OSSMETER: A Software Measurement Platform for Automatically Analysing Open Source Software Projects*

Davide Di Ruscio
University of L'Aquila, Italy
davide.diruscio@univaq.it

Dimitrios S. Kolovos
University of York, UK
dimitris.kolovos@york.ac.uk

Ioannis Korkontzelos
University of Manchester, UK
ioannis.korkontzelos@manchester.ac.uk

Nicholas Matragkas
University of York, UK
nicholas.matragkas@york.ac.uk

Jurgen J. Vinju
Centrum Wiskunde & Informatica, The Netherlands
Jurgen.Vinju@cwi.nl

ABSTRACT

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 as it involves exploring various sources of information. Such sources include OSS source code repositories, communication channels such as newsgroups, forums, and mailing lists, as well as issue tracking systems. OSSMETER is an extensible and scalable platform that can monitor and incrementally analyse a large number of OSS projects. The results of this analysis can be used to assess various aspects of OSS projects, and to directly compare different OSS projects with each other.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Measurement

Keywords

Open source software, Source code analysis, Text mining techniques

1. 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. Various sources of information have to be taken into account including source code repositories, communication channels, bug tracking systems, and other relevant metadata such as the number of downloads, the license(s) under which it is made available, its release history etc. Additional challenges have to be faced when one needs to discover and compare several OSS projects that offer software of similar functionality, and make an evidence-based decision on which one should be selected for the task at hand. 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.

Apart from source code repositories, online channels of communication between OSS users and developers, such as forums, newsgroups, mailing lists and bug trackers, can inform about the interest of users for an OSS project, the engagement of OSS developers with the users and the quality of user support. Online forums, newsgroups and mailing lists allow users to communicate with developers and with each other in order to ask questions, exchange views, report problems, make announcements and suggest enhancements related to the OSS product. Bug trackers allow users and developers to formally provide defect reports and enhancement requests for the OSS. An active and vibrant communication channel is essential for candidate adopters of an OSS product as this is often the main place where they can receive expert support - in contrast to commercial software where support is usually provided under a contract that prescribes a certain quality of service (QoS) level.

In this paper we present OSSMETER[1], a software measurement and analysis platform, which has been developed in the context of the EU project OSSMETER[2]. The platform is capable of tracking repositories associated with OSS projects and evaluate various quality and activity aspects. OSSMETER extends the scope and effectiveness of OSS

*This paper was partially supported by the EU OSSMETER FP7 STREP project (318736). The authors would like to acknowledge the valuable contributions and support of Prof. Richard Paige and Prof. Sophia Ananiadou to this work as Principal Investigators of the OSSMETER project.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2803186

[1]OSSMETER system: `https://github.com/ossmeter`
[2]OSSMETER EU Project: `http://www.ossmeter.eu`

analysis and measurement with novel contributions on source code analysis, Natural Language Processing (NLP) and text mining techniques. OSSMETER has been developed as 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. The OSSMETER system can be a highly valuable supporting tool for:

- **Developers and Project Managers** who are responsible for deciding on the adoption of OSS, as it will enable them to make decisions on hard facts and uniform quality indicators;
- **Developers of OSS** as it will enable them to monitor the quality of the OSS projects they contribute to, promote the OSS they contribute to using independently-calculated and trustworthy quality indicators, and identify related projects for establishing synergies with;
- **Funding Bodies** that are funding ICT projects which produce OSS, as it will allow them to monitor the quality and assess the impact of the produced software even after the end of the projects.

In the following, we present the OSSMETER system and its constituent components.

2. OSSMETER

With the goal of measuring and monitoring numerous information sources of a vast number of OSS projects, the OSSMETER platform needs to be both scalable and extensible. It needs to interface not only with the most widely used issue-tracking systems, version control systems, and user support systems, but also with the most widely-used open source hosting forges (e.g. SourceForge, Google Code, Eclipse, Apache, Github). Moreover, the OSSMETER system needs to be able to support different types of analysis and measurement components. Such components should be able to perform different types of analysis such as source code analysis and natural language processing (e.g. sentiment analysis and thread classification). In addition to extensibility, the OSSMETER system needs to be scalable in order to support the automated analysis of an arbitrary number of OSS projects of arbitrary size.

