# Commercial Stakeholders in the Evolution of OSS Systems

Andrea Capiluppi[1] and Cornelia Boldyreff[2]

University of East London
{a.capiluppi, c.boldyreff}@uel.ac.uk

**Abstract.** It has been lately established that a major success or failure factor of an OSS project is whether it involves a commercial company, or more extremely, when the project management is in the hands of a commercial software corporation. As documented recently, the success of the Eclipse IDE can be largely attributed to the project management of IBM, 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 Free, Libre, Open Source (FLOSS) projects – the Eclipse and jEdit IDE's, 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 FLOSS 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 FLOSS is the involvement of a commercial company, or more radically, when project management is in hands of a commercial entity.

## 1 Introduction

Free, Libre, Open Source Software (FLOSS)[1] has been dramatically changing. The *classic* volunteer-based FLOSS project model is now being accompanied by *sponsored* FLOSS, where commercial stakeholders provide effort besides voluntary programmers. It has been argued that FLOSS projects have become increasingly hybrid with respect to the type of contributing stakeholders [11].

Since their inception in the early 1980s, FLOSS projects were mostly volunteer-based (Plain OSS, right end of Figure 1, adapted from [11]), heavily relying on personal efforts and non-monetary recognitions, and reportedly suffering from communication and coordination problems [13].

Nowadays, *Commercial OSS* projects have also been documented as more similar to *Closed Source* systems (as in far left of Figure 1), where a commercial stakeholder plays a major role in the development and decision making, as in

[1] FLOSS and OSS are used here as synonyms.

the case of the Eclipse IDE by IBM [21, 20, 15, 28]. Finally, *Community OSS* projects have also been reported, as more similar to pure OSS systems: they are driven by a FLOSS 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 [7].

**Fig. 1.** Types of OSS projects – a continuum

Both scenarios, "Commercial" and "Community" FLOSS, are creating 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[2]. In the Eclipse case, for example, the top contributors within have been identified as IBM staff, with only a few external developers acting on the core system [30]. For the second, specially in case of large and complex FLOSS 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 the system's core.

This paper aims to explore these two scenarios and to study whether the involvement of commercial companies can help sustaining the evolution of FLOSS projects. In order to do so, the paper presents different analyses of the evolution of a commercial and a plain 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 [24, 23, 26].

This paper is structured as follows: Section 2 presents the goals, questions and metrics of the study; Sections 3 presents the Eclipse and jEdit case studies, and compare their outcomes both in terms of size, and in the patterns of development. Section 4 focuses instead on the Moodle system as an example of a community OSS system, and explores the relevance of the commercial stakeholders, and how they differ from individual developers. Section 5 presents the conclusions.

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

## 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 [5]. 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 [6] [25].

Bonaccorsi et al. [4] have investigated whether andhow firms contribute to OSS projects. Their study investigated which activities firms undertake in OSS projects, as well as whether the prsence 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. [12], 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. [12] 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 Jokiinen [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. [18] have studied collaboration between a FLOSS community and a company. They conducted case studies of two FLOSS projects: Evolution and Mono.

[27]

### 2.1 Terminology

Several authors have reported on OSS projects that involves companies. However, existing terminology for this has some issues. For instance, OSS projects that are led by firms are referred to as "Commercial OSS", whereas OSS projects that involves 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. Commercial OSS suggests that profit is made from 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.

## 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 will be) differences in the maintenance and evolution activities of FLOSS 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 Metrics

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], [17], [13]).

- **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 [21], *anonymous* and *inner* classes [16] 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[3]. 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 [19].

**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:

- **Resources**: 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[4].

[3] `http://www.stack.nl/~dimitri/doxygen/`, supporting both the Java and PHP languages

[4] 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 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 [29, 15], 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" [14].

As recently reported, the growth of the major releases in Eclipse follows a linearly growing trend [20], 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 increases more than sevenfold (from 35 to 271 *plugins*) [29].

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[5]). 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.

