Empirical Evidence for SOC Dynamics in Software Evolution

Jingwei Wu
Oracle Corporation
400 Oracle Parkway
Redwood City, CA, USA
jingwei.wu@oracle.com

Richard C. Holt
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
holt@plg.uwaterloo.ca

Ahmed E. Hassan
Electrical & Computer Engineering
University of Victoria
Victoria, BC, Canada
ahmed@ece.uvic.ca

Abstract

We examine eleven large open source software systems and present empirical evidence for the existence of fractal structures in software evolution. In our study, fractal structures are measured as power laws throughout the lifetime of each software system. We describe two specific power law related phenomena: the probability distribution of software changes decreases as a power function of change sizes; and the time series of software change exhibits long range correlations with power law behavior. The existence of such spatial (across the system) and temporal (over the system lifetime) power laws suggests that Self-Organized Criticality (SOC) occurs in the evolution of open source systems. As a result, SOC may be useful as a conceptual framework for understanding software evolution dynamics (the cause and mechanism of change or growth). We also discuss the implications of SOC to software practices.

1 Introduction

The laws of software evolution proposed by Lehman [25] represent a major intellectual contribution to understanding software evolution dynamics, *i.e.*, the underlying cause and mechanism of software change or growth. Lehman's eight laws are empirically grounded on observing how a number of closed source industrial software systems such as IBM OS/360 were developed and maintained within a single company using conventional management techniques [27]. The laws suggest that software systems must be continuously adapted to respond to external forces such as new functional requirements and hardware upgrade and to maintain stakeholder satisfaction.

Viewing software evolution as a phenomenon driven by various external forces can make it difficult to accommodate and explain conflicting findings found in different software systems or domains. For example, open source projects in different domains have been found to grow at different rates including super-linear, sub-linear and linear rates [15, 41]. A simple, unified way for explaining evolution dynamics is needed. This paper looks into *Complex Systems* theories to explore new ways for describing and explaining software evolution.

A wealth of knowledge has been gained in understanding the change behavior of complex systems as diverse as sandpiles [3], power blackouts [8], earthquakes [43], and even biological evolution [44]. All these systems are complex in the sense that no single characteristic event size can control their changes or responses over time. Changes in a complex system can be of any size and occur any time. For example, a power blackout can strike a street, a town, a state and all the way up to a country. An intriguing aspect of complex systems is that their statistical properties can be measured as power laws in space and in time [3].

In 1987, Bak *et. al* proposed *Self-Organized Criticality* (SOC) to explain the existence of ubiquitous fractal structures in nature [3]. SOC has two important signatures: (1) the power law distribution of dynamical responses, and (2) long range correlations with power law behavior in time series of response [8, 44]. Simply speaking, large numbers of small changes in a SOC system may be separated occasionally by few large avalanche changes. This seems to apply to software systems where small changes are made constantly and substantial changes are made occasionally to alter the system architecture. Section 2 will provide a brief introduction to fractals, power laws and long range correlations and explain how they are relevant to SOC.

Does a software system follow the SOC dynamics during its evolution? Several previous studies suggest an intricate connection between SOC and software evolution dynamics. In particular, these studies include:

- Biological evolution as self-organized criticality

  Researchers have found that fluctuations in fossil record exhibit long range correlations with power law behavior [44]. The existence of such fractal structures means that, when examining a given time frame, some basic properties such as mean and standard variance remain the same as those obtained from the whole time series if a change of scale is performed. SOC is suggested as a useful way of understanding how long periods of small extinctions are interrupted by mass ex-

tinctions [2, 44]. The structural evolution of a software system exhibits similar characteristics of punctuation as fossil record, suggesting that SOC may also be useful for explaining software evolution.

- Software evolution as punctuated equilibrium

  Wu *et. al* examined the structural evolution of software systems at the level of source files. They observed that three open source systems (OpenSSH, PostgreSQL and Linux Kernel) evolved through an alternation between long periods of small changes and short periods of large avalanche changes [48].

- Overall open source movement as a self-organizing collaborative social network

  In comparison to traditional industrial systems, open source software systems are largely developed based on a less strict control and management model [35, 40]. Spontaneous collaboration is promoted and backed by a decentralized developer community across the Internet [40]. Researchers including Madey and Koch have suggested that open source projects can be seen as a self-organizing phenomenon featuring the self-selection of tasks, spontaneous collaboration and leadership [23, 29]. The main empirical evidence they have presented is the power law distribution of open source project sizes (the numbers of developers) and the power law distribution of developer contributions (the number of commits to the source control repository).

Motivated by these promising findings, we feel that SOC might be established as a useful conceptual framework for describing and explaining software evolution. In this paper, we describe our effort in seeking evidence for SOC in open source software systems and discuss potential implications of SOC to software practices.

The rest of this paper is organized as follows. Section 2 introduces some basic concepts. Section 3 explains empirical data collection with regard to software changes and time series of software change. Section 4 investigates the existence of power laws in the evolution of open source software systems. Section 5 discusses open source software evolution based on SOC. Section 6 discusses some threats to the validity of our work. Section 7 considers related work and Section 8 concludes this paper.

2 Background

This section gives a brief introduction to fractals, power laws, R/S time series analysis and SOC. The reader who is familiar with these concepts can skip to the next section.

2.1 Fractals

Fractals are mathematical or natural objects that are made of parts similar to the whole in "some" way. A fractal has a self-similar structure that occurs at different scales. For example, a small branch of a tree looks like the whole tree due to the existence of branching structures. When the length of a shoreline is measured using the box counting method, the length of any segment can cover the same number of mesh boxes as the whole shoreline if a change of scale is performed. For a detailed explanation of fractals, the reader can refer to Mandelbrot's book *Fractal Geometry of Nature* [30].

2.2 Power Law

A power law is a relationship between two scalar variables x and y, which can be written as follows:

$$y = C{\cdot}x^k$$

where C is the constant of proportionality and k is the exponent of the power law. Such a power law relationship shows as a straight line on a log-log plot since, taking logs of both sides, the above equation is equivalent to

$$log(y) = k{\cdot}log(x) + log(C)$$

which has the same form as the equation for a straight line

$$Y = k{\cdot}X + c$$

The equation $f(x) = C{\cdot}x^k$ has a property that relative scale change $f(sx)/f(x) = s^k$ is independent of x. In this sense, $f(x)$ lacks a characteristic scale or is scale invariant. Consequently, $f(x)$ can be related to fractals because of its scale invariance.

