Information Sciences 423 (2018) 398–411

Contents lists available at ScienceDirect

Information Sciences

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

Comparing the reliability of software systems: A case study on mobile operating systems

Vladimir Ivanov, Alexey Reznik, Giancarlo Succi*

Innopolis University, Universitetskaya Str. 1, Innopolis, 420500, Russian Federation

ARTICLE INFO

Article history:
Received 13 March 2017
Revised 22 August 2017
Accepted 25 August 2017
Available online 21 September 2017

Keywords:
Software reliability
Software quality
Software reliability growth models (SGRM)
GQM

ABSTRACT

Assessment of software reliability is inevitable in modern software production process. Many works aimed at better models for measurement and prediction of reliability of software products. Tens of approaches have been developed and evaluated so far. However, very few works focus on approaches to compare existing systems with respect to reliability. Despite a tremendous importance to practice (and software management area), a complete and sound comparison methodology does not exist. In this paper, we propose a methodology for software reliability comparison. The methodology extensively applies the GQM approach and software reliability growth models. The methodology has been thoroughly evaluated on a case of assessment and comparison of three open source mobile operating systems: Sailfish, Tizen and CyanogenMod.

1. Introduction

Comparing the reliability of software systems is often of paramount importance. Nowadays, problems of software quality assurance and prediction of behavior of software systems have high importance due to the fact that software is included in the most areas of human activity, especially in safety-critical domains (hydrotechnic, transport, chemical industry, warning systems, etc.).

As defined by Lyu [46], software reliability refers to the probability of failure-free software operation for a specified period of time in a specified environment. The reliability of software is typically measured as the number of defects that exist in the source code of the released software product or of failures that happen during its execution [41,72,73].

Research on specific aspects of software reliability modeling has been already performed. Most of works focus on the evaluation and comparison of the software reliability models and pay less attention to the comparison of software products using the developed models. Therefore, there is no well-founded and validated methodology for the effective comparison of reliability of software systems.

The goal of this paper is to propose and validate a methodology for comparing the reliability of software systems. The approach is based on the analysis of the bug reports present in the typical bug tracking systems, nowadays used in most software projects and the validation has been carried out on three industrially used mobile operating systems:

Cyanogen, an Android-like operating system without Google-provided services and with slight modifications to the core of the OS.;

* Corresponding author.
E-mail address: g.succi@innopolis.ru (G. Succi).

http://dx.doi.org/10.1016/j.ins.2017.08.079
0020-0255/

Sailfish, an open source operating system being developed by a Finnish company named Jolla in collaboration with a Russian company named OMP; it is an extension of the old Nokia's Maemo project; Sailfish project consists of two major parts: Mer project (core stack of middleware) and Nemo (Linux distribution for mobile devices)

Tizen, an open source mobile operating system being primarily developed by Samsung. These three projects are in the stage of active development, and have significant size (>50 MLOC). This presents an opportunity to assess the reliability of software based on the engineering processes and their metrics.

The paper is organized as follows. In Section 2 we survey the existing proposals in comparing reliability of software systems. In Section 3 we overview our proposed methodology for the assessment of the reliability of software systems, which is based on the GQM approach (detailed in Section 4) and on Software Reliability Growth Models (SRGM, detailed in Section 2.2); survey the area of software reliability growth model. Section 6 presents description of the datasets that have been used for evaluation. In Sections 7 and 8 we analyze outcomes of the empirical evaluation and provide recommendations relevant for decision making in practice. Section 9 concludes the paper.

2. Background

Comparing the reliability of software systems has been a long standing problem [28] that has permeated the software industry for the last 70 years almost [8,17,24,27]. Various proposals have been devised [65], spanning different levels of the lifecycle, starting from the initial design [6] and covering the entire lifetime [22] and covering different aspects of software systems, from the low levels [2,26,38] networking and intercommunications aspects [32], system integration [24], up to metrics [64] and user interfaces [77], or also considering the whole lifecycle [13,14,33,39,48].

However, there are limited works comparing the reliability of software systems that have been applied or, at least, experimented, with real projects.

2.1. Existing approaches to compare the reliability of software systems

A common goal of a comparative study of reliability is to set up a common approach or a system of criteria to do measurable and reproducible results. Most SRGMs are parametric models with few parameters that may be interpreted and compared in a meaningful way. Therefore, existing methodologies try to find and analyze patterns in software reliability growth and use those patterns in a comparison.

Zhou and Davis [78] reported results of open source software evaluation. They used the Weibull model fitted to bug tracking data. An interesting result is a comparison of open source and closed source projects. Along a development cycle, open source projects demonstrated reliability growth pattern similar to closed source projects. A slightly similar study has been presented by Syed-Mohamad and McBride [68]. Authors use two SRGMs (a concave model and an S-shaped model) to compare two open source products. The main research question: whether open source software has a different defect arrival rate to software developed in-house. Developers of open source software tend to dramatically change source code between subsequent releases in order to meet new expectations such as feature requests and may frequently switch to new technology. Another factor is the large community which easily fulfills new feature requests but rarely do a rigorous quality assurance. This is opposite to results presented in the work of Zhou and Davis [78]. Interesting that both studies interpreted results in a meaningful way. This is a clear evidence for the lack of comprehensive research in the area of methodologies for software reliability comparison. One of the most recent comparison studies is presented by Rossi et al. [63]. Authors aimed at presenting an approach to investigate reliability of open source software projects using several SRGMs. They carefully downloaded and cleaned data from three large open repositories: OpenSuse, Mozilla Firefox and OpenOffice.org and applied eight SRGMs: Goel Okumoto, Goel Okumoto S-shaped, Gompertz, Hossain Dahiya, Logistic, Weibull S-shaped, Weibull more S-shaped and Yamada's model. The comparative study is intended to compare two types of software development (namely open source and closed source), rather than compare software products to each other. Given the difference of the products under the examination, this result is very promising and it is supported in the related study carried out by Li et al. [43].

Overall, the state of the art approaches to comparison of software products reliability are mostly ad-hoc and do not follow a well-founded and validated methodology. Despite the lack of such methodology, there are many case studies in the area of reliability analysis of open-source and closed-source products. The case studies are important for our study, in order to show the whole spectrum of ideas in this filed. Thus, any new comparison methodology should be broad enough to cover major part of the spectrum.

Rahmani et al. [62] compare the goodness-of-fit and prediction capabilities of three reliability models using the failure data of five popular open source software products. Ullah et al. [70] investigate the goodness of fit and prediction capability of eight SRGMs. The comparison involves fifty different failure data sets. Contemporary studies involve additional criteria for evaluation. For instance, in one of the most recent works, Ke et al. [36] use the Parr-curve model with multiple change-points to analyze the consumption of testing-effort.

2.2. Software reliability growth models

The reliability of software systems have been described using multiple approaches, including fuzzy models [29,50,56], granular models [57], program invariants [16] regression analysis [47], Markov Models [42,44], neural networks

[21,23,30,53,55,58], other machine learning techniques [59], testing [11,12], also trying to build a comprehensive framework for it [3,10,49,59], and applied to multiple, also very diverse, contexts [1,79].

Software reliability growth model is a mathematical model that describes software failure-detection or defect discovery phenomena during the system testing and debugging phases. These models are particularly useful in the following aspects as specified by Yamada [76]: (a) the identification of the optimal time to release software, with the associated issue to predict the number of leftover defects at any time, (b) statistical control of the progress of testing, and (c) the optimal allocation of effort for testing. SRGMs may be divided into two groups: models with continuous time between failures (measured as a time of test execution or calendar time) and models with discrete time (measured in a number of test cases' runs).

