Evolution of Open Source Software Systems – A Large-Scale Investigation

Stefan Koch
Department of Information Business, University of Economics and BA
Vienna, Austria
stefan.koch@wu-wien.ac.at

Abstract – **In this paper, the evolution of a large sample of open source software projects will be analysed. The evolution of commercial systems has been an issue that has long been a center of research, thus a coherent theoretical framework of software evolution has been developed and empirically tested. Therefore these results can be used to compare the situation in open source projects to the evolution of commercial projects. This allows to assess whether the underlying software process indeed significantly differs. The data collection methodology relying on a large software repository and the respective source code control systems is described, and an overview on the collected data on several thousand projects is given. The evolutionary behaviour is explored using both a linear and a quadratic model, with the quadratic model significantly outperforming the linear one. The most interesting fact is that while in the mean the growth rate is decreasing over time according to the laws of software evolution, especially larger projects with a higher number of participants might be more often able to sustain super-linear growth.**

I. INTRODUCTION

In the last years, free and open source software has gathered increasing interest, both from the business and academic world. As some projects in different application domains like Linux together with the suite of GNU utilities, GNOME, KDE, Apache, sendmail, bind, and several programming languages have achieved huge success in their respective markets, new business models have been developed and tested by businesses both small and large like Netscape or IBM. Academic interest into this new form of collaborative software development has arisen from very different backgrounds including software engineering, sociology, management or psychology, and has gained increasing prominence.

While many of the successful projects mentioned above are well-known and produce output of high quality, a general assessment of the open source software development paradigm is yet outstanding. Currently, any discussion of this new model is mostly based on a small number of glamorous and successful projects, but these might constitute exceptions. Therefore an analysis needs to be based on quantitative information on the enactment in a variety of project forms and sizes. The main ideas of this development model are described in the seminal work of Raymond, 'The Cathedral and the Bazaar', in which he contrasts the traditional type of software development of a few people planning a cathedral in splendid isolation with the new collaborative bazaar form of open source software development [1]. In this, a large number of developer-turned users come together without monetary compensation to cooperate under a model of rigorous peer-review and take advantage of parallel debugging that leads to innovation and rapid advancement in developing and evolving software products. In order to allow for this to happen and to minimise duplicated work, the source code of the software needs to be accessible, and new versions need to be released often. To this end, software licences that grant the necessary rights to the users, like free redistribution, inclusion of the source code, the possibility for modifications and derived works and some others have been developed. One model for such licences is the Open Source Definition, which lists a number of requirements for specific licences [2]. The most prominent example which fulfils these criteria while still being even more stringent, is the GNU General Public Licence (GPL), developed by the GNU project and advocated by the Free Software Foundation [3].

The advantages and disadvantages of this new development model have been hotly debated [4,5], but mostly on a general level without empirical backing. While some detailed research has been undertaken to uncover issues like the organisation of work in a small number of open source projects [6,7,8] and some aspects have been analysed using larger samples [9,10,11,12,13], several facets of the development process, including the expended effort and its estimation or the evolution of open source systems remain still largely unexplored on a large scale.

In this paper, the evolution of a large sample of open source software projects will be analysed. The evolution of commercial systems has been an issue that has long been a center of research, thus a coherent theoretical framework has been developed and empirically tested. Therefore these results can be used to compare the situation in open source projects to the evolution of commercial projects. This allows to assess whether the underlying software process indeed significantly differs.

The study of software evolution for commercial systems was pioneered by the work of Lehman and Belady on the releases of the IBM OS/360 operating system [14], which has led to many other works (e.g. [15]), in which the laws of software evolution were formulated, expanded and revised. These laws entail a continual need for adaptation of a system. This continual adaptation leads to increased complexity of the system. As it is assumed that constant incremental effort is applied at each time step, average incremental growth naturally declines. Turski has modeled this as an inverse square growth rate [16]. To this date, there is a single case study on the growth behaviour of an open source system: Godfrey and Tu have analysed the most prominent example available, the Linux operating system kernel [17]. Using the size in lines-of-code as a function of the time in days since the first commit, which is used as a proxy for project start date, with one month as time window, they found that the growth behaviour is best fitted by a super-linear rate. This significantly contradicts the prior theory of software evolution, which postulates a decline in growth. This would give an indication of major differences in development modes and their results. On the other hand, Paulson et al. have used a linear approximation, and have not found any differences in growth behaviour between open and closed-source software projects [18]. In this paper we will try to analyse this situation using a larger sample. The next sections will describe the data collection methodology applied, and will give an overview on the collected data. Afterwards, the evolutionary behaviour will be explored.

