
Empirical Software Engineering, 7, 319–343, 2002.

An Empirical Method for Selecting Software Reliability Growth Models

C. STRINGFELLOW stringfellow@mwsu.edu
Computer Science Department, Midwestern State University, Wichita Falls, TX 76308

A. AMSCHLER ANDREWS aandrews@eecs.wsu.edu
School of Electrical Engineering and Computer Science, Washingron State University, Pullman, WA 99164

Abstract. Estimating remaining defects (or failures) in software can help test managers make release decisions during testing. Several methods exist to estimate defect content, among them a variety of software reliability growth models (SRGMs). SRGMs have underlying assumptions that are often violated in practice, but empirical evidence has shown that many are quite robust despite these assumption violations. The problem is that, because of assumption violations, it is often difficult to know which models to apply in practice.

We present an empirical method for selecting SRGMs to make release decisions. The method provides guidelines on how to select among the SRGMs to decide on the best model to use as failures are reported during the test phase. The method applies various SRGMs iteratively during system test. They are fitted to weekly cumulative failure data and used to estimate the expected remaining number of failures in software after release. If the SRGMs pass proposed criteria, they may then be used to make release decisions. The method is applied in a case study using defect reports from system testing of three releases of a large medical record system to determine how well it predicts the expected total number of failures.

Keywords: Software reliability, release decisions.

1. Introduction

Methods that estimate remaining defects (or failures) in software can help test managers make release decisions during testing. Various estimation models exist to estimate the expected number of total defects (or failures) or the expected number of remaining defects (or failures). Static defect estimation models include capture–recapture models (Briand et al., 1997, 1998; Eick et al., 1992; Runeson and Wohlin, 1998; Vander Wiel and Votta, 1993; Wohlin and Runeson, 1995, 1998; Yang and Chao, 1995), curve-fitting methods, such as the Detection Profile Method and the Cumulative Method (Briand et al., 1998; Wohlin and Runeson, 1998), and experience-based methods (Biyani and Santhanam, 1998; Yu et al., 1988). Software reliability growth models (SRGMs) (Goel 1985; Goel and Okumoto, 1979; Kececioglu, 1991; Musa and Ackerman, 1989; Musa et al., 1987; Yamada et al., 1983, 1985, 1986) have also been used to estimate remaining failures.

The problem with using SRGMs to estimate failure content is that they have underlying assumptions that are often violated in practice. Examples of common

assumption violations are: performing functional testing rather than testing an operational profile, varying test effort due to holidays and vacations, imperfect repair and introduction of new errors, calendar time instead of execution time, defect reports instead of failure reports, partial or complete scrubbing of duplicate defect reports, and failure intervals that are not independent of each other, etc. Despite the fact that empirical evidence has shown that many of the models are quite robust in practice, it is often difficult to decide which model to apply in light of these assumption violations. The "best model" can vary across systems, and even releases. We cannot expect to select one model and apply it successfully in later releases or other systems. The underlying problem is the complexity of the factors that interact while influencing software reliability. Cai et al. (1991) consider four major ones: the nature of tests, the characteristics of the software, human intervention, and the debugging process.

A possible strategy is to apply a series of models as soon as testing has progressed to a point in the test plan that it makes sense to worry about whether to stop testing. This will work, if there is a systematic strategy to

- determine the best models as testing progresses and
- determine whether to stop.

This paper proposes a selection method that is able to determine the best model(s) for estimating the total number of failures in the software. From this estimate, the expected number of remaining failures may be calculated, and this may be used to help in making release decisions. A case study validates the selection method empirically on three releases of a large medical record system.

Section 2 describes several well-known SRGMs and their underlying assumptions. It also reports on empirical evaluations that highlight both the fragility and robustness of using SRGMs when assumptions of these models are violated. Section 3 describes the method for model selection used in this study. Section 4 applies the method in a case study. Section 5 presents conclusions.

2. Background

2.1. Software Reliability Models

Both static and dynamic software reliability models exist to assess the quality aspect of software. These models aid in software release decisions (Conte et al., 1986). A static model uses software metrics, like complexity metrics, results of inspections, etc. to estimate the number of defects (or faults) in the software. A dynamic model uses the past failure discovery rate during software execution or cumulative failure profile over time to estimate the number of failures. It includes a time component, typically time between failures.

Musa et al. (1987) distinguish between failures and faults. A failure is a departure from how software should behave during operation according to the requirements. Failures are dynamic: The software must be executing for a failure to occur. A fault is a defect in a program, that when executed causes failure(s). While a fault is a property of the program, a failure is a property of the program's execution. Terminology varies quite a bit with software reliability models. For example, some authors like Trachtenberg (1990) use the term error instead of fault. In test organizations, defects may represent faults or failures depending on how they are reported. If they refer to a single identifiable fault, a defect report may represent the description of a fault. On the other hand, if defects can be reported multiple times, defect reports may reflect failures. Duplicate defect reports in this case represent multiple failures due to the same fault. We try in the following to clarify terms so as to make comparison between methods easier.

Dynamic models measure and model the failure process itself. Because of this, they include a time component, which is typically based on recording times t_i of successive failure i ($i \geq 1$). Time may be recorded as execution time or calendar time. These models focus on the failure history of software. Failure history is influenced by a number of factors, including the environment within which the software is executed and how it is executed. A general assumption of these models is that software must be executed according to its operational profile; that is, test inputs are selected according to their probabilities of occurring during actual operation of the software in a given environment (Lyu, 1996). There is a whole body of knowledge of SRGMs (Goel and Okumoto, 1979; Kececioglu, 1991; Lyu, 1996; Musa et al., 1987; Trachtenberg, 1990; Yamada et al., 1983, 1986) with many studies and applications of the models in various contexts (Musa and Ackerman, 1989; Wood, 1996, 1997). Models differ based on their assumptions about the software and its execution environment.

This study used four common SRGMs, the basic Musa or Goel–Okumoto (G–O) model (Goel and Okumoto, 1979; Musa et al., 1987) the delayed S-shaped model (Yamada et al., 1983), the Gompertz model (Kececioglu, 1991), and the Yamada exponential model (Yamada et al., 1986). We selected these, because they represent a range of assumptions.

The Musa model, the G–O model, the delayed S-shaped model (Yamada et al., 1983), the Gompertz model (Kececioglu, 1991), and the Yamada exponential model (Yamada et al., 1986) all assume testing follows an operational profile. They also assume that the software does not change, except that defects are fixed when discovered (repair is immediate and perfect). The models differ in their assumptions either in terms of workload, error size (how often an error or fault leads to failure), or failure intensity. Details of these models are given in Appendix A.

Some models use a non-homogeneous Poisson process (NHPP) to model the failure process. The NHPP is characterized by its expected value function, $\mu(t)$. This is the cumulative number of failures expected to occur after the software has executed for time t. $\mu(t)$ is non-decreasing in time t with a bounded condition, $\mu(\infty) = a$, where a is the expected number of failures to be encountered if testing

time is infinite. Musa's basic model (Musa et al., 1987), the G–O model, the delayed S-shaped model (Yamada et al., 1983), the Gompertz model (Kececioglu, 1991), and the Yamada exponential model (Yamada et al., 1986) are all based on an NHPP. Table 1 summarizes the expected value functions for each model. Detailed explanations of the functions and their parameters are in Appendix A.