To support the aforementioned requirements of scalability and extensibility, the architecture illustrated in Figure 1 is proposed. Every component of this architecture is designed with these requirements in mind. In the following sections, we will briefly present the main building blocks of this architecture.

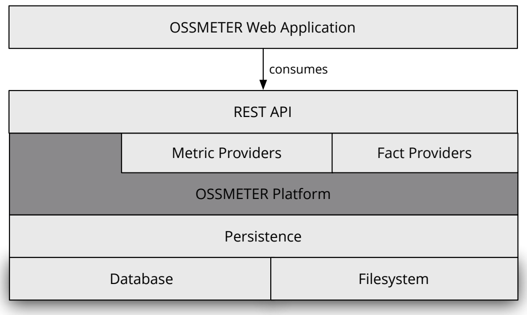

Figure 1: OSSMETER system architecture

2.1 Data Storage and Persistence

The OSSMETER system has to deal with various types of structured and unstructured data, such as source code models and text. Moreover, the system has to deal with high volume data, since there are open source projects which consist of millions of lines of code, and are developed by thousands of contributors. For example, Mozilla Firefox consists of 9,577,883 lines of code, which are developed by 2,492 contributors in 156,467 commits[3]. Monitoring such projects can be a challenging task, especially if one considers the fact that multiple projects will have to be monitored simultaneously.

Due to the nature of the data the OSSMETER platform has to deal with, a scalable and efficient data storage has to be used in order to avoid any performance bottlenecks. The data storage is responsible for storing and retrieving project-specific metadata (e.g., that which was extracted from the hosting forge), and the metric measurements. The database technology we are using is MongoDB[4]. Such a database is scalable, and schemaless supporting the system architecture. Finally, we use local disk storage in order to store temporary data required for the analysis, such as clones of source repositories.

Atop the data storage layer lies the data persistence layer which is responsible for raising the level of abstraction provided by the MongoDB driver. To this end we have developed an object mapping framework for the MongoDB database called Pongo[5]. Pongo is a template-based POJO generator for MongoDB which enables developers to specify a data model in a high level modeling language and to generate strongly-typed Java classes for interacting with the database.

2.2 OSSMETER Platform

The central component of the OSSMETER system is the OSSMETER platform, which is responsible for the integration of the various OSSMETER components, as well as for their scheduling, execution, and orchestration. Another major responsibility of the OSSMETER platform is the mining of the OSS data, which are then passed to the various metrics providers for analysis. To provide this extensibility requirement, the OSSMETER platform is implemented using a plug-in based approach (OSGi). A set of standard platform *extension points* are defined which plug-ins can implement to register services with the platform (such as a new issue-tracking system connector or a new metric). OSGi enables these services to be stopped, started and updated at runtime without stopping the platform.

2.2.1 Incremental Analysis

As the platform supports an arbitrary number of measurement components, each of which require access to one or more of the project information sources (e.g., version control systems, and issue tracking systems), the platform provides a number of information managers. These are modules responsible for connecting to the remote information repositories, and computing a delta of the artefacts that have changed since the last analysis. These deltas are then presented to the measurement components to enable them to perform their analysis without duplicating work. Managers

[3]As measured on the 8th of October 2013
[4]`http://www.mongodb.org`
[5]`https://code.google.com/p/pongo/`

are scheduled to compute deltas for every day of the OSS project's existence. This enables OSSMETER to compute fine-grained analyses of the evolution of each metric, whilst also potentially correlating these analyses to days of the week or time of the year (which has been shown to affect the health of a project [7]).

A major goal of OSSMETER is to avoid storing any data locally whenever this is possible. For example, due to the nature of distributed version control systems such as Git, data from such repositories need to be stored locally. On the other hand, messages from NNTP newsgroups do not need to be stored locally, since their analysis can be performed in an incremental manner over the network. Therefore the daily deltas calculated by the managers enable the analysis to focus on small, incremental chunks of data, and therefore the disk storage requirements of the system are minimised, and the performance is optimised. Daily deltas is a novel feature of OSSMETER in comparison to other similar platforms such as Alitheia Core [5].

