Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

On deterministic chaos in software reliability growth models

O. Yazdanbakhsh, S. Dick (Dr.) (Associate Professor)*, I. Reay, E. Mace

Dept. of Electrical & Computer Engineering, University of Alberta, Edmonton, AB, Canada

A R T I C L E I N F O

Article history:
Received 1 December 2015
Received in revised form 31 May 2016
Accepted 2 August 2016
Available online 5 August 2016

Keywords:
Software reliability
Chaos theory
Time series analysis
Machine learning
Forecasting

A B S T R A C T

Software reliability growth models attempt to forecast the future reliability of a software system, based on observations of the historical occurrences of failures. This allows management to estimate the failure rate of the system in field use, and to set release criteria based on these forecasts. However, the current software reliability growth models have never proven to be accurate enough for widespread industry use. One possible reason is that the model forms themselves may not accurately capture the underlying process of fault injection in software; it has been suggested that fault injection is better modeled as a chaotic process rather than a random one. This possibility, while intriguing, has not yet been evaluated in large-scale, modern software reliability growth datasets.

We report on an analysis of four software reliability growth datasets, including ones drawn from the Android and Mozilla open-source software communities. These are the four largest software reliability growth datasets we are aware of in the public domain, ranging from 1200 to over 86,000 observations. We employ the methods of nonlinear time series analysis to test for chaotic behavior in these time series; we find that three of the four do show evidence of such behavior (specifically, a multifractal attractor). Finally, we compare a deterministic time series forecasting algorithm against a statistical one on both datasets, to evaluate whether exploiting the apparent chaotic behavior might lead to more accurate reliability forecasts.

1. Introduction

The smartphone revolution has embedded software into the daily lives and routines of hundreds of millions of ordinary people. We text and message instantly with friends and acquaintances across the street or across the country. We pull out our phones to search for information at the drop of a hat. Our schedules, emails, and contact lists are literally at our fingertips. Software monitors and controls all major industrial plants, the power grid, sewer and water systems, and public transportation. Government documents are now posted in official online repositories for all citizens to view. News articles, streaming media, online games, etc. pour nearly a zettabyte of data across the Internet each year [80]. Software is intimately woven through every aspect of our lives, and so software failures pose an immediate and critical danger to life, limb and property. It is thus disconcerting that software is more failure-prone than any other engineered product [33]. Software failures likely cost the U.S. economy over $78 billion per year [40]. Improving the reliability of software systems is thus one of the most crucial technical challenges of the 21st century.

While software quality is generally poor, this is not merely a case of shoddy work, but a testament to the sheer intellectual difficulty of developing large software systems. With codebases stretching to hundreds of millions of lines long, and more than $10\hat{2}0$ possible states, software systems are by orders of magnitude the most complex creations mankind has ever attempted to build [20]. By way of comparison, the human brain contains on the order of 60 trillion connections between neurons [73]; thus, it is reasonable to say that software developers are trying to build something orders of magnitude more complex than their own brains. In the face of this complexity, defect-free software is an unachievable goal; instead, we must ensure that the number and severity of the defects remaining when a product is shipped are acceptably low. It does not matter if a few pixels on an in-car entertainment display are the wrong colour; it matters a great deal if the navigation system directs drivers over a cliff.

Software quality assurance almost uniformly follows the develop-and-test paradigm. Code is written, and then its behavior is compared to its specification by executing a number of inputs that should result in known outputs. If the actual and expected outputs match, that test passes; if not, a bug has been found and must

* Corresponding author.
E-mail address: dick@ece.ualberta.ca (S. Dick).

http://dx.doi.org/10.1016/j.asoc.2016.08.006
1568-4946/© 2016 Elsevier B.V. All rights reserved.

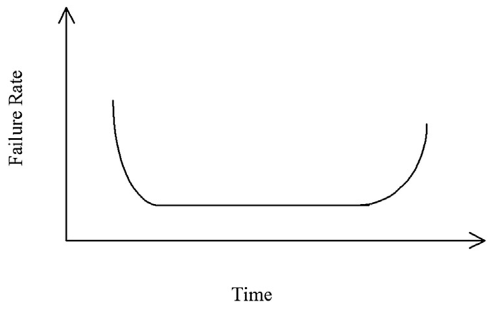

Fig. 1. Failure Rates over Time [41].

be fixed. Thus, the main question for project managers is how to determine *how much* testing is enough for the software to reach an acceptable level of quality. From a management perspective, this means that the expected future cost of maintenance and liability (responding to failures in the field) is acceptably low. When the software reaches this point, it is ready to be released. Ideally, organizations should use the development and testing history of the software system to decide when the release point has been reached. Heuristic criteria that are commonly used include requiring that the number of open minor and moderate bugs be below some threshold, that no serious or safety-critical issues are open, that the rate at which bugs are discovered and their severity are both clearly decreasing, etc. [11]. A more optimal approach would set a threshold of expected future costs below which the software may be released. In order to do this, the probability of a failure in the field over time needs to be estimated; in other words, the future reliability of the system must be forecast. In reliability engineering, this forecast is produced by a *reliability growth model* [41].

Reliability growth models for physical systems are substantially different from those for non-physical systems such as software. In a physical system, failure rates over time generally follow the "bathtub curve" depicted in Fig. 1. After an initial period of high failure rates due to residual design defects (the "infant mortality" stage), the system reaches regular operation, during which time failures are generally due to random external events (hence a constant, low rate of failures). As the system's components wear out, the failure rate again increases, until the system reaches its end of life [41]. The key difference for software is that physical systems are subject to wear and random defects in material components. By contrast, *all* software failures are due to residual design defects; in a very real sense, software never exits the infant mortality stage. Thus, instead of the exponential-class models often used for hardware reliability (e.g. the Weibull distribution [41]) counting models such as Non-Homogenous Poisson Processes (NHPPs) are frequently used for Software Reliability Growth Models (SRGMs), e.g. [53,70].

There has recently been interest in a different class of SRGMs, based on the notion that fault injection in software is a *chaotic* process, rather than a random one. Studies in [12,95] and others have examined SRGM datasets, and found signatures of chaotic behavior in them (chiefly by treating the SRGM dataset as a time series from a dynamic system, and determining a fractal dimension for the state-space attractor). The hypothesis of chaotic behavior is intriguing, but the studies above share a common weakness: publicly-available software reliability growth datasets are small, usually consisting of only a few hundred observations. There may be thousands of failures tracked by these datasets, but they have usually been summarized into counts of failures per time interval. The resulting time series are now usually considered too small to reliably test for chaotic behavior (although a recent advance allowing modeling of even short chaotic time series is reported in [71]). What is needed for a definitive test of the chaotic hypothesis are large inter-failure SRGM datasets drawn from modern large-scale software systems. This preserves the fine-grained structure of the data, and provides enough data for a reliable test. [12] performed their analysis on the largest inter-failure time datasets that were then available, which contained no more than 2000 observations (the Musa dataset [46] is an inter-failure dataset, but only contains 136 observations). This is barely adequate for testing for the simplest form of chaotic behavior (i.e. a monofractal state space attractor); an order of magnitude more data will be needed to reliably test for multifractal behavior. A further necessary test of the chaotic hypothesis is to determine if it leads to a superior model than random behavior. In our view, this means comparing nonlinear deterministic models against probabilistic ones in a forecasting experiment.

Our goal in the current paper is to employ four large-scale software reliability datasets to test for chaotic behavior, and compare probabilistic and deterministic forecasting algorithms on these datasets. We introduce two new SRGM interfailure time datasets, one derived from bug reports and change records for the Android open-source operating system (this is the 2012 MSR Challenge dataset[1]), and a second from the defect-tracking database for the Mozilla open-source Web browser. The Android dataset is an order of magnitude larger than any existing SRGM dataset, at over 20,000 observations; the Mozilla dataset contains more than 86,000 entries. We test for chaotic behavior in these datasets, as well as the two datasets from [12] by estimating the fractal dimension of the underlying attractor. We furthermore compute the multifractal spectrum of these datasets, an analysis which has not previously been attempted. We find that both new datasets, as well as one of the datasets used in [12], have multifractal attractors and are thus chaotic; we discuss how this complex geometry may have arisen in the context of normal software engineering practice. In forecasting experiments, we find that all three datasets also exhibit long-range dependencies, and that fractional ARIMA models and radial basis function networks are equally effective in forecasting them.

Our contributions to the literature are as follows: 1) we develop and analyze two new SRGM datasets, which are an order of magnitude larger than any currently available in the public domain; 2) we determine that the state-space attractors for both of these datasets, as well as an existing one, have a multifractal geometry; 3) we show that stochastic (fractional ARIMA) and deterministic (radial basis function network) models are equally effective in modeling these datasets.

The remainder of this paper is organized as follows. In Section 2, we review essential background and related work, ultimately developing our research hypotheses. We describe our methodology in Section 3, and discuss the creation of our datasets and our initial processing of them in Section 4. We present our extraction of fractal dimensions and multifractal spectra in Section 5, and our predictive modeling experiments in Section 6. We discuss and interpret the totality of our results in Section 7, and close with a summary and discussion of future work in Section 8.

2. Related work

The earliest bespoke SRGMs were the Jelinski-Moranda de-eutrophication model [32], and Schneidewind's Non-Homogenous Poisson Process (NHPP) model [70]. Musa's basic execution model [52], the Yamada S-shaped model [89] and Musa & Okumoto's logarithmic Poisson model [53] were developed a decade later. All of

[1] http://2012.msrconf.org/challenge.php.

these models are variations on the NHPP concept, which is a Poisson process whose mean value is time-varying:

$$P\left\{N(t)=k\right\}=\frac{(m(t))^k}{k!}e^{-m(t)} \tag{1}$$

where $N(t)$ is the number of events observed by time t, and $\mathrm{m}(t)$ is the mean value of the process at time t. The NHPP is one of the main frameworks for developing SRGMs; some other more recent examples include [27,29,58,92]. [25] incorporates a time-varying rate of fault removal into the NHPP model. [28] incorporates the different fault detection rates in system test and field operation into an NHPP model, as well as acknowledging that there might not be a one-to-one relationship between observed failures and faults in the software. [42] incorporate measures of testing coverage into an SRGM by modeling them as a logistic function and incorporating it into the mean-value function of an NHPP. [35] incorporates the effect of testing time and resource constraints in a multi-version software system into an NHPP mean-value function using the Cobb-Douglas equation from economics. The resulting SRGM is then incorporated into the fitness function of a genetic algorithm used for release planning. [36] develops an NHPP that incorporates both imperfect debugging and fault injection during debugging. Another family of SRGMs are rooted in Bayesian inference; the first of these was the Littlewood-Verrall model [44], with other examples including [59,60,76]. Several detailed overviews of the history of SRGM development may be found in [17,57]. A comparison between eight different SRGMs over a large corpus of datasets may be found in [84].