Power Law Distribution

This paper is concerned with a special kind of distribution called power law distribution, in which the Probability Density Function (PDF) of size s is specified as $P(s) \sim s^{-\alpha}$ and the tail Cumulative Distribution Function (CDF) of size s is specified as $D(s) = P(x{\geq}s) \sim s^{-\beta}$. The relationship between α and β is $\beta = \alpha - 1$ [33]. Because β can be conveniently estimated using linear regression on a log-log plot without binning data points, we choose to estimate β rather than α in our study.

Power laws have been observed in many fields such as physics, economics, geography and sociology [33]. For example, the distribution of earthquakes is found to follow $P(E){\sim}E^{-\alpha}$ where E is the amount of energy. The exponent α exhibits some geographical dependence and is found to be in the interval from approximately 1.8 to 2.2 [21]. The distributions of firm sizes (measured as the number of employees) [1, 13] and of open source project sizes (measured as the number of developers or lines of code) [18, 23] also follow power laws.

2.3 Time Series Analysis

Time series are commonly used to characterize the evolution of software systems. For example, Lehman *et al.* studied the evolution of IBM's operating system OS/360 by means of observing system growth measured in terms of the

number of source modules and number of modules changed for each release [6, 26]. Turski has performed a regression analysis of results from these case studies and proposed the inverse-square model, suggesting that system growth is inversely proportional to system complexity and that system complexity is proportional to the square of system size [46]. In this paper, we are interested in studying the fractal properties in time series of software change.

A widely used statistical analysis technique for time series is Rescaled Range Analysis, also referred to as the R/S statistic. It was first formulated by Hurst in 1951 [19]. The R/S statistic calculates the Hurst exponent H to reflect data persistence in a time series. Based on the value of H, natural and/or man-made temporal processes can be classified as follows:

- *Uncorrelated* if $H = 0.5$. A random walk is uncorrelated. Informally, one can think of that future events are not influenced by previous ones and also do not carry memory from the past.
- *Long term anti-correlated* if $H < 0.5$. A process from this category produces anti-persistent time series in which a value above the mean is more likely to be followed by a value below the mean and vice versa. Such behavior is observed in mean-reverting processes such as interest rate change [7].
- *Long term correlated* if $H > 0.5$. A processes from this category has long runs of consecutive values above or below the mean. The Nile River has $H = 0.91$ as calculated by Hurst [19]. This value implies that flood occurrences of the Nile River are not purely random but temporally dependent. Heavier floods are accompanied by above average flood occurrences and minor floods are followed by below average occurrences. Such phenomena are often referred to as long range dependence or long range correlations.

Different time series can be quantified by using the R/S analysis to estimate their associated Hurst exponents. Such a quantification allows us to study similarities between different temporal processes including software system evolution, thereby recognizing underlying unifications that might otherwise have gone unnoticed.

2.4 Self-Organized Criticality

In 1987, Bak *et. al* proposed *Self-Organized Criticality* (SOC) to explain widespread occurrences of spatial fractals and fractal time series (known as $1/f$ noise) in nature [3]. According to SOC, a complex system which consists of interacting components can exhibit some general characteristic behavior in space (self-similar fractals) and in time ($1/f$ noise) spontaneously. Such behavior can often be measured as power laws. More specifically, a SOC system can exhibit power laws in the distribution of its change sizes and long range correlations in the time series of its change history. The power law distribution and long range correlation are the two most important diagnostics of SOC [4].

To investigate whether software systems follow the SOC dynamics, we need access the change history of many software systems. We need recover software changes from the change history of these systems and obtain the time series of change. With such data, we will be able to examine and verify the existence of power law distributions and long range correlations throughout the evolution of many systems, revealing the SOC dynamics in software evolution.

3 Software Change Collection

This section describes how we have collected software changes throughout the lifetime of software systems.

3.1 Software Change

We denote a software change as a set of source files that are modified together for a purpose. For example, a change may contain a number of source files, which a programmer modified in order to remove a defect or to add a feature. We consider two kinds of change: *Logical Changes* recovered from source control repositories (*e.g.*, CVS) and *Structural Changes* obtained by contrasting subsequent snapshot versions (*e.g.*, releases). For the C and C++ programming language, we consider files with the following extensions: `.c`, `.C`, `.cc`, `.cpp`, `.cxx`, `.c++`, `.h`, `.H`, `.hh`, `.hpp`, `.hxx` and `.h++`. The size of a software change is denoted as the number of source files.

Logical Change

A logical change, which is recoverable from a source control repository, contains files checked in by the same developer with the same log message and at the same time. The term "same time" in this context means that files are committed in a short period. Zimmermann *et al.* described a number of methods for recovering logical changes from the CVS repository [49]. One of their methods is the sliding time window protocol, which relies on a maximal time gap to determine whether two subsequent checkins belong to one change. The change log tool *cvs2cl* uses such a protocol to recover changes from the CVS repository automatically [11]. We choose to use *cvs2cl* in our study. Recovered changes may be related to different types of task such as bug fix, feature modification, functional improvement and refactoring. For simplicity they are not differentiated in our study.

