Published in IET Software
Received on 31st January 2010
Revised on 13th October 2010
doi: 10.1049/iet-sen.2010.0007

ISSN 1751-8806

When to stop testing: a study from the perspective of software reliability models

M. Garg[1] *R. Lai*[1] *S. Jen Huang*[2]

[1]*Department of Computer Science and Computer Engineering, La Trobe University, Victoria 3086, Australia*
[2]*Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan*
E-mail: lai@cs.latrobe.edu.au

Abstract: The important question often asked in the software industry is: when to stop testing and to release a software product? Unfortunately, in industrial practices, it is not easy for project manager and developers to be able to answer this question confidently. Software release time is a trade-off between capturing the benefits of an earlier market introduction and the deferral of a product release in order to enhance functionalities or to improve quality. The question has a lot to do with the time required to detect and correct faults in order to ensure a specified reliability goal. During testing, reliability measure is an important criterion in deciding when to release a software product. This study helps answer this question by presenting the perspectives from a study of software reliability models, with focuses on reliability paradigm, efficient management of resources and decision making under uncertainty.

1 Introduction

With the increasing demand to deliver quality software, software development organisations need to manage quality achievement and assessment. Researchers [1, 2] have pointed out that the increasing ubiquity of software stems from its general-purpose nature, causing serious problems in achieving sufficient reliability and demonstrating its achievement. Factors affecting quality assessment are the difficulty and novelty of problems tackled; the complexity of the resulting solutions; the need for short development cycles and the difficulty of gaining assurance of reliability, because of the inherently discrete behaviour of software systems. While testing a piece of software, it is often assumed that the correction of errors does not introduce any new errors and the reliability of the software increases as bugs are uncovered and then fixed. Unfortunately, in industrial practice, it is difficult to decide the time for software release.

During testing, reliability measure is an important criterion in deciding when to release a software product. Reliability, defined as the probability that a product will operate without failure under given conditions for a given time interval, is an important non-functional requirement to take into account when this question is raised. The achieved reliability level determines how long testing should continue before the product is stable enough to be released. Software release time is a trade-off between capturing the benefits of an earlier market introduction and the deferral of a product release in order to enhance functionalities or to improve quality. The software release time question has a lot to do with the time required to detect and correct faults in order to ensure a specified reliability goal. Several other criteria, such as the number of remaining faults, failure intensity, reliability requirements or total costs, may also be used to determine the software release time.

In practice, there has been much effort expended in quantifying the reliability associated with a software system through the development of models which govern software failures based on various underlying assumptions [3]. These models are collectively called software reliability models (SRMs). The main goal of these models is to fit a theoretical distribution to time-between-failure data, to estimate the time-to-failure based on software test data, to estimate software system reliability and to design a stopping rule to determine the appropriate time-to-stop testing and to release the software into the market place [4–6]. However, the success of SRMs depends largely on selecting the model that best satisfies the stakeholder's need. Recent research in the field of modelling software reliability [7–10] addresses the key issue of making the software release decision, that is, deciding whether or not a software product can be transferred from its development phase to operational use. Despite various attempts by researchers [11, 12], this question still stands and there is no stopping rule, which can be applied to all types of data sets.

There are many probabilistic and statistical approaches to model software reliability. While testing software, SRMs are useful in measuring reliability for the quality control and testing process control of software development. In particular, SRMs that describe software failure occurrence or fault-detection phenomenon in the system phase are called software reliability growth models (SRGMs). In the testing and validation phase of the software product life-cycle, the common goal of these models is to support the trade-off between the three dimensions, namely, quality,

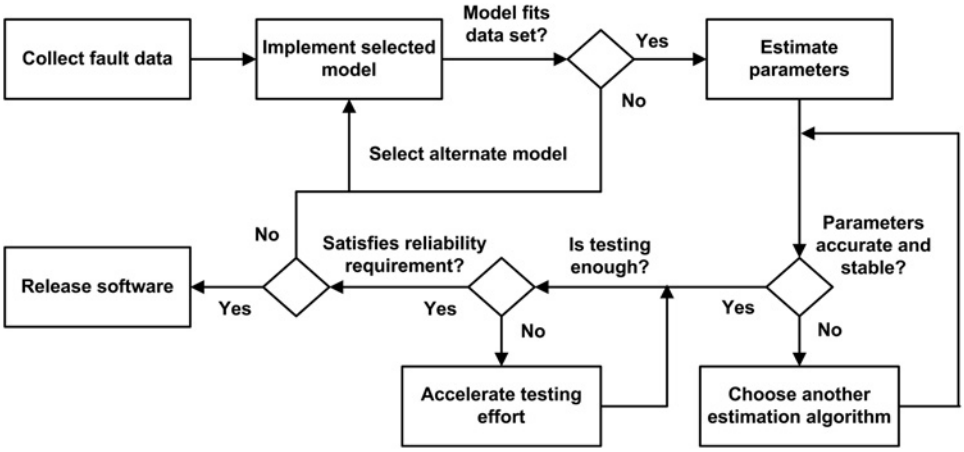

Fig. 1 *Model-based system reliability process*

schedule and cost. Despite their shortcomings – excessive data requirements for even modest reliability claims, difficulty of taking relevant non-measurable factors (such as software complexity, architecture, quality of verification and validation activities and test coverage) into account, etc. – SRMs offer a way to quantify uncertainty that helps in assessing the reliability of a software-based system, and may well provide further evidence in minimising development cost and predicting software release time.

Over the past few decades, quite a number of SRMs have been proposed, which provide a yardstick to predict and estimate reliability during the software development process. In the testing phase, the SRGMs describes the relationship between the cumulative number of detected software faults and the time span of testing [13]. In addition, to predict operational reliability, some SRGMs estimate the number of defects that remain in the code at the time of release to the customers [9, 14, 15]. Each fault removal leads to improved reliability of the system [16, 17]. Detected faults can be removed during debugging; however, new faults may arise in the software, which may cause high failure costs if such software is released in the market.

Independent of how the testing process is organised and how test cases are selected [18, 19], at some point in time during product development, two main questions will arise: how long will the software run before it fails? and how expensive will it be to remove failures from the software? Important steps towards remediation of these problems lie in the ability to manage the testing resources efficiently and an in-depth understanding of system reliability modelling process. The contribution of this paper straddles both research and practice. This paper helps answer this question by presenting the perspectives from a study of SRMs, with focuses on reliability paradigm, efficient management of resources and decision making under uncertainty.

2 Challenges

Determining reliability of a system is a challenging task as at every stage, it is plagued with uncertainties and requires critical decision making. Fig. 1 shows the key evaluation points in a model-based system reliability process and the emerging challenges are highlighted in Table 1. Apart from these challenges, another serious problem is that during the software reliability modelling process, the theory traditionally assumes the software will be tested in the same way it is used in operation. An additional problem when using non-homogeneous poisson process (NHPP)-based SRGMs in operation is the fact that they typically assume instantaneous fault repair time [20]. However, the scenario is not as expected when these problems are confronted in practice.