The expected value function for cumulative failures can be put into two shape classes: concave and S-shaped (Wood, 1996). S-shaped models are first convex, then concave. The S-shaped growth curves start at some fixed point and increase their growth rate monotonically to reach an inflection point. After this point, the growth rate approaches a final value asymptotically. The S-shaped models reflect an assumption that early testing is not as efficient as later testing, so there is a period during which the failure-detection rate increases. This period terminates, resulting in an inflection point in the S-shaped curve, when the failure-detection rate starts to decrease.

Further, these models assume that all failures are observable. The basic Musa and the G–O model also assume that time between failures is exponentially distributed, that the failure rate for each failure interval is constant, and that failure rate, failures remaining, and failure-detection rate are proportional. The delayed S-shaped model assumes a lower failure rate during early stages of testing and a higher rate later. Time to failure for a given fault follows a gamma distribution. The basic Musa model, the G–O model, and the delayed S-shaped model assume constant testing effort. The Yamada exponential model does not. It explicitly includes an exponential testing effort function to account for variable testing effort. For more detail on these models and an explanation of the parameters used to estimate the cumulative number of failures see Appendix A.

Two common ways for estimating the function's parameters from data are the maximum likelihood and regression methods. The maximum likelihood method estimates the parameters by solving a set of simultaneous equations, usually numerically. Methods for estimating parameters for the G–O model and the delayed S-shaped model are provided in Goel and Okumoto (1979) and Yamada et al. (1983), respectively. For the Yamada exponential model, the methods to estimate the initial values are provided in Yamada et al. (1986). Kececioglu (1991) provides methods to estimate initial values for the Gompertz model. Parameter estimates may also be obtained using non-linear regression. This approach fits the curve to the data and

Table 1. SRGMs used in this study.

Model	Type	Equation ($\mu(t)$)	Reference
G–O or Musa	Concave	$a(1-e^{-bt}), a \geq 0, b > 0$	Goel and Okumoto (1979)
Delayed S-shaped	S-shaped	$a(1-(1+bt)e^{-bt}), a \geq 0, b > 0$	Yamada et al. (1983)
Gompertz	S-shaped	$a(b^{c^t}), a \geq 0, 0 \leq b \leq 1, c > 0$	Kececioglu (1991)
Yamada exponential	Concave	$a(1-e^{-bc(1-e^{(-dt)})}), a \geq 0, bc > 0, d > 0$	Yamada et al. (1986)

estimates the parameters from the best fit to the data, where fit is defined as the difference between the data and the curve function fitting the data.

2.2. Software Reliablity Models in Practice

Goel discussed the applicability and limitations of SRGMs during the software development life cycle in Goel (1985). He proposed a step-by-step procedure for fitting a model and applied the procedure to a real-time command and control software system. His procedure selects an appropriate model based on an analysis of the testing process and a model's assumptions. A model whose assumptions are met by the testing process is applied to obtain a fitted model. A goodness-of-fit (GOF) test is performed to check the model fit before obtaining estimates of performance measures to make decisions about additional testing effort. If the model does not fit, additional data is collected or a better model is chosen. He does not describe how to look for a better model. The problem with this method is that, in practice, many of the models' assumptions are violated, hence none of the models are appropriate. Examples of common assumption violations are: performing functional testing rather than testing an operational profile, varying test effort due to holidays and vacations, imperfect repair and introduction of new errors, calendar time instead of execution time, defect reports instead of failure reports, partial or complete scrubbing of duplicate defect reports, and failure intervals that are not statistically independent of each other, etc.

SRGMs have many assumptions that must be met regarding testing and defect repair that are not valid in actual software development and test environments (Wood, 1997). Common realities that violate these assumptions are:

- It is difficult to define operational profiles and perform operational tests.
- Defects may not be repaired immediately.
- Defect repair may introduce new defects.
- New code is frequently introduced during the test period.
- Failures are often reported by many groups with different failure-finding efficiency.
- Some tests are more or less likely to cause failures than others.
- Test effort varies due to vacations and holidays.

For the software practitioner, the assumptions or conditions for the SRGMs are an open problem, because they are often violated in one way or another. Several studies have found that even so, SRGMs perform well in practice. Two industrial applications of reliability measurement are described in Musa and Ackerman (1989).

Despite distributional assumption violations, in both cases, reliability measurements performed well in predicting failure rates, demonstrating that reliability measurement may be used in industry.

In situations where a software reliability program can be set up, the guidelines by the American Institute of Aeronautics and Astronautics can be helpful (AIAA, 1992). They recommend models for initial screening, evaluate tools, and discuss approaches to data collection. Similarly, (Farr and Smith, 1993), Chapter 9, discusses implementation of model selection criteria. Of course, when the data collection and testing process has not been set up according to those guidelines (as is quite often the case), one may be stuck with whatever data collection is set up for defect reporting. This in turn can lead to violation of model assumptions.

Wood compared the assumptions of SRGMs to Tandem's defect removal environment and points out two key differences (Wood, 1997). These include the introduction of new code during system test and varying defect-finding efficiency of tests. Wood also points out the effects of violating the model's assumptions. They are:

- The number of defects increase during testing rather than remaining constant.
- The defect-finding efficiency of tests per unit of time varies rather than remaining constant.

Wood (1997) proposed three approaches to accommodate model assumption violations with advantages and disadvantages discussed for each. The easiest approach is to ignore the violations. This keeps the model simple, but causes some loss of accuracy. Parameter estimation, however, may compensate. The second solution proposed involves modifying the data. This approach is easy to implement, because the standard models can be used with the modified data. Data modification, however, needs to be based on a mathematical model. A third approach is to derive new models to fit the test environment. These models are more complex. While they are more accurate, they are more difficult to derive and to apply. In Wood's experimentation with data from Tandem, the simple models performed reasonably well despite assumption violations, although the confidence limits were wide. In some cases, the data had to be modified.

In 1996, Wood applied eight reliability models to a subset of software products with four releases to determine which model performed the best in predicting the number of residual defects. This study shows that SRGMs based on cumulative defects predict the number of remaining defects that are close to the number of defects reported after release. The cumulative number of defects by week are fitted with SRGMs. If the fit is good (in this case based on correlation), the function is used for predicting the number of remaining defects in the system. While Wood used correlation as the measure of GOF, there are other criteria for comparing software reliability models such as evaluating and comparing predictive validity (Iannino et al., 1984).

Brocklehurst and Littlewood (1996) also provide methods for analyzing predictive accuracy based on the Braun statistic and χ^2. They also introduce the frequential

likelihood ratio to compare accuracy. The u-plot is used to determine whether, on average, the predictions are close to the real distributions. The results of this analysis are used to recalibrate models for improved prediction accuracy for one-step-ahead predictions. Predictions n-step ahead are more difficult to assess, particularly for large numbers of n. The technique was first published in Abdel-Ghaly et al. (1986). It was applied to several data sets in Keiller et al. (1983). For these data sets it concludes that the predictions arising from the Jelinski–Moranda model are not good and suggests to use the u-plot to recalibrate the model predictions. The effect of recalibration is described in Brocklehurst et al. (1990).

