arXiv:0802.3170v1 [physics.soc-ph] 21 Feb 2008

Fat tails, long memory, maturity and ageing in open-source software projects

Damien Challet
Institute for Scientific Interchange, via S. Severo 65, 10113 Turin, Italy and
Département de Physique, Université de Fribourg, Pérolles, 1700 Fribourg, Switzerland[*]

Sergi Valverde
ICREA-Complex Systems Lab, Pompeu Fabra University, Dr. Aiguader 80, 08003 Barcelona, Spain and
Centre de Recherches sur la Cognition Animale, CNRS-UMR 5169,
Université Paul Sabatier, 118, route de Narbonne, 31062 Toulouse Cedex 04 France [†]

We report activity data analysis on several open source software projects, focusing on time between modifications and on the number of files modified at once. Both have fat-tailed distributions, long-term memory, and display systematic non-trivial cross-correlations, suggesting that quiet periods are followed by cascading modifications. In addition the maturity of a software project can be measured from the exponent of the distribution of inter-modification time. Finally, the dynamics of a single file displays ageing, the average rate of modifications decaying as a function of time following a power-law.

I. INTRODUCTION

The time between consecutive events observed in many human activities is neither Poissonian nor Markovian, but exhibits bursts of rapidly occurring events separated by long periods of inactivity. The distribution of interevent times follows heavy-tailed distributions [1, 2, 3, 4, 5].

Although activity patterns resulting from an aggregated behaviour, such as financial markets, have been studied for a long time, recent work focused on the individual behaviour. It has been shown that in some cases non-Poissonian behaviour is not only a by-product of human interaction, but can be traced back to individuals. Some remarkable examples are web surfing and e-mail communication [1, 2, 6] . An important issue is the relationship between individual behaviour and aggregate activity, which we shall investigate with the help of an important, yet under-explored, archive of human activity: open-source software (OSS) development. Distributed communities of programmers develop open-source software projects [7], where several programmers change simultaneously possibly more than one piece of software.

In the past, indices of global development activity have been devised to assess the stability of a software system. A popular measure, called software volatility, measures the number of enhancements per unit of time over a specified time frame [8]. High volatility is often associated to high maintenance costs; according to this point of view, when the volatility exceeds some threshold it may be more economical to rewrite the entire system from scratch instead of maintaining an aged (and unstable) software system [9, 10]. We show here however that the probability distribution of scaled waiting times of the software projects converges towards a universal function, providing *a contrario* a measure of software maturity.

We also investigate the relationship between individual programmer behaviour and global project activity by means of a detailed statistical analysis of both the timing and size of individual actions. Unlike previous studies (e.g. [10]), we do not attempt to make any distinction between different types of modifications.

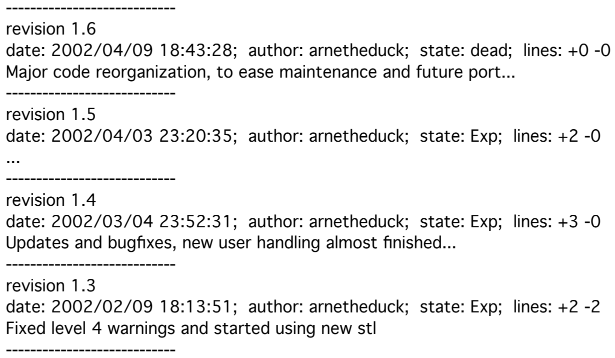
The vast majority of software projects keep track of every single change and its author, using various so-called version control systems. CVS (concurrent versioning system) is a frequently used system in open source projects. In order to avoid costly information losses, the CVS keeps each programmer apart by managing multiple revisions for each project file. These features makes the CVS an invaluable source of information about software evolution and programmer activity patterns. We have analyzed the CVS databases of six large-scale open-source projects from their creation date until November 2005: MOZILLA, APACHE, FREEBSD, OPENBSD, NETBSD, and POSTGRESQL.

[*] Electronic address: challet@isi.it
[†] Electronic address: svalverde@imim.es

WIOPT 2008, 1st − 3rd Apr 2008, Berlin, Germany.
Copyright © 2011 − 2012 ICST ISBN 978-963-9799-18-9
DOI 10.4108/ICST.WIOPT2008.3219

II. ANALYING CVS LOGS

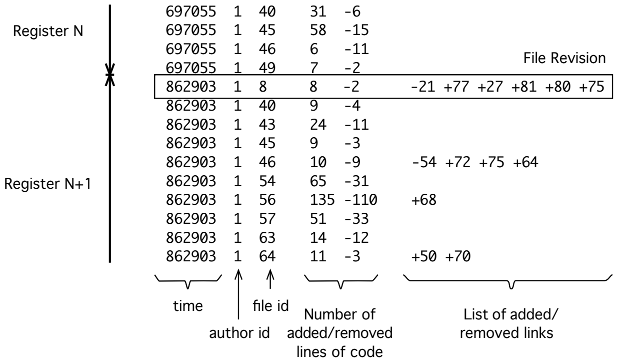
Analyzing operational logs of OSS project developments stored in CVS code repositories requires sometimes delicate ad-hoc pre-processing, without which spurious effects can easily spoil the results in an uncontrollable way. Indeed, while CVS stores all of its information in a text file describing in a human-legible way the nature of each modification (see Fig. 1), there is no standard format definition; it takes therefore some efforts to extract valuable and trustable information from CVS log files.



```
----------------------------
revision 1.6
date: 2002/04/09 18:43:28;  author: arnetheduck;  state: dead;  lines: +0 -0
Major code reorganization, to ease maintenance and future port...
----------------------------
revision 1.5
date: 2002/04/03 23:20:35;  author: arnetheduck;  state: Exp;  lines: +2 -0
...
----------------------------
revision 1.4
date: 2002/03/04 23:52:31;  author: arnetheduck;  state: Exp;  lines: +3 -0
Updates and bugfixes, new user handling almost finished...
----------------------------
revision 1.3
date: 2002/02/09 18:13:51;  author: arnetheduck;  state: Exp;  lines: +2 -2
Fixed level 4 warnings and started using new stl
----------------------------
```

FIG. 1: An example of CVS log entry generated by the *cvs log* command for a file in the DCPP project. Each entry describes a single file revision, indicating revision number, date, author's name, state, number of lines added and removed and a brief description. The raw CVS log is human-readable but difficult to analyze by automated means.

