Supporting custom quality models to analyze and compare open-source software*

Davide Di Ruscio[1], Dimitrios S. Kolovos[2], Ioannis Korkontzelos[3], Nicholas Matragkas[2], Jurgen Vinju[4], and James R. Williams[2]

[1] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
University of L'Aquila, Italy
davide.diruscio@univaq.it
[2] Department of Computer Science
University of York, United Kingdom
{dimitris.kolovos,nicholas.matragkas,james.r.williams}@york.ac.uk
[3] National Centre for Text Mining (NaCTeM)
University of Manchester, United Kingdom
ioannis.korkontzelos@manchester.ac.uk
[4] Centrum Wiskunde & Informatica, The Netherlands
Jurgen.Vinju@cwi.nl

Abstract. Analysing and comparing open source software requires the adoption of quality models that have to be evaluated in order to rank the software systems being compared and to support the final decision about which of them has to be adopted. Since software quality can mean different things in different scenarios, quality models should be flexible in order to accommodate the needs of different users. Over the years several quality models have been proposed. Even though some of them are tool supported, they cannot be extended or customized to better accommodate the requirements of specific business contexts. In this paper, instead of having a fixed model, we propose a workflow and a tool chain to support the specification of custom quality models, which can guide the automated analysis of open source software.

1 Introduction

Deciding whether an open source software (OSS) meets the required standards for adoption in terms of quality, maturity, activity of development and user support is not a straightforward process as it involves exploring various sources of information, including the project's source code repositories, communication channels, and bug tracking systems. This task becomes even more challenging when one needs to discover and compare several OSS projects that offer software of similar functionality (e.g. there are more than 20 open source XML parsers for the Java programming language [1]), and make an evidence-based decision on which one should be selected for the task at hand. Moreover, even when a decision has been made for the adoption of a particular OSS product, decision makers need to be able to monitor whether the OSS project continues

* This paper was partially supported by the EU through the Automated Measurement and Analysis of Open Source Software (OSSMETER) FP7 STREP project (318736).

to be healthy, actively developed and adequately supported throughout its lifecycle, in order to identify and mitigate any risks emerging from a decline in the quality indicators of the project in a timely manner.

The analysis and comparison of open source software requires the specification and quantification of a number of system attributes. Such attributes are usually captured in a quality model. Following [9], a quality model "is the set of characteristics and the relationships between them, which provide the basis for specifying quality requirements, and evaluating quality". Such models are typically organized in a hierarchical manner decomposing successively high-level system attributes into specific metrics at the lowest level.

In the last three decades several quality models have been proposed. Although these models emphasize the need to have quality checks while developing a software system, most of them remain at the conceptual level, and they are not supported by tools and concrete processes. Moreover, even when a tool is provided, this tool and the model are tightly coupled making very difficult to modify or extend a model in order to adjust it to particular scenarios.

Instead of having a fixed model, in this paper we propose a workflow and a tool chain, which permit the definition of custom quality models that best fit a particular business context. The proposed approach has been conceived and implemented in the context of the EU FP7 OSSMETER project, where it is used to perform automatic quality assessment of open source projects.

The paper is structured as follows: Section 2 motivates the research presented in this work. Section 3 presents an overview of the proposed approach. Section 4 presents the tools that have been developed to support the definition of the quality models and the assessment of the specified quality attributes. Section 5 presents the OSSMETER quality metamodel and a sample instantiation of it. Section 6 concludes the paper and presents some perspective work.

2 Background and motivation

Although the most widely accepted software quality models (i.e., *McCall's Quality Model* [11], *Boehm's Quality Model* [3], *ISO 9126* [8] are well established, they do not provide sufficient support for assessing the quality of OSS. This is due to the particularities present in open source software development such as shared artefact repositories, and online communities of developers and users. To fill this gap, researchers and practitioners have developed an array of quality models, which are tailored for the quality assessment of OSS. Such models include the OSMM [6], the OpenBRR [14], the QSOS [12], the OMM [5], the SQO-OSS [13], the QualOSS [7], and the Squale [2] models. Although the strengths and weaknesses of these models have been extensively discussed in the literature, one aspect, which has been neglected is that of operationalisation, i.e. the process through which abstract quality models are translated into actual measurements. OSS *quality model operationalisation* can be decomposed along three axes: *tool support*, *automation*, and *reconfiguration*.

