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 suitability for producing heritable or reusable variants. This 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

Evolvability has been defined in a biological context as ‘an organism’s capacity to generate heritable phenotypic variation’ [14]. By equating phenotypic with the structural and behavioural properties of software, and heritable with reusable, this definition can be paraphrased for the software engineering domain as

the suitability of a software artifact for producing variants that (a) differ in structure and/or behaviour and (b) can become subsequent versions of the same artifact or be reused in other artifacts.

This definition implies that software evolvability can be considered from various perspectives, in particular:

• engineering: which engineering and management techniques are effective in producing variants of software artifacts that maximise stakeholders’ satisfaction? E.g. has the W3C’s patenting policy improved or undermined the evolvability of the Web?

• self-organisation: are there features of software development processes that tend to produce or inhibit reusable variations of a system’s artifacts independently of stakeholders’ intentions?

The work described here is concerned with the latter perspective. We describe our ‘Conditional Growth’ simulation model of software evolution and show how it can be applied to evolvability, e.g. it can be used to explore how evolvability varies over a system’s lifetime or between systems. One of the model’s implications is that much of the variability in the evolution of software systems and their properties, including evolvability, can be accounted for by simple self-organising processes.

Lehman’s Laws of Software Evolution [18] can be understood as an attempt to characterise the self-organising processes in software systems that are most important from an evolution perspective. Several of the laws have implications for a system’s evolvability. In particular, laws I, II and IV — ‘Continuing Change’, ‘Increasing Complexity’, ‘Continuing Growth’, respectively — imply that all E-type [7], [16] software systems have some degree of evolvability because over time they inevitably produce variants that become subsequent versions of the same artifact and/or get incorporated into other artifacts. However, law II again, and also laws VII and VIII — ‘Declining Quality’, ‘Feedback System’ — imply that the system variants which are produced are not necessarily those intended or desired by stakeholders.

Therefore, to understand and successfully manage the evolvability of software systems, it is important to adopt a holistic perspective towards the development and maintenance processes of software systems. Lehman [17] coined the term ‘global software process’ for this concept.

II. SIMULATION AND EVOLVABILITY

In this work we use simulation to explore the 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], [20], 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 create the simplest possible model that still exhibits interesting behaviour. In the case of software evolvability, possible causes include:

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

1 Dawkins [8] defines phenotype as ‘the manifested attributes of an organism, the joint product of its genes and their environment’.
However, in the simulation model described here, all possible causes of evolution in a software system 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 definition of the software evolution process.

Nevertheless, the Conditional Growth model is interesting from an evolvability perspective because a system’s growth rate can be treated as an approximation of its evolvability. A system that grows faster is, other things being equal, more likely to be acquiring components that can be inherited by later versions or can be copied, perhaps with variations, into other systems or artifacts.

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 ‘phylogenetic gradualism’. Bak et al. [3] showed that the Punctuated mode of system dynamics is related to power law distributions [19] 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 [21] research into evolution in hierarchical systems, and our own work with Lehman [7] on the foundations of the SPE classification for evolving software systems. In particular, we note the inherent capacity of $E$-type systems to evolve in unanticipated ways. Empirically, Barry et al. [4] found noticeably 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 effects on system properties, including evolvability. For example, when a system is evolving in Punctuated mode, it is likely that there will be sudden changes in its evolvability. Furthermore, if these changes 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 IV [18]. 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, reproduction and mutation of the system’s components that are applied at each time step in a simulation run:

- **Topology**: a ring of $N$ components, see Section IV-B.  
- **Selection**: as in the Bak-Sneppen model, a ‘fitness’ value $s$ is assigned to each component $c$ and the component with the lowest fitness is selected. The model treats the fitness of a component as an analogy for the level of stakeholders’ satisfaction with a component in a real-world software system.
- **Reproduction**: the reproducibility of the selected component $c_i$, located at position $i$ in the ring, is determined by a step function $R$ ranging over $\{0, 1\}$. If $R(c) = 1$ then the component reproduces by inserting an offspring at position $i + 1$.
- **Mutation**: the target and its neighbour at position $i - 1$ are assigned random values of fitness, similarly to the Bak-Sneppen model.

Mutation is used as an analogy for adding a component to a software system.

B. Model rationale

Both the Bak-Sneppen and the Conditional Growth models simplify ‘fitness’ to a one-dimensional value. In such highly simplified models, there appear to be no strong *a priori* reasons to expect that changes in the fitness of individual components will exhibit a consistent direction, and therefore the mutation rule assigns random fitness values.
Another feature that is shared by the Bak-Sneppen and Conditional Growth models is that interactions between the components of a system are restricted to the two nearest neighbours\(^2\) in a ring topology. In an ecological context, this can be justified as a simplification of the food-chain relationship. In a software system, it can be understood as an analogy of the ‘requires-provides’ relationship between software components.