We parsed CVS logs and generated synthetic files (thereafter *event logs*) which are more amenable to analysis (see Fig. 2 for an example). Every entry in this file describes a single file revision and provides information about the event time, author's identifier, unique file identifier, number of changed lines of code (added and/or removed) and a detailed list of link changes (added and/or removed) (see below).



```
697055  1  40    31  -6
697055  1  45    58  -15
697055  1  46    6   -11
697055  1  49    7   -2
862903  1  8     8   -2      -21 +77 +27 +81 +80 +75
862903  1  40    9   -4
862903  1  43    24  -11
862903  1  45    9   -3
862903  1  46    10  -9      -54 +72 +75 +64
862903  1  54    65  -31
862903  1  56    135 -110    +68
862903  1  57    51  -33
862903  1  63    14  -12
862903  1  64    11  -3      +50 +70
```

Register N
Register N+1
File Revision
time
author id
file id
Number of added/removed lines of code
List of added/ removed links

FIG. 2: An example of *event log* file generated from the CVS log file of the DCPP project. This file format allows for simpler numerical analysis of development patterns.

A. Filename changes

Unfortunately, CVS logs have a number of shortcomings. For instance, CVS does not handle file rename or file moves, making it difficult to rebuild faithfully the evolution of software structure. Instead, every renamed file generates a new file in the CVS, resulting in two entries for the same file and giving rise to specific patterns in the space-time map (see Fig. 3).

There is no general easy way to implement name aliasing. Two simple approaches can address this problem: (i) to deal only with the set of CVS registers before the actual file rename takes place or (ii) to fix the CVS manually by providing a list of filename aliases. Which scheme to choses depends on the size of CVS log. For instance, (i) is a

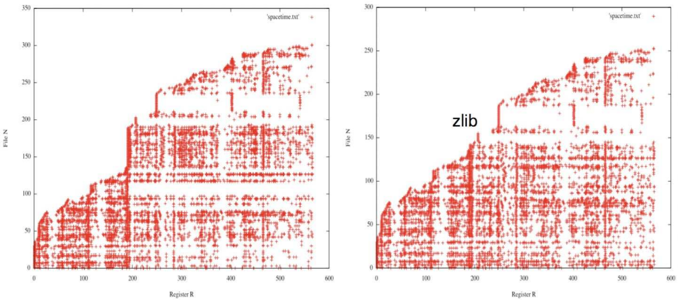


FIG. 3: Activity map of DCPP before and after filename aliasing; each point represents a single file change. In the right hand side figure, the label ZLIB points at a stream of modifications carried out on an auxiliary software library.

suitable approach for very large CVS logs with few file renames. On the other hand, the small size of some CVS logs allows the file moves to be fixed by human inspection. Moreover, one cannot discard registers from small CVS logs without affecting the statistics (for instance, the long tail of the distribution of size of changes).

Some peculiar patterns in the space-time diagram appear as groups of files change together in a remarkably synchronized fashion (see the upper region of the plot in Fig. 3b). These patterns represent synchronization events between auxiliary software components (i.e., libraries or frameworks) and the core software application . The external software library evolves in parallel and is maintained by external software team. However, the interaction between these systems is often asymmetric. Changes to the auxiliary software library are exogenous perturbations in the core application; interaction rarely happens the other way around. A statistical way of detecting such processes was proposed by one of us [11].

B. Software structure

Because of software reuse, files link to each other, thereby defining a network of dependency which plays an important role in software dynamics (see e.g. [12, 13, 14, 15]). Visualizing structural changes helps understanding the microscopic dynamics. However, CVS log files do not contain any structural information.

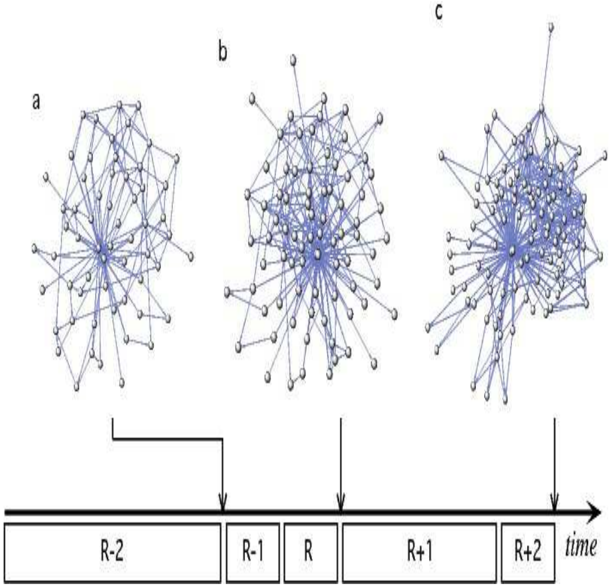


FIG. 4: Mapping between software networks and the stream of CVS registers. Here, software network a maps to register $R-2$, software network b maps to registers $R-1$ and R and finally, software network c maps to registers $R+1$ and $R+2$.

The first step is to obtain enough source code versions for a given project. Since this can generate a large volume of information, one has to limit oneself to a subset of all versions for large OSS projects. Then, it is easy to define a mapping between the sequence of software networks and the time evolution of CVS registers (see Fig. 4) and to add information about insertion/deletion of new files and links.

III. DATA ANALYSIS

From each CVS database history, we analyze the time, the author and the number of modified files of each source code alteration. Each development history comprises a number M of modification registers. Hence, the i-th modification was at time t_i by programmer a_i and concerned $s_i >= 1$ files, $1 \leq i \leq M$. Modification attributes can be obtained as the sum of individual contributions. Lets $c_{jk}(t) = 1$ if programmer j modifies file k at time t and $c_{jk}(t) = 0$ otherwise. Then,

$$S_i = \sum_{1 \leq k \leq N_f} c_{a_i k}(t_i) \tag{1}$$

measures the number of changed files at time t_i, thereafter called modification size.

A. Interevent and modification size distributions

We will be interested in the interevent distribution $P(T)$ of between two consecutive modifications The time elapsed between two consecutive modifications i and $i-1$ is denoted by $T_i = t_i - t_{i-1}$. It reflects a maturation process at a particular stage of software development and depends on many factors, such as the cognitive capabilities of programmers [16], team composition [17], software usage or deadlines. We report the distribution of interevent times at three different scales: project, individual developers, and files. In order to differentiate them, we adopt the following notation: $T_i^{(f)}$ is the time interval between two consecutive modifications of file f, $T_i^{[a]}$, the time interval between two consecutive modifications made by author a, while the absence of a superscript denotes the global project level. Therefore, we focus on $P(T)$, $P(T^{(f)})$, $P(T^{[a]})$, $P(S)$ and $P(S^a)$, and how they are related to each other.

