D5-61 P.25

N87 - 24910

A Study on Fault Prediction and Reliability Assessment in the SEL Environment

Victor R. Basili
Debabrata Patnaik

Department of Computer Science
University of Maryland
at College Park 20742
(301) 454-2002

Abstract

This paper presents an empirical study on estimation and prediction of faults, prediction of fault detection and correction effort, and reliability assessment in the Software Engineering Laboratory environment (SEL).

Fault estimation using empirical relationships and fault prediction using curve fitting method are investigated. Relationships between debugging efforts (fault detection and correction effort) in different test phases are provided, in order to make an early estimate of future debugging effort.

This study concludes with the fault analysis, application of a reliability model and analysis of a normalized metric for reliability assessment and reliability monitoring during development of software.

Keywords: Fault Estimation, Fault Prediction, Debugging, Reliability Assessment and Fault Analysis

This study is funded by NASA grant NSG-5123.

1. Introduction

The software development process is an expensive and complicated process. Software developers spend a considerable amount of time in monitoring the cost and quality of the software with two basic goals, to keep the cost down and quality high. Managers often report difficulty in delivering the software for a certain deadline because of uncertainity about its performance. It has been observed that it is much easier and cheaper to correct a flaw at an early stage than in a later stage [Fischer and Walker 1979]. Therefore, many researchers are stressing requirement level verification to eliminate the bugs that may result from ambiguous or incorrect requirement specifications. There are many tools and techniques available to aid programmers and managers during the development phases. Various testing techniques are employed to remove defects from the software being developed. It has been realized that these tools and techniques are not enough to ensure the quality and reliability of the software. Various software metrics developed in recent years help monitor the software development throughout its life cycle.

One of the main problems that managers encounter during the testing phase is the need to predict the remaining faults in the software so that they can plan to staff and test accordingly. There is no simple solution to this problem. In recent years several reliability models have emerged and more models are yet to appear [Transactions on Reliability, August 1979]. It is very difficult to apply any model in different environments. In some cases, practitioners find it difficult and very expensive to collect data in order to use a model. What they need is a simple but robust model, which can be applied easily and inexpensively to predict and assess the reliability of a software project [Iannino et. al 1984].

This paper addresses several aspects regarding fault prediction and reliability assessment. The objectives of this study were to

evaluate and compare fault estimation and fault prediction models

analyze the fault detection and fault correction effort data at different phases of the software life cycle

investigate the possibility of measuring and monitoring software reliability

The data used in this study were collected from a set of software projects developed at NASA/Goddard by Computer Sciences Corporation. These data were collected by the Software Engineering Laboratory (SEL) using forms and techniques described by [Basili, Zelkowitz, McGarry et al 1977]. These projects were written in FORTRAN and were developed for IBM mainframes running OS/VS and MVS/TSO. The sizes of these projects range from 15 to 168 thousand lines of code. Most of these projects are scientific and predominantly ground support software for satellites. These software projects were developed in a single environment using a common programming pool. In the SEL environment each software life cycle consists of six different phases:

Requirement Specification
Design
Coding and Unit Testing
Integration and System Testing
Acceptance Testing
Operation

Fault data collection starts from the middle of the Unit test phase except for a few exceptions where the data collection starts towards the end of the Unit test phase. The data of interest in this study are described below:

a. The total number of faults (F) is the total number of faults observed during all three test phases.

b. The size of the software is the total number of lines in the software.

c. The faults per KLOC is the number of faults observed per one thousand lines of code.

d. The number of components changed per fault is the number of software components needed to be changed in order to correct a fault.

e. The fault detection effort is the effort (in hours) needed to detect the cause of a fault.

f. The fault correction effort is the effort (in hours) needed to implement the changes required to correct a fault.

g. The sources of faults describe the source of each fault.

One of the objectives of this study is to analyze and compare the fault data across projects. Since duration of the test phases for each project are not the same, each phase was divided into some fractions to facilitate comparisons. The next three sections describe the results of this empirical study.

2. Fault Estimation and Prediction

Fault estimation and prediction at an early phase is important for software developers. In the first case, empirical formulas based on relationships between the total number of faults and other variables are used. In the second case, a curve is fitted to a part of fault data already collected during early testing phases. These two methods are investigated and the results are discussed in 2.1 and 2.2.

2.1 Fault Estimation

There are many empirical formulas which can be used for this purpose. The objective of this section is to find such relationships in the SEL environment. The data used in this study are given in Table 2.1.