Khoshgoftaar and Woodcock (1991) proposed a method to select a reliability model among various alternatives using the log-likelihood function. They apply the method to the failure logs of a project. The method selected an S-shaped model as the most appropriate one.

Lyu and Nikora (1992) implemented GOF model selection criteria in their tool. Common statistical tests for GOF are χ^2, Kolmogorov–Smirnov, Cramer–von Mises, and Anderson–Darling tests as well as the simple R^2 test. For detailed descriptions of these tests see Rigdon (2000).

In Gaudoin et al. (2002) the power of several of these statistical tests for GOF has been evaluated for a variety of reliability models including those based on an NHPP, on a Weibull distribution, and the Miranda model. The power of the following GOF tests was compared: simple R^2, Kolmogorov–Smirnov, Cramer–von Mises, and Anderson–Darling. Interestingly, the R^2 value is, for most of these models, at least as powerful as the other GOF tests and sometimes the most powerful. While our models are different, it motivated us to base our GOF measure on R^2 as well. Obviously it is a very simple measure, but it is also returned automatically by almost all curve fitting programs.

Wood (1996) suggests using a SRGM in industrial applications in two ways:

1. During system test to predict the additional test effort needed to achieve a desirable quality level in terms of number of remaining failures.
2. At the end of system test to predict the number of remaining failures that could be reported after release.

A useful model must become and remain stable. Predictions week by week should not vary much. According to Wood (1996), the prediction should not vary by more than 10% from week to week. Stability requires a reasonable amount of data that may not be available until several weeks of testing have passed. A useful model must also be reasonably accurate at predicting the number of failures after release.

In Wood (1996), predictions were attempted using execution time, calendar time and number of test cases. In the environment in Wood's study (Wood, 1996), execution time was a better measure of amount of testing. Using calendar time, curve fits diverged in earlier test weeks, making it impossible to obtain predictions. In later weeks, predictions were unstable, especially in comparison to execution time.

3. Approach

There are three main risks when applying SRGMs to estimate failures. They are:

- Not any one model will match a company's development and test process exactly. This will result in a curve fit that may not converge or may have a low GOF value.
- Data is usually grouped by week resulting in a smaller amount of data. A smaller amount of data means it will take longer for the predicted total number of failures to stabilize.
- Testing effort may vary week to week. This will be more of problem with a small amount of data, that is when there are few test weeks.

To handle these risks, the approach applies several SRGMs and selects the models that best fit the data.

The selection method applies SRGMs to cumulative failure data grouped by weeks to determine how well the method predicts the expected total number of failures. More specifically, we consider

1. Can one use calendar time (in our case weeks) when the original models assume execution time?
2. How robust are the models used in this case study when assumptions are not met? In this study, defect reports are grouped by week. Defect reports roughly reflect failures since the reports are (mostly) kept, even if they are duplicates (i.e. based on the same underlying fault, but found by different people). Neither grouping by week nor using defect reports as failures exactly captures model assumptions. In addition, software is released in successive drops during a test cycle, meaning that the size of the software changes some.
3. How does one decide in the testing process which model to use, when a plethora of models exist? Not one model may fit data from each release.
4. Can we put a process framework around the models to make the models useful?
5. Can the selection method aid in choosing the model and making decisions to stop testing?

Cumulative number of failures by week are fitted with SRGMs. If the R^2-value is high, the fit is considered good and we assume that the function can predict the number of remaining defects in the system. The predicted number of total failures is statistically estimated from test time and failure data. One may use different models, data, and statistical techniques. The ones we use include:

- Test time: We investigate the use of calendar time, since data on test cases run and execution time are often not available.

- Failure data: We collect defect reports (failure data) during system test. The data is grouped by week.
- SRGMs: We investigate two concave models, the G–O model (Goel and Okumoto, 1979) and the Yamada exponential model (Yamada et al., 1986), and two S-shaped models, the delayed S-shaped model (Yamada et al., 1983) and the Gompertz model (Kececioglu, 1991). These models are applied to the data at the end of each week. We evaluate the models in terms of:
  - Curve fit: How well a curve fits is given by a GOF test. We chose the R^2-value for its simplicity and high power for other models (Gaudoin, 2002).
  - Prediction stability: A prediction is stable if the prediction in week i is within 10% of the prediction in week $i-1$. Setting a threshold for stability is based on subjective judgement. One may require higher or lower values for this threshold. Our rationale for setting this threshold at 10% was motivated by Wood's suggestion of using 10% as a stability threshold (Wood, 1996).
  - Predictive ability: Error of predictive ability is measured in terms of error (estimate − actual) and relative error (error/actual).
- Statistical technique: A mathematical function is fit to the data. Parameter estimates are obtained using non-linear regression.

Figure 1 shows the steps of the approach. The steps are described in more detail as follows:

1. Record the cumulative number of failures found in system test at the end of each week.

2. Determine whether enough of the test plan has been executed to contemplate stopping testing based on applying a model. This requires setting a threshold that determines when to start applying the models to the defect data. This threshold can be set by the test manager based on required test elements. For example, if there is no sensible way to stop testing before 90% of the test plan is complete (e.g. when new functionality is tested), one would usually wait until that time before applying the models. In the absence of such information, we recommend that at least 60% of planned testing be complete before applying the SRGMs. This is because empirical studies in Musa et al. (1987) and Wood (1996) indicate that the models typically do not become stable until 60% of testing is complete. Once at the end of each week (after the testing threshold had passed) we applied a commercial curve fit program that allows the user to enter the equations for the models. The study applying this method uses the G–O, delayed S-shaped, Gompertz, and Yamada exponential models.

3. The curve fit program estimates a model's parameters by attempting to fit the model to the data.

4. If a fit cannot be performed, a model is said to diverge and the curve fit program outputs a message to that effect. This situation occurs if the model is not appropriate

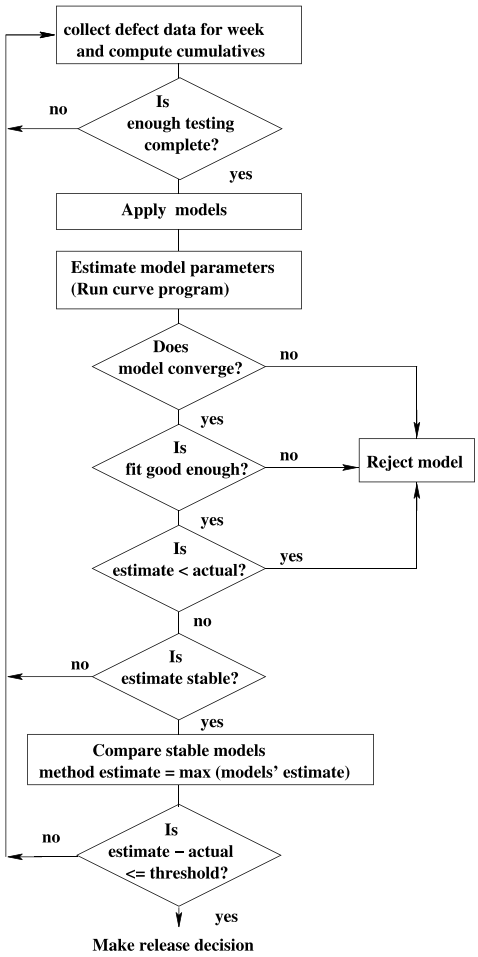

Figure 1. Flowchart for approach.

for the data or not enough data has yet been collected. If this situation occurs after the threshold set for model application, then the model is probably not appropriate for the data and should not be considered in future weeks.

