ROBUST: A Next Generation Software Reliability Engineering Tool*

Naixin Li **Yashwant K. Malaiya**

Computer Science Department
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

Abstract

In this paper we propose an approach for linking the isolated research results together and describe a tool supporting the incorporation of the various existing modeling techniques. The new tool named ROBUST is based on recent research results validated using actual data. It employs knowledge of static software complexity metrics, dynamic failure data and test coverages for software reliability estimation and prediction at different software development phases.

1 Introduction

Software reliability modeling includes static and dynamic approaches. In the static approach, software development process, software categories, software complexity metrics, software engineer's experience, etc. are related to the initial program quality in terms of defect density or error rate. A variety of static models have been proposed by researchers [3, 6]. The static approach has limited accuracy but it can give an early estimate of the software quality and supply valuable information for the project managers.

Software reliability growth models (SRGMs) are used in dynamic approach. Dozens of SRGMs have been proposed [4]. Tools are available to assist reliability engineers in using the SRGMs and to select among different models based on models' goodness of fit [17]. SRGMs can be used only after some testing has been done and some failure data are available. It can give better accuracy than the static approach in reliability estimation and projection. Several techniques has been suggested to improve the predictive quality of SRGMs. Brochlehurst et al have proposed using recalibration to reduce the bias commonly observed when SRGMs are used to make projection [2]. Lyu and Nikora have suggested combining pessimistic and optimistic models for software reliability prediction so that some bias can be cancelled out [11]. Li and Malaiya have studied the effectiveness of techniques for enhancing the predictive quality of SRGMs using a large number of data sets [8].

During early testing phase, failure data is dominated by noise which makes prediction based on SRGMs quite unstable. Static software reliability models can play a role here. Software defect density or error rate can be estimated from static software complexity and process metrics such as average programmers' skill, frequency of specification change and amount of design documentation [18]. The initial number of defects present in a program can be estimated using defect density and program size. The initial number of defects thus estimated may be used as an estimate of one parameter of the finite software reliability growth models. The initial defect density can be used for estimating the initial fault exposure ratio, which can be used to estimate the other parameter of the Exponential model [9].

Traditional SRGMs treats all testing effort equally with no discrimination against differences in test input selection. But in practice, it is observed that the effectiveness of test inputs in revealing defects (failures) can differ significantly. To account for the varying effectiveness of testing effort, coverage based modeling is becoming an active topic recently [13].

Existing software reliability tools are designed only to aid the use of SRGMs [17]. To the best of our knowledge, no tools exist yet that include most recent development of software reliability engineering methods. With increasing availability of data, from both the field and experimental studies, we can now

*This research was supported by a BMDO funded project monitored by ONR

1071-9458/95 $4.00 ©1995 IEEE

attempt to develop engineering methods with sound and validated analytical characterization. ROBUST is a new tool being designed from the beginning to have the following features.

- It uses the familiar Windows/Excel interface, thus allows easy learning. as well as customization of the tool. ROBUST is designed as a system containing well defined modules. These modules can be independently upgraded or replaced. This makes ROBUST an open system.
- It provides a technique to estimate initial defect density based on static metrics.
- In the early test phase, the failure data is dominated by noise which can cause unstable projections. ROBUST provides a choice of stabilization techniques.
- It supports techniques to improve the predictive quality of SRGMs that were validated using extensive actual data.
- It can estimate reliability using either time-based SRGMs or coverage based models.

The rest of this paper is organized as follows. Section 2 presents the technical bases for this tool. The overall structure of ROBUST is described in Section 3. The user interface of ROBUST is illustrated in the next section. Section 5 concludes the paper with a summary of ROBUST.

2 Technical Foundations

As we mentioned in the introduction, the key feature of this paper is to integrate recent research results that were validated and presented separately. A number of recent studies form the basis for this paper. We have attempted to combine them in a careful and systematic way to obtain the specifications for this tool.

