The Journal of Systems and Software xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Identifying exogenous drivers and evolutionary stages in FLOSS projects

Karl Beecher *, Andrea Capiluppi, Cornelia Boldyreff

Centre of Research on Open Source Software – CROSS, Department of Computing and Informatics, University of Lincoln, UK

ARTICLE INFO

Article history:
Received 11 January 2008
Received in revised form 22 October 2008
Accepted 24 October 2008
Available online xxxx

Keywords:
Open source software
Software evolution
Software repositories

ABSTRACT

The success of a Free/Libre/Open Source Software (FLOSS) project has been evaluated in the past through the number of commits made to its configuration management system, number of developers and number of users. Most studies, based on a popular FLOSS repository (SourceForge), have concluded that the vast majority of projects are failures.

This study's empirical results confirm and expand conclusions from an earlier and more limited work. Not only do projects from different repositories display different process and product characteristics, but a more general pattern can be observed. Projects may be considered as early inceptors in highly visible repositories, or as established projects within desktop-wide projects, or finally as structured parts of FLOSS distributions. These three possibilities are formalized into a framework of transitions between repositories.

The framework developed here provides a wider context in which results from FLOSS repository mining can be more effectively presented. Researchers can draw different conclusions based on the overall characteristics studied about an Open Source software project's potential for success, depending on the repository that they mine. These results also provide guidance to OSS developers when choosing where to host their project and how to distribute it to maximize its evolutionary success.

1. Introduction

The environment in which software is deployed is known to have a direct effect on its subsequent evolution. Lehman's first law of software evolution anticipates that useful real-world software systems (i.e., E-type) must undergo continuing change in response to various requirements, in other words they must evolve (Lehman et al., 1997). The LAMP stack (Linux, Apache, MySQL, Perl/PHP/Python), the Mozilla Foundation and the BSDs family are well-known examples of open source E-type software systems, and as such are no exception to this rule.

The successful evolution of such open source software projects has been made possible (among other factors) also by their attraction of large communities of both users and developers, two categories that notably are not mutually exclusive in open source software. Users initiate the need for change and the developers implement it (Mockus et al., 2002). The extent to which an open source project is successful has often been evaluated empirically by measuring *endogenous* characteristics, such as the amount of developer activity, the number of developers, or the size of the project (Crowston et al., 2006; Godfrey and Tu, 2000; Robles et al., 2003). As an example, a thorough study of Sourceforge.net (a popular repository of more than 200,000 open source projects) concluded that the majority of projects housed there should be considered "tragedies" by virtue of their failure to initiate a steady series of releases (English and Schweik, 2007).

The general success of open source software projects has accompanied the wider establishment of organized repositories aiming to facilitate their development and management. In a previous work Beecher (XXXX) the authors examined a collection of open source projects, and studied instead the *exogenous* drivers acting upon them and established to what extent the repositories in which a project is located affects its evolutionary characteristics. By comparing equally sized random samples from two open source repositories and also tracking the evolution of projects that moved between them, this earlier study concluded that a repository typically has statistically significant effects upon characteristics such as the number of contributing developers as well as the period and amount of development activity.

This work extends and expands the previous study in two ways. First, it encompasses a greater number of repositories; instead of the original two, this paper formulates hypotheses and gathers empirical evidence from data extracted from six different FLOSS repositories, and provides further empirical evidence for the earlier assertions. By making multiple comparisons between them, a structured body of knowledge has been constructed regarding the key practical differences between the individual FLOSS repositories being studied. Secondly, the paper formulates a framework of evolution for FLOSS projects, based on the repository to which

* Corresponding author. Tel.: +44 01522 886858.
E-mail address: kbeecher@lincoln.ac.uk (K. Beecher).

0164-1212/$ - see front matter © 2008 Published by Elsevier Inc.
doi:10.1016/j.jss.2008.10.026

they belong, comprising a typical path of evolution between repositories, which exploits better process and product characteristics of projects in particular repositories.

The paper is articulated as follows: Section 2 explores previous work and shows how the findings of this paper extend and expand upon past literature on the subject. Section 3 tailors the Goal-Question-Metric methodology to this specific case study, and introduces the empirical hypotheses on which this study is based, the null hypotheses and their alternative counterparts, and discusses how they have been operationalized. It also describes which repositories have been selected, how the data has been extracted from them, and which attributes have been used to characterize their process and product aspects. Sections 4 and 5 illustrate the results gathered, and verifies whether the hypotheses have to be rejected. Section 6 provides the discussion for the empirical findings and introduces the framework for the evolution of FLOSS projects along repositories; Section 7 explores the threats to the external and internal validity of this empirical study, while Section 8 provides the key findings of this research.

2. Previous work

There are two main types studies found in the FLOSS literature, one termed *external* and the other *internal* to the FLOSS phenomenon (Beecher, XXXX). Based on the availability of FLOSS data, the former has traditionally used FLOSS artefacts in order to propose models (Hindle and German, 2005), test existing or new frameworks (Canfora et al., 2007; Livieri et al., 2007), or build theories (Antoniol et al., 2001) to provide advances in software engineering. The latter includes several other studies that have analyzed the FLOSS phenomenon *per se* (Capiluppi, 2003; German, 2004; Herraiz et al., 2008; Stamelos et al., 2002) with their results aimed at both building a theory of FLOSS, and characterizing the results and their validity specifically as inherent to this type of software and style of development. In this section a selection of works from the latter category is reviewed.

The success and failure of FLOSS projects has been extensively studied in the past; some specific repositories have been analyzed, and metrics have been computed from data extracted from them. Examples include the use of the *vitality* and *popularity* indexes, computed by the SourceForge maintainers, which have been used to predict other factors on the same repository (Stewart and Ammeter, 2002), or to compare the status of the projects between two different observations (Feller et al., 2002). Also data has been collected from SourceForge about community size, bug-fixing time and the popularity of projects, and has been used to review some popular measures for success in information systems related to the FLOSS case (Crowston et al., 2003). Popularity of FLOSS projects has also been assessed using web-search engines (Weiss, 2005). Other studies have observed projects from SourceForge, and from their release numbers, their activity or success within a sample (Crowston et al., 2006) has been inferred; while other research has sampled the whole SourceForge data space, and has concluded that the vast majority of FLOSS projects should be considered as failures (Rainer and Gale, 2005). Finally, other researchers have created 5 categories for the overall SourceForge site, based on dynamic growth attributes, and using the terms "success" and "tragedy" within the FLOSS development. Again, it has been shown that some 50% of the FLOSS projects should be considered as tragedies (English and Schweik, 2007).

This study is intended as an extension of a previous study to amplify a promising set of findings when comparing the characteristics of two different FLOSS repositories, Debian and SourceForge (Beecher, XXXX). It was found that projects in the Debian repository consistently achieved larger sizes and more developer activity than their SourceForge counterparts; in addition, it was found that, within the Debian sample, these increased measures could be observed typically *after* the projects were included in the Debian repository. The present study expands the previous data base and results by considering four other repositories (KDE, GNOME, RubyForge and Savannah), extracts similar samples from each of the resulting six repositories (50 projects each from the repository's "stable" pool), and studies four product and process characteristics of the projects in the samples. Based on these experiments, this study also provides a more general framework for the evolution of FLOSS projects.

There are several tools and data sources which are used to analyze FLOSS projects. FLOSSmole[1] is a single point of access to data gathered from a number of FLOSS repositories (*e.g.*, SourceForge, Freshmeat, Rubyforge). While FLOSSmole provides a simple querying tool, its main function is to act as a source of data for others to analyze. CVSAnaly[2] is a tool which is used to measure any analyses from large FLOSS projects (Robles et al., 2004). It is used in this paper to determine such measures as the number of commits and developers associated with a particular project.

3. Empirical study definition and planning

The Goal-Question-Metric method (GQM) Basili et al. (1994) evaluates whether a goal has been reached by associating that goal with questions that explain it from an operational point of view and provide the basis for applying metrics to answer these questions. The aim of the method is to determine the information and metrics needed to be able to draw conclusions on the achievement of the goal.

In this study, the GQM method is applied firstly to identify the overall goals of this research; then to formulate a number of questions related to FLOSS repositories and their exogenous (or external) effects on the underlying process and product characteristics of the FLOSS projects they comprise; and finally to identify and collect adequate product and process metrics to determine whether the identified goal has been achieved. In the following, the goal, questions and metrics used are introduced and commented upon.