PROJECT	FAULTS	SIZE	$SIZE^{4/3}$
1.DEA	219	66848	2713009
2.DEB	270	68370	2795680
3.SMM	291	98370	4540987
4.ERBS	589	167775	9253435
5.AADS	83	15548	388060
6.AADSIM	61	27443	827768
7.AEM	181	50601	1871596
8.AODS	94	30823	966421
9.DARES	136	25662	756927
10.DEDET	138	17121	441271
11.GLI	42	26923	806921
12.ISEEC	118	75145	3171028
13.MAGSAT	306	89181	3984402

Table 2.1

Fault estimation based on lines of code is useful, since an estimate of the total lines of code can be available at the beginning of the system test (integration test) phase. Based on Halstead's Software Science the total fault content of a software module is considered to be proportional to $Size^{4/3}$. In the following table, the estimating formula for the SEL environment is compared with the same for Akiyama and Lipow [Gaffney84]. Figure 1 displays the linear fit of the the same formula listed in Table 2.2 for the SEL within 95% confidence interval.

Data Source	Language	Formula
Lipow	Jovial	Fault $= 4 + 0.0014 * Size^{4/3}$
Akiyama	Assembly	Fault $= 4.2 + 0.0015 * Size^{4/3}$
SEL	FORTRAN	Fault $= 53.38 + 0.000056 * Size^{4/3}$

Table 2.2

Based on the first two formulas, Gaffney states that the fault content in a software module is independent of language level. This statement would be more likely to be true if the language level would be the only factor that is different between projects analyzed by Lipow and Akiyama. Validity of these two equations has also been questioned by [Lipow 1986]. Independence of language level is not easy to prove in all environments and for all languages.

STUDY OF FAULTS VS SIZES**(4/3)

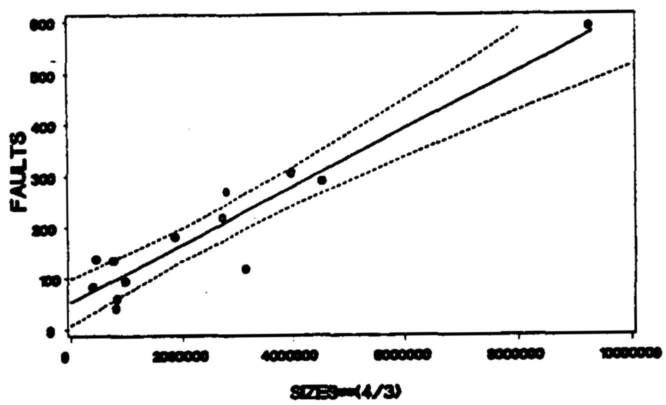

Figure 1.

STUDY OF FAULTS VS SIZES

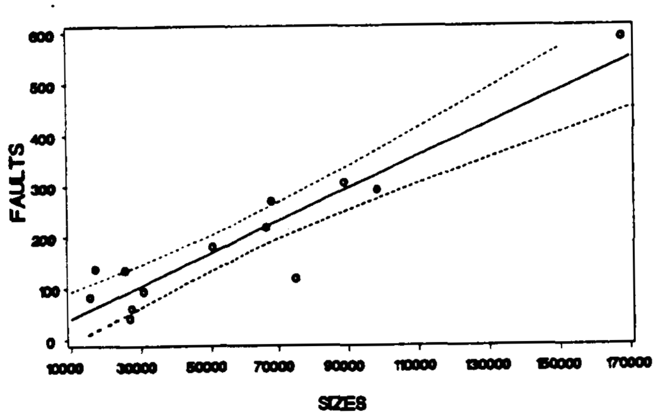

Figure 2.

Assuming that the total fault content is proportional to the size of the software, a linear relationship is obtained for the SEL data, which is

Fault = 8.84 + 0.0032*Size

and Figure 2 shows the linear fit within 95% confidence interval.

The number of faults per KLOC given by Gaffney is six times higher than that of SEL data (Table 2.3). On the average there are 3.3 faults observed for one thousand lines in the SEL environment. This figure is close to the figure for IBM projects [Walston and Felix 1977].

Source	Faults per KLOC	Language
Akiyama	22.7	Assembly
Lipow	21.1	Jovial
Walston-Felix	3.1	Mixed
SEL	3.3	FORTRAN

Table 2.3

In an effort to establish relationships within a project between faults and other variables such as lines of code, number of components changed throughout the development phases, the following observations were made. The number of lines of code do not have any relationship with number of faults observed across the development phase. However the relationship between the number of faults and the number of components changed when compared across projects and also with other projects may reveal intricacy of the faults observed. For example, one could compare the number of components changed per fault at a particular quarter in a phase with the average value for several projects to determine the status of an on-going project [Doerfinger and Basili 1983]. The first four projects listed in Table 2.1 were chosen for detailed analysis. Though these software projects vary in size, their applications are the same. These software systems are Attitude Ground Support Systems that would process telemetry data and provide definitive attitude determination and real time control support for satellite missions. The data for these projects are given in the Appendix. Table 2.4 table lists the number of components changed per fault and the fitted average for the four projects at each quarter starting with the System test phase. A plot of these values is given in Figure 3. This table and the plot clearly reveal that the project 1(DEA) required more than the average number of components to be changed per fault after the end of the second quarter in the System test phase. This information should be helpful to the Software Manager after the middle of System test phase for a project like DEA.

