Reliability and Portability Assessment Tool Based on Hazard Rates for an Embedded Open Source Software

Yoshinobu Tamura
Yamaguchi University/Graduate School of Science and Engineering, Ube, Japan
Email: tamura@yamaguchi-u.ac.jp

Shigeru Yamada
Tottori Univerisity/Graduate School of Engineering, Tottori, Japan
Email: yamada@sse.tottori-u.ac.jp

***Abstract*—An embedded OSS (Open Source Software) known as one of OSS has been gaining a lot of attention in the embedded system area, i.e., Android, BusyBox, TRON, etc. However, the poor handling of quality problem and customer support prohibit the progress of embedded OSS. Therefore, many companies have been hesitant to innovate the embedded OSS because of OSS includes several software versions. Also, it is difficult for developers to assess reliability and portability of the porting-phase in case of installing the embedded OSS on a single-board computer. In this paper, we develop a method of software reliability/portability assessment tool based on a hazard rate model for the embedded OSS. Especially, we analyze actual software failure-occurrence time-interval data to show numerical examples of software reliability/portability assessment for the embedded OSS. Moreover, we show that our model and tool can assist quality improvement for embedded OSS systems development.**

***Index Terms*— reliability, portability, embedded system, open source software, software tool**

I. Introduction

There is growing interest in the next-generation software development and maintenance paradigm by using network computing technologies such as a cloud computing and SaaS (Software as a Service). Also, the software development paradigm based on an open source project is rapidly spreading because of the cost reduction, quick delivery, work saving. For successful example, OSS (Open Source Software) systems which serve as key components of critical infrastructures in the society are still ever-expanding now, i.e., Apache HTTP Server[1], MySQL[2], OpenStack[3], etc. The open source project contains special features so-called software composition by which several geographically-dispersed components are developed in all parts of the world. The successful experience of adopting such open source projects includes Apache HTTP server, Firefox Web browser[4], and GNU/Linux operating system. We focus on the problems in the software quality/reliability, which prohibit the progress of OSS.

Especially, software reliability growth models (SRGM's)[5], [6], [7], [8] and the related hazard rate models[9], [10], [11], [12] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective method of dynamic testing management for new distributed development paradigms as typified by the open source project has only a few presented[13], [14], [15]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for OSS, it is necessary to grasp the situation of registration for bug tracking system, the degree of maturity of OSS, and so on. Especially, an embedded OSS known as one of OSS's has been gaining a lot of attention in the embedded system area, i.e., Android[16], BusyBox[17], TRON, etc. However, the poor handling of quality problem and customer support prohibit the progress of embedded OSS. Also, it is difficult for developers to assess reliability and portability of the porting-phase in case of installing the embedded OSS on a single-board computer. The term "porting-phase" means the rebuilding process in which the developers create an OS/application developed for the specific computer system to suit another computer system in terms of portability. From above mentioned problems, many companies have been hesitant to innovate the embedded OSS.

In this paper, we propose a method of software reliability assessment based on a flexible hazard rate model for embedded OSS. Also, we derive several assessment measures. Especially, we also develop the software reliability/portability assessment tool for the porting-phase of embedded system development by using Java programming language. Then, we analyze actual software failure-occurrence time-interval data to show numerical examples of software reliability/portability analysis for the porting-phase of embedded system development based on the proposed tool. Moreover, we develop the tool considering optimal release problem based on a hazard rate model for the embedded open source software. Also, we add a new feature to our tool in order to compare our model with the existing models. Then, we show that the proposed tool can assist quality improvement for embedded OSS systems development. Furthermore, we investigate a useful software reliability assessment method for the actual open source system development.

II. Related Research

Many fault-counting type SRGM's have been applied to assess the reliability for quality management and testing-progress control of software development. However, it is difficult to apply the SRGM's for assessing quality/reliability of the OSS, because the number of detected faults in the OSS project can not converge to a finite value. In other words, for many OSS's, there are no testing phases for open source development paradigm, in which test personnel may detect and remove the faults as in a usual testing. In fact, there are several SRGM's that can be applied in the above situation, i.e., the Weibull and Log-logistic SRGM's, and so on[5]. Especially, in case that the number of detected faults can not converge to a finite value, it is difficult to assess whether the porting phase will succeed by using reliability assessment measures derived from SRGM's. As another more challenging aspect of the embedded OSS project, the embedded OSS includes several software components in terms of hardware such as device driver, firmware, updater[18], [19]. Also, several software reliability assessment tools have developed by several researchers, i.e., CASRE[20], AT&T SRE Toolkit, SoRel[21], etc. However, it is difficult to assess the porting phase of embedded system by using these software tools from above reasons.

Generally, OSS's of several versions are opened in the website of open source project. Therefore, it is difficult for the embedded software managers to decide the specific version used as the embedded system product. The software managers will be able to judge success and failure of the product planning in the porting phase, if the software managers can assess the reliability of the porting phase. Moreover, it is difficult for the software managers to judge success and failure of the porting phase quantitatively by using the existing SRGM's because of the structural problems of modeling. We summarize the solution approach of these problems as follows:

- We propose a plausible SRGM for the porting phase. The proposed model can assess the trend of reliability regression by using our model.
- We offer the reliability assessment tool for the porting phase. Thereby, the software managers can easily assess the reliability of porting phase without the mathematical knowledge.
- Our software tool can be estimate the optimum software release time. The optimum software release time will be useful for the software managers to minimize the development cost in the porting phase.

III. Hazard Rate Model for Embedded OSS

In this paper, we assume that the software faults detected at the porting-phase of embedded OSS include the following type:

A1. the software failure caused by the latent fault of embedded OSS

A2. the software failure caused by the latent fault of unique software components (i.e., device driver)

In the assumption above, A1 is selected by probability p and A2 selected by probability $(1-p)$. Also, we can not distinguish between assumption A1 and A2 in terms of the software faults. The time interval between successive faults of $(k-1)$-th and k-th is represented as the random variable X_k $(k=1,2,\cdots)$. Therefore, we can define the hazard rate function $z_k(x)$ for X_k as follows:

$$\begin{aligned} z_k(x) &= p\cdot z_k^1(x) + (1-p)\cdot z_k^2(x) \qquad (1)\\ &\quad (k=1,\ 2,\ \cdots;\ 0\le p\le 1),\\ z_k^1(x) &= D(1-\alpha\cdot e^{-\alpha k})^{k-1} \qquad (2)\\ &\quad (k=1,\ 2,\ \cdots;\ -1<\alpha<1,\ D>0),\\ z_k^2(x) &= \phi\{N-(k-1)\} \qquad (3)\\ &\quad (k=1,\ 2,\ \cdots,\ N;\ N>0,\ \phi>0), \end{aligned}$$

where we can define the each parameter as follows:

$z_k^1(x)$	:	the hazard rate for the assumption A1,
α	:	the shape parameter representing the active state of OSS project,
D	:	the initial hazard rate for the 1st software failure,
$z_k^2(x)$	:	the hazard rate for the assumption A2,
N	:	the number of latent faults in unique software components,
ϕ	:	the hazard rate per inherent fault,
p	:	the weight parameter for $z_k^1(x)$.

