Multistage Growth Model for Code Change Events in Open Source Software Development: An Example using Development of Nagios

Hirohisa Aman
Center for Information Technology
Ehime University
Matsuyama, Japan 790–8577
Email: aman@ehime-u.ac.jp

Akiko Yamashita
Department of Computer Science
Ehime University
Matsuyama, Japan 790–8577

Takashi Sasaki and Minoru Kawahara
Center for Information Technology
Ehime University
Matsuyama, Japan 790–8577

***Abstract*—In recent years, many open source software (OSS) products have become popular and widely used in the information technology (IT) business. To successfully run IT business, it is important to properly understand the OSS development status. Having a proper understanding of development status is necessary to evaluate and predict the product quality. However, the OSS development status is not easy to understand, because it is often concurrently developed by many distributed contributors, and its developmental structure is complicated. To aid the understanding of the development status, there is an approach that models the trend of source code change events (evolution) with a growth curve. Although an application of growth curves seems to be a promising approach, there has been a big issue that a single growth curve is often unsuitable for modeling the whole evolution because of its complex evolutionary behavior. This paper proposes a multistage model that divides the whole development period into some stages, and applies a different growth curve to a different stage. The empirical investigation in this paper shows that the switching points of stages have meaningful associations with the release dates.**

I. Introduction

Successful software quality management [1] requires quantitative and useful ways to understand the software development process, status and quality. For instance, in order to evaluate and predict the reliability of software products, many Software Reliability Growth Models (SRGM's) [2] have been studied and utilized in the software industry [3], [4]. Well-known models are ones based on Non-Homogeneous Poisson Process (NHPP), such as the exponential (Goel-Okumoto) model [5] and the delayed S-shaped model [6]. Some trend curves, such as a Gompertz curve [7] and a logistic curve [8], have also been used as empirical growth models in practice.

Recently, there have been studies applying growth curves and/or trend curves to the open source software (OSS) development. OSS products have become increasingly popular for not only personal use but also business use. A successful business with OSS products requires a useful means for evaluating and predicting the product quality based on a proper understanding of the OSS development process. Since the development of OSS products is often concurrently carried out by many distributed developers, its developmental structures and processes are likely to be complicated. For a better understanding of complicated OSS development, there has been an approach that models the occurrence of source code changes (software evolution) with a growth curve [9]. The empirical work reported in [9] shows an applicability of growth curves to explain and predict the evolution of OSS products. However, there still remains a difficulty in explaining the whole trend of code change events with using a single growth curve; it may be hard to model the whole evolutionary trend with a single model because of the complex OSS evolutionary process. In this paper, we propose a multistage model that uses two or more growth curves and/or trend curves to explain the whole trend of OSS evolution. In order to make our model generally and broadly applicable, we will use only data that are available from public code repositories and do not require any special data processing, i.e., data which anyone can easily obtain.

The key contribution of this paper is to propose a novel multistage growth model for software evolution in OSS development from the perspective of "source code change events." The proposed model can be a help in understanding the development of products and their maturity levels. Although data of failures and/or faults detected in the product present more definite information, such failure/fault data are often not available for people who do not directly contribute to the development. Since code repositories of most OSS development projects are accessible to the public, and source file change events are always available from the code repositories, the proposed model is applicable to wider range of OSS development projects.

The remainder of this paper is organized as follows. Section II describes basic growth models and their application to a software evolution analysis. Section III proposes our multistage growth model, and Section IV presents an empirical investigation into the applicability of the proposed model on a popular OSS product. Finally, Section V concludes this paper.

II. Application of Growth Model for Code Changes

This section introduces basic growth models, and describes an application of them to source code change events (evolution) in the OSS development.

A. Empirical Growth Model (Trend Curve)

In the software industry, the reliability of software products is often evaluated by focusing on the number of failures found during their testing activities [3], [10]. One of fundamental evaluating fashions is to model the growth of the cumulative number of failures over time with a mathematical growth model [4], [11], [12]. Some trend curves have been empirically used as growth models [13], [14]. Representative trend curves include Gompertz curves, logistic curves and Weibull curves [15] (see Table I and Fig.1).

