Defining Software Evolvability from a Free/Open-Source Software Perspective

Jean-Christophe Deprez[⊂], Frédéric Fleurial Monfils[⊂], Marcus Ciolkowski[∈] and Martín Soto[∈]
Fraunhofer IESE

[⊂]CETIC (Charleroi, Belgium), [∈]Fraunhofer IESE (Kaiserslautern, Germany)

jean-christophe.deprez@cetic.be, frederic.fleurialmonfils@cetic.be,
ciolkows@iese.fraunhofer.de, soto@iese.fraunhofer.de

Abstract

This paper studies various sources of information to identify factors that influence the evolvability of Free and Open-Source Software (FlOSS) endeavors. The sources reviewed to extract criteria are (1) interviews with FlOSS integrators, (2) the scientific literature, and (3) existing standard, norms as well as (4) three quality assessment methodologies specific to FlOSS , namely, QSOS, OpenBRR and Open Source Maturity Model. This effort fits in the larger scope of QUALOSS, a research project funded by the European Commission, whose goal is to develop a methodology to assess the evolvability and robustness of FlOSS endeavors.[1]

1. Introduction

Many organizations have started to integrate Free (libre) Open-Source Software (F*l*OSS[2]) in their software systems and infrastructures. Furthermore, systems relying on F*l*OSS components become more frequently available to or viewable by customers. In turn, organizations want guarantee regarding the quality of the F*l*OSS components integrated in their solutions but also regarding the complete F*l*OSS endeavor[3]. So, companies are faced with the problem of identifying F*l*OSS endeavors that meet their quality needs, in particular, concerning evolvability so as to avoid having to switch F*l*OSS components after a few years because the initial F*l*OSS endeavor selected has come to a halt.

This paper presents the initial task of QUALOSS, a European-funded research project whose overall objective is to develop a methodology to assess evolvability and robustness of F*l*OSS endeavors in part to facilitate the F*l*OSS acquisition process in Industry but also to address quality concerns of other stakeholders such as F*l*OSS community members. To develop this methodology, QUALOSS draws its inspiration from the Goal-Question-Metric (GQM) paradigm [1]. The GQM is set up in three steps. First, the reasons for wanting to measure quality are identified, i.e., the goal are determined. The second step enumerates questions that will help assess whether a quality goal has been reached or not. Finally, the third step transforms the questions into metrics. When applying GQM to a very specific industrial context, one may directly interview stakeholders to determine exact quality goals. However, the F*l*OSS world is a much broader context hence we slightly adapt the initial step. This paper only covers the initial step of the process, i.e., identify quality goals or criteria.

Incidentally, the current definition of quality as specified in [4] and the different standards and norms related to quality are better suited to the traditional commercial model. For instance, the ISO 9126 standard [2] and the CMMI model [3] are oriented towards traditional organizations. ISO 9126 suggests metric formulæ based on data not commonly found in F*l*OSS project repositories, for example, many metrics require data extracted from design and specification documents. In F*l*OSS project, such information is

[1] Acknowledgment: this work is partly funded by the QUALOSS project, FP6 #033547 and the Belgian project CRAQ-155 convention EP1A1203000073F-130008.
[2] The '*l*' in F*l*OSS is the italicized letter l that stands for *libre*.
[3] A F*l*OSS endeavor is defined later the paper

rarely found in a single, well-identified document. Similarly, CMMI processes such as validation may be problematic to evaluate properly for F/OSS endeavors that heavily relying on user community for testing.

The main contributions of this paper are, first, to explain how we have adapted the initial step of the GQM to apply it to the broad context of F/OSS. Second, to inventory criteria related to evolvability of F/OSS endeavor. Criteria are extracted from several sources of information. In particular, (1) we conducted interviews of several F/OSS integrators; (2) we review the existing ISO9126 standards; (3) we studied the scientific literature on software evolution; (4) we analyzed three existing F/OSS assessment models, namely QSOS, OpenBRR and Open Source Maturity Model.

The paper is organized as follows. In Section 2, we present definitions and explain how we are setting up the initial step of GQM. Section 3 presents our study of existing works to extract criteria relevant to evolvability of F/OSS endeavors. Section 4 presents related efforts. Then, Section 5 concludes and presents our future work.