If none of the models converges, then this method fails and some other approach should be used to estimate quality or make release decisions. We would not expect this method to successfully perform in all conceivable situations. This is why we used a collection of methods to assess quality and make release decisions in Stringfellow (2000).

If a curve can be fit to the data for a model, GOF is evaluated based on the R^2-value. The program also returns the estimate for the expected number of total failures (a parameter).

5. Using the R^2-value as an indicator of GOF requires setting a threshold for the R-value that is good enough for the model to be selected as acceptable. What is considered a good fit is somewhat up to the user's judgement. We chose a threshold of $R = 0.95$. This means that the R^2 value will be over 0.90. For the models evaluated in Gaudoin et al. (2002) this was associated with a very high confidence level. While there is no guarantee of that here, since we are fitting different models, it looks plausible. If the R-value is below that threshold, we consider the model not appropriate for the data. It is not considered in future weeks.

(We are looking for a few models that fit the best. An R^2-value threshold that is too low, may include too many models that must be applied week after week that may not perform as well. Alternatively, an R^2-value threshold set too high may eliminate too many models, in fact it may eliminate all of them.)

6. The curve fit program provides an estimate for the parameter a, which is an estimate for the expected number of total failures. If a model's predictions for expected number of total failures are lower than the actual number of failures already found and have been consistently so in prior weeks, the model chosen is inappropriate for the data and should not be used in future weeks. Note that while an estimate that is close, but lower than the actual number of failures looks alright in terms of error, it would underestimate the number of remaining errors (zero) and give a false sense of security. We are interested in conservative estimates.

7. If no model has a stable prediction for the current week, that is within the stability threshold defined (we chose 10% of the prediction of the previous week as the threshold based on Wood's recommendation (Wood, 1997)), additional testing effort is required. System test continues and collects failure data for another week.

8. If there is at least one stable model, the method estimate is taken to be the maximum estimate of all stable models. (This is a conservative choice. Alternatively, the method estimate could be the minimum or the mean. If there is a large difference in values for the estimate between different models, one may want to either augment this analysis with other quality assessment methods, as shown in Stringfellow (2000), or choose a subset of models based on the GOF indicator.) System test determines whether additional testing effort is required by comparing the method estimate to the actual number of failures found. If the method estimate is much higher than the actual number of failures found, additional testing effort may be considered for at least another week. If the difference between the prediction and the actual number of failures found is below the acceptability threshold, the decision to stop testing may be considered. This acceptability threshold is set through subjective judgement.

9. System test should apply the models that were not eliminated in previous weeks at the end of system test to estimate the number of remaining failures that could be reported after release. The number of failures after release may be estimated by subtracting the number of known failures in system test from the predicted total number of failures.

There are several points at which this method can fail. Curve fit may fail to converge, fit may be below expectations, predictions may not stabilize. We cannot expect one single method to work for all possible cases, particularly when the data is

collected the way it was for our case study. This is why it is important to have failure points clearly identified and to set thresholds appropriately. We now apply the method to a case study to see whether and how we can answer the questions posed at the beginning of this section.

4. Case Study

4.1. Data

The failure data come from three releases of a large medical record system, consisting of 188 software components. Each component contains a number of files. Initially, the software consisted of 173 software components. All three releases added functionality to the product. Over the three releases, 15 components were added. Between three and seven new components were added in each release. Many other components were modified in all three releases as a side effect of the added functionality.

The tracking database records the following attributes for defect reports:

- defect report number
- release identifier
- phase in which defect occurred (development, test, post-release)
- test site reporting defect
- defective entity (code component(s), type of document by subsystem)
- whether the component was new for a release
- the report date
- the "drop" in which the defect occurred.

Software is released to testers in stages or "drops." Developers work on drop "$i+1$" when testers test drop "i." Each successive drop includes more of the functionality expected for a given release. Release 1 has three drops. Release 2 has two drops. Release 3 had one drop.

In system test and after release, a defect report is actually a failure. This study uses failure data from the last drop of system test and after release. Table 2 shows cumulative number of failures by week in the last drop of system test for all three releases. The cumulative number of failures after release are also shown in the last row.

4.2. Selection Method Applied

Tables 3–7 show the data from three releases using the G–O, delayed S-shaped, Gompertz and Yamada exponential SRGMs to predict the total number of failures.

Table 2. Failure data for all three releases.

Test week	Cumulative number of failures		
	Release 1	Release 2	Release 3
1	28	90	9
2	29	107	14
3	29	126	21
4	29	145	28
5	29	171	53
6	37	188	56
7	63	189	58
8	92	190	63
9	116	190	70
10	125	190	75
11	139	192	76
12	152	192	76
13	164	192	77
14	164	192	
15	165	203	
16	168	203	
17	170	204	
18	176		
Post-release	231	245	83

Table 3. Predicted total number of failures for Release 1.

Test week	Failures found	Delayed S-shaped		G–O		Gompertz		Yamada	
		Estimate	*R*-value	Estimate	*R*-value	Estimate	*R*-value	Estimate	*R*-value
11	139	830	0.950	6722	0.916*R*	2431	0.970	–	–*R*
12	152	562	0.962	6910	0.934	796	0.975	–	–
13	164	451	0.970	6889	0.946	412	0.979	–	–
14	164	345	0.972	6054	0.955	276	0.979	–	–
15	165	287	0.971	4906	0.960	227	0.979	5625	0.962
16	168	255	0.972	2574	0.961	207*S*	0.979	6557	0.961
17	170	236*S*	0.973	1539	0.961	197	0.980	4925	0.961
18	176	226	0.974	986	0.962	193	0.981	1008	0.961

Table 4. Final estimates and errors by SRGM models not rejected in Release 1.

Model	Estimate (compare to 231)	*R*-value	Error	Relative error
Delayed S-shaped	226	0.974	−5	−0.022
Gompertz	193	0.981	−38	−0.165

Table 5. Predicted total number of failures for Release 2.

Test week	Failures found	Delayed S-shaped		G–O		Gompertz		Yamada	
		Estimate	*R*-value	Estimate	*R*-value	Estimate	*R*-value	Estimate	*R*-value
11	192	186	0.893*R*	195	0.964	200	0.986	262	0.967
12	192	187	0.898	195	0.966	198	0.986	283S	0.970
13	192	188	0.902	194	0.969	197	0.986	284	0.970
14	192	188	0.906	194	0.969	196	0.986	320*D*	0.971
15	203	190	0.906	195*R*	0.969	197*R*	0.986	286	0.972
16	203	191	0.907	196	0.969	198	0.986	265*S*	0.973
17	204	193	0.907	197	0.969	199	0.986	249	0.973

Table 6. Final estimates and errors by SRGM models not rejected in Release 2.

