Evaluation of FLOSS by Analyzing Its Software Evolution:: An Example using the Moodle Platform

ARTICLE · JULY 2017

DOI: 10.4018/JITR.2015010105

3 AUTHORS, INCLUDING:

Gregorio Robles
King Juan Carlos University
121 PUBLICATIONS **1,301** CITATIONS
SEE PROFILE

Jesus M. Gonzalez-Barahona
King Juan Carlos University
126 PUBLICATIONS **1,023** CITATIONS
SEE PROFILE

---------- Pre-print version ----------

Refence of the final, published version:

Héctor J. Macho, Gregorio Robles and Jesus M. Gonzalez-Barahona: "Evaluation of FLOSS by Analyzing Its Software Evolution". Journal of Information Technology Research 07/2017; 8(1):62-81. DOI: 10.4018/JITR.2015010105

Evaluation of FLOSS by Analyzing its Software Evolution:

An Example Using the Moodle Platform

Héctor J. Macho, *Universidad Rey Juan Carlos, Spain*

Gregorio Robles, *Universidad Rey Juan Carlos, Spain*

Jesús M. González-Barahona, *Universidad Rey Juan Carlos, Spain*

ABSTRACT

In today's world, management often rely on FLOSS (Free/Libre/Open Source Software) systems to run their organizations. However, the nature of FLOSS is different from the software they have been using in the last decades. Its development model is distributed, and its authors are diverse as many volunteers and companies may collaborate in the project. In this paper, we want to shed some light on how to evaluate a FLOSS system by looking at the Moodle platform, which is currently the most used learning management system among educational institutions worldwide. In contrast with other evaluation models that have been proposed so far, the one we present is based on retrieving historical information that can be obtained publicly from the Internet, allowing us to study its evolution. As a result, we will show how by using our methodology management can take informed decisions that lower the risk that organizations face when investing in a FLOSS system.

Keywords: software engineering; software evolution; software evaluation; free software; open source, Moodle; LMS.

INTRODUCTION

Free/Libre/Open Source Software (FLOSS) has gained wide acceptance in today's technological field, and the number of organizations that consider it has grown significantly (Ven, Verelst, & Mannaert, 2008), to the point that many software-intensive organizations have adopted it for tasks that could be considered their core business (Hauge, Ayala, & Conradi, 2010).

Although most FLOSS projects follow an open development model, where transparency allows to take the pulse of the project, managers still are suspicious of a distributed model where seldom a single entity governs the project –as it happens in the proprietary domain, where a software belongs to only one vendor. In recent years, a myriad of models have been proposed to provide managers the tools to perform informed decisions when adopting FLOSS (Groven, Haaland, Glott, Tannenberg, & Darbousset-Chong, 2012). These models take advantage of

the availability of public data of the projects, offering the possibility to perform a structured analysis that takes into consideration the requirements of the organization.

However, these models offer in general a limited perspective of the project as they try to summarize some of the current attributes or characteristics of the projects into a single parameter, usually a final mark that allows managers to rank all possible solutions according to how suitable they are for their organization. In this paper, we argue that the evolution over time of many of these attributes and characteristics provide a wider perspective and allow to have a more detailed view of the project(s) under study. The goal therefore is to show, with the help of a case study (the Moodle learning management system), that software evolution aspects should be taken into consideration when analyzing a FLOSS project for its adoption.

The structure of this paper is as follows: in the next section, related research on evaluation of FLOSS projects is presented. *METHODOLOGY* introduces the proposed methodology for evaluating a FLOSS software from a software evolution point of view. Then, the case study is presented, together with the tools used to apply the methodology on it. *RESULTS* offers the results of using the aforementioned methodology to our case study, Moodle. Finally, conclusions are drawn.

RELATED RESEARCH

This section introduces research and proposals related to the evaluation of FLOSS projects. Several FLOSS evaluation frameworks exist. We will present OpenBRR™, one of the first FLOSS assessment models and undoubtedly the most known one, whose philosophy sums up many of the characteristics that are common to most of the evaluation frameworks.