All of the aforementioned OSS quality model are supported by some kind of tooling apart from OSMM and OpenBRR. The types of tooling range from form-based web

Quality Model	Tool support	Automation	Reconfiguration
OSMM	-	-	-
OpenBRR	-	-	-
QSOS	✓	-	-
OpenBQR	✓	-	-
OMM	✓	-	-
SQO-OSS	✓	✓	-
QualOSS	✓	partial	-
Squale	✓	✓	-

Table 1. Summary of quality models for OSS.

applications (e.g. OpenBQR) to distributed platforms (e.g. SQO-OSS). Tool support is a very important aspect of quality assessment since it reduces the time and effort for applying such methods.

The second aspect of quality model operationalisation is related to the level of automation. Automating the collection of measurements can be of paramount importance for scenarios where the quality of software projects has to be monitored in a constant basis. In such scenarios, human interference can be tedious and time-consuming. On the other hand, automation comes at a cost. There are measurements of specific quality attributes that cannot be automated such as functionality. When automatic is the most important concern, then such measurements are ignored. Only three of the quality models mentioned above address the concern of automation. As shown in Table 1, SQO-OSS and Squale are fully automated in terms of metric collection. On the other hand QualOSS is partially automated, since it supports only the automatic collection of data, but it does not provide support for defining and executing metrics on those data.

The last concern, which is related to the operationalisation of quality models is reconfigurability and extensibility. Quality can mean different things to different people. For example in the context of OSS, an experienced user can approach a software project with the intention of contributing to the code base, while a not so experienced user can approach a project with the intention of using the software product. These two users can have different quality concerns. The first might be interested in source code, which is well-designed, extensible, and maintainable, while the second one is more interested in how quickly bugs are fixed, and how helpful the community of the project is. Therefore since software quality depends on the context it is used in, quality models should be flexible in order to accommodate different users and contexts. One issue is how tools supporting a quality model can adjust to changes to the quality model, in order to support different needs. As shown in Table 1, none of the considered OSS-specific quality models support reconfiguration.

3 The OSSMETER approach to OSS evaluation

To address the limitations of current quality model in terms of operationalisation, the OSSMETER project proposes a different approach. Instead of having a fixed model which is embedded in a supporting tool, OSSMETER proposes a workflow and a tool chain, which can support different quality models on demand depending on the context. The OSSMETER project is an EU FP7 project, which extends the scope and effectiveness of existing OSS analysis and measurement tools. OSSMETER proposes a novel

architecture for such tools as well as it makes contributions in the following areas: *i)* language-specific and language-agnostic methods for source code analysis, *ii)* text mining analysis for communication channels and bug tracking systems, and *iii)* modelling of OSS. OSSMETER tools gather and combine information from various sources in order to provide a comprehensive picture of the quality indicators of OSS projects, and thus facilitate better evidence based decision making and monitoring. In addition, OSSMETER provides a consistent way to capture and measure quality indicators across projects by using rigorous metamodelling approaches.

The main outcome of OSSMETER is a toolset[5], which consists of two main components, i.e., a *platform* and the *web application*. The former mines OSS repositories, orchestrates and automatically executes quality metrics, and finally stores the measurements. The OSSMETER web application provides different visualisations and dashboards in order to support decision making. Moreover, the platform provides a REST API, which can be used by external applications to interact with the platform. In this way the OSSMETER platform can serve simultaneously multiple clients, which could possibly belong to different organisations or stakeholders. These stakeholders can potentially be interested in different aspects of quality and therefore the platform has to accommodate this requirement. Therefore in OSSMETER support for alternative quality models takes place at the client level.

The OSSMETER platform proposes a default quality model, which can be used as soon as the client is installed. The quality model can be fully customisable by users without the need of bespoke tooling. In the following section, we will present in more detail the OSSMETER default quality model and how it has been specified.

4 OSSMETER Workflow

To support automatic and reconfigurable quality model operationalisation, a toolchain has been developed. The proposed tool-chain supports the workflow illustrated in Figure 1.

