From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects

Andrea Capiluppi[1], and Martin Michlmayr[2]
1 University of Lincoln, UK, acapiluppi@lincoln.ac.uk
2 University of Cambridge, UK, martin@michlmayr.org

Abstract. Some free software and open source projects have been extremely successful in the past. The success of a project is often related to the number of developers it can attract: a larger community of developers (the `bazaar') identifies and corrects more software defects and adds more features via a peer-review process. In this paper two free software projects (Wine and Arla) are empirically explored in order to characterize their software lifecycle, development processes and communities. Both the projects show a phase where the number of active developers and the actual work performed on the system is constant, or does not grow: we argued that this phase corresponds to the one termed 'cathedral' in the literature. One of the two projects (Wine) shows also a second phase: a sudden growing amount of developers corresponds to a similar growing output produced: we termed this as the `bazaar' phase, and we also argued that this phase was not achieved for the other system. A further analysis revealed that the transition between `cathedral' and `bazaar' was a phase by itself in Wine, achieved by creating a growing amount of new modules, which attracted new developers.

1 Introduction

Prominent free software (or open source software, OSS) projects such as Linux [32], Apache [27] and FreeBSD [18] have been extremely successful. Anecdotal evidence has been used in the past to characterize successful OSS projects: users/developers acting as "more eyeballs" in the correction of bugs, developers implementing new features independently, skillful project managers dealing with a mostly flat organization, and the resulting coordination costs [28].

Previous studies have provided empirical evidence on the process of successful OSS projects: the definition of various types of developers has been discussed for the Mozilla and the Apache projects, justifying different levels of effort [27], and claiming that the first type (core developers) contribute to the success of a system.

Please use the following format when citing this chapter:

Capiluppi, A. and Michlmayr, M., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 31–44.

Also, social network analyses have shown communication and coordination costs in successful OSS projects [21].

In all these cases, successful projects are studied and characterized, but an analysis in their earlier inception is not given. Therefore, empirical studies on whether the project always benefited of a large number of developers, or built instead its bazaar through several years, are still missing. In order to tackle this missing link, this paper explores the evolution and development processes of two OSS systems, the Wine (a free implementation of Windows on Unix) project and the Arla file system. The first system has been widely adopted and developed by many developers. Arla, on the other hand, is still in a `cathedral' phase when compared Wine: fewer developers are currently collaborating towards its development.

The aim of this paper is to empirically detect and characterize the phases achieved by these two systems, to illustrate whether one phase consequently follow the other, and to establish one of these phases as a `success' for an OSS project. If this is the case, sharing the empirical guidelines on how to achieve this transition could help developers to work on the benefits of the bazaar phase.

Structure of the paper: in Section 2, a theoretical background will be given, as well as two research questions, based on OSS communities. Also, a description of the approach used to acquire and analyses the data employed will be presented. The data will be used to test the presented questions. Section 3 will describe the phases observed in the two systems from the point of view of the activities of developers. This section will also give a detailed description of the activities that underpin the success of a OSS system, as observed in the proposed case studies. Section 4 will deal with related work in this (and other) areas, identifying the main contributions of this paper, and will discuss a number of questions raised in this paper that need further empirical exploration. Finally, Section 5 will give conclusions on the overall process and lifecycle of OSS systems, as well as possible future research directions.

2 Background Research

One of the authors, in a previous work [29], presented a theoretical framework for the activities and phases of the lifecycle of OSS projects. The objective was to provide a more systematic approach for the development of OSS projects, to increase the likelihood of success in new projects. In this paper, the objective is to empirically evaluate the theory contained in that work through two case studies, and to report on best practices of actually successful OSS projects. Since previous studies have shown that many OSS projects must be considered failures [3, 7], it is argued that the latter ones lack some of the characteristics as described in [29], notably the transition between the closed (or `cathedral') and the open (or `bazaar') styles. In his popular essay “The Cathedral and the Bazaar", Eric S. Raymond [28] investigates development structures in OSS projects in light of the success of Linux. The terminology of the `cathedral' and the `bazaar' introduces both a closed approach, found in most commercial entities, where decisions on large software projects are taken by a central management; and an open one, where an entire community is in charge of the whole system.

Instead of viewing these approaches as diametrically opposed, as originally proposed by Raymond, this paper considers these as complimentary events within the same OSS software project. Figure 1 illustrates three basic phases, which this research argues a successful OSS project undergoes. The initial phase of a OSS project does not operate in the context of a community of volunteers. All the characteristics of cathedral style development (like requirements gath- ering, design, implementation and testing) are present, and they are carried out in the typical style of building a cathedral, that is, the work is done by an individual or a small team working in isolation from the community [5]. This development process shows tight control and planning from the central project author, and is referred to as `closed prototyping' by Johnson [17].

