Software Reliability Engineering Study of a Large-Scale Telecommunications Software System

D. W. Carman
Bellcore
444 Hoes Lane
Piscataway, NJ 08854
dwc@soac.bellcore.com

A. A. Dolinsky
Bellcore
6 Corporate Place
Piscataway, NJ 08854
dolinad@cc.bellcore.com

M. R. Lyu*
AT&T Bell Labs
600 Mountain Avenue
Murray Hill, NJ 07974
lyu@research.att.com

J. S. Yu**
Chengdu Institute of
Computer Applications
Chengdu, China
jyu@bellcore.com

Abstract

Software reliability is a crucial factor of performance of telecommunications network elements and operational systems. This paper describes the state-of-practice software reliability engineering (SRE) methods that we selected and organized into an SRE framework for use at Bellcore. This framework comprises several SRE methods: determination of a reliability objective for a product, development and use of operational profiles, reliability modeling and estimation (prediction) to manage system testing, estimation of the product's reliability in the field, and subsequent validation of this estimate using actual field data. Reliability modeling involves assessment of several models according to their predictive accuracy and the use of the most accurate model for reliability estimation. We have successfully tested this framework on several pilot projects. As part of these projects, we tested the usefulness of three different reliability modeling tools (ESTM, CASRE, and SRMP), as well as several different system test time metrics. This paper describes one of these pilot projects, involving a large operational system for networks.

1 Introduction

Software reliability is a crucial factor of performance of telecommunications network elements and operational systems. Its importance is recognized within Bellcore, the regional Bell telephone companies and the supplier community. It has also been recognized that good software reliability engineering (SRE) practices can substantially enhance network and system reliability. Although they have generally been applying SRE methods, developers of telecommunications software are interested in finding out whether their methods are adequate and state-of-the-practice and how they can be enhanced.

As software reliability engineers, we have been assisting various software development organizations at Bellcore by assessing their current SRE methods, providing recommendations and suport for their enhancements, and helping the organizations to analyze the reliability of their products. As part of our efforts, we selected state-of-the-art SRE methods [1, 2] that would be most effective as well as practical at Bellcore and organized them into an SRE framework. This framework was tested on several pilot projects at Bellcore. This paper describes this SRE framework and its application to one of these pilot projects.

2 Software reliability engineering framework

Figure 1 shows the SRE framework that we proposed for Bellcore's use and applied to the pilot project reported in this paper. The elements of this framework are as follows.

1) Determination of a reliability objective. A reliability objective specifies the minimum value of a product's reliability, expressed in terms of an appropriate reliability metric (such as *failure rate*, *mean time to failure* [*MTTF*], or *reliability function*), that is considered to be "acceptable" to the customer. It is to be used as a benchmark against which the reliability of each product release is to be assessed.

2) Development of the operational profile. A product's operational profile quantifies the usage of the product during field operation: it identifies the operations performed by the product in the field and their relative frequencies (probabilities) of occurrence. It is to be used for guiding system testing: test cases should be developed and selected with the guidance of the operational profile. System testing will then resemble the use of the product

* Formerly at Bellcore
** Currently a visiting researcher at Bellcore

1071-9458/95 $4.00 ©1995 IEEE

in the field, which should result in faster reliability improvement and more realistic estimates of field reliability.

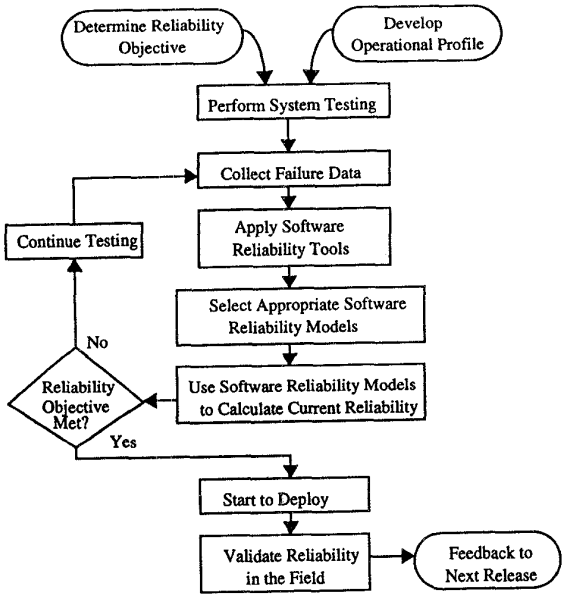

Figure 1: Software reliability engineering process overview

3) Reliability modeling. Reliability modeling is to be used for measuring and estimating (predicting) the reliability of a software release during testing as well as in the field. Modeling uses an appropriate statistical model, which requires appropriate test (or field) failure data (failure counts or the times of their occurrence). Several models should be considered and assessed for their predictive accuracy, in order to select the most accurate model for reliability estimation. This assessment should be performed for each product release, since a different release could have a different "most accurate" model. Assessment of predictive accuracy should use state-of-the-practice measures [3], and should consider factors such as bias, unreal trends, noise, and relative accuracies of models due to all factors combined. An appropriate automated modeling tool should be used.