- *Fault data collection*. A common challenge often faced is poor quality of the failure data, which is needed to estimate the mean value function. There are two fundamental reasons for data quality issues. One is simply a lack of awareness amongst the test and development organisations that the data could be of interest to anyone outside their organisation [9]. Second, it is difficult to gain access to a network of organisations which do not manufacture end-user products (such as telecommunication) and unobtrusively collect the data needed to develop operational profiles. In practice, the test interval often begins with less fault data than required which impugns the prospects of performing operational profile testing. In this context, Bluvband [21] has stated that in order to obtain more trustworthy projections, collected data are the most important aspect to be considered to raise the confidence level of the estimates. Expanding the confidence interval could result in determining accurate

Table 1 Challenges faced during software testing process

Challenge	Description
fault data collection	unobtrusive collected fault data are needed to develop operational profiles
model selection	determination of an appropriate model on the basis of validity of assumptions, performance and prediction of future failure behaviour
parameter estimation	Adequate adjustment of model parameters to obtain accurate and stable estimates
time scale	Selection of an appropriate time scale for testing, such as, execution time instead of calendar time

doi: 10.1049/iet-sen.2010.0007

release time projections by modification of data received at a later date with earlier data.

- *Model selection*. Despite various attempts to solve this ongoing issue by Musa *et al.* [22, 23]; Littlewood [24]; Khoshgoftaar [25]; Kapur *et al.* [26]; Asad *et al.*, [27]; Stringfellow and Andrews [28]; and Li *et al.* [29], there is still no such model that can be applied in all cases. Models that are good in general are not always the best choice for a particular data set, and it is not possible to know in advance what model should be used in a particular scenario. There are two main issues to consider when selecting a model. First, in order to choose a particular model, there is a need for an approach with a high confidence level, which can be applied to various categories of SRMs. Second, identifying the characteristics of software that will ensure that a particular model can be trusted for reliability predictions is ambiguous. Overcoming the difficulty of model selection seems to be a tedious task in the presence of a large number of available SRGMs.
- *Parameter estimation*. Even though statistical models can be used to formulate stopping rules based on predictions of the time to achieve predefined failure intensity, the quality of the prediction is often very poor, and statistical techniques should be involved when deciding the appropriateness of a model for the occasional failure data set [30]. To achieve the required reliability level, there are two procedures worth further consideration. One is to study methods of parameter estimation that can give more stable estimates of the parameters. Another is to consider the variation of the estimates and to take the variation into consideration [3, 31]. If the stable estimates have been achieved but the data from a later phase give unstable results, something must have been changed in the testing process. For correct release time determination, it is useful to have a minimum number of failures or minimum time of testing before stable estimates are expected. Therefore adequately adjusting some specific parameters of an SRGM and adopting the corresponding actions in the proper time interval can greatly help us to more quickly arrive at the desired solution.
- *Time scale*. Selecting the independent variables for the failure intensity function is problematic because of uncertainty in the developer's environment. Therefore determination of an appropriate time scale, such as using the number of person-hours spent; CPU hours; execution time; usage time; calendar time; or the test cases run during testing the software as a measure of exposure time during the test interval, is questionable [21, 32]. Despite of availability of the flexible calibration factors that enable easier alternatives to choose the software execution time scale, current state-of-the-art techniques are far from predicting the precise amount of test effort required in fault fixing and determination of actual software release time.

3 Reliability paradigms

It is typical of the software development industry to demand high levels of product quality because of the significant costs to correct a problem after product release (e.g. recalls in a mass market), the need for high levels of reliability (such as the aerospace industry) or where there are significant safety issues (e.g. medical, aerospace and automotive devices). Software release corresponds with minimising operational cost, and involves a trade-off of functional requirements, time-to-market and development cost. In a paper by Okumoto and Goel [15], the authors state software reliability as the major criteria for stop testing. However, other criterion, such as, failure intensity and cost are also used for the determination of optimal testing time.

- *Failure intensity*. The decision for software release at a given time interval should be determined by checking whether the current failure intensity has reached an acceptably low level [31, 33, 34]. In most of the practical situations, the intensity function of a software system is either unknown or contains uncertain parameters and thus, they must be estimated from the failure data of the software system [17]. To minimise the inequality between the current and the target failure intensity level and to reach the predefined reliability by the specified software release date, different failure intensity functions should be applied. Musa [32] describes that a somewhat more precise approach to determine when to terminate testing would be to compute the pth percent confidence limit for normalised failure intensity and to terminate testing when that value has reached the failure intensity objective. Therefore selection of an appropriate failure intensity function may also be important in determining total effort as well as planning resource allocation during the testing phase.
- *Cost*. Before releasing a software product, an important decision from an economic standpoint is whether to continue testing, stop testing or scrap the software [35, 36]. Delay in software implementation can lead to cost overruns as well as lost business opportunity. If testing at the end of the project stage is stopped too early, significant defects could be released to intended users and the software manufacturer could incur the post-release cost of fixing resultant failures later. Conversely, if testing proceeds too long, the cost of testing and the opportunity cost could be substantial. Hence, cost optimisation is a trade-off where, in theory, the objective is to minimise total cost, that, development cost and risk cost, while maximising reliability claims of a software product [3, 37].

4 Deciding from a reliability perspective

Literature on software testing has been in existence since the beginning of computer science [38, 39]. Software testing serves as a way to measure and improve software reliability. Reliability, being one of the most important quality attributes of commercial software, quantifies software failures during the development process. Despite using good engineering methods that can largely improve software reliability, in practice defect-free software products cannot be easily achieved. The uncertainty in the reliability level makes it difficult to accurately estimate the expected number of post-release defects. This leads to the implementation of a combination of non-analytical methods and activities to decide when a software product is good enough for release. In Table 2, we have summarised these essential activities that contribute evenly in deciding precise scheduled software delivery time.

Testing may be effective in showing the presence of defects, but it is inadequate for showing their absence. The number of tests required to achieve any given level of test coverage increases exponentially with software size. Testing plays an important role in the design, implementation, validation and software release process. In order to increase reliability during testing, it is mandatory to have a comprehensive test plan that ensures all requirements are included and tested. Sommerville [40] has presented literature on Planning of Testing, that is, given the current testing schedule, the time when testing will be completed

Table 2 Desired activities to determine precise delivery schedule

Activities	Description
verification and validation	before the testing and deployment of software products, verification and validation are necessary steps to trigger, locate and remove software defects
prediction or estimation models	software reliability modelling has matured to the point that meaningful results can be obtained by applying suitable models to the problem, but before using them, we must carefully choose the appropriate type of model that can best suit product requirements
module testing	the unfeasibility of completely testing a software module complicates the problem because bug-free software cannot be guaranteed for a moderately complex piece of software
Efficient management of resources	since the testing of software requires resources that are limited, their judicious utilisation is very important. Usage of state-of-the-art algorithms and techniques can facilitate the allocation of limited testing resources efficiently and thus the desired reliability objective during software testing can be better achieved.

can be predicted. A software test plan involves any activity aimed at evaluating an attribute or capability of a software product and determining whether it meets specified requirements [19]. The verification and validation activities are needed to trigger, locate and remove software defects. Also, it is equally important to plan in advance the type of model required to obtain meaningful results. Since software reliability is related to the amount of development resources spent on detecting and correcting latent software faults, it is critical for the project managers to be aware of various techniques to allocate the specified testing resources among all the modules to complete testing within the specified time.

4.1 Verification and validation

Software testing involves both verification and validation of the software product. Verification is the activity that ensures the product is built right. Validation is the demonstration that the software implements each of the stated software requirements, for a specific intended use, correctly and completely, confirming the right product is built. Traditionally, testing only the boundary values was sufficient for verification. However, detecting and removing design defects in modern software is difficult as the possible input combinations thousands of users can make across a given software interface are simply too numerous for testers to apply [41]. Therefore in software programs having multiple different inputs (where timing, unpredictable environmental effects and human interactions generate virtually infinite numbers of distinct input combinations) all possible values need to be tested and verified. Hence, the lack of practical applicability of traditional verification approaches for non-functional requirements, such as reliability and safety, has generated a need for exploration of new approaches.