The OSSMETER system provides managers for the main OSS information sources:

- **Version control systems**: SVN, Git
- **Issue-tracking systems**: Bugzilla, Bitbucket, GitHub Issues, Jira, Redmine, SourceForge
- **Communication channels**: NNTP Newsgroups, SourceForge Forums

Moreover, due to the OSGi-based architecture of the system additional managers can be implemented and integrated with the system with minimal effort.

2.2.2 *Scalability*

The platform needs to scale to support the monitoring of an arbitrary number of OSS projects. To achieve this, we have developed the platform to scale horizontally at both the data and logic layers (Figure 1). Firstly, our choice of data store, MongoDB, is designed to be horizontally scaled through the use of sharding. Secondly, the platform itself can be horizontally scaled through the use of distributed OSGi. New nodes running the OSSMETER platform can be added to the network and register themselves as slave nodes. As each OSS project is treated independently from all others, the scheduler can delegate the analysis of entire projects to any registered node on the network. At the node-level, we optimise the execution by constructing a dependency tree of the different measurement components, and by executing each branch of this tree in a parallel fashion.

2.3 Measurement Components

The architectural layer atop the layer of the OSSMETER platform hosts the platform's measurement components. In the context of OSSMETER, these components are responsible for measuring different aspects of monitored OSS projects. Conceptually the OSSMETER system supports three different types of measurement components: *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.

Currently, OSSMETER provides more than 300 providers. These providers include language-specific and language-agnostic measurement components for source code (e.g., number of committed changes, code churn, cyclomatic complexity, etc.), issue tracking systems components (e.g., average time between creating and closing a bug, number of open bugs, length of discussion for a bug, etc.), and communication channel measurement components (e.g., average response time, number of new messages, number of active users). In addition to the existing providers, new providers can be integrated with the platform with minimal effort by using the provided OSGi extension points.

2.4 Open API

On top of the platform we provide an open Application Programming Interface (API) that enables researchers and developers of 3rd- party applications to access the data persisted by the system. The REST API of the OSSMETER platform[6] is developed using the Restlet framework[7]. The API presents three types of data: project metadata, project metric data, and project metric data summaries, intended for visualisation.

2.5 Web Application

To make the results of the platform immediately accessible, we have designed and implemented a web application[8] that consumes the REST API and enables users to register and discover OSS projects and explore their quality indicators in an intuitive manner to aid decision making. Additionally, users can receive notifications when projects of interest slip below (or above) selected thresholds on the set of metrics that are most important to them. This can, for example, help users plan migrations to a new OSS project if an existing one is discontinued. Finally, the presentation of the information about software projects can be fully customised at the user level and it is based on custom quality models.

3. OSSMETER EVALUATION AND PERFORMANCE

So far we have analysed a number of OSS projects during the evaluation period of the EU OSSMETER project. Several evaluators, both internal and external to the project consortium, have been involved in the evaluation phase including small and medium enterprises like Tecnalia[9], Softeam[10], and UNPARALLEL Innovation[11], and large multinationals like Airbus, and Thales. A sample of the analysed projects is listed in Table 1. During the analysis of those projects we have performed a number of experiments in order to assess the performance of the OSSMETER system. These experiments have shown that on average it takes ≈10 seconds to connect remotely to an information source and create a delta which is then passed to the measurement components. Once the measurement components obtain this delta it takes them on average 20 milliseconds to complete their measurement and store it to the database. For example, Figure 2 illustrates the average time (in milliseconds) of all metrics per day for the Drupal[12] project.

[6]`http://www.ossmeter.com/api/`
[7]`http://restlet.com/`
[8]`http://www.ossmeter.com`
[9]`http://www.tecnalia.com/`
[10]`http://www.softeam.com/`
[11]`http://www.unparallel.pt/`
[12]`https://www.drupal.org/`

Table 1: Some of the projects used in the evaluation

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`

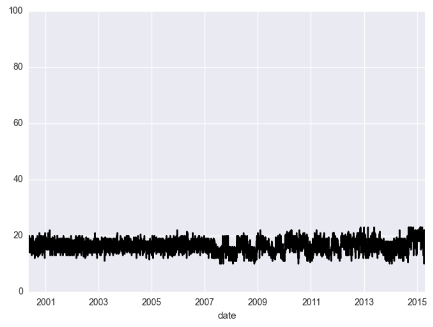

Figure 2: Average metric provider timing for Drupal in milliseconds.