Structural Change

A structural change contains files which must satisfy the following requirements: (1) they have outgoing dependencies added or deleted; and (2) they are connected in an isolated subgraph within a delta graph obtained by contrasting two subsequent snapshot versions. As shown in Figure 1, a delta

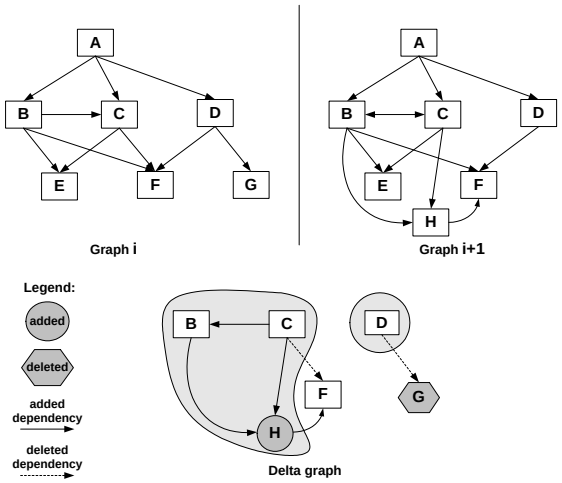


Figure 1: Recover Structural Changes

graph contains five files B, C, D, F, G and H as well as six added and deleted dependencies among these files. Files B, C, F and H form an isolated subgraph in which B, C and H have changed their outgoing dependencies but F does not. According to the definition, B, C and H form a structural change with F excluded. File D forms a structural change by itself.

We use CTSX [47], a robust and efficient source extractor for C and C++, to extract structural dependency graphs on a daily basis throughout the lifetime of a software system. Then, we compare consecutive graphs to recover software structural changes over time.

3.2 Time Series of Change

We use time series to record change fluctuations throughout the lifetime of a software system. We can obtain a time series on the basis of releases or on a daily basis when daily snapshots are available. Using R/S analysis we study the existence of long range correlations in time series of software change to see whether it represents the temporal signature of SOC. We apply R/S based time series on both logical and structural changes.

4 Fractals in Software Evolution

In this section we present empirical evidence for fractal structures found in the evolutionary history of eleven open source software systems. The evidence exists in the form of two fractal related phenomena:

- Power law distribution of change sizes
- Long range correlations in time series of change

We will first discuss in detail the empirical results obtained from GCC (GNU Compiler Collection) [14], then summarize empirical results from more software systems including NetBSD, FreeBSD, OpenBSD, Linux, PostgreSQL, KSDK, KOffice, PHP, OpenSSL and Ruby. Appendix A provides a brief introduction to these software systems.

4.1 Power Law Distribution of Change Sizes

In our analysis of distributions of change sizes for GCC, the quantity being plotted is $D(s)$, *i.e.*, the tail cumulative distribution function (see Section 2.2).

4.1.1 Distribution of Logical Changes in GCC

We recovered a total of 40,034 logical changes from the CVS repository of GCC. These changes span a development period of eight years from 1997/08/11 to 2005/09/09. Fig. 2(a) displays $D(s)$ on a log-log plot which follows approximately a straight line. An ordinary least squares (OLS) linear estimation on the logarithmic scale of base 10 yields the following:

$$D(s) \sim s^{-\beta},\ \beta = 1.3237$$

$D(s)$ has a property that relative change $D(ks)/D(s) = k^{-\beta}$ is independent of size s. $D(s)$ is thereby scale invariant and it can be seen as a self-similar object statistically. In GCC, a larger logical change occurs less frequently than a smaller one. Using the $D(s)$ equation, we can estimate how much rarer large changes are. In Fig. 2(a), the probability of a change occurrence that involves more than 100 source files is extremely low, roughly less than 0.16%.

We have also found that similar power law distributions hold for individual years from 1998 to 2005 in GCC. The obtained scaling exponents vary slightly from 1.29 to 1.34. This suggests that yearly distributions share a similar power law behavior with each other and also with the lifetime distribution.

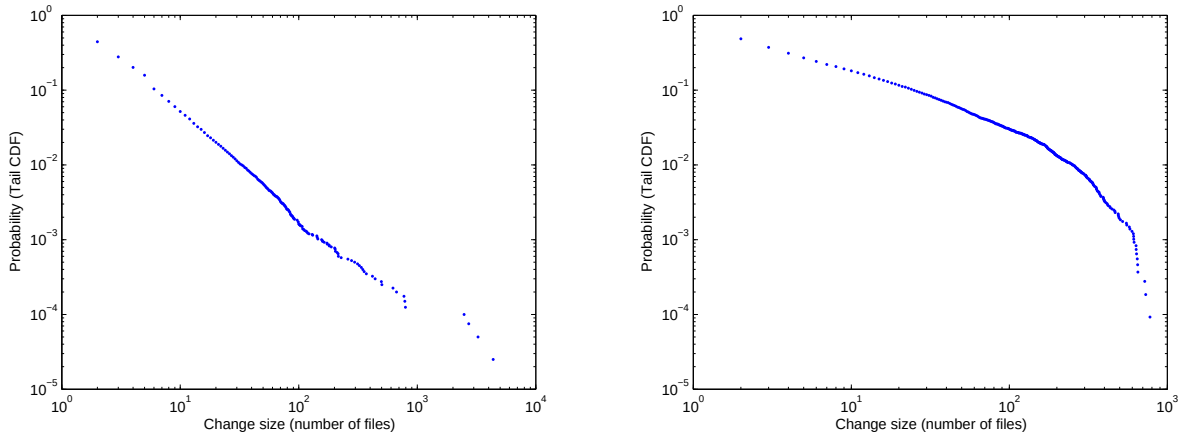
4.1.2 Distribution of Structural Changes in GCC

After observing that the size distribution of logical changes followed a power law relationship, we were curious to know whether other kinds of software change have a similar distribution. We examined the size distribution of daily structural changes over the same development period of GCC. A power law was observed by neglecting changes consisting of more than 100 files. It is shown as a log-log plot in Fig. 2(b). An OLS linear fit on the logarithmic scale gives the following:

$$D(s) \sim s^{-\beta},\ \beta = 0.7482$$

The exponent β is estimated with $1 \leq s \leq 100$. It is interesting that this distribution begins to deviate from power law roughly for $s > 100$. This means that a massive structural change involving more than 100 source files within one day is extremely rare and not governed by the power law. Such a large change may structurally depend on hundreds of other files in the system. It is clearly difficult to modify a substantial number of dependencies among several hundred files.