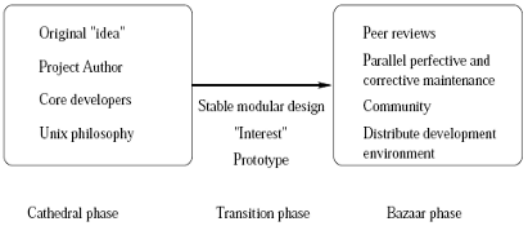

Fig. 1. *OSS development lifecycle*

In order to become a high quality and useful product, [29] argued that an OSS project has to make a transition from the cathedral phase to the bazaar phase (as depicted by the arrow in Figure 1). In this phase, users and developers continuously join the project writing code, submitting patches and correcting bugs. This transition is associated with many complications: it is argued that the majority of free software projects never leave the cathedral phase and therefore do not access the vast resources of manpower and skills the free software community offers [7].

2.1 Research questions

In this paper, historical data on code modifications and additions of large (subsystems) or small scale (modules) sections of a software system are analyzed in order to track how the studied systems evolved over time. Two research questions are presented here: the historical data will be then tested against them, and the results will be evaluated in the next section. The first is based on out- put obtained from input provided, the second on what new developers tend to work on when joining a OSS project. The research questions can be formulated as follows (metrics used to asses each question are also provided):

i) research question 1: the `bazaar' phase involves a growing amount of developers, who join in a self-sustaining cycle. The output obtained in a bazaar phase follows a similar growing trend. OSS projects, while still in the `cathedral' phase, do not benefit from a growing trend in input provided and output achieved.

ii) research question 2: new developers, when joining a software project, tend to work on newest modules first, either by creating the modules themselves, or by contributing to a new module. This can be rationalized saying that new developers might not need insights on all the preexisting functionalities of a system, thus preferring to develop something new. This research question will be used to gather further insights on how Wine could achieve a bazaar phase.

2.2 Empirical approach

The empirical approach involves the extraction of all changes embedded in sources of information of both input (effort provided by developers) and output (that is, additions or changes of subsystems and modules). In the following analysis, the ChangeLog file, recording the whole change history of a project, has been used rather than an analysis of the projects' CVS repositories. From previous research it is known [10, 22] that different development practices have an influence on the best data source, and the ChangeLog file offers more reliable information in the selected case projects [6, 12, 30].

The steps to produce the final data can be summarized in: parse of raw data, and extraction of metrics. As part of the first step, automated Perl scripts are written to parse the raw data contained in the ChangeLog and to extract pre-defined data fields. The data fields which will be considered in this study are: name of the system, name of the module, name of the subsystem containing that module, date of creation or change and unique ID (name and email) of the developer responsible for the change.

2.2.1 Raw data extraction

The analyzed ChangeLog files follow very regular an- notating patterns, thereby allowing a straightforward analysis of the history of changes in a project in a semi-automated way. The following steps have been performed during the extraction of the raw data:

1 – Identification of dates: it was observed in the studied cases that each touch was delimited by a date, using the following or a similar pattern: for example, YYYY-MM-DD, as in “2000-12-31”. Each touch can be associated with one or more than one developers; also, each touch can be associated with one or more than one modules. For each touch there is one and only one date.

2 – Affected modules and subsystems: each touch affects at least one *file*, and is recorded with a plain-text description. In some cases the same touch affects many files: these modifications are referred to the same date. Subsystems are extracted as the *folder* containing the affected file.

3 – Details of developers: All touches concern at least one developer, displayed in various forms inside of the description of the touch. If more than one developers are responsible for a touch, they are recorded together within the touch.

4 – Derivation of metrics: Counts were derived of both effort provided by developers and work produced creating new modules and amending existing ones.