Eq.(2) means the hazard rate for a software failure-occurrence phenomenon for the embedded OSS. On the other hand, Eq.(3) represents the hazard rate for a software failure-occurrence phenomenon for the unique software components. Thus, our model simultaneously describes both the fault detected at the embedded OSS installed to embedded system by Eq.(2) and the fault detected at the unique software components such as the device driver[22].

In particular, our model includes both the modified Moranda model[11] and the conventional Jelinski–Moranda(J-M) model[10]. Eq.(2) based on the Moranda model means that the initial hazard rate for the 1st software failure geometrically decreases with the active state of OSS. Also, we assume that the active state of OSS grows exponentially.

IV. Reliability Assessment Measures

In porting-phase of embedded OSS, the distribution function of $X_k(k=1,\ 2,\ \cdots)$ representing the time-interval between successive faults of $(k-1)$th and k-th is defined as:

$$F_k(x) \equiv \Pr\{X_k \le x\} \qquad (x\ge 0), \qquad (4)$$

where $\Pr\{\mathrm{A}\}$ represents the occurrence probability of event A. Therefore, the following function means the probability density function of X_k:

$$f_k(x) \equiv \frac{dF_k(x)}{dx}. \quad (5)$$

Also, the software reliability can be defined as the probability which a software failure does not occur during the time-interval $(0,\ x]$ after the porting-phase. The software reliability is given by

$$R_k(x) \equiv \Pr\{X_k > x\} = 1 - F_k(x). \quad (6)$$

From Eqs.(4) and (5), the hazard rate is given by the following equation:

$$z_k(x) \equiv \frac{f_k(x)}{1 - F_k(x)} = \frac{f_k(x)}{R_k(x)}, \quad (7)$$

where the hazard rate means the software failure rate after the porting-phase when the software failure does not occur during the time-interval $(0,\ x]$.

Therefore, we can obtain the software reliability assessment measures from our hazard rate model represented by Eq.(1). The probability density function can be derived as

$$\begin{aligned} f_k(x) &= \Big\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \\ &+ (1-p)\phi(N-k+1)\Big\} \\ &\cdot \exp\Big[-\Big\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \\ &+ (1-p)\phi(N-k+1)\Big\}\cdot x\Big]. \quad (8) \end{aligned}$$

Especially, we can give the following expressions as software reliability assessment measures derived from our hazard rate model:

- *MTBF*
  The mean time between software failures(MTBF) is useful to measure the property of the frequency of software failure-occurrence, and is given by

$$\begin{aligned} \mathrm{E}[X_k] &= 1 \Big/ \Big\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \\ &+ (1-p)\phi(N-k+1)\Big\}. \quad (9) \end{aligned}$$

- *Software reliability*
  Also, the software reliability can be defined as the probability which a software failure does not occur during the time-interval $(t, t+x]$ $(t \geq 0, x \geq 0)$ given that the testing-time of porting-phase is t. The software reliability is given as follows:

$$\begin{aligned} R_k(x) &= \exp\Big[-\Big\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \\ &+ (1-p)\phi(N-k+1)\Big\}\cdot x\Big]. \quad (10) \end{aligned}$$

- *Porting stability*
  Moreover, we can estimate the porting stability from our hazard rate model. We define the porting stability as the value of model parameter w_1.
- *MSE*
  The MSE (mean square error) can be obtained by dividing the sum of square errors between the observed value, y_k, and its estimated one, $\hat{y}_k$, by the number of data pairs; n. That is,

$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y}_k)^2. \quad (11)$$

$\hat{y}_k$ $(k = 1, 2, \cdots, n)$ in Eq. (11) is obtained from the estimated value. The mean square error indicates that the selected model fits better to the observed data as MSE becomes small. We compare the proposed flexible hazard rate model for embedded OSS with the following typical conventional hazard rate models:

for Moranda model:

$$\begin{aligned} z_k(x) &= Dc^{k-1} \\ &\quad (D > 0,\ 0 < c < 1;\ k = 1,\ 2,\ \cdots), \quad (12) \\ \mathrm{E}[X_k] &= \frac{1}{Dc^{k-1}}. \quad (13) \end{aligned}$$

for Jelinski–Moranda(J-M) model:

$$\begin{aligned} z_k(x) &= \phi(N-k+1) \\ &\quad (N > 0,\ \phi > 0;\ k = 1,\ 2,\ \cdots,\ N), \quad (14) \\ \mathrm{E}[X_k] &= \frac{1}{\phi(N-k+1)}. \quad (15) \end{aligned}$$

The model parameters in Eq. (12) and (14) are defined as follows:

D :	the initial hazard rate for 1st software failure,
c :	the reduction factor of hazard rate,
ϕ :	the hazard rate per remaining fault,
N :	the latent fault in software system.

- *Predicted relative error*
  Furthermore, we adopt the value of the predicted relative error as comparison criteria of goodness-of-fit in our tool. The predicted relative error is defined as a relative error between the predicted and observed values of the software failure-occurrence time-interval. It is given by

$$R_p = \frac{\hat{A}(t_q) - q}{q}, \quad (16)$$

where t_q is the termination time of the porting and q is the observed number of faults detected in the time-interval $[0, t_q)$. $\hat{A}(t_q)$ in Eq. (16) is the estimated value of the MTBF at the termination time t_q where $A(t)$ is estimated by using the actual data observed up to arbitrary porting time $t_p (0 \leq t_p \leq t_q)$.

V. Laplace Trend Test

Also, we use the Laplace trend test[18], [23] of the data set to determine which hazard rate models are useful to investigate. In case of the fault detection time-interval data, the Laplace trend test statistic $u(i)$ is given by the

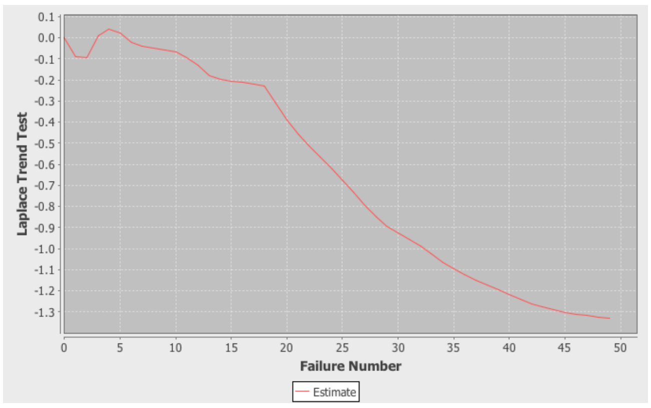


Figure 3. The estimated results of the Laplace trend test in case of Android 1.0 SDK.

