Collaborative Maintenance in Large Open-Source Projects

Matthijs den Besten[1,2], Jean-Michel Dalle[1,2] and Fabrice Galia[3]
[1] Université Pierre et Marie Curie, Paris, France
<jean-michel.dalle@upmc.fr>
[2] Université Paris-Dauphine, Paris, France
<matthijs.denbesten@lamsade.dauphine.fr>
[3] Université Panthéon-Assas Paris II, Paris, France
<galia@u-paris2.fr>

Abstract. The paper investigates collaborative work among maintainers of open source software by analyzing the logs of a set of 10 large projects. We inquire whether teamwork can be influenced by several characteristics of code. Preliminary results suggest that collaboration among maintainers in most large open-source projects seems to be positively influenced by file vintage and by Halstead volume of files, and negatively by McCabe complexity and size measured in SLOCs. These results could be consistent with an increased attractivity of files created early in the history of a project, and with maintainers being less attracted by more verbose code and by more complex code, although in this last case it might also reflect the fact that more complex files would be de facto more exclusive in terms of maintenance.

1 Introduction

Teams in general, and virtual teams in particular, enjoy an increasing interest from scholars in organizational science.[1,2] In the absence of a strong managerial hand, it is not obvious indeed how team members collaborate – especially when the members are located in various parts of the world. Nonetheless, in many circumstances virtual teams appear to be remarkably successful and until now, no real and clear understanding exists of the conditions of their success and efficiency.

In this context, the work of virtual teams is at least partly traceable in the activity logs that those teams leave behind in their virtual environments. Open source software projects are natural candidates in this respect, i.e. for quantitative empirical studies of virtual teams, given their increasing economic success and the free and easy access they typically provide to such data.[4] Several steps in this direction have already been made by others[3]. This conviction that the by-products of collaboration provide a wealth of data that could be harnessed is also behind the study of collaborative maintenance activity in open source project logs that we present here..

Section 2 introduces open source software and reviews some of the research done in that area. In section 3, we describe the database we studied and how we created it, and we introduce a few important methodological caveats. It is followed, in section 4,

Please use the following format when citing this chapter:

den Besten, M., Dalle, J.-M., and Galia, F., 2006, in IFIP International Federation for Information Processing, Volume 203, Open Source Systems, eds. Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G., (Boston: Springer), pp. 233-244

by an analysis of the results of our investigations. We conclude by briefly pointing out several avenues for further research.

2 Open Source Software

Open source software (OSS) is a type of software that has become increasingly prevalent over recent years. In contrast to closed source software, in OSS the human readable source code of the software program is distributed along with the program itself. With this source code it becomes then possible for users of the program to scrutinize the inner workings of the program and to adapt the program to their needs. The most famous example of OSS is Linux, an operating system developed based on Unix that is developed by Linus Torvalds and many other developers.[5] Microsoft, a dominant player in the market for operating systems, acknowledged the strength of Linux very early on, in what is now known as the "Halloween document"[6], and since then, the software industry has looked for ways to adapt features of the open source development model in more traditional closed environments.[7,8]

Yet, there is still something particular, and largely puzzling, about the OSS development model. In general, what is understood as the OSS development model is that it corresponds to the community-based voluntary self-organizing effort of various virtual teams of physically dispersed computer programmers to develop software – that is itself open to inspection to everyone who is interested. Eric Raymond famously likened the OSS development model to the interactions that are going on in a "bazaar".[9] However, since then, several case studies of open source software projects showed that in many projects hierarchies tend to persist and that there is larger diversity in organizational forms from one project to the other than would have been expected.[10] Indeed, in so far as there is a OSS development model, recent research seems to point towards an "onion model" of organization in which a core team of just a few developers is aided by a larger group of co-developers who are in turn aided by an even larger group of bug-submitters and feature-requesters, etc.[11] That is, open source development typically involves the participation of a large number of users who report bugs and request features, to be compared to a more limited number of co-developers who suggest software code that addresses those bugs and features; and to yet a smaller set of core developers who review the suggested code contributions and incorporate them in the existing code base.