Model	Estimate (compare to 245)	*R*-value	Error	Relative error
Yamada exponential	249	0.973	4	0.016

Table 7. Predicted total number of failures for Release 3.

Test week	Failures found	Delayed S-shaped		G–O		Gompertz		Yamada	
		Estimate	*R*-value	Estimate	*R*-value	Estimate	*R*-value	Estimate	*R*-value
8	63	83	0.974	396	0.966	74	0.979	779	0.966
9	70	84*S*	0.980	293	0.972	77*S*	0.983	297	0.972
10	75	86	0.984	214	0.977	80	0.986	216	0.977
11	76	85	0.986	159	0.978	81	0.988	161	0.978
12	76	84	0.987	129	0.977	80	0.989	130	0.977
13	77	83	0.988	114	0.976	80	0.990	115	0.976

The columns show the test week, the cumulative number of failures found by test week, the prediction for total number of failures (parameter *a*), and the *R*-values (GOF). The week a model is rejected is indicated by an '*R*'. The week a model stabilizes is indicated by an '*S*' in the estimate column. (If a model destabilizes, it is indicated by a '*D*'.)

Table 3 shows that according to our process framework, the G–O model and the Yamada exponential model would be rejected as appropriate models at week 11 (60% of the way through testing). The G–O model is rejected because the *R*-value is 0.916, less than the 0.95 threshold chosen. The Yamada exponential model is rejected because it does not converge in week 11, indicating that a good curve fit with this model is not possible for the data (both rejected models overestimate by a very large amount, if one had used them later on).

Because visual inspection of the cumulative failure curve in Release 1 indicated that it was more S-shaped than concave, we expect the delayed S-shaped model and the Gompertz model to perform better. Figure 2 shows the plot of the failure data from Release 1. It also shows the curves for the delayed S-shaped model and the Gompertz curve. The figure clearly shows that the data has an S-shaped curve.

The correlation values in Table 3 show that the S-shaped models provided a good fit to the data in Release 1. The Gompertz model applied to Release 1 is the first model to stabilize. It stabilizes at week 16. This is the point at which decisions about continuing or stopping testing can be made. The Gompertz model predicts the number of failures at week 16 as 207. Since the delayed S-shaped model is the only stable model at week 16, the method estimate is taken to be the estimate of the delayed S-shaped model. The predicted number of failures, 207, is compared to the actual number of failures, 168. The prediction is 23% more than the actual number and probably not below the acceptability threshold. If this is the case, testing continues.

In weeks 17 and 18, both the delayed S-shaped and Gompertz models are stable. The selection method estimate is the maximum estimate of the two models, 236 in week 17 and 26 in week 18. Week 17 and week 18 estimates are 39 and 28% more, respectively, than the actual number of failures. If this is below the acceptability threshold, testing should continue. System test actually stopped during week 18.

Used as a guide for system testing, the selection method suggests that system testing should have continued beyond week 18. Considering that after release, 55 more failures were found, system testing should probably have continued. This

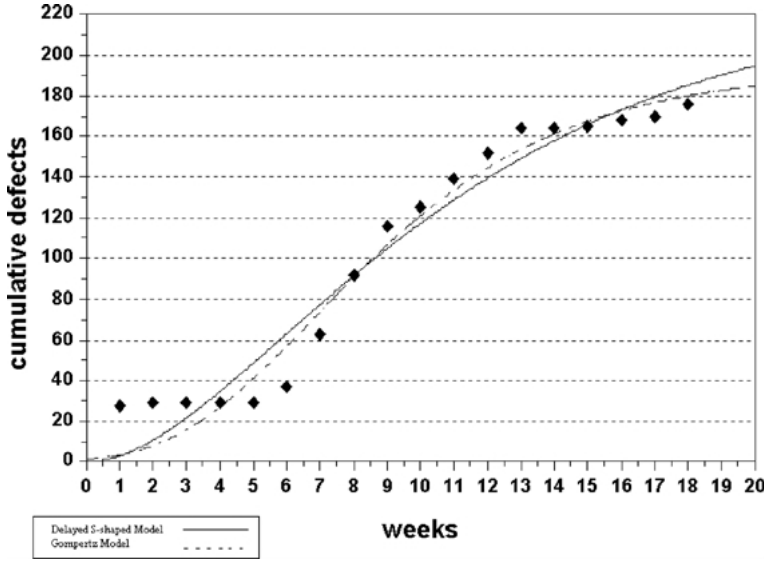

Figure 2. Plot of Release 1 data and SRGM models not rejected.

decision is supported by the opinion of one tester who believed that too many defects were found after Release 1 and that testing should have continued a bit longer.

Together with the 176 failures found by the end of system test and the 55 failures found after release, the total number of failures is 231 in Release 1. The estimates given by the models that were not rejected should be compared to this value. Table 4 shows the prediction provided by the delayed S-shaped model underestimated by five failures, with a relative prediction error of −0.02. This is a very good performance for the model. The Gompertz model underestimated by 38 failures, with a relative error of −0.165. While the Gompertz model provided a better curve fit to the data than did the delayed S-shaped model according to the *R*-value, it underestimated more and had a higher relative error in its prediction of total number of failures. The selection method correctly chose the best stable model, the delayed S-shaped model, to make release decisions in Release 1.

The method for selecting SRGMs based on cumulative failures performed very well in Release 1 in predicting the number of failures close to the total number of failures reported in test and after release. The approach was applied in Releases 2 and 3, as well.

Table 5 shows the SRGMs applied to Release 2 data. All models, except for the Yamada exponential model would be rejected according to the approach. Sixty percent of testing is completed by week 11 in Release 2. At that time, the delayed S-shaped model has both an unacceptably low *R*-value and an estimate for the total number of failures that is less than the actual number of failures found—both reasons to reject the model. The Gompertz model has an acceptable *R*-value, but is rejected in week 15 because it estimates less than the number of problems already found. Table 5 shows that there are several weeks in which no new failures were found. These include weeks 12, 13, 14, and 16. This may reflect a decrease in testing effort by system test and it may affect the fit of some of the S-shaped models and the prediction values.

The S-shaped models, the delayed S-shaped and the Gompertz models, did not fit the data for Release 2 well. Because visual inspection of the cumulative curve indicated that it was more concave than S-shaped, we expect the G–O concave model and the Yamada exponential concave model to perform better.

We evaluated the G–O concave model and the Yamada exponential concave model on Release 2 on a week by week basis. Table 5 shows that the G–O concave model stabilized at week 7, well before 60% of testing was complete. As a predictor of total number of failures, the G–O concave model did not perform well, it predicted fewer total failures than the actual number already found after week 14. According to the approach described, the G–O model should be applied until week 15, until the estimate becomes less than the actual number of failures found. At this point, the G–O model is rejected.

The Yamada exponential concave model performed very well. At week 11, for example, the prediction was 262 failures, while only 192 had been found in system test by that time. The Yamada exponential model would suggest that system testing continue, if 70 failures were unacceptable. By week 17, the model predicts a total of

249 failures, while only 204 have been found. Again, system test should have probably continued testing beyond week 17, since 41 failures were found after release. The Yamada exponential model attempts to account for variations in testing effort and this may be the reason it works better on Release 2.

