An Evolution of Reliability of Large Scale Software of a Switching System

대형 교환 시스템의 소프트웨어 신뢰도 성장

J.K. Lee (이재기) Switching Technology Department (시스템종합팀 선임기술원)
S.K. Shin (신상권) Switching Technology Department (시스템종합팀 기술기능원)
S.S. Nam (남상식) Switching Technology Department (시스템종합팀 책임연구원, 팀장)
K.C. Park (박권철) Switching Technology Department (교환기술연구부 책임연구원, 부장)

In this paper, we summarize the lessons learned from the applications of the software reliability engineering to a large-scale software project. The considered software is the software system of the TDX-10 ISDN switching system. The considered software consists of many components, called functional blocks. These functional blocks serve as the unit of coding and test. The software is continuing to be developed by adding new functional blocks. We are mainly concerned with the analysis of the effects of these software components to software reliability and with the analysis of the reliability evolution. We analyze the static characteristics of the software related to software reliability using failure data collected during system test. We also discussed a pattern which represents a local and global growth of the software reliability as version evolves. To find the pattern of software of the TDX-10 ISDN system, we apply the S-shaped model to a collection of failure data sets of each evolutionary version and the Goel-Okumoto (G-O) model to a grouped overall failure data set. We expect this pattern analysis will be helpful to plan and manage necessary human/resources for a new similar software project which is developed under the same developing circumstances by estimating the total software failures with respect to its size and time.

I. Introduction

A large-scale software project is generally defined in terms of the amount of necessary human/resources, more than 100 people, and necessary duration, more than one year. Software of a switching system is a typical example of this type. In [1], L. Bernstein reported the difficulties of a large-scale project management and he suggested some management principles emphasizing the deployment of people working on the project.

In general, management of software projects are improved through a review of the past experiences. In this paper, we summarize experience obtained from the studies of software reliability applied to a large-scale software project of the TDX-10 ISDN switching system. The

TDX-10 ISDN switching system, released at 1995, is now successfully in use in Korea.

The software has been designed by using functional components which are called functional blocks. A functional block not only provides the behavior required by its specification, but also serves as the unit which try to identify possible software faults in the system testing. The software is continued to be developed by adding new functions in several categories.

There is a recent study which deals with the general description for the evolution of software reliability for the telecommunications system [2]. Motivated by the studies about management and evolution of software reliability, we analyze the effects of these functional blocks to software reliability and the reliability evolution.

To investigate the structural effect to software reliability, we analyze the static characteristics of the software related to software reliability using failure data collected during system test. We, also, analyze the pattern which represents a local and global behavior of a software reliability growth. With this pattern, we can easily get practical information used to determine the time of adding new functions keeping the software reliability. Here by the local behavior, we mean the reliability growth of each of the evolutionary version and by the global behavior, we mean the overall reliability growth of the software.

In section 2, we give a description for the software components of the TDX-10 ISDN switching system. System test methods and failure management based on the functional blocks, are explained in section 3. In section 4, we give statistics for failures with respect to the functional blocks. The evolution of reliability is analyzed in section 5.

II. Software Components

Five years and 80 ~ 120 humans / year are invested in developing this software. Depending on the implemented features, we divide the software into three official versions, referred to as N3.3, N3.4, and N3.5. N3.3 (N3.4) is further divided into N3.3a and N3.3b (N3.4a and N 34.b) depending on the implemented services.

The first official version, N3.3, includes all the features of PSTN (Public Switched Telephone Network) and basic features of ISDN (Integrated Services Digital Network) with CCS (Common Channel Signaling) No.7 facilities. N3.4 includes all the features of ISDN. N3.5 is the last version and is released to the field. The size of this software is 1,330 in kilolines of source codes (KLOC).

The TDX-10 ISDN system is designed with a fully distributed architecture. Depending on this architecture, the software is designed by the modular atomic components, referred to as functional blocks. These functional blocks constitute the units of development, and also serve as the unit of identifying software faults in the system test. In N3.5, the final 140 functional blocks are developed. The size of each functional block range over 2 – 10 KLOC.

A user function is performed by many functional blocks interacting in a sequential manner. Interactions among functional blocks are accomplished by exchanging signals via a high speed inter-processor communication link.