What makes open source software projects particularly attractive as a topic for research is that virtually the whole development process is recorded and that the archives of these recordings are freely available for investigation. More in particular, open source software projects typically feature mailing lists where developers discuss their work and non-developers submit requests or ask for help. In addition, there may be discussion forums and bug tracking tools. Last, but not least, the source code is available and, when, as is often the case, a version control system is employed, in fact all old versions of the source code so that the development process can be traced back to the start. Researchers of software engineering have started to make use of this wealth of data to inform their investigations. Notable examples are the work of Walt

Scacchi[12], who performed an in-depth ethnographical analysis of the implicit ways in which requirements are gathered in open source projects, and that of Mockus and Herbsleb[13], who studied the pace with which bugs were resolved based on information in mailing lists and software logs. Hashler and Koch[14] propose a larger scale mining of the available information and discuss what kind of questions could be explored on the basis of that information.

The data that we looked at for our particular investigation of the allocation of tasks in open source software project teams was extracted from logs of development activity that are maintained by software version control systems. Version control systems are used by development teams in order to keep track of what was contributed, when and by whom. If conflicts arise due to a change in the code, a version control system makes it possible to undo that change and revert to the source code as it was before the change was made. Note, however, that in most OSS projects, a possible change has already been thoroughly reviewed before it is applied to the source code. Also, the people who commit the change are not necessarily the ones who wrote the code incorporated in that change. Rather, they are likely to be the *maintainers* of a part of the source code, who after a review of a change suggested by others, decide it is a good change and apply it to their part of the source code. In some cases, each change has to be approved of by a committee of core developers. In other cases, the review of suggested changes is completely up to the digression of the maintainer of the part of the source code to which the change is applied.

3 Database & Caveats

To create a database adapted to our investigations, we selected a set of open-source projects, attempting to obtain a set that was diverse in terms of product complexity, task uncertainty, and target audience. In addition, the projects needed to have a minimum amount of code, contributors and development history: in the list below, the logs typically span a period of five to ten years. Obviously, only those projects that provided easy access to their code repositories could qualify. In the end we settled for ten projects: An operating system – *NetBSD*, a data base – *PostgreSQL*, a web server – *Apache*, a web browser – *Mozilla*, an instant messaging application – *Gaim*, a secure networking protocol – *OpenSSH*, a programming language – *Python*, a compiler – *GCC*, an interpreter for the PostScript language and for PDF – *Ghostcript*, and a version control system – *CVS*. Several of these projects, most notably Mozilla and Apache, have already received a lot of attention from researchers. Others, like Gaim, stand out because of the amount of activity or because of the sheer length of activity. Finally, and although we only selected "large" projects, we selected projects whose sizes belong to different orders of magnitude (in terms #contributors, #files, #years of history), which could have an impact on their characteristics, and we would precisely like to discriminate between characteristics of projects and features more generally associated with the open-source mode of software development. There are also strong and potentially relevant differences among these projects in terms of organization and in terms of maintenance policies.

We extracted CVS logs for all these projects. CVS is the most widely used version control system for open source software development and its logs are relatively easy to parse.[15] The log lists for each file each revision of that file and for each revision when the revision was made, who was responsible for the revision and how many lines of code were added to and deleted from the file as a result of the revision (example given in Annex). At this level of analysis, we have restrained our sample to all the files that contain source code written in C or C++ i.e. to files with .c, .C, .cc, or .cpp suffixes. However, in some projects, e.g. *Python*, most code is obviously written in another language (e.g. *python*, precisely). In others, specially in *gcc*, there is a large portion of test files.

For each of the 10 projects, we computed descriptive data similar to what is available for various open-source projects[18], reported partially in Table 1. Then, more specifically for the purpose of studying collaborative maintenance, for each file that was studied and for each month we computed how many distinct maintainers had committed a change to that file during that period, and how many commits the file had received during the same period.