Reliability measurement and estimation should be performed at intermediate points and at the end of system test. At intermediate points, reliability calculations will provide a measure of the product's reliability growth and a prediction of the length of additional time required until the product can be released to customers. At the end of testing, reliability calculations will enable one to validate the accuracy of this prediction and will also show the difference, if any, between the product's reliability and the reliability objective.

4) Field reliability estimation. The product's reliability measurement at the end of system test can be used to estimate its reliability in the field. To do this, one must know the value of the product's *testing compression factor* (TCF), which provides a connection between the product's test and operational phases, as explained in Section 4.3.

5) Field reliability validation. This involves comparison of the predicted field reliability of the product with the actual reliability measured from field failure data. This validation not only establishes benchmarks and accuracies for the reliability estimates, but also provides feedback to the SRE process improvement and better parameter tuning. For example, it can help us establish model validity, determine reliability growth, refine the testing compression factor (if required), etc.

3 Project characteristics

The SRE methods were tested on several pilot projects. The pilot project for which we are reporting the results in this paper is a key telecommunication software system for daily telephone operations. This system has been in existence for over 10 years and has gone through a number of releases. It is deployed by the regional Bell telephone companies, usually at multiple sites, with multiple machines running the software per site. It must interact, through a series of complex, automatic interactions, with several other large-scale telecommunications systems. It has human users as well: service order representatives and system administrators. The whole system includes about one million lines of C source code, but the main application is composed of 700K lines of code.

During field operation, system failures are classified by the customer according to severity into one of four severity classes (critical, high, medium and low) and reported to the Bellcore development organization. There, the failure data is entered into a database. The failures are then investigated and the software defects responsible for them identified and repaired. The Bellcore development organization also keeps track of the number of machines operating the system in the field (there are a number of sites and one or more machines per site) and when the machines came on line with each system release. From this data it is possible to determine for each release the cumulative calendar time of field

operation over all machines (cumulative machine months).

The test process for this system is mature and stable across releases. Although it can vary somewhat between releases, typically system testing involves 15 testers over a period of about a dozen weeks. Much of the testing environment, including regression testing, is automated. A number of test metrics are collected automatically. Test failure and related software fault data is collected in an internal problem tracking system. Test failures are classified by severity.

We initiated the pilot study just before one of the latest releases of this system (containing about 250K of new and changed lines of code) was scheduled to start system testing. We were therefore able to introduce test data collection enhancements, measure this release's reliability during system testing, and track its improvement. After this release was deployed in the field, we were able to obtain data about its field failures. We also had test and field failure data for two prior releases that were already deployed in the field. All this information was used to investigate the feasibility and practicality of the SRE methods in the SRE framework we had proposed.

4 Methods, models, tools and data collection

We used the following reliability methods, models, tools and methods of data collection.

4.1 Operational profile development

An operational profile describes how users employ a product (or system): it identifies all the tasks (i.e., the smallest units of work performed by the sytem) that can be initiated by external intervention (by a human user or operator or by another system), together with the probability of occurrence of each task in the system's operational environment. Such tasks correspond to computer or software *runs*. In an operational profile, *runs* are generally grouped into *operations*, which are groupings of *runs* that correspond to similar "work" performed by the system and that utilize similar software. An example of an *operation* in the case of a local telephone switching system would be a local two-party telephone call. A *run* belonging to this *operation* would be a specific local two-party call.

We adopted the method for developing operational profiles described by Musa [2, 4] (although other methods exist, we did not test them). This method has shown itself so far to be flexible and adaptable to a variety of software projects we have worked with. We have been able to apply it to projects where software *operations* are statistically independent as well as to projects where *operations* are arranged in highly correlated sequences that must be represented by *operational scenarios* [2]. When developing an operational profile, we utilize a variety of internal experts who have knowledge relevant to the operational profile. We have found it particularly useful to utilize Bellcore engineers who had worked at one time in a telephone company and are familiar with the environment and application of the product in the field. For some projects it has also been possible to collect data from the users in the field by means of a questionnaire or interviews of the users.