Recent developments in SRGMs have tended to focus on machine learning approaches. Neural network SRGMs have been investigated for over 20 years. [37] studied both feed-forward and recurrent neural network SRGMs. [79] found that neural networks were more accurate predictors of reliability than parametric SRGMs. [3] evaluated back-propagation neural network SRGMs, as did [62]. [23] developed a new recurrent neural network SRGM. [83] created an evolutionary neural network SRGM. [81] developed a specialized neural network with individual neuron transfer functions mimicked existing SRGMs, and the network is trained to weight these different models. A similar approach, but using decision trees as the weighting framework, is reported in [55], while [66] compared feedforward and recurrent neural networks for weighting. [12] employed radial basis function networks as an SRGM. [26] use recurrent neural networks to build an SRGM that models both fault detection and removal. [93] builds an SRGM from an ensemble of feedforward neural networks. [39] created an SRGM from a heterogeneous ensemble of several different neural networks, decision trees, multiple adaptive regression splines and neuro-fuzzy systems, all fused together by a backpropagation neural network acting as the meta-predictor.

Numerous other machine learning algorithms have been explored. Markov processes were used as an SRGM in [63]. [94] constructed an SRGM from multivariate adaptive regression splines. [90] used support vector regression as an SRGM; [54] does as well, but they additionally employ simulated annealing to determine the hyper-parameters of the model. [10] builds an SRGM based on genetic programming. [51] employ an ensemble of genetic programming and the Group Method of Data Handling technique, fused by a genetic-programming meta-predictor. [50] examine ant colony optimization for predicting software reliability.

SRGMs based on time-series forecasting have also been investigated, usually employing the Auto-Regressive Moving Average (ARMA) family of models. [77] employed a simple first-order auto-regressive (AR(1)) model, while the more complex Auto-Regressive Integrated Moving Average (ARIMA) model was explored in [38]. That model also incorporated a few software complexity metrics as exogenous variables to improve the prediction.

As noted in [12], software reliability models have traditionally assumed that failures occur randomly; however, there is little or no evidence offered to support this assumption. The most common situation is that authors will offer a few qualitative arguments that failures happen randomly, justifying their treatment of failure counts or interfailure times as random variables. Some examples include, “Since the number of failures occurring in infinite time is dependent on the specific execution history of the program, it is best viewed as a random variable...” [53], or simply stating, “The life lengths of the software at each stage of development are random variables...” [48]. A more detailed argument due to Musa [52] is that while programmers do not make errors at random, the location of errors within source code is random, and the test cases used to detect them are applied randomly. A moment’s reflection, however, shows that this is nonsensical. One of the fundamental rules of thumb for software testers is that faults will cluster in certain modules in a program... usually those that are more complex. Indeed, Singupurwalla & Wilson [78] admit this to be the case: “...the different input types can be envisaged as arriving to the software randomly, leading to the detection of errors in a random way. So although software failure may not be *generated* stochastically, it may be *detected* in such a manner.” The authors are thus agreeing that inputs are random, but not the errors within the source code. With the location of errors in source code disposed of, let us consider random inputs. Test cases, and the order of their presentation, are *not* chosen at random, but crafted as a part of a test plan [4,43,70,95]. The whole point is to *expose* failures as rapidly and efficiently as possible. Boundary value analysis, equivalence partitioning, dataflow testing, branch and loop testing, all rely on choosing specific, meaningful test inputs; they are emphatically *not* random. Thus, there appears to be no mechanism by which software failures would randomly arise; and yet, they are not predictable. What, then, might the underlying mechanism be?

[12] proposed that, instead of being randomly distributed in code, errors might follow a fractal pattern. Specifically, it is well-known that errors in source code are associated with a set of input values (a region of the input space) that will *trigger* the error (i.e. drive program execution through the line(s) of code containing the error, resulting in a deviation of the program state from its “correct” value.) The regions were dubbed “error crystals” in [14,15,19], and it was found that there is an inverse power-law relationship between the size (hypervolume) of an error crystal and the prevalence of crystals of that size. This is a known characteristic of fractal sets, and so Dick et al. hypothesize that “*the union of all error crystals within the input space of a software system forms a fractal subset of the input space*” [12].

A small number of studies have sought to employ chaos theory in software reliability modeling. [95] studied three well-known SRGM datasets, extracting a fractal dimension and creating a locally linear model for each. However, the datasets only range from 34 to 136 observations in size; the correlation dimension is calculated without excluding temporal correlations; and the locally linear model will not predict effectively beyond a very short time horizon. [12] corrects these failings by studying two much longer datasets (1207 and 2008 observations, respectively); performing a test for determinism and excluding temporal correlations based on the space-time separation plot; and using a nonlinear forecasting algorithm (radial-basis function networks), which should lengthen the prediction horizon. [6,7] also extract a fractal dimension from three small failure datasets. Rather than interfailure times, they focus on the relationship between the total failures observed and the time t. A straight line is fitted to a log–log plot of this statistic for each dataset. However, no attempt to remove temporal correlations has been made. [71,72] creates an SRGM out of series non-uniform rational B-splines (S-NURBS). They propose that this model, once fitted to a small dataset suspected to be chaotic, can be used to up-sample the dataset to find reliable estimates of chaotic invariants such as the correlation dimension. As the S-NURBS function is a smooth continuous curve, we are skeptical of dimension estimates from such a source; time-series behaviors at small length scales are likely incorrect due to the smoothing effect. Furthermore, the correlation dimension estimates again do not appear to have accounted for temporal correlations. Finally, a textbook by Lu [45] estimates that the well-known Musa SRGM dataset 1 has a fractal dimension of 1.49.

2.1. *Self-similarity, chaos, and long-range dependency*

Our discussion to this point has not defined *precisely* what is meant by "chaotic behavior," or its connection with fractal sets. We will now discuss the mathematical definitions of these terms. We begin with the notion of self-similarity, discussed by Mandelbrot in his celebrated work on fractal sets [47]. A self-similar set is one that appears to be identical to itself when viewed at many different length scales. The Mandelbrot set, the Cantor set, the Koch curve, are all examples of exactly self-similar sets, having an identical appearance at all length scales. *Statistical* self-similarity is a related concept, in which subsets have the same statistical properties as the entire set, even if they are not exactly self-similar (e.g. the coastline of Britain has the same statistical properties at all length scales, even if you will not find a 1-millimeter-long river Thames) [47]. Self-similar sets are an important subclass of the fractal sets, which were Mandelbrot's extension of the concept of a continuous yet nowhere-differentiable curve to geometric objects. His original definition of a fractal set has now been extended to include all sets (self-similar or not) whose topological dimension D_T is strictly greater than their Hausdorff dimension D_H [16].

Next, we discuss what we mean by *chaotic behavior*. When we informally speak of things being chaotic, we are stating that they are irregular and unpredictable. A more precise definition of chaos is based on the state-space behavior of a system. Consider two state-space trajectories (each representing some part of the time evolution of the system). A chaotic system has the unique property that any two such trajectories will diverge at an *exponential* rate through time. Now consider the forecasting problem: given the current and past states of a system, find its future state(s). In a chaotic system, any error in measuring the current state will be magnified exponentially through time, until the forecast error is as large as the actual signal amplitude, and this limit will be reached in just a short time horizon. Hence the properties of *unpredictabilitiy* and *sensitivity to initial conditions* that have been made famous in the popular press (e.g. the "butterfly effect") [16]. This uncertainty in future behavior is distinct from randomness, in that it can arise from a *purely deterministic* system; random system inputs are not required to generate it. We can term this uncertainty "irregularity," and note that it includes phenomena such as bifurcations and intermittency.

The connection between chaotic systems and fractal sets is quite simple. A dissipative system with chaotic dynamics will have a fractal state-space attractor (i.e. the locus of points that make up the attractor form a fractal set in the state space). The methods of nonlinear time series analysis revolve around detecting and quantifying the fractal geometry of that attractor. The commonly-discussed correlation dimension, for instance, is the most robust estimate of the attractor's fractal dimension available [34].

Long-Range Dependency (LRD) is a statistical property that self-similar time series exhibit. It refers to an unusually slow decay of the autocorrelations in a signal, resulting in the presence of many more "extreme" values. The more familiar "short-memory" processes exhibit an exponential rate of decay in their autocorrelations, whereas LRD processes exhibit a power-law rate of decay [69]. LRD is quantified by the well-known Hurst parameter H. Given a sequence of n observations,

$$E\left[\frac{R(n)}{\sigma(n)}\right] = C \cdot n^H, \quad n \to \infty \tag{2}$$

where $R(n)$ is the range of those n observations, $\sigma(n)$ is their standard deviation, E is the expectation operator, H is the Hurst parameter (properly the Hurst *exponent*), and C is a constant [18]. Values of H between 0.5 and 1 are associated with LRD in a stationary process, while values greater than 1 indicate non-stationarity. (Some authors have suggested that LRD represents a "middle ground" between the classes of stationary and non-stationary processes [69].) Plainly, H cannot be directly computed from a data sample, as it is defined over the limit of an infinite sequence. Several versions of maximum-likelihood estimates of H (or more precisely, the related value d, discussed in Section 3.2) have been developed [1]; as will be seen in our results, they can lead to substantially different values for d and H.

2.2. *Research hypotheses*

Our goal in this article is to perform a thorough evaluation of the hypothesis of chaotic behavior in software reliability datasets. Specifically, we advance and evaluate the following four hypotheses:

H1. Software reliability growth data, in the form of a sequence of interfailure times, forms a stationary time series with a fractal state-space attractor

Recalling the proposition from [12] – that the union of all error crystals forms a fractal set in the system input space – we ask how this proposition can in practice be tested. We are not able to directly test for fractal geometry in a system input space, because no generally-accepted definition of an "input space" *exists* for modern software systems possessing a graphical user interface (GUI). Instead, we must identify an acceptable proxy. If the above proposition is true, one of the consequences should be that the times between fault discovery should also be irregular, exhibiting self-similarity (in the distributional sense) even under a planned test sequence. Even more importantly, *random* inputs to the software (i.e. field usage by a broad user population) should also exhibit this characteristic, albeit at a lower failure intensity as inputs follow the system's operational profile instead of a test plan [28]. The fractal error set thus leads to a time series having a fractal attractor. We assert that the time series should be stationary so long as the software developer's quality-assurance process remains essentially static; plainly, any major process change would imply a non-stationarity.

H2. The time series from (H1) exhibit stationarity and long-range dependence, as measured by the Hurst parameter

The property of long-range dependence arises directly from the assertion in (H1) that the time series is self-similar, and we have already asserted the hypothesis of stationarity. Testing these two assertions via the Hurst parameter gives us an independent cross-check on our results from the methods of nonlinear time series analysis.

