Exploring the Role of Commercial Stakeholders in Open Source Software Evolution

Andrea Capiluppi, Klaas-Jan Stol, Cornelia Boldyreff

Exploring the Role of Commercial Stakeholders in Open Source Software Evolution

Andrea Capiluppi[1], Klaas-Jan Stol[2] and Cornelia Boldyreff[3]

[1]Brunel University, United Kingdom
[2]Lero—The Irish Software Engineering Research Centre
University of Limerick, Ireland
[3]University of East London, United Kingdom
andrea.capiluppi@brunel.ac.uk, klaas-jan.stol@lero.ie,
c.boldyreff@uel.ac.uk

Abstract. It has been lately established that a major success or failure factor of an OSS project is whether or not it involves a commercial company, or more extremely, when a project is managed by a commercial software corporation. As documented recently, the success of the Eclipse project can be largely attributed to IBM's project management, since the upper part of the developer hierarchy is dominated by its staff. This paper reports on the study of the evolution of three different Open Source (OSS) projects — the Eclipse and jEdit IDEs and the Moodle e-learning system — looking at whether they have benefited from the contribution of commercial companies. With the involvement of commercial companies, it is found that OSS projects achieve sustained productivity, increasing amounts of output produced and intake of new developers. It is also found that individual and commercial contributions show similar stages: developer intake, learning effect, sustained contributions and, finally, abandonment of the project. This preliminary evidence suggests that a major success factor for OSS is the involvement of a commercial company, or more radically, when project management is in hands of a commercial entity.

1 Introduction

Governance and control in Open Source Software (OSS) has been dramatically changing [30]. The *traditional* volunteer-based OSS project model is now being accompanied by *sponsored* OSS, where commercial stakeholders provide effort beyond voluntary programmers. It has been argued that OSS projects have become increasingly hybrid with respect to this type of contributing stakeholders [12].

Since their inception in the early 1980s, OSS projects were mostly volunteer-based (or *Traditional OSS*, right end of Figure 1), heavily relying on personal efforts and non-monetary recognition, and reportedly suffering from communication and coordination problems [14].

Nowadays, so-called *Sponsored OSS* projects have also been documented as more similar to *Closed Source* systems (as in far left of Figure 1). They could be industry-led OSS projects, where a commercial stakeholder plays a major role in the development and decision making, as in the case of the Eclipse project by IBM [25, 24, 19, 33].

They can also be *industry-involved* OSS projects when they are driven by an OSS community, but often have one or several companies or institutions (e.g., universities) among their stakeholders, as in the case of the Moodle Content Management System (CMS) [8].

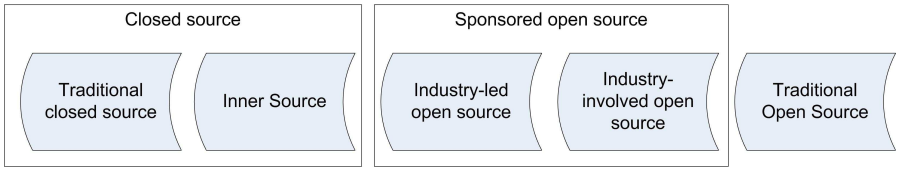

Fig. 1. Scenarios of development of OSS projects

Both the *industry-led* and *industry-involved* scenarios introduce new challenges to OSS projects: the first is based on one (or a small subset of) critical stakeholder(s), which could eventually halt the project if they decide to abandon it[1]. In the case of Eclipse, for example, IBM staff have been identified as the top contributors, with only a few external developers working on the core system [36]. For *Community* OSS, especially in the case of large and complex OSS systems, there is a need of proper incentives for different types of stakeholders, with complementary expertise and requirements, in particular when their contributions are relevant to a system's core functionality.

This paper aims to explore these three scenarios and to study whether the involvement of commercial companies can help sustaining the evolution of OSS projects. To that end, the paper presents different analyses of the evolution of a *commercial* and a *traditional* OSS systems (Eclipse and jEdit), sharing the same application domain, and one *community* OSS project (Moodle).

By exploring the type of activities performed by commercial stakeholders, and by comparing the results achieved by similar OSS projects (sharing the same application domain, but with different involvement of stakeholders), this paper explores a research area that only recently started to be covered in the literature [28, 27, 31].

1.1 Terminology

Existing literature typically distinguishes between *community-managed* (also called *autonomous* [32]) and *sponsored* communities. However, current terminology for this has some issues. For instance, OSS projects that are led by firms are referred to as *commercial OSS*, whereas an OSS project that involves commercial stakeholders (companies), but is led by an OSS community (consisting of "traditional" community members) is referred to as *Community OSS*. We argue that both terms are not precisely defined and need further refinement. Commercial OSS suggests that profit is made from

[1] This happened with Netscape Navigator (then Mozilla) when Netscape Communications Corporation (NCC) released it as open source, but without further evolving it.

the OSS project. The term "Community OSS" does not clearly distinguish projects that involve companies from "traditional" OSS projects (that do not involve companies). Therefore, in this paper we propose the following new terminology for the various models of involvement:

Traditional OSS projects are those projects in which no companies are involved.

Industry-involved OSS projects are projects in which commercial firms are involved as contributors, but the project is still managed by the "community".

Industry-led OSS projects are projects that are led by a commercial firms. The wider community can contribute (as with any OSS project), but since a company has control over the project, it defines the evolution strategy.

Together, industry-involved and industry-led projects are **Sponsored** OSS projects, whereas industry-involved and traditional projects are both forms of **Community** projects (see Figure 1) as they are led by a community (as opposed to a company).

1.2 Structure of this paper

The remainder of this paper is structured as follows: Section 2 presents the goals, questions and metrics of the study. Section 3 presents the research design. Sections 4 and 5 presents the Eclipse and jEdit case studies, respectively. Section 6 focuses on the Moodle system as an example of an industry-involved OSS system, and explores the relevance of the commercial stakeholders, and how they differ from individual developers. Section 7 discusses the results followed by conclusions in Section 8.

2 Background and Related Work

This section provides a brief overview of relevant background and related studies. Most reports on participation of firms in OSS projects present results from large-scale surveys.

Bonaccorsi and Rossi studied contributions to OSS projects by commercial firms. They conducted a large-scale survey among 146 Italian companies that provide software solutions and services based on Open Source Software [6]. One of the findings was that approximately 20 per cent of companies were coordinating an OSS project. Furthermore, almost half of the companies (46.2%) had never joined an OSS project. It is important to note that these results were published in 2004, and that these numbers may have changed significantly over the last eight years; we suggest that a replication of this study would be a valuable contribution.

Bonaccorsi and Rossi have further studied (using data from the same survey) motivations of firms to contribute to OSS projects [7, 29].

Bonaccorsi et al. [5] have investigated whether and how firms contribute to OSS projects. Their study investigated which activities firms undertake in OSS projects, as well as whether the presence of firms affect the evolution of OSS projects. To address these questions, Bonaccorsi et al. conducted a survey of 300 OSS projects hosted on SourceForge.net. They found that almost one in three of the studied projects had one or more firms involved. In a survey of 1,302 OSS projects by Capra et al. [13], similar

results were found, namely that firms were involved in 31% of the projects. Different types of involvement were identified: (1) project coordination, (2) collaboration in code development, and (3) provision of code. Capra et al. [13] made a slightly different classification of participation models: the *Management model* (for project coordination), the *Support model* (sponsoring through financial or logistic support) and the *Coding model* (contributing code, bug fixes, customization, etc.). In most cases, it was found that the firms founded the OSS project, but in some cases firms took over by replacing a project's coordinator.