Before we proceed to presenting our investigations and their results, a few caveats have to be mentioned, which appeared as we progressed in the series of experiments that we conducted with our database.

1. About the constitution of the database and its suitability for econometric inquiries, it is not fully clear where the boundaries of a given project are. For instance, Apache and Mozilla have their own repositories but both host multiple applications. Lacking a clear rule for now about where to draw these limits, we decided that in the case of Apache, we would restrict ourselves to the logs concerning Apache HTTP Server 2.0. In the case of Mozilla, we considered the whole suite. In the case of NetBSD, we only looked at the kernel of the operating system, while in the case of OpenSSH, which is part of OpenBSD, we focused at the subdirectory within OpenBSD where OpenSSH resides.
2. The first date recorded in the repository does not necessarily coincide with the creation date of the project. However, the earliest record in the log does not necessarily coincide with the start of the project itself as the decision to adopt CVS could have been made well into the development of the project: A case in point is GCC, which started well before the first recorded commit in 1997.
3. For now, we only consider the main branch and ignore activity in other development branches. More generally, it is not completely clear when a file is really part of the project's code base. That is, some files are explicitly deleted when they are no longer needed, but we cannot be sure that this policy is always enforced. Some files are "born dead" (which happens when a file is created in a branch other than the main branch). Sometimes files that are registered as dead are "revived". All of this is mainly *CVS*-specific.

4. Finally, it might be necessary to investigate at some point whether CVS accounts could be used by more than one maintainer, which could create another potential source of bias.

4 Empirical Investigations

To study collaborative maintenance activity, the econometric tests presented in this paper address two different measures for each file, the average number of maintainers per month ("maint's"), and the average number of revisions per month ("revisions"). The first measure can be considered as an indicator of collaborative maintenance while the second addresses activity more specifically.

However, previous investigations[20] have attracted our attention to the time variability of collaborative maintenance and activity on a given file. We had typically found that in 80 to 90% of the cases, only one maintainer had committed a change to a given file during a given month. As a consequence, we investigate also two other variables: the maximum number of maintainers per month over the period ("max maint's") and the maximum number of revisions of files per month over the period ("max revisions") in order to address this issue. These last two variables focus on intensive periods of maintenance and activity to deal with the fact that there are large periods of low activity, which is rather intuitive once said, but which we fear might create a significant bias: in doing so, they allow us to focus specially on periods of *teamwork*.

We run several specifications for all 10 projects, trying to explain four dependant variables (maint's, max maint's, revisions and max revisions) by the size of the file defined as its number of single lines of code ("SLOCs"), the maximum McCabe complexity index for all functions in the file ("McCabe"), Halstead volume ("Halstead") of the file, and the date of creation of the file ("Relative creation date").

Taking *Apache* as an example (Table 2), we find that:

a. maint's is explained positively by the relative creation date of the files: even controlling by their age, younger files attract on average more maintainers than older ones. A similar, but opposite, dependence characterizes max maint's: in that case, the older the file the higher the maximum number of maintainers during one month. Similar dependencies (positive for revisions and negative for max revisions), and therefore similar tentative explanations, characterize activity: still controlling by their age, younger files attract more activity on average, but a lower maximal activity per month. Generally, younger files tend to attract a higher average number of maintainers per month, and a higher average number of revisions per month, but lower maxima in both cases.

→ This could be explained by a larger global audience of the project, meaning that more recent files could attract more numerous maintainers just because the population of developers would be larger, because the growth of the total number of maintainers for the project over time, meaning that the files could therefore be "touched" by more maintainers simply because there are more maintainers in the project. At the same

time, older files have more intense (collaborative) maintenance & activity peaks: this could maybe be related to older files – files with an older *vintage* – being more attractive to development and maintenance activity because of their importance in the project, or to the fact that early development activity was more collaborative in itself, due for instance to the role of initial core teams.

b. File size, measured in SLOCs, does not explain the average number of maintainers per month on a file, nor the average number of revisions per month, except when associated with Halstead, but has always some explanatory power for the related maxima. McCabe is not significant for maint's, whereas it is for the 3 other dependent variables. Halstead is significant for the estimations of all 4 dependent variables, and renders SLOCs insignificant: indeed, Halstead is more strongly correlated to SLOCs (though both complexity variables actually are). Adjusted R2 are considerably higher for both maxima with Halstead.