Figure 3 shows the plot of the failure data from Release 2. It also shows the curve for the Yamada exponential model. The figure shows that the data is concave.

Table 6 shows the final *R*-value, predicted total failures and the relative error for the Yamada exponential model, the only one not rejected for Release 2 data. The Yamada exponential model has a fairly good *R*-value of 0.973. It predicts 249 total failures. The actual total number of failures in Release 2 was 245. The Yamada exponential model over-estimates by four failures and has a relative error of 0.016. This model performed very well for Release 2. Other models also investigated in Wood (1996) did not fit much better than the ones shown in Table 5. The models had *R*-values between 0.867 and 0.978 predicted total failures in the range of 203–212, underestimating by 33–42 failures.

Table 7 shows the S-shaped models had a good fit to the data in Release 3. The *R*-values for the S-shaped models are even better than in Release 1, with *R*-values between 0.974 and 0.979 at week 8, 60% of the way through testing.

The selection method did not reject any of the models investigated in Release 3. Only two models, the S-shaped models, stabilized, both at week 9. The delayed S-shaped model is very stable for the last 6 weeks with predicted failures ranging between 83 and 86. The Gompertz model is also quite stable with predicted failures ranging between 77 and 80.

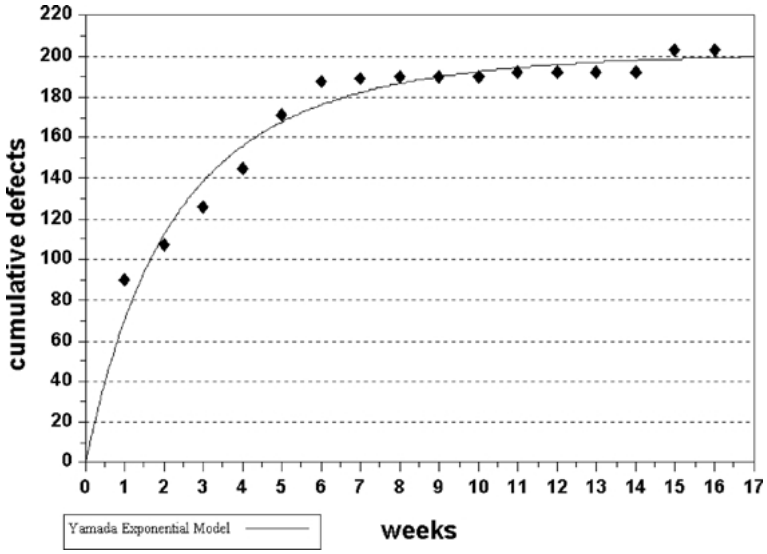

Figure 3. Plot of Release 2 data and SRGM models not rejected.

Table 7 shows that the concave models had a high enough *R*-value at week 8 and through the remainder of test, but the predictions never stabilized. (The '—' in the tables indicate that the models did not converge.) So while all four models must be applied according to the approach, the estimates for the concave models should not be used to make decisions to stop testing.

The selection method uses the estimates of the delayed S-shaped model, because those estimates are higher. At week 8, the delayed S-shaped model predicts the total number of failures is 83 and the actual number of failures found by that time is 63. This is a 20% difference between the estimate and the actual number of failures. This might recommend a continuation of testing, if it is below the acceptability threshold. At week 12, the delayed S-shaped model predicts eight more failures than the actual number of failures that have occurred. This is within 10%. If this is above the acceptability threshold, the system test manager may consider stopping test.

Figure 4 shows the plot of the failure data from Release 3. It also shows the curves for the delayed S-shaped model and the Gompertz model.

In Release 3, post-release reported six failures. Together with the 77 failures found by the end of system test, this is 83 failures. Table 8 shows that the final prediction value using the delayed S-shaped model is exactly 83 failures, giving an error of 0, while the final prediction value using the Gompertz model is 80 defects, an underestimate of 3. As in Release 1, the Gompertz model has a higher *R*-value, and thus a better curve fit to the data, but the estimate of the total number of failures has a higher error—the Gompertz model underestimates more than the delayed S-shaped model. Based on this observation, the estimates based on the delayed S-shaped

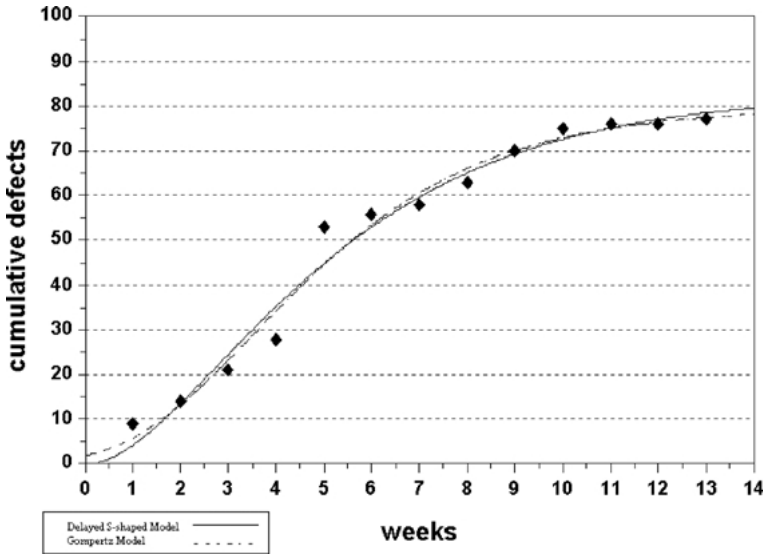

Figure 4. Plot of Release 3 data and SRGM models not rejected.

Table 8. Final estimates and errors by SRGM models not rejected in Release 3.

Model	Estimate (compare to 83)	*R*-value	Error	Relative error
Delayed S-shaped	83	0.988	0	0
Gompertz	80	0.990	−3	−0.036

model are better than the Gompertz model, if testers want to be more conservative. The selection method correctly chooses the best model, the delayed S-shaped model, for Release 2.

5. Conclusion

Results show that the selection method based on empirical data works well in choosing a SRGM that predicts number of failures. The selection method is robust in the sense that it is is able to adjust to the differences in the data. This enables it to differentiate between the models: Different models were selected in the releases.

At least one model of those investigated is acceptable by the time testing is complete enough to consider stopping and releasing the software. In the first and third release, the S-shaped models performed well in predicting the total number of failures. These two releases had defect data that exhibited an S-shape. The data in Release 2 was concave, rather than S-shaped. It is no surprise that the S-shaped models did not perform well on this data. The Yamada exponential model, however, performed very well on the data from Release 2. (Other concave models underpredict the total number of failures.)

SRGMs may provide good predictions of the total number of failures or the number of remaining failures. Wood's empirical study (Wood, 1996) has shown that predictions from simple models of cumulative defects based on execution time correlate well with field data. In our study, predictions from simple models based on calendar time correlate well with data from our environment. The empirical selection method described in this paper helped in choosing an appropriate model. We would like to caution, though, that this does not guarantee universal success. It is always useful to employ complementary techniques for assessment, as for example in Stringfellow (2000).

