Письма в ЭЧАЯ. 2008. Т. 5, № 3(145). С. 353–361



# SCALING LAWS IN EVOLUTION OF LARGE COMPUTER PROGRAMS

*A. A. Gorshenev, Yu. M. Pis'mak*

Department of Theoretical Physics, State University of Saint-Petersburg, Russia

An approach based on a paradigm of self-organized criticality is proposed for experimental investigation and theoretical modelling of software evolution. The dynamics of modifications is studied for three free open source programs Mozilla, Free-BSD and Emacs using the data from version control systems. Scaling laws typical of self-organization criticality are found. The model of software evolution presenting the natural selection principle is proposed. The results of numerical and analytical investigation of the model are presented. They are in good agreement with the data collected for the real-world software.

Для экспериментального исследования и теоретического моделирования эволюции программного обеспечения предложен подход, основанный на парадигме самоорганизованной критичности. Динамика модификаций изучена для трех программ с открытым кодом: Mozilla, Free-BSD и Emacs на основе данных VCS-системы. Обнаружены типичные для самоорганизованной критичности скейлинговые законы. Построена модель эволюции программного обеспечения на основе принципа естественного отбора. Представлены результаты ее численного и аналитического исследования. Они находятся в хорошем согласии с имеющимися для реальных компьютерных программ данными.

PACS: 02.70.-C; 02.60.Pn

## INTRODUCTION

In recent time there have been a lot of attempts to reveal basic evolutionary mechanisms and universal laws of elementary interaction forming dynamics in real complex systems. The construction of theory of self-organized criticality (SOC) is the essential achievement of these investigations [1]. It was shown that the behaviour of many «in a natural way» arisen systems can be qualitatively described in the framework of simple mathematical models [2, 3, 5]. It makes it possible to understand inherent dynamical features of the complex system being important both for theoretical considerations and for practical applications [4–7].

In this paper we present based on the SOC-paradigm approach of investigation of evolution of large computer program [8]. This process is a good example of dynamics formed by natural selection in complex system, and the study of it is a vital topic nowadays [10].

Large software systems must evolve constantly or they risk losing the market share. However, maintaining of such a system is extraordinary difficult, complicated and time consuming. The task of adding new features, adding support for new hardware devices and platforms, system tuning, and defect fixing becomes more difficult as a system ages and grows [10, 12].

Despite of the fact that the first papers of software evolution study are now decades old, the basic mechanisms of evolution of computer programs are unclear. Most of the existing research is directed on quantitative description of changes in a system, and the question about underlying mechanism is left aside [12, 13]. In this paper we propose a formal model of software evolution that would simulate at least qualitatively the most essential features of processes in the real-world software systems.

The structure of the paper is as follows. Section 1 is a short review of the state of the field. In Sec. 2 we describe the experimental techniques and the results of its application to three real-world systems. In Sec. 3 we propose the model of software evolution and give the results of computer simulation. In Sec. 4 we present analytical results obtained for the model with random neighbor interactions.

## 1. GENERAL PROPERTIES OF SOFTWARE CHANGES

The source code of the majority of existing computer programs changes. There are a lot of reasons to change a program. The basic motives are as follows [9, 14]: defect correction, adding a new feature or possibility, adding new platform or device support, system tuning cosmetic changes.

The first item in this list had a lot of attention attracted. There is a bunch of research and papers dedicated to fixing of the bug area. However (and the situation has not changed essentially for the last thirty years), the search and correction of defects is still an actual problem [9, 10].

A multitude of research papers propose some kinds of statistical methods and metrics to ease the task of search and correction of defects and to answer the question about quality of computer software [16, 17]. The term «defect» can be defined in different ways. The usual definition is a deviation from specifications or expectations [9].

Generally, efforts have tended to concentrate on the following three problem perspectives: predicting the number of defects in the system; estimating the reliability of the system in terms of time to failure; understanding the impact of design and testing processes on defect counts and failure densities [9].

The majority of work on predicting the number of defects are based on some kind of metrics describing complexity, size, volume, etc., of a system. First research in this area began in early seventies. Halstead [17] proposed a number of complexity metrics. From our point of view, the main disadvantage of such an approach is that taking into account changes going on due to defect correction one loses changes occurred because of other reasons. Even the best in the world static metrics that predicts a number of improvements for a program to correspond a specification becomes useless if the specification changes in time essentially.