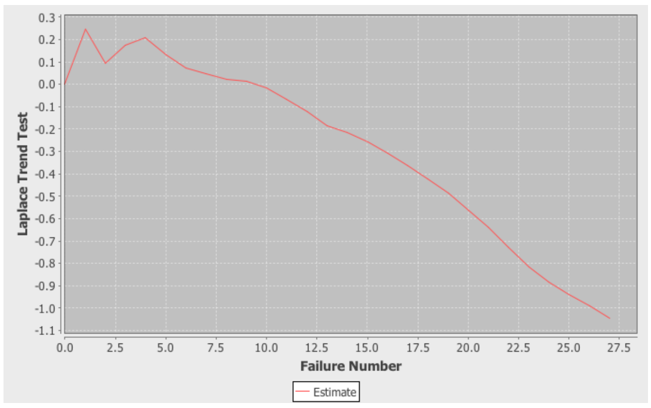


Figure 5. The estimated results of the Laplace trend test in case of Android 1.5 NDK.

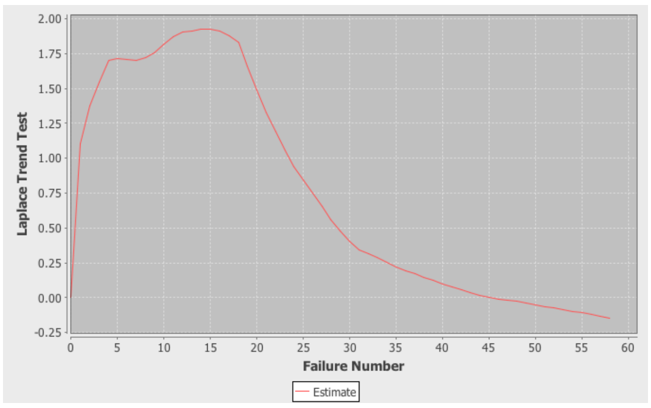


Figure 4. The estimated results of the Laplace trend test in case of Android 1.1 SDK.

following equation[18], [23].

$$u(i) = \frac{\frac{1}{i-1}\sum_{n=1}^{i-1}\sum_{j=1}^{n}\theta_j - \frac{\sum_{j=1}^{i}\theta_j}{2}}{\sum_{j=1}^{i}\theta_j\sqrt{\frac{1}{12(i-1)}}}, \tag{17}$$

where i is the failure number, θ_j means the j-th the fault detection time-interval.

Especially, we focus on the embedded OSS in order to evaluate the performance of our tool, i.e., Android[16] and BusyBox[17]. BusyBox includes 4 components. In this paper, we show numerical examples of the Android OS developed for mobile phone. Also, we consider the case of installing BusyBox to Android as the porting environment. We use the data collected from the following versions as Figures 1 and 2.

- 2008 - Android 1.0 SDK Release 1 available (first actual 1.0-compatible SDK)
- Android 1.1 SDK, Release 1
- Android 1.5 NDK, Release 1

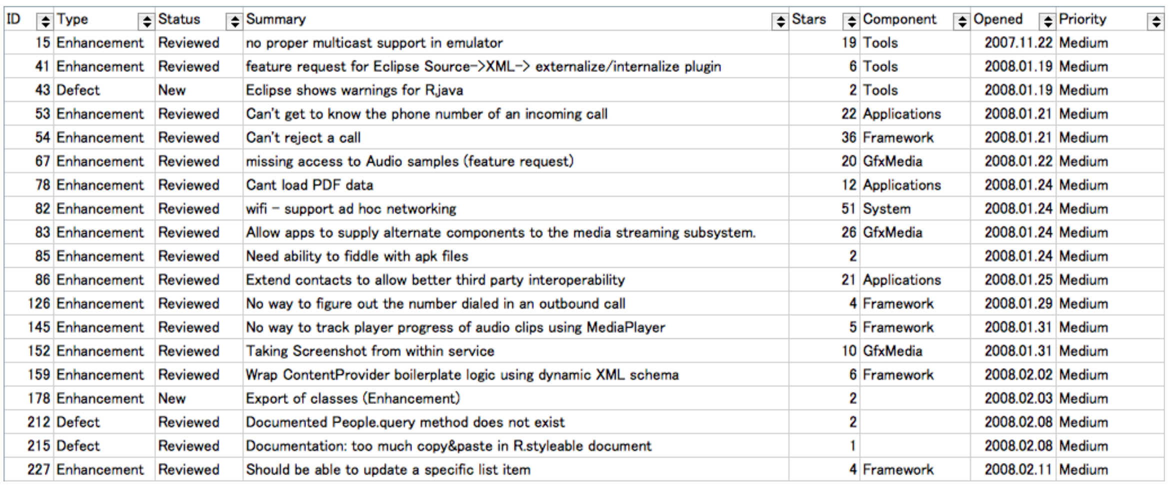
We investigate the reliability regression and growth trends by using the Laplace trend test. The estimation results of Laplace trend test for each version are shown from Figure 3 to Figure 5. First, the Laplace trend test in case of Android 1.0 SDK shows a reliability growth trend in all area from Figure 3. Figure 3 means that the embedded software has been thoroughly-tested because of the first version Android 1.0 SDK as the introduced and retail version. Second, the Laplace trend test in case of Android 1.1 SDK shows a reliability regression trend in all area from Figure 4. We found that Figure 4 includes both the reliability regression trend. Especially, Figure 4 means that the fault report is very heavy on the bug tracking system because of the second version Android 1.1 SDK as the widespread utilization version. This implies that it is difficult to apply the conventional hazard rate models. Moreover, the Laplace trend test in case of Android 1.5 NDK shows a reliability regression trend in all area before the failure No. 10 from Figure 5. We found that Figure 5 includes both the reliability regression and growth trend because of the stable version Android 1.5 NDK. Above mentioned results, it is difficult for the software developers to assess the reliability of the embedded software porting phase.

We show the performance illustrations of our tool in Section VIII by using the data sets of the above mentioned two versions in terms of Android 1.0 SDK and Android 1.5 NDK. Then, we confirm that our tool can assess the reliability of the porting phase by using the other data set. Also, our tool can comprehend both the trends of reliability growth and retrogression.

VI. Optimal Release Problem for the Porting-Phase

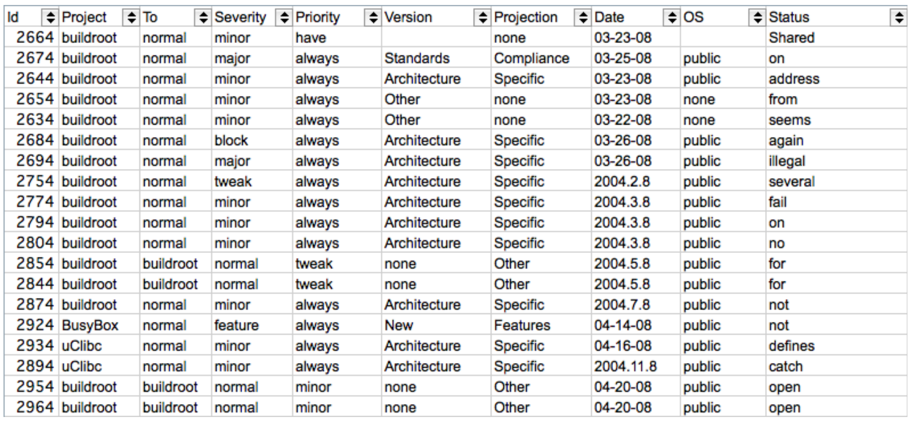
We find the optimal release time of porting-phase by minimizing the total expected software cost in this section. Then, we discuss about the determination of optimal software release times minimizing the total expected software cost[24]. We introduce the following cost parameters:

c_1	the testing cost per porting-time ($c_1 > 0$),
c_2	the fixing cost per fault during the porting-phase ($c_2 > 0$),