A vast number of models have been proposed so far. One of the first contributions was made by Musa et al. [52]. Since then, many models for analysis of failure arrival process have been developed. A comprehensive analysis of SRGMs can be found in surveys like in [9,60,63,63], and in comparative studies carried by Succi et al. [67] and Ullah et al. [70]. However, so far there is no "one fits all approach" to model reliability someone could choose with respect to a set of reasonable assessment criteria, such as goodness of fit, prediction of leftover defects, model's robustness, etc. [15,54,67,69]. There are also several sophisticated classification approaches for SRGMs due to their large number. For instance, Asad et al. [5] distinguish between failure rate models and failure intensity models. Yadav and Khan [75] divide models by type, e.g., exponential failure time model, Weibull and Gamma failure models, Infinite failure time models and Bayesian models. One of the contemporary and a comprehensive classifications of SRGMs proposed by Kapur et al. [35]. Further, depending on a model's output, one may distinguish between models for assessment a probability of failure and models that are able to predict a number of leftover defects. There are lots of comprehensive surveys of SRGM studies [9,60,61,70].

An underlying defect counting process is usually assumed to be a non-homogeneous Poisson process (NHPP); The area of NHPP-based SRGMs have attracted the most of research interest. Tens of NHPP-based SRGMs have been proposed to model and predict the rate of arrival of failures and they have been investigated and compared to each other in numerous studies, like the studies of Stringfellow and Andrews [66] and of Succi et al. [67]. The typical statement in SRGM research is that the best models work well under a certain set of predefined assumptions, and there is no "best mode" for every situation, as it is presented in [66], [67], [35], and [71].

Modern approaches to software reliability growth extend the state of the art results in a number of ways. Here are the two most popular of the ways: (i) add a new dimension (e.g., test coverage or cost), or (ii) try to remove some assumptions (e.g., consider imperfect debugging process).

2.3. Summary of our contribution

Altogether, we can summarize that: I. Most of the existing approaches compare SRGMs, not products; II. There is no common methodology for application of SRGMs and comparison software systems using the results; III. At the same time there are a lot of possibilities to apply SRGMs; such a variety (of criteria, data formats) makes comparison of case studies less valid. In this study we fill the gaps providing a new methodology for comparison reliability of software systems, defining a set of criteria for assessment quality of an SRGM and a set of experiments that validate the methodology.

3. Overall structure of the proposed methodology

In this section we present a methodology for comparison of reliability of software products. A short overview of the methodology is presented below.

I. Collect defect-related data from issue trackers of the software systems under comparison. This is an inevitable step for an empirical evaluation of software systems; we describe the step in Section 6.1.

II. Do a GQM-based analysis of system comparison process with respect to the reliability; this step is described in Section 4 and includes selection of Goals, selection of Questions and selection of Metrics.

III. Choose and apply SRGMs to model defect-related data; this step described in Sections 5.4 and 5; it includes selection and characterization of parameters of SRGMs, selection of criteria for assessment of SRGMs and fitting SRGMs to the data and calculation of values of parameters.

IV. Compare parameters of SRGMs for different selected software systems. This step is guided by the results of GQM analysis, while results of SRGM-based analysis serve as empirical evidence for comparison. This step is described in Section 7.

4. Using GQM to compare the reliability of software products

In the literature, the terms defects, failures, and reliability have often been almost interchanged. To clarify the scope of our work, we follow the precise definition of Fenton and Pfleeger [20].

A failure is the occurrence of a behavior of a (software) system that deviates from its specifications or expectations. A defect is the code that leads to failures in operations. The reliability of a system is the possibility that such system will operate according to its specifications for a given time.

The connection between defects, failures, and reliability is controversial, essentially because of Pareto's Principle: "a very small proportion of the defects in a system will lead to almost all the observed failures in a given period of time; conversely most defects in a system will not lead to failures in the same given period of time" [34].

Reliability management models operate on data describing the time between failures, its reciprocal that is the failure rate, or the cumulative count of failures over time.

4.1. Selection of relevant GQM questions

Altogether, to understand the overall reliability of the three target project, we can take a GQM approach by Basili et al. [7], and, given the goal to determine the overall reliability of the target mobile and open source operating systems, we can consider the three following questions:

(Q.1) How likely is the code to have a failure?
(Q.2) How likely is the code to have a fix for an existing failure, and, if it has, how fast such fix occurs?
(Q.3) What is the rate at which failures are detected?

A defect can be analyzed from different points of view. We focus on defects triggering back actions on developers during testing, following the work of Gokhale et al. [25]. Therefore, we use the term "Modification Request" (MR).

MRs arise during the design and development process, during the code inspection, or as a result of the testing process. Our data is about MRs arising during the actual early use of the software. We do not take into account any ripple effect due to defect repair.

Fenton and Neil [18] and Fenton and Ohlsson [19] review the defect prediction models. According to Wood [73], we use reliability management models with MRs-detection rate instead of failures rate and in general MRs instead of failure. A software reliability model provides a description of the growth of MRs over time. This is expressed by a mathematical function giving the cumulative number of MRs arriving within a fixed time interval.

Following the established approach of Musa [51], we assume that all MRs contribute equally to the failure rate. We also assume that fixing modification request does influence the MRs detection process as proposed by Littlewood [45]. These two assumptions have been taken also by Succi et al. [67] without any loss of generality.

As mentioned, our goal is to compare the ability of the three core open source projects to detect defects, taking into account that each model has different effectiveness in forecasting the arrival of MRs. Therefore we select suitable SRGMs and we use them for the purpose of determining the ability of each project to detect defects. SRGMs have already been applied in Open Source projects, see for instance the evaluation by Rossi et al. [63].

Finally, we assume the independence of successive times, each with a specified distribution function.

4.2. Selection of relevant GQM metrics

Given these premises, to answer the questions above, we will consider the following metrics. First, to answer Q.1 we use:

(M.1) The number of MRs that are present.
(M.2) The rate at which MRs are issued (i.e., MRs / Total time elapsed for the project).
(M.3) The density of MRs over the physical size of the project (i.e., MRs / Total size).

Then, to answer Q.2 we use:

(M.4) The percentage of MRs that are fixed.
(M.5) The speed at which MRs are fixed, when they are fixed.
(M.6) A subjective evaluation of the overall fixing process, performed analyzing the curve of fixes.

Eventually, to answer Q.3 we use:

(M.1) The parameters of the SRGMs referring to the effectiveness of detecting bugs (i.e., in issuing MRs) across the target models as defined in Section 2.2.
(M.2) A subjective evaluation of the timings of arrivals of MRs, performed analyzing the curves of arrivals.

Below we summarize the questions and the metrics which we have selected:

- Q.1. How likely is the code to have a failure? :
  - M.1. The number of MRs that are present.
  - M.2 The rate at which MRs are issued.
  - M.3.The density of MRs over the physical size of the project.
- Q.2. What is the rate at which failures are detected?
  - M.4. The percentage of MRs that are fixed
  - M.5. The speed at which MRs are fixed, when they are fixed.
  - M.6. A subjective evaluation of the overall fixing process, performed analyzing the curve of fixing of MRs.
- Q.3. What is the rate at which failures are detected?
  - M.7. The parameters of the SRGMs
  - M.8. A subjective evaluation of the timings of arrivals of MRs, performed analyzing the curves of arrivals.

5. Using SRGMs to measure and predict reliability of software products

As mentioned, our goal is to compare the ability of the three core open source projects to detect defects using SRGMs, taking into account that each model has different effectiveness in forecasting the arrival of MRs.

