OSSMETER: Automated Measurement and Analysis of Open Source Software

Bruno Almeida[1], Sophia Ananiadou[2], Alessandra Bagnato[3], Alberto Berreteaga Barbero[4], Juri Di Rocco[5], Davide Di Ruscio[5], Dimitrios S. Kolovos[6], Ioannis Korkontzelos[2], Scott Hansen[7], Pedro Maló[8], Nicholas Matragkas[6], Richard F. Paige[6], and Jurgen Vinju[8]

[1] UNPARALLEL, Portugal
bruno.almeida@unparallel.pt
[2] National Centre for Text Mining (NaCTeM)
University of Manchester, United Kingdom
sophia.ananiadou,ioannis.korkontzelos@manchester.ac.uk
[3] SOFTEAM, France
alessandra.bagnato@softeam.fr
[4] TECNALIA, Spain
Alberto.Berreteaga@tecnalia.com
[5] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
University of L'Aquila, Italy
{juri.diruscio,davide.diruscio}@univaq.it
[6] Department of Computer Science
University of York, United Kingdom
{dimitris.kolovos,nicholas.matragkas,richard.paige}@york.ac.uk
[7] The Open Group, Belgium
s.hansen@opengroup.org
[8] UNINOVA, Portugal
pmm@uninova.pt
[9] Centrum Wiskunde & Informatica, The Netherlands
Jurgen.Vinju@cwi.nl

Abstract. 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. It involves analysing various sources of information, including the project's source code repositories, communication channels, and bug tracking systems. OSSMETER extends state-of-the-art techniques in the field of automated analysis and measurement of open-source software (OSS), and develops a platform that supports decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of opensource software. To achieve this, OSSMETER computes trustworthy quality indicators by performing advanced analysis and integration of information from diverse sources including the project metadata, source code repositories, communication channels and bug tracking systems of OSS projects.

1 Project data

- **Acronym:** OSSMETER
- **Title:** Automated Measurement and Analysis of Open Source Software

– **Partners:** The Open Group - *Project Coordinator* (Belgium), University of York - *Technical Coordinator* (UK), University of L'Aquila (IT), Centrum Wiskunde & Informatica (NL), University of Manchester (UK), Tecnalia Research and Innovation (ES), UNINOVA (PT), SOFTEAM (FR), Unparallel Innovation (PT)
– **Start date:** 1 October 2012, **Duration:** 30 months
– **Website:** `http://www.ossmeter.eu`

2 Introduction

Deciding whether an open source software (OSS) project meets the required standards for adoption in terms of quality, maturity, activity of development and user support is not a straightforward process; it involves analysing various sources of information – including its source code repositories – to identify how actively the code is developed, which programming languages are used, how well the code is commented, whether there are unit tests etc. Additional information may be pertitent to the analysis, including that from communication channels such as newsgroups, forums and mailing lists to identify whether user questions are answered in a timely and satisfactory manner, to estimate the number of experts and users of the software, its bug tracking system to identify whether the software has many open bugs and at which rate bugs are fixed, and other relevant metadata such as the number of downloads, the license(s) under which it is made available, its release history etc. 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), 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 to be healthy, actively developed and adequately supported throughout the lifecycle of the software development project in which it is used, in order to identify and mitigate any risks emerging from a decline in the quality indicators of the project in a timely manner. Previous work in the field of OSS analysis and measurement has mainly concentrated on analysing the source code behind OSS software to calculate quality indicators and metrics.

OSSMETER extends the scope and effectiveness of OSS analysis and measurement with novel contributions on language-agnostic and language-specific methods for source code analysis, but also proposes using state-of-the-art Natural Language Processing (NLP) and text mining techniques such as question/answer extraction, sentiment analysis and thread clustering to analyse and integrate relevant information extracted from communication channels (newsgroups, forums, mailing lists), and bug tracking systems supporting OSS projects, in order to provide a more comprehensive picture of the quality indicators of OSS projects, and facilitate better evidence-based decision making and monitoring. OSSMETER also provides metamodels for capturing the meta-information relevant to OSS projects, and effective quality indicators, in a rigorous and consistent manner that enable direct comparison between OSS projects. These contributions are integrated in the form of an extensible cloud-based platform through which users can register, discover and compare OSS projects, but which can also be

extended in order to support quality analysis and monitoring of proprietary software development projects. To summarize the scientific and technological objectives achieved by OSSMETER are:

- comprehensive *domain modelling* for the domain of open source software development; identification and formal representation of the meta-information that needs to be captured in order to extract meaningful quality indicators for OSS projects;
- extraction of quality metrics by *analysing* aspects related to the *source code* and the development team behind an OSS project;
- extraction of quality metrics related to the *communication channels*, and *bug tracking facilities* of OSS projects using Natural Language Processing and text mining techniques;
- development of an *extensible cloud-based platform* that can monitor and incrementally analyse a large number of OSS projects, and a web-application to present their related quality metrics in an intuitive manner that aids decision making.

In the next sections such objectives are described. For each of them the progress beyond the state of the art is also discussed.

3 Domain Modeling and OSS project Lifecycle Analysis

State of the art: Modeling and abstracting open source software and its management have been the focus of a number of projects and research activities aiming at understanding the current practice in OSS projects e.g., for information and documentation purposes. The Qualipso project[10] analysed many OSS projects in order to identify typical roles (e.g., user, maintainer, and developer), information sources (e.g., help documents, release notes, and source code repositories), and their relations. Qualipso analysed also widely used forges (e.g., SourceForge, and Google Code) in order to identify services, which are typically provided to forge users, and the metadata which is used to describe and support OSS projects. This has been done since there is not a common agreement about the formats and metadata, which have to be used in the whole lifecycle of OSS projects. This hampers the definition of homogeneous treatments of projects maintained in different forges.

Other works (e.g., [9,10]) created abstract models of OSS projects in order to understand their architecture, and their evolution over time. In particular, [9] addresses the structural characteristics of OSS projects, explicitly the organization of the software's constituent components. In [10] the authors, by leveraging the "4+1" view model [17], and the four architectural views of software systems defined in [14], focus on the views which are closer to the work of OSS software developers, such as, for instance, the directory and the file level. The work in [8] proposes models and metrics to support the defect prediction for OSS projects. In particular, in addition to static code attributes for modeling software data in defect prediction, the authors introduce alternative metric sets, such as history and organizational metrics.

To improve both the quality and the trustworthiness perception of OSS products, [20] introduces the idea of certifying the testing process of an OSS system. In this respect,

[10] Qualipso: Leveraging Open Source for Boosting Industry Growth. `http://www.qualipso.org/`

the authors identify peculiar characteristics of OSS projects, that might influence the testing process. The work defines also a certification model that companies, developers, and final users can follow to evaluate the maturity level of an OSS testing process.
Innovation: According to the works previously outlined, the whole life-cycle of OSS projects can be analyzed by means of ad-hoc techniques specifically defined to retrieve heterogeneous information available from different sources in different formats. OSSMETER advances state-of-the-art techniques by providing the means to create models representing in a homogeneous manner different aspects of OSS projects in order to enable objective comparisons of OSS alternatives with respect to user needs, and quality requirements [22]. In particular, OSSMETER has developed:

- Metamodels for the specification of models representing the whole lifecycle of OSS projects. By considering and enhancing existing domain models, a set of EMF/Ecore[11] based metamodels and supporting tools have been conceived in order to enable the representations of OSS projects;
- Metamodels for OSS project metrics to enable automated measurement of open source software.

4 Source Code Quality and Activity Analysis

State of the art: Software metrics are a widely studied subject and are used in practice, for instance in the form of Function Points (FP) to measure the size of software (see International Function Point User Group, IFPUG[12]). Software metrics are widely used for the global analysis of productivity and quality of software [12,15]. All work on activity analysis is ultimately based on the original work of Lehman [18] who also coined the term *software evolution*. There is a wide range of tools available for performing specific analyses on source code as well as for computing various metrics. Regarding analyses, it is not easy to combine the results of different analyses and for metrics the same holds: the results produced by different tools are incomparable since they use different definitions for the underlying metrics. In addition, most of these tools are hand-coded and have to be reimplemented for different languages.
Innovation: OSSMETER provides an integrated view and corresponding tooling to do analyses, metrics calculations and activity analysis on several implementation languages. The main innovation are:

- Definition and of a coherent set of indicators for code quality and activity analysis. These indicators are usable across different implementation languages, different implementation platforms, and different version repository systems;
- Generation of the required tooling from declarative metrics descriptions using innovative model-driven/ generation-based techniques.