H3. The fractal attractor from (H1) will have a multifractal geometry

Let us consider fault injection in a large-scale software system. Obviously, there will be a large number of developers working on the software. Each of these developers will make some mistakes, injecting faults into the codebase, each associated with an error crystal. The overall (hypothetically fractal) error set of the software system is clearly the union of the individual developers' error sets. Now, consider further the errors made by developers. Every developer is a human being, with a particular mindset and way of thinking. It is our own experience – and we believe, that of developers in general – that we find ourselves making similar *kinds* of mistakes across many different programs. Not all fault injection involves repeated mistakes – hitting the wrong key late at night on one's fifth cup of coffee is an all-too-common experience – but multiple lines of evidence indirectly support this overall contention. Common-mode failures in N-version programming, for instance, have been traced to using common languages between versions, or having the different versions developed by the same corporation (due to organizational biases in how to construct software) [61]. Furthermore, studies of developer behavior show that a large fraction of defects arise from "systematic" errors, mistakes which tend to be repeated [8]. However, the errors produced by an individual developer have not proven to be any more predictable than the overall errors in a software system.

If each developer has their own, irregular pattern of errors they *tend* to make, it seems plausible that the union of their own error crystals itself forms a fractal set. We do not have adequate data to test this hypothesis; it would require reconstruction of the authorship of every line of code in the software system, an assignment of each failure to the responsible developer (a process that is never error-free, as even determining the root cause of a failure is itself an error-prone process), and then construction of an individual SRGM dataset for each developer. This is not practical, even in an open-source project; furthermore, there is a serious ethical problem, in that open-source developers never consented to have a personally-identifiable quantification of their personal biases and productivity created when they began development work. While one might argue that the act of posting their own code in a public repository constitutes implicit consent to such an analysis, ethical research concerning human subjects requires explicit, informed consent. The key point to note here is the definition of "human subjects research;" per U.S. federal regulations,[2] which closely parallels the Belmont Report [67], this is limited to interactions or interventions with another person in order to gather information, or when *the researcher* records information that can directly or indirectly identify a person. Clearly, assembling an interfailure-time software reliability growth dataset for a whole project does not fall under this definition,[3] but individualized ones likely would [86].

However, these is a direct implication of the individual fractal-set hypothesis that we can test. Individual modules in a software system are partitioned according to their functionality, not according to their input sets; thus the system's response to a given input is potentially (perhaps even likely) defined by code written by many different developers. Thus, it is likely that the fractal sets from each developer will overlap in a complex manner in forming the overall error set. Each individual fractal furthermore likely has its own characteristics – including its own scaling law (dimensionality). This precisely matches the definition of a multifractal set: a fractal set with many different dimensionalities, each holding on a monofractal subset (having a single, constant scaling law) of the overall object. Thus, following the same arguments as in (H1), we hypothesize that SRGM datasets have multifractal attractors [64].

H4. A nonlinear deterministic forecasting algorithm will exhibit lower overall forecast error on the datasets from (H1) than a stochastic forecasting algorithm

[2] U.S. Code of Federal Regulations,Title 45, Part 46, Protection of Human Subjects.
[3] The authors' Research Ethics Board determined that the research reported herein was not human subjects research, and did not require further review.

If indeed SRGM data arises from a nonlinear deterministic process, then a forecasting algorithm that exploits this characteristic should be superior to one that does not. Equally, if SRGM data is fundamentally stochastic with long-range dependencies, a model that utilizes this fact should be superior to one that does not.

3. Methodology

3.1. Datasets

Our four datasets are depicted in Figs. 2–5.

The ODC1 and ODC4 datasets were collected by the IBM Corporation during their Orthogonal Defect Classification project. They consist of 1207 and 2008 bug reports, respectively, with each report dated [46]. These datasets are not originally interfailure times, but we follow a recommendation from [46] to convert them to interfailure times by assuming that the failures arrive at random times during the day. Interestingly, this is very similar to a recommendation in [34] for dealing with discretization noise: add white noise in the range [−0.5,0.5] to the signal, then process it with a nonlinear noise reduction algorithm. These two datasets are identical to those analyzed in [12].

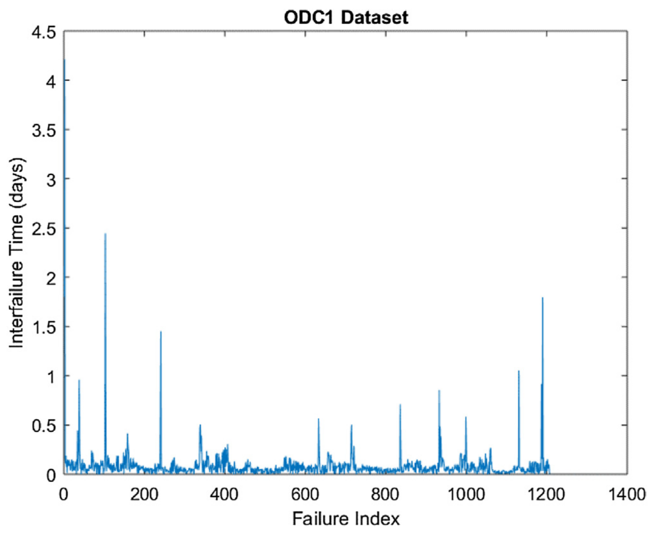

Fig. 2. ODC1 Dataset.

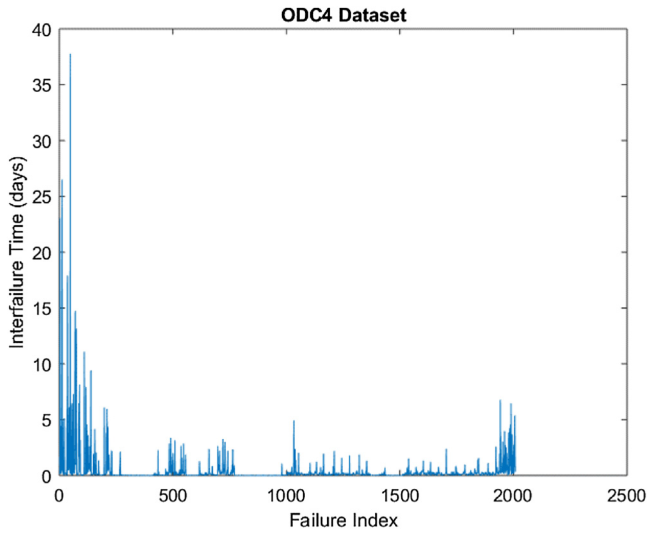

Fig. 3. ODC4 Data.

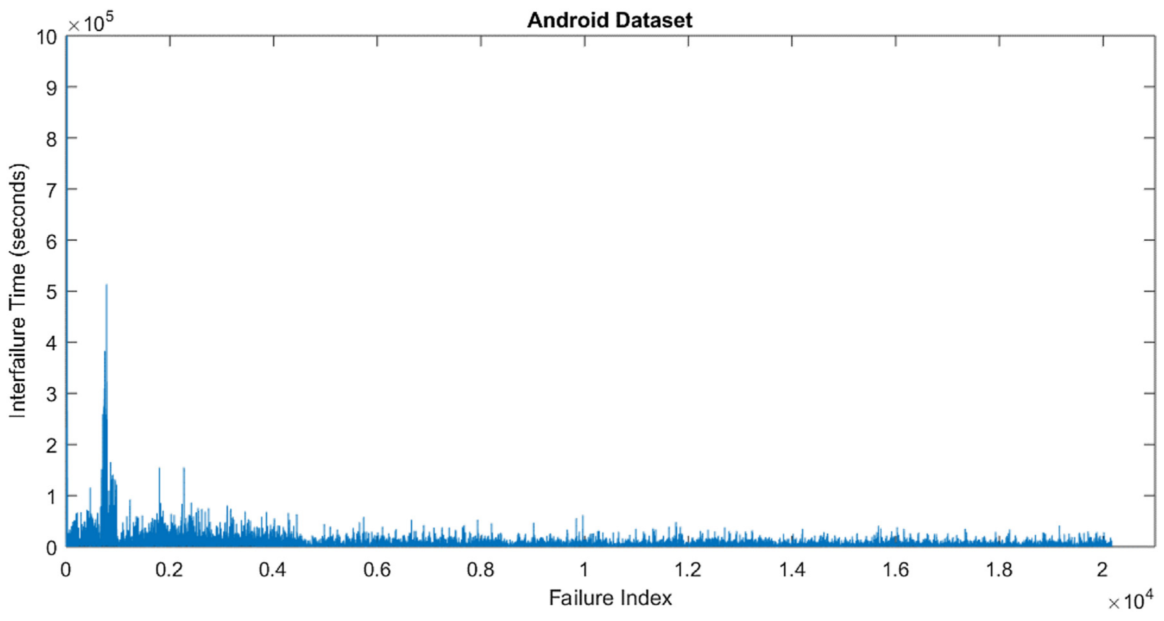

Fig. 4. Android dataset.

The Android dataset was derived from the 2012 Mining Software Repositories Challenge dataset, consisting of bug reports and changes from the Android open-source project. We directly process the date/time stamps on each bug report, computing the interfailure times to an accuracy of one second. The resulting dataset contains 20,168 observations.

The Mozilla dataset is based on a copy of the Bugzilla defect-tracking database used by the Mozilla project, which was donated to our lab in 2003. Bugzilla is a front-end to a standard MySQL database; documentation on the database is limited, but the database schema was reverse-engineered in [68]. In previous work, a software defect prediction dataset was extracted from this source [56]; we have now also derived an SRGM dataset from it. We select all non-duplicate, confirmed bugs that were assigned again the development trunk since 1999 (as with other long-lived software systems, Mozilla exists as a source tree with multiple release branches; the "trunk" is the evolving codebase that represents ongoing development). We again extract the timestamps of all selected bugs, and record the interfailure times between them, yielding 86,077 observations.

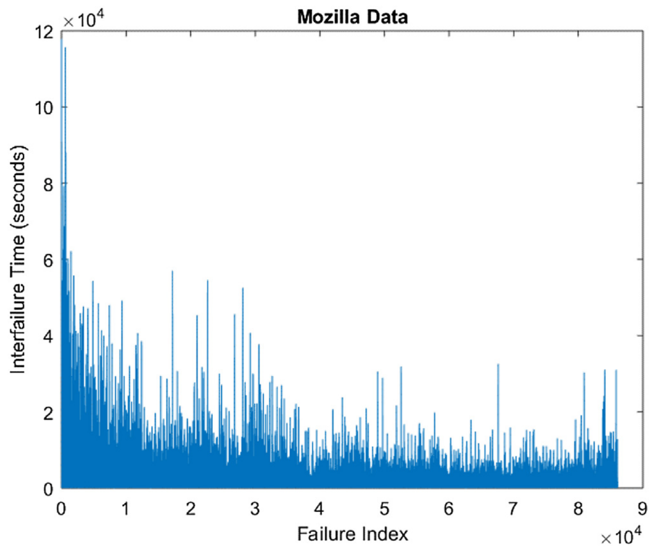

Fig. 5. Mozilla Dataset.