Moreover we evaluated how scalable OSSMETER is in terms of disk storage. Our results show that for the largest project in terms of activity from the list in Table 1, namely Odoo, the data obtained for this project is ≈4GB. These data correspond to the measurements of 315 metric providers for the entire life of the project. Although the results indicated that we need considerable amount of disk space, we observed that measurements of 4 metric providers correspond to more than half or the required disk space. These metrics are responsible for very fine grained analysis such as keeping track of every time a developer has performed an action on a file. In a scenario where disk space is a system constraint, the modular architecture of OSSMETER permits the deactivation of such costly metric providers without affecting the execution of the rest of the system.

4. RELATED WORK

Several projects have provided platforms that support automated measurement of open source software in the last decade. In our view, the most significant of these projects of this kind are Flossmetrics [4], Qualoss [8], SQO-OSS (Alitheia Core) [5] and Ohloh [1]. Also, many OSS forges such as SourceForge, Google Code and GitHub provide built-in annotation and measurement facilities for the OSS projects they host. OSSMETER differentiates itself from these by addressing the entire community around an OSS project, rather than focusing on a single aspect (such as source code). OSSMETER's novel daily deltas provide a fine-grained view of the evolution of an OSS project's health indicators. The open API of OSSMETER enables third party developers to benefit from our data analysis efforts and, optionally, contribute ther results back to the community. Furthermore, OSSMETER facilitates long-term monitoring of OSS projects, by issuing notifications when quality indicators fall below a user-defined level.

There are many existing metrics that are used to evaluate the quality of source code [3], bug tracking systems [2], and newsgroups [6]. OSSMETER will support these metrics whilst also extending the state-of-the-art in programming language-agnostic and language-specific analysis methods, and applying advanced NLP and text-mining techniques to OSS information sources.

5. CONCLUSIONS

This paper has presented OSSMETER, an extensible system for monitoring the health of OSS projects. Its architecture and design enables the platform to be extensible and scalable. Extensibility is achieved in multiple ways. First, since we are using a schemaless database, the platform supports heterogeneous data models, which can be tailored according to the needs of different metric providers. Moreover, the use of OSGi enables the integration with the platform of an arbitrary number of information managers and metric providers. Scalability at the data level is supported by the choice of an appropriate database technology, namely MongoDB, which supports sharding. Furthermore, the platform can be deployed in a distributed manner across multiple computational nodes. In addition computation at the node-level can be performed in a parallel manner.

6. REFERENCES

[1] Ohloh Project. http://www.ohloh.net/.

[2] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, and W. Liu. Search-based duplicate defect detection: An industrial experience. In *Proc. MSR'13*, pages 173–182, 2013.

[3] N.E. Fenton and S.L. Pfleeger. *Software Metrics.* Pws Pub Co, 1996.

[4] J. M. Gonzalez-Barahona, G. Robles, and S. Dueñas. Collecting data about floss development: the flossmetrics experience. In *Procs of FLOSS2010*, pages 29–34. ACM, 2010.

[5] G. Gousios and D. Spinellis. Alitheia core: An extensible software quality monitoring platform. In *Procs of ICSE 2009*, pages 579–582. IEEE Computer Society, 2009.

[6] B. Lundell J. Gamalielsson and B. Lings. Responsiveness as a measure for assessing the health of OSS ecosystems. In *Proc. OSCOMM'10*, 2010.

[7] J. Sliwerski, T. Zimmermann, and A. Zeller. Don't program on fridays! how to locate fix-inducing changes. In *Proc. WSR'05*, May 2005.

[8] M. Soto and M. Ciolkowski. The qualoss open source assessment model measuring the performance of open source communities. In *Procs of ESEM 2009*, pages 498–501. IEEE Computer Society, 2009.