Research on Open-Source Software Reliability Models Considering Fault Introduction During the Debugging Process (Postprint)

Authors: Xiaoping Mi, Wang Jinyong

Date: 2018-07-23T00:00:00+00:00

Abstract

In recent years, open-source software has gained significant popularity in the software industry. However, the reliability of open-source software has been widely questioned. Evaluating the reliability of open-source software represents an important research problem. Compared with traditional closed-source software, establishing reliability models for open-source software must account for two critical factors: the delay time between fault introduction and fault detection and correction. This paper proposes an open-source software reliability model that incorporates the debugging process and imperfect debugging phenomena. Furthermore, we validate the fitting and predictive performance of the proposed model using two open-source software failure datasets. Experimental results demonstrate that the proposed model achieves favorable fitting and predictive performance in open-source software reliability assessment. The proposed model can be applied to reliability evaluation of open-source software in practical development processes.

Full Text

Preamble

Title: Research on Open Source Software Reliability Models Considering Fault Introduction During the Correction Process

Authors: Mi Xiaoping, Wang Jinyong† (School of Software Engineering, Shanxi University, Taiyuan 030006, China)

Abstract: In recent years, open source software has gained significant popularity in the software industry. However, the reliability of open source software has been widely questioned. Evaluating the reliability of open source software represents an important challenge. Compared with traditional closed-source

software, two factors must be considered when building open source software reliability models: fault introduction and the time delay between fault detection and correction. This paper proposes corresponding open source software reliability models that incorporate the correction process and imperfect debugging phenomena. Two open source software fault datasets are used to validate the fitting and predictive performance of the proposed models. Experimental results demonstrate that the proposed models exhibit good fitting and predictive capabilities in open source software reliability evaluation. The proposed models can be applied to reliability assessment of open source software in actual development processes.

Keywords: software reliability; software reliability model; correction process; imperfect debugging; open source software

Classification: TP311

Introduction

In modern information society, software applications have become increasingly pervasive. Not only has the scale and functionality of software code grown substantially, but development methodologies have also undergone significant transformation. For instance, the open source software development approach has gained popularity in recent years. The development of open source software differs fundamentally from traditional closed-source software development. Raymond [1] characterized open source software development as the "Bazaar" model, while referring to traditional closed-source development as the "Cathedral" model. Moreover, testing of open source software is conducted by developers, community volunteers, and users. Due to the open and dynamic nature of open source software development, its reliability has been subject to widespread skepticism [2].

Generally, software reliability growth models (SRGM) can be employed to evaluate software reliability and predict the number of remaining faults. For traditional closed-source software, numerous software reliability models have been developed over the past four decades. Most of these models focus solely on the fault detection process when establishing software reliability models [3-10]. For example, (a) perfect debugging models: Goel and Okumoto [3] proposed a non-homogeneous Poisson process (NHPP) based software reliability growth model, assuming that the failure rate is proportional to the number of remaining faults during software testing; (b) imperfect debugging models: Goel [6] was the first to propose an imperfect debugging model.

On the other hand, for closed-source software, it is typically assumed that faults detected during testing are immediately removed. In other words, the number of detected faults equals the number of corrected faults. This assumption clearly does not align with actual software testing practices, as a time delay exists be-

tween fault detection and correction in reality. Therefore, Schneidewind [11] argued that fault correction is not synchronized with fault detection; otherwise, the number of remaining faults in software would be underestimated. He proposed a modeling approach that incorporates a time delay variable for the fault correction process. Xie and Zhao [12] modified Schneidewind' s model and proposed a software reliability model where the time delay is an increasing function. Subsequently, researchers have proposed a series of closed-source software reliability models related to time delay [13-17].

Regarding open source software reliability modeling, Tamura and Yamada [18] proposed a software reliability model based on stochastic differential equations, but considered only the fault detection process. Zou and Davis [19] evaluated the reliability of open source software using several traditional closed-source software reliability models. Experimental results indicated that traditional closed-source software reliability models can be used to assess open source software reliability, with the Weibull distribution-based software reliability model demonstrating superior fitting and predictive performance. Li et al. [20] considered how volunteer interest changes over testing time during open source software development and proposed an open source software reliability model based on a fault detection rate that first increases and then decreases over time.