Previous work produced plots *en masse* of the number of files modified by a given author, or the number of modifications contributed per author, without further analysis such as fitting, scaling or discussion of stationarity [18]. A related work consisted in measuring the distributions of number of lines of code added or deleted per modification, which are both clean power-laws with exponent around $-3/2$ [19], whose origin is still unclear. Previous work has compared Apache and Mozilla in a qualitative way [20]. Finally, recent work studied the size of modifications and carried out a detrended fluctuation analys [21]

B. Time series analysis: caveats

Human beings have intrinsic time scales. Despite this obvious fact, one finds frequent bursts of modifications from the same programmer occurring with super-natural high frequencies. This comes from the way some programmers submit their modifications to CVS with automated scripts, for instance every 5 seconds. One must therefore coarsen the time series so as to remove non-human dynamics by merging the modifications i and $i+1$ separated by less than δT seconds, i.e., $t_{i+1} - t_i < \delta T$; the influence of δT is discussed thereafter.

The other fundamental problem is that the dynamics of software projects is seldom stationary during their whole histories. Indeed, one characteristics of the successful large-scale open-source projects analyzed here is to gradually attract the attention of the public and of companies, resulting into an increasing number of programmers, denoted with. $N_a(t)$. In addition, in any software project, the number of files $N_f(t)$ increases as a well. Since we are first focusing on global activity patterns, this may not be a problem if for instance the activity of each programmer, or the rate of modifications of a given file decreases in a related manner.

A simple and effective way to measure the change of programming pattern is to plot measures of cumulated activity. The cumulative number of modified files as a function of the time t, defined as

$$\Sigma(t) = \sum_{t_i \leq t} S_i \tag{2}$$

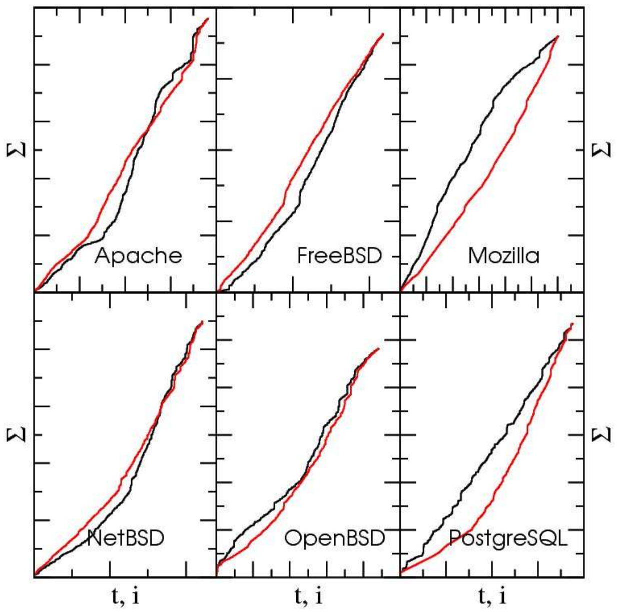


FIG. 5: Cumulative number of modified files Σ as a function of time t (black lines) and of modification number i (red lines). The units of t and i, $\Sigma(t)$ and $\Sigma(i)$ have been chosen so as to make the end of the time series coincide.

is a good candidate, as its slope is equal to the rate of modified files per unit of time and reflects therefore the global activity of the project. Fig 5 reports $\Sigma(t)$ for six software projects. Some of them have a relatively constant rate of modification after a transient period (FreeBSD, NetBSD), while PostgreSQL has a remarkably constant modification rate. Apache has a more erratic behaviour [28], while OpenBSD experiences various episodes of high and low activity, the activity reducing lately. Finally, Mozilla's activity has been decreasing for much of its history, which can be a sign of software maturing.

$\Sigma(t)$ does not characterize entirely the activity of a project. Let us define a related quantity, the cumulative number $\Sigma(i)$ of modified files as a function of modification number i:

$$\Sigma(i) = \sum_{j \leq i} S_j \tag{3}$$

The slope of $\Sigma(i)$ is equal to the average modification size $\langle S \rangle$ and therefore provides an additional criterion of stationarity. The example of PostgreSQL is striking in this respect: although the slope of $\Sigma(t)$ is remarkably constant, $\Sigma(i)$ increases superlinearly for at least half of the time series before reaching a constant slope, meaning that the number of modifications per batches has increased. This increase, although usually less spectacular, is seen in almost all the projects. This could reflect the evolution of the software structure: changes are likely to propagate to nearest neighbors, hence $\langle \Sigma(i) \rangle$ follows in part the evolution of the average number of nearest neighbours; and indeed one of us [11], found an initial increase of propagation of changes to nearest neighbors and then a saturation, confirming the above interpretation. The only exception here is Mozilla whose modification rate decreases at the end of its time series, while the average number of modified files shows no sign of decrease; this is the signature of a genuine slowdown of the development.

In short, a transient state is generally present at the beginning of projects histories and a steady state is reached after some time. Mozilla is clearly not in a steady state in the last part of the history considered here, its modification rate having slowed down significantly. OpenBSD behaviour changes much with time, and Apache is erratic. At any rate, the history of a software project is generally not very smooth, the first part of its development is different from the following ones, hinting at how crucial it is to split the timeseries into several parts, which will be confirmed in the next subsections.

Plotting S_i as a function of t_i, and T_i and S_i as a function of i reveals a highly non-trivial temporal structure (Fig 6). In particular, this figure displays two distinctive features found in all the datasets we studied: non-Gaussianity and clustered activity, which we will characterize in details.