ID	Type	Status	Summary	Stars	Component	Opened	Priority
15	Enhancement	Reviewed	no proper multicast support in emulator	19	Tools	2007.11.22	Medium
41	Enhancement	Reviewed	feature request for Eclipse Source->XML-> externalize/internalize plugin	6	Tools	2008.01.19	Medium
43	Defect	New	Eclipse shows warnings for R.java	2	Tools	2008.01.19	Medium
53	Enhancement	Reviewed	Can't get to know the phone number of an incoming call	22	Applications	2008.01.21	Medium
54	Enhancement	Reviewed	Can't reject a call	36	Framework	2008.01.21	Medium
67	Enhancement	Reviewed	missing access to Audio samples (feature request)	20	GfxMedia	2008.01.22	Medium
78	Enhancement	Reviewed	Cant load PDF data	12	Applications	2008.01.24	Medium
82	Enhancement	Reviewed	wifi – support ad hoc networking	51	System	2008.01.24	Medium
83	Enhancement	Reviewed	Allow apps to supply alternate components to the media streaming subsystem.	26	GfxMedia	2008.01.24	Medium
85	Enhancement	Reviewed	Need ability to fiddle with apk files	2		2008.01.24	Medium
86	Enhancement	Reviewed	Extend contacts to allow better third party interoperability	21	Applications	2008.01.25	Medium
126	Enhancement	Reviewed	No way to figure out the number dialed in an outbound call	4	Framework	2008.01.29	Medium
145	Enhancement	Reviewed	No way to track player progress of audio clips using MediaPlayer	5	Framework	2008.01.31	Medium
152	Enhancement	Reviewed	Taking Screenshot from within service	10	GfxMedia	2008.01.31	Medium
159	Enhancement	Reviewed	Wrap ContentProvider boilerplate logic using dynamic XML schema	6	Framework	2008.02.02	Medium
178	Enhancement	New	Export of classes (Enhancement)	2		2008.02.03	Medium
212	Defect	Reviewed	Documented People.query method does not exist	2		2008.02.08	Medium
215	Defect	Reviewed	Documentation: too much copy&paste in R.styleable document	1		2008.02.08	Medium
227	Enhancement	Reviewed	Should be able to update a specific list item	4	Framework	2008.02.11	Medium

Figure 1. The partial source data in Android.



Id	Project	To	Severity	Priority	Version	Projection	Date	OS	Status
2664	buildroot	normal	minor	have		none	03-23-08		Shared
2674	buildroot	normal	major	always	Standards	Compliance	03-25-08	public	on
2644	buildroot	normal	minor	always	Architecture	Specific	03-23-08	public	address
2654	buildroot	normal	minor	always	Other	none	03-23-08	none	from
2634	buildroot	normal	minor	always	Other	none	03-22-08	none	seems
2684	buildroot	normal	block	always	Architecture	Specific	03-26-08	public	again
2694	buildroot	normal	major	always	Architecture	Specific	03-26-08	public	illegal
2754	buildroot	normal	tweak	always	Architecture	Specific	2004.2.8	public	several
2774	buildroot	normal	minor	always	Architecture	Specific	2004.3.8	public	fail
2794	buildroot	normal	minor	always	Architecture	Specific	2004.3.8	public	on
2804	buildroot	normal	minor	always	Architecture	Specific	2004.3.8	public	no
2854	buildroot	buildroot	normal	tweak	none	Other	2004.5.8	public	for
2844	buildroot	buildroot	normal	tweak	none	Other	2004.5.8	public	for
2874	buildroot	normal	minor	always	Architecture	Specific	2004.7.8	public	not
2924	BusyBox	normal	feature	always	New	Features	04-14-08	public	not
2934	uClibc	normal	minor	always	Architecture	Specific	04-16-08	public	defines
2894	uClibc	normal	minor	always	Architecture	Specific	2004.11.8	public	catch
2954	buildroot	buildroot	normal	minor	none	Other	04-20-08	public	open
2964	buildroot	buildroot	normal	minor	none	Other	04-20-08	public	open

Figure 2. The partial source data in BusyBox.

c_3 the fixing cost per fault after the release $(c_3 > c_2)$.

Then, the expected software cost of OSS can be formulated as:

$$C_1(l) = c_1 \sum_{k=1}^{l} \mathrm{E}[X_k] + c_2 l, \tag{18}$$

where l is the number of software failure-occurrence.

Also, we can define the expected software cost for software components as follows:

$$C_2(l) = c_3 (N - l) \,. \tag{19}$$

Consequently, from Eqs. (18) and (19), the total expected software cost is given by

$$C(l) = C_1(l) + C_2(l). \tag{20}$$

From l^* obtained by minimizing l, we can estimate the optimum software release time $\sum_{k=1}^{l^*} \mathrm{E}[X_k]$.

Therefore, we can give the following expressions such as software release time derived from our hazard rate model:

- *Optimum software release time*
  Moreover, we can estimate the optimal software release time minimizing the expected total software cost based on our hazard rate model.
- *Total expected software cost*
  We can estimate the total expected software cost in case of the estimated optimum software release time $\sum_{k=1}^{l^*} \mathrm{E}[X_k]$.

VII. Reliability/Portability Assessment Tool

A. Specification requirement

The specification requirements of the reliability/portability assessment tool for embedded OSS are shown as follows:

1. This tool should be operated by clicking the mouse button and typing on the keyboard to input the data through GUI system.
2. An object-oriented language, Java, should be used to implement the program. This tool is developed as a stand-alone application on Windows[1], Unix[2], and Macintosh[3] operating system.

[1]Windows is a registered trademark licensed to Microsoft Corp.
[2]Unix is a registered trademark licensed to the Open group.
[3]Macintosh is a trademark of Macintosh Laboratory, Inc. licensed to Apple Computer, Inc.

3. This tool treats the hazard rate model for embedded OSS, and illustrate the MTBF, software reliability, porting stability, and predicted relative error as software reliability/portability assessment measures.
4. In case of the fault detection time-interval data, the Laplace trend test statistic is illustrated.
5. This tool calculate the MSE for several existing models.

B. *Software reliability assessment procedures*

The procedures of reliability/portability assessment built into the proposed tool for embedded OSS are shown as follows:

1. This tool processes the data file in terms of the software failure-occurrence time-interval data in the porting-phase of the embedded system for reliability/portability assessment.
2. Using the data obtained from the porting-phase, we analyze the data for input data.
3. This tool estimates the unknown parameters included in our hazard rate model. We assume as $w_1 = pD$ and $w_2 = (1-p)\phi$ for the simplification technique. Moreover, we define w_1 and w_2 as the scale of the porting stability.
4. This tool illustrates the MTBF, software reliability, porting stability, and predicted relative error as software reliability/portability assessment measures.
5. This tool illustrates the Laplace trend test statistic of the data set.
6. We focus on optimal software release problems based on our hazard rate model for the porting-phase. Especially, the expected total software cost and the optimal software release time minimizing the cost for our model are plotted on the CRT.