3.2. Chaotic invariants: preliminaries

A time series based on observations of a deterministic process can be considered a low-dimensional, nonlinear projection of that system's state space trajectory (the time history of states the system has passed through in its evolution during the observation period). Takens' embedding theorem [82] shows that it is possible to reconstruct this trajectory in an "embedding space" that is equivalent to the original state space. The method requires us to concatenate time-ordered prior observations (lags of the time series) into vectors of sufficiently high dimensionality (at least $2D+1$, where D is the dimensionality of the original space [5]). That dimensionality has to be determined heuristically; one very common method for doing so is the technique of False Nearest Neighbors (FNN). In essence, we compare the one-step-ahead time evolution of points that initially lie within an e-neighborhood of each other, and determine if any of them cease to be close neighbors after that time step. Any that do are considered "false" neighbors, in that they only appeared to be within the neighborhood due to unresolved projections of the embedding space. If a substantial number of false neighbors are found, this indicates that the current embedding dimensionality is insufficient [34].

In addition to the dimensionality, the embedding *delay* must also be determined heuristically. This parameter controls the temporal width of an embedding vector; we could select each consecutive prior observation, every second one, every third, and so on. Mathematically, all delays are equivalent (in the limit of an infinite noise-free time series); but in practice, the value of the delay parameter can have a substantial impact on further analyses. Heuristics for the embedding delay include the first minimum of the time-delayed mutual information of the signal, and the first zero-crossing of the autocorrelation function [34].

In [5], the FNN technique was compared against three methods of directly estimating the attractor dimension: Kégl's algorithm (also called the Box-Counting dimension), the Grassberger-Procaccia algorithm (the well-known correlation dimension), and the Levina-Bickel algorithm (a maximum likelihood estimate of the correlation dimension). Generally the embedding dimensionality obtained by FNN was similar to that inferred from the three attractor dimensionalities, and yielded similar errors when a support-vector regression algorithm was trained on the resulting embedding vectors. Thus, the FNN technique was supported as an effective means of estimating the embedding dimension. The lone exception was on a chaotic dataset: the well-known Santa Fe A (laser) dataset from the Santa Fe times series forecasting competition described in [87]. This is the output amplitude of a far-infrared laser governed by the Lorenz equations for turbulent flow. On this dataset, the FNN method found that a three-dimensional embedding would suffice; the other methods found a "true" dimensionality of 2.0–2.35, implying the need for a five-dimensional embedding. The model based on the FNN findings yielded a substantially higher mean squared error than the others. Note that in [5], the time delay is always identically 1; no examination of other values is attempted, and indeed the time delay is not even discussed as an embedding parameter.

Let us now consider an earlier analysis of the Laser A data, from its original developers. An analysis of this dataset in [30] (which did not use the FNN technique) found that the embedding dimension needs to be at least 7; a time delay of 2–3 steps was the most effective in exposing the attractor geometry. As the authors again note, their dimension estimates do not formally depend on the delay time, but a "good" choice does help the analysis. More specifically, changing the time delay alters the total amount of the attractor that is covered by the delay vector. A fractal attractor takes the form of a complex orbit that never quite repeats itself, and a delay vector will sample this orbit. Increasing the delay between lags thus had the effect of downsampling the orbit, and capturing more of the spatial features [30]. In summation, the findings in [5] are at variance with established literature on this dataset. It seems plausible that fixing the delay to 1 might be part of the reason for this discrepancy; delay vectors must be formed to identify nearest neighbors, meaning that FNN is in part dependent on the time delay parameter. Indeed, our own prior experience indicates that the estimated dimensions form the FNN technique do change when the delay parameter is fixed at 1 versus being heuristically determined [91].

Noise reduction is carried out by taking an ε-neighborhood around each delay vector, and then replacing the central coordinate (the $m/2$-th component of an m-dimensional vector) with the average value of that coordinate in the ε-neighborhood. We run this algorithm until convergence, for several values of ε, inspecting a phase plot of the resulting time series after each run; we discard those runs that produce obvious phase-space artifacts, and keep the run using the largest value of ε within the remaining group. This should satisfy the requirement that ε be large enough to cover the amplitude of the signal noise, but smaller than the average radius of curvature in the state trajectory [34].

The methods of nonlinear time series analysis (e.g. extracting the correlation dimension) are effective at characterizing deterministic behavior when it is known to be present. However, they are not presently suitable for determining *whether or not* determinism is present. This being the case, a separate test for the existence of determinism in a dataset is required. Currently, no test statistic is known that distinguishes between determinism and randomness for a given time series. The best option currently available is a Monte Carlo approach, known as the method of *surrogate data* [22,34].

The fundamental idea is simple: we choose our null hypothesis to be that the data under analysis is a product of some stochastic process. We then create a number of artificial data sequences (surrogates) that match that hypothesis, while preserving the statistical moments of the original time series. We choose a test statistic that measures some aspect of determinism, and compare the values of that statistic on the original data and the surrogates; if the original data is substantially different than the surrogates, we reject the null hypothesis in favor of the data originating from a deterministic process [22,34].

How we generate the surrogates controls how general our null hypothesis actually is; we want them to represent the broadest possible subset of the class of stochastic processes. In the literature, the most general null hypothesis available is that the data come from a linear Gaussian process, which has been distorted by a monotonic nonlinear observation function. To generate these surrogates, we first take the Fourier transform of the original data, then multiply each element by a random phase with unit magnitude. This maintains the power spectrum of the sequence the same even though the data has been randomized. Transforming back, we have a randomized sequence with the same power spectrum. We use this sequence as a template for a shuffle of the original data, matching the rank-ordering of the random sequence. This is now one surrogate dataset [22,34].

The test statistics to be used must quantify some aspect of determinism. For instance, a deterministic prediction algorithm should yield a lower prediction error on the deterministic original data than on the surrogates. Alternatively, a deterministic process will not be symmetrical under a reversal of time, whereas a stochastic one should be. The time reversal asymmetry statistic is given by [22,34]:

$$x_i \cdot x_{i+1}^2 - x_i^2 \cdot x_{i+1} \tag{3}$$

where x_i is the i-th observation of the time series. At this point, we must acknowledge that the distribution of the above statistics (or any of the others that have been used in the literature) are unknown, and quite possibly strongly non-normal. Thus, we cannot compute a critical value for a given level of significance. Instead, we use a rank based approach: for a desired two-tailed significance of α we generate $(2/\alpha)$-1 surrogates, and reject the null hypothesis iff the statistic value for the original data is either greater than that for all surrogates, or less than them all [22,34].

Long-range dependency is a characteristic of self-similar data, which we can test for relatively simply. We begin with the Auto-Regressive Integrated Moving Average (ARIMA), wich is a three-parameter forecasting model. An ARIMA(p,d,q) model is defined by the equation

$$\left(1-\sum_{i=1}^{p}\alpha_i L^i\right)\cdot(1-L)^d X_t = \left(1+\sum_{j=1}^{q}\beta_i L^i\right)\cdot\varepsilon_t \tag{4}$$

where p is the order of the auto-regression part of the model, $_{\alpha i}$ is the auto-regression coefficient of the i-th lag, L is the lag operator, d is the order of differencing, X_t is the t-th element of a time series, q is the order of the moving-average portion of the model, $_{\beta i}$ is the moving-average coefficient of the i-th lag, and ε_t is the t-th error term; errors are assumed to be independent, identically distributed random variables drawn from a zero-mean normal distribution [49]. In an ARIMA model, p, d, and q are necessarily integers. The FARIMA model generalizes ARMIA by allowing d to be a non-integer. By construction in [24], one very interesting result of that generalization is that d is directly related to the well-known Hurst parameter H via $d = H - 0.5$. Thus, values of $d < 0$ indicate a stationary short-memory process, $0 \le d < 0.5$ indicates a stationary long-memory process (i.e. self-similarity), and $d \ge 0.5$ indicates a non-stationary process [1,21,65]. Our final descriptive analysis thus focuses on estimating the fractional differencing parameter d for our time series.

3.3. Monofractal and multifractal analysis

Given that our surrogate data test has led us to reject the hypothesis of a stochastic process, we still need to determine if the time series appears to be stationary. While the value of the Hurst parameter provides evidence for stationarity, the space-time separation plot provides additional, vital information. Curves of constant probability for pairs of points to be within an e-neighborhood of one another when they are separated by different lengths of time are plotted; if all the curves reach a plateau or level oscillation, then the time series is stationary. Furthermore, the number of time steps required to reach that plateau or oscillation defines the extent of temporal correlations in the data [22,34].

We can now proceed to test for the presence of chaotic behavior. As discussed, this can be confirmed by the presence of a fractal attractor in the reconstructed state space. In the case of a finite and potentially noisy time series, identifying a fractal attractor is best attempted by determining the value of known invariants of a fractal attractor, and inquiring if they indicate its presence. The maximal Lyapunov exponent (which defines the average exponential rate at which trajectories in state space diverge) is one such; the correlation dimension (an estimate of the fractal set's dimensionality) is another. The Grassberger correlation dimension is currently accepted as the most robust of the fractal invariants, and is the one most commonly computed in studies of chaotic time series (and specifically in studies of chaos in SRGM data [7,45,72,95]). Firstly, the correlation sum is given by

$$C(e,m) = \frac{2}{(N-n_{\min})\cdot(N-n_{\min}-1)}\cdot\sum_{i=1}^{N}\sum_{j=i+n_{\min}}^{N}\Theta(e-\|\vec{x}_i-\vec{x}_j\|) \tag{5}$$

where N is the number of m-dimensional delay vectors x, e is a neighborhood, Θ is the Heaviside step function, and $n_{\min}$ is the Theiler window, whose function is to exclude points closer than $n_{\min}$ time steps from the correlation sum. This is done to prevent temporal correlations from biasing the dimension estimate; only *spatial* correlations properly define the attractor. The Theiler window is determined from the space-time separation plot, as previously discussed. We compute the correlation sum for a range of neighborhood sizes e, and a range of embedding dimensions m, and then plot the slopes of the correlation sum against the neighborhood size e on a log-linear plot. For each embedding dimension, there will be a different curve; if these curves all saturate in a common plateau, the y-value of that plateau is the estimated correlation dimension [22,34].

The correlation dimension is one of many dimensions that can be computed for a given attractor. However, one important caveat is that they all assume that the attractor is a monofractal; that is, there is a single scaling law that defines all of the self-similarity in the time series. In general, this is not so; the attractor may in fact be a deeply interwoven union between *many* fractal sets, each with its own scaling law; thus a single aggregated dimension estimate does not give a true picture of the attractor geometry. Such objects are referred to as *multifractals*, and they can be expected to possess a *spectrum* of dimensions. Again, there are many ways to estimate this spectrum; the Large Deviation Spectrum (implemented in FracLab) is one of the most robust [85].