OpenBRR™

The Business Readiness Rating (OpenBRR™)[1] is a model is based on the identification of functional evaluation criteria for the software under study. These criteria are supposed to be extracted from the project repositories with the help of automated tools and available for the evaluators in form of a spreadsheet that groups them according to several aspects. A final step allows evaluators to specify different weights to the aspects, in accordance to their subjective importance for adoption, and to obtain a final mark for the project, which can be compared to other FLOSS projects. Figure 1(a) offers a graphical perspective of the model: normalized metrics are obtained from the FLOSS project, which are weighted to provide an intermediate rating to a set of predefined categories that range from functionality to usability. These ratings are used (provided that they are weighted according to their importance for the evaluating organization) to obtain a global rating. Ratings range from 1 (unacceptable) to 5 (excellent). Figure 1(b) shows how its proponents have conceived that the model should be used. They have identified in total four different phases. In a first one, all projects of a given domain are quickly assessed, so that the ones that most likely seem not to be suitable for the organization are filtered out. This step avoids having to perform the rest of phases, which may be costly in time and resources. Then, the target usage should be considered, as this will result in values for the weights in the model. By means of tools, data on the candidate FLOSS projects is to be collected and processed in a third phase. The last phase transforms the collected data to the various category ratings in the model, and finally to the final rating. An example of the use of OpenBRR™ on a FLOSS business intelligence suite can be found in (Marinheiro & Bernardino, 2013).

Figure 1: (a) OpenBRR™ model. (b) OpenBRR™ assessment phases[2].

OpenBRR™ has served as basis for many other research efforts. Germán et al. discuss the challenges of automating the process to obtain a quantitative analysis of FLOSS projects (German, Robles, & Gonzalez-Barahona, 2006), while Das et al. use FLOSSMole data, a public repository that offers publicly data from SourceForge and many other FLOSS software development platforms, to determine OpenBRR™ for FLOSS projects (Das & Wasserman, 2007). Cau et al. extend the OpenBRR™ model with metrics for object oriented FLOSS projects (Cau, Concas, & Marchesi, 2006).

Other Models

Other FLOSS evaluation models exist (in (Stol & Babar, 2010) up to 20 models have been identified), the most notorious ones are briefly presented here. QSOS™ (which stands for Qualification and Selection of Open Source Software2) and the OSMM™ (Open Source Maturity Model) (Golden,2005) are industry-led proposals. In addition, from an academic-industrial consortium, the QualiPSo OpenSource

Maturity Model (OMM) has been proposed (Petrinja, Nambakam, & Sillitti, 2009): this model approaches evaluation from the perspective of the well-known and popular Capability Maturity Model and tries to adapt it to the FLOSS domain. Finally, we can find SQO-OSS, a model that measures quality of FLOSS projects (Samoladas, Gousios, Spinellis, & Stamelos, 2008).

Comparisons

Given the presence of so many models, comparisons among them have been made. As an example, we can cite:

Between OpenBRR™ and QSOS™ (Deprez & Alexandre, 2008), where the parameters under study, the methodology and the flexibility are considered.

Among OpenBRR™, QSOS™ and OMM (Petrinja, Sillitti, & Succi, 2010), being all of them applied to Mozilla Firefox and the FLOSS version of the Chrome web browser (Chromium). The three models offered comparable evaluations for the two projects, although the authors noted that in all of them there were unclear aspects that made the evaluation work sometimes difficult.

A framework, called Framework fOr Comparing Open Source software Evaluation Methods (FOCOSEM) and based on the systematic NIMSAD methodology comprehension and evaluation meta-framework (Jayaratna, 1994), that helps in the comparison of evaluation models has also been proposed (Stol & Babar, 2010). This framework classifies the parameters in several components: context, user, process, and evaluation.

Other perspectives

Some authors have noted that the evaluation models lack information regarding some perspective that they find important, and have proposed to introduce them. So for instance, Groven et al. argue to include security measurements in the frameworks (Groven, Haaland, Glott, & Tannenberg, 2010), while Izquierdo et al. see a lack of human factors in the models and propose to include also quality models that take the development communities into consideration (Izquierdo-Cortazar, Gonzalez-Barahona, Duenas, & Robles,2010).

Our approach could be also seen as adding a new perspective to the existing adoption models. In the opinion of the authors, the models should include historical information of the projects in the assessment. This idea is not new, as it has been already been worked on by Deprez et al. who have been investigating to define software evolvability for FLOSS projects (Deprez, Monfilsc, Ciolkowski, & Soto, 2007); Izquierdo et al. extend the concept to include –in addition to the software artifacts– the community of contributors (Izquierdo-Cortazar, González-Barahona, Robles, Deprez, & Auvray, 2010).

Fogel provides a cookbook-like guide to how to run FLOSS projects successfully, including many of the details that are considered in the previous models (Fogel, 2005). However, the aim of Fogel is far from evaluating the projects, but to obtain parameters that help running a healthy software project.

The previous models, including our approach, are concerned with how FLOSS software gets introduced into a company or institution. Kilamo et al. offer an evaluation framework, called R3 (Release Readiness Rating) for companies wanting to release their software (usually offered previously in a proprietary fashion) as FLOSS (Kilamo, Hammouda, Mikkonen, & Aaltonen, 2012).