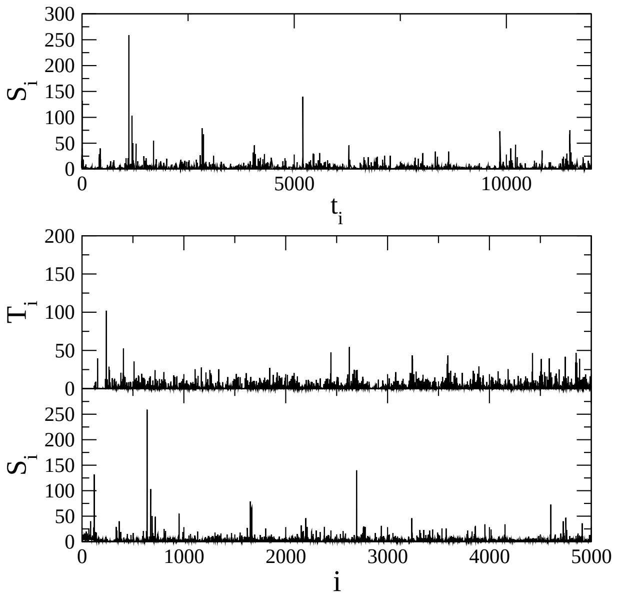


FIG. 6: Activity patterns: number of modifications vs time of modification (upper graph), time interval between two modifications (middle graph) and number of modified files (lower graph) as a function of the modification number (Inkscape).

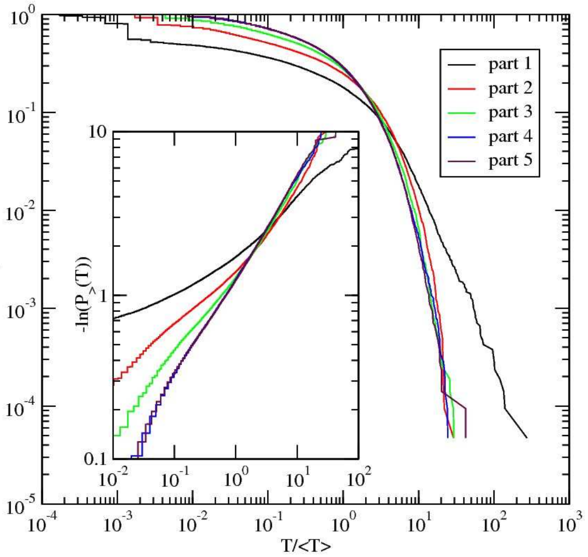


FIG. 7: Rescaled cumulative distribution of time delay between two modifications of FreeBSD; each part contains an equal number of modification batches.

C. Time between modifications

Non-stationarity, particularly frequent at the beginning of the history of the projects, suggest to split the time series into several parts. We performed it in such a way that the number of modification batches is the same in each part. Focusing on FreeBSD and plotting the cumulative distribution of T, denoted by $P_>(T) = \int_T^\infty P(t)dt$ for the five parts reveals a pattern shared by all the projects studied here (Fig. 7): the first parts have a broader distribution than the subsequent ones, sometimes with a clear power-law tail, and then converges to a stable law whose tail is very well fitted with a stretched exponential, i.e. a Weibull distribution,

$$P_>(T) \simeq e^{-\left(\frac{T}{\langle T\rangle c}\right)^{\alpha}} \tag{4}$$

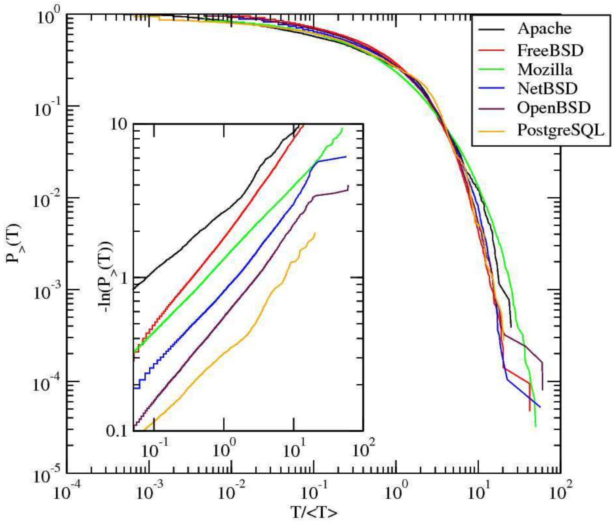


FIG. 8: Rescaled cumulative distribution of time delay between two modifications of the last part of various projects. Inset: $-\log P_>(T)$, shifted vertically for the sake of clarity.

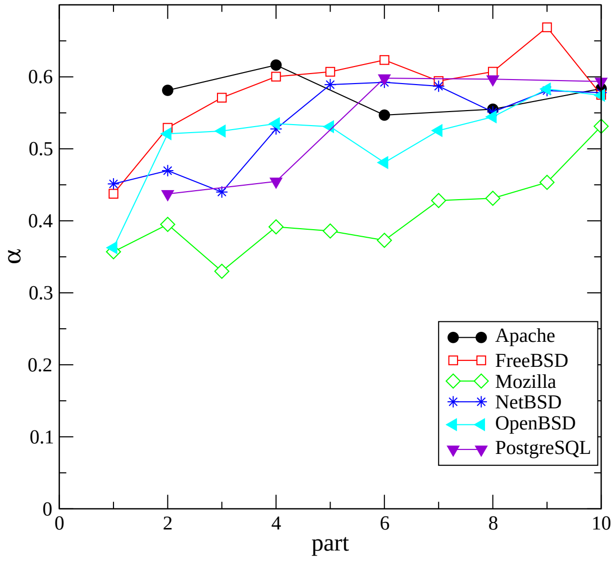


FIG. 9: Time evolution of the stretched exponential exponent α as a function of the time. Time series divided into 10 parts, except for Apache and PostgreSQL (5 parts).

for large $T/\langle T\rangle$, which can be seen by the straight line behaviour of $-\log P_>(T)$ in a log-log plot (inset of Fig 7). For instance, the cumulative distribution functions of the last two parts of FreeBSD are indistinguishable. In all the projects analysed, $P_>(T)$ seem to converge to the same functional shape given by Eq. (4) (Fig 8), although less clearly for PostgreSQL and Apache. The parameters α were obtained for the last part of the time series by maximum likelyhood estimation (see e.g. [22]) for $T/\langle T\rangle > 0.1$, resulting in 0.62 (FreeBSD), 0.57 (NetBSD), 0.58 (OpenBSD), 0.58 (Apache), 0.48 (Mozilla), 0.59 (PostgreSQL). One sees therefore that α is consistently about 0.58, except for FreeBSD and Mozilla. The time evolution of α is reported in Fig 9: generally speaking, α increases as a function of time and then saturates at $0.58 - 0.60$. Mozilla's α is still increasing, hence, may reach ~ 0.59 someday, while PostgreSQL's $\alpha = 0.59$ is stable. Coarsening the time series T_i in order to group batches of modifications does not alter the generic shape of $P_>(T)$, but changes the exponent α.