Lehman et al. have built the largest and best known body of research on software evolution of large, long-lived software systems [10, 11]. Lehman's laws of software evolution suggest that as system grows in size, it becomes increasingly difficult to add new code unless explicit steps are taken to reorganize the overall design. There were some systems examined both at system level and within the top-level subsystems [12, 13]. It has been noted that subsystems can behave quite differently from the system as a whole. In [14, 15] the «code decay» metaphor has been proposed to describe the continuous process that makes the software more brittle over time.

To study software evolution processes it is necessary to have an information about the state of the system in different moments of time. The usual source of this knowledge is different versions or releases of a product [10, 12, 13]. Unfortunately, the number of releases seldom exceeds a couple of tens. It is significantly decreases our possibility to study the dynamics of software evolution. The better sources of information about changes in computer programs are version control systems such as CVS, Teamware/SCCS, etc. These systems keep information about changes that happened in much frequent moments of time.

Following [12] we should note that the majority of research in the software evolution area done with closed systems, developed using traditional process and management. So the results of such a research cannot be reproduced or denied by independent research (which is one of the basic principles of scientific work). And even more: sometimes the published data contain intentional distortions to keep commercial secrets.

In the last couple of decades, due to Free Software movement success we have perfect possibility to experimentally study software evolution. There are a lot of open source projects that are successfully developing through years. Some of these projects we used in our present work as experimental material [18–20].

One of the unclear topics in software engineering is an existence of «abnormally» large fluctuations during the development of software. In spite of using a variety of software development technologies and development process improvements, large projects encounter the necessity to rewrite large pieces of code, comparable with the size of the system as a whole.

In physical systems large fluctuations are distinctive feature of the so-called critical behaviour. Critical phenomena arouse in the point of second order phase transitions. Developed turbulence is an example of critical dynamics. SOC is called the critical regime of dynamics arising without fine tuning of system parameters. The SOC dynamics is peculiar to many complex systems in the Nature [1], therefore we started our investigations with checking of hypothesis that evolution or real-world software should be a SOC process.

## 2. EXPERIMENTAL STUDY OF SOFTWARE CHANGES

Since most important features of critical behaviour are scale invariance and universality, the aim of our experimental investigation was to reveal them in evolution of computer programs. In our work we studied the histories of such software projects as Mozilla web browser, Free-BSD Operating System and Gnu Emacs text editor [18–20]. For each of these projects we studied only files written in basic for the project language. For Mozilla it is C++, for Free-BSD it is C and for Emacs it is Lisp. Header files for C/C++ were not studied. Total amount of files processed is 9000, 11000, 900 approximately for Mozilla, Free-BSD, Emacs in order. Total length of RCS-files constitutes $1 \cdot 10^7$, $1 \cdot 10^7$ and $2 \cdot 10^6$ lines. Total amount of data processed exceeds 2 GB. Due to some resource limitations only part of Free-BSD CVS storage is processed.

Histories of all three projects are stored under control of Concurrent Versioning System (CVS) and were publicly available during our research period from the corresponding Internet servers [18–20]. CVS allows one to store old versions of files and information about who, when and why made some changes to data. The second useful feature of CVS is allowing

multiple people to work with one file simultaneously. At any given moment of time the system keeps current state of all the data space and contains enough information to restore any past state. Each data file is kept in the following way: a text of the last (current) version; changes form the previous state (delta); previous delta; etc. The deltas carry information about which lines one must delete and which lines must be added to a version of file to get the previous version. For each delta of each file an amount $D$ of deleted lines and an amount $A$ of lines added were collected. Empty lines and comments were collected together with the rest of the data. A number of lines in the very first version of each file was not counted.