METHODOLOGY

To understand and evaluate the evolution of a FLOSS project we have chosen a set of criteria that includes information from the software product, the software process, the community and economic impact. These criteria are:

Growth of lines of code: software size is a common characteristic used as a proxy for the total complexity of a software component. Although initially measured in modules (i.e., number of source code files), it is currently widely accepted to be measured as well in number of lines of code (Herraiz, Robles, González-Barahona, Capiluppi, & Ramil, 2006), without considering blank lines or comments. The evolution of the size of a software has been the matter of research by Lehman and colleagues for long time and is the basis of the “laws” of software evolution (Lehman & Belady, 1985; Lehman, Ramil, Wernick, Perry, & Turski, 1997). The evolution of the number of lines of code is a measure of the vivacity of a project, as one of the “laws” of software evolution states that the software has to constantly be adapted to changing requirements by users.

Similarity between releases: We define similarity between releases as the

percentage of source code from a previous version that can still be found in a particular version. As the software evolves, changes are introduced. Some of them may be minor changes, which may be corrective or adaptive in nature and as such do not affect other parts of the code base. However, from time to time, developers introduce large amounts of changes that may induce to software inconsistencies. The study of similarity over time allows decision makers to assess the stability of a FLOSS project. If a major change occurs, they may use this information to take decisions that affect external modules or plug-ins.

Growth of cyclomatic complexity: The cyclomatic complexity, also known as conditional complexity, provides a measure of the complexity of the code (McCabe, 1976). It is known that cyclomatic complexity, among other complexity measures, correlates with lines of code (Herraiz & Hassan, 2010), so its absolute value would provide the same information as the growth in lines of code. Nonetheless, we introduce this measure in relation to the one of lines of code, obtaining it as a fraction. Hence, we will consider the evolution of the cyclomatic complexity per line of code for the total of the project, which will allow us to see how the complexity of the code base is evolving.

Commits per week: Most FLOSS projects have a common versioning repository where developers synchronize their contributions (which is known as to perform a commit). The number of commits to the versioning system is one of the measures of the activity in the project. Its evolution over time gives information on how the project is gaining or losing activity over time.

Tickets/issues per week: The other major activity in the project can be observed in a bug-tracking system, where not only bug reports but enhancement suggestions are managed. These systems are mostly web-based and serve as a communication means between developers themselves, but between developers and users as well. In this regard, the number of tickets/issues per week is a complimentary measure to commits in order to study the activity of a project.

Authors per week: A committer is the person who performs the commit to the versioning repository. However, a committer may not be the original author of the source code, as committing requires certain privileges. Therefore, modern versioning systems include in the commit meta-data information about the author in addition to the committer. Studying the number of authors is a way of analyzing the number of participants in the project. The analysis of the authors in a time frame provides information about how many active participants the project has. If this measure is seen over time it allows to see how the number of active contributors is changing.

Demographic information: Contributors may come and go. The previous measure about authors per week hides this information as a developer may not be constantly participating. With a demographics analysis, we divide contributors in cohorts and calculate two measures which we have called aging (the number of years in the project of active contributors) and birth (the number of new contributors in the community). These measures will allow us to determine how the human resources of a project are evolving over time.

Companies involved: FLOSS has gained in the last decade much attention from the software industry. As a consequence, many companies are involved in FLOSS projects and collaborate with a community composed of volunteers and other companies. The analysis of the number of companies in a project may provide valuable insight to managers, as usually the existence of a company in a FLOSS project means that additional, commercial services may be obtained. In addition, if there are many active companies in a project, it can be seen as an additional sign for the mid-term sustainability of the project. It may also mean that services may be acquired from several companies that have advanced know-how on the project, avoiding a monopolistic situation.

CASE STUDY

Moodle is a FLOSS learning management system (LMS) widely used currently in many educational institutions. Moodle is a web-based tool implemented in the

PHP programming language with a MySQL database back-end. Its design is modular, so it offers the possibility to develop modules and plug-ins which can be aggregated to the main project. Due to its popularity, Moodle accounts with a large user base, an active community, versions in over 100 languages and several companies that offer several types of support, ranging from development of modules to hosting and maintenance.

Although probably the most used nowadays, Moodle is only one of the many FLOSS LMS that exist today. Other popular LMS are Claroline, Dokeos, Chamilo or .LRN.

To apply the methodology introduced in *METHODOLOGY* to Moodle, we have used a set of tools and procedures which are available to other researchers for replication due to their FLOSS nature. The data sources are as well public. Thus, the authors have released a replication package to allow others to replicate or even build on top of our efforts[3]. For every analysis, the tools that have been used are:

Growth of lines of code: For this analysis we have used three tools, CLOC[4], SLOCCount[5] and Ohcount[6], as they use different heuristic algorithms. Hence, it is more important to see if the results given by the three tools are consistent (in terms of offering a similar growth pattern) than to see if they are the same, as this may depend on the heuristics, the programming languages that they identify, etc.

