Models of OSS Project Meta-Information: A Dataset of Three Forges[*]

James R. Williams
Department of Computer Science
University of York
York, United Kingdom
james.r.williams@york.ac.uk

Davide Di Ruscio
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Universitá degli Studi di L'Aquila
L'Aquila, Italy
davide.diruscio@univaq.it

Nicholas Matragkas
Department of Computer Science
University of York
York, United Kingdom
nicholas.matragkas@york.ac.uk

Juri Di Rocco
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Universitá degli Studi di L'Aquila
L'Aquila, Italy
juri.dirocco@univaq.it

Dimitrios S. Kolovos
Department of Computer Science
University of York
York, United Kingdom
dimitris.kolovos@york.ac.uk

ABSTRACT

The process of selecting open-source software (OSS) for adoption is not straightforward as it involves exploring various sources of information to determine the quality, maturity, activity, and user support of each project. In the context of the *OSSMETER* project, we have developed a forge-agnostic *metamodel* that captures the meta-information common to all OSS projects. We specialise this metamodel for popular OSS forges in order to capture forge-specific meta-information. In this paper we present a dataset conforming to these metamodels for over 500,000 OSS projects hosted on three popular OSS forges: Eclipse, SourceForge, and GitHub. The dataset enables different kinds of automatic analysis and supports objective comparisons of cross-forge OSS alternatives with respect to a user's needs and quality requirements.

Categories and Subject Descriptors

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

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

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597132

General Terms

Theory, Measurement

Keywords

Data mining

1. INTRODUCTION

The creation and adoption of open-source software (OSS) has been increasing in the last few years [8]. The task of selecting appropriate OSS for adoption, however, is not straightforward as it involves exploring numerous, disparate sources of information to determine whether the project meets the user's needs and quality requirements. Unlike commercial software, which is typically developed within the context of a particular organisation with a well-established business plan and commitment to the maintenance, documentation and support of the software, OSS is very often developed in a public, collaborative, and loosely-coordinated manner. This has several implications on the level of quality of different OSS software as well as on the level of support that different OSS communities provide to users of the software they produce. On the one hand, there exist several high-quality and mature OSS projects which deliver stable and well-documented products. Such projects typically also foster a vibrant expert and user community which provides remarkable levels of support both in answering user questions and in repairing reported defects (bugs) in the provided software [2]. On the other hand, there is also a substantial number of OSS projects that have inactive development teams, have little user documentation, have a small inactive community, or have been abandoned entirely [7].

As part of the *OSSMETER* project[1], we are developing a

[1]`www.ossmeter.eu`

monitoring and analysis platform that enables OSS projects to be compared on a large number of metrics relating to all aspects of OSS. This includes metrics for analysing the source code of the project, the communication channels (e.g. mailing lists, newsgroups) with which users and developers communicate, and the bug tracking system(s) of the project. In this paper we present a dataset consisting of the forge-related meta-information of OSS projects. This dataset enables this forge-related meta-information to be automatically analysed in a forge-independent and forge-specific manner. Furthermore, it also enables the OSS projects from different forges to be compared objectively. The dataset consists of all 233 projects currently hosted by the Eclipse Foundation, nearly 90,000 projects hosted on SourceForge, and nearly 470,000 projects hosted on GitHub. The dataset is available at:
`https://github.com/ossmeter/msr14-showcase-forges`.

Section 2 presents our domain analysis of OSS projects and forges which led to the development of *metamodels* that capture the meta-information of OSS forges. Section 3 describes how these metamodels represent the schemata for a database that stores this meta-information in a homogeneous manner. Section 4 summarises how we have populated the database for three popular OSS forges, whilst section 5 describes the challenges and limitations of doing so. Section 6 presents related work.

2. DOMAIN ANALYSIS OF OSS PROJECTS

Model-Driven Engineering (MDE) [6] is an engineering approach that treats models as first-class development artefacts. Through a series of automatic transformations, high-level models can be used to generate the final production system. Focusing development on models allows them to remain useful and up-to-date, but also allows engineers to develop systems at the level of the application domain.

The specification of precise models that can be automatically analysed and queried requires a precise definition of the abstract syntax of those models and the inter-relationships between model elements. *Metamodelling* is the process of defining a collection of concepts and their relationships that capture a certain domain and expressing them in the form of a *metamodel* [6]. Metamodels, therefore, prescribe the structure and semantics of any models for the domain.

