An Evolution of Software Reliability in a Large Scale Switching System: using the software

Jae Ki Lee *Regular Member*, Sang Sik Nam, Chang Bong Kim* *A Lifelong Members*

Summary

In this paper, an evolution of software reliability engineering in a large-scale software project is summarized. The considered software consists of many components, called functional blocks in software of switching system. These functional blocks are served as the unit of coding and test, and the software is continuously updated by adding new functional blocks. We are mainly concerned with the analysis of the effects of these software components in software reliability and reliability evolution.

We analyze the static characteristics of the software related to software reliability using collected failure data 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 system software, 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.

Key words: *Failure intensity, fault density, failure, software reliability, system test, software functional block (SB), change request (CR).*

1. Introduction

A large-scale software project is generally defined in terms of the amount of required human resources and duration, Software of a switching system is called a large-scale, for required person power is over 100 and duration is also over one year. In [1], L. Bernstein reported the difficulties of a large-scale project management and hesuggested some management principles emphasizing the deployment of people working on the project. In general, management of software projects isimproved 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 switching system. The switching system, released at 1995, is now successfully in operation in Korea.

The software has been designed by using functional components, which are called functional blocks. A functional block not only provides the behaviorrequired by its specification, but also serves as the unit that tries 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

* 한양대학교 전자전기컴퓨터공학부 (skpark@hanyang.ac.kr)
논문번호 : 030478-1103, 접수일자 : 2003년 11월 3일

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 to determine the time of adding new functions by keeping the software reliability. Here by the local behavior, the reliability growth of each evolutionary version and global behavior, we mean the overall reliability growth of the software.

In section 2, we give a description for the software components of the 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.

2. Software Components and Versions

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 N34.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 kilo-lines of source codes (KLOC).

The switchingsystem 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 blocks range is over 2 - 10 KLOC.

Many functional blocks interacting in a sequential manner perform a user function. 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:

- System Kernel (SK),
- 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 (KLOC)	Functions
CP	45	390.7	270
DH	14	171.9	33
Ad	44	315.3	290
OM	29	240.7	241
SK	8	217.5	-
Total	140	1336.1	834

3. System test and failure management

A. 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 tomeasure 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.

- Test procedure and activity

The main tests, executed at each development phase of switching system developed in ETRI are unit test, function test, integration test and system test. During these tests, system errors (hardware and software error) are collected and analyzed by the CRMS(Software Change Request Management System). The CRMS (meaning of the defect tracking system)is test tools for software reliability, providing the progress and processing activities of the project development schedule, as well as software test data.

Each error is summarized with CR(Change Request) sheet which was divided into the cause and kind of errors by using CRMS.

The occurred errors during system test were recorded and placed in one of the following 3-type of categories. We also could classify the occurred errors as category 2, 3.

In case of emergency error, we considered as serious affect service. These errors areprocessing of high priority and corrected for the error.

- Category of error type

Type 1 : logical error, interface(internal/ external) error, wrong data type, error in use of resource, coding standard violation error, rule or convention error, error in previous phase, commentary error

Type 2: call processing, operation, maintenance, data base, operating system, others

Type 3: critical (emergency), high, medium, low, others

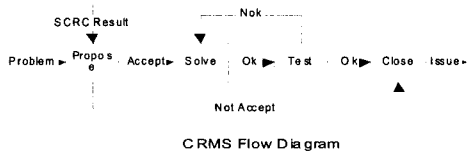
B. Management of Failures and Correction of Software Faults

All abnormal behaviors observed during the period of a system test are recorded to a FMS (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 submits a change request to the SCMC (Software Change Management Committee). On a reception of CR, SCMC keeps track of the state of the proposed CR.

Correctedsoftware 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 offunctions. If the correction is verified, the state of a proposed CR is closed. But the 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 are controlled by CRMS (see to Fig. 1)

The CRMS contains 2,000 CR collected during the 4-years. The software change requests constitute the most important part of data with system development process. The following figure illustrates some of the most commonly used state transitions. However, many other state transitions are also possible. For example, we can move a problem from Propose to close or re-solve a problem by moving it from test to solve. Our organization can customize the states and state transitions available on a class-by-class basis. [9],[10] In these studies, Recently reported papers of assessing the evidence from Change Management Data, we founded [18-19]



SCRC : Software Change Review Committee

Fig. 1 Control flow chart of the CRMS

- Failure specify

Detection of failures during system test is classified to cause analysis and semantic analysis method. Occurred to failure, it's analysis of location, cause and content. Also, the failure location divided two methods. 1'st analysis are category type 1, function, blocks for a location, 2'nd analysis is detailed to classify for a content of failure.