2. Overview of QUALOSS and Initiating GQM

QUALOSS [20] proposes to apply the Goal-Question-Metric paradigm (GQM) [4] to create an assessment methodology specifically tailored to evaluate the evolvability and robustness of F/OSS endeavors. Data to feed to the methodologies will be extracted from various data sources, among others, software distributions list, version control repositories, issue tracking data, mailing lists, websites, and even datasets outside the direct scope of F/OSS projects such CVE [21] and NVD [22] security databases, publication databases [23, 24], and eventually other F/OSS data repositories created by other projects such as FlossMole [25], FLOSSMETRICS [26], or ohloh [27].

As Wernick shown in [19], the GQM approach has been successfully applied in industry to help organizations enter in a continuous cycle of product-quality assessment and improvement but it is a challenge to apply GQM to a wider context such as, in our particular case, the whole of F/OSS world. When dealing with a broader context, it becomes harder to identify all relevant stakeholders, which, in turn, increases the risk of missing important quality goals. We present a more detail account of our effort to inventory and categorize our goals and our questions in [28], which also include our validation plan.

In order to identify as complete a set of relevant quality goals, we first need to define what evolvability means. In turn, a definition of F/OSS endeavor is also required.

The definition of F/OSS endeavor is guided by concepts of activity theory, which highlights the elements involved in a human-group activity [5]. It is in complete agreement with the view presented by CMMI where, in [3], product development is shown to be supported by the three pillars: people, tools, and processes.

Definition: A F/OSS endeavor includes

- The software product (which itself eventually includes code, documentation, and tests),
- The community (members closely or remotely connected to the endeavor),
- The rules and division of labor that community members obliged to when performing activities (aka development processes),
- The tools used by the community during their respective activities. A tool can be a version control system, a code or parser generator, or a library developed by another F/OSS endeavor.

Our definition does not impose a specific scope on software product however once defined, it transitively defines the scope of community, processes, and tools. A large scope on software product will likely make it more complex to identify clearly the complete scope of community, development processes, and tools associated. Here are a few examples of software product scopes. The most common scope is likely to be defined at the level of a single F/OSS project such as BIRT under Eclipse or Apache Jakarta and all its subprojects; in both cases all versions of the software product would fall under the selected scopes. Conversely, a scope could also be much smaller such as a specific version of a F/OSS product e.g., Azureus 2.4.

Definition: Evolvability of F/OSS endeavor is the degree to which a F/OSS endeavor can evolve or is perceived to be evolvable.

Pragmatically, we instantiate the first part of the *or* conjunction in the definition over each element of a F/OSS endeavor. In turn, evolvability of a F/OSS endeavor is the degree to which the software product, the community, the development processes and the tools used can evolve.

Furthermore, the second part of the *or* conjunction the definition “is perceived to be evolvable” indicates that there is always a degree of uncertainty in determining evolvability, in turn, factors that affect the degree of confidence granted to a F/OSS endeavor's evolvability could themselves influence the evolvability

of a F*l*OSS endeavor. For example, the age of project or the number of bugs reported in the past may influence the adoption rate of the product hence the user community evolvability.

To have a complete understanding of evolvability, we must therefore include criteria that directly influence or impact evolvability but also those that are perceived to influence evolvability. Moreover, given that we apply the GQM approach, we must inventory as many goals as stakeholders may have. And such goals could definitely be defined on the age of a F*l*OSS endeavor.

The next section inventories a long list of criteria that influence evolvability. This indeed will help continue with the initial step of GQM, which consist in defining our measurement sub-goals. These criteria are our quality goals. They specify the aspects to monitor about a F*l*OSS endeavor in order to assess its evolvability.

3. Criteria of Evolvability for F*l*OSS endeavors

Section 3.1 presents the quality criteria identified by interviewing F*l*OSS integrators. Section 3.2 reviews characteristics related to software product evolvability and robustness present in ISO 9126 [2]. Section 3.3 enumerates quality goals extracted from the scientific literature, and Section 3.4, identifies the quality goals used by the three F*l*OSS assessment methodologies, QSOS [6], OpenBRR [7], and OSMM [8].

It is worth noting that we also tried to maintain the review of standard and of the scientific literature separated from the analysis of the F*l*OSS assessment methodologies in order to keep a neutral viewpoint not influenced by any F*l*OSS specifics. In turn, we started our analysis by interviewing F*l*OSS users (or adopters), then went on with review of the ISO 9126 standards and of the scientific literature in parallel. Finally, we finished our analysis by reviewing the F*l*OSS assessment methodologies, which, a priori, seems to be a richer source of information for our purpose.

