

Practical Software Reliability Modeling

Dolores R. Wallace
SRS Information Services
Software Assurance Technology Center
NASA Goddard Space Flight Center
Greenbelt, MD 20771
dwallac@pop300.gsfc.nasa.gov

Abstract

NASA is increasingly dependent upon software in systems critical to the success of NASA's mission. The capability to accurately measure the reliability of the software in these systems is essential to ensuring that NASA systems will meet mission requirements. The Software Assurance Technology Center at the NASA Goddard Space Flight Center explored software reliability modeling as a practical measurement technique. This paper describes software reliability modeling, the effort required to use it, and potential improvements to the models to allow for fault correction.

1. Introduction

NASA is increasingly dependent upon systems in which software is a major component. These systems are critical to the success of NASA's mission and must execute successfully for a specified time under specified conditions, that is, they must be reliable. The capability to accurately measure the reliability of the software in these systems is an essential part of ensuring that NASA systems will meet mission requirements. The Software Assurance Technology Center (SATC) at the NASA Goddard Space Flight Center (GSFC) conducted a study that addressed the relationships between hardware and software reliability to identify potential improvements to the software reliability models and to determine how readily the approach could be applied to typical GSFC projects. A paper, "Application and Improvement of Software Reliability Models," describing the complete study is located on the SATC WEB site at <http://satc.gsfc.nasa.gov/support/index.html>.

Software reliability engineering mathematically models the behavior of a software system based on its failures. Predictions such as time-to-next failure, mean time to failure, total number of faults remaining in the system, number of faults remaining at time t are examples of measurements derived from the reliability models and project failure data. These measurements provide an indicator of reliability growth, for example, as the mean time to failure gets larger, then the software is considered to have shown reliability growth.

The equations for the models have parameters estimated from techniques like least squares or maximum likelihood estimation. Then the equation of the models, often containing exponents or logarithms, must be executed. Tedious computationally, mathematical and statistical functions provide the predictions and degrees of confidence for the predictions. Verifying that the selected model is valid for the particular data set may require iteration and study of the model assumptions. While the entire process may be tedious and error-prone when performed manually, software tools reduce the difficulties. It provides managers with a valuable measurement to be used with other reliability metrics in making important decisions such as test or maintenance schedules and product release.

We used a software tool to model GSFC data to determine if this measurement technique is practical. We identify the type of effort needed by the project staff to use this measurement technique successfully. Specific issues that may affect usage of these models include the modeling process, data collection requirements, availability of software tools to support these models, and difficulties of using the models and interpreting their results. We show that a modification to the Schneidewind model to account for the fault correction process may be a useful improvement for software reliability modeling.

2. The modeling process

The current software reliability modeling process is based on features of hardware. While some models compensate for some differences between hardware and software, overall they do not. Schneidewind's model does by discounting the earlier part of the software failure history because of an assumption that after correction, the earlier number of failures does not affect the current failure rate. Some models identifying the mean time to next failure assume that the system has been tested after debugging with no failures observed, but for software latent faults often exist and cause failure in operation. While the models assume that the system is exercised in circumstances very similar to the final operating environment, it is often extremely difficult to test software in its operating environment [1].

The AIAA's *Recommended Practice for Software Reliability* defines a formal procedure of eleven steps for software reliability estimation [2]. The first three pertain to establishing requirements for the system and are outside this discussion. Another step, defining failure, is accomplished by NASA guidelines on defining non-conformances [3]. The remaining 7 steps are specific to software reliability modeling and are discussed in the remaining paragraphs of this section. Some steps may require interaction between project members and the reliability analyst. They include characterizing the operational environment, selecting tests, selecting the models, collecting the data, estimating parameters, validating the models and performing analysis. We examined these steps to determine if software reliability modeling techniques could be practically used at GSFC.

2.1 Characterizing the operational environment

Reliability measurement assigns some reliability measure to a system, such as predicting the number of faults remaining, the number of failures expected in a given time, how much time is needed to find a specified number of faults, or the probability of operating without failure in a specified time. Reliability measures may be determined for system components or for the entire system. Hence, the analyst needs to know the system configuration either to allocate system reliability to component reliabilities or to combine component reliabilities to establish system reliability. The system may evolve with new code or components and these may affect usage of the reliability estimations. Finally, the system operational profile may show how different modes are utilized and may need to have separate reliability measurements for them.