4.2 Reliability modeling tools

We did not invent new reliability models or software tools but adapted and tested, for Bellcore applications, models and tools that are published in the technical literature or are available on the market. About half a dozen computerized tools are currently available. We tested three of these computerized tools: the *Economic Stop Test Model* (ESTM), *Computer-Aided Software Reliability Estimation* (CASRE), and *Software Reliability Modelling Programs* (SRMP).

The ESTM tool [5] is primarily an economic model to help managers decide when to stop testing a software product and release it to customers [6]. This model is based on a trade-off between competing sets of costs experienced by a development organization. On the one hand, there is the cost of performing system test and repairing the software faults detected as failures during testing. This cost is proportional to the testing effort (number of testers involved and the length of testing) and to the number of software faults repaired. On the other hand, there is the cost of diagnosing the software failures encountered in the field and repairing the software defects responsible for those failures. This cost is proportional to the number of expected field failures, which in turn is proportional to the expected number of software faults not removed during system testing. ESTM calculates the cumulative net benefit (net cost savings accrued) as a function of test time (e.g., tester-hours), from which the most economical test time can be determined. To perform its calculations, ESTM needs to estimate the expected number of software faults as a function of test time. ESTM uses the Goel-Okumoto Model (GO) for this purpose [7].

The CASRE tool [8, 9] calculates a product's reliability and represents it in terms of several measures (reliability function, expected number of failures, failure intensity, etc.) as functions of time. They can be used the

to study reliability improvement during a product's testing or operational phase. CASRE permits an analyst to estimate a product's reliability using a number of reliability models and, furthermore, to determine which reliability model possesses the best predictive accuracy for the available test (or operational) data. CASRE calculates four statistical measures for analyzing the predictive accuracy of models: relative accuracy, bias, bias trend and noise [10]. The first three of these are measured using *prequential likelihood*, *u-plot*, and *y-plot*, respectively, first introduced into SRE by Littlewood [11]. We found these measures useful, for they enable us not only to select the relatively best reliability model, but also to determine whether or not this model was sufficiently accurate for the failure data being analyzed and to assess the type and seriousness of the inaccuracies present.

CASRE incorporates a library of twelve reliability models. These models, listed in Table 1, have been adopted from the SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software) tool [12, 13] and are categorized into two classes based on their input data: Time-Between-Failures (TBF) models that take the sequence of time between failures as input data, and Failure-Count (FC) models that take number of failures per time interval as input data. CASRE also incorporates methods for converting TBF input data into FC data (by data grouping) and FC data into TBF data (by randomly distributing failures, using a uniform probability distribution, across the time interval to which they belong, or by uniformly distributing the failures across the time interval). As a result, either class of models can be used to analyze failure data. According to [14], conversion of FC data into TBF data by randomly distributing failures should result in small errors. We verified this on some of our test failure data: we found the differences in the calculated reliability measures to be less than five percent.

The SRMP tool [11, 15] is based on the same philosophy as CASRE and uses the same four statistical measures of predictive accuracy, which are referred to in SRMP as prequential likelihood [16], u-plot, y-plot and sum of deviance. SRMP differs from CASRE mainly in its user interface, inputs, measures of reliability, and many model implementations in its model library. For example, it can only take TBF input data [15]; FC data must therefore be converted into TBF data. SRMP's model library contains the nine reliability models listed in Table 1.

4.3 Testing compression factor

Testing compression factor (TCF) is defined as "the ratio of execution time required in the operational phase to execution time required in the test phase to cover the input space of the program" [14]. It provides a connection between the reliability of the product during system testing and during field operation, for according to [14]: "...the failure times obtained in test can be used to estimate test phase failure intensities...These failure intensities should then be *divided* by the testing compression factor C to obtain the corresponding failure intensities to be expected in operation." This connection can be used to predict a product release's reliability in the field from its reliability measured during system testing. This approach is expected to provide realistic predictions from the vantage point of the customer only if system testing is performed according to the release's operational profile.

To use this approach, the value of TCF must be known. This value can be calculated from the release's operational profile, together with information on the run times of all the types of *runs* that can be initiated from the product release's input space [14]. This information is frequently unavailable or difficult to obtain, however, and the calculation of TCF difficult, even when this information is available. Another, and often simpler, method of calculating TCF is to take the ratio of the failure rate (intensity) at the end of system test to the failure rate at the start of field operation, and this is the method we have been using on our projects.