### 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[6].

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

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

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

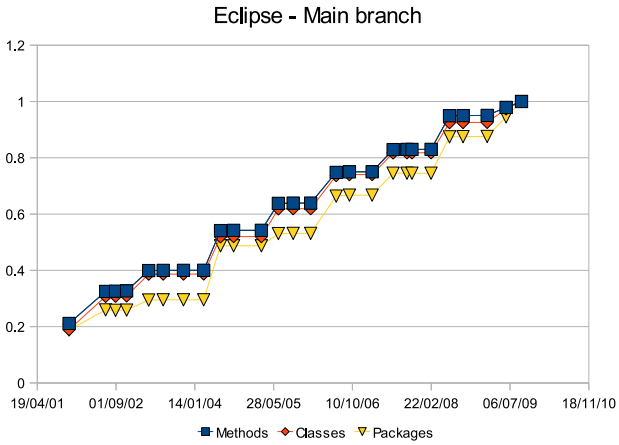


**Fig. 2.** Step-wise growth in the "main" branch.

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 [29].

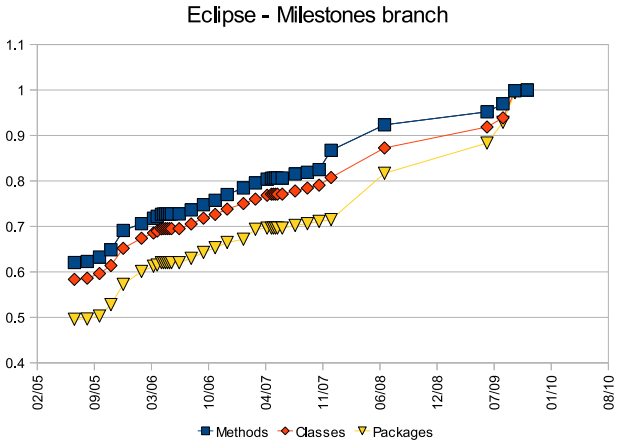


**Fig. 3.** Continuous growth in the maintenance patterns in the "milestones" branch of Eclipse.

### 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. $10 \leq cc_i < 15$
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 following a similar evolutionary pattern). The amount of highly complex methods ($cc > 15$, [19]) 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].

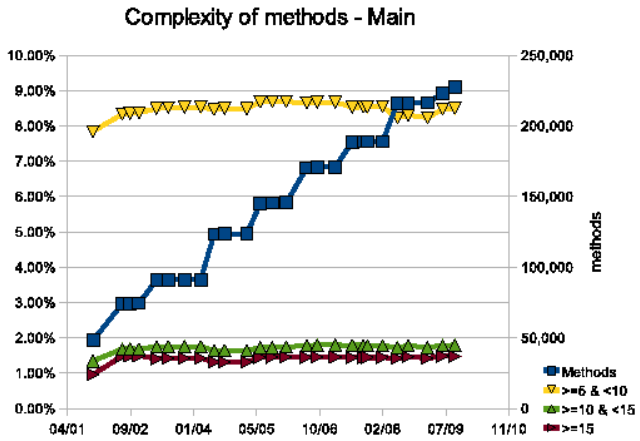


**Fig. 4.** Patterns of highly complex methods (McCabe index > 15) in the two branches of Eclipse.

### 4.3 Results – Eclipse Coupling

The amount 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 visible, these findings confirm previous ones [29] 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.

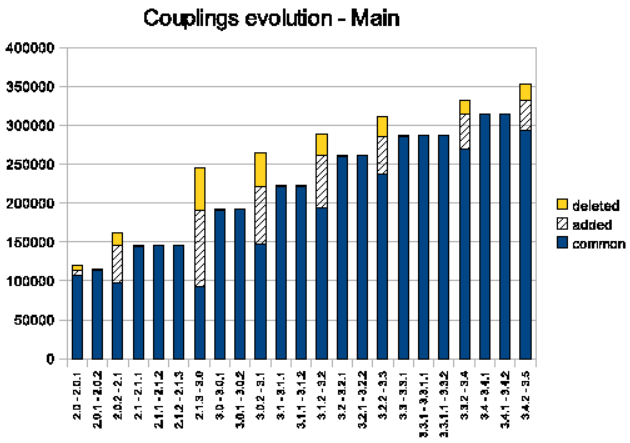


**Fig. 5.** Distribution of coupling in the main branch of Eclipse

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 amount of elements connected with other internal elements, and then cumulated for all the classes or packages. Figure 7 (right) shows the evolution of cohesion at the package level, and it confirms the observations achieved when evaluating the highly complex methods (Figure 7 left). Albeit 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 lifecycle 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 stabilises to some $69-70\%$ for the last 6 years. Table **??** illustrates the similarities of the findings in the patterns of evolution for the selected metrics.