2.2 Test approach

The analyst needs to understand the testing approach such as the integration of components or the approach to correction of faults. Without this knowledge, failure data may be incorrectly associated with wrong components or the assigned time intervals may not represent reality.

2.3 Model selection

Model selection is complex. Criteria for model selection include predictive validity, ease of parameter measurement (e.g., the amount of data should be ~5 times as much as the number of parameters), quality of assumptions, capability, applicability, simplicity, insensitivity to noise (calendar instead of execution time) [2].

Predictive validity addresses the forecast quality of each model and is usually determined by a goodness-of-fit test. Most models have parameters that need to be determined before the equations of the model can be solved. Fortunately the tool used in this study performs these determinations.

On the other hand, matching a project to the assumptions of a model may not be easy. The assumptions need to match the actual project testing and operational environment as closely as possible. These include features such as ability of test inputs to encounter faults, independence of effects of all failures, test coverage, observation of all failures when they occur, or assurance that faults removed on discovery are not counted again. The development and test staffs need to interact with the analyst to determine which assumptions hold for a specific project. This function requires intellectual effort and project knowledge by the analyst prior to the steps performed by software reliability modeling tools.

Capability refers to the ability of a model to estimate other reliability measurements such as the mean-time-to-failure (MTTF) or the confidence intervals for estimated parameters. The tool used in this study provides these measurements according to each model's equations.

Each model may accommodate different development and operational environments and therefore should consider features such as evolving software, failure severity classification, incomplete failure data, multiple installations of the same software, and project environments departing from the model assumptions.

Simplicity refers to three time-consuming and expensive parts of reliability measurement: the data collection process, the modeling concepts, and implementation. Data collection and implementation are addressed in sections 2.4, 3, and 4 of this paper. The simpler the modeling concepts, the more easily an

analyst may understand the assumptions, estimate the parameters, and interpret the results, overall assuring more accurate results from use of these techniques.

A model's results should not be biased by noise, such as extraneous data or incorrect data. For software reliability, the usual noise is the time component. The hardware reliability models are based on a continuous variable of time. For hardware, failure data may be collected in calendar time that may be a continuous variable, especially for wear-out. Calendar time and execution time may be the same. For software, failure data usually are provided in calendar time, which is not usually the same as execution time, especially during testing, and therefore is not necessarily a continuous variable. Adjustments need to be made to get as close as possible to execution time. Calendar time can be especially difficult to adjust if the data collection system records only the date of the failure but not the number of hours (or days) spent testing. The analyst can assign a test interval of a day but must know whether testing occurs on weekends and holidays.

2.4 Data collection

Data collection of any type, in any environment, in any domain, is generally resisted and difficult to achieve due to time and perception that data collection interferes with the "real" work. To overcome resistance, the analyst must clearly state the objectives for the data and request only the data essential to successful use of the models. Data are crucial to the success of software reliability modeling. Section 3 describes features of data collected in a project at GSFC and the transformation of that data for use with software reliability models.

The analyst must remember that if data requests are too intrusive on the project, costs and schedules suffer and project cooperation may go rapidly to zero. The project and the analyst will benefit from discussions about the project's development and test processes and about the system description. The analyst must keep the data collectors motivated, get access to the data quickly and review it promptly.

2.5 Parameter estimation

Three common methods for parameter estimation include the method of moments, least squares, and maximum likelihood estimation. The tool used in this study performs maximum likelihood estimation, the most commonly used approach.

2.6 Model validation

All of the software models supported by the tool used in this study have been used in industry, often the

industry for which the model's designer first exercised the model. As discussed in Section 2.3, model selection is based on the assumptions of the model. The analyst needs to examine the model's assumptions very carefully to assure that the model is valid in the specific domain and even for a new project within a domain. Also, different models may not necessarily produce similar results for the same data. A caution is provided by Littlewood, "Different software reliability models can produce very different answers when called upon to predict future reliability in a reliability growth context. Users need to know which, if any, of the competing predictions are trustworthy. Some techniques are presented which form the basis of a partial solution to this problem. In addition, it is shown that this approach can point the way towards more accurate prediction via models which learn from past behaviour" [4].

