Doctoral Dissertation
博士論文

Software Reliability Growth Model Considering Debug Activity of Open Source Software
（オープンソースソフトウェアのデバッグ活動を考慮したソフトウェア信頼度成長モデル）

March 2025
2025年3月

Shoichiro Miyamoto
宮本 翔一郎

Graduate School of Sciences and Technology for Innovation
Yamaguchi University
山口大学大学院創成科学研究科

Abstract

In recent years, an open source software (OSS) has been increasingly utilized in various fields of software development. Additionally, its use in organizations that have significant societal impacts, such as governments and financial institutions. It is growing under OSS's social standing. According to Synopsys's open source security and risk analysis (OSSRA), 96% of commercial codebase is embedded OSS's source code. Recently, the OSS has been incorporated into various systems. Many systems are being developed at low cost and in short delivery times using OSS.

On the other hand, the evaluation of OSS quality is needed to ensure the quality from the perspective of the software quality management. Since OSS is the software created by volunteer software engineer, the quality of the software varies depending on the OSS project. Therefore, it is necessary to check whether there are any problems with the quality of the software before adopting the OSS.

In the past, the software reliability model (SRM) was proposed as a reliability assessment tool for the proprietary software to evaluate the software reliability. It is the model of the trends of reliability indicators. Several SRMs can easily and quickly evaluation the reliability of proprietary software. Especially, the fault detection based on software reliability growth model (SRGM) have been proposed for the proprietary software as a method of the software reliability evaluation. The SRGMs have been utilized in various development settings. However, it is difficult to apply several traditional SRGMs to OSS due to the differing debugging activity of OSS directly. Therefore, it is needed to develop the SRGM that can evaluate the reliability for OSS. This thesis focuses on the following three debugging activities and discusses an SRGM to enable reliability evaluation in OSS:

1. Variability in the number of users: Debugging ability varies depending on the number of OSS users.

2. Differences in fault reporting systems: Fault recognition is delayed due to differences in fault reporting system process.

3. Software updates: Fault occurrence trends depend on software updates.

In terms of "Variability in the number of users," this thesis analyzes the impact of user fluctuation on existing SRGMs. It proposes execution time that accounts for the number of users. In traditional SRGM for proprietary software, the number of users remains relatively stable during the testing phase. This stability results in a consistent testing environment. However, the number of users frequently fluctuates due to factors such as software popularity and functionality. These fluctuations make it challenging to evaluate software reliability using traditional SRGM. To address this issue, this thesis introduces execution time that considers user fluctuations. By introducing the execution time considering the number of users, this issue will be resolved. Applying the execution time to the SRGM smooths the debugging capacity according to the number of users. This approach enables OSS reliability evaluations using existing models.

In "Differences in fault reporting systems," this thesis focuses on the effects of differences in the software fault reporting process. This thesis proposes the method that allows for construction of OSS reliability evaluation models quickly. OSS projects use the bug tracking systems (BTS) in order to manage the faults like as the proprietary software development. Since OSS's BTS is an Open BTS and anyone can report the faults, the quality of the reported faults varies. In addition, since the OSS projects depends on the volunteer developers. Then, the developers have the limited time to develop software compared with the proprietary software. It takes a long time to confirm the reported faults. Therefore, it takes time to analyze the fault data. This thesis analyzes the Open BTS fault data and analyze the data required to build a SRGM for OSS. Furthermore, this thesis uses the analysis results to propose the SRGM that utilizes unclosed fault data. By applying the proposed method, the reliability of OSS can be evaluated more quickly.

In “Software updates,” this thesis proposes the SRGM considering the frequent software updates in OSS. OSS frequently undergoes several software updates to fix the faults and improve functionality. Because users select versions according to the situation, the OSS versions used are highly fragmented. No research has been conducted to analyze the impact of this on SRGM. There has been no research of the SRGM considering the correlation with the download counts. Therefore, this thesis makes a software repository mining for analyzing the effect of software updates. In addition, this thesis uses the data on the number of downloads by version to analyze the relationship between the adoption status on the repository and the number of software downloads. Furthermore, this thesis proposes the SRGM considering the effect of software update.

In conclusion, this thesis has identified several relationships between OSS debugging activities and fault occurrence phenomena. Moreover, the proposed method was found to be highly effective as a reliability evaluation method for OSS. Furthermore, the proposed method can be applied to the tools used by many OSS projects. It has the potential to be used as a powerful tool selecting OSS in software development.

Contents

Chapter 1

Introduction

1.1 Research background

In recent years, incorporating open source software (OSS), the OSS is available for free. The IT system development with OSS in order to reduce the development costs has become increasingly common, not only for individuals but also at the business level. According to Synopsys's open source security and risk analysis (OSSRA), 96% of commercial codebases contain OSS [1]. Additionally, in 2022, the U.S. Department of Defense announced a policy prioritizing OSS in software procurement, signaling the growing adoption of OSS at the national level [2]. OSS is becoming a crucial part of today's social infrastructure, and its influence on society is growing significantly. On the other hand, in terms of adopting OSS, it is essential to conduct a pre-assessment of software quality from the perspective of the system quality assurance. Software quality is defined by ISO/IEC 25010, as shown in the figure [3]. Among these quality attributes, the reliability is specified as an indispensable element. It is important to select OSS with high reliability when adopting it.

However, it is difficult to develop the quantitative method for OSS reliability evaluation. Research on software reliability has predominantly focused on proprietary software. Among these studies, some software reliability models have been statistically proposed by fault data obtained from the software testing phase. Figure 1.1 shows a classification of the various software reliability models (SRM) that have been proposed in the past [4–7]. In particular, the dynamic models are re-

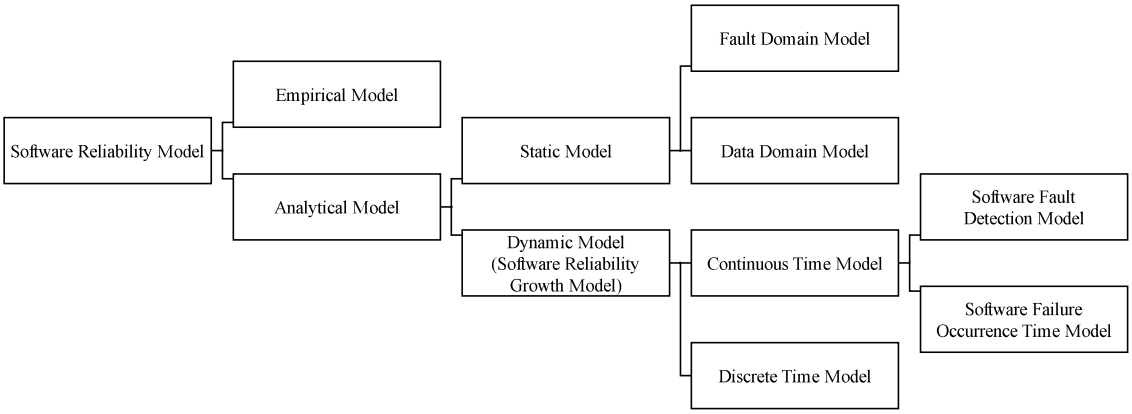


Figure 1.1: The classification of SRM.

ferred to as Software Reliability Growth Models (SRGMs). These SRGMs have been widely used to evaluate the reliability of proprietary software, due to their applicability to test data and their ability to represent the fault occurrence trends. The representative SRGMs include as follows [8–11]:

- Exponential SRGM (Exp)
- Delayed S-shaped SRGM (Delayed)
- Inflection S-shaped SRGM (Inf)
- Logarithmic Poisson Execution Time Model (Log)

However, these models were designed for the proprietary software. It is well known that applying these models to OSS is challenging due to the significant differences in the characteristics of proprietary software and OSS. It is difficult to apply the conventional SRGMs to OSS because of the following factors:

1. Variability in the number of users
2. Differences in fault reporting process
3. Software updates

From above factors it difficult to evaluate OSS reliability using the traditional SRMGs. As a result, the selection of OSS in actual development projects has required developers with a deep understanding of the characteristics and properties of OSS. Therefore, there is a need for an SRGM tailored to OSS that can quantitatively evaluating software reliability.

1.2 Purpose

This thesis focuses on the developing SRGM for OSS. Especially, this thesis focuses on the three issues listed in Section 1.1 that is problem for developing SRGM for OSS.

In terms of "Variability in the number of users," this thesis analyzes the impact of user fluctuation on existing SRGMs. It proposes execution time that accounts for the number of users. In traditional SRGM for proprietary software, the number of users remains relatively stable during the testing phase. This stability results in a consistent testing environment. However, the number of users frequently fluctuates due to factors such as software popularity and functionality. These fluctuations make it challenging to evaluate software reliability using traditional SRGM. To address this issue, this thesis introduces execution time that considers user fluctuations. By introducing the execution time considering the number of users, this issue will be resolved. Applying the execution time to the SRGM smooths the debugging capacity according to the number of users. This approach enables OSS reliability evaluations using existing models.

In "Differences in fault reporting systems," this thesis focuses on the effects of differences in the software fault reporting process. This thesis proposes the method that allows for construction of OSS reliability evaluation models quickly. OSS projects use the bug tracking systems (BTS) in order to manage the faults like as the proprietary software development. Since OSS's BTS is an Open BTS and anyone can report the faults, the quality of the reported faults varies. In addition, since the OSS projects depends on the volunteer developers. Then, the developers have the limited time to develop software compared with the proprietary software. It takes a long time to confirm the reported faults. Therefore, it takes time to analyze the fault data. This thesis analyzes the Open BTS fault data and analyze the data required to build a SRGM for OSS. Furthermore, this thesis uses the

analysis results to propose the SRGM that utilizes unclosed fault data. By applying the proposed method, the reliability of OSS can be evaluated more quickly.

In "Software updates," this thesis proposes the SRGM considering the frequent software updates in OSS. OSS frequently undergoes several software updates to fix the faults and improve functionality. Because users select versions according to the situation, the OSS versions used are highly fragmented. No research has been conducted to analyze the impact of this on SRGM. There has been no research of the SRGM considering the correlation with the download counts. Therefore, this thesis makes a software repository mining for analyzing the effect of software updates. In addition, this thesis uses the data on the number of downloads by version to analyze the relationship between the adoption status on the repository and the number of software downloads. Furthermore, this thesis proposes the SRGM considering the effect of software update.