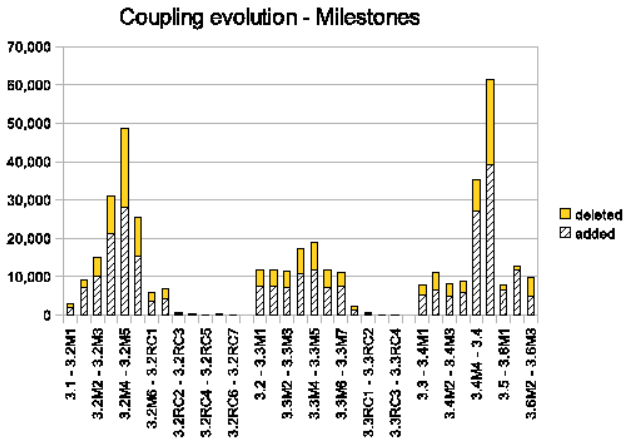


**Fig. 6.** Distribution of coupling in the milestones branch of Eclipse

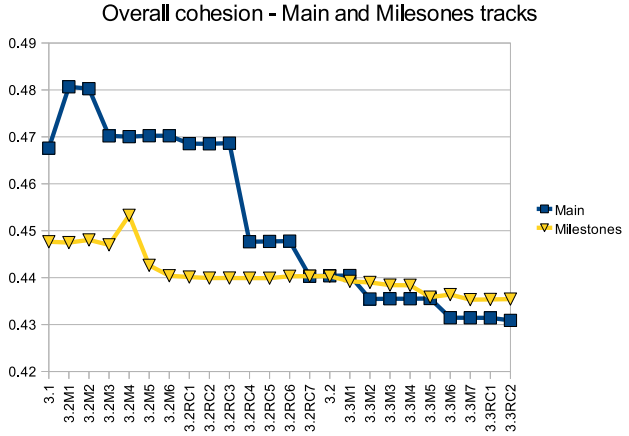


**Fig. 7.** Patterns of cohesion of the two branches.

## 5 Traditional Open Source Project: jEdit

Given the results from the above study, a *community-driven* FLOSS project (i.e., where no commercial company is "sponsoring" the development [11]) was studied in a similar way to evaluate and compare in some way the quantitative results of Eclipse. Albeit 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. Albeit 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 FLOSS 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

Also the second system 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.

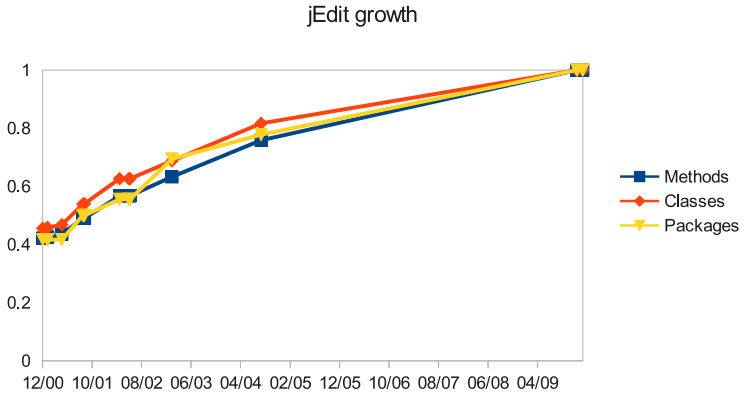


**Fig. 8.** Evolution of size in jEdit

### 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

FLOSS systems [8]. Large and complex methods are typically a deterrent to the understandability 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 relevant 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 appear not 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.