Farr cautions us to be careful about a model's assumptions, for some are unforgiving [5]. For example, Schneidewind's model assumes the time intervals over which the failures are observed are equal. In practice, this means establishing equal intervals and including those in which no failures occurred. Other assumptions may be violated, such as the distributional one about the number of failures per unit time, and still credibly fit the data. The Brocklehurst-Littlewood chapter in [6] provides a partial solution to this problem of model validation. It is difficult to characterize programs that fit specific models because programs differ so widely in the problem being solved, development practices, architecture, degree of fault tolerance and other features. The models themselves make fairly crude assumptions about what may be a complex failure process. A way around this situation may be to evaluate each model's predictive accuracy upon each data set that is analyzed, that is, compare a prediction with an actual observation of a program.

Potential solutions to model validity may seem complex, but discussion in Section 4 about an application with a software tool indicates that model validity is manageable. The tool exercises several models, either time-between-failure models or failure count models. The tool provides statistical analyses concerning the accuracy and validity of each model relative to the input data.

2.7 Analysis

Once all the previous steps have been performed, the analyst executes the tool for the selected model(s). The difficult part is that the analyst must study the results (1) to determine if the timeline should be changed and (2) to decide what values to enter to enable predictions. The analyst may have selected several models and may need to determine which is the better fit for the project.

3. Collection of data

At GSFC, some projects use the Distributed Defect Tracking System (DDTS) to collect testing and operational nonconformance data, primarily for managing and tracking non-conformances. In the context of this study, failures are nonconformances that caused the system or component not to perform a required function within specified limits, or the termination of the ability of a functional unit to perform its required function, or caused program operation to depart from program requirements.

For reliability prediction and estimation, it is important for the analyst to ensure data content and validity by reviewing the data when collected to ensure that all necessary data for each fault have been submitted. The analyst needs to keep information about the system and all the activities in developing, testing, and debugging and correcting it to ensure that the data match data each model's assumptions and are organized the data correctly for use with the models.

Several projects have their own implementation of the DDTS, each possibly with slight variations in the data collected. Each is a distributed system with access rights given to many people with different project responsibilities. Data are entered and housed in the resident database for each project's system. In some cases, data from that database may be offloaded elsewhere for other purposes. It is likely that an analyst who may not be directly aligned with the project will work with the offloaded data. The analyst needs to ensure that all necessary supporting information is collected and included in the transferred data.

Beyond content, the analyst must be concerned with organizing the DDTS data to provide the form of data required by the software reliability models, either time-between-failure or failure counts by time interval data. This data reduction activity may involve considerable manipulation and transformation activities.

3.1 Data content

The software reliability modeling data requirements, number of faults in a time interval or the time between failures, appear to be simple. But, when projects collect data, the intent is often to provide information prioritizing corrections and for tracking the fault correction process. The DDTS contains a large amount of data that must be reduced to the simplicity of the required input.

Most likely, for a specific project, data from all testing activities for all components are entered into the system. Specific information that the analyst must be able to extract consists of the activity that found the failure, the date of the failure, and the severity of the

failure. In Section 6, we show that the date the correction began and date the correction was completed are important for improvements to software reliability modeling for the fault correction process. The analyst must be able to define equal time intervals, e.g., 1 day. When the timeline is long and occurs over holidays or weekends, then it is important to know if testing occurred during those times. When there are intervals with no failures, the analyst needs to know if either testing or fault correction occurred. The size or number of test intervals must be adjusted to accommodate intervals with more than the usual staff size. These issues are important in order to get a discrete variable as close to continuous as possible.

Because the defect tracking systems appear to be used across the entire project, the analyst should understand the testing approach concerning integration of components and how they are identified within the tracking system. Data must be sorted by components as they are tested. The analyst needs to discuss with the project staff exactly what data are stored in the DDTS to have valid data for the modeling. If the project does not collect appropriate data, then either modeling cannot be used for that project or the requirements for data collection need to be changed. It would be best for the analyst to look at a project's DDTS frequently to validate that the data have been entered correctly and in a timely manner.