5 Communication Channel and Bug Tracking System Analysis

State of the art: Structuring and analysing textual data in forum, newsgroup and community-based question and answer threads is a newly emerging and complex problem

[11] Eclipse EMF: https://www.eclipse.org/modeling/emf/
[12] http://www.ifpug.org/

in text mining. Peer users are the cornerstone of managing software defects in OSS, due to their involvement in online forums [5]. Nevertheless, empirical studies regarding open source quality assurance activities and quality claims are rare [11]. OSS forums and bug-tracking systems concentrate vast amounts of knowledge generated daily about problems and their solutions as well as feedback to requests for OSS improvement.

Mining this textual data can match solutions to problems, evaluate solutions quality and impressively enhance user access to solutions and support [7,21,13]. Due to the size of this textual information, extracting, managing and evaluating it without manual intervention is a demanding, costly, impractical and probably impossible task. Text mining tools that automatically analyse, extract, summarise and assess information found in the threads of discussions on online forums are valuable for supporting OSS. Although text mining techniques have been used extensively in domains such as biomedicine [6], finance [19,16], competitive intelligence [23], very little work has been accomplished on applying text mining techniques for analysing threads of online forums.

Innovation: The target of this analysis is to extract from OSS forums and bug-tracking systems as many indicators about the characteristics and the quality of the communication that takes place as possible. Due to the complexity of the problem, a number of text mining technologies have been combined and structured in levels: after collecting online forum threads, the first level consists of identifying the types of each post as question, answer or supplementary text (context). In succession, posts are classified into more fine-grained categories and similarity-based methods are employed to identify chains of questions, contexts and answers within each thread, i.e. identify which answers and context correspond to which question. Thirdly, posts are analysed as far as sentiment and attitude is concerned. The output of this stage is a fundamental source of evidence useful for quality assessments. Finally, clustering together semantically similar threads and labelling the resulting clusters provides hints about the error-prone aspects of each OSS or its parts that need to be improved. The output of each level is two-fold: a number of indicators about the input posts quality that concerns the specific aspect that the corresponding component exploits; and also, supplementary output useful for the following components, but not necessarily part of the overall system output.

6 OSSMETER platform

State of the art: In the last decade several projects have provided platforms that support automated measurement of open source software including FLOSSMETRICS [1], Qualoss [3], SQO-OSS (Alitheia Core) [4] and Ohloh [2]. Also, many OSS forges such as SourceForge, Google Code and GitHub provide built-in annotation and measurement facilities for the OSS projects they host.

The aim of the FLOSS [13] project was to develop indicators of *non-monetary/trans-monetary* economic activity through a case study of OSS, and to assess OSS business models and best practices, and policy/regulatory impact. Its successor FLOSSMETRICS project [1] integrated a number of source code and bug tracking and mailing list extraction tools into a web-based platform which monitors a selection of open source projects and provides the extracted data in the form of SQL files which then need to be injected into a local database in order to be further analysed.

[13] http://www.flossproject.org/

Qualoss [3] aimed at automating the quality measurement of open source software. The Qualoss platform has been conceived to analyse two types of data: source code and project-repository information and does not appear to be measuring aspects related to communication channels or bug tracking systems of OSS projects.

Alitheia-Core [4] is a platform which aims at enabling software engineering research targeting OSS projects. Alitheia-Core provides support for processing source code repositories, emails from mailing lists and bug tracking systems through an API that developers can use in order to implement *metrics* and *experiments*. The design of Alitheia-Core is similar to the envisioned design of the OSSMETER platform, but the platform itself does not appear to be providing any implemented metrics related to mailing list and bug tracking systems.

Ohloh [2] is a free but proprietary and closed source system, only provided as a hosted service. Ohloh only analyses information related to the source code of OSS projects and does not take communication channels or bug tracking systems into consideration. However, it provides OSS project classification facilities (through user-defined tags), enables OSS project discovery and comparison, and presents source-code and activity-related metrics in an intuitive and understandable manner. On the downside, beyond not taking communication channels and bug tracking systems into consideration, being closed source means that organisations cannot run their own local instance of Ohloh through which they could monitor only the open source projects they are interested in, or their own proprietary projects. Also, as the system is proprietary, developers cannot extend it with features such as support for new metrics, access to additional sources of information, or integration with custom version control management systems.