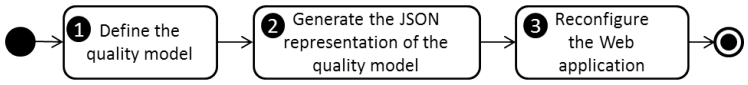

Fig. 1. Definition and use of custom quality models

In activity ❶ of this workflow users define a quality model, which is fit for their purpose. To support this activity we opted for a generic and lightweight technology, namely yEd[6], instead of a highly specialised and heavy-weight modelling technology such as the Eclipse Modelling Framework (EMF) [7]. The reason for choosing a general purpose drawing tool such as yEd for defining quality models has to do with our intention to make the tool approachable not only to engineers familiar with modelling technologies,

[5] https://github.com/ossmeter/ossmeter
[6] http://www.yworks.com/en/products/yfiles/yed/
[7] http://eclipse.org/modeling/emf/

but also to less technical-savvy users such as project and community managers. In fact, the proposed approach and tooling is not only restricted to yEd but it works with any GraphML-compliant tool [4]. Moreover, although we opted for flexibility and ease of use, we still wanted to enjoy some of the perks of formal modelling, such as model validation transformation. To this end we used the flexible modelling approach proposed in [10]. Figure 2 illustrates an excerpt of the default quality model in the yEd editor.

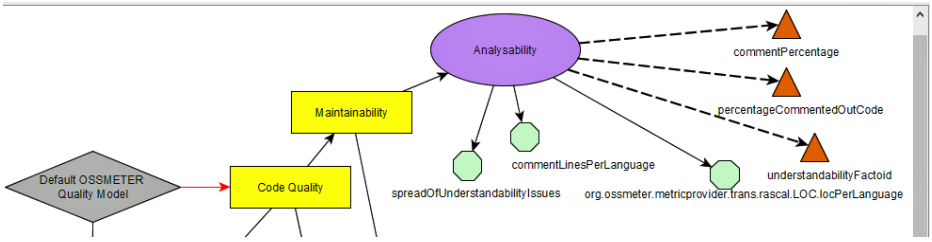

Fig. 2. Excerpt of the default OSSMETER quality model in yEd

Once a quality model is specified in GraphML, it can be automatically transformed into a JSON representation which can be passed around over the network and it can be consumed by the OSSMETER clients (see activity ❷ of the process in Fig. 1). To generate the JSON representation, the user just needs to upload the GraphML file on a dedicated webpage and then the JSON representation is generated automatically. Finally the user just needs to place the generated JSON into the *config* folder of the OSSMETER web application in order to reconfigure it accordingly (see activity ❸ in Fig. 1).

5 The OSSMETER quality metamodel and the default quality model

With the aim of collecting the characteristics that necessarily have to be considered when defining a quality model and the ways to evaluate it, we have analysed the OSS-specific quality models discussed in Section 2. The outcome of the analysis consists of a number of concepts which are precisely represented in the metamodel shown in Fig. 3. In particular, the `QualityAttribute` metaclass represents a quality aspect that is considered to be relevant for contributing the assessment of the quality of a considered product. Each quality attribute has associated a number of *facts* that are obtained by corresponding `Factoid`s. A quality attribute, like *code quality* can be an aggregation of other attributes, like *maintainability*, *maturity*, and *testability*. Each quality attribute specifies how to aggregate the contained attributes in order to provide an overall quality score for the considered attribute. The `Factoid` metaclass represents a fact about the considered product. Such a fact is obtained by considering the values of related metrics. In other words a factoid is an aggregation unit depending on a number of related metrics. Since metric aggregation and metric combination is a very active area of research, OSSMETER does not commit to a specific approach. Thus the factoid mechanism is proposed as a means to aggregate heterogeneous metric providers into a 1-4 star system and a piece of text with an interpretation of the numerical value. The metaclass