II. METHODOLOGY

For performing the proposed analysis of the evolutionary behaviour of open source software projects, the information contained in software development repositories will be explored. These repositories contain a plethora of information on the underlying software and the associated development processes [19,20]. Studying software systems and development processes using these sources of data offers several advantages [19]: This approach is very cost-effective, as no additional instrumentation is necessary, and it does not influence the software process under consideration. In addition, longitudinal data is available, allowing for analyses considering the whole project history.

Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems, bug reporting systems, or mailing lists. Many of these have already been used as information sources for closed source software development projects. For example, Cook et al. present a case study to illustrate their proposed methodology of analyzing in-place software processes [19]. They describe an update process for large telecommunications software, analysing several instances of this process using event data from customer request database, source code control, modification request tracking database and inspection information database. Atkins et al. use data from a version control system in order to quantify the impact of a software tool, a version-sensitive editor, on developer effort [20].

In open source software development projects, repositories in several forms are also in use, in fact form the most important communication and coordination channels, as the participants in any project are not collocated. Therefore only a small amount of information can not be captured by repository analyses because it is transmitted inter-personally. As a side effect, the repositories in use must be available openly and publicly, in order to enable as many persons as possible to access them and to participate in the project. Therefore open source software development repositories form an optimal data source for studying the associated type of software development.

Given this situation, repository data have already been used in research on open source software development. This includes in-depth analyses of small numbers of successful projects like Apache and Mozilla [6,7], GNOME [8] using mostly information provided by version-control-systems, but sometimes in combination with other repository data like from mailing list archives. Large-scale quantitative investigations spanning several projects going into software development issues are not yet as common, and have mostly been limited to using aggregated data provided by software project repositories [9,10,11]), meta-information included in Linux Software Map entries [12], or data retrieved directly from the source code itself [13].

In this paper, we will follow this approach of using publicly available data from software repositories to study the evolutionary behaviour of open source projects and the underlying software process.

III. SOURCEFORGE.NET: A PROJECT ECOLOGY REPOSITORY

For this analysis, a large data set covering a diverse population of large, small, successful und failed projects was needed. SourceForge.net, the software development and hosting site, was chosen as the source of data. The mission of SourceForge.net is ‘to enrich the open source community by providing a centralized place for open source developers to control and manage open source software development’. To fulfil this mission goal, a variety of services is offered to hosted projects, including tools for managing support, mailing lists and discussion forums, web server space, shell services and compile farm, and source code control. While SourceForge.net publishes several statistics, e.g. on activity in their hosted projects, this information was not detailed enough for the proposed analysis. For example, Crowston and Scozzi used the available data for validating a theory for competency rallying, which suggests factors important for the success of a project [9]. Hunt and Johnson have analysed the number of downloads of projects occurring [10], and Krishnamurthy used the available data of the 100 most active mature projects for an analysis [11].

The data collection method utilized is described in detail in [21]. It is based on automatically extracting data from the web pages and especially the source code control system, in the form of CVS [22], and subsequently parsing the results and storing the relevant results, e.g. size, date and programmer of each checkin, in a database. Most of this work was done using Perl scripts. All following analyses are based on the 8,621 projects for which all relevant information could be retrieved. More detailled analyses of these data can be found in [23], in the following a short overview is provided.

Within these projects, a total of 7,734,082 commits have been made, with 663,801,121 LOCs having been added and 87,405,383 having been deleted. The projects consist of 2,474,175 single files, and an overall number of 12,395 distinct programmers have contributed with at least one commit. The distribution of both the assets available, i.e. the programmers, and the resulting outcome, i.e. commits, lines-of-code and project status within the project ecology is very skewed. Table I gives descriptive statistics for the associated variables.