Utilising metamodels to capture different aspects of OSS projects, in particular the meta-information from the hosting forge, enables these projects to be represented in a homogeneous manner. A standardised representation of an OSS project's forge-related meta-information enables the unified analysis of usually-disparate OSS projects. For example, one can analyse the size of teams and their roles across forges, or calculate descriptive statistics such as the proportion of projects with multiple source code repositories. Moreover, it enables cross-forge comparisons, allowing users to contrast OSS alternatives with respect to their quality requirements.

To develop metamodels of OSS forges, we performed an iterative analysis of OSS projects: specifying the models of real OSS projects, and refining their metamodels as new constructs were encountered. Each OSS project shares a common set of information, which we capture independent of the forge-specific information. We now present the metamodel that expresses the common features of all OSS projects, and also describe a specialisation of this metamodel for the Eclipse OSS forge.

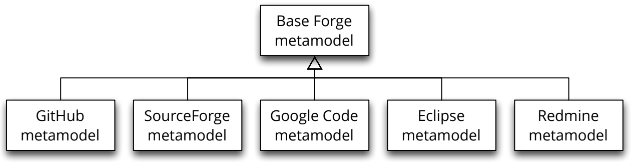

Figure 1: The forge metamodel hierarchy: the common meta-data is captured in a forge-agnostic metamodel, which is specialised for each OSS forge.

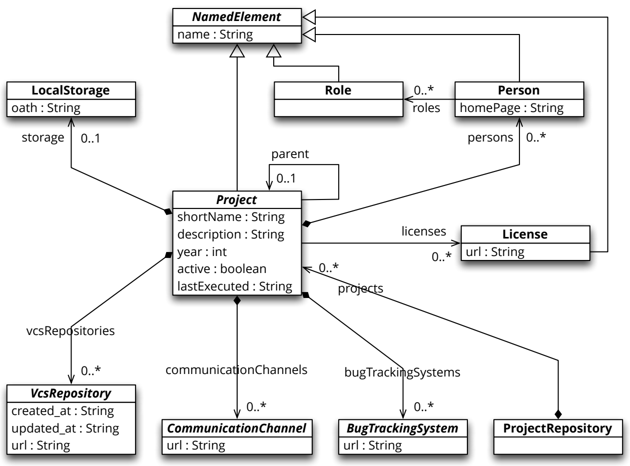

Figure 2: The metamodel capturing concepts common between OSS forges.

2.1 The Common Forge Metamodel

There is a set of concepts related to OSS projects that are independent from the forge that hosts the project. This set includes: information related to the project's version control system(s), communication channels (e.g. forums, mailing lists), and bug tracking system(s), as well as the licenses of the project, and information related to the people who contribute in some manner to the project.

We capture this information in a forge-agnostic metamodel; each OSS forge then specialises the metamodel to include any extra meta-information it provides, in a similar way to which object-oriented code can be specialised. Figure 1 illustrates the metamodel hierarchy, whilst the forge-agnostic metamodel, developed by analysing a number of forges, is shown in figure 2. We now briefly describe the specialisation for the Eclipse forge. Specialisations and descriptions for other forges, including GitHub, SourceForge and Google Code can be found in [5].

2.2 Eclipse Forge Specialisation

The Eclipse Foundation[2] provide a hosting system for projects that contribute to the Eclipse integrated development environment. Unlike forges such as SourceForge or GitHub, Eclipse prescribes a rigorous set of requirements that a project must meet in order to be hosted in their forge. Furthermore, new projects are initially entered into an *incubation* period until they meet the standards to be accepted as a full Eclipse project.

[2]`www.eclipse.org`

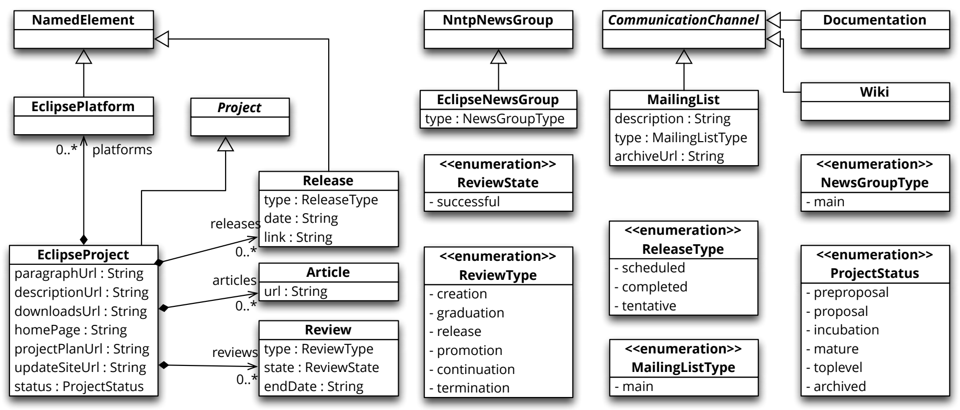

Figure 3: The specialisation of the common metamodel for the Eclipse forge.