We begin with the binomial measure μ, defined recursively as follows: randomly partition the unit interval into sub-intervals I_0 and I_1, dividing the total probability mass of the parent interval between them as m_0 and $1-m_0$, respectively. Then, recursively repeat this operation on each sub-interval, denoting the new children intervals as $I_{\varepsilon 1.\varepsilon n}$, $\varepsilon i \in \{0,1\}$. The binomial measure of any $I_{\varepsilon 1.\varepsilon n}$ is then the product of the masses $m_{\varepsilon 1}\cdot\ldots\cdot m_{\varepsilon n}$ defined at each partitioning. Now, consider how the measure changes as $I_{\varepsilon 1.\varepsilon n}$ shrinks in size to a single point x; the total mass remaining in $I_{\varepsilon 1.\varepsilon n}$ decreases at an exponential rate, roughly $2^{-n\alpha(x)}$. These values $\alpha(x)$ are given by

$$\alpha_n(x) = \frac{\log\mu(I_{\varepsilon 1\ldots\varepsilon n})}{\log|I_{\varepsilon 1\ldots\varepsilon n}|} \tag{6}$$

$$\alpha(x) = \lim_{n\to\infty}\alpha_n(x) \tag{7}$$

and are known as the coarse Hölder exponents of μ. In essence, the Hölder exponent defines the scaling behavior of $I_{\varepsilon 1.\varepsilon n}$; it is its fractal dimension. The overall geometry of *all* the $I_{\varepsilon 1.\varepsilon n}$ is defined by the spectrum of Hölder exponents; each one will hold on a fractal subset of the overall unit interval, with these sets closely interwoven. The concept of Hölder exponents generalizes beyond the binomial measure to measures of fractal sets in general; the spectrum of Hölder exponents is one way to express the spectrum of dimensions for a multifractal. To test for multifractal behavior, we compute the relationship between a given Hölder exponent α, and its frequency of occurrence f_G in the fractal object:

$$f_G(\alpha) = \lim_{\varepsilon\to 0}\lim_{\delta\to 0}\sup\left(\frac{\log N_\delta(\alpha,\varepsilon)}{\log(1/\delta)}\right) \tag{8}$$

where N_δ is the number of cubes C of size δ for which the Hölder exponent $\alpha(C)$ is close to α. A plot of $f_G(\alpha)$ against α will be a single point for a monofractal object, and a curve for a multifractal [64].

3.4. Predictive modeling

Our first consideration is the choice of models to compare. We expect that long-range dependency will be an important characteristic of all four of our datasets. Thus, the models we employ should have a strong track record of modeling LRD data. On the deterministic side, prediction models such as neural networks, support vector regression, and regression trees are all possibilities. In the literature on chaotic time series forecasting, the Radial Basis Function Network (RBFN) is a common choice [34], and a well-known and −understood kernel-based algorithm. They scale well to large time series (in previous work, we have used them to model time series of up to 1 million observations [13]). We thus adopt them as our deterministic model. On the statistical side, Fractional Auto-Regressive Integrated Moving Average (FARIMA) models [24] are a very common choice for modeling LRD data (with network traffic data being a prime example [74]). The principal alternative to FARIMA models appears to be the Generalized Auto-Regressive Conditional Heteroskedasticity (GARCH) family of models [2]. We make the (admittedly arbitrary) choice to use FARIMA models in this research.

Our experimental design is a typical approach in the forecasting community. We execute a chronologically-ordered single-split of each dataset, reserving the most recent partition as a holdout test sample. Parameter exploration is then carried out in the earlier, "training" portion of the dataset. For the RBFN, we form delay vectors using the delay and dimension parameters determined in Section 4, and then carry out parameter exploration using a tenfold cross-validation experimental design in the "training" partition only (time ordering can be ignored in this step because a delay vector in theory encapsulates all of the state-space dynamics necessary to predict the next observation). Tenfold cross-validation is not appropriate for the FARIMA model; we instead determine maximum likelihood estimates of the parameters on the training set, and model performance is then immediately evaluated on the testing set.

The performance measure we employ is the Mean Absolute Error (MAE). While this measure may not be as commonly used in the machine learning literature as the Mean Square Error (MSE) and its derivatives, MAE has significant advantages over it for our situation. As discussed in [31,88], MSE is highly sensitive to outliers, and thus model comparisons based on it are biased by extreme values. We do not assert that this invalidates RMSE (there likely are situations where one wishes to compare the behavior of models under unusual or stressful conditions), but it is not congruent to our purpose in this experiment. Our intent is to discern whether

FARIMA or RBFN models are *generally* more accurate SRGM predictors. This means we are principally interested in comparing the ordinary behavior of these models, and hence a measure that is less sensitive to outliers is desirable. We thus elect to use the MAE measure, as it is a commonly-used measure in the forecasting community that meets this requirement.

To assess the statistical significance of our results, we employ two tests. We first check for statistical significance using the paired Wilcoxon signed-rank test, comparing the prediction results for FARIMA and RBFN on each holdout sample. The Wilcoxon test is a rank-based (non-parametric) statistic, and thus does not assume a distributional form for the samples being compared; we have no evidence that the residuals from the RBFN or FARIMA models would be normally distributed, and the LRD nature of the datasets makes it plausible that they are *not*. One first calculates the absolute difference $|x_i - y_i|$ and the sign of the difference $sgn(x_i - y_i)$, $i = 1,2,\ldots,n$ for all n samples in the holdout set, where x_i is the i-th prediction from FARIMA and y_i is the i-th prediction from the RBFN. Samples are ranked by their absolute difference, and their rank R_i recorded. We then compute the test statistic

$$W = \sum_i sgn(x_i - y_i) \cdot R_i \tag{9}$$

The null hypothesis is that the ranks R_i are symmetrically distributed around 0 (i.e. there is no systematic difference in the prediction errors between FARIMA and RBFN). Under this null hypothesis, the test statistic W follows a known but complex distribution; for large n this distribution can be approximated by the Normal distribution [75].

Given that our holdout sets can contain tens of thousands of samples, we also compute an effect size for each dataset. Cohen's d measures the "strength" of a difference (properly the standardized mean difference) between two sampled populations as

$$d = \frac{\bar{x} - \bar{y}}{s} \tag{10}$$

where $\bar{x}$ and $\bar{y}$ are the sample means of population x and y, and the pooled sample standard deviation s is given by

$$s = \sqrt{\frac{(n-1)s_x^2 + (n-1)s_y^2}{2 \cdot n - 2}} \tag{11}$$

for the case when the two sample sets are of equal size, where n is that size, ${s_x}^2$ is the sample variance of x, and ${s_y}^2$ is the sample variance of y. Cohen suggests that an effect size less than 0.2 indicates no real relationship, $0.2 \le d < 0.5$ indicates a weak effect, $0.5 \le d < 0.8$ is a moderate effect, and $0.8 \le d$ indicates a large effect [9].

4. Characterization of the datasets

In this section we discuss the initial processing of our four datasets. We determine a delay embedding for each one, perform noise reduction, test for apparent deterministic behavior, test for stationarity, and test for the presence of long-range dependencies. The first four procedures are carried out using the TISEAN software package, while the latter was performed using several packages in R.

4.1. Findings

The embedding delay and dimension we determined for the Android and Mozilla datasets, as well as those determined for ODC1 and ODC4 in [12], are given in Table 1. We thus reconstruct the ODC1 dataset in 3 dimensions with time delay 2; the ODC4 dataset in 2 dimensions with time delay 5; the Android dataset in 18 dimensions with time delay 2; and the Mozilla dataset in 16 dimensions with time delay 6. Noise reduction was carried out using the locally-constant scheme previously discussed.

Table 1
Embedding delays and dimensions.

	Dimensions	Delay
ODC1	3	3
ODC4	2	5
Android	18	2
Mozilla	16	6

Table 2
Nonlinear Prediction Error Statistics on Surrogate Data, $\alpha = 0.05$.

	Original	Surrogate Min	Surrogate Max
ODC1	0.131924823	0.129554912	0.132290289
ODC4	0.742800474	1.1041894	1.54841113
Android	13364.1104	128971.334	13370.2861
Mozilla	2719.82129	2805.56055	2812.08423

Table 3
Time Reversal Asymmetry Statistics on Surrogate Data, $\alpha = 0.025$.

	Original	Surrogate Min	Surrogate Max
ODC1	0.145726815	−0.199396521	0.274040103
Android	−34763.4297	−18730.5176	25476.2246

Table 4
Estimates of the Fractional Differencing Parameter d.

	arfima	*fracdiff*	*fdGPH*	*fdSpresio*
ODC1	0.04608492	0.3537307	0.0001439979	0.03152876
ODC4	0.2988599	0.1859035	0.6238698	0.5321577
Android	0.2536651	0.1877835	0.3585572	0.3223998
Mozilla	0.2074765	0.2188145	0.3598718	0.2927557

In testing for determinism we first employ the error from a locally-constant prediction algorithm as our test statistic (this is a different application of the same technique we used for noise reduction). The value of the statistic for the original datasets, along with the minimum and maximum for the surrogates, are presented in Table 2.

From Table 2, ODC4 and Mozilla appear to be deterministic, while the explanation of a Gaussian process cannot be rejected for ODC1 and Android. We cross-check these last two results using the time-reversal asymmetry statistic (anecdotally considered less powerful than the prediction statistic). As this is a repeated test, we apply Bonferroni's correction to the significance level, meaning we must test at $\alpha = 0.025$ to achieve an actual significance of $\alpha = 0.05$. These results are presented in Table 3.

In this second test, we are again unable to reject the null hypothesis for ODC1, but we do reject it for the Android dataset.

We next test for stationarity. Space-time separation plots for all four datasets are presented in Fig. 6(a)–(d). Based on these results we set the Theiler windows as follows: 20 times steps for ODC1, 25 time steps for ODC4, 30 time steps for Android, and 36 time steps for Mozilla.

We now evaluate research hypothesis (H2). There are a number of approaches for estimating the fractional differencing parameter d (which is properly defined over an infinite time series) for a finite and possibly noisy time series. We employ four methods, which are implemented in the R environment: estimates from the functions *arfima* and *fracdiff*, by Hyndman; the Geweke and Porter-Hudak Estimator, implemented by the *fdGPH* function; and the Spresio Estimator, implemented by the *fdSpresio* function. We present these results in Table 4.