TABLE I.
DESCRIPTIVE STATISTICS FOR PROJECT VARIABLES FROM SOURCEFORGE.NET DATA SET (N=8,621)

	Min.	**Max.**	**Mean**	**Std. Dev.**	**Median**
Number of programmers	1	88	1.86	2.61	1
Commits	1	133,759	897.12	3,840.90	192
LOC added	0	12,951K	77K	459K	10,801
LOC deleted	0	3,847K	10K	73K	373
Files	1	42,674	285.46	1,317.74	69
Development Status	0	6	2.67	1.77	3

As can be seen, the vast majority of projects have only a very small number of programmers (67.5 per cent have only 1 programmer), only 1.3 per cent have more than 10 programmers. This number of programmers can be shown to follow a power law (or Pareto or Zipf) distribution [23], like Hunt and Johnson have also found for the number of downloads of projects [10]. These numbers also correspond to the findings of Krishnamurthy [11], who showed that most of the projects had only a small number of participants (median of 4). Only 19 per cent had more than 10, 22 per cent only 1 developer. While this percentage is much smaller than found here, this is not surprising as Krishnamurthy only used the 100 most active projects.

Regarding the output of the projects, a similar situation can be seen, the vast majority of projects achieves only a small number of commits and is of small size, leading to the assumption that input and output of projects a correlated, i.e. that projects with a small number of programmers only achieve small numbers of commits and lines-of-code. This intuitive relationship can be ascertained. For example, the total number of programmers of a project correlates positively at significance 1 percent with coefficients 0.472 with number of commits and 0.408 with total lines-of-code added [23].

Regarding the situation within projects, most prior studies as cited [6,7,8,13,24] have found a distinctly skewed distribution of effort between the participants (see Table II for variables of programmer participation in SourceForge.net). Similar results can also be found at the project ecology under consideration. The top decile is responsible for 79 per cent of the total SourceForge.net code base, the second decile for additional 11 per cent. Defining a simple measure for the inequality of work distribution within the development team [25], the effects of an increasingly inequal distribution of work on other project variables shows rather small, but positive correlations with total number of commits and sum of lines-of-code added. There is no correlation with the age of the project, so the inequality does not increase simply with time. Of course, the direction of the relationship is not ascertained, so the results do not necessarily indicate that more activity in projects is caused by a more unequal distribution of contributions, as the other way would also give a possible explanation, i.e. as the project grows, the inequality grows as a result.

TABLE II.
DESCRIPTIVE STATISTICS FOR PROGRAMMER VARIABLES FROM SOURCEFORGE.NET DATA SET (N=12,395)

	Min.	**Max.**	**Mean**	**Std. Dev.**	**Median**
Commits	1	132,736	624	2,815	112
LOC added	0	16,152,866	53,554	374,750	5,469
LOC deleted	0	3,846,863	7,052	54,547	372
Files	1	44,271	230	1,105	48
Projects	1	15	1.29	0.81	1

The next possible influence on productivity in a project is the number of active programmers, following the reasoning of Brooks's Law [26], "Adding manpower to a late project makes it later". Therefore, the number of active programmers and the achieved progress in each project was analysed on a monthly basis. Although the number of active programmers has a significant and positive relationship with the overall output, the coefficient is much smaller than previously found by Koch and Schneider in their analysis of the GNOME project [8]. Brooks's Law itself is next to non-existent, as the correlation between the number of active programmers and the mean output per programmer in a period is (although negative) only -0.013 with both commits and lines-of-code used as output measures (significant at 5 per cent).

IV. EVOLUTION OF OPEN SOURCE SOFTWARE

Using the data retrieved for the projects from the SourceForge.net project ecology as detailled above, the evolutionary behaviour of these projects is explored. To this end, both a linear and a quadratic model were computed for each project, taking the size in lines-of-code *S* as a function of the time in days since the first commit *t*, which is used as project start date, and using one month as time window. This approach is taken in accordance with Godfrey and Tu [17]. Therefore model A was formulated simply as

$$S_A(t) = a * t + b \tag{1}$$

and model B as

$$S_B(t) = a * t^2 + t * b + c. \tag{2}$$

The necessary parameters were estimated using regression techniques. This was possible for 4,047 of the projects, as the other lacked necessary information (i.e. had too few data points). In order to compare the resulting fit of these models, the adjusted R-squared measure is used which accounts for the number of parameters. The quadratic model B outperformed the linear one with a mean adjusted R-squared of 0.831 (median 0.918) against 0.592 (median 0.727). This difference was tested for significance, using a Wilcoxon signed rank test instead of a paired-samples t-test because the difference scores are not normal distributed. The hypothesis that the distributions are equal is indeed rejected (at 1 per cent significance).