Distributions $P(A)$ and $P(D)$ were evaluated for these two arrays $A_i$ and $D_i$. As an example the data for the Free-BSD are shown in Figs. 1 and 2, for Emacs in Figs. 3 and 4 in log–log scale. One can see that these graphs can be perfectly approximated by linear function. It means that $P(A)$ and $P(D)$ can be well approximated by power function $P(A) \sim A^{\mu_a}, P(D) \sim D^{\mu_d}$. The values of exponents are the following:

Free-BSD: $\mu_a = -1.44 \pm 0.02, \quad \mu_d = -1.48 \pm 0.02;$
Mozilla: $\mu_a = -1.43 \pm 0.02, \quad \mu_d = -1.47 \pm 0.02;$
Emacs: $\mu_a = -1.39 \pm 0.03, \quad \mu_d = -1.49 \pm 0.04.$

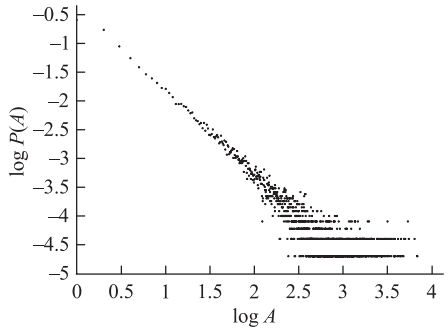


Fig. 1. Distribution $P(A)$ for Free-BSD

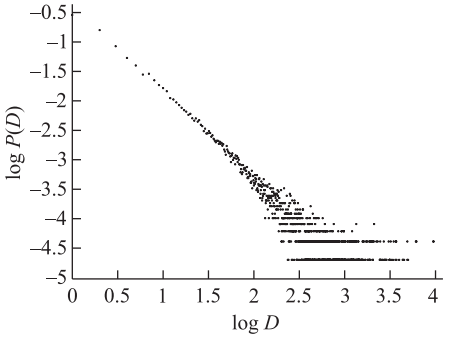


Fig. 2. Distribution $P(D)$ for Free-BSD

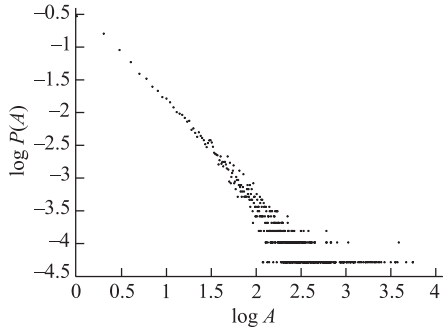


Fig. 3. Distribution $P(A)$ for Emacs

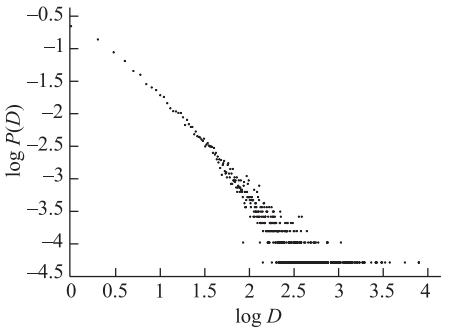


Fig. 4. Distribution $P(D)$ for Emacs

Thus, the amounts of changes in the system between two metastable states can be described by power functions with nontrivial exponents. It is interesting to note that the value of $\mu_a$

for Emacs somewhat differs from the similar values for the other two systems. One can speculate that this is because of different topological structures of the language or because of differences in development process. Of course this question needs the further study. The obtained scaling laws can be considered as a confirmation of hypothesis that evolution of large computer program is a scale invariant universal SOC dynamics.

There is widely used term «avalanche» in the theory of critical processes. It denotes bursts of activity during which long range correlations are built in a system. The SOC systems (i.e., systems coming to the critical state not because of parameter tuning, but because of dynamics of the system itself) change their metastable states via avalanches.

For the software evolution process the close analog of the avalanche is a set of changes going on from version to version. An important feature of software changes is that one programmer can modify a program at only one point at a time (at least using a traditional development tools). In a given time a programmer can make a lot of changes in a lot of places. The question is: what determines the order in which the changes are made? A possible answer can be given by consideration of the extremal dynamics systems. This is the class of the systems in which the changes are going on only in places characterized by an extremal value of some variable. In analogy with the above-said a programmer has some subjective estimation of parts of a program and makes changes not in any place, but in place where this estimation reaches extremely nonsatisfactory values.

These analogies between SOC systems, extremal dynamics and software development process gave us the idea that it is possible to apply methods used to study SOC systems for the analysis and study of evolution processes in large real-world software projects.