Appendix A

Trachtenberg (1990) presents a framework for dynamic SRGMs. The rate at which failures are experienced is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \lambda = \frac{\mathrm{d}f}{\mathrm{d}e}\frac{\mathrm{d}e}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}t} = sdw$$

where x is the number of executed instructions,

e the number of encountered errors (faults)[1],

f the number of failures,

S, s are the initial and current average size of remaining errors (faults) (Thus error size is measured by how many failures were encountered for a particular error (fault)). The average size of remaining errors is proportional to the probability of failure of the remaining software error and d the apparent error (fault) density (measured as number of encountered errors (faults per executed instruction). $D = \mathrm{d}R/r$ is the actual error (fault) density, R and r are the initial and current remaining errors (faults), respectively and W, w are the initial and current workload (measured as the number of instructions per unit time).

The classical models of software reliability can be derived from Trachtenberg's general model by modifying assumptions for the parameters, s, d, and w. Musa's model (Musa et al., 1987), for example, assumes that the average size of remaining errors (faults), s, is constant, that the apparent error (fault) density, d, is the same as the actual error (fault) density, D, and that the workload, w, is constant. In this model, the failure intensity decreases a constant amount each time a defect is removed. (The general assumption is that all defects are corrected when discovered.) The failure rate decreases at the same rate at which remaining errors decrease. The failure intensity, λ, is given in Musa et al. (1987) as

$$\lambda(\mu) = \lambda_0(1 - \mu/v_0)$$

where λ_0 is the initial failure intensity (or hazard rate). v_0 the total number of failures that would occur in infinite time and μ the expected number of failures at a time t.

The failure intensity function can also be expressed in Trachtenberg's model as

$$\lambda = -\frac{\mathrm{d}r}{\mathrm{d}t} = SDWe^{(SDWt/R)}$$

Musa's basic model (Musa et al., 1987) uses failure data from execution to characterize the failure process. The expected number of failures is expressed as a function of time t.

$$\mu(t) = v_0(1 - \mathrm{e}^{-(\lambda_0/v_0)t}) \tag{A1}$$

In the absence of failure data, λ_0 and v_0 must be predicted. If failure data exists, both parameters are estimated using maximum likelihood estimation or some other suitable method.

Given a target failure intensity, one can derive the number of additional failures or the additional amount of execution time needed to reach the desired failure intensity. This provides valuable information to system testers in making decisions to release software.

Musa's basic model (Musa et al., 1987) makes the following assumptions:

- Test input is selected randomly from a complete set of inputs anticipated in actual operation. (The operational profile specifies the probability that a given input will be selected.)
- All software failures are observed.
- Failure intervals are independent of each other.
- The execution time between failures is exponentially distributed and the failure rate is constant during the interval between failures. (The failure rate changes at the correction of each defect.)
- The failure rate is proportional to the number of failures remaining.
- The failure detection rate is proportional to the failure rate.

Software reliability growth models predict the number of failures, μ, at time t, or $\mu(t)$. The G–O model Goel and Okumoto, (1979) (similar to the Musa model) is a concave model. It uses the function equation

$$\mu(t) = a(1 - e^{-bt}), \quad a \geq 0, \quad b > 0 \tag{A2}$$

where a is the expected total number of failures that would occur if testing was infinite. It is the upper limit that the reliability (or number of failures) approaches asymptotically as t approaches infinity ($a = v_0$ in Equation (A1)), b is the rate at which the failure-detection rate decreases. It is a shape factor for the curve (b is λ_0/v_0 in Equation (A1)).

The assumptions for this model are the same as for the Musa model.

The delayed S-shaped model (Yamada et al., 1983) is a modification of the G–O model to make it S-shaped. An S-shaped reliability growth curve describes a reliability growth trend with a lower rate of failures occuring during the early stages of development and a higher rate later. It is given by:

$$\mu(t) = a(1 - (1 + bt)e^{-bt}), \quad a \geq 0, \quad b > 0 \tag{A3}$$

where a is the expected total number of failures that would occur if testing was infinite, ($a = v_0$ in Equation (A1)), b the failure detection rate during the steady-state, that is the value to which the rate converges as t approaches infinity. (The failure intensity rate initially increases from $t = 0$ to $t = 1/b$ and then gradually decreases, approaching zero.)

The delayed S-shaped model makes the following assumptions:

- The failure detection process is a non-homogeneous process, that is the characteristics of the probability distribution vary over time.

- The time to failure of an individual fault follows a gamma distribution with a shape parameter of 2.
- Each time a failure occurs, the error that caused it is immediately fixed, and no other errors are introduced.

So far, models have two parameters; other models may have more. The Yamada exponential model and the Gompertz model are two such examples. The Yamada exponential model, a concave model (Yamada et al., 1986), attempts to account for differences in testing effort. It does not assume that testing effort is constant over the testing period. It is given by the equation:

$$\mu(t) = a(1 - \mathrm{e}^{-bc(1-e^{(-dt)})}), \quad a \geq 0, \quad bc > 0, \quad d > 0 \tag{A4}$$

where a is the expected total number of failures that would occur if testing was infinite. $a = v_0$ in Equation (A1)), b the failure-detection rate per unit testing-effort (b is λ_0/v_0 in Equation (A1)), c and d are parameters in the testing-effort function. To account for a variable amount of effort, c and d are based on assuming an exponential form for the testing effort function (Basili and Zelkowitz, 1978). The parameters are estimated using least-squares.

The model makes the following assumptions:

- The failure process is a non-homogeneous process, that is the characteristics of the probability distribution vary over time.
- Each time a failure occurs, the error that caused it is immediately fixed, and no other errors are introduced.
- Testing-effort is described by an exponential curve.
- The expected number of failures in a time interval to the current testing-effort expenditures is proportional to the expected number of remaining errors.

Another popular model to estimate remaining failures is the Gompertz model (Kececioglu, 1991). It has been widely used to estimate software error content (Yamada et al., 1983). It works by fitting a curve to the data using regression analysis. The Gompertz model (Kececioglu, 1991) is an S-shaped model. It is given by the following equation:

$$\mu(t) = a(b^{c^t}), \quad a \geq 0, \quad 0 \leq b \leq 1, \quad 0 < c < 1 \tag{A5}$$

where a is the expected total number of failures that would occur if testing was infinite ($a = v_0$ in Equation (A1)), b the rate at which the failure detection rate decreases. (b is λ_0/v_0 in Equation (A1)), c models the growth pattern (small values model rapid early reliability growth, and large values model slow reliability growth).

Note

1. Trachtenberg uses the term error while others use the term fault.

References

Abdel-Ghaly, A., Chan, P., and Littlewood, B. 1986. Evaluation of Competing Software Reliability Predictions: *IEEE Transactions on Software Engineering* SE-12(9): 950–967.

AIAA SBOS/COS Software Reliability Working Group, *AIAA Software Reliability Engineering Recommended Practice*, R-013-1992, American Institute of Aeronautics and Astronautics.

Basili, V., and Zelkowitz, M. 1978. Analyzing medium-scale software development. *Proceedings of the Third International Conference on Software Engineering* 116–123.