To use this second method, however, the values of the failure rate at the end of system test and at the start of field operation must be available. The TCF calculation can therefore be performed only for a product release that has already been deployed in the field, for which the reliability in the field can be measured. If one wishes to predict the field reliability of a new product release that has just completed system testing, one may have to use the TCF value calculated for previous releases of the same product, assuming that the TCF value remains approximately constant across the releases, including the new release. This should be the case if the changes in the operational profile between releases, as well as the run times of the software *run* types, are such that they do not have a significant effect on the value of TCF. We investigated this issue for each project that involved the calculation of TCF. As a verification of the constancy of TCF across releases, we calculated a value of TCF for several earlier releases and compared their values. These values were considered to be approximately constant if they did not differ by more than 20 percent from each other.

4.4 Reliability objective

For the pilot project reported here, the customers had not specified a quantified reliability objective for the product. The software development organization had, therefore, to determine one. It was decided to express this objective as a failure rate, as this was the reliability metric that had the most intuitive appeal to the testing organization. To make any business sense, the reliability objective had to be such that all the customers would consider it acceptable.

CASRE Models	SRMP Models
TBF Models:	JM Model (same as that in CASRE)
Geometric Model (GEO)	Bayesian Jelinski-Moranda Model (BJM)
Jelinski–Moranda Model (JM)	Goel-Okumoto Model (GO) (same as that in ESTM)
Littlewood–Verrall Model (LV)	MO Model (same as that in CASRE)
Musa Basic Model (MB)	Duane Model (DU)
Musa–Okumoto Model (MO)	Littlewood Model (LM)
Nonhomogeneous Poisson Process Model for TBF (NHPP-TBF)	Littlewood Non-Homogeneous Poisson Process Model (LNHPP)
FC Models:	LV Model (same as that in CASRE)
Brooks and Motley Binomial Model (BMB)	Keiller-Littlewood Model (KL)
Brooks and Motley Poisson Model (BMP)	
Generalized Poisson Model (GP)	
Nonhomogeneous Poisson Process Model for FC (NHPP-FC)	
Schneidewind Model (SM)	
Yamada S-shaped Model (YM)	

Table 1: List of the reliability models contained in the CASRE, SRMP, and ESTM model libraries

Since the product has been deployed in the field for a number of years, has gone through a number of releases, and was being supported by the development organization when deployed in the field, there was considerable amount of information on the product's performance in the field: field failures reported by the customers, as well as opinions expressed by the customers to the development organization as to which releases they regarded as "good" and which as "bad." In addition, the product development personnel had information about their own perception of the quality of the different releases: they were able to identify which releases had required a large amount of maintenance effort after deployment in the field and which releases did not. We used this information to develop a reliability objective for the pilot project.

We selected one of the releases of the product and used its failure rate in the field as the reliability objective for the product in the pilot study reported here. This release was required to have the following characteristics. It had been deployed in the field for a sufficiently long period of time and at many sites and was, therefore, able to provide a good measurement of its field failure rate. The customers, as well as developers, considered this release to be of good quality. This release was a "major" one: it incorporated new product features and new lines of code, in addition to correcting problems detected in the field in earlier releases. Test data was available that allowed us to calculate the failure rate during system testing and the TCF value for this release. This TCF value was approximately equal to the value to be used for predicting the field failure of a new release.

4.5 Failure data collection

To measure the reliability in the field for the pilot project, we used the field data stored in Bellcore's databases. Because of the limitations of this data, it was only possible to calculate an "average" failure rate of each product release in the field. A decrease of the field failure rate with operating time could not be measured even though the product was maintained during field operations and included repair of serious defects. Furthermore, the only time metric available for these calculations was cumulative calendar time over all the machines running the software in the field (machine-months). We explored the possibility of enhancing the field data collection procedure to allow the collection of more precise software usage time data, but found it to be not feasible at the present time: a new data collection procedure would interfere with the customers' operations and would require a costly development effort.

To measure the reliability during system testing, we needed a more accurate approximation to the actual software usage time (CPU or execution time) than was provided by calendar time, which was already being collected automatically on the pilot project. For calendar time had shown itself an inaccurate software usage time metric during system testing of other projects we had investigated. Calendar time had included not only the time testers spent on product testing but also the time testers spent on activities unrelated to testing (such as staff meetings and sick leave). As a result, the SRE modeling methods could be unable to detect any reliability growth during testing, even though other evidence indicated its presence. To amend this situation, we enhanced the data collection for the pilot project by

developing a simple manual data sheet to collect both failure counts and staff time (hours spent testing) for each tester on a daily basis.

Staff time did not account for all the software usage time, however, because regression time was automated. We accounted for this additional software usage time by multiplying the staff time by a constant (greater than 1). It was necessary to do this because the additional software usage time was not being captured, with sufficient accuracy for reliability modeling purposes, by the automated data collection system. An analysis of the available data showed, nevertheless, that this additional software usage time was proportional to the staff time, to a good approximation, and this proportionality factor could be estimated. The total software usage time, combining staff time and automated test time, was called staff-execution time.

