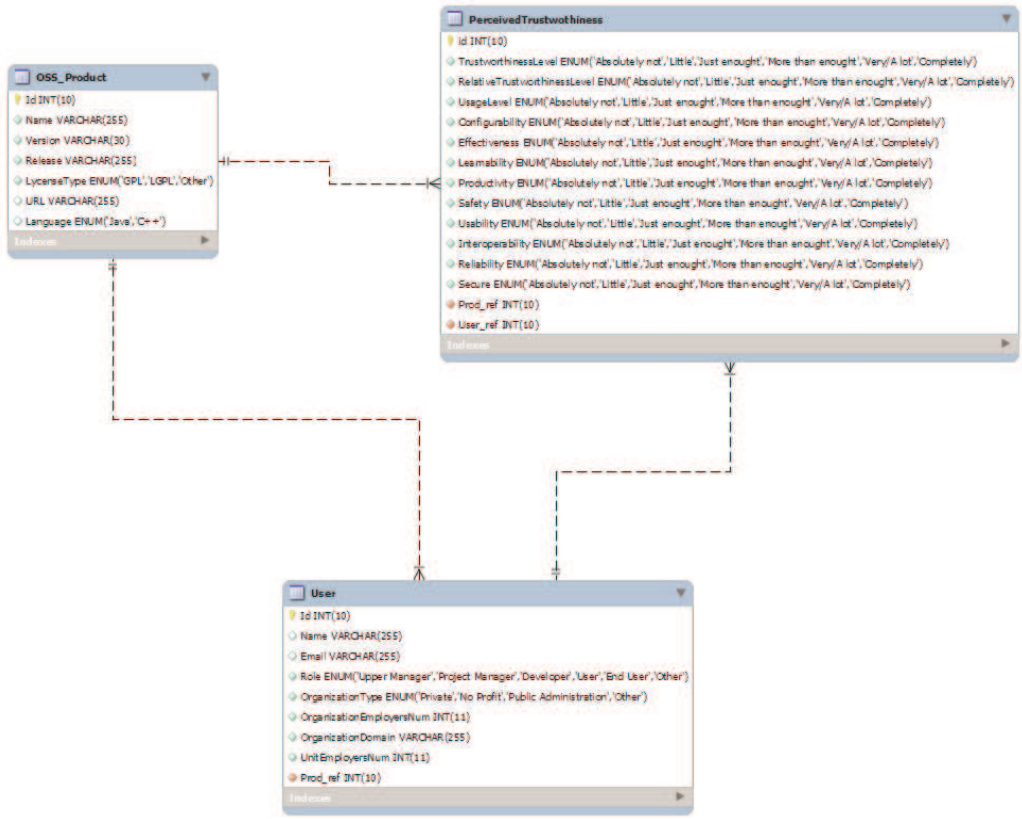

Figure 22. Conceptual model including all user perceived aspects of trustworthiness.

Figure 23 illustrates the tables that were designed to contain the data concerning the objective measures of the product characteristics. There is a table for each element

(class, method, attribute, ...) and granularity level (application, package, class, ...) for which measures can be defined. In addition to these tables, there are three tables for storing the measures form the ”foreign” tools that we are planning to use, namely PMD, FindBugs, and Checkstyle. Finally there is a table for storing data from any additional measurement tool that we could decide to use in the future.

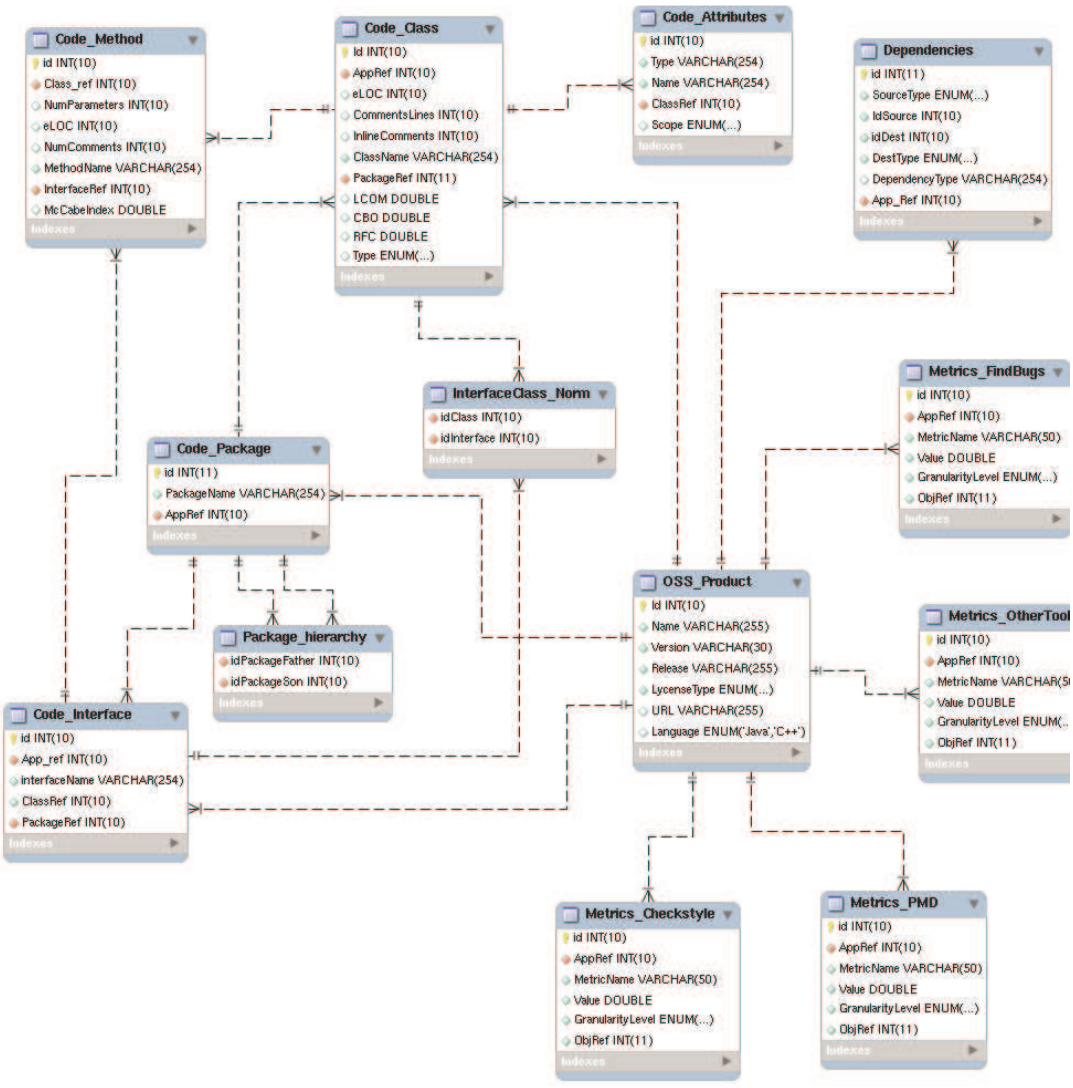

Figure 23. Conceptual model including the objective data.

6.3 Subjective evaluation of perceived trustworthiness

The collection of the subjective evaluation of the various aspects of trustworthiness by users proceeded through a series of steps:

- A first version of a questionnaire (concerning 11 projects) was released. This version proved to be too detailed: people would not spend the time required to provide all the requested information.
- A second version of the questionnaire was released. This version contained only a few questions, with the possibility to answer them for multiple products. As already mentioned in section 6.1.3, the questionnaire includes questions on a set of 22 Java programs and a set of 22 C++ programs. The questionnaire is reported in appendix (section Appendix C:).
- An on-line version of the questionnaire was published, in order to ease the collection of data. See http://qualipso.dscpi.uninsubria.it/survey. The screenshot of the initial page is reported in Figure 24.

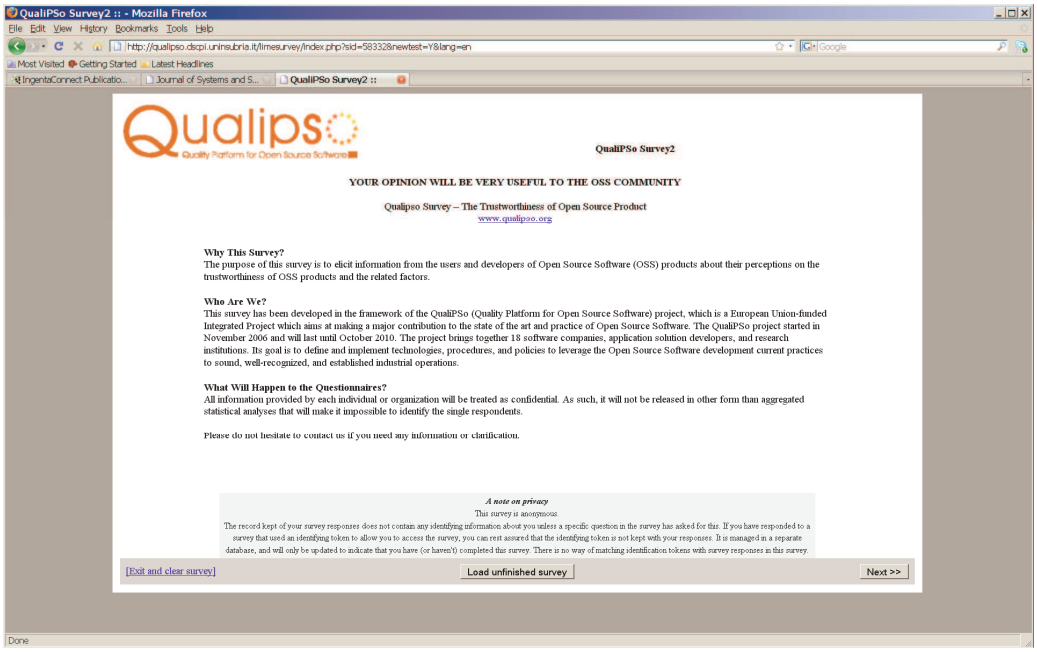

Figure 24. A screenshot of the welcome page of the online questionnaire.

Up to the end of June 2009, 532 questionnaires were collected. Overall, they account for 3809 evaluations.

Table 13 reports the questions in the questionnaire while Table 14 lists the number of evaluations per project collected.

QUESTION IN THE QUESTIONNAIRE	PERCEIVED QUALITY (short name)
How familiar are you with the product?	Familiarity
How usable is the product?	Usability
How portable is the product?	Portability
How much does/did the product satisfy your functional requirements when you use/used it?	Functional Requirements
How interoperable is the product?	Interoperability
How reliable is the product?	Reliability
How secure is the product?	Security
How useful is the product developer community to you?	Community
How well documented is the product?	Documentation
How fast is the product?	Fastness
How much do you trust the product, compared to its Open Source competitors?	Trust_wrt_oss
How much do you trust the product, compared to its non Open Source competitors?	Trust_wrt_non_oss
How much do you trust the product, overall?	Trustworthiness

Table 13. The evaluated characteristics in the questionnaire

PRODUCT NAME	PROGRAMMING LANGUAGE	EVALUATIONS
Ant	C/C++	109
Axis	C/C++	15
BusyBox	C/C++	72
Checkstyle	JAVA	27
CVS	C/C++	154
CygWin	C/C++	112
DDD	C/C++	17
Eclipse	JAVA	341
Findbugs	JAVA	34
Firefox	C/C++	128
GDB	C/C++	107
Gnu C Library	C/C++	141
Gnu GCC	C/C++	163
Hibernate	JAVA	103
HttpUnit	JAVA	35
Jack.Commons IO	JAVA	13
Jasper Reports	JAVA	37
JBoss	JAVA	94
JFreeChart	JAVA	36
JMeter	JAVA	40
Lib XML	C/C++	62
Linux Debian	C/C++	58
Linux Kernel	C/C++	205
Log4J	JAVA	118

Mono	C/C++	37
MySQL	C/C++	273
OpeLDAP	C/C++	54
Open Office	C/C++	127
Open Pegasus	C/C++	5
Open SSL	C/C++	117
Perl	C/C++	139
PMD	JAVA	28
PosgreSQL	C/C++	106
Saxon	JAVA	27
Servicemix	C/C++	5
SpiderMonkey	C/C++	10
Spring Framework	JAVA	57
SQLite	C/C++	108
Struts	JAVA	55
Subversion	C/C++	188
Tapestry	C/C++	7
TCL/Tk	C/C++	21
TPTP	JAVA	5
Velocity	JAVA	19
Weka	JAVA	11
Xalan	JAVA	34
Xerces	JAVA	55

Table 14. Number of evaluations per project

6.4 Objective evaluations of OSS product characteristics

19 of the 22 Java projects whose trustworthiness is being evaluated were also measured (i.e., their characteristics were objectively evaluated).

We report the measures concerning only a small set of products, in order to illustrate the kind of measures that were collected.

Table 15 reports the measure of the level of coverage of the tests that are available for a given set of OSS products. These measures are related to reliability (the higher the coverage the more effective the testing, the more correct, hence reliable, the released product).

OSS product	Jmeter	Log4J	PMD	HSQLDB	Junit
version	2.3.2	1.2.15	5.0	1.9 Alpha 2	4.6

2010

Size (bytecode instructions)	161,385	34,848	133,727	277,533	11,019
Number of Classes with Exception Handlers	285	75	73	200	41
Number of Methods with Exception Handlers	625	201	374	683	63
Cov Req / All-Nodes_ei	0.38	0.41	0.37	0.20	0.31
Cov Req / All-Edges_ei	0.28	0.38	0.29	0.17	0.26
Cov Req / All-Uses_ei	0.27	0.41	0.23	0.16	0.24
Cov Req / All-Pot-Nodes_ei	0.26	0.38	0.15	0.15	0.24
Cov Req / All-Nodes_ed	0.03	0.06	0.16	0.07	0.08
Cov Req / All-Edges_ed	0.01	0.01	0.03	0.01	0.05
Cov Req / All-Uses_ed	0.02	0.07	0.11	0.09	0.07
Cov Req / All-Pot-Nodes_ed	0.02	0.05	0.08	0.09	0.05

Table 15. Coverage measures

Table 16 reports the ECA (Expert Code Assessment) measures that have been for the usual small set of sample projects. There measures were identified out of Siemens' experience and are expected to indicate poorly designed and/or implemented code, which is more error prone.

OSS product	Jmeter	Log4J	PMD
ECARules_EqualsNotDefinedWithHash	0.015	0	0.045
ECARules_EqualsNotDefinedWithHash	0.015	0.03	0.045
ECARules_CatchingThrowable	0.15	0.303	1.681
ECARules_MissingBreakInSwitch	0	0	0
ECARules_MissingBreakInSwitch	0	0.03	0.166
ECARules_EmptyCatchBlock	0.714	0.424	0.454
ECARules_ClassNameWithLowerCase	0	0	0
ECARules_ClassNameWithLowerCase	0	0	0
ECARules_ExcessiveClassLength	0.047	0.03	0.075
ECARules_DubiousFloatingPointComparison	0	0	0
ECARules_DubiousStringComparison	0	0.03	0
ECARules_DubiousStringComparison	0	0.03	0
ECARules_ConstructorCallsOverridableMethod	1.15	0.969	0.257
ECARules_DuplicatedCode	1.079	0.242	5.742
ECARules_MissingBracesIfStmts	0.015	4.242	29.712
ECARules_MissingBracesWhileLoops	0	0.03	0.045
ECARules_MissingBracesIfElseStmts	0	1.878	2.287
ECARules_MissingBracesForLoops	0	0.03	0.545
ECARules_FieldNeverInitializedProperly	0.031	0	0
ECARules_ExcessiveMethodLength	0.206	0.09	0.5
ECARules_NullPointerDereference	0	0	0
ECARules_HidingField	0	0.03	0.09
ECARules_ReferenceToMutableObject_ReturnsArray	0	0.03	0
ECARules_ReferenceToMutableObject_FinalArray	0.023	0	0
ECARules_ReferenceToMutableObject_FinalHashTable	0	0	0
ECARules_ReferenceToMutableObject_ReturnsObject	0.087	0.03	0.03
ECARules_DubiousArrayComparison	0	0	0
ECARules_MissingDefaultInSwitch	0.063	0.06	0.212
ECARules_UnreadField	0.031	0.212	0.03
ECARules_UnusedField	0	0	0.015
ECARules_UnusedField	0.047	0.06	0.045
ECARules_UnusedPrivateMethod	0	0	0
ECARules_UnusedPrivateMethod	0.039	0	0.075

Table 16. ECA (Expert Code Assessment) rules measures

Table 17 reports the static code measures of OSS products. The table accounts for size metrics of various types (from LOCs to number of classes, methods, etc.) and for typical object-oriented metrics (namely those proposed by Chidamber and Kemerer).

Metric Name		JMeterV2.3R3	Log4JV1.2R0	PMDV4.3R5
eLOC	median	36	47	20
	avg	63.41	71.84	53.56
	tot	18517	12717	35243
	max	584	913	2797
	min	2	-54	2
	std_dev	6769.86	10180.64	25613.15
McCabe	median	4	6	3
	avg	8.69	11.23	6.32
	max	98	111	149
	min	0	0	0
	std_dev	10.68	13.53	12.05
LCOM	median	6	6	1
	avg	80.39	80.44	79.06
	max	4361	5132	6441
	min	0	0	0
	std_dev	305.28	616.67	464.54
numAttributesPerClass	median	2	4	1
	avg	4.04	4.01	2.41
	tot	1289	828	1759
	max	67	37	123
	min	0	0	0
	std_dev	7.60	6.03	9.47
numClassesWithDefinedAttributes	tot	229	154	349
numClassesWithDefinedMethods	tot	299	197	686
CBO	median	8	10	6
	avg	12.06	13.85	11.69
	max	97	120	228
	min	0	0	0
	std_dev	14.02	13.63	20.32
commentLinesPerClass	median	62	80	9
	avg	106.09	129.45	31.91
	tot	30979	22914	20998
	max	1083	1083	772
	min	17	16	0
	std_dev	120.45	129.98	61.66
numAttributesPerClass	median	2	4	1
	avg	4.04	4.01	2.41
	tot	1289	828	1759
	max	67	37	123
	min	0	0	0
	std_dev	7.60	6.03	9.47
numClassesWithDefinedAttributes	tot	229	154	349

numClassesWithDefinedMethods	tot	299	197	686
numClasses	tot	292	177	658
numInterfaces	tot	0	0	0
numInterfacesPerClass	median	0	0	0
	avg	0.69	0.34	0.24
	tot	204	61	162
	max	6	3	10
	min	0	0	0
	std_dev	0.94	0.51	0.67
numMethods	tot	2454	1431	4066
numMethodsPerClass	median	5	7	3
	avg	8.40	8.08	6.17
	tot	2454	1431	4066
	max	98	104	114
	min	0	0	0
	std_dev	10.33	9.79	11.32
numMethodsPerInterface	median	0	0	0
	avg	0	0	0
	tot	0	0	0
	max	0	0	0
	min	0	0	0
	std_dev	0	0	0
numPackages	tot	41	21	72
numParametersPerMethod	median	4	5	5
	avg	7.15	8.41	8.91
	tot	2089	1489	5866
	max	68	79	226
	min	0	0	0
	std_dev	9.37	9.49	18.81
RFC	median	4	4	3
	avg	8.45	8.19	6.16
	max	98	108	100
	min	0	0	0
	std_dev	10.44	10.09	10.95

Table 17. Static code measures

Table 18 reports that defect measures extracted from bug repositories. Data on defects is clearly important to explain reliability, to assess maturity, and to relate code characteristics to reliability.

OSS Product	Defect status					
	New	Assigned	Reopened			Total
JMETER V2.3R3	4	0	1			5
	Open					Total
CHECKSTYLE V5.0R2505	7					7
	New	Assigned	Resolved	Verified	Closed	Total
ECLIPSE JDT CORE V3.5R0	140	3	44	178	7	372
	Open					Total
FINDBUGS no version	70					70
	Open	Reopened				Total
HIBERNATE no version	62	1				63
	Open					Total
HTTPUNIT no version	13					13
	New	Acknowledged	Confirmed	Assigned		Total
JASPER V3.5.2	43	20	4	100		167
	Open					Total
JBOSS V3.2.6 Final	3					3
JBOSS V5.1.0 GA	27					27
	Open					Total
JFREECHART V1.0.X	46					46
	New	Assigned	Reopened			Total
LOG4J V1.2R0	43	8	7			58
	Open	Pending				Total
PMD	134	6				140
	Open					Total
SAXON V9.1	4					4
	Open	In Progress	Reopened			Total
STRUTS V2.1.6	40	0	1			41
	Open	In Progress	Reopened			Total
VELOCITY V1.6.2	4	0	1			5
	Open	In Progress	Reopened			Total
XALAN V2.7.1	62	0	13			75
	Open	In Progress	Reopened			Total
XERCES V2.9.1	16	0	0			16

Table 18. Defect measures

Chapter 7

Analysis

7.1 Introduction

The goal of the analysis reported in this section is to evaluate –through statistical methods– whether there are relations that link the subjective perception of OSS product qualities with objective measures of the software. For instance, the goal of the analysis includes the verification of the existence of relations of the trustworthiness, reliability, portability, etc., with characteristics like software size, complexity and modularity...

For space reasons, we describe the results of the analysis of java projects. The detailed results for Java projects are available in Appendix F: while results on C/C++ are listed in Appendix G:

7.1.1 Tools

The analysis of the data provided in the previous section was carried out using appropriate tools. In particular, statistical tools were needed in order to perform the necessary computations and verify whether the factors identified in Section 4.4 are actually influential on the trustworthiness of the OSS products and artifacts.

The analysis performed in this section is characterized by quite classical statistical techniques. Therefore we did not look for a particularly sophisticated tool; rather we sought a tool that:

- Can be integrated at the data level with the measurement repository. In fact, we need to extract the required data from the repository and feed them to the analysis tools in a simple and efficient way.
- Provides all the required statistical tools, in particular logistic regression.

- Is programmable, in order to let us define the statistic procedures to be applied repetitively.

On the basis of these requirements, it was decided to use R, the tool that was already successfully used in Section 3. R is a GPL-licensed language and 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/).

7.1.2 Analysis procedures

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

Since we have interviewed several users about a given quality of a given OSS product, 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[2] 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.

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.

[2] 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.

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 subjective 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, in the analysis reported in section **Errore. L'origine riferimento non è stata trovata.** 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 is a dichotomy and the independents 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 25, 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.

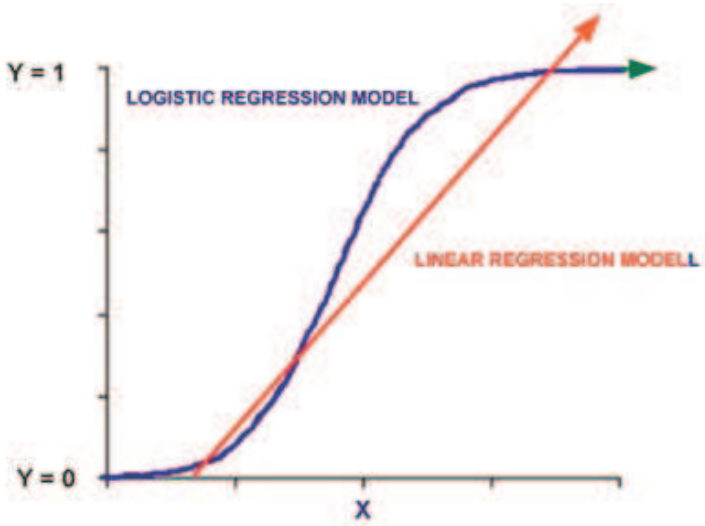

Figure 25. Logistic vs. linear regression curves.

The analysis procedures based on logistic regression are reported in detail in the appendixes, in the form of scripts for the R toolset.

7.1.3 The dataset

The analyses reported in this document are based on the users evaluations and measures collected up to September 30 2010.

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.
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. Table 14 reports the number of evaluations per project while Table 13 reports the list of characteristics evaluated by the users. Detailed results on the users' evaluation can be found in Table 2 or in [24]

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 18 products, depending on the specific quality being considered (users were free to express opinions only on a subset of the products' qualities) (Table 13).

7.2 Analysis of Java products

In this section, we summarize the results of the analysis carried out for Java projects. The detailed results are reported in Appendix F.

7.2.1 Reliability

Of the statistically significant relations found, the most precise and reasonably explained is the one that links Reliability with the number of interfaces per class. Such relationships indicates that defining multiple interfaces for a single class can be a dangerous practice, which leads to a decrease in reliability perceivable by the end users. The regression line is reported in Figure 26.

The distribution of relative residuals is reported in Figure 27.

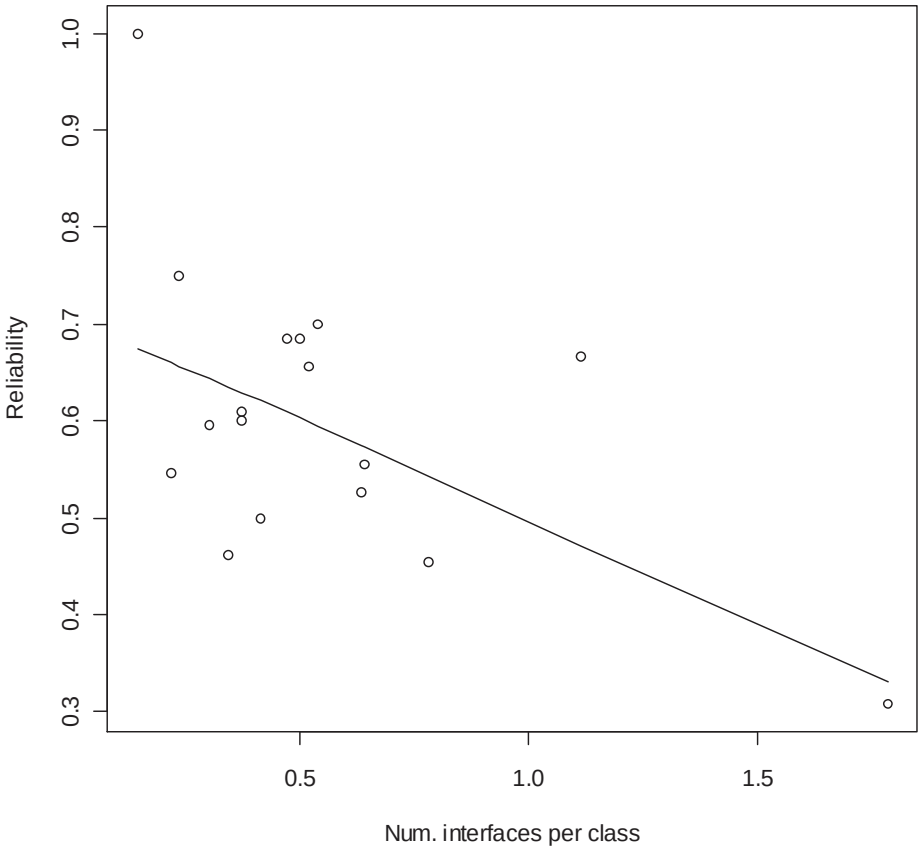

Figure 26. Reliability vs. The number of interfaces per class: regression line.

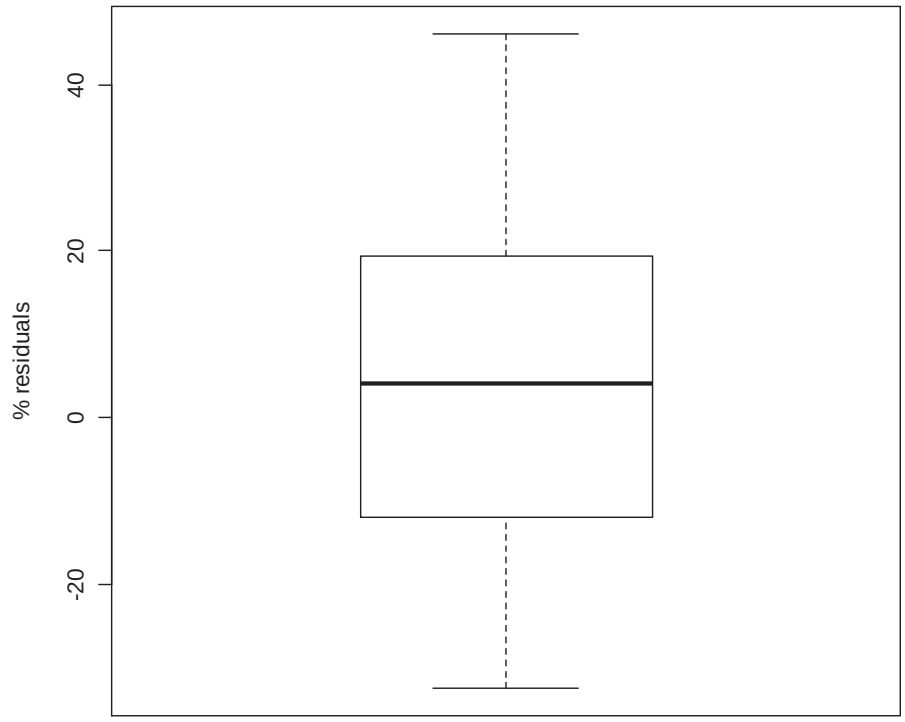

Figure 27. Reliability vs. The number of interfaces per class: boxplot of relative residuals.

7.2.2 Usability

Among the statistically significant relations found, the most precise and reasonably explained are the ones that link Usability with the number of interfaces and the number of methods per interfaces. This appears quite understandable: the more interfaces and methods are provided, the more probable is that the user is given the function he/she needs in a way that is considered easy to use.

Of course it is difficult to establish a really reliable and credible relation between a quality that is based almost exclusively on external elements (e.g., the user interface) and the measures of the internal qualities. Anyway, it seems that several of the correlations found can at least be considered reasonable.

Another interesting correlation found is the one that indicates that Usability grows with the size of classes (eLOC per class) and the number of parameters per method, while it decreases with the global size of the application (eLOC). This seems to indicate that smaller applications, with big classes and highly parameterized methods tend to be more usable.

7.2.3 Portability

The results found for portability seem very reasonable, and generally conformant to the expectations.

The fact that portability grows with the NOC (number of children, i.e., the number of sub-classes) seems to indicate that portability is favoured by rich generalization hierarchies. This seems reasonable: the richer the hierarchy, the easier to encapsulate and share the required adaptations.

The fact that portability grows with the number of packages seems to indicate that in a system with several packages it is easier to find packages that do not depend on the specific platform, and can thus be ported with little effort.

The fact that portability grows with the number of effective LOC per class, while at the same time it decreases with the number of interfaces per class can be explained considering that ‘big’ classes are easier to port (i.e., the porting effort probably depends on the number of classes, rather than on their size) and that many interface increase the difficulty of porting.

The fact that portability grows with the McCabe complexity, while at the same time it decreases with the number of parameter per method seems to confirm the previous observation. In fact, complexity can be seen –like size– as a measure of how much computation is performed in a class, while a high number of methods per class increases the probability that some of these parameters depends on the platform, thus posing porting problems.

Interestingly, the correlations found are very precise. Moreover, the multivariate regressions are characterized by the absence of outliers. The distribution of relative residuals of the Portability vs. McCabe and number of parameter per method is reported in Figure 27.

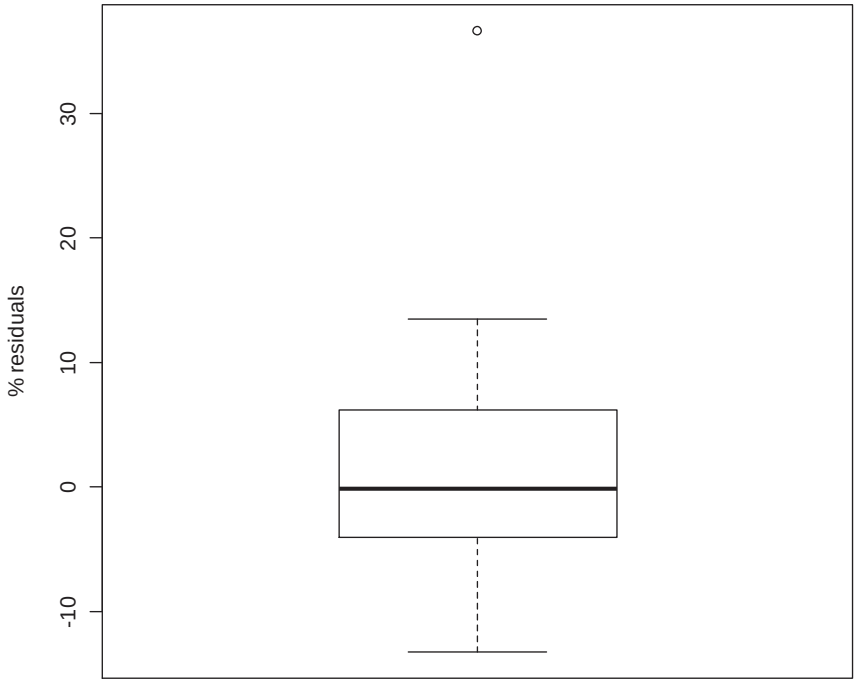

Figure 28. Portability vs. McCabe and number of parameter per method: boxplot of relative residuals.

7.2.4 How well are functional requirements satisfied

Of the statistically significant relations found, several are acceptably precise and reasonably explained. Among these, are the correlations of the degree of satisfaction of functional requirements with the following factors (a minus in parentheses indicates that the factor contributes negatively to the perceived satisfaction of functional requirements):

- eLOC (-) , McCabe
- McCabe , Num. Methods (-)
- McCabe , Num. methods per class (-)
- Num. attributes per class , Num. methods per interface

The first two correlations seem to indicate that smaller and more complex applications are more likely to satisfy users' requirements. The third correlation says the same, but at the class level. The last one seems to indicate that classes rich in data and in exported methods are more likely to contribute to satisfy users' requirements.

7.2.5 Interoperability

Of the statistically significant relations found, the most precise is also the one most reasonably explained. In fact, the correlation indicating that Interoperability grows with number of attributes per class, while it decreases with number of public methods confirms the well known notion that a good encapsulation favours interoperability.

7.2.6 Security

Only one statistically significant correlation involving Security was found. This seems to confirm that security is difficult to evaluate on the basis of internal characteristics; in particular, measures that indicate a good design are not able to support the perception of a good security level.

Quite interestingly, the only correlation found says that the bigger the application, the more secure it is, according to end users. How this result should be interpreted is not clear. An hypothesis could be that bigger application are the result of wide, well organized development efforts that are more likely to pay attention to security.

7.2.7 Speed

Quite interestingly, all the statistically significant correlations found are characterized by negative coefficients, i.e., they indicate the factors that decrease the efficiency of the analyzed applications. Among these factors are:

- The size of classes (eLOC per class)
- The number of interfaces or public methods per class
- The amount of data managed by a class (number of attributes per class)
- The lack of cohesion of classes

The correlation characterized by the better precision indicates that efficiency is most hindered by the combination of lack of cohesion and the a large number of public methods.