4.2 Prediction and estimation models

Software reliability evaluation measurements and models are increasingly important in developing and testing new software products. In the literature, two types of SRMs are described, supporting a software manufacturer to make quantitative statements about reliability prior to a release decision [42]:

- *Software reliability prediction models*. Software reliability prediction models address the reliability of the software early in the life-cycle, at the requirements, design or coding level, using historical data. Hence, these models are also referred to as quality management models [43]. Reliability is, for example, predicted using fault density models and code characteristics, such as lines of code and nesting of loops, to estimate the number of faults in the software.
- *Software reliability estimation models*. Software reliability estimation models evaluate current and future reliability from faults, beginning with the integration or system testing, of the software. Thus, they are commonly referred to as SRGMs [43]. The estimation is based on test data. These models attempt to statistically correlate defect detection data with known functions, such as an exponential function.

Although software reliability prediction models can be applied during the entire product development process, software reliability estimation models have been formulated to find the optimal release time for software products. Table 3 summarises the main differences between these two types of models. Practically, a favorable choice is to use software reliability prediction models instead, that is, using historical data to make predictions of the densities of expected defects in the different development phases. It requires, however, the availability of such historical data. With regard to reliability, it will support the use of software

Table 3 Differences between prediction and estimation models [42]

Issues	Prediction models	Estimation models
data reference	uses historical data	uses data from the current software development effort
when used in the development cycle	usually made prior to coding and test phases; can be used as early as requirements phase	usually made later in the product life-cycle during testing (after some data have been collected)
time frame	predict reliability at some future time	estimate reliability at either present or some future time

doi: 10.1049/iet-sen.2010.0007

reliability prediction models to estimate pre-release development costs for further testing and the number of residual faults after product release.

Despite the numerous estimation models proposed in the literature, the usefulness of the software reliability estimation models has been heavily criticised. In the first place, most of these models assume a way of working that does not reflect reality [41, 44–46], meaning that the quality of assumptions is low. As a result, several models can produce dramatically different results for the same data set [35, 47], meaning that their predictive validity is also limited. In this context, Fenton and Neil [48] studied the most-widely used SRGMs and observed that these types of models provide little support for determining the reliability of a software product. Their study also showed that the number of pre-release faults is not a reliable indicator to estimate the number of post-release failures, which is normally assumed when using these models in practice. Secondly, Grams [49] claims that these models make strong statements for basically unaltered software, but in the case of new innovative software products, the models are completely worthless. This may explain why these models are mostly applied only in high-reliability industries such as avionics and telecommunications, where high-reliability is achieved during extensive operational use [2].

Since no two models provide exactly the same answers, care should be taken to select the most appropriate model for a project, and too much weight should not be given to the value of the results [50]. In this light, Rae and Robert [51] define evaluation methods for reliability and other non-functional requirements, and proposed different methods for different risk levels.

4.3 Module testing

Large software consists of modules, which are tested individually prior to system testing, known as integration testing. The software release time is usually based on the modular structure of the software and the reliability of each module [31]. Though no conclusion can be drawn about system reliability during module testing, it is desirable to remove as many faults as possible. Littlewood [24] has discussed this issue explicitly where a software system with multiple modules is formulated as a semi-markov process and the asymptotic distribution of the total cost for running the program is analysed, whereas Kubat and Koch [52] discussed the optimal allocation of test time to the module.

In another attempt, Masuda *et al.* [53] developed statistical procedures for estimating the distribution parameters and the number of software bugs for individual modules. For exponential distribution, both the faults x and parameter a are obtained using function

$$f(x) = a\ \exp(-ax) \tag{1}$$

whereas, for a Weibull distribution, the two distribution parameters a and b in a given module are evaluated assuming that the number of faults is unknown

$$f(x) = abx^{b-1}\ \exp(-ax^{b}) \tag{2}$$

Using (1) and (2), the authors proposed a model to determine optimum release time of software with a modular structure. The decision is given, under some general assumptions, for determining whether the software will be released or further testing for another period of time is necessary.

4.4 Efficient management of resources

Software reliability is closely related to the amount of testing effort which is spent on detecting and correcting software faults. Musa *et al.* [23] demonstrated that execution time (i.e. test effort) is a better time domain for software reliability modelling than calendar time. The testing effort is measured by testing resources such as man-power, CPU hours and executed test-cases spent during the testing phase [54]. The failure detection process based on calendar time depends strongly on the time distribution of the test effort. Since the test effort is not homogeneously distributed during testing, the reliability growth analysis based on calendar time data cannot be as accurate as that based on the effort index or on execution time data [55].

Unlike Jelinski and Moranda [56], Yamada *et al.* [54] proposed particular functional forms for the test effort function. According to them test effort dependent model assumes that the test effort follows either Exponential, Weibull or Rayleigh distributions. In another paper, Yamada *et al.* [57] describe the explicit relationship between the calendar time, the amount of test effort, and the number of software faults detected by the testing. They proposed SRGMs incorporating the effect of test effort (time lag between failure observation and fault removal) on the software reliability growth assuming that the number of detected faults to the current test effort expenditure is proportional to the current fault content; and both the test effort during the testing and the software development effort can be described by either Exponential, Weibull, Rayleigh, Logistic or Generalised-Logistic curves. However, in many software testing situations, it is sometimes difficult to describe the test effort expenditure only by these curves, since actual software data can show various expenditure patterns.

4.4.1 Fault-detection rate: From practical field studies, it is evident that one can estimate the testing efforts consumption pattern and predict the trends of the fault-detection rate (FDR). Knowing the number of residual faults can help to determine whether the software is suitable for customers to use or not, and how much more testing resources are required. It can provide an estimate of the number of failures that will eventually be encountered by the customers. The FDR with respect to test effort intensity is proportional to the current fault content in the software and the proportionality increases linearly with each additional fault removal. The optimal release time increases as the FDR in each model decreases, and the difference between the fault finding cost during test phase and that during operational phase increases [58]. Software fault detection during testing phenomena is described by different SRGMs based on NHPP [13]. Schneidewind's model [59] studies the expected number of faults/failures at a given time, whereas Ohba's model [60] describes the fault identification process with respect to time. Furthermore, in the time-dependent FDR model by Goel and Okumoto [61] the time-dependent behaviour of test effort and the consequent reliability growth has been studied.

The FDR is used to measure the effectiveness of fault detection by test techniques and test cases [62]. Test effort should not be assumed constant throughout the testing phase. In fact, instantaneous test effort ultimately decreases during the testing life-cycle so that the cumulative testing effort

approaches a finite limit. The FDR per fault as a software reliability growth index is formulated by $d(t) = rw(t)$, where r is FDR per unit test effort and $w(t)$ is the current test effort expended at testing time t.

4.4.2 Modelling test effort: Traditionally, all time-based SRGMs implicitly assumed that the test intensity is constant over time. However, later research [32] showed that the time measure used to model test effort could be either calendar time or person-hours spent during testing. Jelinski and Moranda [56] pointed out the problem of a time-varying test effort in their original paper. They further refined their model and accounted for the time-varying test effort by a function named exposure rate. Using their model, it was possible to transform the time scale in accordance with the proposed function or express the program hazard rate after the removal of the $(i-1)$st fault. In another attempt, Yamada *et al.* [54] extended the Goel-Okumoto model [61] following the same line of thought as Jelinski and Moranda. They stated that the ratio between the expected number of software failures occurring in a specified time and the current test effort expenditures are proportional to the expected number of undetected faults. Grottke [63] compared these two models and proved that the hazard rate of each fault remaining in the software is not constant, but varies because of the changing intensity of testing.