5.1. General structure of SRGMs

A software reliability growth model is a mathematical model that describes software failure-detection or defect discovery phenomena during the system testing and debugging phases. Such models are intended to assess the reliability growth during the testing-progress and predict software reliability at later stages of product development. SRGMs may be divided into two groups: models with continuous time between failures (measured as a time of test execution or calendar time) and models with discrete time (measured in a number of test cases' runs).

In SRGMs with discrete time, the cumulative number of MRs arriving from 0 to t, n(t), is usually modeled as the output of a random process. An underlying defect counting process is usually assumed to be a non-homogeneous Poisson process (NHPP); in this case an SRGM is represented by a parametric mean value function ($m(t)$) of the NHPP. The mean value function denotes an expected number of defects discovered by time t. Its shape and scale depend on free parameters which are estimated from a dataset in place.

We briefly review models based on NHPP, because of their higher relevance to our study. This class of models may be further divided into two subclasses: S-shaped models and Concave models. One of the earliest contributions was the Goel-Okumoto model, where the mean value function, $m(t) = a(1 - e^{-bt})$, derived from the following equation: $\frac{d}{dt}m(t) = b(a - m(t))$.

Both concave and exponential models share an assumption that a process of defect removal is uniform. Later S-Shaped models have been introduced under an assumption of a non-uniform testing profile. A Delayed S-Shaped model represents testing in two stages: a failure observation stage and a fault removal (or correction) stage. Hence it has two mean value functions: $m_f(t) = a(1 - e^{-bt})$, $m_r(t) = a(1 - (1 + e^{-bt}))$.

These two functions are combined to build a mean value function of a more complex model. The idea of combining of several models into one also present in works that focus on complexity of faults.

The complexity of a defect is represented as a number of necessary stages to remove the defect; and each such stage has some distinct delay. For instance, for simple defects mean value function has one stage (a defect is removed immediately): $m(t) = a(1 - e^{-bt})$. For hard defects, detection and removal is a two-staged process, which may be represented with the following mean value function: $m_2(t) = a_2(1 - (1 + b_2t)e^{-b_2t})$. For severe faults, the process will be a three-staged one, with even more complex mean value function: $m_3(t) = a_3(1 - (1 + b_3t + b_3^2t^2/2)e^{-b_3t})$.

The Generalized Erlangian software reliability growth model proposed by Khoshgoftaar and Woodcock [37] with n types of defects will have the following mean value function (a is a total number of defects):

$$m(t) = \sum_{i=1}^{n} a_i \left(1 - e^{-b_i t} \sum_{j=0}^{i-1} \left((b_j t)^j / j!\right)\right), \sum_{i=1}^{n} a_i = a.$$

5.2. About the parameters of the SRGMs

Concave models simply represent the fact that initially defects are discovered and reported faster, and, as they get found, there are less to find, so the rate of detection and reporting slows down. S-shaped models are first convex and then concave, with a period during which the MRs-detection rate increases, reflecting the initial learning phase. At the beginning the user is not familiar with the software, hence the usage of the system is limited and the detection is slow, but after a certain time the team gains experience and knowledge about the behaviors of the product. This implies higher rates of MRs detection until a large number of MRs have been detected. Then, it becomes harder and harder to detect new MRs. So the MRs detection rates initially increase then decrease. Of course the second derivative of $\mu(t)$ is responsible of its shape. Some of the function we consider may be concave or S-shaped according to the values of their parameters.

The performances of SRGMs have been measured in a number of comparative studies[31]. evaluates parameters affecting the software reliability[70]. selected eight models: Musa Okumoto, Inflection S-Shaped, Goel Okumoto, Delayed S-Shaped, Logistic, Gompertz, Yamada Exponential and Generalized Goel model due to their prevalence among other software reliability growth models.

Table 1summarizes some of the most preeminent SRGMs and their performances. The list of selected models has been elaborated based on Table 1 from [67] and Table A from [73]. Indeed, every model carries a value that depends on the goodness of fit, which is employed in numerous SRGM studies. However, in case of software reliability analysis other criteria are also important. In the next subsection, we represent the relevant criteria for evaluation of SRGMS.

As evident, typically, the parameter a represents the total number of MRs, while the parameters b and c refer to the speed of receiving MRs. Please, keep in mind that, while the values of the parameters a are comparable across models, the same is not true for b and (when present) c, which, even when they have the same interpretation, still are incomparable across models and should be compared only within a given model, indeed, if the model proves to be a good description of

Table 1
Summary of the main proposals for SRGMs useful in assessing the goodness of the defect detection process based on Table 1 from [67], and Table A from [73]. 𝔈 stands for "Concave," 𝔖 for "S-Shaped," ectnd for "expected cumulative total number of defects," dd for "defect detection.".

Model, 𝔈/𝔖	Formula	Interpretation
Goel-Okumoto (G-O), 𝔈	$a(1-e^{-bt})a>0,\ b>0$	a = ectnd; b = dd rate
G-O S-Shaped (GO-S), 𝔖	$a(1-(1+bt)e^{-bt})a>0,\ b>0$	a = ectnd; b = dd rate
Logistics (L), 𝔖	$\frac{a}{(1+be^{-ct})}\ a>0,\ b>0,\ c>0$	a = ectnd; c = dd rate; b = dd inefficiency
Hossain-Dahiya (HD), 𝔖	$a\frac{(1-e^{-bt})}{(1+ce^{-bt})}\ a\geq 0,\ b>0,\ c>0$	a = ectnd; b = dd rate; c = dd inefficiency
Weibull (W), 𝔖	$a(1-e^{-b\times t^c}),\ a>0,\ b>0,\ c>0$	a = ectnd; b = dd rate; c = dd rate booster
W more S-Shaped (W-S), 𝔖	$a(1-(1+b\times t^c)e^{-b\times t^c});\ a>0,\ b>0,\ c>0$	a = ectnd; b = dd rate; c = dd rate booster
Yamada Exp. (YE), 𝔈	$a(1-e^{-b(1-e^{-c\times t})})a>0,\ b>0,\ c>0$	a = ectnd; b = dd rate; c = dd rate booster
Yamada Raleigh (YR), 𝔖	$a(1-e^{-b(1-e^{-c\frac{t^2}{2}})})a>0,\ b>0,\ c>0$	a = ectnd; b = dd rate; c = dd stronger booster

Table 2
Influences of the parameters of the models in Table 1 on the speed of receiving MRs.

Model	b	c
G-O	only influential	absent
GO-S	only influential	absent
L	negative impact	positive impact
HD	positive impact	negative impact
W	positive, secondarily influential	positive, most influential
W-S	positive, secondarily influential	positive, most influential
YE	positive, secondarily influential	positive, most influential
YR	positive, secondarily influential	positive, most influential

the underlying data. Moreover, the influence of b and c on the detection rate, when present, can have different impact, depending on the role of each parameter in the equation (sometimes even with a negative impact). Therefore, to facilitate the interpretation of each parameter, Table 2 summarizes, model per model, the most influential parameter and the parameter with a negative impact, if present.

5.3. Multi-stage SRGMs

A special case is multi-stage SRGMs that represent the development process as a sequence of releases, after each of which the process of defect detection and removal starts again. A number of multi-stage SRGMs have been proposed in the past decade. In this section we review approaches that avoid the single-release assumption by using a concept of "multi-stage SRGM" which was originally introduced by Musa et al. [52].

Wood [74] mentioned a multi-stage SRGM, but with an assumption that the moments of releases are known a-priori which, in turn, leads to the application of a single-stage model several times. The original model proposed by Wood for multi-release modeling was based on a simplification that can hardly reflect the reality, i.e. each release has a completely independent defect identification process. In other words, in a given release all the detected defects come from the release and not from previously developed and tested code. The reality is more complex, though.