7.2.8 Documentation Quality

Establishing a correlation between the perceived quality of documentation and the measured internal qualities appears difficult. The only significant correlation found is not very precise (MMRE is close to 40%). However, quite interestingly, the correlation involves the comment lines and the comment lines per class, thus showing that the developers' attention to commenting the code is somehow correlated to the amount and quality of the documentation made available to the end user.

7.2.9 Trustworthiness with respect to non Open Source products

Establishing a correlation between the perceived trustworthiness with respect to non Open Source products and the measured internal qualities appears difficult. The only significant correlation found indicates that the more complex is the OSS product being examined, the more likely it is that it is preferred to non OS alternatives. The precision of the correlation is quite good (MMRE close to 10%).

The explanation of the correlation lays probably in the fact that the most complex OSS products (like PMD, for instance) do not have popular non OSS competitors.

7.2.10 Trustworthiness

A univariate regression tend to suggest that the trustworthiness of OSS products decreases with the size expressed in terms of number of classes, methods, or public methods.

The trustworthiness vs. Number of methods regression line is illustrated in Figure 29. It is fairly precise (MMRE=17.7%). The boxplot of relative residuals is reported in Figure 30.

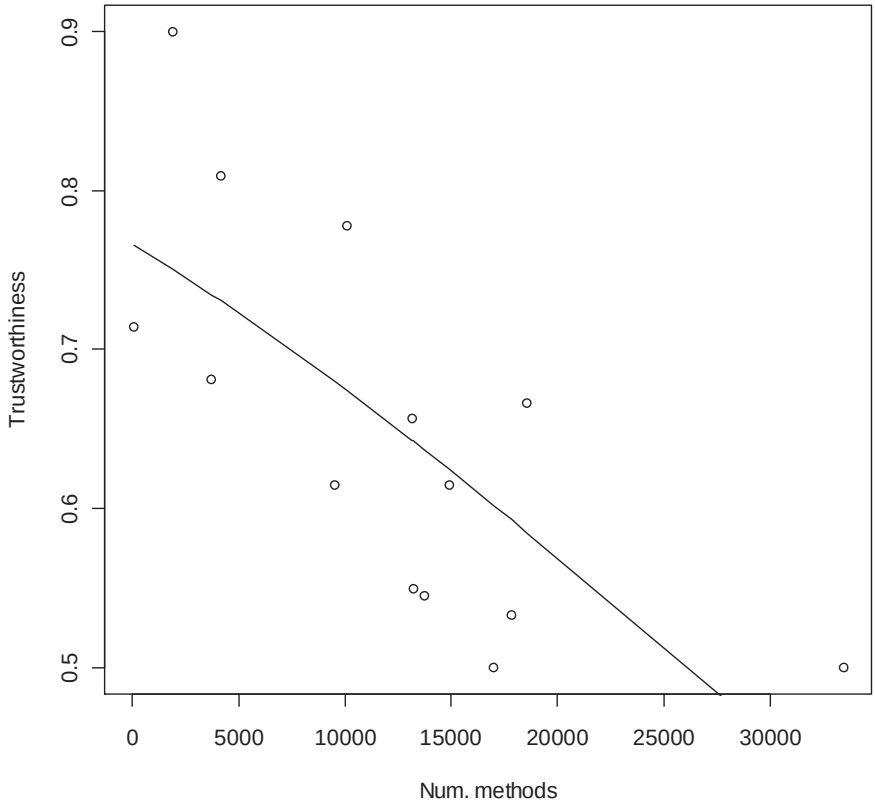

Figure 29. Trustworthiness vs. Number of methods: regression line

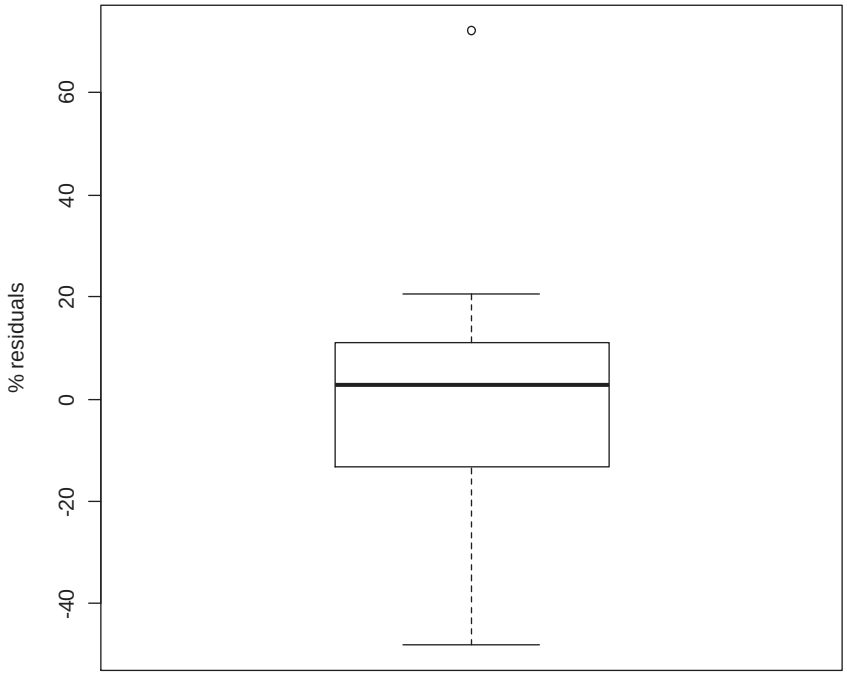

Figure 30. Trustworthiness vs. Number of methods: boxplot of relative residuals

7.3 Threats to Validity

A number of threats may exist to the validity of a correlational study like ours. We now examine some of the most relevant ones.

Like with any other correlational study, the threats to the external validity of our study need to be identified and assessed. The most important issue is about the fact that our sample may not be fully "balanced," and that may have somewhat influenced the results. While this may be true, the following points need to be taken into account.

- It was not possible to interview several additional people that could have made our sample more "balanced," because they were not available or had no or little interest in answering our questionnaire.
- 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.
- The most popular products were assessed by several users, while only a few evaluations were collected about specific-purpose products. This effect is

clearly very difficult to avoid. In any case, we excluded from the evaluation the products that had been evaluated by too few users.

- Like in many correlational studies, we used a so-called "convenience sample," composed of respondents who agreed to answer our questions. We collected information about the respondents' experience, application field, etc., but we did not make any screening. Excluding respondents based on some criteria, which must have been perforce subjective, may have resulted in an "unbalanced" sample, which may have biased the results.
- We dealt with motivated interviewees, so this ensured a good level for the quality of responses.
- There is no researcher's bias in our survey, since we simply wanted to collect and analyze data from the field, and not provide evidence supporting or refuting some theory.

The measures of the products' code used in this study concern a specific release for each product, namely the most recent release available. On the contrary, when we collected the users' opinions about the qualities of products, we did not ask for the release being evaluated. This choice was due to the considerations that a) most users use the most recent release available; b) most users would not remember the exact version they are using; c) since products we investigated are quite mature, we do not expect relevant changes in quality between releases.

An additional threat concerns the fact that the measures used to quantify the relevant factors may not be adequate. This work deals with trustworthiness, which is an intrinsically subjective quality, so the only way to measure it is to carry out a survey.

As for reliability, quite a large number of measures have been proposed to represent it from an objective point of view. However, here we are dealing with the users' perception of reliability, so, again, a survey is adequate to collect information about this quality. The correlation of the subjective perception of reliability with the traditional measures of reliability (e.g., defect density) is a possible subject for future work.

7.4 Discussion

The activities reported above were largely successful, in the sense that they identified the existence of several statistically valid models of the subjective qualities as functions of the internal, objectively measurable qualities.

The analyses reported in Appendix F and Appendix G allow us to state that the qualities of OSS products that are subjectively evaluated by users can be linked to the internal, measurable qualities of the software both for Java than for C/C++ OSS products.

In fact, two kinds of quantitative, statistically significant models were derived:

- monovariate models that correlate a subjective quality with a single objective quality
- multivariate models that correlate a subjective quality with several objective qualities

As an example of monovariate model, it was found that the trustworthiness of OSS products decreases as the size, expressed as number of classes, increases both for Java than C/C++ projects.

As summarized in Table 19 we found a monovariate correlation for each perceived quality except for the documentation quality and the perceived trustworthiness with respect to other OSS products.

Perceived quality (short name)	Objective Quality (metrics)
Usability	Number of methods per Interface
Portability	Number of methods per class
Functional Requirements	Number of interface per class
Interoperability	Coupling Between Objects (CBO)
Reliability	Number of Packages
Security	Effective Lines of Code (eLOC)
Community	Coupling Between Objects (CBO)
Documentation	
Fastness	Number of attributes per class
Trust_wrt_oss	
Trust_wrt_non_oss	Ciclomatic Complexity (Mc Cabe)
Trustworthiness	Number of classes

Table 19. Univariate Correlation for java OSS products

We also found out several multivariate models. As an example, this are the seven models we identified for trustworthiness:

1. LCOM and eLOC

2. RFC and LCOM
3. eLOC per class and Number of interface per class
4. Number of packages and eLOC per class
5. eLOC and Number of attributes per class
6. CBO, Comment Line per class and Number of interfaces per class
7. Comment Lines. Ciclomatic Complexity, Number of attributes per class

Raw results and an explanation on how to read the results can be found in Appendix F for Java projects and Appendix G for C/C++ projects.

Chapter 8

Conclusions and future work

The evaluation of the trustworthiness of OSS is important because of the ever increasing importance of OSS in software development and practical applications. However, lacking objective measures, OSS users and stakeholders rely on their own somewhat subjective evaluations when deciding to adopt an OSS product.

Some trustworthiness evaluation methods have been proposed to let potential users assess the quality of OSS products before possibly adopting them. Such methods –like the OpenBQR [2] and other similar approaches [10][11][12]– typically face two problems: what are the factors that should be taken into consideration, and what is the relative importance of factors? Generally these decisions are left to the user, who has to choose the qualities in a usually long list and assign weights. So, the work reported here improves our knowledge of the user-perceived qualities and trustworthiness of OSS products and of trustworthiness models.

In this work we defined a trustworthiness model for OSS projects.

First we carried out a survey to identified a set of trustworthiness factors based the users' perception of trustworthiness and a number of other qualities of OSS products.

Then, in order to test the feasibility of deriving a correct, complete and reliable evaluation of trustworthiness on the basis of the factors identified, a set of well-known OSS projects, widely adopted and generally considered trustable, were chosen to be used as references. Afterwards, a first quick analysis was carried out, checking which factors were readily available on each project's web site. The idea was to emulate the search for information carried out by a potential user, who browses the project's web sites, but is not willing to spend too much effort and time in carrying out a complete analysis.

Based on the trustworthiness factors identified by means of the questionnaires, we defined a GQM plan for the trustworthiness evaluation.

We identified set of tool to collect objective data from OSS projects and we developed MacXim, a Java static code analysis tool.
We selected 22 Java and 22 C++ products, and we collected objective data by means of the identified tools and subjective data by means of more than 500 questionnaires. We studied their popularity, the influence of the implementation language on trustworthiness, and whether OSS products are rated better than CSS products.
Finally we look for existing correlations among objective and subjective data.

The activities reported above were largely successful, in the sense that they identified the existence of several statistically valid models of the subjective qualities as functions of the internal, objectively measurable qualities.
In fact, we identified two kinds of quantitative, statistically significant: 1) univariate models that correlate a subjective quality with a single objective quality and 2) multivariate models that correlate a subjective quality with several objective qualities.
As an example of univariate model, it was found that the trustworthiness of OSS products decreases as the size, expressed as number of classes, increases both for Java than C/C++ projects.

8.1.1 Usage of the results

The main result of the activity reported in this document does not consist just in having found that relationships to exist between trustworthiness (and reliability) and objectively measurable characteristics of the OSS. A really important point is that we were able to *quantify* 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 by a given quality of their OSS product) into threshold of quality metrics that must be met by their code.

The procedure for using the quantitative knowledge of relations is exemplified below, considering the dependency of trustworthiness on McCabe complexity.
For users the procedure is simple: given a product, if for instance its McCabe complexity is, 8 then a user can expect that the product will be satisfactory (with probability > 60%, see Figure 31).

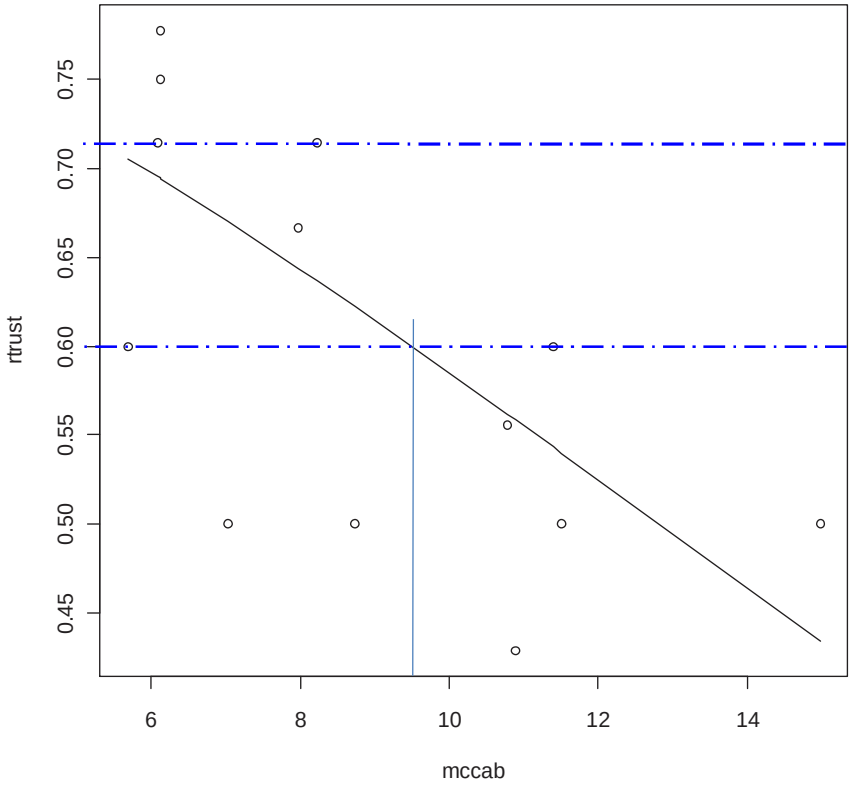

Figure 31. McCabe complexity corresponding to median and best quartile of Trustworthiness.

For developers the procedure is a bit more complex. In Figure 31 –which shows the function that links trustworthiness to McCabe complexity– the median value of the trustworthiness for the observed user population is reported, together with the highest quartile. It is easy to see that the median trustworthiness corresponds to a value of McCabe complexity between 9 and 10, while the 25% most trustworthy process are expected to have McCabe complexity below 6. Therefore, if the goal of the developer is

that its product is considered trustworthy by over 75% of the users, he/she must aim at a complexity not greater than 6. On the contrary, if the goal of the developer is that to satisfy the majority of the users with respect to trustworthiness, he/she must aim at a complexity not greater than 10.

In conclusion, unlike several discussions that are based on –sometimes interested– opinions about the quality of OSS, this study aims at deriving statistically significant models that are based on repeatable measures and user evaluations provided by a reasonably large sample of OSS users.

Publications

[1] D.Taibi, L.Lavazza, S.Morasca: "OpenBQR: a framework for the assessment of OSS" in Open Source Development, Adoption and Innovation - Springer - pp 173-186

[2] D. Taibi, M.Chinosi, V.Del Bianco, L.Lavazza: "A GQM plan for the evaluation of the trustworthiness of open-source software" TOSS, 1st International Workshop on Trust in Open Source Software, Co-located with OSS 2007, Limerick, Ireland, 14 June 2007.

[3] D. Taibi, M. Chinosi, L. Lavazza:BlogMeter: Web Sentiment Platform, TOSS 1st International Workshop on Trust in Open Source Software, Co-located with OSS 2007, Limerick, Ireland, 14 June 2007.

[4] V. del Bianco, M. Chinosi, L. Lavazza, S. Morasca, D. Taibi: How European software industry perceives OSS trustworthiness and what are the specific criteria to establish trust in OSS, October 2007, available on-line at http://wiki.qualipso.org/xwiki/bin/view/Wiki/WP51.

[5] 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" published in OSS 2008 proceedings

[6] V.Del Bianco, L.Lavazza, S. Morasca, D. Taibi: "Analysis of relevant open source projects and artefacts", QualiPSo report, April 2008

[7] D.Taibi: "Defining an Open Source Software Trustworthiness Model", 3rd International Doctoral Symposium on Empirical Software Engineering, Co-Located with ESEM (Empirical Software Engineering and Measurement)

[8] V.Del Bianco, L.Lavazza, S. Morasca, D. Taibi: "Open Source:mi devo fidare?" Conferenza Italiana sul Software Libero 2008 - Trento

[9] V.Del Bianco, L.Lavazza, S. Morasca, D. Taibi: "The observed characteristics and relevant factors used for assessing the trustworthiness of OSS products and artefacts" October 2008

[10] V.Del Bianco, L.Lavazza, S. Morasca, D. Taibi: "Report: How the trustworthiness of OSS products and artifacts can be assessed and predicted"

[11] V.Del Bianco, L.Lavazza, S. Morasca, D. Taibi: "Analysis of relevant OSS products and artifacts"

[12] S.Morasca, M.Proto, D.Taibi: "Quantitative Analysis of Reliability and Defects of Sourceforge Projects" Submitted to OSS2009

[13] M.Keikha, S.Gerani, M.Carman, R.Gwadera, D.Taibi, F. Crestani: "University of Lugano at TREC 2008 Blog Track"

[14] V. del Bianco, L. Lavazza, S.Morasca, D.Taibi: "Quality of Open Source Software: The QualiPSo Trustworthiness Model". OSS 2009 – published in IFIP International Federation for Information Processing

[15] S.Morasca, D.Taibi, D.Tosi: "Towards Certifying the Testing Process of Open-Source Software: new challenges or old methodologies?" Emerging Trends in FLOSS Research and Development Co-located with ICSE 2009 Vancouver, Canada (http://icse.libresoft.es/)

[16] L.Lavazza, S.Morasca, D. Taibi, D.Tosi: “La certificazione dei portali web che ospitano software open source” Conferenza Italiana sul Software Libero 2009

[17] L.Lavazza, S.Morasca, D. Taibi, D.Tosi: "Quality 2010 - oss 2010 - An Investigation of the users perception of OSS quality ". OSS 2010 – Notre Dame

[18] L.Lavazza, S.Morasca, D. Taibi, D.Tosi: "T-DOC: a Tool for the Automatic Generation of Testing Documentation for OSS Products". OSS 2010 – Notre Dame

[19] L.Lavazza, S.Morasca, D. Taibi, D.Tosi: “Applying SCRUM in an OSS Development Process”. XP2010, Trondheim.

[20] V.Del Bianco, L.Lavazza, S.Morasca, D. Taibi, D.Tosi: “A Survey on the Importance of Some Economic Factors in the Adoption of Open Source Software“. SERA (Software Engineering Research, Management & Applications) 2010, Montreal

[21] L.Lavazza, S.Morasca, D.Taibi, D.Tosi: “Predicting OSS Trustworthiness on the Basis of Elementary Code Assessment.” International Symposium on Empirical Software Engineering and Measurement (ESEM). Bolzano, Italy

[22] V.Del Bianco, L.Lavazza, S.Morasca, D.Taibi, D.Tosi: “The QualiSPo approach to OSS product quality evaluation” Emerging Trends in FLOSS Research and Development Co-located with ICSE 2010 Cape Town, South Africa

[23] D.Taibi: “Floss. To Trust or not to trust?” Workshop organization. fOSSA 2010, Grenoble, France

[24] V.Del Bianco, L.Lavazza, S.Morasca, D.Taibi, D.Tosi: *“A Survey on Open Source Software Trustworthiness: Which Factors Influence Trust in OSS?” Emerging Trends in FLOSS*” IEEE Computer. Accepted for pubblication

References

[25] Anderson J. P. Computer Security Technology Planning study, ESD-TR-73-51, Vol1, AD758 206, ESD/AFSC, Hanscom AFB, Bedford, M.A., October, 1972

[26] ISO/IEC, Information Technology-Security Techniques- Evaluation Criteria for IT security, Pat1: Introduction and genera Model 2nd ed. 2005-10-01

[27] Trusted computing group. TCG Architecture overview. V1. 2, 28 April 2004

[28] Basili V., and Rombach H.D., The TAME project: towards improvement-oriented software environments, IEEE Transactions on Software Engineering, June 1988.

[29] V. Basili and D. Weiss, “A methodology for collecting valid software engineering data,” IEEE Transactions on Software Engineering, vol. SE-10, no. 6, pp. 728-738, 1984.

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

[31] C. Mundie, B.Gates: How Microsoft Is Refocusing on Security, Reliability, Privacy, and More, as Part of Trustworthy Computing Initiative, February 2002 (http://www.microsoft.com/PressPass/features/2002/feb02/02 - 20mundieqa.mspx)

[32] Amoroso, E., Taylor, C., Watson, J., and Weiss, J. "A process-oriented methodology for assessing and improving software trustworthiness". In Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 39-50, 1994.

[33] Neumann, P. G., "Robust Nonproprietary Software", In Proceedings of the IEEE Symposium on Security and Privacy, pp. 122-123, 2000.

[34] Hertzum, M. "The importance of trust in software engineers’ assessment and choice of information sources". In Information and Organization, vol. 12, no. 1, pp. 1-18, 2002.

[35] "Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU". UNU-Merit report, 2006.

[36] Bernstein, L., "Trustworthy software systems", ACM SIGSOFT Software Engineering Notes, vol. 30, no. 1, pp. 4-5, 2005.

[37] Dustin, E., "The Software Trustworthiness Framework", Zero in a Bit, 30 January 2007.

[38] Fuggetta, A., "Open source software, an evaluation", The Journal of Systems and Software, vol. 66, pp. 77-90, 2003.

[39] Wheeler, D. A., "Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers", dwheeler.com Report, 2007.

[40] Fisher, R. A., "On the interpretation of $\chi 2$ from contingency tables, and the calculation of P", Journal of the Royal Statistical Society, vol. 85, no. 1, pp. 87-94, 1922.

[41] Abdi, H., "Binomial Distribution: Binomial and Sign Tests", in Encyclopedia of Measurement and Statistics, Thousand Oaks, 2007.

[42] Mann, H. B., Whitney, D. R., "On a test whether one of two random variables is stochastically larger than the other", in Annals of Mathematical Statistics, vol. 18, pp. 50-60, 1947.

[43] Mood, A. M., "Introduction to the Theory of Statistics", third edition, McGraw-Hill, 1974.

[44] The Open Source Initiative. Internet Website. Accessed on June 2007. Available online via http://www.opensource.org/.

[45] The Free Software Foundation. Internet Website. Accessed in June 2007. Available online via http://www.fsf.org.

[46] Raymond, E.S., "Goodbye, free software; hello, open source", internet website, accessed in June 2007, available online via http://www.catb.org/esr/open-source.html.

[47] European Working Group on Libre Software, Internet Website, accessed in June 2007, available online via http://eu.conecta.it.

[48] European Commission. "Free/Libre and Open Source Software: Survey and Study", Internet Website, accessed in June 2007, available online via http://www.infonomics.nl/FLOSS.

[49] Cerri, D. and Fuggetta, A., "Open Standards, Open Formats, and Open Source." CEFRIEL, Politecnico di Milano, version 5.0 final draft, January 2007.

[50] Fuggetta, A., "Open Source and Free Software: A New Model for The Software Development Process?", UPGRADE, European Journal for the Informatics Professional, Vol. V, No. 5, October 2004.

[51] Antikainen M., "Is trust based on cognitive factors in OSS communities?", Trust in Open-Source Software (TOSS) 2007, Limerick, June 2007.

[52] German D. M., Gonzales-Barahona J. M., Robles G., "In what do you trust when you trust? The importance of dependencies in trust analysis", Trust in Open-Source Software (TOSS) 2007, Limerick, June 2007.

[53] Hansen M. Ks., Úhntopp K., Pfitzmann A., "The Open Source Approach – Opportunities and Limitations with Respect to Security and Privacy". Computers & Security, vol. 21/5, pp. 461-471, 2002.

[54] Hasselbring W., Reussner R., "Toward Trustworthy Software Systems", US Army Research Laboratories Information Assurance Center, IEEE. 2006.

[55] Lawrieand T., Gacek C., "Issues of Dependability in Open Source Software Development", ACM SIGSOFT, Software Engineering Notes, vol. 27, n. 3, pp 34, May 2002.

[56] Goode S., "Something for Nothing: Management Rejection of Open Source Software in Australia's Top Firms", Information & Management, vol. 42, pp. 669 – 681, 2005.

[57] "A Collaborative Fact Finding Study. Open Source Business Opportunities for Canada's Information and Communications Technology Sector", e-Cology Corporation, available on-line via http://www.e-cology.ca/canfloss/report, 2003.

[58] "Business Readiness Rating for Open Source", OpenBRR.org, BRR2005 - RFC 1, accessed in June 2007, available on-line via http://www.openbrr.org.

[59] "Method for Qualification and Selection of open Source software (QSOS)", Atos Origin, version 1.6, accessed June 2007, available on-line via http://www.qsos.org.

[60] Golden B., "Making Open Source Ready for the Enterprise: The open Source Maturity Model", extracted from "Succeeding with Open Source", Addison-Wesley, 2005.

[61] Duijnhouwer F. W., Widdows C., "Open Source Maturity Model", accessed June 2007, available on-line via http://www.seriouslyopen.org.

[62] Ruffatti G., Oltolina S., Tura D., Damiani E., Bellettini C., Colombo A., Frati F., "New Trends Towards Process Modelling: Spago4Q", Trust in Open-Source Software (TOSS), Limerick, June 2007.

[63] Weiss D., "Measuring Success of Open Source Projects Using Web Search Engines", in proceedings of the Open-Source Software Conference (OSS) 2005.

[64] Hahn R.W. (editor), "Government Policy toward Open Source Software", 2002.

[65] Krishnamurthy S., "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects", 2002.

[66] Stamelos I., Angelis L., Oikonomou A., Bleris G., "Code quality analysis in open source software development", Systems J, volume 12, pp. 43-60, 2002.

[67] "Object Management Group (OMG)", Internet Website, accessed in June 2007, available on-line via http://omg.org/.

[68] "ISO/IEC-9126:2001 Information technology, Product Quality", International Standard ISO/IEC 9126, International Standard Organization, June, 2001.

[69] "Centro Nazionale per l'Informatica nella Pubblica Amministrazione (CNIPA)", Internet Website, accessed in June 2007, available on-line via http://www.cnipa.gov.it.

[70] "Osservatorio Open Source (OOS)", Internet Website, accessed in June 2007, available on-line via http://www.osspa.cnipa.it.

[71] "Riusabilità del software e delle applicazioni informatiche nella pubblica amministrazione", CNIPA, 2004.

[72] Cattaneo F., del Bianco V., Fuggetta A., "Fattori abilitanti al riuso del software nella pubblica amministrazione", CEFRIEL, 2004.

[73] "OpenPA", Internet Website, accessed in June 2007, available on-line via http://www.lugroma.org/openpa.

[74] "Consorzio per il Sistema Informativo (CSI)", Internet Website, accessed in June 2007, available on-line via http://www.csipiemonte.it.

[75] "Piattaforma per la gestione documentale OASI", Internet Website, accessed in June 2007, available on-line via http://www.csipiemonte.it/progetti/oasi.

[76] "Strategie Digitali", Internet Website, accessed in April 2007, available on-line via http://www.strategiedigitali.it.

[77] "FLOSS Piemonte", Internet Website, accessed in April 2007, available on-line via http://floss.piemonte.it.

[78] Fenton N., Pfleeger S. L., "Software metrics: a rigorous and practical approach", PWS Publishing, 1998.

[79] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M., Characteristics of Software Quality, North Holland, 1978.

[80] Boehm, Barry W., Brown, J. R, and Lipow, M.: Quantitative evaluation of software quality, International Conference on Software Engineering, Proceedings of the 2nd international conference on Software engineering, 1976.

[81] McCall, J. A., Richards, P. K., and Walters, G. F., "Factors in Software Quality", Nat'l Tech. Information Service, Vol. 1, 2 and 3, 1977.

[82] J. Smith David, D. Schuff, R. St. Louis, "Managing your total IT cost of ownership", Communications of the ACM, Volume 45, n. 1 (January 2002).

[83] R. Kaplan and D. Norton, The Balanced Scorecard: Translating Strategy into Action, Harvard Business School Press, September 1996.

[84] Dave Zubrow, Measuring Software Product Quality: the ISO 25000 Series and CMMI, European SEPG, June 14, 2004

[85] V. Basili, G. Caldiera, and D. Rombach, "Goal/Question/Metric Paradigm," in En-cyclopedia of Software Engineering, vol. 1, J. C. Marciniak, Ed.: John Wiley & Sons, 1994, pp. 528-532.

[86] R. van Solingen and E. Berghout, The Goal/Question/Metric Method, McGraw-Hill, 1999. http://www.gqm.nl/

[87] Sandro Morasca, "On the Use of Weighted Sums in the Definition of Measures." WETSoM, Cape Town, South Africa, May 4, 2010.

[88] 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

[89] The Merriam Webster online Dictionary. http://www.merriam-webster.com/

[90] Hyperdictionary online dictionary. http://www.hyperdictionary.com/

[91] Wikipedia. http://www.wikipedia.org

Appendix A: The questionnaire on OSS selection

The purpose of this questionnaire is to elicit information from the users of FLOSS products about their goals when they use/customize/modify/develop FLOSS products and about their development and FLOSS product selection processes.

This questionnaire has been developed in the framework of the QualiPSo (Quality Platform for Open Source Software) project, which is a European Union-funded Integrated Project which aims at making a major contribution to the state of the art and practice of Open Source Software. The QualiPSo project started in November 2006 and will last until October 2010. The project brings together over twenty software companies, application solution developers, and research institutions. Its goal is to define and implement technologies, procedures and policies to leverage the Open Source Software development current practices to sound, well-recognized, and established industrial operations.

All information provided by each individual or organization will be treated as confidential. As such, it will not be released in other form than aggregated statistical analyses that will make it impossible to identify the single respondents.

Please do not hesitate to contact us if you need any information or clarification.

Personal information

- Name:
- Role:
- Unit:
- Education:
- Time in the company:
- E-mail:

Company information

- Type of organization (private, no profit, Public Administration, etc.):
- Number of employees:
- Domain(s) (Public Administration, avionics, banking/finance, …):
- Number of employees of the organizational unit:

- Domain(s) (Public Administration, avionics, banking/finance, …) of the organizational unit:

Role of the organization with respect to OSS

- Is the company a producer, user, mixed (user/modifier), value adder (customizer, …) of OSS?
- Choose all that applies:
  1. OSS products are used to support SW development
  2. OSS products are used as part of other product
  3. OSS products are customized/configured
  4. OSS products are used to support the internal process
  5. OSS products are used to provide services to the outside world.
- Is OSS the development platform?
- Is OSS the target/usage platform?

Issues that can be taken into account when deciding whether to adopt OSS

Economics

- Do you choose OSS considering (please rank, from 0-irrelevant to 10-essential)
  1. The TCO (Total Cost of Ownership)? E.g., is OSS used because it is less expensive then commercial alternatives?
2. The ROI (Return On Investment)? E.g., is OSS chosen to reduce effort?
3. Any other issues related to your business model?

License

- What types of licenses do you have in the OSS you deal with?
  - Academic Free License
  - Adaptive Public License (APL)
  - Apache Software License
  - Apple Public Source License
  - Artistic License
  - Attribution Assurance Licenses
  - BSD License
  - Computer Associates Trusted Open Source License
  - Common Development and Distribution License
  - Common Public License

- CUA Office Public License
- EU DataGrid Software License
- Eclipse Public License
- Educational Community License
- Eiffel Forum License
- Entessa Public License
- Fair License
- Frameworx License
- GNU General Public License (GPL)
- GNU Lesser General Public License (LGPL)
- Historical Permission Notice and Disclaimer
- IBM Public License
- Intel Open Source License
- Jabber Open Source License
- Lucent Public License
- MIT License
- MITRE Collaborative Virtual Workspace License (CVW License)
- Motosoto License
- Mozilla Public License (MPL) 1.0 and 1.1
- NASA Open Source Agreement
- Naumen Public License
- NetHack General Public License
- Nokia Open Source License
- OCLC Research Public License
- Open Group Test Suite License
- Open Software License
- PHP License
- Python License
- Python Software Foundation License
- Qt Public License (QPL)
- RealNetworks Public Source License
- Reciprocal Public License

- Ricoh Source Code Public License
- Sleepycat License
- Sun Industry Standards Source License (SISSL)
- Sun Public License (SPL)
- Sybase Open Watcom Public License
- University of Illinois/NCSA Open Source License
- Vovida Software License v. 1.0
- W3C License
- wxWindows Library License
- X.Net License
- zlib-libpng license
- Zope Public License
- Other _____

- What should the license allow/restrict to users, developers, modifiers, integrators?
  - Hackers dislike accepting code under it
  - Cannot combine with proprietary and redistribute
  - Cannot combine with GPL'ed code and redistribute
  - Can redistribute binaries without source
  - Apply to everyone who receives the program, without the need for any additional agreements
  - Allow distribution with any other software agreements
  - Allow distribution in any form
  - Grant to distribute the program themselves, including the right to charge money for it
  - Grant the right to distribute modified versions of the program
  - Grant access to the program's source code
  - Grant the right to modify the program

Selection Process

- Do you have a process for selecting OSS to use?
- If so, what is it like?
- Which OSS evaluation methods do you use?
  - QSOS (www.qsos.org)
  - OpenBRR (www.openbrr.org)

- OSMM – Navica (www.navicasoft.com/pages/osmm.htm)
- OSMM – Capgemini (www.SeriouslyOpen.org)
- OpenBQR (http://www.taibi.it/OpenBQR)

- What is the context process in which it is used?
- Do you choose OSS products considering (please rank, from 0-irrelevant to 10-essential)
  1. the type of licenses used?
  2. the availability of tools for developing/modifying/customizing … OSS products?
  3. the availability of best practices on the specific OSS products?
  4. the availability of technical documentation/user manual?
  5. environmental issues (platforms, preferences and needs of personnel, …)?
  6. the availability of training, guidelines, etc.?
  7. the mid/long term existence of a user community?
  8. the mid/long term existence of a maintainer organization / "sponsor"?
  9. the short term support (problem resolution, correction of bugs, etc.)?
  10. the reputation of the OSS provider?
  11. the programming language uniformity?
  12. the existence of a sufficiently large community of users of the OSS software that can witness its quality?
  13. the existence of benchmarks, test suites that witness for the quality of OSS?
  14. other (please specify)?
- What other characteristics that are not commonly available about OSS development processes would you like to have and use?

