A Simulation Model of Self-organising Evolvability in Software Systems

Stephen Cook and Rachel Harrison
School of Systems Engineering
University of Reading
Reading, UK
Email: {s.c.cook, rachel.harrison}@reading.ac.uk

Paul Wernick
Department of Computer Science
University of Hertfordshire
Hatfield, UK
Email: p.d.wernick@herts.ac.uk

Abstract— **The evolvability of a software artifact is its capacity for producing heritable or reusable variants; the inverse quality is the artifact's inertia or resistance to evolutionary change. Evolvability in software systems may arise from engineering and/or self-organising processes. We describe our 'Conditional Growth' simulation model of software evolution and show how it can be used to investigate evolvability from a self-organisation perspective. The model is derived from the Bak-Sneppen family of 'self-organised criticality' simulations. It shows good qualitative agreement with Lehman's 'laws of software evolution' and reproduces phenomena that have been observed empirically. The model suggests interesting predictions about the dynamics of evolvability and implies that much of the observed variability in software evolution can be accounted for by comparatively simple self-organising processes.**

I. Introduction

A. The concept of evolvability

Evolvability has been defined in a biological context as 'an organism's capacity to generate heritable phenotypic variation' [13], i.e. the capacity to generate variations in 'the manifested attributes of an organism, the joint product of its genes and their environment [8]' that can be inherited by offspring. For example, female cuckoos lay their eggs in the nests of other birds. Cuckoos vary in their choice of foster species and in their deception techniques, e.g. egg mimicry. These variations in cuckoo phenotypes appear to be inherited and also to vary over time as the unwilling foster birds develop defensive tactics against the cuckoo. Thus, cuckoos need to have greater evolvability than, say, robins to stay ahead in the 'arms race' [8] between them.

By abstracting the general characteristics of evolvability from its original biological context, the concept can be applied to artificial systems [19], including software systems. Generalised evolvability can be understood as the inverse of inertia in relation to a replicator's capacity for evolutionary change. A replicator [8] is anything — e.g. a gene, a software component — that can be copied, possibly with variations. The copying operation requires the expenditure of energy, partly to overcome inertia and partly to create the copy.

Inertia to evolutionary change can arise as a natural consequence of previous successful adaptations. When replicators are competing to survive and replicate, long-term success for the replicator — but not necessarily for its host — depends not only on finding a 'winning formula' but also on defending it against random changes and resisting being dislodged by competing replicators.

Software components have those characteristics. They exhibit a form of heritability, both in the object-oriented sense of inheritance and also in a more general sense where, for example, each successive version of a component is a copy-with-possible-variation of its preceding version. Such variations may affect the component's 'manifested attributes', usually its behaviour or interface. Software components also vary in how easily they can be changed. Thus, they have the core attributes of evolvability, as identified in biological terms by Kirshner and Gerhart [13] above. The inverse quality of inertia or resistance to evolutionary change is also frequently observed in software.

B. Evolvability and self-organising processes

When replicators are competing to occupy scarce niches, the population dynamics can be complex. It is sometimes tempting to explain such changes in a metaphoric or anthropomorphic way, e.g. in terms of the 'desires' or 'intentions' of genes, organisms, etc. However, biologists in general prefer theories that rely on self-organising mechanisms to explain evolutionary change. For example, Maynard Smith [18] used game theory to explain the statistical distribution of alternative heritable behaviours in a population.

Care is needed when applying similar explanations to software systems, because clearly some changes are the direct result of stakeholders' intentions. However, it is also apparent from the accumulated work, both empirical and theoretical, on software evolution that there are emergent, unintended effects when systems evolve over significant periods of time. It is therefore reasonable to investigate whether these findings can be understood as evidence of self-organising processes that may or may not be similar to those found in other kinds of evolving systems, including biological systems. In this work, we examine an approach to modelling self-organising processes that is particularly associated with Bak [1], and we consider how it can be applied to a simple model of changing evolvability in an evolving software system.

Two related contributions towards a theory of software evolution are particularly relevant to this work, namely the SPE [7], [15] classification of evolving software systems and Lehman's proposed 'laws of software evolution' [17]. The SPE classification is related to evolvability in the following way:

- *S*-type systems do not evolve: therefore, by definition, any apparent capacity for generating phenotypic change is never manifested;
- the evolution of *P*-type systems is constrained by stakeholders' policies;
- so long as an *E*-type system remains useful to its stakeholders, its evolution is inevitable and the outcome is difficult to predict.