Aaltonen and Jokinen [1] studied the influence in the Linux kernel community and found that firms have a large impact in the project's development.

Martinez-Romo et al. [22] have studied collaboration between an OSS community and a company. They conducted case studies of two OSS projects: Evolution and Mono.

Companies can sponsor OSS projects in different ways. Berdou [4] investigated the dynamics of cooperation in community-led projects that involve paid contributors, and proposed a framework to understand this relationship.

Dahlander and Magnusson [15] proposed a typology consisting of *symbiotic* (win-win), *commensalistic* (firm gains, community indifferent) and *parasitic* (firm gains, community loses) approaches to characterize firm–community relationships. These relationships only apply in community-led projects.

The last decade of research in OSS has well established the relationship between firms and OSS projects. This relationship has been shown to have a direct effect on a project's sustainability. However, what kind of effects this relationship has on a project's evolution has not been studied. Therefore, we set out to explore this by means of a comparative case study. The next section outlines the research design.

3 Research Design

This section presents the research design of the empirical study following the *Goal-Question-Metric* (GQM) approach [3].

3.1 Goal

The long term objective of this research is to understand whether there are (and there likely will be) differences in the maintenance and evolution activities of OSS projects as long as commercial stakeholders join or drive the development.

3.2 Questions

This paper addresses the following research questions:

1. Are there differences in the evolution of similar-scoped OSS applications, as long as one (or more) commercial stakeholders play a major role in the development?

2. When considering projects in the same application domain, are different "categories" achieving different results or patterns of maintenance?
3. From an effort perspective, do commercial stakeholders behave similarly to individual developers?

3.3 Method and metrics

Given the exploratory nature of this topic, we decided to perform an exploratory multiple case study. Since this topic has not been studied in depth, this multiple case study can be considered as a *revelatory* case study [37]. Rather than seeking to make generalizations with respect to the influence of commercial stakeholders in OSS evolution, we have aimed at exploring this phenomenon with the purpose to identify more precise hypotheses that can be studied in more depth and with different research methods (e.g., surveys).

The choice of the studied projects was grounded in the fact that they are appropriate examples of the three types of involvement models mentioned earlier, and was also supported by the fact that the first author was familiar with these projects through previous studies [8, 9, 35] as well as ongoing (as of yet unpublished) studies.

Our study is a quantitative analysis of the studied projects, for which two types of metrics are used: *code* metrics and *effort* metrics. These are discussed below.

Code Metrics Given the available (public) releases, a set of data was extracted from the studied projects: two systems (Eclipse and jEdit) are implemented mostly in Java, while Moodle is implemented in PHP, and partially relying on OO features, evidenced by a visible number of PHP classes. The terminology and associated definitions for these metrics are extracted from related and well-known past studies, for example, the definition of *common* and *control* coupling ([2, 21, 16]).

– **Methods** (or functions in PHP): the lowest level of granularity of the present analysis. Within this attribute, the union of the sets of OO methods, interfaces, constructors and abstract methods was extracted.
– **Classes**: as containers of methods, the number of classes composing the systems has been extracted. Differently from past studies [25], *anonymous* and *inner* classes [20] were also considered as part of the analysed systems.
– **Size**: the growth in size was evaluated in number of SLOCs (physical lines of code), number of methods, classes and packages.
– **Coupling**: this is the union of all the *dependencies* and *method calls* (*i.e.*, the common and control coupling) of all source files as extracted through Doxygen[2]. The three aggregations introduced above (methods, classes and packages) were considered for the same level of granularity (the *method-to-method*, *class-to-class* and the *package-to-package* couplings). A strong coupling link between package A and B is found when many elements within A call elements of package B.
– **Complexity**: the complexity was evaluated at the method level. Each method's complexity was evaluated via its McCabe index [23].

[2] `http://www.doxygen.nl`, supporting both the Java and PHP languages

Effort Metrics A second set of data was extracted based on the availability of CMS servers: this data source represents a regular, highly parsable set of atomic transactions (i.e., 'commits') which details the actions that developers (i.e., 'committers') perform on the code composing the system. Two metrics were extracted:

- **Effort**: the effort of developers was evaluated by counting the number of unique (or *distinct*, in a SQL-like terminology) developers in a month.
- **Output metrics**: the work produced was evaluated by counting the monthly creations of, or modifications to, classes or packages. Several modifications to the same file were also filtered with the SQL *distinct* clause, in order to observe how many different entities were modified in a month[3].

4 Industry-led Open Source Project: Eclipse IDE

The Eclipse project has attracted a vast amount of attention by researchers and practitioners, in part due to the availability of its source code, and the openness of its development process. Among the recent publications, several have been focused on the "architectural layer" of this system [34, 19], extracting the relevant information from special-purposed XML files used to describe Eclipse's features and extensions (*i.e.*, plugins) implementing them, in this way representing some sort of "module architecture view" [18].

As recently reported, the growth of the major releases in Eclipse follows a linearly growing trend [24], when studying the evolution of its lines of code, number of files and classes. The study on Eclipse's meta-data indicated that, over all releases, the size of the architecture has increased more than sevenfold (from 35 to 271 *plugins*) [34].

The present study is instead performed at the method level, and on two release streams (*trunk* and *milestones*). Regarding Eclipse, 26 releases composing the stream of "major" and "minor" releases of Eclipse (from 1.0 to 3.5.1) and some 30 additional releases tagged as "milestones" (M) or "release candidates" (RC), were considered in this study, spanning some 8 years of evolution. For each release, we performed an analysis of the source code with the Doxygen tool. This latter analysis lasted a few hours for the early releases, but it required more than one day of parsing for the latest available releases, mostly due to the explosion in size of the project (490,000 SLOCs found in the 1.0 release of Eclipse, up to more than 3 million SLOCs found in the 3.6 releases[4]). Overall, it required more than one month to perform the analysis on the whole batch of Eclipse releases.

The remainder of this section presents the results of the analysis of Eclipse. Subsection 4.1 presents the results of the evolution of the size of Eclipse. Subsection 4.2 presents the evolution of Eclipse's complexity.

[3] In specific cases, specific committer IDs were excluded, when it was clear that they are responsible for automatic, uninteresting, commits; it was also excluded from this metric any activity concerning the 'Attic' CMS location (which denotes deleted source material).

[4] Statistics were collected with SLOCCount, `http://www.dwheeler.com/sloccount/`

4.1 Results – Eclipse Size

This study considered the "main" releases (3.0, 3.1, etc.), and the "milestone" releases (e.g., 3.2M1, 3.2M2, etc.) and "release candidates" (e.g., 3.3RC1, 3.3RC2, etc.) release streams of the Eclipse project. The overall growth is almost fivefold, while it is also evident from Figure 2 that the main stream of releases has a stepwise growth, the steps being the major releases[5].