2.2.2 Metrics choice and description

The analysis of the two OSS systems involved three types of metrics, used differently to discuss the research questions. A list is proposed in the following:

i - Input metrics: the effort of developers was evaluated by counting the number of unique (or distinct, in a SQL-like terminology) developers during a specific interval of time. The chosen granularity of time was based on months: different approaches may be used, as on a weekly or on a daily basis, but it is believed that the month represented a larger grained unit of time to gather the number of active developers. This metrics was used to evaluate the first research

question. For instance, in February 2006 it was found that the Wine system had 73 distinct developers who wrote code for this system in that month.

ii - Output metrics: the work produced was evaluated by counting the touches to modules or subsystems during the same interval of time. Smaller- grained metrics, like lines of code, were not considered in this study: evaluating how many lines of code are produced by OSS developers could be subject to strong limitations[1]. In the following section this metric will be used also as an indicator of parallel development work performed in successful projects. This metrics was also used to evaluate the first research question. As above, in February 2006 it was detected that the Wine system had 820 distinct modules which were touched in that month.

iii - New Input and Output metrics: the newly-added effort was evaluated counting the new developers joining the project. The work produced by these new developers was also isolated: the objective is to determine how much of this work has been focused on existing parts of the system, and how much goes to new parts. This metrics served to evaluate the second research question, i.e. to explore if new developers tend to work either on old or new parts of the system. As above, in February 2006 it was detected that the Wine system had 73 new developers (i.e. not detected in any of the previous touches). It was also empirically detected that these new developers worked in part on old modules, and in part on new modules, i.e. added in the same month. It was observed that 75% of their work concerned newer modules, and 25% on existing modules.

2.3 Case studies

The choice of the case studies was based on the recognized, objective success of one of the systems (Wine), while the second analyzed system (Arla) seems to have suffered from an inability of recruiting new developers, and achieved a much smaller overall size. Both of them have been used in the past for other empirical case studies, and their development style and growth pattern have been extensively studied.

The authors recognize that the two systems have two very different application domains: Wine is a tool to run Windows applications on Linux and other operating systems, while Arla is a networked file system. The main objective of the present study was not to evaluate the exogenous reasons behind successfully recruiting projects (like the presence of recognized “gurus” in a project, the good reputation of the existing community, etc [9]). On the contrary, this study focuses on evaluating the presence of three different stages in successful projects. The research presented here proposes a theoretical framework for OSS projects, independently from their domain, and empirically evaluates the mechanisms of forming a community around OSS projects.

Attribute/System	Arla	Wine
Earliest found entry	10/1997	07/1993
Latest studied entry	03/2006	03/2006
Change or creation points	7,000	88,000
Global, distinct developers	83	880

Table 1: *summary of information in the two systems.*

The choice of the information sources was restricted to two classes of items, the CVS commits and the ChangeLog

[1] Lines of code produced are biased by the skills of the developer, the programming language and, in general, the context of the modifications.

records. The CVS repository of Arla was found to be *incomplete*, since it does not contain the complete evolution history of the project. This is probably due to the fact that the CVS has been adopted at some point after the project's first inception. It was also observed that the CVS server of Wine is *inaccurate*: a query for active developers shows only 2 committers, against a much larger number of developers found in the ChangeLog records. That probably means a restriction in the write access to the Wine CVS. ChangeLogs were therefore preferred over CVS logs.

As a means to characterize the two systems, Table 1 displays some basic information about their ChangeLog files, the time span, and the amount of distinct developers which were found actively contributing to the project.

3 Results and discussion of the phases

In the following section, the two research questions are discussed, and the three phases (cathedral and bazaar, separated by a transition phase) as presented in [29] are evaluated, based on the empirical data from the case studies. Apart from this evaluation, it is also planned to identify some practical actions that OSS developers should consider in order to enhance the evolutionary success of their projects, and to ease the transition between the cathedral and the bazaar phases.

3.1 The cathedral phase

One of the main differences between closed, traditional software and OSS development is the ownership of the code. In the first environment, the development is typically driven by a team of individuals, while users do not contribute to, nor access the source code. In the latter, potentially everyone has the right to access and modify the source code underlying an application. It is argued that a typical OSS system will follow a cathedral approach in its first evolution history.

Arla system – input: Figure 2 (left) shows the distribution of distinct developers per month in the Arla system. Even though a sum of over 80 developers have contributed code, patches and fixes to the project (see Table 1), the number of distinct developers working on the development each month is much lower: on average only about five distinct developers work on the code base each month. As stated above, the first research question is not confirmed by the empirical findings: in the Arla project, the evolution of distinct, active developers in a month shows a regular, constant pattern.

Arla system – output: Figure 2 (right), on the contrary, shows the amount of distinct modules and subsystems that Arla developers have worked on since its inception: the distribution is fairly regular, and that could mean that new developers, when joining the project, are not expanding it into new areas, but that they rather work on existing functionality, together with the core developers. This will be tested in the section dedicated to the transition phase. These output findings, i.e. a constant, not growing pattern in output produced, confirm that the first research question does not apply for the Arla system.