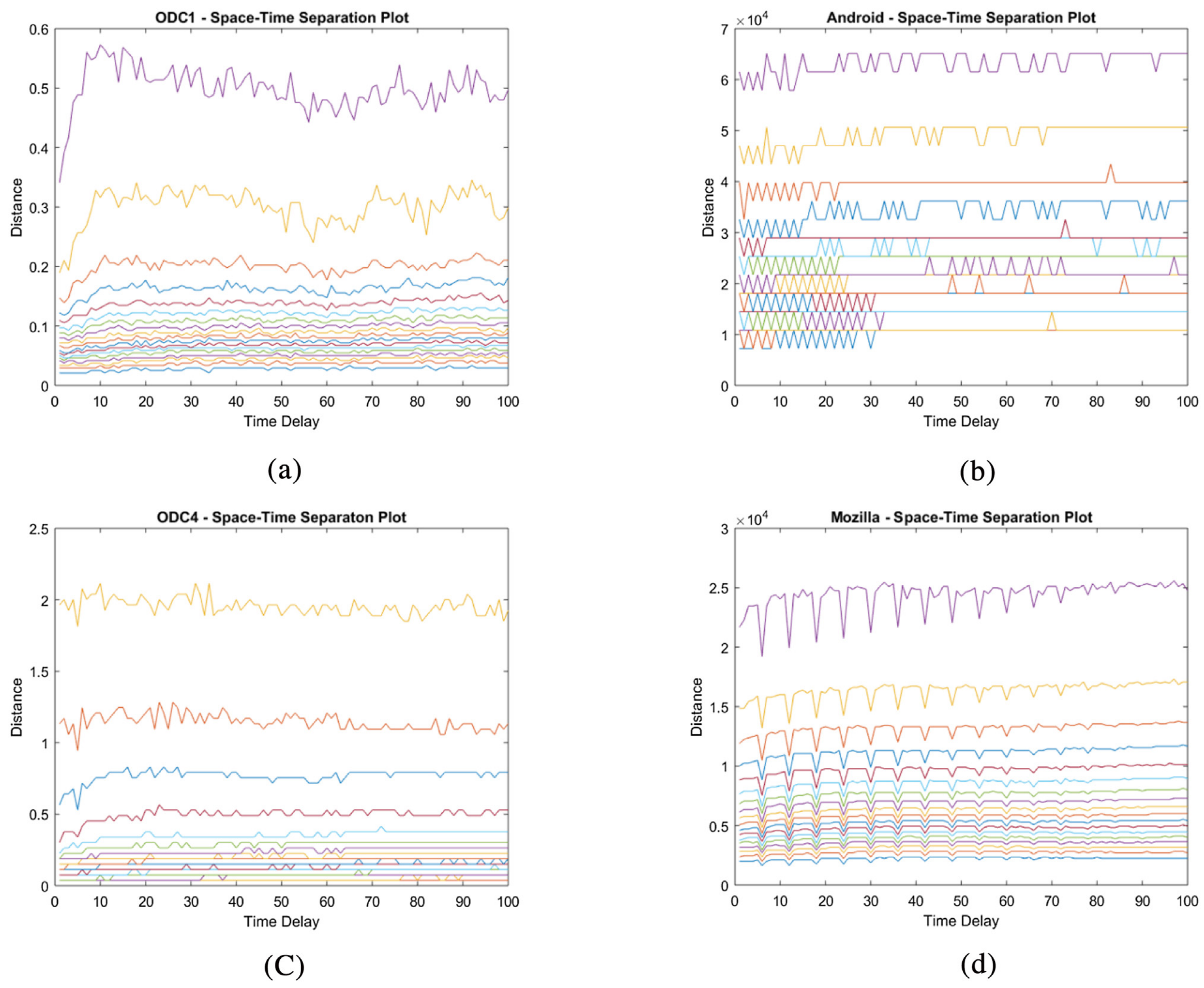

Fig. 6. Space-Time Separation Plots: (a) ODC1, (b) ODC4, (c) Android, (d) Mozilla.

We find that the d estimated for ODC1 is very near to zero for three of the four estimators, indicating a very weak long-range dependence (in fact, it seems close to the transition to a short-memory process, which by definition would not be self-similar). However, all estimates do still indicate a stationary LRD process, so (H2) is supported on this dataset. Two of the estimators indicate that ODC4 is a non-stationary process ($d > 0.5$); the other two indicate a stationary LRD process. Thus, the evidence for (H2) on this dataset is inconclusive. All estimators indicate that the Android and Mozilla datasets are stationary LRD, so (H2) is supported on both datasets.

4.2. Discussion

Only two of our four datasets yield unambiguous results: Mozilla appears to be stationary based on both the space-time separation plot and the estimated value of d. It furthermore appears to arise from a deterministic process. A Theiler window has been determined, and we will proceed to extract fractal invariants from, and conduct predictive modeling on, this dataset.

The case of ODC1 also seems clear. The dataset is clearly stationary, but LRD is very weak. Both our initial test and re-test for deterministic behavior failed to reject the null hypothesis. It is therefore inappropriate to attempt to extract fractal invariants, and we will not do so. We will, however, include ODC1 in our predictive modeling experiments.

ODC4 is also clearly deterministic, and the space-time separation plot indicates it is stationary. Two of the four estimators of d contradict that last point, but the remaining two support it. Based on the totality of the evidence, we will proceed with extracting fractal invariants and predictive modeling in this dataset.

The Android dataset is the most muddled picture. We failed to reject the null hypothesis in our test for determinism using the prediction error statistic, but did reject it using the (supposedly) less powerful time reversal asymmetry statistic [34]. Both the space-time separation plot and the estimate of d show that the dataset is stationary, and it exhibits LRD. What then to make of the surrogate data results? We ultimately choose to accept the time-reversal result, for the following reason: the hypothesis of a linear Gaussian process means that our surrogates are *not* LRD, but the original dataset clearly *is*, from the estimates of d. Furthermore, unlike ODC1, LRD is clear and strong in this dataset; thus, it stands to reason that the null hypothesis should be rejected by any test that identifies and takes advantage of "structure" in the dataset. The lack of rejection by the prediction error statistic is thus puzzling, but the totality of evidence available seems to favor accepting the Android dataset as likely being deterministic. Given this decision, we will extract fractal invariants and conduct predictive modeling in this dataset.

5. Fractal dimension and multifractals

In this section, we evaluate both (H1) and (H3). These hypotheses are not independent of one another; if a dataset has an attractor with a multifractal geometry (H3), then the attractor is a fractal and the time series exhibits chaotic behavior. On the other hand, (H1) does not imply (H3), as the attractor could be a monofractal. We begin with the correlation dimension; the slopes of the correlation sum for each dataset are presented in Figs. 7–9.

For ODC4, no plateau is visible; however, there is a region of common behavior from $e = 3 \times 10^{-1}$ to 6×10^{-1}. This common behavior seems to run from $C(e,m) = 0$ to $C(e,m) = 1$. In the Android dataset, a possible scaling region is visible from $e = 2 \times 10^4$ to 3×10^4. The clearest evidence seems to come from the Mozilla dataset, where there appears to be a scaling region from $e = 3 \times 10^3$ up to 10^4.

The large deviation spectrum for our three datasets is depicted in Figs. 10–12, giving clear indications of multifractality. A very broad range of scaling exponents occurring very frequently can be found in all of our datasets. This perhaps is a reason why the evidence from the correlation dimension is so weak; finding a single average value

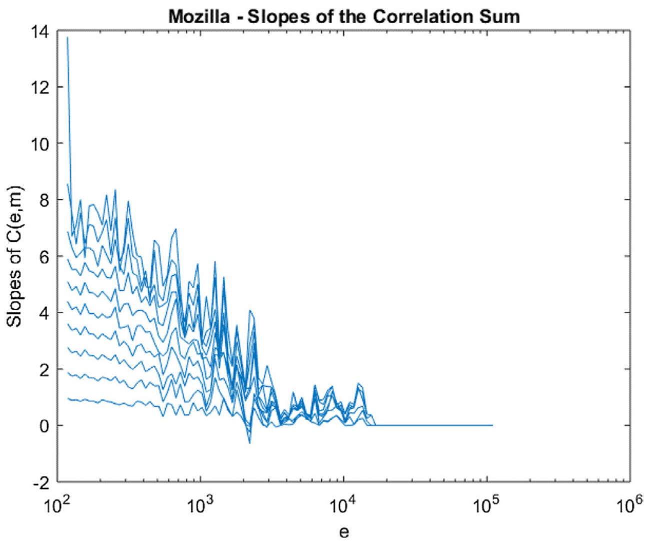

Fig. 9. Slopes of the Correlation Sum for Mozilla.

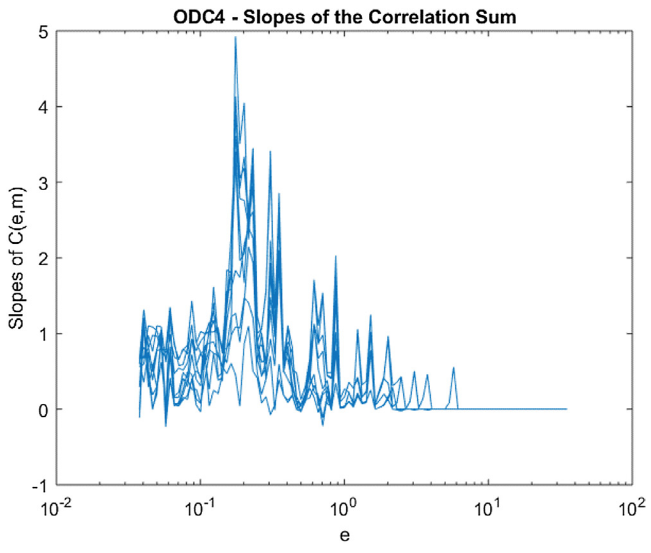

Fig. 7. Slopes of the Correlation Sum for ODC4.

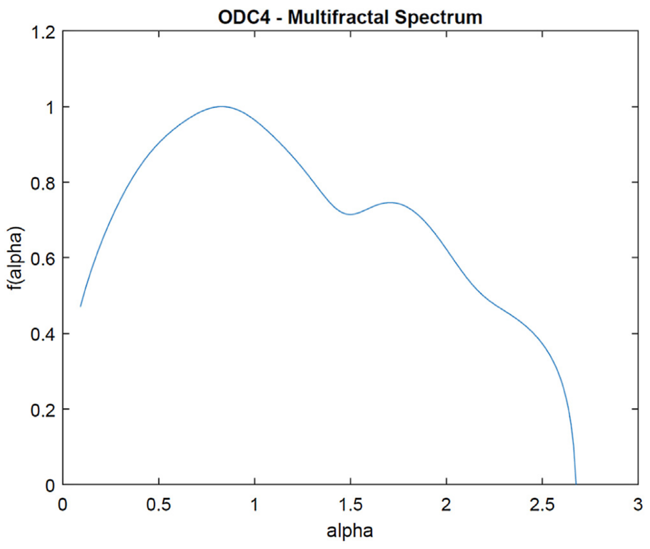

Fig. 10. Multifractal Spectrum for ODC4.

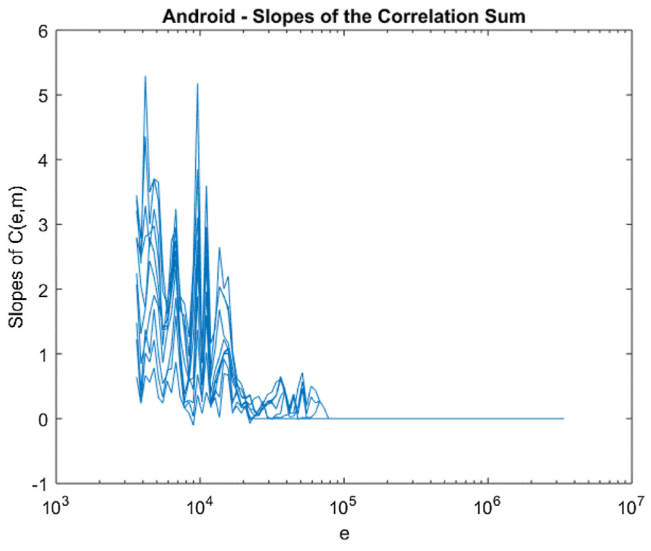

Fig. 8. Slopes of the Correlation Sum for Android.