A Gompertz curve is an S-shaped curve which was originally proposed to express the law of human mortality. It has been applied to various fields of science and technology for modeling biological phenomena, economic phenomena and so on [16]. A logistic curve is an S-shaped curve which was studied to model the population growth. It has also been widely applied to the scientific world as well as Gompertz curves [17]. Since the cumulative number of failures often makes an S-shaped curve over testing time, a Gompertz curve and/or a logistic curve are fitted to actual data and used for evaluating the testing activity progress or for predicting the number of remaining (potential) failures.

A Weibull curve is also useful to model the growth of cumulative number of failures. It is more flexible curve than the above two trend curves since Weibull curve has a shape parameter (d in Table I). Weibull curves can be used for modeling a software reliability growth as well [18], [19].

Table I presents the equations of these three curves, and Fig.1 shows examples of them. For all curves in the table, parameter a signifies the value to which $y(t)$ converges, i.e., $a = y(\infty)$; parameter b decides the initial value of $y(t)$, i.e., $y(0)$, and it mainly affects the shape of curve in early stages of growth; parameter c gives the rate of growth: a larger value of c corresponds to a faster growth. Parameter d of Weibull curve is its shape parameter related to the probability distribution of target event occurrence: in simpler terms, $d > 0$ makes an S-shaped curve, and $d \leq 0$ makes an exponential curve.

TABLE I. REPRESENTATIVE TREND CURVES.

Model	Formula
Gompertz curve	$y(t) = ab^{c^t}$, $(0 < a, \quad 0 < b < 1, \quad 0 < c < 1)$.
logistic curve	$y(t) = \frac{a}{1 + be^{-ct}}$, $(0 < a, b, c)$.
Weibull curve	$y(t) = a - be^{-ct^d}$, $(0 < a, b, c, d)$.

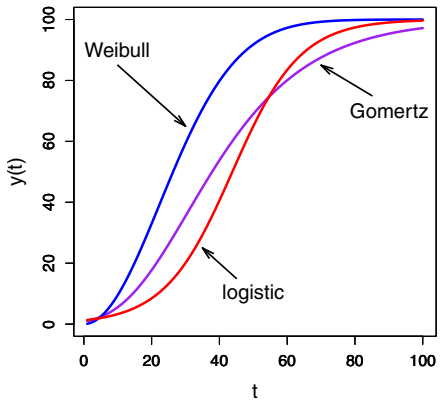

Fig. 1. Examples of trend curves shown in Table I.
($a = 100, b = 0.008, c = 0.95$ for Gompertz curve)
($a = b = 100, c = 0.001, d = 2$ for Weibull curve)
($a = 100, b = 80, c = 0.1$ for logistic curve)

B. Non-Homogenous Poisson Process-based Growth Model

In the field of SRGM study, most approaches adopt the following fashion: a failure (or a fault) detection is regarded as a stochastic event, and this detection process is modeled by the Non-Homogeneous Poisson Process (NHPP). Now let $h(t)$ be the failure (or fault) detection rate at time t, and let $y(t)$ be the expected cumulative number of failures (or faults) found by time t. We can write the relation between them as

$$h(t) = \frac{dy(t)}{dt} \ . \tag{1}$$

Then, assume that the failure/fault detection rate $h(t)$ depends on the number of remaining failures/faults as

$$\frac{dy(t)}{dt} = b(t) \cdot \{ \ a - y(t) \ \} \ , \tag{2}$$

where $a = h(\infty)$ and $b(t)$ is the proportionality factor at time t.

Equation (2) presents the basic form of NHPP-based models. There are numerous variations of such equation, and a different equation leads to a different model. Fundamental models are the exponential (Goel-Okumoto) model, the delayed S-shaped model and the inflection S-shaped model [20]. Table II gives the formulas of $y(t)$ in these three models, and Fig.2 shows examples of them.

The exponential model is derived from Eq.(2) by letting $b(t)$ be a constant, i.e., $b(t) = b$ which corresponds to parameter b in Table II. Thus, the growth in the exponential model constantly slows since the failure detection rate $h(t)$ is decided by the constant b and the remaining failures $(a - y(t))$ (see Eq.(2)).