2.1 A New Static Model

It is well known that many factors contribute to defects in a program. For instance, it is related to human behavior, professional experience, design methodology, quality control, tool support, interaction with target users and programmers' understanding of both software development and target field. Maturity of software technology (languages, tools, reuse, etc.) may also have a general effect on defect density. Experience in developing similar systems may reduce the defect density. Inherent target system complexity also effects on the final defect density. In the past, researchers have examined this problem from different perspectives.

Takahashi and Kamayachi [18] noticed that programmer's skill, frequency of program specification change, and volume of program design document are significant factors with respect to error rate in a program. A model was proposed taking those factors into consideration for predicting the error rate in a program. Examination of the model and actual parameter values would reveal that programmer's skill is the most significant factor and the other factors only slightly affect the error rate.

Software complexity also affects the software reliability. Many complexity measures have been proposed in the past. Crawford, et al found that no other single complexity metric performs significantly better than the number of lines of code (LOC) in predicting the number of faults in a C program [3]. Most of the existing complexity measures are highly correlated and some are simply linear combinations of others. Munson and Khoshgoftaar used factor analysis to reduce 16 complexity metrics into 5 relatively independent complexity domains and map those domains into a single relative complexity metric [15].

Gaffney and Pietrolewicz proposed a process phase based model for early error prediction in the software development process [5]. They suggested that the error detection profile with respect to phases can be described by the Rayleigh model. Although the phase based model is not quite accurate, it allows projections of errors found in later phases and operational uses even before the code is executed. Motivated by their work, phase dependency is taken into consideration in our implementation.

The RADC model [1] estimates defect density using multiplicative factors including application type, development and software characteristics. Our static model takes a similar format, i.e. the total number of defects is estimated by multiplying several factors. It is observed that 50-60% of variation in the number of faults is accounted for by software size alone [16]. We choose software size (KLOC) as one of the multipliers. Other factors affect the defect density which is measured in terms of the number of defects per thousand lines of code. Equation 1 describes the defect density model:

$$D = C * F_c \times F_p \times F_e \times F_m \quad (1)$$

where C is a constant. It can be viewed as the defect density for the "average case" which occurs when the values of F_c, F_p, F_e, and F_m are equal to the default value of 1. Based on existent data, C is chosen to be

20 [16, 18]. F_c depends on the overall software complexity, which can be a measure similar to Munson and Khoshgoftaar's relative complexity metric. The value of F_p reflects the phase at which the estimate is made. F_e measures the average of the programmers' experience. It is noticed that the maturity of software development process including technology, programming language, tool support, amount of reuse, process maturity, quality control and so on, has a significant effect on defect density [7]. The term F_m is used to characterize the maturity level of the software development process. A preliminary version of the model is implemented in the ROBUST tool. Modeling the factors F_c, F_p, F_e, and F_m as a function of related measures is discussed in [10].

2.2 Early estimation and prediction

Static models can be used to estimate the initial defect density or initial total number of defects in a program. They do not tell about reliability growth during testing. The parameters of SRGMs, such as the exponential model, have specific meaning associated with them. Their value may be estimated from static measures. For example, the exponential model as described by the equation below,

$$\mu(t) = \beta_0 \times (1 - exp(-\beta_1 \times t)) \tag{2}$$

has two parameters β_0 and β_1. β_0 is the expected total number of defects initially present in a program. As we mentioned above, it can be estimated using static models. β_1 is the per fault hazard rate which can be estimated using the following equation:

$$\beta_1 = \frac{K}{T_L} = \frac{K \times r}{I_s Q} \tag{3}$$

where T_L is the linear execution time of the program. I_s is the number of source lines of code; Q_r is the average object instructions per source statement; r is the testing CPU instruction rate. K is the fault exposure ratio [16] and reflects the average effectiveness of testing effort. Li and Malaiya modelled the relationship between defect density and fault exposure ratio [9]. Their model, as given below, was validated using empirical data.

$$K = \frac{b_0}{D} e^{b_1 D} \tag{4}$$

Since the defect density can be estimated statically, and we know that K varies only in a small range as described by the above model, we can plug the values of b_0 and b_1 from earlier projects into the model to estimate the value of K. β_1 can then be estimated using static parameters and Equation 3. Thus, we obtain a software reliability growth model from static metrics. Such empirical estimation can be used to make early projections about software reliability growth.