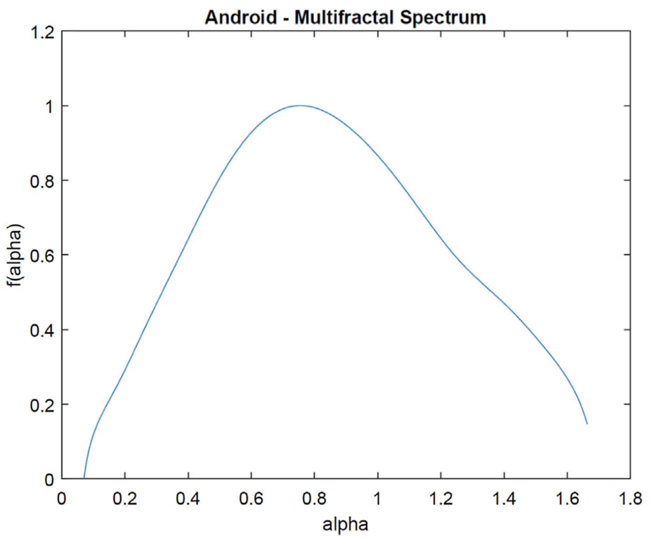

Fig. 11. Multifractal Spectrum for Android.

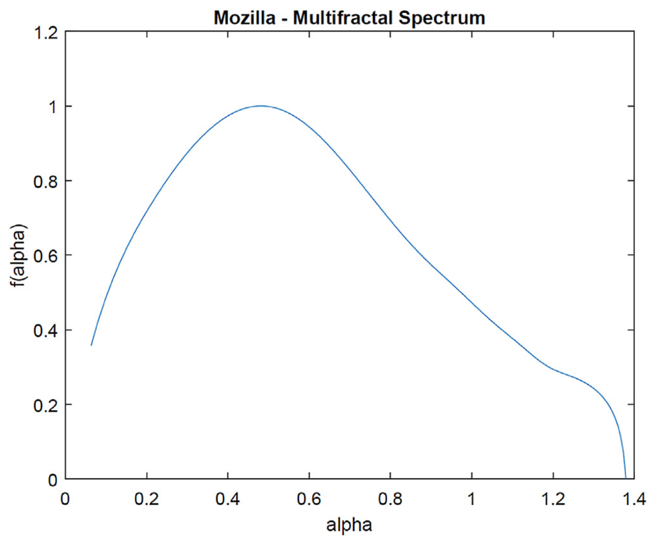

Fig. 12. Multifractal Spectrum for Mozilla.

Table 5
SRGM Prediction Experiment Results (MAE).

	FARIMA	RBFN	*W*-Statistic	*p*-Value	Effect Size
ODC1	0.032	0.028	2.7e+4	8.3e-7	0.055
ODC4	0.031	0.033	1.2e+5	3.6e-4	0.020
Android	0.064	0.066	1.1e+7	1.7e-4	0.020
Mozilla	0.027	0.027	1.9e+8	<2.2e-16	0.018

for the fractal dimension is going to be at once difficult and less meaningful, as the variance in scaling behaviors is so large in any of the attractors. Based on the totality of evidence in this section, we find that both (H1) and (H3) are supported for all three datasets.

6. Comparing statistical versus deterministic models

We now evaluate hypothesis (H4), comparing RBFN and FARIMA models on our four datasets.

6.1. Experimental results

We compare the final out-of-sample test results in Table 5. We present the MAE for FARIMA and RBFN on each dataset in the second and third columns, respectively. In the fourth column, we provide the *W*-statistic between the RBFN and FARIMA results, with its *p*-value for the Wilcoxon test in the fifth column. The sixth and final column is the effect size between the RBFN and FARIMA results.

From Table 5, we see that there are only small differences in the prediction error between FARIMA and RBFN on each dataset. The p-values of these differences are all significant at the 0.001 level, normally a very strong statement; but the effect sizes are all well under 0.1. Examining the *W*-statistic column actually yields an explanation for these contradictory results: the values of the *W*-statistic correlate perfectly with the size of the datasets. The very strong significance results on these datasets appear to be due to large sample size effects (which are a known source of bias in statistical hypothesis tests), not an inherent difference in the average performance of the FARIMA and RBFN methods. We thus interpret Table 5 as revealing a statistical tie between FARIMA and RBFN on each of our four datasets. Thus, (H4) is not supported.

7. Discussion

In this work, we have gathered a considerable amount of evidence on the possible presence of chaotic behavior in software reliability growth datasets – and by extension, the existence of chaotic dynamics in software development and quality assurance. Here, we summarize and interpret the various lines of evidence.

We first note that our predictive modeling experiments did not go as expected. Instead of a clear distinction between model performances when nonlinear determinism is apparent, both RBFN and FARIMA models performed equally well on all datasets. Hypothesis (H4) is thus not supported as stated. However, we also did not find that a stochastic prediction model was superior to a deterministic one. In the end, we must find that this portion of our experiment was inconclusive.

The ODC1 dataset is stationary, but only weakly LRD; it appears to be close to the boundary between LRD and short-memory processes. There is no evidence of deterministic behavior, and so no fractal invariants were extracted. Thus, for this dataset, (H2) is weakly supported, while (H1) and (H3) are not supported. ODC4 appears (despite some contrary evidence) to be stationary; but the evidence for LRD (and thus H2) is inconclusive. The presence of chaotic behavior appears to be confirmed, as an attractor with a multifractal geometry was detected, supporting (H1) and (H3). The Android dataset is also stationary LRD, and on balance appears to be deterministic. An attractor with a multifractal geometry was detected, indicating the presence of chaos in this dataset as well; thus, (H1), (H2), and (H3) are supported on this dataset. Finally, the Mozilla dataset is unambiguously deterministic, and stationary LRD. An attractor with a multifractal geometry was also detected. (H1), (H2), and (H3) are all supported on this dataset.

We can say in general that software reliability data exhibits long-range dependency; it is self-similar, and can be modeled equally well with RBFNs and FARIMA models. Chaotic, even multifractal behavior appears to be common. These conclusions are strongest in the new Android and Mozilla datasets, which represent modern, large-scale software systems developed under the open-source model. The detection of multifractal behavior tends to lend support to our contention that individual developers might have quantifiable patterns in the faults they accidentally inject into code. Our finding mean that, at the very least, future SRGMs should account for LRD and self-similarity in their models; and there is good reason to incorporate chaotic dynamics into them as well.

8. Summary and future work

We have sought to confirm or refute the hypothesis of chaotic behavior in software reliability growth data. We have assembled two new software reliability growth datasets, which are an order of magnitude larger than any previous studied in this area. We find that three of them are stationary LRD, with each exhibiting a multifractal attractor in delay embedding space. Predictive modeling showed that the RBFN and FARMIA techniques worried equally well on all datasets. Future SRGM models will need to account for all of these characteristics.

Our own future work will revolve around the question of individual developer mistake patterns. We use the word “patterns” deliberately here; previous work on eliminating systematic errors [8] showed that those efforts had a clear impact on quality improvement. It may be that by collecting development anti-patterns, we can raise awareness of systematic defects, and how to avoid or remove them. We will attempt to identify such anti-patterns, and communicate them within an organization, to aid developers in avoiding time-consuming mistakes. There is also the question of whether or not our results would be replicated on large-scale proprietary software; the development processes for such systems tend to be quite different from open-source software, and the evidence from our two proprietary-software datasets (which are admittedly smaller, older, and noisier) is nowhere near as conclusive on the presence or absence of chaotic behavior.

Acknowledgement

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, under grant no. RGPIN 262-151.