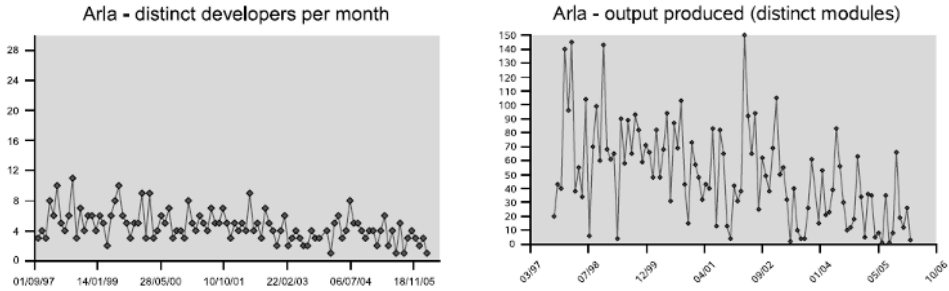

Figure 2: *Development input (left) and output produced (right) in Arla*

While these findings do not necessarily imply that Arla is a failure compared to Wine (as in the overall amount of developers from Table 1), it raises some interesting questions: for instance, it should be studied why only a small, but constant, number of developers is contributing code. As a possible explanation of its (reduced) success in recruiting new developers, one could argue that the system could be perceived as mature already [8], and that little further work was needed. Similar problems have been observed in the past for the OpenOffice.org and Mozilla systems: they represent two extremely complex applications and required a huge investment in the study, before developers could actually contribute directly.

In the next sections, practical guidelines will be evaluated on how an OSS system could tackle the issues faced by the Arla project, and in order to benefit of the efforts of a larger pool of developers.

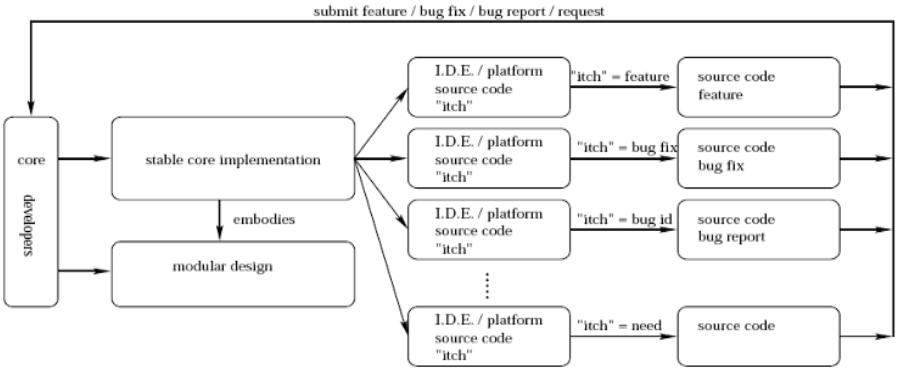

Figure 3: *Detailed bazaar phase*

3.2 Bazaar phase

The aim of many OSS projects is to reach a stage where a community of users can actively contribute to its further development. Some of the key characteristics of the bazaar phase are visualized in Figure 3, and can be summarized as follows:

- Contributions: the bazaar style makes source code publicly available and contributions are actively encouraged, particularly from people using the software. Contributions can come in many different forms and at any time. Non-technical users can suggest new requirements, write user documentation and tutorials, or

point out usability problems (represented as low-level "itches" in Figure 3); technical users can implement features, fix defects and even extend the design of the software (the high-level "itches" of Figure 3).

- Software quality: increased levels of quality comes from thorough, parallel inspections of the software, carried out by a large community of users and developers. These benefits are consistent with software engineering principles: the `debugging process' of a OSS project is synonymous with the maintenance phase of a traditional software lifecycle.
- Community: a network of users and developers review and modify the code associated with a software system. The old adage “many hands make light work” is appropriate in describing the reasons for the success of some OSS projects [27].