In early software testing phase when only a few data points are available, noise can dominate information and the failure data might not show the reliability growth. SRGM parameters evaluated using such data can be unstable and quite inaccurate. The above technique for obtaining dynamic model from static metrics provides a way to stabilize the initial parameter values. Suppose β_0^s and β_1^s are evaluated from static metrics, β_0^d and β_1^d are estimated from early software failure data, then the final parameters can be computed using:

$$\beta_0 = k \times \beta_0^s + (1 - k) \times \beta_0^d \tag{5}$$

$$\beta_1 = k \times \beta_1^s + (1 - k) \times \beta_1^d \tag{6}$$

where k is a weighing factor, which can be adjusted at user's discrimination and/or based on the noise level in the data.

Stabilizing techniques are model dependent in general. The above shows the method for the exponential model. For the logarithmic model, the two parameters can be approximated using the parameters from exponential model and Equations 7 and 8 [14]. This may offset the optimism typically associated with finite SRGMs.

$$\beta_0^L = \frac{1}{a} \beta_0^E \tag{7}$$

$$\beta_1^L = a \beta_1^E \tag{8}$$

where the value of a is in the vicinity of 5.

Another possible way to stabilizing logarithmic model parameters is given by Malaiya [12]: $\lambda_m = \beta_0 \times \beta_1$ is the maximum failure intensity which may be assumed constant for the initial short period of testing and can be estimated simply by taking the average failure intensity value from the initial failure data. β_1 is estimated by:

$$\beta_1 = \frac{\lambda_m}{\beta_0} = \frac{a \lambda_m}{DS} \tag{9}$$

where D is initial defect density; S is the size of the code; the value of a is in the vicinity of 5.

2.3 Enhancing predictive accuracy

Despite the existence of many SRGMs, accurate prediction of software reliability using failure data is not so easy. First software failure data needs to be

carefully collected. The predictive accuracy of SRGMs can only be as good as the failure data itself. Secondly, projections of different SRGMs do not often agree with each other. Different models present different reliability prediction quality for different projects, although it was observed that some models, such as the logarithmic model, generally performs better than others. Finally the process of applying SRGMs to software failure data has also an important role in predictability.

We empirically studied the effect of different enhancing techniques on predictive accuracy based on over 20 well collected real project data. It was observed that proper selection of the enhancing techniques, such as grouping and recalibration, can significantly enhance the predictive quality [8]. ROBUST supports noise reduction as well as recalibration.

2.4 Coverage based reliability modeling

Traditional SRGMs use testing effort as the only drive for reliability and neglect the difference in testing effectiveness. That may be improved by taking test coverages into consideration. Malaiya, et al [13] modeled various test coverages as a logarithmic function of the number of test cases. The model can be applied to code based coverages like *block coverage*, *branch coverage*, *p-use coverage*, etc. as well as defect coverage. A model that relates defect coverage to test coverage is proposed as given below:

$$C^0 = a_0^i \ln(1 + a_1^i(exp(a_2^i C^i - 1)) \qquad (10)$$

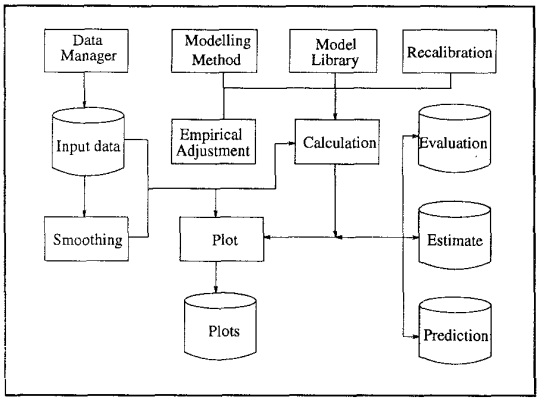
where C_0 is the defect coverage, C^i is some kind of test coverage such as block coverage. a_0^i, a_1^i, and a_2^i is the model parameters. ROBUST includes this model in its library so that test coverage data, if available, can be used to improve the reliability prediction.