This tool is composed of several function components such as fault analysis, estimation of unknown parameters, goodness-of-fit test for the estimated model, graphical representation of fault data, and results of estimation. The structure of reliability/portability assessment tool for embedded OSS is shown in Figure 6. Moreover, we show the activity diagram and the sequence diagram of our tool in Figure 7.

VIII. Performance Illustrations

In this section, we analyze a set of actual software failure-occurrence time-interval data to show performance illustrations of software reliability/portability measurement for application of our tool[4]. We show numerical examples for reliability/portability assessment of the porting-phase of embedded system development by using embedded OSS. Moreover, we illustrate the estimation results of our tool by using the data sets assumed the porting environment such as Section V. Considering the realities of the field use, we show the performance illustrations of our tool by using the data sets in terms of "Android 1.0 SDK Release 1" and "Android 1.5 NDK".

[4]Reliability/Portability Assessment Tool for Embedded OSS (RPAT for Embedded OSS), URL: http://sourceforge.net/projects/rpatforeoss/

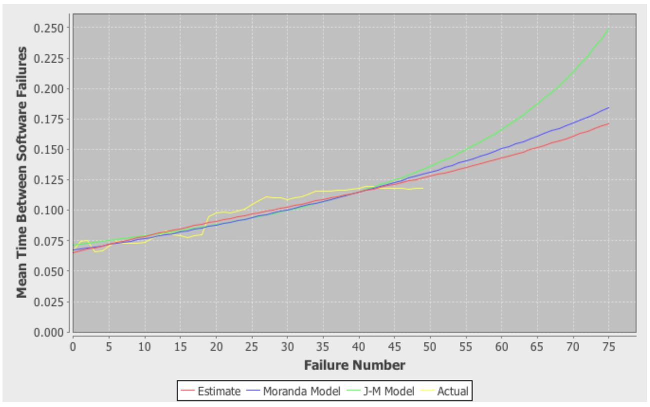


Figure 9. The estimated MTBF in case of Android 1.5 NDK.

A. *Reliability assessment results*

The estimated results of the unknown parameters of our hazard rate model are shown in Figure 8 in case of Android 1.5 NDK. Also, the estimated MTBF in Eq. (9) is shown in Figure 9 in case of Android 1.5 NDK. The horizontal axis of Figure 9 means the porting time k. From Figure 9, we can confirm that the MTBF grows as porting procedures go on. Moreover, Figure 10 shows the estimated software reliability in case of Android 1.5 NDK. The horizontal axis of Figure 10 means the elapsed time after the end of the detected faults $k = 50$. From Figure 10, if the embedded system is operated after the under the same strict conditions as the testing phase after the 50 faults at the time of Jun. 1, 2009, the software reliability after 12 hours after from the beginning of its operation shows about 0.5. Therefore, we found that at least one software failure may occur with the probability of 50% within 12 hours. Next, Figure 11 shows the behaviors of the predicted relative error in case of Android 1.5 NDK. As shown in Figure 11, the variation of the model becomes stable when the porting progress ratio exceeds 80%.

Similarly, we illustrate several reliability assessment measures of Android 1.0 SDK in Figures 12$\sim$14. From Figures 12$\sim$14, we found that our tool can be used for the other data. Especially, Figures 11 and 14 mean that the estimated predicted relative errors in case of two versions equal to or lower than 0.15 through all porting progress ratio. Therefore, we found that our tool can be used regardless of the difference in the data set from Figures 11 and 14.

B. *Optimal release problem with reliability of embedded OSS*

Also, we consider that the developer of embedded system can estimate the optimal release time with considering reliability by combining the total expected software cost with the reliability. We assume that the reliability requirement as follows:

$$R_k(0.1) = 0.5.$$

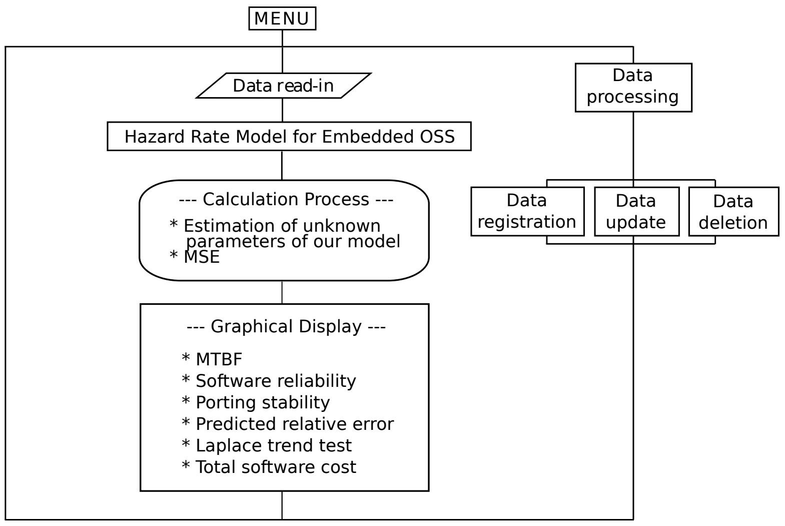


Figure 6. The structure of reliability/portability assessment tool for embedded OSS.

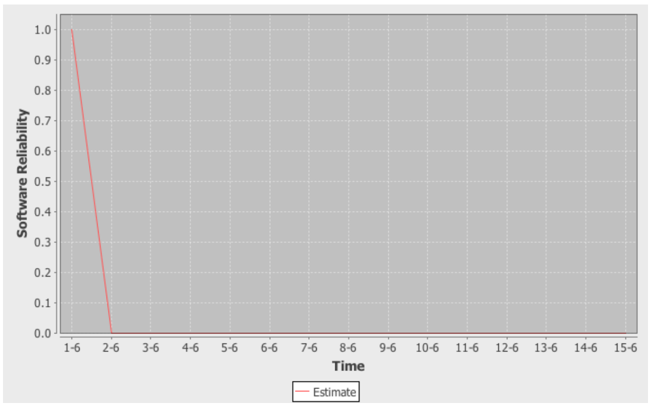


Figure 10. The estimated software reliability in case of Android 1.5 NDK.

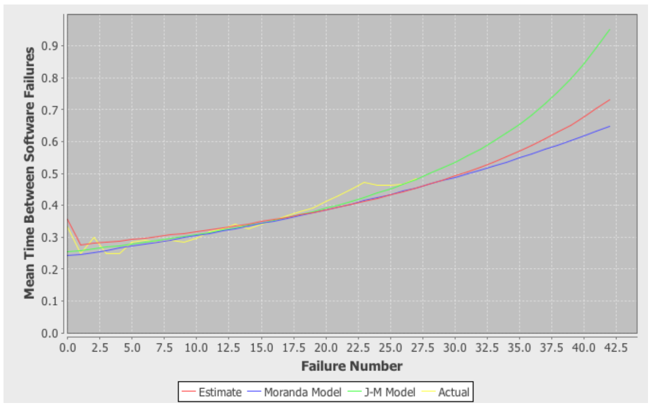


Figure 12. The estimated MTBF in case of Android 1.0 SDK.