Wine system – input: From the empirical standpoint, Figure 4 (left) shows the distribution of distinct developers per month in the Wine system. In total, over 800 developers have contributed code, patches and fixes (Table 1). Even though this project has a longer time span, which could have facilitated the growth of a developers basis, a clear distinction between a first phase (cathedral) and a later phase (bazaar) can be identified in the number of developers. Around July 1998, the Wine system has undergone a massive evolution in the number of distinct developers involved in the project. The sustainability of this new bazaar phase is demonstrated by the further, continual increasing number of new distinct developers in the Wine system. The first research question finds an empirical evidence analyzing the Wine system, a growing pattern of active developers signals the presence of the bazaar

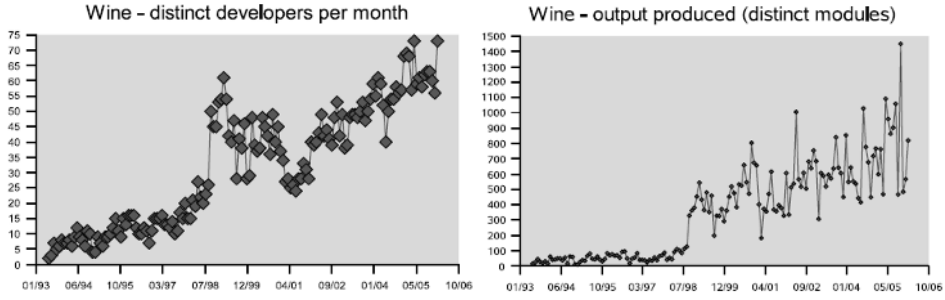

Figure 3: *Development input (left) and output produced (right) in Wine*

phase. The sustainability of the input process is visible in the ever-changing amount of distinct developers which participate in the evolution of the system.

Wine system – output: The bazaar phase is characterized by an open process in which input from volunteers defines the direction of the project, including the requirements. The initial implementation is mainly based on the requirements of the project author. In the bazaar phase, projects benefit from the involvement of a diverse range of users (with different requirements) who work together to increase the functionality and appeal of the software.

This parallel development behavior is achieved successfully in the Wine project. During the investigation of this system, the evolving scope of the project became apparent through the amount of distinct modules which developers work on each month. Figure 3 (right) shows the amount of distinct modules and subsystems that developers have worked on since its inception: the distribution is growing abruptly

around the same time when an increase of distinct authors is observed Figure 3, right). This means that the project, with new developers joining constantly, is actively expanding it into new areas. The growing pattern of active developers sustains a growing pattern of output produced: as above, the first research question helps signaling the presence of the bazaar phase when such a growing pattern occurs.

3.3 Transition phase – defining new avenues of development

The theoretical framework represented in Figure 1 assigns a fundamental role to the transition phase, since it requires a drastic restructuring of the project, especially in the way the project is managed. One important aspect is commencing the transition at the right time. This is a crucial step and a hurdle many projects fail to overcome [11]. Since volunteers have to be attracted during the transition, the prototype needs to be functional but still in need of improvement [17, 28, 2].

If the prototype does not have sufficient functionality or stability, potential volunteers may not get involved. On the other hand, if the prototype is too advanced, new volunteers have little incentive to join the project because the code base is complex or the features they require have already been implemented. In both cases, adding future directions to the system could provide potential new developers further avenues for the development.

Based on the second research question, new developers, when joining a software project, tend to work on new modules, rather than old ones. As a consequence, the core developers should expand the original system into new directions and provide new code to work on: this would foster the recruitment of new developers and facilitate the transition phase.

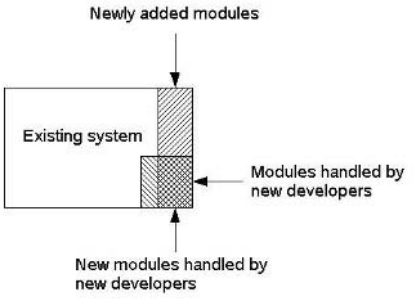

Figure 4: *Design of research question 2.*

To evaluate this question, an experiment was designed: at first, the newly added modules were extracted in every month. In parallel, the amount of new developers was also extracted. Finally, what new developers worked on was defined as the percentage of new modules they handled: Figure 4 graphically summaries this process.

The empirical results were extracted for the two systems Arla and Wine and are displayed in a box-plot, spanning all the releases for the two systems. Figure 8 is a description, on a percentile basis, of the modules as handled by newest developers.

Transition achieved – Wine: this system reveals that new developers, when joining the project, tend to work more easily on new modules than on older ones. In fact, more than 50% (on average) of what they work on is newly added in the same month, either by themselves or the core developers (right boxplot of Figure 5). Also, the average value of the boxplot was found to be larger when considering only the `bazaar' phase of Wine.

This first result is confirmed by plotting the amount of new modules created by the developers (Figure 5, right). A growing pattern is detected, similar to the one observed in the global evolution of the system (Figure 3): new developers join in, working on newest parts of the code, while core developers sustain the community of the project by continuously adding new modules.