3 The Structure of ROBUST

ROBUST has been developed using Microsoft Visual Basic for Applications that comes with Microsoft Excel. It provides a user-friendly interface. ROBUST is an event-driven program consisting of modules in the following categories: data editing, file management, model library, enhancing techniques, estimate, prediction, plotting, etc. Figure 1 depicts the main modules and logical structure of the ROBUST tool.

- *Data Manager:* This part of modules allow users to enter new data in the provided format, retrieve and/or edit existing data sets. The data can be static metrics, dynamic software failure data or test coverages. Data sets are organized into files. Existing data in text format can be imported to ROBUST for reliability evaluation or prediction.

Figure 1: Overall Structure of ROBUST



- *Smoothing:* Proper noise filtering through data grouping or windowing can significantly improve the predictability of SRGMs [8]. ROBUST supports noise smoothing by allowing user to choose from fixed size grouping, lump grouping or windowing. As Li and Malaiya have shown in [8], proper grouping can reduce the noise yet preserve the general trend, while excessive grouping can result in loss of information present in a data. By default, ROBUST uses the optimal grouping guidelines which were obtained using a large set of real data. Users can also specify the degree of grouping for a particular data.
- *Plot:* Plots provide graphic presentation of software failure data. Various plots can be made to show the reliability growth trend.
- *Modeling Methods:* ROBUST supports static modeling, time based modeling, and coverage-based modeling. For static modeling, it uses static software metrics to estimate the defect density, fault exposure ratio, and parameters of the exponential SRGMs. For SRGMs, ROBUST use failure data to evaluate parameters for user selected models and use the fitted model to estimate or predict the program's reliability. If test coverage data is available, ROBUST can measure the program's reliability use coverage based reliability model.
- *Model Library:* ROBUST uses the combined model introduced earlier to estimate defect density. Li and Malaiya's method is used to estimate fault exposure ratio [9]. The logarithmic model, exponential model, and delayed-S shaped

model are implemented as SRGMs [16]. Malaiya, et al's coverage based reliability model is adopted for coverage modeling. Inclusion of other models in each category is relatively easy with the current platform.

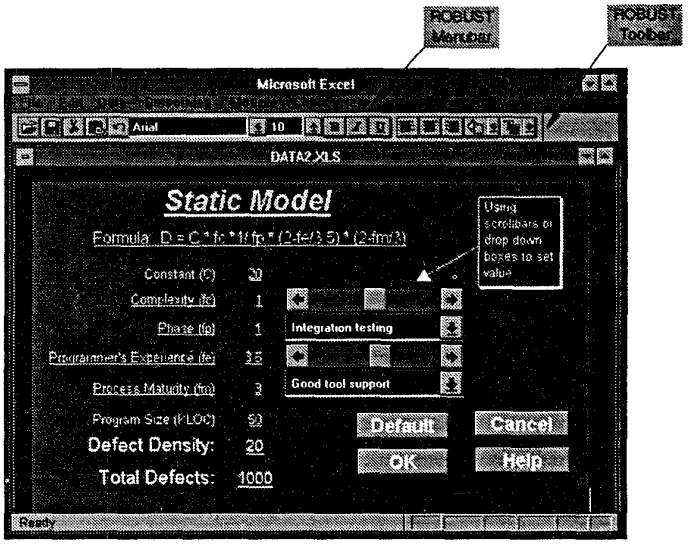
- *Recalibration:* ROBUST uses recalibration to counteract the tendency of overestimation or underestimation of SRGMs.
- *Stabilization:* In early testing phase, software failure data is dominated by noise. Reliability projection based on such data can be quite inaccurate and unstable. ROBUST uses parameter estimation from static metrics to stabilize the parameter values of SRGMs.
- *Calculation:* This part of the tool will interact with users and invoke other corresponding functional modules of ROBUST to make estimation or prediction of software reliability.

