A Framework for Analysing and Visualising Open Source Software Ecosystems

Mathieu Goeminne  
Software Engineering Lab  
University of Mons - UMONS  
Place du Parc 20, Mons, Belgium  
mathieu.goeminne@umons.ac.be

Tom Mens  
Software Engineering Lab  
University of Mons - UMONS  
Place du Parc 20, Mons, Belgium  
tom.mens@umons.ac.be

ABSTRACT

Nowadays, most empirical studies in open source software evolution are based on the analysis of program code alone. In order to get a better understanding of how software evolves over time, many more entities that are part of the software ecosystem need to be taken into account. We present a general framework to automate the analysis of the evolution of software ecosystems. The framework incorporates a database that stores all relevant information obtained thanks to several mining tools, and provides a unified data source to visualisation tools. One such visualisation tool is integrated in order to get a first quick overview of the evolution of different aspects of the software project under study. The framework is extensible in order to accommodate more and different types of input and output, depending on the needs of the user. The framework is extensible in order to accommodate more and different types of input and output, depending on the needs of the user. We compare our framework against existing solutions, and show how we can use this framework for carrying out concrete ecosystem evolution experiments.

Categories and Subject Descriptors

D.2.8 [Software Engineering]: Metrics—product metrics, process metrics; K.6.3 [Management of Computing and Information Systems]: Software Management—software maintenance

General Terms

Experimentation, Human Factors, Measurement

1. INTRODUCTION

Traditionally, most software studies rely only on the software source code to analyse and predict how the software evolves [2, 6]. We need to focus on more elements to get a full picture of the software evolution. In particular, the human aspect plays a significant role in how and why the software evolves over time. The communication among developers on the one hand and between developers and users on the other hand is decisive in the way in which they transmit their requirements and report observed issues. To ensure the success of a software development, all forms of human interaction (that is, the software ecosystem) should be taken into account [3, 4, 11, 14].

We define the ecosystem as the source code together with the user and developer communities surrounding the software. Alternative definitions of software ecosystems exist, but are outside the scope of this paper. Lungu [9] defines it as a set of software projects for which some people are involved in both of them.

The novelty of our approach is to complement source code information with knowledge about the community (consisting of developers and users) that surrounds it. Firstly, we are interested in the effect that a community of communicating and collaborating software developers has on the (evolution of) software quality and vice versa. Secondly, we want to study the relation between the popularity of an open source software (OSS) project and its quality [12].

To realise our goals, we developed a generic and extensible framework enabling the empirical study, analysis, visualisation and comparison of OSS projects. An OSS project typically provides all data related to its evolution, thanks to publicly available tools, such as a version control tool, one or more mailing list(s) and a bug tracker. Using the knowledge acquired during our empirical studies, we expect to reach the following medium-term goals with our framework:

- offer better support for the software development process, by providing concrete suggestions to developers on how to improve communication and collaboration within the development team;
- offer support to end-users, by providing recommendations to help them to choose the ‘best’ OSS, according to their goals and constraints;
- improve the OSS quality, by taking into account not only the source code data but also all the collected ecosystem information;
- provide reliable information to OSS developers about how their software is used, in order to allow them to understand what are the key features to develop or improve;
- provide a better insight to researchers in how software evolves and what are the major factors (both technical and non technical) affecting it.
2. EXISTING WORK

Even if most studies focus only on the source code, some research teams try to broaden the research field by analysing communities surrounding OSS projects. Mockus et al. [10] made a comparative study of Apache and Mozilla using both the source code repository and the mailing list of these projects. They highlight that there is a set of implicit conventions among developers that implies an intensive communication. Because the communication is not scalable (one cannot linearly increase the communication intensity without adding more human resources), a strategy is needed to restrain the number and the size of communications. Apache seems to have a very efficient approach that consists of a minimal server core with a well-defined interface.

Stephany [15] suggests Maispion, a tool able to display several source code and mail metrics from a restrained set of sources. Abreu and Premraj [1] studied the correlation between developer communication and software quality. They showed a statistically significant correlation between communication frequency and number of injected bugs in the software. The Libresoft team [13, 14] analyses OSS evolution and correlations between software quality and developer communities. They highlight the interest to study not only source code, but also the surrounding ecosystem.

To get a better insight into how the ecosystem surrounding an OSS project affects its evolution, researchers have implemented dedicated tools. Generally these are tailored tools, designed for one or a few scientific experiments. Sometimes a more generic and reusable tool is produced, but the views about the software it produces are too static or too specific to highlight the interaction between the evolution of software quality and the evolution of the ecosystem. Our framework and our tool are designed to bridge this gap.