Thus, evolvability can be an important property of *E*- and *P*-type systems and components but is not relevant to *S*-type. For example, the architecture of a *P*- or *E*-type system can be viewed from an evolvability perspective as a collection of design commitments that embody stakeholders' assumptions and policies about the expected evolution of the system.

Lehman's 'laws of software evolution' identify several self-organising processes in software systems. Laws I, II and VI — 'Continuing Change', 'Increasing Complexity', 'Continuing Growth', respectively — refer implicitly to evolvability.

II. Simulation and Evolvability

In this work we use simulated changes in the evolvability of *components* to explore the *system* dynamics of evolution processes. Simulation can abstract from situations that are known empirically to be complex and diverse. It can also be used to model the dynamic effects of theories and of generalised observations such as Lehman's Laws. Various simulation techniques, e.g. [6], [21], have been found helpful for investigating software evolution.

The simulation approach used in this work involves merging many possible causes of the target phenomenon to a simple model that still exhibits interesting behaviour. Possible causes of change in a system or component's evolvability, include:

- engineering and project management actions
- uncertainty / volatility in requirements
- tolerances and rigidities of implementations
- innovative use of the system

However, in the simulation model described here, all possible causes of change in evolvability are collapsed into a single mechanism.

Our simulation model is called the Conditional Growth model and is described in detail in Section IV. Its purpose is to explore whether various behaviours that are generally considered to be normal in evolving software systems can be simulated by models derived from the concept of self-organised criticality (SOC) [3], described in Section III. At this stage, the Conditional Growth model cannot be proposed as a complete and accurate characterisation of the software evolution process.

III. Progressive and Punctuated Evolution

Systems in general can exhibit many kinds of dynamic behaviour. From an evolution perspective, two modes of system dynamics are particularly interesting:

- *Progressive* : successive states of the system are closely related and tend to exhibit change in a consistent direction over relatively long periods;
- *Punctuated* : the system repeatedly reaches 'poised' or 'critical' states, far out of equilibrium, from which further change occurs as an 'avalanche' of unpredictable size, timing, direction, etc.

Under the influence of Darwin, evolution researchers in both biological and artificial systems often assumed that the Progressive mode predominated. However, it has become recognised that both modes are found. Eldredge and Gould [9] proposed that 'punctuated equilibrium' is a more convincing explanation for the emergence of many species than the more conventional theory of 'phyletic gradualism'. Bak *et al.* [3] showed that the Punctuated mode of system dynamics is related to power law distributions [20] in system properties and can be found in many natural and artificial systems, e.g. the distribution of earthquake sizes in a fault zone. Bak uses the term *self-organised criticality* to refer to the capacity of some systems in Punctuated mode to consistently and repeatedly reach a critical state without tuning or other external intervention.

There are both theoretical and empirical grounds for conjecturing that the Punctuated mode of evolution can occur in software systems. We have found conceptual similarities between the descriptions of punctuated equilibrium and SOC, Simon's [22] research into evolution in hierarchical systems, and our own work [7] with Lehman on the foundations of SPE. In particular, we note the inherent capacity of *E*-type systems to evolve in unanticipated ways. Empirically, Barry *et al.* [4] found noticably non-uniform evolution in a portfolio of 23 business applications. Wu *et al.* [24] found punctuated evolution in a survey of three open-source systems.

The Progressive and Punctuated modes of evolution have different implications for system properties, including evolvability. For example, the Progressive mode is more likely to be associated with constant or slowly changing evolvability than with sudden changes. If changes in evolvability are a result of self-organising processes, they need not have any external cause. Consequently, stakeholders may perceive such changes as unexplained or disproportionate to their supposed cause. Simulation can increase our understanding of these situations.

IV. The Conditional Growth Model of Evolution

Bak and Sneppen [2] developed a simple simulation model of an evolving ecosystem that robustly achieves SOC. However, the Bak-Sneppen model is not immediately applicable to evolving software systems because one of its simplifications is to conserve the number of 'species' in the 'ecosystem', whereas software systems tend to grow, as described by Lehman's Law VI [17]. In this work, we define a 'Conditional Growth' variant of the Bak-Sneppen model that allows the number of components in the system to increase.