Product quality

- Do you choose OSS products considering (please rank, from 0-irrelevant to 10-essential)
  1. the degree to which an OSS product satisfies/covers functional requirements
  2. the degree to which other qualities are satisfied, e.g., the qualities of ISO9126
     1. reliability
     2. performance
     3. usability
     4. maintainability
     5. portability
     6. other (e.g., reusability)

3. design and code qualities:
    1. size
    2. complexity
    3. modularity
    4. standard architecture
    5. patterns
    6. other (Please specify)
4. standard compliance
5. self-containedness (the product does not need other "products" to work correctly)
6. the interoperability (data level, formats, etc.)
7. the human interface language / localization of the OSS product

- What other characteristics that are not commonly available about OSS product quality would you like to have and use?

Features supporting the customer requirements

- What features do you take into account when choosing OSS? (please rank, from 0-irrelevant to 10-essential)

1. Customer satisfaction
2. Interoperability issues
3. Law conformance (e.g., for Public Administrations)
4. Standard imposed
5. other (please specify)

Processes

Trust

- What are the elements (practices, tools, techniques, etc.) in the process that allow you to trust the quality of the final result?

Quality assurance

- What are the aspects for verifying quality of he product you use/produce?
- Who is testing the product?
- Which manually test methods are used? (internal/user testing)
- Which automated testing techniques are used?
- How often, how much and what do you test?
- Are new releases scheduled?
- How regularly are releases rolled out?
- Is it planned in which release which :

- Features will be added?
- Bugs will be solved?

- How is the work managed in the time of delivering a new release?

General questions

- Which open source software are used within the company/unit?
- If there is a commercial alternative available, why do you choose OSS?
- Is an OSS product usually used/developed/modified/customized in a single location within the company or at several locations?
- When did the project start?
- Where did the project start?
  - Within the company?
  - Did the project already have roots/backgrounds (outside of the company), that the company improved?
- How long does it last (approximately)?

Roles and responsibilities

- How many people were/are working in the project?
  1. 1-15
  2. 16-25
  3. 26-50
  4. 51-100
  5. 101-500
  6. More than 500
- How much is the turnover? (annual rate of people getting into/leaving the project)
  1. 1%-10%
  2. 11%-20%
  3. 21%-40%
  4. 41%-60%
  5. 61%-80%
  6. 81%-100%
- Please determine:
  - The standard roles:
    1. users (yes/no)
    2. developers (yes/no)

3. committers (yes/no)
4. PMC members (yes/no)
5. other (yes/no)

- The number of the participants of the project:
  1. users
  2. developers
  3. committers
  4. PMC members
  5. Other
- The responsibilities:
  1. users
  2. developers
  3. committers
  4. PMC members
  5. Other

- How can one become a developer, committer, PMC member?
- Is there any community within or outside of the company which makes decisions?
- How are decision processes arranged?
- How do you decide about code modification, giving rights, package releases, etc? (voting, responsibilities, etc.)

Architecture definition

- How is the technical architecture of the project managed?
- Is it planned before, incremental?
- What are the most important technical requirements?
- Which technologies are used?

Development techniques and practices

- Which development methodology do you use?
- Can you describe it? (if it is not standard)
- Which practices do you use? (describe it)
  - Test first
  - Unit test
  - Continuous integration
  - Code reviews

- Other (please specify)

- How do you collect and manage requirements?
- Do you use any coding standards?
- How is the maintenance of the existing code worked out?

Tools used

- On which operating system is the project implemented?
- Is it running on other OS?
- If yes, on which one(s)?
  - Windows
  - Linux
  - Solaris
  - Other (please specify)
- Which programming language is used for the implementation?
  - Java
  - C++
  - C
  - Visual Basic 6
  - Perl
  - Pyton
  - Other (Please specify)
- On which platform?
  - Windows
  - Linux
  - Solaris
  - Other (please specify)
- Which development tools are used in the project?
  - Eclipse
  - Visual Studio
  - Vi
  - Emacs
  - Other (please specify)
- Do you use any tool developed in house? (yes, no)

- Do you make these tools available to others? (yes, no)
- Do you use other open source or commercial software? (yes/no)

Features to implement

- Considering the new features; Who:
  1. Makes suggestions for new features? (Is there any mailing list/newsgroups for doing this?)
  2. Is deciding about new features?
  3. Has to implement the new features?
- Is there a time plan
  1. For implementing the features?
  2. Which feature should be implemented first? (ranking of features by priorities)
  3. How priorities are assigned?

Documentation, bug management

- Do you have documentation of the project?
- Who writes the documentation and where? (in the implementation, in a separate documentation, etc.)
- Does the project have a roadmap?
- Is it useful for the developers?
- Which tools are used for bug-tracking?
- If there are several in use, which tool has the highest priority?
- Are the bug-tracking tools specialized for different persons (users, developers, etc), or do they use the same tool for reporting bugs?
- How many bug reports do you get?
- Can the bug-tracking tool be used for other purposes too? (e.g.: making suggestions, looking for tasks to resolve them, etc.)
- How long does it take to solve a bug?
- How are priorities assigned?

Version control and people management

- Which version control system is used for the project?
- Is this tool freely available for everybody (user, company, etc.)?
- Who has access to the version control system and which rights?
- Who and how can get more rights and which ones?
- Who can be the owner of a module?
- How are the tasks assigned? Can one choose what to implement?

Business model

- Are developers employee?
- Which advantages/disadvantages, benefits has the developer for contributing?
- What is the goal of the project?
- Does the company sell this product?
- Are there any additional services (e.g. courses, support, extensions, etc)?
- If yes, which one(s)?

Workflows of the processes identified

- Please describe the following processes:
  1. Development techniques
  2. Release development
  3. Testing
  4. Quality assurance

Appendix B: Trustworthiness Factors Analysis

The main goal of the analysis is to obtain the information that is quickly available through a project's website.

Some factor have been analyzed, while others need some tools to be developed.

In this section, we analyze all the factors and their measures reported in Table 4 referring to the set of 32 projects. We correlate users and developers requirements with the actual availability of the trustworthiness factors into web portals. Finally, we provide some guidelines useful to developers of OSS products in order to better highlight trustworthiness factors into their web portals.

The next sections reflect the structure of the Questionnaire presented in Appendix A with reference to the four categories: Economics, Development, Quality, and Customer.

Economics: Economic Issues When Choosing OSS

Here, we do not analyze these factors due to their subjectivity.

Developments: OSS Development Process

We collected information to understand both the main development related factors when choosing an OSS product, and also the available attributes that are taken into consideration. In this section, we try to identify the attributes that the selected projects currently provide to the final users.

The identified factors, hereafter analyzed, are:

- License Issues When Choosing OSS
- The availability of tools for developing modifying customizing OSS products
- The availability of best practices on the specific OSS products
- The availability of technical documentation / user manual
- Environmental issues
- The availability of training and guidelines
- The mid-long term existence of a user community
- The mid-long term existence of a maintainer organization / sponsor

- The short-term support
- The reputation of the OSS provider
- The distribution channel
- The programming language uniformity
- The existence of a sufficiently large community of users that can witness its quality
- The existence of benchmarks / test suites that witness for the quality of the OSS product

License Issues When Choosing OSS

As emerged in our questionnaire, the factor "The types of licenses used" takes a fairly high importance for developers and final users.

In Table 20 we show the distribution of the licenses types: The vast majority of the projects use a GPL/LGPL license (48% GPL and 17% LGPL); seven projects use an Apache License while the remaining projects use other types of licenses.

As expected, this important factor is properly reported into the analyzed web portals.

Type of license used	Values	Percentage
Apache Licence	7/32	24%
GPL	14/32	48%
LGPL	5/32	17%
CDDL	1/32	3%
CPL or EPL	1/32	3%
BSD	1/32	3%
Common Public Licence 1.0	1/32	3%
Apache BSD style	1/32	3%
Unknown	1/32	3%

Table 20: Type of licenses used

2010

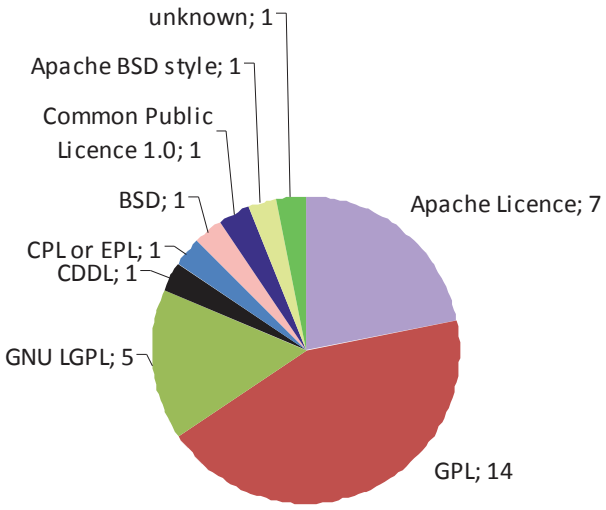

Figure 32: The license distribution

The availability of tools for developing modifying customizing OSS products

As emerged in our questionnaire, the factor "The availability of tools for developing modifying customizing OSS products" takes a fairly high importance for developers.

In Table 21 we summarize the data collected for the tools availability. We can see that more than 50% of the projects have special purpose-built tools and more than 75% of projects have some purpose-built tools and less than 35% of the projects have other useful tools.

As expected, this important factor is properly reported into the analyzed web portals.

Purpose-build tools for product	Values	Percentage
yes	17	53%
no	15	47%
Other useful tools	Values	Percentage
yes	11	34%
no	21	66%
Build customizing scripts	Values	Percentage
yes	21	66%
no	11	34%
Integration with development tools	Values	Percentage
yes	9	28%
no	23	72%
Documentation on customization	Values	Percentage
yes	24	75%
no	8	25%
Built in customization facilities	Values	Percentage
yes	25	78%
no	7	22%

Implementation with customization in focus	Values	Percentage
yes	8	25%
no	24	75%

Table 21: The availability of tools for developing modifying customizing OSS products

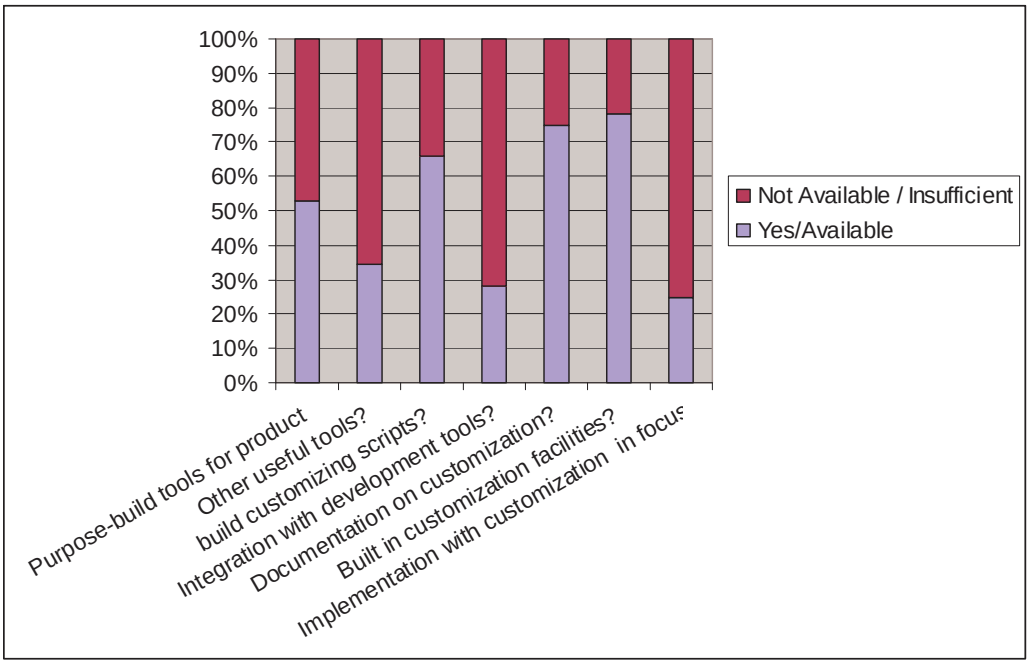

Figure 33: The availability of tools for developing modifying customizing OSS products

The availability of best practices on the specific OSS products

As emerged in our questionnaire, the factor "The availability of best practices on the specific OSS products" takes a low importance for users and developers.

In Table 22 we summarize the data collected for the availability of best practices.

As expected, best practices were not available mostly in all projects (only one project up to 32 shows best practices on its website) while more than half projects have some code examples listed in the website.

Best Practices Area	Values	**Percentage**
Yes	1	3%
No	31	97%
Code Examples	Values	**Percentage**
Yes	20	63%
No	12	38%

Table 22 : The availability of best practices on the specific OSS products

Figure 34: The availability of best practices on the specific OSS products

The availability of technical documentation / user manual

As emerged in our questionnaire, the factor "The availability of technical documentation / user manual" takes a low importance for users and developers.

In Table 23 we summarize the data collected for the availability of user documentation, while in Table 24 we summarize the availability of technical documentation.

As expected, almost every project has an up-to-date user documentation (manuals, getting started guides and installation guides) and there is a good level of interaction between users and developers by means of forums and mailing lists. Conversely, the existence of technical documentation is not so frequent: approximately half of the projects provide technical documentation, forums and mailing list, while only less than half of the projects have updated F.A.Q. and technical forums.

We point out that OSS users consider product and design documentation as a major issue. We suggest developers to make always available up-to-date technical documentation.

User manual	Values	Percentage
Updated	25	78%
Not Updated	3	9%
Not Available	5	16%
Getting started guide	Values	Percentage
Updated	18	56%
Not Updated	3	9%
Not Available	11	34%
User related F.A.Q.	Values	Percentage
Updated	23	72%
Not Updated	1	3%
Not Available	8	25%
Mailing list	Values	Percentage
Yes	27	84%
No	5	16%

Table 23: The availability of user documentation

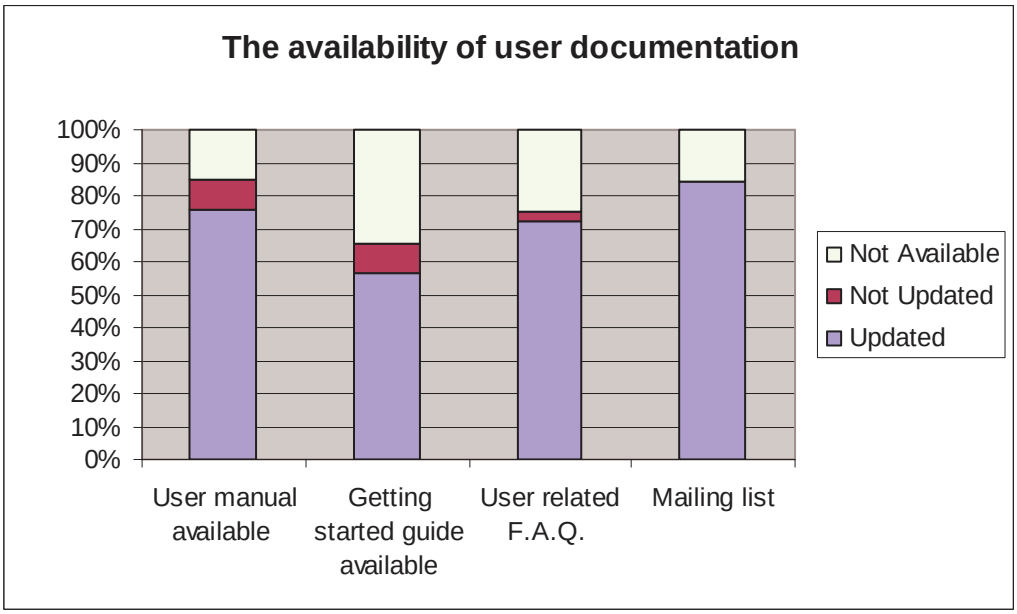

Figure 35: The availability of user documentation

Technical manual	Values	Percentage
Updated	16	50%
Not Updated	2	6%
Not Available	14	44%
Technical documentation (like Javadoc)	Values	Percentage
Yes	14	44%
No	18	56%
Installation guide	Values	Percentage
Updated	25	78%
Not Updated	2	6%
Not Available	5	16%

Technical related F.A.Q.	Values	Percentage
Updated	11	34%
Not Updated / unknown if related to the latest version	7	22%
Not Available	14	44%
Technical forum	Values	Percentage
Yes	12	38%
No	20	63%
Technical related Mailing list	Values	Percentage
Yes	16	50%
No	16	50%

Table 24: The availability of technical documentation

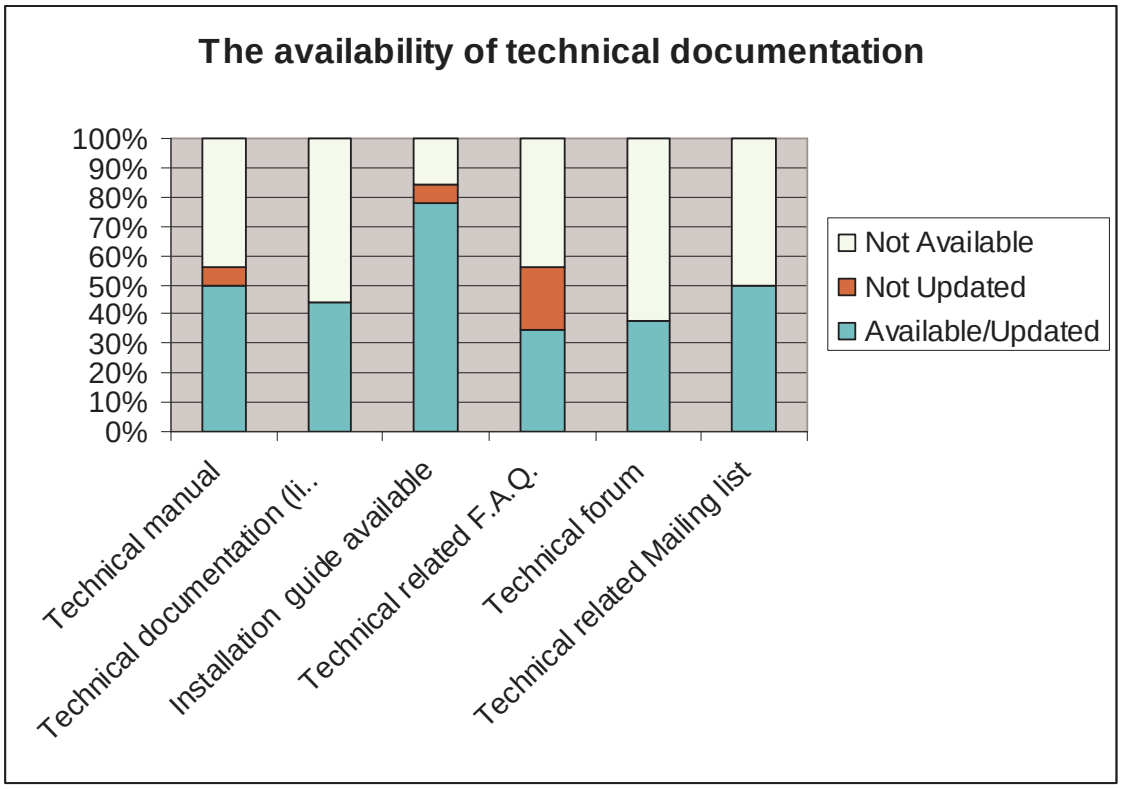

Figure 36: The availability of technical documentation

Environmental issues

Environmental issues describe software and hardware capabilities for each component of the environment. Due to the high subjectivity of this factor, we excluded it in our analysis.

The availability of training and guidelines

As emerged in our questionnaire, the factor "The availability of training and guidelines" takes a very low importance for both users and developers.

In Table 25 we summarize the data collected for the availability of training information and guidelines.

Unexpectedly, guidelines and training guides are mostly available and updated on the project websites. Only 7 project up to 32 have out-of-date guidelines and one project doesn't provide any guideline.

Considering the availability of official training courses, only 8 projects out of 32 provide it.

Availability of training, guidelines	Values	Percentage
Some updated materials	24	80%
Out of date materials	1	3%
No training materials	7	17%
Availability of official training course	**Values**	**Percentage**
Yes	8	30%
No	24	80%

Table 25: The availability of training and guidelines

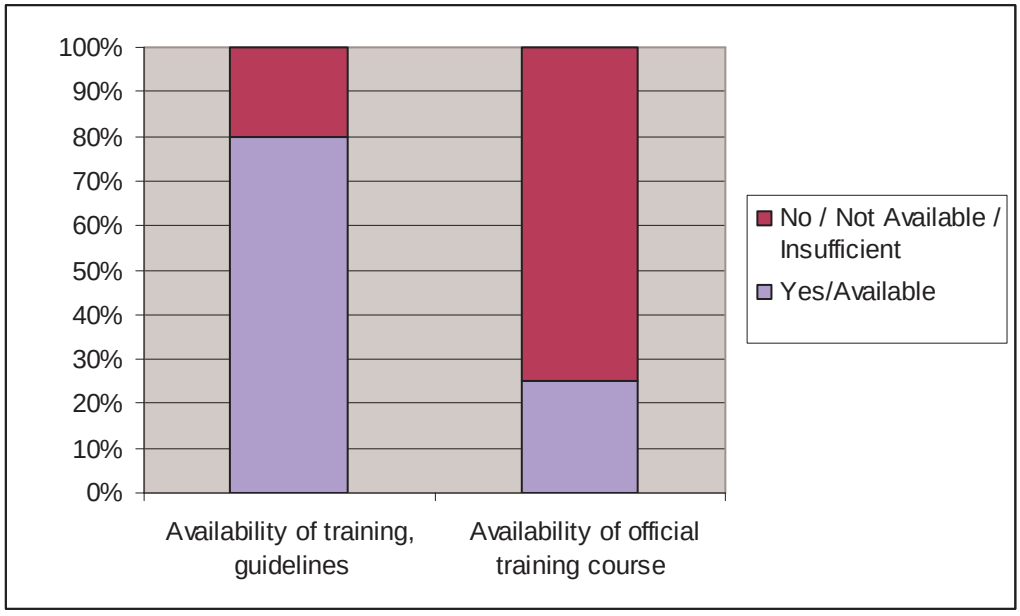

Figure 37; The availability of training and guidelines

The mid-long term existence of a user community

As emerged in our questionnaire, the factor "The mid-long term existence of a user community" takes a high importance for both users and developers. OSS users often pay attention to the vitality of the user community both in terms of its duration and also in terms of the number of people involved.

In Table 26 we summarize the data collected for the existence of a user community.

Unfortunately, the dimension of the user community is not measurable unless explicit data is provided on the website. In our analysis, only two projects up to 32 show the

community size. Considering the vitality of a community in correlation with the number of patches and releases, not all websites clearly show this data. Some websites show only the number of patches/releases of the last 6 months, others only the total number of patches/releases, while others show both data. An interesting result is provided by the availability of several community groups identified through different mailing lists (technical related, user related, translator related…).

Despite our expectation, data related to the size and the vitality of the communities is not well highlighted in the considered web portals. We suggest developers to clearly show this information.

Actual dimension of user community	Values	Percentage
Found	2	6%
not found	30	94%
Number of patches/releases (total)	Values	Percentage
0-25	13	41%
26-50	8	25%
>=50	5	16%
Not found	6	19%
Number of patches/releases (last 6 months)	Values	Percentage
0-5	17	53%
6-10	2	6%
>10	2	6%
Not found	11	34%
Project Age	Values	Percentage
0-5	18	56%
6-10	5	16%
>10	5	16%
Not found	4	13%
Age of the community	Values	Percentage
0-5	5	16%
6-10	2	6%
>10	0	0%
Not found	25	78%
Number of contributors of the community	Values	Percentage
0-5	5	16%
6-10	2	6%
>10	6	19%
Not found	19	59%
Existence of several different community groups	Values	Percentage
Many mailing lists	22	69%
Several user groups	6	19%
Not available	4	13%
Number of subscribers of the mailing lists	Values	Percentage
Available	1	3%
Not Available	31	97%

Table 26: The mid-long term existence of a user community

The mid-long term existence of a maintainer organization / sponsor

As emerged in our questionnaire, the factor "The mid-long term existence a maintainer organization / sponsor" takes a very low importance.

In Table 27 we summarize the data collected for the existence of a maintainer organization / sponsor.

The analysis carried out on this factor shows that the vast majority of the projects have several maintainers/sponsors. As shown in Table 27, only 8 projects out of 32 (25%) don't have a maintainer/sponsor or a supporting organization.

Active maintainer organization / sponsor	Values	Percentage
yes	24	75%
no	8	25%
Type of the maintainer	Values	Percentage
individuals	3	9%
small	6	19%
large	8	25%
all	3	9%
not found	12	38%
Supporting organizations	Values	Percentage
1-5	6	19%
6-10	2	6%
>10	11	35%
not found	13	41%

Table 27: The mid-long term existence of a maintainer organization / sponsor

The short-term support

As emerged in our questionnaire, the factor "The short term support" takes a fairly high importance for users and developers.

In Table 28 we summarize the data collected for the availability of short term support.

As expected, the short term support is mostly assessable. As we can see, most of the projects publish their bugs-tracker and provide professional services that can guarantee a short-term resolution of bugs.

Bug number available	Values	Percentage
Yes	15	47%
No	17	53%
Professional services	Values	Percentage
Yes	23	72%
No	9	28%
Bug tracking	Values	Percentage

Yes	25	78%
No	7	22%
Bug workflow	Values	Percentage
Yes	2	6%
No	30	94%

Table 28: The short-term support

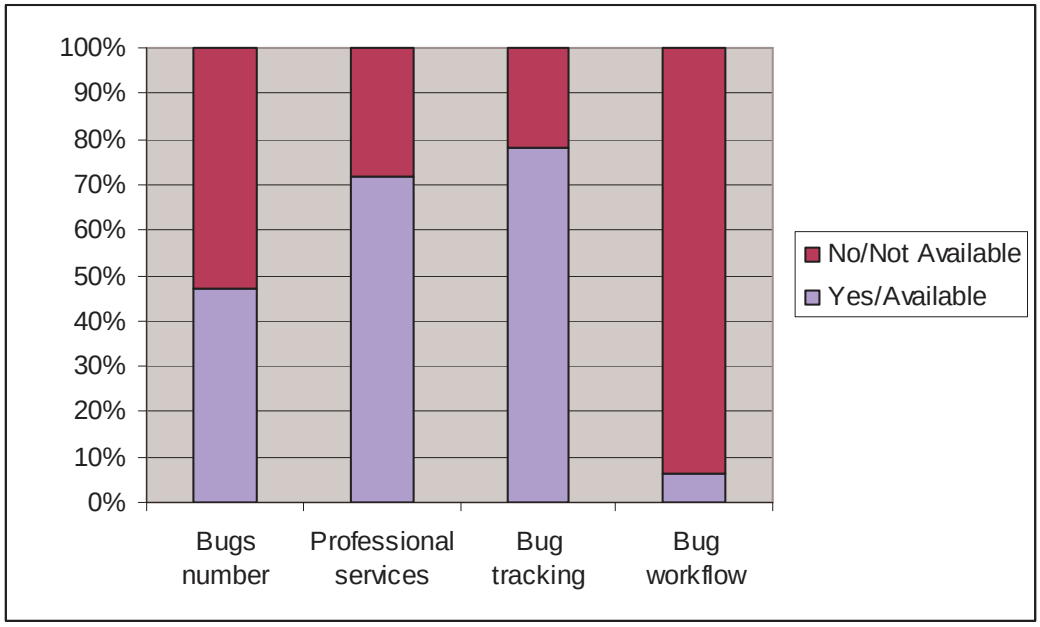

Figure 38: The short-term support

The reputation of the OSS provider

This factor is not analyzed, due to its high subjectivity.

The distribution channel

As emerged in our questionnaire, the factor "The distribution channel" takes a negligible importance.

In Table 29 we summarize the data collected for the available distribution channels.

As expected, all the projects freely provide their source code via internet. The big majority provides both the source code, the binaries and access to the source code repository.

Only few projects are available via CD/DVD or p2p networks (such as Torrent, or eMule).

Source code download	Values	Percentage
Yes	32	100%
No	0	0%
Binaries download	Values	Percentage
Yes	24	75%

No	8	25%
Repository access	Values	Percentage
Anonymous	28	88%
No	3	9%
Private Login	1	3%
CD/DVD	Values	Percentage
Yes	6	19%
No	26	81%
p2p	Values	Percentage
Yes	2	6%
no	30	94%

Table 29: The distribution channel

The programming language uniformity

As emerged in our questionnaire, the factor "programming language uniformity" takes a low importance.

In Table 30 we summarize the data collected for the language uniformity.

In this analysis, more than 60% of the projects use only one programming language, but only half of these projects explain why they use one language instead of another one, or why they use a variety of languages for their projects.

Only one language used	Values	Pencentage
Yes	20	63%
No	12	38%
Reasoning why different languages are used	Values	Pencentage
Yes	18	56%
No	14	44%

Table 30: The programming language uniformity

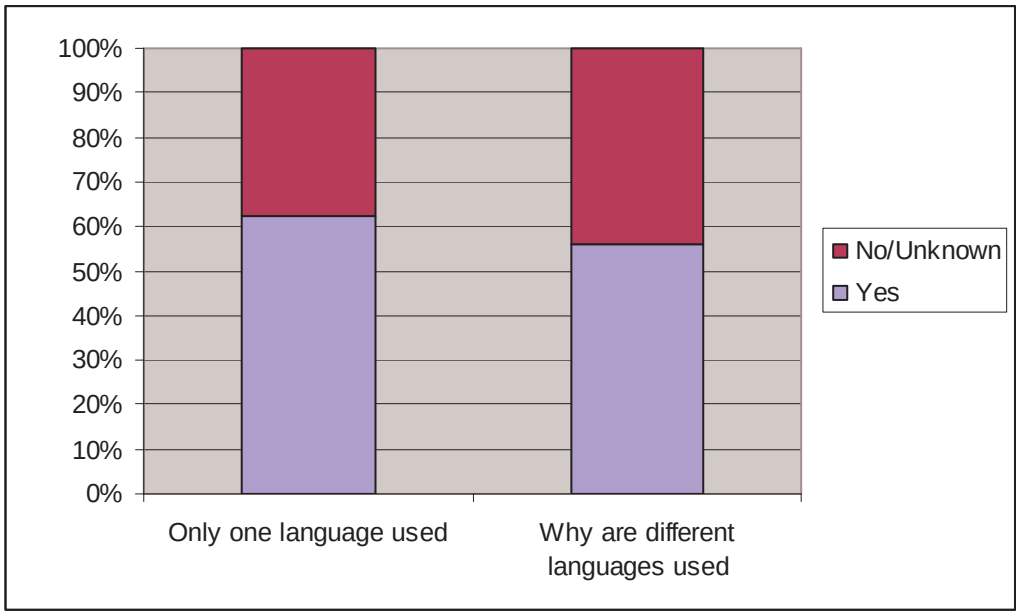

Figure 39: The programming language uniformity

The existence of a sufficiently large community of users that can witness its quality

As emerged in our questionnaire, the factor "The existence of a sufficiently large community of users that can witness its quality" takes a fairly high importance.

In Table 31 we summarize the data collected for the language uniformity.

Unexpectedly, most of the project do not provide an official forum (20 up to 32), while the others have forums with a lot of activity (in some cases with more than 100.000 posts).

We suggest developers to always maintain active forums and vital communities.

Number of post available on forums/blogs/newsgroup	Values	Percentage
0-5000	3	9%
5001-100000	6	19%
>100000	3	9%
unknown	20	63%

Table 31: The existence of a sufficiently large community of users that can witness its quality

The existence of benchmarks / test suites that witness for the quality of OSS

As emerged in our questionnaire, the factor "The existence of benchmarks / test suites that witness for the quality of OSS" takes a low importance.

In Table 32 we summarize the data collected for the existence of benchmarks. Unexpectedly, most of the project do not show if they use any test framework and test suites. 18% of projects try to encourage the community to contribute to their quality efforts and 41% shows links to articles on the results of benchmarks.

Existence of test suites	Values	Percentage
YES	2	6%
NO	30	94%
Existence of benchmarks		Percentage
YES	6	19%
NO	26	81%
Usage of a test framework		Percentage
YES	1	3%
NO	31	97%
Results of test suite runs published		Percentage
YES	0	0%
NO	32	100%
Activity to encourage the community to contribute to quality efforts		Percentage
YES	6	19%
NO	26	81%
(Links to) articles on the results of benchmarks		Percentage
YES	13	41%
NO	19	59%
Explicitly named individuals or sub-communities which focus on these topics		Percentage
YES	0	0%
NO	32	100%
Which kind of tests are available		Percentage
performance test	9	28%
function test	5	16%
unknown	18	56%

Table 32 The existence of benchmarks / test suites that witness for the quality of OSS

Quality

This information is collected in order to check the availability of the quality related factors that OSS users take into account when selecting OSS products.

The degree to which a OSS product satisfies / covers functional requirements

As emerged in our questionnaire, the factor "The degree to which a OSS product satisfies / covers functional requirements" is fundamental.

In Table 33 we summarize the data collected for this factor.

Unexpectedly, the situation is negative: less than half of the projects do not provide a comprehensive list of supported functionalities and product samples (such as screenshots, static or dynamic demos or excerpts of code). The majority of the projects (19 out of 32) do not discuss functional requirements (or often the provided information is incomplete). Only releases notes are widely provided (59% of projects).

We suggest developers to focus their attention to this fundamental factor, discussing and reporting how their products satisfy/cover functional requirements.

Features list	Values	Percentage
poor free text description	13	41%
incomplete feature list available	6	19%
comprehensive feature list available	13	41%
Release notes	Values	Percentage
contains features information	19	59%
No feature information	13	41%
Products examples	Values	Percentage
live	5	16%
demo	15	47%
screenshot	10	31%
none	19	59%

Table 33: Functional Requirements Analysis

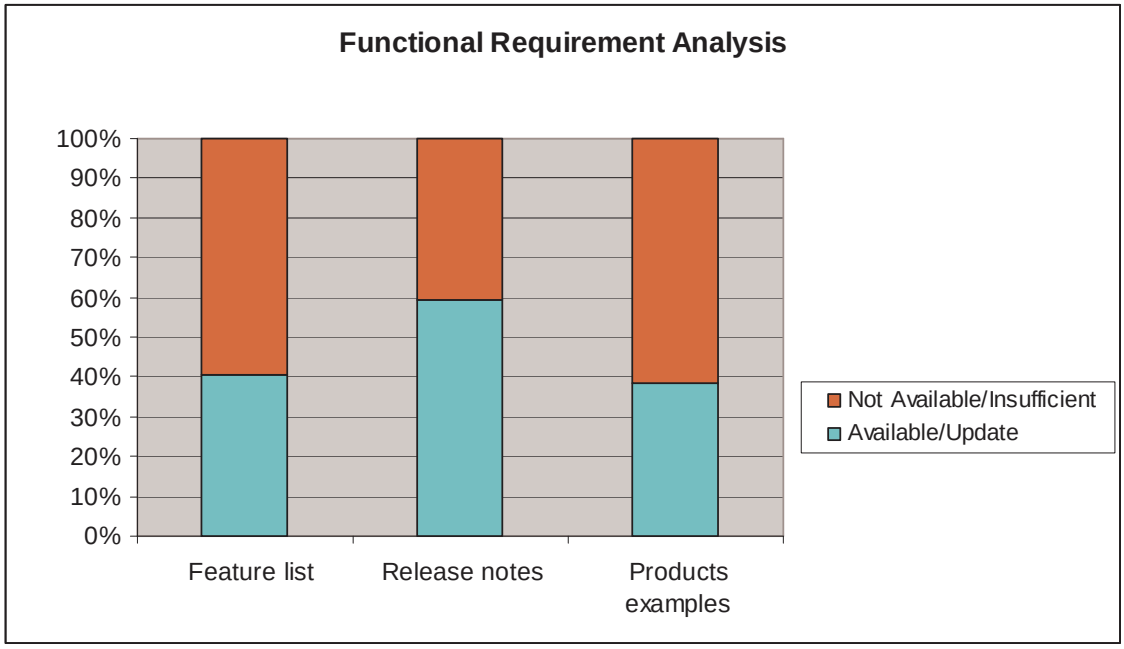

Figure 40: Functional Requirements Analysis

External quality – Performances

As emerged in our questionnaire, the factor "External quality - performances" takes a fairly high importance.