Similarity between releases: We have authored a Python script to compare the two source code trees and to calculate the similarities (we have applied the script to one version and all its previous versions). The resulting index gives us an idea about the modifications that have been done and the addition/removal of files. Hence, two releases are similar if they have an index of 1. If the releases have nothing in common, their similarity index is 0. Values in between give a degree of similarity.

Growth of cyclomatic complexity: The tool chosen for this analysis was PHP Depend[7], given that the most of the Moodle source code is written in PHP (there are other programming languages in the code such as Javascript). As Moodle is a web-based project, elements written in HTML, CSS or SQL may add complexity to the final product which has not been included in our analysis.

Commits and authors per week: We have obtained the data from Moodle's git repository (which contains the history of the CVS repository, too) using the CVSAnalY[8] tool that is part of the MetricsGrimoire[9] platform. The figures shown in this paper for this analysis are the result of using VizGrimoire[10]. The scripts provided with VizGrimoire can identify duplicated users (i.e., when the same user uses several email addresses) with the same email or the same name for modern repositories. In the case of CVS, as no email is used (a nickname is the way of identifying in such systems), the script cannot merge duplicated users. Using external information, we have done this process manually.

Companies involved: We have extracted information from the e-mail domain of the authors in the versioning repository in order to assign them to companies. The e-mail address is obtained by means of CVSAnalY. The assignation has been manually verified. Moodle initially used CVS, where email information is not provided, so we have only analyzed the data from the git repository.

Demographic information: We have analyzed the information given by MetricsGrimoire with VizGrimoire in order to obtain the results and to show them as a graph.

Metrics about the tickets/issues: To obtain the information of the Issue Tracking System we need another tool, Bicho[11] (from the MetricsGrimoire toolkit, as CVSAnalY). The ITS used by Moodle is JIRA and the support of Bicho for it is not complete, so we had to modify the source code of Bicho partially (specifically the JIRA backend). The obtained information has been processed with VizGrimoire in order to obtain information about the issues opened and closed over time.

RESULTS

In this section, we present the results of applying our methodology to Moodle.

The analyses of lines of code, similarity and cyclomatic complexity have been performed on all versions released by the Moodle project between August 20th 2002 (version 1.0.0) and January 13th 2014 (version 2.6.1); both minor and major releases have been included.

In order to normalize the results for the analysis of source code, it has been necessary to remove all non-English translations found in the versioning system between versions 1.2.0 and 1.5.4 (both included). The reason for doing this is that outside these versions, translations are distributed in other packages.

For the analyses involving the versioning system, we have used the data between November 22nd 2001 and January 22nd 2014,

spanning the whole (publicly available) life of the project. In the case of the Issues Tracking System, the first issue is dated April 25th 2002. Analyzing the commit log we found the date "2009.10.23 05:23:47" as the date of the first commit in Git, with no others commits on CVS after it (we have found a previous commit in Git, but this was followed by commits to CVS). We have chosen this commit as the first commit of the companies analyze with all subsequent time commits.

Two bots ("Moodle Robot" and "AMOS Bot") that automatically perform commits to the repository have been identified. These bots have been filtered out in our activity, authorship and company involvement analysis.

Growth and Similarity

When talking about source lines of code (SLoC) in this section, we are referring to those source lines of code that are not blank nor comment lines. Although comment lines are very important and needed to understand the source code of a program they may provide a false results in evolution terms, so it is general research practice to not consider them for software evolution studies or effort estimations.

Figure 2: (a) Growth in Source Lines of Code (SLoC) as provided by SLOCCount, CLOC and Ohcount. (b) Heat Map Chart of similarity between different versions. Dark colors are indicative for a high amount of common source code.

Figure 2(a) provides the growth in SLoC of Moodle release after release. It should be noted that although the three tools provide different results, they show the same trend. The difference among the tools are due to various reasons, but mainly because SLOCCount does not take the JavaScript language code into consideration. There are some minor differences as well due to the heuristics used in the programs. Except for the quasi-vertical line that appears between the 1.9.19 and the 2.0.0 versions, the growth rate is quasi-linear. The gap for 2.0.0 is because Moodle 2 is a special effort that included a many new functionalities, but as well a severe architectural change. As such, its development was done in parallel with the latest 1.x versions during several years. Not considering this jump, linear growth seems to be found consistently through the whole lifetime of Moodle. Similar linear growth has been found in many other FLOSS projects (Robles, Amor, Gonzalez-Barahona, & Herraiz, 2005) in the past; sometimes even, as it is the case of Linux, super-linearity has been reported (Godfrey & Tu, 2000). This is an interesting finding as the "laws" of software evolution indicate that the growth pattern of software projects should become slower with increasing size (Lehman, 1996), the rationale being that in addition to adding new functionality, effort should be devoted to maintain the already developed codebase. Hence, this is indicative for a sustainable community in Moodle, that has achieved a balance that allows to grow (i.e., add more functionality) while at the same time performing the required maintenance on the code.