Some initial considerations for modeling purposes include:

- the data should be collected from integration test through operation
- extensive changes should not be routinely made, that is, software cannot be changing so fast that data gathered one day is radically different from another day
- the data should provide an indicator of the type of testing, e.g., component testing, integration testing, or system testing, including names of components.

To emphasize again, the analyst must have information about the validity of the data, including but not limited to its history, other events occurring when the data were collected, how accurate the date submitted is relative to the date actually found and especially the dates of testing.

Some of the models may be used at unit or integration test, but most are used during system testing and operation. Failure data in the form of time between failure or the failure count within equal time intervals are input to software reliability modeling. Each model uses data that meet assumptions about failure rate and intensity and fit a curve implied by the model's mathematics.

3.2 Data manipulation

When data are provided to an analyst in a database or spreadsheet, sorting data into appropriate categories is simpler than when the data is presented in text files. In this case, a text file for each of thousands of data records requires considerable effort to sort out extraneous information and consolidate the necessary data into a manageable spreadsheet or database.

When the analyst has direct access to a project's DDTS or a database derived from it, many of the difficulties may be eliminated because of the ease of sorting and transforming usually provided by database capabilities. The data may be further reduced by some of the project parameters. For example, faults at severity 1-3 may be corrected at a higher priority than those at severity 4 and 5, and so data for the severity 4 and 5 faults can be eliminated. Data may be needed only for certain components and must be sorted out.

The data for an analysis must come from the same testing activity and are extracted from the date of discovery and the number of faults found on the same date. Sorting the data may require considerable manipulation of the DDTS data. The specific tool used in this study requires a text file of either the number of faults found in every time interval or the number of time intervals between faults. Manipulating the data into the appropriate text format may require varying amounts of manual effort depending on how readily the database or spreadsheet produce output accepted by the software reliability modeling tools.

4. Software tool availability

Software support is a valuable resource because the reliability models require several complicated steps engaging mathematical functions using logarithms and exponents and statistical techniques such as maximum likelihood estimation and Chi square goodness of fit. As an example, in Schneidewind's model, the time to next failure(s) at time t is computed by Equation 1.

$$T_{F(t)} = \left[\frac{\log(\alpha / (\alpha - \beta(X_{s,t} + F_t)))}{\beta} \right] - (t - s + 1) \quad (1)$$

F_t is the number of faults for which the prediction Time to next failure(s) $T_{F(t)}$ is made at time t ; α , β are the parametric values computed for the model; s is the starting interval for using observed failure data in parameter estimation for the Schneidewind model. A software tool making invisible these computations and the need to understand them expertly would be valuable to the user.

Recent WEB searches identify basically the same software reliability modeling tools from a survey conducted in the early 1990s [7] and those listed in the AIAA recommended practices [2]. A compact disc (CD) containing several tools is provided with the *Software Reliability Handbook* [6]. One of these is SMERFS, a public domain tool developed by Dr. William Farr of the Naval Surface Warfare Laboratory and employs several of the models. We selected it for use with our experiments.

Table 1. SMERFS^3 software reliability models

Interval Data Models
Brooks and Motley's Binomial Model
Brooks and Motley's Poisson Model
Generalized Poisson Model (2 versions)
Non-homogeneous Poisson Model
Schneidewind's Model (3 treatments)
Yamada's S-Shaped Model
Time Between Failure Models
Geometric Model
Jelinski / Moranda Model
Littlewood and Verrall Linear Model
Littlewood and Verrall Quadratic Model
Musa's Basic Model
Musa's Logarithmic Model
Non-homogeneous Poisson Model

SMERFS has been modernized and is now called SMERFS^3 to indicate the latest version. Table 1 shows the models contained in SMERFS^3. While this version has not been officially released, it is easier to use than earlier versions because of the user interface. Because we had access to the tool and its developer, it became the baseline tool supporting this study.

After examining a project's characteristics, an analyst may reasonably permit SMERFS^3 to select the appropriate models. SMERFS^3 provides other services that free the analyst to worry only about the goodness of the failure data and the interpretation of the results. The tool computes the parameters needed for the various models and checks results for validity and accuracy of each model. It provides confidence intervals for the parameters for the models utilizing them.