- Details failure classification

Software structure, signal, control and assignment, data generation, packaging or environment of development

This classify idea is consider a question in all it's aspects of software design and programming.

Figure 2 plots a number of fault density in software functional blocks (SBs) each version. Two curves have similar shapes but the fault expose ratio (N3.3 after) in software blocks is smaller than N3.3 before. For this reason, we discussed section 4-A. In other words, N3.4 version is stable than N3.3.

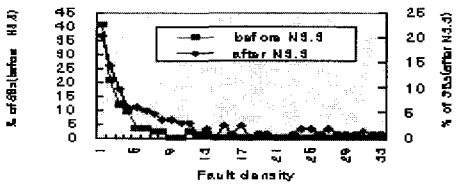


Fig. 2 Fault density distribution among the SBs

Figure 3 present that the statuses of propose and closed the software change request per month. The analysis of time evolution between proposed and closed of CRs gives an insight into the correction effort to solve the problems (or system failures) and to trace of the efficiency of failure analysis process like as front of mentions. Among the 2,200 failure report, we have solved only 2,000 failures (real data : 1680 CRs). The remained failures are canceled not corrected. These failures (about 300 failures) are determined to be a duplicate of an existing failure report or not a fault.

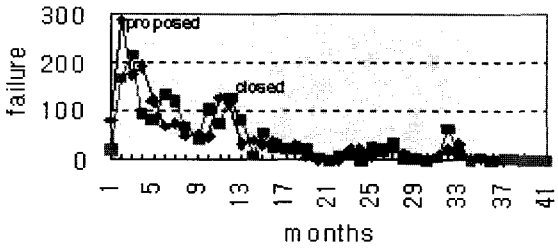
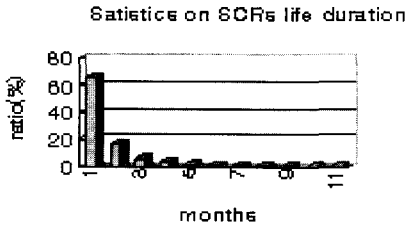


Fig. 3 Number of proposed and closed software CR

In figure 3, the number of proposed and closed failure increased to radicallyin 3, 11 and 33-month start of the system test. The cause of added, we have trial to practical and commercial test adaptation of field in service. So, the purpose of detection failures and the fixed of software bugs during the system test, we change to test scenario with operational profile and add to test effort compatible with real environment.

- Review of CR data

We review the CR data;conclude the following two possible major reasons for it. One is the existence of well-defined development methodology, which makes the programmer's skill to be independent of the implementation of the product. Another is that the developer's career over a certain degree hardly affects the number of faults. Also, we showed the duration of time for a solved software change request (SCR) as fellows



4. 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 faults 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.

A. Software failure intensities and priority of test

To view the reliability evolution, we use failure intensity, indicating the number of grouped failures per unit time (one month). Figure 4shows the relationship between all failures and the versions of the software during the period of system tests (the unit of x-axis is month).

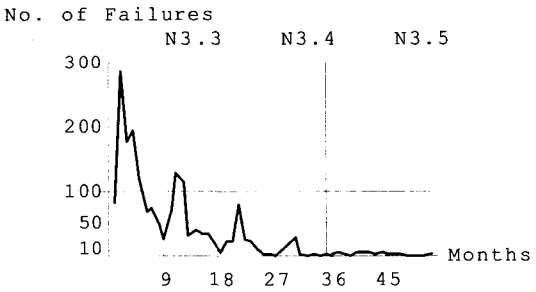


Fig. 4 Failure distributions to versions

Software version N3.3 is initial software package added to some of functions for our first demonstration version (N3.2). This package was attempted to commercial and field test. Also, N3.4 is a full package included all functions in our switching system. In this version, we performed integration test and system tests following the priority of test made reference to table 2. Last version (N3.5) is release of field for a in service. The priority of test is followingthe table 2 with functional category and each version.