## 3. MODEL OF SOFTWARE EVOLUTION

In our model we represent computer program as a sequence of similar elements — lines of code. There is a number corresponding to each line — a barrier. The state of the system of $N$ elements is fully given by $b_i(t), i = 1, 2, \ldots, N$. The $i$th barrier $b_i(t)$ as a function of discrete time $t$ takes its value from interval $[0, 1]$: $0 \leqslant b_i(t) \leqslant 1$.

The dynamics of the model is as follows: in the beginning the value of every barrier is a random number. On every step of discrete time the node with minimal barrier is found. After that there are two possibilities. It can be deleted from the system with probability $\alpha$, and the barriers of the nodes it has been connected with (its neighbors) are set random. Or, some new node connected with the minimal one is inserted into system with probability $1 - \alpha$; after that barriers of minimal one, its neighbors and the new node are set into random values. So the size of the system decreases or increases by one during each step. It is supposed that there is minimal number $K$ of nodes in the system. By definition, if the current number of nodes is equal $K$, deleting of ones is impossible.

It is easy to see that our model is a modification of well-known Simple Model of Biological Evolution introduced by Bak and Sneppen [3]. The significant difference of our model is a variable number of elements in the system. In our study we have chosen two modifications of the model: the two-nearest-neighbors system and random-neighbor system. For the two-nearest-neighbors system the nodes are organized into one-dimensional lattice. So if the minimal barrier is found at the $n$th node its nearest neighbors are $n + 1$th and $n - 1$th.

There is a periodic boundary condition in the model, so the first and the $N$th nodes are the neighbors. For the random-neighbors system there is no order of nodes. At every step of time any $k$ random nodes are chosen as neighbors. This system can be seen as a mean field approximation for a lattice model [4].

Dynamics in this kind of systems is usually described as the so-called avalanche process. In extremal dynamic systems there are two kinds of avalanches: $\lambda$-avalanches and transient avalanches [3, 5, 6]. In our model we are interested mostly in transient avalanches. They can be defined as follows: let at the moment $t_0$ of discrete time the minimal barrier have the value $f_0$. The sequence of $S$ time steps, during which the minimal barrier $b_{\min}(t), t_0 < t < t_0 + S$ does not exceed $f_0$ finished at the $t_0 + S$ step, at which the value of minimal becomes larger than $f_0$, is called transient avalanche or just avalanche.

Usually one is interested in distributions $P(S)$ of avalanche temporal durations and $P(R)$ of avalanche spatial volume. But in our model the size of the system changes during the avalanche. So $P(R)$ has no meaning for our model. There is a pair of variables distributions which correspond to $P(R)$ in our case. $A$ is a number of new elements appeared in the system at the end of avalanche. Note that this is not the total number of elements appeared in the system during the avalanche with probability $1 - \alpha$. Because some of the newly appeared elements can be killed by the subsequent deletions during the very same avalanche. $D$ is a number of elements disappeared from the system at the end of the avalanche. Note again that this is not the total amount of deleted during avalanche elements. As we could delete elements created during the same avalanche, so they do not appear neither in initial state nor in the final state. There is an equal description in terms of $A + D$ and $A - D$. But we think that our choice of variables $A$ and $D$ somewhat a little bit more natural.

In our work we studied mostly the distributions $P(S), P(A), P(D)$ of temporal and spatial characteristics of avalanches. Moreover, there are some characteristics of the model that can be interesting to study from the SOC point of view, such as distribution of barriers in critical state, distribution of minimal barrier.

We made numerical experiments for two modifications of the model: one-dimensional lattice model (1DM) with periodic boundary conditions and model with one random neighbor (RNM). The $\alpha$ coefficient always was taken as 1/2.

The initial size of the system was 8000 elements. The experiment went on until one million of avalanches registered. We got the following results: $P(S) \sim S^{-\tau}$, $P(A) \sim A^{-\mu_a}$, $P(D) \sim D^{-\mu_d}$ with exponents $\tau = -1.358 \pm 0.005$, $\mu_a = -1.45 \pm 0.01$, $\mu_d = -1.47 \pm 0.02$ for 1DM and $\tau = -1.901 \pm 0.008$, $\mu_a - 1.98 \pm 0.01$, $\mu_d = -2.10 \pm 0.02$ for RNM.