Even though the manual data sheet for collecting staff time data was simple, the testers considered it burdensome and preferred a totally automated software usage time collection system. We investigated therefore other possible metrics for usage time as well, to see if they could measure reliability growth during testing as well as staff-execution time could. These additional test time metrics were number of messages received (which measured the number of transactions), number of all tests, number of unique tests, number of tests run for the first time during testing, and calendar time.

All these test time data, as well as test failure data, were collected on a daily basis and were, as a result, in a failures-per-time-interval format. To analyze them with the TBF reliability models of the CASRE tool or with the SRMP tool, we used random numbers to prepare time-between-failures input data for these tools.

5 Pilot project results

The pilot project reported here (as well as other pilot projects) was supposed to answer a number of questions about practical feasibility of the SRE methods for reliability management of actual projects.

5.1 System testing and operational profile

We reviewed the system testing procedures for the pilot project. Though developed before a formally documented operational profile became available, these procedures did emulate the use of the product in the field and were consistent with the operational profile. This was due to the fact that they had been developed under the guidance and supervision of the product's lead tester who was very experienced and had good understanding of how the product was used by the customers.

Statistical sampling procedures were not formally used for selecting tests and test sequences. Rather, tests were executed so that system features and operations were selected "deterministically," proportional to the frequency of their occurrence in the field. Occasionally, a test was performed earlier than it should have been according to the operational profile, however, in order to minimize frequent changes in test setup, so that tests with the same or similar test setup could be run together. Individual *runs* within operations were selected more or less randomly. Repetitions of tests were restricted: limited repetitions were primarily used for regression testing, and to verify and diagnose a test failure. There was very little code churn during testing, as system testers usually had all the code at the start of testing.

5.2 Reliability growth during system test

We knew that there must be a tendency for reliability growth because test failures were being repaired during testing and the testers and developers on this project were experienced and efficient. The issue was whether the SRE methods would be capable of detecting and measuring it, and if so, which automatically collected test time metrics were able to do this as well as the staff-execution time metric.

We were able to detect reliability growth with the SRE methods. Figure 2 shows a plot of the failure rate (intensity) as a function of test time for the pilot project, calculated by CASRE using three reliability models. (The purpose of this plot was just to exhibit reliability growth with test time, and not to analyze fluctuations in real-world data or calculate variance.) Expressing this result in another and more compact and convenient way, we defined a Reliability Growth Factor (RGF) as the ratio of the initial failure rate to the final failure rate, at the start and end of test, respectively (or as the ratio of the final MTTF to the initial MTTF). The larger the value of the RGF, the greater the reliability growth measured by a test time metric (and vice versa). Table 2 shows the RGF values for the different test time metrics for the pilot project. These values were calculated using the reliability model that was identified as the most accurate individually for each time metric.

Table 2 shows high RGF values for all test time metrics except the *number of tests run for the first time* metric. Apparently the *number of tests run for the first time* metric was not a good test time metric for tracking reliability growth for this pilot project (it could, however, be different for some other project with reliability growth, where the *number of tests run for the first time* metric could conceivably show a high RGF value). All the other time metrics show approximately comparable

RGF values, similar to the RGF value for the *staff-execution time* metric. All the test time metrics, with the exception of *number of tests run for the first time*, can therefore be used to track reliability growth for the pilot project.

Somewhat surprising to us was the performance of the *calendar time* metric for the pilot project, as it was able to track significant reliability growth, especially since the *calendar time* metric did not perform well on other software development projects we had investigated. It appears that in the case of the pilot project we are dealing with a mature and stable test process containing a significant amount of automation. Also, the testing effort was expended on a fairly uniform basis throughout the entire test period, as indicated by an analysis of the test data collected. This very likely accounted for the good performance of the *calendar time* metric on the pilot project.