Table 2: Priority of test

Versions	Priority
N3.3a	CP > DH > Ad > OM
N3.3b	Ad > OM > CP > DH
N3.4a	CP > Ad > OM
N3.4b	Ad > CP > OM

B. Failure distribution among categories

Table 3 shows the failure distribution among the categories mentioned in section 2.

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.

Table 3: Software Failure Distribution to categories

Version	CP	DH	Ad	OM	SK	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

In Table 3, 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.

C. 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 co-relations among functional blocks.

Table 4 is a statistical data indicating the number of modified blocks to correct software faults. The fault reduction factor estimated from Table 5 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 that are corrected by modifying 1 - 3 blocksis 94%. This 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. These functional blocks having many interactions are of fault-prone.

Table 4: Modified Number of Blocks and CRs

No. of Modified Blocks	CRs	Ratio (%)
1	1401	78
2	201	11
3	89	5
> 4	109	6

D. 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 5, 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.

Table 5: 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
SK	0.28 = 71/ 257K
Average	1.632 = 2219/1373K

E. Correction Time Distribution

The distribution of time required to correct faults is useful in predicting software maintenance effort during the field operation. In Table 6, 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 failures are caused due to control and assignment error within a block.

Table 6: 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(%)

5. Evolution of Software Reliability

The generalized form of software reliability model can be obtained if the distribution of failures is given.

Using the generalizedform, the software reliability growth research using the first stage of test occurred failures are to be stupid. (For example, selection of model and analysis of model used parameters, inclusion and deletion variable from model) So, general form is proposal to unity method of the research our project. Others, it is an easy distinction of software reliability growth models and compare of models.

A. Selection of model

In [11], we have experiencedthat our software reliability models fit G-O and S-Shaped model using a general form of software reliability growth models. As an application of generalized form, we classify three mentioned models according to the form of $b(t)$ and $f(N(t))$, Also, we present a case study applying the generalized form. $b(t)$ is the failure rate per software fault at time t. and $N(t)$ is the number remaining faults.

B. Evolution of two models

We analyze the software reliability growth models for the specified period from the viewpoint of theory of differential equations. We defined a generalized form of reliability models as fellows:

$$\frac{dN(t)}{dt} = b(t)f(N(t)) \qquad (1)$$

In other words, software reliability model shown that the reliability model represent differential equation between delta *N(t)* and *b(t)*. Also, we show the well-known software reliability growth models (such as NHPP models) G-O, S-Shaped model are special cases of the generalized form. Detailed of two generalized form of reliability model evolution is reference to appendix.

From (1), we also, extend the generalized form into an expended form being,

$$\frac{dN(t)}{dt} = b(t;\gamma)f(N(t)) \qquad (2)$$

γ : *test effort,* $0 < \gamma \leq 1$

- General form of software reliability growth model

In general hypothesis, the transformation of time ratio for remaining faults in software is a multiple of *N(t)* and *b(t)* until the time t. So, that function (1), we have called software reliability general model. In this function (1), *b(t)* is fault detection ratio and *F(N(t))* is extra function for variable of *N(t)*. Once again, software faults are detected and removed in software during the system test.

Rewrite of Function (1),

$$\frac{d\mu(t)}{dt} = -\frac{dN(t)}{dt} = b(t)f(N(t)) = b(t)f(N(0) - \mu(t))$$

$\mu(t)$: *cumulative fault (or failure),*
N(0) : remaining fault(before start of test)

time t, differential time ratio for the $\mu(t)$ is subordination between *b(t) and N(0),* $\mu(t)$.
N(0) = *a,* software reliability growth curve $\mu(t)$ decided to intermediary variable a and *b(t).*

- Expended form of software reliability

growth model

Software fault detection ratio is variable to found failures at system test. It described that fault detection ratio is consider of time and test-effort in system tests for fault detection. In case of variable to *b(t)* for some of parameters(eg. consumed of test time and test effort, resource, etc.), general form of software reliability differential function shown as fellows

$$\frac{dN(t)}{dt} = b(t;\gamma) f(N(t))$$

Function (2) is presented to parameter γ and we have called expended form of software reliability growth model.