In general, the development environment of open source software is far more complex than that of closed-source software. Open source software involves more diverse stakeholders in development and testing compared to closed-source software. Open source software management is more loosely structured, and its development process is more dynamic. Consequently, using closed-source software reliability models to evaluate open source software reliability does not fully correspond to the actual software testing environment. Although some open source software reliability models have been developed and can be used to evaluate open source software reliability under certain testing conditions, most are based on fault detection alone and do not consider the time delay between fault correction and detection. In reality, during open source software testing, a time delay exists between fault detection and correction. For example, in the Apache open source software project, detected bugs are tagged with creation, update, and resolution times. The status of detected faults can be open, reopened, resolved, or closed. Therefore, the time delay problem between fault detection and correction in open source software is clearly evident. Additionally, since faults marked as resolved or closed can be reopened later, this indicates that the original fault was not completely eliminated or that new faults were introduced during correction. Therefore, when building open source software reliability models, the phenomenon of fault introduction must be comprehensively considered.

This paper proposes software reliability models that incorporate both the fault correction process and fault introduction phenomena during open source software testing. Model parameters are estimated using Least Squares Estimation (LSE). The model' s performance is validated using two fault datasets from open

source software. Experimental results demonstrate that the software reliability model considering fault correction processes and fault introduction phenomena exhibits good fault fitting and predictive performance.

The contributions of this paper are as follows: (a) we propose an open source software reliability model that considers both fault correction and fault introduction; (b) the time delay between fault detection and correction and the fault introduction phenomenon are two important factors in building open source software reliability models; (c) the proposed model can be used to evaluate the reliability of open source software in actual development processes.

Notation used in this paper: - $a(t)$: Fault content function (total number) - a: Expected initial number of faults in software - $b(t)$: Fault detection rate - $c(t)$: Fault correction rate - $m_d(t)$: Number of faults actually detected - $m_c(t)$: Number of faults actually corrected - α: Fault introduction rate - k: Sample size

---

1 Related Work

This section introduces existing software reliability models that consider fault correction during software testing. Previous research has investigated fault detection and correction processes. For closed-source software reliability modeling, Schneidewind [11] treated the time delay between fault detection and correction as a random variable following an exponential distribution and proposed a software reliability model incorporating the fault correction process. Xie and Zhao [12] argued that the time delay between fault detection and correction gradually increases over time and proposed an improved software reliability model considering the time delay as an increasing function. Lo and Huang [13] considered the assumption that detected faults are immediately removed during software testing to be unrealistic and proposed building corresponding software reliability models by integrating fault detection and correction processes. Huang and Lin [14] also proposed a software reliability model considering fault dependency and the time delay between fault detection and correction. Wu et al. [15] considered fault dependency and established another software reliability model incorporating the time delay between fault detection and correction. Xie et al. [16] considered the existence of time delays between fault detection and correction processes and proposed corresponding software reliability models, along with an optimal software release time cost model considering both processes. Peng et al. [21] proposed a closed-source software reliability model by integrating testing effort and fault introduction phenomena.

Recently, Liu et al. [17] proposed a Markov-based software reliability evaluation method and employed weighted least squares estimation for model parameter estimation. Liu et al. [22] considered the time delay between fault detection and correction processes as a random variable following exponential or Weibull distributions and proposed corresponding software reliability models. They also

built multi-version software reliability models and validated model performance using open source software fault datasets. Furthermore, Yang et al. [23] assumed that the time delay between fault detection and correction processes follows a Gamma distribution and built multi-version software reliability models, using two open source software fault datasets to validate model performance. Due to the complexity of open source software development and testing environments, building reliability models for open source software is extremely difficult. Ullah et al. [24] proposed an optimal model selection method to optimize the selection of established software reliability models, ultimately choosing the best model for open source software reliability evaluation.

Although existing open source software reliability models consider the time delay between fault detection and correction, they do not account for the possibility that the fault removal process may introduce new faults. Therefore, the assumptions underlying these time-delay-based open source software reliability models do not align with actual fault removal scenarios in open source software, and the validity of their assumptions is questionable.

This paper proposes an imperfect debugging open source software reliability model that fully considers the actual time delay between fault detection and removal in open source software and accounts for the possibility of fault introduction during removal. By considering the potential for fault introduction during open source software fault removal, the proposed open source software reliability model better reflects the actual fault detection and removal process, making its underlying assumptions more reasonable and credible.