Recently, many models are developed to control the test effort [7, 10] and to determine the required amount of test effort to detect the specified number of software faults during a given testing time interval. The consumed test effort during testing indicates how the faults are detected effectively in the software during testing and can be modelled by the variations of various statistical distributions, such as, Exponential, Rayleigh, Weibull and Logistic. The comparison criteria of these distributions are usually objective. In research papers [7, 8, 11, 20, 55, 62, 64, 65], the authors have provided data tables (please refer to Table 4 for sources of available data sets) and graphical representation of the results obtained from the comparison among their proposed model and other candidate models describing efficiency to manage test effort during testing.

4.4.3 Test effort patterns: Test effort should not be assumed constant throughout the testing phase. In fact, instantaneous test effort ultimately decreases during the testing life-cycle because the cumulative test effort approaches a finite limit. This assumption is reasonable because no software company will spend infinite resources on software testing. Assuming this, Yamada *et al.* have discussed the Exponential [54] and Rayleigh [57] density functions to model test effort. The Exponential and Rayleigh curves are used to describe the relationship between the test effort consumption and testing time, while in their other papers [70, 71], the more general Weibull density function is used. The test effort described by a Weibull-type distribution can be made to fit or approximate many distributions and represents a flexible test effort by controlling the shape parameter. Thus, it can be used with a wide variety of possible expenditure patterns in actual software projects. Recently, Ahmad *et al.* [7] described the Exponential Weibull (EW) test effort by adding the generalised exponential and Burr type X functions to existing density functions, whereas Huang *et al.* [65] suggested the Logistic density function to describe logistic test effort patterns. The Logistic test effort function was used to derive the resource consumption curve of a software project over its life-cycle. The advantage of the Logistic test effort function is its applicability to various kinds of models [55].

Table 5 shows the various types of test effort patterns observed. The cumulative test effort spent at time t is $W(t)$, whereas b, θ, A and c are constant parameters to specify the function form. The parameter α represents the total amount of test effort required by software testing in the logistic test effort function, b represents the consumption rate for test effort expenditures, a is the total amount of test effort eventually consumed and c represents the shape parameter. As c can take any value greater than zero, various types of time-dependent test effort curves can be described. When $c = 1$ and $c = 2$, we have Exponential and Rayleigh test effort functions, respectively, although the Weibull-type curve can fit the data well under a general software development environment. However, it will have an extreme peak work rate when the shape parameter $c \geq 3$. That is, the test effort curves, when $c = 3$, 4 and 5, have an apparent peak point during the software development process.

Table 4 Sources of available data sets

Project description	Observation time	Faults found	Reference
PL/1 application software (IBM)	19 weeks	328	Ohba [60]
tandem (version 1, HP)	20 weeks	100	Wood [66]
Rome Air Development Centre (RADC) – radar system	152 weeks	1301	Brooks and Motley [67]
RADC- T1 system for real-time command and control application	21 weeks	136	Musa *et al.* [23]
Online Data Entry Software Package	3 weeks	46	Pham [68]
real-time test and debug data collected from software development projects	3 weeks	86	Tohma *et al.* [69]

Table 5 Types of test effort patterns

Curve type	Parameters	Cumulative test effort	Research
exponential	$a > 0$, $b > 0$, $c = 1$, $\theta = 1$	$W(t) = a[1 - \exp(-bt)]$	Yamada *et al.* [54]
Rayleigh	$a > 0$, $b > 0$, $c = 2$, $\theta = 1$	$W(t) = a\left[1 - \exp\left(-\left(\frac{b}{2}\right)t^2\right)\right]$	Yamada and Ohtera [57]
Weibull	$a > 0$, $b > 0$, $c > 0$, $\theta = 1$	$W(t) = a[1 - \exp(-bt^c)]$	Yamada *et al.* [70]
Exponential Weibull	$a > 0$, $b > 0$, $c = 2$, $\theta > 0$	$W(t) = a[1 - \exp(-bt^2)]$	Ahmad *et al.* [7]
logistic	$a > 0$, $A > 0$, $\alpha > 0$	$W(t) = \frac{a}{1+A\exp(-\alpha t)}$	Huang and Lyu [65]

doi: 10.1049/iet-sen.2010.0007

Table 6 Values obtained for different distribution types using Ohba's [60] data set

Distribution	â	r	A	α	AE%	MSE	BIAS	VAR	MRE	$PE_{\text{end of test}}$
exponential	828.252	0.01178	–	–	131.35	140.66	−0.394	1.37	1.42	1.27
Rayleigh	459.08	0.02733	–	–	28.23	268.42	0.830	2.17	2.02	2.50
Weibull	565.35	0.01965	–	–	57.91	122.09	0.034	0.96	0.96	−0.38
exponential Weibull	565.643	0.01964	–	–	57.98	113.10	0.038	0.94	0.94	−0.30
logistic	394.08	0.04272	13.0334	0.2263	10.06	118.59	− 0.032	1.11	1.91	1.05

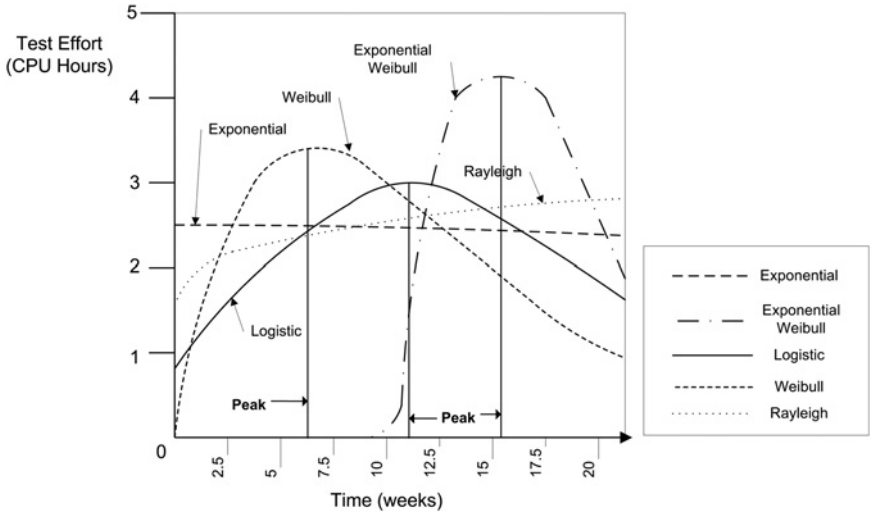

Fig. 2 *Estimated test effort against time for different distribution types using Ohba's [60] data set*

While determining the amount of test effort required to reach the reliability goal specified by the project stakeholders, parameters a, b, c and θ are minimised using the method of least square estimation (LSE) to obtain values of $\hat{a}$ and r for a given distribution. Where, $\hat{a}$ is the expected number of initial faults in software and r is the FDR per unit test effort. Table 6 shows the values obtained for different distributions using Ohba's [60] data set. Fig. 2 shows the different types of curves obtained for these distributions by plotting estimated values in test effort against time graph. The peak of the curves helps in determining the flexibility of a test effort function to predict the actual expenditure patterns during software testing. There are various ways to compare the SRGM for its performance and efficiency. The most preferred method to compare a model is to use either method of accuracy of estimation (AE) [54] or mean square error (MSE) [62]. Lower value of MSE indicates less fitting error, thus better goodness-of-fit. Alternatively, as shown by Huang and Lyu [65], the values for Bias, variation (VAR) and magnitude of relative error (MRE) could also be used as comparison criteria. Whereas, Ahmad *et al.* [7], Kapur *et al.* [10, 72] and Huang [64] have used Kolmogorov–Smirnov goodness-of-fit test to check if the error curve of the model fits within a specified level of significance.