These first results are not surprising and are still in line with the laws of software evolution. As a next step, it is therefore necessary to explore whether or not the growth rate is decreasing over time according to these laws . This can be checked by analysing the second derivate of the quadratic model $S_B(t)'$, or more conveniently directly the coefficient of the quadratic term *a*. As a first step, it is confirmed that this parameter indeed is different from zero, using a t-test at 1 percent significance. Then, the distribution of this term is explored. In the mean, it has a slightly negative value of -0.504 with a median of -0.020. This would indeed indicate a decreasing growth rate in accordance with the laws of software evolution, but contrary to the findings of Godfrey and Tu for Linux [17]. In fact, for 61 percent of the projects this term is negative, for the remaining minority of 1,578 positive. This shows that a rather large number of projects exhibits super-linear growth. To check the validity of these results, only projects having achieved a given maturity (status productive or mature) have been included in a separate analysis, which leaves about a quarter (1,087 projects) of the data, but the results stay the same.

As a nex step, it is explored whether there are any characteristics of projects that lead to this super-linear growth behaviour. Therefore, the projects are divided into two groups according to their growth behaviour, i.e. whether it is super-linear or not. Using (Spearman) correlations and Mann-Whitney U-tests a small but significant (at level 0.01) relationship with size (in both lines-of-code and commits) and number of programmers can be found, indicating that larger projects with a higher number of participants might be more often able to sustain super-linear growth (see also Fig. I). This would be in accordance to the findings of Godfrey and Tu [17], as Linux is a relatively large project, but would contradict the assumption of software evolution that increased size leads to more complexity and interdependencies, thus decreasing growth rate.

As a further explanatory variable, the inequality of work distribution within the team was explored. Again, a small but significant positive relationship (significance level 0.05) was found (see also Fig. II).