The similarity heatmap in Figure 2(b) offers a different perspective of the evolution of Moodle, as it provides for each pair of releases how much in common (i.e., similar) their source code base is. This is an important measure that complements the growth in SLoC, as the former is usually granted as a proxy of the growth in functionality. The similarity measure offers information about how much the source code base is modified in absolute terms, including all types of maintenance (corrective, adaptive and perfective (Swanson, 1976)) in addition to the inclusion of new features.

The horizontal and vertical axis start with the first version of Moodle (1.0.0) and end with the last one (2.6.1); versions grow horizontally from left to right and from top to bottom. As the heatmap is symmetric, only half of it is shown. Versions with few differences are given by a black or a dark gray dot (for instance, the diagonal is completely black as versions have no difference with themselves), while the color becomes lighter the more versions differ. Given a version of Moodle, we can see its differences to previous versions by following the vertical colors; the differences with future versions are given by following the release horizontally.

Major version	Prev. minor version	Similar
1.1.0	1.0.9	54%
1.2.0	1.1.1	47%
1.3.0	1.2.1	89%
1.4.0	1.3.5	71%
1.5.0	1.4.5	55%
1.6.0	1.5.4	60%
1.7.0	1.6.9	68%
1.8.0	1.7.7	82%
1.9.0	1.8.14	77%

2.0.0	1.9.19	26%
2.1.0	2.0.10	91%
2.2.0	2.1.10	88%
2.3.0	2.2.11	84%
2.4.0	2.3.11	92%
2.5.0	2.4.8	81%
2.6.0	2.5.4	87%

Table 1: Similarity measure between last minor and next major version.

If we analyze the similarity heatmap, we can see that the architecture of Moodle has become more stable over time, i.e., there are less differences in recent versions than in older ones. This can be observed from the size of the dark gray triangles. The 2.0 version of Moodle can be easily identified as it is complete restructuring of the previous versions. When comparing Moodle 2.0 with other previous major versions such as 1.8 and 1.9 we can see that stability is higher in the latter. Table 1 gives detailed numbers of the similarity measure calculated between the last minor version and the next major version which gives a detailed overview of this phenomenon.

Observation #1: Growth in terms of SLoC offers hints of the future sustainability of the project by observing past trends; in the case of Moodle, a steady growth is indicative for a healthy community. Similarity provides insight into the stability of a software project; in the case of Moodle, newer versions are more stable than the older ones.

Activity and Authors

Several questions should be answered before performing an activity analysis. These are: Are the commits of all the branches important to understand the evolution of a project? Do the resulting commits of a merge branch contribute to this kind of analysis?

In our case, we think that all the branches are important, because the development of a project in not only dependent on the master branch, but the activity that happens in all the branches is meaningful. However, if we analyze the commits of all branches it has no sense to consider the commits due to merging, which is what happens when information (i.e., source code) from one branch is transferred into another one, usually the main one.

Figure 3: (a) Commits (aggregated). (b) Commits per week.

Figure 4: (a) Author Activity (aggregated). (b) Author Activity per week.

Figures 3 and 4 provide information on the activity of the Moodle project. Figure 3(a) shows the growth over time of the number of commits in an aggregated manner. As in the case of the growth in SLoC, a quasi-linear trend can be identified, which makes us assume that a constant effort by developers can be recognized. However, from the amount of commits per week shown in Figure 3(b), we see that the activity in later phases of the project is slightly increasing, meaning that in reality we have a slight super-linear trend.

Figure 4(b) shows the number of authors with at least one commit in a week. To calculate this graph, in addition to the merging of distinct author emails performed by the tool, we have manually merged 52 identifiers in CVS using information from Github (and other forges) and Ohloh.

The number of distinct total authors in the project can be seen from Figure 4(a). Three different (linear) phases can be identified from it: a first phase from the beginning of the project until 2005 with a significant growth, a more calmed growth from 2005 until 2010, and a higher growth rate from 2011 onwards. By inspecting the release dates, we see that the first breaking point is close to the 1.5 release, while the second one corresponds to the release of Moodle 2.0. We are not in the position to provide reasons for the slowdown in version 1.5, but the acceleration since Moodle 2.0 is probably due to a more modularized architecture, one of the main aims of the new platform.