The proposed model reproduces basic mechanisms of software evolution. Changes are made by programmer locally in the place where these changes most of all needed. But if a programmer changes a program in one place, he often has to change it in other points in some way connected to the first one. For example, in order to change the number of arguments of subroutine call, we need to change not only the line containing the call operator but the definition of the subroutine either. This would lead to some subsequent changes of all the calls to the subroutine in all the program. Another example: if we add the line in which we read some data from disk, we should add some lines to check if the data read successfully, which, in turn, can require some change in the list of the modules included, which, in turn, can cause a name conflict, which, in turn, can ..., etc. Such an avalanche process ends up when all the parts of the program code are more or less satisfy some subjective and implicit criteria of our programmer. This is why we study transient avalanches of self-organization

period and not the $\lambda$ avalanches of the stationary mode. After the avalanche the value of minimal barrier becomes greater than it was before the avalanche. Naively speaking, the program as a whole becomes «a little bit better» after the avalanche.

As we can see, important dynamic characteristics of the model are described by the power function distributions which allows one to make a conclusion about the observation of self-organized criticality.

## 4. ANALYTICAL RESULTS FOR RNM

For the random-neighbor version of our model one can write exact master equations. Let us chose a parameter $0 < \lambda < 1$ and define the probability $P_{kN}(t)$ that at the time point $t$ the system has $N$ nodes and $k$ barriers of them are less than $\lambda$. It follows from the dynamical rules that $P_{kN}(t)$ fulfils the equation: we shall use the following notations: we denote the probability $P_{n,N}(t)$ that at the time point $t$ there are $N$ elements in the system and $n$ barriers are less than $\lambda$. Master equation is of the form

$$P_{n,N}(t+1) = (\alpha + \beta\delta_{N,K+1})P^a_{n,N}(t) + \beta P^d_{n,N}(t),$$

where $\beta = 1 - \alpha$ and in terms of $\mu = 1 - \lambda$, $\rho_{n,N} = (n-1)/(N-1)$ the quantities $P^d_{n,N}(t)$ and $P^a_{n,N}(t)$ can be presented by the following relations:

$$\begin{aligned} P^a_{n,N}(t) = A^a_{n+2N-1}P_{n+2,N-1}(t) + B^a_{n+1N-1}P_{n+1,N-1}(t) + \\ + C^a_{nN-1}P_{n,N-1}(t) + D^a_{n-1N-1}P_{n-1,N-1}(t) + E^a_{n-2N-1}P_{n-2,N-1}(t) + \\ + (\mu^3\delta_{n,0} + 3\lambda\iota\mu^2\delta_{n,1} + 3\lambda^2\mu\delta_{n,2} + \lambda^3\delta_{n,3})P_{0,N-1}(t), \end{aligned}$$

$$\begin{aligned} P^d_{n,N}(t) = A^d_{n+2N+1}P_{n+2,N+1}(t) + B^d_{n+1N+1}P_{n+1,N+1}(t) + \\ + C^d_{nN+1}P_{n,N+1}(t) + (\mu\delta_{n,0} + \lambda\delta_{n,1})P_{0,N+1}(t). \end{aligned}$$

Here,

$$\begin{aligned} A^a_{n,N} &= \mu^3\rho_{n,N}, \\ B^a_{n,N} &= 3\lambda\mu^2\rho_{n,N} + \mu^3(1-\rho_{n,N}), \\ C^a_{n,N} &= 3\lambda\mu^2(1-\rho_{n,N}) + 3\lambda^2\mu\rho_{n,N}, \\ D^a_{n,N} &= 3\lambda^2\mu(1-\rho_{n,N}) + \lambda^3\rho_{n,N}, \\ E^a_{n,N} &= \lambda^3(1-\rho_{n,N}),\ A^d_{n,N} = \mu\rho_{n,N}, \\ B^d_{n,N} &= \mu(1-\rho_{n,N}) + \lambda\rho_{n,N}, \\ C^d_{n,N} &= \lambda(1-\rho_{n,N}) \end{aligned}$$

and all these coefficients are zero when $n \leqslant 0$, or $n > N$.