→ This could be consistent with the idea suggested above that there are limited periods of intense activity for files, outside of which "normal" activity is less relevant for this kind of analysis. In all circumstances, Halstead has a strong explanatory power, which is relatively intuitive is we analyse it as a combination of size and complexity of code.

c. Results with Halstead are therefore presented in synthetic form for all 10 others projects in Table 3. There are only few differences such as the absence of explicative power of the relative creation date for *gaim*, except for maint's, which would notably deserve further and more specific investigations. The significance of SLOCs, and the sign of the dependence when it exist, appear more subject to variations than for Halstead, but might point more to a measurement issue more than to actual differences among projects, save at least for Python where it is probably in relation with the number of files written in python, precisely, and which have therefore been excluded for now from our analysis.

→ These results confirm the robustness of the findings and interpretations presented above, and suggest that these characteristics could generally characterize the open-source mode of development in large projects. Together with results obtained for Apache, they might also suggest more subtle dependencies associated with other measures of code size (SLOCs) or complexity (McCabe).

d. In this last respect, and turning back to Apache, Table 4 presents an additional estimation of max maint's using Halstead, SLOCs, McCabe, and Functions (which gives the number of functions in a file). Interestingly, all these variables are significant: a higher number of functions tends to significantly increase the maximum number of maintainers in a file; on the contrary, higher McCabe and SLOCs significantly decreases the number of maintainers.

→ This finding could be consistent with an enhanced division of labour between maintainers inside a given file when more modular, i.e. allowing for more maintainers when there are more functions; and with more complex and longer files being more difficult to maintain and less attractive for maintainers respectively.

e. Finally, Table 5 also presents a more complete estimation of max revisions using Halstead, SLOCs, McCabe, Functions and now Max maint's, as it appears reasonable indeed to suggest that the maximal activity on a file could be explained by the maximum number of maintainers. It is indeed so, and the relative creation date, SLOCs and Functions lose all statistical significance, which Halstead and McCabe retain.

→ This validates the idea that vintage explains the maximum number of maintainers on a file and thus indirectly its maximum activity, and also that the division of a file into functions is consistent with organizing maintainer collaboration more than with explaining activity per se. Halstead and McCabe have a strong positive and negative explanatory power vis a vis activity, respectively, controlling by the number of maintainers: therefore, they could also provide explanations for the attractivity of a file per se (in terms of contributions).

Generally speaking, and awaiting further confirmation of these results on a larger collection of open-source projects, our investigations suggest that a metrics of code size and complexity such as Halstead volume and file vintage are major determinants of teamwork on files. In this respect, the significance of vintage could be consistent with the idea that core teams play a specially significant role when projects are recent. In this general framework, more modular code – here, more functions in files – is associated with more maintainers, which is consistent with insights from modularity theory and with a more efficient division of labour. Still in this context, more complex files attract a lower number of collaborative maintainers, maybe because they induce a more exclusive selection of who could maintain a given piece of specially complex code. Finally, more "verbose" code – more lines of code for a given complexity – is less attractive for maintainers, perhaps because it could correspond to less attractive features inside projects. These findings appear consistent with suggestions[18,19] according to which maintainers would respond to technical considerations, either based on use value or on challenge and peer regard, in their motivations and in their choices among modules, and therefore in the global allocation of efforts in large open-source software projects.