Figure 4(b) offers the information on a weekly basis, without aggregating. The graph supports the aforementioned results: it can be observed that Moodle had a first period of activity growth up to 2005 (with a peak of around 25 developers per week), then a second period of lower activity (with around 10 to 15 developers/week), and that after the release of Moodle 2.0 the number of participants has boosted (in recent times over 40 developers/week).

Figure 5: (a) Opened issues (aggregated). (b) Opened issues per week.

Finally, Figure 5(b) shows the number of aggregated opened issues, and Figure 5(a) the same information on a weekly manner. The growth trend of the number of issues in the aggregated graph follows a quasi-quadratic trend, implying that the activity in issues is growing faster than the ones of SLoC and commits. To understand the reasons behind this, it should be noted that FLOSS projects are modeled as onions, with a small core of very active developers, a larger amount of contributors, a larger amount of occasional contributors, an even larger amount of users (Crowston & Howison, 2005). Projects where the growth of issues outperform the one in commits/code base are those where the outer parts of the onion are growing faster than the inner ones. In other words, this result is indicative for the Moodle user base growing faster than the number of contributors.

Observation #2: Activity measures over time give information of the project's vitality and its future sustainability. In the case of Moodle, the number of commits is constant over time, while with version 2.0 the number of authors is growing significantly. The growth of issues outperforms the one of commits, meaning that the user base is growing faster than the number of contributors.

Mean Complexity

We have investigated the evolution of the complexity of the source code by means of the Cyclomatic complexity measure. The Cyclomatic complexity number gives the number of linearly independent paths of a fragment of source code (in our case, we have calculated the number of different paths of all the source code of a particular version). As it can be seen from Figure 6(a), the graph shows a similar behavior than the one of the growth of source lines of code. This is consistent with previous literature (Herraiz & Hassan, 2010), as usually by adding new functionality there is inherently a rise in the complexity of the code.

Figure 6: (a) Growth of Cyclomatic Complexity. (b) Cyclomatic Complexity / Lines of Code.

Nonetheless, the interesting comparison in this case is related to the rise in complexity per line of code. By inspecting Figure 6(b), we see that this increase is not parallel to the number of lines of code. Versions prior to 2.0 have, with exceptions for the 1.2.x and 1.3.x releases, mainly values around 0.23. Starting with version 2.0.0 we can observe an important decrease in the calculated cyclomatic complexity per line of code, reaching values close to 0.2 in the most recent ones. This suggests that the Moodle project has been working hard on reducing complexity, by performing preventive maintenance.

Observation #3: Complexity measures over time allow to know if the rise in functionality is not achieved at the expense of poor code quality. In the case of Moodle, although the code base is much larger in the latest versions, effort has been made to lower the relative complexity.

Involvement of Companies

Modern FLOSS projects are turning into complex software ecosystems (Messerschmitt & Szyperski, 2005) where volunteers collaborate with for-profit companies and non-profit institutions in a project.

The main company involved in Moodle is Moodle Pty Ltd (also known as Moodle.com and Moodle Headquarters, based in Perth, Western Australia), an Australian company which performs the majority of the development of the core Moodle platform. Figure 7(a) shows the number of commits by company and Figure 7(b) the number of authors by company. As we can see in the graphs, although the number of professional authors working for Moodle Pty Ltd it is not very high (about 6.69% of the total authors), the number of commits by those authors is more than a third part of the total number commits for the project (36.04%). This gives an idea of the central position of this company in the development of Moodle.

Figure 7: Activities per company in number of commits and distinct authors.

Among the rest of companies, only the Open University (a British on-line university) has a significant amount of commits (although still

only around one fifth of the ones of Moodle Pty Ltd). Considering the number of authors, in addition to Moodle Pty Ltd, only three other companies have devoted ten or more developers to the project (The Open University, NETSPOT and Catalyst IT).

Some of the involved companies are Moodle Partners, which means that they are part of the Moodle Partner network, a commercial arm in the Moodle environment that have several benefits (such as using the Moodle trademark) or may offer official services (like certification and support). It is interesting to analyze the results of these companies as only one company, to see how much commits and authors they contribute to the project.

Figure 8: Activities per company in number of commits and distinct authors (Moodle Partners grouped together).

We can see the results of grouping all Moodle Partners together in Figure 8. As a result, when comparing Figure 8 with Figure 7, we can see that the number of professional authors affiliated to a Moodle Partner surpasses those of any other company, including Moodle Pty Ltd. However, if we consider their activity, they are well below Moodle Pty Ltd. and even the Open University, which is not part of the Moodle Partner network. This can be understood as Moodle Partners providing mainly services around Moodle, and not being that much involved in the development of it.