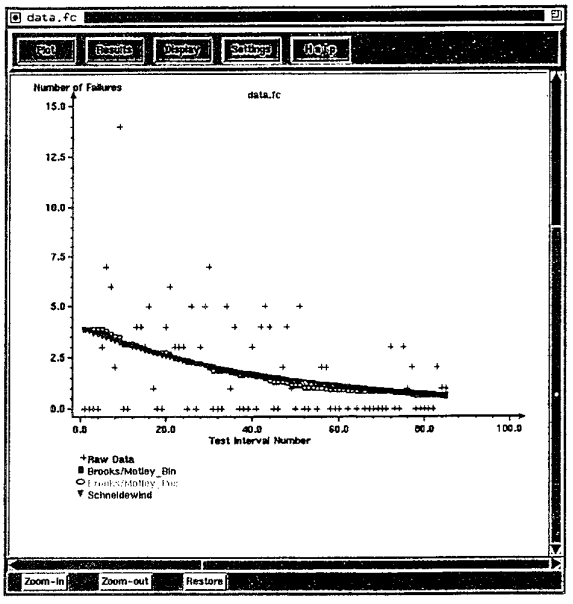

Figure 2: Failure rate as a function of test time for the pilot project, calculated by CASRE

Most (though not all) of our subsequent effort was concentrated on the use of *staff-execution time* and *messages received* as test time metrics. *Staff-execution time* had shown itself to be a good test time metric on all other projects, whereas the other metrics either were not applicable to other projects or did not track reliability growth well. *Messages received* was collected automatically and relatively easily during system testing of the pilot project and was therefore the metric preferred by the test group.

Test Time Metric	RGF
calendar time	6.522
staff-execution time	4.665
number of messages received	5.453
number of all tests	4.348
number of unique tests	5.805
number of tests run for the first time	1.573

Table 2: RGF values for different test time metrics

5.3 Reliability model selection and estimation

We found the ESTM tool capable of providing guidelines on when-to-stop-testing, as well as of exhibiting reliability growth of the product during testing. Figure 3 illustrates the net benefits of testing (explained in section 4.2) as a function of test time measured in terms of the *messages received* metric.. The graph can be used to determine the range of test time when the net benefits are positive, as well as the length of test time when the benefits reach their maximum value. Figure 4 shows cumulative number of faults found as a function of test time, calculated from the reliability model used by ESTM. The model tracks the actual fault data reasonably well. It also has a continuously decreasing slope, which corresponds to a decreasing failure rate during testing.

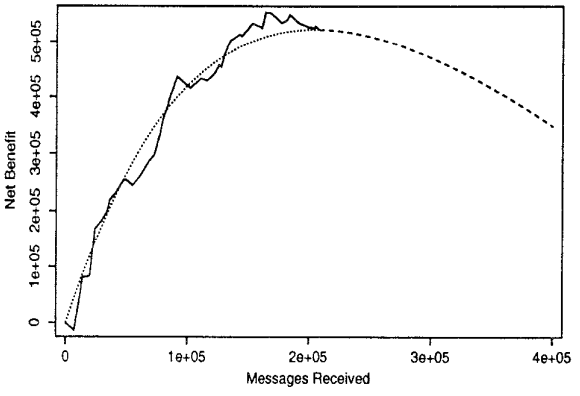

Figure 3: Net benefit of testing as a function of test time (measured in terms of messages received), calculated by ESTM

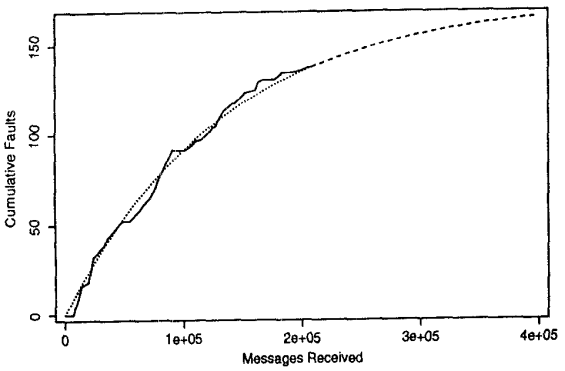

Figure 4: Cumulative number of software faults discovered during testing as a function of test time (measured in terms of messages received), calculated by ESTM

For the other two tools, CASRE and SRMP, the questions posed were: Can either or both tools be readily used to identify the most accurate reliability model for a product release undergoing system testing and calculate the release's reliability? Can this be performed repeatedly at various stages of system testing when enough test data is available to perform the calculations? And can this be performed for various test time metrics? We found this to be the case for both CASRE and SRMP. Although we discuss this here for the pilot project, we found CASRE and SRMP to be equally applicable to other projects as well. For example, Table 3 presents the results for the pilot project, calculated by the SRMP tool at the end of system test, using the *staff-execution time* metric.