3.1 Evolvability from User Interviews

The goal of the interviews was to probe the relevant quality aspects used by F*l*OSS users in industry when assessing F*l*OSS components. The approach we followed was to elicit goals from industrial partners through a structured interview. In total, we interviewed nine practitioners, in most cases responsible for IT in their organizations, from five different domains: OSS developer/integrator, general IT, (web-based) Services, Health care, and public administration.

Preliminary results indicate that in all cases, the evaluation criteria for F*l*OSS components are ad-hoc; that is, usually one expert does the evaluation using (often implicit) criteria for evaluating required product qualities. In the interviews, we tried to elicit these criteria, as they can serve as basis for a definition of evolvability and robustness from the practitioners' viewpoint. At that stage, we had not yet defined evolvability as presented in chapter 2. In fact, one of the goals for the interviews was to help us come up with a definition. In turn, the interviews clearly show that evolvability is not limited to the product but also includes other elements such as community and processes.

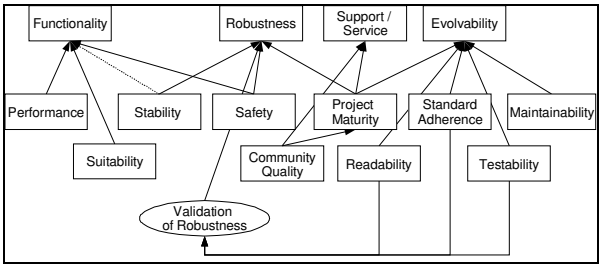

Figure 1: Preliminary Definition of Robustness and Evolvability from Interviews. An arrow indicates a relationship of the kind contributes to

Figure 1 shows the preliminary results from the interviews concerning relevant quality criteria for evaluation of F*l*OSS components. There are four top-level constructs that users are interested in: Functionality, robustness, level of support, and evolvability.

Functionality: Typical evaluation criteria are performance and suitability to the problem to be solved; in some cases, stability and safety are considered as part of the required functionality. These criteria are usually evaluated by conducting "ad-hoc" testing.

Robustness: Typical evaluation criteria are stability, safety, maturity, and community quality. In addition, readability, adherence to standards, and testability are considered as enhancing robustness, as they increase the evaluator's trust into the product.

Support: Evaluation criteria are community quality, and maturity

Evolvability: Typical evaluation criteria are maturity, readability, adherence to standards, interoperability, testability, and maintainability.

In the following, we elaborate on the criteria used to evaluate robustness and evolvability.

Community quality: Evaluation criteria used are the continuity of the community (i.e., whether the project

will go on further), activity in mailing lists, whether developers with reputation are in the team, the size of the community, whether company support exists for the community, quality of responses to questions / ability of the community to explain questions, and response time for bug fixes.

Continuity of the community is evaluated by looking at the frequency of releases, ease of updates, whether the project is based on standards, existence of a roadmap and evaluation of previous timeline, and the project is sufficiently focused to guarantee future survival.

Project maturity: This is evaluated by using criteria such as that the project should be "not too young" (i.e., age of the project), neither should it be a closed project, success cases should exist in large companies, a stable version of the product should be available, and user opinions and OSS community views should be generally positive.

Stability and safety: These criteria are evaluated by looking at user opinions in mailing lists, and by doing ad-hoc tests.

Readability, ***maintainability*** and ***testability***: These criteria are evaluated by looking at the clarity of code, use of standards in the project, and at code documentation. That is, the interviewed persons do not distinguish between these different quality attributes, except by using different implicit criteria in testing.

Standard adherence: Main evaluation criterion is whether the project adheres to relevant development standards, such as design patterns or existing libraries. Concrete criteria used are typically defined ad-hoc by the evaluators.

The insights from the interviews will provide one input for defining evolvability and robustness of F/OSS endeavor. In Figure 1, all the criteria that are the source of an arrow going directly or indirectly to evolvability are of interest.

3.2 Evolvability and Robustness in ISO9126

One aspect became quite clear after conducting our interviews with F/OSS integrators: evolvability and robustness are concepts that in the mind of people covered a broad spectrum. Interviewee included factors whose contribution to evolvability and robustness were not always obvious and direct, for instance, readability of the source code. In turn, to identify as complete a set of goals relevant to many stakeholders, we have to keep an open mind and identify a large range of quality criteria that contributes to evolvability and robustness even if such a contribution is not necessarily major. In turn, based on our long list of quality criteria, all stakeholders will be able to express their quality needs (or goals).