In Table 34 we summarize the data collected for the language uniformity.

Unexpectedly, this factor is omitted by most of developers: the majority of the projects do not provide any description about quality performances (for example, by means of specific documentations, reports of performance tests, benchmarks).

We suggest developers to pay attention to this important factor, discussing and reporting how their products satisfy non-functional requirements.

Quality performances description	Values	Percentage
yes	9	28%
no	23	72%
Is performance one of the goal of the project	Values	Percentage
yes	12	37%
no	20	63%
Performance tests	Values	Percentage
yes	8	25%
no	24	75%
Links to articles	Values	Percentage
yes	14	44%
no	18	56%
Performance-related documentation	Values	Percentage
yes	11	34%

no	21	66%
Performance-Oriented Implementation	Values	Percentage
yes	7	22%
no	0	0%
unknown	25	78%

Table 34: External quality – performance

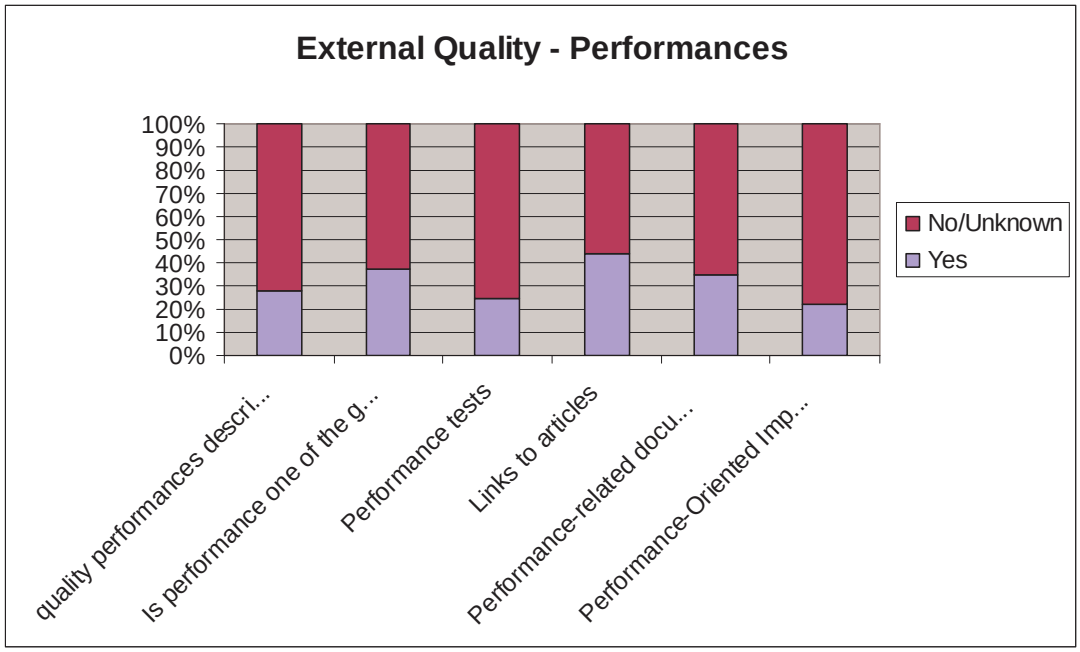

Figure 41: External quality - Performances

External quality - Maintainability

As emerged in our questionnaire, the factor "External quality - maintainability" takes a fairly high importance.

In Table 35 we summarize the data collected for the maintainability.

Unexpectedly, the only measure that is easily retrievable from the analyzed web portals is the existence of maintenance releases. The other measures are almost never retrievable, while half of the projects show the usage of some coding standards.

We suggest developers to better highlight this important factor into their web portals, discussing and reporting how their products are maintainable.

Adaptation description	Values	Percentage
yes	6	19%
no	26	81%
Architecture description	Values	Percentage
yes	9	28%

no	16	50%
Plugin interface	Values	Percentage
yes	12	38%
no	20	63%
Maintenance releases	Values	Percentage
yes	29	91%
unknow	3	9%
Existence Coding standards	Values	Percentage
yes	16	50%
no	16	50%
Coding standard check	Values	Percentage
yes	5	16%
no	27	84%

Table 35: External Quality - Maintainability

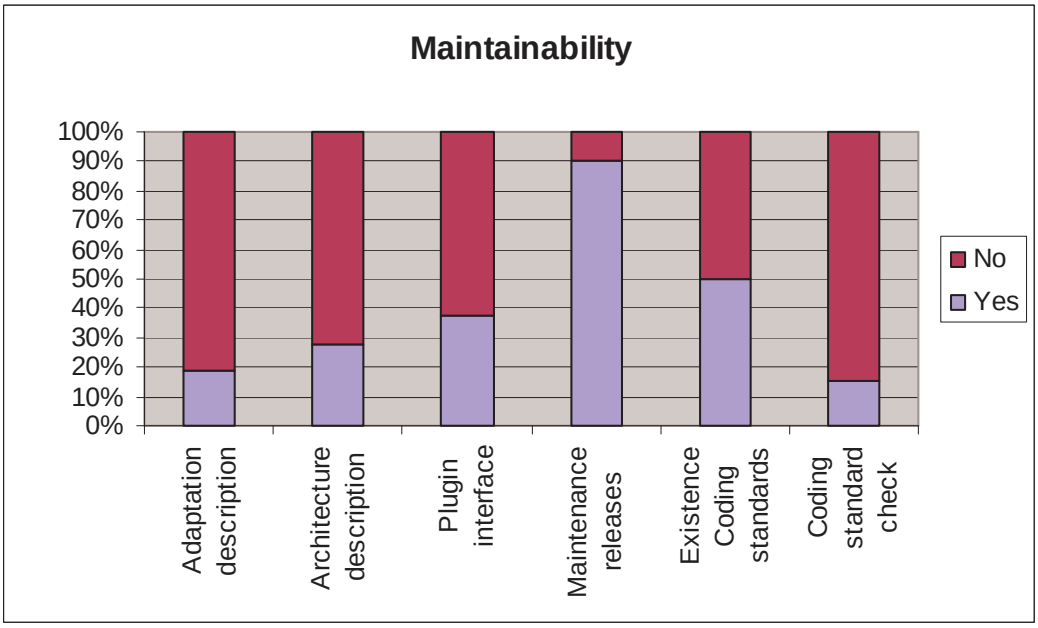

Figure 42: External Quality - Maintainability

External quality - Portability

As emerged in our questionnaire, the factor “External quality - portability” takes a fairly high importance.

In Table 36 we summarize the data collected for the portability issue.

As expected, the analysis shows that more than 70% of the projects use a portable language (e.g. Java), but only 38% of the projects show their supported environments.

Supported environments list	Values	Percentage
yes	12	38%
no	20	63%

Usage of an easy portable language (e.g. Java)	Values	Percentage
yes	23	72%
no	9	28%

Table 36: External Quality - Portability

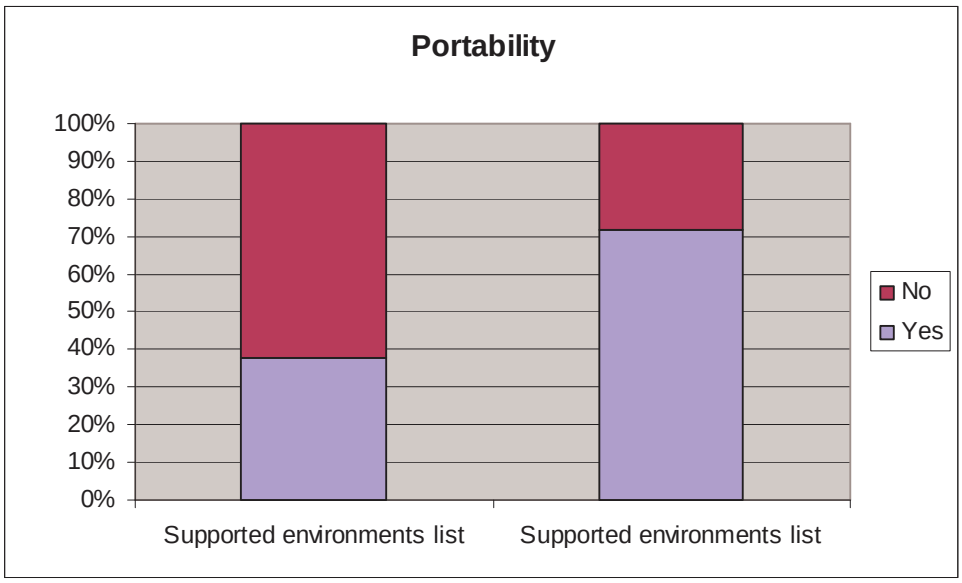

Figure 43: External Quality – Portability

External quality – Reliability

As emerged in our questionnaire, the factor "External quality - reliability" takes a very high importance.

In Table 37 we summarize the data collected for the reliability characteristic.

Unexpectedly, almost no project report its development status and only half of the projects use some coding standards and check it regularly.

We suggest developers to better highlight this important factor into their web portals, discussing and reporting data that demonstrate the degree of reliability of their products.

Development Status	Values	Percentage
Available	4	13%
Not Available	28	88%
Intentions - not objective description	Values	Percentage
Yes	10	31%
No	22	69%
Reliability focused Implementation	Values	Percentage
Yes	3	9%
No	5	16%
Unknown	24	75%
Minor/patch releases	Values	Percentage
1-4	10	31%

5-15	5	16%
>15	6	19%
Unknown	10	31%
Existence of coding standards	Values	Percentage
Yes	16	50%
No	16	50%
Coding standard check	Values	Percentage
Yes	5	16%
No	27	84%

Table 37: External Quality Reliability

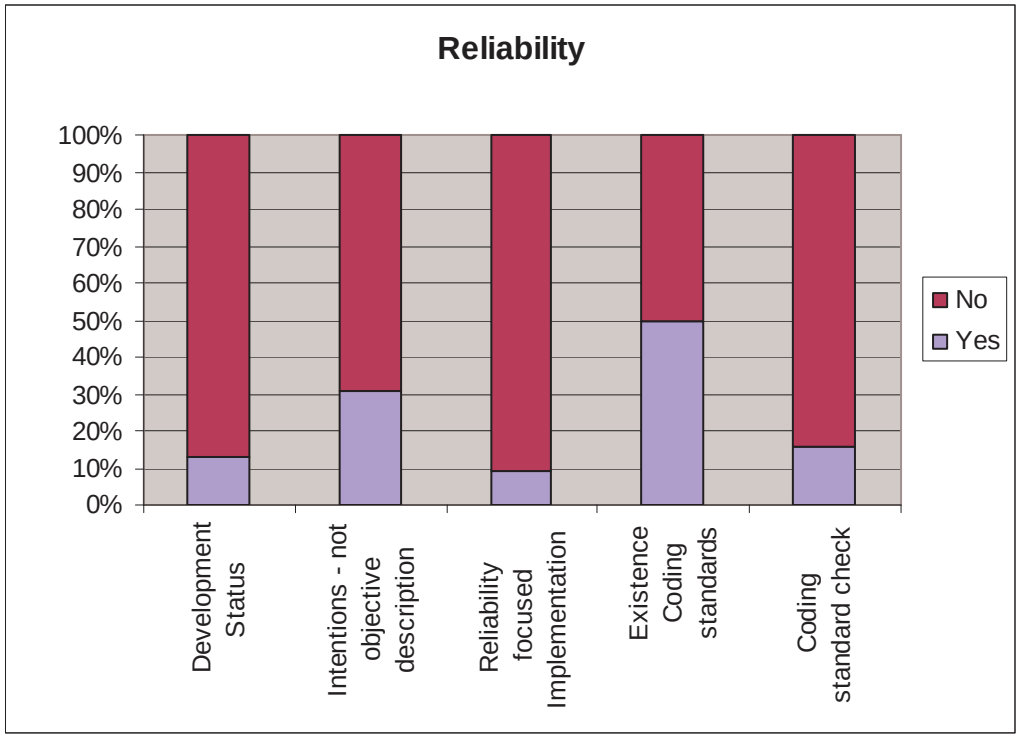

Figure 44: External Quality – Reliability

Internal Quality - Complexity

As emerged in our questionnaire, the factor "Internal quality - complexity" takes a low importance for users and developers.

This is reflected by the incompleteness of statistical data about the code complexity.

Internal Quality - Modularity

As emerged in our questionnaire, the factor "Internal quality - modularity" takes a high importance.

In Table 38 we summarize the data collected for the modularity.

Inline with the desire of users and providers, more than 60% of the analyzed projects provide plug-ins or interfaces that increase the modularity of the project.

Plug-ins / Interfaces provided	Values	Percentage
yes	20	63%
no	12	38%

Table 38: Internal Quality - Modularity

Internal Quality - usage of Standard Architecture

As emerged in our questionnaire, the factor "Internal quality – Standard Architecture" takes a high importance both for developers and users.

In Table 39 we summarize the data collected for the usage of standard architectures and design patterns.

Unexpectedly, only half of the projects provide some architectural documentation and a description of the adopted standards. Moreover, only 14 out of 32 web portals describe the design patterns applied into the project.

We suggest developers to simplify the retrieval of this important factor into their web portals, increasing the information and the documentation related to architectural choices they used.

Availability of architectural documentation	Values	Percentage
Yes	16	50%
No	16	50%
Any architectural standard/pattern used into the description/manual	Values	Percentage
Yes	14	44%
No	18	56%

Table 39: Usage of standard Architecture

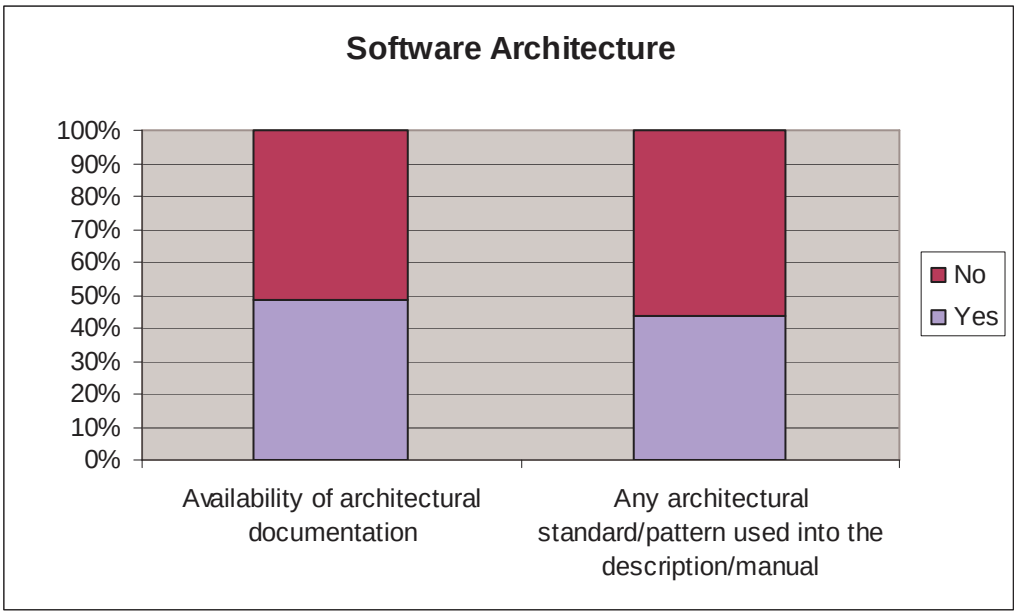

Figure 45: Usage of standard architecture

Standard compliance

As emerged in our questionnaire, the factor "Standard Compliance" takes a high importance.

In Table 40 we summarize the data collected about the compliance of the projects with available standards.

Unexpectedly, the possibility to assess the standard compliance reflects the already discussed quality factors. Two measures are in common with the factor "Internal Quality – Reliability": the use of standards during the coding phase and the check of the achieved standardization. Another measure we considered for this factor is the availability of information about the implemented standards (e.g., HTTP 1.0, SQL 97…). Unfortunately, only half of the projects report data about the compliance of the project with available standards.

We suggest developers to point out this important factor into their web portals, discussing and reporting data about the used standards in order to improve the global comprehension of the project.

Information about standards implemented	Values	Percentage
Yes	17	53%
No	15	47%
Coding standards	Values	Percentage
Yes	16	50%
No	16	50%

Coding standards checks	Values	Percentage
Yes	5	16%
No	27	84%

Table 40: Standard Compliance

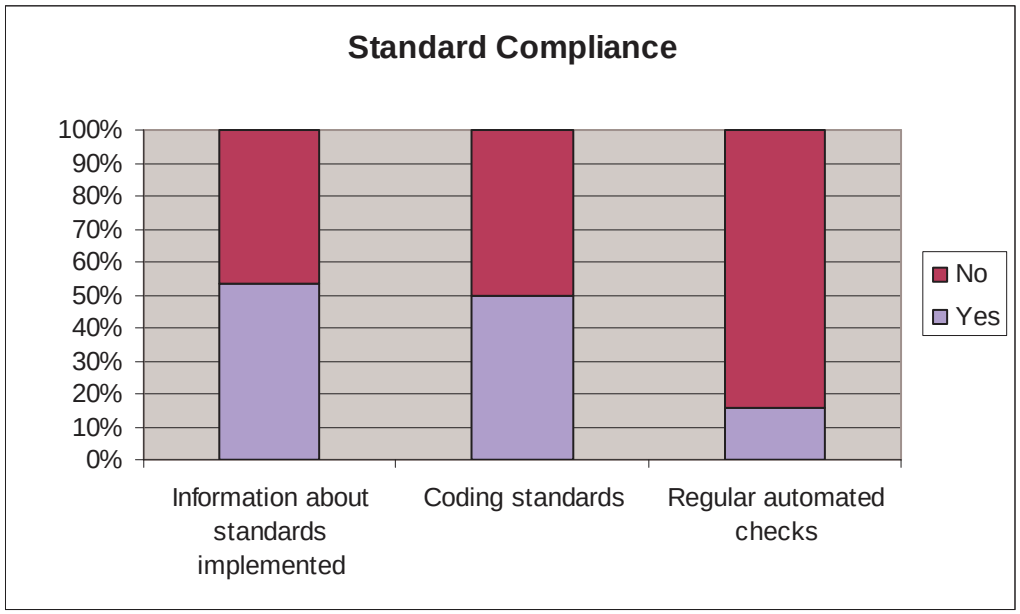

Figure 46: Standard Compliance

Self containedness

As emerged in our questionnaire, the factor "Self Containedness" takes a low importance.

In Table 41 we summarize the data collected for self containedness.

More than 70% of projects can run out of the box without any other tool or library. As expected, some projects use third parties products but only half of them describe integration issues in their documentation.

Can run "out of the box"?	Values	Percentage
Yes	23	72%
No	9	28%
no data	3	9%
Are third parties products used?	Values	Percentage
Yes	11	34%
No	15	47%
no data	3	9%
Documented which third parties products are used?	Values	Percentage
Yes	5	16%
No	26	81%
no data	1	3%

Table 41: Self Containedness

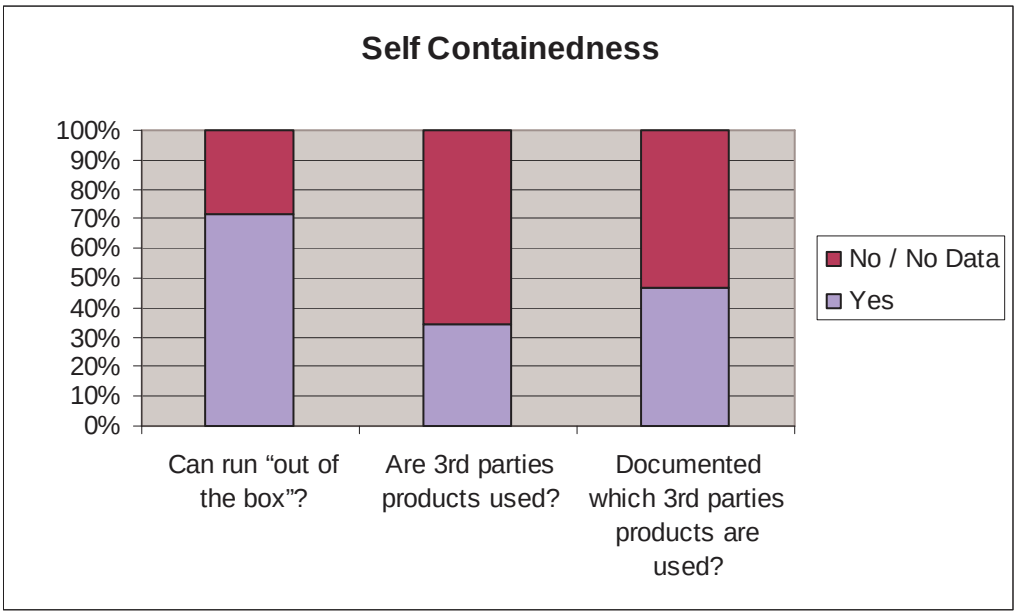

Figure 47: Self Containedness

Interoperability

As emerged in our questionnaire, the factor "Interoperability" takes a very high importance.

In Table 42 we summarize the data collected for the interoperability.

As expected, most of the projects are equipped with information about the interoperability issues (e.g., whether they communicate or not with other systems, and if they provide plug-ins or interfaces).

Communication with other systems	Values	Percentage
Yes	24	75%
No	8	25%
Plugin / Interfaces provided	Values	Percentage
Yes	20	63%
No	12	38%

Table 42: Interoperability

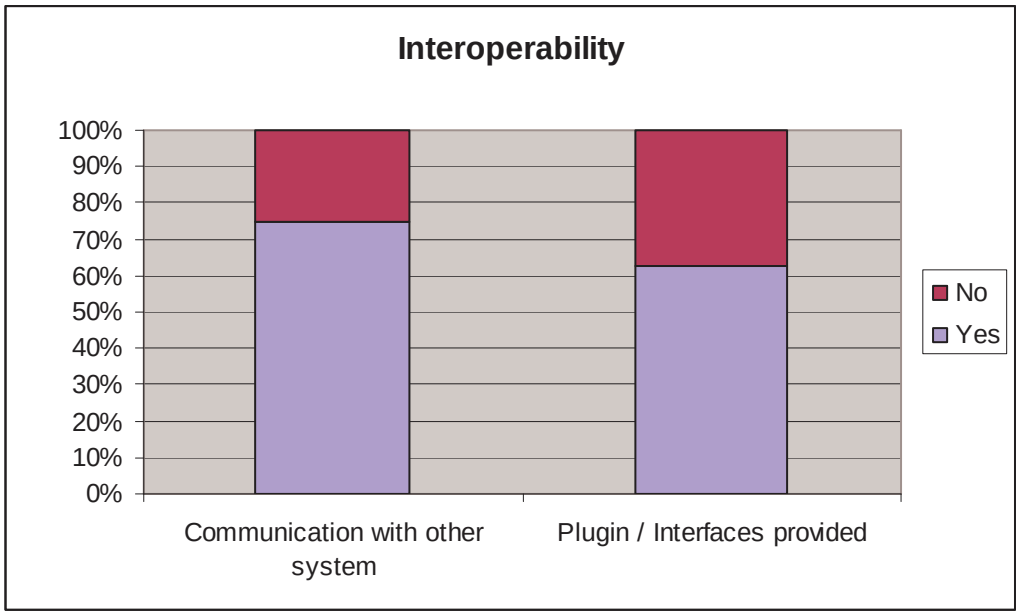

Figure 48: Interoperability

Human interface language / localization

As emerged in our questionnaire, the factor "Human interface language / localization" takes a low importance.

In Table 43 we summarize the data collected for the localizability aspect.

Only 11 projects out of 32 provide the localization support, and support more languages. This reflects the results of the questionnaire.

Localization support	Values	Percentage
Yes	11	34%
No	17	53%
Unknown	4	13%
Availability of different localizations	Values	Percentage
Yes	11	34%
No	17	53%
Unknown	4	13%
Possibility to add more languages	Values	Percentage
Yes	5	16%
No	25	78%
Unknown	2	6%

Table 43: Human interface language / localization

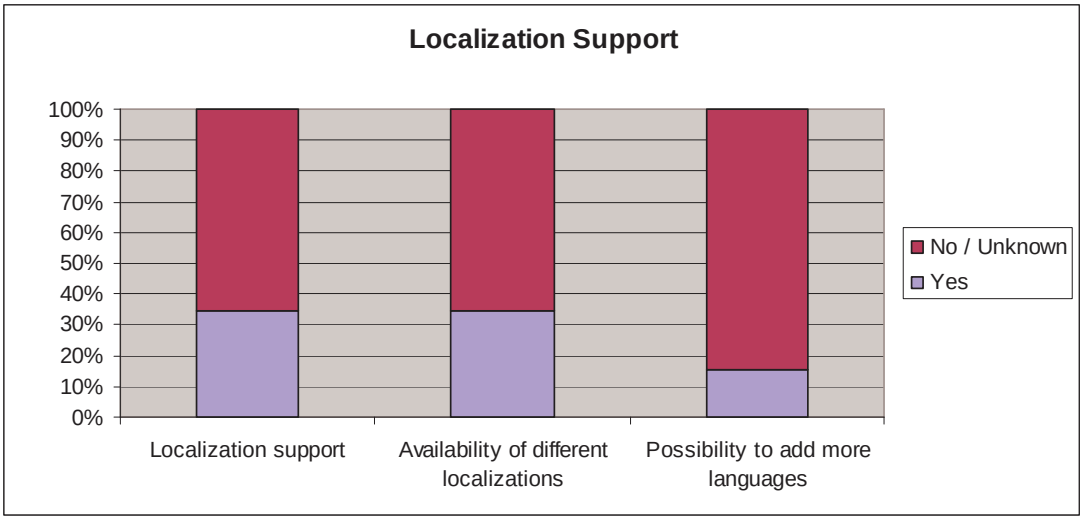

Figure 49: Human interface language / localization

Customer

By customer, we mean the person that has requested a service, a system, a library, a tool, etc. We intend to stress out a purchaser role.

An analysis of the customer-related factors is impossible due to the subjectivity of this category.

Appendix C: The questionnaire for assessing the perceived trustworthiness of OSS

Here follows the questionnaire for evaluating the users' perceived trustworthiness.

Why This Survey?

The purpose of this survey is to elicit information from the users and developers of Open Source Software (OSS) products about their perceptions on the trustworthiness of OSS products and the related factors.

Who Are We?

This survey has been developed in the framework of the QualiPSo (Quality Platform for Open Source Software) project, which is a European Union-funded Integrated Project which aims at making a major contribution to the state of the art and practice of Open Source Software. The QualiPSo project started in November 2006 and will last until October 2010. The project brings together 18 software companies, application solution developers, and research institutions. Its goal is to define and implement technologies, procedures, and policies to leverage the Open Source Software development current practices to sound, well-recognized, and established industrial operations.

What Will Happen to the Questionnaires?

All information provided by each individual or organization will be treated as confidential. As such, it will not be released in other form than aggregated statistical analyses that will make it impossible to identify the single respondents.

Some questions may not be applicable to you: just skip them. When you answer, *please always give <u>your</u> personal opinion*.

Your name (optional) _____
Your email address (optional) _____
Your role in your organization ☐ Upper Manager ☐ Project manager ☐ Developer ☐ Other
Type of organization ☐ private ☐ no profit ☐ Public Administration
Number of employees of your organization: _____
Organization's domain(s) (Public Administration, banking/finance, …)
_____
Number of employees of your specific unit in your organization: _____
Unit's domain(s) (Public Administration, avionics, banking/finance, …, same as organization's):
_____

Your use of the OSS product **Java Projects**	Checkstyle	Eclipse	Findbugs	Hibernate	HttpUnit	Jack.Commons IO	Jasper Reports	JBoss	JFreeChart	JMeter	Log4J	PMD	Saxon	Spring Framework	Struts	Tapestry	TPTP	Velocity	Weka	Xalan	Xerces	Servicemix
Do you use the product?																						
Yes																						
Maybe in the future																						
No																						
What version of the product are you using																						
The last one																						
A recent one																						
What is your relationship with the OSS product																						
User of the product 'as is'																						
Integrator/customizer																						
Producer																						
Other																						

Please give us **your opinion** for the projects you are familiar with, by ranking the factors below on a 1 to 6 scale, where 1= absolutely not; 2=little; 3=just enough; 4=more than enough; 5= very/a lot; 6= completely
Just skip the projects you are not familiar with

Quality of the OSS product **Java Projects**	Checkstyle	Eclipse	Findbugs	Hibernate	HttpUnit	Jack.Commons IO	Jasper Reports	JBoss	JFreeChart	JMeter	Log4J	PMD	Saxon	Spring Framework	Struts	Tapestry	TPTP	Velocity	Weka	Xalan	Xerces	Servicemix
How **familiar** are you with the product?																						
How **usable** is the product?																						
How **portable** is the product?																						
How much does/did the product **satisfy** your **functional requirements** when you use/used it?																						
How **interoperable** is the product?																						
How **reliable** is the product?																						
How **secure** is the product?																						
How useful is the **product developer community** to you?																						
How **fast** is the product?																						
How **well documented** is the product?																						

Based on your answers to the questions above:

How much do you **trust** the product, overall?																						
How much do you **trust** the product, **compared to its Open Source competitors**?																						
How much do you **trust** the product, **compared to its non Open Source competitors**?																						

Please tick the correct answer.

Your use of the OSS product **C/C++ Projects**	Ant	Axis	BusyBox	CVS	CygWin	DDD	GDB	Gnu C Library	Gnu GCC	Lib XML	Linux Kernel	Mono	MySQL	OpeLDAP	Open Pegasus	Open SSL	Perl	PosgreSQL	SpiderMonkey	SQLite	Subversion	TCL/Tk
Do you use the product?																						
Yes																						
Maybe in the future																						
No																						
What version of the product are you using																						
The last one																						
A recent one																						
What is your relationship with the OSS product																						
User of the product ‘as is’																						
Integrator/customizer																						
Producer																						
Other																						

Please give us **your opinion** for the projects you are familiar with, by ranking the factors below on a 1 to 6 scale, where 1= absolutely not; 2=little; 3=just enough; 4=more than enough; 5= very/a lot; 6= completely

Just skip the projects you are not familiar with

Quality of the OSS product **C/C++ Projects**	Ant	Axis	BusyBox	CVS	CygWin	DDD	GDB	Gnu C Library	Gnu GCC	Lib XML	Linux Kernel	Mono	MySQL	OpeLDAP	Open Pegasus	Open SSL	Perl	PosgreSQL	SpiderMonkey	SQLite	Subversion	TCL/Tk
How **familiar** are you with the product?																						
How **usable** is the product?																						
How **portable** is the product?																						
How much does/did the product **satisfy** your **functional requirements** when you use/used it?																						
How **interoperable** is the product?																						
How **reliable** is the product?																						
How **secure** is the product?																						
How useful is the **product developer community** to you?																						
How **fast** is the product?																						
How **well documented** is the product?																						

Based on your answers to the questions above:

How much do you **trust** the product, overall?																						
How much do you **trust** the product, **compared to its Open Source competitors**?																						
How much do you **trust** the product, **compared to its non Open Source competitors**?																						

Appendix D: The modified elements of the GQM plan

Q_Actual_Reliability

How much is the OSS product reliable

This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the ability of the software not to fail, i.e., to perform its function satisfactorily.

- Correctness: Correctness: is the OSS product correct?
  - Correctness_wrt_tests (Absolute)
    What is the % of functional tests passed
    Origin: Junit, awtaf
  - Test_instruction_coverage (Nominal)
    Origin: coverage tools (Jabuti; cobertura; etc.)
  - Test_condition_coverage (Nominal)
    Origin: coverage tools (Jabuti; cobertura; etc.)
  - Test_path_coverage (Nominal)
    Origin: coverage tools (Jabuti; cobertura; etc.)

- Robustness
  - Robustness - same as correctness, but outside specs (Ordinal)
    Percentage of cases not conforming the specifications in which the SW behaves acceptably

- Dependability: Is the OSS product dependable?
  - DependabilityEvidence (Ordinal)
    Is the OSS product dependable
    Origin: manual

- ProductMaturity: How mature is the OSS product?
  This question is meant to distinguished mature products from those that have just been made available and have yet to reach stability, concerning both faultiness and functionality (i.e., matching user needs).
  - ProductMaturityLevel (Ordinal)
    How much is the OSS product mature
    Origin: StatCVS/SVN; bugzilla; bugtrackers; other projects (
  - Bug_trend (Nominal)
    Origin: StatCVS/SVN; bugzilla; bugtrackers
  - Releases_trend (Nominal)
    Origin: some indicator from projects that amalyse forges

- FailureFrequency: What is the frequency of failures of the OSS product
  - FailuresFrequency (Absolute)
    The number of failures for every hour of usage
    Origin: Jmeter; crash report repository?

- How probable are problems according to code construction
  - Unexpected situation handling index (Absolute)
    number of situations not handled

Q_Actual_Functionality

The degree to which a OSS product satisfies / covers functional requirements
This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates the degree to which the considered OSS product satisfies / covers functional requirements.

Q_Actual_Functionality_Suitability

- RequirementsSatisfaction: To what extent does the OSS product satisfy the requirements
  The user requirements vary from user to user. Therefore here we consider a set of user requirements which is "typical" for the class of applications to which the product belongs.
  - RequirementsSatisfactionDegree (Absolute)
    Percentage of requirements that are satisfied

- LicenseCharacteristics: What are the characteristics of the license under which the OSS product is released
  In case no standard license is used, it is necessary to characterize the specific license under which the considered product is released. Otherwise, all the interesfing features should be determined by the standard license itself.
  - DistributionAgreement (Ordinal)
    Does the license allow distribution with different licence agreements
  - FeeAllowed (Ordinal)
    Does the license allow redistribution with fee
  - SourceCodeAccessible (Ordinal)
    Does the licence allow access to source code
  - ModificationOfCodeAllowed (Ordinal)
    Modification of source code allowed
  - NumberOfLicences (Absolute)
    Origin: OSLC, FOSSology
  - copyrightedMaterial (Ordinal)

Q_Actual_Functionality_Accuracy

- FunctionalityQuality: How ell are the requirements satisfied by the OSS product
  - FunctionalityImplementationQuality (Ordinal)
    For each requirement how well it is implemented by the OSS product

- LawConformance: Does the OSS product conform to applicable laws
  - ApplicableLaw (Nominal)
    Identitity of applicable law
  - LawBreakEvidence (Ordinal)

Q_Actual_Interoperability

How well does the OSS product support interoperability with other software

This quality is desirable in general, i.e., both if the product is used as-is, or if it is changed. It indicates how well the OSS product operates in conjunction with (i.e., exchanging data or control information with) other software products.