3. OUR FRAMEWORK

3.1 Overview

To be able to gather data from several sources, we created a generic multi-layer framework presented in Figure 1. The framework provides tools for each kind of entity involved in the software development:

1. A set of project databases represents all the software data sources. We can distinguish between the source code repository containing all versions of the source code, the bug tracker containing all feature requests and problem reports as well as all the resolution process, and the mailing list(s) containing all the mails exchanged among developers and between users and developers;

2. The mining layer contains tools to extract useful data (and metadata) from the project databases. For instance, a mailing list contains e-mails and a source code repository contains commits, which are pieces of atomic changes done on the source code;

3. The extracted information is analysed by the analysis layer. The analysis allows to compute more complex metrics based on the extracted information.

4. All the obtained metrics are stored in a persistent database. The application layer contains applications able to consult this database and to present relevant information under the form of statistics, graphical outputs, wizards, guidelines, reports and so on. These tools offer a means to reach our goals. They are able to present metrics about the software quality, to summarise the collected information, to compare ecosystems against each other, to automatically highlight evolution trends, communication patterns, and so on. The possibilities are endless and may be adapted to the studied behaviours.

Figure 1: Framework for extracting, computing, collecting and storing ecosystem metrics.

Because each layer realises a specific task and operates on various types of data, we need to use specific tools adapted to each possible context. For example, the framework needs tools to extract artefacts from each kind of source code repository, tools to extract metrics from source files written in different programming languages, and so on. To achieve this, the framework is a mixture of home-brewed, free and commercial tools to reuse as much as possible existing work. Unfortunately, these tools sometimes can have different definitions of a same metric. This makes it hard to directly compare two software projects analysed by different tools. The framework therefore offers a glue to present all the data in a uniform way. Because tools may use different definitions...
of computed metrics, the framework has to deal with inconsistencies and redundancies. Our approach is to reconcile the divergent results by taking into account the specificity of used tools. If they provide unreconcilable results, the inconsistency must be highlighted to offer the opportunity to correct the problem, or at least to be aware of it.

### 3.2 FLOSSMetrics

Because it is a hard and unnecessary work to reinvent the wheel, our framework exploits as much as possible external tools and existing databases. The FLOSSMetrics project provides a database scheme for the persistence support and populated databases respecting this scheme [7]. It is a popular means to study the software evolution [8]. It also provides tools to extract data from source code repositories, mailing lists and bug trackers. These tools support Subversion, CVS and Git thanks to CVSAnaly2; mailbox files thanks to Ml-stats; and the Sourceforge bug tracker thanks to Bicho.

Others tools can be used, as long as the produced results are stored in a database respecting the FLOSSMetrics conventions. Because the collection and analysis of software ecosystem data is a time-consuming and error-prone process, we use public databases provided by FLOSSMetrics. We can also directly use the FLOSSMetrics tools to populate a database. FLOSSMetrics can thus be viewed as a part of our framework.

### 3.3 Herdsman

Herdsman is an essential part of the application layer of our framework. The tool provides a (partially) automated way to visualise and explore collected metrics. Typically, researchers use their own tools to extract and visualise metrics. To provide a more reusable and effortless solution, we propose an extensible, generic tool allowing researchers to define and visualise metrics about the software ecosystem. The FLOSSMetrics-compliant database scheme used by Herdsman is extensible by external tools to improve the knowledge of the studied software. Herdsman can produce visualisations about the project evolution from source code repositories, bug trackers or mailing lists metrics, and combinations thereof.

#### Merging identities.

If the information we wish to display is obtained through a combination of different data sources, one of the most important issues is the identification of entities. We need to determine what are the mappings between the entities in the different data sources. For instance, we have to identify what are the identities of a committer in the mailing list or the bug tracker. To help the user in this recognition process, we provide a semi-automatic tool inspired by [15]. Because of the nature of the merging process, merging data cannot be stored in the project metrics database. The merging tool resides in the application layer and is available for the other applications. We exploit metadata contained in artefacts to characterise the identity of the individuals involved: for commits, the nickname of the committers and, if available, their e-mail address; for mails, the e-mail adress of the mailers; for issue reports, the login and the e-mail address of users and developers.