Figure 3 presents our specialisation of the forge-agnostic metamodel for the Eclipse forge. The specialisation adds extra meta-information – such as a project description, downloads, and the project's homepage on the Eclipse website – as well as information related to the reviewing process of the project, any related documentation articles, and its list of releases. Furthermore, contributors to a project are assigned *rôles*, namely *mentor*, *committer*, and *leader*.

3. THE DATABASE SCHEMA

The forge metamodel definitions in the previous section are used to define the database schema. If a relational database was chosen to store the forge meta-data, one could use an *object-relational mapping* from each class in the metamodel to tables in the database. However, forge-specific information would need to be stored in separate tables. As the forge-specific metamodels all inherit from the same forge-agnostic metamodel, we can store all projects together, no matter which forge they belong to, by using a schema-less database.

For this, we use MongoDB as the database, and use *Pongo*[3] as a means of defining the schema. Pongo is a template-based POJO generator for MongoDB, which is built atop the MongoDB Java driver[4]. With Pongo, an engineer can define a model of the data to be stored using a textual modelling language called *Emfatic*[5]. Pongo then uses this model to generate strongly-typed Java classes, which can be used to work with the database at a more convenient level of abstraction. For each class in the data model, a POJO class is created that extends the core `Pongo` class and provides support for querying the database in an intuitive manner. Any changes made to objects are cached until a *synchronise* method is invoked.

The four metamodels discussed in the previous section – forge-agnostic, GitHub, SourceForge, and Eclipse – have all been specified in Emfatic and we have used Pongo to generate the appropriate Java code. The next section describes the modules that we have developed to import projects from these three OSS forges. Each importer uses the Pongo-generated classes to create new entries in the database.

[3]`http://code.google.com/p/pongo/`
[4]`http://docs.mongodb.org/ecosystem/drivers/java/`
[5]`http://www.eclipse.org/epsilon/doc/articles/emfatic/`

4. DATA COLLECTION PROCESS

In this section we summarize how the data has been collected from the different forges. Importers have been developed by taking into account the specificities of each forge. The importers have been developed in Java, and conceptually each importer consists of a corresponding Java class implementing two methods: `importAll`, and `importProject`. The former is the entry point of the importer and it retrieves the list of every project hosted by the considered forge. For each project, the `importProject` method is executed and it uses the Pongo-generated code to create and save the project information using the retrieved meta-data. All projects, no matter their forge, are stored in the same MongoDB collection. In the following, the `importAll` and `importProject` methods of each developed importer are summarized.

4.1 Eclipse forge

The Eclipse forge data has been collected by exploiting a public API that permits the gathering of project meta-data from a JSON representation of them. The `importAll` method retrieves the list of all the hosted Eclipse projects. The details of each project are then retrieved by means of the method `importProject`, identifying projects by a unique *projectId*. However, not all of the information of a given Eclipse project is available via this API. For instance, the committers and the list of supported platforms of a given project are available only from the Web pages of the project. This has required the development of specific HTML parsers for retrieving the meta-data not available via the API but is necessary to fully represent an Eclipse project.

4.2 GitHub

GitHub provides users with complete and advanced APIs that enable the retrieval of all the information of the hosted projects. This allowed us to develop the `importAll` and `importProject` methods without any workarounds, such as HTML parsing. Unfortunately, there is not a way to directly retrieve the list of all the hosted projects. At the link `https://api.github.com/repositories?since=id` it is possible to retrieve 100 projects having identifiers greater than *id*. Thus the `importAll` method iterates on the value of the attribute *id*, and for each value executes the `importProject` method on sets of 100 projects. The meta-data of a given project is available at `https://api.github.com/repos/projectId`, where *projectId* is a string identifying the considered project.