(1) **Goal**: The long-term objective of this research is to evaluate characteristics and associated metrics to identify successful FLOSS projects, and to investigate whether different repositories can be held *externally* responsible for this success. In this particular work, the aim is to establish whether (and if so to what extent) inclusion of a project within a repository causes a project to increase its "success", and hence establish a cause-effect relationship. As a corollary goal, this work aims to provide guidelines to FLOSS developers about practical actions to take in order to foster the successful evolution of their applications.

(2) **Question:** The purpose of this study is to establish differences between samples of FLOSS projects extracted from various repositories. Two research questions have been formulated for evaluation; one is thoroughly comparative, and one is related to a formulated framework of reference for FLOSS repositories. The first deals with a direct comparison; the evolutionary characteristics of the projects have been compared with projects from other repositories. The second clusters the repositories in distinct groups and formulates hypotheses based on the effectiveness of each group on the observed characteristics. As a summary, the two main questions underlying this study can be formulated as follows:

[1] http://ossmole.sourceforge.net/.
[2] http://cvsanaly.tigris.org/.

(a) Are the various repositories significantly different from each other, in terms of both process and product characteristics?
(b) Based on the same characteristics, do repositories cluster in different groups? Are these groups significantly different from each other, when comparing these characteristics?

(3) **Metrics**: This study uses two sources of data to answer the above questions: the repositories themselves, which have been mined to select random samples of projects; and each project's own repository (either their CVS or SVN) which has been studied to analyze activity and the outputs recorded by the configuration management systems. In each case the metrics have been taken from the project's source code repository log (CVS or SVN); exceptions and filtering of noise in data has also been performed, as detailed in the following sections. Two types of metrics have been collected throughout this study, and these are detailed below:

Process metrics	number of distinct developers and developers' effort (in the form of distinct touches);
Product metrics	size achieved in terms of SLOC (sources lines of code) and duration of the effort (number of days of activity observed in either the CVS or SVN repository).

3.1. Code repositories

This study mines data from six FLOSS repositories (listed below) to address the above questions. In order to achieve a reasonable comparison, in all cases, the samples have been drawn from a pool of "stable" projects from these repositories described below:

(1) The Debian project (http://www.debian.org/) hosts a large number of FLOSS projects under a common name. At the time of writing, more than 20,000 projects are listed under the "stable" label of the latest version. Projects analyzed in this study must have this label.
(2) GNOME (http://gnome.org) is a desktop environment and development platform for the GNU/Linux operating system. Its software repository is organized into more than 600 software programs. Whilst GNOME has no method of explicitly designating projects as "stable", projects sampled for this study come only from the main development trunk – other locations such as branches or incubators are not considered.
(3) KDE (http://kde.org) is another desktop environment and development platform for the GNU/Linux operating system. Like GNOME, KDE does not explicitly label the development status of projects, so samples are drawn from the main development trunk only.
(4) The RubyForge website (http://rubyforge.org) acts as a development management system for over 4,500 projects programmed in the Ruby programming language, 360 of which are labelled as "Production/Stable".
(5) The Savannah project acts as a central point for the development of approximately 2850 free software projects (http://savannah.gnu.org and http://savannah.nongnu.org). The Savannah sample has been drawn exclusively from the set of projects marked "Production/Stable".
(6) Finally, the SourceForge site (http://sourceforge.net) hosts more than 150,000 projects. The sample from SourceForge has been extracted from only the pool of projects whose core developers have labelled the status of the project with the tag "Production/Stable".

Each repository has a sample of 50 individual projects chosen from it by a randomizer, and a checkout has been performed on each member project of each sample (from either their CVS or SVN source control repositories). The list of analyzed projects is shown in Appendix. Each of these sources has been analyzed to obtain the metrics needed to perform the investigation; the measures for the study are introduced in the section below.

3.2. Measured characteristics

In order to compare these repositories with their different characteristics, scope and underlying communities, the following, common characteristics have been measured to build a table of results for each project. Those noted in **bold** are the actual attributes used in the paper to evaluate and test the empirical hypotheses. The relevant definitions and measured characteristics are as follows:

- Commit: the atomic action of a developer checking in one or more files (being source code or other) into a central repository.
- Committers: this information has been recorded in two ways: firstly by assigning the activity to the actual developer who placed the file into the repository, and automatically recorded it; and secondly by using any further developers who were mentioned in the commit, by means of mentioning his/her involvement in the coding or patching. This information has been used to characterize the involvement of distinct developers in each project.
- Modules and subsystems: at a fine granular level, both CVS and SVN repositories record activity on files (here termed as "modules") and their containing folder (termed "subsystem").
- Date: CVS/SVN repositories record the time when the module and its subsystem was modified or created from scratch, typically in the ISO formatting "YYYY-MM-DD". For the purpose of this work, a date in the form "YYYY-MM" has been recorded, hence this work has analyzed the activity and the involvement of developers on a monthly basis only.
- **Size**: the size of the project has been detected from the source code, which is measured in SLOCs. It has been shown in the past that this metric, when used within a sample of FLOSS projects, had similar growth patterns to other metrics related to size, specifically number of files or modules (Herraiz et al., 2006) and number of folders containing source code (Capiluppi et al., 2004). Moreover this metric is part of the most widely used techniques in various research fields, for example cost estimation (Boehm et al., 2000). Although several criticisms were raised in the past, it is possible to postulate that it is a viable approach for measuring size of a software system. Alternative measures to SLOCs such as function points are also affected by problems and weaknesses, given their strong theoretical support based on Halstead's software science (Hamer and Frewin, 1982; Shen et al., 1983).
- **Duration**: the age of the project is measured by number of days over which the project has been developed. This has been evaluated using the earliest and the latest available dates in the CVS/SVN repositories. This measurement shows the time-span over which FLOSS developers adopted a Software Configuration Management server in order to enable distributed development to be properly performed.
- **Distinct touches**: since many modules and subsystems can be committed in the repository within the same commit, and the same module could have been modified by more than one developer in the same commit, the term "touch" has been used to isolate the atomic information of a unique developer, unique date

and unique union on module and subsystem. As an example, if a project obtains 200 commits in a given month, but those apply to the unions {subsystemA/moduleA, *e.g.* 100 commits}, {subsystemB/moduleB, *e.g.* 80 commits} and {subsystemC/moduleC, *e.g.* 20 commits}, the activity recorded for that specific month is only 3. This information is gained automatically by parsing the CVS/SVN log for the project (see Fig. 1); in this case, the committer (yakk) is held responsible for the effort together with the author of the patch (buddy@email.com).

- **Distinct developers**: information on distinct contributing developers is obtained by analyzing commit logs (see the sample in Fig. 1). Every touch has an committer ID attached to it and this is recorded. The information is then filtered through a "distinct" clause: specifically, the number of unique developers per month is recorded, in order to record the real involvement of developers and to avoid cumulative effects.

3.3. Distribution of data in the repositories

The distributions of the projects within the samples are represented in boxplots (Tukey, 1977) in Fig. 2. Both the distributions of SLOCs and Project Duration (top left and top right, respectively) are composed of one measurement per project per repository: the SLOCs at the time of extraction (October 2007) and the length (in days) of the configuration management logs retrieved respectively for each project. The other two boxplots represent a measure of the activity, in terms of distinct commits, and the involvement of distinct developers (bottom left and bottom right, respectively). Since nearly all the projects in each sample span several months, a single value in each case (*i.e.*the median) has been recorded, per project and per repository, to summarize the evolution of these attributes.

As is visible in Fig. 2, the sample from the Debian repository appears to achieve the largest sizes in SLOCs, with several outliers

```
------------------------------------------------------------------------
r7 | yakk | 2003-11-04 09:07:21 +0000 (Tue, 04 Nov 2003) | 2 lines
Changed paths:
M /trunk/ChangeLog
M /trunk/src/Makefile.am

little tiny build fix, patched by   buddy@email.com   '
------------------------------------------------------------------------
```

Fig. 1. Excerpt from a sample SVN log file showing one commit that touches two files.