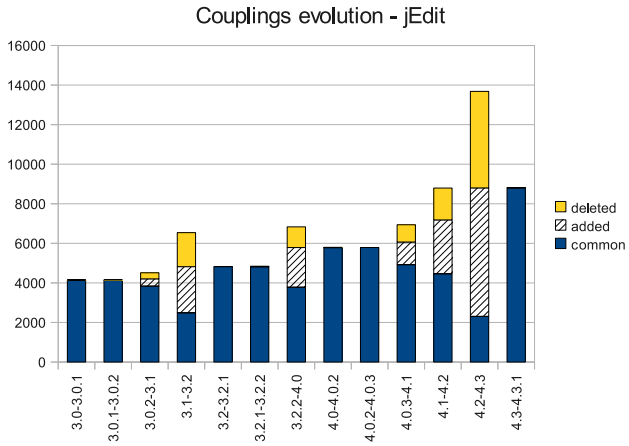


**Fig. 9.** Coupling in jEdit

## 6 Community-led Open Source Project: Moodle

As per the definition of a "Community 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 defined as "Moodle partners". The study presented here is based on the analysis of the Moodle CMS, that yields more interesting insights than the study of publicly available releases.

Similarly to the other two case studies, we extracted the size, complexity and cohesion of the PHP code contained in the publicly available releases [7]. The project maintained a 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 separated from the others.

### 6.1 Results – Moodle Size

As seen in Eclipse, the evolution of Moodle does resemble a step-like pattern (see Figure 10, top), 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 the 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 10 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 to define and maintain various branches at the same time.

### 6.2 Results – Moodle Complexity

Since Moodle is based on the PHP programming language, and this is based on functional and OO behaviours, 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 11 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 the releases 1.x and 2.x: in the latter, a larger number of functions were introduced, in a step-wise growth, while parellel work was done to reduce the amount of complexty in existing and new functions, with a spep-wise descent of highly complex functions.

[7] A list of the releases since 2002 is available at `http://docs.moodle.org/dev/Releases`

### 6.3 Results – Moodle Coupling

## 7 Discussion

### 7.1 Staffing and activity in Moodle

Two directories are found in the CMS server: the core 'Moodle' directory (which makes for the public releases), and the 'contrib' folder, organized in 'plugins', 'patches' and 'tools' (but not wrapped in the official releases). As visible in Figure 12 (top), the evolution of the core Moodle system follows the typical pattern of an early (or 'cathedral' [22]) 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 [9]. The main difference with the jEdit intake of developers is visible in Figure 12 (below): albeit the 'core' (or 'trunk') is separated from the 'plugins', few contributors were added in the latter, following a cyclic development pattern overall. For this plain OSS system, the intake of contributors does not follow a linear pattern.

On the contrary, 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.

### 7.2 Three-layered Contributions

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 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:

(a) **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.

(b) **Seasonal** developers: as reported recently [24], 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.

(c) **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 13 (left), 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 13, right), 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.

## 8 Conclusions and Future Work

The terminology around the OSS phenomenon have been radically changing in the past few years: this paper has studied how commercial stakeholders can have an influence on the evolution and maintenance of OSS systems. Eclipse has been studied as a "Commercial OSS" system, since it is backed by the IBM corporation; the Java IDE jEdit was selected as an exemplar of a "Plain OSS" system; while Moodle was chosen as an exemplar of a "Community OSS" system, built mostly by the OSS communities, although several commercial stakeholders have write-access to it.

The *commercial* OSS system presents several "best practices" of software engineering: low complexity of units, continuous evolution and regular maintenance cyles. The *plain* 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. The *community* OSS project shows that the amount of active developers and "net" output produced follow an increasing, linear trend. Factors for thede trends were found in the increasing amount of contributions and plug-ins, and the presence of commercial partners driving the evolution. Differently from Eclipse, the studied commercial stakeholder is a *seasonal* contributor, after some time peaking off and leaving the project.

What these findings demonstrate could have a profound impact on what we intend as "open source" development: is the presence of commercial stakeholders a necessary condition to achieve sustained evolution? are the "plain" OSS projects eventually destined to tail 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 in order to understand what the OSS phenomenon will become in the future.