1.3 Structure

This thesis is structured as follows:

Chapter 1 describes the background and purpose of this research.

Chapter 2 compares the proprietary software and the OSS by focusing on differences in development methods and characteristics. Then, this thesis summarizes the challenges of conducting reliability evaluations for OSS.

In Chapter 3, this thesis attempts to adapt the existing non homogeneous Poisson process (NHPP)-based SRGMs for OSS by applying the execution time based on the number of users in several OSS projects. The NHPP-based SRGMs are one of SRGM used for evaluating software reliability. The NHPP-based SRGMs have the characteristic of determining the fault convergence when the debugging activity is constant. In this chapter, this thesis addresses the problem of difficulty in determining the fault convergence using NHPP-based SRGMs for OSS by applying the execution time that considers the number of software users.

Chapter 4, this thesis analyzes the trends in faults reported in OSS projects. Also, this thesis attempts to develop the methods to accelerate the reliability evaluations. This thesis analyzes the

range of data that can be used for software reliability evaluations by examining the fault reporting trends in OSS projects. This allows us to expand the range of data needed to build NHPP-based SRGMs and develop a model using more recent data. This thesis also focuses on the optimized predictive values.

Chapter 5, this thesis uses the repository mining to analyze software usage. In particular, this thesis analyzes the process by the users migrated to new versions of OSS. This thesis also analyzes the effect of version upgrades from the perspective of a versioning method based on the semantic versioning that is widely used in software development. Based on the results of our analysis, this thesis proposes the model considering the effect of version upgrades.

Chapter 6, this thesis proposes the extended model by arranging the model proposed in Chapter 5. This thesis discusses the necessary input data in Chapter 5. Then, this thesis proposes the method to modeling with fewer data than Chapter 5.

Finally, Chapter 7 provides the summary of this research and proposes future directions. In particular, this thesis focuses on the reliability evaluation using the data from software repository. Historically, many researchers have proposed the probabilistic models aimed at evaluating the reliability of commercial software. Almost researches focus on the specific BTS. On the other hand, recent OSS projects uses different BTS such as repositories hosting service's BTS.

This thesis proposes the method to evaluate software reliability using widely used software repository's fault data. By applying the proposed method, it will enable to evaluate the software reliability of many OSS projects. Moreover, the proposed method is the practical method for evaluating the reliability of OSS projects.

Chapter 2

Background of Open Source Software and Software Repository

2.1 OSS

2.1.1 OSS in society

OSS is software that can be used freely without charge. It is utilized in the development of many systems in today's society. The use of software source code has been practiced since the early days when computers began to run software. It was widely adopted in research institutions and academic fields. In the recent software development, the concept of "reinventing the wheel" is widely recognized as a software development anti-pattern [12, 13]. The use of OSS has benefits in terms of development costs, productivity, and quality [14]. There are several definitions of OSS, one of the most widely accepted is as follows [15]:

- Free Redistribution:

  The license must allow selling or giving away the software as part of a larger distribution, without royalties.

- Source Code:

  The program must include source code. The source code distribution must be allowed.

- Derived Works:

  The license must allow modifications and redistribution of derived works under the same terms.

- Integrity of The Author's Source Code:

  The license may restrict modified source distribution. However, it must allow patch files to modify the program at build time. Additionally, it must permit redistribution of software built from modified source code.

- No Discrimination Against Persons or Groups:

  The license must not discriminate against any person or group.

- No Discrimination Against Fields of Endeavor:

  The license must not restrict use in any field, such as business or research.

- Distribution of License:

  The rights must apply to all recipients without needing an additional license.

- License Must Not Be Specific to a Product:

  Even when bundled with other software, the original OSS license must remain independent.

- License Must Not Restrict Other Software:

  The license must not impose restrictions on other software distributed with it.

- License Must Be Technology-Neutral:

  The license must not be dependent on any particular technology or interface.

As described above, the restrictions on distribution and modification are weaker than those for proprietary software. Thus, OSS is easy to use for both individuals and companies. In general, these definitions are explicitly stated in the OSS licenses. Typical examples are the MIT License and the GNU General Public License [16,17]. According to a survey by Japan Ministry of Economy, Trade and Industry, the benefits of using OSS is as follows: [18]

- Reduction of development costs and shortening of development time through increasing efficiency.
- Ensuring high stability, quality, and transparency.
- Creation of new value through a wide variety of options and avoidance of vendor lock-in.

Due to these benefits, OSS has increasingly been adopted in commercial systems. In 2023, 66% of organizations had established either an Open Source Program Office or OSS initiatives within their companies for the promotion use of OSS [19].

2.1.2 Selection of OSS

Today, numerous OSS projects are underway as the use of OSS becomes more active. As a result, there are often multiple OSS options available for selection. Therefore, the developers need to choose the most suitable OSS for their projects. According to a survey by the Japan Ministry of Economy, Trade, and Industry, the essential for selecting OSS is as follows [18]:

- Selection Evaluation
- Licensing
- Vulnerability Response
- Maintenance and Quality Assurance
- Supply Chain Management
- Personal Competence and Education
- Organizational Structure
- Community Activities

It is necessary to select the OSS from these perspectives. The software reliability is included in "Selection Evaluation"Ḣowever, the quality of software reviews depends on the skills of the software developers. Then, it is needs to develop the methods that support for reliability evaluation.

In particular, the OSS is used by multiple users with the software. Therefore, there is a high risk of similar attacks being executed when an OSS-specific vulnerability was discovered. This risk was particularly exposed in 2014 with the OpenSSL "HeartBleed" fault [20, 21]. The HeartBleed fault was injected to maintain secure sockets layer (SSL) sessions for the extended periods in accordance with the mechanism defined in "RFC 6520 Transport Layer Security and Datagram Transport Layer Security Heartbeat Extension" [22]. It had been discovered by several developers before it was reported. However, it took long time for it to fix. Even after it became public, the attacks targeting the vulnerability continued. It caused further issues such as private data leakage. Since it is virtually impossible to develop fault-free software, it has become increasingly important to select OSS with as low a risk of faults as possible. The SRM supports to select OSS by reliability evaluation.

2.2 SRGM

2.2.1 SRGM in reliability evaluation

The SRGM is positioned as a reliability model within the dynamic models of SRM. In the software development, it is extremely difficult to create software without defects. Additionally, due to the vast number of test cases in the software, there are limitations to methods that rely on the source code or the test cases in order to measure the reliability. The SRGM is a model that evaluates the reliability by modeling the transition of reliability-related indicators, e.g., the number of faults detected during software operation or testing. In software development, SRGM is known as the progress control of development, the improvement of software reliability, and the indicators of reliability evaluation. In particular, the fault detection models represent the reduction in the number of faults detected as time procedures go on. It has been empirically observed that the fault detection process varies depending on the software development approach and the skills of the developers. Therefore, various researchers have proposed models that evaluate the reliability [23–26]. These

models are classified into three categories: the time measurement models related to occurrence time, the count measurement models related to the number of occurrences, and the availability models focusing on the temporal behavior of the software. In particular, the count measurement model is based on the concept that the fault occurrence as shown in Figure 2.1.

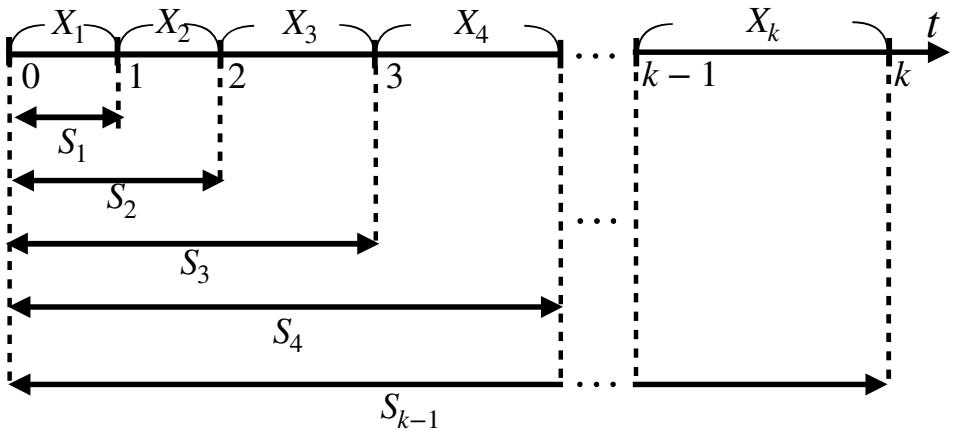


Figure 2.1: The software-failure occurrence phenomenon.

Where S_k is the k-th fault occurrence time, X_k is the time-interval. In this case, the occurrence of faults in the subsequent process can be treated as an NHPP.

1. The number of detected faults is 0 at time $t = 0$.

2. The number of latent faults is finite except for some models.

3. The software fault detection phenomena.

4. The number of detected faults depends on the number of remaining faults.

5. The fault is detected. Then, the detected faults are removed immediately.

The probability function for the number of faults n to be detected by time t is obtained as follows:

$$\Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!} \exp(-F(t)) \quad (n = 0, 1, 2, ...), \tag{2.1}$$

where $N(t)$ is the number of faults up to time t, and $H(t)$ is the mean value function. The representative mean function models include the exponential SRGM, delayed S-shaped SRGM, inflection S-shaped SRGM, and logarithmic Poisson execution time model. [8–11]. Examples of the waveforms of these models are shown in Figures 2.2-2.5.

The exponential SRGM is a simple model in which the number of detected faults decreases exponentially according to the time procedure [8]. It shows the process of finding many bugs in the early stages of testing and decreasing the number of remaining faults as the test progresses. It is as follows:

$$E(t) = a(1 - e^{-bt}), \tag{2.2}$$

where a is the number of latent faults, b is the fault detection rate.

The delayed S-shaped type SRGM considers the delay in fault detection and recognition [9]. This assumes that the detected faults are corrected immediately. However, it is difficult to fix faults. Moreover, the failures may not always be recognized immediately. The mean value function is as follows:

$$S(t) = a\left\{1 - (1 + bt)e^{-bt}\right\}, \tag{2.3}$$

where a is the number of the latent faults and b is the fault detection rate.