5 Further Work

This paper documented investigations of detailed development records to study collaborative maintenance in open-source projects. The success that many of these projects have had in recent years and the voluntary nature of their development process make them extremely interesting to study, especially since abundant documentation of the development history of each project is readily available on the Internet. We came to the conclusion that collaborative maintenance in large open-source projects seems to be generally influenced by Halstead volume and also by the vintage of the files in a given project. Further studies are needed to uncover the role played by various factors which would be candidates to increase the explanatory power of the simple econometric models presented in this paper, including notably more technical characteristics of files. Furthermore, the extent to which maintainers

actually coordinate their work is not yet clear, nor are the dynamic interplay of the variables we have studied or the fact that such dynamics can give birth to hot spots. It could be interesting too to study more qualitatively subsets of files, and more deeply the interactions between maintainers within files.

Acknowledgments

Our research has been partly supported by *Calibre*, a EU FP6 Coordination Action. The support of the US National Science Foundation (NSF Grant No. IIS-0326529 from the DTS Program) is also gratefully acknowledged by one of us (JMD).

REFERENCES

1. C. U. Ciborra, *Teams, Markets and Systems* (Cambridge University Press, 1993).
2. J. Olson and K. M. Branch, in: Communication, Management Benchmark Study, edited by E. L. Malone (Office of Science, Department of Energy, Washington D.C., 2002) pp. 133–142.
3. W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters, Workflow Mining: A Survey of Issues and Approaches, *Data & Knowledge Engineering*, **47**, 237–267 (2003).
4. S. Koch and G. Schneider, Effort, Cooperation and Coordination in an Open Source Software Project: Gnome, *Information Systems Journal*, **12**(1), 27–42, (2002).
5. J. Y. Moon and L. Sproull, Essence of Distributed Work: The Case of the Linux Kernel, *First Monday*, **5**, (2000).
6. V. Valloppillil, Open source software: A (new?) development methodology. Microsoft memo, 1998 (unpublished).
7. J. Matusow, S. McGibbon, and D. Rowe, in: Proceedings of the 1st International Conference on Open Source Systems, edited by M. Scotto and G. Succi, (Genoa, 2005), pp. 263–266.
8. G. D. Prato and D. Gagliardi, in: Proceedings of the 1st International Conference on Open Source Systems, edited by M. Scotto and G. Succi, (Genoa, 2005), pp. 237–240.
9. E. S. Raymond, The Cathedral and the Bazaar, *First Monday*, **3** (1998).
10. S. Krishnamurthy, Cave or Community? An Empirical Investigation of 100 Mature Open Source Projects, *First Monday*, **6** (2002).
11. K. Crowston and J. Howison, The Social Structure of Free and Open Source Software Development, *First Monday*, **10** (2005).
12. W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings – Software*, **149**, 24–39 (2002).

13. A. Mockus, R. T. Fielding, and J. D. Herbsleb, Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, **11**, 309–346 (2002).
14. M. Hahsler and S. Koch, Discussion of a Large-Scale Open Source Data Collection Methodology, *Proc. HICSS* **38**, (2005).
15. G. Robles, S. Koch, and J. M. González Barahona, Remote Analysis and Measurement of Libre Software Systems by Means of the CVSAnalY Tool, *Proc. ICSE* **2**, (2004).
16. J. Howison, M. Conklin, and K. Crowston, in: Proceedings of the 1st International Conference on Open Source Systems, edited by M. Scotto and G. Succi, (Genoa, 2005), pp. 54–60.
17. A. Capiluppi, A. E. Faria, and J. F. Ramil, Exploring the Relationship between Cumulative Change and Complexity in an Open Source System, *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering* (2005).
18. J.-M. Dalle and P. David, Simulating Code Growth in Libre (Open-Source) Mode, in: *The Economics of the Internet*, edited by N. Curien and E. Brousseau, (Cambridge University Press, 2005).
19. J.-M. Dalle and P. David, The Allocation of Software Development Resources in 'Open Source' Production Mode, Discussion Paper 02-27, 2003 (Stanford Institute for Economic Policy Research).
20. M. den Besten and J.-M. Dalle, Assessing the Impact of Product Complexity on Organizational Design in Open Source Software: Findings & Future Work, *Proceedings of the ECCS 2005 Satellite Workshop: Embracing Complexity in Design* (2005).