Thus, models with additional parameters that are able to describe a number of releases and release times attract more interest. Multi-stage software reliability growth models describe the evolution of a software product across releases. Values of key parameters coming from a model allow to compare several similar products and even types of products as well as software development processes.

We denote a number of releases with n. Hence the process of defect detection has n stages each of which may be represented as a separate SRGM. We call each of these SRGMs "a basic level model", because they act as building blocks in a combined models, a multi-stage SRGM. We denote a mean value function of a basic level SRGM as $f(a, b, c, t)$[1].

We represent an approach to build a complex SRGM as a combination of n basic level SRGMs, as proposed by Wood [74]. This approach allows to build predictive models which work under assumption of multiple release software development. In other words, given fixed n and a fixed basic level SRGM (e.g., from the Table 1) we use the following general expression

[1] Some of SRGMs have two parameters: a and b. In this case we use corresponding functional form without the parameter c, $f(a, b, t)$.

to construct the mean value function for the multi-stage SRGM:

$$\Phi_n(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, t) = \sum_{i=1}^{n} f(a_i, b_i, c_i, t - d_{i-1}), \tag{1}$$

where **a**, **b**, **c** are n-dimensional vectors, that represent parameters of the combined model, and **d** is also an n-dimensional vector holding shifts in time that correspond to the ends of releases. For example, d_2 denotes the moment when the testing of the second release was ended. By definition $d_0 \equiv 0$. In practice, a_i, b_i, c_i are the parameters of a basic level SRGM; they denote the i-th release.

Following the existing research of Almering et al. [4], Koziolek et al. [40] and Wood [74], we assume the two additional constraints to hold:

1) the function $f(a_i, b_i, c_i, t - d_{i-1})$ equals to zero for all values of $t \le d_{i-1}$, in other terms, there is no back effect in time of a future release; however, note that indeed f still affects the total number of defects detected in the subsequent releases, when $t > d_{i-1}$;
2) We assume that parameters related to the environment should not change dramatically from release to release[2]. Thus, we set an additional constraints: $\forall i = 1..n, \exists b, c : b_i = b; c_i = c$.

The second assumption significantly decreases model's degrees of freedom, so we can rewrite the basic level SRGM as:

$$\hat{f}(a_i, b, c, d_{i-1}, t) = \begin{cases} f(a_i, b, c, t - d_{i-1}), & \text{if } t \ge d_{i-1}, \\ 0, & \text{otherwise;} \end{cases} \tag{2}$$

and finally simplify (1) into the following expression of mean value function at the n-staged SRGM:

$$F_n(\mathbf{a}, b, c, \mathbf{d}, t) = \sum_{i=1}^{n} \hat{f}(a_i, b, c, d_{i-1}, t), \tag{3}$$

$$a = \sum_{i=1}^{n} a_i. \tag{4}$$

We assume the total number of defects (a) will follow the Equation (4) and the time spans defined by the vector d will cover the whole testing period.

5.4. Criteria to evaluate SRGMs

In the present study we consider the performances of the SRGMs with respect to a set of relevant criteria. Out of the criteria proposed by Succi et al. [67], we have selected the following: goodness of fit (GoF), relative precision of fit (RPoF) and coverage of fit (CoF).

The **goodness of fit (GoF)** shows how well the model fits the original data. The unit used for GoF is the number of bugs. GoF is calculated as a sum of squared residuals divided by the number of degrees of freedom.

The **relative precision of fit (RPoF)** is the area of the bootstrap 95% confidence interval of the stripe of the model, normalized over the analyzed time. The unit used for this measure is based on the product of the time (expressed in days) with the number of defects. Therefore, models with low value of the RPoF typically have high capability of providing useful information about the occurrence of defects.

The **coverage of fit (CoF)** is degree to which the bootstrap 95% confidence interval of the stripe of the model captures the observed defects. The unit used for the coverage of fit is the fraction of the total number of defects within the bootstrap 95% confidence interval. The RPoF and the CoF represent two complementary aspects of the model, which need to be considered together. Indeed, a very large stripe could cover practically 100% of the data points, but would have high value of relative precision as well. On converse a very low value of RPoF could cover only a small fraction of the data.

It is important to emphasize that these criteria cover different aspects of the performances of the models that could be also individually conflicting. As an example, a very positive value of coverage of fit could be obtained through a very large confidence interval. However, such situation would penalize the relative precision of fit. Therefore, we think that altogether these criteria represent a solid description of the overall quality of a SRGM.

6. Empirical data for the evaluation

6.1. Data from issue tracking systems

As mentioned, we validate the methodology on three open source mobile operating systems: CyanogenMod, Sailfish and Tizen.

We collect data from original Open Source repositories, namely:

[2] Basic level SRGMs have two or three parameters, one of which is typically the total number of defects in the released code (the parameter named a in Table 1), other parameters usually related to the ability to identify defects or to other environmental factors (e.g. learning rate).

Table 3
Summary of the collected data for the MRs of the target projects.

Project	Starting date	Total number of MRs
Sailfish	November 2013	3425
Tizen	January 2012	10,092
CyanogenMod	July 2009	19,771

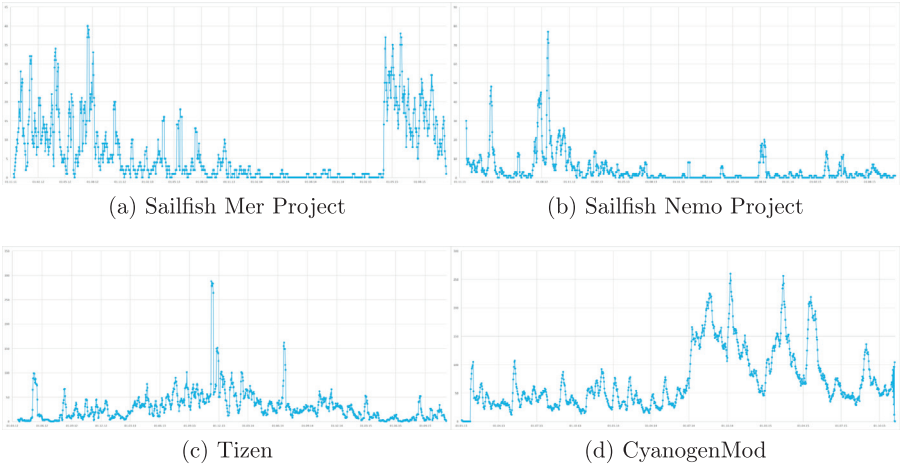


Fig. 1. Arrival of MRs as a function of time.