4 User Interface

ROBUST is implemented as an ADD-IN application for Microsoft Excel. The menu "Robust" appears in the Excel menu bar when ROBUST is installed. To run ROBUST, select "Robust|Run" from Excel menu bar. ROBUST menu bar will replace Excel menu bar and a customized toolbar replaces any previous Excel toolbars. ROBUST menu bar includes such menus as *File, Edit, Data, Smoothing, Models, Chart,* and *Help*. The toolbar provides a convenient way for user to invoke frequently used functions.

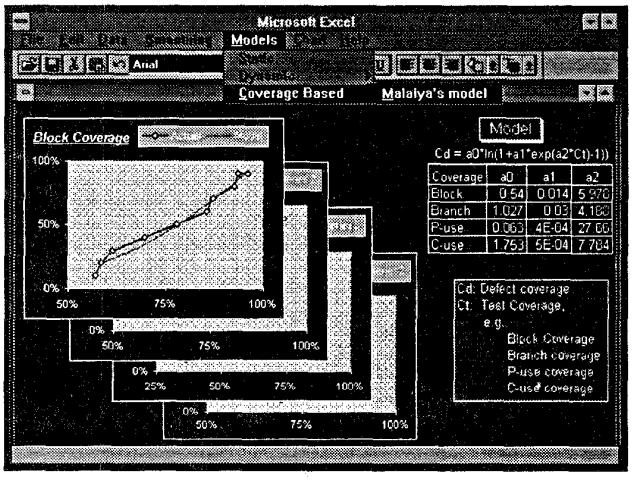
- *File menu:* When a new file is opened, ROBUST provides a template workbook for the user to work with. The template workbook consists of a few forms to be filled by users with static software metrics and dynamic failure data. Users can *import* existing failure data stored as text file using "File|Import".
- *Edit menu:* The Edit menu has common menu items such as Undo, Cut, Copy, Paste for editing. However, it is more convenient to use the toolbar buttons.
- *Data menu:* The Data menu lets users to choose among static metrics, failure data, and coverage data. For failure data, user can enter either failure interval or failure times. Figure 2 shows the screen when static data are selected. To estimate the defect density, users enter data using the scrollbars and drop down boxes. Default values are supplied where applicable.

Figure 2: Static Data and Model



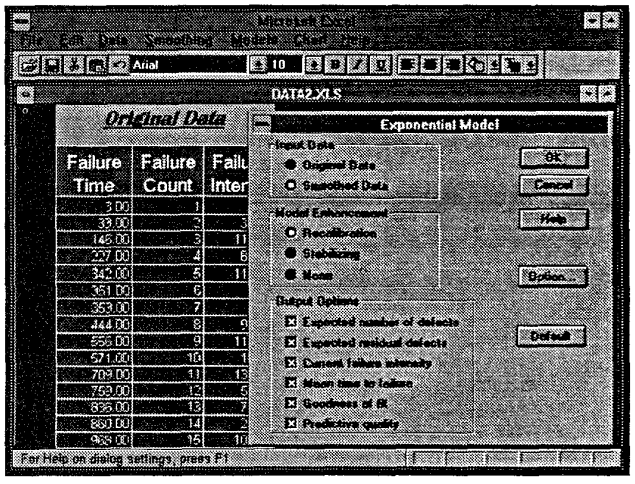
- *Smoothing menu:* Three data smoothing methods are supported: *Fixed Size Grouping, Lump Grouping, Windowing.* For fixed size grouping, an optimal degree of grouping which is data dependent[8] is supplied as the default. Users can change the preset degree of grouping.
- *Models menu:* ROBUST supports the static model as discussed in section 2, SRGMs including the Exponential model, the Logarithmic model, and the Delayed-S Shaped model, and Malaiya's coverage based model [13]. Figure 3 shows the screen for coverage based modeling. Figure 4

Figure 3: Coverage Based Reliability Modeling



show the scenario when a user selects the Exponential model from "Models|Dynamic" and a dialog box pops up letting users to choose the input data (original or smoothed), model enhancement (use recalibration, stabilizing or none), output options and so on.