A. Model definition

The Conditional Growth model can be defined in terms of the topology of the evolving system and the rules for selection and adaptation of the system's components that are applied at each time step in a simulation run:

- **Topology** : a ring of N components.
- **Selection** : as in the Bak-Sneppen model, an 'evolutionary inertia' value s is assigned to each component c. At each time step, the component with the lowest inertia c_i, located at position i in the ring, is selected for adaptation.
- **Adaptation** of the selected component c_i affects the system in several ways:
  - **local adaptation** of c_i is represented by assigning it a new random value of inertia s';
  - **consequential adaptation** of a minimal subset of the selected component's neighbours is represented by assigning a new random value of inertia to component c_{i-1};
  - **reproduction** : the reproductivity of c_i is determined by a step function $\mathcal{R}$ ranging over $\{0, 1\}$. If $\mathcal{R}(c_i) = 1$ then the component reproduces by inserting an offspring at position $i + 1$.

Thus, at each time step in a simulation, one component is selected for adaptation. That component and one of its neighbours receive new random inertia values. The value of $\mathcal{R}$ determines whether the system size increases by one or is static.

B. Histories and scenarios

Different kinds of *histories*, i.e. individual runs or iterations of the simulation model, can be generated by configuring the model with different parameters and/or redefining the $\mathcal{R}$ function. Each distinct definition of $\mathcal{R}$ is a *policy*. We use the term *scenario* to refer to a policy and a collection of parameter values that define the starting conditions for one or more histories.

In the simplest possible scenario, $\mathcal{R}$ is a constant, e.g. 1. In this case, the number of components N would increase in each time step. This does not produce interesting histories. They exhibit uniform, linear growth, which is inconsistent with both Lehman's Laws and empirical observations. However, more complex policies can be devised that model more features of Lehman's Laws and exhibit more interesting behaviour.

C. Simulation framework

The authors have designed and implemented a framework for simulating the Conditional Growth model in Java. The framework allows histories to be generated for different policies and scenarios.

V. Description of $\mathcal{R}_{\mathcal{U}}$ Policy

To illustrate the role of the reproductivity function $\mathcal{R}$ in the Conditional Growth model, we consider a policy $\mathcal{R}_{\mathcal{U}}$ that models a conjecture derived from Lehman's Law IV 'Conservation of organisational stability' or 'Invariant work rate'. The conjecture is that both *reductions* in inertia, e.g. due to refactoring, and *significant increases* in inertia, e.g. due to modifying a component, tend to absorb stakeholder effort that might otherwise be used to add new components (functionality) to the system.

$\mathcal{R}_{\mathcal{U}}$ models this conjecture by ensuring that $\mathcal{R}$ is only mapped to 1 when the inertia increment $s' - s$ is small and positive, otherwise $\mathcal{R}$ is mapped to 0. This is achieved by defining $\mathcal{R}_{\mathcal{U}}$ in terms of $s' - s$, a fixed 'threshold' value ϑ, and a feedback component $U(c_i)$, defined as the number of previous updates or 'mutations' of component c_i:

$$\mathcal{R}_{\mathcal{U}} = \begin{cases} (s'-s) < 0 & \mapsto 0 \\ 0 < (s'-s) < \vartheta/U(c_i) & \mapsto 1 \\ \vartheta/U(c_i) < (s'-s) & \mapsto 0 \end{cases}$$

The ϑ parameter allows some coarse-grained tuning of the model's behaviour within the $\mathcal{R}_{\mathcal{U}}$ policy. As $\vartheta \to 0$, the behaviour of the model approximates the 'no growth' scenario associated with *S*-type systems. As $\vartheta \to 1$, the growth of the system becomes steeper and smoother.

VI. Qualitative Analysis

If the Conditional Growth model is configured with the $\mathcal{R}_{\mathcal{U}}$ policy, it models Lehman's Laws to the following extent:

I **Continuing Change** : evolutionary inertia levels change at every time step, representing the effect of continual adaptation of the system in a continually changing environment.

II **Increasing Complexity** : the increasing complexity of the system is represented by the values of $U(c_i)$, which increase as the system evolves.