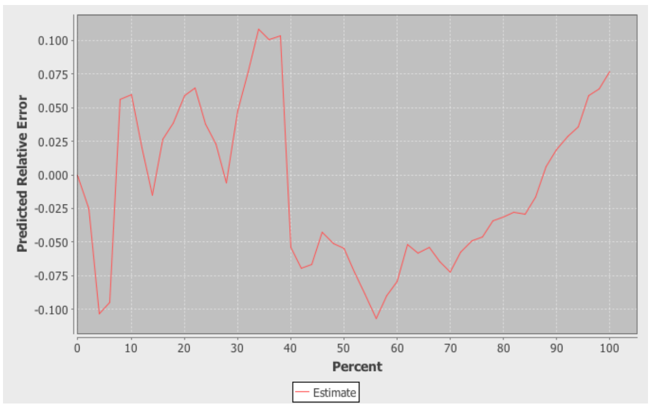


Figure 11. The estimated predicted relative errors in case of Android 1.5 NDK.

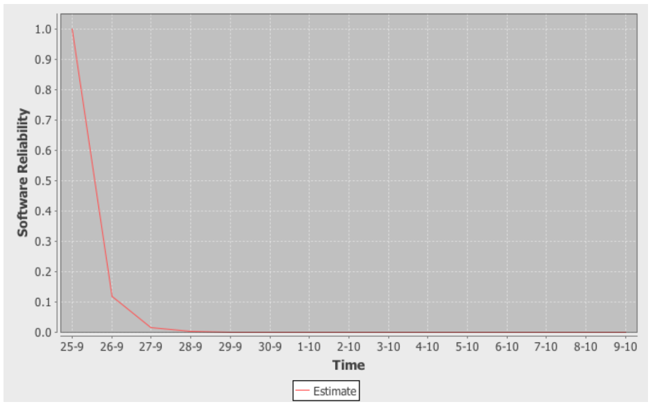


Figure 13. The estimated software reliability in case of Android 1.0 SDK.

We obtain the reliability $R_{51}(0.1) = 0.462$ when the optimum release time is $t' = 6.6$ days where $l' = 51$. Then, we need to lengthen the porting-time, which is shown in Figure 15 as $(t' - t^*) = 8.8 - 6.6 = 2.2$ days. Figure 15 illustrates the optimum release time with reliability objective $R_{61}(0.1) = 0.5$. From Figure 15, we have found that the optimum release time become lengthen, and the total expected software cost increases. When the porting time and the reliability objective are assumed as $R_k(0.1) = 0.5$, we obtain $t'^* = 8.8$ from $l^* = 61$. Then, the total expected software cost is 1502.8.

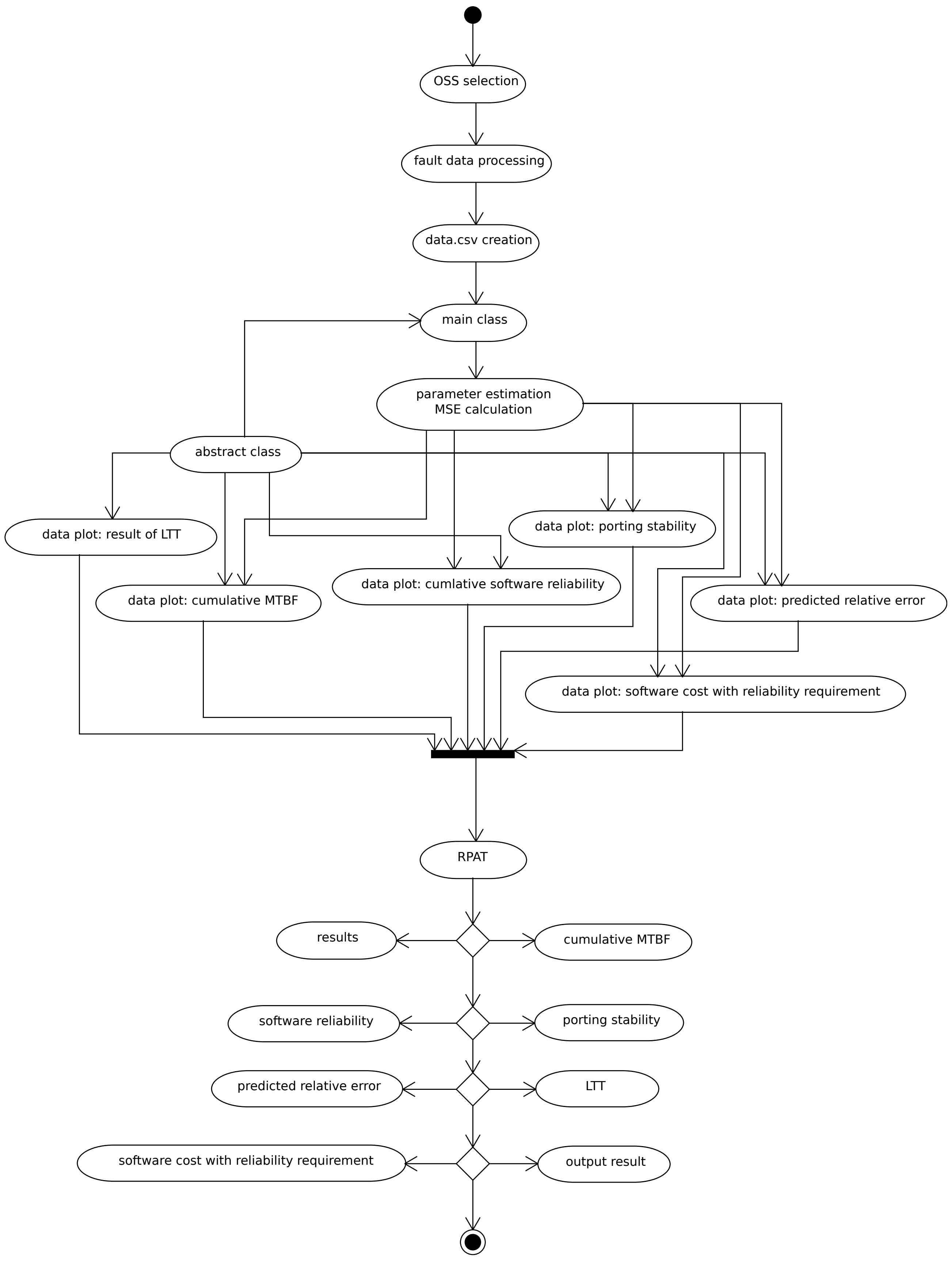
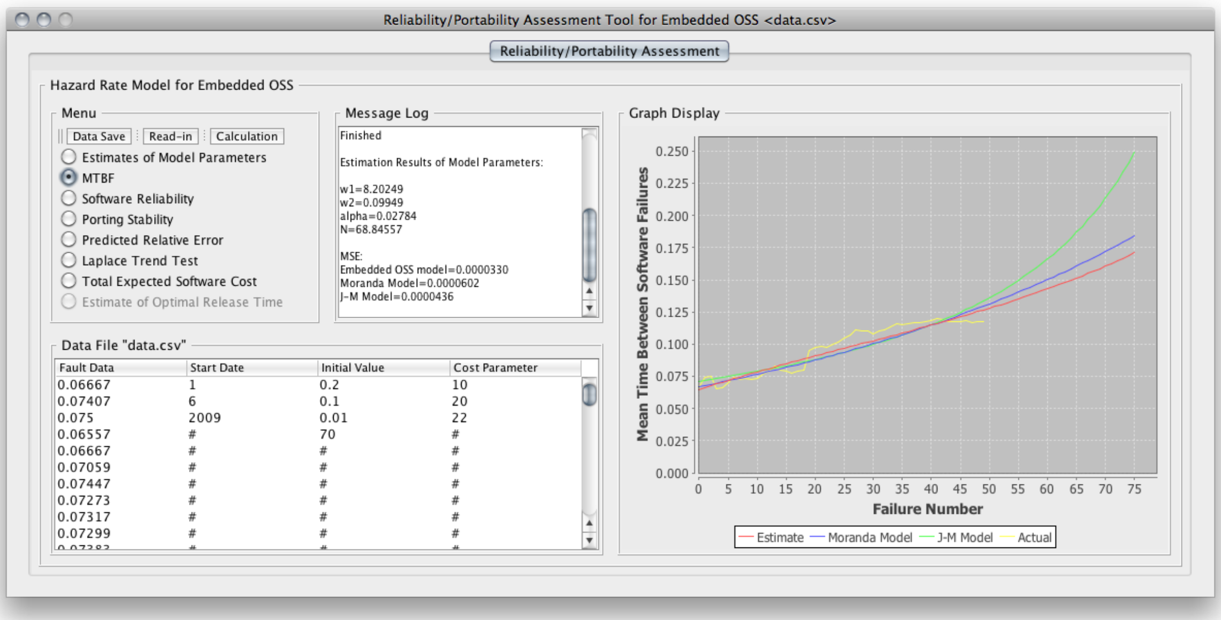