- EaseOfDataExchange: How easy is it to import/export data to/from the OSS product
  - StandardDataFormatSupported (Ordinal)
    Does the product read/write data according to a sintactically well defined format
  - UserDefinedData (Ordinal)
    Can the user define the format of the data to be used (read and written)

  - EaseOfDataParsing/Unparsing (Ordinal)
    Is the code that parses/unparses the In/Out data easy to modify
  - AutomaticalMagaementOfOtherSoftwareData (Ordinal)
    Is the application able to automatically manage hidden (e.g., configuration) data from other software?
  - SemanticallyWellDefinedDataFormat (Nominal)
    Does the product read/write data according to a semantically well defined format

- EaseOfIntegration: How easy is it to integrate the OSS product with other software
  At the control level.
  - StandardApplicationInterface (Ordinal)
    Does the OSS product support standard interfacing mechanisms
  - StandardInterfaceConformityEvidence (Ordinal)

- LocationIndependence: Are the locations of applications that interoperate with the considered OSS product made not relevant?
  - LocationIndependenceSupport (Ordinal)
    To what extent is location independence supported

- ProtocolBasedDataExchange
  - StandardProtocolSupported (Ordinal)
    like data, but for protocols
  - StandardProtocolSupportEvidence (Ordinal)

Q_Actual_Exploit_in_dev_Maintainability

The ease of maintaining the OSS product

Maintainability: A quality of the software that relates to the effort needed to make specified modifications. According to the traditional classification of maintenance activities, the required changes can be aimed at removing defects, extend the product functionality, or adapt it to environmental changes.

- BugRemovalEfficiency: How fast are bugs removed from the OSS product
  An aspect of maintenance is bug removal. Fast bug removal is an indicator of good maintainability Here it is intended that bug removeal is performed by the developers/maintainers, not by the user.
  - BugRemovalRate (Absolute)
    Number of bugs removed per month
    Origin: tools from flossmole, etc.
  - BugClosedPercentageRate (Absolute)
    Percentage of bugs closed per month
  - BugClosureRatePerDeveloper (Nominal)

- NewReleaseRate: How frequently are new versions released
  - NewReleasesPerYear (Absolute)
    Number of releases per year that do not involve just bug corrections
  - ChangedLOCSperYear (Absolute)
    Number of LOCS added or modified per year
  - BugsReportingRateperKLOC (Absolute)
    Number of new bugs found per month per KLOC

- Analyzability_for_maintenance: How easy is it to analyze the code of the OSS product for the purpose of maintenance

  - CodingStandards (Absolute)
    Is a coding standard defined for the product
  - CodeDocumentation (Absolute)
    Lines of comments to total lines of code ratio
    Origin: Macxim
  - CodeModularity (Absolute)
    How much modular is the code
    Origin: Macxim
  - MaintainabilityOrientedArchitecture (Ordinal)
    Does the product feature an architecture easying maintenance
  - RunTimeModularity (Nominal)
    Origin: AOP by Siemens
  - CodingStandardEnforcement (Nominal)
    Is the usage of a coding standard verified

- Maintenance_Testability: Is support to testing available
  - TestCases (Absolute)
    Number of test cases available
  - TestResultsAvailability (Ordinal)
    Availability of the results of previous tests
  - McCabeCyclomaticNumber (Absolute)
    McCabe Cyclomatic complexity of the OSS product code

- Maintenance_Stability
  - Number_of_failures_due_to_maintenance (Absolute)
    How many failures are caused by defects introduced by change activities
    Origin: work by Zeller, Stroulia
  - DesignPatternUsage (Ordinal)
    Usage of design patterns
    Origin: tool, like PTIDEJ, FUJABA,SPQR, CodeCrawler.

- SupportingToolAvailability: Are tools available to support the adoption of the OSS product
  - ToolSupport (Ordinal)
    The level of support provided by tools

- StandardArchitecture: Does the OSS product feature a standard architecture
  - StandardArchitecture (Ordinal)
    Does the OSS product feature a standard architecture

- CodeSize
  - CodeSizeinLOC (Absolute)
    Code size measured in effective LOC
    Origin: Static cxode measurement
  - OOCodeSize (Ratio)
    What is the size in terms of object-oriented constructs
    Origin: code measurement

Appendix E: why do our logistic regressions look linear?

One could observe that the logistic regression lines reported in several of the above sections do not look like the typical regression line, which is illustrated in **Figure 50**.

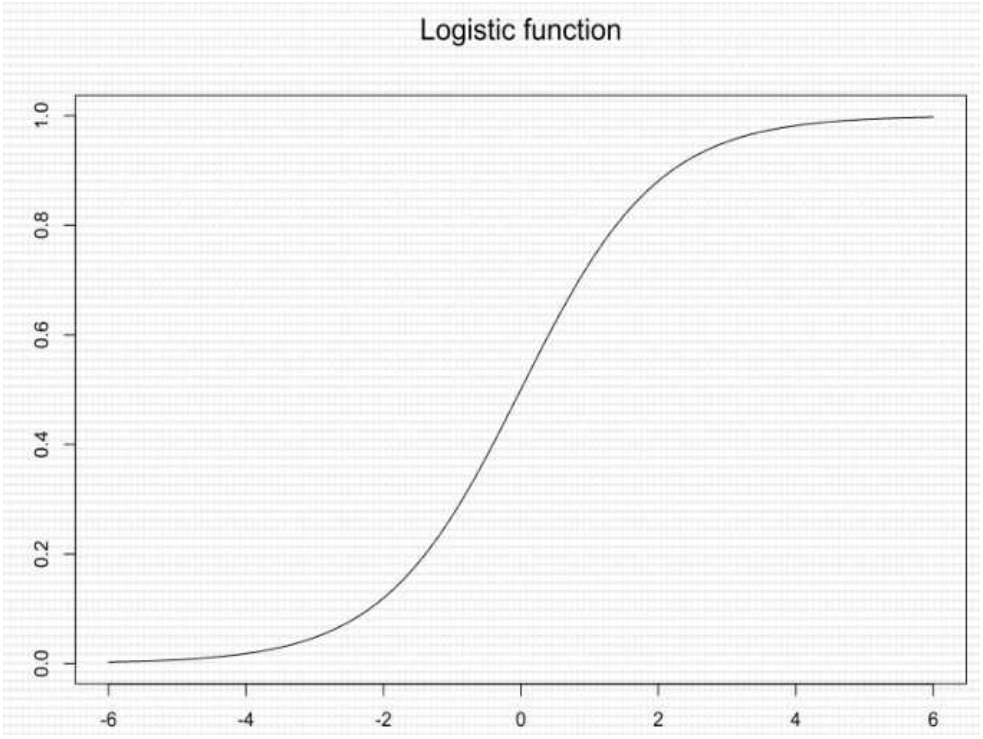

Figure 50. The logistic function $y=1/(1+e^{-(a+bx)})$.

The reason is that the range of variability of our independent variable is a relatively small interval.

Consider for instance the function $y = 1/ (1+e^{-(1.227363--0.006746\ x)})$, which is reported in **Errore. L'origine riferimento non è stata trovata.**, limitedly to the 25..250 range. The same function, plotted for whole x axis is reported in **Figure 51**. The highlighted region corresponds to the portion of the function illustrated in **Figure 51**. It is easy to see that in such region the function is approximately linear.

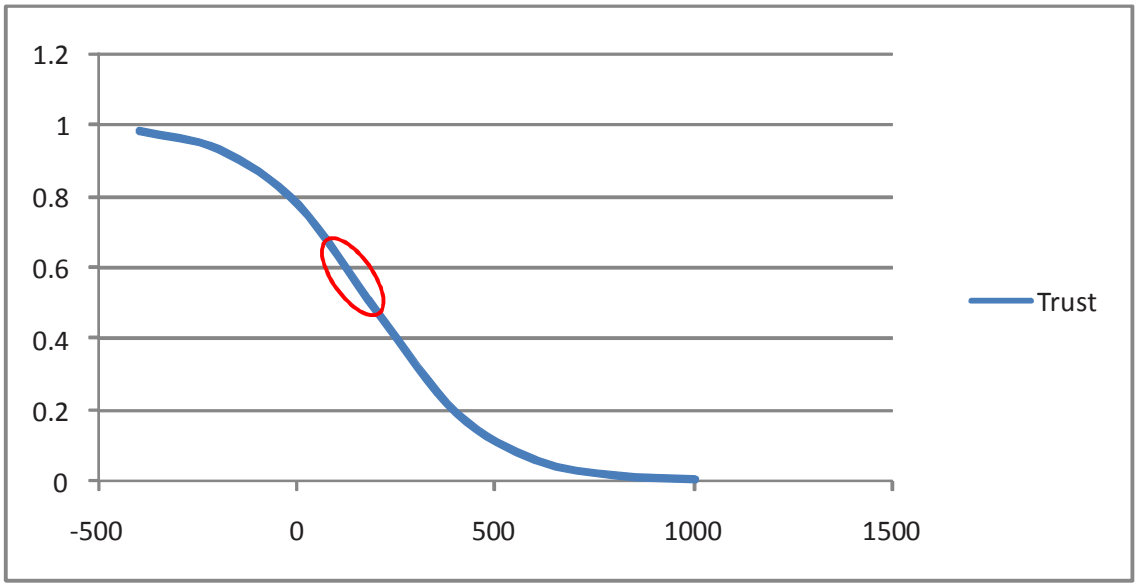

Figure 51. The logistic function y = 1/ (1+e-(1.48142-0.08547 x)).

Appendix F: Analysis of Java products

Here we report the detailed data on for java projects the correlations between subjective and objective data.

How to read the results

Every correlation found is illustrated by means of a set of results from the statistical analysis as illustrated in Figure 52.

```
Reliability  vs.  LCOM ,  Num. public methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  2.095484e+00 5.153696e-01  4.065982 4.783053e-05
x1          -1.371423e-03 3.936123e-04 -3.484196 4.936172e-04
x2          -3.414151e-05 1.468522e-05 -2.324889 2.007788e-02
R2log =  0.8483479
Excluded as outliers:  Eclipse Ant HttpUnit Log4J  ( 4 / 17 )
MMRE =  21.68755
Pred(25) =  82.35294
Error range = [ -19.12005 .. 98.45401 ]
```

Figure 52. Data about a correlation.

The first line indicates the correlation being reported: the corretalion reported in in Figure 52 concerns reliability vs. LCOM (the lack of cohesion between methods) and the number of public methods.

The following tables 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. public methods). Therefore, $z = 2.095484 - 0.001371423\, x_1 - 0.00003414151 x_2$ and $\text{Reliability}(z) = \frac{1}{1+e^{-z}}$.

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.

R2log is the value of R^2_{log}, a measure of goodness of fit defined in [88] 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, Ant, HttpUnit and Log4J) were excluded as outliers.

The last three lines give some indication on the precision of the fitting. MMRE Mean Magnitude 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).

Reliability

The significant models found for OSS products Reliability are reported below.

```
============================================================
Reliability  vs.  Num. interfaces per class
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.8540080  0.2158944  3.955674 7.631912e-05
x1          -0.8735424  0.3750497 -2.329138 1.985178e-02
R2log =  0.8465696
Excluded as outliers:  Eclipse  ( 1 / 18 )
MMRE =  16.97731
Pred(25) =  77.77778
Error range = [ -32.56562 .. 46.18897 ]
============================================================
Reliability  vs.  Num. packages
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  0.580388458 0.1478487288  3.925556 8.652965e-05
x1          -0.003433091 0.0008863225 -3.873410 1.073229e-04
R2log =  0.8822593
Excluded as outliers:  Hibernate  ( 1 / 18 )
MMRE =  21.56865
Pred(25) =  77.77778
Error range = [ -60.01954 .. 98.18703 ]
============================================================
Reliability  vs.  CBO ,  Num. abstract classes
                Estimate  Std. Error   z value    Pr(>|z|)
(Intercept)  1.78359998 0.620729144  2.873395 0.004060861
x1          -0.45494183 0.190859853 -2.383643 0.017142204
x2           0.00705526 0.003555181  1.984501 0.047199987
R2log =  0.8538135
Excluded as outliers:  Eclipse Xalan Saxon  ( 3 / 17 )
MMRE =  24.2585
Pred(25) =  58.82353
Error range = [ -100 .. 37.34153 ]
============================================================
Reliability  vs.  LCOM ,  McCabe
                 Estimate   Std. Error    z value    Pr(>|z|)
(Intercept) -0.485624340 0.7177832658 -0.6765612 0.498684375
x1          -0.001164551 0.0003753264 -3.1027695 0.001917189
x2           0.832067653 0.4064419644  2.0471992 0.040638533
```

```
R2log =  0.8462217
Excluded as outliers:  Eclipse Ant Xerces  ( 3 / 18 )
MMRE =  17.49821
Pred(25) =  77.77778
Error range = [ -24.06588 .. 71.11811 ]
============================================================
Reliability  vs.  LCOM ,  Num. public methods
                  Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  2.095484e+00 5.153696e-01  4.065982 4.783053e-05
x1          -1.371423e-03 3.936123e-04 -3.484196 4.936172e-04
x2          -3.414151e-05 1.468522e-05 -2.324889 2.007788e-02
R2log =  0.8483479
Excluded as outliers:  Eclipse Ant HttpUnit Log4J  ( 4 / 17 )
MMRE =  21.68755
Pred(25) =  82.35294
Error range = [ -19.12005 .. 98.45401 ]
============================================================
Reliability  vs.  avg_loc_changed_per_year
                  Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  6.070026e-01 1.867163e-01  3.250935 0.001150262
x1          -4.904745e-05 2.110291e-05 -2.324204 0.020114586
R2log =  0.8563513
Excluded as outliers:  Eclipse Xalan Hibernate  ( 3 / 15 )
MMRE =  27.95078
Pred(25) =  66.66667
Error range = [ -98.0998 .. 30.23309 ]
============================================================
```

Usability

```
============================================================
Usability  vs.  CBO
             Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  1.8900494  0.6666344  2.835212 0.004579535
x1          -0.4521127  0.1750141 -2.583293 0.009786204
R2log =  0.840735
Excluded as outliers:  Eclipse Saxon Struts Xalan Hibernate  ( 5
/ 18 )
MMRE =  37.77437
Pred(25) =  33.33333
Error range = [ -100 .. 101.6746 ]
============================================================
Usability  vs.  LCOM
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -0.2463181527 0.2029179427 -1.213881 0.224793325
x1           0.0009124617 0.0003209740  2.842790 0.004472058
R2log =  0.8568704
Excluded as outliers:  JFreeChart Eclipse Checkstyle JMeter  ( 4
/ 18 )
MMRE =  31.74570
Pred(25) =  55.55556
Error range = [ -50.70977 .. 154.6519 ]
============================================================
Usability  vs.  Comment lines
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  7.293996e-01 2.165911e-01  3.367634 0.0007581608
x1          -7.331410e-06 2.945903e-06 -2.488680 0.0128218180
R2log =  0.8349655
Excluded as outliers:  Eclipse Struts HttpUnit  ( 3 / 18 )
```

```
MMRE =  27.32437
Pred(25) =  61.11111
Error range = [ -47.10666 .. 84.03233 ]
================================================================
Usability  vs.  Comment lines per class
              Estimate  Std. Error    z value   Pr(>|z|)
(Intercept) -0.19540460 0.292126050 -0.6689051 0.50355602
x1           0.01534695 0.007654782  2.0048838 0.04497547
R2log =  0.8863142
Excluded as outliers:  JFreeChart Ant Weka Xerces  ( 4 / 18 )
MMRE =  39.06681
Pred(25) =  38.88889
Error range = [ -37.87529 .. 181.9824 ]
================================================================
Usability  vs.  eLOC per class
              Estimate  Std. Error   z value   Pr(>|z|)
(Intercept) -1.08623822 0.607241023 -1.788809 0.07364558
x1           0.02207583 0.009168053  2.407909 0.01604420
R2log =  0.8895461
Excluded as outliers:  JFreeChart Weka Xerces Ant  ( 4 / 18 )
MMRE =  38.34607
Pred(25) =  44.44444
Error range = [ -43.70425 .. 174.1539 ]
================================================================
Usability  vs.  McCabe
             Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -2.585271  0.9440127 -2.738598 0.006170175
x1           1.318073  0.4381872  3.008014 0.002629607
R2log =  0.7773205
Excluded as outliers:  Eclipse Xerces Ant Hibernate Log4J  ( 5 /
18 )
MMRE =  27.85867
Pred(25) =  55.55556
Error range = [ -35.14576 .. 108.9347 ]
================================================================
Usability  vs.  Num. abstract classes
              Estimate  Std. Error   z value   Pr(>|z|)
(Intercept)  0.70679556 0.258363763  2.735661 0.00622552
x1          -0.01090757 0.005052382 -2.158897 0.03085819
R2log =  0.8316732
Excluded as outliers:  Hibernate Eclipse Saxon  ( 3 / 17 )
MMRE =  27.87937
Pred(25) =  58.82353
Error range = [ -63.27699 .. 91.26135 ]
================================================================
Usability  vs.  Num. classes
               Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -0.3406731180 0.2819005583 -1.208487 0.22686005
x1           0.0004166072 0.0001997859  2.085268 0.03704497
R2log =  0.8801109
Excluded as outliers:  Hibernate Log4J Checkstyle  ( 3 / 18 )
MMRE =  30.68464
Pred(25) =  55.55556
Error range = [ -50.5957 .. 100.9276 ]
================================================================
Usability  vs.  Num. intefaces
              Estimate  Std. Error   z value   Pr(>|z|)
(Intercept) -0.238526704 0.216620191 -1.101129 0.27084066
x1           0.001866302 0.000803439  2.322892 0.02018498
R2log =  0.8819345
```

```
Excluded as outliers:  Log4J Hibernate Checkstyle  ( 3 / 18 )
MMRE =  25.31939
Pred(25) =  50
Error range = [ -50.16849 .. 71.46671 ]
================================================================
Usability  vs.  Num. methods per class
              Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -0.78918991 0.38396156 -2.055388 0.039841560
x1           0.07067751 0.02543282  2.778988 0.005452852
R2log =  0.855624
Excluded as outliers:  Eclipse JFreeChart Checkstyle JMeter  ( 4
/ 18 )
MMRE =  33.31810
Pred(25) =  55.55556
Error range = [ -53.38943 .. 150.4082 ]
================================================================
Usability  vs.  Num. methods per interface
             Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.4220748  0.2461684 -1.714578 0.08642265
x1           0.2956812  0.1174300  2.517936 0.01180447
R2log =  0.8359456
Excluded as outliers:  Eclipse Log4J  ( 2 / 16 )
MMRE =  19.11263
Pred(25) =  75
Error range = [ -51.04846 .. 48.20157 ]
================================================================
Usability  vs.  LCOM ,  Num. interfaces per class
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  0.4938351234 0.2198629305  2.246105 0.024697314
x1           0.0008107104 0.0003247587  2.496347 0.012547984
x2          -1.5492446451 0.4857427649 -3.189434 0.001425515
R2log =  0.8407305
Excluded as outliers:  Eclipse  ( 1 / 18 )
MMRE =  16.73436
Pred(25) =  72.22222
Error range = [ -42.89694 .. 44.26262 ]
================================================================
Usability  vs.  Comment lines per class ,  Num. abstract classes
                Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  1.306207571 0.343302303  3.804832 0.0001419005
x1          -0.013041367 0.004869261 -2.678305 0.0073995818
x2          -0.005707365 0.002264978 -2.519832 0.0117410849
R2log =  0.8924783
Excluded as outliers:  HttpUnit  ( 1 / 17 )
MMRE =  21.92274
Pred(25) =  64.70588
Error range = [ -49.71311 .. 96.47 ]
================================================================
Usability  vs.  Comment lines per class ,  Num. interfaces per
class
              Estimate  Std. Error   z value    Pr(>|z|)
(Intercept)  0.69131658 0.219740236  3.146063 0.001654845
x1           0.01582193 0.007266965  2.177240 0.029462634
x2          -2.11435537 0.647118645 -3.267338 0.001085640
R2log =  0.8865136
Excluded as outliers:  Ant  ( 1 / 18 )
MMRE =  20.18166
Pred(25) =  77.77778
Error range = [ -40.73157 .. 59.05492 ]
================================================================
```

```
Usability  vs.  eLOC per class ,  Num. interfaces per class
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.59063979 0.83879334 -1.896343 0.0579147181
x1           0.05654653 0.01614373  3.502692 0.0004605812
x2          -4.16542796 1.32155395 -3.151917 0.0016220253
R2log =  0.9093642
Excluded as outliers:  Ant JFreeChart Xerces Struts Weka  ( 5 /
18 )
MMRE =  30.05399
Pred(25) =  66.66667
Error range = [ -69.21335 .. 119.1383 ]
================================================================
Usability  vs.  McCabe ,  Num. abstract classes
               Estimate  Std. Error   z value    Pr(>|z|)
(Intercept) -2.69824151 1.117092945 -2.415414 0.015717350
x1           1.72745291 0.556610584  3.103521 0.001912323
x2          -0.01814791 0.006294499 -2.883138 0.003937343
R2log =  0.8347016
Excluded as outliers:  Eclipse Hibernate Xerces Jasper Reports
( 4 / 17 )
MMRE =  31.15396
Pred(25) =  52.94118
Error range = [ -95.33598 .. 95.93583 ]
================================================================
Usability  vs.  McCabe ,  Num. classes
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.1706580369 1.3638094822 -3.058094 0.002227495
x1           2.3871912375 0.7296164279  3.271844 0.001068485
x2          -0.0005741655 0.0002860659 -2.007109 0.044738046
R2log =  0.8161358
Excluded as outliers:  Eclipse Hibernate Xerces Ant PMD  ( 5 /
18 )
MMRE =  35.56762
Pred(25) =  50
Error range = [ -92.81493 .. 91.73914 ]
================================================================
Usability  vs.  McCabe ,  Num. interfaces per class
             Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -1.149280  0.7553382 -1.521543 0.128123600
x1           1.099552  0.3568063  3.081649 0.002058572
x2          -1.645729  0.7942253 -2.072118 0.038254422
R2log =  0.8427765
Excluded as outliers:  Eclipse Ant JFreeChart Xerces  ( 4 / 18 )
MMRE =  22.68806
Pred(25) =  61.11111
Error range = [ -51.17379 .. 53.35328 ]
================================================================
Usability  vs.  McCabe ,  Num. methods
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.724667e+00 1.066611e+00 -2.554509 0.010633759
x1           1.672664e+00 5.453123e-01  3.067350 0.002159655
x2          -5.673673e-05 2.159146e-05 -2.627739 0.008595446
R2log =  0.838209
Excluded as outliers:  Eclipse Hibernate Xerces Ant Saxon  ( 5 /
18 )
MMRE =  29.2469
Pred(25) =  55.55556
Error range = [ -86.96785 .. 62.25866 ]
================================================================
Usability  vs.  McCabe ,  Num. public methods
```

```
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.347257e+00 1.101606e+00 -2.130760 0.033108923
x1           1.567818e+00 5.513584e-01  2.843555 0.004461337
x2          -7.893846e-05 2.644862e-05 -2.984597 0.002839525
R2log =  0.8454107
Excluded as outliers:  Eclipse Hibernate Xerces Ant Saxon  ( 5 /
17 )
MMRE =  27.89225
Pred(25) =  58.82353
Error range = [ -89.9932 .. 60.93035 ]
================================================================
Usability  vs.  McCabe ,  RFC
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.752224e+00 1.069091e+00 -2.574358 0.010042643
x1           1.687290e+00 5.452864e-01  3.094320 0.001972647
x2          -3.104551e-05 1.147287e-05 -2.705993 0.006810042
R2log =  0.8398207
Excluded as outliers:  Eclipse Hibernate Xerces Ant Saxon  ( 5 /
18 )
MMRE =  28.41228
Pred(25) =  61.11111
Error range = [ -87.3 .. 62.42777 ]
================================================================
Usability  vs.  NOC ,  Num. interfaces per class
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.5427735  0.4689350  1.157460 0.24708452
x1           0.8628447  0.4058809  2.125857 0.03351519
x2          -1.9245142  0.7835257 -2.456223 0.01404058
R2log =  0.803971
Excluded as outliers:  JBoss Ant JFreeChart Eclipse Hibernate  (
5 / 18 )
MMRE =  23.85434
Pred(25) =  66.66667
Error range = [ -69.32601 .. 67.94469 ]
================================================================
Usability  vs.  Num. abstract classes ,  Num. methods per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  3.01555755 0.777681713  3.877624 0.0001054814
x1          -0.01614820 0.006249208 -2.584040 0.0097650424
x2          -0.11122450 0.035442046 -3.138208 0.0016998432
R2log =  0.8580966
Excluded as outliers:  Hibernate Saxon Eclipse Ant HttpUnit  ( 5
/ 17 )
MMRE =  26.34150
Pred(25) =  70.58824
Error range = [ -69.3333 .. 117.7895 ]
================================================================
Usability  vs.  Num. interfaces per class ,  Num. methods per
class
               Estimate Std. Error    z value    Pr(>|z|)
(Intercept) -0.19565028 0.40222876 -0.4864154 0.626672640
x1          -1.70975697 0.83624398 -2.0445672 0.040897550
x2           0.07797737 0.02705926  2.8817255 0.003955041
R2log =  0.8552978
Excluded as outliers:  Eclipse JFreeChart Checkstyle  ( 3 / 18 )
MMRE =  19.26414
Pred(25) =  72.22222
Error range = [ -42.50574 .. 44.08862 ]
================================================================
Usability  vs.  LCOM ,  eLOC ,  McCabe
```

```
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -3.315320e+00 1.089383e+00 -3.043301 0.0023399848
x1          -1.007403e-03 4.155108e-04 -2.424494 0.0153297546
x2          -9.283670e-06 3.768757e-06 -2.463324 0.0137655369
x3           2.264528e+00 5.847839e-01  3.872418 0.0001077609
R2log =  0.836629
Excluded as outliers:  Eclipse Hibernate Xerces Ant Struts  ( 5
/ 18 )
MMRE =  28.57176
Pred(25) =  50
Error range = [ -72.82899 .. 66.58874 ]
================================================================
Usability  vs.  LCOM ,  McCabe ,  NOC
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -4.567965357 1.1256249681 -4.058159 4.946104e-05
x1          -0.001288068 0.0004399631 -2.927674 3.415084e-03
x2           2.408657412 0.5689756525  4.233322 2.302640e-05
x3           1.276032873 0.4570472283  2.791906 5.239858e-03
R2log =  0.8703066
Excluded as outliers:  Ant Xerces Eclipse Weka Xalan  ( 5 / 18 )
MMRE =  32.07286
Pred(25) =  66.66667
Error range = [ -25.67032 .. 177.2891 ]
================================================================
Usability  vs.  LCOM ,  McCabe ,  Num. abstract classes
              Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -2.58753423 0.9913854929 -2.610018 0.0090537392
x1          -0.00094592 0.0004016964 -2.354813 0.0185320094
x2           1.93654227 0.5206681085  3.719341 0.0001997432
x3          -0.01179430 0.0058714161 -2.008765 0.0445620353
R2log =  0.8294042
Excluded as outliers:  Hibernate Eclipse Xerces Ant  ( 4 / 17 )
MMRE =  29.37844
Pred(25) =  58.82353
Error range = [ -82.48251 .. 93.57965 ]
================================================================
Usability  vs.  Comment lines ,  eLOC per class ,  Num.
parameters per method
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.569105e+00 1.746996e+00 -2.615407 0.008912115
x1          -8.696668e-06 4.081206e-06 -2.130906 0.033096866
x2           2.543978e-02 1.060271e-02  2.399367 0.016423460
x3           3.990756e+00 1.626565e+00  2.453486 0.014147895
R2log =  0.8289981
Excluded as outliers:  Eclipse JFreeChart Xerces Spring
Framework Hibernate  ( 5 /18 )
MMRE =  37.65118
Pred(25) =  50
Error range = [ -57.23838 .. 210.4919 ]
================================================================
Usability  vs.  Comment lines per class ,  McCabe ,  Num.
interfaces per class
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.18055552 0.74714504 -1.580089 0.1140865308
x1           0.02911286 0.01186272  2.454146 0.0141219568
x2           0.96756192 0.37792357  2.560205 0.0104610349
x3          -3.18227887 0.90979475 -3.497799 0.0004691151
R2log =  0.9036471
Excluded as outliers:  Ant Xerces Xalan Hibernate  ( 4 / 18 )
MMRE =  24.63878
```

```
Pred(25) =  72.22222
Error range = [ -29.13776 .. 88.03527 ]
================================================================
Usability  vs.  Comment lines per class ,  NOC ,  Num. methods
per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -1.59123947 0.796091911 -1.998814 0.0456285081
x1          -0.02710334 0.008099853 -3.346152 0.0008194146
x2           1.80096820 0.607362473  2.965228 0.0030245869
x3           0.10212771 0.036325585  2.811454 0.0049318145
R2log =  0.8938025
Excluded as outliers:  Hibernate JMeter Struts JBoss  ( 4 / 18 )
MMRE =  28.68584
Pred(25) =  55.55556
Error range = [ -62.68749 .. 104.6778 ]
================================================================
Usability  vs.  Comment lines per class ,  Num. classes ,  Num.
parameters per method
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -2.792284069 1.0853653696 -2.572667 0.0100918148
x1          -0.022731099 0.0072836336 -3.120846 0.0018033226
x2           0.001146167 0.0003180059  3.604231 0.0003130786
x3           3.085578531 1.1647931329  2.649036 0.0080721817
R2log =  0.9034009
Excluded as outliers: Hibernate Findbugs Log4J Checkstyle (4/18)
MMRE =  28.68404
Pred(25) =  55.55556
Error range = [ -58.83847 .. 94.19984 ]
================================================================
Usability  vs.  eLOC per class ,  eLOC ,  Num. parameters per
method
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -5.008987e+00 1.793211e+00 -2.793307 0.005217219
x1           2.846816e-02 1.121502e-02  2.538396 0.011136179
x2          -9.724296e-06 4.271110e-06 -2.276761 0.022800530
x3           4.332511e+00 1.641753e+00  2.638955 0.008316204
R2log =  0.8311528
Excluded as outliers:  JFreeChart Eclipse Xerces Hibernate
Spring Framework  ( 5 / 18 )
MMRE =  39.81476
Pred(25) =  38.88889
Error range = [ -62.56738 .. 217.0889 ]
================================================================
Usability  vs.  McCabe ,  Num. interfaces per class ,  Num.
parameters per method
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  4.411831  1.8951227  2.327992 0.0199125138
x1           2.262855  0.6754384  3.350202 0.0008075256
x2          -4.280543  1.2465845 -3.433817 0.0005951464
x3          -7.684348  2.7963556 -2.747987 0.0059962453
R2log =  0.8421222
Excluded as outliers:  Eclipse Ant JFreeChart Spring Framework
Hibernate  ( 5 / 18 )
MMRE =  26.66088
Pred(25) =  61.11111
Error range = [ -99.94252 .. 81.56915 ]
================================================================
Usability  vs.  NOC ,  Num. classes ,  Num. interfaces per class
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  0.659571228 0.4935547265  1.336369 0.181428699
```

```
x1          -0.871233340 0.4077327364 -2.136776 0.032616252
x2           0.001057646 0.0003371302  3.137203 0.001705679
x3          -1.935677414 0.8973713429 -2.157053 0.031001547
R2log =  0.901401
Excluded as outliers:  Hibernate Log4J JFreeChart Findbugs
Checkstyle  ( 5 / 18 )
MMRE =  31.23992
Pred(25) =  50
Error range = [ -72.2469 .. 91.79771 ]
================================================================
Usability  vs.  Num. abstract classes ,  Num. attributes per
class ,  Num. parameters per method
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  4.77629374 1.332900667  3.583383 0.0003391726
x1          -0.02842533 0.008478548 -3.352618 0.0008005107
x2          -0.13833203 0.052459386 -2.636936 0.0083658678
x3          -2.07287411 0.860830245 -2.407994 0.0160404423
R2log =  0.9169942
Excluded as outliers:  Hibernate Xalan HttpUnit Saxon Findbugs
( 5 / 17 )
MMRE =  34.39253
Pred(25) =  52.94118
Error range = [ -96.55657 .. 123.6709 ]
================================================================
Usability  vs.  LCOM ,  Comment lines per class ,  NOC ,  Num.
abstract classes
                Estimate   Std. Error    z value    Pr(>|z|)
(Intercept) -0.3885141047 0.5997093370 -0.6478373 0.517090158
x1           0.0008921735 0.0003906076  2.2840657 0.022367670
x2          -0.0207495637 0.0065122369 -3.1862421 0.001441339
x3           1.5910160665 0.5598925139  2.8416455 0.004488136
x4          -0.0062013338 0.0024322630 -2.5496148 0.010784199
R2log =  0.8977364
Excluded as outliers:  JBoss JMeter  ( 2 / 17 )
MMRE =  19.71771
Pred(25) =  70.58824
Error range = [ -44.54104 .. 89.00059 ]
================================================================
Usability  vs.  LCOM ,  Comment lines per class ,  NOC ,  Num.
packages
                Estimate   Std. Error    z value     Pr(>|z|)
(Intercept) -0.1283061585 0.3722834065 -0.3446465 0.7303601727
x1           0.0006465408 0.0002948998  2.1924086 0.0283500195
x2          -0.0171102621 0.0061554603 -2.7796885 0.0054411069
x3           1.6358959351 0.4964274864  3.2953371 0.0009830366
x4          -0.0155278370 0.0062883510 -2.4693019 0.0135376948
R2log =  0.8293402
Excluded as outliers:  Eclipse Hibernate JBoss  ( 3 / 18 )
MMRE =  27.28973
Pred(25) =  66.66667
Error range = [ -99.61787 .. 71.11291 ]
================================================================
Usability  vs.  Comment lines per class ,  McCabe ,  NOC ,  Num.
packages
               Estimate  Std. Error   z value    Pr(>|z|)
(Intercept) -2.67364797 0.970312712 -2.755450 0.005861150
x1          -0.01145254 0.005810965 -1.970849 0.048741108
x2           1.46692206 0.503994693  2.910590 0.003607467
x3           1.28072502 0.516901521  2.477696 0.013223362
x4          -0.01689701 0.007021844 -2.406349 0.016112854
```

```
R2log =  0.8281923
Excluded as outliers:  Eclipse Hibernate JBoss Xerces Ant  ( 5 /
18 )
MMRE =  30.75647
Pred(25) =  50
Error range = [ -99.91814 .. 64.18284 ]
================================================================
Usability  vs.  Comment lines per class ,  McCabe ,  Num.
interfaces per class ,  Num. public methods
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.581208e+00 1.182140e+00 -2.183504 0.028998716
x1           4.166011e-02 1.541404e-02  2.702737 0.006877108
x2           1.266308e+00 4.225337e-01  2.996941 0.002727035
x3          -3.249089e+00 1.349876e+00 -2.406955 0.016086172
x4           3.829047e-05 1.822942e-05  2.100476 0.035686958
R2log =  0.9213594
Excluded as outliers:  Ant Xerces Xalan JFreeChart Weka  ( 5 /
17 )
MMRE =  35.13904
Pred(25) =  64.70588
Error range = [ -16.00766 .. 162.0532 ]
================================================================
Usability  vs.  eLOC per class ,  Num. interfaces per class ,
Num. methods per interface ,  Num. packages
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -3.80679301 1.38079091 -2.756966 0.0058340493
x1           0.09263505 0.02748807  3.370009 0.0007516573
x2          -4.48969476 1.20712765 -3.719321 0.0001997594
x3           0.69129507 0.25955821  2.663353 0.0077366279
x4          -0.03339003 0.01045659 -3.193206 0.0014070262
R2log =  0.8624878
Excluded as outliers:  Hibernate Eclipse Xerces JMeter JBoss  (
5 / 16 )
MMRE =  35.71356
Pred(25) =  56.25
Error range = [ -99.99997 .. 84.25082 ]
================================================================
Usability  vs.  loc_total
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  7.778421e-01 2.004441e-01  3.880595 0.0001042014
x1          -3.245123e-06 1.239917e-06 -2.617210 0.0088651730
R2log =  0.8985792
Excluded as outliers:  Hibernate Xalan HttpUnit  ( 3 / 15 )
MMRE =  27.93642
Pred(25) =  53.33333
Error range = [ -43.28456 .. 91.28609 ]
================================================================
Usability  vs.  number_of_developers
               Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -0.63378245 0.30410013 -2.084124 0.037148866
x1           0.03160120 0.01198271  2.637234 0.008358512
R2log =  0.853871
Excluded as outliers:  Saxon Eclipse Log4J  ( 3 / 15 )
MMRE =  17.91213
Pred(25) =  66.66667
Error range = [ -64.61552 .. 27.46912 ]
================================================================
Usability  vs.  number_of_commits
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -0.3444293370 0.2539060679 -1.356523 0.17493293
```

```
x1           0.0002540772 0.0001024132  2.480904 0.01310497
R2log =  0.9107858
Excluded as outliers:  Findbugs Saxon Xerces JBoss  ( 4 / 15 )
MMRE =  27.94476
Pred(25) =  40
Error range = [ -58.5081 .. 74.32911 ]
================================================================
Usability  vs.  avg_major_release_per_year
              Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.1523779  0.1989726 -0.7658238 0.44378115
x1           0.8204048  0.3683445  2.2272756 0.02592886
R2log =  0.8529176
Excluded as outliers:  Eclipse Spring Framework Saxon ( 3 / 15 )
MMRE =  26.68525
Pred(25) =  66.66667
Error range = [ -53.8021 .. 94.36208 ]
================================================================
Usability  vs.  avg_file_size
                Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.533360460 0.149853780  3.559206 0.0003719778
x1          -0.005106254 0.001960993 -2.603913 0.0092166256
R2log =  0.895109
Excluded as outliers:  JBoss Xalan  ( 2 / 15 )
MMRE =  22.26312
Pred(25) =  60
Error range = [ -40.9427 .. 72.14782 ]
================================================================
Usability  vs.  loc_total ,  number_of_developers
                 Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -2.677970e-01 3.390873e-01 -0.7897583 0.42966893
x1          -2.513747e-06 1.206899e-06 -2.0828151 0.03726808
x2           4.718344e-02 1.990810e-02  2.3700628 0.01778506
R2log =  0.9051717
Excluded as outliers:  Hibernate Saxon JBoss  ( 3 / 15 )
MMRE =  22.61816
Pred(25) =  73.33333
Error range = [ -56.12443 .. 39.99063 ]
================================================================
Usability  vs.  loc_total ,  avg_files_added_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  4.414103e-01 2.207972e-01  1.999167 0.045590317
x1          -3.414311e-06 1.220162e-06 -2.798244 0.005138134
x2           9.138793e-04 4.316301e-04  2.117274 0.034236583
R2log =  0.8418982
Excluded as outliers:  Spring Framework Eclipse Xalan  ( 3 / 15
)
MMRE =  31.49756
Pred(25) =  60
Error range = [ -47.12594 .. 87.55008 ]
================================================================
Usability vs. number_of_developers, avg_minor_release_per_year
               Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.11587350 0.24781534 -0.467580 0.64008495
x1           0.02846462 0.01208565  2.355241 0.01851070
x2          -0.14208698 0.06298908 -2.255740 0.02408692
R2log =  0.9136968
Excluded as outliers:  PMD Saxon  ( 2 / 15 )
MMRE =  19.71357
Pred(25) =  73.33333
Error range = [ -63.58232 .. 50.00066 ]
```

```
================================================================
Usability  vs.  avg_loc_del_per_year ,  avg_files_rem_per_year
                 Estimate   Std. Error    z value    Pr(>|z|)
(Intercept) -1.570248e-01 2.138302e-01 -0.7343434 0.462739448
x1          -9.934319e-06 3.341753e-06 -2.9727871 0.002951091
x2           3.996071e-03 1.220320e-03  3.2746094 0.001058081
R2log =  0.8719533
Excluded as outliers: Spring Framework Struts Eclipse ( 3 / 15 )
MMRE =  25.19746
Pred(25) =  66.66667
Error range = [ -18.17691 .. 105.1296 ]
================================================================
Usability  vs.  files_count_total ,  number_of_commits ,
avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  3.1192780826 8.764064e-01  3.559168 0.0003720309
x1          -0.0002417612 8.904118e-05 -2.715162 0.0066243402
x2           0.0005509842 1.803138e-04  3.055696 0.0022453875
x3          -0.5800600276 1.782248e-01 -3.254654 0.0011353048
R2log =  0.8622824
Excluded as outliers: HttpUnit Eclipse Jasper Reports ( 3 / 15 )
MMRE =  24.14804
Pred(25) =  66.66667
Error range = [ -90.28245 .. 51.10075 ]
================================================================
Usability  vs. files_count_total,  number_of_commits,
avg_loc_del_per_year,  avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  3.620110e+00 9.310747e-01  3.888099 0.0001010325
x1          -2.330507e-04 8.670948e-05 -2.687719 0.0071941882
x2           5.942523e-04 1.762725e-04  3.371213 0.0007483800
x3          -4.121874e-06 1.776910e-06 -2.319686 0.0203578578
x4          -6.585444e-01 1.824790e-01 -3.608877 0.0003075252
R2log =  0.9157867
Excluded as outliers:  HttpUnit Jasper Reports  ( 2 / 15 )
MMRE =  20.25183
Pred(25) =  66.66667
Error range = [ -93.3342 .. 38.24876 ]
================================================================
Usability  vs.  number_of_commits ,  avg_loc_added_per_year ,

avg_files_added_per_year ,  avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  3.262223e+00 8.237199e-01  3.960354 7.483863e-05
x1           5.647300e-04 1.799399e-04  3.138437 1.698513e-03
x2          -1.631638e-05 7.213468e-06 -2.261932 2.370159e-02
x3          -3.359794e-04 1.368256e-04 -2.455531 1.406768e-02
x4          -6.231798e-01 1.756165e-01 -3.548527 3.873927e-04
R2log =  0.8589273
Excluded as outliers:  Eclipse Hibernate HttpUnit Jasper Reports
( 4 / 15 )
MMRE =  30.31601
Pred(25) =  66.66667
Error range = [ -94.62795 .. 15.53882 ]
================================================================
Usability  vs.  number_of_commits ,  avg_loc_del_per_year ,

avg_major_release_per_year ,  avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  4.867469e+00 1.503066e+00  3.238361 0.001202188
```

```
x1           6.755161e-04 2.165131e-04  3.119977 0.001808649
x2          -2.135499e-05 8.198418e-06 -2.604770 0.009193595
x3          -1.239872e+00 5.372658e-01 -2.307745 0.021013348
x4          -8.364053e-01 2.591819e-01 -3.227098 0.001250528
R2log =  0.8635258
Excluded as outliers:  Eclipse HttpUnit Hibernate Jasper Reports
( 4 / 15 )
MMRE =  26.87195
Pred(25) =  66.66667
Error range = [ -97.1518 .. 49.09983 ]
================================================================
Usability  vs.  number_of_commits ,  avg_loc_del_per_year ,
avg_files_added_per_year ,  avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  3.255355e+00 8.273757e-01  3.934555 8.335114e-05
x1           5.539546e-04 1.769552e-04  3.130479 1.745214e-03
x2          -1.674359e-05 7.106329e-06 -2.356151 1.846539e-02
x3          -3.319648e-04 1.360686e-04 -2.439688 1.469996e-02
x4          -6.266876e-01 1.773827e-01 -3.532969 4.109205e-04
R2log =  0.8609328
Excluded as outliers:  Eclipse Hibernate HttpUnit Jasper Reports
( 4 / 15 )
MMRE =  27.32718
Pred(25) =  66.66667
Error range = [ -93.27653 .. 14.87284 ]
================================================================
Usability vs. avg_loc_changed_per_year,
avg_minor_release_per_year,avg_file_size, avg_files_rem_per_year
                  Estimate   Std. Error    z value    Pr(>|z|)
(Intercept)  2.316585e-01 3.595069e-01  0.6443786 0.519329952
x1          -1.792039e-05 6.301740e-06 -2.8437212 0.004459005
x2          -3.466954e-01 1.080971e-01 -3.2072585 0.001340065
x3           6.996326e-03 3.075602e-03  2.2747823 0.022918994
x4           2.723345e-03 1.035564e-03  2.6298170 0.008543083
R2log =  0.918734
Excluded as outliers: Spring Framework PMD JFreeChart ( 3 / 15 )
MMRE =  30.86335
Pred(25) =  73.33333
Error range = [ -82.74213 .. 160.4847 ]
================================================================
```