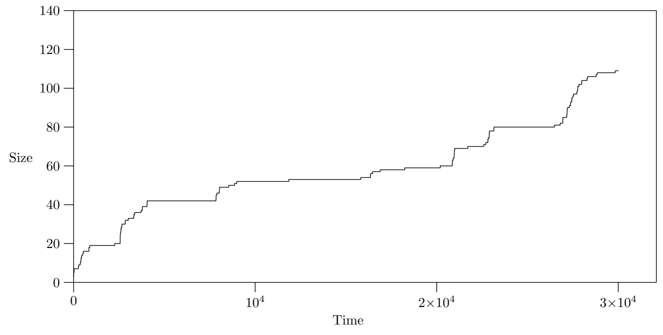
III **Self Regulation** : the model exhibits consistently convergent features and also properties with non-normal probability distributions, see Section VII.

IV **Conservation of Organisational Stability** : the definition of $\mathcal{R}_{\mathcal{U}}$ is derived from this Law, see section V.

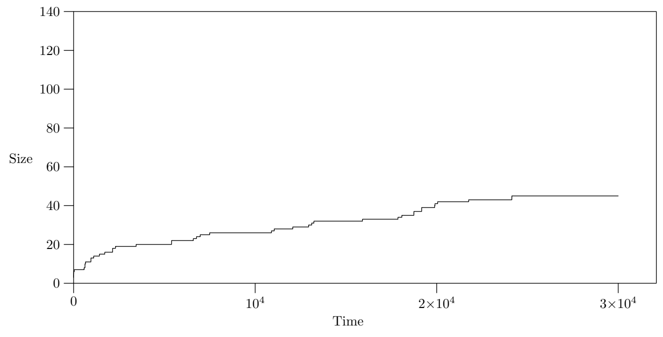
V **Conservation of Familiarity** : 'releases' are not modelled explicitly; the distribution of stasis lengths, see Section VII, implies that $\mathcal{R}_{\mathcal{U}}$ histories do not have consistent periodicity.

VI **Continuing Growth** : the system has the potential to grow indefinitely at extremely variable short-term rates.

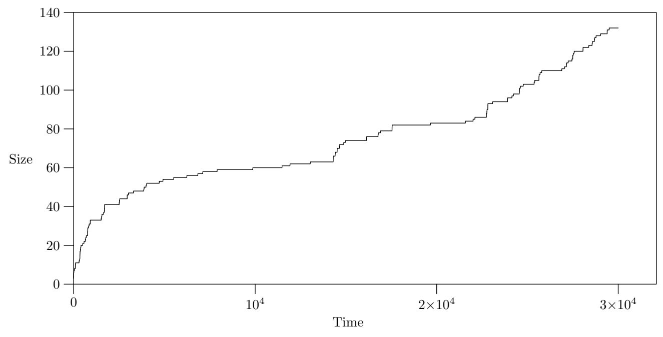
VII **Declining Quality** : the definition of $\mathcal{R}_{\mathcal{U}}$ can also be interpreted as a tradeoff between increasing functionality and other effort-consuming activities, e.g. improving quality; the tradeoff for each component becomes steeper as it ages, represented by increasing values of $U(c_i)$.



(a)



(b)



(c)

Figure 1: Examples of growth curves generated by scenario $R_U(\vartheta = 0.3)$

VIII **Feedback System** : each time-step sets the subsequent step's starting conditions, which include the explicit feedback provided by the values of $U(c_i)$.

VII. Statistical Analysis

This section presents some preliminary results from our analysis of the $\mathcal{R_U}$ policy that are particularly relevant to the self-organising dynamics of software evolvability.

A. Short-term evolution and evolvability

As $\vartheta \to 0$, the $\mathcal{R_U}$ growth policy produces very diverse growth curves, often with many abrupt changes of gradient. Some examples are shown in Fig. 1. They show similarities to growth curves found in empirical investigations, e.g. [24]. Thus, short-term evolution in $\mathcal{R_U}$ histories shows visual evidence of the Punctuated mode of evolution.

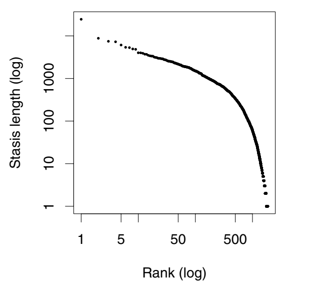
The distribution of the length of *stasis periods*, i.e. the number of consecutive time steps when $\mathcal{R} = 0$, provides a convenient way of characterising the model's micro-evolutionary behaviour that is analagous to the analysis of avalanche sizes by Bak and his colleagues.