Figure 7. The activity diagram of reliability/portability assessment tool for embedded OSS.


Reliability/Portability Assessment Tool for Embedded OSS <data.csv>
Reliability/Portability Assessment
Hazard Rate Model for Embedded OSS
Menu
Data Save
Read-in
Calculation
Estimates of Model Parameters
MTBF
Software Reliability
Porting Stability
Predicted Relative Error
Laplace Trend Test
Total Expected Software Cost
Estimate of Optimal Release Time
Message Log
Data File "data.csv"
Graph Display
Mean Time Between Software Failures
Failure Number
Estimate
Moranda Model
J-M Model
Actual

( ) Estimates of Model Parameters
[✓ MTBF]
( ) Software Reliability
( ) Porting Stability
( ) Predicted Relative Error
( ) Laplace Trend Test
( ) Total Expected Software Cost
( ) Estimate of Optimal Release Time

```
Finished

Estimation Results of Model Parameters:

w1=8.20249
w2=0.09949
alpha=0.02784
N=68.84557

MSE:
Embedded OSS model=0.0000330
Moranda Model=0.0000602
J-M Model=0.0000436
```

Fault Data	Start Date	Initial Value	Cost Parameter
0.06667	1	0.2	10
0.07407	6	0.1	20
0.075	2009	0.01	22
0.06557	#	70	#
0.06667	#	#	#
0.07059	#	#	#
0.07447	#	#	#
0.07273	#	#	#
0.07317	#	#	#
0.07299	#	#	#

Figure 8. The estimated results of the model parameters in case of Android 1.5 NDK.

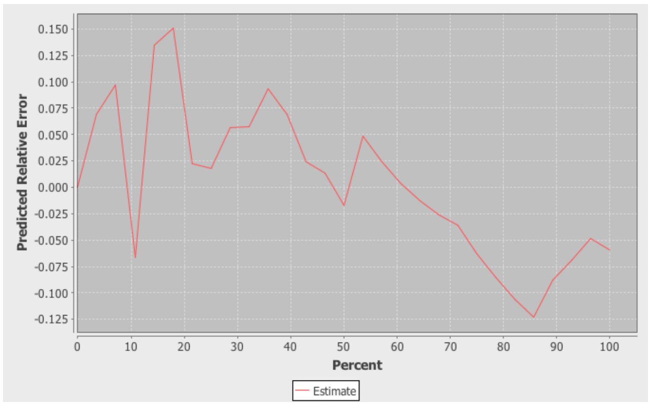


Figure 14. The estimated predicted relative errors in case of Android 1.0 SDK.

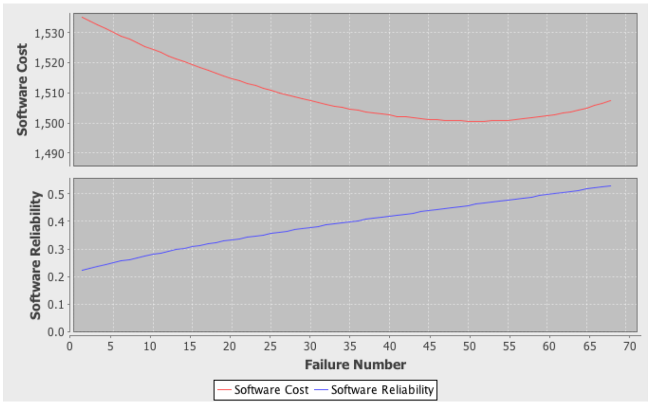


Figure 15. The optimum software release time with reliability requirement, t'^* in case of Android 1.5 NDK.

IX. Concluding Remarks

It is important for software developers to control the porting-phase in embedded system development by using software reliability/portability assessment tool without knowing the details of the process of the faults data analysis. In this paper, we have proposed a software reliability/portability assessment tool based on the hazard rate model for embedded system development by using Java programming language.

In this paper, we have shown that our method can grasp both the embedded OSS and the unique software components such as the device driver. Additionally, we have presented several performance illustrations for the actual data. Moreover, it is useful for embedded software developers to understand the debugging progress in porting-phase of embedded system development by using a software reliability/portability assessment tool without knowing the details of the process of the fault data analysis. Furthermore, our tool proposed here is useful for embedded system developers in terms of the management of the debugging process in the porting-phase of embedded system development. The results of data analysis are represented simply and visually by GUI and this tool prepares the expandability, portability, and maintainability by using Java.

The proposed model is used as a simple combination technique of two exponential models by using the reproducing property. In particular, the proposed model is assumed for reliability assessment in unusual circumstances such as the OSS porting phase. However, the readers can describe another combination models by using some other models. Moreover, the embedded system developers can modify our tool.

Finally, we have focused on the reliability/portability under the porting-phase of an embedded system development by using the embedded OSS. Distributed development environment typified by such embedded OSS will evolve at a rapid pace in the future. Our reliability/portability assessment tool is useful as a method of reliability/portability assessment to solve the problems which many companies have been hesitant to innovate the embedded OSS.