Fault detection ratio put in weibull distribution,

$$b(t;\gamma) = -b\gamma t^{\gamma-1}, \gamma > 0 \quad (3)$$

in front of section B, applying to solved differential equation,

$$N(t,\gamma) = ae^{-bt^{\gamma}} \quad (4)$$

in case of $\gamma = 1$, $b(t, 1) = -b$,
N(t, 1) is same N(t) = ae-bt (meaning of remaining faults in software for G-O model)

C. Goodness of fit judgment

The results of NHPP model fitness areshown in Table 10. NHPP model mean function is

$$H(t) = B0 * (1 - e^{-B1t}) / 1 + PSI * e^{-B1t} \quad (5)$$

PSI: error detection efficiency

In inflection S-shaped model, PSI is ability of error detection by test team or tester for the test program (eg. using tool, experience of programming language, carrier, etc.) this model is well defined in [12], [13], [14].

The most widely used method for selection of best regression model is to examine iteratively some potential independent variables. That is referred to as a stepwise regression procedure. The details of this procedure are excellently discussed in [15], [16].

In Table 10, R-square is a general measure of the variance in the data explained by the model. It is a percentage between zero and 100. Traditionally, the R2 statistic is used almost exclusively empirical studies in software engineering. R2 is defined a ratio of sum of squares;

$$R^2 = \frac{SSR}{SST},$$

SSR: regression sum of squares for the fitted subset regression model with parameter
SST: the total sum squares

A modification of R2 statistic (*adjR2*) is the adjusted coefficient of multiple determinations, which does not attempt to correct for the parameters in regression model. This adjusted coefficient of multiple determinations,

$$adjR^2 = 1 - (\frac{n-1}{n-p}) \frac{SSE}{SST},$$

SSE: error sum of squares for fitted subset regression model with parameters
n: number of observation, p: use of parameters in model

The larger *R2* and *adjR2* are explains the variance of a dependent variable. Repeated, the largest R2corresponding to an exponential distribution of error count over time. This provides strong confirmatory evidence for the adequacy of NHPP model in describing error

occurrence in software. Therefore our project is suited to exponential model and provides a plausible characterization of error count over time. We applied to non-linear growth curve model using goodness of fit test statistics.[17]

- Corrected actual R-square (*Rc2*)

$$R_c^2 = 1 - \frac{SSE}{CSSA}$$

$$SSE = \sum_{t=1}^{N} (\hat{y}_t - y_t)^2, SSA = \sum_{t=1}^{N} y_t^2$$

$$CSSA = \sum_{t=1}^{N} y_t^2 - (\sum_{t=1}^{N} y_t)^2 = \sum_{t=1}^{N} (y_t - \bar{y})^2$$

(6)

y_t : *actual value,* $\hat{y}_t$: *predicted value*

SSA: sum of squares in actual value

$\bar{y}$: *actual mean*

In function (6), the coefficient of determinate is differ to linear regression model of coefficient.

- Corrected actual adjusted R-square (adj*Rc2*)

$$adjR_c^2 = \frac{(n-1)R_c^2 - k}{n-k-1} \quad (7)$$

$k = p-1$, *n: number of observation,*

p: use of parameters in mode (or predicted of population parameter)

The result data in Table 10 was determined by solving (5), (6), (7) using SAS (statistical analysis system).

D. Analysis results of data

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 (NHPP) models, in particular, the S-shaped [5],[6],[7] and Goel-Okumoto(G-O)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 that implies the assumption of the failure intensity depends on the tester's experience.

Recall that the cumulative failure count u(t) in the S-shaped model is given by

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

(8)

a : expected number of failures, b : failure rate

and the cumulative failure count of G-O model is

$$\mu_{GO}(t) = a(1-e^{-bt}), (a,b>0) \quad (9)$$

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 7, 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 7, as shown in the Fig. 4, we obtain the estimated values of parameters, a and b for the S-shaped model as in Table 8.

Table 7: 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 8: Estimation a, b in S-shaped model

Version No.	a	b
N3.3a	1118.14	0.552831
N3.3b	490.31	0.641667
N3.4a	188.59	0.697083
N3.4b	62.13	0.937313

In Fig. 5, we show the local behavior of the failure intensity growths of each version. The finite times that maximize the failure intensity are varied within 1-2 months in our cases.