The long tail deviation from a power law might be due to the finite-size effect [4]. If the number of files in GCC



(a) Size distribution of logical changes ($\beta = 1.3237$)

(b) Size distribution of structural changes ($\beta = 0.7482$)

Figure 2: Tail Cumulative Distributions of Change Sizes for GCC

System	Period	Logical change (LC)			Structural change (SC)		
		#LC	β	R^2	#SC	β	R^2
NetBSD	1993/03/20–2005/08/17	86,280	1.3072	0.9952	–	–	–
FreeBSD	1993/06/06–2005/08/17	72,021	1.3435	0.9916	–	–	–
OpenBSD	1995/10/18–2005/08/17	47,969	1.1796	0.9963	–	–	–
Linux*	1994/03/13–2005/07/15	–	–	–	5,042	0.3420	0.9902
PostgreSQL	1996/07/09–2005/09/09	10,797	1.2866	0.9907	3,140	0.8573	0.9929
GCC	1997/08/11–2005/09/09	40,034	1.3237	0.9853	10,835	0.7482	0.9915
KSDK	1999/01/01–2004/09/15	4,012	1.4305	0.9851	1,112	0.7096	0.9655
KOffice	1999/01/01–2004/09/15	2,2948	1.4326	0.9899	19,913	0.5634	0.9624
OpenSSL	1999/01/01–2005/07/16	3,934	1.2989	0.9912	872	0.8208	0.9815
PHP	1999/04/07–2005/09/09	15,558	1.2749	0.9882	2,198	0.7701	0.9822
Ruby	1999/08/13–2005/09/09	3,655	1.5022	0.9935	443	0.9101	0.9227

$*$: The structural changes of Linux were obtained by comparing consecutive public releases over time.

Table 1: Scaling Exponents for Distributions of Software Changes

grows significantly in the future, we suspect that the deviation could appear at a larger size. We also observed similar deviations around $s = 100$ in several other open source software systems including KSDK, KOffice, OpenSSL, PHP, PostgreSQL and Ruby. In comparison to GCC, these systems have a smaller or roughly equivalent size. For Linux which is many times larger than GCC, the deviation appears around $s = 350$. Such deviations can be seen in log-log plots shown in Appendix B.

4.1.3 Power Distributions for More Systems

We examined ten more open source systems and found they followed power laws to varying degrees. The obtained scaling exponents are summarized in Table 1. The obtained log-log plots are shown in Appendix B.

We did not analyze distributions of structural change sizes for three BSD variants because each of them is actually composed of a large number of smaller applications and libraries. It is beyond the scope of this paper to study the evolution of structural dependencies among a collection of interacting applications. Unlike structural changes, logical changes are mostly within the boundary of each smaller application or library. Thus we only studied distributions of logical changes for three BSDs.

Because Linux does not have a source control repository such as CVS for public access, we were not able to analyze change log information or perform daily structural comparisons. We instead examined its structural evolution through comparing 524 releases (from 1.0 to 2.6.12.3). These releases were chosen and ordered according to release dates to create a historical sequence. This sequence contains both stable and experimental releases.

Here are some of our observations on the eleven studied open source software systems.

- All eleven software systems we studied have a power law distribution with respect to both logical and structural changes. The quality of fit (R^2) values shown in

Table 1 indicate a strong linear relationship between function $D(s)$ and change size (the number of source files) on the logarithmic scale. Such power law distributions of software changes are observed in different software domains, not limited to just one system. This presents empirical evidence for the spatial signature of SOC in software evolution.

- The distribution of logical changes has a larger scaling exponent than distribution of structural changes. This is limited to a size threshold, above which a large structural change can have a lower probability if compared to a logical change of the same size. This can be seen from Figure 2. The distribution in Fig. 2(b) deviates from the power law at larger sizes ($s > 100$) and drops toward probability zero. No structural change is found to involve more than 800 source files in GCC on a daily basis. This kind of deviation is apparently more common in the distribution of structural changes than in the distribution of logical changes.
- OpenBSD has a scaling exponent different from those of FreeBSD and NetBSD. This suggests that products from the same product family may exhibit slightly different behaviors. It perhaps is because that FreeBSD and NetBSD have a longer history and thus have more logical changes than OpenBSD. Further examination will be needed for understanding what causes such differences in β from system to system.
- For Linux, time intervals between any two adjacent releases vary between 5 days and 37 days except that release 2.3.99-pre9 is approximately 6 months away from release 2.4.0. Such a sampling frequency favors larger structural changes rather than smaller ones. As a result, Linux has the smallest scaling exponent.

4.2 Long Range Correlations in Time Series

The fundamental nature of software system evolution is change occurring spatially (across the system) and temporally (over the system lifetime). Our observation of scale invariance in distributions of change sizes leads us to wonder whether the evolution of a software system exhibits fractal structures in time, *i.e.*, long range correlations with power law behavior. As pointed out in Section 2.3, the R/S analysis can be used to analyze long range correlations in time series. To address the above question, we only need to determine whether a time series of software change has a Hurst exponent greater than 0.5 (the characteristic value of random noise).

4.2.1 R/S Analysis of GCC

The two R/S statistic plots shown in Figure 3 have Hurst exponents with $H = 0.7711$ for time series of logical change and $H = 0.6841$ for time series of structural change. These exponents are significantly above 0.5, thus indicating strong long range correlations in the evolution of GCC. The results can be verified by means of randomly shuffling the original time series to eliminate correlations and re-applying R/S analysis. For GCC, a random shuffling always results in a reduction of H towards 0.5.

4.2.2 R/S Analysis for More Systems

We performed the R/S analysis on the same set of systems examined in Section 4.1. We estimated Hurst exponents by considering time lags smaller than 365 (days). The largest time lag for Linux is 174 (releases). This is because Linux's time series were obtained by comparing subsequent releases but not daily snapshot versions. The time lag 174 accounts approximately for one third of 524 Linux releases we studied. The obtained Hurst exponents are shown in Table 2.

We also defined *Time Series Coverage* (TSC) as the ratio of the number of non-zero values to the total number of values in a time series. TSC measures how often changes occur over a system's lifetime. Table 2 shows that Ruby has the smallest TSC (14.7%) with regard to structural change activities, indicating that Ruby's structure was changed approximately once every seven days on average. Smaller systems such as KSDK, PHP and Ruby are less prone to logical and structural changes. Larger systems such as GCC, Linux, KOffice and three BSD variants tend to change every day. This is not surprising because more developers are usually involved in a larger project and change activities occur more frequently.