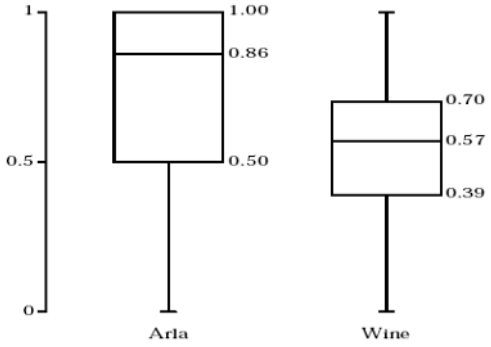

Figure 5: *Description of effort for new developers*

Transition not achieved – Arla: this second system provides a much more interesting box-plot: the tendency of new developers is clearly towards working on something new, rather than on old modules (left boxplot of Figure 5). The main difference with the Wine project is that, for most of the periods, there are no new developers joining in the Arla development. Based on the assumptions of the second research question, new developers still prefer to start something new, or work on newly added code: still, this project could not ease the transition phase by not recruiting new developers. Therefore, it is possible to conclude that the original developers in Arla failed in providing new directions for the system, by creating new modules or subsystems. This conclusion is backed by the amount of new modules created by the developers (Figure 6, left): a decreasing pattern is detected, which confirms that new developers (and the community around the project), albeit willing to work on the system, were not adequately stimulated by the core developers.

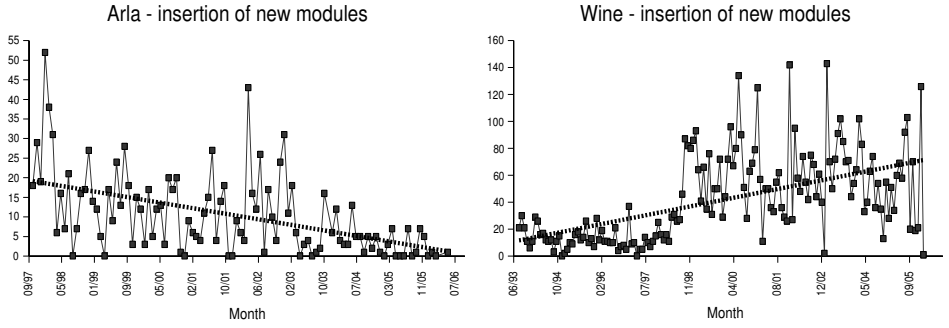

Figure 6: *Creation of new modules in the Arla and Wine systems*

In summary, considering the second research question stated above, we found similar evidences for both the systems: when joining the development of an OSS system, new developers tend to work on (i.e., add or modify) new modules rather than old ones. As a proposed corollary to these results, the transition to a bazaar phase should be actively sought by the core developers: potential new developers should be actively fostered adding new ideas or directions to the project.

4 Related work

In this section the present work is related to various fields, specifically empirical studies on software systems and effort evaluation. Since this work is in a larger research context, related to the study of the evolution of OSS systems, empirical studies of OSS are also relevant to this research.

The earliest studies of the evolution of software systems were achieved through the proprietary operating system OS/360 [4]. The initial studied observed some 20 releases of OS/360, and the results that emerged from this investigation, and subsequent studies of other proprietary commercial software [20], included the SPE program classification and a set of laws of software evolution.

The present research has been conducted similarly, but evaluating both the input (as effort) provided, and the output (as changes made to the code base) achieved. The research questions which this paper is based upon derives from [29], and is based on the presence of two distinct phases in the software lifecycle of OSS systems, namely the cathedral phase and the bazaar phase [28]. This in contrast with Raymond's suggestion that the bazaar is the typical style of open source projects [15, 28]: an empirical evaluation was achieved by studying the lifecycle of two large free software projects, of which only one has made the transition to the bazaar phase and attracted a large community of developers. It is believed by the authors that too much emphasis has been put on highly popular projects in the past which are not necessarily representative of the OSS community as a whole [13, 15, 16, 26]. Few projects make a transition to the bazaar, attracting a large and active developer community along the way.

Having a large bazaar surrounding a project has several advantages, such as the ability to incorporate feedback from a diverse base of users and developers. Nevertheless, this is not to say that projects which are not in the bazaar phase are necessarily failures – they neither have to be unsuccessful nor of low quality.

Interestingly enough, in contrast to Raymond's model, there are a number of applications, such as GNU *coreutils* and *tar*, which form a core part of every Linux system and which clearly follow the cathedral. Similarly, there are many projects entirely developed by a single, extremely competent developer which show high levels of quality. Due to the lack of better theories and empirical research, quality in OSS projects is explained through the bazaar with its peer review [1, 26, 28]. However, not every project with high quality actually exhibits a large bazaar and significant peer review.