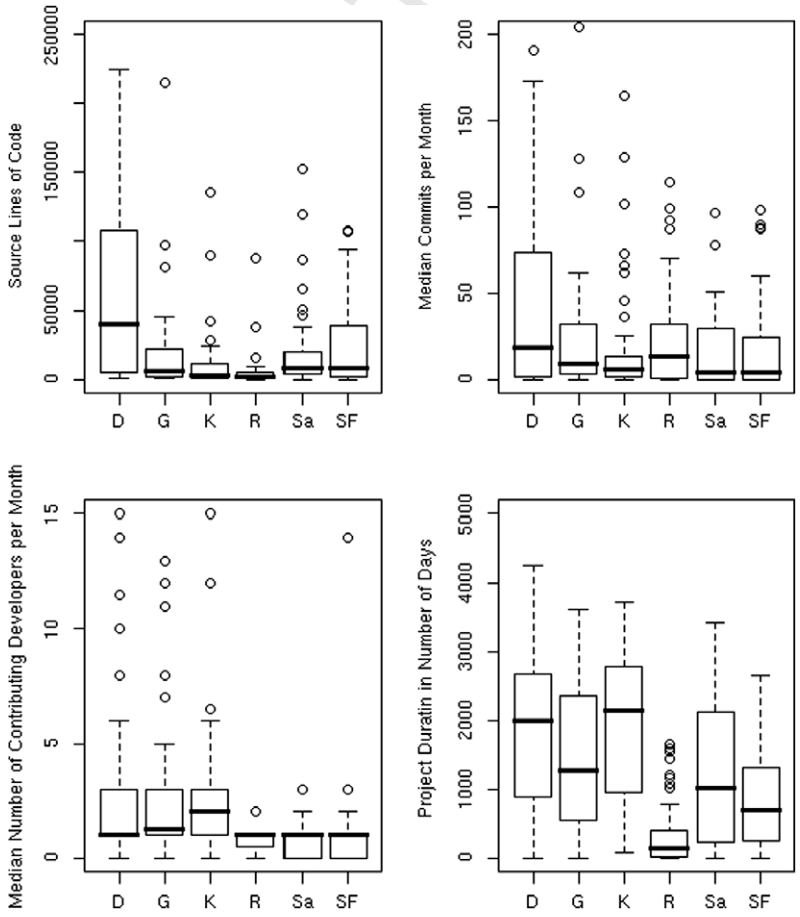

Fig. 2. Boxplots of the four attributes under investigation for (D)ebian, (G)NOME, (K)DE, (R)ubyForge, (Sa)vannah and SourceForge (SF).

Please cite this article in press as: Beecher, K. et al., Identifying exogenous drivers and evolutionary stages in FLOSS projects, J. Syst. Software (2008), doi:10.1016/j.jss.2008.10.026

skewing its distribution (median: 39,590 SLOCs). The samples from the other repositories, excluding the outliers, appear comparable with one another, in terms of medians. In contrast, observing the three distributions of duration, commits and developers, a first group (Debian, KDE and GNOME) appears as achieving consistently larger values than a second group (RubyForge, Savannah and SourceForge). This is especially true for the number of distinct developers per month, which is typically capped by at most "one developer" in the second group; and for the activity of monthly commits, which has a larger variance in the first group.

The application domains of the six samples has also been studied. Table 1 summarizes the domains as collected for every project composing all the samples, and shows the relative percentage of each domain of the sample. These domains are those used within SourceForge site to effectively cluster the FLOSS projects. As visible, the back-end applications ("System" and "Software Development") together form some 40% of all the topics. Appendix details the domains for each sample. Several different domains are detected also in KDE and GNOME, although one would expect that most of their projects to fall into a "Desktop Environment" category (see Table 2).

The initial observation of these distributions has led to the formulation of the following research questions:

- Is it true that all the repositories produce similar results in terms of the process and product characteristics studied?
- Is it possible to group repositories into categories that achieve statistically different results?

In the following Section 4, the first of these questions will be analyzed statistically, by comparing each distribution with the others on the four presented characteristics.

3.4. Statistical Tests

Each data set is compared to another using the unpaired Wilcoxon test. The critical information of this test is as follows:

Table 1
Summary of the application domains. The shortcut is also used in Appendix.

Application domain	Shortcut	Ratio
Communications	A	9%
Database	B	3%
Desktop environment	C	5%
Education	D	1%
Formats and protocols	E	2%
Games/entertainment	F	8%
Internet	G	13%
Multimedia	H	8%
Office/business	I	1%
Other/nonlisted topic	J	1%
Printing	K	0
Rails	L	1%
Scientific/engineering	M	7%
Security	N	%
Software development	O	20%
System	P	19%
Terminals	Q	0
Text editor	R	2%

Table 2
Summary of statistical tests used and their purpose per hypothesis.

	Statistical test	What the test establishes
Hypothesis 1	Bi-directional unpaired Wilcoxon	If the two samples are from different probability distributions
Hypothesis 2	One-directional unpaired Wilcoxon	If one sample is from a probability distribution that differs from another in a specified direction

- Its **purpose** is to test whether or not the two samples are from different probability distributions.
- It **assumes** that the probability distribution is non-parametric, and that the two samples are independent.

The resultant value (the p-value) informs of the probability that random sampling of two populations would lead to a difference between the sample means as large (or larger) than that observed in the samples. By convention 0.05 or less is the desired p-value.

4. Results: research question 1 – hypotheses

The further research to answer the first question has been designed as a direct comparison between the six samples of stable projects extracted from the repositories, and its objective is to highlight any significant difference based on the chosen characteristics. Four hypotheses have been formulated and evaluated empirically. Given the null hypothesis and the alternative hypotheses in the second and third columns of Table 3, a statistical test (Wilcoxon, 1945) will allow the null hypothesis to be either rejected or confirmed. The threshold for the significance of the p-values will be modified with the Bonferroni correction (Cabin and Mitchell, 2000); although this approach has suffered from criticisms in the past (Perneger, 1998), it is relevant here because multiple tests are being carried out. The R programming language has been used to carry out these tests based on evaluation of the data extracted earlier from the respective repositories Dalgaard, 2002. A summary of the tests and their results will be provided at the end of this section to draw together the relevant conclusions. Each hypothesis is briefly introduced in the following:

- *Hypothesis 1.1 – Size achieved:* The first hypothesis postulates that the typical size of a project differs significantly for each repository, in terms of SLOC, with the null hypothesis stating that all the repositories have similar project sizes, to be rejected if project sizes are shown to be significantly different.
- *Hypothesis 1.2 – Activity (touches):* The second hypothesis for question 1 postulates that the amount of observed activity (or output) differs among repositories. Specifically, the null hypothesis states that, on average, individual projects in one repository will have a number of distinct touches that does not differ significantly from that found among the others. This may be rejected if it is shown that specific repositories tend to have significantly more active projects than others.

Table 3
Summary of the hypotheses, tests and metrics.

Hypothesis 1.1: distribution of size – Test T1.1		
H1.1 (null): Projects from all repositories have a similar size	H1.1 (alternative): Projects from repositories have different sizes	SLOCs
Hypothesis 1.2: overall touches – Test T1.2		
H1.2 (null): Projects from all repositories have a similar amount of touches	H2.1: Projects from repositories have significantly more or fewer touches	Distinct touches
Hypothesis 1.3: distinct developers – Test T1.3		
H1.3 (null): Projects from all repositories have a similar amount of developers	H1.3 (alternative): Projects from repositories have significantly more or fewer developers	Distinct committers
Hypothesis 1.4: days of evolution – Test T1.4		
H1.4 (null): All projects have similar time-spans	H1.4 (alternative): Projects from repositories have significantly longer or shorter time-spans	Days
Statistical test used: bi-directional unpaired Wilcoxon test, tolerance 5%		

Please cite this article in press as: Beecher, K. et al., Identifying exogenous drivers and evolutionary stages in FLOSS projects, J. Syst. Software (2008), doi:10.1016/j.jss.2008.10.026

Table 4
Results – question 1. T1 = SLOCs, T2 = commits, T3 = committers, T4 = days. Underlined entries have a non-significant p-value, but a large Cohen's Effect Size.