Phase	Quarter	Project 1 (DEA)	Project 2 (DEB)	Project 3 (SMM)	Project 4 (ERBS)	Average (fitted)
System	1	1.68	1.73	1.50	1.36	1.51
	2	1.47	1.41	1.34	1.53	1.54
	3	1.90	1.48	1.33	1.50	1.57
	4	1.90	1.44	1.68	1.56	1.60
Accept	1	1.89	1.51	1.64	1.59	1.63
	2	1.97	1.53	1.58	1.62	1.65
	3	1.97	1.57	1.63	1.60	1.68
	4	1.92	1.56	1.60	1.60	1.71

Table 2.4

The analysis of the results are as follows. Fault estimation based on size is the most simple and the easiest to use empirical formula. It has been shown that the fault estimations based on size of the software is as good as those based on other metrics such as size of vocabulary [Halstead 1977]. However the formulas can not be generalized for all environments. Since the relationships between number of components changed and number of faults are reasonably consistent across three out of four projects, the metric (number of component changed per fault) could be used as a dynamic variable to monitor the development of a software project.

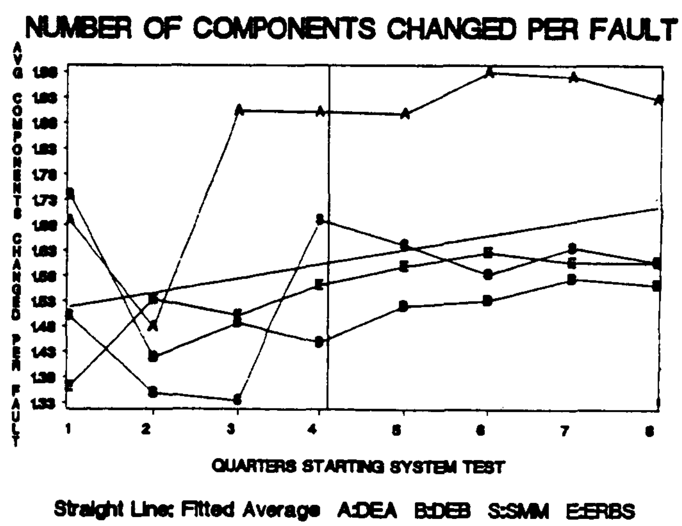

Figure 3.

2.2 Fault Prediction

The main objective of this section is to investigate the approach of curve fitting to predict faults for the four projects. In this method a curve is fitted to previous data using non linear regression and then the fitted curve is extrapolated into the future. An advantage of this approach is it is not dependent on any specific test phase data so one could fit curves at any section of the fault data. Since cumulative fault distributions seem to be asymptotic, a NHPP model was chosen.

The model was fitted to the fault data starting from System test phase and up to half of the acceptance test phase. A total of 15 points were used from the data given in the Appendix and the rest of the acceptance test faults were extrapolated. Figure 4 through Figure 7 display the results for the projects 1 through 4 respectively. The observed faults are also plotted along with the predicted faults for comparison. The fitted models are listed in Table 2.5.

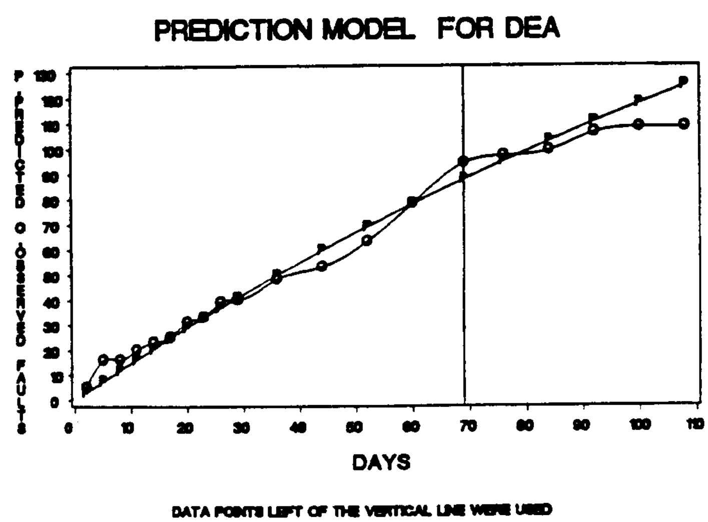

Figure 4.