A project in the cathedral phase can be highly successful and of high quality [31]. However, there are some restrictions a project in the cathedral phase faces as well as a number of potential problems which are less severe if the project had a large developer community. For example, while it is possible for a single developer to write an application with a limited scope (such as a boot loader), only a full community can complete a project with a larger scope (such as a full desktop environment). Furthermore, a project written by one developer may be of high quality but it also faces a high risk of failure due to the reliance on one person who is a volunteer [23, 25]. Having a large community around a project makes the project more sustainable.

This discussion shows the lack of research in a number of areas related to OSS projects. While a uniformed model for all OSS projects has been assumed in the past, it is increasingly becoming clear that there is a great variety in terms of development processes [9, 19, 14]. Better theories about success and quality in OSS projects are needed [24], as are further comparisons between projects with different levels of success and quality. Finally, it should not be assumed that the bazaar is necessarily the optimal phase for every project, or that it is not associated with any problems. There is a general assumption that it is beneficial for a OSS project to be open, but too much openness can also be harmful when it leads to incompetent developers or people who demotivate important contributors getting involved [9].

5 Conclusions and future work

Successful OSS projects have been studied and characterized in the past, but an empirical demonstration on how they achieved their status has not been proven yet. In order to tackle this missing link, this paper has presented an empirical exploration of two OSS projects, Arla and Wine, to illustrate different phases in their lifecycle, their development processes and the communities which formed around them. Their ChangeLog records were analyzed and all the changes and additions, performed by the developers over the years, were recorded.

The assumption underpinning this paper is that the `cathedral' and `bazaar' phases, as initially proposed and depicted by Raymond in [28], are not mutually exclusive: OSS projects start out in the cathedral phase, and potentially move to a bazaar later. The cathedral phase is characterized by closed development performed by a small group or developer, with much in common with traditional software development. The bazaar phase exploits a larger number of volunteers who contribute to the development of the software through defect reports, additional requirements, bug fixes and features. The transition between the two phases was argued to be by itself a phase too, which has to be accommodated by specific, active actions of the core developers or project author. It was also argued that this transition is a necessary factor for truly successful and popular projects.

A first research question has proposed the study of the difference between the cathedral and the bazaar phases: the first system (Arla) has remained, through its lifecycle, an effort of a limited number of developers, or in a cathedral phase. It was also argued that this should not be interpreted as a sign of the overall failure of an OSS project, but as a potentially missed opportunity to establish a thriving community around a project. On the contrary, the second system (Wine) only shows an initial phase that is similar to what observed in the Arla system: a second, longer phase (bazaar) has a growing amount of active developers and a continuous expansion of the system.

Through a second research question, the focus was moved to the preferences of new developers joining an OSS project: results on both the systems show that new developers prefer to work on newly added modules, rather than older ones. In the Wine system, existing developers eased the transition phase by adding many new modules which new developers could work on. On the other hand, new developers in

Arla, although eager to work on new code, were not yet given enough new directions of the project, and an overall poor ability in recruiting new developers was resulting.

The future work has been identified in a replication of the study with other OSS projects, especially those belonging to the same application domain: the results as obtained in this study have analyzed the creation of a community from a neutral point of view, that is, without considering exogenous drivers. Our next step is to introduce these drivers into the research, and analyze large projects which currently compete with each other for the scarce resource of developers.

References

[1] A. Aoki, K. Hayashi, K. Kishida, K. Nakakoji, Y. Nishinaka, B. Reeves, A. Takashima, and Y. Yamamoto. A case study of the evolution of jun: an object-oriented open-source 3d multimedia library. In Proceedings of the 23rd International Conference on Software Engineering, pages 524-533, Toronto, Canada, 2001.

[2] B. Arief, C. Gacek, and T. Lawrie. Software architectures and open source software – where can research leverage the most? In Proceedings of the 1st Workshop on Open Source Software Engineering, Toronto, Canada, 2001.

[3] R. Austen and G. Stephen. Evaluating the quality and quantity of data on open source software projects. In Proceedings of 1st International Conference on Open Source Systems, Genova, Italy, June 2005.

[4] L. A. Belady and M. M. Lehman. A model of large program development. IBM Systems Journal, 15(3):225-252, 1976.

[5] M. Bergquist and J. Ljungberg. The power of gifts: Organising social relationships in open source communities. Information Systems Journal, 11(4):305-320, 2001.