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

## References


1. T. Aaltonen and J. Jokinen. Influence in the linux kernel community. In *OSS2007: Open Source Development, Adoption and Innovation (IFIP 2.13)*, volume 234/2007 of *IFIP International Federation for Information Processing*, pages 203 – 208. Springer, Springer, 2007/// 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*. John Wiley & Sons, 1994. See also http://sdqweb.ipd.uka.de/wiki/GQM.
4. A. Bonaccorsi, D. Lorenzi, M. Merito, and C. Rossi. Business firms' engagement in community projects. empirical evidence and further developments of the research. In *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development*, FLOSS '07, pages 13–, Washington, DC, USA, 2007. IEEE Computer Society.
5. A. Bonaccorsi and C. Rossi. Contributing to os projects. a comparison between individual and firms. pages 18–22, 2004.
6. A. Bonaccorsi and C. Rossi. Intrinsic motivations and profit-oriented firms. do firms practise what they preach? In *OSS2005: Open Source Systems*, pages 241–245, 2005.
7. A. Capiluppi, A. Baravalle, and N. W. Heap. Engaging without over-powering: a case study of a floss project. In *Proc. 6th Int'l Conf. on Open Source Systems*, June 2010.
8. A. Capiluppi and J. Fernández-Ramil. Studying the evolution of open source systems at different levels of granularity: Two case studies. In *IWPSE*, pages 113–118, 2004.
9. 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. International Federation for Information Processing, Springer, 2007.
10. A. Capiluppi and J. F. Ramil. Studying the evolution of open source systems at different levels of granularity: Two case studies. In *IWPSE '04: Proceedings of the Principles of Software Evolution, 7th International Workshop*, pages 113–118, Washington, DC, USA, 2004. IEEE Computer Society.
11. 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 Trans. Softw. Eng.*, 34(6):765–782, 2008.
12. E. Capra, C. Francalanci, F. Merlo, and C. R. Lamastra. A survey on firms' participation in open source community projects. In *OSS'09*, pages 225–236, 2009.
13. N. E. Fenton and S. L. Pfleeger. *Software metrics: a practical and rigorous approach*. Thomson, 1996.
14. C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture: A Practical Guide for Software Designers*. Addison-Wesley Professional, 2000.
15. D. Hou. Studying the evolution of the Eclipse Java editor. In *eclipse '07: Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pages 65–69, New York, NY, USA, 2007. ACM.

16. A. Igarashi and B. C. Pierce. On inner classes. *Information and Computation*, 177(1):56 – 89, 2002.
17. W. Li and S. Henry. Object-oriented metrics that predict maintainability. *J. Syst. Softw.*, 23(2):111–122, 1993.
18. J. Martinez-Romo, G. Robles, J. M. Gonzlez-Barahona, and M. Ortuo-Perez. Using social network analysis techniques to study collaboration between a floss community and a company. In *OSS'08*, pages 171–186, 2008.
19. T. J. McCabe and C. W. Butler. Design complexity measurement and testing. *Communications of the ACM*, pages 1415–1425, December 1989.
20. T. Mens, J. Fernández-Ramil, and S. Degrandsart. The evolution of Eclipse. In *Proc. 24th Int'l Conf. on Software Maintenance*, pages 386–395, October 2008.
21. E. Merlo, G. Antoniol, M. Di Penta, and V. F. Rollo. Linear complexity object-oriented similarity for clone detection and software evolution analyses. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 412–416, Washington, DC, USA, 2004. IEEE Computer Society.
22. E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.
23. 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, *OSS*, volume 234 of *IFIP*, pages 121–132. Springer, 2007.
24. G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution of the core team of developers in libre software projects. In *MSR '09: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 167–170, Washington, DC, USA, 2009. IEEE Computer Society.
25. C. Rossi and A. Bonaccorsi. Why profit-oriented companies enter the os field?: intrinsic vs. extrinsic incentives. In *Proceedings of the fifth workshop on Open source software engineering*, 5-WOSSE, pages 1–5, New York, NY, USA, 2005. ACM.
26. M. Schaarschmidt and H. F. von Kortzflieisch. Divide et impera! the role of firms in large open source software consortia. In *AMCIS 2009 Proceedings. Paper 309*, 2009.
27. B. Shibuya and T. Tamai. Understanding the process of participating in open source communities. May 2009.
28. M. Wermelinger and Y. Yu. Analyzing the evolution of eclipse plugins. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, pages 133–136, New York, NY, USA, 2008. ACM.
29. M. Wermelinger, Y. Yu, and A. Lozano. Design principles in architectural evolution: a case study. In *Proc. 24th Intl Conf. on Software Maintenance*, pages 396–405, October 2008.
30. M. Wermelinger, Y. Yu, and M. Strohmaier. Using formal concept analysis to construct and visualise hierarchies of socio-technical relations. In *Proc. 31st Int'l Conf. on Software Eng., companion volume*, pages 327–330. IEEE, May 2009.