The tool allows the user to select a specific model or to have SMERFS^3 select those that are appropriate. All interval data models use the same input and all time-between-failures use the same. The program provides some transformation assistance between the two types of data. It is speedy: exercising several reliability models on a data file of over 300 time intervals took only seconds.

SMERFS^3 is only a tool. It does not interpret results, that is, it will not replace the intellectual ability of the analyst. The analyst must rely on knowledge about the

project and judgment to understand what the results mean relative to the system's reliability. Even though the tool computes all the mathematics, a fair amount of understanding of the underlying mathematics and statistics is needed to interpret the results. The tool does perform curve fitting on the input data and will refuse to exercise the data on inappropriate models. Of the models it does exercise, results may vary widely. The parametric values of the models are provided, but the analyst needs to understand them to decide which output results are meaningful. After an analyst acquires experience, interpretation and judgment may become easier and quicker. Execution time is in seconds but the time to prepare the data may be substantial. The time to interpret results, alter the length of the data set, and repeat the process may be overwhelming. The example in Section 5 provides evidence that most of the difficulties can be overcome with a little experience.

5. Example of software reliability modeling

We describe this experience with software reliability modeling to characterize the advantages and the constraints of using this measurement technique. While the tool reduces manual computations, questions remain about other tasks. How much knowledge of the mathematics would a user need to interpret the results? We describe the steps necessary before applying SMERFS^S and the process of using the tool. We show sample output. Finally we discuss some lessons learned.

5.1 Initial process steps

Of several projects available to us, we discuss only one project. We had a description of the variables in the DDTS and some information about testing schedules. We could not develop a true characterization of the operational environment or the various test activities and their relationships to the pieces of the software system. Given those constraints, we prepared the data for the modeling process.

Of the many data fields of the DDTS, only a few were significant. These were the dates the failures occurred, the activity or phase in which they were found, and their severity level. The severity level mattered because corrections of non-conformances at lower severity levels were deferred and may have reappeared in later testing. The activity or phase was the only information for sorting the failure data by software component or subsystem. Because their names were not always consistent, we asked a project person to sort failures into their proper activities. The date provided the link to

time-between-failure and to the failure count for a time interval. For time intervals of a day, we located calendars to eliminate weekends and holidays.

While it was much simpler to use intervals of a week or a month, we needed to map those time periods properly and to ensure we had enough data for these larger intervals. Regardless of whether the data was in a database or spreadsheet, manipulation and reduction to forms needed by the models took a significant amount of time. The process is error-prone and requires verification. Macros may help to reduce the effort.

5.2 Exercising the models

SMERFS³'s pull-down menus are concise and leave little room for misunderstanding. The user is asked to specify whether the input is time-between-failure or interval data. For interval models, failure counts for every interval, even those without failures, must be provided in a text file with care not to have a blank line at the end of the file. The user may select the models to be executed but usually it is better to let SMERFS³ do the selections with accuracy analyses for the models. If the data are grossly inappropriate for a model, the tool will inform the user. Table 2 shows a sample of input data, for nine intervals of one week. Week 7 had no failures and must be shown. The count of failures is in column 1 and column 2 indicates the data are for software failures.

Table 2. Sample interval data for SMERFS³

1	1.
6	1.
7	1.
3	1.
7	1.
5	1.
0	1.
3	1.
1	1.

5.3 Sample Executions

From several project activities, we chose integration testing for subsystem 1 spanning 26 months, or, 110 weeks, with 130 failures. We chose intervals of one week. We exercised the data at 69 weeks, and 110 weeks to see if predictions made at 69 weeks would indicate the full 130 faults found at 110 weeks. There were 129 faults at 69 weeks. The circles in Figure 1 indicate the observed faults, and the models, beginning at the bottom to the top are Brooks and Motley's binomial, Brooks and Motley's Poisson, Schick-Wolverton Poisson, generalized Poisson weighting 2, non-homogenous Poisson and Schneidewind (treatment 1). The number of

observed failures varied widely in the first weeks, as is anticipated. The number of failures leveled off and reliability growth occurred, that is, reliability was higher.