Each functional block is developed independently by a team consisting of two or three members.

We classify these functional blocks, following the organization of development teams, into the following categories:

- Kernel (K),
- Call Processing (CP),
- Data Handling (DH),
- Administration (Ad),
- Operation and Maintenance (OM).

The kernel category includes the concurrent real-time operating system (CROS) and real-time relational data base management system (RDBMS). The functions for services of PSTN, ISDN, CCS No.7, and X.25/X.75 packet are collected into the CP category.

Functions related to handling data of call processing and system configuration are collected into the DH category. Functions for administration are collected into the Ad category, and functions for operation and maintenance are into the OM category. The number of functional blocks, the size, and the number of associated system functions per category is given in Table 1 for the final version N3.5.

<Table 1> No. of functional blocks, size and system functions per category.

Categories	No. of Block	Size (K)	Functions
CP	45	390.7	270
DH	14	171.9	33
Ad	44	315.3	290
OM	29	240.7	241
K	8	217.5	–
Total	140	1336.1	834

III. System Test and Failure Management

1. System Test

Since user functions are accomplished by many functional blocks, they can only be completely validated by a system test. And so, the period of a system test is relatively long. System tests, lasting about three months, are repeated by three times to each version. Thus, we take nine months as the unit of analyzing failure data of each version, and one month as the unit of grouping the failure data.

In the beginning of test, focuses of test are put to verify system's functionality, but as version evolves, focuses are moved to measure system's qualities. A scenario based testing strategy is used, because of the large size and the lack of precise usage information. Test scenarios are chosen within a framework which describes user's usage situations as much as possible. But, once some software failures are corrected, a series of test cases related to the detected software failure are conducted to verify related functions.

During a system test, all failure data are gathered for accessing and predicting the software reliability. Since a unit of execution in the software is a functional block, the location of software failures can be easily identified in a system test.

2. Management of Failures and Correction of Software Faults

All abnormal behaviors observed during the period of a system test are recorded to a FMS

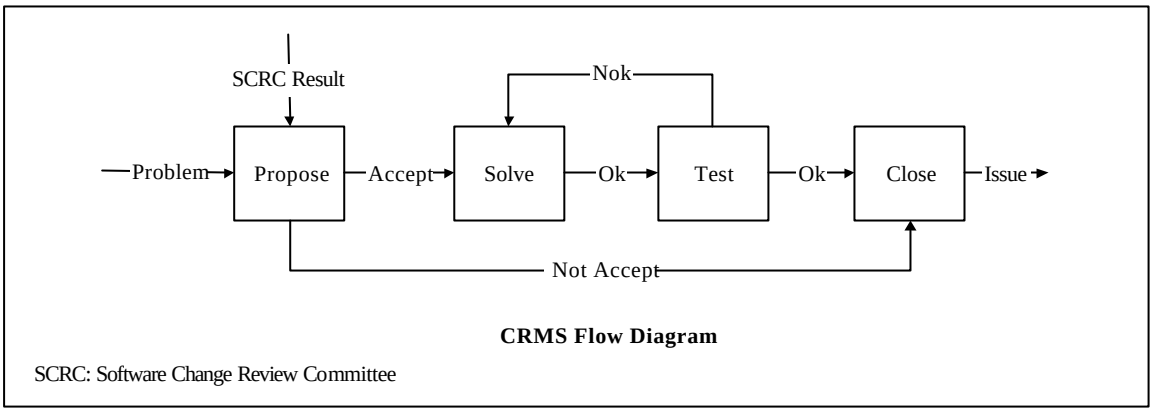

Fig. 1. Control folw chart of the CRMS.

(Failure Management System) in an on-line fashion. Failures reported by FMS are examined by FMCC (Failure Management Control Committee), held in a week basis, to distinguish software failures. Members of FMCC consist of persons of design teams, development teams, packaging team, and test team. In FMCC, cause analysis is performed to each failure at the functional block level. Results of cause analysis are announced to the concerned developers to identify and correct software faults. Corrections of software faults are accomplished through three stages. To correct a software fault, a developer submit a change request (CR) to the SCMC (Software Change Management Committee). On a reception of CR, SCMC keeps track of the state of the proposed CR. A corrected software is built into a temporal package, and a regression test for this package is performed to verify the corrections by repeating tests depending on the tree of functions. If the correction is verified, the state of a proposed CR is closed. If a verification fails, the status of a CR remains a continued state, and the same test procedures are repeated. SCMC keeps all data of CRs during these periods and these data are used to reliability analysis.