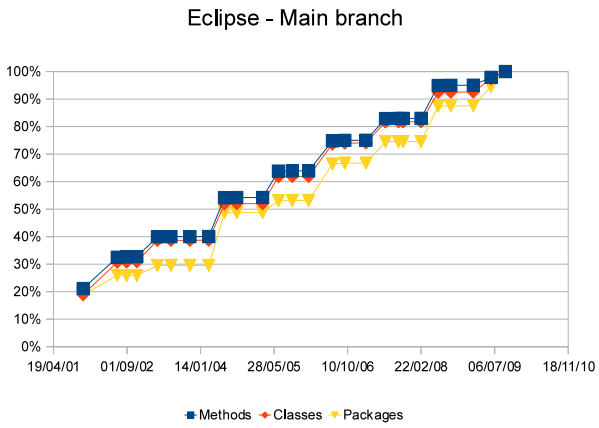

Fig. 2. Growth in the "main" branch of Eclipse

Major releases of Eclipse are regularly devoted to new features, while milestone and release candidates releases are devoted to maintaining existing ones (Figure 3). The milestones stream has a more linear path: plotting the number of methods against the "build date" of the relative release, a linear fit is found with an appropriate goodness of fit ($R^2 = 0.98$). The step-wise growth for the main release stream, and the linear trend for the milestones release also reflect what was found when studying the evolution of Eclipse at a larger granularity level, i.e. its plugins [34].

4.2 Results – Eclipse Complexity

The study at the method level shows a distribution of the McCabe cyclomatic indexes which is constant along the two streams of releases (main and milestones) of Eclipse. This is visible when assigning the cyclomatic complexity of each method (cc_i) in the four following clusters:

1. $cc_i < 5$
2. $5 \leq cc_i < 10$
3. $10 \leq cc_i < 15$

[5] The overall size growth has been normalized to 1 for easing the reading of the graph.

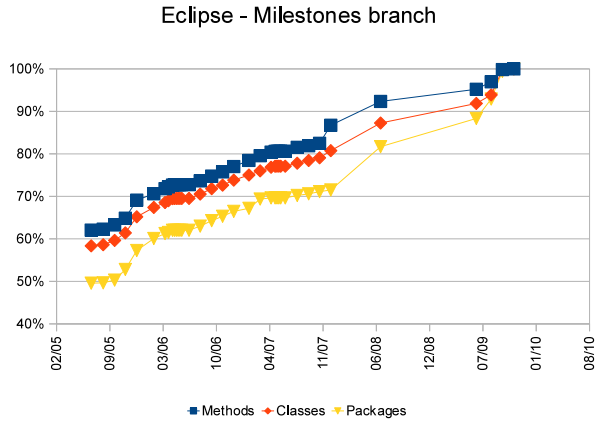

Fig. 3. Growth and maintenance patterns in the "milestones" branch of Eclipse.

4. $cc_i \geq 15$

Figure 4 shows the relative evolution of the fourth cluster, and reveals a quasi-constant evolutionary trend (for reason of clarity, the other trends are not displayed, although they follow a similar evolutionary pattern). The amount of highly complex methods ($cc > 15$, [23]) present in the system never reaches the 2% of the overall system. As reported in other works, this shows a profound difference from other traditional Open Source projects, where this ratio (for C and C++ projects) has been observed at around 10% of the system [10].

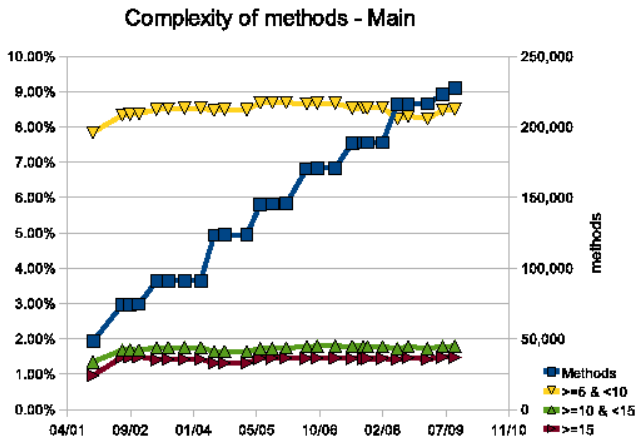

Fig. 4. Patterns of highly complex methods (McCabe index > 15) in the main branch of Eclipse.

4.3 Results – Eclipse Coupling

The number of couplings (i.e., unique method calls) has been counted for each of the two streams of releases. The set of added, deleted and kept couplings has been evaluated between two subsequent releases in each stream, and plotted in Figure 5. As shown, these findings confirm previous ones [34] regarding Eclipse's maintenance patterns: in the main stream, a large amount of modifications to its existing connections is made between minor and major releases, reaching more than 60% of new couplings added during the transition between the subsequent versions 2.1.3 and 3.0.

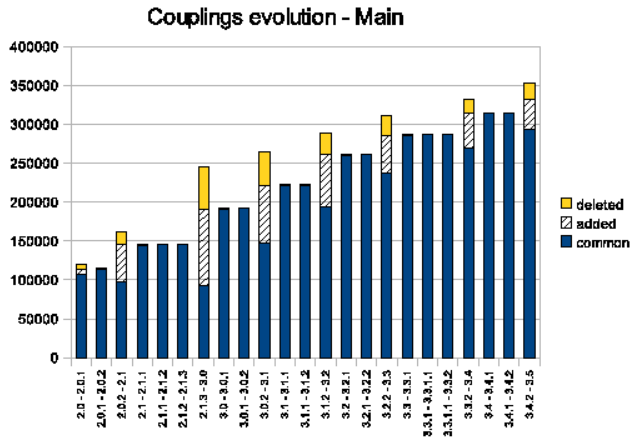

Fig. 5. Distribution of coupling in the main branch of Eclipse.

On the other hand, the Milestones stream (Figure 6) confirms a recurring pattern, where the milestones show a great deal of added and removed couplings, whereas the Release Candidates (RC's) show a much lower activity in the same activity of coupling restructurings (the amount of shared couplings between two subsequent releases is not shown for clarity purposes).

4.4 Results – Eclipse Cohesion

The cohesion of classes or packages was measured by counting the number of elements connected with other internal elements, and then cumulated for all the classes or packages. Figure 7 shows the evolution of cohesion at the package level, and it confirms the observations achieved when evaluating the highly complex methods (Figure 4). Although there is a vast increase in the number of methods and classes, most of the connections are confined within the same package, keeping the cohesion constant throughout the life-cycle until the latest observed release. This measurement is also found higher in the earliest releases (some 73%), and declining sharply until release 3.0, where it stabilizes to some $69 - 70\%$ for the last 6 years.

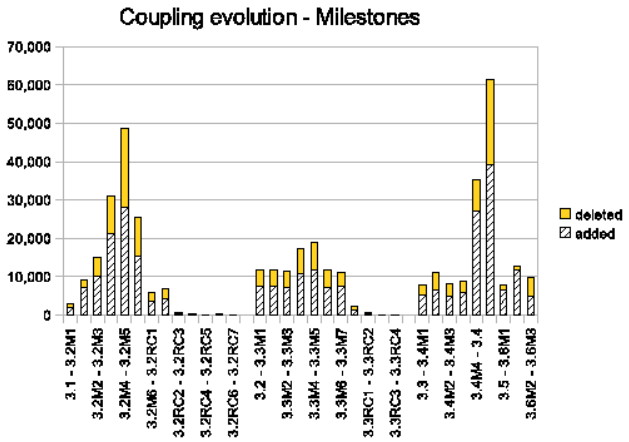

Fig. 6. Distribution of coupling in the milestones branch of Eclipse.