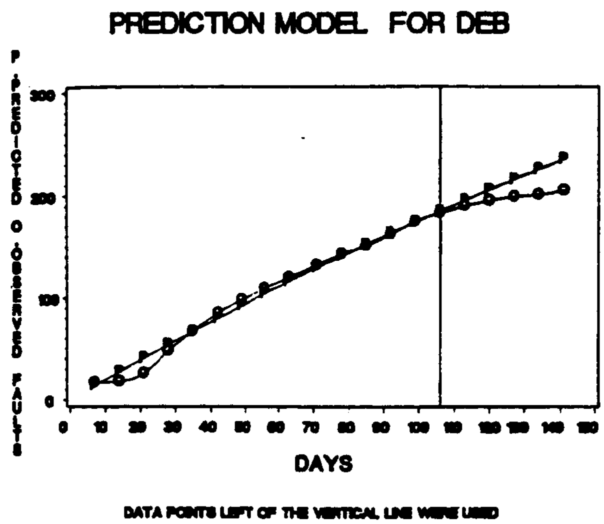

Figure 5.

Project	Prediction Model Equation
1.DEA	$282.06\ (1 - \exp^{-0.00539\,days})$
2.DEB	$755.51\ (1 - \exp^{-0.00265\,days})$
3.SMM	$1150.57\ (1 - \exp^{-0.00065\,days})$
4.ERBS	$656.46\ (1 - \exp^{-0.00385\,days})$

Table 2.5

Project	Data Used	Observed Fault	Predicted Fault	% of Error in Prediction
1.DEA	System & Part of Acceptance	108	124	14.8
2.DEB	System & Part of Acceptance	205	235	14.6
3.SMM	System & Part of Acceptance	122	138	13.1
4.ERBS	System & Part of Acceptance	296	290	2.0

Table 2.6

Table 2.6 lists the total number of faults observed (starting with the system test phase), the predicted faults and the percentage of error in prediction for each project.

For the first two projects, the model overestimates by approximately 15 percent, for the third project by 13 percent and for the last project the predicted and observed faults are close. This method is an alternative to using empirical formulas. There are obvious reasons why it may go wrong in some cases. First, it does not consider the intensity of test cases; second, calendar time, in reality, is not a true representation of test effort. Though several researchers such as [Ellingson 1967], [Coutinho 1973], and [Nathan 1979] have used a similar approach for fault prediction, a more comprehensive model is sought [Shooman 1983]. However, this simple approach provides a scenario of the possible outcome in the future testing period. In this section only one class of curve was applied, but it would be interesting to compare the predicting capabilities of different classes of growth curves.

3. Effort Estimation

Estimation of effort required to detect and correct faults is as important as estimation of faults. In order to predict fault detection and correction effort, we need to know the predicted faults and also detection and correction effort per fault in each phase. The main objective of this section is to investigate the possibility of predicting detection and correction effort per fault in a later phase based on the data from the earlier phases.

Shooman and Bolsky reported, by analyzing a program with 52 faults, that the difficulty of debugging is independent of testing phases [Shooman & Bolsky 1975]. Later Tratchenburg challenged the result by showing that for four medium sized projects, the

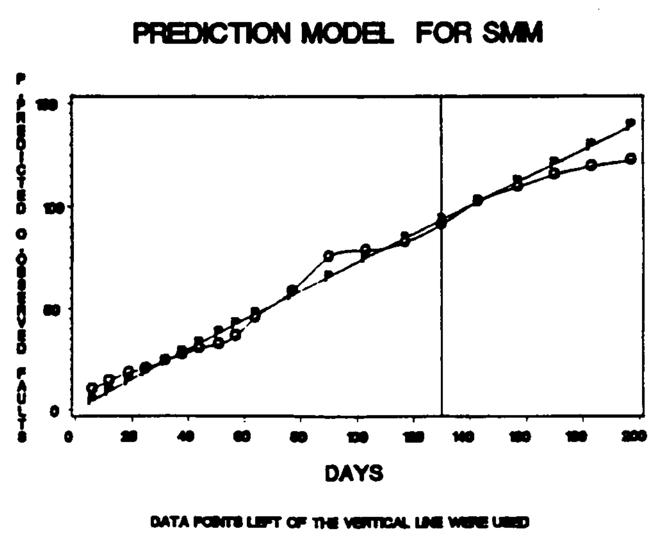

Figure 6.

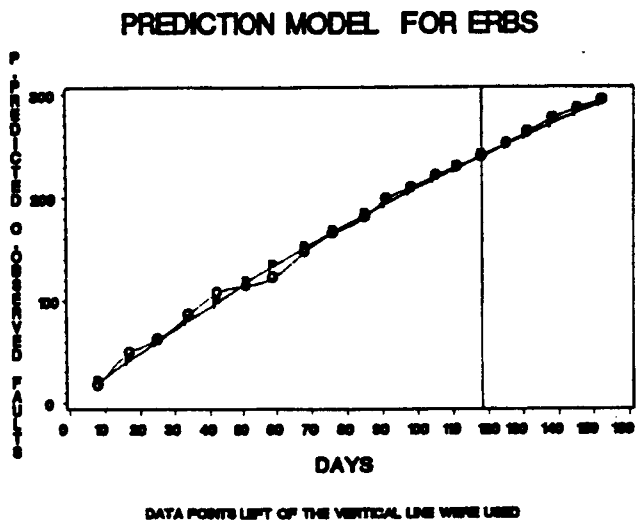

Figure 7.