Fig. 5 Local evolution of reliability growth

The total failure counts per version are shown in Table 9.

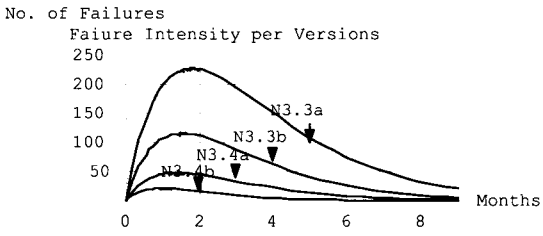


Table 9: Number of total failures of each version

Version	N3.3a	N3.3b	N3.4a	N3.4b	N3.5	field
Failures	1072	482	186	62	20	4

To getthe global software reliability growth pattern, we apply the G-O model to the failure data in Table 9. The maximum likelihood 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

$$\mu'_{GO}(t) = 1842.02 * 0.101597 e^{-0.101597t} \quad (10)$$

Using (10), we predicted the failure intensity to be 0.77452 during the next 9 monthsit is comparable to the real value 0.44444. Also, the cumulative failures of S-shaped model are shown in Fig. 6. InFig. 6, the real data (pot line) shows the contributions to the overall failure count arising from software errors related to development cycle (eg. During system test)

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. 7.

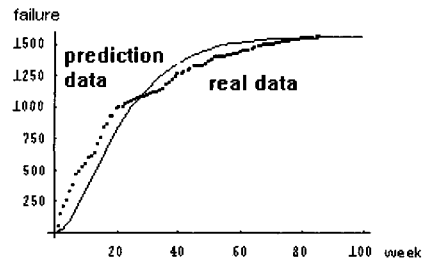


Fig. 6 Cumulative failures of S-Shaped model

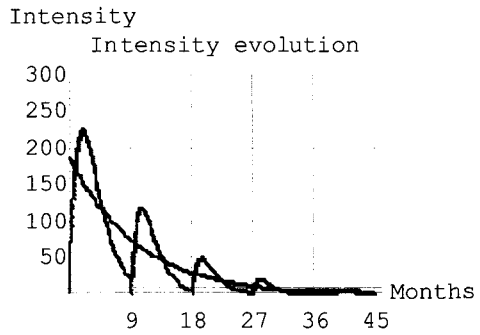


Fig. 7 Evolution of failure intensity

B1	0.160962	9.8586713953	0.15653150.0018738456	0.1653934

R-Square = 0.970937 $R_c^2 = 1$ SSE/CSE , R_c^2: coefficient of statistic measuring for non-linear model, SSE : the variance of the full model, CSE : the variance of the mean model

[N3.4]

Parameter	Estimate	Asymptotic std. Error	Asymptotic 95% Confidence interval	
			Lower	Upper
B0	559.2257800		Lower	Upper
B1	0.2340925	5.2404657926	546.83394804	571.61761201

R-Square = 0.976044 $R_c^2 = 1$ SSE/CSE , R^2: coefficient of statistic measuring for non-linear model, SSE : the variance of the full model, CSE : the variance of the mean model0.245158540.223026320.0046798135

[N3.5]

Table 10: Estimation of reliability for NHPP

A) Regression NHPP for each version

Version & model	Source	d.f	Sum of squares(s.s)	mean squares(m.s)	R_c^2	adj R_c^2
N3.3 (NHPP)	Regression	2	5814022.6978	2907011.3489	0.970937	0.970916
	Residual	7	26996.3022	3856.6146		
	Corrected total	8	928885.5556			
N3.4 (NHPP)	Regression	2	1247685.5250	623842.7625	0.976044	0.975947
	Residual	7	3666.4750	523.7821		
	Corrected total	8	153048.000			
N3.5 (NHPP)	Regression	2	203419.11053	101709.55527	0.906452	0.902554
	Residual	7	3018.88947	431.26992		
	Corrected total	8	32270.8889			

B) Model predictive quality (for each version)

[N3.3]

Parameter	Estimate	Asymptotic std. Error	Asymptotic 95% Confidence interval	
			Lower	Upper
B0	1460.502008		1437.1897676	1483.8142489

Parameter	Estimate	Asymptotic std. Error	Asymptotic 95% Confidence interval	
			Lower	Upper
B0	235.1020588	6.03947399937	220.82085745	249.38326010
B1	0.2144906	0.0111351381	0.18815992	0.24082118

R-Square = 0.906452 $R_c^2 = 1$ SSE/CSE , R^2: coefficient of statistic for non-linear model, SSE : the variance of the full model, CSE : the variance of the mean model

6. 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], [3], [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 new similar software by estimating the total software failures with respect to its size and time.