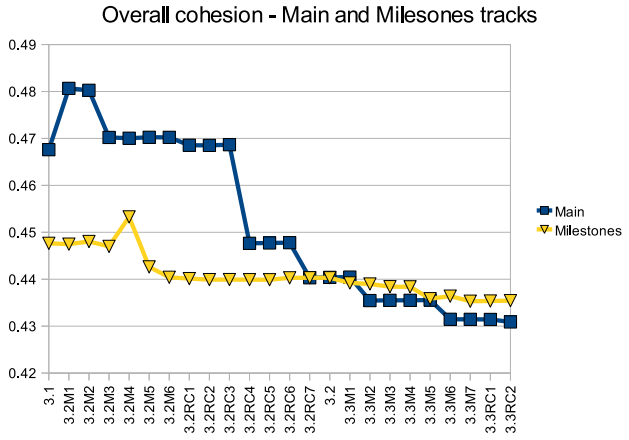

Fig. 7. Patterns of cohesion of the two branches of Eclipse.

5 Traditional Open Source Project: jEdit

Given the results from the above study, a *community-driven* OSS project (i.e., where no commercial company is "sponsoring" the development [12]) was studied in a similar way to evaluate and compare in some way the quantitative results of Eclipse. Although not exactly implementing all the features within Eclipse, the jEdit project also aims to be a fully-fledged IDE, benefiting from a large number of add-ons and plugins, independently developed and pluggable in the core system. Though any two software systems are always different to some degree, this study was not performed for the purpose of comparing features, but for the sake of observing whether the patterns observed

in a very large and articulated project are similarly found in a much smaller project, and whether good practices should be inferred in any direction.

Similarly to the Eclipse project, the 14 releases available of jEdit were therefore collected on the largest OSS portal (*i.e.*, SourceForge), from 3.0 to 4.3.1 (earlier releases do not provide the source code). Being a much smaller project, collecting the information via Doxygen was much quicker, both at the beginning of the sequence (57 kSLOCs, jEdit-3.0) and at the end (190 kSLOCs, jEdit-4.3.1). The 14 considered releases are the ones made available to the community, and span some 10 years of development.

5.1 Results – jEdit Size

The second system also shows a linear growth, with an adequate goodness of fit ($R^2 = 0.97$), albeit with a lower slope than what found in Eclipse, as to summarise a slower linear growth in Figure 8. A similar linear trend is found in the evolution of methods, classes and packages. The most evident difference with the evolution of Eclipse is the pace of the public releases in jEdit: between releases 4.2 and 4.3 some 5 years passed, although the jEdit configuration management system contains information on the ongoing activity by developers.

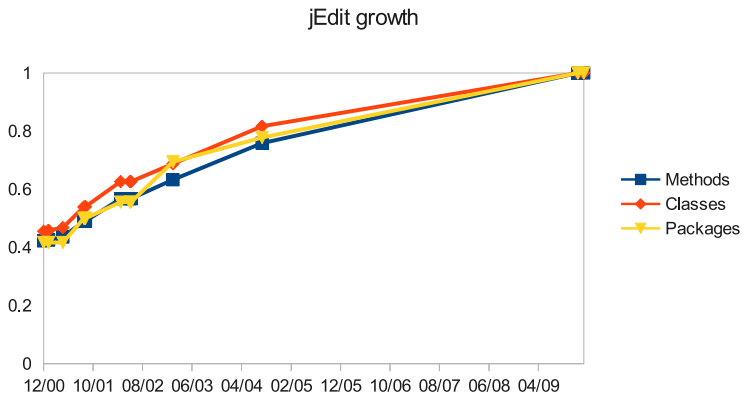

Fig. 8. Evolution of size in jEdit

5.2 Results – jEdit Complexity

Regarding jEdit, the evolution of the complexity at the methods' level brings an interesting insight: for this project, it was found that more than 25% of the methods are constantly over a threshold of high complexity, at any time of jEdit's evolution. This complexity pattern has been observed also in other OSS systems [10]. Large and complex methods are typically a deterrent to the understandability and maintainability of a

software system, and a vast refactoring of these methods has been achieved in the last two public releases, as visible in the graph, where a significant drop of highly complex methods is achieved even in the presence of a net increase in the number of methods.

5.3 Results – jEdit Coupling

The maintenance patterns of jEdit present a more discontinuous profile, with changes between major releases typically presenting large additions of new couplings (see Figure 9, bottom), and minor releases where less of such modifications were made. More importantly, the maintenance of couplings appears not to be planned, where the largest modifications (between 4.2 and 4.3) appear after a long hiatus of five years, and represent a full restructuring of the underlying code architecture, with added and deleted couplings representing three-times and twice as many couplings as the maintained ones, respectively.

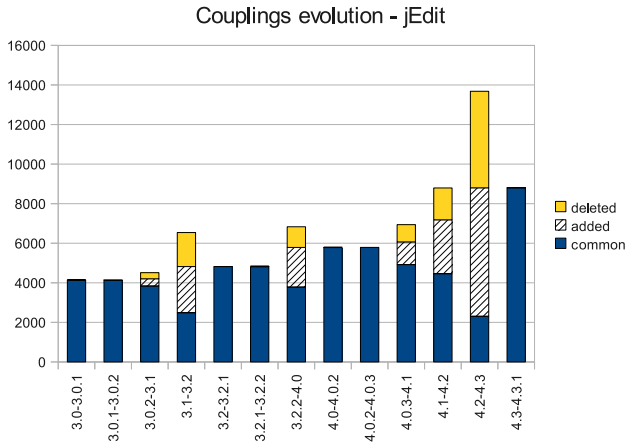

Fig. 9. Coupling in jEdit.

6 Industry-involved OSS Project: Moodle

As per the definition of an *industry-involved* OSS project, Moodle's development is primarily centered around the OSS community, but various other actors have interest in its development. A number of organizations across the world are directly contributing to the development of Moodle by way of funding or contributing their expertise, and have been defined as "Moodle partners".

Similarly to the other two case studies, we extracted the size, complexity and cohesion of the PHP code contained in the publicly available releases[6]: overall we studied some 90 releases of this project. By checking on the official website, it can be observed that Moodle was evolved in one single stream of release until version 1.7: from 1.8 onwards, several branches have been evolved at the same time (e.g., 1.7.x, 1.8.x, 1.9.x etc). For each of these branches we kept the results on size, coupling and complexity separated from the other branches.

6.1 Results – Moodle Size

As observed in Eclipse, the evolution of Moodle resembles a step-like pattern (see Figure 10), where the major releases consist of the addition of a large number of files, classes and functions, and the minor releases show smaller additions in all the measured metrics. From release 1.8 onwards, all the various branches maintain the same pattern as well, albeit the growth is intertwined in time with all the other branches (Figures 11 middle and bottom): during the interim releases between minor (e.g., 1.8) and development (e.g., 1.8.1) releases, the growth in number of functions, classes and source files is minimal, while the step-wise growth pattern is observed between minor releases (e.g., between 1.8 and 1.9). Therefore, for this system the increase in size has changed the approach to development, requiring the project to define and maintain various branches at the same time.