ANNEX

Table 1: Descriptive elements of the sample in the database. Other statistics available upon request.

	First month of act.	Files (#)	"c" files (#)	maint's (total #)	maint's (av)	max maint's (av)	revisions (av)	max revisions (av)	McCabe (av)	Halstead (av)	SLOCs (av)
apache	07/96	4133	657	79	7.67	2.60	32.38	5.96	18.24	14483.73	523.85
cvs	12/94	1062	287	30	3.67	1.41	23.74	3.01	19.25	16643.53	1456.00
gaim	03/00	5158	681	39	3.62	1.74	26.91	4.62	17.10	25181.14	3581.71
gcc	08/97	34757	16405	250	2.56	1.19	6.30	1.46	17.62	4526.51	3546.63
gostscript	03/00	2819	932	23	3.68	1.76	9.08	1.76	25.04	21445.66	3197.25
mozilla	03/98	40545	8370	595	7.77	1.90	21.11	3.31	15.39	18064.63	1606.94
NetBSD	03/93	19514	7081	267	6.48	1.66	18.00	2.94	10.03	15846.91	7805.33
openssh	09/99	289	138	50	5.32	2.21	35.56	4.93	19.67	13779.09	9230.17
postgresql	07/96	4102	904	25	4.53	1.92	42.00	4.38	18.75	17190.52	1246.06
python	08/90	4643	419	88	5.94	1.94	31.59	4.78	21.53	33965.03	14453.06

Table 2: Econometric estimations (OLS) for *Apache*. Dependent variables: average number of maintainers per month, maximum number of maintainers per month, average number of revisions per month, and maximum number of revisions per month (parameter estimate, above, and standard error, below). Stars signal confidence levels – 95% = *, 99% = **, and 99.9% = ***.

	maint's	maint's	Max maint's	Max maint's	revisions	revisions	Max revisions	Max revisions
Intercept	0.14039***	0.12214***	3.21049***	2.93030***	0.60443***	0.39800***	6.56893***	4.98504***
	0.02860	0.02683	0.11575	0.09999	0.12597	0.11208	0.58216	0.48336
SLOCs	1.165E-5	4.30E-6	4.090E-5**	-1.189E-4**	1.3514E-4	1.579E-5***	7.047E-4***	-2.0944E-4
	1.227E-5	1.212E-5	5.047E-5	4.553E-5	5.404E-5	5.064E-5	2.5383E-4	2.2009E-4
Mc Cabe	6.8159E-4	------	7.39E-3***	------	8 89E-3**	------	4.469E-2***	------
	6.4246E-4	------	2.56E-3	------	2.83E-3	------	1.288E-2	------
Halstead	------	2.12E-6***	------	3.280E-5***	------	2.786E-***	------	1.855E-4***
	------	7.54826E-7	------	2.83E-6	------	1.5E-6	------	1.370E-5
Relative creation date	9.11E-3***	8.95E-3***	-3.433E-2***	-3.057E-2***	1.824E-2***	1.948E-2***	-7.575E-2***	-5.449E-2***
	1.05E-3	9.9214E-4	4.10E-3	3.58E-3	4.61E-3	4.15E-3	2.060E-2	1.733E-2

Table 3: Summary of econometric tests (OLS) for all 10 projects with variable Halstead. Full results, including results with variable McCabe, available upon request. Stars signal confidence levels – 95% = *, 99% = **, and 99.9% = ***; (-) signals a negative coefficient.