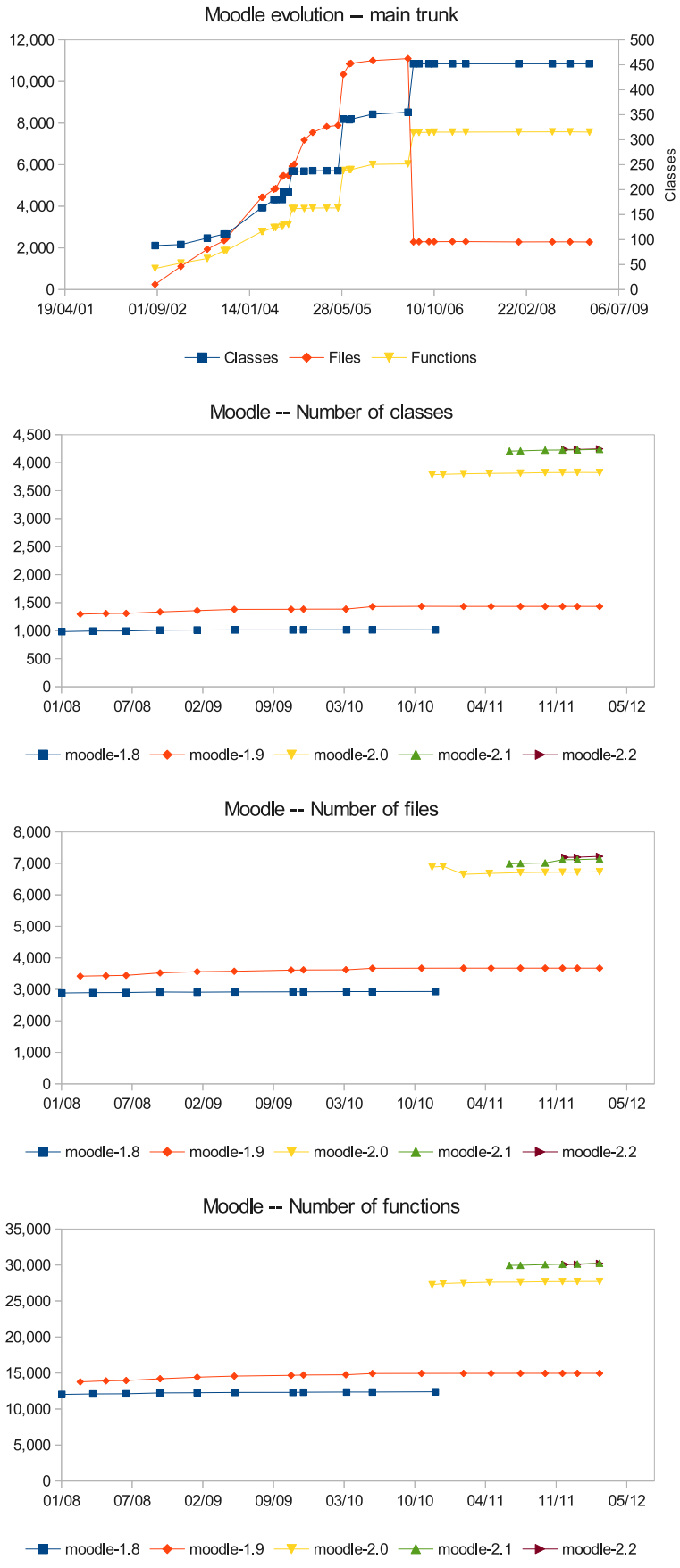


**Fig. 10.** Size in the main branch of moodle (up to release 1.7)

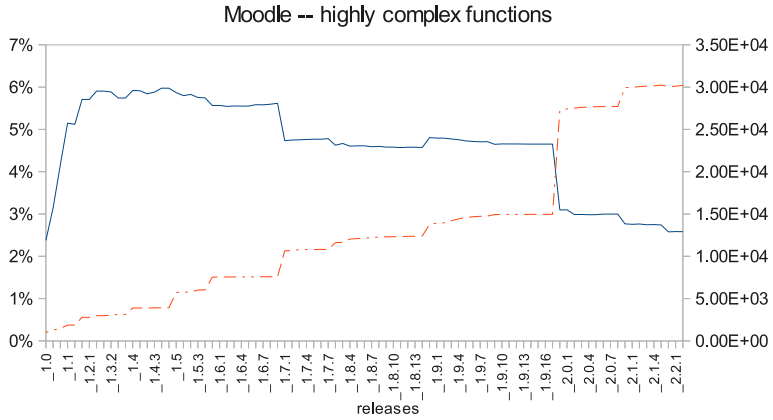


**Fig. 11.** Complexity in Moodle

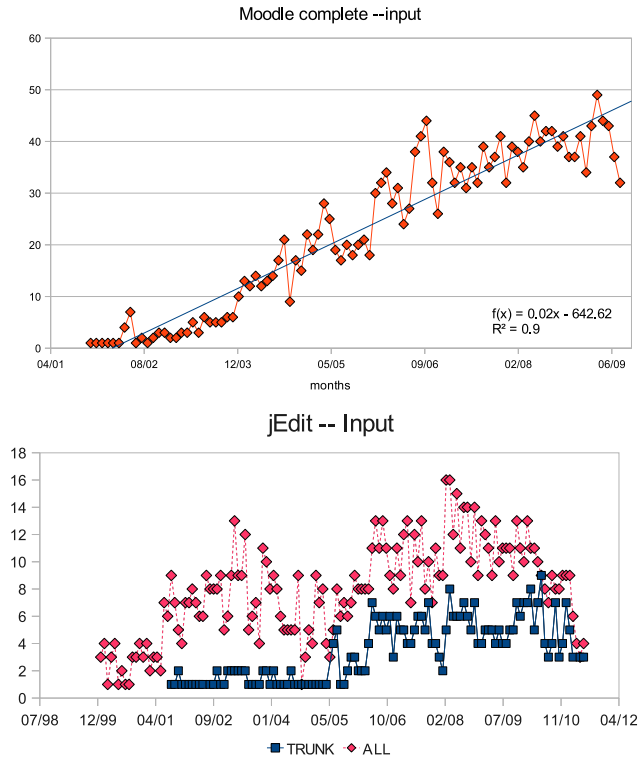