In other contexts, several works report a power-law behaviour of $P_>(T)$: the cumulative distribution of intervals between print requests is $P_>(T) \propto T^{-3/4}$ [23], while that of financial market transactions has a fat tail [24, 25]. Software development is therefore clearly different, but the fact that $P_>(T)$ is not Poissonian indicates that there is some kind of interaction/correlation between the programmers.

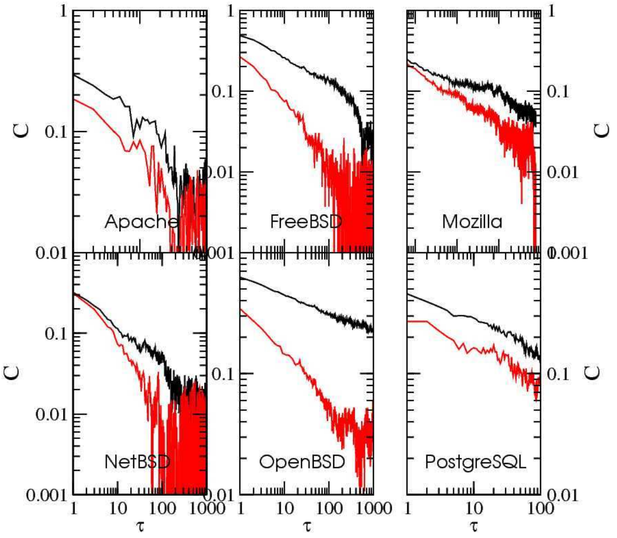


FIG. 10: Auto-correlation function of time interval between two modifications of various programs (black lines: first part, red line: second part of the project history)

In order to test the presence of long memory in T_i, we computed its auto-correlation function

$$C(\tau) = \frac{\langle T_i(t+\tau)T_i(t)\rangle - \langle T_i\rangle^2}{\langle T_i^2\rangle - \langle T_i\rangle^2}. \tag{5}$$

A slow decrease of $C(\tau)$, i.e., $C(\tau) \propto \tau^{-\beta}$ with $\beta < 1$, is a sign of long-memory. Unveiling long memory requires as long time series as possible, but we face the problem that the time series are not stationary. Hence we split the time series into two parts of equal size. According to Fig. 10, the auto-correlation functions of the first parts are always larger than those of the second parts, which is possibly due to the increase of the number of programmers. The exponent β of the second parts are comprised between 0.42 and 0.61 β (see Table A in appendix).

The presence of long memory at the macroscopic scale, that is, from the point of view of the CVS system collecting the modifications, means intuitively that periods of high activity are likely to be followed by periods of high activity, and reversely, suggesting the existence of cascading modifications. However, it is possible to convince oneself that large submissions of size S drawn from a power-law distribution $P(S) = \alpha S^{-\alpha-1}$ split into a number of chunks proportional to S give a power-law to $C(\tau)$ (see [26] for more details). The way to make sure that this long memory is genuine is to take a $\delta T > 0$ and check that $C_{\delta T}(\tau)$ still decreases as a power-law. Since the distribution of T has fat tails, detecting long memory is made easier by computing the auto-correlation function of $\log T_i$, which decreases the importance of large fluctuations of T_i; the resulting auto-correlation function is much less noisy but has a different exponent. Increasing δT of course decreases $C(\tau)$, which becomes more noisy. We found that $C(\tau)$ is still a power-law for $\delta T \leq 0.5$ hour, whereas the rapid sequences of modifications are typically separated by at a few seconds at most. Therefore we conclude that the long memory of T_i is genuine and that there are cascades of modifications.

D. Modifications

Fig. 11 reports the cumulative distributions of modification size $P_>(S)$. The groups of submissions separated by less than one minute each have been merged in order to give cleaner distributions. From the plot one concludes that $P_>(S)$ has generally a fat tail. Fitting the data with a power-law $P_>(S) \propto S^{-\gamma+1}$ should be done carefully,[29] as most cases do not have a pure power-law, and also because all the data sets do not have the same functional form. For instance, all the BSDs have a power-law core with exponent $\gamma \simeq 2$ with a cut-off. Mozilla has a power-law tail with $\gamma \simeq 2.4$ and no cut-off, while Apache has $\gamma \simeq 2.5$ with a strong cut-off. PostgreSQL is clearly irregular, hence we do not try to find its γ.

When $\gamma \simeq 2$, $P_>(S)$ is stable if one splits the time series into several parts, as it is Levy stable. On the other hand, the exponent of Mozilla changes from part to part, which is of an other clue of the non-stationarity of the project. In short, the exponent γ has no universal value. In any case, γ is markedly different from that of the distribution of added or deleted lines, which follows a clean power-law with exponent 3/2 [19]. A recent study found independently

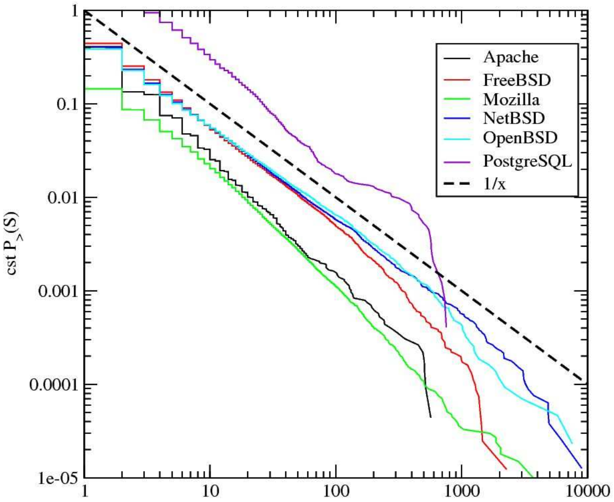


FIG. 11: Cumulative distributions of the number of modified files per modification batch ($\delta T = 0.017$ hours $= 1$ minute). The distributions have been shifted for the sake of clarity.