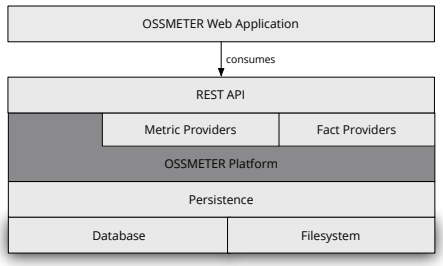

Fig. 1. OSSMETER system architecture

As mentioned above, OSS forges such as SourceForge, Google Code and GitHub provide built-in facilities for capturing additional information (metadata) about projects such as the category they belong to, the languages they are implemented in, relevant news feeds, and activity indicators such as user reviews, number of developers, and number of downloads. However, each OSS forge captures a different set of metadata and as such, projects hosted in different forges are not directly comparable. Moreover, none of these forges provides advanced source code, communication channel, and bug tracking system content analysis features such as those proposed by OSSMETER .

Innovation: The OSSMETER platform integrates and extends components and results produced by the projects discussed above in order to provide the comprehensive system shown in Fig. 1 for analysing and monitoring OSS projects. The novel features of the OSSMETER system are:

▷ a scalable and efficient data storage, which is responsible for storing and retrieving project specific metadata, and metric measurements. The use of local disk storage is

Projects	LOC	Age of Code (days)	# Developers	# Commits	Repository	URL
odoo	2,016,254	2,955	315	93,180	GitHub	https://github.com/odoo/odoo
Joomla	865,282	3,465	561	21,831	GitHub	https://github.com/joomla/joomla-cms
Drupal	609,987	4,680	107	16,450	GitHub	https://github.com/drupal/drupal
Ossmeter	537,343	349	7	1,796	GitHub	https://github.com/ossmeter/ossmeter
Assimp	310,235	2,555	77	2,610	GitHub	https://github.com/assimp/assimp
Libreplan	291,744	1,251	35	9,346	GitHub	https://github.com/Igalia/libreplan
BIMServer	224,000	1,716	15	2,775	GitHub	https://github.com/opensourceBIM/BIMserver
Hudson	223,033	3,345	73	1,476	Eclipse	https://projects.eclipse.org/projects/technology.hudson
Alitheia-Core	53,874	2,675	11	4,815	GitHub	https://github.com/istlab/Alitheia-Core
Epsilon	5,483,784	2,920	6	5,122	Eclipse	https://projects.eclipse.org/projects/modeling.epsilon
ATL	563,439	1,684	8	3,661	Eclipse	https://projects.eclipse.org/projects/modeling.mmt.atl

Table 1. List of projects used in the evaluation

also enabled to store temporary data required for the analysis, such as clones of source repositories.

▷ support for automated classification of OSS projects and discovery of related projects based on source code, communication channel and bug tracking system analysis through the use of advanced NLP and text mining techniques. To this end different kinds of measure components are provided, namely *fact providers*, *metric providers*, and *factoids*. Fact providers perform utility measurements and store factual data that can be consumed by other fact/metric providers. Metric providers optionally use computed facts to measure one or more project aspects and store the result in the database. Finally, factoids can aggregate heterogeneous metric providers into a four-star system.

▷ an extensible platform implemented using a plug-in based approach (OSGi), which is responsible for the integration of the various OSSMETER components, as well as for their scheduling, execution, and orchestration. The OSSMETER platform is also responsible of mining the OSS data, which are then passed to the various metrics providers for analysis.

▷ a REST API that enables software engineering researchers to access calculated quality indicators in order to perform additional analysis, and developers of 3rd party software to provide added-value services on top of the OSSMETER platform.

▷ a usable web-application developed on top of the platform that enables end-users to explore and compare OSS software in an intuitive manner. The presentation of the information about software projects can be fully customised at the user level and it is based on custom quality models.

7 Conclusion

OSSMETER is officially ended on March 31, 2015. When writing this document, the use case providers were performing the evaluation of the OSSMETER technologies by considering real OSS projects from different application domains. Some of the projects considered during the evaluation are shown in Table 1. These projects were chosen based on their characteristics, such as size, age, number of developers, and number of commits. The code of the OSSMETER platform is publicly available online at `https://github.com/ossmeter/ossmeter`. It is possible to download a locally-deployable version of the OSSMETER system that users can install locally – and if needed extend – in order to monitor a custom selection of OSS projects of interest and/or internal software development projects. By mentioning some facts updated at June 2015, the OSSMETER GitHub repository counted more than 650K lines of code, 1,800 commits, 3 branches, 8 releases, and 8 contributors. More than 30 technical deliverables were produced to present the technologies developed during the project. The official installation of OSSMETER is available at `www.ossmeter.com`.