2.1 General NHPP-Based Software Reliability Models

Assume that $\{N(t), t \geq 0\}$ is a counting process representing the cumulative number of detected faults up to time t, which follows a non-homogeneous Poisson process (NHPP). The mean value function (MVF) of NHPP-based software reliability growth models can be expressed as:

$$m_d(t) = a(t)(1 - \exp(-b(t)t))$$

Generally, most NHPP-based software reliability growth models assume that the number of instantaneously detected faults is proportional to the number of remaining faults in the software, such as the G-O model [3], delayed S-shaped model [4], and inflected S-shaped model [5]. This can be expressed by the following equation:

$$\frac{dm_d(t)}{dt} = b(t)(a(t) - m_d(t))$$

In Equation (3), when $a(t) = a$ and $b(t) = b$, solving the equation yields $m_d(t) = a(1 - \exp(-bt))$, which is the G-O model; when $a(t) = a$ and $b(t) = \frac{b^2 t}{1+bt}$, solving the equation yields $m_d(t) = a(1-(1+bt)\exp(-bt))$; when $a(t) = a$ and $b(t) = \frac{b}{1+\beta\exp(-bt)}$, solving the equation yields $m_d(t) = a\left(\frac{1-\exp(-bt)}{1+\beta\exp(-bt)}\right)$.

2.2 Model Integrating Imperfect Debugging and Correction Process

Assumptions of the proposed model:

1. Both fault detection and fault correction processes follow a non-homogeneous Poisson process (NHPP) [12, 23].
2. The number of instantaneously detected faults is proportional to the number of remaining faults in the software [12].
3. The number of instantaneously corrected faults is proportional to the number of detected but not yet removed faults [12].
4. Detected faults are not immediately removed; during correction, new faults may be introduced with probability α.

Based on these assumptions, the following differential equations can be established:

$$\frac{dm_d(t)}{dt} = b(t)(a(t) - m_d(t))$$

$$\frac{dm_c(t)}{dt} = c(t)(m_d(t) - m_c(t))$$

$$\frac{da(t)}{dt} = \alpha\frac{dm_c(t)}{dt}$$

Given the condition $a(0) = a$, Equation (8) yields:

$$a(t) = a(1 + \alpha m_c(t))$$

When $b(t) = b$, substituting Equation (9) into Equation (6) yields:

$$\frac{dm_d(t)}{dt} = b(a(1 + \alpha m_c(t)) - m_d(t))$$

When $c(t) = c$, substituting Equation (9) into Equation (7) yields:

$$\frac{dm_c(t)}{dt} = c(m_d(t) - m_c(t))$$

2.3 Derivation of the Proposed Model

Since a time delay exists between fault detection and correction during software testing, Schneidewind [11] proposed a time-delay model considering non-synchronization between fault correction and detection. It can be expressed as:

$$m_c(t) = m_d(t - \Delta t)$$

where Δt is a delay time variable. Xie and Zhao [12] modified Schneidewind' s model and proposed alternative forms. For example, in the time interval $(t, t + \Delta t)$, the correction rate is proportional to the number of detected but not yet removed faults. It can be expressed as:

$$\frac{dm_c(t)}{dt} = c(t)(m_d(t) - m_c(t))$$

1) Substituting Equation (9) into Equation (6), Equation (6) can be expressed as:

$$\frac{dm_d(t)}{dt} = b\,(a(1 + \alpha m_c(t)) - m_d(t))$$

Integrating both sides:

$$\int \frac{dm_d(t)}{a(1 + \alpha m_c(t)) - m_d(t)} = \int b\,dt$$

where C_1 is a constant. When $t = 0$, $m_d(0) = 0$. Therefore:

$$m_d(t) = a(1 - \exp(-(1 + \alpha)bt))$$

2) Let $C(t) = ct$, then Equation (7) can be rewritten as:

$$\exp(ct)\,dm_c(t) + \exp(ct)c\,m_c(t)dt = \exp(ct)c\,m_d(t)dt$$

Integrating both sides:

$$\exp(ct)m_c(t) = \int_0^t \exp(cx)c\,m_d(x)dx$$

Substituting Equation (9) into Equation (16):

$$m_c(t) = \exp(-ct)\int_0^t \exp(cx)c\,m_d(x)dx$$

Therefore, Equations (14) and (17) represent the software reliability models for fault detection and imperfect fault correction, respectively.