first half of testing requires 1/2 less effort for fault detection and 1/3 more effort for fault correction than that for the second half [Tratchenburg 1983]. Our results differ from both Shooman's and Tratchenburg's results. We find that there is a pattern (see following tables) of fault fixing effort (detection effort+correction effort) during the phases irrespective of fault detection or fault correction effort. The 1st half represents the first half of the testing phases, that is unit test and first half of system test phase and 2nd half represents the second half of system test phase and acceptance test phase. As stated before the units of efforts are in hours.

Average Effort to Detect & Correct

Project	1st half	2nd half	Ratio
DEA	5.16	7.07	0.73
DEB	5.13	5.49	0.93
SMM	4.92	5.30	0.93
ERBS	5.80	10.03	0.58
Wt. Avg.[1]			0.74

Table 3.1

[1]Weighted by the number of faults

Average Effort per Fault

PROJECT	UNIT	SYSTEM	ACCEPTANCE
DEA	4.9	6.9	7.1
DEB	2.7	5.1	6.2
SMM	4.5	6.8	5.1
ERBS	4.8	9.1	10.1
Wt.Avg	4.7	7.3	7.5

Table 3.2

In contrast to the second half, approximately 1/4 less effort is required for both fault detection and fault correction in the first half. Also, the average effort required to fix a fault in the system test phase is higher than the effort required in the unit test phase. The same relationship holds between the effort required in the acceptance test phase and the system test phase. Although this proportionality varies across projects, we can conclude that the difficulty in debugging is certainly greater in a later phase than in earlier phases in the SEL environment. As stated by Tratchenburg, analysis of this kind combined with fault estimation can help a developer to predict the effort required for debugging in the rest of the testing period. But the results do not appear to be generalizable across environments.

4. Reliability Assessment

To measure the reliability of software, there have been many reliability models proposed [Goel], [Musa], [Jelsinki-Moranda]. These models are mathematical models used to assess the reliability of software from specified parameters which are measured from observations or experiments on software. Another type of reliability study deals with quantitative evaluation of the characteristics of the software which are sensed as associated with high reliability or the lack of it. Complexity of the software has been considered to be such a characteristic [Thayer et.al.78] which leads to low reliability. Difficulty of debugging is considered to be one of the consequences of complexity. In this section, we will address the following questions:

1. Which types of fault are difficult to debug ?
2. What do we learn from applying a reliability model ?
3. How can we monitor the reliability of software during its development ?

The first question is discussed in section 4.1. The second and the third questions are discussed in sections 4.2 and 4.3 respectively.

4.1 Fault Analysis

Fault Analysis helps in identifying the software development methodology and test strategies to prevent and detect faults. Therefore efficient fault categorization has continued to be of great interest to both researchers and software developers. One of the goals in this study is to identify the fault sources that are expensive in terms of detection and correction efforts.

Fault data for the same four projects were analyzed. For each project average effort to fix a fault was calculated for each fault source (SRCERR[2]) in each test phase (Unit, System, Acceptance and Total).

Table 4.1 lists the average effort[3] required to detect and correct a fault in each test phase and for each fault source. The average effort is weighted by the corresponding number of faults in each category and in each project.

Average Effort per Fault

SRCERR	UNIT	SYST	ACCE	TOTAL
REQUIRE	5.00	26.50	1.00	10.83
FUNSPEC	12.00	8.00	8.88	9.59
DESIGN	6.58	7.28	8.16	7.23
CODE	4.14	7.53	7.32	5.85
PREV. C	4.50	5.86	6.93	5.44
OTHER	0.75	1.00	0.00	0.80

Table 4.1

Irrespective of fault sources, the average effort spent to fix a fault for all four projects is 6.14 hours. It is observed that the faults due to incorrect requirements, function specification and design require more than average effort. Therefore, faults resulting from these three sources are considered the most difficult to debug.

---

[2]*Sources of Faults*

REQUIR - Incorrect requirement specification
FUNSPE - Incorrect function specification
DESIGN - Incorrect design
CODE - Incorrect code (possibly semantic fault)
PREV. C - Fault resulting from a previous change
OTHER - None of the above

[3]Average of all four projects

4.2 Reliability Models

Reliability models are at times expensive and difficult to apply. In the past, attempts had been made to use the Musa model in the SEL environment [Miller 1980]. The study showed that reliability models over predicted remaining faults (under predicting the reliability) of the systems to an extent to make the models impractical for this environment. It was argued that the model needed more accurate data on run history than it was feasible to collect, since data was collected in groups of time rather than after each fault. In this study, we tried to use the Goel-Okumoto model [Goel & Okumoto 1980]. This is an NHPP model, which handles group data. Given a history of faults observed over time, it can predict the number of faults to be observed by a particular time. The model is:

$$n(t) = a(1 - e^{-bt})$$

where

n(t) is the number of faults observed by time t,
a predicts the maximum number of faults to be observed,
b is the steepness of the curve.