	Debian	GNOME	KDE	RubyForge	Savannah	SourceForge
Debian	–	T1: $\mathbf{p = 0.226 \times 10^{-3}}$ T2: $\underline{\mathbf{p = 0.464}}$ T3: $p = 0.279$ T4: $p = 0.0553$	T1: $\mathbf{p = 2.29 \times 10^{-6}}$ T2: $p = 0.136$ T3: $p = 0.0246$ T4: $p = 0.986$	T1: $\mathbf{p = 0.537 \times 10^{-9}}$ T2: $\underline{p = 0.295}$ T3: $\mathbf{p = 4.45 \times 10^{-3}}$ T4: $\mathbf{p = 0.185 \times 10^{-9}}$	T1: $\mathbf{p = 2.53 \times 10^{-3}}$ T2: $\mathbf{p = 0.0117}$ T3: $\mathbf{p = 0.0121}$ T4: $\mathbf{p = 0.0180}$	T1: $\mathbf{p = 0.0149 \times 10^{-3}}$ T2: $\underline{p = 0.0235}$ T3: $\mathbf{p = 4.80 \times 10^{-3}}$ T4:$\mathbf{p = 14.0 \times 10^{-6}}$
GNOME		–	T1: $p = 0.830$ T2: $p = 0.130$ T3: $p = 0.176$ T4: $p = 0.0343$	T1: $\mathbf{p = 0.404 \times 10^{-6}}$ T2: $\underline{p = 0.7636}$ T3: $\mathbf{p = 2.14 \times 10^{-6}}$ T4: $\mathbf{p = 92.1 \times 10^{-9}}$	T1: $p = 0.491$ T2: $p = 0.0248$ T3: $\mathbf{p = 3.90 \times 10^{-5}}$ T4: $p = 0.537$	T1: $p = 0.216$ T2: $\underline{p = 0.0512}$ T3: $\mathbf{p = 1.04 \times 10^{-5}}$ T4: $\mathbf{p = 0.0106}$
KDE			–	T1: $\mathbf{p = 5.64 \times 10 - 3}$ T2: $\underline{p = 0.417}$ T3: $\mathbf{p = 2.12 \times 10^{-10}}$ T4: $\mathbf{p = 0.691 \times 10^{-12}}$	T1: $\mathbf{p = 0.0150}$ T2: $p = 0.258$ T3: $\mathbf{p = 2.64 \times 10^{-8}}$ T4: $\mathbf{p = 6.33 \times 10^{-3}}$	T1: $p = 0.0390$ T2: $\underline{p = 0.253}$ T3: $\mathbf{p = 6.94 \times 10^{-9}}$ T4: $\mathbf{p = 1.58 \times 10^{-6}}$
RubyForge				–	T1: $\mathbf{p = 1.24 \times 10^{-6}}$ T2: $\underline{p = 0.175}$ T3: $p = 0.928$ T4: $\mathbf{p = 3.04 \times 10^{-6}}$	T1: $\mathbf{p = 2.94 \times 10^{-5}}$ T2: $\underline{p = 0.144}$ T3: $p = 0.652$ T4: $\mathbf{p = 3.07 \times 10^{-5}}$
Savannah					–	T1: $p = 0.850$ T2: $\underline{p = 0.946}$ T3: $p = 0.639$ T4: $p = 0.0983$
SourceForge						–

- *Hypothesis 1.3 – Developers:* This hypothesis posits that the number of distinct developers that work on a project monthly is, on average, significantly different for each repository, measured according to the number of distinct developers who have contributed source code. The null hypothesis states that all the projects have approximately an equal number of contributing developers, to be rejected if this is not the case.
- *Hypothesis 1.4 – Period of activity:* The final hypothesis posits that the duration of time that projects from each repository have been evolved over differs significantly, measured by the number of days for which activity could be observed on a project's repository. In statistical terms, the null hypothesis underlies the presumption that FLOSS projects come from different populations; based on their original repository, they have a different timespan. The null hypothesis should be rejected if the sample projects display significant differences between repositories.

4.1. Results of the tests – research question 1

The empirical evaluation of the first research question has led to a total of 15 direct comparisons among the 6 repositories, and has been based on the 4 hypotheses expounded above. The results of the tests are reported in Table 4 where each comparison between two repositories is displayed along with the test name (T1–T4) and a report of the p-value from the Unpaired Wilcoxon Test Wilcoxon, 1945. Since the stringent difference between two repositories is being tested, a two-sided test has been performed in all cases; the Bonferroni correction has been applied giving a significance threshold of $p\text{-value} \leqslant 0.01$. Bold figures denote p-values that are lower than (or extremely close to) this threshold; underlined figures denote p-values that are higher than this threshold, but correspond with a large Cohen's Effect Size (see Table 5).

The following observations have been made, based on the direct comparisons among repositories: (see Table 6)

(1) **Size achieved:** Table 4 clearly shows significant differences between repositories based on their project sizes. Observations are as follows:
   - Debian differs significantly from all other repositories;
   - GNOME and KDE do not differ – both repositories' results are mixed when compared to others;
   - Savannah does not differ from SourceForge, whereas RubyForge differs from them both.

(2) **Commits:** After performing corrections on the p-values, no repositories indicate significantly different levels of activity from each other. The results also suggest that RubyForge activity is comparable to that of all other repositories (with the exception of KDE). However it can be seen in Fig. 2 that RubyForge projects have a significantly shorter life than those found in all other repositories, which increases their perceived rate of activity. Hence the perceived rate of activity of RubyForge projects should be treated with suspicion.

(3) **Committers:** Table 4 allows the following observations to be made about the number of contributing developers to each repository:
   - Debian does not differ significantly from GNOME and KDE, but the p-values are borderline when compared to RubyForge and SourceForge;

Table 5
Summary of Cohen's effect size – (SLOCs; commits; committers; days)

	Debian	GNOME	KDE	RubyForge	Savanah	SourceForge
Debian	–	(0.59; 0.41; 0.06; 0.37)	(0.65; 0.08; 0; 0.52)	(0.71; 1.82; 0.85; 0.43)	(0.25; 0.43; 0.77; 0.04)	(0.49; 1.12; 0.59; 0.55)
GNOME		–	(0.24; 0.34; 0.07; 0.29)	(0.56; 1.36; 1.05; 0.07)	(0.25; 0.04; 0.92; 0.18)	(0.24; 0.66; 0.65; 0.33)
KDE			–	(0.34; 1.82; 1.24; 0.26)	(0.32; 0.37; 1.09; 0.26)	(0.46; 1.08; 0.77; 0.02)
RubyForge				–	(0.38; 1.21; 0.14; 0.2)	(0.71; 0.84; 0.19; 0.3)
Savanah					–	(0.2; 0.88; 0.72; 0.1)
SourceForge						–

Table 6
Summary of the hypotheses, tests and metrics

Hypothesis 2.1: distribution of size – Test T2.1		
H2.1 (null): Projects from the two groups have similar size	H2.1 (alternative): Projects from the first group have larger sizes	SLOCs
Hypothesis 2.2: overall touches – Test T2.2		
H2.2 (null): Projects from the two groups have a similar amount of touches	H2.2: Projects from the first group have significantly more touches	Distinct touches
Hypothesis 2.3: distinct developers – Test T2.3		
H2.3 (null): Projects from all the two groups have a similar amount of developers	H2.3 (alternative): Projects from the first group have significantly more developers	Distinct committers
Hypothesis 2.4: days of evolution – Test T2.4		
H2.4 (null): Projects from the two groups have similar time-spans	H2.4 (alternative): Projects from the first group have significantly longer time-spans	Days
Statistical test used: one-directional unpaired Wilcoxon test, tolerance 5%		

- GNOME and KDE do not differ significantly from each other, but they both show a significant difference from RubyForge, Savannah and SourceForge;
- RubyForge, Savannah and SourceForge do not differ from one another.

(4) **Duration:** In Table 4 the absence of significant differences between repositories suggests:

- Debian and KDE have a significantly different average project age from RubyForge and SourceForge;
- GNOME has a significantly different project age from RubyForge;
- Savannah does not differ from SourceForge.

In summary, a significant difference has been identified between the characteristics of the repositories, dividing them firmly into two groups:

- **Group 1**: Debian, GNOME and KDE (with GNOME and KDE displaying very notable similarities);
- **Group 2**: RubyForge, Savannah and SourceForge (with Savannah and SourceForge displaying very notable similarities).

5. Results: research question 2 – hypotheses

The first research question has been designed as a comparison between all repositories, and its objective is to highlight any significant differences between them. Having established these differences the second question is designed to establish their direction and hence the supremacy of one repository over any other with regards to the characteristics under study.

In each hypothesis it has been posited that the repositories in group 1 (labelled in Section 4.1) have a superior value to those of group 2.