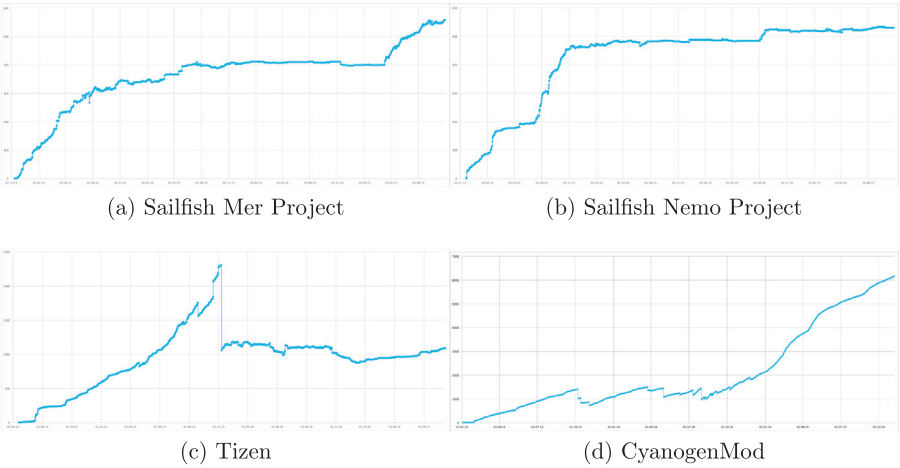


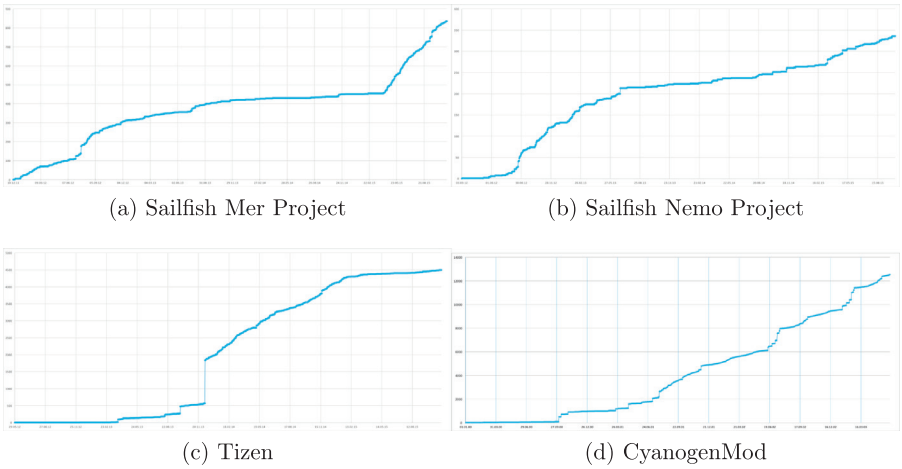
Fig. 2. Number of not resolved MRs as a function of time.

1) For Sailfish: https://bugs.nemomobile.org and https://bugs.merproject.org, based both on Bugzilla
2) For Tizen: https://bugs.tizen.org, based on Jira
3) For CyanogenMod: https://jira.cyanogenmod.org, based on Jira

These data have been collected directly by the authors at the end of November 2015 and are reported in Table 3. In the following figures we represent temporal distribution of the arrival of MRs (Fig. 1), temporal distribution of the not resolved MRs (Fig. 2), temporal distribution of the closed MRs (Fig. 3), and cumulative number of arrived MRs (Fig. 4).

6.2. Project sizes

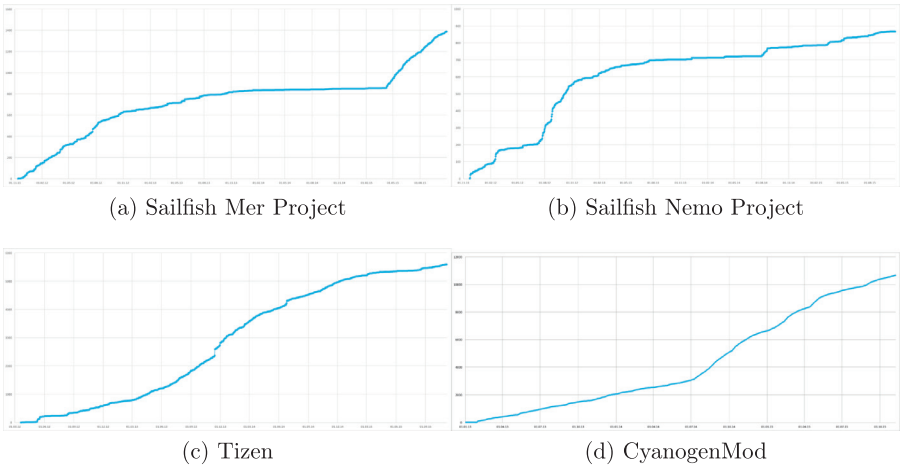
To compute the metrics about the size of the project, we have analysed the code repository. The projects are written in a multitude of programming languages (including C, C++, and Java), therefore it appeared that for the purpose of this chapter the most reliable approach is to consider to complementary measures - one being the considering size as a count of the



(a) Sailfish Mer Project (b) Sailfish Nemo Project

(c) Tizen (d) CyanogenMod

Fig. 3. Number of closed MRs as a function of time.



(a) Sailfish Mer Project (b) Sailfish Nemo Project

(c) Tizen (d) CyanogenMod

Fig. 4. Cumulative mumber of MRs arrived as a function of time.

Table 4
Size in number of files and LOCs for the projects.

Project	Total number of files	Lines of Code as per `cloc` command
Sailfish	864,998	97,041,969
Tizen	987,273	122,166,906
Cyanogen	457,649	50,523,609

number of files (NOF) present in the projects, and the other size as sum of the lines of code (LOC) of each file, where LOC is the output of the Linux command `cloc`[3]. They are listed in Table 4.

7. Results of evaluation

7.1. Analysis of the values of the metrics related to the likelihood that the projects contain failures (Q.1)

As mentioned, metrics related to the likelihood that the projects contain failures (Q.1) are:

M.1. The number of MRs that are present.
M.2. The rate at which MRs are issued (i.e., MRs / total time elapsed for the project).
M.3. The density of MRs over the physical size of the project (i.e., MRs / total size considered both in terms of number of files (NOF) and of lines of code (LOC)).

[3] The description is present in a multitude of online resources like http://linux.die.net/man/1/cloc.

Table 5
Metrics of failures for the target projects (Q.1).

Project	M.1.	M.2.	M.3. (NOF)	M.3. (LOC)
Sailfish	*3,43*	*4.51*	3.96×10^{-3}	35.29×10^{-6}
Tizen	10,09	**9.22**	10.22×10^{-3}	82.61×10^{-6}
CyanogenMod	**19,77**	8.44	43.20×10^{-3}	391.32×10^{-6}

Table 6
Metrics of fixes for the target projects (Q.2).

Project	M.4.	M.5.
Sailfish	**34.36%**	**1.55 MRs/day**
Tizen	44.61%	*4.12 MRs/day*
CyanogenMod	*46.01%*	3.88 MRs/day

The values of the metrics related to the likelihood that the projects contain failures (Q.1) are summarized in Table 5 below.

It is evident that in all these three cases the project that scores best (in *italic*) is Sailfish, while in general the worst (in **bold**) is CyanogenMod. However, it is worth mentioning that this could be due to the fact that these metrics depend on how much a project has been actually used by the end users, since projects with a lower usage would have a lower number of MRs.

7.2. Analysis of the values of the metrics related the likelihood and the speed of fixing issues (Q.2)

As mentioned, metrics related to the likelihood and the speed of fixing issues (Q.2) are:

M.4. The percentage of MRs that are fixed.
M.5. The speed at which MRs are fixed, when they are fixed.
M.6. A subjective evaluation of the overall fixing process, performed analysing the curve of fixes.

The values of the first two metrics related to the likelihood and the speed of fixing (M.5 and M.6) are summarized in Table 6. Also here we have highlighted the best results with italic. We notice that for this question Tizen and CyanogenMod perform remarkably better. In M.5 CyanogenMod scores the best and it is about 3% better than Tizen and in M.6 Tizen scores the best and it is about 6% better than CyanogenMod.