All of the CRs is controlled by CRMS (Software Change Request Management System) (see Fig. 1).

IV. Statistics of Failure Data

In this section, we analyze the statistics of software failures in order to investigate the characteristics of the software from the software reliability viewpoint. These statistics are used to identify function blocks which are the most fault prone, and to estimate the measures of the software reliability. In addition to this analysis, there is a study on attributes to the software reliability [8].

In [8], we applied to the failure data collected on during the system test using of the poission regression model. the majority of purpose to this model is promotion of the software

quality. In other words, the concept of the poission regression is introduced to explain the dependency between software failure data and several explanatory variables. for example, dependence of software complexity as to failure criticality, size of source line, etc. therefore, this model is proper to our project in development of switching system and software failure count modeling.

1. Software Failure Intensities

To view the reliability evolution, we use failure intensity, indicating the number of grouped failures per unit time (one month). Figure 2 shows the relationship between all failures and the versions of the software during the period of system tests (the unit of x-axis is month).

2. Failure Distribution among Categories

Table 2 shows the failure distribution among the categories mentioned in section 2.

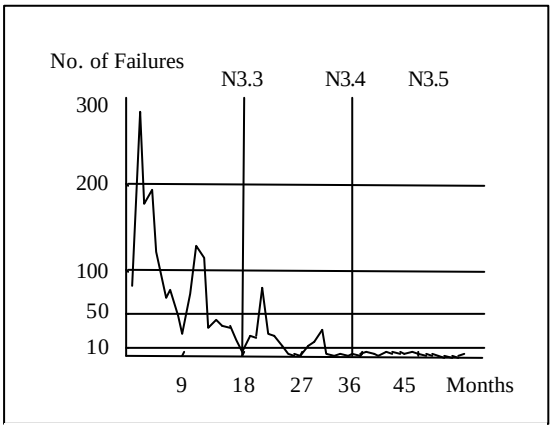

Fig. 2. Failure distribution to versions.

As expected, the distribution of failures among categories depends on the variety of functions of a category. The Ad and OM categories, taking 42 % of the software, contains 65 % of failures.

In Table 2, N3.3 and N3.4 have the similar distribution of failures to each functional category. But in N3.5, the number of failures in the CP category is relatively high compared with that of N3.3 and N3.4. This phenomenon explains that, in a field test, functions in the CP category are more frequently used than other categories.

<Table 2> Software failure distribution to categories.

Version	CP	DH	Ad	OM	K	Total (%)
N3.3	25.3	3.7	42.0	25.7	3.3	100
N3.4	24.4	4.0	42.2	26.1	3.3	100
N3.5	33.0	0.5	42.8	21.1	2.6	100

3. Faults Reduction Factor

By fault reduction factor, known as the ratio of software faults and the number of modified blocks, we can measure the functional corelations among functional blocks.

Table 3 is a statistical data indicating the number of modified blocks to correct software faults. The fault reduction factor estimated from Table 3 is around 0.90. This value can be comparable to fault reduction values of other case studies in [2], [4].

The portion of software faults which are corrected by modifying 1 – 3 blocks is 94%. This

<Table 3> Modified number of blocks and CRs.

No. of Modified Blocks	CRs	Ratio (%)
1	1,401	78
2	201	11
3	89	5
> 4	109	6

shows that the most of blocks keeps a functional independence. But, blocks, modified more than 4 times per one fault, have many interactions with other blocks. This say that functional blocks having many interactions are of fault-prone.

4. Fault Density Distribution among Categories

A common measure used to evaluate code quality is a fault density, defined as the number of software faults divided by size. In Table 4, we show the fault density distribution among categories. The average fault density is less than 2.0 Fault/KLOC. Fault density will be used to estimate the amount of faults in a new similar project.