Project	*Apache*				*CVS*				*Gaim*			
	maint's	max maint's	revisions	max revisions	maint's	max maint's	revisions	max revisions	maint's	max maint's	revisions	max revisions
Intercept	***	***	***	***	***	***	***	***		***	***	***
SLOCs			***								***	**
Halstead	***	***	***	***		***	***	***		***	***	***
Relative creation date	***	(-)***	***	(-)***	***	(-)***	***	(-)***	***			
Adjusted R2	0.1456	0.3272	0.1792	0.3210	0.1668	0.2392	0.2233	0.3601	0.2116	0.0256	0.5750	0.0615

Project	*GCC*				*Ghostcript*				*Mozilla*			
	maint's	max maint's	revisions	max revisions	maint's	max maint's	revisions	max revisions	maint's	max maint's	revisions	max revisions
Intercept	***	***	***	***	***	***	***	***	***	***	***	***
SLOCs	(-)***	***	(-)***	***	(-)**	***	***	***	(-)*	***		**
Halstead	***	***	***	***		***	***	***	***	***	***	***
Relative creation date	***	(-)***	***	(-)***	***	(-)***	***	(-)***	***	(-)***		(-)***
Adjusted R2	0.2259	0.3091	0.2731	0.3334	0.0360	0 4648	0 2028	0.4648	0.0313	0.2623	0. 1341	0.2862

Project	*NetBSD*				*OpenSSH*				*PostgreSQL*			
	maint's	max maint's	revisions	max revisions	maint's	max maint's	revisions	max revisions	maint's	max maint's	revisions	max revisions
Intercept	***	***	***	***	***	***	***	***	**	***	***	***
SLOCs		***	***	***	***	***		***		**		
Halstead	***	***	***	***		***	***	***		***	***	***
Relative creation date	***	(-)***	***	(-)***		(-)***	(-)***	(-)***	***	(-)***	***	(-)***
Adjusted R2	0.0780	0.2765	0.1001	0 2496	0.1657	0.5289	0.3576	0.4869	0.1413	0.2593	0.1814	0.3229

Project	*Python*			
	maint's	max maint's	revisions	max revisions
Intercept		***		***
SLOCs	(-)**	(-)***	(-)**	(-)***
Halstead		***	***	***
Relative creation date	***	(-)***	***	
Adjusted R2	0.0946	0.2830	0.0553	0.1137

Table 4: Further econometric estimations (OLS) for *Apache*. Dependent variables: maximum number of maintainers per month and maximum number of revisions per month (parameter estimate, above, and standard error, below). Stars signal confidence levels – 95% = *, 99% = **, and 99.9% = ***.

	Max maint's	Max maint's	Max maint's	Max revisions	Max revisions	Max revisions
Intercept	3.21049***	2.93030***	3.07637***	6.56893***	4.98504***	-2.61835***
	0.11575	0.09999	0.10384	0.58216	0.48336	0.75211
SLOCs	4.090E-5**	-1.1890E-4**	-5.1764E-4***	7.047E-4***	-2.0944E-4	1.00E-3
	5.047E-5	4.553E-5	1.2736E-4	2.5383E-4	2.2009E-4	5.4907E-4
Mc Cabe	7.39E-3***	------	-9.55E-3**	4.469E-2***	------	-3.457E-2**
	2.56E-3	------	2.97E-3	1.288E-2	------	1.272E-2
Halstead	------	3.280E-5***	4.708E-5***	------	1.855E-4***	1.0795E-4***
	------	2.83E-6	3.96E-6	------	1.370E-5	1.918E-5
Functions	------	------	8.26E-3***	------	------	-1.657E-2
	------	------	2.41E-3	------	------	1.032E-2
Max maint's	------	------	------	------	------	2.64696***
	------	------	------	------	------	0.19833
Relative creation date	-3.433E-2***	-3.057E-2***	-3.053E-2***	-7.575E-2***	-5.449E-2***	-2.358E-2
	4.10E-3	3.58E-3	3.61E-3	2.060E-2	1.733E-2	1.643E-2
Adjusted R2	0.1580	0.3272	0.3669	0.0909	0.3210	0.5243