To analyze M.6. we can look jointly at the curves of arrivals of MRs in Figures 1 a (Sailfish Mer), 1 b (Sailfish Nemo), 1 c (Tizen), and 1 d (CyanogenMod), and of MRs still open in Figs. 2a (Sailfish Mer), 2 b (Sailfish Nemo), 2 c (Tizen), and 2 d (CyanogenMod). Overall, we notice that the detecting and fixing projects lags somehow behind in Sailfish while appears occurring consistently in Tizen and CyanogenMod. For Tizen there has been a time in which it looks like there has been a massive commit of previous fixes (around December 2013), but for the rest the process is quite regular, with fixes submitted all over the process. For CyanogenMod there are multiple but smaller times of commit of fixes, and, under this perspective the process appears less regular than CyanogenMod. These findings are confirmed looking at the curves of the cumulative closed MRs in Fig. 3a for Sailfish Mer, in Fig. 3b for Sailfish Nemo, in Fig. 3c for Tizen, in Fig. 3d for CyanogenMod. Altogether, *our subjective evaluation for M.6 produces a slightly better assessment for Tizen than for CyanogenMod* while **Sailfish scores the worst**.

7.3. Analysis of the values of the metrics related to the rate at which failures are detected (Q.3)

As mentioned, metrics related to the rate at which failures are detected (Q.3) are:

M.7: The parameters of the SRGMs referring to the effectiveness of detecting bugs (i.e., in issuing MRs) across the target models as defined in Section 3.
M.8: A subjective evaluation of the timings of arrivals of MRs, performed analyzing the curves of arrivals.

Focusing our attention to the SRGMs, Table 7 contains performances on the data of the three operating systems of the models discussed in Table 1.

From this analysis (Table 7), it appears that in the cases under consideration, the models GO-S, L, HD, and W-S perform the best. We can now consider the results of the fitted models in terms of the desired parameters (Table 8); in this table it is evident that in terms of the ability to detect bugs *Sailfish performs the best*. This is consistently present in all models and in the models that fit best the curve (we mark these rows with a star(*) in Table 8), namely GO-S, L, HD, and W-S. In terms of the worst model, it is difficult to discern between Tizen and CyanogenMod.

On one side both perform best in one case, and Tizen performs worst more often. On the other side, Tizen perform best in one of the most accurate models (W-S) and when it performs worst, it is in many cases very close to CyanogenMod.

Table 7
Values of the parameters of models in Table 1 on the three considered operating systems.

Model	Sailfish			Tizen			CyanogenMod		
	GoF	RPoF	CoF	GoF	RPoF	CoF	GoF	RPoF	CoF
G-O	4.07	1047	*1*	46.12	154	0.99	48.94	1296	0.97
GO-S	6.08	*939*	*1*	36.81	*131*	0.99	40.66	1583	0.98
L	4.72	945	0.81	35.19	1252	0.56	*37.88*	*776*	0.41
HD	5.31	1363	*1*	*34.99*	348	*1*	38.16	2008	0.96
W	3.86	2393	*1*	46.27	7211	0.99	49.57	5475	0.88
W-S	*3.53*	8450	*1*	44.81	3567	*1.00*	60.08	5571	*1*
YE	3.63	10,263	*1*	36.11	45,079	*1*	50.34	57,597	*1*
YR	5.00	16,200	*1*	36.70	33,926	*1*	42.92	131,810	*1*

Table 8
Performances of the models in Table 2 for the three considered operating systems.

Model	Parameter	Sailfish	Tizen	CyanogenMod
G-O	b (positive)	*0.05*	**0.003**	0.0068
GO-S*	b (positive)	0.025	**0.013**	0.016
L*	c (positive)	*0.087*	**0.026**	0.045
	b (negative)	*3.99*	14.09	**27.56**
HD*	b (positive)	*0.034*	0.021	**0.018**
	c (negative)	*0.77*	**9.49**	3.66
W	c (positive, most influential)	0.71	0.76	**0.67**
	b (positive, secondarily influential)	*0.06*	**0.007**	0.0084
W-S*	c (positive, most influential)	0.38	*0.89*	0.61
	b (positive, secondarily influential)	*0.625*	**0.016**	0.08
YE	c (positive, most influential)	*0.0519*	**0.004**	0.0044
	b (positive, secondarily influential)	0.59	0.47	*1.1*
YR	c (positive, most influential)	*0.0015*	**0.0001**	**0.0001**
	b (positive, secondarily influential)	*1.16*	0.91	**0.87**

Altogether, we conclude that **CyanogenMod is the worst performing OSs** in terms of rate at which failures are detected. To evaluate the timings of arrivals of MRs, we analyze curves of the cumulative MRs arrived at an instant of time. These curves are for Sailfish Mer in Fig. 4a, for Sailfish Nemo in Fig. 4b, for Tizen in Fig. 4c, for CyanogenMod in Fig. 4d. In this case we notice *a very consistent time of arrivals of MRs for Tizen (the best performing Operating System)*, while these lines are less constant for CyanogenMod, while **the lines are very irregular for Sailfish (the worst performing Operating System)**.

8. Discussion of the results

The primary goal of the study, is development and validation of new methodology for comparing reliability. Thus we split the discussion into two parts: development of the methodology and its validation in our case.

8.1. On the methodology to compare of reliability of software systems

The methodology has both advantages and flaws. Main contribution to the current state of the art in software reliability comparison is twofold: (i) extensive application of GQM-approach and (ii) application of SRGMs. The first component (GQM) provides a powerful tool to apply the methodology and to guide the following steps by proper selection of relevant GQM Goals, Questions and Metrics. The second component (SRGM) gives objectivity and flexibility, due to a wide variety of existing SRGMs and evaluation criteria.

Limitations and disadvantages of the methodology are the following. Sometimes, data we use in SRGM-analysis are not available for closed source products. This limits the area of application of the methodology. Even if data is available it should be properly preprocessed. This preprocessing step is hard to avoid or unify. For instance dataset may contain information about defects discovered in a series of releases, while a selected SRGM was intended to model single-release data. Dealing with such issues its is a major drawback of the comparison methodology.

8.2. On the comparison of the reliability of open source mobile OS

A byproduct of our study is a comparison of the three analyzed Open Source mobile operating systems. The goal of this research is to determine which OS is stronger in terms of reliability. To this end we have performed a GQM analysis and we have identified 3 questions and 8 metrics. Below are the results of the analysis for each question and each project:

M.1: best performing OS: Sailfish, worst performing OS: Cyanogen.
M.2: best performing OS: Sailfish, worst performing OS: Tizen.

M.3: best performing OS: Sailfish, worst performing OS: Cyanogen.
M.4: best performing OS: Cyanogen, worst performing OS: Sailfish.
M.5: best OS: Tizen, worst performing OS: Sailfish.
M.6: best performing OS: Tizen, worst performing OS: Sailfish.
M.7: best performing OS: Sailfish, worst performing OS: Cyanogen.
M.8: best performing OS: Tizen, worst performing OS: Sailfish.

Considering the rankings of the metrics, overall, it appears that Sailfish is in most case the best performing OS (4 times out of 8). However, it is also the OS that performs the worst in most cases (again 4 cases out of 8). On the contrary, Tizen scores the best in 3 cases out of 8, but the worst only in one case out of 8.

9. Conclusion and further work

In this study we present a methodology for comparison of software systems with respect to reliability dimension. We have validated the methodology comparing three software products. The proposed methodology has the following features that fulfill the initial goal of the study: it provides certain level of flexibility and abstraction while keeping objectivity, i.e. providing measurable comparison criteria.

Finally, given the comparison methodology with a set of SRGMs and evaluation criteria it becomes much easier to disseminate information about reliability of wide range of software systems.

During the application of the methodology to mobile OS we have found:

A) Tizen OS is the most effective mobile operating system out of the three considered in terms of reliability of the core modules and components.
B) CyanogenMod is the worst performing OSs in terms of rate at which failures are detected.
C) In terms of the ability to detect bugs Sailfish performs the best.