There are several ways this model can be used. One way is to apply the model at different times and observe whether there is any improvement in the estimated parameters. The other way is to compare the reliability measure against past projects and use one's own judgement based on one's knowledge about the past projects at that particular time. We will look at both approaches.

Using the first approach, the model was applied to the four projects at different times in the test life cycle and comparisons of estimated parameters were made to observe any improvements in reliability measurements. Data given in the appendix were used and the parameters ("a" and "b") were estimated using the algorithm given in [Goel 1982]. The results are shown in Table 4.2 through Table 4.6.

Table 4.2 displays result for only system test data. Table 4.3 and Table 4.4 display the results as more points were added. The parameter "a" is observed to increase and then decrease.

Data Used: System Test

Project	a	b	Faults Observed Before	Faults Predicted	Faults Observed Later
DEA	70.3	0.036	40	30	68
DEB	291.4	0.011	132	159	73
SMM	112.6	0.010	45	67	77
ERBS	378.7	0.010	181	197	112

Table 4.2

Data Used: System + Half of Acceptance Test

Project	a	b	Faults Observed Before	Faults Predicted	Faults Observed Later
DEA	226.06	0.010	93	133	15
DEB	327.03	0.010	183	144	22
SMM	143.9	0.010	91	53	31
ERBS	406.31	0.010	246	160	47

Table 4.3

Data Used: System + Acceptance Test

Project	a	b	Faults Observed Before	Faults Predicted	Faults Observed Later
DEA	141.4	0.013	108	33	-
DEB	271.2	0.010	205	66	-
SMM	141.7	0.010	122	19	-
ERBS	375.0	0.010	293	93	-

Table 4.4

As more data were added the estimated parameters look worse. When only acceptance test data were used the model shows improvement of reliability parameters for all the projects with one exception. It predicts a large number of remaining faults for project 4 (ERBS). The results are shown in Table 4.5 and 4.6.

Data Used: Only Acceptance Test(partial)

Project	a	b	Faults Observed Before	Faults Predicted	Faults Observed Later
DEA	103.78	0.020	59	44	9
DEB	126.8	0.017	63	63	10
SMM	123.9	0.011	64	60	13
ERBS	213.5	0.012	83	130	29

Table 4.5

Data Used: Acceptance Test(full)

Project	a	b	Faults Observed Before	Faults Predicted	Faults Observed Later
DEA	83.6	0.027	68	15	-
DEB	94.6	0.026	73	21	-
SMM	104.9	0.013	77	28	-
ERBS	242.0	0.011	112	130	-

Table 4.6

Fault detection rate is dependent on complexity and amount of code covered by successive test cases. Usually, in the SEL environment, simple tests are performed initially and followed by more complex tests. This study confirms the results by [Miller] on a different set of projects. Reliability models do not appear to be useful in this environment.

4.3 Reliability Metrics

One of the important tasks of a software manager is to monitor the quality and reliability of the software being developed. Fault data for the past projects in the same environment help in the comparison of reliability metrics for an on-going project with past projects. Since different projects have different sizes and different duration of testing time, normalized reliability metrics can be compared at each quarter in each phase. One would expect normalized reliability metrics, such as fault rate (faults per day) or fault density (faults per day per line of code), to be constant and same during the test life cycle. This does not seem to be true, even in the same environment and for similar applications written in same language. The sizes of the four projects varies from 66K to 160K. This may have contributed to the variations of the normalized metrics. In this situation, using the fault data for the four representative projects, we present an acceptable fault density range at different intervals during the testing phases in order to facilitate reliability monitoring. The following tables summarize the fault densities for the projects and the acceptable ranges for the fault density at each interval, where acceptable ranges have been calculated using "mean +/- variance".

Fault Density

Project	Unit			
	25%	50%	75%	100%
DEA	2.08	2.94	3.08	3.00
DEB	3.22	6.45	4.84	3.76
SMM	3.97	2.95	2.11	1.54
ERBS	0.06	0.05	0.99	0.85

Project	System			
	25%	50%	75%	100%
DEA	3.04	2.89	2.73	2.65
DEB	2.68	3.19	3.20	3.01
SMM	1.38	1.22	1.15	1.11
ERBS	0.86	0.85	0.79	0.83

Project	Acceptance			
	25%	50%	75%	100%
DEA	2.47	2.48	2.26	2.08
DEB	2.89	2.75	2.56	2.22
SMM	1.05	0.97	0.94	0.82
ERBS	0.84	0.83	0.83	0.82