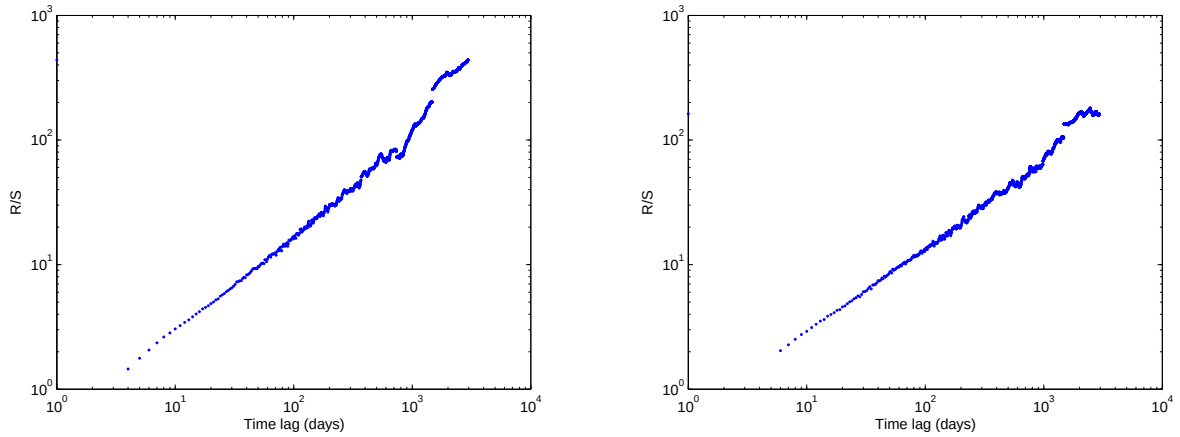
The H values obtained for all the time series of logical change vary between 0.7 and 0.8. This indicates that these time series appear approximately equally correlated in the long run. By contrast, all the H values obtained for structural change have a wider span from 0.6 to 0.8 with Ruby excluded. The Hurst exponent for Ruby is 0.5129, a close indicator of random noise. This perhaps is because the time series of Ruby is too sparse (TSC $= 14.7\%$) to yield notable correlations.

4.3 Summary

We presented evidence for the existence of fractal structures in the evolution of eleven large open source software systems. The fractal structures are measured as power laws in space (across the system) and in time (over the system lifetime). Both logical and structural changes follow power laws. However, logical changes yield stronger indications than structural changes.

5 Discussion

The eleven open source systems we examined are from different domains and have evolved over many years. However, their change dynamics share a common power law behavior in space and in time and appear to be independent of the individual details of each system. In order to understand such scale free dynamics we need to take into consid-



(a) R/S for time series of logical change ($H = 0.7711$)

(b) R/S for time series of structural change ($H = 0.6841$)

Figure 3: R/S Analysis of Daily Time Series for GCC

System	Logical change			Structural change		
	TSC	H	R^2	TSC	H	R^2
NetBSD	98.6%	0.7340	0.9984	–	–	–
FreeBSD	96.4%	0.7586	0.9952	–	–	–
OpenBSD	96.5%	0.7181	0.9962	–	–	–
Linux*	–	–	–	96.2%	0.7491	0.9893
PostgreSQL	80.5%	0.7637	0.9969	52.1%	0.7029	0.9972
GCC	98.9%	0.7711	0.9973	84.8%	0.6841	0.9964
KSDK	53.5%	0.8096	0.9811	33.0%	0.7909	0.9872
KOffice	96.6%	0.8092	0.9921	90.3%	0.6967	0.9967
OpenSSL	56.6%	0.7354	0.9941	26.5%	0.7163	0.9894
PHP	94.7%	0.7545	0.9968	59.2%	0.6186	0.9948
Ruby	61.2%	0.6980	0.9936	14.7%	0.5129	0.9864

$*$: The time series of Linux was obtained by comparing consecutive releases over time.

Table 2: Hurst Exponents from R/S Analysis of Daily Time Series

eration developers behavior and development processes in open source software development.

In large open source projects, developers commonly collaborate with one another for some common purposes spontaneously and they have freedom to modify and redistribute the source code and to work on the segments of their own interest. A central organization in the name of core group or steering committee [5] often exists. It plays an important role in providing general development guidelines and road maps, anchoring a broader community of developers and users, and nurturing leadership and collaboration across the community [17]. However, such an organization does not control or command what and how an individual developer should modify the source code. Spontaneous collaboration activities between developers ensure the delivery of working releases and sustain the future evolution of the system. Such spontaneous self-organizing behavior of open source developers shall not be confused with Lehman's third law of software evolution, *Self-Regulation* [25]. Self-regulation is a control notion suggesting that both positive and negative feedback controls are constantly and pervasively exercised during a system's evolution. By contrast, self-organization is a configurational notion indicating spontaneous developer collaboration which is neither entirely directed by a central organization nor prescribed by a published software process guideline [20].

Agile Software Development

Agile Software Development is a conceptual framework for undertaking software engineering projects with the help of lightweight methodologies such as Extreme Programming [10] and Adaptive Software Development [17]. Generally speaking, agile methodologies value

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The four core values are regarded as the canonical definition of agile software development and commonly referred to as *Agile Manifesto* [31].

Highsmith has advocated adaptive software development as an alternate approach between the monumental and accidental software development models in today's turbulent e-business world [17]. His idea is drawn from theories for complex adaptive systems (including SOC). He emphasizes adaptability, speed and collaboration as the key elements to the success of software project teams who develop and manage high-speed, high-change and high-uncertainty projects in real life. Other agile methodologies share a similar idea.

Agile software development has grown in recognition of complex systems theories which value adaptation over prediction and/or optimization [31]. The agile manifesto was summarized by many agile software practitioners based on their hands-on experience in developing and managing large numbers of real life projects over years. The fractal related phenomena we observed in open source software evolution provide empirical evidence for the existence of SOC. This can lend a hand to agile software practitioners in justifying their methodologies.