Observation #4: The study of the involvement of companies sheds some light into the software ecosystem of a FLOSS project, especially signaling structures of power. In the case of Moodle, the importance of a company, Moodle Pty Ltd, is clear for its development. A high number of other companies are involved in Moodle, although their main activity is to offer services around it.

Demographics

Figure 9 shows the results of the demographics study. Aging is “the age of the active contributors, considering their first contribution” and birth is “the number of contributors joining the community”. Age is calculated as the time between the first commit/issue and the last one for a given developer.

Figure 9: Demographics in (a) the versioning system, and (b) the ticket/issues system.

Figure 9(a) provides the results of the demographic study applied to the information from the source code repository. The size of the birth bars at the bottom suggest that the community is attracting more developers in recent times. On the other hand, the size of the aging bars at the bottom point out that new developers are being retained. The size of the aging bars at the top is very small, so current developers are not very old. In short, the results show that, while there are some developers that have been contributing for long in Moodle, a vast majority of them have been attracted in recent times.

Figure 9(b) provides the results of the demographic study applied to the Issues Tracking System. Again, the size of the birth bars at the bottom suggests that the community attract more contributors lately, while the size of the aging bars at the bottom and at the top are indicative for new contributors are being retained and many old contributors still active (there are about 30 contributors still active with an age of 7 years).

Observation #5: The demography analysis provides information on the human resources of the project. In the case of Moodle, a high number of contributors have joined recently, and the project is successfully retaining them.

CONCLUSIONS

Software runs the world, and in recent times many organizations are looking for solutions from the FLOSS domain. However, many managers are still skeptic about its adoption and fear hidden risks.

Many FLOSS evaluation models have been proposed in the last years to assess FLOSS for its readiness in a professional context. These models try to provide managers with a tool so that they can take informed decisions in the adoption of a FLOSS project. This is done by handling information that can be obtained from publicly available repositories, and which are processed so that a final score of the suitability of a FLOSS is provided.

In this paper we argue that the previous models have severe limitations as they do not offer the overview that evolution of many interesting parameters offer. So, we have shown by means of the evolution several parameters how additional information can be obtained that is of significant importance in the decision of adopting a FLOSS software or not. To show its convenience, we have applied them to a case study, the well-known learning management system Moodle. With it, we have proven that the study of the evolution over time offers many facets of the software that are not to be neglected and that barely can be introduced into any mark that is composed of a simple range between 1 and 5.

Although our approach offers a wide range of information and data, it is computationally not much more difficult the current frameworks, as the input data is in general the same. What is different is how this data is offered and how it is analyzed and interpreted.

All in all, in opinion of the authors, we see evaluation models and frameworks turning into other paradigms to include this type of information. We envision in this sense the possibility of having dashboards or other, more visual tools that help managers in taking the best informed decisions.

ACKNOWLEDGMENTS

The work of Héctor J. Macho, Gregorio Robles and Jesús M. González-Barahona has been funded in part by the Spanish Government under project SobreSale (TIN2011-28110) and by eMadrid, S2009/TIC-1650, “Investigación y Desarrollo de tecnologías para el e-learning en la Comunidad de Madrid funded by the Region of Madrid. We would like to thank Bitergia S.L. for the support given with the tools.

References

Cau, A., Concas, G., & Marchesi, M. (2006). Extending openbrr with automated metrics to measure object oriented open source project success. In The workshop on evaluation frameworks for open source software.

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. First Monday, 10(2).

Das, A., & Wasserman, A. I. (2007). Using flossmole data in determining business readiness ratings. In Workshop on public data about software development, the 3rd international conference on open source systems, ifip (Vol. 2).

Deprez, J.-C., & Alexandre, S. (2008). Comparing assessment methodologies for free/open source software: Openbrr and qsos. In Product-focused software process improvement (pp. 189–203). Springer.

Deprez, J.-C., Monfilsc, F., Ciolkowski, M., & Soto, M. (2007). Defining software evolvability from a free/open-source software. In Software evolvability, 2007 third international ieee workshop on (pp. 29–35).

Fogel, K. (2005). Producing open source software: How to run a successful free software project. O’Reilly Media, Inc.

German, D. M., Robles, G., & Gonzalez-Barahona, J. M. (2006). The challenges of automated quantitative analysis of open source software projects. In The workshop on evaluation frameworks for open source software.

Godfrey, M. W., & Tu, Q. (2000). Evolution in Open Source software: A case study. In Proceedings of the international conference on software maintenance (p. 131-142). San Jose, California.

Golden, B. (2005). Succeeding with open source. Addison-Wesley Professional.

Groven, A.-K., Haaland, K., Glott, R., & Tannenberg, A. (2010). Security measurements within the framework of quality assessment models for free/libre open source software. In Proceedings of the fourth european conference on software architecture: Companion volume (pp. 229–235).