Model / Measure	JM	BJM	GO	MO	DU	LM	LV	KL
Prequential Likelihood (-ln PL)	360.826 (8)*	358.849 (4)	360.563 (6)	359.932 (5)	358.660 (3)	360.825 (7)	355.548 (1)	355.898 (2)
U-plot Kolmogorov Statistic	0.1393 (7)*	0.1414 (8)	0.1162 (5)	0.1160 (4)	0.1044 (2)	0.1391 (6)	0.1132 (3)	0.0833 (1)
Bias at significance level:								
0.01	No	No	No	No	No	No	No	No
0.05	No	No	No	No	No	No	No	No
0.10	Yes	Yes	No	No	No	Yes	No	No
0.20	Yes	Yes	No	No	No	Yes	No	No
Y-plot Kolmogorov Statistic	0.1311 (7)*	0.1232 (3)	0.1244 (4)	0.1275 (5)	0.1424 (8)	0.1309 (6)	0.0936 (2)	0.0831 (1)
Trends at significance level:								
0.01	No	No	No	No	No	No	No	No
0.05	No	No	No	No	No	No	No	No
0.10	No	No	No	No	Yes	No	No	No
0.20	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Mean Time To Failure (staff-execution hours)	55.804	N/A**	N/A	46.273	N/A	55.836	46.768	53.410
Median Time To Failure (staff-execution hours)	38.680	38.291	37.427	31.847	26.110	38.702	24.110	21.678
Failure Rate (failures per staff-execution hour)	0.01792	0.01856	0.01867	0.02185	0.02658	0.01791	0.03243	0.03839

* The SR model's rank order for this measure ** Not Applicable (not mathematically defined for this SR model)

Table 3: Summary test reliability results at the end of test for the pilot project, obtained from SRMP computer runs using staff-execution time metric

Table 3 shows the reliability models analyzed (JM, BJM, etc.) in the first row. It then shows, for each model, three statistical measures to be used for assessing the model's predictive accuracy - prequential likelihood (presented as the negative of its natural logarithm) and u-plot and y-plot Kolmogorov-Smirnov statistics. This is followed by three measures of estimated product reliability at the end of system test.

We used all the statistical measures when assessing a model's predictive accuracy. In Table 3, for example, the smaller a model's absolute value of the logarithm of the prequential likelihood the better the model's predictive accuracy. Accordingly, the LV and KL models should be the most accurate (although the KL model's prequential likelihood value is slightly higher, this small difference could be due to data noise and numerical rounding in the calculations and should not be considered significant).

The prequential likelihood values are not the whole story, however. They only indicate which among the models are the relatively "best." They do not show us whether any of the models, even the best one, is sufficiently accurate and what types of inaccuracies it contains, for prequential likelihood measures a model's inaccuracy due to all causes combined. This has to be assessed by considering the other measures of predictive accuracy - the u-plot and y-plot Kolmogorov-Smirnov statistics (goodness-of-fit meaures) in Table 3, which test the presence or absence of bias and unreal trends in each model. It is up to the analyst to decide what statistical risk, measured by the significance level in Table 3, to assume when using the SRMP results for the u-plot and y-plot. The significance level indicates the probability of making the wrong decision: the data indicate a poor fit between the model and the data, suggesting the presence of bias or bias trends, when in reality this is not the case. In Table 3 we use several values for the significance level (0.01, 0.05, 0.10 and 0.20) and indicate the presence of bias or bias trends by a "Yes."

In Table 3, the LV and KL models, which were previously shown to be the relatively best by the prequential likelihood, are now shown as well to have neither bias nor bias trends at the significance levels selected. Considering the measures of predictive accuracy shown in Table 3, the LV and KL models appear to be sufficiently accurate for the reliability calculations on the pilot project. None of the other six models appear to be as accurate: they either show some evidence of bias or bias trends.

Similar results were obtained with SRMP for other test time metrics. For example, using the *number of messages received* time metric, the LV and KL models were the most accurate: they had the best relative accuracies as measured by the prequential likelihood and showed no statistical evidence of bias or bias trends.

Also, the SRMP results were generally consistent with the CASRE results, even though CASRE and SRMP use a number of reliability models that are different between the two tools. For example, Table 4 shows the rank order of the six TBF models used by CASRE for the different test time metrics. The LV model is the "best" overall model. As a result, this model was used for calculating reliability estimates for the pilot project.