[6] A. Capiluppi. Models for the evolution of OS projects. In Proceedings of International Conference on Software Maintenance, pages65-74, Amsterdam, Netherlands, 2003.

[7] A. Capiluppi, P. Lago, and M. Morisio. Evidences in the evolution of OS projects through changelog analyses. In Proceedings of the 3rd Workshop on Open Source Software Engineering, Portland, OR, USA, 2003.

[8] A. Capiluppi, M. Morisio, and J. F. Ramil. Structural evolution of an open source system: A case study. In Proceedings of the 12th International Workshop on Program Comprehension (IWPC), pages 172-182, Bari, Italy, 2004.

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

[10] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In Proceedings of International Conference on Software Maintenance, pages 23-32, Amsterdam, Netherlands, 2003.

[11] K. F. Fogel. Open Source Development with CVS. The Coriolis Group, Scottsdale, Arizona, 1st edition, 1999.

[12] D. M. German. An empirical study of fine-grained software modifications. pages 316-325, Chicago, IL, USA, 2004.

[13] D. M. German. Using software trails to reconstruct the evolution of software. Journal of Software Maintenance and Evolution:Research and Practice, 16(6):367-384, 2004.

[14] D. M. German and A. Mockus. Automating the measurement of open source projects. In Proceedings of the 3rd Workshop on Open Source Software Engineering, Portland, OR, USA, 2003.

[15] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In Proceedings of the International Conference on Software Maintenance, pages 131-142, San Jose, CA, USA, 2000.
[16] J. Howison and K. Crowston. The perils and pitfalls of mining SourceForge. In Proceedings of the International Workshop on Mining Software Repositories (MSR 2004), pages 7-11, Edinburgh, UK, 2004.
[17] K. Johnson. A descriptive process model for open-source software development. Master's thesis, Department of Computer Science,University of Calgary, 2001. http://sern.ucalgary.ca/students/theses/KimJohnson/thesis.htm
[18]N. Jørgensen. Putting it all in the trunk: Incremental software engineering in the FreeBSD open source project. Information Systems Journal, 11(4):321-336, 2001.
[19] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: GNOME. Information Systems Journal, 12(1):27-42, 2002.
[20] M. M. Lehman and L. A. Belady, editors. Program evolution: Processes of software change. Academic Press Professional, Inc., San Diego, CA, USA, 1985.
[21] L. Lopez, J. G. Barahona, I. Herraiz, and G. Robles. Applying social network analysis techniques to community-driven libre software projects. International Journal of Information Technology and Web Engineering, 11(4):321-336, 2006.
[22] T. Mens, J. F. Ramil, and M. W. Godfrey. Analyzing the evolution of large-scale software: Guest editorial. Journal of Software Maintenance and Evolution, 16(6):363-365, 2004.
[23] M. Michlmayr. Managing volunteer activity in free software projects. In Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track, pages 93-102, Boston, USA, 2004.
[24] M. Michlmayr. Software process maturity and the success of free software projects. In K. Zielinski and T. Szmuc, editors, Software Engineering: Evolution and Emerging Technologies, pages 3-14, Krakow, Poland, 2005. IOS Press.
[25] M. Michlmayr and B. M. Hill. Quality and the reliance on individuals in free software projects. In Proceedings of the 3rd Workshop on Open Source Software Engineering, pages 105-109, Portland, OR, USA, 2003.
[26] M. Michlmayr, F. Hunt, and D. Probert. Quality practices and problems in free software projects. In M. Scotto and G. Succi, editors,Proceedings of the First International Conference on Open Source Systems, pages 24-28, Genova, Italy, 2005.
[27] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3):309-346, 2002.
[28] E. S. Raymond. The Cathedral and the Bazaar. O'Reilly & Associates, Sebastopol, CA, USA, 1999.
[29] A. Senyard and M. Michlmayr. How to have a successful free software project. In Proceedings of the 11th Asia-Pacific Software Engineering Conference, pages 84-91, Busan, Korea, 2004. IEEE Computer Society.
[30] N. Smith, A. Capiluppi, and J. F. Ramil. Agent-based simulation of open source evolution. Software Process: Improvement and Practice, 11(4):423-434, 2006.
[31] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris. Code quality analysis in open-source software development. Information Systems Journal, 12(1):43-60, 2002.
[32] L. Torvalds. The Linux edge. In C. DiBona, S. Ockman, and M. Stone, editors, Open Sources: Voices from the Open Source Revolution, pages 101-111. O'Reilly & Associates, Sebastapol, CA, USA, 1999.