It might be expected that the distribution of stasis lengths would approximate a power law, given that the Conditional Growth model is derived from the Bak-Sneppen model in which avalanches consistently show a power law distribution. However, it appears that this is not the case, at least for histories up to 75 000 time-steps.

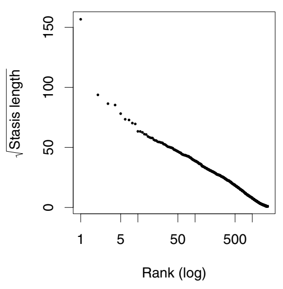
Nevertheless, the distribution of stasis lengths within each history is highly skewed, with a long tail of very large values. Fig. 2 shows the fit of the empirical data against 'stretched exponential' [14] and power law distributions. In each case, a perfect fit would result in a linear plot. Thus, Fig. 2a indicates a poor fit to a power law distribution. Fig. 2b shows a better fit to a stretched exponential distribution of the form

$$\sqrt{x_n} = b - a\log(n)$$

where x_n is the stasis length value at rank n of N stasis lengths, such that $0 < n < N + 1$ and $x_1 \geq x_2 \geq \cdots \geq x_N$ and a, b are constants.



(a) Power law



(b) Stretched exponential

Figure 2: Probability distribution fitness for stasis lengths data

Although the *overall* distribution of stasis lengths is similar between histories generated by the same scenario, successive stasis lengths within the *same* history are scarcely correlated. Treating each history's sequence of stasis lengths as a time series [5], we found very low values of the autocorrelation function. This confirms the visual impression from Fig. 1 that $\mathcal{R_U}$ growth curves have very low smoothness.

Thus, when a system's short-term evolution resembles the $\mathcal{R_U}$ policy, micro-trends in properties that are evolutionarily sensitive cannot be predicted with confidence. This is because (a) the relevant probability distributions are highly skewed with very large variances, and (b) the recent past is a poor predictor of the immediate future.

B. Long-term evolution and evolvability

Boxplots of system size over time provide a convenient way of describing histories from a macro-evolution perspective. The growth curves of nine histories each of 75 000 time-steps are summarised in Fig. 3 using this technique. The boxplots show that the variability of short-term growth rates is also found at larger scales. This implies wide variations in long-term evolution between $\mathcal{R}_{\mathcal{U}}$ histories.

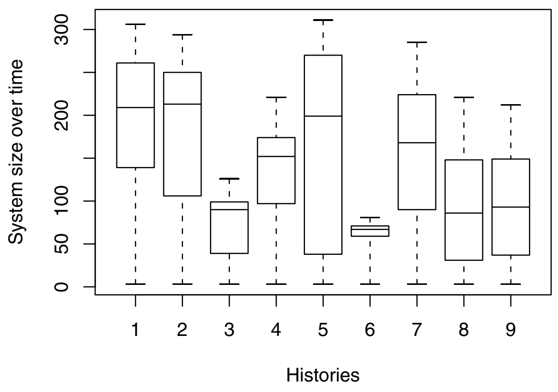


Figure 3: Boxplots of system growth over time for 9 histories of 75 000 time-steps generated by scenario $\mathcal{R}_{\mathcal{U}}(\vartheta = 0.3)$. The divided boxes show the growth achieved during the two middle quartiles of time, and the system size at the median time, i.e. $t = 37500$.

Nevertheless, the underlying long-term growth trend for $\mathcal{R}_{\mathcal{U}}$ histories is linear, i.e. mean and standard deviation of system size increase linearly with history length. This reveals an aspect of $\mathcal{R}_{\mathcal{U}}$ that lacks validity, since both theory [23] and empirical observations [6] suggest that system growth rates tend to decline in the long term.

VIII. Discussion and Conclusions

A. *Probability distributions in evolving software systems*

The Conditional Growth model shows that simple models of evolution can produce effects with various probability distributions. Lehman's Law III proposes that

> '*E*-type system evolution process is self regulating with distribution of product and process measures close to normal.' [17]

Our simulation results suggest that Law III may understate the variability of software evolution processes. We conjecture that the applicability of Law III could be improved by revising it along the following lines:

> '*E*-type system evolution processes are self-organising, producing distributions of system property values that approximate various well-defined probability distributions, including at least normal, log-normal and exponential.'