Mean value of the system element number $n(t)$ at time point $t$ can be calculated exactly and has the following asymptotic for large $t$:
for $\alpha > 1/2$,

$$n(t) = (2\alpha - 1)t + n(0) - \frac{2\alpha\beta}{(\alpha^2 - \beta^2)} + \frac{(4\alpha\beta)^{t/2}g_1(t)}{t^{3/2}},$$

for $\alpha = 1/2$,

$$n(t) = \sqrt{\frac{2t}{\pi}} + \frac{g_2(t)}{t^{1/2}}$$

and for $\alpha < 1/2$,

$$n(t) = K + \frac{[1 + (-1)^{t+K}(1-2\alpha)^2(p_{\rm ev} - p_{\rm od})]}{2(1-2\alpha)} + \frac{(4\alpha\beta)^{t/2} g_3(t)}{t^{3/2}}.$$

Here we denoted the probability $p_{\rm ev}\,(p_{\rm od})$ that the initial number $N(0)$ of nodes is even (odd). The function $g_i(t)$, $i = 1, 2, 3$ is bounded for large $t$, i.e., there are constants $T$, $M$ that $|g_i(t)| < M$, if $t > T$. Since $4\alpha\beta < 1$ for $\alpha \neq 1/2$, the function $f(t)$ decreases exponentially fast for large $t$.

The asymptotic behaviour of $n(t)$ demonstrates the dynamical phase transition at the point $\alpha = 1/2$. For $\alpha < 1/2$, the volume of system remains finite, but for $\alpha \geqslant 1/2$, it can became as large as one likes. At the point $\alpha = 1/2$, the dynamics of the system is a critical one.

## CONCLUSION

The avalanche-like processes seem to be natural for modifications of programs. The obtained statistical characteristics of avalanches make it possible to conclude that self-organized criricality (SOC) is the dominating dynamical regime in evolution of free software. We demonstrated that the natural selection can create this type of «punctuated equilibrium» for such complex «virtual beings» in info-sphere.

We believe that in the framework of the proposed approach the methods of investigation of the SOC dynamics can be very effective for studies of universal aspects of software evolution. Our results could be seen also as a theoretical prerequisite for the development of new tools and methods for advanced measures of software engineering quality.

The work of Yu. M. P. is partially supported by grants RSS-5538.2006.2 and RNP.2.1.1.112.

## REFERENCES

1. *Bak P.* How Narure Works: The Science of Self-Organized Criticality. Oxford Univ. Press, 1997.
2. *Bak P., Thang C., Wiesenfeld K.* // Phys. Rev. A. 1988. V. 38. P. 364.
3. *Bak P., Sneppen K.* // Phys. Rev. Lett. 1993. V. 71. P. 4083.
4. *de Boer J.* // Phys. Rev. Lett. 1994. V. 73. P. 906.
5. *Paczuski M., Maslov S., Bak P.* // Phys. Rev. E. 1996. V. 53. P. 414.
6. *Maslov S.* adap-org/9601003. 1996.
7. *Pis'mak Yu. M.* // J. Phys. A. 1995. V. 28. P. 3109.
8. *Gorshenev A. A., Pis'mak Yu. M.* // Phys. Rev. E. 2004. V. 70. P. 067103.
9. *Fenton N. E., Neil M.* // IEEE Trans. on Software Engin. 1999. V. 25, No. 5.

10. *Lehman M. M., Ramil J. F., Wernick P. D.* Metrics and Laws of Software Evolution — The Nineties View // Proc. of the 4th Intern. Software Metrics Symp. 1997.

11. *Lehman M. M., Belady L. A.* Program Evolution: Process of Software Change. 1985.

12. *Godfrey M., Tu Q.* Evolution in Open Source Software: A Case Study // Proc. of the Intern. Conf. of Software Maintenance. 2000.

13. *Gall H. et al.* Software Evolution Observations Based on Product Release History // Proc. of the Intern. Conf. of Software Maintenance. 1997.

14. *Eick S. G. et al.* Does Code Decay? Assessing the Evidence form Change Management Data // IEEE Trans. on Software Engin. 1999.

15. *Parnas L. D.* Software Aging // Proc. of the 4th Intern. Software Metrics Symp. 1997.

16. *Akiyama F.* An Example of Software System Debugging // Inform. Processing. 1971. V. 71.

17. *Halstead M. H.* Elements of Software Science. 1975.

18. The Free-BSD homepage. www.freebsd.org

19. The Mozilla project homepage. www.mozilla.org

20. The GNU project homepage. www.gnu.org