Portability

```
================================================================
Portability  vs.  NOC
               Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.08946496  0.3003584 -0.2978606 0.76580953
x1           0.60899752  0.2794756  2.1790725 0.02932628
R2log =  0.8921334
Excluded as outliers:  JBoss  ( 1 / 18 )
MMRE =  11.80279
Pred(25) =  88.88889
Error range = [ -24.37454 .. 52.83443 ]
================================================================
Portability  vs.  Num. packages
               Estimate  Std. Error    z value   Pr(>|z|)
(Intercept) -0.03566876 0.278962725 -0.1278621 0.89825809
x1           0.01030754 0.005156904  1.9987845 0.04563167
R2log =  0.8320353
Excluded as outliers:  Hibernate Eclipse Log4J  ( 3 / 18 )
```

```
MMRE =  18.88812
Pred(25) =  83.33333
Error range = [ -18.62236 .. 72.92966 ]
================================================================
Portability  vs.  eLOC per class ,  Num. interfaces per class
               Estimate  Std. Error    z value    Pr(>|z|)
(Intercept)  0.05574562 0.406595952  0.1371032 0.890949199
x1           0.01682395 0.007527396  2.2350290 0.025415445
x2          -1.45046472 0.516305378 -2.8093155 0.004964696
R2log =  0.8986532
Excluded as outliers:   ( 0 / 18 )
MMRE =  7.343792
Pred(25) =  94.44444
Error range = [ -13.23063 .. 36.66999 ]
================================================================
Portability  vs.  McCabe ,  Num. parameters per method
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.7759051  0.6195523  1.252364 0.21043720
x1           0.6348988  0.2915224  2.177873 0.02941549
x2          -1.7554705  0.7593420 -2.311831 0.02078699
R2log =  0.8961848
Excluded as outliers:   ( 0 / 18 )
MMRE =  10.73470
Pred(25) =  88.88889
Error range = [ -17.90362 .. 33.29071 ]
================================================================
```

How well are functional requirements satisfied

```
================================================================
Functionality  vs.  CBO ,  eLOC per class
               Estimate  Std. Error   z value   Pr(>|z|)
(Intercept)  0.57318841 0.462322826  1.239801 0.21504892
x1           0.40247103 0.193051783  2.084783 0.03708902
x2          -0.02246685 0.009353803 -2.401895 0.01631039
R2log =  0.8652167
Excluded as outliers:  Eclipse Hibernate HttpUnit Xalan Saxon  (
5 / 18 )
MMRE =  31.71418
Pred(25) =  66.66667
Error range = [ -21.12416 .. 112.9904 ]
================================================================
Functionality  vs.  Comment lines per class ,  NOC
               Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  0.75324411 0.34681827  2.171870 0.029865491
x1          -0.03019243 0.01130772 -2.670072 0.007583506
x2           1.18332002 0.49346299  2.397991 0.016485250
R2log =  0.8677023
Excluded as outliers:  Hibernate Eclipse JFreeChart HttpUnit  (
4 / 18 )
MMRE =  24.35349
Pred(25) =  72.22222
Error range = [ -84.35382 .. 98.25808 ]
================================================================
Functionality  vs.  eLOC ,  McCabe
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.691198e+00 1.075275e+00 -2.502800 0.012321521
x1          -7.895661e-06 3.804496e-06 -2.075350 0.037954094
```

```
x2           1.508519e+00 5.352988e-01  2.818087 0.004831068
R2log =  0.8661916
Excluded as outliers:  Eclipse Hibernate Xerces Jasper Reports
( 4 / 18 )
MMRE =  20.18275
Pred(25) =  72.22222
Error range = [ -69.7415 .. 44.32345 ]
================================================================
Functionality  vs.  McCabe ,  Num. interfaces per class
              Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.3181390  0.5992554 -0.5308905 0.59549465
x1           0.5367714  0.2548872  2.1059174 0.03521153
x2          -1.3836324  0.6945088 -1.9922462 0.04634405
R2log =  0.866334
Excluded as outliers:  JFreeChart Eclipse  ( 2 / 18 )
MMRE =  17.77286
Pred(25) =  83.33333
Error range = [ -67.45904 .. 42.14344 ]
================================================================
Functionality  vs.  McCabe ,  Num. methods
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.459630e+00 1.000538e+00 -2.458308 0.013959340
x1           1.430944e+00 5.002081e-01  2.860698 0.004227099
x2          -4.273505e-05 2.030307e-05 -2.104857 0.035303759
R2log =  0.866268
Excluded as outliers:  Eclipse Hibernate Xerces Jasper Reports
( 4 / 18 )
MMRE =  21.10365
Pred(25) =  66.66667
Error range = [ -77.84831 .. 46.04495 ]
================================================================
Functionality  vs.  McCabe ,  Num. methods per class
               Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.44703391 0.46984673 -0.9514463 0.34137789
x1           0.55810373 0.25377921  2.1991704 0.02786581
x2          -0.03706245 0.01864084 -1.9882395 0.04678521
R2log =  0.8956569
Excluded as outliers:  Log4J  ( 1 / 18 )
MMRE =  17.57399
Pred(25) =  88.88889
Error range = [ -20.81359 .. 66.0328 ]
================================================================
Functionality  vs.  Num. attributes per class ,  Num. methods
per interface
              Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -2.4070712 0.95773904 -2.513285 0.011961267
x1           0.1377325 0.06042544  2.279379 0.022644534
x2           0.8465572 0.31828442  2.659751 0.007819852
R2log =  0.8466062
Excluded as outliers:  Eclipse JBoss Xalan Ant Log4J  ( 5 / 16 )
MMRE =  25.61057
Pred(25) =  68.75
Error range = [ -53.02712 .. 100.5544 ]
================================================================
Functionality  vs.  LCOM ,  Num. abstract classes ,  Num.
interfaces per class
                 Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -0.3440648918 0.5759262184 -0.5974114 0.55023274
x1           0.0009067416 0.0003972887  2.2823244 0.02247020
x2           0.0161623534 0.0076233293  2.1201174 0.03399614
```

```
x3          -1.9472984747 0.8499405928 -2.2910995 0.02195766
R2log =  0.8607087
Excluded as outliers:  Hibernate JFreeChart Eclipse Log4J
Checkstyle  ( 5 / 17 )
MMRE =  22.33808
Pred(25) =  70.58824
Error range = [ -71.09287 .. 93.00308 ]
================================================================
Functionality  vs.  Comment lines per class ,  McCabe ,  NOC
               Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.90524604 0.63925970 -1.416085 0.15675063
x1          -0.02397907 0.01025939 -2.337280 0.01942465
x2           0.74996722 0.33629560  2.230083 0.02574191
x3           0.70611936 0.32847338  2.149700 0.03157892
R2log =  0.9037505
Excluded as outliers:  JFreeChart Hibernate  ( 2 / 18 )
MMRE =  18.80307
Pred(25) =  77.77778
Error range = [ -76.74598 .. 54.52898 ]
================================================================
Functionality  vs.  Comment lines per class ,  NOC ,  Num.
packages
                Estimate  Std. Error   z value    Pr(>|z|)
(Intercept)  0.861668404 0.360508339  2.390148 0.016841569
x1          -0.031499653 0.011249000 -2.800218 0.005106810
x2           1.364286113 0.487051252  2.801114 0.005092651
x3          -0.003893344 0.001285312 -3.029105 0.002452796
R2log =  0.912421
Excluded as outliers:  JFreeChart HttpUnit  ( 2 / 18 )
MMRE =  18.5524
Pred(25) =  83.33333
Error range = [ -85.57875 .. 106.8532 ]
================================================================
Functionality  vs.  eLOC per class ,  NOC ,  Num. interfaces per
class
               Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  0.95175740 0.52179774  1.823997 0.068152558
x1           0.02836477 0.01219481  2.325970 0.020020157
x2          -1.15776465 0.55836437 -2.073493 0.038126416
x3          -3.77770708 1.38547711 -2.726647 0.006398144
R2log =  0.866799
Excluded as outliers:  JFreeChart JBoss Eclipse JMeter  ( 4 / 18
)
MMRE =  22.85829
Pred(25) =  66.66667
Error range = [ -91.15188 .. 59.93853 ]
================================================================
Functionality  vs.  files_count_total
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -0.3352351315 0.2751952614 -1.218172 0.22315866
x1           0.0003001768 0.0001195007  2.511926 0.01200744
R2log =  0.858213
Excluded as outliers:  Spring Framework Hibernate Eclipse Struts
( 4 / 15 )
MMRE =  21.33821
Pred(25) =  73.33333
Error range = [ -23.62299 .. 78.58765 ]
================================================================
Functionality  vs.  number_of_commits
                 Estimate   Std. Error    z value   Pr(>|z|)
```

```
(Intercept) -0.1317928171 1.779582e-01 -0.7405832 0.45894620
x1           0.0001525140 6.316434e-05  2.4145581 0.01575431
R2log =  0.905931
Excluded as outliers:   ( 0 / 15 )
MMRE =  12.83085
Pred(25) =  93.33333
Error range = [ -22.13112 .. 38.01295 ]
================================================================
Functionality  vs.  files_count_total ,  avg_loc_del_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.365897e-01 2.877580e-01 -1.517212 0.129213271
x1           4.041612e-04 1.426846e-04  2.832549 0.004617842
x2          -9.277479e-06 3.565440e-06 -2.602057 0.009266636
R2log =  0.8743656
Excluded as outliers: Spring Framework Eclipse Struts ( 3 / 15 )
MMRE =  19.43088
Pred(25) =  80
Error range = [ -38.53268 .. 89.27538 ]
================================================================
Functionality vs. avg_loc_added_per_year, avg_files_rem_per_year
                 Estimate   Std. Error    z value    Pr(>|z|)
(Intercept) -7.130892e-02 1.894508e-01 -0.3763980 0.706620997
x1          -5.884354e-06 2.515269e-06 -2.3394532 0.019311990
x2           2.782665e-03 1.074880e-03  2.5888146 0.009630693
R2log =  0.914996
Excluded as outliers:  Spring Framework Saxon Struts  ( 3 / 15 )
MMRE =  22.46386
Pred(25) =  66.66667
Error range = [ -29.36840 .. 99.97384 ]
================================================================
Functionality vs. avg_loc_del_per_year,
avg_major_release_per_year, avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.104056e+00 8.988395e-01 -2.340859 0.019239437
x1           7.850266e-06 3.386075e-06  2.318397 0.020427749
x2           1.298017e+00 5.913246e-01  2.195100 0.028156395
x3           2.286458e-01 8.857330e-02  2.581430 0.009839187
R2log =  0.9198223
Excluded as outliers: Spring Framework Saxon HttpUnit ( 3 / 15 )
MMRE =  21.49962
Pred(25) =  80
Error range = [ -73.17863 .. 87.19398 ]
================================================================
Functionality  vs.  avg_major_release_per_year ,
avg_files_added_per_year ,

avg_number_of_revisions_per_file
                  Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.493288276 1.0328643728 -2.413955 0.015780405
x1           1.516011947 0.5879548786  2.578449 0.009924481
x2           0.001159953 0.0004417274  2.625948 0.008640811
x3           0.211383251 0.0883636005  2.392198 0.016747793
R2log =  0.9138229
Excluded as outliers:  Spring Framework Saxon  ( 2 / 15 )
MMRE =  20.58450
Pred(25) =  80
Error range = [ -55.80254 .. 99.82894 ]
================================================================
Functionality vs. avg_major_release_per_year,
avg_files_rem_per_year, avg_number_of_revisions_per_file
```

```
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -1.480215550 0.6558593996 -2.256910 0.02401371
x1           0.810028823 0.3943639308  2.054013 0.03997439
x2           0.001904427 0.0008035833  2.369919 0.01779198
x3           0.134256715 0.0641735824  2.092087 0.03643074
R2log =  0.8705076
Excluded as outliers:  Spring Framework Eclipse  ( 2 / 15 )
MMRE =  16.58737
Pred(25) =  86.66667
Error range = [ -17.02919 .. 99.78273 ]
================================================================
```

Interoperability

```
================================================================
Interoperability  vs.  Comment lines per class ,  Num. methods
per interface
               Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -1.70738769  0.8020219 -2.128854 0.03326632
x1           0.02061113  0.0099192  2.077902 0.03771838
x2           0.66748273  0.2689258  2.482033 0.01306351
R2log =  0.8273493
Excluded as outliers:  Eclipse JFreeChart JBoss Log4J Hibernate
( 5 / 16 )
MMRE =  27.77385
Pred(25) =  62.5
Error range = [ -32.49771 .. 100.7163 ]
================================================================
Interoperability  vs.  NOC ,  RFC
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  3.995809e-01 3.199987e-01  1.248695 0.21177651
x1           1.055025e+00 4.824227e-01  2.186931 0.02874754
x2          -3.045737e-05 1.321377e-05 -2.304973 0.02116811
R2log =  0.8147798
Excluded as outliers: Eclipse Hibernate Ant Log4J HttpUnit
(5/18)
MMRE =  26.1796
Pred(25) =  61.11111
Error range = [ -49.58174 .. 109.7635 ]
================================================================
Interoperability  vs.  Num. attributes per class ,  Num. public
methods
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  3.414012e-01 3.052651e-01  1.118376 0.26340637
x1           4.984285e-02 2.291856e-02  2.174781 0.02964653
x2          -5.186182e-05 2.562597e-05 -2.023799 0.04299082
R2log =  0.8861481
Excluded as outliers:  Hibernate Log4J  ( 2 / 17 )
MMRE =  22.49075
Pred(25) =  64.70588
Error range = [ -64.2106 .. 79.68061 ]
================================================================
Interoperability  vs.  Comment lines ,  Comment lines per class,
Num. abstract classes
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  1.439551e+00 5.532372e-01  2.602051 0.009266816
x1           2.243767e-05 1.005081e-05  2.232425 0.025586893
```

```
x2          -2.472401e-02 1.118238e-02 -2.210979 0.027037321
x3          -2.506639e-02 1.123042e-02 -2.232009 0.025614349
R2log =  0.8330125
Excluded as outliers: Eclipse Hibernate Spring Framework (3/17)
MMRE =  24.87753
Pred(25) =  58.82353
Error range = [ -74.0165 .. 74.21646 ]
================================================================
Interoperability  vs.  avg_loc_added_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  5.417006e-01 2.059174e-01  2.630669 0.008521697
x1          -3.194263e-05 1.608166e-05 -1.986277 0.047002564
R2log =  0.8383662
Excluded as outliers:  Eclipse Hibernate PMD JMeter  ( 4 / 15 )
MMRE =  37.57031
Pred(25) =  53.33333
Error range = [ -99.94197 .. 77.66751 ]
================================================================
Interoperability  vs.  loc_total ,  avg_major_release_per_year
                  Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -5.634165e-01 3.754854e-01 -1.500501 0.13348456
x1           2.749139e-06 1.316279e-06  2.088569 0.03674658
x2           6.755571e-01 3.153776e-01  2.142058 0.03218883
R2log =  0.8356012
Excluded as outliers:  Hibernate Eclipse  ( 2 / 15 )
MMRE =  14.39924
Pred(25) =  86.66667
Error range = [ -23.43169 .. 43.91985 ]
================================================================
Interoperability  vs.  avg_major_release_per_year ,
avg_file_size
                Estimate  Std. Error   z value   Pr(>|z|)
(Intercept) -0.796471575 0.412101375 -1.932708 0.05327219
x1           1.200617119 0.539953697  2.223556 0.02617835
x2           0.009012478 0.003470130  2.597158 0.00939986
R2log =  0.8541156
Excluded as outliers:  JFreeChart JBoss Eclipse Spring Framework
( 4 / 15 )
MMRE =  21.00227
Pred(25) =  73.33333
Error range = [ -22.35049 .. 116.9828 ]
================================================================
Interoperability  vs.  loc_total ,  avg_loc_del_per_year ,
avg_files_rem_per_year
                  Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -5.110442e-01 3.325558e-01 -1.536717 0.124362576
x1           3.136311e-06 1.110728e-06  2.823653 0.004747977
x2          -1.296071e-05 4.984656e-06 -2.600121 0.009319091
x3           3.028319e-03 1.307945e-03  2.315326 0.020595094
R2log =  0.9116604
Excluded as outliers:  Spring Framework Struts Saxon  ( 3 / 15 )
MMRE =  24.67779
Pred(25) =  66.66667
Error range = [ -32.02462 .. 110.5184 ]
================================================================
```

Security

```
============================================================
Security  vs.  eLOC
                Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -1.287702e-01 2.129536e-01 -0.6046866 0.54538725
x1           7.596619e-06 3.789195e-06  2.0048110 0.04498326
R2log =  0.8443458
Excluded as outliers:  Eclipse Hibernate  ( 2 / 17 )
MMRE =  20.90020
Pred(25) =  76.47059
Error range = [ -20.31237 .. 68.18477 ]
============================================================
Security  vs.  avg_loc_added_per_year
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  2.925696e-01 1.685849e-01  1.735444 0.08266219
x1          -2.183443e-06 8.883412e-07 -2.457887 0.01397571
R2log =  0.8936093
Excluded as outliers:  Log4J  ( 1 / 15 )
MMRE =  14.76664
Pred(25) =  73.33333
Error range = [ -26.76824 .. 36.76835 ]
============================================================
Security  vs.  avg_files_added_per_year
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  0.4634404040 0.2320813363  1.996888 0.04583737
x1          -0.0004971651 0.0001987493 -2.501469 0.01236793
R2log =  0.8967454
Excluded as outliers:  Spring Framework Log4J  ( 2 / 15 )
MMRE =  19.82717
Pred(25) =  66.66667
Error range = [ -79.1386 .. 38.40918 ]
============================================================
Security  vs.  avg_files_rem_per_year
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  0.275131080 0.1919297926  1.433499 0.15171541
x1          -0.000798813 0.0003687744 -2.166129 0.03030134
R2log =  0.8949154
Excluded as outliers:  Spring Framework  ( 1 / 15 )
MMRE =  18.47417
Pred(25) =  86.66667
Error range = [ -84.43458 .. 24.74118 ]
============================================================
Security  vs.  loc_total ,  files_count_total
                Estimate   Std. Error    z value   Pr(>|z|)
(Intercept)  1.097497e-01 3.251229e-01  0.3375638 0.73569191
x1           2.900640e-06 1.262621e-06  2.2973169 0.02160070
x2          -2.372309e-04 1.064364e-04 -2.2288501 0.02582388
R2log =  0.9049571
Excluded as outliers:  Spring Framework Hibernate PMD  ( 3 / 15
)
MMRE =  18.60600
Pred(25) =  73.33333
Error range = [ -76.15355 .. 33.53208 ]
============================================================
```

Speed

```
============================================================
```

```
Speed  vs.  LCOM
                Estimate   Std. Error    z value   Pr(>|z|)
(Intercept)  0.0593518625 0.1842068505  0.3222023 0.74729948
x1          -0.0005567389 0.0002533943 -2.1971246 0.02801155
R2log =  0.8597402
Excluded as outliers:  Eclipse  ( 1 / 18 )
MMRE =  20.49962
Pred(25) =  66.66667
Error range = [ -34.1952 .. 107.4788 ]
============================================================
Speed  vs.  eLOC per class
               Estimate  Std. Error   z value   Pr(>|z|)
(Intercept)  0.57573620 0.417045975  1.380510 0.16742962
x1          -0.01103935 0.005529275 -1.996527 0.04587658
R2log =  0.8507292
Excluded as outliers:  Eclipse Hibernate  ( 2 / 18 )
MMRE =  24.35536
Pred(25) =  61.11111
Error range = [ -39.44572 .. 85.54478 ]
============================================================
Speed  vs.  Num. attributes per class
               Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.47525150 0.36681951  1.295600 0.19511319
x1          -0.08452554 0.04175597 -2.024274 0.04294193
R2log =  0.8368939
Excluded as outliers:  Eclipse Xalan Hibernate Log4J  ( 4 / 18 )
MMRE =  32.14329
Pred(25) =  50
Error range = [ -82.81606 .. 109.7569 ]
============================================================
Speed  vs.  Num. intefaces
                Estimate  Std. Error    z value   Pr(>|z|)
(Intercept)  0.159461399 0.228882144  0.6966965 0.48599272
x1          -0.004512853 0.002012041 -2.2429233 0.02490176
R2log =  0.850113
Excluded as outliers:  Eclipse JBoss Hibernate Jasper Reports  (
4 / 18 )
MMRE =  24.31349
Pred(25) =  66.66667
Error range = [ -71.74088 .. 45.28392 ]
============================================================
Speed  vs.  Num. public methods
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  4.002294e-01 3.149863e-01  1.270625 0.20386221
x1          -6.594226e-05 2.927509e-05 -2.252504 0.02429044
R2log =  0.8403567
Excluded as outliers:  Hibernate Eclipse Weka Log4J  ( 4 / 17 )
MMRE =  28.49705
Pred(25) =  47.05882
Error range = [ -68.76518 .. 64.67592 ]
============================================================
Speed  vs.  LCOM ,  Num. public methods
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  5.795781e-01 3.273662e-01  1.770427 0.07665599
x1          -8.215582e-04 3.277375e-04 -2.506757 0.01218445
x2          -2.217034e-05 1.111118e-05 -1.995319 0.04600813
R2log =  0.8593725
Excluded as outliers:  Eclipse Ant  ( 2 / 17 )
MMRE =  20.40114
Pred(25) =  82.35294
```

```
Error range = [ -27.64666 .. 131.4864 ]
================================================================
Speed  vs.  Num. abstract classes ,  Num. attributes per class ,
Num. parameters per method
                Estimate  Std. Error   z value    Pr(>|z|)
(Intercept)  3.034775191 1.151299155  2.635957 0.008390034
x1          -0.005798768 0.002619608 -2.213601 0.026856244
x2          -0.092617308 0.041098177 -2.253562 0.024223709
x3          -2.249889428 0.994261315 -2.262875 0.023643374
R2log =  0.872414
Excluded as outliers:  Eclipse Xalan  ( 2 / 17 )
MMRE =  24.65641
Pred(25) =  88.2353
Error range = [ -88.89463 .. 155.4849 ]
================================================================
```