Owing to the complexity of the software systems and the incomplete understanding of the software requirements/specifications, the testing team may not be able to remove the faults perfectly and the original fault is replaced by another fault. The new fault may generate new failures when this component of the software system is traversed during the testing. The multiple removal of faults slow down the fault removal process and gives rise to the S-shaped growth curve. Kapur *et al.* [73] developed an S-shaped SRGM based on NHPP to model the relationship between fault removals under imperfect debugging for test effort consumption. One of the advantages of using fault isolation data is that some faults are removed during the fault isolation process of other faults without failure detection by a test team. Hence, the reliability growth may become S-shaped if there is a peak of effort in the latter half of the test period.

The time-dependent behaviour of test effort expenditure integrated into software reliability growth in practice is generally expressed by a Weibull curve, which is flexible in describing the number of test effort expenditure patterns. The expected number of faults that will be removed at any time in the future can be forecast, if the effort follows a known pattern. When the forecast number of faults falls below the desired number, the test effort needs to be controlled. To increase the intensity of test effort, more staff needs to be employed, but if resources are limited, this may not be feasible.

4.4.4 Change point: SRGMs incorporate a realistic situation encountered in software development where the FDR is not constant over the entire testing process, but changes because of variations in resource allocation, failure density, running environment and testing strategy (called the change point). A change point can occur when the testing strategy and resource allocation are changed as the FDR is not constant throughout the testing process (please refer to Fig. 3). Experimental results obtained by Kapur *et al.* [10] shows that the incorporation of both test effort and change point for SRGM has a fairly accurate prediction capability. Thus, the FDR may not be constant and can be changed at some time moment called the change point [64]. Equations (3) and (4) show the recent and changed forms

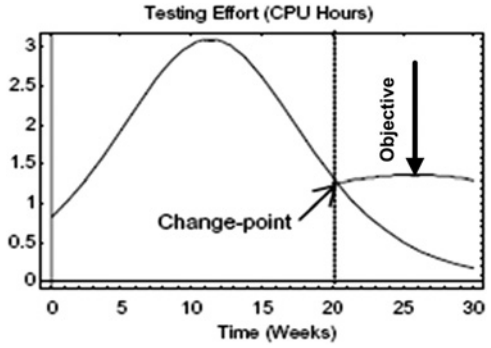

Fig. 3 *Modified test effort curve because of incorporation of change point*

of failure intensity function $\lambda(t)$

$$\lambda(t) = ar_1 w(t)\ \exp\{-r_1 W*(t)\},\quad 0 \le t \le \tau \tag{3}$$

$$\lambda(t) = ar_2 w(t)\ \exp\{-(r_1(W*(\tau) + r_2(W(t) - W(\tau)))\},\quad t > \tau \tag{4}$$

where τ is the change point, r_1 is the FDR before change and r_2 is the FDR when the change-point is exhibited.

4.4.5 Learning factor: Xia *et al.* [74] incorporated the concept of a learning factor in the model developed by Goel and Okumoto [61]. According to them, the learning factor $f(t)$ is proportional to the experience of the tester, which increases with time

$$f(t) = \frac{\alpha + \beta t}{1 + bt},\quad \alpha > 0,\quad \beta > 0 \tag{5}$$

In (5), b is the error detection rate per error per test effort and α, β are constant parameters. Yamada *et al.* [54, 70] and Xia *et al.* [74], have considered the effects of test effort and the learning factor to describe the failure process independently; but according to Chatterjee *et al.* [75], the test effort and the learning factor are dependent on each other. The test effort and the learning factor are inversely proportional to each other and they have a joint effect on the software development process. They assumed that test effort $w(t)$ is inversely proportional to pth power of the learning factor $f(t)$

$$w(t) = \frac{\kappa}{[f(t)^p]} \tag{6}$$

In (6), κ is the proportionality constant and the learning factor $f(t)$ is an increasing function of time t.

5 Deciding under uncertainity

The making of decisions under uncertainty is the aim of statistical decision theory, be it based on the Bayesian or the frequentist paradigm. Decision theory is intimately associated with probability. Although probability theory is a coherent method of quantifying uncertainties, decision theory tries to build an analogous approach to the problem of making decisions. The optimal testing time is a function of many variables, such as, size of the software, level of reliability desired, personnel available, market conditions and penalties of in-process failure. Balancing the desire for high reliability are criteria that favour a short testing time, such as the cost of testing and debugging and the risk of product obsolescence.

The decision-making process to release a product will normally involve different stakeholders, who will not necessarily have the same preferences for the decision outcome. A decision is only considered successful if there is congruence between the expected outcome and the actual outcome [1], which sets requirements for decision implementation. Although most of the optimal software release problems have been discussed within the NHPP framework, a few Bayesian discussions exist in the literature. Researchers [48] argue that a Bayesian set of models takes into account the crucial concepts missing from classical NHPP models. We next explore how Bayesian models have bridged this gap to determine the software release time from a statistical point of view.

5.1 Statistical stopping rules

Stopping rules came into effect to signal the end of software debugging. Statistical issues such as model verification, parameter estimation and the development of a procedure to formalise the debugging of software emanated from the use of SRMs [76]. The major drawback of statistical models used in practice was that they fail to identify estimates for the achieved level of reliability. Secondly, intuition prevails where sharing convincing information is required. In this light, Musa and Ackerman [33] commented that 'system testing becomes guesswork – unless you set it up to apply statistical analysis'. Although there are some statistically proposed stopping rules, unfortunately they are not general and are not widely accepted by software practitioners due to the complexity of the mathematical calculations involved.

Using the prediction capability and future reliability properties, statistical models formulates stopping rules based on predictions of the time to achieve a predefined small failure rate or insignificant fault content. An initial stopping rule based on the Jelinski and Moranda model was proposed by Forman and Singpurwalla [76], but in this empirical study, the authors have neither taken into account the sequential aspect of the decision, nor provided the statistical risk associated with their stopping rule [77]. Their effort is to obtain more accurate estimates by applying statistical techniques such as the profile likelihood and relative likelihood functions [30, 78] instead of the maximum likelihood estimation method owing to instability of the maximum likelihood estimators, especially for the number of initial faults in the software [31]. Later, Jewell [79] extended the Jelinski and Moranda model introducing Bayesian inference for the programs' failure rate and for the number of initial faults. The number of undetected faults at a given time interval is formulated as a Poisson process and his stopping rule is based on the predictive density of this number [30].