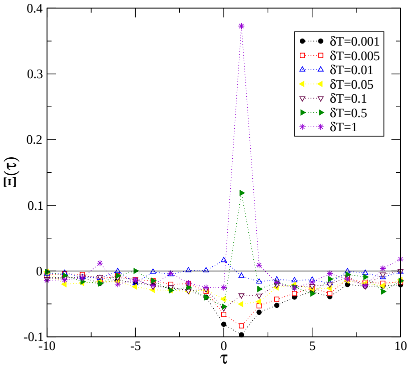


FIG. 12: Log Cross-correlation $\Xi(\tau)$ of FreeBSD for increasing δT expressed in hours

fat tails for $P_>(S)$, although with different exponents [21]. It is very tempting to relate this finding to the exponent of incoming links in the software network, which as an exponent ranging from 2 to 2.4 depending on the way one measures it [14, 15, 27]: sometimes, when a programmer modifies the code of a given file, it is necessary to change all the files linking to it as well.

The timeseries of number of changed files $S_i(t)$ has also long memory, although the auto-correlation function $C(\tau)$ is noisier than the one for time intervals, preventing us to try and obtain an exponent. However, once again, the auto-correlation function of $\log S$ is cleaner and we obtained the exponent δ of the log auto-correlation function of two largest datasets, i.e., FreeBSD (0.78) and Mozilla (0.48) ; the auto-correlation functions of the other projects, while clearly displaying long memory, were too noisy to be fitted. Non-trivial detrended fluctuation plots of modification size in [21] also indicate long memory.

E. Cross-correlations

The cross-correlation between the time intervals T and modification size S is defined as

$$\frac{\langle T(t)S(t+\tau)\rangle - \langle T\rangle\langle S\rangle}{\sqrt{\langle (T-\langle T\rangle)^2\rangle\langle (S-\langle S\rangle)^2\rangle}} \tag{6}$$

where $|\tau|$ indicates the time delay. The fat-tailed nature of their underlying distributions of $P(T)$ and $P(S)$ makes it difficult to detect any cross-correlation pattern. This problem is overcome by computing the cross-correlation of $\log T$

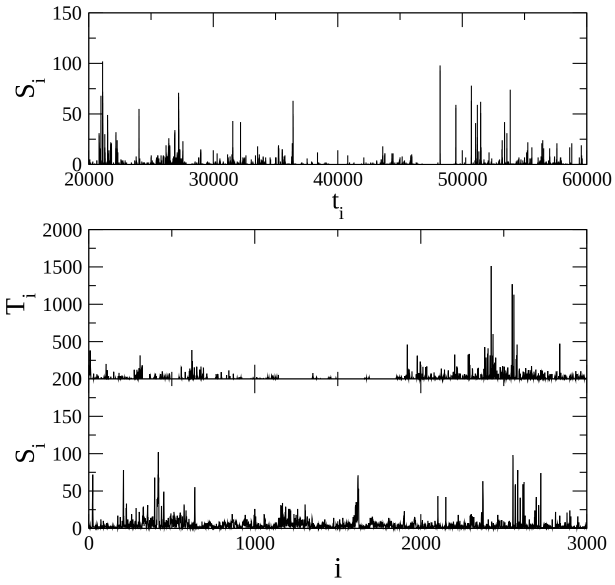


FIG. 13: Activity patterns of developer jst of Mozilla: number of modifications vs time of modification (upper graph), time interval between two modifications (middle graph) and number of modified files (lower graph) as a function of the modification number.

and $\log S$, denoted by $\Xi(\tau)$. The most interesting part of Ξ is for small values of $|\tau|$. For example, if $\Xi(0) < 0$, this means that a longer than usual wait results in a smaller than usual number of modified files. Reversely, $\Xi(0) > 0$ if a programmer works for a while on many files and then submits the changes at once. Therefore $\Xi(0)$ is much influenced by steady submissions of modification batches separated by few seconds. Fig 12 reports that $\Xi(0)$ is significantly negative when $\delta T = 0$, then increases as a function of δT until it reaches the noise level for δT of the order of a minute, and then decreases again. This not only supports the hypothesis of cascading modifications, but also shows that there are fewer than average files modified after a long wait. $\Xi(1)$ is markedly different: first significantly negative, it increases as a function of δT, becomes positive and very large for $\delta T > 0.5$ hour. Its increase is yet another sign that modifications are submitted by cascades.

IV. INDIVIDUAL DEVELOPERS

The same analysis can be performed at the level of individual developers. Fig. 13 plots $T^{(\mathrm{jst})}$, the time between two modifications of a Mozilla developper nicknamed jst: the individual time series shows a much greater variability than that of a whole project. Individual actions have also long memory, as confirmed by Fig. 14.

When studying the dynamics of individual, the question of stationary state is of utter importance, and plots of $\Sigma^{(\mathrm{a})}$ as a function of i and t must be carried out, since the activity pattern of a programmer may change abruptly. We took therefore care of selecting stationary periods when plotting of $P_>(T^{(\mathrm{a})})$ for the programmers studied here. Of these developers, only the ones labelled 1 and 91 were still contributing at the end of the time series, developer 91 being active in the second half of the history of the project. There is an obvious change of behaviour at $T^{(\mathrm{a})} > 24$ hours: $P_>(T^{(1)})$ and $P_>(T^{(91)})$ are stretched exponentials for $0.5 < T^{(a)} < 24$ hours, and a power-law with exponent 2 for longer times; the other two programmers (52 and 16) have not the same waiting time distributions. This may reflect the variety of personal behaviour, but also be related to the particular type of work done by each programmer: for instance creating a whole new part of a program is more complex than translating its user interface, hence the power-law of waiting times of programmers 1 and 91 may reflect the structure of the program on which they worked.

Finally, the collapse plot of S (Fig. 16) is also convincing and suggests that $P_>(S)$ is a superposition of single individual distributions of roughly the same functional form as the global distribution.

All the above provides evidence that some global properties of software projects are found again at a microscopic level.