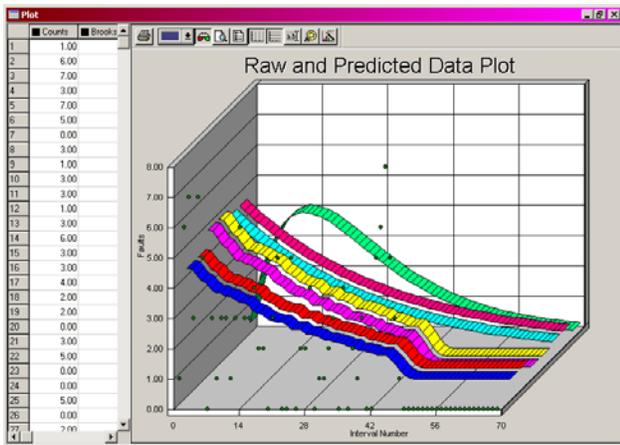


Figure 1. Sample graphs of models

SMERFS³ generates a chart for each model with the estimates of total faults and total faults remaining produced by the models. The prediction section requires interaction with the analyst. The queries vary slightly depending on the model and produce answers such as how many failures to expect within a specific test time or how long it would take to encounter n faults. Figure 2 is the output for Yamada's S-shaped model. Of course if the estimated number of faults is less than 1, then a prediction cannot be made.

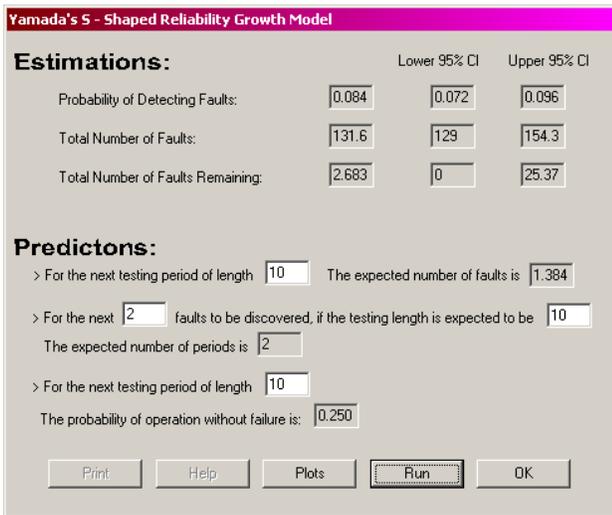


Figure 2. Sample output with predictions

Table 3 shows some possible predictions with a summary of other output for the interval data of 69 weeks, for the models that produce confidence intervals. The models of Table 3, Generalized Poisson –2, non-homogeneous Poisson, and Yamada, produce Chi-square

values to indicate how good the curve fit is to the input data. Using Chi-square fits and confidence intervals helps as does comparing results of same data using one shorter and one longer set of observed values for predictions. The model that produced the prediction closest to the 110 weeks of data is Yamada. The analyst uses all of this information to determine which model to use. Analyzing all this information helps an analyst to avoid selection of a poor model. Ideally the analyst exercises the models for time beyond the time of the observed data to enable predictions far out.

Table 3. Predictions with confidence intervals

	LC I	P	UCI	LCI	TNF	UCI
GP2	.027	.035	.043	137.8	139.3	140.8
NHP P	.022	.032	.042	129	144	171
YAM	.072	.084	.096	129	131.7	154.3
	LC I	TNF R	UCI	CHI 2	Pred	
GP2	8.88	10.34	11.85	127	.362	
NHP P	0	15	42.3	58.6	4.21	
YAM	0	2.68	25.37	117.8	1.38	

P: probability of detecting faults
 Pred: # faults predicted in next 10 test periods
 TNF: total # faults
 TNFR: total # faults remaining
 Chi2: Chi-square value

5.4 Lessons Learned

When we initiated the software reliability modeling process on the first set of data, we had no experience with either SMERFS³ or data from any of the defect tracking systems. To ease difficulty in learning how to use SMERFS³, we recommend that the tool contain a brief tutorial with one set of input data of interval type and one of time between failure type, along with output and interpretation for each. Within a couple hours anyone could understand how this tool can be used for software reliability modeling. We believe that once an analyst has applied the SMERFS³ tool, he will have little or no future difficulty utilizing it.