Future works would focus on multi-release data modeling, as we consider this as a major limitation to application of the derived results.

Acknowledgements

We thank Innopolis University for generously funding this research.

References

[1] M. Abd-El-Barr, F. Gebali, Reliability analysis and fault tolerance for hypercube multi-computer networks, Inf. Sci. (Ny) 276 (2014) 295–318.
[2] T. Aldemir, S. Guarro, J. Kirschenbaum, D. Mandelli, L. Mangan, P. Bucci, M. Yau, B. Johnson, C. Elks, E. Ekici, et al., A benchmark implementation of two dynamic methodologies for the reliability modeling of digital instrumentation and control systems, Nureg/Cr-6985. Washington, DC: US Nuclear Regulatory Commission, 2009.
[3] R. Aliev, W. Pedrycz, V. Kreinovich, O. Huseynov, The general theory of decisions, Inf. Sci. (Ny) 327 (2016) 125–148.
[4] V. Almering, M. van Genuchten, G. Cloudt, P. Sonnemans, Using software reliability growth models in practice, Softw., IEEE 24 (6) (2007) 82–88, doi:10.1109/MS.2007.182.
[5] C.A. Asad, M.I. Ullah, M.J.U. Rehman, An approach for software reliability model selection, in: Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International, vol.1, 2004, pp. 534–539.
[6] G. Avontuur, K. van der Werff, An implementation of reliability analysis in the conceptual design phase of drive trains, Reliab. Eng. Syst. Saf. 73 (2) (2001) 155–165.
[7] V.R. Basili, G. Caldiera, H.D. Rombach, The goal question metric approach, Encyclopedia of Software Engineering, Wiley, 1994.
[8] J. Baussaron, B. Mihaela, G.-R. Léo, G. Fabrice, S. Paul, Reliability assessment based on degradation measurements: how to compare some models? Reliab. Eng. Syst. Saf. 131 (2014) 236–241.
[9] M.K. Bhuyan, D.P. Mohapatra, S. Sethi, A survey of computational intelligence approaches for software reliability prediction, SIGSOFT Softw. Eng. Notes 39 (2) (2014) 1–10.
[10] K.-Y. Cai, Towards a conceptual framework of software run reliability modeling, Inf. Sci. (Ny) 126 (14) (2000) 137–163.
[11] K.-Y. Cai, Z. Dong, K. Liu, Software testing processes as a linear dynamic system, Inf. Sci. (Ny) 178 (6) (2008) 1558–1597.
[12] P. Cao, Z. Dong, K. Liu, K.-Y. Cai, Quantitative effects of software testing on reliability improvement in the presence of imperfect debugging, Inf. Sci. (Ny) 218 (2013) 119–132.
[13] I.D. Coman, A. Sillitti, G. Succi, Investigating the usefulness of pair-programming in a mature agile team, in: Agile Processes in Software Engineering and Extreme Programming: 9th International Conference, XP 2008, Limerick, Ireland. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 127–136.
[14] L. Corral, A. Sillitti, G. Succi, A. Garibbo, P. Ramella, Evolution of mobile software development from platform-specific to web-based multiplatform paradigm, in: Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, in: Onward! 2011, ACM, New York, NY, USA, 2011, pp. 181–183.
[15] E.O. Costa, G.A. de Souza, A.T.R. Pozo, S.R. Vergilio, Exploring genetic programming and boosting techniques to model software reliability, IEEE Trans. Reliab. 56 (3) (2007) 422–434.
[16] Z. Ding, M.-H. Chen, X. Li, Online reliability computing of composite services based on program invariants, Inf. Sci. (Ny) 264 (2014) 340–348. Serious Games.
[17] K.M. El-Said, M.S. El-Sherbeny, Comparing of reliability characteristics between two different systems, Appl. Math. Comput. 173 (2) (2006) 1183–1199.
[18] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Trans. Softw. Eng. 25 (5) (1999) 675–689.
[19] N.E. Fenton, N. Ohlsson, Quantitative analysis of faults and failures in a complex software system, IEEE Trans. Softw. Eng. 26 (8) (2000) 797–814.
[20] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd edition, PWS Publishing Co., Boston, MA, USA, 1998.
[21] O. Fink, E. Zio, U. Weidmann, Quantifying the reliability of fault classifiers, Inf. Sci. (Ny) 266 (2014) 65–74.
[22] M. Finkelstein, On some comparisons of lifetimes for reliability analysis, Reliab. Eng. Syst. Saf. 119 (2013) 300–304.