- *Hypothesis 2.1 – Period of activity:* This hypothesis posits that projects from group 1 (Debian, GNOME and KDE) have been developed for greater periods of time than group 2 (RubyForge, Savannah and SourceForge), and hence possess a significantly longer duration of activity.
- *Hypothesis 2.2 – Size achieved:* This hypothesis postulates that the group 1 projects are typically larger than their group 2 counterparts and hence have significantly larger SLOCs.
- *Hypothesis 2.3 – Developers:* This hypothesis posits that the projects from Debian, GNOME and KDE are more successful at attracting developers than projects from RubyForge, Savannah and SourceForge. They should therefore show evidence of a significantly greater number of developers.
- *Hypothesis 2.4 – Activity (touches):* The final hypothesis postulates that group 1 projects typically receive more development effort than those of group 2, evidenced by a significantly larger rate of touches.

Each repository has been compared to each counterpart repository of the opposing group. With three repositories in each group this has resulted in nine such comparisons. In each such comparison it has been hypothesized that there is a significant difference in a specified direction, which has been estimated by inspecting the boxplot for the relevant attribute (Fig. 2). The results of the tests are summarized in Table 7. As in Section 4.1, each comparison is displayed by test name (T1–T4) showing the resulting *p*-value

Table 7
Results – question 2. T1 = SLOCs, T2 = commits, T3 = committers, T4 = days

	RubyForge	Savannah	SourceForge.net
Debian	T1: $W = 2101$, $\boldsymbol{p} = 0.447 \times 10^{-9}$ T2: $W = 1402$, $p = 0.1472$ T3: $W = 1626$, $\boldsymbol{p} = 2.22 \times 10^{-3}$ T4: $W = 2174$, $\boldsymbol{p} = 92.4 \times 10^{-12}$	T1: $W = 1626$, $\boldsymbol{p} = 1.27 \times 10^{-3}$ T2: $W = 1561$, $\boldsymbol{p} = 0.00885$ T3: $W = 1596$, $\boldsymbol{p} = 6.08 \times 10^{-3}$ T4: $W = 1594$, $p = 9.02 \times 10^{-3}$	T1: $W = 1365$, $\boldsymbol{p} = 7.50 \times 10^{-3}$ T2: $W = 1575$, $p = 0.0118$ T3: $W = 1637$, $\boldsymbol{p} = 2.40 \times 10^{-3}$ T4: $W = 1881$, $\boldsymbol{p} = 7.02 \times 10^{-6}$
GNOME	T1: $W = 1846$, $\boldsymbol{p} = 2.02 \times 10^{-5}$ T2: $W = 1294$, $p = 0.382$ T3: $W = 1876$, $\boldsymbol{p} = 1.07 \times 10^{-6}$ T4: $W = 2024$, $\boldsymbol{p} = 46.1 \times 10^{-9}$	T1: $W = 1126$, $p = 0.805$ T2: $W = 1545$, $p = 0.0124$ T3: $W = 1819$, $\boldsymbol{p} = 1.95 \times 10^{-5}$ T4: $W = 1340$, $p = 0.269$	T1: $W = 914$, $p = 0.738$ T2: $W = 1532$, $p = 0.0256$ T3: $W = 1858$, $\boldsymbol{p} = 0.520 \times 10^{-5}$ T4: $W = 1621$, $\boldsymbol{p} = 0.00532 \times 10^{-3}$
KDE	T1: $W = 1652$, $\boldsymbol{p} = 2.82 \times 10^{-3}$ T2: $W = 1132$, $p = 0.794$ T3: $W = 2102$, $\boldsymbol{p} = 1.06 \times 10^{-10}$ T4: $W = 2292$, $\boldsymbol{p} = 0.0345 \times 10^{-12}$	T1: $W = 877$, $p = 0.995$ T2: $W = 1413$, $p = 0.130$ T3: $W = 2026$, $\boldsymbol{p} = 1.32 \times 10^{-8}$ T4: $W = 1646$, $\boldsymbol{p} = 3.17 \times 10^{-3}$	T1: $W = 950$, $p = 0.981$ T2: $W = 1416$, $p = 0.127$ T3: $W = 2056$, $\boldsymbol{p} = 3.47 \times 10^{-9}$ T4: $W = 1947$, $\boldsymbol{p} = 0.787 \times 10^{-6}$

Please cite this article in press as: Beecher, K. et al., Identifying exogenous drivers and evolutionary stages in FLOSS projects, J. Syst. Software (2008), doi:10.1016/j.jss.2008.10.026

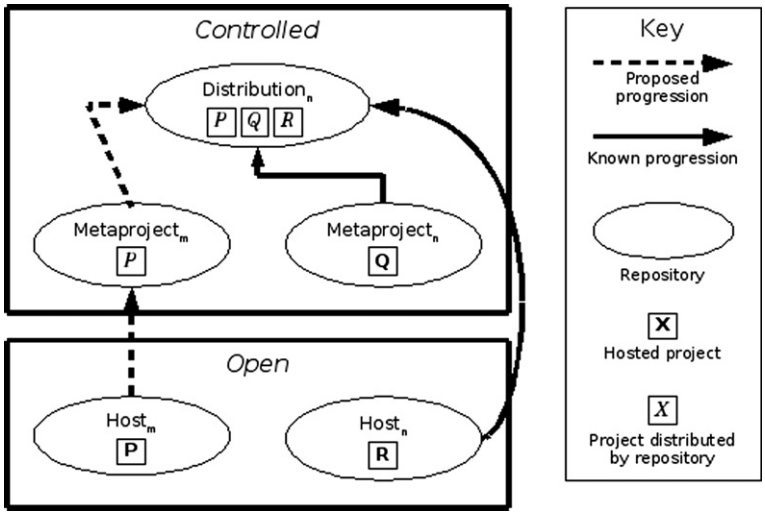

Fig. 3. Framework of progression for FLOSS projects through various types of repositories.

obtained from a one-sided unpaired Wilcoxon test; the Bonferroni correction has been applied giving a significance threshold of $p \leqslant 0.0083$.

5.1. Results of the tests – research question 2

The following observations can be made in this case:

- *Activity:* the directional Wilcoxon tests (Table 7) show that the median commits per month is the only indicator that does not display any significant differences, although on average Debian, GNOME and KDE all have greater average commit rates (between 5 and 18) than Savannah and SourceForge (around 4 commits per month each). Recall that RubyForge commit rates are suspected of being artificially high.
- *Size achieved:* the directional Wilcoxon test (see Table 7) confirms that KDE and GNOME projects are smaller on average than Savannah and SourceForge projects. Debian contains the largest projects of all repositories under study.
- *Number of developers:* the directional Wilcoxon tests (Table 7) show that there is a divide between Debian, GNOME and KDE as one group, and RubyForge, Savannah and SourceForge as another. The former group have between 20 and 32 projects with a number of contributors exceeding one, whereas the Sourceforge sample (the best performing for this indicator in the latter group) had only 6 projects with more than one contributor.
- *Project duration:* the directional Wilcoxon tests again shows a clear divide between Debian, GNOME and KDE which consistently have older projects and RubyForge, Savannah and SourceForge (which consistently have younger projects).

6. Discussion – a framework for transitions of FLOSS projects

The empirical evidence gathered by analyzing data to answer the two research questions above shows a significant divide between one group (Debian, KDE and GNOME) and a second group of repositories (RubyForge, Savannah and SourceForge). In previous works, those of the first group have already been characterized. The Debian repository has been extensively studied, and its internal product and process characteristics have been described in terms of a successful product Michlmayr and Senyard, 2006, 2007. Also KDE and GNOME have been evaluated as successful repositories, based on the characterization of their development processes and recruitment rates Hemetsberger and Reinhardt (2004) and Koch et al. (2002) respectively.

As already reported above, an earlier work highlighted the differences between SourceForge and Debian (Beecher, ~~XXXX~~). These were not only statistically significant, but also directional (Debian achieves better results than SourceForge). Investigating further, it was also found that a subset of projects had transited from SourceForge to Debian. In these cases, the projects being incorporated into Debian from SourceForge achieved, from that point in time on, an improvement in the overall activity and an increased number of developers.

6.1. Transition framework – types of repositories

The earlier results and the differences between the two groups outlined in this work are here contextualized by a wider framework of evolution developed as part of this research, visualized here in Fig. 3. The terms used within the framework formulation are as follows:

(a) Open Forge – the term is applied to those FLOSS repositories with a low barrier to entry: RubyForge, Savannah and SourceForge all guarantee any FLOSS developer the availability of web-space and management tools (e.g. CVS, forums) to host a software project (see lower part of Fig. 3).