2.4 Model Parameter Estimation Method

This paper employs the Least Squares Estimation (LSE) method for parameter estimation. Since the fault correction process depends on the fault detection process, both fault detection and correction models must be considered simultaneously during parameter estimation. Therefore, considering both fault detection and correction processes [16], model parameters can be estimated using the following formula:

$$S = \sum_{i=1}^{k} \left[(m_d(t_i) - m_i)^2 + (m_c(t_i) - n_i)^2 \right]$$

Taking partial derivatives of the above formula with respect to the proposed model' s parameter variables yields:

$$\frac{\partial S}{\partial a} = 0, \quad \frac{\partial S}{\partial b} = 0, \quad \frac{\partial S}{\partial c} = 0, \quad \frac{\partial S}{\partial \alpha} = 0$$

Solving the above system of differential equations simultaneously yields the parameter values for the proposed model.

---

3 Model Fitting and Prediction Performance Comparison

This section uses two open source software fault datasets to estimate the parameters of the models employed in this paper. Four model comparison criteria are used to evaluate model fitting performance: MSE_d, MSE_c, R_d^2, and R_c^2. Two additional model comparison criteria are used to evaluate prediction performance: RE_d and RE_c. Sensitivity analysis of the proposed model' s parameters is also conducted to further investigate which parameters have significant impact on the model.

3.1 Open Source Software Fault Datasets

The two open source software fault datasets used in this paper are obtained from literature [23]. They were collected and reorganized from the online bug tracking system Bugzilla for two open source software projects: Mozilla (https://bugzilla.mozilla.org/) and GNOME (https://bugzilla.gnome.org/). One dataset includes three consecutive versions of Firefox (3.0, 3.5, and 3.6) collected from the Appendix list. The other dataset is from the GNOME open

source software project, including four consecutive versions (2.0x, 2.1x, 2.2x, and 2.3x) also collected from the Appendix list. For more detailed information about the fault datasets used, please refer to literature [23].

3.2 Model Comparison Criteria

1) Goodness-of-fit criteria

$$MSE_d = \frac{1}{k}\sum_{i=1}^{k}(m_d(t_i) - m_i)^2$$

$$MSE_c = \frac{1}{k}\sum_{i=1}^{k}(m_c(t_i) - n_i)^2$$

where MSE_d and MSE_c represent the mean squared errors in fault detection and fault correction processes, respectively. Smaller MSE_d and MSE_c values indicate better model fitting performance.

$$R_d^2 = 1 - \frac{\sum_{i=1}^{k}(m_d(t_i) - m_i)^2}{\sum_{i=1}^{k}(m_i - \bar{m})^2}$$

$$R_c^2 = 1 - \frac{\sum_{i=1}^{k}(m_c(t_i) - n_i)^2}{\sum_{i=1}^{k}(n_i - \bar{n})^2}$$

where R_d^2 and R_c^2 are used to evaluate model fitting performance in fault detection and fault correction processes, respectively. Values closer to 1 indicate better model fitting performance.

2) Prediction performance criteria

$$RE_d = \frac{\hat{m}_d(t_{i+1}) - m_{i+1}}{m_{i+1}}$$

$$RE_c = \frac{\hat{m}_c(t_{i+1}) - n_{i+1}}{n_{i+1}}$$

RE_d and RE_c represent the relative errors in fault detection and fault correction, respectively. Relative error is used for one-step prediction to measure model predictive performance. Equations (24) and (25) calculate the number of detected or corrected faults at time t_{i+1} using parameters estimated from fault data up to time t_i. RE_d and RE_c values closer to zero indicate better predictive performance. When RE_d and RE_c are greater than zero, the model overestimates the number of detected and corrected faults; when less than zero, it underestimates them.

3.3 Model Performance Comparison and Analysis

3.3.1 Model fitting performance comparison

From Table 1 , we can see that MSE_d for the proposed model gradually decreases across Firefox versions 3.0, 3.5, and 3.6, indicating increasingly better fault detection fitting performance. However, MSE_c for the proposed model first decreases then increases across Firefox versions 3.0, 3.5, and 3.6, suggesting that the fault correction process is complex and influenced by many subjective factors such as debugger capability, debugging skills, and psychological changes during debugging. Therefore, the fault correction process exhibits greater volatility than the fault detection process. Additionally, R_d^2 and R_c^2 show the same trend as MSE_d and MSE_c in terms of fitting performance changes.

From Figure 1 [Figure 1: see original paper], we can also observe the changes in detected and corrected fault counts over testing time for the proposed model. Comparing Figure 1(a) with Figures 1(b) and 1(c), the fitting performance of the proposed model for fault detection and correction processes is slightly worse. Comparing Figure 1(d) with Figures 1(e) and 1(g), the fitting performance is relatively poorer.