This conjecture should be tested by empirical studies of a wide range of evolving software systems. Our understanding of software evolution processes would increase significantly if we found consistent statistical relationships between evolutionary properties in software systems and the probability distributions of their values.

B. *Power laws in software evolution*

We have not yet found clear evidence of power law distributions in $\mathcal{R}_{\mathcal{U}}$ histories. This is unexpected, given the similarities between the Bak-Sneppen and Conditional Growth models, and contrary to the results from the adaptation of the Bak-Sneppen model devised by Gorshenev and Pis'mak [12]. However, we note that several researchers [11], [14], [20] have cautioned against over-diagnosis of power laws and SOC. This question may be resolved by running longer histories, since it is known, e.g. from Bak's [1] experiments with rice piles, that SOC properties take time to emerge.

C. *Modelling Punctuated evolution*

The evidence from $\mathcal{R}_{\mathcal{U}}$ histories suggests that this policy is more successful at modelling the Punctuated mode of evolution in the short than in the long term. Further work is required to improve the validity of the model for longer histories. We plan to investigate alternative policies to $\mathcal{R}_{\mathcal{U}}$, e.g. by considering whether additional feedback factors can improve the model's validity.

D. *Representations of time in models of software evolution*

One of the significant differences between the Conditional Growth model and some previous work on measuring and modelling software evolution is the treatment of time. The Conditional Growth model uses absolute, but scale-free, units of time. On the other hand, it rounds each increment in system size to 0 or 1. However, other researchers have made different decisions. For example, in his empirical investigations, Lehman measures size increments absolutely, using 'module' units, but effectively normalises the lengths of stasis periods between increments by measuring time in 'release sequence numbers' rather than absolute units.

Both practices, and indeed others, can be justified but it is important to understand their different implications for discarding information and thus possibly losing insights into the phenomena under study. Further analysis of previous empirical studies of software evolution might reveal evidence of Punctuated evolution that had been overlooked.

E. *Non-smooth dynamics in software evolution*

The discovery that simple policies such as R_U can produce non-smooth growth curves is significant. Empirical studies of evolving software systems have found instances where an underlying trend of declining growth was apparently interrupted or restarted by short-term bursts of more rapid growth. Lehman suggested that this variability in short-term growth rates could be attributed to exceptional, local causes such as 'anti-regressive' or refactoring work by the system development team.

Punctuated evolution offers an alternative, simpler explanation, namely that when a system is evolving in Punctuated mode, it will exhibit non-smooth change as an emergent feature that does not *necessarily* require further explanation. For example, if Figs. 1a and 1c were plots of individual case studies rather than simulation data, it would be tempting to interpret the abrupt increases in short-term growth rates as 'regeneration points' and to seek a local explanation in that system's specific 'global software process'. However, the Conditional Growth model shows that this could be an over-interpretation of the data.

F. What is 'normal' evolutionary behaviour?

The results obtained from simulating the Conditional Growth model provide strong grounds for expecting that the evolutionary behaviour of real-world software systems can be both complex and diverse even if the underlying process is eventually found to be simple and uniform. In other words, software evolution researchers should not assume that complex phenomena must be the result of multiple or complex causes, nor should they assume that diverse behaviour must be the result of different processes. For example, the apparently contrasting findings of, say, Godfrey [10] and Lehman *et al.* [16], may have a simple explanation in the large variances and skewed probability distributions of software evolution metrics.

Claims of significant differences in evolutionary properties between, say, open-source and proprietary systems or between *E*-type and *P*-type systems, should be based on statistically valid sampling. The highly variable histories generated by single scenarios of the R_U policy, demonstrate that generalisations from single case studies or micro-samples must be treated cautiously.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers for their helpful and perceptive comments on the originally submitted version of this paper.

REFERENCES

[1] P. Bak, *How Nature Works: The Science of Self-Organized Criticality*. Oxford: OUP, 1997.

[2] P. Bak and K. Sneppen, "Punctuated equilibrium and criticality in a simple model of evolution", *Phys. Rev. Let.*, vol. 71, no. 24, pp. 4083–4086, 1993.