The inflection S-shaped SRGM considers the skill of programmer debugging [10]. In general, the software developers become accustomed to debugging as time elapsed from the initial software development. The inflection S-shaped SRGM considers the feature. The mean value function is as follows:

$$I(t) = a \cdot \frac{1 - e^{-bt}}{1 + ce^{-bt}}, \tag{2.4}$$

where a is the number of latent faults. b is the fault detection rate, c is the inflection rate.

The logarithmic Poisson execution model is structured from the initial failure intensity and the decreasing rate of the failure intensity [11]. In Eqs. (2.2)-(2.4), this is assumed that the number of latent fault is the finite. It can be useful when it is needed to estimate the number of faults. However,

most of OSS projects continue to develop. The source code will be changed with time elapsed. In this case, the number of latent fault will be also changed until the end of OSS development. The logarithmic Poisson execution time model assumes that the number of latent fault is infinity. It is not affected by changing the number of latent fault. This model is as follows:

$$F(t) = \frac{1}{\theta}\log(\lambda\theta t + 1), \tag{2.5}$$

where λ is the initial failure intensity, and θ is the decreasing rate of the failure intensity for each software failure. By estimating the parameters of these mean value functions using NHPP, it is possible to predict future trends based on fault detection data. The parameters can be estimated using the maximum likelihood estimation method. When observing y_1 to y_n faults between times t_1 to t_n, the likelihood is given as follows:

$$\begin{aligned} L &= P(t_1)P(t_2)\cdots P(t_n) \\ &= \Pr\{N(t_1) = y_1\}\Pr\{N(t_2) = y_2\}\cdots\Pr\{N(t_n) = y_n\}. \end{aligned} \tag{2.6}$$

In Poisson process, the number of faults occurs from u to s as follows:

$$\begin{aligned} n &= N(s) - N(u) \\ &= y_s - y_u \quad (s > u). \end{aligned} \tag{2.7}$$

Then, the Eq. (2.6) is as follows:

$$\begin{aligned} L = \Pr\{N(t_1) = y_1\}\Pr\{N(t_2) - N(t_1) = y_2 - y_1\}\Pr\{N(t_3) - N(t_2) = y_3 - y_2\}\cdots \\ \Pr\{N(t_n) - N(t_{n-1}) = y_n - y_{n-1}\}. \end{aligned} \tag{2.8}$$

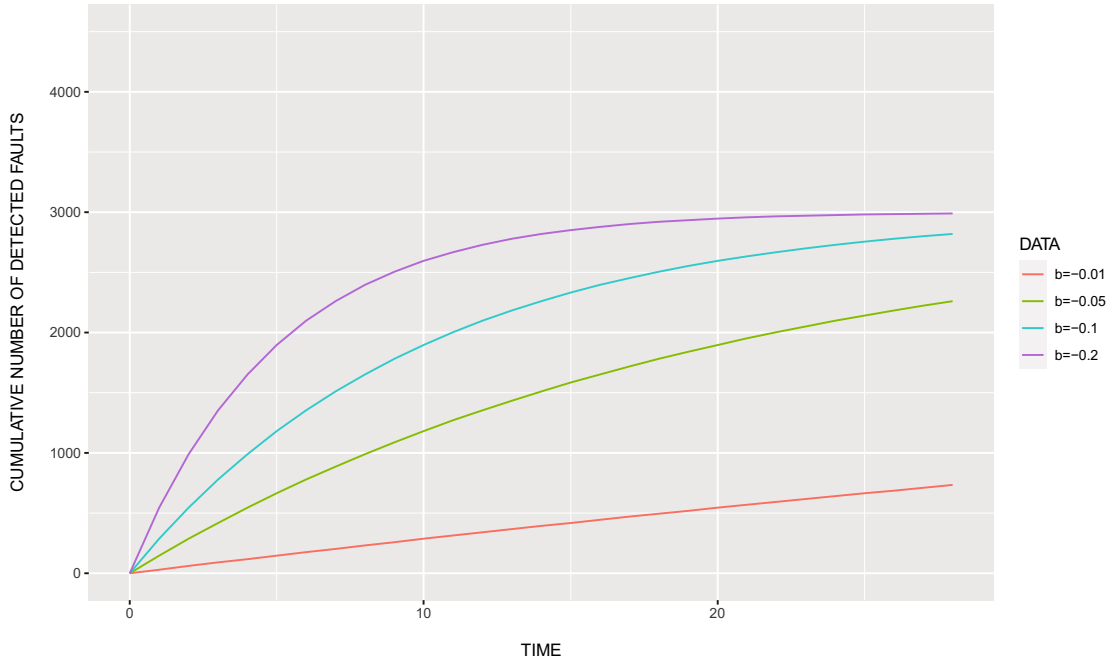


Figure 2.2: Exponential SRGM.

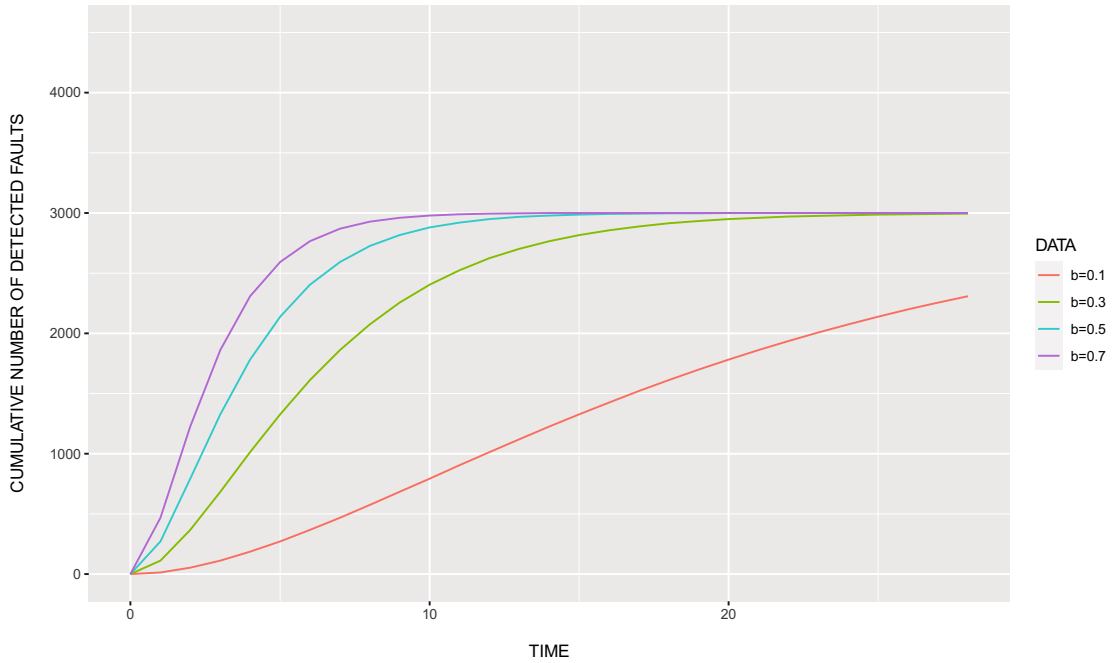


Figure 2.3: Delayed S-shaped SRGM.

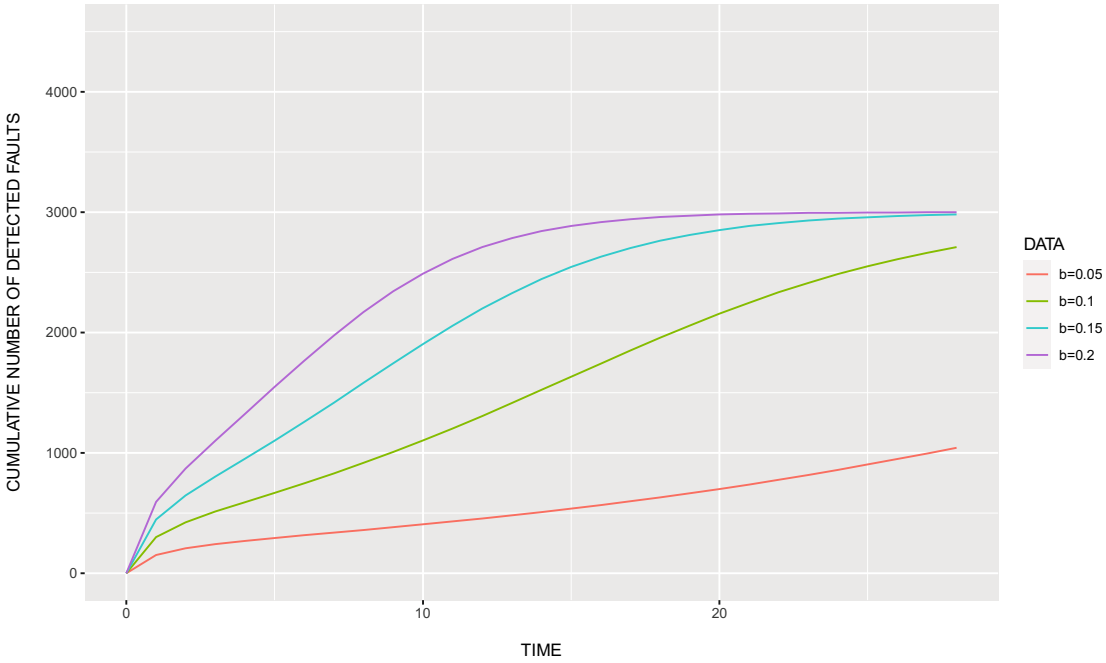


Figure 2.4: Inflection S-shaped SRGM.