[23] D. Fisch, A. Hofmann, B. Sick, On the versatility of radial basis function neural networks: a case study in the field of intrusion detection, Inf. Sci. (Ny) 180 (12) (2010) 2421–2439.
[24] S.T. Garren, D.S.P. Richards, General conditions for comparing the reliability functions of systems of components sharing a common environment, J. Appl. Probab. 35 (1) (1998) 124–135.
[25] S. Gokhale, M. Lyu, K. Trivedi, Software reliability analysis incorporating fault detection and debugging activities, in: Proceedings of the The Ninth International Symposium on Software Reliability Engineering, in: ISSRE '98, IEEE Computer Society, Washington, DC, USA, 1998, p. 202.
[26] M. González, J.P. Proenza, L. Almeida, Quantitative characterization of the reliability of simplex buses and stars to compare their benefits in fieldbuses, Reliab. Eng. Syst. Saf. 138 (.) (2015) 163–175.
[27] S.D. Guikema, A comparison of reliability estimation methods for binary systems, Reliab. Eng. Syst. Saf. 87 (3) (2005) 365–376.
[28] R.S. Hanmer, D.T. McBride, V.B. Mendiratta, Comparing reliability and security: concepts, requirements, and techniques, Bell Labs Tech. J. 12 (3) (2007) 65–78.
[29] L. He, X. Zhang, Fuzzy reliability analysis using cellular automata for network systems, Inf. Sci. (Ny) 348 (2016) 322–336.
[30] L. Hu, F. Sun, H. Xu, H. Liu, X. Zhang, Mutation hopfield neural network and its applications, Inf. Sci. (Ny) 181 (1) (2011) 92–105.
[31] A. Jatain, Y. Mehta, Metrics and models for software reliability: a systematic review, in: Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on, 2014, pp. 210–214.
[32] B. Javadi, D. Kondo, A. Iosup, D. Epema, The failure trace archive: enabling the comparison of failure measurements and models of distributed systems, J. Parallel Distrib. Comput. 73 (8) (2013) 1208–1223.
[33] A. Jermakovics, A. Sillitti, G. Succi, Mining and visualizing developer networks from version control systems, in: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, in: CHASE '11, ACM, 2011, pp. 24–31.
[34] J.M. Juran, Managerial Breakthrough, McGraw-Hill, New York, NY, USA, 1964.
[35] P.K. Kapur, H. Pham, A. Gupta, P.C. Jha, Software Reliability Assessment with OR Applications, Springer London, London, 2011, pp. 313–346.
[36] S.-Z. Ke, C.-Y. Huang, K.-L. Peng, Software reliability analysis considering the variation of testing-effort and change-point, in: Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices, in: InnoSWDev 2014, ACM, New York, NY, USA, 2014, pp. 30–39.
[37] T.M. Khoshgoftaar, T.G. Woodcock, Software reliability model selection: a cast study, in: Software Reliability Engineering, 1991. Proceedings., 1991 International Symposium on, IEEE, 1991, pp. 183–191.
[38] J. Kirschenbaum, P. Bucci, M. Stovsky, D. Mandelli, T. Aldemir, M. Yau, S. Guarro, E. Ekici, S.A. Arndt, A benchmark system for comparing reliability modeling approaches for digital instrumentation and control systems, Nucl. Technol. 165 (1) (2009) 53–95.
[39] G.L. Kovács, S. Drozdik, P. Zuliani, G. Succi, Open source software for the public administration, in: Proceedings of the 6th International Workshop on Computer Science and Information Technologies, 2004.
[40] H. Koziolek, B. Schlich, C. Bilich, A large-scale industrial case study on architecture-based software reliability analysis, in: Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, 2010, pp. 279–288, doi:10.1109/ISSRE.2010.15.
[41] U. Kumar, A. Ahmadi, A.K. Verma, P. Varde, Current Trends in Reliability, Availability, Maintainability and Safety: An Industry Perspective, Springer International Publishing, 2015.
[42] X. Lei, F. Wang, F.-X. Wu, A. Zhang, W. Pedrycz, Protein complex identification through Markov clustering with firefly algorithm on dynamic protein interaction networks, Inf. Sci. (Ny) 329 (2016) 303–316. Special issue on Discovery Science
[43] P.L. Li, J. Herbsleb, M. Shaw, Forecasting field defect rates using a combined time-based and metrics-based approach: a case study of openbsd, in: Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on, IEEE, 2005, pp. 10–pp.
[44] M. Liron, B. Melamed, S.S. Yau, Markov reliability models of fault-tolerant distributed computing systems, Inf. Sci. 40 (3) (1986) 183–206.
[45] B. Littlewood, Stochastic reliability growth: a model for fault removal in computer programs and hardware design, IEEE Trans. Reliab. 30 (4) (1981) 313–320.
[46] Lyu, M. R. (Ed.), 1996. Handbook of Software Reliability Engineering. McGraw-Hill, Inc., Hightstown, NJ, USA.
[47] E.Y. Matsumoto, E. Del-Moral-Hernandez, Improving regression predictions using individual point reliability estimates based on critical error scenarios, Inf. Sci. (Ny) 374 (2016) 65–84.
[48] F. Maurer, G. Succi, H. Holz, B. Kötting, S. Goldmann, B. Dellen, Software process support over the internet, in: Proceedings of the 21st International Conference on Software Engineering, in: ICSE '99, ACM, 1999, pp. 642–645.
[49] I. Moser, M. Gheorghita, A. Aleti, Investigating the correlation between indicators of predictive diagnostic optimisation and search result quality, Inf. Sci. (Ny) 372 (2016) 162–180.
[50] Q. Mou, Z. Xu, H. Liao, An intuitionistic fuzzy multiplicative best-worst method for multi-criteria group decision making, Inf. Sci. (Ny) 374 (2016) 224–239.
[51] J. Musa, Handbook of Software Engineering, Van Nostrand Reinhold, 1984, pp. 392–412.
[52] J.D. Musa, A. Iannino, K. Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, Inc., 1987.
[53] S.-K. Oh, W. Pedrycz, The design of self-organizing polynomial neural networks, Inf. Sci. (Ny) 141 (34) (2002) 237–258.
[54] H. Okamura, T. Dohi, A novel framework of software reliability evaluation with software reliability growth models and software metrics, in: High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on, 2014, pp. 97–104.
[55] B.-J. Park, S.-K. Oh, W. Pedrycz, The design of polynomial function-based neural network predictors for detection of software defects, Inf. Sci. (Ny) 229 (2013) 40–57.
[56] W. Pedrycz, Relevancy of fuzzy models, Inf. Sci. 52 (3) (1990) 285–302.
[57] W. Pedrycz, A. Gacek, Temporal granulation and its application to signal analysis, Inf. Sci. (Ny) 143 (14) (2002) 47–71.
[58] W. Pedrycz, E. Roventa, A hierarchical neural model of matching, Inf. Sci. 78 (3–4) (1994) 215–227.
[59] W. Pedrycz, G. Succi, A. Sillitti, J. Iljazi, Data description: a general framework of information granules, Knowl.-Based Syst. 80 (2015) 98–108.
[60] H. Pham, Recent Studies in Software Reliability Engineering, Springer, 2003.
[61] H. Pham, System software reliability, Springer Science & Business Media, 2007.
[62] C. Rahmani, A.H. Azadmanesh, L. Najjar, A comparative analysis of open source software reliability., JSW 5 (12) (2010) 1384–1394.
[63] B. Rossi, B. Russo, G. Succi, Modelling failures occurrences of open source software with reliability growth, in: P. Agerfalk, C. Boldyreff, J.M. Gonzalez-Barahona, G.R. Madey, J. Noll (Eds.), Open Source Software: New Horizons, IFIP Advances in Information and Communication Technology, Vol. 319, Springer Berlin Heidelberg, 2010, pp. 268–280.
[64] M. Scotto, A. Sillitti, G. Succi, T. Vernazza, A relational approach to software metrics, in: Proceedings of the 2004 ACM Symposium on Applied Computing, in: SAC '04, ACM, 2004, pp. 1536–1540.
[65] R. Sehgal, O. Gandhi, S. Angra, Reliability evaluation and selection of rolling element bearings, Reliab. Eng. Syst. Saf. 68 (1) (2000) 39–52.
[66] C. Stringfellow, A.A. Andrews, An empirical method for selecting software reliability growth models, Empir. Softw. Eng. 7 (4) (2002) 319–343.
[67] G. Succi, W. Pedrycz, M. Stefanovic, B. Russo, An investigation on the occurrence of service requests in commercial software applications, Empir. Softw. Engg. 8 (2) (2003) 197–215.
[68] S.M. Syed-Mohamad, T. McBride, Reliability growth of open source software using defect analysis, in: Computer Science and Software Engineering, 2008 International Conference on, 2, IEEE, 2008, pp. 662–667.
[69] N. Ullah, A method for predicting open source software residual defects, Softw. Qual. J. 23 (1) (2015) 55–76.
[70] N. Ullah, M. Morisio, A. Vetro, A comparative analysis of software reliability growth models using defects data of closed and open source software, in: Software Engineering Workshop (SEW), 2012 35th Annual IEEE, IEEE, 2012, pp. 187–192.
[71] N. Ullah, M. Morisio, A. Vetro, Selecting the best reliability model to predict residual defects in open source software, IEEE Comput. 48 (6) (2015) 50–58.

[72] A.K. Verma, S. Ajit, D.R. Karanki, Reliability and Safety Engineering, Springer-Verlag London, 2016.
[73] A. Wood, Predicting software reliability, Computer (Long Beach Calif) 29 (11) (1996) 69–77.
[74] A. Wood, Software reliability growth models, Tandem Technical Report 96 (130056) (1996).
[75] A. Yadav, R. Khan, Critical review on software reliability models 1, Int. J. Recent Trends Eng. 2 (3) (2009) 114–116.
[76] S. Yamada, Software Reliability Modeling Fundamentals and Applications, Springer Japan, 2014.
[77] Z. Yang, S. Yang, Z. Yu, B. Yin, C. Bai, Graphical User Interface Reliability Prediction Based on Architecture and Event Handler Interaction, Springer International Publishing, Cham, 2015, pp. 1003–1010.
[78] Y. Zhou, J. Davis, Open source software reliability model: an empirical approach, in: ACM SIGSOFT Software Engineering Notes, Vol. 30, ACM, 2005, pp. 1–6.
[79] Q. Zhu, J.-M. Xu, X. Hou, M. Xu, On reliability of the folded hypercubes, Inf. Sci. (Ny) 177 (8) (2007) 1782–1788.