In another attempt, Ross [34] gave a procedure to estimate system failure rate when fault rate is not learned. Ross also determines a stopping rule based on the failure rate of the software, and risk level associated with the time to conduct the tests. Yang *et al.* [80] compared various stopping rules and SRMs by their ability to deal with two criteria: when the reliability has reached a given threshold and when the gain in reliability cannot justify the testing cost. Ohtera *et al.* [13] discuss the optimal software release problem taking into consideration software fault detection during testing as well as operational phases. According to Deely

doi: 10.1049/iet-sen.2010.0007

Table 7 Multi-stage sequential testing types

Test plan	Description
one-stage look-ahead	where decisions are made without an accounting of all the possible future stages of testing. The testing plan is simply a sequence of single stage tests; at each stage, the decision must be made to continue test or release using the information obtained from previous tests
fixed-time look-ahead	a sequence of times is specified at to which testing is to stop and, when each time is reached, a decision is made as to whether to release or test until the next time in the sequence
one-bug look-ahead	testing is conducted until a bug is found and fixed, at which point it is decided whether to release or continue testing until another bug is found

and Keats [81], even though there are a wide variety of individual models, the vast majority can be categorised either into decision theoretic models, which require an explicit cost/reliability function to be formulated or some form of the sequential probability ratio test. Statistical stopping rules provide perhaps the most straightforward means of estimating a release time and the obtainable reliability level. Regardless of which approach and which specific model, interest is centred in producing a stopping rule, which gives the user an algorithm for deciding, after each observation, whether or not to stop the test.

5.2 Decision approach

Theoretically, the Bayesian approach provides a coherent framework for making decisions with regard to optimal stopping during the software development phase. The Bayesian decision theoretic formulation of the problem enables one to process information sequentially as one learns more about the failure behaviour of the software and how to adapt with the optimal stopping strategy, accordingly. In other words, the Bayesian approach is adaptive in that it allows for revision of the stopping strategies as well as uncertainties in a dynamic manner [82], thus making it very suitable to answer sequential decision problems. Another strength of the decision theoretic models is that they are designed to generate an optimal stopping rule dynamically by using the concept of mean-utility as a criterion for stop testing [81].

In this context, Dalal and Mallows [4] proposed a Bayesian decision theoretic approach to the problem, but this has the drawback of having an asymptotic flavour (where the number of bugs is indefinitely large), and also that it ignores the history between test times. To overcome these limitations, Singpurwalla [78] proposed another decision theoretic framework, in which the decision as to whether or not to stop testing is based on the maximisation of expected reliability and the minimisation of cost. According to him, at each observation, the decision to stop is based on whether the mean gain from information in future possible observations is worth the additional cost when compared to the information contained in the sample values obtained thus far.

5.3 Sequential testing

In sequential testing, the number of stages to test is random, and the decision on whether to test, and if so, for how long, is made at the completion of the previous stage. According to Caspi and Kouka [16], a sequential stopping rule allows defining iteratively after each fault detection and correction a run time without failure at the end of which it will be possible to decide that the system is fault free at a given statistical risk. This rule can be seen as a robust one since it does not require any parameter estimation.

In the Bayesian formulation for a sequential hypothesis-test, one derives a boundary based on the posterior distribution and invokes various probability levels as the observations are taken to produce a stopping rule [81]. Morali and Soyer [82] presented a Bayesian decision theoretic approach by formulating the optimal release problem as a sequential decision problem. By using a non-Gaussian Kalman filter-type model (originally proposed by Chen and Singpurwalla [83]) to track software reliability, the authors accommodated the step testing case involving sequential decision making. Further, they obtained tractable expressions for inference and determined a one-stage look-ahead stopping rule, where at the end of each test stage, corrections and modifications are made to the software resulting in reliability growth.

The one-stage look-ahead theorem [83, 84], implies that it is optimal to stop testing an additional stage when the expected increase in cost is greater than the expected increase in reliability because of the improvement in failure rate. Thus, it is ideal to stop at a balance point between under testing and over testing. However, a one-stage look-ahead decision is not the only way. In Table 7, we have summarised multi-stage test plans that enable a test manager to decide the ideal time-to-stop testing software. These multi-strategies, such as two-stage decision making, are shown to be superior based on Singpurwalla and Wilson's taxonomy in their book [84]. In a multi-strategy testing plan, one-stage look ahead testing is further fortified by a second stage (fixed-time look-ahead plan) or third stage (one-bug look-ahead) testing if needed.

6 Conclusions

From a practical standpoint, the decision to release a software product is difficult as it is often unstructured, and usually involves various stakeholders; and there might, for example, be reasons to release a system or software product prematurely, despite knowing it still contains defects. The optimal testing time is a function of many variables, such as size of the software, level of reliability desired, personnel available and penalties of in-process failure. Throughout the testing process, corrections and modifications are made to the software to find those critical faults not easily exposed during the earlier manual testing phase. Thus, software testing serves as a way to measure and enhance software reliability. Despite numerous SRGMs proposed to streamline the software-testing process and to improve reliability, no such model exists which could be applied in general to all scenarios. In industrial practice, the underlying reason to use SRGMs during testing is to meet the project manager's expectations to derive precise

software release time projections and to determine how much more testing and repair activity is needed to reach the target reliability threshold within budget, or whether the target reliability can be reached within budget at all.

To answer the question of when to stop testing a piece of software, in this paper we have studied the dimensions of this problem from the viewpoint of both SRGMs and statistical stopping rules discussing software release time issues under the following headings; reliability paradigms, deciding from a reliability perspective and deciding under uncertainty, to provide project managers a better view on these important aspects in making a stop test decision. From our survey, it is evident that most researchers have a good understanding of one of the stated aspects, which helps to decide the exact time-to-stop testing. We have redetermined that from a reliability perspective, it is optimal to stop testing when the failure intensity of software has reached a given threshold. We have also highlighted some of the activities which need more focus in the process to improve reliability of a software product, such as: verification and validation; selection of an appropriate model; module testing and management of testing resources. We ascertain that the lack of practical applicability of traditional verification approaches for non-functional requirements, such as reliability and safety, has generated a need for the exploration of new methods to deal with the numerous input combinations modern software can make in an interface. It is equally important to choose an appropriate type of model to help assess the reliability of a software-based system. In addition, SRGMs provide a measure to evaluate a satisfactory reliability level for large software systems consisting of modules where even a very small portion of errors may result in catastrophic failures. Further, we observed that SRGMs are used widely for the efficient management of testing resources to control the test effort expenditures and to detect more faults in a prescribed time interval to meet the prescribed release schedule.

Since the decision to release software is based on judgments, we have surveyed existing stopping rules and decision theoretic approaches based on Bayesian statistics. We ascertain that in order to achieve the required reliability level proposed by project stakeholders, the approach to determining the appropriate time-to-stop testing shall follow an iterative process, which seeks to balance software quality, capabilities, cost and schedule instead of a pure sequential process. In this paper, we conclude that to predict accurate software release time, the current state-of-the-art techniques used in practice are insufficient as new type of software faults, which are difficult to detect and correct, arise frequently in the modern software and therefore the software development industry is still in need of robust stopping rules; new algorithms and approaches tailored as per the requirements of the industry; and improved SRGMs for more accurate prediction of future reliability properties. In addition, reliability models themselves are far less sufficient in answering the question of when to stop testing, in that, (i) reliability models can only address one aspect of testing goals, which might involve many other quality attributes such as availability, performance, security, etc. (ii) when to stop testing depends on not only the required software reliability level but also various other stakeholder goals including business goals. For future research, there is a need to perform a study of SRGMs focusing on the relationship between development cost and schedule delivery of the software product and the total software cost including the risk costs, such as the penalty cost incurred because of late delivery of software product and the cost of fixing a software fault during the warranty period.