From Table 2 , we can see that MSE_d for the proposed model gradually decreases across Gnome2 Control-center versions 2.0x, 2.1x, 2.2x, and 2.3x. MSE_c also gradually decreases across these versions. Additionally, R_d^2 generally increases gradually, while R_c^2 first decreases then increases from version 2.0x to 2.3x. This variation indicates that the correction process remains complex and irregular. For example, the lower R_c^2 value for the proposed model in Gnome2 Control-center 2.2x also demonstrates the uncertainty in the open source software correction process.

From Tables 1 and 2, we can also observe that MSE_d for the proposed model shows a consistent decreasing trend in both Firefox and Gnome projects. However, MSE_c shows inconsistent trends—first decreasing then increasing in Firefox, but gradually decreasing in Gnome. Furthermore, R_d^2 shows consistent trends in both projects (gradually increasing), while R_c^2 shows opposite trends (first increasing then decreasing in Firefox, first decreasing then increasing in Gnome).

From the above analysis, we can draw the following conclusions: a) The proposed model' s fitting performance for the fault detection process is better than that for the fault correction process. b) In early versions of open source software, the fitting performance for both fault detection and correction is comparable and generally moderate. c) During open source software development and testing, the fault correction process is more difficult, complex, and uncertain than the fault detection process. d) Generally, the proposed model demonstrates good fitting performance across entire open source software fault datasets (including both detection and correction datasets).

3.3.2 Model prediction performance comparison

From Figure 2 [Figure 2: see original paper], we can see that the proposed model exhibits better prediction performance for the fault correction process than for fault detection in intermediate versions of open source software. However, in later versions, the proposed model shows higher prediction accuracy for fault detection than for fault correction. This demonstrates that the fault correction process is more complex and variable than the fault detection process. Generally, Figure 2 shows that the proposed model has good prediction performance in both fault detection and correction processes.

From Figure 3 [Figure 3: see original paper], we can see that the proposed model demonstrates better prediction performance for fault detection than for fault correction across the entire fault dataset. This indicates that predicting the number of corrected faults is much more difficult than predicting the number of detected faults. Additionally, Figure 3 shows that the proposed model has good predictive capability for future failures and correction behaviors in both fault detection and correction processes.

3.3.3 Sensitivity analysis of the proposed model

Sensitivity analysis is a method to determine which parameters in a model have significant impact. The general approach is to vary the value of one parameter while keeping other parameter values constant [25].

From Figure 4 [Figure 4: see original paper], we can clearly see that parameters a, α, and c in the proposed model have important impacts. Intuitively, parameter α in the proposed model is related to fault introduction during open source software development and testing. Parameter c is related to the fault correction phenomenon, while parameter a is related to the initial total number of faults. All are associated with important factors affecting software reliability modeling in open source software development and testing, such as the number of introduced faults, time delay between fault detection and correction, and the initial total number of faults. Therefore, the sensitivity analysis results are consistent with actual testing scenarios in open source software. Fault introduction and time delay between fault detection and correction are two important factors that must be carefully considered when building open source software reliability models.

---

4 Conclusion

This paper proposes an open source software reliability model that integrates fault correction and fault introduction processes. In particular, the fault introduction phenomenon is studied and, for the first time, integrated into the open source software correction process. Two real open source software fault datasets are used to validate the model' s fitting and predictive performance. Experimental results demonstrate that the proposed model exhibits good fitting and predictive performance and can be applied to reliability testing processes

for open source software. This paper also discusses and analyzes the parameter sensitivity of the proposed model. Results show that parameters a, c, and α have significant impact on the model. Moreover, fault introduction and time delay between fault detection and correction are two important factors that must be carefully considered when building open source software reliability models.

Although the proposed model can effectively evaluate the reliability of open source software in certain environments, further research is needed. For example, phenomena such as irregular fluctuations in the remaining uncorrected correction rate over testing time and nonlinear variations in fault introduction represent future research directions for open source software reliability modeling.

References

[1] Raymond E S. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary [J]. 1999 (2): 105-107.

[2] Farber D. Six barriers to open source adoption [R/OL], (2004-03). http://techupdate.zdnet.com/techupdate/stories/main/.

[3] Goel A L, Okumoto K. Time-dependent error-detection rate model for software reliability and other performance measures [J]. IEEE Trans on Reliability. 1979 (R-28): 206-211.