---

1. [www.flossmetrics.org](http://www.flossmetrics.org)

### Types of diagrams.

Herdsman is able to produce a wide variety of diagrams, since it is based on JFreeChart2. All these diagrams can roughly be classified in two categories: snapshots and temporal diagrams. Snapshots offer a representation of metrics at a given point in time. Temporal diagrams present the evolution of some metrics as a function of time. In a single picture, one can view what is the evolution of these metrics. Both kinds of views are complementary because a temporal diagram offers a simple, all-in-one information; a snapshot can present more detailed information. Figure 2 shows how our tool uses a slider to change the time of the displayed snapshot. Sliding the cursor from the start of the project to its end, enables an animated view of the snapshot evolution, which is impossible to do with a temporal diagram.

For temporal diagrams, such as the one in Figure 5, different time axes are available. The id axis is based on the cardinality of the artefact from which data is collected. For instance, the revision id is used to display data about commits in a version repository. The date axis provides a linear time representation. The tag axis (available for commit metrics) is based on tags created by developers. They mark milestones in the software development.

![Figure 2: A visualisation using a slider.](www.jfree.org/jfreechart)

#### Daily and hourly activities.

To try to find patterns in the behaviour of entities, one can analyse the time when they are active. For example, we can expect that different groups of developers have different work hours. It is a reasonable assumption that professional developers mainly work during office hours and work days. At the opposite side, volunteers will mainly work during their free time, in the evening or during holidays. Herdsman provides a way to verify this hypothesis by displaying the
daily and hourly distribution of committing, mail sending, bug track collaboration or a combination of them. Figure 3 shows a snapshot of the hourly commits and e-mail activities for a recent version of the Evince project.

Figure 3: Snapshot (on 15 April 2009) comparing hourly commit and mail activities for Evince.

To try to find patterns in the commit and mail activities, we can use the tabular view of Figure 4, displaying a snapshot of the e-mail activity for a given day against a given hour. The darker a field of the table is, the less intense is the activity for the related day and hour. We observe, for example, that every day of the week the activity between 2 and 7 AM is much lower than for the other hours of the day. Figure 2 provides an alternative view of the same data using colored and stacked bar charts. Each hour of the day is assigned a different colour code, and the amount of activity is represented by the height of the corresponding block.

Figure 4: Snapshot (on 1 March 2010) comparing daily and hourly mail activity for Evince. Lighter is more intensive.

4. EXPERIENCE

4.1 Case study

We have started to use our framework to empirically study the evolution of the Evince ecosystem. This is a popular document viewer written in C and mainly used on the GNOME Desktop. It has been selected for the analysis because of its popularity, its age (the project started in 1999) and the ease of collecting mail list data: FLOSSMetrics provides up-to-date commit and mail databases for this project. We are currently using Herdsman visualisations to conduct a qualitative study of the developers commit and mail activity; some of these visualisations are presented in this paper.

As illustrated in Figure 5, we observed an increasing interest of developers for Evince: after a relatively slow progression of the number of committers until 2003, the number of involved developers exploded in 2004 and 2005. These new committers quickly started to lead the project, publishing a lot of commits. The number of new participating committers per revision is roughly constant.

Figure 5: Temporal evolution of number of participating committers for Evince.

According to Figure 3, generated with Herdsman, the commit and mail activities are mostly concentrated from 10 AM till midnight. In order to gain a deeper understanding of the e-mail activities, we can try to interpret Figure 4, also generated with Herdsman, which provides a more detailed picture of the hourly distribution of e-mail activities on different days of the week. We clearly identify a dark area of low activity in the early hours between 2 and 8 AM. The most e-mail activity appears to occur between 10 and 12 AM.

Figure 6: Boxplots of daily e-mail activity for Evince.

In order to understand how the e-mail activity is dispersed over different days of the week, we generated a set of boxplots, shown in Figure 6. Each boxplot is based on a data set of 24 values (one for each hour) representing the number of mails sent in that particular hour. The boxplots reveal an important decrease in e-mail activity over the weekend,
as can be expected. There is also not a lot of variation in the number of mails sent during the weekend. Finally, without any exception, when analysing all outliers visualised in the boxplot, we find that they correspond to a significantly higher e-mail activity between 10 and 12 AM (as we observed in Figure 3). For example, the outlier on Tuesday represents the fact that (over the analysed Evince timespan), 44 e-mails were sent between 10 AM and 11 AM.