The delayed S-shaped model is composed of two exponential models: the first model represents the failure detection process which is the same as the exponential model, and the second one corresponds to the fault recognition process. In the delayed S-shaped model, a failure detection event and a fault recognition event are distinguished, and the fault recognition is considered to be doable after the corresponding failure was detected. The formula of delayed S-shaped model is derived from another version of Eq.(2) as

$$\begin{cases} m(t) = a(1 - e^{-bt}), \\ \dfrac{dy(t)}{dt} = b \cdot \{ \ m(t) - y(t) \ \} \ , \end{cases} \tag{3}$$

where $m(t)$ and $y(t)$ signifies the expected cumulative number of detected failures and the expected cumulative number of recognized faults, respectively. The fault recognition rate is decided by the number of remaining unrecognized (already detected, but not recognized) faults $(m(t) - y(t))$, whose concept is the same as the exponential model.

TABLE II. FUNDAMENTAL NHPP-BASED GROWTH MODELS.

Model	Formula
Exponential model	$y(t) = a(1 - e^{-bt}),$ $(0 < a, b).$
Delayed S-shaped model	$y(t) = a\left\{ 1 - (1 + bt)e^{-bt} \right\},$ $(0 < a, b).$
Inflection S-shaped model	$y(t) = \frac{a(1 - e^{-bt})}{1 + ce^{-bt}},$ $(0 < a, b, c).$

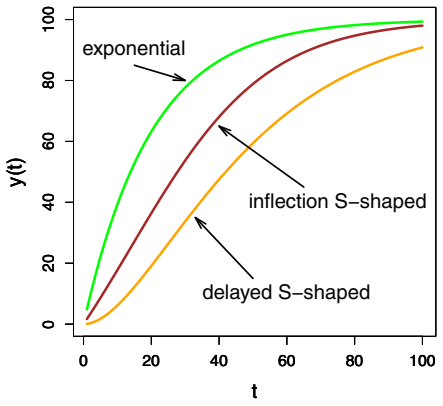

Fig. 2. Examples of NHPP-based models shown in Table II.
($a = 100, b = 0.05$ for exponential model)
($a = 100, b = 0.04$ for delayed S-shaped model)
($a = 100, b = 0.05, c = 2$ for inflection S-shaped model)

The inflection S-shaped model is also based on Eq.(2) but it uses another $b(t)$ which varies over testing time. The model introduces another parameter l in addition to parameters a and b. Parameter l represents a learning effect in the failure detection, and define $b(t)$ as

$$b(t) = b\left\{ l + (1 - l)\frac{y(t)}{a} \right\}. \quad (4)$$

If $l \to 1$ then $b(t) \simeq b$, so the model is almost identical with the exponential model. If $l \to 0$ then $b(t) \simeq y(t)/a$, i.e., $b(t)$ is approximately proportional to the ratio of detected failures by time t. The formula of the inflection S-shaped model in Table II is derived from Eq.(2) and Eq.(4), where $c = (1 - l)/l$ and $0 < l < 1$.

The exponential model shows a growth curve which is convex upward, while the delayed S-shaped model draws an S-shaped curve as shown in Fig.2. Although the inflection S-shaped model also draws an S-shaped curve, it can be a curve similar to the exponential model when $c \simeq 0$. For all models, parameter a signifies the value to which $y(t)$ converges, and parameter b corresponds to the growth rate. Parameter c of the inflection S-shaped model can be a shape parameter which is like the one of Weibull model. These models have been widely known as useful ways to evaluate the testing activities in the software industry [3], [13], [21].

C. Growth Model for Code Change Events

The growth models described in Sections II-A and II-B have been used for modeling the growth of the cumulative number of failures (or faults). Such models can also be applied to other types of data by focusing on software development events other than failure detection. One of the most interesting events is "source code change" that reflects the evolution of software products. OSS development is often concurrently carried out by many distributed developers, so timings of when code changes are made depend on individual developers. Since it is difficult to model individual developers' behaviors, there is an approach from the point of view of stochastic events—regard a code change event as a stochastic event, and model the occurrence of code changes [9], i.e., a modeling of code changes (software evolution) with a growth model.

Figure 3 shows the cumulative number of monthly code change events occurring in the development of Nagios[1], and a Gompertz curve fitted to the actual data. If a growth model explains the trend of code changes well, we would be able to evaluate the development status or predict the occurrence of code changes (evolution) based on the growth model.