Table XII

Acceptable Ranges for Fault Density

UNIT Test Phase

25%	50%	75%	100%
$2.33 \pm Z_{\alpha/2}\,0.85$	$3.09 \pm Z_{\alpha/2}\,1.30$	$2.75 \pm Z_{\alpha/2}\,0.81$	$2.28 \pm Z_{\alpha/2}\,0.66$

SYSTEM Test Phase

25%	50%	75%	100%
$1.99 \pm Z_{\alpha/2}\,0.51$	$2.03 \pm Z_{\alpha/2}\,0.58$	$1.96 \pm Z_{\alpha/2}\,0.58$	$1.90 \pm Z_{\alpha/2}\,0.54$

ACCEPTANCE Test Phase

25%	50%	75%	100%
$1.81 \pm Z_{\alpha/2}\,0.50$	$1.75 \pm Z_{\alpha/2}\,0.49$	$1.64 \pm Z_{\alpha/2}\,0.44$	$1.48 \pm Z_{\alpha/2}\,0.38$

Table XIII

There are two important observations obtained from analyzing these fault densities for different projects. First, the fault density at the end of unit test phase is greater than the fault density at the end of the system test phase and the same relationship holds for the system test phase and acceptance test phase. Second, the fault density measured during the acceptance test phase shows a gradual decrease. Notice that the change of fault density for project ERBS during the Acceptance test phase is almost steady. This is different from other projects though the fault density for this project is lower than other projects. This is, in a way, a reflection of increase in fault rate because of more complex tests being performed during the second half of the acceptance test.

It was observed that the faults due to incorrect requirements, function specification and design require more than average effort. We also investigated the possibility and criteria for using a reliability model in the SEL environment and it was found that the reliability models look worse and worse as more data is added. Finally, we provided a basis for comparison of fault distributions in order to monitor reliability of an on-going project.

5. Conclusion

In summary, an appropriate estimation formula or a fitted curve may be used to predict the number of faults in the future and therefore fault correction effort also can be predicted. Faults in software originate from several sources; it has been found that the longer the fault stays in the system, the more costly it is to remove. It has been observed that the fault resulting from requirement specifications, function specification and design require more effort to fix than faults resulting from other sources. This statement is based on the data for only four projects. Comparison of fault density at different intervals may be made with the same metric for other past projects to assure the status of the project. Fault densities are observed to decrease from the unit test phase to the end of the acceptance test phase. Also within the acceptance test phase the fault densities at an interval is lower than in the previous interval. Both horizontal comparison, i.e. comparing with other projects at a particular interval, and vertical comparison, i.e. comparing within the same project at different intervals, were considered for the application of a reliability model, but were not effective in this environment.

Acknowledgments

The authors are grateful to Dr. Amrit Goel of Syracuse University, Frank McGarry of NASA/Goddard Space Flight Center, Dr. Gerald Page of Computer Sciences Corporation, Dr. Dieter Rombach and Nora M. Panlilio-Yap of the University of Maryland for their suggestions and support. Computer support was provided by NASA/Goddard Space Flight Center, Department of Computer Science, Computer Science Center and Sea Grant College at the University of Maryland.

References

[1] Victor R. Basili and Barry T. Perricone, "*Software Errors and Complexity: An Empirical Investigation,*" Computer Science, Univ. of Maryland, 1982, UOM-1195.

[2] Victor R. Basili and David M. Weiss, "*Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*", IEEE Transaction on Software Engineering, Vol. SE-11, No.12, pp 157.

[3] V. R. Basili, M. V. Zelkowitz, F. E. McGarry, R. W. Reiter, W. F. Truszkowski, and D. L. Weiss, "*The Software Engineering Laboratory*", Tech. Rep. TR-535, Department of Computer Science, University of Maryland, College Park, May 1977.

[4] J. S. Coutinho, "*Software Reliability Growth*", IEEE Symp. Comp. Software Reliability, 1973.

[5] C. W. Doerflinger and V. R. Basili, "*Monitoring Software Development Through Dynamic Variables*", IEEE Transaction on Software Engineering, Vol. SE-11, No. 9, Sept. 1985, pp. 978.

[6] O. E. Ellingson, "*Computer Program and Change Control*", IEEE Symp. Comp. Software Reliability, 1973.

[7] Kurt F. Fisher and Michael G. Walker, "*Improved Software Reliability Through Requirements Verification*", IEEE Transactions on Reliability, August 1979.

[8] J.E. Gaffney, Jr., "*Estimating the Number of Faults in Code*", IEEE Transaction on Software Engineering, Vol. SE-10, No.4, July 1984, pp 459-464.

[9] A. L. Goel and K. Okumoto, "*A Time Dependent Error Detection Model for Software Performance* Assessment with Applications", annual report to RADC, Department of Industrial Engineering and Operations Research, Syracuse University, New York, March 1980.