Utility of the product developer community

```
================================================================
Community  vs.  CBO
             Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.9677939  0.4394584  2.202242 0.02764820
x1          -0.2614979  0.1155526 -2.263020 0.02363444
R2log =  0.8386677
Excluded as outliers:  Eclipse Hibernate Log4J  ( 3 / 16 )
MMRE =  27.88297
Pred(25) =  56.25
Error range = [ -99.99969 .. 74.70896 ]
================================================================
Community  vs.  LCOM ,  Num. interfaces per class
               Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -0.9956297435 0.5119546850 -1.944761 0.05180369
x1          -0.0008882355 0.0004296822 -2.067191 0.03871612
x2           3.2623873793 1.3738567668  2.374620 0.01756704
R2log =  0.8692818
Excluded as outliers:  JFreeChart Eclipse PMD JMeter  ( 4 / 16 )
MMRE =  31.54234
Pred(25) =  56.25
Error range = [ -67.54058 .. 93.10946 ]
================================================================
Community  vs.  NOC ,  Num. attributes per class
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.86101589 0.39627300  2.172785 0.02979653
x1          -0.52963631 0.26543432 -1.995357 0.04600392
x2          -0.06147217 0.03048413 -2.016530 0.04374458
R2log =  0.9018403
Excluded as outliers:  Xalan  ( 1 / 16 )
MMRE =  21.82209
Pred(25) =  81.25
Error range = [ -75.64863 .. 110.1620 ]
================================================================
Community  vs.  McCabe ,  Num. attributes per class ,  Num.
interfaces per class
              Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -2.9151999 1.31719787 -2.213183 0.026885034
x1           1.1303664 0.52643170  2.147223 0.031775502
x2          -0.2355771 0.09133947 -2.579138 0.009904711
x3           5.5958856 2.15647225  2.594926 0.009461133
R2log =  0.8685826
```

```
Excluded as outliers:  Xalan JFreeChart Xerces Eclipse Log4J  (
5 / 16 )
MMRE =  33.97902
Pred(25) =  62.5
Error range = [ -99.84973 .. 99.80573 ]
================================================================
Community  vs.  LCOM ,  Comment lines per class ,  NOC ,  Num.
packages
                Estimate   Std. Error    z value    Pr(>|z|)
(Intercept)  0.0460336297 0.4411546737  0.1043480 0.916893148
x1          -0.0007782302 0.0003437119 -2.2641930 0.023562247
x2           0.0364207528 0.0149238137  2.4404454 0.014669162
x3          -1.9128102380 0.6182737628 -3.0937917 0.001976162
x4           0.0032385674 0.0014897368  2.1739192 0.029711205
R2log =  0.9135916
Excluded as outliers:  JFreeChart PMD  ( 2 / 16 )
MMRE =  20.92991
Pred(25) =  68.75
Error range = [ -73.59067 .. 83.28782 ]
================================================================
Community  vs.  loc_total ,  files_count_total
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -1.070226e+00 3.464006e-01 -3.089560 0.002004530
x1           2.529075e-06 1.176559e-06  2.149553 0.031590615
x2           1.773488e-04 7.402238e-05  2.395881 0.016580476
R2log =  0.9035187
Excluded as outliers:  Hibernate JBoss  ( 2 / 14 )
MMRE =  20.3453
Pred(25) =  71.42857
Error range = [ -34.34001 .. 54.29452 ]
================================================================
Community  vs.  loc_total ,  avg_major_release_per_year
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -1.292636e+00 4.478521e-01 -2.886301 0.003897995
x1           4.146629e-06 1.490031e-06  2.782915 0.005387292
x2           8.455055e-01 3.530812e-01  2.394649 0.016636294
R2log =  0.846061
Excluded as outliers:  Eclipse Hibernate  ( 2 / 14 )
MMRE =  17.86593
Pred(25) =  64.28571
Error range = [ -34.0511 .. 45.39039 ]
================================================================
Community  vs.  loc_total ,  avg_files_added_per_year
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -7.369177e-01 2.653171e-01 -2.777498 0.005477914
x1           2.808696e-06 1.215897e-06  2.309979 0.020889301
x2           2.209786e-04 1.073166e-04  2.059128 0.039481999
R2log =  0.8988746
Excluded as outliers:  Hibernate  ( 1 / 14 )
MMRE =  21.02635
Pred(25) =  64.28571
Error range = [ -22.79198 .. 54.85847 ]
================================================================
Community  vs.  loc_total ,  avg_number_of_revisions_per_file
                Estimate   Std. Error    z value   Pr(>|z|)
(Intercept)  2.523388e-01 3.437476e-01  0.7340815 0.46289908
x1           2.851668e-06 1.225212e-06  2.3274896 0.01993922
x2          -1.232537e-01 5.870196e-02 -2.0996517 0.03575949
R2log =  0.8997666
Excluded as outliers:  Hibernate  ( 1 / 14 )
```

```
MMRE =  21.14715
Pred(25) =  78.57143
Error range = [ -23.13899 .. 65.07981 ]
================================================================
Community  vs.  avg_loc_del_per_year ,  avg_files_added_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -5.693474e-01 2.311753e-01 -2.462838 0.013784223
x1          -1.110716e-05 4.481462e-06 -2.478469 0.013194775
x2           1.507830e-03 5.719926e-04  2.636101 0.008386473
R2log =  0.9059014
Excluded as outliers:  Spring Framework  ( 1 / 14 )
MMRE =  18.99841
Pred(25) =  57.14286
Error range = [ -29.68839 .. 72.54569 ]
================================================================
Community vs. avg_major_release_per_year,
avg_minor_release_per_year
              Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -0.9762137  0.3385586 -2.883441 0.003933566
x1           0.5779543  0.2803257  2.061724 0.039234036
x2           0.2406570  0.0889429  2.705748 0.006815083
R2log =  0.8611813
Excluded as outliers:  Eclipse PMD  ( 2 / 14 )
MMRE =  26.37751
Pred(25) =  71.42857
Error range = [ -38.92617 .. 83.0344 ]
================================================================
Community vs. avg_minor_release_per_year,
avg_files_added_per_year
                  Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -0.7566435522 0.2716702479 -2.785154 0.005350226
x1           0.1956204075 0.0807010714  2.424013 0.015350076
x2           0.0002111372 0.0001076161  1.961949 0.049768450
R2log =  0.8598602
Excluded as outliers:  PMD Eclipse  ( 2 / 14 )
MMRE =  21.87745
Pred(25) =  71.42857
Error range = [ -27.86525 .. 66.71854 ]
================================================================
Community  vs.  loc_total ,  files_count_total ,
number_of_commits
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -4.600312e-01 3.463251e-01 -1.328322 0.18407179
x1           3.196406e-06 1.253326e-06  2.550339 0.01076181
x2           1.572793e-04 7.189490e-05  2.187627 0.02869675
x3          -2.250664e-04 1.091472e-04 -2.062044 0.03920356
R2log =  0.9094875
Excluded as outliers:  Hibernate Findbugs  ( 2 / 14 )
MMRE =  21.05915
Pred(25) =  64.28571
Error range = [ -54.33324 .. 79.84851 ]
================================================================
Community vs. files_count_total, number_of_commits,
avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -2.7416309685 1.1164573016 -2.455652 0.01406291
x1           0.0003873338 0.0001635913  2.367692 0.01789943
x2          -0.0003644824 0.0001465291 -2.487441 0.01286660
x3           0.3651017260 0.1555691968  2.346877 0.01893152
R2log =  0.8645516
```

```
Excluded as outliers:  HttpUnit JBoss Eclipse Spring Framework
( 4 / 14 )
MMRE =  28.55432
Pred(25) =  78.57143
Error range = [ -50.90839 .. 191.5555 ]
================================================================
Community  vs.  number_of_commits ,  avg_files_rem_per_year ,

avg_number_of_revisions_per_file
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.6776874060 1.0424045922 -2.568760 0.010206306
x1          -0.0003904030 0.0001491933 -2.616760 0.008876867
x2           0.0032843184 0.0013142444  2.499017 0.012453850
x3           0.4173302751 0.1651916681  2.526340 0.011525796
R2log =  0.8675318
Excluded as outliers:  HttpUnit Spring Framework Eclipse JBoss
( 4 / 14 )
MMRE =  34.34006
Pred(25) =  71.42857
Error range = [ -44.38507 .. 220.2552 ]
================================================================
Community  vs.  avg_loc_added_per_year ,  avg_file_size ,
avg_files_rem_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -1.201678e+00 3.932606e-01 -3.055679 0.002245517
x1          -4.341128e-06 2.169166e-06 -2.001289 0.045361215
x2           6.579496e-03 2.381818e-03  2.762384 0.005738096
x3           2.823072e-03 1.089072e-03  2.592182 0.009536927
R2log =  0.909533
Excluded as outliers:  Spring Framework  ( 1 / 14 )
MMRE =  16.94885
Pred(25) =  85.71429
Error range = [ -21.70795 .. 72.6659 ]
================================================================
Community  vs.  avg_loc_changed_per_year ,
avg_major_release_per_year , avg_minor_release_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -1.451040e+00 4.859238e-01 -2.986147 0.002825171
x1           1.127296e-05 4.483431e-06  2.514361 0.011924823
x2           9.130654e-01 3.686978e-01  2.476460 0.013269254
x3           2.903518e-01 9.890319e-02  2.935718 0.003327770
R2log =  0.9095433
Excluded as outliers:  PMD  ( 1 / 14 )
MMRE =  20.55266
Pred(25) =  71.42857
Error range = [ -31.99209 .. 89.71474 ]
================================================================
Community  vs.  avg_loc_changed_per_year ,  avg_file_size ,
avg_files_rem_per_year
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -1.177210e+00 3.889463e-01 -3.026665 0.002472681
x1          -1.623266e-05 7.798504e-06 -2.081510 0.037387260
x2           7.732515e-03 2.598664e-03  2.975573 0.002924416
x3           2.428498e-03 9.600547e-04  2.529541 0.011421183
R2log =  0.8669093
Excluded as outliers:  Spring Framework Eclipse  ( 2 / 14 )
MMRE =  17.74764
Pred(25) =  85.71429
Error range = [ -40.91654 .. 72.51251 ]
================================================================
```

Documentation Quality

```
================================================================
DocQuality  vs.  Comment lines ,  Comment lines per class ,
Num. abstract classes , Num. methods per interface
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.320117e+00 1.504010e+00 -2.872399 0.004073678
x1          -2.002578e-05 9.417799e-06 -2.126376 0.033471931
x2           5.127531e-02 1.963857e-02  2.610949 0.009029128
x3           3.050114e-02 1.346274e-02  2.265597 0.023476057
x4           6.536364e-01 2.769001e-01  2.360549 0.018247889
R2log =  0.8391869
Excluded as outliers:  JFreeChart Eclipse Log4J Hibernate JBoss
( 5 / 16 )
MMRE =  39.98816
Pred(25) =  50
Error range = [ -68.60609 .. 138.0597 ]
================================================================
DocQuality  vs.  files_count_total
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -0.5504057808 2.143248e-01 -2.568092 0.01022601
x1           0.0001326518 6.079125e-05  2.182087 0.02910308
R2log =  0.8923422
Excluded as outliers:   ( 0 / 15 )
MMRE =  35.72839
Pred(25) =  73.33333
Error range = [ -40.87004 .. 310.0619 ]
================================================================
DocQuality  vs.  avg_files_added_per_year
               Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.010502834 0.2800142132 -3.608756 0.0003076694
x1           0.001912882 0.0005337721  3.583705 0.0003387547
R2log =  0.8355119
Excluded as outliers:  Eclipse Spring Framework Hibernate Log4J
( 4 / 15 )
MMRE =  39.36985
Pred(25) =  60
Error range = [ -26.66932 .. 203.0511 ]
================================================================
DocQuality  vs.  avg_files_rem_per_year
              Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -0.520109781 0.1973799444 -2.635069 0.008412017
x1           0.001303054 0.0006290435  2.071485 0.038313495
R2log =  0.8558501
Excluded as outliers:  Spring Framework Eclipse Struts  ( 3 / 15
)
MMRE =  38.28485
Pred(25) =  66.66667
Error range = [ -28.9158 .. 310.9139 ]
================================================================
DocQuality  vs.  loc_total ,  avg_files_rem_per_year
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.418152e+00 4.175500e-01 -3.396365 0.0006828728
x1           3.506074e-06 1.466122e-06  2.391393 0.0167845618
x2           2.327952e-03 6.778834e-04  3.434148 0.0005944194
R2log =  0.838631
```

```
Excluded as outliers:  Spring Framework Eclipse Hibernate Log4J
( 4 / 15 )
MMRE =  30.21146
Pred(25) =  66.66667
Error range = [ -38.0504 .. 117.1942 ]
================================================================
DocQuality  vs.  files_count_total ,  avg_major_release_per_year
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.7664435378 0.4980051773 -3.547039 0.0003895877
x1           0.0004586585 0.0001403948  3.266919 0.0010872465
x2           1.3576051873 0.6072020353  2.235838 0.0253624060
R2log =  0.914498
Excluded as outliers:  Spring Framework PMD Saxon Log4J  ( 4 /
15 )
MMRE =  29.59114
Pred(25) =  66.66667
Error range = [ -69.09649 .. 96.3461 ]
================================================================
DocQuality  vs.  files_count_total ,
avg_number_of_revisions_per_file
                Estimate   Std. Error    z value    Pr(>|z|)
(Intercept)  0.170199021 0.5176067638  0.3288192 0.742292381
x1           0.000361532 0.0001323872  2.7308677 0.006316782
x2          -0.152779532 0.0616389406 -2.4786203 0.013189161
R2log =  0.8608512
Excluded as outliers:  Eclipse Hibernate Spring Framework PMD  (
4 / 15 )
MMRE =  29.5962
Pred(25) =  53.33333
Error range = [ -26.45458 .. 95.9311 ]
================================================================
DocQuality  vs.  files_count_total ,  number_of_commits ,
avg_loc_del_per_year
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -5.549910e-01 2.862640e-01 -1.938738 0.052533219
x1           4.161277e-04 1.281786e-04  3.246467 0.001168468
x2          -1.480103e-04 7.499386e-05 -1.973632 0.048423635
x3          -8.319345e-06 3.559567e-06 -2.337179 0.019429886
R2log =  0.8603327
Excluded as outliers:  Spring Framework Eclipse  ( 2 / 15 )
MMRE =  33.39720
Pred(25) =  73.33333
Error range = [ -34.94137 .. 297.0179 ]
================================================================
DocQuality  vs.  files_count_total ,  avg_loc_del_per_year ,

avg_number_of_revisions_per_file
                Estimate   Std. Error    z value    Pr(>|z|)
(Intercept)  3.121962e-01 4.977493e-01  0.6272158 0.530517813
x1           3.119420e-04 1.147533e-04  2.7183711 0.006560421
x2          -8.265360e-06 3.370102e-06 -2.4525548 0.014184579
x3          -1.500173e-01 5.999843e-02 -2.5003530 0.012406960
R2log =  0.8673331
Excluded as outliers:  Spring Framework Eclipse  ( 2 / 15 )
MMRE =  22.96958
Pred(25) =  73.33333
Error range = [ -28.16463 .. 121.7120 ]
================================================================
```

```
DocQuality  vs.  loc_total ,  avg_loc_del_per_year ,
avg_major_release_per_year ,

avg_files_rem_per_year
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.706635e+00 5.384801e-01 -3.169355 0.0015277773
x1           2.724761e-06 9.755779e-07  2.792972 0.0052226286
x2          -5.430967e-06 2.709636e-06 -2.004316 0.0450362510
x3           1.194318e+00 5.960473e-01  2.003731 0.0450989043
x4           2.855538e-03 8.043877e-04  3.549952 0.0003853015
R2log =  0.916607
Excluded as outliers:  Spring Framework PMD  ( 2 / 15 )
MMRE =  21.48755
Pred(25) =  73.33333
Error range = [ -38.5442 .. 87.9971 ]
================================================================
```

Trustworthiness with respect to non Open Source products

```
================================================================
CssCompetitors  vs.  McCabe
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.7262621  0.6202630 -1.170894 0.24164150
x1           0.7263045  0.3141709  2.311813 0.02078797
R2log =  0.8945558
Excluded as outliers:  Xerces  ( 1 / 17 )
MMRE =  10.54384
Pred(25) =  94.11765
Error range = [ -20.37711 .. 39.18234 ]
================================================================
```

Trustworthiness

```
================================================================
Trustworthiness  vs.  Num. classes
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  1.0608998429 0.2438356102  4.350881 1.355913e-05
x1          -0.0003682820 0.0001778050 -2.071269 3.833363e-02
R2log =  0.8939507
```

```
Excluded as outliers:  Hibernate HttpUnit  ( 2 / 18 )
MMRE =  17.47932
Pred(25) =  83.33333
Error range = [ -44.54244 .. 68.36605 ]
================================================================
Trustworthiness  vs.  Num. methods
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  1.186386e+00 2.443949e-01  4.854380 1.207638e-06
x1          -4.542541e-05 1.955227e-05 -2.323281 2.016408e-02
R2log =  0.8540174
Excluded as outliers:  Hibernate HttpUnit Eclipse Saxon  ( 4 /
18 )
MMRE =  17.70697
Pred(25) =  83.33333
Error range = [ -48.04843 .. 72.17629 ]
================================================================
Trustworthiness  vs.  Num. public methods
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  1.8316962419 4.406502e-01  4.156803 3.227318e-05
x1          -0.0001149826 3.825743e-05 -3.005499 2.651459e-03
R2log =  0.9108067
Excluded as outliers:  Hibernate Saxon HttpUnit Log4J Weka  ( 5
/ 17 )
MMRE =  25.20285
Pred(25) =  76.47059
Error range = [ -86.89772 .. 91.48193 ]
================================================================
Trustworthiness  vs.  CBO ,  McCabe
             Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -1.618633  1.0829440 -1.494660 0.135003108
x1          -0.258561  0.1070114 -2.416201 0.015683401
x2           1.559883  0.6003321  2.598367 0.009366838
R2log =  0.8521789
Excluded as outliers:  Eclipse Xerces Hibernate PMD  ( 4 / 18 )
MMRE =  22.89241
Pred(25) =  72.22222
Error range = [ -99.99965 .. 63.35738 ]
================================================================
Trustworthiness  vs.  Comment lines ,  McCabe
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -1.588621e+00 1.184741e+00 -1.340902 0.17995243
x1          -9.162756e-06 3.859641e-06 -2.373992 0.01759694
x2           1.244082e+00 6.047269e-01  2.057263 0.03966092
R2log =  0.8448486
Excluded as outliers:  Xerces Ant Eclipse Hibernate PMD  ( 5 /
18 )
MMRE =  18.98965
Pred(25) =  72.22222
Error range = [ -41.52189 .. 51.91282 ]
================================================================
Trustworthiness  vs.  eLOC per class ,  Num. interfaces per
class
               Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.21290835 0.65647941 -0.3243184 0.74569699
x1           0.02834693 0.01298528  2.1830053 0.02903542
x2          -2.76562686 1.03193673 -2.6800353 0.00736144
R2log =  0.9100124
Excluded as outliers:  JFreeChart Struts Xerces Spring Framework
( 4 / 18 )
MMRE =  19.31615
```

```
Pred(25) =  72.22222
Error range = [ -66.09687 .. 39.8693 ]
================================================================
Trustworthiness  vs.  eLOC per class ,  Num. packages
               Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  1.632318935 0.4475924585  3.646887 0.000265437
x1          -0.011585483 0.0054908635 -2.109956 0.034862109
x2          -0.001314071 0.0006522147 -2.014783 0.043927402
R2log =  0.9004286
Excluded as outliers:  HttpUnit  ( 1 / 18 )
MMRE =  13.48572
Pred(25) =  94.44444
Error range = [ -18.35019 .. 74.0536 ]
================================================================
Trustworthiness  vs.  eLOC ,  Num. attributes per class
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  1.641993e+00 4.176497e-01  3.931508 8.441468e-05
x1          -4.608735e-06 2.272133e-06 -2.028374 4.252209e-02
x2          -7.085604e-02 3.447366e-02 -2.055367 3.984355e-02
R2log =  0.864489
Excluded as outliers:  Xalan HttpUnit Eclipse  ( 3 / 18 )
MMRE =  18.30190
Pred(25) =  83.33333
Error range = [ -68.91717 .. 83.19392 ]
================================================================
Trustworthiness  vs.  McCabe ,  Num. classes
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.3428163610 1.1946285338 -1.961125 0.049864396
x1           1.7569920711 0.6358366558  2.763276 0.005722438
x2          -0.0006916421 0.0002721924 -2.541005 0.011053446
R2log =  0.8540814
Excluded as outliers:  Hibernate Xerces Eclipse PMD  ( 4 / 18 )
MMRE =  23.11587
Pred(25) =  66.66667
Error range = [ -91.78244 .. 67.7372 ]
================================================================
Trustworthiness  vs.  McCabe ,  Num. methods
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.173007e+00 1.140033e+00 -1.906091 0.056638371
x1           1.650310e+00 5.909619e-01  2.792582 0.005228925
x2          -5.897458e-05 2.127650e-05 -2.771818 0.005574412
R2log =  0.8575264
Excluded as outliers:  Hibernate Xerces PMD Eclipse  ( 4 / 18 )
MMRE =  20.57024
Pred(25) =  77.77778
Error range = [ -83.98747 .. 63.97956 ]
================================================================
Trustworthiness  vs.  McCabe ,  Num. public methods
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -2.358158e+00 1.1815106308 -1.995884 0.045946602
x1           1.718108e+00 0.6006091748  2.860609 0.004228280
x2          -6.732879e-05 0.0000251374 -2.678431 0.007396804
R2log =  0.8580578
Excluded as outliers:  Hibernate Xerces PMD Eclipse  ( 4 / 17 )
MMRE =  21.45107
Pred(25) =  82.35294
Error range = [ -84.7867 .. 65.1984 ]
================================================================
Trustworthiness  vs.  McCabe ,  RFC
                 Estimate   Std. Error   z value    Pr(>|z|)
```

```
(Intercept) -2.180365e+00 1.142897e+00 -1.907754 0.056423067
x1           1.646798e+00 5.918837e-01  2.782301 0.005397498
x2          -3.092225e-05 1.127234e-05 -2.743197 0.006084411
R2log =  0.856996
Excluded as outliers:  Hibernate Xerces PMD Eclipse  ( 4 / 18 )
MMRE =  20.20410
Pred(25) =  77.77778
Error range = [ -82.93462 .. 64.02911 ]
================================================================
Trustworthiness  vs.  CBO ,  Comment lines per class ,  Num.
interfaces per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  1.58742120 0.411172485  3.860718 0.0001130541
x1          -0.26585967 0.122101202 -2.177371 0.0294528638
x2           0.01632332 0.007812906  2.089277 0.0366828377
x3          -1.43930560 0.541748077 -2.656780 0.0078890844
R2log =  0.8576064
Excluded as outliers:  Eclipse Hibernate Saxon  ( 3 / 18 )
MMRE =  18.78701
Pred(25) =  77.77778
Error range = [ -99.99963 .. 35.7058 ]
================================================================
Trustworthiness  vs.  Comment lines ,  McCabe ,  Num. attributes
per class
                 Estimate   Std. Error    z value   Pr(>|z|)
(Intercept)  1.091825e-01 8.107740e-01  0.1346645 0.89287711
x1          -6.271132e-06 2.476323e-06 -2.5324374 0.01132726
x2           8.797515e-01 4.164251e-01  2.1126284 0.03463259
x3          -9.077953e-02 3.719986e-02 -2.4403189 0.01467430
R2log =  0.9156736
Excluded as outliers:  Xalan HttpUnit PMD Xerces  ( 4 / 18 )
MMRE =  19.14223
Pred(25) =  77.77778
Error range = [ -75.85427 .. 71.77117 ]
================================================================
Trustworthiness  vs.  avg_number_of_revisions_per_file
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  1.5178204 0.42895087  3.538448 0.0004024865
x1          -0.1472291 0.06109325 -2.409908 0.0159565406
R2log =  0.8615347
Excluded as outliers:  Eclipse Findbugs Hibernate  ( 3 / 15 )
MMRE =  11.98052
Pred(25) =  86.66667
Error range = [ -38.54624 .. 29.83004 ]
```

Appendix G: Analysis of C++ products

Here we report the detailed data on for java projects the correlations between subjective and objective data.

Reliability

```
================================================================
Reliability  vs.  Num. attributes per class ,  Num. attribute
per method
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -1.51023767 0.300354006 -5.028192 4.951255e-07
x1          -0.02444745 0.009473188 -2.580699 9.860053e-03
x2           1.53281304 0.256598128  5.973594 2.320830e-09
R2log =  0.9490417
Excluded as outliers:  linux perl sqlite openssl  ( 4 / 13 )
MMRE =  23.44636
Pred(25) =  69.23077
Error range = [ -45.0891 .. 108.1098 ]
================================================================
```

Usability

```
================================================================
Usability  vs.  LCOM
               Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -0.38713612 0.15332815 -2.524886 0.011573576
x1           0.07920131 0.02444924  3.239418 0.001197741
R2log =  0.9493277
Excluded as outliers:  linux glibc gdb  ( 3 / 13 )
MMRE =  14.46090
Pred(25) =  76.92308
Error range = [ -30.74551 .. 71.75337 ]
================================================================
Usability  vs.  Num. attributes per class
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.2699555 0.13598832 -1.985137 0.04712922
x1           0.0254235 0.01118158  2.273696 0.02298430
R2log =  0.9477184
Excluded as outliers:  linux gdb perl sqlite  ( 4 / 13 )
MMRE =  17.65493
Pred(25) =  84.61538
Error range = [ -23.37585 .. 74.628 ]
================================================================
Usability  vs.  Num. methods
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  4.021547e-01 1.300177e-01  3.093077 0.001980928
x1          -1.999896e-05 7.147359e-06 -2.798090 0.005140573
R2log =  0.952941
Excluded as outliers:  linux ldap-src subversion  ( 3 / 13 )
MMRE =  22.73113
Pred(25) =  84.61538
```

```
Error range = [ -99.63284 .. 110.6074 ]
================================================================
Usability  vs.  Num. public methods
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  3.922633e-01 1.282449e-01  3.058705 0.002222956
x1          -1.973909e-05 7.151699e-06 -2.760057 0.005779136
R2log =  0.9527443
Excluded as outliers:  linux ldap-src subversion  ( 3 / 13 )
MMRE =  22.73732
Pred(25) =  84.61538
Error range = [ -99.60504 .. 109.8661 ]
================================================================
Usability  vs.  LCOM ,  McCabe
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.3235288 0.17879399 -1.809506 0.0703724202
x1           0.2667431 0.07667804  3.478742 0.0005037728
x2          -0.4111575 0.13336043 -3.083055 0.0020488755
R2log =  0.951955
Excluded as outliers:  glibc perl sqlite linux  ( 4 / 13 )
MMRE =  17.44336
Pred(25) =  76.92308
Error range = [ -58.85103 .. 37.42888 ]
================================================================
```

Portability

```
================================================================
Portability  vs.  Num. methods per class
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.1136125 0.25915091 -4.297158 1.730018e-05
x1           0.0637316 0.01145698  5.562689 2.656487e-08
R2log =  0.9559484
Excluded as outliers:  glibc openssl libxml2  ( 3 / 13 )
MMRE =  16.31748
Pred(25) =  76.92308
Error range = [ -51.96173 .. 51.02023 ]
================================================================
```

How well are functional requirements satisfied

```
================================================================
Functionality  vs.  ACC
              Estimate Std. Error   z value    Pr(>|z|)
(Intercept) -0.2689996 0.20511091 -1.311484 0.189694418
x1           0.1232661 0.04645936  2.653203 0.007973198
R2log =  0.9477128
Excluded as outliers:  linux perl gdb posgresql  ( 4 / 13 )
MMRE =  16.80887
Pred(25) =  76.92308
Error range = [ -21.79552 .. 77.48773 ]
================================================================
Functionality  vs.  CBO
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.2504962 0.20755474 -1.206892 0.22747363
x1           0.1214927 0.04818648  2.521303 0.01169211
R2log =  0.9469887
Excluded as outliers:  linux perl gdb posgresql  ( 4 / 13 )
MMRE =  17.03314
```

```
Pred(25) =  76.92308
Error range = [ -21.28353 .. 78.48593 ]
================================================================
Functionality  vs.  LCOM
             Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -0.4697571 0.21130989 -2.223072 0.02621093
x1           0.1048625 0.02919295  3.592048 0.00032809
R2log =  0.9450293
Excluded as outliers:  linux glibc subversion openssl ( 4 / 13 )
MMRE =  18.41709
Pred(25) =  69.23077
Error range = [ -28.51859 .. 57.13447 ]
================================================================
Functionality  vs.  eLOC per method
              Estimate  Std. Error    z value    Pr(>|z|)
(Intercept) -0.15824472 0.160398990 -0.9865693 0.323853837
x1           0.01882910 0.006427766  2.9293379 0.003396850
R2log =  0.94873
Excluded as outliers:  linux perl  ( 2 / 13 )
MMRE =  16.38215
Pred(25) =  76.92308
Error range = [ -27.65076 .. 64.11788 ]
================================================================
Functionality  vs.  eLOC per class
               Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -0.075716271 0.1686317253 -0.4490037 0.65342898
x1           0.001558958 0.0005701948  2.7340802 0.00625548
R2log =  0.947368
Excluded as outliers:  libxml2 linux gdb sqlite  ( 4 / 13 )
MMRE =  26.96493
Pred(25) =  61.53846
Error range = [ -10.09838 .. 89.42949 ]
================================================================
Functionality  vs.  McCabe
              Estimate Std. Error    z value    Pr(>|z|)
(Intercept) -0.08322601 0.16954678 -0.4908734 0.623515966
x1           0.16735464 0.06351721  2.6347919 0.008418887
R2log =  0.9456217
Excluded as outliers:  linux gdb sqlite  ( 3 / 13 )
MMRE =  20.59543
Pred(25) =  84.61538
Error range = [ -21.57827 .. 73.84595 ]
================================================================
Functionality  vs.  Num. methods
                Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -2.206761e-02 1.662996e-01 -0.1326979 0.89443227
x1           3.393236e-05 1.424471e-05  2.3821027 0.01721410
R2log =  0.9457637
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  23.38308
Pred(25) =  61.53846
Error range = [ -15.52365 .. 92.69175 ]
================================================================
Functionality  vs.  Num. attribute per method
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.6351755  0.2764945 -2.297245 0.0216047931
x1           0.7121362  0.2073896  3.433808 0.0005951656
R2log =  0.9498425
Excluded as outliers:  linux perl sqlite gdb  ( 4 / 13 )
MMRE =  18.71853
```

```
Pred(25) =  69.23077
Error range = [ -26.55506 .. 79.08917 ]
================================================================
Functionality  vs.  Num. public methods
                 Estimate   Std. Error     z value   Pr(>|z|)
(Intercept) -1.444779e-02 1.676456e-01 -0.08618055 0.93132289
x1           3.417787e-05 1.481457e-05  2.30704425 0.02105235
R2log =  0.9453948
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  23.64714
Pred(25) =  61.53846
Error range = [ -15.21326 .. 93.50963 ]
================================================================
Functionality  vs.  LCOM ,  eLOC
                 Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.877843e-01 2.853327e-01 -1.709528 0.087353173
x1           8.639986e-02 2.909423e-02  2.969656 0.002981334
x2           7.961439e-07 3.643925e-07  2.184852 0.028899674
R2log =  0.9422352
Excluded as outliers:  linux gdb subversion  ( 3 / 13 )
MMRE =  20.96989
Pred(25) =  69.23077
Error range = [ -13.95603 .. 72.81018 ]
================================================================
Functionality  vs.  LCOM ,  Num. methods
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -4.293522e-01 3.111793e-01 -1.379758 0.16766111
x1           6.975180e-02 3.277884e-02  2.127952 0.03334107
x2           4.934959e-05 2.134364e-05  2.312144 0.02076973
R2log =  0.9482143
Excluded as outliers:  linux gdb cygwin posgresql  ( 4 / 13 )
MMRE =  25.01579
Pred(25) =  61.53846
Error range = [ -6.535332 .. 92.3553 ]
================================================================
Functionality  vs.  eLOC per method ,  Num. attributes per class
               Estimate Std. Error    z value     Pr(>|z|)
(Intercept) -0.15147724 0.16091658 -0.9413401 0.3465305817
x1           0.02913454 0.00816428  3.5685382 0.0003589786
x2          -0.01658191 0.00802182 -2.0671013 0.0387246116
R2log =  0.9522783
Excluded as outliers:  linux perl  ( 2 / 13 )
MMRE =  15.36017
Pred(25) =  76.92308
Error range = [ -41.72362 .. 63.00066 ]
================================================================
Functionality  vs.  McCabe ,  Num. attributes per class
               Estimate Std. Error    z value     Pr(>|z|)
(Intercept) -0.10281958 0.16414106 -0.6264099 0.5310461481
x1           0.33580539 0.10141678  3.3111423 0.0009291595
x2          -0.03683576 0.01254156 -2.9370945 0.0033130310
R2log =  0.9492425
Excluded as outliers:  linux glibc  ( 2 / 13 )
MMRE =  17.38857
Pred(25) =  76.92308
Error range = [ -52.28476 .. 69.44363 ]
================================================================
Functionality  vs.  Num. attributes per class ,  Num. attribute
per method
               Estimate  Std. Error   z value   Pr(>|z|)
```

```
(Intercept) -0.95941354 0.413152993 -2.322175 0.02022352
x1          -0.02048793 0.009062966 -2.260620 0.02378278
x2           1.08703780 0.327806655  3.316094 0.00091285
R2log =  0.9515106
Excluded as outliers:  linux perl sqlite cygwin  ( 4 / 13 )
MMRE =  17.68123
Pred(25) =  76.92308
Error range = [ -33.05436 .. 63.06829 ]
================================================================
```

Interoperability

```
================================================================
Interoperability  vs.  LCOM
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.8282344 0.21208493 -3.905202 9.414684e-05
x1           0.1350092 0.02623372  5.146401 2.655314e-07
R2log =  0.9459
Excluded as outliers:  openssl glibc subversion  ( 3 / 13 )
MMRE =  15.54680
Pred(25) =  84.61538
Error range = [ -42.98863 .. 16.45856 ]
================================================================
Interoperability  vs.  Num. methods
                 Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -1.578726e-01 1.625064e-01 -0.9714855 0.33130655
x1           3.934332e-05 1.563712e-05  2.5160208 0.01186882
R2log =  0.9473539
Excluded as outliers:  linux cygwin gdb openssl  ( 4 / 13 )
MMRE =  23.97699
Pred(25) =  69.23077
Error range = [ -16.00141 .. 100.3220 ]
================================================================
Interoperability  vs.  Num. public methods
                 Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -1.574113e-01 1.621623e-01 -0.9707023 0.33169655
x1           3.937568e-05 1.562725e-05  2.5196809 0.01174613
R2log =  0.9473764
Excluded as outliers:  linux cygwin gdb openssl  ( 4 / 13 )
MMRE =  24.14464
Pred(25) =  69.23077
Error range = [ -18.65682 .. 99.77417 ]
================================================================
```

Security

```
================================================================
Security  vs.  ACC
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.4946254 0.14246159 -3.471991 5.166134e-04
x1           0.1225129 0.02003996  6.113429 9.751256e-10
R2log =  0.949365
Excluded as outliers:  perl openssl  ( 2 / 13 )
MMRE =  12.14888
Pred(25) =  84.61538
Error range = [ -26.29964 .. 18.11794 ]
================================================================
Security  vs.  McCabe
```

```
              Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.1410507 0.20875037 -0.6756907 0.49923702
x1           0.1425878 0.06652634  2.1433290 0.03208669
R2log =  0.946116
Excluded as outliers:  cygwin linux openssl sqlite  ( 4 / 13 )
MMRE =  18.59930
Pred(25) =  69.23077
Error range = [ -26.34763 .. 46.87301 ]
================================================================
Security  vs.  Num. methods
                 Estimate   Std. Error    z value   Pr(>|z|)
(Intercept) -1.000220e-01 1.619874e-01 -0.6174678 0.53692624
x1           3.820375e-05 1.557583e-05  2.4527588 0.01417654
R2log =  0.9471986
Excluded as outliers:  linux cygwin gdb openssl  ( 4 / 13 )
MMRE =  22.99840
Pred(25) =  61.53846
Error range = [ -16.40444 .. 97.19032 ]
================================================================
Security  vs.  Num. attribute per method
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.0499715  0.2895565 -3.626137 2.876924e-04
x1           0.8783335  0.2062723  4.258126 2.061477e-05
R2log =  0.9492961
Excluded as outliers:  sqlite perl linux openssl  ( 4 / 13 )
MMRE =  16.92961
Pred(25) =  69.23077
Error range = [ -33.71983 .. 48.96089 ]
================================================================
Security  vs.  Num. public methods
                 Estimate   Std. Error   z value   Pr(>|z|)
(Intercept) -9.903799e-02 1.616395e-01 -0.612709 0.54006879
x1           3.817564e-05 1.556492e-05  2.452672 0.01417997
R2log =  0.9471987
Excluded as outliers:  linux cygwin gdb openssl  ( 4 / 13 )
MMRE =  23.15043
Pred(25) =  61.53846
Error range = [ -18.91692 .. 96.61892 ]
================================================================
```