III. MULTISTAGE GROWTH MODEL

While the previous work applied a growth model to code change events (software evolution), notice that such an approach assumes the growth model is well-fitted to actual data and explains the trend of code changes well. Such an assumption may not always apply in reality. For instance, the Gompertz curve shown in Fig.3 is not well-fitted to actual data. Thus, the model does not explain the whole trend of code changes well, and it does not work for evaluation and prediction of evolution well.

Now, we can find temporally (locally) stable states of the growth in Fig.3—e.g., around the 40th month and around the 100th month. That is to say, they seem to have some cycles between "actively-developed (unstable)" states and "temporally stable" states (see Fig.4). Based on this idea, we propose to switch our growth models in which we encounter temporally

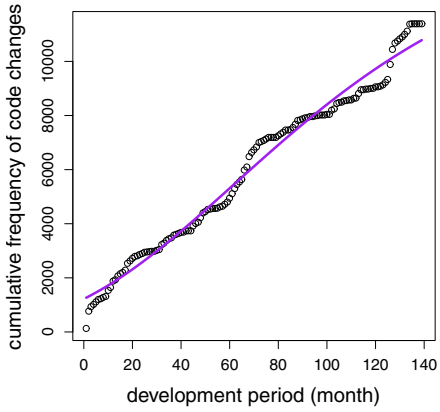

Fig. 3. Gompertz curve fitted to the cumulative frequency of code changes in the development of Nagios.

[1] `http://www.nagios.org/`. A popular application for monitoring information infrastructure systems.

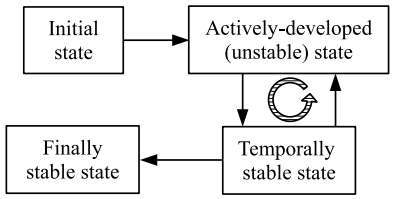

Fig. 4. Development state changes.

stable states; our growth model gets a fresh start at a temporally stable point. We will call this a "multistage growth model." The following Procedure 1 shows our model building steps.

Procedure 1 (Building a multistage growth model):

Suppose M growth models $y_i(t)$ (for $i = 1, \ldots, M$) are available in our computational environment. Let $c(t)$ be the cumulative number of code change events occurred by the t-th month of the development, and suppose we have the actual data of $c(t)$ for $0 < t \leq N$, where the N-th month is the latest month in our observation of development. Let the s-th month be the switching point at which the growth model gets a fresh start (restart).

1) Start our model building at the first month: $s \leftarrow 1$.
2) Apply a growth model $y_i(t)$ to actual data $c(t)$ for $s \leq t \leq s+x$, where x is a natural number which is grater than or equal to six[2]. For all possible x and for all available growth models, perform the above application of a growth model to actual data.
3) Find the best-fitted growth model and the duration, i.e., the pair of $y_i(t)$ and x such that the following fitness index $\phi_{i,s}(x)$ has the minimum value:

$$\phi_{i,s}(x) = \frac{\mathrm{AIC}_{i,s}(x)}{x+1} \ ,$$

where $\mathrm{AIC}_{i,s}(x)$ is the fitness of model $y_i(t)$ to the actual data $c(t)$ for $s \leq t \leq s+x$, evaluated by Akaike information criterion (AIC) [22]. A smaller value signifies a better fitting. Since AIC values get larger as the number of data increases, the above index is normalized by the number of months (the length of stage period).

Then, decide to use the best-fitted model $y_i(t)$ for the duration $s \leq t \leq s+x$.

4) If $s + x < N$, i.e., the $(s+x)$-th month is not the final month, $s \leftarrow s + x + 1$ and go back to 2) in order to get another fresh start of the next stage.

□

IV. Empirical Investigation

This section presents an empirical investigation into the applicability of the proposed model, with using the code change data of Nagios (see Fig.3 and Table III). Nagios is a popular application for monitoring information infrastructure systems, whose core development language is C. It has been actively developed and maintained for over 10 years with getting many developers and users involved in, through their support forum Web site[3], SNS (including twitter, Facebook and LinkedIn), mailing lists[4], etc.

TABLE III. Empirical object: Development of Nagios.

Data collection period	#code changes	Development language
2/28, 2002 – 9/10, 2013 (139 months)	11, 392	C

A. Data Collection

We can observe all source-file-change events tracked in a version control system by checking the working log on the system. Our data collection is conducted in the following procedure.