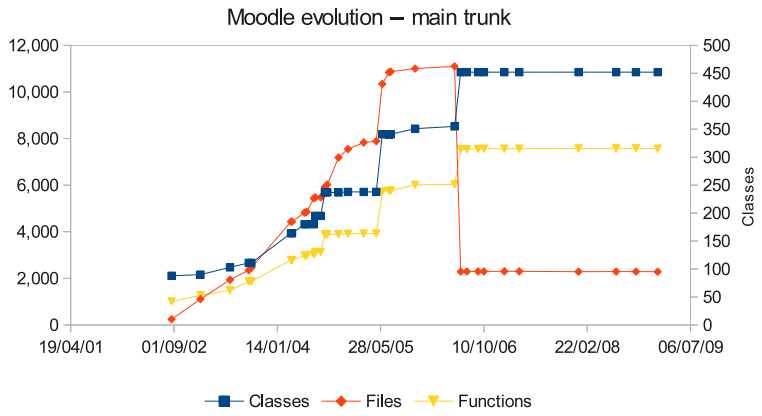

Fig. 10. Growth of size in the main branch of Moodle (up to release 1.7).

[6] A list of the releases (with the relative releasing date) since 2002 is available at `http://docs.moodle.org/dev/Releases`

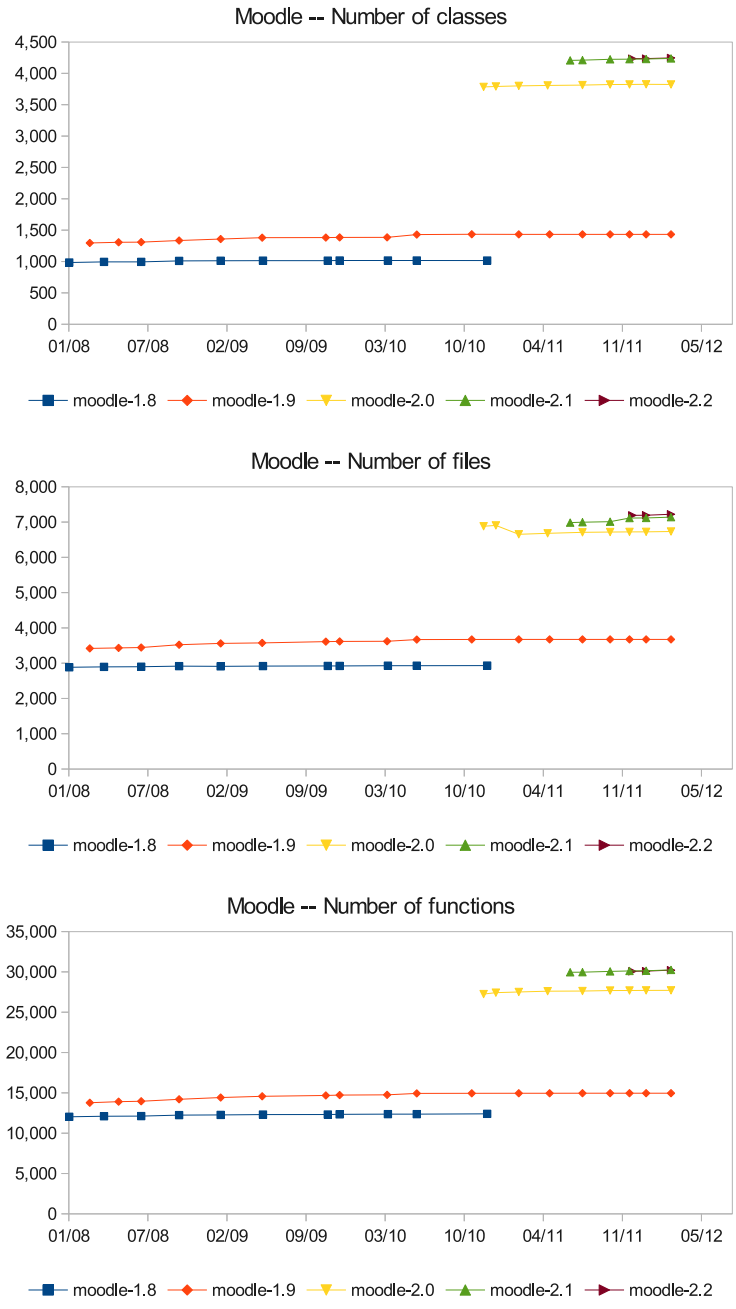

Fig. 11. Growth of size in the parallel branches of Moodle (after release 1.7).

6.2 Results – Moodle Complexity

Since Moodle is written in the PHP programming language, which is based on procedural and object-oriented constructs, we evaluated the complexity of the functions contained in the source code. This was plotted per release, as above, and the percentage of highly complex functions tracked throughout. The summary in Figure 12 shows how the excessive complexity (i.e., the sum of functions whose McCabe cyclomatic index is > 15, and depicted in the continuous line) has been kept under control even though the system constantly increases the number of its functions (depicted as a continuous line in the same figure). What is quite evident is also the major refactoring that was undertaken between releases 1.x and 2.x. In the latter, a larger number of functions were introduced, in a step-wise growth, while parallel work was done to reduce the amount of complexity in existing and new functions, with a step-wise descent of highly complex functions.

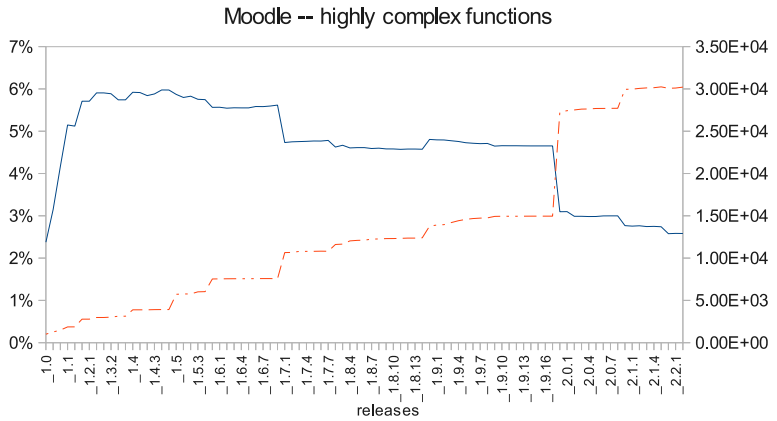

Fig. 12. Evolution of complexity in Moodle (continuous line) and overall increase in number of functions (dashed).

6.3 Results – Moodle Coupling

The functions composing the releases of Moodle were also analysed in terms of their connections, and which of the connections were added or removed between major and minor releases, and between branches. As done for the previous cases, the releases were analysed by the Doxygen engine, extracting all the links between low level entities, that were later lifted to file-to-file dependencies.