(b) Controlled Forge – this term is instead used for those repositories which apply various filters and guidelines to newly joining projects (top part of Fig. 3). Debian, KDE and GNOME clearly have underlying rules, standards and specific tools for developers to adhere to or to adopt when joining. Debian accepts a new project only after an advocate from within Debian issues a request to include it (Laat and Paul, 2007). KDE requests new developers to adhere to programming standards, and to comply with an existing C++ code-base (Kuniavsky and Raghavan, 2005). Finally, GNOME requests the knowledge of the basic GTK graphical platform, and its Application Programming Interfaces (API) (German, 2004). Within this group, the nature of the repository and the empirical results showed the following distinction between two subgroups:

(b.1) *Distributions*: Debian should be considered a distribution, since the FLOSS projects it hosts are all part of a larger Linux operating system. From the point of view

of the process, Debian developers are not typically programming for other Debian projects.

(b.2) *Meta-Projects*: KDE and GNOME should be instead considered as meta-projects because the projects they contain are subsystems of a wider system (the KDE and GNOME desktop environments, respectively), and developers work on several glue projects.

The main difference between distributions and meta-projects is in the higher acceptance threshold of the former. As already mentioned, a software project becomes part of a distribution only under specific conditions, such as, for instance, the Debian advocates with new FLOSS projects (Laat and Paul, 2007).

6.2. Transition framework – types of transitions

The transitions among the repositories are noted with two types of arrows:

- As per the **bold arrow** of Fig. 3, the study reported above (Beecher, XXXX) empirically showed not only how a subset of projects transited from an Open Forge (SourceForge) into a Controlled Forge (Debian). It also showed that this transition had an effect on the two selected aspects of product and process; the transited projects were shown to have benefited from more developers and displayed an increased activity. FLOSS projects within the KDE and GNOME repositories have an easier entry point within Debian, since the distribution typically ships both the two meta-projects; for this reason, also this transition is depicted as a bold line. Projects belonging to "open repositories" have a less straight-forward entry-level into major distributions; at least one Debian developer has to act as an "advocate", or "sponsor" for its introduction within the distribution, otherwise it will be considered as "non-interesting".[3]
- The **dashed lines** depicted in Fig. 3 represent instead proposed transitions between repositories; projects have been observed to migrate along the dashed lines, but no empirical study on the benefits of these transitions has been performed in this research. Among the sample from SourceForge, for instance, it is possible already to detect projects which transited to a meta-project (*e.g.*, the *kpictorial* project is also included in the KDE repository, Appendix). A transition from a meta-project to a distribution has also been observed; a subset of projects have been observed transiting between the KDE meta-project to the Debian distribution (*e.g.*, the *ark* from the KDE repository, Appendix). As also stated in (Michlmayr, 2007), one of the advantages of being part of a larger distribution consist of having a more pressing schedule due to the *release management*. Open Forges do not use a formal way of imposing schedule constraints and deadlines; controlled Forges typically do, thus placing an heavier burden on the the developers and requiring higher productivity within the contained projects.

6.3. Transition Framework – discussion

Regarding the general goal stated in Section 3, this research shows that a general framework relating different *types* of FLOSS repositories provides a better context to describe the variety of results (in terms of success) of the average FLOSS project. This framework establishes the possible routes that may be taken to achieve these results and their relative benefits and challenges.

The corollary of this objective, also stated above, is that useful findings and practical actions could be extrapolated for the use of developers and practitioners. As shown in (Capiluppi and Michlmayr, 2007) both of the so-called "cathedral" and "bazaar" modes of operation can co-exist within FLOSS; projects begin in a cathedral mode and may, if they wish, change later to a bazaar mode and thereby increase visibility, activity and size. Fig. 4, taken from (Capiluppi and Michlmayr, 2007), summarizes that FLOSS projects can just achieve one state (the "cathedral", left part of figure), while in the life cycle of other projects, the "bazaar" state can follow the cathedral phase, thus achieving an increased effort and greater output. Building on this earlier result, we can claim that some forges (SourceForge, RubyForge, Savannah) on average host projects mostly in their cathedral phase; if developers wish to upgrade the status of their project, and exploit the advantages of the wider FLOSS communities, they should consider being included into a meta-project or a distribution repository. This, however, is not a mandatory move; and making such a transition may require the project to alter their working practices and follow a more managed release strategy.

7. Threats to the validity of this study

The following aspects have been identified which could lead to threats to validity of the present empirical study; they have been grouped into threats to construct, internal and external validity as follows.

- *Construct validity (relationship between theory and observation)*Missing historical data – the study has been able to make use only of available data. It is possible, for example, that the project initialization pre-dates the first measurable piece of historical data and is therefore beyond the reach of our analysis.
- *Internal validity (confounding factors can influence the findings)*
  (1) Status of the projects – as indicated, all projects studied are chosen for being "stable", in order to counter the problems of comparing projects at differing stages of evolution. However the lesser threat remains that the projects studied are at differing stages of evolution because the definition of "stable" varies across the repositories and is somewhat subjective.
  (2) Outliers – a very small number of outliers were discovered within the data, and were subsequently excluded from the analysis. Specifically the following have been excluded:
    - Size: a single extreme outlier was identified in each of the Debian and Savannah samples.
    - Activity: a single outlier was identified in the Savannah sample. Upon investigation, it appeared that an existing project with a long history had been imported all at once and was then never worked upon again. Since the addition of each file constitutes a commit, the average of commits per month was artificially large.
- *External validity (how results can be generalized)*

(1) Union of sets – the permissive nature of FLOSS development means that it is possible, even encouraged, for individual projects, or parts of them, to be included in more than one repository. Hence, when randomly sampling projects from individual repositories, it is possible that a sampled project may be found in another location and that its evolution is also influenced by this unknown repository. The assumption is therefore made that any such confounding effect, if present, is negligible.

[3] The specific Debian process to become a developer, or how to include a new project within the distribution is detailed under http://people.debian.org/mpalmer/debian-mentors_FAQ.html.

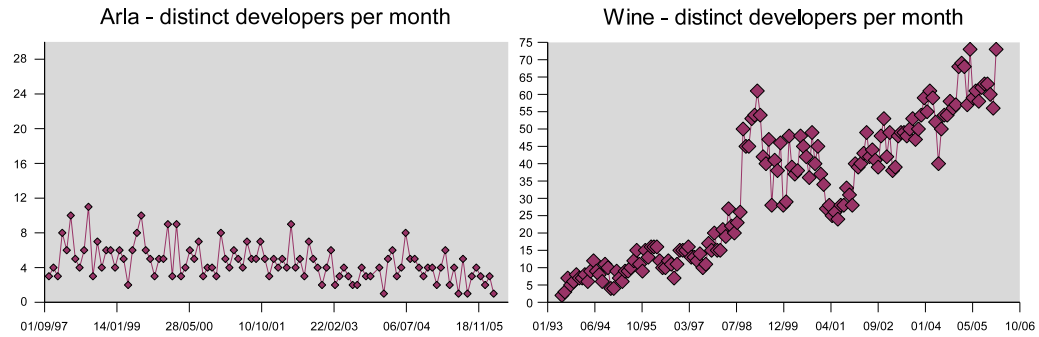

Fig. 4. Transition between Cathedral and Bazaar, taken from (Capiluppi and Michlmayr, 2007).

(2) Size of the populations – perhaps the most variable characteristic of the repositories was the number of available projects. To promote the fairness with which projects could be compared we endeavoured to select ones considered "stable". Consequently the number of projects available ranged from approximately 400 (in the case of RubyForge) up to the order of 20,000 (for SourceForge). The static sample size of 50 resulted in each sample being not necessarily proportionate to all others.
(3) Further generalization – for each repository type proposed within this work, it has been represented by a small number of repositories (between one and three). This approach holds the risk that results may be biased by any peculiarities of individual repositories. However, this study is relative easily replicable on other repositories of the types identified. For example, other meta-projects exist with development emphasis on other domains not covered in this study, such as Mozilla or Apache.

8. Conclusions and future works

This study has been carried out as an extension of previous research (Beecher, XXXX), aiming to amplify a promising set of findings obtained when comparing the characteristics of two different FLOSS repositories, Debian and SourceForge. The present study expanded the previous data base with four other repositories, extracted similar samples from each of the resulting six repositories (50 projects each from the repository's "stable" pool), and studied four product and process characteristics of the projects in the samples.