Software Change Propagation

If an open source software system follows the SOC dynamics during its evolution, can one foresee the extent to which changes propagate through the system? According to SOC, a complex system is evolving at the edge between chaos and order where no single characteristic event size can control the evolution of the system [21]. For instance, the electricity power grid is postulated to operate in such a narrow region where power blackouts can not be limited to a certain small size such as a street block or a small town [8]. It is difficult to predict where a power blackout can occur and how far the blackout can propagate.

As a software system evolves to respond to the changing environment and requirements, changes of varying sizes occur at different locations within the system. As indicated by the power law distributions we observed in open source systems, a software change can be up to any sizes, from one source file to the entire system. The occurrences of change covering a significant part of a system or even the entire system, though rare, appear inevitable in open source projects. Necessary measures such as the Collective Code Ownership [34] should be adopted to facilitate communication and collaboration between developers, preparing them for a large unexpected change.

6 Validity Threats and Limitations

There are several threats to the validity of our work.

- The eleven systems we studied are all successful, large open source software systems. There are a large number of small open source projects which neither attract many developers nor are maintained actively over a long period of time. Therefore, the systems we studied are not representative of small or failed open source projects.
- It is unknown whether closed source industrial systems exhibit similar power laws in space and in time. This needs to be empirically checked through future work.
- The definition of a structural change is affected by how one interprets changes to the system structure and how frequently structural snapshots are captured. Furthermore, CTSX that we used to extract program structural dependencies is inaccurate, especially in handling C++ programs. This perhaps explains why KOffice implemented in C++ does not to follow power laws during its structural evolution (see Fig. 4(f)).
- A number of simulation models for SOC [2, 44] can be adapted to simulate software evolution and verify our empirical findings from a theoretical point of view. For example, Cook *et. al* have presented a conditional growth model to reveal the existence of SOC in software growth [9]. By adapting their model to software changes, we might be able to obtain a theoretical proof of SOC in software evolution.

7 Related Work

Several researchers have studied power law distributions related to open source projects hosted on the SourceForge Web site [45]. The mission of SourceForge is to enrich the open source community by providing a centralized place for developers to manage open source projects.

Hunt and Johnson studied the downloads of open source projects at SourceForge and found that projects sizes (measured as the number of software downloads) follow a rank-based power law distribution [18]. According to their study, there are a small number of exceptionally popular projects such as Linux, Apache and Mozilla while most SourceForge projects are less popular. They suggest that studying median size projects might be useful for identifying best practice for developing open source software.

Madey *et al.* also reported similar power law distributions about open source projects [29]. They choose to measure project sizes using the number of developers. Their results indicate that most open source projects at SourceForge have only one developer and only a small percentage have a larger ongoing team. In addition, they modeled open source movement as a collaborative social network with developers as nodes and joint membership in projects as links between nodes. The clusters (development teams) in the social network are connected by linchpin nodes which are developers playing an important role in transferring ideas and technology between different development teams. Madey reported that cluster sizes also followed power laws.

Koch analyzed individual programmers' contribution to open source projects at SourceForge and also found power laws [23]. Koch measured the contribution of a programmer using the number of commits made by the programmer or the number of projects the programmer worked on. The observed power laws indicate that a minority of programmers are in fact responsible for the major growth of open source projects. In addition, the collocation of projects in a virtual hosting environment such as SourceForge does not significantly increase co-participation over different projects.

These empirical studies model the open source software community as an open ecology in which individual programmers collaborate with one another, ideas are nurtured, and projects are delivered. Madey and Koch have suggested that the overall open source software development is a self-organizing system within which spontaneous collaboration as well as leadership (*e.g.*, chief programmer) contribute to the success of open source projects. By contrast, we focused on individual systems instead of cross-project behavior. We found power laws throughout the evolution of several open source systems and suggested that open source systems follow the SOC dynamics at the level of source files.

The work of Gorshenev and Pis'mak on explaining software evolution dynamics based on SOC perhaps is the most relevant to our work [16]. They have observed that distributions of added lines (or deleted lines) follow power laws in three open source systems which are Mozilla, FreeBSD and Emacs. We examined logical and structural changes at the source file level and observed power laws in distributions of change sizes as well as in time series of change.

It is worth to note that power laws have been observed at three levels of abstraction for open source software systems: the project level [18][23][29], the file level (our work), and the level of Lines of Code (LOC) [16]. These power laws make it promising to explain the evolution of open source communities as well as the evolution of individual projects using SOC. A unified framework built on SOC may be constructed and then enhanced in the future, within which software evolution may be better explained and understood.

In comparison to prior work, our work showed evidence for the existence of power laws in multiple open source software systems across different domains. We presented such evidence not only for logical changes but also for structural changes. Furthermore, we validated the existence of long range correlations in time series of software change, which is missing from prior work.

8 Conclusion

We have presented empirical evidence for the existence of fractal structures in the evolution of open source software systems. Fractal structures are identified and measured as power laws in space (across the system) and in time (over the system lifetime). Specifically, our findings are presented in the form of power law distribution of change sizes and long range correlations in time series of change. Such spatial and temporal fractal structures suggest that open source software systems do follow the SOC dynamics through their evolution.

However, it does remains a challenging task to establish SOC as a useful framework for understanding software evolution and maintaining practical software systems. We hope that our findings will spur further interests in studying SOC-related phenomena in software evolution and bridging the gap between SOC formulae and software practice.

References

[1] R. L. Axtell. Zipf distribution of U.S. firm sizes. *Science*, 293:1818–1820, September 2001.

[2] P. Bak and K. Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71(24):4083–4086, December 1993.

[3] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of $1/f$ noise. *Pysical Review Letters*, 59(4):381–384, July 1987.

[4] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality. *Pysical Review A*, 38(1):364–374, July 1988.

[5] A. Bauer and M. Pizka. The contribution of free software to software evolution. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 170–179, Helsinki, Finland, September 2003.

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

[7] J. Beran. *Statistics for Long-Memory Processes*. Chapman and Hall, 1994.

[8] B. A. Carreras, D. E. Newman, I. Dobson, and A. B. Poole. Evidence for self-organized criticality in a time series of electric power system blackouts. *IEEE Transactions on Circuits and Systems I*, 51(9):1733–1740, September 2004.