[10] A. L. Goel, "*Software Reliability Modeling and Estimating Techniques*", RADC-TR-82-263, Air Force Systems Command, Griffiss Air Force Base, NY 13441.

[11] M. Halstead, "*Elements of Software Science*", Elsevier North-Holland, Inc., New York, 1977.

[12] A. Iannino, J.D. Musa, K. Okumoto and B. Littlewood, "*Criteria for Software Reliability Model Comparisons*", IEEE Transactions on Software Engineering, vol. SE-10, No.6, November 1984, pp 687-691.

[13] Myron Lipow, "*Comments on "Estimating the Number of Faults in Code" and Two Corrections to Published Data*", IEEE Transactions on Software Engineering, Vol. SE-12, No. 4, April 1986, pp 584-585.

[14] Irwin Nathan, "*A Deterministic Model to Predict Error-Free Status of Computer Software Development*", Proc. Workshop on Quantitative Software Models, Kiamesha Lake, NY, IEEE, Oct 9, 1979.

[15] Anna-Mary B. Miller, "*A Study of the Musa Reliability Model*", Master's Thesis, Computer Science Dept., Univ. of Maryland, 1980.

[16] J. D. Musa, "*A Theory of Software Reliability and Its Application*", IEEE Transactions on Software Engineering, vol. SE-1, No. 3, September 1975, pp 312-327.

[17] M. L. Shooman, "*Software Engineering*", Computer Science Series, McGraw Hill Publications, 1983.

[18] M. L. Shooman and M. J. Bolsky, "*Types, distributions and test and correction time for programming errors*", ACM SIGPLAN Notices, Vol. 10, pp. 347-357, June 1975.

[19] M. Trachtenberg, "*Order and Difficulty of Debugging*", IEEE Transactions on Software Engineering, Vol. SE-9, No.6, November 1983, pp 746-747.

[20] Thomas A. Thayer, Myron Lipow & Eldred C. Nelson, "*Software Reliability*", TRW Series of Software Technology, Volume 2, 1978, North-Holland Publishing Company.

[21] C.F. Walston and C.P. Felix, "*A Method of Programming Measurement and Estimation*", IBM System Journal, Vol. 16, No.1, 1977, pp 54-73.

Project 1 (DEA)				
Phase	Quarters in Phase	Days Elapsed	Faults	Components Changed
Unit	1	20	25	37
	2	41	76	122
	3	62	124	236
	4	83	162	296
System		85	167	
		88	178	
	1	90	178	323
		91	178	
		94	182	
	2	97	185	330
		100	187	
		103	193	
	3	104	193	355
		106	195	
		109	201	
	4	112	202	372
Acceptance		119	210	
		127	215	
	1	131	220	406
		135	225	
		143	240	
	2	152	256	480
		159	259	
		167	261	
	3	171	263	493
		175	268	
		183	270	
	4	191	270	504

Table A.1

Project 2 (DEB)				
Phase	Quarters in Phase	Days Elapsed	Faults	Components Changed
Unit	1	1	2	2
	2	3	10	10
	3	4	12	12
	4	6	14	22
System		13	31	
		20	32	
	1	23	37	62
		27	40	
		34	62	
	2	41	81	117
		48	99	
		55	112	
	3	59	119	178
		62	123	
		69	134	
	4	77	146	213
Acceptance		84	157	
		91	165	
	1	94	174	265
		98	176	
		105	189	
	2	112	197	302
		119	204	
		126	209	
	3	129	211	332
		133	213	
		140	215	
	4	147	219	342

Table A.2

Project 3 (SMM)				
Phase	Quarters in Phase	Days Elapsed	Faults	Components Changed
Unit	1	42	45	63
	2	84	100	149
	3	126	127	192
	4	168	169	243
System		174	176	
		180	180	
	1	184	185	267
		187	185	
		193	186	
	2	200	190	274
		206	193	
		212	196	
	3	216	198	283
		219	198	
		225	202	
	4	232	214	319
Acceptance		245	224	
		258	241	
	1	265	244	366
		271	244	
		285	248	
	2	298	257	387
		311	268	
		325	275	
	3	331	281	426
		338	281	
		351	285	
	4	365	291	

Table A.3

Project 4 (ERBS)				
Phase	Quarters in Phase	Days Elapsed	Faults	Components Changed
Unit	1	68	0	0
	2	136	6	8
	3	204	120	154
	4	272	293	383
System		280	310	
		289	342	
	1	293	343	451
		297	355	
		306	379	
	2	314	400	547
		323	407	
		331	415	
	3	335	423	578
		340	440	
		348	458	
	4	357	475	667
Acceptance		363	491	
		370	502	
	1	373	507	725
		377	514	
		383	522	
	2	390	532	773
		397	544	
		403	555	
	3	407	562	815
		410	568	
		417	578	
	4	424	589	855

Table A.4