7 References

1 Grady, R.B., Caswell, D.L.: 'Software metrics: establishing a company-wide program' (Prentice-Hall Inc., Upper Saddle River, NJ, USA, 1987)
2 Littlewood, B., Strigini, L.: 'Software reliability and dependability: a roadmap'. Proc. 22nd Int. Conf. on Software Engineering, 2000
3 Xie, M.: 'Software reliability modelling' (World Scientific, Singapore, 1991)
4 Dalal, S.R., Mallows, C.L.: 'When should one stop testing software?', *J. Am. Stat. Assoc.*, 1988, **83**, (403), pp. 872–879
5 Xia, G., Zeephongsekul, P., Kumar, S.: 'Optimal software release policy with a learning factor for imperfect debugging', *Microelectron. Reliab.*, 1993, **33**, pp. 81–86
6 Zeephongsekul, P., Xia, C., Kumar, S.: 'A software reliability growth model primary errors generating secondary errors under imperfect debugging', *IEEE Trans. Reliab.*, 1994, **R-43**, (3), pp. 408–413
7 Ahmad, N., Bokhari, M.U., Quadri, S.M.K., Khan, M.G.M.: 'The exponentiated Weibull software reliability growth model with various testing-efforts and optimal release policy', *Int. J. Quality Reliab. Manag.*, 2008, **25**, (2), pp. 211–235
8 Huang, C.: 'Cost-reliability-optimal release policy for software reliability models incorporating improvements in testing efficiency', *J. Syst. Softw.*, 2005, **77**, pp. 139–155
9 Jeske, D.R., Zhang, X.: 'Some successful approaches to software reliability modelling in industry', *J. Syst. Softw.*, 2005, **74**, pp. 85–99
10 Kapur, P.K., Singh, V.B., Anand, S., Yadavalli, V.S.S.: 'Software reliability growth model with change-point and effort control using a power function of the testing time', *Int. J. Prod. Res.*, 2006, **46**, (3), pp. 771–787
11 Li, P.L., Shaw, M., Herbsleb, J., Ray, B., Santhanam, P.: 'Empirical evaluation of defect projection models for widely-deployed production software systems'. Proc. 12th ACM SIGSOFT, SIGSOFT '04/FSE-12 12th Int. Symp. on Foundations of Software Engineering, New York, USA, 2004, pp. 263–272
12 Myrtveit, I., Stensrud, E., Shepperd, M.: 'Reliability and validity in comparative studies of software prediction models', *IEEE Trans. Softw. Eng.*, 2005, **31**, (5), pp. 380–391
13 Ohtera, H., Yamada, S.: 'Optimum software-release time considering an error-detection phenomenon during operation', *IEEE Trans. Reliab.*, 1990, **39**, (5), pp. 596–599
14 Kenney, G.Q.: 'Estimating defects in commercial software during operational use', *IEEE Trans. Reliab.*, 1993, **42**, (1), pp. 107–115
15 Okumoto, K., Goel, A.L.: 'Optimum release time for software systems based on reliability and cost criteria', *J. Syst. Softw.*, 1980, **1**, pp. 315–318
16 Caspi, P.A., Kouka, E.F.: 'Stopping rules for a debugging process based on different software reliability models.' Int. Conf. on Fault-Tolerant Computing, 1984, pp. 114–119
17 Kim, H., Yamada, S., Park, D.: 'Bayesian approach to optimal release policy of software system', *IEICE Trans. Fund.*, 2005, **E88-A**, (12), pp. 3618–3626
18 Beizer, B.: 'Black box testing' (John Wiley & Sons, 1995)
19 Hetzel, B.: 'The complete guide to software testing' (John Wiley & Sons, 1993)
20 Almering, V., Genuchten, M., Cloudt, G., Sonnemans, P.J.M.: 'Using software reliability growth models in practice', *Proc. IEEE Software*, 2007, **24**, (6), pp. 82–88
21 Bluvband, Z.: 'Reliability centered software testing' Reliability and Maintainability Symp., 2002, pp. 300–305
22 Musa, J.D.: 'Software reliability engineering' (McGraw-Hill, Inc., USA, 1998)
23 Musa, J.D., Iannino, A., Okumoto, K.: 'Software reliability: measurement, prediction, application' (McGraw-Hill, Inc., USA, 1987)
24 Littlewood, B., Miller, D.R.: 'Conceptual modeling of coincidental failures in multi-version software', *IEEE Trans. Softw. Eng.*, 1989, **15**, (12), pp. 1596–1614
25 Khoshgoftaar, T.M., Woodcock, T.G.: 'Software reliability model selection: a case study' Int. Symp. on software reliability Engineering, (ISSRE), 1991, pp. 183–191
26 Kapur, P.K., Garg, R.B., Kumar, S.: 'Contributions to hardware and software reliability' (World Scientific, Singapore, 1999)
27 Asad, Ch.A., Ullah, M.I., Rehman, M.J.: 'An approach for software reliability model selection'. Int. Computer Software and Applications Conf., (COMPSAC), 2004, pp. 534–539