References

1. FLOSSMETRICS: Free/Libre/Open Source Software Metrics. http://www.flossmetrics.org/.
2. Ohloh Project. http://www.sqo-oss.org/.
3. QUALOSS: Quality in Open Source Software. http://www.qualoss.org/.
4. SQO-OSS: Alitheia Core. http://www.sqo-oss.org/.
5. Faheem Ahmed, Piers Campbell, Ahmad Jaffar, and Luiz Fernando Capretz. Managing support requests in open source software project: The role of online forums. In *ICCSIT 2009*, pages 590–594. IEEE, August 2009.
6. Sophia Ananiadou and John Mcnaught. *Text Mining for Biology And Biomedicine*. Artech House, Inc., Norwood, MA, USA, 2005.
7. Timothy Baldwin, David Martinez, Richard B. Penman, Su N. Kim, Marco Lui, Li Wang, and Andrew MacKinlay. Intelligent linux information access by data mining: the ILIAD project. In *Procs. NAACL HLT 2010*, pages 15–16. Association for Computational Linguistics, 2010.
8. Bora Caglayan, Ayse Bener, and Stefan Koch. Merits of using repository metrics in defect prediction for open source projects. In *Procs. of FLOSS'09*, pages 31–36. IEEE Computer Society, 2009.
9. Andrea Capiluppi and Karl Beecher. Structural complexity and decay in floss systems: An inter-repository study. In *Procs. of CSMR '09*, pages 169–178, 2009.
10. Andrea Capiluppi, Cornelia Boldyreff, and Klaas-Jan Stol. Successful reuse of software components: A report from the open source perspective. In *Open Source Systems: Grounding Research - 7th IFIP WG 2.13 International Conference, OSS 2011*, pages 159–176, 2011.
11. Robert Glass. Is open source software more reliable? an elusive answer. *The Software Practitioner*, 11(6), 2001.
12. R. Grady. *Effective Software Measurements*. Prentice Hall, 1992.
13. Ahmed Hassan, Vahed Qazvinian, and Dragomir Radev. What's with the attitude?: identifying sentences with attitude in online discussions. In *Procs. EMNLP'10*, pages 1245–1255, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
14. Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley, 2000.
15. Capers Jones. *Applied Software Measurement: Global Analysis of Productivity and Quality, Third Edition*. McGraw Hill, 2008.
16. A. Kloptchenko, T. Eklund, J. Karlsson, B. Back, H. Vanharanta, and A. Visa. Combining data and text mining techniques for analysing financial reports: Research articles. *Int. Journ. of Intelligent Systems in Accounting, Finance and Management*, 12:29–41, January 2004.
17. Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(5):88–93, 1995.
18. M.M. Lehman. Programs, life cycles, and laws of software evolution. In *Proceedings IEEE*, volume 68, pages 1060–1976, 1980.
19. Hsin-Min Lu, Hsinchun Chen, Tsai-Jyh Chen, Mao-Wei Hung, and Shu-Hsing Li. Financial text mining: Supporting decision making using web 2.0 content. *IEEE Intelligent Systems*, pages 78–82, 2010.
20. Sandro Morasca, Davide Taibi, and Davide Tosi. Towards certifying the testing process of open-source software: New challenges or old methodologies? In *Procs. FLOSS'09*, pages 25–30, Washington, DC, USA, 2009. IEEE Computer Society.
21. Li Wang, Su N. Kim, and Timothy Baldwin. Thread-level analysis over technical user forum data. In *Procs. of the Australasian Language Technology Association Workshop 2010*, pages 27–31, Melbourne, Australia, December 2010.
22. James R. Williams, Davide Di Ruscio, Juri Di Rocco, and Dimitrios S. Kolovos. Models of OSS Project Meta-Information: A Dataset of Three Forges. In *MSR2014 at ICSE2014*, 2014.
23. Alessandro Zanasi. *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*. WIT Press, 2007.