[9] S. Cook, R. Harrison, and P. Wernick. A simulation model of self-organising evolvability in software systems. In *Proceedings of the IEEE International Workshop on Software Evolvability*, pages 17–22, Budapest, Hungary, September 2005.

[10] Extreme Programming: A Gentle Introduction. Website: http://www.extremeprogramming.org/, 2004.

[11] K. Fogel, M. O'neill, and M. J. Pearce. CVS log message to change log message conversion script. Website: http://www.red-bean.com/cvs2cl/, 2002.

[12] FreeBSD. Website: http://www.freebsd.org, 2004.

[13] E. Gaffeo, M. Gallegati, and A. Palestrini. On the size distribution of firms: Additional evidence from G7 countries. *Physica A*, 324:117–123, 2003.

[14] GCC. Website: http://gcc.gnu.org, 2002.

[15] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *Proceedings of the 16th International Conference on Software Maintenance*, pages 131–142, San Jose, California, October 2000.

[16] A. A. Gorshenev and Yu. M. Pis'mak. Punctuated equilibrium in software evolution. *DOI: cond-mat/0307201*, 2003.

[17] J. A. Highsmith. *Adaptive Software Development – A Collaborative Approach to Managing Complex Systems*. Dorset House, New York, NY, USA, 1999.

[18] F. Hunt and P. Johnson. On the Pareto distribution of Sourceforge projects. In *Proceedings of Open Source Software Development Workshop*, pages 122–129, Newcastle, UK, 2002.

[19] H. E. Hurst. Long-term storage capacity of reservoirs. *Transactions of American Society of Civil Engineers*, 116:770–799, 1951.

[20] C. Jensen and W. Scacchi. Simulating an automated approach to discovery and modeling of open source software development. In *Proceedings of Software Process Simulation and Modeling Workshop*, Portland, OR, USA, May 2003.

[21] H. J. Jensen. *Self-Organized Criticality - Emergent Complex Behavior in Physical and Biological Systems*. Cambridge University Press, 1998.

[22] KDE. Website: http://www.kde.org, 2004.

[23] S. Koch. Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2):77–88, July 2004.

[24] KOffice. Website: http://www.koffice.org, 2004.

[25] M. M. Lehman. Laws of software evolution revisited. *Lecture Notes in Computer Science*, 1149:108–124, 1997.

[26] M. M. Lehman and L. A. Belady. *Program Evolution – Processes of Software Change*. Academic Press, UK, 1985.

[27] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution - the nineties view. In *Proceedings of the 4th International Software Metrics Symposium*, pages 20–32, Albuquerque, NM, November 1997.

[28] Linux Kernel. Website: http://www.kernel.org, 2004.

[29] G. Madey, V. Freeh, and R. Tynan. The open source software development phenomenon: An analysis based on social network theory. In *Proceedings of Americas Conference on Information Systems*, pages 1806–1813, Dallas, TX, 2002.

[30] B. B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman & Company, 1982.

[31] Manifesto for Agile Software Development. Website: http://agilemanifesto.org/, 2001.

[32] NetBSD. Website: http://www.netbsd.org, 2002.

[33] M. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, Sept. 2005.

[34] M. E. Nordberg. Managing code ownership. *IEEE Software*, 20(2):26–33, January 2003.

[35] Open Source Definition. Website: http://www.opensource.org/docs/definition.php, 2005.

[36] OpenBSD. Website: http://www.openbsd.org, 2004.

[37] OpenSSL. Website: http://www.openssl.org, 2004.

[38] PHP. Website: http://www.php.net, 2004.

[39] PostgreSQL. Website: http://www.postgresql.org, 2003.

[40] E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly and Associates, 1999.

[41] G. Robles, J. J. Amor, J. M. Gonzléz-Barahona, and I. Herraiz. Evolution and growth in large libre software projects. In *Proceedings of the International Workshop on Principles of Software Evolution*, Lisbon, Portugal, September 2005.

[42] Ruby. Website: http://www.ruby-lang.org, 2004.

[43] C. H. Scholz. *The Mechanics of Earthquakes and Faulting*. Cambridge University Press, 1990.

[44] R. V. Solé, S. C. Manrubia, M. Benton, and P. Bak. Self-similarity of extinction statistics in the fossil record. *Nature*, 388(21):764–767, August 1997.

[45] SourceForge. Website: http://sourceforge.net/, 2005.

[46] W. M. Turski. The reference model for smooth growth of software systems revisited. *IEEE Transactions on Software Engineering*, 228(8):814–815, August 2002.

[47] J. Wu and R. C. Holt. A program extractor suite for c and c++: Choosing the right tool for the job. Technical Report CS-2006-17, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, May 2006.

[48] J. Wu, C. W. Spitzer, A. E. Hassan, and R. C. Holt. Evolution spectrographs: Visualizing punctuated change in software evolution. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 57–66, Kyoto, Japan, September 2004.

[49] T. Zimmermann and P. Weißgerber. Preprocessing CVS data for fine-grained analysis. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR)*, pages 2–6, Edinburgh, UK, May 2004.

A Open Source Systems Studied

BSD is the UNIX derivative distributed by University of California at Berkeley. The development of BSD at Berkeley ceased after 4.4BSD-Lite was delivered in 1995. Since then, several distributions based on 4.4BSD-Lite have been in active development. **FreeBSD** is developed as a complete operating system with the kernel and miscellaneous utilities included [12]. **NetBSD** is a unified, portable, production-quality operating system [32]. It is often used in embedded systems. **OpenBSD** focuses on open source and documentation, standardization and security [36].

GCC [14] stands for GNU Compiler Collection and it has a set of compilers for C, C++, Objective C, Fortran, Java and Ada, as well as libraries for these languages. GCC is the key component of the GNU tool chain.

KOffice [24] is an integrated office suite developed for the K Desktop Environment (KDE). KOffice is part of the KDE Project and consists of 12 main applications: KPresenter, KWord, KChart, KSpread, Kivio, Karbon14, Krita, Kugar, KPlato, Kexi, KFormular and Filters that permit KOffice to interoperate with other popular office suites such as OpenOffice and Microsoft Office.