Groven, A.-K., Haaland, K., Glott, R., Tannenberg, A., & Darbousset-Chong, X. (2012). 5 quality assessment of foss. ITLED4240 Compendium Spring 2012, 79.

Hauge, Ø., Ayala, C., & Conradi, R. (2010). Adoption of open source software in software-intensive organizations–a systematic literature review. Information and Software Technology, 52(11), 1133–1154.

Herraiz, I., & Hassan, A. E. (2010). Beyond lines of code: Do we need more complexity metrics? O'Reilly Media.

Herraiz, I., Robles, G., González-Barahona, J. M., Capiluppi, A., & Ramil, J. (2006). Comparison between slocs and number of files as size metrics for software evolution analysis. In Software maintenance and reengineering, 2006. csmr 2006. proceedings of the 10th european conference on (pp. 8–pp).

Izquierdo-Cortazar, D., Gonzalez-Barahona, J. M., Duenas, S., & Robles, G. (2010). Towards automated quality models for software development communities: The qualoss and flossmetrics case. In Quality of information and communications technology (quatic), 2010 seventh international conference on the (pp. 364–369).

Izquierdo-Cortazar, D., González-Barahona, J. M., Robles, G., Deprez, J.-C., & Auvray, V. (2010). Floss communities: Analyzing evolvability and robustness from an industrial perspective. In Open source software: New horizons (pp. 336–341). Springer.

Jayaratna, N. (1994). Understanding and evaluating methodologies: Nimsad, a systematic framework. McGraw-Hill, Inc.

Kilamo, T., Hammouda, I., Mikkonen, T., & Aaltonen, T. (2012). From proprietary to open sourcegrowing an open source ecosystem. Journal of Systems and Software, 85(7), 1467–1478.

Lehman, M. M. (1996). Laws of software evolution revisited. In Software process technology (pp. 108–124). Springer.

Lehman, M. M., & Belady, L. A. (Eds.). (1985). Program evolution: Processes of software change. San Diego, CA, USA: Academic Press Professional, Inc.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., & Turski, W. M. (1997, nov). Metrics and laws of software evolution - the nineties view. In Metrics '97: Proceedings of the 4th international symposium on software metrics (p. 20).

Marinheiro, A., & Bernardino, J. (2013). Openbrr evaluation of an open source bi suite. In Proceedings of the international c* conference on computer science and software engineering (pp. 134–135).

McCabe, T. J. (1976, December). A complexity measure. IEEE Transactions on Software Engineering, 2(4), 308-320.

Messerschmitt, D. G., & Szyperski, C. (2005). Software ecosystem: understanding an indispensable technology and industry. MIT Press Books, 1.

Petrinja, E., Nambakam, R., & Sillitti, A. (2009). Introducing the opensource maturity model. In Proceedings of the 2009 icse workshop on emerging trends in free/libre/open source software research and development (pp. 37–41).

Petrinja, E., Sillitti, A., & Succi, G. (2010). Comparing openbrr, qsos, and omm assessment models. In Open source software: New horizons (pp. 224–238). Springer.

Robles, G., Amor, J. J., Gonzalez-Barahona, J. M., & Herraiz, I. (2005, September). Evolution and growth in large libre software projects. In Proceedings of the international workshop on principles in software evolution (pp. 165–174). Lisbon, Portugal.

Samoladas, I., Gousios, G., Spinellis, D., & Stamelos, I. (2008). The sqo-oss quality model: measurement based open source software evaluation. In Open source development, communities and quality (pp. 237–248). Springer.

Stol, K.-J., & Babar, M. A. (2010). A comparison framework for open source software evaluation methods. In Open source software: New horizons (pp. 389–394). Springer.

Swanson, E. B. (1976). The dimensions of maintenance. In Proceedings of the 2nd international conference on software engineering (pp. 492–497).

Ven, K., Verelst, J., & Mannaert, H. (2008). Should you adopt open source software? Software, IEEE, 25(3), 54–59.

Notes

[1] http://www.openbrr.org/

[2] Source of both figures: OpenBRR web page (http://www.openbrr.com).

[3] http://www.qsos.org/

[4] The complete replication package, including

original data sources, software and results can be

downloaded from http://gsyc.urjc.es/~grex/repro/2014-ijitsa-moodle/

[5] http://cloc.sourceforge.net/

[6] http://www.dwheeler.com/sloccount/

[7] https://github.com/blackducksw/ohcount

[8] http://pdepend.org

[9] http://metricsgrimoire.github.io/CVSAnalY/

[10] http://metricsgrimoire.github.io

[11] http://vizgrimoire.bitergia.org/

[12] http://metricsgrimoire.github.io/Bicho/