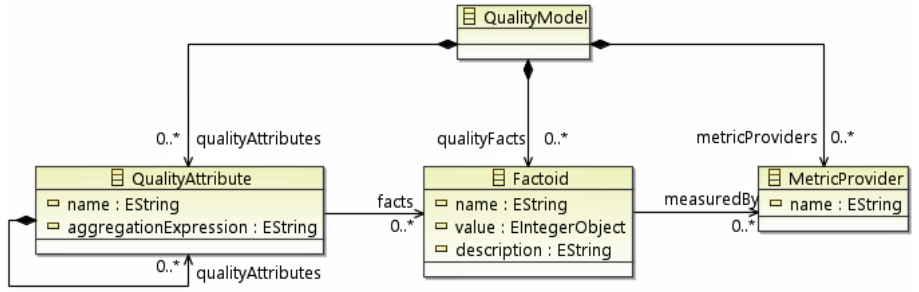

Fig. 3. The OSSMETER quality metamodel

`Metric provider` represents the software component that implements a metric by using the functionality provided by the OSSMETER platform. Lines of Code (LOC) is a very simple example of metric that can be implemented with a corresponding metric provider.

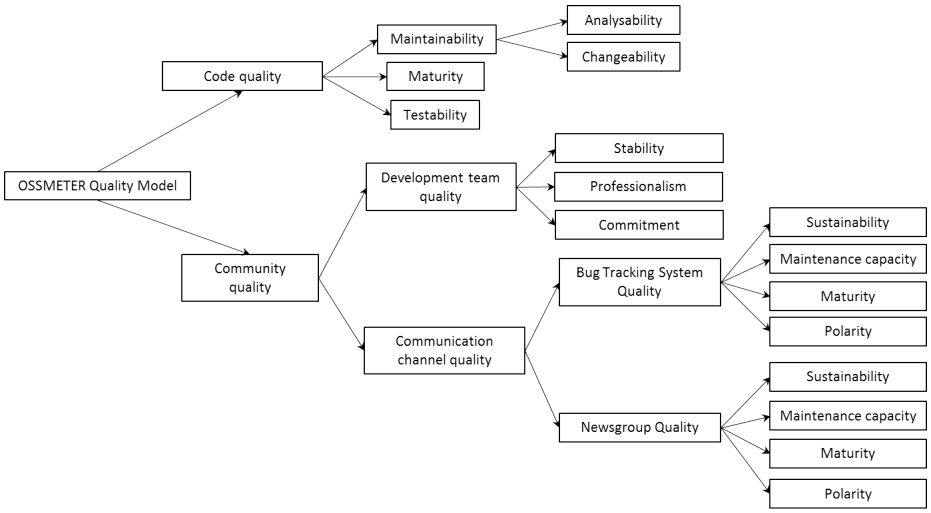

Fig. 4. Graphical representation of the default OSSMETER quality model

The quality model conforming to the metamodel shown in Fig. 3 and that is provided as default in the OSSMETER platform is shown in Fig. 4. It is important to recall that such a model is used to customize the front-end of the OSSMETER web application. For each quality aspect, the factoids (and consequently the metrics providers) that are used for the measurements are also specified. An example of the factoid view of the web application is illustrated in Figure 5. This view shows the factoids associated to the bug tracking system of a project.

Users can extend and refine such model by using the concepts included in the OSSMETER metamodel and the means provided by the tools of the OSSMETER platform outlined in the next section. The proposed default quality model consists of attributes

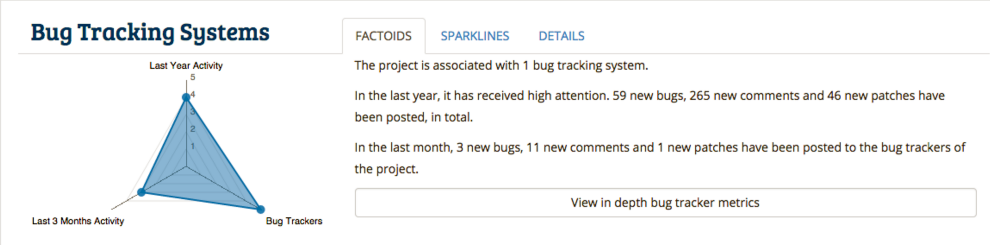

Fig. 5. The factoids page for a sample project

hierarchically organized that are distinguished into those related to the quality of the product code (Section 5.1), and those related to the community built around the considered product (Section 5.2) as described in the next sections.