In order to get a precise picture of how the activities are distributed over the different hours of the day, we created another set of boxplots in Figure 7. This time, the y-axis represents the 24 hours of the day, allowing us to see at which time of the day most of the e-mail activity occurred. In this figure, we do not observe any outliers, and the difference between weekdays and weekend becomes more clear. During the weekend, the activity is much more condensed (smaller boxes), and the activity starts and ends earlier in the day: most of the activity during the weekend is spent before noon. During weekdays, the activities span a wider time range: most of the activity tends to occur between 6 AM and 22 PM.

![Boxplots illustrating the e-mail activity for Evince at different hours of the day.](image)

Figure 7: Boxplots illustrating the e-mail activity for Evince at different hours of the day.

We exploited the export functionality of our framework, to export the values used in Figure 3, and to import them into Microsoft Excel for further processing and generation of Figures 6 and 7. In the future, we aim to combine our framework with other applications, such as the statistical tool R, to provide more detailed statistical analysis. We will also extend the proposed visualisations to improve the usability of Herdsman, for example by providing built-in generic support for visualising boxplots.

The current Evince case study only served to illustrate how our proposed framework can be used in practice. In the future, we will carry out more case studies, in order to compare different software ecosystems (including Evince) to try and find patterns of recurrent behaviour over different projects, and to try to understand why certain ecosystems behave differently than others.

### 4.2 Threats to Validity

Besides the traditional threats to validity one encounters during studies on the evolution of OSS [5], some specific threats to validity are relevant to our empirical study. The first one is the reliability of the data that has been extracted by FLOSSMetrics and its third-party tools. Because several tools are used to populate the database, we have to take into account potential inconsistencies and redundancies due to differences in metrics definitions, bugs in the tools and usage of two tools for extracting the same metric. Sometimes, metrics provided by FLOSSMetrics seem wrong, and a particular focus on the meaning of used values is needed. For instance, a naive interpretation of the database content reveals that there are more deleted files than created files. This issue is due to the nature of the source code repository information for which a file copy doesn’t create any file.

We observed that FLOSSMetrics does not always respect its own database scheme. A recurrent issue is to have a mailing list database without correlation between a mailer id and data about physical persons. This makes it impossible to give a name or email address to a mailer. Our identity merging tool is useless in this case, and some visualisations are not possible. The merging tool itself is not perfect either: it cannot always find identical persons due to a lack of information: because a similarity distance is used to match entities, an arbitrary threshold needs to be defined to determine if two identities represent the same person.

For time-based metrics, a recurrent problem is the lack of information about the timezone difference between the client and the server storing the activity data. This is for example the case for commits. This issue partially distorts the results. The only way to resolve it would be to discover the geographical location of the client. For instance, if a committer has an email address ending by .de, it is reasonable to assume that this committer resides in Germany. This issue is perhaps less problematic for commercial software projects which are often developed in a single geographical location. Eventually, Herdsman will be able to use the identity merging tool to determine the timezone of any person that has sent e-mails. Each time somebody sends an e-mail, its timezone is marked in the mail headers, so it’s possible to approximate a person’s timezone based in this information.

More general threats need to be addressed as well. Internal validity threats concern potential defects in Herdsman and auxiliary home-made tools due to the immaturity of our framework. External validity threats concern external tools used to populate the databases. They are harder to find and fix because they require a thorough knowledge of these tools.

### 5. CONCLUSION

The framework proposed in this paper allows to study and improve the knowledge of the evolution of OSS ecosystems. The empirical study we started to carry out attests that the framework, and the Herdsman tool in particular, can be easily used to visually represent a wide range of metrics relative to OSS ecosystems. Our framework provides a comprehensive, dynamic way to study evolution patterns in software ecosystems. It is built upon, and takes advantage of existing tools (like FLOSSMetrics and JFreeChart) that have proven their use in the past. The framework will continue to be extended in numerous ways: developing a more reliable identity merging tool; making the framework interoperable with more external tools and databases; adding more visualisations; adding the possibility to combine metrics and to compare different ecosystems; providing wizards to help users to choose the best software for them and to help developers to understand the software evolution and how to improve it.
Acknowledgments
This work has been supported by the F.R.S. - FNRS through FRFC project 2.4515.09 “Research Center on Software Adaptability”, and by research project AUWB-08/12-UMH “Model-Driven Software Evolution”, an Action de Recherche Concertée financed by the Ministère de la Communauté française – Direction générale de l’Enseignement non obligatoire et de la Recherche scientifique, Belgium.

6. REFERENCES