Testing whether similar results can be obtained by researchers when studying any FLOSS repository, it was found that not only do repositories differ from each other in terms of product or process characteristics (or both); but also that two groups showed significant differences between them. A first group (Debian, KDE and Debian) showed consistently different characteristics in comparison with a second group (RubyForge, Savannah and SourceForge). Later, it was also shown that two repositories (Debian and KDE) in the first group achieve significantly better results than those in the second group.

Combining the two above findings, a framework for the evolution of FLOSS repositories has been proposed. RubyForge, SourceForge and Savannah do not pose any barriers to entry to software projects (although RubyForge projects are generally expected to be related to the Ruby programming language) and named "open forges". On the other hand, repositories such as Debian ("controlled forges") set up a higher threshold to admittance, typically by introducing new projects in a stricter, controlled way. Among these, both full distributions such as Debian, and meta-projects, such as KDE or GNOME, offer the potential of a wider spectrum of developers and increased activity.

Transitions were also studied: FLOSS projects transiting from an "open forge" to a "controlled forge", will be able to exploit the benefits of a larger audience of users and developers, and become, on average, also larger projects. The transition between a "meta-project" to a "distribution" has also been postulated as a further advance in the evolution of a FLOSS project; however, from the KDE and GNOME samples, it was not possible to observe any cases where a project was introduced in the Debian distribution, in order to evaluate the effects of the Debian *treatment*.

The presented work has two main research strands which we propose to consider further in our future work. The first is to introduce other repositories (or forges) into the quantitative study, in order to achieve an improved understanding of the distribution of FLOSS projects within repositories. Major repositories such as Tigris[4] or FreshMeat[5] could be analyzed, following the same approach, characteristics and hypotheses used above. Given their policies, which make these two more similar to SourceForge than Debian, the framework as proposed in Section 6 would place them into the "Open Forges" category. A research hypothesis would then be used to test whether these two repositories achieve (on average) worse results than KDE, GNOME or Debian. Other, specialized forges such as OW2[6] could be also analyzed; given its policy of semi-openness, our framework would place it under the "Controlled Forges", hence (in theory) achieving better results than an open forge.

The second research strand yet to be pursued is a closer, quantitative, investigation of the transitions as proposed in the framework; we propose to conduct an observational study where a known subset of projects from either KDE or GNOME, which have been introduced at some stage in the Debian distribution, is studied both before and after their introduction. This will give a stronger, empirical foundation to the framework, and allow the dashed (i.e., proposed) transitions to be replaced with continuous (i.e., observed) lines.

Overall, additional metrics could provide even more insights into FLOSS quality and support for understanding better how projects from the various forges perform on quality aspects (apart from their already studied productivity). We plan to introduce metrics related to software complexity (such as the cyclomatic number of methods and functions, or the coupling among methods, files and packages), and a better characterisation of the *touch* metric, by considering the amount of code modified in each touch (i.e. with the ratio SLOCs/touch) in our future studies.

[4] http://www.tigris.org.
[5] http://freshmeat.net.
[6] http://objectweb.org.

Appendix A. List of projects and application domains

DEBIAN		GNOME		KDE		RF		SAV		SF	
acpidump	P	alacarte	C	ark	P	actsasfo-rmatted	O	a2ps	H	Aquila	G
apmud	P	anjuta	R	dolphin	P	classifier	O	acct	P	audiobo-okcutter	H
boson	F	astrolabe	P	fifteen-applet	F	cmdctrl	P	alive	P	Beobach-ter	P
cdpara-noia	H	atspi	O	kaddress-book	A	debug-print	P	autoconf	O	cdlite	H
cherokee	G	bakery	O	kamera	H	explain-pmt	A	avrdude	P	cotvnc	P
clamav	P	cheese	H	kate	R	family-connect	G	bubble-mon	P	cpia	G
dia	M	criawips	I	kback-gammon	F	forkma-nager	G	carbon-kernel	M	criticalcare	F
enigmail	A	damnedlies	P	kbattle-ship	F	geokit	L	cdump	P	csUnit	O
EtoileWi-ldMenus	O	daybook	G	kcron	P	gewee	F	cflow	O	eas3pkg	M
fig2ps	H	esound	H	kdebug-dialog	O	hatena-graphup	G	clustersim	M	edict	J
flac	H	evolutionjescs	A	kfeed	A	inifile	G	codeeditor	F	expreval	M
fte	O	evolutionsharp	A	kfileplugins	P	iowa	G	fluxus	P	fitnesse	O
geomview	M	garnome	P	kfind	P	jabber4r	A	freehoo	F	fnjavabot	A
gosa	G	gdm	P	kgamma	P	matlabruby	M	freepooma	H	formproc	G
grass6	M	geadow	R	khangman	F	mechanize	G	gcl	O	fourever	E
grub	P	gfloppy	P	khtml	P	mms2r	A	gfo	M	freemind	H
gwenview	H	gimphelp2	H	kioclient	G	morse	A	ghome-mover	P	galeon	C
jToolkit	R	glibjava	O	kjsembed	O	netnetrc	G	gnumed	A	genromfs	P
kdegames	F	gnomebuild	O	kjs	E	object-graph	O	gtktalog	P	gvision	O
kdenet-work	G	gnomefileselector	P	klink-status	G	pseudo-cursors	P	gvpe	H	hge	F
kmouth	H	gnomereset	P	kmag	H	qwik	A	hitweb	R	icsDrone	F
kphoneSI	A	gnometestspecs	P	kmailcvt	A	railshasflags	B	libmath-eval	M	interme-zzo	P
libax25	O	gnomewebwml	G	kmoon	F	randomdata	M	mcron	O	jtrac	O
liboil	O	gob	O	kmouse-tool	C	rapt	L	mp3tag	H	juel	O
libsoup	O	gopersist	C	kmouth	H	rateable-plugin	L	myspwi-zard	B	kpictorial	F
mimede-code	P	greg	P	knetwalk	F	roxml	O	oroborus	C	modaspdotnet	G
modauthkerb	F	gthumb	H	knetwork-conf	P	rparsec	O	osip	O	moses	A
myphp-money	M	gtkmmroot	O	knewsti-cker	A	rriki	B	phpcom-pta	G	nbcheck-style	O
noteedit	R	guikachu	O	kpat	F	rssfwd	A	phpgroup-ware	G	neocrypt	N
octaveforge	G	imperl	A	kppp	G	rtplan	I	ply	A	netstrain	A
openafs	G	libbonobojava	O	krfb	P	rubyamf	G	psg	P	ogce	G
Pike	O	libglass	O	kross-python	O	rubyexiv2	H	radius	G	oliver	G
prcs1	H	libgnomeui	O	ksim	P	rubyibm	B	radius-plugin	G	ozone	B
preludemanager	G	libpreview	I	ksquares	F	rubypytst	O	ratpoison	C	perpojo	A
ProofGene-ral	M	libwnck	C	kstart	C	ruport	B	sather	P	pf	P
rlplot	H	libxmlpp	O	kteatime	F	sahara	O	sdcdc	O	Qpolymer	M
ruby	O	narwhal	B	ktnef	A	s7eep	O	sins	F	seagull	O
scid	F	nautilusmozilla	G	kuiserver	P	simplesi-debar	C	stow	P	simple-soap	D
shorewall	P	nautilusrc	G	kxmlrpc-client	A	snmplib	A	texi2html	E	simplexml	O
skel	P	nautilussendto	P	kxsconfig	C	soks	G	texinfo	R	source	G
sylpheed	A	oaf	O	liloconfig	P	sstruct	J	tiger	P	swtjaspe-rviewer	K
syncekde	C	present	C	lskat	F	stdlibdoc	O	tong	F	toolchest	C
tcl	O	pygtk	O	marble	F	timcha-rper	G	twinlisp	O	txt2xml	O
tdb	P	pygtksou-rceview	O	nntp	A	trie	M	vihmauss	F	uniportio	P
tiobench	P	rhythmbox	H	qtruby	O	ttt2db	I	wasp	P	ustl	O
txt2html	E	SashCom-ponents	O	shell	P	utilrb	O	webpu-blish	G	whiteboard	D
vlc	H	sawfish	C	solid	O	verhoeff	M	xmakemol	M	winssh-askpass	Q
wxWidgets	O	silkywww	G	sonnet	O	voruby	M	xmod	O	wxactivex	G
xmakemol	M	viciousbuildscripts	O	strigianalyzer	P	watchcat	G	xsltxt	E	xmlnuke	G
yaml4r	M	vte	P	umbrello	O	xspf	O	yafsplash	C	xqilla	B

References

Antoniol, G., Casazza, G., Penta, M.D., Merlo, E., 2001. Modeling clones evolution through time series. In: Proceedings of the IEEE International Conference on Software Maintenance 2001 (ICSM 2001), Fiorence, Italy, pp. 273–280.