5.1 Product Code quality

It identifies the quality factors that influence the ability to understand software product, change and test it, as well as it permits to assess the maturity of the considered product by considering the way it is developed. Product code quality is assessed by considering the following attributes:
▷ *Maintainability*: it is an important attribute that refers to how easy is the software to understand and modify. Thus it is further decomposed into the following attributes:

- *Analyzability*: it refers at what extent the source code of the considered product is properly designed and implemented. Adherence to coding style and usage of comments in the code are only some examples of facts that can affect the analyzability of the product code.
- *Changeability*: it refers to the effort required to change the product e.g., to address unforeseen requirements. The size of code, the used programming languages, the amount of cloned code, or even characteristics like coupling and cohesion are examples of facts that can affect the changeability property of a given product.

▷ *Maturity*: it refers to the possibility of having software failures. The amount of error prone or even inefficient code can reduce the overall maturity of the considered product.
▷ *Testability*: software systems are evolving entities and as a such a sign of quality is also the effort needed to test the system when for some reason it has been changed. The availability of unit tests is an example of fact that can affect the testability factor of a given product.

5.2 Community quality

Considering characteristics of the the community strongly influences the overall product quality, especially when they observed over an extended period of time. Moreover, the ability of a OSS community to remain active over time is obviously very important for product survival, and thus very relevant when considering sustainability, i.e., the capability of a product to sustain its self. In the following, we present factors that are related to the quality of product communities.

Development team quality There is a strong correlation between characteristics of a development team and the quality of a software product. The decision of adopting an open source product takes into consideration for sure if the project will be maintained in the long-run. In this respect the following quality attributes are considered:

▷ *Stability*: it refers to the availability of a stable number of developers working on the project. The number of commits over time per developer, is an example of measure that can contribute to understand if the team working on a product is growing or shrinking.

▷ *Professionalism*: it refers to the experience of the developers working on the product. This can be measured by considering their activity e.g., the number of commits per developer, or the number of commits per day, etc.

▷ *Commitment*: if a project is developed and maintained by a representative number of developers with a high level of commitment to the project is of course a good sign. The commitment of a developer can be assessed by analysing his/her activity e.g., if there a commits done over the weekend, or if the amount of code committed by the developer is increasing.

Communication channel quality Further than the development team, the community of a project consists also of people interacting with the bug tracking systems and the newsgroups of the product. In this respect, the following quality attributes have been considered:

▷ *Bug Tracking System Quality*: it refers to quality aspects that can be assessed by considering information retrieved from the bug tracking systems of a product. By considering such systems it is possible to assess the following quality attributes:

- *Sustainability*: the responsiveness to bug reports and to feature requests is an important measure that permits to assess if there is a strong community maintaining the product and its development over time.
- *Maintenance capacity*: it refers to what extent potential users might take advantage of the information available from bug tracking systems and thus being supported during the adoption of the product. The number of available bug tracking systems and the number of comments therein are only some of the facts related to the maintenance capacity of a product.
- *Maturity*: this attribute permits to assess if a considered bug tracking system is mature with respect to the ways bugs are replied (e.g., properly or inadequately), the number of bugs closed, fixed, and even duplicated.
- *Polarity*: sentimental and emotional polarities are important indicators about the overall consideration of the users that are adopting the considered product. For instance, if the average sentimental and emotional polarities in all bug tracking systems associated with the project are negative then it is possible to conclude that people that are using the project and are trying to ask for support are somehow unhappy and consequently the overall quality of the product is affected.

▷ *Newsgroup Quality*: it refers to quality aspects that can be assessed by considering information retrieved from the newsgroups of a product. The quality attributes related to this aspect are the same of those defined for bug tracking systems, i.e., *sustainability*, *maintenance capacity*, *maturity* and *polarity*.

5.3 Measuring quality attributes by means of factoids

As said above, the quality attributes defined in the OSSMETER quality model are measured by means of a set of factoids. A sample of them are shown in Table 2, and Table 3. The second column of the tables shows sample (template) strings that the corresponding factoids emit when executed. It is important to recall that each project, which is monitored by the OSSMETER platform by means of the proposed quality model has a 1-4 stars score related. Currently, such a score is obtained by simply calculating the arithmetic mean of all the factoid scores. This solution is not intended to be optimal and aims at giving just an indicator about the project quality. Of course, such a behaviour can be changed by extending the platform and/or refining the quality model.