ISO9126 does not cover community characteristics however it presents an interesting list of characteristics for product quality. Since evolvability and robustness also covers the software products, ISO9126 provides appropriate information for our identifying our criteria.

ISO9126 contains 6 main characteristics that in turn contain sub-characteristics, 27 in total. The main characteristics are: functionality, usability, efficiency, maintainability, reliability, portability. Although we could segregate and select based on these six main characteristics, we could miss some sub-characteristics. To support our point, a survey led by Ho-Won Jung *et al*. showed that for many people the ISO9126 classification was not intuitive [9].

Based on the definition provided by ISO9126, we have identified the following characteristics and sub-characteristics to be relevant to evolvability.

All characteristics under maintainability, including it, were found to be relevant: ***maintainability***, ***analyzability***, ***changeability***, stability (however, from the definition, we prefer to substitute it with the term ***flexibility***), ***testability***, and ***compliance to maintainability standards***. Many characteristics under portability including it, could also contribute to evolvability: ***portability***, ***adaptability***, ***installability***, ***coexistence***, ***compliance to portability standards***. Under other main characteristics, we have also identified the following sub-characteristics as maybe impacting the evolvability of a F/OSS endeavour: ***interoperability*** (under functionality), ***usability*** characteristics (***understandability***, ***learnability***, ***operability***, ***attractiveness***) are likely to influence rate of adoption and also to motivate the developer community hence they influence community evolvability.

3.3 Evolvability in the Scientific Literature

We reviewed the literature for definitions of evolvability and robustness. Many researchers have studied software evolution starting in 1976 with Lehman's laws of software evolutions [10, 11]. The most thorough definition of software evolution seems to come from Perry [12], who identifies three dimensions that influence software evolution: evolution in domain, in experience, and in process. The text box below presents Perry's 3 dimensions and their types of evolution. The text in bold represents the quality criteria we identified for each type of evolution. Furthermore, the text box lists characteristics extracted from Seifert and Pizka's definition [13] that point to the importance of refactoring. Finally, we also refer to

research on modeling software evolution based on system dynamics [14, 15, 16]. Although not the primary intent of that research, they are the only pieces of scientific literature we could find that emphasize the need to integrate new members in the team producing a software product.

Table 1: Characteristics related to evolvability extracted from Perry's definition of software evolution, from Seifert and Pizka's and from research in system dynamics modeling research. (continues on next page)

Perry's definition *and related quality characteristics*
Evolution in the Domains
Evolution of the domains in the Real World (new requirements, influence of the systems once introduced in the Real World).
• **Stability of Users' Needs**
• **Stability of norms and standards implemented by the software product**
• **Stability in Laws, Regulations implemented by the software product**
Evolution of the abstract representation of the Real World domains
• **Stability/Maturity of the design (abstraction) in modeling the particular real world domain implemented by the software product.**
Evolution in Experience
Evolution of Understanding based on Feedback
• **Capacity to listen criticism from the users' community (related to software defects)**
• **Capacity to listen to suggestion from the users' community (related to feature requests)**
Evolution of Understanding Experiments
• **Willingness to measure work activities and work products**
Evolution in process
Evolution of Methods (Theories and Experiences)
• **Stability of the theories implemented in the software product**
Evolution of technologies
• **Stability/Maturity of the technologies used for implementing the software product**
• **Stability/Maturity of competing technologies**
Evolution of organizations
• **Willingness of the developing community to follow organizational process**
• **Maturity of the development process**
• **Willingness of the developing community to follow new organizational process**
• **Willingness of the developing community to record new types of data**
Additional quality characteristics from ***Seifert and Pizka's definition*** *of software evolution:*
• **Willingness of the developer community to refactor and reengineer a system (this characteristic should be repeated respectively for communities of analyst/architect and management)**

(Continue from Table 1)

Additional quality characteristic extracted from software evolution from ***system dynamics modeling research****:*
• ***Capability of experts in the community to integrate new members.***

Table 2: Quality criteria extracted from QSOS, OpenBRR and OSMM.