4.3 SourceForge

Similarly to the previous forges, SourceForge provides APIs for retrieving information of the hosted projects. In particular, given a *projectId*, the corresponding meta-data represented in a JSON document is available at `http://sourceforge.net/api/project/name/projectId`. Unfortunately, the API also does not provide the means to retrieve the list of every project hosted by SourceForge. A workaround we have implemented to solve this problem exploits the Web interface of the SourceForge projects directory. In particular, at the link `http://sourceforge.net/directory/?page=id` it is possible to retrieve a list of 25 projects where *id* is an integer identifying a directory page. The `importAll` method iterates on the identifier *id*, and for each value it parses the corresponding HTML page in order to retrieve the list of 25 project identifiers to be used for executing the `importProject` method.

5. CHALLENGES AND LIMITATIONS

The development of the importers summarized in the previous section has demanded some efforts to deal with a number of challenges. In particular, there are two main cases that we had to manage with respect to the different ways that each forge provides for accessing its data:

- Forges like GitHub provide open APIs or Web Services that can be used to extract the meta-data in a structured way;
- Forges like SourceForge and Eclipse require parsing HTML pages in order to retrieve meta-data that is not available in the exported APIs.

The development of the different importers had to take into account the APIs restrictions that some of the forges apply. In particular, the rate limit for the GitHub APIs is 5,000 requests per hour for authenticated users, or 60 requests per hour for unauthenticated users. As such, our GitHub importer will enter an idle state once the rate limit is reached, and will automatically restart when possible. Due to network problems that may occur during the importing process, we have implemented mechanisms that permit the importers to restart from where they left off. Without such a mechanism, in case of network disconnections, importers have to be re-executed from scratch, losing the meta-data of the projects already imported.

The main limitations of the proposed approach are related to the time required to collect data. For instance, importing the meta-data of 400,000 GitHub projects can take more than two days. This might raise some problems in case of keeping the dataset up to date with the latest information on the forges.

6. RELATED WORK

In [3] the authors presents *GHTorrent*, a tool suite for monitoring the GitHub public event time line. In particular, *GHTorrent* uses the GitHub API to collect data and extract, archive and share queryable meta-data. In this paper we go further by proposing a collection of interrelated metamodels that allow OSS projects to be represented in a homogeneous manner, even though these projects exist in heterogeneous sources (i.e. not limited to GitHub).

FLOSSMole [4] is a similar initiative to OSSMETER; it aims to collect and freely redistribute in different formats the data of open source software. Differently from OSSMETER, however, the FLOSSMole project does not provide the instruments to analyse data, that are simply collected and made publicly available.

In [1] the authors exploit the facilities of FAMIX[6] to integrate information retrieved from different sources, e.g. version control systems, bug reports and source code. We share the idea of having a single model to represent concepts retrieved from different sources, however in the metamodels discussed in this paper we focus on project meta-data. The information we capture (e.g. SVN and Bugzilla URLs) can be used by further tools to access and analyse those parts of the project. Consequently, as opposed to [1], we do not store fine grained information like a specific line of source code, or a particular thread in the bug tracking system.

7. CONCLUSION

In this paper we have presented a dataset containing the forge-related meta-information of over 500,000 projects. The dataset enables cross-forge comparisons of OSS projects, as well as provides the basis for the OSSMETER project to apply standardised quality metrics to any OSS project, independent of where it is hosted.

8. REFERENCES

[1] G. Antonio, M. Di Penta, H. Gall, and M. Pinzger. Towards the integration of versioning systems, bug reports and source code meta-models. In *Procs of Workshop on Software Evolution Through Transformation (SETra 2004)*, pages 83–94, Amsterdam, 2004. Elsevier.

[2] K. Crowston, H. Annabi, and J. Howison. Defining open source software project success. In *Procs of the 24th International Conference on Information Systems (ICIS 2003)*, pages 327–340, 2003.

[3] G. Gousios. The ghtorrent dataset and tool suite. In *Procs of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[4] J. Howison, M. Conklin, and K. Crowston. Flossmole: A collaborative repository for floss research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, 2006.

[5] U. of L'Aquila. D2.1 - domain analysis of OSS projects. Technical report, University of L'Aquila, 2013. Available at http://www.ossmeter.org/publications.

[6] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

[7] C. M. Schweik, R. English, Q. Paienjton, and S. Haire. Success and abandonment in open source commons: Selected findings from an empirical study of sourceforge.net projects. In *Procs of the 2nd Workshop on Building Sustainable Open Source Communities (OSSCOMM 2010)*, pages 91–101, 2010.

[8] D. Spinellis and V. Giannikas. Organizational adoption of open source software. *Journal of Systems and Software*, 85(3):666–682, Mar. 2012.

[6]`http://www.moosetechnology.org/docs/famix`