ACKNOWLEDGMENTS

This work was supported in part by the Grant-in-Aid for Scientific Research (C), Grant No. 24500066 and No. 25350445 from the Ministry of Education, Culture, Sports, Science, and Technology of Japan.

REFERENCES

[1] The Apache HTTP Server Project, The Apache Software Foundation, 2013, http://httpd.apache.org/
[2] Oracle Corporation and/or Its Affiliates, MySQL, 2013, http://mysql.com/
[3] The OpenStack project, OpenStack, 2013, http://www.openstack.org/
[4] Mozilla.org, Mozilla Foundation, 2013 http://www.mozilla.org/
[5] M.R. Lyu, ed. *Handbook of Software Reliability Engineering*, Los Alamitos, CA: IEEE Computer Society Press, 1996.
[6] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, New York: McGraw-Hill, 1987.
[7] S. Yamada, *Software Reliability Modeling: Fundamentals and Applications*, Tokyo/Heidelberg: Springer–Verlag, 2013.
[8] K.Y. Cai, D.B. Hu, C. Bai, H. Hu, and T. Jing, Does software reliability growth behavior follow a non-homogeneous Poisson process, *Information & Software Technology*, vol. 50, no. 12, pp. 1232-1247, 2008.
[9] G.J. Schick and R.W. Wolverton, An Analysis of Competing Software Reliability Models, *IEEE Transactions Software Engineering*, vol. SE-4, no. 2, pp. 104–120, 1978.
[10] Z. Jelinski and P.B. Moranda, *Software Reliability Research, in Statistical Computer Performance Evaluation*, Freiberger, pp. 465–484, New York: Academic Press, 1972.
[11] P.B. Moranda, Event–altered Rate Models for General Reliability Analysis, *IEEE Transactions Reliability*, vol. R–28, no. 5, pp. 376–381, 1979.
[12] M. Xie, On a Generalization of the J-M Model, *Proceedings Reliability '89*, 5, Ba/3/1–5 Ba/3/7, 1989.
[13] Y. Zhou and J. Davis, "Open source software reliability model: an empirical approach,"*Proceedings of the Fifth Workshop on Open Source Software Engineering*, pp.67–72, 2005.
[14] P. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, *Proceeding of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)*, pp. 263–272, 2004.
[15] J. Norris, Mission-critical development with open source software, *IEEE Software Magazine*, vol. 21, no. 1, pp. 42–49, 2004.
[16] Open Handset Alliance, Android, 2013, http://www.android.com/
[17] Erik Andersen, BUSYBOX, 2013, http://www.busybox.net/
[18] V. Almering, M.V. Genuchten, G. Cloudt, and P.J.M. Sonnemants, Using software reliability growth models in practice, *IEEE Software*, pp. 82–88, 2007.
[19] Y. Tamura and S. Yamada, Reliability analysis methods for an open source software with their comparison of goodness-of-fit, *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 208–212, 2009.
[20] Open Channel Foundation, CASRE 3.0, 2011, http://www.openchannelsoftware.com/projects/CASRE_3.0
[21] Karama Kanoun, SoRel, 2011, http://homepages.laas.fr/surf4tst/sorel/sorel.html
[22] Y. Tamura and S. Yamada, Reliability assessment based on hazard rate model for an embedded OSS porting phase, *Journal of Software Testing, Verification and Reliability*, vol. 23, no. 1, pp. 77–88, 2013.
[23] P.A. Keiler and T.A. Mazzuchi, Enhancing the predictive performance of the Goel-Okumoto software reliability growth model, *Proceedings Annual Reliability and Maintainability Symposium*, IEEE Press, pp. 106–112, 2000.
[24] S. Yamada and S. Osaki, Cost-reliability optimal software release policies for software systems, *IEEE Transactions on Reliability*, vol. 34, no. 5, pp. 422–424, 1985.

Yoshinobu Tamura received the B.S.E., M.S., and Ph.D. degrees from Tottori University in 1998, 2000, and 2003, respectively. From 2003 to 2006, he was a Research Assistant at Tottori University of Environmental Studies. From 2006 to 2009, he was a Lecturer and Associate Professor at Faculty of Applied Information Science of Hiroshima Institute of Technology, Hiroshima, Japan. Since 2009, he has been working as a Associate Professor at the Graduate School of Science and Engineering, Yamaguchi University, Ube, Japan. His research interests include reliability assessment for open source software. He is a regular member of the Institute of Electronics, Information and Communication Engineers of Japan, the Information Processing Society of Japan, the Operations Research Society of Japan, the Society of Project Management of Japan, and the IEEE. Dr. Tamura received the Presentation Award of the Seventh International Conference on Industrial Management in 2004, the IEEE Reliability Society Japan Chapter Awards in 2007, the Research Leadership Award in Area of Reliability from the ICRITO in 2010, and the Best Paper Award of the IEEE International Conference on Industrial Engineering and Engineering Management in 2012.

Shigeru Yamada received the B.S.E., M.S., and Ph.D. degrees from Hiroshima University, Japan, in 1975, 1977, and 1985, respectively. Since 1993, he has been working as a professor at the Department of Social Management Engineering, Graduate School of Engineering, Tottori University, Tottori-shi, Japan. He has published over 500 reviewed technical papers in the area of software reliability engineering, project management, reliability engineering, and quality control. He has authored several books entitled such as Introduction to Software Management Model (Kyoritsu Shuppan, 1993), Software Reliability Models: Fundamentals and Applications (JUSE, Tokyo, 1994), Statistical Quality Control for TQM (Corona Publishing, Tokyo, 1998), Software Reliability: Model, Tool, Management (The Society of Project Management, 2004), Quality-Oriented Software Management (Morikita Shuppan, 2007), Elements of Software Reliability Modeling Approach- (Kyoritsu Shuppan, 2011), and Project Management (Kyoritsu Shuppan, 2012). Dr. Yamada received the Best Author Award from the Information Processing Society of Japan in 1992, the TELECOM System

Technology Award from the Telecommunications Advancement Foundation in 1993, the Best Paper Award from the Reliability Engineering Association of Japan in 1999, the International Leadership Award in Reliability Engg. Research from the IC-QRIT/SREQOM in 2003, the Best Paper Award at the 2004 International Computer Symposium, the Best Paper Award from the Society of Project Management in 2006, the Leadership Award from the ISSAT in 2007, the Outstanding Paper Award at the IEEE International Conference on Industrial Engineering and Engineering Management (IEEM208) in 2008, the International Leadership and Pioneering Research Award in Software Reliability Engineering from the SREQOM/ICQRIT in 2009, the Exceptional International Leadership and Contribution Award in Software Reliability at the ICRITO' 2010, and 2011 Best Paper Award from the IEEE Reliability Society Japan Chapter in 2012. He is a regular member of the IEICE, the Information Processing Society of Japan, the Operations Research Society of Japan, the Japan SIAM, the Reliability Engineering Association of Japan, Japan Industrial Management Association, the Japanese Society for Quality Control, the Society of Project Management, and the IEEE.