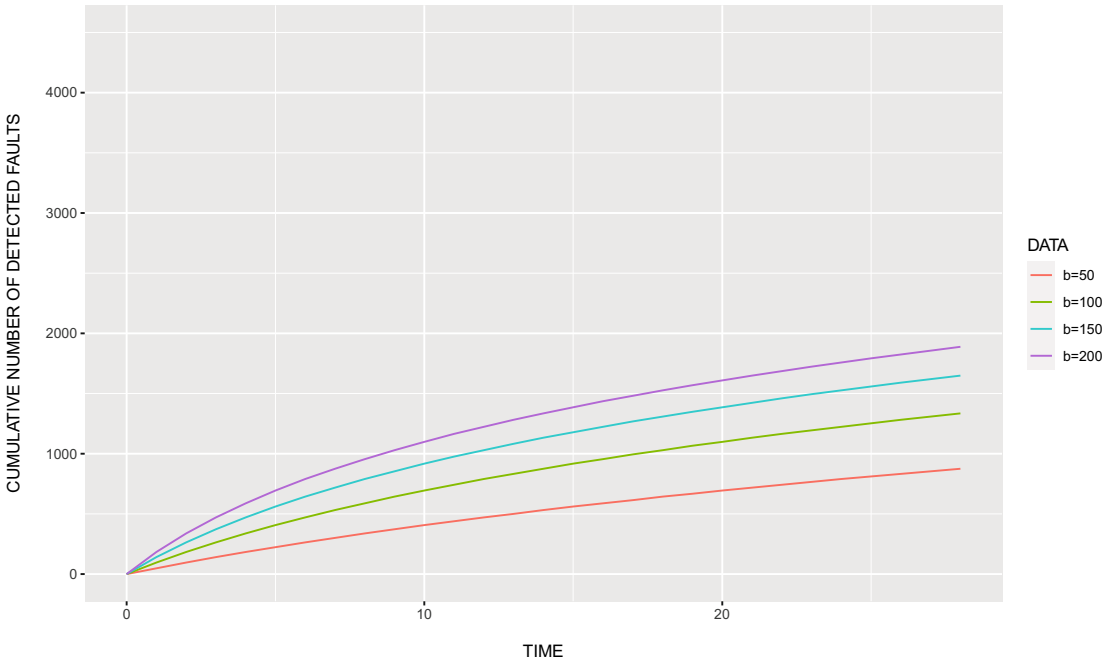


Figure 2.5: Logarithmic Poisson execution time model.

Then, the likelihood function is as follows:

$$
\begin{aligned}
L = & \frac{\{H(t_1)\}^{y_1}}{y_1!}\exp(-H(t_1))\frac{\{H(t_1)-H(t_1)\}^{y_2-y_1}}{(y_2-y_1)!}\exp(-(H(t_2)-H(t_1)))\cdots \\
& \frac{\{H(t_n)-H(t_{n-1})\}^{y_n-y_{n-1}}}{(y_n-y_{n-1})!}\exp(-(H(t_n)-H(t_{n-1}))) \\
= & \prod_{k=1}^{n}\frac{\{H(t_k)-H(t_{k-1})\}^{y_k-y_{k-1}}}{(y_k-y_{k-1})!}\exp(-H(t_n)).
\end{aligned}
\tag{2.9}
$$

Thus, the log-likelihood function is given by

$$
\ln L = \sum_{k=1}^{K}(y_k-y_{k-1})\ln[H(t_k)-H(t_{k-1})] - H(t_K) - \sum_{k=1}^{K}\ln[(y_k-y_{k-1})!], \tag{2.10}
$$

where t_k is time at k. k is the number of datasets. y_k is the cumulative number of detected faults at k. By using the optimization methods such as the simplex method [27] to find the parameters that maximize the Eq. (2.10), optimal parameters of mean value function can be obtained. After completing the parameter estimation, it is possible to estimate the future trend of fault occurrence by drawing the SRGM. If the fault does not tend to converge, you can respond by estimating the number of testing steps or increasing the number of efforts. It is sometimes used in combination with the progress management methods such as earned value management to manage the development cost [28, 29]. By using the SRGM, it is possible to determine the tendency of faults. On the other hand, the fault tendency differs from project to project in software development. In the software development, the tendency of fault occurrence differs from project to project. Therefore, the optimal SRGM differs depending on the project. Therefore, it is necessary to evaluate models using the evaluation criteria for selecting model [30].

2.2.2 Evaluation criteria

The occurrence trend of faults varies depending on the software project. Therefore, it is important to select the optimal model in order to evaluate the software reliability. Additionally, several evaluation metrics are used for model selection. The representative evaluation criteria include the mean absolute error (MAE), the mean absolute percentage error (MAPE), and Akaike's Information

Criterion (AIC) [31].

MAE is an evaluation criterion that takes the average of the errors. It can be used to evaluate not only NHPP models but also models constructed using machine learning. Since it handles all errors as they are, it is considered a robust metric against outliers. MAE is expressed by the follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|, \tag{2.11}$$

where N is the number of the data, $\hat{y}_i$ is the predicted value, y_i is the actual value.

MAPE is an evaluation criterion of error expressed as a percentage of the actual value and averaged. It has the advantage that the error is independent of the unit. On the other hand, if the actual value is small, it may not be possible to make an appropriate evaluation.

$$MAPE = \frac{100}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{y_i}, \tag{2.12}$$

where N is the number of the data, $\hat{y}_i$ is the predicted value, y_i is the actual value.

AIC is the evaluation metrics that is considering the balance between the fitting and over-fitting caused by complex model. When creating a model, complex model tends to fitting the training data [31]. However, the parameters may overfit the noise. It causes decreases the model estimation performance. It is necessary to reduce the number of parameters to prevent over-fitting. On the other hand, is difficult to create a high-performing model with a few parameters. AIC gives the one of the answers. AIC is given by

$$\mathrm{AIC} = -2\ln L + 2m, \tag{2.13}$$

where L is the likelihood function of the model, m is the number of the free parameters. In most of the cases, the optimum model is the case that the AIC becomes small.

2.3 Software repository

2.3.1 Relationship between software repository and OSS

The disclosure of source code is required for OSS. In the early days of OSS, distribution was done using personal FTP servers, among other methods. For example, the Linux kernel was distributed via FTP, and patches were shared using mailing lists [32]. With the advancement of software development technologies, by the late 1990s, the Concurrent Versions System (CVS) was developed, enabling distribution through repositories utilizing version control systems [33]. Today, distribution primarily relies on Git [34].

2.3.2 Version control system

In OSS, version control systems (VCS) are made public on repositories to track software changes, support distributed development, and ensure source code transparency. Using VCS allows development to proceed while avoiding conflicts in changes. The CVS and Subversion have been widely used for software version control in the past [33, 35]. Today, Git is the predominant tool that allows for the distributed management of the entire source code among clients. An example of development using branches is shown in Figure 2.6.

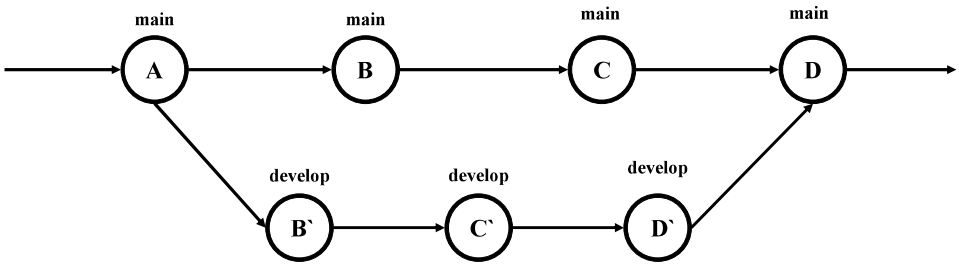


Figure 2.6: The example of branch.

Branches can be created as needed for individual software units, versions, or faults, allowing edits to be made without affecting other files. Additionally, by analyzing these edit histories through repository mining, the data can be used for reliability evaluations. Many researchers have already conducted analyzes of software using repository mining [36–40]. These studies have advanced the

analysis of fault identification, the developer contributions, and the analysis of software development. On the other hand, there are few models of SRGM by performing the repository mining for OSS. The detailed information obtained from repository mining can further advance the study of SRGM.

2.3.3 Bug tracking system

The detected faults in the software are reported to the developer for the fixing. The bug tracking system (BTS) is widely used for the fault management. Bugzilla [41], Redmine [42], GitHub Issues [43], and GitLab Issues [44] are known as the typical BTS. The information registered in BTS records the information about faults occurred in software. Various methods of software reliability assessment have been proposed the research in the past [45–48]. In addition, several researchers have proposed the research that converts detailed fault data registered in the BTS into features. Several researchers use these features to predict the time to fix the fault [49–51] and the fault severity [52–54] using deep learning. Similarly, the fault management is carried out using the public BTS in OSS. Figure 2.7 shows the process of a user reporting a fault in an open BTS.

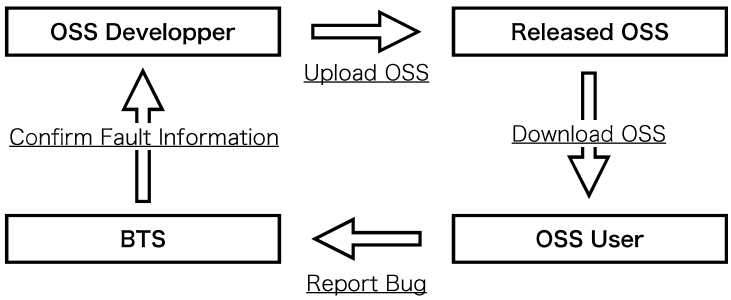


Figure 2.7: The fault fixing process.

The fault discoverer submits the information necessary for fixing the fault, such as the environment information, the software version, the reproduction steps, and the description of the fault. The OSS maintainer assigns the OSS developer to fix it based on the urgency of the fault and the availability of personnel. In long-established projects like Linux [32], the traditional BTS such as Bugzilla [41] and Redmine [42] have been used. In recent years, bug tracking functionality has been integrated into software repositories themselves, such as GitHub Issues [43] and GitLab Issues [44].

The source code is easily linked to the fault on the repository. Many OSS projects are now utilizing the built-in BTS of their repositories. The fault is assigned to a person in charge, after the fault reporting. Then, the fault is fixed through the process illustrated in Figure 2.8.

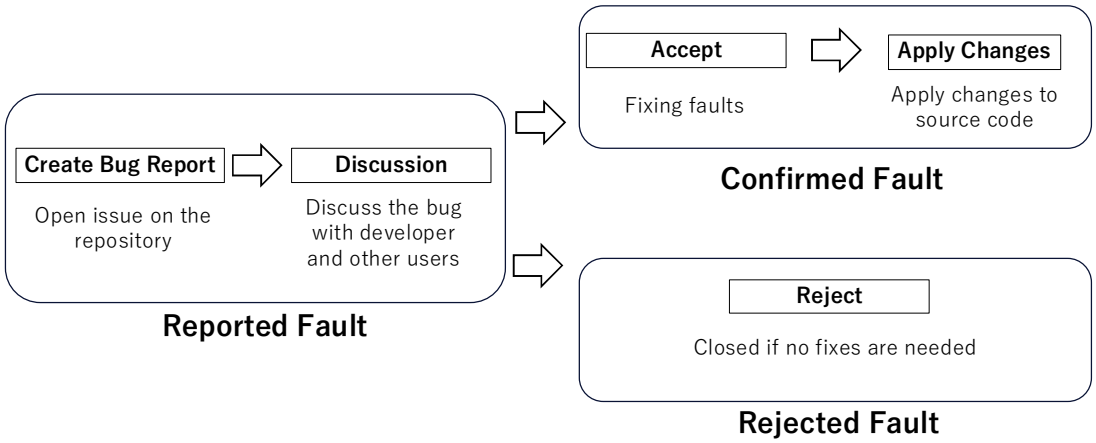


Figure 2.8: The confirmation process for the fault reporting.