As reported for the size growth, it becomes clear that the minor and development releases have become central in Moodle to perform several adjustments, that trail off in proximity of the next release, similarly to what is found in the Eclipse environment (see Figure 13, displaying the 1.7.x and 1.8.x branches of releases). This has evolved

in Moodle: the earlier branches (e.g., Moodle-1.1.x or Moodle-1.2.x) did not display long sequences of development (e.g., only Moodle-1.1 and Moodle-1.1.1 have been released within the Moodle-1.1.x branch). With more recent releases, the pattern observed in Eclipse is also visible in Moodle, with longer sequences of development releases (14 development releases in Moodle-1.8.x, 17 in Moodle-1.9.x), in which fewer and fewer couplings are added and removed, until the release is being discontinued and not supported further.

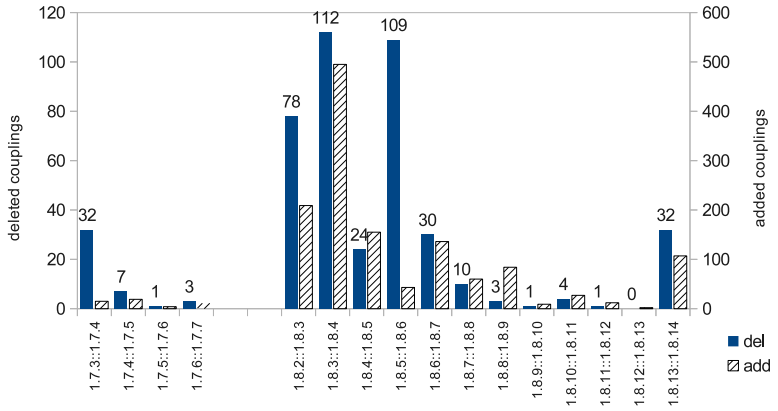

Fig. 13. Added and removed couplings in Moodle (branches 1.7.x and 1.8.x).

7 Discussion

The two cases of Moodle and jEdit show that similar issues are faced by the developers: even if companies are involved in development of the Moodle project, they do not *drive* the development, as for Eclipse. Given it is taken for granted that industry-led OSS projects do not have an issue of long-term sustainability, industry-involved and traditional OSS projects need to address the issue of how to attract and maintain the existing contributors in the development loop. In the following subsections, we analyse how effectively developers and contributors are attracted and maintained within the two projects, and whether lessons learned can be drawn in both cases.

7.1 Contributions on the Periphery

In both the Moodle and jEdit projects, the “core” of the system is separated from the “plugins” or “contributors” section. We assume that contributing to the “core” of a project is more time-consuming, and requires more skills, than contributing to the

"modules" or the "plugins" sections[7]. Therefore we investigated whether a sustained intake of contributors is achieved in Moodle and jEdit, or whether these projects face an issue in this respect.

Moodle – Two main directories are found in the CMS server: the core 'Moodle' directory (which makes for the public releases, that we consider as "core"), and the 'contrib' folder, organized in 'plugins', 'patches' and 'tools' (but not wrapped in the official releases). As visible in Figure 14 (left), the evolution of the core Moodle system follows the typical pattern of an early (or 'cathedral' [26]) OSS project: few contributors are visible in the first months (mostly the main Moodle developer), with few other contributors being active in a discontinuous way. A further, sustained period is also visible, where the number of active developers follows a growing trend with peaks of over 30 developers a month contributing, and revealing a 'bazaar' phase [11]. The main issue that is visible in the Moodle "core" system is revealed at around $3/4$ of its life-cycle, where the number of active developers start to decline. From the point of view of the sustainability, we posit that this could represent a serious issue in the long-term evolution of this system.

On the other hand, the activity of Moodle has been devoted more and more to the 'contrib' folder, rather than in the 'core': this reflects a more and more distributed participation to the Moodle development, and a low barrier to entry, albeit not all the contributed modules are selected for inclusion in the publicly available releases. The overall distribution of changes throughout the Moodle evolution proceeds on a linear trend ($R^2 = 0.78$): in recent months, the inflection of productivity in the "core" Moodle has been balanced by the late growth of contributions to the other parts. That reflects a more and more distributed participation to the Moodle development, and a low barrier to entry, but several of the proposed modules have not been selected for inclusion in the main Moodle system.

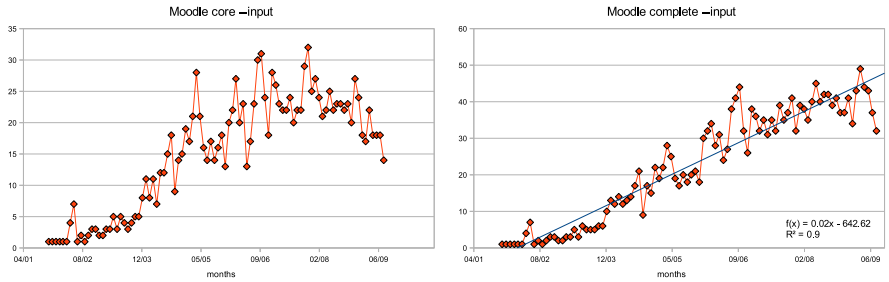

Fig. 14. Active monthly contributors in the "core" (left) and in the overall Moodle project (right)

jEdit – The main difference between jEdit and Moodle in the intake of developers is visible in Figure 15 (below): albeit the 'core' (or 'trunk') is separated from the

[7] This is because writing plugins or additional modules, where the system is modular, should be possible without modifying other files, but just using the system's APIs.

'plugins', few contributors were added in the latter, following a cyclic development pattern overall. Differently from Moodle, the intake of contributors does not follow a linear pattern: the presence of developers in the "core" declines at around $3/4$ of the life-cycle, and so does the number of contributors working on the periphery of the system. This makes jEdit even more brittle to sustainability issues, specifically around the intake of new developers.

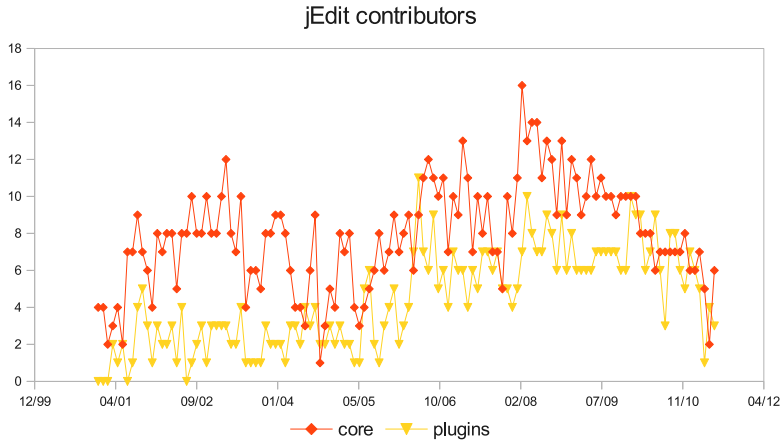

Fig. 15. Active monthly contributors in the "core" and in the "plugins" parts of jEdit

7.2 Three-layered Contributions

The study of Moodle as an industry-involved OSS project resulted in an in-depth analysis of the types of contributors who actively produce code for the system. Interesting insights were discovered when studying each developer's actual contribution to the code: in a first attempt to categorize the intake, the contributions, and the developers leaving the project, three categories are clearly distinguishable, not based on the amount of effort inputed in the system, but purely on the length of the activity of each developer:

1. **Sporadic** developers: this refers to the extremely low presence of certain contributors in the development. Within Moodle, 60 developers have been active for just one month; other 70 developers have been active between 2 and 6 (not necessarily consecutive) months.
2. **Seasonal** developers: as reported recently [28], most OSS projects benefit seasonal developers, i.e., those developers who are active for a short period of time (we are not referring to 'recurring' or 'returning' developers).
3. **Stable** developers: those developers showing a sustained involvement (say, more than 24 months for the Moodle system). Both seasonal and stable developers can

be part of the top 20% developing most of the system, as in the definition of 'generation of OSS developers' given in the past [5].