[4] Yamada S, Ohba M, Osaki S. S-shaped reliability growth modeling for software error detection [J]. IEEE Trans on Reliability. 1983 (32): 475-484.

[5] M. Ohba. Inflection S-shaped software reliability growth models. Stochastic Models in Reliability Theory [M]. Berlin Heidelberg: Springer, 1984: 144-162.

[6] Goel A L. Software Reliability Models Assumptions, Limitations and Applicability [J]. IEEE Trans on Software Engineering. 1985, 11 (12): 1411-1423.

[7] Pham H, Nordmann L, Zhang Xuemei. A general imperfect software debugging model with S-shaped fault detection rate [J]. IEEE Trans On Reliability. 1999, 48 (2): 169–175.

[8] Zhang Xuemei, Teng Xiaolin, Pham H. Considering fault removal efficiency in software reliability assessment [J]. IEEE Trans on Systems, Man, and Cybernetics–Part A: Systems and Humans. 2003, 33 (1): 114-120.

[9] Wang Jinyong, Wu Zhibo, Shu Yanjun, et al. An Imperfect Software Debugging Model Considering Log-logistic Distribution Fault Content Function [J]. Journal of Systems and Software. 2015, 100 (c): 167-181.

[10] Wang Jinyong, Wu zhibo, Shu Yanjun, et al. An Optimized Method for Software Reliability Model Based on Nonhomogeneous Poisson Process [J]. Applied Mathematical Modelling. 2016, 40 (13-14): 6324-6339.

[11] Schneidewind N F. Modelling the fault correction process [C]// Proc of the 12th International Symposium on Software Reliability Engineering. Washington, DC, USA: IEEE Computer Society, 2001: 185–190.

[12] Xie Min, Zhao Min. The Schneidewind software reliability model revisited [C]// Proc of the 3rd International Symposium on Software Reliability Engineering. Washington, DC, USA: IEEE Computer Society, 1992: 184-192.

[13] Lo Jung-Hua, Huang Chin-Yu. An integration of fault detection and correction processes in software reliability analysis [J]. Journal of Systems & Software. 2006, 79 (9): 1312-1323.

[14] Huang Chin-Yu, Lin Chu-Ti. Software reliability analysis by considering fault dependency and debugging time lag [J]. IEEE Trans On Reliability. 2006, 55 (3): 436-450.

[15] Wu Y P, Hu Q P, Xie Min, et al. Modeling and analysis of software fault detection and correction process by considering time dependency [J]. IEEE Trans on Reliability. 2007, 56 (4): 629–642.

[16] Xie Min, Hu Q P, Wu Y P, et al. A study of the modeling and analysis of software fault-detection and fault-correction processes [J]. Quality & Reliability Engineering. 2007, 23 (4): 459–470.

[17] Liu Yu, Li Duo, Wang Lujia, et al. A general modeling and analysis framework for software fault detection and correction process [J]. Software Testing Verification & Reliability, 2016, 26 (5) 351-365.

[18] Tamura Y, Yamada S. Optimisation analysis for reliability assessment based on stochastic differential equation modelling for open source software [J]. International Journal of Systems Science. 2009, 40 (4): 429-438.

[19] Zhou Ying, Davis Joseph. Open source software reliability model: an empirical approach [C]// Proc of the 15th Workshop on Open Source Software Engineering, New York: ACM Press, 2005: 1–6.

[20] Li Xiang, Li Yan-Fu, Xie Min, et al. Reliability analysis and optimal version-updating for open source software [J]. Information and Software Technology. 2011, 53 (9): 929–936.

[21] Peng R, Li Y F, Zhang W J, et al. Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction [J]. Reliability Engineering & System Safety. 2014, 126 (2): 37–43.

[22] Liu Yu, Xie Min, Yang Jianfeng, et al. A new framework and application of software reliability estimation based on fault detection and correction processes [C]// IEEE International Conference on Software Quality, Reliability and Security. Washington, DC, USA: IEEE Computer Society, 2015: 65-74.

[23] Yang Jianfeng, Liu Yu, Xie Min, et al. Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes [J]. Journal of Systems & Software. 2016, 115 (c): 102-110.

[24] Ullah N, Morisio M, Vetro A. Selecting the best reliability model to predict residual defects in open source software [J]. Computer. 2015, 48 (6): 56-63.

[25] Li Xiang, Xie Min, Ng S H. Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points [J]. Applied Mathematical Modelling 2010, 34 (11): 3560–3570.

Note: Figure translations are in progress. See original paper for figures.