The reported faults are discussed in the case. After a discussion between the reporter and other users, including developers, the developers determine whether it is a bug and proceed with the fix. A similar bug identification process exists in the proprietary software. However, since OSS is maintained by volunteer developers, there are the constraints on the available personnel and time for development. As a result, software reliability can be evaluated only with the old fault data due to prolonged confirmation of the faults. Therefore, it is necessary to develop a method to evaluate the reliability using the pre-confirmation fault data.

Chapter 3

Execution Time

3.1 Introduction

In the software development, the testing is an important process that determines the quality of the software. Because software is created by humans, it is difficult to avoid the faults caused by misunderstanding of the design or miss-definition of variables. In the large-scale system development, there are an enormous number of system states. It makes faults more likely to occur. Therefore, it is necessary to ensure the quality of the software by checking the input and output of the software in the testing and discovering faults as much as possible. There are numerous software development models. Especially, the waterfall model is known as a famous model [55]. Figure 3.1 shows the process of the waterfall model. Testing is positioned as a process in software development. When there are many people performing tests in the software testing process, more tests can be performed on the software in a shorter period of time. On the other hand, it has been reported that having too many people do not lead to shorter delivery times [56]. In addition, the costs are required according to the number of people. Therefore, the software development managers need to develop software by balancing the human resources, effort, and quality. The SRGM have been used by many developers as tools to grasp the progress of testing and the estimation of work hour. In particular, the software fault detection model is one type of SRGM. This model has been used to understand the convergence status of the number of faults.

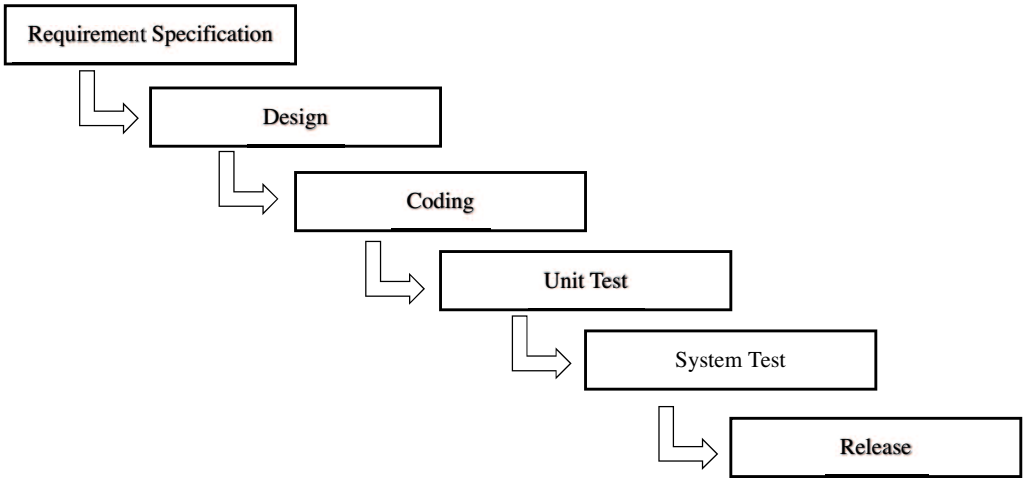


Figure 3.1: Waterfall development process.

The fault detection model consists of the cumulative number of faults on the vertical axis and the number of test cases or time on the horizontal axis. In the waterfall development, it is possible to evaluate the convergence status of the accumulated number of faults by applying the fault data of the system test.

On the other hand, in the case of OSS, the development for the purpose of fixing faults and modifying functions to the source code is conducted even after release. Therefore, it is difficult to apply the existing model to OSS. In OSS, the agile-like development such as Figure 3.2 proceeds. Since the faults are discovered and corrected even after release, it is necessary to apply all reported fault data after the software release to the SRGM when applying the fault data of OSS.

It is the OSS users who report faults, and the number of users fluctuates depending on the functionality of the OSS. Therefore, the number of testers and debugging capacity fluctuates. As a result, the probability of discovering faults also varies. At this time, there is a discrepancy between the conventional SRGM assumption and the actual. Therefore, it is necessary to standardize the testing effort depending on the number of users. This thesis considers new model for OSS. In the

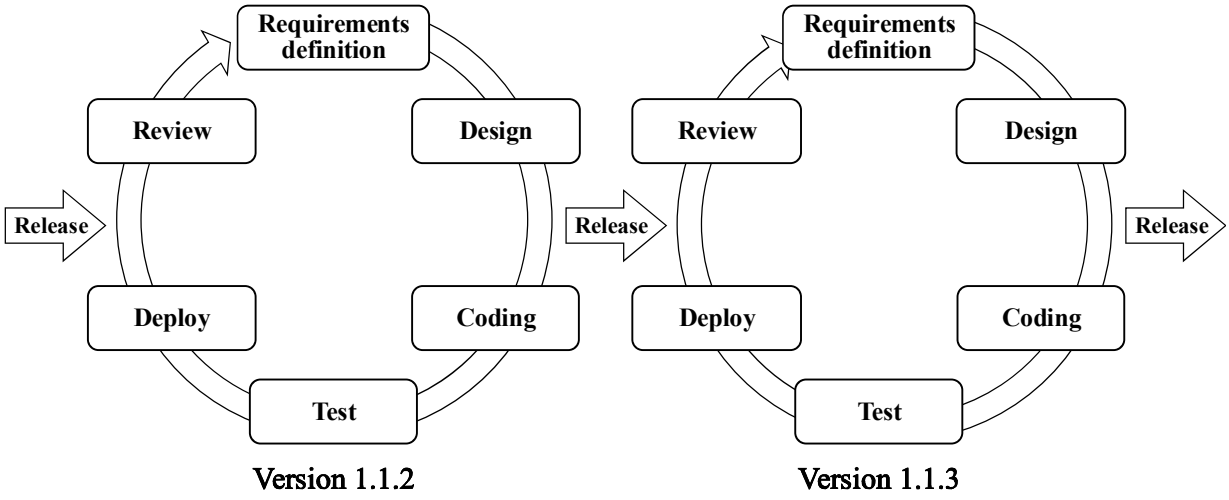


Figure 3.2: Agile development process.

past, the researches has been standardized the testing process for complex systems [57]. Moreover, the research papers have also been published by considering the variability of debugging capabilities [58]. These studies have demonstrated the effectiveness of fault data standardization. However, there has been no research on standardization that takes into account the number of testers.

This chapter focuses on the relationship between the software users and the faults by analyzing the data. This thesis also proposes the execution time model that takes into account the tester variations. Moreover, this thesis proposes the conventional model applicable to OSS.

3.2 Data collection

It is known that the number of users affects the number of issues in fault tracking systems. To investigate this phenomenon, fault data was collected from GitHub. This thesis focuses on the effects of an increasing number of users. Accordingly, the fault data was collected as follows:

- Selected Node.js library because the number of users is increasing.
- Selected the building tool library that number of users tends to fluctuate.

- Collected the number of downloads as users from software downloader.
- Collected the fault data from software repository.

This thesis focuses on GitHub as a software repository [43]. Moreover, npm is used as the distributor [59]. As a result, following OSS projects faults data were collected:

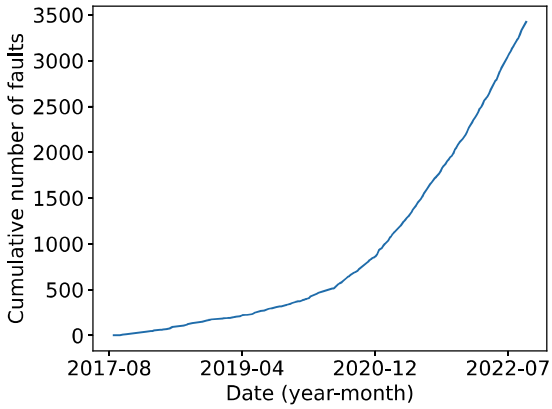
- N-DS1: Nx [60]
- N-DS2: SWC [61]
- N-DS3: webpack [62]
- N-DS4: Vite [63]

The software faults in the collected dataset have been classified according to the reporter. The software faults have been categorized by the bot, user, and development members, respectively. This thesis has removed the reported faults from bot because it isn't caused by the internal source code.

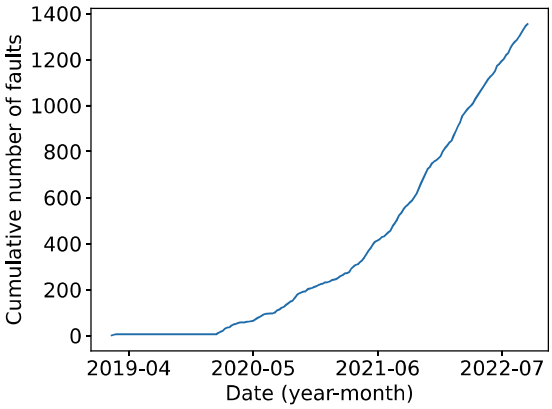
3.3 Effects of the number of users to the number of faults

3.3.1 The relationship between number of cumulative faults and time

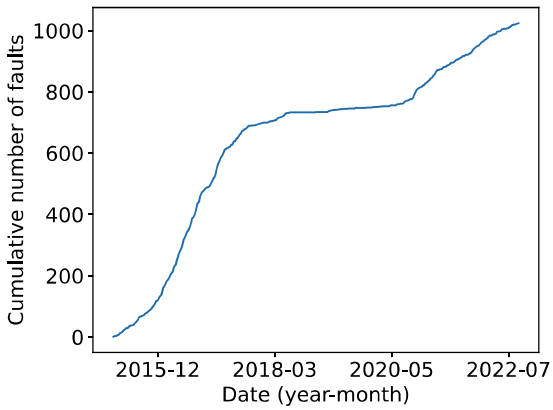
Figure 3.3 shows the relationship between the cumulative number of faults and the time. It can be seen that the cumulative number of faults tends to be less likely to increase in the early stages of release for N-DS1, N-DS2, and N-DS4. In addition, the cumulative faults of N-DS1 and N-DS2 increase exponentially. Furthermore, it is difficult to grasp the convergence trend of faults from any OSS. In conventional SRGM, the software reliability has been evaluated based on the convergence of faults. However, Figure 3.3 shows that it is difficult to evaluate software reliability from the convergence trend. Therefore, it is needed to develop other model for evaluating software reliability.