KSDK [22] is a software development toolkit designed for the KDE project. KSDK offers a collection of tools for developing and debugging various kinds of KDE applications.

Linux [28] is a cloned operating system UNIX, written from scratch by Linus Torvalds and then subsequently worked on by hundreds of developers who are loosely connected through the Internet. It aims towards POSIX and Single UNIX Specification compliance. Linux 1.0 was delivered in March 1994.

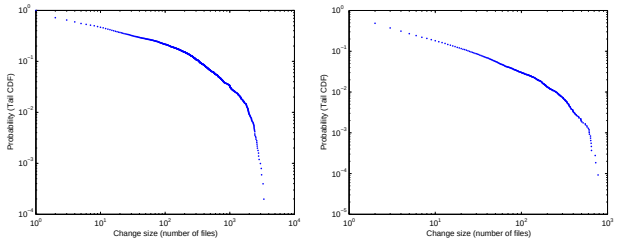
OpenSSL [37] is a cryptography toolkit implementing the Secure Sockets Layer (SSL 2.0/3.0) and Transport Layer Security (TLS 1.0) protocols and a general purpose cryptography library.

PostgreSQL [39] is a popular SQL-compliant object-relational database management system (DBMS). It has more than 15 years history and a globally distributed development team.

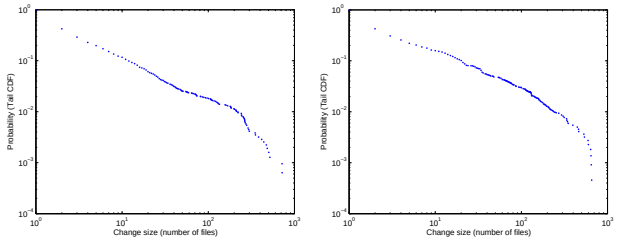
PHP [38], short for "the Hypertext Preprocessor", is a reflective programming language which is widely used for developing server-side applications and dynamic web content.

Ruby [42] is an interpreted scripting language for quick and easy object-oriented programming. It has many convenient features for text file processing and system management.

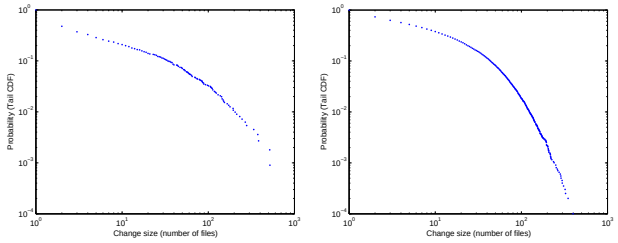
B Distributions of Change Sizes



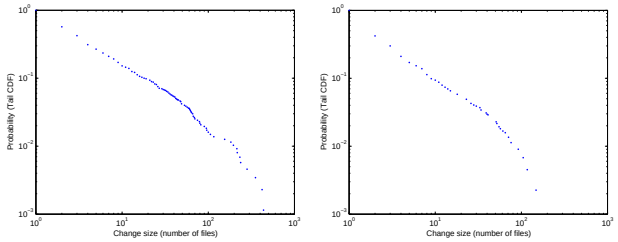
(a) Linux Kernel (b) GCC



(c) PostgreSQL (d) PHP

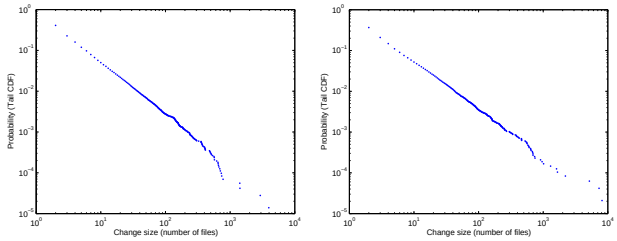


(e) KSDK (f) KOffice

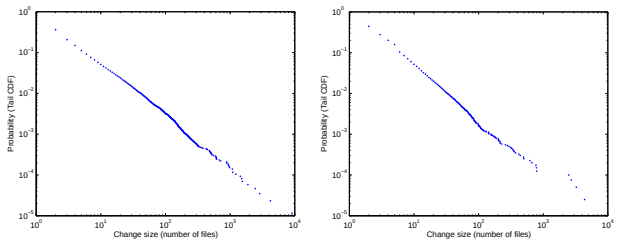


(g) OpenSSL (h) Ruby

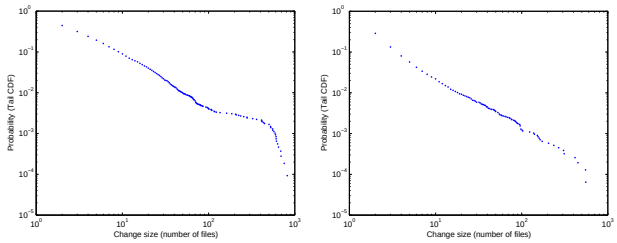
Figure 4: Tail CDF of Structural Changes



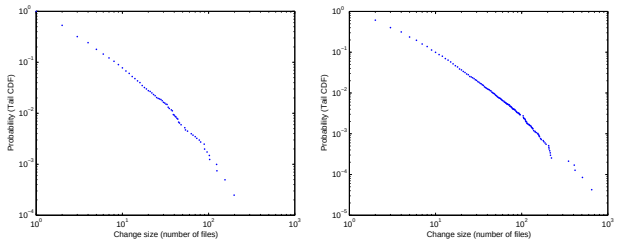
(a) FreeBSD (b) OpenBSD



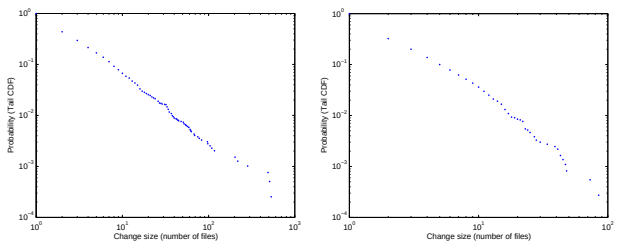
(c) NetBSD (d) GCC



(e) PostgreSQL (f) PHP



(g) KSDK (h) KOffice



(i) OpenSSL (j) Ruby

Figure 5: Tail CDF of Logical Changes