Procedure 2 (Data Collection):

1) Make a local copy of the code repository.

   For a public code repository, anyone can make a copy on their local disk. Such a copying is referred to as "checking out" or "cloning" of the repository, and it is a common function among all version control systems. For instance, we can make a local copy of any Git repository by using "git clone" command. Figure 5 shows the command we executed to make a copy of code repository of Nagios, where directory "*nagios*" is the root of our local copy.

2) Count the numbers of code change events occurred every month from the working log on the repository.

   We can obtain the change history from our local copy of repository, with using a log-output function which is also common among version control systems. In the case of Git, we can get the list of source files changed in specified month, by using "git log" command. Then, we can count the number of the changed files with "egrep" command in Unix/Linux operating system. Figure 6 shows an example for counting the number of source-file-change events occurred in November, 2010. In the figure, "git log" command outputs the working logs corresponding to file change events tracked between November 1st and November 30th, 2010, where the status of changed files are displayed as "M." Then, "egrep" counts the corresponding source files which are matched to the regular expression "`^M.*\.c\$`."

□

```
git clone  git://git.code.sf.net/p/nagios/nagioscore  nagios
```

Fig. 5. Making a copy of the code repository of Nagios.

```
git log --since=2010-11-01 --until=2010-11-30
        --name-status --diff-filter=M | egrep -c "^M.*\.c\$"
```

Fig. 6. Counting the number of source-file-change events occurred in November, 2010.

[2]We set $x \geq 6$ in order to avoid the overfittings. The setting of a proper lower limit of x will be our future work.

[3]http://support.nagios.com/forum/

[4]http://sourceforge.net/p/nagios/mailman/

B. Model Building and Results

Let us consider that the six models shown in Tables I and II are available, and try to apply our multistage growth model to the code change events observed in Nagios development.

(1) Let the first month be the starting point of the stage: $s \leftarrow 1$.
(2) For each of possible x's, apply each of six growth models $y_i(t)$ (for $i = 1, \ldots, 6$) to actual data $c(t)$ for $1 \leq t \leq 1+x$. Figure 7 shows one of these results: values of $\phi_{1,1}(x)$ for Gompertz model with $s = 1$. In the Gompertz model, $\phi_{1,1}(x)$ gives the minimum value at $x = 8$, so the model fitted to the actual data $c(t)$ in interval $[1, 1+8]$ is the best model.
Through our six models, the fitness of Gompertz model to the actual data was the best[5].
(3) Then, we get a fresh start at the 10th month (starting of the second stage): $s \leftarrow s + x + 1 = 1 + 8 + 1 = 10$.
(4) Similarly, the exponential growth model applied to interval $10 \leq t \leq 17$ was the best model in the second stage; the third stage was started from the 18th month.
(5) By iterating similar steps, the whole period was divided into 12 stages. Table IV and Fig. 8 show their results.

C. Discussion

Let us discuss the applicability of proposed model to the software evolution from the perspective of their release dates.

Nagios had 21 release versions during the data collection period: "1.0 – 1.4, 2.0 – 2.9, 3.0 – 3.5." A release is usually made around or after a timing of when the development becomes locally stable. Therefore, we can expect that a release would be made around a switching time of stages in our multistage growth model. Figure 9 shows the numbers of releases and their gaps from the nearest stage-switching month. For example, the gap of "-1" in Fig.9 means one month before the stage switching, and there were 4 releases at this time.

In this case, 17 out of 21 releases were made around the switching month: from -3 to $+3$. Especially, the time period of three months before (-3) had the most releases (6 releases). Since the switching-month represents the final month

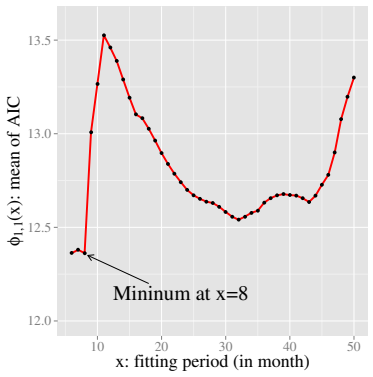

Fig. 7. $\phi_s(x)$ for Gompertz model ($s = 1$).