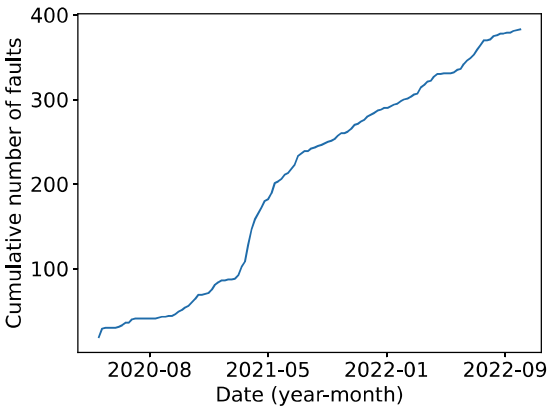
(a) N-DS1



(b) N-DS2



(c) N-DS3



(d) N-DS4

Figure 3.3: Relationship between cumulative detected faults and time.

3.3.2 The relationship between number of faults and number of downloads

Figure 3.4 shows the relationship between the number of faults and the number of downloads. In case of N-DS3, the number of downloads has been gradually increasing after its initial release. Many faults in N-DS3 were discovered in the early stages of its release. On the other hand, the number of faults and users of N-DS1 and N-DS2 have continued to increase since the release. It can be seen that numerous faults were not reported even after the number of users became very large. N-DS4 shows a similar trend to that observed in the early stages of N-DS3. The number of detected faults also increases around 2021, when the number of downloads of N-DS1 and N-DS2 increases sharply.

It is generally known that the number of faults discovered in OSS projects tends to increase as the number of users increases. From Figure 3.4, it can be seen that the number of faults does not increase in proportion to the increase in the number of users.

3.3.3 Summary of the OSS fault data features

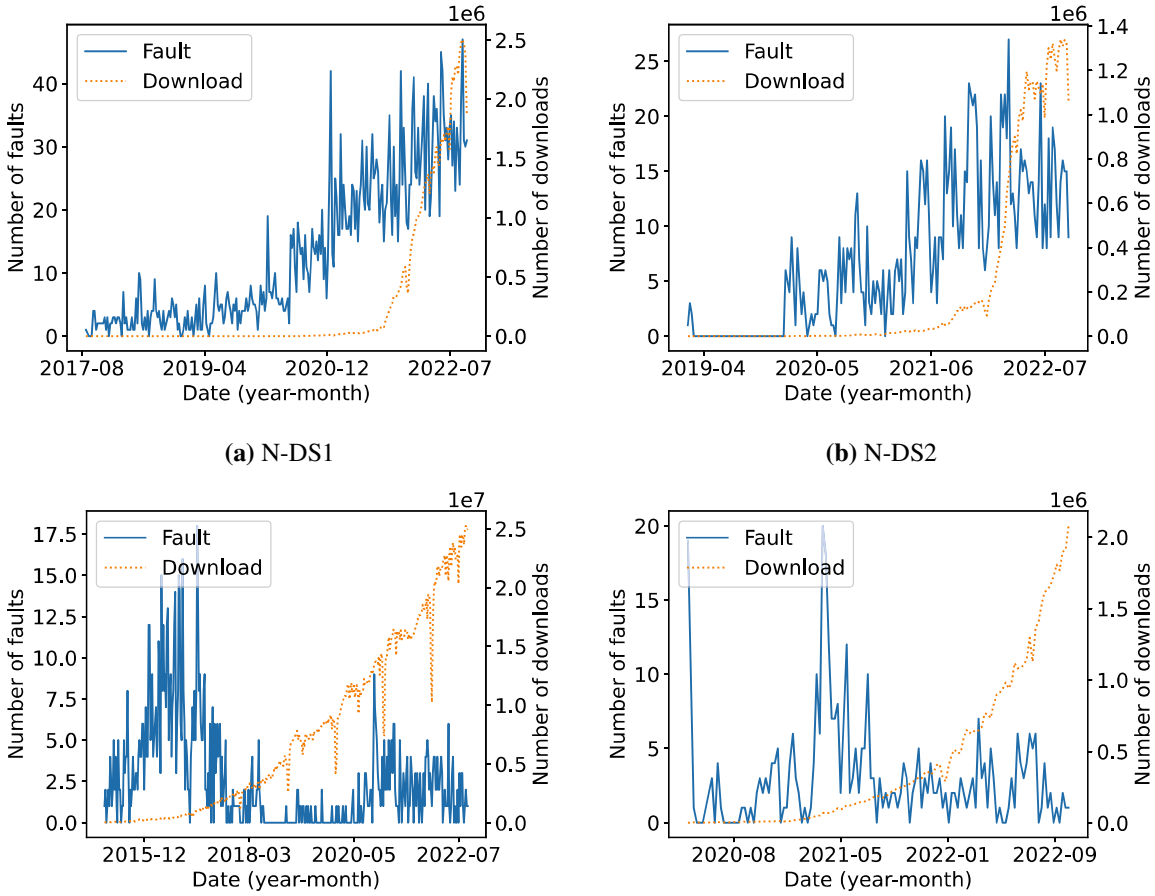
From Sections 3.3.1 and 3.3.2, these results show as follows:

- It is not always the case that many faults will be discovered in the early stages of an OSS release.
- It is difficult to observe the convergence of the number of faults on calendar time.
- The number of faults is not proportional to the number of downloads.

From these results, the previous SRGM is not suitable method to assess the reliability. Furthermore, previous mean value function was not assessed the feature of OSS because it was made for general development software. Thus, a re-examination of SRGM for OSS is required.

3.4 Proposed method

In Section 3.3.3, it was shown that the previous method is not suitable in the number of users fluctuates greatly situations. Thus, OSS SRGM needs to be re-examined. Generally, in case of



(a) N-DS1 **(b)** N-DS2

(c) N-DS3 **(d)** N-DS4

Figure 3.4: Relationship between the number of detected faults and the number of downloads.

more there are testers, the software testing process can be completed faster. Moreover, the number of downloads means the number of users. Furthermore, SRGM's mean value function supposes the software testing process will proceed according to time. However, it doesn't consider the testing speed fluctuation. Thus, it is needed to correct the time to actual testing time. Ideally, it is the best to use the cumulative actual using time by all user. However, it is difficult to collect. On the other hand, the number of downloads is an indicator to the number of the users. Multiply the number of the downloads by elapsed time to get an indicator that corresponds to the actual execution time. It is represented by

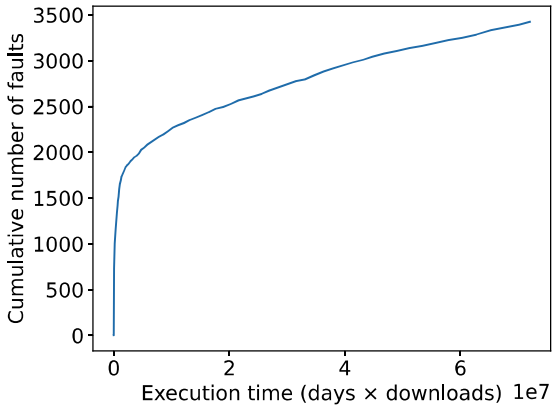
$$T_i = \sum_{k=1}^{i} \Delta t_k N_k, \tag{3.1}$$

where i is the number of unit time, Δt_k is the elapsed time at k-th, N_k is the number of downloads at k-th. The number of testing user is constant in the general software. In this case, N in Eq. (3.1) is a constant. Thus, software execution time is an extension of time. Moreover, it can also assess general software. This method can calculate the time considering the variation of the number of the users. Figure 3.5 shows the result when the proposed method is applied. From Figure 3.5, most of the fault are found early phase after the release. Moreover, it can observe the convergence of SRGM. It could not be evaluated by previous methods.

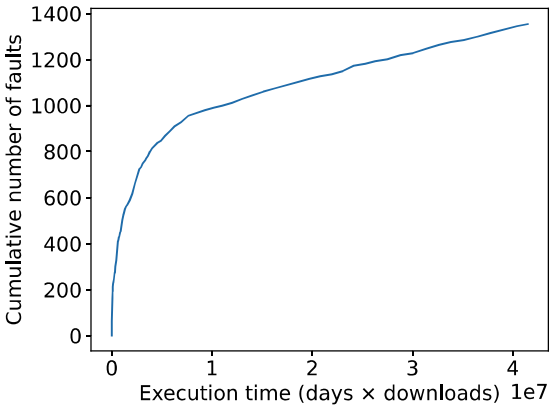
3.5 Numerical example

In order to examine the proposed method, this thesis compare it with previous methods. 95% of the data collected in Section 3.2 was used as training data. The training data was then optimized using the simplex method [27], and Eq. (2.10) was maximized. Optimization was performed using Eqs. (2.2)-(2.5) as the mean value function. Furthermore, the SRGM was evaluated using Eqs. (2.11)-(2.13) to compare the results. Figure 3.6 shows the optimization result of previous method. From Figure 3.6, the logarithmic Poisson execution time model has the transition closest to the measured value. On the other hand, the other methods are less accurate.