Table 2: Measuring the Source Code *Analysability* quality attribute

Factoids	**Example of output strings**
`percentageCommentedOutCode`	The percentage of commented out code over all measured languages is *totalPercentage*. The percentages per language are *commaSeparatedTable*.
`commentPercentage`	The percentage of lines containing comments over all measured languages is *totalPercentage*.
`understandabilityFactoid`	Currently, there is hardly any understandable code in this project and this situation is stable in the last six months. Currently, hard to understand code practices are widely spread throughout the project, but the situation has been improving over the last six months.

Table 3: Measuring the BTS *Sustainability* quality attribute

Factoids	**Example of output strings**
`org.ossmeter.factoid .bugs.channelusage`	The project is associated with *X* bug tracking systems. In the last year, they have *(received high — received much — received some — not received much)* attention. *A* new bugs, *B* new comments and *C* new patches have been posted, in total. *D* new bugs, *E* new comments and *F* new patches have been posted to bug tracking system *G*. In the last month, *H* new bugs, *I* new comments and *J* new patches have been posted to the bug trackers of the project. *K* new bugs, *L* new comments and *M* new patches have been posted to bug tracking system *N*.)

6 Conclusions and Future Work

Analysing and comparing open source projects is a difficult task that involves the adoption of proper quality models defining the attributes to be considered and evaluated. Over the last three decades several quality models have been proposed, and even though

they are well established, they do not provide sufficient support for refinements and customizations in order to better address specific user needs. In this paper we proposed a tool supported approach, which permits to define custom quality models, and to automatically execute the corresponding quality assessment. The generation script, as well as the JSON and GraphML representations of the default quality model can be found online at `https://github.com/ossmeter/oss2015`. Moreover, the OSSMETER platform is publicly available online at `https://github.com/ossmeter/ossmeter`. In the future, we want to improve the proposed approach in two ways. First we want to provide customisation support at the user level and not at the client level. Moreover, although we have evaluated the technical feasibility of the proposed approached in the context of OSSMETER, we are in the process of evaluating other aspects such as usability. The result of this evaluation process will be reported in the future.

References

1. Open source XML parsers in Java. http://java-source.net/open-source/xml-parsers.
2. Alexandre Bergel, Simon Denier, Stéphane Ducasse, Jannik Laval, Fabrice Bellingard, Philippe Vaillergues, and Françoise Balmasand Karine Mordal-Manet. Squale – software quality enhancement. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), European Projects Track*, March 2009.
3. B. Boehm, J. Brown, J. Kaspar, and M. Lipow. *Characteristics of Software Quality*. North Holland, 1978.
4. Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. *Graph markup language (GraphML)*. Bibliothek der Universität Konstanz, 2010.
5. Vieri Del Bianco, Luigi Lavazza, Sandro Morasca, and Davide Taibi. Quality of open source software: The qualipso trustworthiness model. In *Open Source Ecosystems: Diverse Communities Interacting*, pages 199–212. Springer, 2009.
6. F.W. Duijnhouwer and C. Widdows. Open source maturity model. *Capgemini Expert Letter*, 2003.
7. IESE. D4.2 - Metrics and Indicators of the Standard QualOSS Assessment Method, July 2009.
8. ISO. ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model. Technical report, International Organization for Standardization, 2001.
9. ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.
10. Dimitrios S Kolovos, Nicholas Drivalos Matragkas, Horacio Hoyos Rodríguez, and Richard F Paige. Programmatic muddle management. In *XM@ MoDELS*, pages 2–10, 2013.
11. J McCall. *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*, volume 1-3. General Electric, November 1977.
12. Atos Origin. Method for qualification and selection of open source software (qsos), version 1.6. *Disponible en Internet:¡ http://www. qsos. org/download/qsos-1.6-en. pdf*, 2006.
13. Ioannis Samoladas, Georgios Gousios, Diomidis Spinellis, and Ioannis Stamelos. The sqo-oss quality model: measurement based open source software evaluation. In *Open source development, communities and quality*, pages 237–248. Springer, 2008.
14. Anthony I Wasserman, Murugan Pal, and Christopher Chan. The business readiness rating: a framework for evaluating open source. *EFOSS-Evaluation Framework for Open Source Software*, 2006.