Some of the Moodle partners have been found acting as *seasonal* developers; the *Catalyst* partner[8] has so far provided a large number of modifications to the core Moodle, by deploying several developers who became active contributors within the community. The profile of the contributed outputs is visible in Figure 16 (bottom), and can be defined as a 'seasonal' effort pattern, meaning a large contribution on a very specific time interval, and lower levels of effort before and after it. Comparing this curve to a selection of seasonal Moodle individual developers (Figure 16, top), a similar pattern is visible: an initial period of low commit rates, followed by a peak were a high level of contributions is observed, finally a leveling-off.

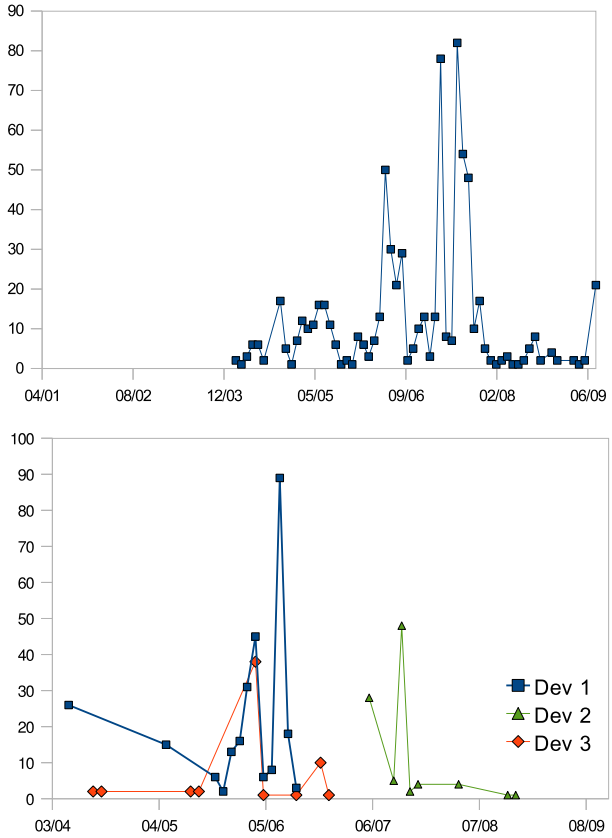

Fig. 16. Output produced by one of the partners (Catalyst, top), as compared to seasonal developers in Moodle

[8] http://www.catalyst.net.nz/

7.3 Limitations of this study

We are aware of a few limitations of this study, which we discuss below. Yin [37] lists four types of threats to validity, namely, *construct*, *internal* and *external* validity, and *reliability*.

Construct validity Construct validity is concerned with *establishing correct operational measures for the concepts that are studied* [37]. In this study, construct validity relates to the measures we have used to collect and analyze the data, namely, code metrics such as size, coupling and complexity, and effort metrics such as number of developers and number of modifications made. We argue that these are well established metrics that appropriately represent the concepts being studied.

Internal validity Internal validity is concerned with establishing a causal relationship. In our study, the relationship that we have explored is between the nature of the stakeholders (i.e., commercial versus non-commercial) and the evolution of OSS. The results of this exploratory study suggest that there is, in fact, an influence from the presence of commercial stakeholders. However, changes in evolutionary patterns may not be due to the involvement of commercial stakeholders. Further research is needed to establish the nature of this relationship in more detail.

External validity External validity is concerned with the extent to which findings of a study can be generalized to other settings. A common critique of the case study methodology is that findings cannot be generalized. However, the purpose of conducting case study research is not to look for *statistical* generalizability, such as aimed for in large-scale quantitative surveys, but rather to seek *theoretical* generalization [37]. In other words, in this paper we have started to explore a theory relating to the influence of commercial stakeholders on the evolution of OSS. We like to emphasize that our study is of *exploratory* nature, and as such serves the purpose of exploring our initial ideas and defining more focused hypotheses for further research.

Reliability Reliability of a study refers to the degree to which a study can be repeated and attaining the same results. One strategy to increase a study's reliability is to establish an audit trail [17]. Our audit trail consists of the extracted data as well as spreadsheets that contain the analysis.

8 Conclusions and Future Work

The terminology around the OSS phenomenon has been radically changing in the past few years. This research has studied how commercial stakeholders can have an influence on the evolution and maintenance of OSS systems. Eclipse has been studied as an *industry-led* OSS system, since it is backed by the IBM corporation; the Java IDE jEdit was selected as an exemplar of a *traditional* OSS system; while Moodle was chosen as an exemplar of *industry-involved* system, built mostly by the OSS community, although several commercial stakeholders have write-access to it. The public releases

of each system, and their configuration management systems (CMS), were jointly analyzed, to determine the best type of information to draw results from.

The study of the releases allowed us to focus on the main points along the evolution of the studied systems. The industry-led OSS system presents several "best practices" of software engineering: low complexity of units, continuous evolution and regular maintenance cycles. The traditional OSS system, in the same application domain, achieves very different results: 1 in 4 units are too complex, discontinuous evolution, and the maintenance is not regularly achieved. Finally, the industry-involved system shows more and more regular patterns of evolution, increasing control of complexity and alignment of its maintenance cycles to multi-branch, large software systems with parallel maintained releases.

On the other hand, the study of the CMSs allowed the effort of the contributors to be tracked along the life-cycle of these systems, with the specific objective of determining issues in the sustainability of OSS systems. Analysing the industry-led project, we posit that it does not present (yet) issues of sustainability, as it is backed by a large corporation. The industry-involved project shows that the amount of active developers and the output produced follow an increasing, linear trend. Factors for these trends were found in the increasing number of contributions and plug-ins, and the presence of commercial partners driving the evolution, that act exactly as typical developers, joining in the projects, producing contributions, and then leaving. As observed, and different from Eclipse, the studied commercial stakeholder in Moodle is a *seasonal* contributor, after some time trailing off and leaving the project.

The study of the effort in the traditional system shows instead that, even with a sustained number of releases, jEdit has fewer and fewer developers in both the "core" system as well as in the periphery, showing more issues of sustainability than the other two cases.

What these findings demonstrate could have a profound impact on what is considered as "Open Source" development and raises the following questions:

– Is the presence of commercial stakeholders a necessary condition to achieve sustained evolution?
– Are "traditional" OSS projects eventually destined to trail off and be abandoned?
– Is the lack of adherence to basic software engineering principles an obstacle to OSS development?

These are fundamental questions to be answered by further research studies in order to understand how the OSS phenomenon will change in the future.