While each run with SMERFS³ is almost instantaneous, an analyst needs to have reports with the charts and plots to show project staff. While a printout of the models' numeric values for the charts is available,

the predictions are not printed. Saving the output data to a file accessible by other people took some manipulation. As SMERFS³ matures, some of the problems with saving output may disappear. Until then, the lesson is for a user to save each screen output to a WORD file. A macro may ease the difficulty of manipulating screen outputs in a file.

Data issues are somewhat more involved and concern either data collection or data manipulation. Most GSFC projects span several years. The defect tracking systems provide data for monitoring the status of non-conformances. Records for test schedules and staffing levels do not appear in the database but without this information the time between failures may be incorrect. It may also be misleading when there are no failures for consecutive intervals because it is not clear if testing has occurred during those intervals. If not, then the intervals need to be adjusted. Similarly two people conducting tests instead of one person changes the interval size. The lesson is that test schedule information and staffing levels should be required information in the tracking system and should be made available to the analyst.

Data input must be constructed so that the failures are from the same software component; therefore failures should relate to the software component. The “activity” field almost accomplishes this, but could be enhanced with another field naming the involved component(s). The lesson here is to require the software component name field in the tracking system. Also, if some fault corrections are deferred indefinitely, then they should be labeled accordingly. Addressing these lessons should satisfy the data collection concerns.

The analyst needs to manipulate the data considerably to organize it by software type and test activity, then by data and finally by count. These are manual and highly error-prone tasks. Macros to ease the difficulty of manipulating the project data in a spreadsheet may mitigate these concerns.

The remaining step, interpretation of results, is probably the most difficult aspect of software reliability modeling. It also depends on how well the data were interpreted in the first place. Therefore, the first lesson is that the analyst must work carefully with project staff to understand the organization, test schedule, and operational environment of the software. SMERFS³ produces statistical information to aid the analyst in selecting the best model for the data and in evaluating the estimates. Another lesson is that training in understanding the results and applying to decisions concerning the project would remove the final hurdle to making this technology a practical instrument at GSFC for software reliability measurement.

6. Modeling the fault correction process

In general, software reliability models have been based on hardware reliability models and hence have focused on failure data and predicting failure occurrence. They have not given equal priority to modeling the fault correction process. However, there is a need for fault correction prediction, stemming from the fact that the fault correction process is vital to ensuring high quality software. If we only address failure prediction, reliability assessment will be incomplete because it would not reflect the reliability of the software resulting from fault correction. In addition to achieving greater accuracy in reliability prediction, there are by-product benefits associated with fault correction prediction as follows:

- Predicting whether reliability goals have been achieved: If no predictions are made of the number of faults to be corrected, fault correction rate, and fault correction time, accurate prediction of reliability cannot be obtained.
- Providing stopping rules for testing as follows: (1) The predicted number of remaining faults is less than or equal to a specified critical value and (2) The fault correction rate asymptotically approaches zero.
- Prioritizing tests and allocating test resources: Software with high values of number of remaining faults and low fault correction rates are given high priority in testing and allocation of resources, such as personnel and computer time.

Dr. Norman Schneidewind developed a fault correction modification for SATC and applied it initially to his model [8]. The modification assumes that the number of corrected faults has the same general form as the number of detected faults but with a variable delay dT , that is, Equations 2 and 3 are similar for $D(T)$ and $C(T)$, the predicted number of faults detected at time T , and the number of faults corrected at time T , respectively:

$$D(T) = (\alpha/\beta)[1 - \exp(-\beta((T-s+1)))] + X_{s-1} \quad (2)$$

$$C(T) = (\alpha/\beta)[1 - \exp(-\beta((T-s+1) - dT))] + C_{s-1} \quad (3)$$

The parameters α and β are computed from the maximum likelihood function in SMERFS³, and s , also computed from SMERFS³, is the starting interval for using observed failure data in parameter estimation. X_{s-1} and C_{s-1} are the observed fault count and fault correction count in the range $[1, s-1]$, respectively. Using these equations and the cumulative probability distribution, the new model yields new measures for the number of remaining faults, the time required to correct C faults, the proportion of faults corrected, the fault correction

rate, and other prediction capabilities such as whether reliability goals have been achieved. Data from the space shuttle and the project already described in this paper, but for 35 weeks, were used to validate the model. The derivation, equations and validation are provided in [8].