[5]We used one of the most popular environment for statistical computing, R (`http://www.r-project.org/`). The fitting of $y_i(t)$ and computing of AIC values can be done through the function "nls" provided in R.

TABLE IV. STAGES OF DEVELOPMENT OF NAGIOS, ESTIMATED BY PROPOSED MULTISTAGE GROWTH MODEL.

No.	period (in month)	duration (in month)	fitted model
1	1st – 9th	9	Gompertz
2	10th – 17th	8	Exponential
3	18th – 29th	12	Gompertz
4	30th – 44th	15	Weibull
5	45th – 55th	11	Weibull
6	56th – 63rd	8	Logistic
7	64th – 80th	17	Gompertz
8	81st – 101st	21	Logistic
9	102nd – 112nd	11	Exponential
10	113th – 119th	7	Gompertz
11	120th – 129th	11	Weibull
12	130th – 139th	11	Weibull

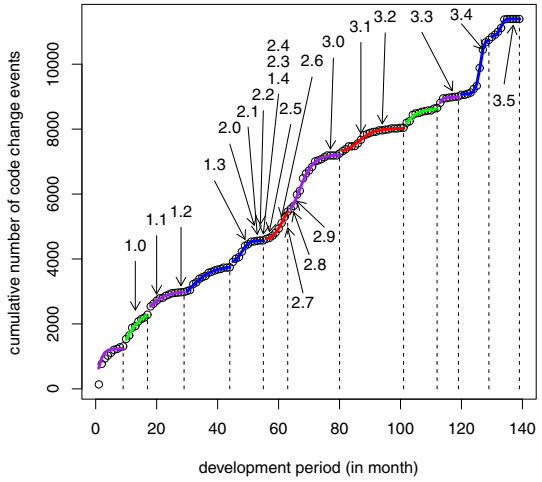

Fig. 8. Stages, fitted models and release times.

of the local growth model, the development is likely to show a temporally stable state around such a month. Although our multistage growth model uses only simple data of when code changes were made, the stage-switching points seem to reflect the actual development status.

There were releases that have a bit larger gaps from the switching month: $+6$ and -8 in Fig.9. These are corresponding to the releases of versions 3.1 and 3.2, which were made in

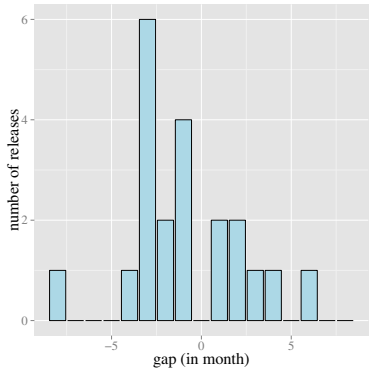

Fig. 9. Number of releases vs. gaps from the stage-switching month.

the 8th stage (the 81st month to the 101st month). Although they are in intermediate points of the growth model, the 8th stage grows at a slower pace, and this is completely stable, so those releases were made regardless of the stage switching.

Our multistage growth model can flexibly represent the software evolution from the perspective of code changes, and it would be more useful in evaluating and predicting OSS development than the conventional model using a single growth model.

D. Limitation

Our multistage growth model assumes that OSS development has cycles between actively-developed states and temporally stable states (see Fig.4). If we face a project such that the development is always active and their code changes are constantly made, our model may not work well. We would have to consider another approach for such a project.

As mentioned in Section I, we used only simple data that anyone can easily obtain. Although richer data from code analysis, mailing list analysis and bug tracking system analysis would produce better explanation models, such rich data may not always be available or easy to collect. The aim of this paper is to utilize OSS development data that are available to anyone, and to offer something useful derived from such simple data.

V. Conclusion and Future Work

This paper focused on the source code change events that are available to anyone and are easy to collect from OSS development projects. Then it proposed a multistage growth model that models the software evolution by using two or more growth models, and conducted an empirical investigation into the applicability of the proposed model on the code change data of a popular OSS product (Nagios). The empirical investigation showed that the proposed model could reflect the development cycle between actively-developed states and temporally stable states, and properly explained the timings of when the new versions are released.

In the future, we plan to conduct empirical work for evaluating the quality of OSS products by using our multistage growth model. Moreover, we will analyze the gap between the actual release time periods and the stage-switching periods in order to predict the quality of the released version. For example, the releases of Firefox are scheduled in advance, so the analysis of their gaps would be interesting findings. The switching points of growth models can tell changing points of developmental stages, and such information can be a help in analyzing the relationships between a product maturity and its preferred time of release.