APPENDIX

A. Goel-Okumoto(G-O) model

Let, b(t) = -b (b : constant),

The software reliability growth model for specified period, we defined general form as followings differential equation (A-1)

$$\frac{dN(t)}{dt} = -bN(t) \qquad \text{(A-1)}$$

N(t) : remaining faults at time t

b(t) : failure rate at time t

function (A-1), expressed differential display,

$$\frac{dN(t)}{N(t)} = -bdt \qquad \text{(A-2)}$$

and integrated calculus of the function (A-2),

$$nN\ (t) = -\int_0^t bds\ +\ c \qquad \text{(A-3)}$$

c : integral constant

rewriting the function (A-3) for N(t),

$$N(t) = Ce^{-bt},$$

At this, on the assumption that initial condition is

N(0) = a, c = a

$$N(t) = ae^{-bt},$$

Then, we obtained following the result as

$$N(0) - N(t) = a - ae^{-bt} = a(1 - e^{-bt}) = \mu_{G-O}(t)$$

B. S-Shaped model

S-shaped model is proposedof Yamada et al. and fault detection ratio is not constant, it's time function.

Let, $b(t) = -\frac{b^2 t}{1+bt}$,

$$\frac{dN(t)}{dt} = -\frac{b^2 t}{1+bt} N(t) \qquad \text{(B-1)}$$

rewrite differential form (B-1)

$$\frac{dN(t)}{N(t)} = -\frac{b^2 t}{1+bt} dt \qquad \text{(B-2)}$$

Also, integrated calculus of the function (B-2),

$$\ln N(t) = -\int_0^t b \frac{bs}{1+bs} ds + c = -b\int_0^t (1 - \frac{1}{1+bs}) ds + c$$
$$= -bt + \ln(1+bt) + c$$

c : integral constant

so, $N(t) = Ce^{-bt} e^{\ln(1+bt)} = C(1+bt)e^{-bt}$

Let, N(0) = C = a, using $N(t) = a(1+bt)e^{-bt}$

we conformed as fellows

$$N(0) - N(t) = a(1-(1+bt)e^{-bt}) = \mu_s(t)$$

C. An analysis of reliability growth intermediary variable: parameter b(t)

Let, Probability distribution function F(t) for time t, probability of fault detection at (t, t+dt) is

$$b(t)dt = \frac{P[t<x<t+dt]}{P[x>t]} = \frac{F'(t)dt}{1-F(t)} \qquad \text{(C-1)}$$

at this, Fault detection hazard rate at time t,

$$b(t) = \frac{F'(t)}{1-F(t)} \qquad \text{(C-2)}$$

1) *b(t) in Goel-Okumoto model*

the fault density function f(x) is

$f(x) = be^{-bx}$ (<= *hypothesis, exponential density*)

using (C-2), confirmed $b(t) = \frac{be^{-bt}}{1-(1-e^{-bt})} = b$

2) *b(t) in S-Shaped model*

Let, probability density function f(x) is

$f(x) = b^2 xe^{-bx}$ (<= *hypothesis, gamma density*)

$$F(x) = (1 - bxe^{-bx} - e^{-bx}),$$

using (C-2), confirmed $b(t) = \frac{b^2 te^{-bt}}{bte^{-bt} + e^{-bt}} = \frac{b^2 t}{1+bt}$

others, in case of weibull distribution b(t),

density function is

$$f(t) = b\gamma t^{-\gamma-1} e^{-bt^{\gamma}},$$

the distribution function F(t) is

$$F(t) = 1 - e^{-bt^{\gamma}},$$

$$(\because b(t) = \frac{f(t)}{1-F(t)} = b\gamma t^{\gamma-1})$$

it's same function, section 5-B-(3)

References

[1] L. Bernstein, "Software in the Large'', AT & T Technical Journal, pp. 5-14, Jan/Feb. 1996.