doi: 10.1049/iet-sen.2010.0007

28 Stringfellow, C., Andrews, A.A.: 'An empirical method for selecting software reliability growth models', *Empir. Softw. Eng.*, 2002, **4**, pp. 319–343
29 Li, S., Yin, Q., Guo, P., Lyu, M.R.: 'A hierarchical mixture of software reliability prediction', *Appl. Math. Comput.*, 2007, **185**, pp. 1120–1130
30 Petrova, E., Malevris, N.: 'Rules and criteria for when to stop testing a piece of software', *Microelectron. Reliab.*, 1992, **32**, (1/2), pp. 101–117
31 Xie, M.: 'On the determination of optimum software release time'. Proc. Int. Symp. on Software Reliability Engineering (ISSRE), 1991, pp. 218–224
32 Musa, J.D.: 'Software reliability engineering: more reliable software faster and cheaper' (Author House, 2004, 2nd edn.)
33 Musa, J.D., Ackerman, A.F.: 'Quantifying software validation: when to stop testing?', *IEEE Softw.*, 1989, **6**, (3), pp. 19–27
34 Ross, S.M.: 'Software reliability: the stopping rule problem', *IEEE Trans. Softw. Eng.*, 1985, **SE-11**, (12), pp. 1472–1476
35 Fenton, N.E., Pfleeger, S.L.: 'Software metrics: a rigorous and practical approach' (PWS Publishing Company, 1997)
36 Shanthikumar, J.G.: 'Software reliability models: a review', *Microelectron. Reliab.*, 1983, **23**, pp. 903–949
37 Pham, H., Zhang, X.: 'A Software cost model with warranty and risk costs', *IEEE Trans. Comput.*, 1999, **48**, (1), pp. 71–75
38 Myers, G.J.: 'The art of software testing' (John Wiley & Sons, 1979)
39 Whittaker, J.B., Gerhart, S.L.: 'Toward a theory of test data selection', *Trans. Softw. Eng.*, 1975, pp. 156–173
40 Sommerville, I.: 'Software engineering' (Addison-Wesley, 2001, 6th edn.)
41 Whittaker, J.A.: 'What is software testing? And why is it so hard?', *IEEE Softw.*, 2000, **17**, (1), pp. 70–79
42 RAC: 'Introduction to software reliability: a state-of-the-art review', *Reliability Analysis Center*, 1996
43 Kan, S.H.: 'Metrics and models in software quality engineering' (Addison-Wesley, 2003)
44 Hamlet, D.: 'Are we testing for true reliability?', *IEEE Softw.*, 1992, **9**, (4), pp. 21–27
45 Hecht, H., Hecht M., Tang, D., Brill, R.W.: 'Quantitative reliability and availability assessment for critical systems including software'. Proc. 12th Annual Conf. on Computer Assurance, 1997, pp. 147–158
46 Wood, A.: 'Software reliability growth models: assumptions vs. reality'. Proc. Int. Symp. on Software Reliability Engineering (ISSRE), 1997, pp. 136–143
47 Gokhale, S.S., Marinos, P.N., Trivedi, K.S.: 'Important milestones in software reliability modeling.' SAE Int Communications in Reliability, Maintainability and Serviceability, 1996
48 Fenton, N.E., Neil, M.: 'A critique of software defect prediction research', *IEEE Trans. Softw. Eng.*, 1999, **25**, (5), pp. 675–689
49 Grams, T.: 'The poverty of reliability growth models'. Proc. 10th Int. Symp. on Software Reliability Engineering, 1999
50 Wallace, D., Coleman, C.: 'Hardware and software reliability: application and improvement of software reliability models', *Software Assurance Technology Center, Report 323–08, NASA*, 2001
51 Rae, H., Robert, P.: 'Software product evaluation' (McGraw-Hill, Inc., 1995)
52 Kubat, P., Koch, H.S.: 'Managing test-procedure to achieve reliable software', *IEEE Trans. Reliab.*, 1983, **32**, (3), pp. 299–303
53 Masuda, Y., Miyawaki, N., Sumita, U., Yokohama, S.: 'A statistical approach for determining release time of software system with modular structure', *IEEE Trans. Reliab.*, 1989, **38**, (3), pp. 365–372
54 Yamada, S., Ohtera, H., Narihisa, H.: 'Software reliability growth models with testing-effort', *IEEE Trans. Reliab.*, 1986, **R-35**, (1), pp. 19–23
55 Huang, C., Kuo, S., Lyu, M.R., Lo, J.: 'Effort-index-based software reliability growth models and performance assessment' Int. Computer Software and Applications Conf., (COMPSAC), 2000, pp. 454–459
56 Jelinski, Z., Moranda, P.B.: 'Software reliability research,' in Freiberger, W. (Ed.): 'Statistical computer performance evaluation' (Academic Press, New York, 1972), pp. 465–497
57 Yamada, S., Ohtera, H.: 'Software reliability growth models for testing-effort control', *Eur. J. Oper. Res.*, 1990, **46**, pp. 343–349
58 Yun, W.Y., Bai, D.S.: 'Optimum software release policy with random life cycle', *IEEE Trans. Reliab.*, 1990, **39**, (2), pp. 167–170
59 Schneidewind, N.F.: 'Analysis of error processes in computer software', *SIGPLAN Not.*, 1975, **10**, pp. 337–346
60 Ohba, M.: 'Software reliability analysis models', *Nontropical Issue, IBM J. Res. Dev.*, 1984, **28**, (4), pp. 428–443
61 Goel, A.L., Okumoto, K.: 'Time-dependent error detection rate model for software reliability and other performance measures', *IEEE Trans. Reliab.*, 1979, **R-28**, (3), pp. 206–211
62 Kuo, S., Huang, C., Lyu, M.R.: 'Framework for modeling software reliability, using various testing-efforts and fault-detection rates', *IEEE Trans. Reliab.*, 2001, **50**, (3), pp. 310–320
63 Grottke, M.: 'Software reliability model study', Project PETS, 2001
64 Huang, C.: 'Performance analysis of software reliability growth models with testing-effort and change point', *J. Syst. Softw.*, 2005, **76**, pp. 181–194
65 Huang, C., Lyu, M.R.: 'Optimal release time for software systems considering cost, testing-effort, and test efficiency', *IEEE Trans. Reliab.*, 2005, **54**, (4), pp. 583–591
66 Wood, A.: 'Software reliability growth models.' Tandem Technical Report 96.1, 1996
67 Brooks, W.D., Motley, R.W.: 'Analysis of discrete software reliability models'. Technical Report RADC-TR-80-84, Rome Air Development Centre, 1980
68 Pham, H.: 'Software reliability' (Springer-Verlag London Limited, USA, 2000)
69 Tohma, Y., Jacoby, R., Murata, Y., Yamamoto, M.: 'Hyper-geometric distribution model to estimate the number of residual software fault.' Proc. COMPSAC-89, 1989, pp. 610–617
70 Yamada, S., Hishitani, J., Osaki, S.: 'Software-reliability growth with a Weibull test-effort: a model and application', *IEEE Trans. Reliab.*, 1993, **42**, (1), pp. 100–106
71 Yamada, S., Ohtera, H., Narihisa, H.: 'A testing-effort dependent software reliability model and its application', *Microelectron. Reliab.*, 1987, **27**, (3), pp. 507–522
72 Kapur, P.K., Goswami, D.N., Bardhan, A., Singh, O.: 'Flexible software reliability growth model with testing effort dependent learning process', *Int. J. Appl. Math. Model.*, 2008, **32**, pp. 1298–1307
73 Kapur, P.K., Grover, P.S., Younes, S.: 'Modelling an imperfect debugging phenomenon with testing effort' Int. Symp. of Software Reliability Engineering, (ISSRE), 1994, pp. 178–183
74 Xia, G., Zeephongsekul, P., Kumar, S.: 'Optimal software release policies for models incorporating learning in testing', *Asia Pacific J. Oper. Res.*, 1992, **9**, pp. 221–234
75 Chatterjee, S., Mishra, R.B., Alam, S.S.: 'Joint effect of test effort and learning factor on software reliability and optimal release policy', *Int. J. Syst. Sci.*, 1997, **28**, (4), pp. 391–396
76 Forman, E.H., Singpurwalla, N.D.: 'An empirical stopping rule for debugging and testing computer software', *J. Am. Stat. Assoc.*, 1977, **72**, (360), pp. 750–757
77 Forman, E.H., Singpurwalla, N.D.: 'Optimal time intervals for testing hypotheses on computer software errors', *IEEE Trans. Reliab.*, 1979, **28**, (3), pp. 250–257
78 Singpurwalla, N.D.: 'Determining an optimal time interval for testing and debugging software', *IEEE Trans. Softw. Eng.*, 1991, **17**, (4), pp. 313–319
79 Jewell, W.S.: 'Bayesian extensions to a basic model of software reliability', *IEEE Trans. Softw. Eng.*, 1985, **SE-11**, (12), pp. 1465–1471
80 Yang, M.C.K., Chao, A.: 'Reliability-estimation and stopping-rules for software testing, based on repeated appearance of bugs', *IEEE Trans. Reliab.*, 1995, **44**, (2), pp. 315–321
81 Deely, J.J., Keats, J.B.: 'Bayes stopping rules for reliability testing with the exponential distribution', *IEEE Trans. Reliab.*, 1994, **43**, (2), pp. 288–293
82 Morali, N., Soyer, R.: 'Optimal stopping in software testing', *Naval Res. Logist.*, 2002, **50**, (1), pp. 88–104
83 Chen, Y., Singpurwalla, N.D.: 'A non-Guassian Kalman filter model for tracking software reliability', *Stat. Sin.*, 1994, **4**, pp. 535–548
84 Singpurwalla, N.D., Wilson, S.P.: 'Statistical methods in software engineering: reliability and risk' (Springer-Verlag, 1999)