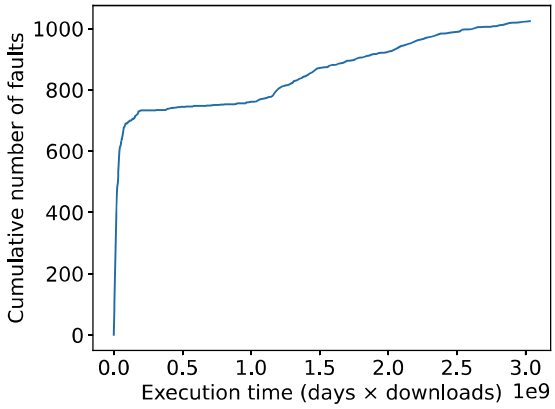
Figure 3.7 shows the optimization result of previous method. From Figure 3.7, the logarithmic Poisson execution time model has also the transition closest to the measured value. Furthermore,



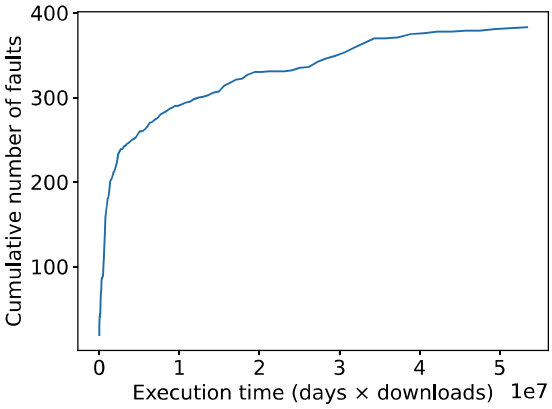
(a) N-DS1



(b) N-DS2

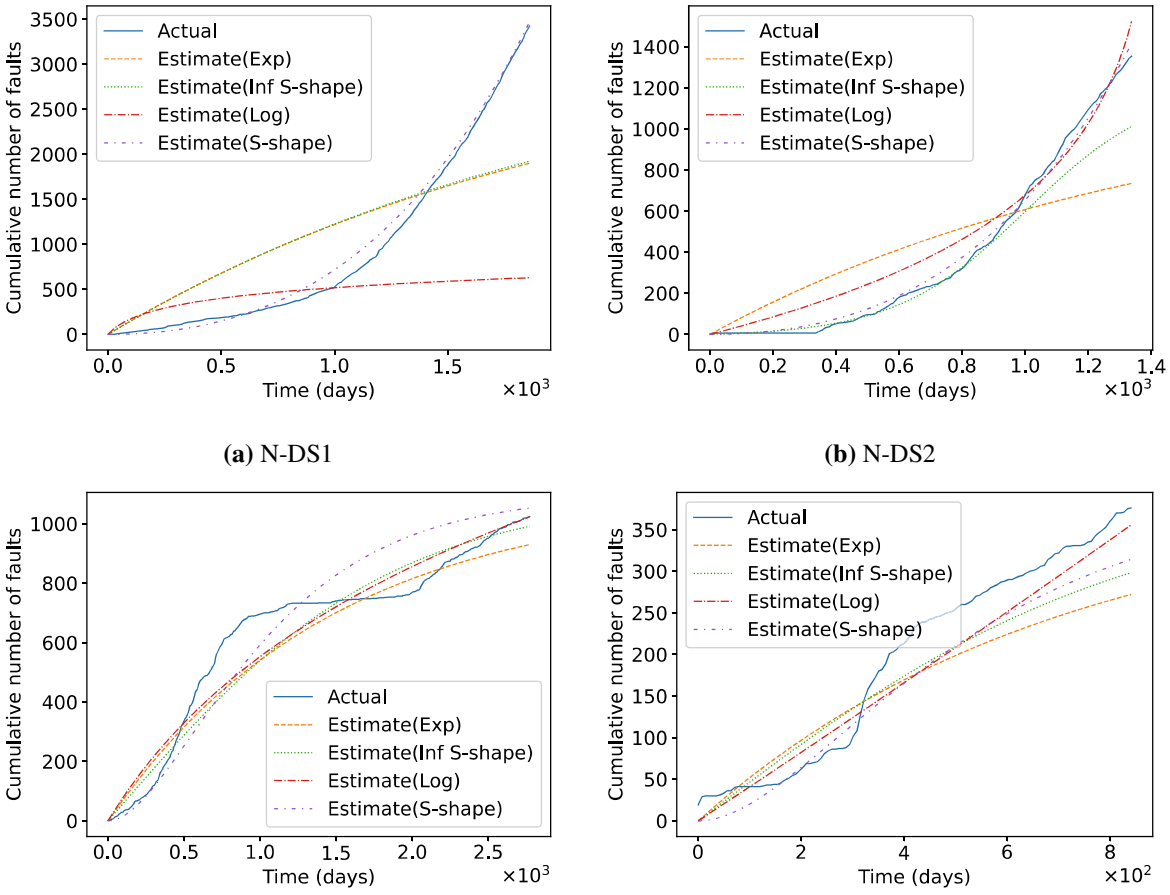


(c) N-DS3



(d) N-DS4

Figure 3.5: The estimated curves by using the proposed method.



(a) N-DS1 **(b)** N-DS2

(c) N-DS3 **(d)** N-DS4

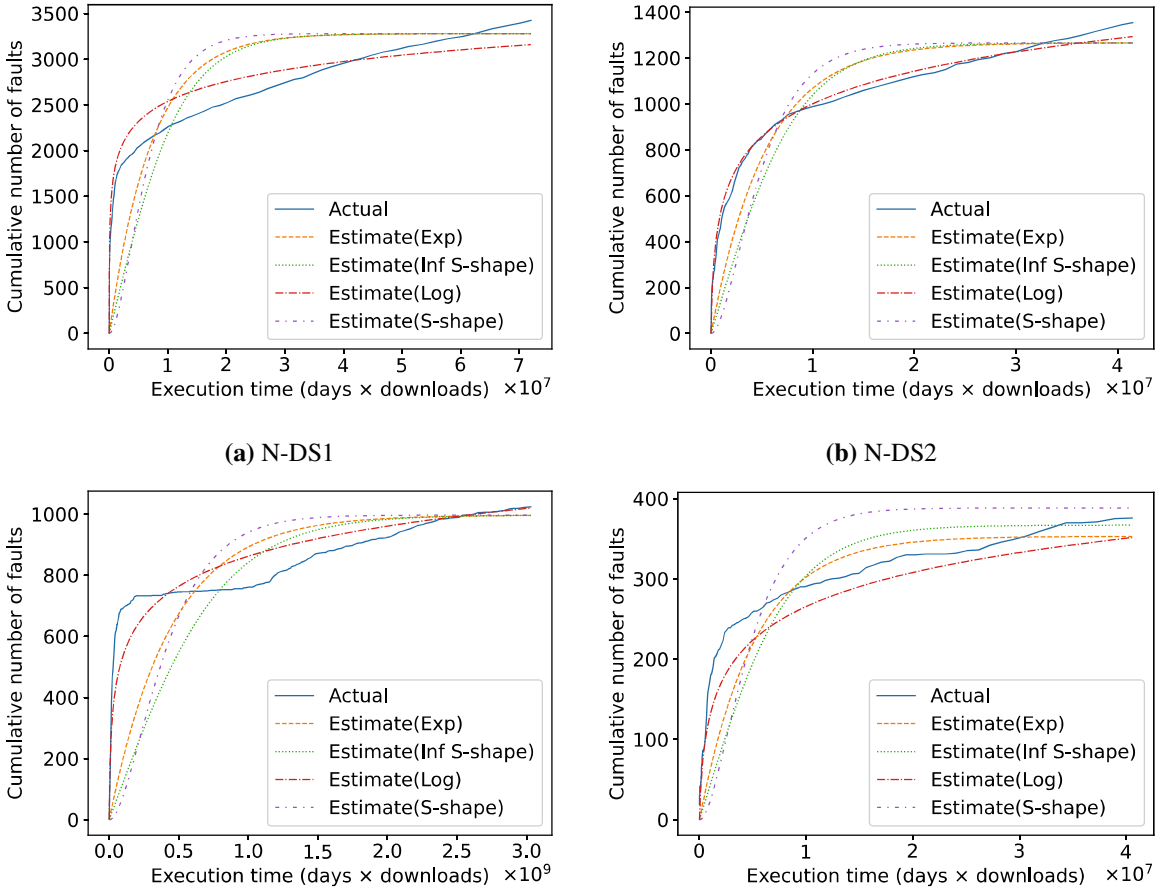
Figure 3.6: The relationship between the number of faults and the time using previous method.

the proposed method showed high accuracy even in cases where existing methods were significantly less accurate.

Tables 3.1-3.4 show the result of model evaluation metrics. The proposed model shows the smaller values of Eqs. (2.11)-(2.13) in most of the cases. Comparing the proposed method and the previous method, the proposed method gives better results.

3.6 Summary

In this chapter, this thesis has discussed the effect the number of users to the number of faults in OSS. As a result of data collection, it was found that the rapid increase in the number of users affects the number of reported faults. Moreover, there is no linear relationship between the number



(a) N-DS1

(b) N-DS2

(c) N-DS3

(d) N-DS4

Figure 3.7: The relationship between the number of faults and the execution time using the proposed method.

Table 3.1: The comparison results of goodness-of-fit (N-DS1).

	MAE		MAPE		AIC	
	Previous	Propose	Previous	Propose	Previous	Propose
Exp	1.34e+03	1.20e+02	7.20e-01	4.00e-02	5.45e+03	9.55e+03
Delayed	3.41e+01	1.22e+02	1.00e-02	4.00e-02	1.52e+03	2.29e+04
Inf	1.32e+03	1.21e+02	7.00e-01	4.00e-02	5.36e+03	1.10e+04
Log	2.59e+03	1.31e+02	4.15e+00	4.00e-02	1.33e+04	2.28e+03

Table 3.2: The comparison results of goodness-of-fit (N-DS2).

	MAE		MAPE		AIC	
	Previous	Propose	Previous	Propose	Previous	Propose
Exp	5.69e+02	3.98e+01	7.90e-01	3.00e-02	2.66e+03	1.82e+03
Delayed	3.14e+01	4.00e+01	2.00e-02	3.00e-02	9.54e+02	4.35e+03
Inf	3.10e+02	3.98e+01	3.20e-01	3.00e-02	1.01e+03	2.25e+03
Log	7.78e+01	3.24e+01	5.00e-02	3.00e-02	1.18e+03	9.41e+02

Table 3.3: The comparison results of goodness-of-fit (N-DS3).