Project age	Modularity of Architecture
Product stability (	Ease of build-ability
Management ability to solve crisis	Ease of extensibility or plug-ability
Professionalism of Process for proposition of modifications	License permissiveness
Road map availability and precision	License protection against proprietary forks
Driving force behind production	Size of copyright owning team
Strategical Independence	Source comment volume
Fork probability	Use of design patterns
Project popularity	Technological dispersion
Project referencing (used by)	Intrinsic complexity of algorithms
Management style	Code expertise availability
Developer identification	Protocol for external communication (product)
Developer turnover	Diversity of user community mailing lists
Activity on bugs	Ease of Vanilla Deploy-ability (Installability or Configurability)
Bug reporting activity (in a specified period of time)	Community involvement on security issues
Reactivity on critical bugs	Testability for preformance (including presence of testsuite and benchmark reports)
Reporting activity on product vulnerability	Tunability & Configurability (on user's side)
Reactivity on security bugs	Scalability (of design/architecture)
Activity on functionalities	Feature Configurability (on deployment or by user)
Activity on releases	Diversity of documentation contributors
Independence of development	Diversity of actual deployments (depends on diversity of user community)
Training diversity (geographical, cultural and level)	Integrability / permeability of core developer team
Support (level of commitment dedicated to support)	Volume of contributing community
Support professionalism	Diversity of contributing community
Consulting diversity (geographical, cultural and level)	Volume of book published
Documentation availability	Leading team size
Documentation recency	
Definition and Documentation of QA Process	
Tools Used for Project Management and for QA activities	
Diversity of packaged distributions (with source, for what *nix distribution)	

3.4 Evolvability in QSOS, OpenBRR and OSMM

In this section, we present quality criteria found in the three F/OSS assessment methodologies, QSOS [6], OpenBRR [7] and OSMM [8]. The criteria in the table below are sometimes reworded from the original form to bring them to an appropriate level of abstraction. For

example, OpenBRR words them as metrics e.g., "Number of bugs reported in the last 6 months", which we changed into "Bug reporting activity (within a given period of time)". Some of the criteria are very closely related. In some cases, we could get rid of some, in others we could organize them in a hierarchy or if a hierarchy is not intuitive, we could clearly express the relationships between two or more criteria. This exercise is left for future work.

4. Related Works

As shown in section 3, ISO 9126 influenced our work. On the other hand, the ISO 9126 quality characteristics are often not precise enough for our needs. Many stakeholders will need quality goals refined to a much finer level so they become quite concrete. Furthermore, ISO9126 focuses mainly on software product; it does not intend to characterize community, processes and tools.

More directly related to our work, the three F*l*OSS assessment methodologies, QSOS, OpenBRR and OSMM also inventory criteria of F*l*OSS endeavor. They had a strong influence on our work. However, they lack depth in certain aspect. For example, they do not provide rigorous definition of what they intend to characterize. Hence, each of the three methodologies, when taken on its own, overlooks certain criteria. Moreover, none of them investigate in details the areas of development processes or tools used. Most importantly, none are directive enough in asking evaluators to record the data sources to measure. This could lead to very subjective results in the end.

5. Conclusions and Future Work

The QUALOSS project is applying GQM to F*l*OSS endeavors in order to assess their evolvability and robustness. So far, we have applied the initial step of GQM, i.e., identification of the top-level measurement goals and we have also inventoried a list of criteria that have an impact of the evolvability of a F*l*OSS endeavor. To obtain this long list of quality criteria, we interviewed F*l*OSS integrators, surveyed the scientific literature, and reviewed the three existing F*l*OSS assessment methodologies, QSOS, OpenBRR and OSMM. These criteria will then be used to either elaborate more specific measurement sub-goals and also to enumerate questions whose answer will determine whether or not a F*l*OSS endeavors meets our goals.

The following steps of our research are:

- Review the literature on F*l*OSS development process and also extract the practices of CMMI that could be of interest to F*l*OSS endeavors. From these review we expect to extract new criteria related evolvability.
- Compare and prune our criteria to obtain a single comprehensive list that combine results from different sources.
- Organize the list hierarchically, similarly to ISO9126, QSOS, OpenBRR and OSMM.

In parallel to identifying a comprehensive list of quality goals, we have started inventorying indicators and metrics that can be used to measure the criteria specified. The next part of QUALOSS will connect these criteria or goals these indicators and metrics continuing with the GQM approach.

6. References

[1] Victor R. Basili, Software Modeling and Measurement: The Goal Question Metric Paradigm, Computer Science Technical Report Series CS-TR-2956 (UMIACS-TR-92-96), September 1992, University of Maryland , College Park, MD, USA.