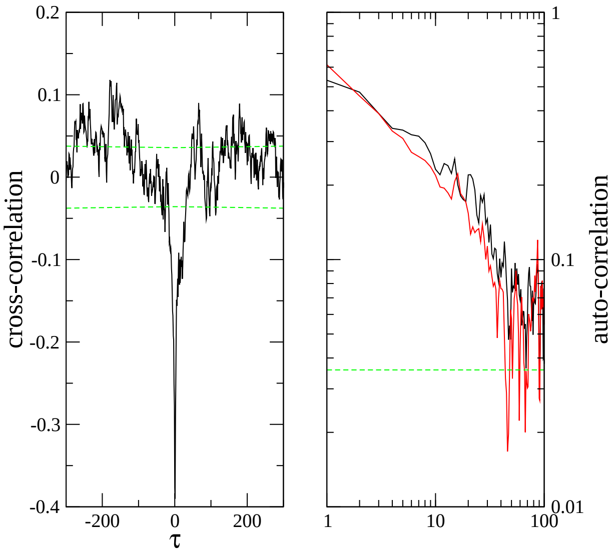


FIG. 14: Left figure: cross-correlation function of $T^{(jst)}$ and $S^{(jst)}$. Right figure: auto-correlation of $\log T^{(jst)}$. The dotted lines delimit noise at 99% confidence.

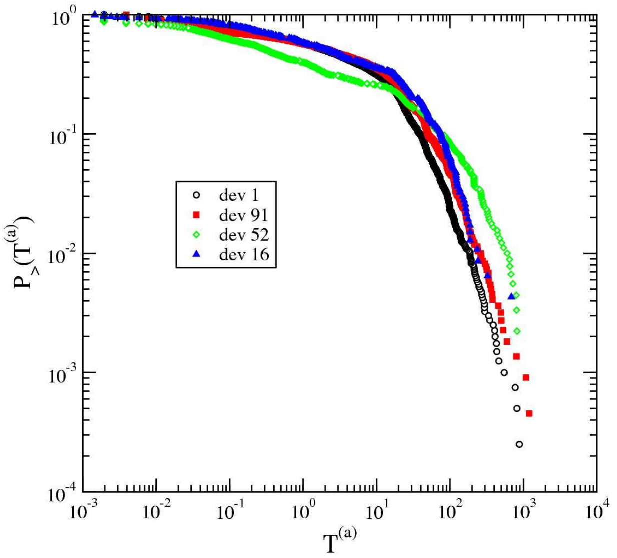


FIG. 15: Time interval cumulative distributions of the four most active developers of OpenBSD.

V. FILES

The source files are not modified at a constant rate after their creation, but less and less frequently on average. In other words, they age as they converge slowly to an acceptable state, with bursts of modifications from time to time. A way of visualising ageing is to plot the time interval between two modifications of a given file $T^{[f]}$ as a function of the modification number: $T^{[f]}$ tends to increase (see Fig 17) and display larger and larger spikes, suggesting once again a non-constant dynamics. The bursts of new activity are either due to the implementation of a new feature, or to a tentative bug fix; in the latter case, the long quiet period reflects the time needed to find and correct a bug. The clustered activity at the level of a single file is yet another clue of trial and error, or cascading modifications.

A better statistical characterisation of this process is done by plotting the cumulated number of modifications as a function of the time elapsed since the file's creation, $t_i^{[f]} - t_0^{[f]}$ (see Fig 18). If the rate of modification is constant, both quantities depend linearly from each other; if the rate of modifications slows down with time, the dependence is sub-linear. Fitting our datasets with two-parameter function $c(T_i - T_0)^z$, we found z ranging from 0.6 to 0.9: the cumulative number of modifications increase sub-linearly as a function of time (Fig 18), echoing a decreases of activity.

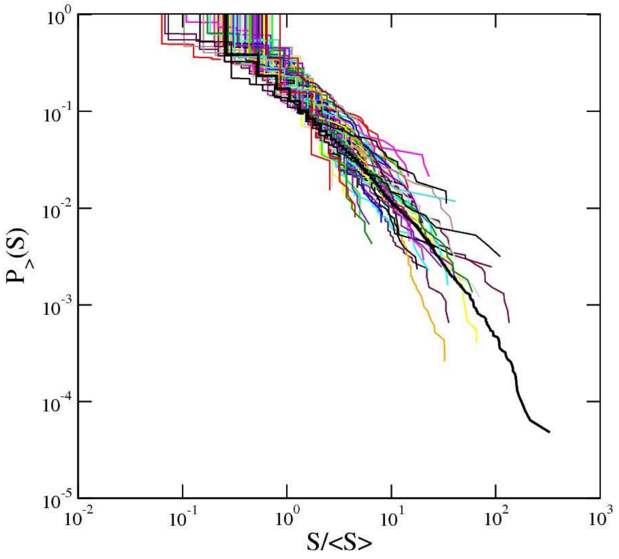


FIG. 16: Modification cumulative distributions of developers of OpenBSD, and OpenBSD itself.

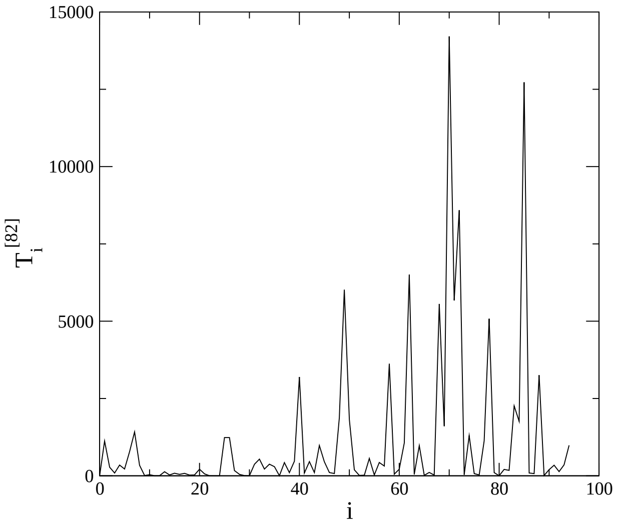


FIG. 17: Time interval between two modifications of file 82 of OpenBSD.

VI. CONCLUSION

In short, we have provided evidence that the process of software development does not follow a Poissonian process as often assumed in software engineering, but that it shares many properties with other kinds of human activity, be it submitting printing jobs, trading in financial markets, or answering emails and letters. Remarkably, our study suggests that software development does not belong to the universality classes previously reported in the literature. In addition, we wish to point out that open-source software provides most detailed data: contrarily to financial markets, one has full access to the most microscopic actions.

Our results point at the non-smooth trial-and-error processes that underly software projects: the correlations due to the interaction of programmers and to the structures of the software itself cause large fluctuations of both the time between two modification submissions and size of the modifications itself. Nevertheless, all the projects analyzed here have a remarkable degree of statistical regularity and reach a stationary, or mature, state

We thank Matthijs den Besten and Paul David for useful suggestions.