Basili, V.R., Caldiera, G., Rombach, D.H., 1994. The goal question metric approach. In: Encyclopedia of Software Engineering, John Wiley & Sons, pp. 528–532, see also <http://sdqweb.ipd.uka.de/wiki/GQM>.

Beecher, K., Boldyreff, C., Capiluppi, A., Rank, S. Evolutionary success of open source software: An investigation into exogenous drivers, Electronic Communications of the EASST 8.

Boehm, B.W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., Steece, B., 2000. Software Cost Estimation with Cocomo II with Cdrom. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Cabin, R.J., Mitchell, R.J., 2000. To bonferroni or not to bonferroni: when and how are the questions. Bulletin of the Ecological Society of America, 246–248.

Canfora, G., Cerulo, L., Penta, M.D., 2007. Identifying changed source code lines from version repositories. Mining Software Repositories 0, 14.

Capiluppi, A., 2003. Models for the evolution of OS projects. In: Proceedings of ICSM 2003 2003 IEEE, Amsterdam, The Netherlands.

Capiluppi, A., Michlmayr, M., 2007. From the cathedral to the bazaar: an empirical study of the lifecycle of volunteer community projects. In: Feller, J., Fitzgerald, B., Scacchi, W., Silitti, A. (Eds.), Open Source Development, Adoption and Innovation, International Federation for Information Processing. Springer, pp. 31–44.

Capiluppi, A., Morisio, M., Ramil, J.F., 2004. Structural evolution of an open source system: a case study. IWPC, 172–182.

Crowston, K., Annabi, H., Howison, J., 2003. Defining open source software project success. In: Proceedings of ICIS 2003, Seattle, Washington, USA.

Crowston, K., Howison, J., Annabi, H., 2006. Information systems success in free and open source software development: theory and measures. Software Process Improvement and Practice, 123–148.

Dalgaard, P., 2002. Introductory Statistics with R. Springer.

English, R., Schweik, C., 2007. Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects. In: Proceedings of the 1st International Workshop on Emerging Trends in FLOSS Research and Development, ICSE, Minneapolis, MN.

Feller, J., Fitzgerald, B., Hecker, F., Hissam, S., Lakhani, K., van der Hoek, A. (Eds.), . Characterizing the OSS Process. ACM.

German, D.M., 2004. The gnome project: a case study of open source, global software development, software process: improvement and Practice 8 (4) 201–215 <URL http://dx.doi.org/10.1002/spip.189>.

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

Please cite this article in press as: Beecher, K. et al., Identifying exogenous drivers and evolutionary stages in FLOSS projects, J. Syst. Software (2008), doi:10.1016/j.jss.2008.10.026

Godfrey, M.W., Tu, Q., 2000. Eovlution in open source software: a case study. In: Proceedings of 16th IEEE International Conference on Software Maintenance.

Hamer, P.G., Frewin, G.D., 1982. M.h. halstead's software science – a critical examination. In: ICSE'82: Proceedings of the Sixth International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 197–206.

Hemetsberger, A., Reinhardt, C., 2004. Sharing and creating knowledge in open-source communities: the case of kde. In: Proceedings of the Fifth European Conference on Organizational Knowledge, Learning and Capabilities (OKLC), Insbruck University.

Herraiz, I., Gonzlez-Barahona, J.M., Robles, G., 2008. Determinism and evolution. In: Hassan, A.E., Lanza, M., Godfrey, M.W. (Eds.), Mining Software Repositories. ACM, pp. 1–10.

Herraiz, I., Robles, G., González-Barahona, J.M., 2006. Comparison between slocs and number of files as size metrics for software evolution analysis. CSMR, 206–213.

Hindle, A., German, D.M., 2005. Scql: a formal model and a query language for source control repositories. SIGSOFT Software Engineering Notes 30 (4), 1–5.

Koch, S., Schneider, G., 2002. Effort cooperation and coordination in an open source software project: Gnome. Information Systems Journal 12 (1), 27–42.

Kuniavsky, M., Raghavan, S., 2005. Guidelines are a tool: building a design knowledge management system for programmers. In: DUX'05: Proceedings of the 2005 Conference on Designing for User Experience. AIGA: American Institute of Graphic Arts New York, NY, USA.

Laat, Paul, 2007. Governance of open source software: state of the art, Journal of Management & Governance 11 (2), 165–177 URL <http://dx.doi.org/10.1007/s10997-007-9022-9>.

Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M., 1997. Metrics and laws of software evolution – The nineties view. In: El Eman, K., Madhavji, N.H. (Eds.), Elements of Software Process Assessment and Improvement. IEEE CS Press, Albuquerque, New Mexico, pp. 20–32.

Livieri, S., Higo, Y., Matushita, M., Inoue, K., 2007. Very-large scale code clone analysis and visualization of open source programs using distributed ccfinder: D-ccfinder. In: ICSE'07: Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA.

Michlmayr, M., 2007. Quality improvement in volunteer free and open source software projects: exploring the impact of release management. Ph.D. Thesis, University of Cambridge, Cambridge, UK. URL <http://www.cyrius.com/publications/michlmayr-phd.pdf>.

Michlmayr, M., Senyard, A., 2006. A statistical analysis of defects in Debian and strategies for improving quality in free software projects. In: Bitzer, J., Schrder, P.J.H. (Eds.), The Economics of Open Source Software Development. Elsevier, Amsterdam, The Netherlands, pp. 131–148.

Mockus, A., Fielding, R.T., Herbsleb, J., 2002. Two case studies of open source software development: apache and mozilla. ACM Transactions on Software Engineering and Methodology 11 (3), 309–346.

Perneger, T.V., 1998. What's wrong with Bonferroni adjustments, vol. 316, <URL http://bmj.bmjjournals.com/cgi/content/full/316/7139/1236>.

Rainer, A., Gale, S., 2005. Evaluating the quality and quantity of data on open source software projects. In: Feller, J., Fitzgerald, B., Scacchi, W., Silitti, A. (Eds.), First International Conference on Open Source Systems.

Robles, G., Dueñas, S., González-Barahona, J.M., 2007. Corporate involvement of libre software: study of presence in debian code over time. In: Feller, J., Fitzgerald, B., Scacchi, W., Sillitti, A. (Eds.), OSS of IFIP, vol. 234. Springer, pp. 121–132.

Robles, G., Gonzlez-Barahona, J.M., Centeno-Gonzalez, J., Matellan-Olivera, V., Rodero-Merino, L., 2003. Studying the evolution of libre software projects using publicly available data. In: Proceedings of the Third Workshop on Open Source Software Engineering, pp. 111–115.

Robles, G., Koch, S., González-Barahona, J.M., 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In: Proceedings of the Second ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS'04), 26th International Conference on Software Engineering, Edinburgh, UK.

Shen, V.Y., Conte, S.D., Dunsmore, H.E., 1983. Software science revisited: a critical analysis of the theory and its empirical support. IEEE Transactions of Software Engineering 9 (2), 155–165.

Stamelos, I., Angelis, L., Oikonomou, A., Bleris, G.L., 2002. Code quality analysis in open-source software development. Information Systems Journal 12 (1), 43–60.

Stewart, K.J., Ammeter, T., 2002. An exploratory study of factors influencing the level of vitality and popularity of open source projects. In: ICIS 2002, Proceedings of International Conference on Information Systems 2002.

Tukey, J.W., 1977. Exploratory Data Analysis, Addison-Wesley Series in Behavioral Science: Quantitative Methods, Reading, Mass. Addison-Wesley.

Weiss, D., 2005. Measuring success of open source projects using web search engines. In: Scotto, M., Succi, G. (Eds.), Proceedings of The First International Conference on Open Source Systems (OSS 2005), Genova, Italy, pp. 93–99.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics Bulletin 1 (6), 80–83.