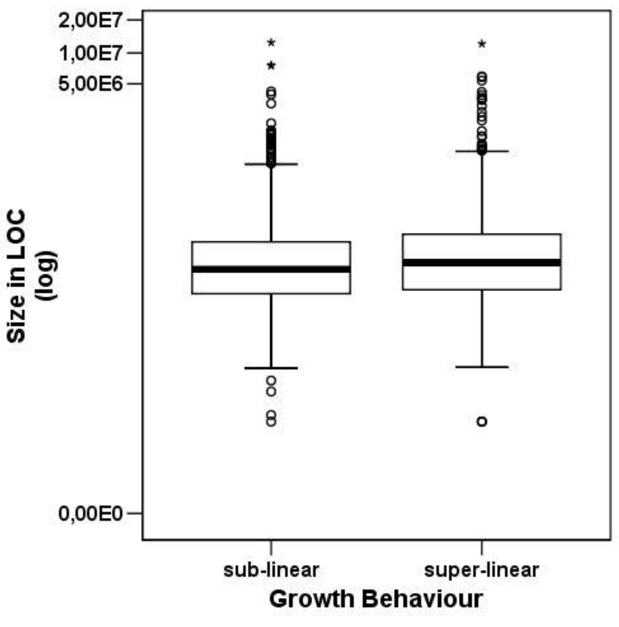

FIGURE I.
BOXPLOT OF PROJECT SIZE DISTRIBUTION DEPENDING ON GROWTH BEHAVIOUR

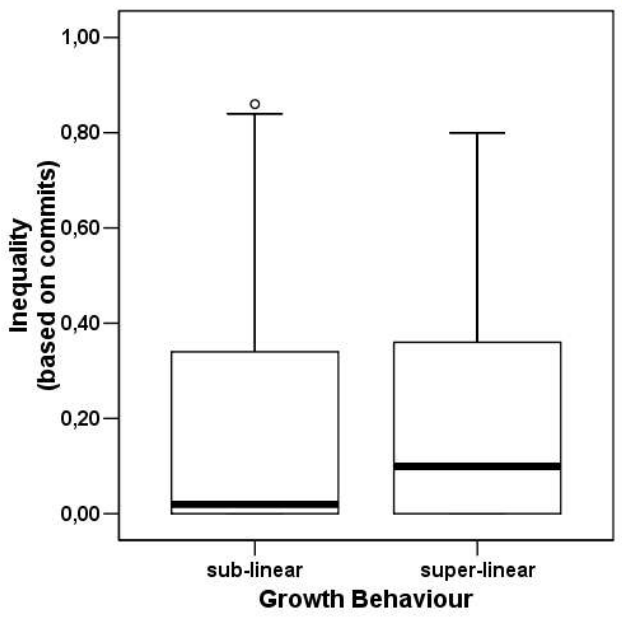

FIGURE II.
BOXPLOT OF PROJECT INEQUALITY DISTRIBUTION DEPENDING ON GROWTH BEHAVIOUR

This indicates that in the projects with super-linear growth rate, the distribution is in general more inequal. At first glance, this seems to contradict the prior result of these projects having more participants, but indeed seems to point at a certain development model. A project with super-linear growth has a higher number of participants, but this does not lead to a more equal distribution of output within this group. Therefore, a few people within this larger group seem to do most of the work, while the others assist them in large numbers. This organisation has been proposed decades ago by Harlan D. Mills [27] as 'chief programmer team organisation' [28], also termed 'surgical team' by Brooks [26], in which system development is divided into tasks each handled by a chief programmer who is responsible for the most part of the actual design and coding, supported by a larger number of other specialists like a documentation writer or a tester. This finding will need to be further explored, as this form of organisation seems to be able to overcome the problems associated with increased complexity during software evolution.

V. CONCLUSION

In this paper, the evolution of a large sample of open source software projects has been analysed. The evolution of commercial systems has been an issue that has long been a center of research, and therefore a coherent theoretical framework of software evolution has been developed and empirically tested. The data collection methodology relying on a large software repository and the respective source code control systems has been described, and an overview on the collected data on several thousand projects was given. The evolutionary behaviour is found to be significantly better modelled using a quadratic model in comparison to a linear one. The most interesting fact is that while in the mean the growth rate is decreasing over time according to the laws of software evolution, especially larger projects with a higher number of participants might be more often able to sustain super-linear growth. In addition, this group of projects seem to sport a higher inequality in the distribution of work within the development team. This might hint at a certain organisational model, the chief programmer team. This form, present in open source software development, through measures like strict modularization and self-selection for tasks seems to be able to at least delay the negative effects arising during evolution. In addition, especially the fourth law of software evolution, 'conservation of organisational stability' [15], implying constant incremental effort, might be violated especially in large projects which attract an ever increasing number of participants.

VI. REFERENCES

[1] E.S. Raymond, *The Cathedral and the Bazaar,* Cambridge, Massachusetts: O'Reilly & Associates, 1999.

[2] B. Perens, "The Open Source Definition", in DiBona, C. et al. (eds.), *Open Sources: Voices from the Open Source Revolution,* Cambridge, Massachusetts: O'Reilly & Associates, 1999

[3] R.M. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman,* Boston, Massachusetts: GNU Press, 2002.

[4] P. Vixie, "Software Engineering", in DiBona, C. et al. (eds.), *Open Sources: Voices from the Open Source Revolution,* Cambridge, Massachusetts: O'Reilly & Associates, 1999.

[5] S. McConnell, "Open-source methodology: Ready for prime time?", *IEEE Software,* vol.16, no. 4, 1999, pp. 6-8.

[6] A. Mockus, R. Fielding and J: Herbsleb, "A Case Study of Open Source Software Development: The Apache Server", in *Proceedings of the 22nd International Conference on Software Engineering,* Limerick, Ireland, 2000, pp. 263-272.

[7] A. Mockus, R. Fielding and J: Herbsleb, "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology,* vol. 11, no. 3, 2002, pp. 309-346.