This work has been supported in part by the E.U. within the 6th Framework Program under contract 001907

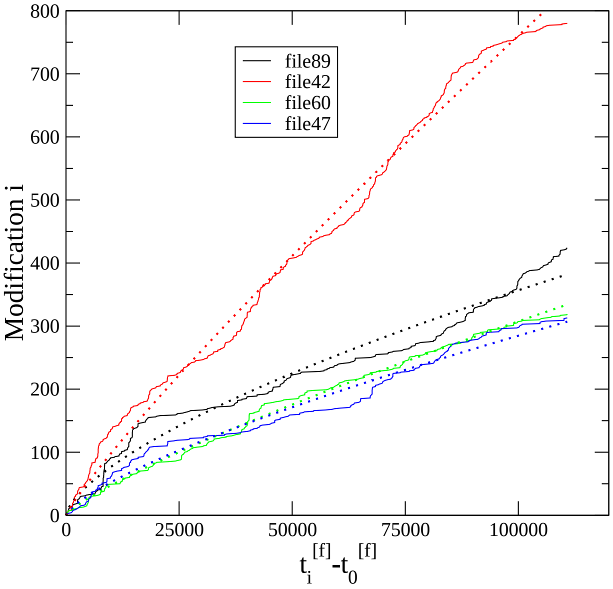


FIG. 18: Modification number versus time from file creation for various files of NetBSD. Dotted lines are best fits with $c(T_i - T_0)^{z'}$

(DELIS).

[1] A. Johansen. Response time of internauts. *Physica A*, 296:539–546, 2001.

[2] A.-L. Barabási. The origin of bursts and heavy tails in human dynamics. *Nature*, 2005.

[3] Alexei Vázquez et al. Exact results for the Barabási model of human dynamics. *Phys. Rev. Lett.*, 2005. preprint physics/0506126.

[4] Alexei Vázquez et al. Modeling bursts and heavy tails in human dynamics. 2005. preprint physics/0510117.

[5] Daniel B. Stouffer, R. Dean Malmgren, and Luis A. N. Amaral. ”comment on Barabasi, Nature 435, 207 (2005)”. 2005.

[6] A. Lukács B. Rácz I. Szakadát A.-L. Barabási Z. Dezso, E. Almaas. Fifteen minutes of fame: the dynamics of information access on the web. 2005. cond-mat/0505087.

[7] S. Valverde and R. V. Sole. Self-organization patterns in wasp and open source communities. *IEEE Intelligent Systems*, 21(2):36–40, 2006.

[8] R. D. Banker and S. A. Slaughter. The moderating effects of structure on volatility and complexity in software enhancement. *Inf. Syst. Res.*, 11(3):219–240, 2000.

[9] S. L. Chung T. Chan and T.-H. Ho. An economic model to estimate software rewriting and replacement times. *IEEE Trans. Soft. Eng.*, 22(8):580–598, 1996.

[10] J. Heales. A model of factors affecting an information system’s change in state. *J. Softw. Maint. Evol.: Res. Pract.*, 14:409–427, 2002.

[11] Sergi Valverde. Crossover from endogenous to exogenous activity in opensource software development. *Eur. Phys. Lett.*, 77:20002, 2007.

[12] S. Valverde, R. Ferrer-Cancho, and R. V. Solé. Scale-free networks from optimal design. *Europhys. Lett*, 60:512–517, 2002.

[13] S. Valverde and R. V. Sole. Logarithmic growth dynamics in software networks. *Europhys. Lett.*, 72, 2005.

[14] C. R. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Phys. Rev. E*, 68:046116, 2003. cond-mat/0305575.

[15] Damien Challet and Andrea Lombardoni. Bug propagation and debugging in asymmetric software structures. *Phys. Rev. E*, 70:046109, 2004.

[16] M. Borst B. S. Curtis, P. Milliman and T. Love. Measuring psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Trans. Soft. Eng.*, 5(2):96–104, 1997.

[17] K. Clark and S. Wheelwright. *Managing product and process development*. The Free Press, New York, 1998.

[18] `http://libresoft.urjc.es/cvsanaly/`.

[19] A. Gorshenev and Yuri Pis’mak. Punctuated equilibrium in software evolution. *Phys. Rev. E*, 70:067103, 2004.

[20] R. T. Fielding A. Mockus and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Meth.*, 11(3):309–346, 2002.

[21] Jingwei Wu, Richard C. Holt, and Ahmed E. Hassan. Empirical evidence for soc dynamics in software evolution. *IEEE*

conference on Software Maintenance, ICSM 2007., pages 244–254, 2007.

[22] J. Laherrre and D. Sornette. Stretched exponential distributions in nature and economy: fat tails with characteristic scales. *Eur. Phys. J. B*, 2(4):525–539, 1998.

[23] Uli Harder and Maya Paczuski. Correlated dynamics in human printing behavior. 2004. preprint cs.PF/0412027.

[24] Jaume Masoliver, Miquel Montero, and George H. Weiss. Continuous-time random-walk model for financial distributions. *Phys. Rev. E*, 2003. cond-mat/0210513.

[25] Z. Eisler and J. Kertész. Size matters: some stylized facts of the stock market revisited. *Eur. Phys. J. B*, 51:145–154, 2006.

[26] F. Lillo, S. Mike, and J. D. Farmer. A theory for long memory in supply and demand. *Phys. Rev. E*, 71:66122, 2005.

[27] S. Valverde and R. V. Solé. Hierarchical small worlds in software architecture. *Dynamics of Continuous Discrete and Impulsive Systems: Series B: Applications and Algorithms*, 14:1–11, 2007.

[28] This is in line with the qualitative study of Ref. [20]

[29] We used both Hill estimator and direct fitting.

APPENDIX A: EXPONENTS

Program	Number of points	α	β	γ	δ
Apache	12992	0.58	0.45 ± 0.06	2.5	?
FreeBSD	105843	0.61	0.62±0.01	2.0	0.78
Mozilla	154852	0.48	0.44±0.02	2.0	0.48
NetBSD	95568	0.57	0.57±0.02	2.0	?
OpenBSD	62347	0.58	0.54±0.02	2.0	?
PostgreSQL	17934	0.59	0.60±0.06	?	?