Figure 4: Dialog Box for the Exponential Model



- *Chart menu:* This menu plots the software failure data to visually present the software reliability growth trend.

5 Concluding Remarks

We present a new integrated method and a new tool for software reliability evaluation and projection. Unlike existing software reliability tools, ROBUST is not just a tool for SRGMs. It embraces recent development of software reliability engineering methods that are not available in current reliability tools. It enables estimate and prediction of a program's reliability even before test begins. In early test phase, due to low information-to-noise ratio, measurement of reliability using SRGMs can be quite inaccurate and unstable, ROBUST stabilize reliability prediction using static metrics and empirical data. During middle and later testing phase, ROBUST's noise smoothing function can significantly improve the predictive quality of SRGMs. When test coverage data are available, ROBUST can use such data in reliability evaluation and prediction.

References

[1] J. R. Adams. Software reliability predictions are practical today. In *7th Annual Software Reliability Symposium*, Denver, Colorado, May 1989.

[2] S. Brocklehurst, P. Chan, B. Littlewood, and J. Snell. Recalibrating software reliability models. *IEEE Trans. Software Engineering*, 16:456–470, April 1990.

[3] S. Crawford, A. McIntosh, and D. Pregibon. An analysis of static metrics and faults in c software. *The journal of systems and software*, 5:37–48, 1985.

[4] W. H. Farr. A survey of software reliability modelling and estimation. TR 82-171, Naval Surface Weapons Center, Sept. 1983.

[5] J. Gaffney and J. Pietrolewicz. An automated model for early error prediction in the software development process. In *8th Annual Software Reliability Symposium*, Denver, Colorado, June 1990.

[6] S. Henry and C. Selig. Predicting source-code complexity at the design stage. *IEEE software*, pages 36–44, Mar. 1990.

[7] K. C. Keene and S. J. Keene. Concurrent engineering aspects of software development. In *International Symposium on Software Reliability Engineering*, pages 51–62, 1992.

[8] N. Li and Y. K. Malaiya. Enhancing accuracy of software reliability prediction. In *4th International Symposium on Software Reliability Engineering*, pages 71–79, Devner, Nov. 1993.

[9] N. Li and Y. K. Malaiya. Fault exposure ratio and software reliability. In *3rd Workshop on Issues in Software Reliability and Testing*, pages 6.3.1–6.3.18, Boulder, Colorado, Oct. 1993.

[10] N. Li and Y. K. Malaiya. A new empirical model for software defect density. Technical report, Computer Science Department, Colorado State University, 1995.

[11] M. R. Lyu and A. Nikora. Applying reliability models more effectively. *IEEE Software*, 9:43–52, July 1992.

[12] Y. K. Malaiya. Early characterization of the defect removal process. In *9th Annual Software Reliability Symposium*, Denver, Colorado, May 1991.

[13] Y. K. Malaiya, N. Li, J. Bieman, R. Karcich, and B. Skibble. The relationship between test coverage and reliability. In *5th International Symposium on Software Reliability Engineering*, Monterey, California, Nov. 1994.

[14] Y. K. Malaiya, S. Sur, N. Karunanithi, and Y. Sun. Implementation considerations for software reliability. In *8th Annual Software Reliability Symposium*, Denver, Colorado, June 1990.

[15] J. C. Munson and T. M. Khoshgoftaar. Measuring dynamic program complexity. *IEEE Software*, 9:48–55, Nov. 1992. Relative complexity combines the features of many complexity metrics to predict performance and reliability.

[16] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability - Measurement, Prediction, Applications*. McGraw-Hill, 1987.

[17] G. E. Stark. A survey of software reliability measurement tools. In *International Symposium of Software Reliability Engineering*, pages 90–97, Austin, Texas, May 1991.

[18] M. Takahashi and Y. Kamayachi. An empirical study of a model for program error prediction. *IEEE Trans. Software Engineering*, pages 82–86, Jan. 1989.