[8] S. Koch and G. Schneider, "Effort, Cooperation and Coordination in an Open Source Software Project: Gnome", *Information Systems Journal,* vol. 12, no. 1, 2002. pp. 27-42.

[9] K. Crowston and B. Scozzi, "Open source software projects as virtual organizations: Competency rallying for software development", *IEE Proceedings - Software Engineering,* vol. 149, no. 1, 2002, pp. 3-17.

[10] F. Hunt and P. Johnson, "On the pareto distribution of sourceforge projects", in *Proceedings of the Open Source Software Development Workshop,* Newcastle, UK, 2002, pp. 122-129.

[11] S. Krishnamurthy, "Cave or community? an empirical investigation of 100 mature open source projects", *First Monday,* vol. 7, no. 6, 2002.

[12] B.J. Dempsey, D. Weiss, P. Jones and J. Greenberg, "Who is an open source software developer?", *Communications of the ACM,* vol. 45, no. 2, 2002, pp. 67-72.

[13] R. Ghosh and V.V. Prakash, "The Orbiten Free Software Survey", *First Monday,* vol.. 5, no. 7, 2000.

[14] L.A. Belady and M.M. Lehman, "A model of large program development", *IBM Systems Journal,* vol. 15, no. 3, 1976, pp. 225-252.

[15] M.M. Lehman and J.F. Ramil, "Rules and Tools for Software Evolution Planning and Management", *Annals of Software Engineering,* vol. 11, 2001, pp. 15-44.

[16] W.M. Turski, "Reference Model for Smooth Growth of Software Systems", *IEEE Transactions on Software Engineering,* vol. 22, no. 8, 1996, pp. 599-600.

[17] M.W. Godfrey and Q. Tu, "Evolution in Open Source software: A case study", in *Proceedings of the International Conference on Software Maintenance (ICSM 2000),* San Jose, California, 2000, pp. 131-142.

[18] J.W. Paulson, G. Succi, and A. Eberlein, "'An empirical study of open-source and closed-source software products", *IEEE Transactions on Software Engineering,* vol. 30, no.4, 2004, pp. 246-256.

[19] J.E. Cook, L.G. Votta and A.L. Wolf, "Cost-effective analysis of in-place software processes", *IEEE Transactions on Software Engineering,* vol. 24, no. 8, 1998, pp. 650-663.

[20] S. Atkins, T. Ball, T. Graves and A. Mockus, "Using Version Control Data to Evaluate the Impact of

Software Tools", in *Proceedings of the 21st International Conference on Software Engineering,* Los Angeles, CA, 1999, pp. 324-333.

[21] M. Hahsler and S. Koch, "Discussion of a Large-Scale Open Source Data Collection Methodology", in *Proceedings of the Hawaii International Conference on System Sciences (HICSS-38),* Big Island, Hawaii, 2005.

[22] K. Fogel, *Open Source Development with CVS,* Scottsdale, Arizona: CoriolisOpen Press, 1999.

[23] S. Koch, "Profiling an Open Source Project Ecology and Its Programmers", *Electronic Markets,* vol. 10, no. 2, 2004, pp. 77-88.

[24] G. Hertel, S. Niedner and S. Hermann, "Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel", *Research Policy,* vol. 32, no. 7, 2003, pp. 1159-1177.

[25] G. Robles, S. Koch and J.M. Gonzalez-Barahona, "'Remote analysis and measurement of libre software systems by means of the CVSanalY tool", in *ICSE 2004 - Proceedings of the Second International Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04),* Edinburgh, Scotland, 2004, pp. 51-55.

[26] F.P. Brooks jr., *The Mythical Man-Month: Essays on Software Engineering,* Anniversary ed., Reading, Massachusetts: Addison-Wesley, 1995.

[27] H.D. Mills, "Chief Programmer Teams: Principles and Procedures", *Report FSC 71-5108,* IBM Federal Systems Division, Gaithersburg, Maryland, 1971.

[28] T.F. Baker, "Chief Programmer Team Management of Production Programming", *IBM Systems Journal,* vol. 11, no. 1, 1972, pp. 56-73.