Model	GEO	JM	LV	MB	MO	NHPP
calendar time	(5)	(3)	(1)	(2)	(4)	(6)
staff-exec. time	(5)	(3)	(1)	(2)	(4)	(6)
all tests	(5)	(1)	(1)	(3)	(4)	(6)
unique tests	(5)	(1)	(2)	(3)	(4)	(6)
first-time tests	(5)	(4)	(2)	(3)	(1)	(6)
messages rec.	(2)	(3)	(1)	(3)	(5)	(6)
Sum of Rank	27	15	8	16	22	36
Total Rank	(5)	(2)	(1)	(3)	(4)	(6)

Table 4: Overall comparison of CASRE's TBF models

Product Release	Value of TCF	Difference from average value (%)
Release 1	30.140	-2.3
Release 2	31.534	+2.3
Average value	30.837	------

Table 5: Testing compression factor (TCF) values for the pilot project

5.4 Testing compression factor and prediction of field reliability

Only two previous releases of the product were used for the calculation of the *testing compression factor* . Table 5 shows the calculated TCF values. These values are almost identical. There was no evidence from the operational profile that these values should be different or that they are inapplicable to a new release of the product. These TCF values were used to predict an average value of the failure rate of the new Release 3 in the field, and this prediction was later compared with the measured value of the field failure rate when field failure data for Release 3 became available. Table 6 shows the results. We see excellent agreement between the predicted and actual values.

Basis on which Failure Rate of Release 3 is calculated	Value of Field Failure Rate (failures per machine-month)	Difference between prediction and actual value (%)
Prediction based on TCF value for Release 1	0.0731	+3.0
Prediction based on TCF value for Release 2	0.0698	-1.7
Actual value of Field Failure Rate	0.0710	-----

Table 6: Comparison between predicted and actual field failure values for the pilot project

We were at first surprised that the TCF values in Table 5 are so large, which indicates that there are, on average, many more repetitions of the same transaction types during the running of the product in the field than during system test. This is attributable to the highly non-uniform nature of the operational profile for this product. For example, we found substantially lower values of TCF (e.g., in the range 6 to 8) for other products with more uniform operational profiles.

6 Conclusions

The SRE methods discussed in this paper were found to be successful on all the projects to which we applied them. We found through experience that they have to be adapted, to different degrees, to each project and used judiciously. For example, test time and field operation time metrics have to be selected carefully. On the one hand, one would like to use metrics that approximate software execution time as closely as possible in order to be able to measure reliability growth during testing as well as during field operation (if the product is maintained in the field and at least some of the faults are repaired). On the other hand, one has to be realistic and not unduly burden the software testers and especially the customers with additional time-consuming data collection effort, substantially beyond the data collection effort they normally expend when testing and operating the product. By a judicious balance between the two requirements, which has to be performed individually on every project, we have found that it is possible to obtain useful reliability measurements and estimates and to use them for reliability management on a project.

References

[1] IEEE, *Charter and Organization of the IEEE/TCSE Software Reliability Engineering Committee (SREC)*, 1995.

[2] M. R. Lyu (ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill Book Company and IEEE Computer Society Press, New York, New York, 1995.

[3] *American National Standard, Recommended Practice for Software Reliability*, ANSI/AIAA R-013-1992.

[4] J. D. Musa, "Operational Profiles in Software Reliability Engineering," *IEEE Software*, vol.10, no. 2, pp.14-32, March 1993.

[5] S. R. Dalal and C. L. Mallows, "Some Graphical Aids for Deciding When to Stop Testing Software," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 2, pp. 169-175, February 1990.

[6] S. R. Dalal and C. L. Mallows, "When Should One Stop Testing Software?", *Journal American Statistical Association*, vol. 83, pp. 872-879, 1988.

[7] A. L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, vol. R-28, pp.206-211, 1979.

[8] M. R. Lyu and A. Nikora, "CASRE – A Computer Aided Software Reliability Estimation Tool," *CASE 92 Proceedings*, pp. 264-275, Montreal, Canada, July 1992.

[9] M. R. Lyu, A. Nikora, and W. Farr, "A Systematic and Comprehensive Tool for Software Reliability Modeling and Measurement," *Proceedings FTCS-23*, pp. 648-653, Toulouse, France, June 1993.

[10] M. R. Lyu and A. Nikora, "Applying Reliability Models More Effectively," *IEEE Software*, pp. 43-52, July 1992.

[11] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 9, September 1986, pp. 950-967.

[12] W. H. Farr and O. D. Smith, "Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide," *NAVSWC TR 84-373, Revision 2*, December 1991.

[13] W. H. Farr, "A Survey of Software Reliability Modeling and Estimation," *NSWC TR 82-171*, September 1983.

[14] J. D. Musa, A. Iannino, K. Okumoto, *Software Reliability – Measurement, Prediction, Application*, McGraw-Hill Book Company, New York, New York, 1987.

[15] *User's Manual for the Software Reliability Modelling Programs* (SRMP), developed by Reliability and Statistical Consultants, Ltd., 5 Jocelyn Road, Richmond, Surrey TW9 2TJ, United Kingdom.

[16] A. P. Dawid, "Statistical Theory: The Prequential Approach," *J. Royal Statistics Soc. A*, Vol. 147, pp. 278-292.