5. Correction Time Distribution

The distribution of time required to correct faults is useful in predicting software maintenance effort during the field operation. In Table 5, we show the statistics of duration of CRs. Required time for a failure correction is highly correlated with its contents of fault. The failures due to design fault require longer correction time, but do not occur frequently. Through semantic analysis, we see that 65% of failure

<Table 4> Fault density distribution among categories.

Category	Fault density
CP	2.13 = 833/390K
DH	2.32 = 397/171K
Ad	1.24 = 392/315K
OM	2.19 = 526/240K
K	0.28 = 71/ 257K
Average	1.632 = 2,219/1,373K

<Table 5> Life duration of CRs.

Month	Numbers of CR	Percent (%)
1	826	65.7
2	210	16.7
3	81	6.44
4	46	3.70
5	28	2.23
6	23	1.80
7	13	1.03
8	12	0.96
9	12	0.96
10	3	0.24
11	3	0.24

are caused due to control and assignment error within a block.

V. Evolution of Software Reliability

To get the pattern of software reliability growth, we apply the software reliability growth theory to the failure data of each version and to the grouped overall failure data.

Since we are interested in the reliability growth pattern related to evolution of versions, assuming that the arrival of failures are independent in each version, we consider two non-

homogeneous Poisson process models, in particular, the S-shaped [5]-[7] and Goel-Okumoto model [3], [4]. In the G-O model, the failure intensity decrease from the initial time, but in the S-shaped model it increase to some finite time then gradually decrease. The S-shaped model is a model which implies the assumption of the failure intensity depends on the testers experience.

Recall that the cumulative failure count in the S-shaped model is given by

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

a : expected number of failures,
b : failure rate

and the cumulative failure count of G-O model is

$$m_{GO}(t) = a(1-e^{-bt}),\ a,b>0. \quad (2)$$

Here a is the expected number of failures that will be eventually be detected and b is the failure rate. Also, b is the failure occurrence rate per fault. In the Table 6, we give the total number of failures of each version. N3.3a and N3.3b (N3.4a and N3.4b) are the two temporal versions of N3.3 (N3.4).

Applying the maximum likelihood estimation method to the failure data in Table 6, as shown in the Fig. 2, we obtain the estimated values of parameters, a and b for the S-shaped model as in Table 7.

In Fig. 3, we show the local behavior of the failure intensity growths of each version. The finite times which maximize the failure intensity are varied within 1-2 months in our cases.

<Table 6> Failure data of N3.3 and N3.4.

Month	N3.3a	N3.3b	N3.4a	N3.4b
1	83	69	23	11
2	287	129	21	16
3	177	117	81	30
4	193	31	24	2
5	120	40	22	0
6	67	34	12	1
7	75	35	2	0
8	46	20	1	2
9	24	5	0	0

<Table 7> Estimation a, b in S-shaped model.

Ver. No.	a	b
N3.3a	1,118.14	0.552831
N3.3b	90.31	0.641667
N3.4a	188.59	0.697083
N3.4b	62.13	0.937313

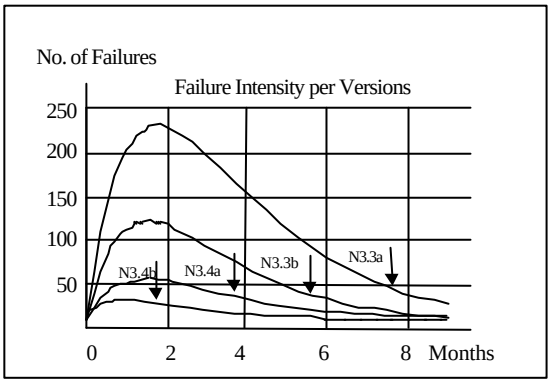

Fig. 3. Local evolution of reliability growth.

The total failure counts per version are shown in Table 8.

To get the global software reliability growth pattern, we apply the G-O model to the failure data in Table 8. The maximum likelihood

<Table 8> Number of total failures of each version.