A 1-D ring topology is also attractive for the Conditional Growth model because of its simplicity. Although other structures, e.g. trees, might be more realistic for modelling software systems, each additional dimension in the topology requires more complexity in the simulation algorithm. For example, in a tree, the simulation algorithm must decide whether to grow it horizontally or vertically.

The Conditional Growth model can be used to explore features of evolutionary dynamics that appear to be ‘missing’ from current models of software evolution. For example, an important assumption of the Bak-Sneppen model, which is carried forward into the Conditional Growth model, is that at each point in time the locus of evolutionary change in a system is the component with the lowest fitness. Bak adopted this assumption from Wright’s [23] work on ‘fitness landscapes’, which concluded that the inertial barriers to evolutionary change were likely to be lowest in the least fit species or gene in an ecosystem. It should be possible to investigate empirically whether this assumption is valid for evolving software systems, and this might lead to the formulation of additional laws of software evolution.

C. 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.

D. 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.

\(^2\) Bak et al. [10] also considered other configurations, e.g. random neighbours.

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

To illustrate the role of the reproductive function \(\mathcal{R}\) in the Conditional Growth model, we consider a policy \(\mathcal{R}_{U}\) which defines \(\mathcal{R}\) in terms of the fitness increment \(s' - s\), a fixed ‘Threshold’ value \(\vartheta\), and a feedback component \(U(c_i)\), defined as the number of previous updates or ‘mutations’ of component \(c_i\):

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

The meaning of \(\mathcal{R}_{U}\) can be summarised informally as:

The system tends to grow when the least fit component is improving in fitness and has rarely been updated in the past.

The \(\vartheta\) parameter allows some coarse-grained tuning of the model’s behaviour within the \(\mathcal{R}_{U}\) policy. As \(\vartheta \to 0\), the behaviour of the model approximates the ‘no growth’ scenario associated with \(S\)-type [7], [16] 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 \(\mathcal{R}_{U}\) policy, it models Lehman’s Laws to the following extent:

I Continuing Change: stakeholder satisfaction 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 approximated by the array \(U\), whose values 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 same algorithm is performed at each time step

V Conservation of Familiarity: ‘releases’ are not modelled explicitly; the distribution of stasis lengths, see Section VII, implies that \(\mathcal{R}_{U}\) 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 \(\mathcal{R}_{U}\) can be interpreted as a tradeoff between improving quality and increasing functionality, because growth only occurs when the improvement in the target component’s fitness is smaller than \(\vartheta\); the tradeoff becomes steeper as components age, represented by increasing values in array \(U\)

VIII Feedback System: each time-step sets the subsequent step’s starting conditions, which include the explicit feedback provided by \(U(c)\)
A. Short-term evolution and evolvability

As \( \vartheta \to 0 \), the \( R_U \) 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 \( R_U \) 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 \( R = 0 \), provides a convenient way of characterising the model’s micro-evolutionary behaviour that is analogous to the analysis of avalanche sizes by Bak and his colleagues. Fig. 1 shows that \( R_U \) growth curves approximate the ‘devil’s staircase’ property, which Bak associated with SOC. This implies that the growth curves of software systems could be fractal.

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’ [15] 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 \( x_1 \geq x_2 \geq \ldots x_N \) and \( a, b \) are constants.

VII. STATISTICAL ANALYSIS

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

A. Short-term evolution and evolvability

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 \( R_U \) growth curves have very low smoothness.

Thus, when a system’s short-term evolution resembles the \( R_U \) policy, micro-trends in properties such as evolvability 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.

A devil’s staircase is fractal, i.e. the distribution of step sizes is self-similar at all scales.
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 evolvability between $R_U$ histories.

![Boxplots of system growth over time for 9 histories of 75,000 time-steps generated by scenario $R_{U, \vartheta} = 0.3$.](image)

Figure 3: Boxplots of system growth over time for 9 histories of 75,000 time-steps generated by scenario $R_{U, \vartheta} = 0.3$. 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 $R_U$ histories is linear, i.e. mean and standard deviation of system size increase linearly with history length. This reveals an aspect of $R_U$ that lacks validity, since both theory 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 very simple models of evolution can produce effects with various probability distributions. Lehman’s Law III states

‘$E$-type system evolution process is self-regulating with distribution of product and process measures close to normal.’ [18]

We suggest the following revision:

‘$E$-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 finding has implications for the expected variance in system properties such as evolvability. When a system is evolving in Punctuated mode, estimates of future values must take account of the possibility of non-normal probability distributions.

B. Power laws in software evolution

We have not yet found clear evidence of power law distributions in $R_U$ 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 [13]. However, we note that several researchers [12], [15], [19] 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 $R_U$ 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 in simulating long-run trends in properties such as evolvability. We plan to investigate alternative policies to $R_U$, 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 [11] and Lehman, 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 samples. The highly variable histories generated by single scenarios of the $R_C$ policy, demonstrate that it would be unsafe to rely on generalisations from case studies or micro-samples that are assumed to be representative of different classes of evolving system.

**REFERENCES**