[3] P. Bak, C. Tang, and K. Wiesenfeld, "Self-organized criticality: an explanation of the 1/f noise", *Phys. Rev. Let.*, vol. 59, pp. 381- 384, 1987.

[4] E. J. Barry, C. F. Kemerer, and S. A. Slaughter, "On the uniformity of software evolution patterns" in *Proc. 25th Int. Conf. Software Engineering (ICSE'03).*, 2003, pp. 106–113.

[5] C. Chatfield, *The Analysis of Time Series: An Introduction*, 5th ed. Boca Raton, FL: Chapman & Hall/CRC, 1996.

[6] B. W. Chatters, M. M. Lehman, J. F. Ramil, and P. Wernick, "Modelling a software evolution process: a long-term case study", *J. Software Process Improve. Pract.*, vol. 5, no. 2–3, pp. 95–102, 2000.

[7] S. Cook, R. Harrison, M. M. Lehman, and P. Wernick, "Evolution in software systems: foundations of the SPE classification scheme", accepted for publication in 'Journal of Software Maintenance and Evolution: Research and Practice', 2005.

[8] R. Dawkins, *The Extended Phenotype: The Long Reach of the Gene*, revised ed. Oxford: OUP, 1999.

[9] N. Eldredge and S. J. Gould, "Punctuated equilibria: an alternative to phyletic gradualism" in *Models in Paleobiology*, T. J. M. Schopf, Ed. San Francisco, CA: Freeman, Cooper, 1972, pp. 82–115.

[10] M. W. Godfrey and Q. Tu, "Evolution in open source software: a case study" in *Proc. Int. Conf. Software Maintenance (ICSM'00)*, B. Werner, Ed. Los Alamitos, CA: IEEE Computer Society, 2000, pp. 131–142.

[11] M. L. Goldstein, S. A. Morris, and G. G. Yen, "Problems with fitting to the power-law distribution", *Eur. Phys. J., B*, vol. 41, pp. 255–258, 2004.

[12] A. A. Gorshenev and Y. M. Pis'mak, "Punctuated equilibrium in software evolution", submission to arXiv e-print archive, 2003. URL: http://arxiv.org/abs/cond-mat/0307201

[13] M. Kirschner and J. Gerhart, "Evolvability", *Proc. Natl. Acad. Sci. USA*, vol. 95, no. 15, pp. 8420–8427, 1998.

[14] J. Laherrere and D. Sornette, "Stretched exponential distributions in nature and economy: 'fat tails' with characteristic scales", *Eur. Phys. J., B*, vol. 2, pp. 525–539, 1998.

[15] M. M. Lehman, "Programs, life cycles, and laws of software evolution", *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[16] M. M. Lehman and L. A. Belady, Eds., *Program Evolution: Processes of Software Change*, A.P.I.C. Studies in Data Processing, 27. London: Academic Press, 1985.

[17] 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 *Proc. 4th Int. Symposium On Software Metrics (Metrics 97)*. Los Alamitos, CA: IEEE Computer Society, 1997, pp. 20–32.

[18] J. Maynard Smith, *Evolution and the Theory of Games*. Cambridge, UK: CUP, 1982.

[19] C. L. Nehaniv, "Evolvability in biology, artifacts, and software systems" in *Proc. Artificial Life VII Workshops*, C. C. Maley and E. Boudreau, Eds. MIT Press, 2000, pp. 17–21.

[20] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law", revised version, posted to arXiv 9 Jan, 2005. URL: http://arxiv.org/abs/cond-mat/0412004

[21] J. F. Ramil and N. Smith, "Qualitative simulation of models of software evolution", *Softw. Proc. Impr. Prac.*, vol. 7, pp. 95–112, 2002.

[22] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, MA: MIT Press, 1996.

[23] W. M. Turski, "Reference model for smooth growth of software systems", *IEEE Trans. Software Eng.*, vol. 22, no. 8, pp. 599–600, 1996.

[24] J. Wu, C. W. Spitzer, A. E. Hassan, and R. C. Holt, "Evolution spectrographs: visualizing punctuated change in software evolution" in *Proc. 7th Int. Workshop On Principles of Software Evolution.*, 2004, pp. 57- 66.