Acknowledgment

The authors would like to thank the anonymous reviewers for their thoughtful comments and helpful suggestions on the first version of this paper. This work was partially supported by JSPS KAKENHI Grant Number 25330083.

References

[1] SQuBOK Project Team, "SQuBOK Guide, Guide to the Software Quality Body of Knowledge," http://www.juse.or.jp/software/squbok-eng.html, 2007.

[2] H. A. Stieber, "A family of software reliability growth models," in *Proc. 31st Annual Int'l Computer Softw. and Applications Conf.*, vol. 2, 2007, pp. 217–224.

[3] A. Wood, "Predicting software reliability," *Computer*, vol. 29, no. 11, pp. 69–77, Nov. 1996.

[4] S. Yamada, "Software quality/reliability measurement and assessment: Software reliability growth models and data analysis," *Journal of Information Processing*, vol. 14, no. 3, pp. 254–266, Jun. 1991.

[5] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. Reliability*, vol. R-28, no. 3, pp. 206–211, Aug. 1979.

[6] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Trans. Reliability*, vol. R-32, no. 5, pp. 475–484, Dec. 1983.

[7] N. Bowers, *Actuarial Mathematics*. Illinois: Society of Actuaries, 1997.

[8] R. Stone, "Sigmoids," *Bulletin in Applied Statistics*, vol. 7, no. 1, pp. 59–119, 1980.

[9] H. Aman, "A proposal of nhpp-based method for predicting code change in open source development," in *Proc. the Joint Conf. 21st Int'l Workshop Softw. Measurement and 6th Int'l Conf. Softw. Process and Product Measurement (IWSM-MENSURA 2011)*, Nov. 2011, pp. 38–47.

[10] D. Kececioglu, *Reliability Engineering Handbook, Vol.2*. N.J.: DEStech Publications, 2002.

[11] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability*. New York: McGraw-Hill, 1987.

[12] S. M. Rafi and shaheda akthar, "Software reliability growth model with gompertz tef and optimal release time determination by improving the test efficiency," *Int'l Journal of Computer Applications*, vol. 7, no. 11, pp. 34–43, Oct. 2010.

[13] N. Ullah, M. Morisio, and A. Vetro, "A comparative analysis of software reliability growth models using defects data of closed and open source software," in *Proc. 35th Annual IEEE Softw. Eng. Workshop*, 2012, pp. 187–192.

[14] D. Satoh, "A discrete gompertz equation and a software reliability growth model," *IEICE Trans. Inf. & Syst.*, vol. E83-D, no. 7, pp. 1508–1513, July 2000.

[15] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*. Berkshire: McGraw-Hill, 2002.

[16] C. P. Winsor, "The gompertz curve as a growth curve," in *Proc. National Academy of Sc.*, vol. 18, no. 1, Jan. 1932, pp. 1–8.

[17] A. Tsoularis and J. Wallace, "Analysis of logistic growth models," *Mathematical Biosciences*, vol. 179, no. 1, pp. 21 – 55, July-Aug. 2002.

[18] S. Yamada, J. Hishitani, and S. Osaki, "Software-reliability growth with a weibull test-effort: a model and application," *IEEE Trans. Reliability*, vol. 42, no. 1, pp. 100–106, Mar. 1993.

[19] N. Ahmad, M. Bokhari, S. Quadri, and M. G. Khan, "The exponentiated weibull software reliability growth model with various testing-efforts and optimal release policy: a performance analysis," *Int'l Journal of Quality and Reliability Management*, vol. 25, no. 2, pp. 211–235, Feb. 2008.

[20] M. Ohba, "Inflection s-shaped software reliability growth model," in *Stochastic Models in Reliability Theory*, ser. Lecture Notes in Economics and Mathematical Systems, S. Osaki and Y. Hatoyama, Eds. Berlin: Springer, Apr. 1984, vol. 235, pp. 144–162.

[21] S. H. Kan, *Metrics and Models in Software Quality Engineering*. N.J.: Pearson Education, 2003.

[22] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Automatic Control*, vol. 19, no. 6, pp. 716–723, Dec. 1974.