[2] ISO/JTC 1/SC 7, ISO/IEC 9126 Software Engineering - Product Quality (Part1-4), 2001.

[3] Mary Beth Chrissis et al., CMMI — Guidelines for Process Integration and Product Improvement, August 2005, Adison-Wesley, Boston, USA, (9th Printing).

[4] ISO/TC 176/SC 1, ISO 8402 - Quality management and quality assurance -- Vocabulary, 1994

[5] Bjørke, S.Å. (2005) 'The Concepts of Communities of Practice, Activity Theory and Implications for Distributed Learning' in Lager, R. (ed) *Artikler om høgskolepedagogikk*, *Skriftserien nr 118*, Agder University College, Kristiansand 2005, ISBN: 82-7117-558-0

[6] Atos Origin, Method for Qualification and Selection of Open Source software (QSOS) v1.6, April 2006, Licensed under GNU Free Documentation License . (http://www.qsos.org/download/qsos-1.6-en.pdf)

[7] OpenBRR.org (SkipeSource, Carnegie Mellon West, Intel), Business Readiness Rating for Open Source, 2005, BRR_whitepaper_2005RFC1.pdf. (http://www.openbrr.org)

[8] Frans-Willem Duijnhouwer and Chris Widdows (Capgemini), Open Source Maturity Model 1.5.3, August 2003, (Expert Letter) (http://www.seriouslyopen.org)

[9] Ho-Won Jung, Seung-Gweon Kim and Chang-Shin Chung, Measuring Software Product Quality: A Survey of ISO/IEC 9126 , IEEE Software, September/October 2004, pp 88-92.

[10] Belady, L., and Lehman, M., A Model of Large Program Development, IBM Systems Journal, Vol. 15(1), 1976, pp 225-252.

[11] Lehman, M. Meir, Programs, lifecycles and the Laws of Software Evolution, Proceedings of the IEEE, Sept. 1980, Vol. 68(9), pp 1060-1076, ISBN:0018-9219.

[12] Dewayne E. Perry, Dimensions of Software Evolution, 1994, Proc. of the Int. Conf. on Software Maintenance (Ed. Hausi A. Muller and Mari Georges), pp 196-203.

[13] Tilman Seifert and Markus Pizka, Supporting Software-Evolution at the Process Level, Net.ObjectDays 2003, ,September 2003, Erfurt, Germany, Ed. tranSIT GmbH.

[14] Marc I. Kellner, Raymond J. Madachy, and David M. Raffo, Software Process Simulation Modeling: Why? What? How?, Journal of Systems and Software, 46(2/3), April 1999.

[15] Manny M. Lehman and Juan F. Ramil, Modeling Process Dynamics in Software Evolution Processes - Some Issues, Workshop on Software Change and Evolution (SCE'99), May 17, 1999.

[16] Manny M. Lehman and Paul Wernick, System Dynamics Models of Software Evolution Processes, Int. Wrkshp on the Principles of Soft. Evol., IWPSE-98 (ICSE-20), April 1998, Kyoto, Japan, pp 6-10.

[17] IEEE, IEEE Standard 601.12-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE Press, 1990

[18] D. Firesmith, Common Concepts Underlying Safety, Security, and Survivability Engineering, SEI Technical Report CMU/SEI-2003-TN-033, 2003

[19] P. Wernick, Identifying and Justifying Metrics for Software Evolution Investigations Using the Goal-Question Metric Method, FEAST 2000 Workshop, Imperial College, London, 10-12 July 2000.

[20] QUALOSS, Quality of Open Source Software, http://www.qualoss.eu

[21] CVE, Common Vulnerabilities and Exposures, http://cve.mitre.org

[22] NVD, NIST National Vulnerability Database, http://nvd.nist.gov

[23] Amazon, http://www.amazon.com

[24] Citeseer, http://citeseer.ist.psu.edu

[25] FlossMole, http://www.flossmole.org

[26] FLOSSMETRICS, http://flossmetrics.org

[27] ohloh. http://www.ohloh.net

[28] Jean-Christophe Deprez, Marcus Ciolkowski and Martín Soto, Deliverable D1.2 Measurement Requirements Specifications – version 1.3, March 16, 2007, http://www.qualoss.org/about/Progress/deliverables/WP1_Deliverable1.2_final.pdf