Biyani, S., and Santhanam, P. 1998. Exploring defect data from development and customer usage on software modules over multiple releases. *Proceedings of the Ninth International Conference on Software Reliability Engineering*. Paderborn, Germany, pp. 316–320.

Briand, L., El Emam, K., and Freimut, B. 1998. A comparison and integration of capture–recapture models and the detection profile method. *Proceedings of the Ninth International Conference on Software Reliability Engineering*. Paderborn, Germany, pp. 32–41.

Briand, L., El Emam, K., Freimut, B., Laitenberger, B., and Laitenberger, O. 1997. Quantitative evaluation of capture–recapture models to control software inspections. *Proceedings of the Eighth International Conference on Software Reliability Engineering*. Albuquerque, NM, pp. 234–244.

Brocklehurst, S., and Littlewood, B. 1996. Techniques for Prediction, Analysis, and Recalibration, In: M. Lyu (ed.): *Handbook of Software Reliability Engineering*. Los Alamitos, CA: McGraw-Hill and IEEE Computer Society press Chapter 4, pp. 119–166.

Brocklehurst, S., Chan, P., Littlewood, B., and Snell, J. 1990. ''Recalibrating software reliability models. *IEEE Transactions on Software Engineering* SE-16(4): pp. 458–470.

Cai, K., Wen, C., and Zhang, M. 1991. A critical review on software reliability modeling. *Reliability Engineering and System Safety* 32: 357–371.

Conte, S., Dunsmore, H., and Shen, V. 1986. *Software Engineering Metrics and Models*. Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc.

Eick, S., Loader, C., Long, M., Votta, L., and VanderWeil, S. 1992. Estimating software fault content before coding. *Proceedings of the International Conference on Software Engineering*. Melbourne, Australia, pp. 59–65.

Farr, W., and Smith, O. 1993. *Statistical Modeling and Estimation of Reliability Functions for Software* (*SMERFS*) *User's Guide*. Dahlgren, VA: Naval Surface Warfare Center.

Frankl, P., Hamlet, R., Littlewood, B., and Strigini, L. 1998. Evaluating testing methods by delivered reliability. *IEEE Transactions on Software Engineering* 24(8): 586–601.

Gaudoin, O., Xie, M., and Yang, B. 2002. A simple goodness-of-fit test for the power-law process, based on the Duane plot. *IEEE Transactions on Reliability* (in press).

Goel, A. L. 1985. Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Reliability* 11(12): 1411–1421.

Goel, A. L., and Okumoto, K. 1979. A time dependent error detection model for software reliability and other performance measures. *IEEE Transactions on Reliability* 28(3): 206–211.

Iannino, A., Musa, J., Okumoto, K., and Littlewood, B. 1984. Criteria for software model comparisons. *IEEE Transactions on Software Engineering* SE-10(6): 687–691.

Kececioglu, D. 1991. *Reliability Engineering Handbook*, Vol. 2, Englewood Cliffs, NJ: Prentice-Hall.

Keiller, P., Littlewood, B., Miller, D., and Sofer, A. 1983. Comparison of software reliability predictions. *Proceedings of the Thirteenth International Symposium on Fault-tolerant Computing*. pp. 128–134.

Khoshgoftaar, T., and Woodcock, T. 1991. Software reliabiity model selection: A case study. *Proceedings of the Second International Symposium on Software Reliability Engineering*. IEEE Computer Society Press, Austin, TX: pp. 183–191.

Lyu, M. (ed.): 1996. *Handbook of Software Reliability Engineering*. New York: McGraw-Hill.

Lyu, M., and Nikora, A. 1992. CASREA—A computer-aided software reliability estimation tool, *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering*, Montreal, CA, pp. 264–275.

Musa, J. 1998. Applying failure data to guide decisions. *Software Reliability Engineering*. New York: McGraw-Hill.

Musa, J., and Ackerman, A. 1989. Quantifying software validation: When to stop testing. *IEEE Software*. 19–27.

Musa, J., Iannino, A., and Okumoto, K. 1987. *Software Reliability*: *Measurement*, *Prediction*, *Application*. New York: McGraw-Hill.

Rigdon, S. 2000. *Statistical Methods for the Reliability of Repairable Systems*. New York: Wiley.

Runeson, P., and Wohlin, C. 1998. An experimental evaluation of an experience-based capture-recapture method in software code inspections. *Empirical Software Engineering*: *An International Journal* 3(4): 381–406.

Stringfellow, C. 2000. An integrated method for improving testing effectiveness and efficiency. PhD Dissertation, Colorado State University.

Trachtenberg, M. 1990. A general theory of software-reliability modeling. *IEEE Transactions on Reliability* 39(1): 92–96.

Vander Wiel, S., and Votta, L. 1993. Assessing software designs using capture–recapture methods. *IEEE Transactions on Software Engineering* 19(11): 1045–1054.

Wohlin, C., and Runeson, P. 1995. An experimental evaluation of capture–recapture in software inspections. *Journal of Software Testing*, *Verification and Reliability* 5(4): 213–232.

Wohlin, C., and Runeson, P. 1998. Defect content estimations from review data. *Proceedings of the International Conference on Software Engineering*. Kyoto, Japan, pp. 400–409.

Wood, A. 1996. Predicting software reliability. *IEEE Computer* 29(11): 69–78.

Wood, A. 1997. Software reliability growth models: Assumptions vs. reality. *Proceedings of the International Symposium on Software Reliability Engineering* 23(11): 136–141.

Yamada, S., Ohba, M., and Osaki, S. 1983. S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability* 32(5): 475–478.

Yamada, S., Ohba, M., and Osaki, S. 1985. Software reliability growth modeling: Models and applications. *IEEE Transactions on Reliability* 11(12): 1431–1437.

Yamada, S., Ohtera, H., and Narihisa, H. 1986. Software reliability growth models with testing effort. *IEEE Transactions on Reliability* 35(1): 19–23.

Yang, M., and Chao, A. 1995. Reliability-estimation & stopping-rules for software testing based on repeated appearances of bugs. *IEEE Transactions on Reliability* 44(2): 315–321.

Yu, T., Shen, V., and Dunsmore, H. 1988. An analysis of several software defect models. *IEEE Transactions on Software Engineering* 14(9): 1261–1270.

Catherine Stringfellow received her B.S. in Math and Computer Science, Cameron University, M.S. in Computer Information Sciences, The Ohio State University, Ph.D. in Computer Science, Colorado State University. Currently an Associate Professor of Computer Science at Midwestern State University, Wichita Falls, Texas. Research interests include software testing and maintenance, software process management, and scientific visualization.

Dr. Anneliese Amschler Andrews holds the Huie-Rogers endowed chair in software engineering at Washington State University. She is the author of a text book and numerous articles in the area of Software Engineering, particularly software testing and maintenance. She holds an M.S and Ph.D. from Duke University and a Dipl.-Inf. from the Technical University of Karlsruhe. She is currently Editor in Chief of the IEEE Transactions on Software Engineering, She has also served on several other editorial boards including the IEEE Transactions on Reliability, the Empirical Software Engineering Journal and the Journal of Software Maintenance. She contributes to many conferences in various capacities including general or program chair, or as a member of the steering or program committee.