References

[1] J. Beran, Statistics for Long-Memory Processes, CRC Press, Boca Raton, FL, USA, 1994.
[2] T. Bollerslev, Generalized autoregressive conditional heteroskedasticity, J. Econom. 31 (1986) 307–327.
[3] K.-Y. Cai, et al., On the neural network approach in software reliability modeling, J. Syst. Softw. 58 (2001) 47–62.
[4] K.-Y. Cai, et al., A critical review on software reliability modeling, Reliab. Eng. Syst. Saf. 32 (1991) 357–371.
[5] F. Camastra, M. Filippone, A comparative evaluation of nonlinear dynamics methods for time series prediction, Neural Comput. Appl. 18 (2009) 1021–1029.
[6] Y. Cao, Q.-X. Zhu, The software reliability forecasting method using fractals, Inf. Technol. J. 9 (2010) 331–336.
[7] Y. Cao, Q. Zhu, The Software Failure Prediction Based on Fractal presented at the Advanced Software Engineering and Its Applications, Hainan Island, China (2008).
[8] D.N. Card, Learning from our mistakes with defect causal analysis, IEEE Softw. 15 (1998) 56–63.
[9] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, 2nd ed., Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1988.
[10] E.O. Costa, et al., A genetic programming approach for software reliability modeling, IEEE Trans. Reliab. 59 (2010) 222–230.
[11] R.D. Craig, S.P. Jaskiel, Systematic Software Testing, Artech House, Norwood, MA, USA, 2002.
[12] S. Dick, et al., Software reliability modeling: the case for deterministic behavior, IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. 37 (2007) 106–119.
[13] S. Dick, et al., An empirical investigation of web session workloads: can self-similarity be explained by deterministic chaos? Inf. Process. Manage. 50 (2014) 41–53.
[14] J.R. Dunham, Experiments in software reliability: life-critical applications, IEEE Trans. Softw. Eng. 12 (1986) 110–123.
[15] J.R. Dunham, G.B. Finelli, Real-time software failure characterization, presented at the COMPASS, Gaithersburg, MD, USA, in: 90, 5th Annual Conference on Computer Assurance, 8217, 1990.
[16] K. Falconer, Fractal Geometry: Mathematical Foundations and Applications, John Wiley & Sons, New York, NY, USA, 1990.
[17] W. Farr, in: M.R. Lyu (Ed.), SoftwareReliabilityModelingSurvey in Handbook of Software Reliability Engineering, McGraw-Hill, New York, NY, USA, 1996, pp. 71–115.
[18] J. Feder, Fractals, Plenum Press, New York, NY, USA, 1998.
[19] G.B. Finelli, NASA software failure characterization experiments, Reliab. Eng. Syst. Saf. 32 (1991) 155–169.
[20] M.A. Friedman, J.M. Voas, Software Assessment: Reliability, Safety Testability, John Wiley & Sons, New York, NY, USA, 1995.
[21] C.W.J. Granger, R. Joyeux, An introduction to long-memory time series models and fractional differencing, J. Time Ser. Anal. 1 (1980) 15–30.
[22] R. Hegger, et al., Practical implementation of nonlinear time series methods: the TISEAN package, CHAOS 9 (1999) 413–435.
[23] S.L. Ho, et al., A study of the connectionist models for software reliability prediction, Comput. Math. Appl. 46 (2003) 1037–1045.
[24] J.R.M. Hosking, Fractional differencing, Biometrika 68 (1981) 165–176.
[25] C.-J. Hsu, et al., Enhancing software reliability modeling and prediction through the introduction of time-variable fault reduction factor, Appl. Math. Modell. 35 (2011) 506–521.
[26] Q.P. Hu, et al., Robust recurrent neural network modeling for software fault detection and correction prediction, Reliab. Eng. Syst. Saf. 92 (2007) 332–340.
[27] C.-Y. Huang, S.-Y. Kuo, Analysis of incorporating logistic testing-effort function into software reliability modeling, IEEE Trans. Reliab. 51 (2002) 261–270.
[28] C.-Y. Huang, C.-T. Lin, Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship, IEEE Trans. Comput. 59 (2010) 283–288.
[29] C.-Y. Huang, et al., A unified scheme of some nonhomogenous Poisson process models for software reliability, IEEE Trans. Softw. Eng. 29 (2003) 261–269.
[30] U. Hubner, et al., Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared NH3 laser, Phys. Rev. A 40 (1989) 6354–6365.
[31] R.J. Hyndman, A.B. Koehler, Another look at measures of forecast accuracy, Int. J. Forecast. 22 (2006) 679–688.
[32] Z. Jelinski, P.B. Moranda, Software reliability research, presented at the Statistical Computer Performance Evaluation, Providence, RI USA (1971).
[33] C. Jones, Software Assessments, Benchmarks, and Best Practices, Addison-Wesley, New York, NY, USA, 2000.
[34] H. Kantz, T. Schreiber, Nonlinear Time Series Analysis, Cambridge University Press, New York, NY, USA, 1997.
[35] P.K. Kapur, et al., Two dimensional multi-release software reliability modeling and optimal release planning, IEEE Trans. Reliab. 61 (2012) 758–768.
[36] P.K. Kapur, et al., A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation, IEEE Trans. Reliab. 60 (2011) 331–340.
[37] N. Karunanithi, et al., Prediction of software reliability using connectionist models, IEEE Trans. Softw. Eng. 18 (1992) 563–574.
[38] T.M. Khoshgoftaar, R.M. Szabo, Investigating ARIMA models of software system quality, Softw. Q. J. 4 (1995) 33–48.
[39] N.R. Kiran, V. Ravi, Software reliability prediction by soft computing techniques, J. Syst. Softw. 81 (2008) 576–583.
[40] M. Levinson, (2001 October 15) Let's stop wasting $78 billion per year. CIO Magazine (2016).
[41] E.E. Lewis, Introduction to Reliability Engineering, 2nd ed., John Wiley and Sons, New York, NY, USA, 1996.
[42] H. Li, et al., Software Reliability Modeling with Logistic Test Coverage Function Presented at the Int. Symp., Software Reliability Engineering, Seattle, WA, USA, 2008.
[43] B. Littlewood, Theories of software reliability: how good are they and how can they be improved? IEEE Trans. Softw. Eng. 6 (1980) 489–500.
[44] B. Littlewood, J.L. Verrall, A Bayesian reliability model with a stochastically monotone failure rate, IEEE Trans. Reliab. 23 (1974) 108–114.
[45] J.H. Lu, et al., Chaotic Time Series Analysis and Application, University Press, Wuhan: Wuhan, 2002.
[46] M.R. Lyu, Handbook of Software Reliability Engineering, McGraw-Hill, New York, NY, 1996.
[47] B.B. Mandelbrot, The Fractal Geometry of Nature, W.H Freeman and Co, New York, NY, USA,
1983.
[48] T.A. Mazzuchi, R. Soyer, A Bayes-empirical Bayes model for software reliability, IEEE Trans. Reliab. 37 (1988) 248–254.
[49] T.C. Mills, Time Series Techniques for Economists, University Press, Boston, MA, USA: Cambridge, 1990.
[50] R. Mohanthy, *et al.* Predicting Software Reliability Using Ant Colony Optimization Technique, presented at the Int. C. Communication Systems and Network Technologies Bhopal, India (2014).
[51] R. Mohanty, et al., Hybrid intelligent systems for predicting software reliability, Appl. Soft Comput. 13 (2013) 189–200.
[52] J.D. Musa, Validity of execution-time theory of software reliability, IEEE Trans. Reliab. 28 (1979) 181–191.
[53] J.D. Musa, K. Okumoto, A logarithmic Poisson execution time model for software reliability measurement, presented at the Int C. Software Engineering, Orlando, FL, USA (1984).
[54] P.-F. Pai, W.-C. Hong, Software reliability forecasting by support vector machines with simulated annealing algorithms, J. Syst. Softw. 79 (2006) 747–755.
[55] J. Park, J. Baik, Improving software reliability prediction through multi-criteria based dynamic model selection and combination, J. Syst. Softw. 101 (2015) 236–244.
[56] L. Pelayo, et al., Predicting object-oriented software quality: a case study, Int. J. Intell. Control Syst. 14 (2009) 180–192.
[57] H. Pham, Software Reliability, Springer-Verlag, New York, NY, USA, 2000.
[58] H. Pham, et al., A general imperfect-software-debugging model with S-shaped fault-detection rate, IEEE Trans. Reliab. 48 (1999) 169–175.
[59] L. Pham, H. Pham, A Bayesian predictive software reliability model with pseudo-failures, IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. 31 (2001) 233–238.
[60] L. Pham, H. Pham, Software reliability models with time-dependent hazard function based on Bayesian approach, IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. 30 (2000) 25–35.
[61] L.L. Pullum, Software Fault Tolerance Techniques and Implementation, Artech House, Norwood, MA, USA, 2001.
[62] Y.-D. Qi, *et al.* A BP neural network based hybrid model for software reliability prediction presented at the Int. C. Computer Application and System Modeling, Taiyuan, China (2010).
[63] J. Qu, et al., Software reliability analysis in air traffic control system presented at the Integrated Communication, in: Navigation, and Surveillance Conference, Herdon, VA, USA, 2015.
[64] R.H. Riedi, Introduction to Multifractals, Rice University, Houston TX, USA, 1999.
[65] O. Rose, Estimation of the Hurst Parameter of Long Range Dependent Time Series, Institute of Computer Science, University of Wurzburg, Wurzburg Germany, 1996.
[66] P. Roy, et al., Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction, Appl. Soft Comput. 22 (2014) 629–637.

[67] K.J. Ryan, et al., Ethical Principles and Guidelines for the Protection of Human Subjects of Research, United States Department of Health, Education, and Welfare, Washington, D.C USA, 1979.
[68] A. Sadia, Mining Software Quality Data From a Large-Scale Open-Source Software System, M.Sc., Electrical & Computer Engineering, University of Alberta, Edmonton, AB Canada, 2005.
[69] G. Samorodnitsky, Long range dependence, Found. Trends Stochastic Syst. 1 (2006) 163–257.
[70] N.F. Schneidewind, Analysis of error processes in computer software, SIGPLAN Notices 10 (1975) 337–346.
[71] C. Shao, et al., Recovering chaotic properties from small data, IEEE Trans. Cybern. 44 (2014) 2545–2556.
[72] C. Shao, et al., Series-non uniform rational B-Spline (S-NURBS) model: a geometrical interpolation framework for chaotic data, Chaos 23 (2013) 033132-1–033132-11.
[73] G.M. Shepherd, C. Koch, Introduction to synaptic circuits, in: G.M. Shepherd (Ed.), The Synaptic Organization of the Brain, Oxford University Press, New York, NY, USA, 1990, pp. 3–31.
[74] Y. Shu, *et al*. Traffic prediction using FARIMA models, presented at the IEEE Int C. Communications Vancouver, BC, Canada (1999).
[75] S. Siegel, Non-parametric Statistics for the Behavioral Sciences, McGraw-Hill, New York, NY, USA, 1956.
[76] N.D. Singpurwalla, Determining an optimal time interval for testing and debugging software, IEEE Trans. Softw. Eng. 17 (1991) 313–319.
[77] N.D. Singpurwalla, R. Soyer, Assessing (software) reliability growth using a random coefficient autoregressive process and its ramifications, IEEE Trans. Softw. Eng. 11 (1985) 1456–1464.
[78] N.D. Singpurwalla, S.P. Wilson, Software reliability modeling, Int. Stat. Rev. 62 (1994) 289–317.
[79] R. Sitte, Comparison of software-reliability-growth predictions: neural networks vs parametric recalibration, IEEE Trans. Reliab. 48 (1999) 285–291.
[80] Staff, The Zettabyte Era: Trends and Analysis, Cisco, San Jose, CA USA, 2015.
[81] Y.-S. Su, C.-Y. Huang, Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models, J. Syst. Softw. 80 (2007) 606–615.
[82] F. Takens, Detecting strange attractors in turbulence, Lect. Notes Math. 898 (1981) 366–381.
[83] L. Tian, A. Noore, On-line prediction of software reliability using an evolutionary connectionist model, J. Syst. Softw. 77 (2005) 173–180.
[84] N. Ullah, *et al*. A Comparative Analysis of Software Reliability Growth Models using Defects Data of Closed and Open Source Software presented at the IEEE Soft. Eng. Wkshp, Heraclion, Crete, Greece (2012).
[85] J.L. Vehel, P. Legrand, Signal and image processing with Fraclab presented at the Int. Multidisc. C. Complexity and Fractals in Nature (2004).
[86] J.B. Walther, Research ethics in Internet-enabled research: human subjects issues and methodological myopia, Ethics Inf. Technol. 4 (2002) 205–216.
[87] A.S. Weigend, N.A. Gershenfeld, Time Series Prediction: Forecasting the Future and Understanding the Past, Addison-Wesley, Reading, MA, USA, 1994.
[88] C.J. Willmott, K. Matsuura, Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance, Clim. Res. 30 (2005) 79.
[89] S. Yamada, et al., S-shaped reliability growth modeling for software error detection, IEEE Trans. Reliab. 32 (1983) 475–478.
[90] B. Yang, X. Li, A study on software reliability prediction based on support vector machines presented at the IEEE Int. C. Industrial Engineering and Engineering Management Singapore (2007).
[91] O. Yazdanbakhsh, S. Dick, Time-Series forecasting via complex fuzzy logic, in: A. Sadeghian, H. Tahayori (Eds.), Frontiers of Higher Order Fuzzy Sets, Springer-Verlag, New York, NY, 2015, pp. 147–165.
[92] X. Zhang, et al., Considering fault removal efficiency in software reliability assessment, IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. 33 (2003) 114–120.
[93] J. Zheng, Predicting software reliability with neural network ensembles, Expert Syst. Appl. 36 (2009) 2116–2122.
[94] Y. Zhou, H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, J. Syst. Softw. 80 (2007) 1349–1361.
[95] F.-Z. Zou, C.-X. Li, A chaotic model for software reliability, Chin. J. Comput. 3 (2001) 281–291.