Acknowledgments

The authors wish to thank Dr Fernández-Ramil for his extensive comments on an earlier draft of this paper. We thank the two anonymous reviewers who provided constructive feedback on this paper. This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero—The Irish Software Engineering Research Centre (`www.lero.ie`).

References

1. T. Aaltonen and J. Jokinen. Influence in the linux kernel community. In J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, editors, *Open Source Development, Adoption and Innovation*, pages 203–208. Springer, 2007.
2. E. Arisholm, L. C. Briand, and A. Foyen. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):491–506, 2004.
3. V. R. Basili, G. Caldiera, and D. H. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley & Sons, 1994.
4. E. Berdou. Insiders and outsiders: paid contributors and the dynamics of cooperation in community led f/os projects. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, editors, *Open Source Systems*, pages 201–208. Springer, 2006.
5. A. Bonaccorsi, D. Lorenzi, M. Merito, and C. Rossi. Business firms' engagement in community projects. empirical evidence and further developments of the research. In *Proc. First International Workshop on Emerging Trends in FLOSS Research and Development*, Washington, DC, USA, 2007. IEEE Computer Society.
6. A. Bonaccorsi and C. Rossi. Contributing to os projects. a comparison between individual and firms. In *Proc. 4th Workshop on Open Source Software Engineering (WOSSE)*, pages 18–22, 2004.
7. A. Bonaccorsi and C. Rossi. Intrinsic motivations and profit-oriented firms. do firms practise what they preach? In *Proc. First International Conference on Open Source Systems*, pages 241–245, 2005.
8. A. Capiluppi, A. Baravalle, and N. W. Heap. Engaging without over-powering: a case study of a floss project. In P. Ågerfalk, C. Boldyreff, J. M. González-Barahona, G. R. Madey, and J. Noll, editors, *Open Source Software: New Horizons*, pages 29–41. Springer, 2010.
9. A. Capiluppi, A. Baravalle, and N. W. Heap. From "community" to "commercial" floss – the case of moodle. In *Proc. Third Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 11–16. ACM, 2010.
10. A. Capiluppi and J. Fernández-Ramil. Studying the evolution of open source systems at different levels of granularity: Two case studies. In *Proc. 7th International Workshop on Principles of Software Evolution (IWPSE)*, pages 113–118, Washington, DC, USA, 2004. IEEE Computer Society.
11. A. Capiluppi and M. Michlmayr. From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In J. Feller, B. Fitzgerald, W. Scacchi, and A. Silitti, editors, *Open Source Development, Adoption and Innovation*, pages 31–44. Springer, 2007.
12. E. Capra, C. Francalanci, and F. Merlo. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *IEEE Transactions on Software Engineering*, 34(6):765–782, 2008.
13. E. Capra, C. Francalanci, F. Merlo, and C. R. Lamastra. A survey on firms' participation in open source community projects. In C. Boldyreff, K. Crowston, B. Lundell, and A. Wasserman, editors, *Open Source Ecosystems: Diverse Communities Interacting*, pages 225–236. Springer, 2009.
14. K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys*, 44(2), 2012.
15. L. Dahlander and M. G. Magnusson. Relationships between open source software companies and communities: Observations from nordic firms. *Research Policy*, 34(4):481–493, 2005.

16. N. E. Fenton and S. L. Pfleeger. *Software metrics: a practical and rigorous approach*. Thomson, 1996.
17. E. Guba. Criteria for assessing the trustworthiness of naturalistic inquiries. *Educational Communication and Technology*, 29(2):75–91, 1981.
18. C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture: A Practical Guide for Software Designers*. Addison-Wesley Professional, 2000.
19. D. Hou. Studying the evolution of the Eclipse Java editor. In *eclipse '07: Proc. OOPSLA workshop on eclipse technology eXchange*, pages 65–69, New York, NY, USA, 2007. ACM.
20. A. Igarashi and B. C. Pierce. On inner classes. *Information and Computation*, 177(1):56–89, 2002.
21. W. Li and S. Henry. Object-oriented metrics that predict maintainability. *The Journal of Systems and Software*, 23(2):111–122, 1993.
22. J. Martinez-Romo, G. Robles, J. M. González-Barahona, and M. Ortuño-Perez. Using social network analysis techniques to study collaboration between a floss community and a company. In B. Russo, E. Damiani, B. L. Scott Hissam, and G. Succi, editors, *Open Source Development, Communities and Quality*, pages 171–186. Springer, 2008.
23. T. J. McCabe and C. W. Butler. Design complexity measurement and testing. *Communications of the ACM*, pages 1415–1425, December 1989.
24. T. Mens, J. Fernández-Ramil, and S. Degrandsart. The evolution of Eclipse. In *Proc. 24th International Conference on Software Maintenance (ICSM)*, pages 386–395, October 2008.
25. E. Merlo, G. Antoniol, M. Di Penta, and V. F. Rollo. Linear complexity object-oriented similarity for clone detection and software evolution analyses. In *Proc. 20th IEEE International Conference on Software Maintenance (ICSM)*, pages 412–416, Washington, DC, USA, 2004. IEEE Computer Society.
26. E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.
27. G. Robles, S. Dueñas, and J. M. González-Barahona. Corporate involvement of libre software: Study of presence in debian code over time. In J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, editors, *Open Source Development, Adoption and Innovation*, pages 121–132. Springer, 2007.
28. G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution of the core team of developers in libre software projects. In *Proc. 6th IEEE International Working Conference on Mining Software Repositories (MSR)*, pages 167–170, 2009.
29. C. Rossi and A. Bonaccorsi. Why profit-oriented companies enter the os field?: intrinsic vs. extrinsic incentives. In *Proc. 5th Workshop on Open Source Software Engineering (WOSSE)*, New York, NY, USA, 2005. ACM.
30. C. D. Santos Jr., G. Kuk, F. Kon, and R. Suguiura. The inextricable role of organizational sponsorship for open source sustainability. In *Proc. 2nd workshop Towards Sustainable Open Source*, 2011.
31. M. Schaarschmidt and H. F. von Kortzflieisch. Divide et impera! the role of firms in large open source software consortia. In *Proc. 15th Americas Conference on Information Systems (AMCIS)*, 2009.
32. B. Shibuya and T. Tamai. Understanding the process of participating in open source communities. In *Proc. 2nd Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, 2009.
33. M. Wermelinger and Y. Yu. Analyzing the evolution of eclipse plugins. In *Proc. International working conference on Mining Software Repositories (MSR)*, pages 133–136, New York, NY, USA, 2008. ACM.

34. M. Wermelinger, Y. Yu, and A. Lozano. Design principles in architectural evolution: a case study. In *Proc. 24th International Conference on Software Maintenance (ICSM)*, pages 396–405, 2008.
35. M. Wermelinger, Y. Yu, A. Lozano, and A. Capiluppi. Assessing architectural evolution: a case study. *International Journal of Empirical Software Engineering*, pages 623–666, 2011.
36. M. Wermelinger, Y. Yu, and M. Strohmaier. Using formal concept analysis to construct and visualise hierarchies of socio-technical relations. In *Proc. 31st International Conference on Software Engineering (ICSE), companion volume*, pages 327–330. IEEE, 2009.
37. R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, 3rd edition, 2003.