Ver.	N3.3a	N3.3b	N3.4a	N3.4b	N3.5	Field
Failures	1072	482	186	62	20	4

estimation for a and b for G-O model is calculated as a = 1842.02 and b = 0.101597. With these values, we get the global failure intensity function as

$$m'_{GO}(t) = 1842.02 * 0.101597\, e^{-0.101597t} \quad (3)$$

Others, the cumulative failures of S-shaped Model shown in Fig. 4.

If we compare the local failure intensities of each version with the global one, we have the curve which shows the pattern of software reliability evolution pattern as shown in Fig. 5.

Using (3), we predicted the failure intensity to be 0.77452 during the next 9 months, it is comparable to the real value 0.44444.

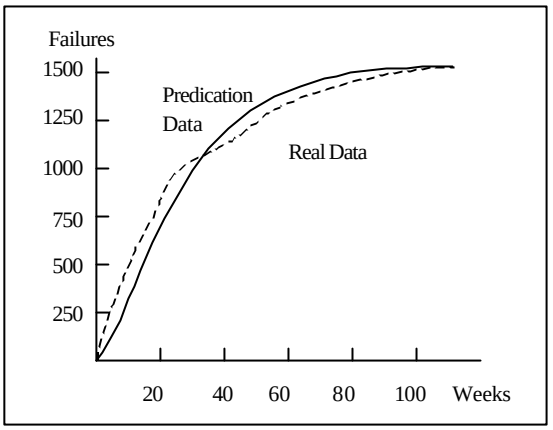

Fig. 4. Cumulative failures of S-Shaped model.

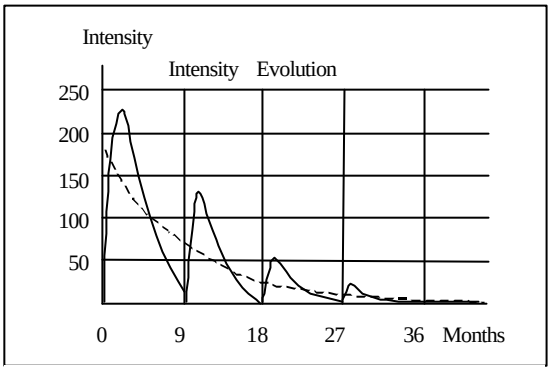

Fig. 5. Evolution of failure intensity.

VI. Conclusion

We analyzed the several issues, related to the software reliability, of the software designed by the functional block components. We found that, regardless of the structuring components of the software, the estimated values related the software reliability, such as, fault density, failure rates, etc, are comparable to that of founded in literature [2]-[4]. But, the evolution of the failure intensity, heavily depending on the test strategy and the deployed humans, show different behavior that of reported in [2]. Taking into considerations of our test methods, we applied two software reliability growth models to a collection of failure data sets and get a pattern of evolution failure intensity which showing the local and global software reliability growth.

As versions evolve, the size and complexity of software will be increased and test methods complicated, and so the detection of software failures will be more difficult. And so, more systematic resources management will be needed. The knowledge about the pattern of a local and global software reliability growth will

be helpful to plan the developing process and to manage allocations of resources to improve quality of a new similar software by estimating the total software failures with respect to its size and time.

Reference

[1] L. Bernstein, "Software in the Large," *AT & T Technical Journal*, Jan/Feb. 1996, pp. 5 – 14.

[2] M. Kaaniche, K. Kanoun, "Reliability of a Commercial Telecommunications System," *ISSRE '96*, 1996, pp. 207 – 211.

[3] Michael R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1995.

[4] J. Musa, A. Lannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1987.

[5] S. Yamada and S. Osaki, "Non-homogeneous Error Detection Rate Models for Software Reliability Growth," *Stochastic Models in Reliability Theory*, Springer-Verlag, Berlin, 1984, pp.

120 – 143.

[6] S. Yamada *et al.*, "Software Reliability Growth Model of Two Types of Errors," R.*A.I.R.O. Operations Research*, Vol. 19, No. 1, Feb. 1985, pp. 8 – 104.

[7] Shigeru Yamada and Hiroshi Ohtera, "Software Reliability: Theory and Practical Application," *SE Series*, Soft Research Center, Feb. 1990, pp. 135 – 196.

[8] S.Y. Lee, "An Application of Poisson Regression to a Switching Software Failures," *ISSAT'98*, 1998.