From a practical standpoint, the question is “How can one utilize these new equations?” Dr. Schneidewind has provided SATC with a spreadsheet to compute these new measures. The values of α , β and s are computed from SMERFS³ and are entered into the spreadsheet, along with the interval failure data. The analyst must exercise care in entering the data but can also make and control predictions directly within a worksheet. These can readily be delivered to the project member requesting the predictions. By using a spreadsheet, instead of a software tool in executable form, the analyst can enter new equations or models. Not least, with control of the output, the analyst can choose how to present the results in graphs readily produced by the spreadsheet.

7. Conclusions

Software reliability modeling is one of many methods to aid in measuring software reliability. Others may include inspections, testing, change control boards, metrics gathered from these and other methods. In a paper addressing software reliability modeling for Shuttle software, Schneidewind suggests that perhaps experience and judgment are most important for understanding software reliability measurements [9].

With careful application of this technique, project managers have a valuable tool to assist them in making major decisions. These include but are not limited to establishing or altering test schedules for integration and system testing, for staff allocation, for product release, including release of some components for the next stage of integration, and for maintenance scheduling of released products.

This study has shown that software tools can perform most of the mathematical and statistical functions of software reliability modeling. Analysts can perform the technique within reasonable amount of time and without a deep understanding of the mathematics. The two major issues are data reduction and interpretation of results. The first of these can be simplified by macros for either a data base or spreadsheet used in preparing the data for SMERFS³. The latter can be enabled with experience applying the technique.

The SATC can enable a project to gain access to this critical measurement technology in a number of ways: 1) by training its engineers, 2) through teaming and partnering on a phase-driven basis, or 3) by serving as independent analysts, i.e., assessing the data and

formally reporting results and conclusions. The choice of option remains a project manager's prerogative; the critical issue is to achieve understanding and reduce the project's risk.

Finally, the new Schneidewind model adjusts for fault correction, which the earlier models do not. Applying this technique requires the use of both a software reliability modeling tool to get parametric values and a spreadsheet for the actual correction model. This appears to be a promising approach to software reliability measurement. More experience applying this model to actual project data is needed to judge its value.

Overall, this study has shown that software reliability modeling has a valuable role in software measurement and that the technique can be applied in a practical manner.

8. References

- [1] D. Hamlet, “Are we testing for true reliability?” *IEEE Software*, Vol. 9, No. 4, July 1992, PP. 21-27.
- [2] American Institute of Aeronautics and Astronautics, *Recommended Practice for Software Reliability*, ANSI/AIAA R-013-1992, February 1993.
- [3] Corrective and Preventive Action, Goddard Procedures and Guidelines, GPG 1710.1E, November, 1999.
- [4] B. Littewood, A.A. Abdel Ghaly, and P.Y Chan,, “Tools for the Analysis of the Accuracy of Software Reliability Predictions,” *Software System Design Methods*, Edited by J. K. Skwirzynski, NATO ASI Series, Vol. F22, Springer-Verlag, 1986, PP.299-333.
- [5] W. B. Farr, “Software Reliability Modeling Survey”, *Handbook of Software Reliability Engineering*, Michael R. Lyu (Ed), IEEE Computer Society Press, McGraw Hill, 1996.
- [6] M.R. Lyu., Editor, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, McGraw Hill, 1996.
- [7] G. E. Stark, “A Survey of Software Reliability Measurement Tools,” Proceedings of the 1991 International Symposium on Software Reliability Engineering, IEEE Computer Society, pp. 90-97, 1991.
- [8] N. F. Schneidewind, "Modeling the Fault Correction Process," Proceedings of the International Symposium on Software Reliability Engineering, IEEE Computer Society, November, 2001.
- [9] N. F. Schneidewind, “Reliability Modeling for Safety Critical Software,” *IEEE Transactions on Reliability*, Vol. 46, No. 1, March 1997, PP. 88-98.