Speed

```
================================================================
Speed  vs.  LCOM
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.8776853 0.20230724 -4.338378 1.435382e-05
x1           0.1351164 0.03524085  3.834085 1.260328e-04
R2log =  0.9491585
Excluded as outliers:  linux glibc openssl perl  ( 4 / 13 )
MMRE =  17.71915
Pred(25) =  69.23077
Error range = [ -44.71081 .. 50.90545 ]
================================================================
Speed  vs.  eLOC per method
                Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -0.79642571 0.173572193 -4.588441 4.465692e-06
x1           0.02781879 0.006757576  4.116682 3.843656e-05
R2log =  0.9519118
Excluded as outliers:  linux openssl glibc perl  ( 4 / 13 )
```

```
MMRE =  13.69598
Pred(25) =  84.61538
Error range = [ -40.7392 .. 16.77493 ]
================================================================
Speed  vs.  McCabe
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.6921361 0.16569838 -4.177084 2.952699e-05
x1           0.1931513 0.05239635  3.686350 2.274931e-04
R2log =  0.9498962
Excluded as outliers:  linux openssl glibc  ( 3 / 13 )
MMRE =  15.29002
Pred(25) =  76.92308
Error range = [ -39.96996 .. 33.50838 ]
================================================================
Speed  vs.  ACC ,  RFC
                 Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -0.860165517 0.192798883 -4.461465 8.140114e-06
x1           0.074291962 0.029095916  2.553347 1.066932e-02
x2           0.003429174 0.001400897  2.447841 1.437149e-02
R2log =  0.9513538
Excluded as outliers:  linux openssl glibc perl  ( 4 / 13 )
MMRE =  13.93381
Pred(25) =  76.92308
Error range = [ -41.90643 .. 10.69179 ]
================================================================
Speed  vs.  CBO ,  RFC
                 Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -0.859270402 0.192564299 -4.462252 8.110282e-06
x1           0.074258204 0.029081133  2.553484 1.066512e-02
x2           0.003424515 0.001401322  2.443774 1.453454e-02
R2log =  0.9513547
Excluded as outliers:  linux openssl glibc perl  ( 4 / 13 )
MMRE =  14.05334
Pred(25) =  76.92308
Error range = [ -43.4835 .. 10.75888 ]
================================================================
Speed  vs.  eLOC ,  Num. attribute per method
                  Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.563039e+00 3.022566e-01 -5.171231 2.325571e-07
x1           1.021710e-06 3.321474e-07  3.076073 2.097467e-03
x2           7.590314e-01 1.657819e-01  4.578494 4.683358e-06
R2log =  0.9521471
Excluded as outliers:  openssl linux gdb perl  ( 4 / 13 )
MMRE =  19.78608
Pred(25) =  61.53846
Error range = [ -36.37149 .. 64.60323 ]
================================================================
```

Utility of the product developer community

```
================================================================
Community  vs.  ACC
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.61265952 0.13653385 -4.487235 7.215339e-06
x1           0.08204031 0.01866126  4.396290 1.101167e-05
R2log =  0.951876
Excluded as outliers:  perl  ( 1 / 13 )
MMRE =  10.78645
Pred(25) =  92.3077
```

```
Error range = [ -29.54724 .. 16.67387 ]
================================================================
Community  vs.  CBO
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.5972677 0.13472153 -4.433350 9.278006e-06
x1           0.0804353 0.01852726  4.341457 1.415412e-05
R2log =  0.9514845
Excluded as outliers:  perl  ( 1 / 13 )
MMRE =  10.98834
Pred(25) =  92.3077
Error range = [ -29.08407 .. 16.76074 ]
================================================================
Community  vs.  LCOM
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.0004855 0.21354228 -4.685187 2.797043e-06
x1           0.1250674 0.02573718  4.859406 1.177385e-06
R2log =  0.9505509
Excluded as outliers:  glibc openssl subversion  ( 3 / 13 )
MMRE =  13.54013
Pred(25) =  84.61538
Error range = [ -36.9326 .. 15.53158 ]
================================================================
Community  vs.  eLOC per class
               Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -0.398719764 0.1830548126 -2.178144 0.029395309
x1           0.001367252 0.0004695217  2.912010 0.003591107
R2log =  0.9568126
Excluded as outliers:  libxml2 sqlite gdb cygwin  ( 4 / 13 )
MMRE =  24.54869
Pred(25) =  69.23077
Error range = [ -10.06239 .. 121.9292 ]
================================================================
Community  vs.  Num. methods
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -5.904692e-01 1.789469e-01 -3.299690 0.0009679158
x1           4.561845e-05 1.489235e-05  3.063215 0.0021897297
R2log =  0.9527587
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  22.31116
Pred(25) =  76.92308
Error range = [ -24.90588 .. 99.2933 ]
================================================================
Community  vs.  Num. attribute per method
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.0062447  0.2857989 -3.520813 0.0004302255
x1           0.6290865  0.2035320  3.090848 0.0019958597
R2log =  0.9477246
Excluded as outliers:  perl sqlite linux  ( 3 / 13 )
MMRE =  15.34681
Pred(25) =  84.61538
Error range = [ -34.37621 .. 49.48352 ]
================================================================
Community  vs.  Num. public methods
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -5.966396e-01 1.800204e-01 -3.314289 0.000918764
x1           4.757811e-05 1.546137e-05  3.077226 0.002089369
R2log =  0.9528778
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  22.45235
Pred(25) =  76.92308
```

```
Error range = [ -24.71196 .. 101.5722 ]
================================================================
Community  vs.  LCOM ,  Num. classes
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -0.6856567873 2.331556e-01 -2.940769 0.003273986
x1           0.0764484006 2.820048e-02  2.710891 0.006710278
x2           0.0001266623 6.132476e-05  2.065435 0.038881878
R2log =  0.9504459
Excluded as outliers:  linux cygwin  ( 2 / 13 )
MMRE =  16.88189
Pred(25) =  84.61538
Error range = [ -10.49582 .. 98.67139 ]
================================================================
Community  vs.  LCOM ,  RFC
                Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -0.287286211 0.151444177 -1.896978 0.0578308951
x1           0.102913348 0.028434327  3.619335 0.0002953613
x2          -0.004124779 0.001676067 -2.460987 0.0138555574
R2log =  0.9533986
Excluded as outliers:  cygwin perl  ( 2 / 13 )
MMRE =  12.37445
Pred(25) =  84.61538
Error range = [ -9.100908 .. 52.02749 ]
================================================================
Community  vs.  eLOC ,  McCabe
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.811913e-01 1.688362e-01 -2.850048 0.004371261
x1          -4.692812e-07 2.144884e-07 -2.187909 0.028676196
x2           2.353229e-01 7.190807e-02  3.272552 0.001065813
R2log =  0.947836
Excluded as outliers:  linux sqlite  ( 2 / 13 )
MMRE =  20.60234
Pred(25) =  76.92308
Error range = [ -95.40849 .. 59.77883 ]
================================================================
Community  vs.  McCabe ,  Num. methods per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -1.11728356 0.274209075 -4.074568 4.609981e-05
x1           0.29962652 0.086098962  3.480025 5.013674e-04
x2           0.02156189 0.007833908  2.752380 5.916391e-03
R2log =  0.9566052
Excluded as outliers:  sqlite gdb libxml2 posgresql  ( 4 / 13 )
MMRE =  20.76168
Pred(25) =  76.92308
Error range = [ -23.56640 .. 76.0099 ]
================================================================
```

Documentation Quality

```
================================================================
DocQuality  vs.  LCOM
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.1404971 0.21335438 -5.345553 9.014150e-08
x1           0.1542199 0.02593226  5.947030 2.730517e-09
R2log =  0.9480892
Excluded as outliers:  glibc subversion openssl  ( 3 / 13 )
MMRE =  23.70122
Pred(25) =  69.23077
Error range = [ -46.27743 .. 119.9662 ]
```

```
================================================================
DocQuality  vs.  Num. attributes per class
               Estimate  Std. Error   z value   Pr(>|z|)
(Intercept)  0.22116677 0.130340642  1.696837 0.08972756
x1          -0.01723115 0.007226257 -2.384519 0.01710146
R2log =  0.9434227
Excluded as outliers:  linux posgresql cygwin perl  ( 4 / 13 )
MMRE =  32.08148
Pred(25) =  61.53846
Error range = [ -37.52433 .. 193.3229 ]
================================================================
DocQuality  vs.  Num. methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -4.715958e-01 1.736558e-01 -2.715692 0.0066137345
x1           5.406815e-05 1.472338e-05  3.672265 0.0002404099
R2log =  0.9486783
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  37.56731
Pred(25) =  53.84615
Error range = [ -25.53044 .. 137.6285 ]
================================================================
DocQuality  vs.  Num. attribute per method
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.292514  0.2824477 -4.576117 4.736849e-06
x1           1.026637  0.1986796  5.167302 2.374968e-07
R2log =  0.9519507
Excluded as outliers:  perl sqlite gdb  ( 3 / 13 )
MMRE =  30.99149
Pred(25) =  69.23077
Error range = [ -40.33495 .. 187.7889 ]
================================================================
DocQuality  vs.  Num. public methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -4.868995e-01 0.1750802812 -2.781007 0.0054190549
x1           5.717341e-05 0.0000153281  3.729974 0.0001914995
R2log =  0.949231
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  37.65285
Pred(25) =  53.84615
Error range = [ -25.44634 .. 135.8436 ]
================================================================
DocQuality  vs.  eLOC per method ,  Num. attributes per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -0.69849295 0.195143189 -3.579387 3.444014e-04
x1           0.04214317 0.008682181  4.853984 1.210054e-06
x2          -0.02535074 0.008939206 -2.835905 4.569602e-03
R2log =  0.943998
Excluded as outliers:  linux perl subversion  ( 3 / 13 )
MMRE =  18.77071
Pred(25) =  76.92308
Error range = [ -50.86176 .. 57.22592 ]
================================================================
DocQuality  vs.  eLOC per method ,  Num. methods per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -1.44503976 0.289977744 -4.983278 6.251597e-07
x1           0.01873411 0.007456299  2.512522 1.198718e-02
x2           0.05578795 0.011489008  4.855767 1.199214e-06
R2log =  0.9534666
Excluded as outliers:  sqlite glibc libxml2 openssl  ( 4 / 13 )
MMRE =  25.35241
```

```
Pred(25) =  46.15385
Error range = [ -43.15462 .. 64.01202 ]
================================================================
DocQuality  vs.  McCabe ,  Num. methods per class
               Estimate  Std. Error   z value    Pr(>|z|)
(Intercept) -0.60502529 0.290256245 -2.084452 0.037119048
x1           0.19394146 0.073571927  2.636080 0.008387004
x2           0.01900285 0.008307099  2.287543 0.022164123
R2log =  0.9529295
Excluded as outliers:  cygwin sqlite gdb libxml2  ( 4 / 13 )
MMRE =  33.09373
Pred(25) =  46.15385
Error range = [ -8.402078 .. 149.5282 ]
================================================================
```

Trustworthiness with respect to other Open Source products

```
================================================================
OssCompetitors  vs.  ACC
             Estimate Std. Error  z value    Pr(>|z|)
(Intercept) 0.1386089 0.17739221 0.781370 0.434584939
x1          0.0819205 0.03033434 2.700586 0.006921743
R2log =  0.9310557
Excluded as outliers:  linux perl subversion  ( 3 / 13 )
MMRE =  12.38624
Pred(25) =  92.3077
Error range = [ -15.18194 .. 47.22463 ]
================================================================
OssCompetitors  vs.  LCOM
               Estimate Std. Error    z value    Pr(>|z|)
(Intercept) -0.08099502 0.18356583 -0.4412315 0.659045429
x1           0.08684293 0.02823579  3.0756335 0.002100559
R2log =  0.9441202
Excluded as outliers:  linux glibc openssl  ( 3 / 13 )
MMRE =  13.71208
Pred(25) =  84.61538
Error range = [ -25.85867 .. 54.82388 ]
================================================================
OssCompetitors  vs.  eLOC per method
               Estimate  Std. Error    z value    Pr(>|z|)
(Intercept) -0.02596338 0.183890653 -0.1411893 0.887720435
x1           0.02595006 0.007918353  3.2772041 0.001048406
R2log =  0.940268
Excluded as outliers:  linux sqlite perl  ( 3 / 13 )
MMRE =  13.40685
Pred(25) =  84.61538
Error range = [ -23.61703 .. 38.00108 ]
================================================================
OssCompetitors  vs.  Num. methods
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.679869e-01 1.812274e-01 -0.926940 3.539577e-01
```

```
x1           6.562055e-05 1.593747e-05  4.117376 3.832105e-05
R2log =  0.9457584
Excluded as outliers:  linux gdb cygwin perl  ( 4 / 13 )
MMRE =  14.59442
Pred(25) =  76.92308
Error range = [ -18.64501 .. 42.84904 ]
================================================================
OssCompetitors  vs.  Num. attribute per method
             Estimate Std. Error    z value   Pr(>|z|)
(Intercept) 0.01036333  0.2962005 0.03498755 0.97208967
x1          0.45483522  0.2128316 2.13706584 0.03259264
R2log =  0.9269898
Excluded as outliers:  sqlite linux perl subversion  ( 4 / 13 )
MMRE =  16.98973
Pred(25) =  84.61538
Error range = [ -16.05440 .. 74.18972 ]
================================================================
OssCompetitors  vs.  Num. public methods
               Estimate   Std. Error    z value     Pr(>|z|)
(Intercept) -1.479425e-01 1.830022e-01 -0.8084192 4.188493e-01
x1           6.542355e-05 1.657909e-05  3.9461494 7.941808e-05
R2log =  0.944015
Excluded as outliers:  linux gdb cygwin perl  ( 4 / 13 )
MMRE =  14.81329
Pred(25) =  76.92308
Error range = [ -18.04017 .. 42.77925 ]
================================================================
OssCompetitors  vs.  RFC
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.819423729 0.13754190  5.957630 2.559220e-09
x1          -0.003289863 0.00138025 -2.383527 1.714762e-02
R2log =  0.9322374
Excluded as outliers:  linux posgresql subversion cygwin  ( 4 /
13 )
MMRE =  16.57449
Pred(25) =  84.61538
Error range = [ -25.13926 .. 89.81073 ]
================================================================
OssCompetitors  vs.  ACC ,  LCOM
             Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  0.5266014 0.25347752  2.077507 0.037754745
x1           0.1091626 0.04195605  2.601833 0.009272708
x2          -0.1225675 0.05597316 -2.189755 0.028542016
R2log =  0.9350203
Excluded as outliers:  perl linux posgresql subversion  ( 4 / 13
)
MMRE =  14.32073
Pred(25) =  84.61538
Error range = [ -45.54541 .. 41.54445 ]
================================================================
OssCompetitors  vs.  ACC ,  Num. attributes per class
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.44246548 0.16407607  2.696709 0.0070028359
x1           0.14275718 0.03704286  3.853837 0.0001162808
x2          -0.03142094 0.00955415 -3.288722 0.0010064337
R2log =  0.9462454
Excluded as outliers:  linux subversion ldap-src  ( 3 / 13 )
MMRE =  10.5708
Pred(25) =  76.92308
Error range = [ -11.07156 .. 34.98033 ]
```

```
================================================================
OssCompetitors  vs.  ACC ,  Num. methods per class
               Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  0.35956129 0.22042323  1.631231 0.102841502
x1           0.08876927 0.03009009  2.950116 0.003176545
x2          -0.01888204 0.00874729 -2.158616 0.030879991
R2log =  0.9416606
Excluded as outliers:  perl linux  ( 2 / 13 )
MMRE =  12.84397
Pred(25) =  84.61538
Error range = [ -28.60415 .. 53.48325 ]
================================================================
OssCompetitors  vs.  CBO ,  LCOM
              Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.5598569 0.25126418  2.228160 0.02586983
x1           0.1035329 0.04294751  2.410684 0.01592262
x2          -0.1219968 0.05664703 -2.153631 0.03126911
R2log =  0.9336252
Excluded as outliers:  perl linux posgresql subversion  ( 4 / 13
)
MMRE =  14.60993
Pred(25) =  84.61538
Error range = [ -44.68897 .. 43.96367 ]
================================================================
OssCompetitors  vs.  CBO ,  Num. attributes per class
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.46015031 0.16238377  2.833721 0.0046009474
x1           0.14601634 0.03820869  3.821548 0.0001326168
x2          -0.03281393 0.00979372 -3.350507 0.0008066386
R2log =  0.9459533
Excluded as outliers:  linux subversion ldap-src  ( 3 / 13 )
MMRE =  10.95897
Pred(25) =  76.92308
Error range = [ -10.98415 .. 34.72803 ]
================================================================
OssCompetitors  vs.  CBO ,  Num. methods per class
               Estimate Std. Error   z value   Pr(>|z|)
(Intercept)  0.39496998 0.21718994  1.818546 0.06898067
x1           0.08458445 0.03013316  2.807022 0.00500018
x2          -0.01935374 0.00876653 -2.207686 0.02726618
R2log =  0.9408186
Excluded as outliers:  perl linux  ( 2 / 13 )
MMRE =  13.01269
Pred(25) =  84.61538
Error range = [ -28.25762 .. 55.65066 ]
================================================================
OssCompetitors  vs.  LCOM ,  eLOC
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -3.845058e-01 2.482176e-01 -1.549067 0.1213655654
x1           8.364675e-02 2.860610e-02  2.924087 0.0034546770
x2           1.252165e-06 3.623181e-07  3.455983 0.0005482906
R2log =  0.943718
Excluded as outliers:  linux gdb  ( 2 / 13 )
MMRE =  11.57405
Pred(25) =  84.61538
Error range = [ -16.57002 .. 39.9556 ]
================================================================
OssCompetitors  vs.  LCOM ,  Num. methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -7.727836e-01 3.430907e-01 -2.252418 2.429589e-02
```

```
x1           9.428574e-02 3.673456e-02  2.566677 1.026782e-02
x2           9.406765e-05 2.389677e-05  3.936417 8.270722e-05
R2log =  0.9489377
Excluded as outliers:  linux gdb cygwin posgresql  ( 4 / 13 )
MMRE =  14.62395
Pred(25) =  84.61538
Error range = [ -12.21201 .. 51.75187 ]
================================================================
OssCompetitors  vs.  eLOC per method ,  Num. attributes per
class
               Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  0.52183316 0.20240936  2.578108 0.009934297
x1           0.02515103 0.01120142  2.245342 0.024746157
x2          -0.04171287 0.01296373 -3.217660 0.001292408
R2log =  0.9383593
Excluded as outliers:  linux gdb subversion posgresql  ( 4 / 13
)
MMRE =  15.51862
Pred(25) =  76.92308
Error range = [ -56.93294 .. 31.37763 ]
================================================================
OssCompetitors  vs.  eLOC per class ,  eLOC
                Estimate   Std. Error   z value    Pr(>|z|)
(Intercept) -4.950762e-01 3.088323e-01 -1.603058 0.108921758
x1           2.239849e-03 7.025664e-04  3.188095 0.001432133
x2           1.116911e-06 4.059276e-07  2.751503 0.005932249
R2log =  0.9466387
Excluded as outliers:  linux gdb libxml2 sqlite  ( 4 / 13 )
MMRE =  19.26473
Pred(25) =  76.92308
Error range = [ -13.97349 .. 84.44037 ]
================================================================
OssCompetitors  vs.  eLOC ,  Num. methods per class
                Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.121878e+00 4.810261e-01 -2.332261 0.0196869768
x1           1.997001e-06 5.674061e-07  3.519527 0.0004323164
x2           4.485541e-02 1.502577e-02  2.985231 0.0028336419
R2log =  0.9500572
Excluded as outliers:  linux gdb openssl libxml2  ( 4 / 13 )
MMRE =  14.48514
Pred(25) =  84.61538
Error range = [ -23.89715 .. 47.10292 ]
================================================================
OssCompetitors  vs.  McCabe ,  Num. attributes per class
              Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  0.51153704 0.21910237  2.334694 0.019559399
x1           0.19714412 0.10031768  1.965198 0.049391303
x2          -0.04479094 0.01456916 -3.074367 0.002109495
R2log =  0.936697
Excluded as outliers:  linux gdb posgresql subversion  ( 4 / 13
)
MMRE =  15.87538
Pred(25) =  76.92308
Error range = [ -58.46338 .. 34.63136 ]
================================================================
OssCompetitors  vs.  ACC ,  LCOM ,  RFC
                Estimate  Std. Error    z value     Pr(>|z|)
(Intercept)  0.038757382 0.154348340  0.2511033 0.8017342225
x1           0.084924252 0.023802604  3.5678555 0.0003599149
x2           0.060483535 0.026229528  2.3059330 0.0211143760
```

```
x3          -0.003332799 0.001349377 -2.4698807 0.0135158110
R2log =  0.944446
Excluded as outliers:   ( 0 / 13 )
MMRE =  9.961673
Pred(25) =  92.3077
Error range = [ -14.25141 .. 55.02372 ]
================================================================
```

Trustworthiness with respect to non Open Source products

```
================================================================
CssCompetitors  vs.  ACC
              Estimate Std. Error  z value    Pr(>|z|)
(Intercept) 0.53722334 0.18056441 2.975245 0.002927548
x1          0.07268813 0.03205261 2.267776 0.023342879
R2log =  0.9326331
Excluded as outliers:  linux perl  ( 2 / 13 )
MMRE =  11.80516
Pred(25) =  84.61538
Error range = [ -10.72870 .. 38.99635 ]
================================================================
CssCompetitors  vs.  eLOC per method
              Estimate Std. Error  z value   Pr(>|z|)
(Intercept) 0.43049430 0.22419318 1.920194 0.05483345
x1          0.02916432 0.01189664 2.451475 0.01422719
R2log =  0.9345065
Excluded as outliers:  linux sqlite posgresql  ( 3 / 13 )
MMRE =  13.30016
Pred(25) =  76.92308
Error range = [ -15.56892 .. 35.27322 ]
================================================================
CssCompetitors  vs.  Num. methods
               Estimate   Std. Error  z value    Pr(>|z|)
(Intercept) 3.950109e-01 1.970246e-01 2.004881 0.044975753
x1          6.262022e-05 2.145549e-05 2.918611 0.003515944
R2log =  0.935521
Excluded as outliers:  linux cygwin gdb posgresql  ( 4 / 13 )
MMRE =  14.22091
Pred(25) =  76.92308
Error range = [ -8.552595 .. 42.52655 ]
================================================================
CssCompetitors  vs.  Num. methods per class
              Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  1.22640911 0.181819582  6.745198 1.528183e-11
x1          -0.01835388 0.009316272 -1.970088 4.882829e-02
R2log =  0.9311549
Excluded as outliers:  linux perl  ( 2 / 13 )
MMRE =  14.50143
Pred(25) =  84.61538
Error range = [ -22.16676 .. 65.08489 ]
================================================================
CssCompetitors  vs.  Num. attribute per method
             Estimate Std. Error   z value   Pr(>|z|)
(Intercept) 0.2408754  0.3068030 0.7851143 0.43238657
x1          0.5279231  0.2233258 2.3639146 0.01808298
R2log =  0.9326764
Excluded as outliers:  sqlite linux perl  ( 3 / 13 )
MMRE =  13.40067
Pred(25) =  92.3077
```

```
Error range = [ -13.25562 .. 54.75226 ]
================================================================
CssCompetitors  vs.  Num. public methods
                Estimate   Std. Error  z value    Pr(>|z|)
(Intercept) 0.3804268597 0.2056252850 1.850098 0.064299474
x1          0.0000664253 0.0000232928 2.851752 0.004347897
R2log =  0.934901
Excluded as outliers:  linux cygwin gdb posgresql  ( 4 / 13 )
MMRE =  14.65793
Pred(25) =  69.23077
Error range = [ -8.355594 .. 43.49516 ]
================================================================
CssCompetitors  vs.  ACC ,  LCOM
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.7169864 0.19559360  3.665695 0.0002466682
x1           0.1329127 0.04136043  3.213523 0.0013111727
x2          -0.1051667 0.04715965 -2.230015 0.0257464489
R2log =  0.9381416
Excluded as outliers:  perl linux  ( 2 / 13 )
MMRE =  11.22235
Pred(25) =  76.92308
Error range = [ -36.11098 .. 32.8361 ]
================================================================
CssCompetitors  vs.  ACC ,  Num. attributes per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.72397163 0.163229462  4.435300 9.194418e-06
x1           0.09662670 0.035255696  2.740740 6.130099e-03
x2          -0.01865060 0.008544485 -2.182765 2.905311e-02
R2log =  0.9377748
Excluded as outliers:  linux  ( 1 / 13 )
MMRE =  10.20937
Pred(25) =  84.61538
Error range = [ -12.18411 .. 37.36086 ]
================================================================
CssCompetitors  vs.  ACC ,  Num. methods per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.84932423 0.235593317  3.605044 0.0003121003
x1           0.07718665 0.032058483  2.407683 0.0160541275
x2          -0.01954108 0.009176351 -2.129504 0.0332125471
R2log =  0.9376741
Excluded as outliers:  perl linux  ( 2 / 13 )
MMRE =  10.74531
Pred(25) =  92.3077
Error range = [ -20.09363 .. 47.8937 ]
================================================================
CssCompetitors  vs.  ACC ,  RFC
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.769213468 0.170627568  4.508143 6.539751e-06
x1           0.074170885 0.031231570  2.374869 1.755518e-02
x2          -0.002835036 0.001334122 -2.125020 3.358498e-02
R2log =  0.9374866
Excluded as outliers:  linux  ( 1 / 13 )
MMRE =  10.73369
Pred(25) =  84.61538
Error range = [ -8.457621 .. 49.71875 ]
================================================================
CssCompetitors  vs.  CBO ,  LCOM
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.7412328 0.19409335  3.818950 0.0001340211
x1           0.1292410 0.04213871  3.067037 0.0021619188
```

```
x2          -0.1044355 0.04797735 -2.176766 0.0294980418
R2log =  0.9371468
Excluded as outliers:  perl linux  ( 2 / 13 )
MMRE =  11.49438
Pred(25) =  76.92308
Error range = [ -35.37259 .. 33.39255 ]
================================================================
CssCompetitors  vs.  CBO ,  Num. attributes per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.74012709 0.161815681  4.573890 4.787517e-06
x1           0.09507526 0.035909064  2.647668 8.104919e-03
x2          -0.01886545 0.008670435 -2.175836 2.956751e-02
R2log =  0.9372796
Excluded as outliers:  linux  ( 1 / 13 )
MMRE =  10.40584
Pred(25) =  84.61538
Error range = [ -12.42110 .. 37.71880 ]
================================================================
CssCompetitors  vs.  CBO ,  Num. methods per class
               Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.87780950 0.232389759  3.777316 0.0001585277
x1           0.07406512 0.032130215  2.305155 0.0211579023
x2          -0.01997669 0.009196706 -2.172157 0.0298438119
R2log =  0.937137
Excluded as outliers:  linux perl  ( 2 / 13 )
MMRE =  10.93107
Pred(25) =  84.61538
Error range = [ -19.88941 .. 49.1187 ]
================================================================
CssCompetitors  vs.  CBO ,  RFC
                Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.787394464 0.169172644  4.654384 3.249505e-06
x1           0.071414697 0.031454377  2.270422 2.318202e-02
x2          -0.002843488 0.001340449 -2.121295 3.389700e-02
R2log =  0.937005
Excluded as outliers:  linux  ( 1 / 13 )
MMRE =  10.90620
Pred(25) =  84.61538
Error range = [ -8.771909 .. 50.57831 ]
================================================================
CssCompetitors  vs.  LCOM ,  eLOC per method
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.7668079 0.21743061  3.526679 0.0004208073
x1          -0.1749729 0.07347917 -2.381259 0.0172535867
x2           0.0493524 0.01547119  3.189954 0.0014229527
R2log =  0.9370105
Excluded as outliers:  linux perl sqlite glibc  ( 4 / 13 )
MMRE =  15.9887
Pred(25) =  76.92308
Error range = [ -47.14434 .. 25.71036 ]
================================================================
CssCompetitors  vs.  eLOC per method ,  Num. attributes per
class
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.74731307 0.17225582  4.338391 1.435297e-05
x1           0.03227598 0.01000799  3.225022 1.259631e-03
x2          -0.04365922 0.01298912 -3.361215 7.760045e-04
R2log =  0.9418964
Excluded as outliers:  linux gdb  ( 2 / 13 )
MMRE =  12.53779
```

```
Pred(25) =  76.92308
Error range = [ -51.62918 .. 29.09189 ]
================================================================
CssCompetitors  vs.  eLOC per method ,  RFC
                Estimate  Std. Error   z value     Pr(>|z|)
(Intercept)  0.747060085 0.174237363  4.287600 1.806141e-05
x1           0.020965968 0.008483986  2.471240 1.346453e-02
x2          -0.003949659 0.001529169 -2.582879 9.797979e-03
R2log =  0.9379307
Excluded as outliers:  linux  ( 1 / 13 )
MMRE =  11.65961
Pred(25) =  84.61538
Error range = [ -22.29256 .. 49.51463 ]
================================================================
CssCompetitors  vs.  McCabe ,  Num. attributes per class
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.75011045 0.18473932  4.060372 4.899451e-05
x1           0.26300399 0.09933235  2.647717 8.103722e-03
x2          -0.04622622 0.01528033 -3.025210 2.484602e-03
R2log =  0.9383112
Excluded as outliers:  linux gdb  ( 2 / 13 )
MMRE =  13.39119
Pred(25) =  84.61538
Error range = [ -50.786 .. 30.44502 ]
================================================================
CssCompetitors  vs.  Num. attributes per class ,  Num. attribute
per method
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  0.65070308 0.23546012  2.763538 0.0057178357
x1          -0.05202552 0.01548510 -3.359714 0.0007802314
x2           0.62230991 0.24495212  2.540537 0.0110682387
R2log =  0.9398034
Excluded as outliers:  linux gdb posgresql  ( 3 / 13 )
MMRE =  15.54464
Pred(25) =  69.23077
Error range = [ -52.57854 .. 41.16742 ]
================================================================
CssCompetitors  vs.  Num. attributes per class ,  Num. public
methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)  7.413532e-01 2.199416e-01  3.370682 0.0007498234
x1          -2.789214e-02 1.397088e-02 -1.996448 0.0458851417
x2           5.176754e-05 1.913798e-05  2.704963 0.0068311985
R2log =  0.9371734
Excluded as outliers:  linux cygwin sqlite gdb  ( 4 / 13 )
MMRE =  12.83062
Pred(25) =  76.92308
Error range = [ -18.43080 .. 44.29059 ]
================================================================
```

Trustworthiness

```
================================================================
Trustworthiness  vs.  ACC
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) 0.06107287 0.16333649 0.3739083 0.7084725473
x1          0.10715451 0.02948285 3.6344685 0.0002785543
```

```
R2log =  0.9505937
Excluded as outliers:  linux perl ldap-src  ( 3 / 13 )
MMRE =  12.52097
Pred(25) =  92.3077
Error range = [ -23.33907 .. 72.07474 ]
================================================================
Trustworthiness  vs.  CBO
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) 0.08130587 0.16163887 0.5030094 0.6149576697
x1          0.10471561 0.02954980 3.5436990 0.0003945554
R2log =  0.949967
Excluded as outliers:  linux perl ldap-src  ( 3 / 13 )
MMRE =  12.77652
Pred(25) =  92.3077
Error range = [ -22.90171 .. 72.18718 ]
================================================================
Trustworthiness  vs.  LCOM
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.5367880 0.22263195 -2.411101 1.590446e-02
x1           0.1632395 0.03165883  5.156208 2.520014e-07
R2log =  0.942918
Excluded as outliers:  linux glibc openssl subversion  ( 4 / 13
)
MMRE =  13.25279
Pred(25) =  76.92308
Error range = [ -36.55212 .. 26.49659 ]
================================================================
Trustworthiness  vs.  eLOC per method
               Estimate Std. Error    z value     Pr(>|z|)
(Intercept) -0.06468410  0.1750661 -0.3694839 7.117671e-01
x1           0.02970275  0.0076190  3.8985095 9.678659e-05
R2log =  0.9414993
Excluded as outliers:  linux perl sqlite  ( 3 / 13 )
MMRE =  13.79388
Pred(25) =  84.61538
Error range = [ -25.21418 .. 61.3659 ]
================================================================
Trustworthiness  vs.  McCabe
              Estimate Std. Error  z value     Pr(>|z|)
(Intercept) 0.07913687 0.17483670 0.452633 0.6508130099
x1          0.20597148 0.05997392 3.434351 0.0005939744
R2log =  0.9494126
Excluded as outliers:  linux sqlite perl ldap-src  ( 4 / 13 )
MMRE =  14.86738
Pred(25) =  84.61538
Error range = [ -21.12578 .. 69.69928 ]
================================================================
Trustworthiness  vs.  Num. methods
                 Estimate   Std. Error    z value     Pr(>|z|)
(Intercept) -5.107505e-02 1.733888e-01 -0.2945694 7.683228e-01
x1           6.133483e-05 1.521912e-05  4.0301175 5.574897e-05
R2log =  0.9460112
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  16.89503
Pred(25) =  76.92308
Error range = [ -22.98601 .. 59.99431 ]
================================================================
Trustworthiness  vs.  Num. attribute per method
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -0.5984576  0.2765321 -2.164152 3.045266e-02
```

```
x1           0.8732914  0.2015840  4.332147 1.476624e-05
R2log =  0.9446666
Excluded as outliers:  sqlite perl linux  ( 3 / 13 )
MMRE =  14.56759
Pred(25) =  76.92308
Error range = [ -31.77072 .. 43.18899 ]
================================================================
Trustworthiness  vs.  Num. public methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -5.649428e-02 1.753479e-01 -0.322184 7.473133e-01
x1           6.368713e-05 1.589916e-05  4.005692 6.183616e-05
R2log =  0.945877
Excluded as outliers:  linux cygwin gdb perl  ( 4 / 13 )
MMRE =  17.25905
Pred(25) =  76.92308
Error range = [ -22.75501 .. 61.06993 ]
================================================================
Trustworthiness  vs.  eLOC per method ,  eLOC per class
                Estimate   Std. Error    z value     Pr(>|z|)
(Intercept)  0.040254753 0.1772362984  0.2271248 0.8203267279
x1           0.043615357 0.0113909548  3.8289466 0.0001286929
x2          -0.001606256 0.0006217851 -2.5832979 0.0097860770
R2log =  0.9437019
Excluded as outliers:  libxml2 linux perl gdb  ( 4 / 13 )
MMRE =  21.79792
Pred(25) =  69.23077
Error range = [ -77.90352 .. 54.46817 ]
================================================================
Trustworthiness  vs.  Num. methods ,  Num. methods per class
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.005593e+00 3.681290e-01 -2.731632 6.302148e-03
x1           2.388715e-05 8.933793e-06  2.673797 7.499777e-03
x2           5.906534e-02 1.350339e-02  4.374113 1.219274e-05
R2log =  0.9441127
Excluded as outliers:  linux glibc openssl libxml2  ( 4 / 13 )
MMRE =  19.41418
Pred(25) =  69.23077
Error range = [ -46.35815 .. 40.311 ]
================================================================
Trustworthiness  vs.  Num. methods per class ,  Num. public
methods
                 Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -1.002688e+00 3.668939e-01 -2.732910 6.277752e-03
x1           5.894094e-02 1.347003e-02  4.375710 1.210377e-05
x2           2.395724e-05 8.942985e-06  2.678886 7.386752e-03
R2log =  0.9441462
Excluded as outliers:  linux glibc openssl libxml2  ( 4 / 13 )
MMRE =  19.52612
Pred(25) =  69.23077
Error range = [ -46.24506 .. 40.26507 ]
```