**Fig. 12.** Growth of Moodle

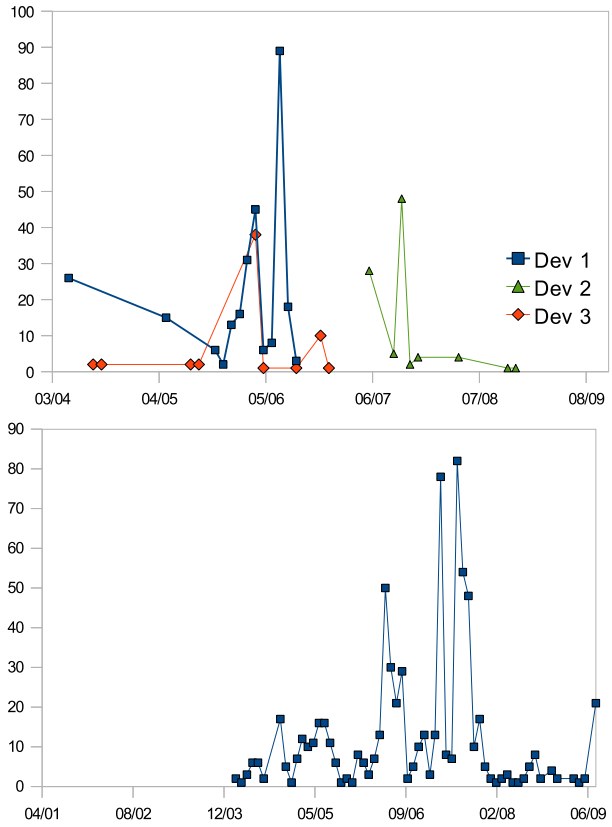


**Fig. 13.** Output produced by one of the partners (Catalyst, left), as compared to seasonal developers in Moodle