[2] Kaaniche M., Kanoun K, "Reliability of a Commercial Telecommunications System'', ISSRE96, pp. 207-211, 1996.

[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, pp 120-143, Springer-Verlag, Berlin, 1984.

[6] Yamada. S, et al, "Software Reliability growth Model of two types of errors'', R.A.I.R.O. Operations Research, Vol. 19, No. 1, pp. 87-104, Feb. 1985.

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

[8] S. Y. Lee, An Application of Poisson Regression to a Switching Software failures'', ISSAT98, 1998.

[9] J.K. Lee, et al, "A study on hypothetical switching software through of the analysis of failure data", KICS journal, 98-8 Vol.23, No. 8, pp.1915-1925, Aug. 1998.

[10] J.K. Lee, et al, "An Examination of fault Exposure rate of switching software of TDX series from empirical failure data", IEEK journal, Vol. 36 S, No. 3, pp. 297-305, Mar. 1999.

[11] J.N. Yoo, "A generalized form of software reliability growth models", IEEK journal, Vol. 35 C, No. 5, pp. 331-336, May. 1998.

[12] M. Ohba, "Inflection S-shaped software reliability growth model", in stochastic models in Reliability Theory, S. Osaki and Y. Hotoyama (eds), pp. 144-162, Spring-Verlag, Berlin, 1984.

[13] S. Yamada, "Software reliability estimation technology Introduction of software reliability growth model", HBJ publishing integrated libraries No. 42, pp.101-115, 1989.

[14] S. Yamada, Hiroshi Ohtera, "Software reliability: theory and practical application", SE series software research center, pp. 135-173, Feb. 1990.

[15] N. draper and H. smith, "Applied regression analysis" 2nd edition, John Wiley and Sons, New York, 1981.

[16] J. Neter, W. wasserman and H. kurner, "Applied linear statistical models", IRWIN, Homewood, 1985.

[17] J.Y. HWNG, "A study on the Analysis procedures of non-linear growth curve models", journal of KSQM, Vol. 25 No. 1, pp. 44-55, Mar. 1997.

[18] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.s. Marron, Audris Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data", IEEE Transaction on software engineering, vol. 27, No. 1, pp. 1-12, JAN. 2001.

[19] Todd L. Graves, Alan F. Karr, J.s. Marron, and Harvey Siy, "Predicting fault incidence using software change history", IEEE Transaction on software engineering, vol. 28, No. 7, pp 653-661, July 2000.

이 재 기(Jae Ki Lee) 정회원

1985.2. 서울산업대학교 전자공학과 졸업

1989.5. 청주대학교 대학원 전자공학과 졸업

2002.3. - 공주대학교 대학원 전기전자정보통신공학과 박사과정

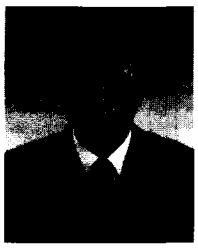
1983.3. - 현재 한국전자통신연구원 책임연구원

(주관심 분야) 소프트웨어 신뢰도, 시험 및 검증, 소프트웨어 품질 향상

남 상 식(Sang Sik Kim) 종신회원

1981. 2. 단국대학교 전자공학과 졸업(공학사)
1983. 2. 단국대학교 대학원 전자공학과 졸업(공학석사)
1999. 2. 단국대학교 대학원 전자공학과 졸업(공학박사)
1985. 3. ~ 현재 한국전자통신 연구원 책임연구원(팀장)
(주관심 분야) ATM Technology, Signal Integrity, NGN

김 창 봉(Chang-Bong Kim) 종신회원



1983년: 고려대학교 전자공학과(공학사)
1988년: Florida Tech 대학 전자공학과(공학석사)
1992년: Texas A&M 대학 전자공학과(공학박사)
1993년~현재: 공주대학교 정보통신공학과 교수
2000년~현재:미국전기전자공학회(IEEE) Senior Member
(주관심분야) 광통신망, 광전송