	MAE		MAPE		AIC	
	Previous	Propose	Previous	Propose	Previous	Propose
Exp	8.87e+01	1.60e+01	1.00e-01	2.00e-02	1.90e+03	3.92e+03
Delayed	3.73e+01	1.41e+01	4.00e-02	1.00e-02	1.97e+03	7.52e+03
Inf	2.72e+01	1.61e+01	3.00e-02	2.00e-02	1.93e+03	4.53e+03
Log	1.16e+00	2.47e+00	1.00e-02	1.00e-02	1.89e+03	1.86e+03

Table 3.4: The comparison results of goodness-of-fit (N-DS4).

	MAE		MAPE		AIC	
	Previous	Propose	Previous	Propose	Previous	Propose
Exp	1.34e+03	1.66e+01	7.20e-01	5.00e-02	5.45e+03	7.64e+02
Delayed	3.41e+01	1.90e+01	1.00e-02	5.00e-02	1.52e+03	1.42e+03
Inf	1.32e+03	5.43e+00	7.00e-01	1.00e-02	5.36e+03	9.04e+02
Log	2.59e+03	2.54e+01	4.15e+00	7.00e-02	1.33e+04	5.36e+02

of users and the number of faults. These show the impact of a rapid increase in the number of users and the number of faults. In particular, the SRGM can assess the number of cumulative faults. The SRGM is also affected from the cumulative detected faults. It is difficult to assess the reliability by using SRGM in case that the number of the user increases rapidly.

This thesis has studied NHPP based SRGM by considering the number of users. This thesis proposed the execution time of SRGM that considers the number of downloads as the number

of users. The results have shown that the appropriate SRGM even in case of the fluctuation for the number of users. Moreover, the results have shown that most faults are found early in the software's release period. It is similar to general proprietary software. Furthermore, this thesis has demonstrated the numerical example by comparing previous method with the proposed one. As the result, the proposed method has shown better results in most cases. Especially, it is difficult to apply SRGM to the data using Eqs. (2.2)-(2.4) in previous methods. Since the proposed method corrected to the execution time, it is easy to apply it to actual situations.

On the other hand, there may be some possible limitations in the proposed method. In the proposed method, the execution time per user is treated to be the same. However, it is possible that the users' OSS execution times differ depending on the start time using the software by users. Therefore, the execution time may not be estimated accurately for some OSS.

In the future, the execution time per user will be analyzed from software repositories to estimate more accurate execution times using repository mining. Moreover, the proposed method will be applied to other NHPP-based SRGMs to compare estimation accuracy.

Papers Related in this Chapter

1. S. Miyamoto, Y. Tamura and S. Yamada, "A method of reliability assessment based on trend analysis for open source software," International Journal of Reliability, Quality and Safety Engineering, Vol. 31, No. 4, World Scientific, pp. 2450004-1–2450004-15, 2024.

2. S. Miyamoto, Y. Tamura, and S. Yamada, "A Reliability Assessment Method for Public Software Repository Based on Deep Learning, "RIMS kokyuroku "Mathematical Decision Making Under Uncertainty and Related Topics," No. 2242, pp. 69–80, January, 2023 (in Japanese).

Chapter 4

Considering Uncertain Fault Model

4.1 Introduction

The fault data registered in the BTS is frequently used in order to evaluate the software reliability. In the software reliability evaluation, the typical evaluation metrics include the mean time between failure, the mean time to failure, and the availability. Many researchers have proposed the reliability evaluation indices [64–67]. The data is based on the time of fault occurrence. Therefore, there are research papers focus on calculating these indices using the BTS. It has also been used in many studies to build SRM. Evaluating the reliability of software, it is important to use the most up-to-date data. In particular, in the case of software that is frequently updated, the reliability of the software frequently changes due to the updates. Therefore, by utilizing the most up-to-date data when constructing SRGM, the accuracy of extrapolation estimation can be improved. It is crucial to keep the fault data obtained from BTS as current as possible.

On the other hand, the open BTS includes the confirm process shown in Figure 2.8. Additionally, it can take time for faults to be confirmed. This can lead to delays in fault confirming the fault in BTS. The period of the obtained data becomes outdated. This thesis compares the SRGM constructed using the number of reported faults and the number of confirmed faults. Based on the analysis results, this thesis proposes the SRGM that by utilizing the fault data before confirm.

4.2 Data collection

The fault dataset was collected to analyze the difference in trends between reported faults and confirmed faults from the open BTS. This thesis focuses on hyper text transfer protocol (HTTP) client OSS that is technically mature and does not have numerous functions. Moreover, popular HTTP client OSS from the Python Package Index were narrowed down to collect a small noise dataset [68]. The selection of OSS is based on the following criteria:

- Including “HTTP ” word in the OSS description on PyPI.
- Include HTTP Client function in the OSS.
- Top 4 libraries in terms of downloads that meet the above two criteria on PyPI from March 24th to 31st, 2024.
- Not maintained by company.

As a result, HTTP client OSS were selected as shown in Table 4.1. Among the top 100 software downloads on PyPI, 11 OSS were tagged with HTTP. Seven of these were non-HTTP client OSS, such as HTTP servers. Therefore, this thesis selected following OSSs:

- P-DS1: urllib3 [69]
- P-DS2: requests [70]
- P-DS3: aiohttp [71]
- P-DS4: h11 [72]

This thesis analyzed the four selected OSS using fault data reported in the issue section of GitHub [43] and the number of version-specific downloads data on BigQuery [73] collected by the Linehaul [74].

Table 4.1: Top 4 most downloaded HTTP clients on PyPI from March 24th to 31st, 2024.

Total rank	OSS	Number of downloads
3	P-DS1	128802654
5	P-DS2	110197755
63	P-DS3	27269399
99	P-DS4	18254159

4.3 Analysis of fault reported on the open BTS

4.3.1 Fault detection trends

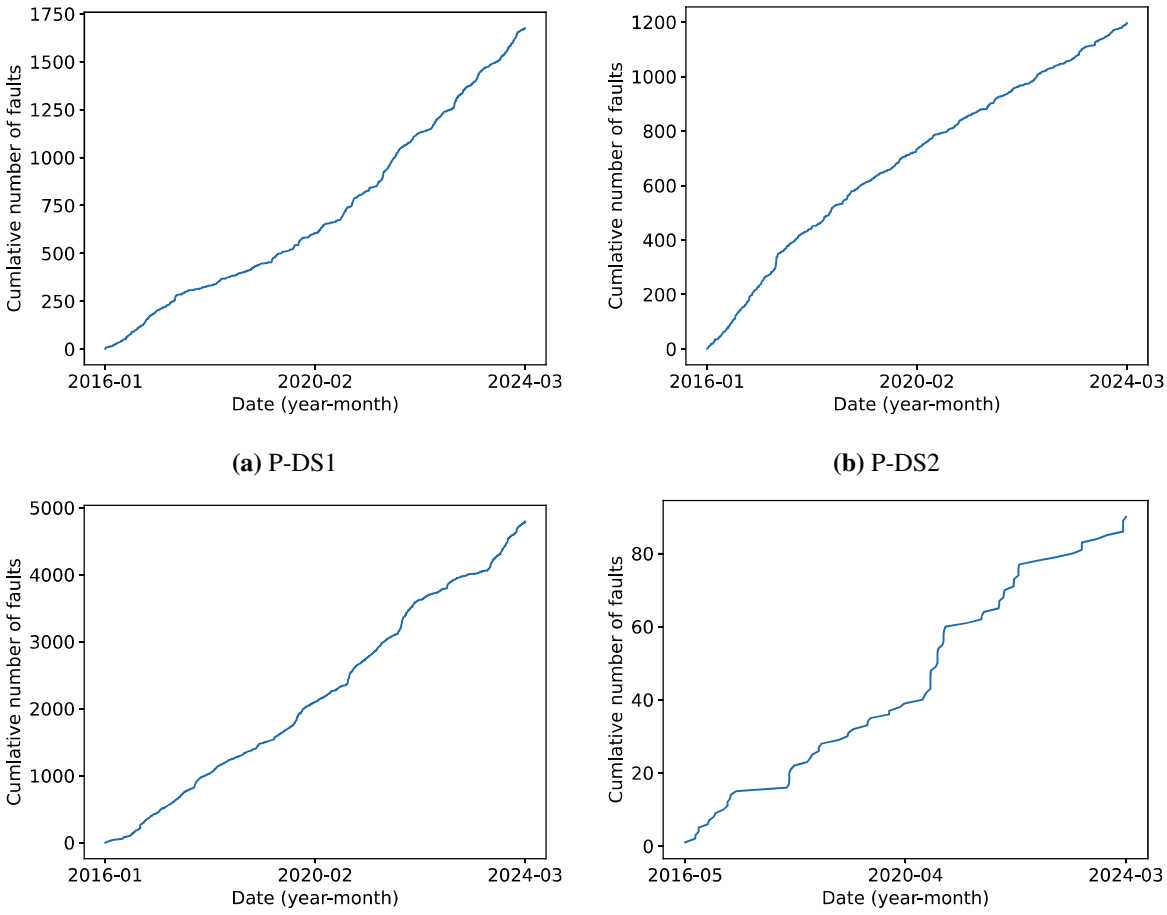
It is known that the number of faults in OSS tends to not converge easily. Therefore, it is verified whether the selected OSS exhibits this characteristic. Figure 4.1 shows the relationship between cumulative faults and time. From Figure 4.1, it can be observed that in all OSS, faults shows the nearly linear increase. It can be seen that convergence trend isn't observed. Thus, it is confirmed that reliability evaluation based on Figure 4.1 is difficult. By applying Eq. (3.1) to Time, the relationship between the execution time and the cumulative faults is shown in Figure 4.2. Eq. (3.1) reveals that in all OSS, a convergence trend in faults is observed. It makes reliability evaluation possible.

4.3.2 Relationship the confirmed fault and the rejected one

This thesis considers the SRGM based on NHPP curve. The SRGM based on NHPP is calculated to measure the degree of fault convergence. In popular OSS, the number of reported and confirmed faults is enormous. Therefore, if the tendency of each occurrence is the same, the SRGM based on NHPP curves are expected to be similar. This thesis allows to compare the two curves using normalization. This thesis normalized the x-axis and y-axis of the SRGM based on NHPP by dividing by each maximum value. The SRGM is built using the execution time proposed in chapter 3. Figure 4.3 shows the normalized SRGM based on NHPP.

The fitting of the curve is also considered based on the mean absolute error (MAE). Given the coordinates (x_1, y_1) and (x_2, y_2) of two points, (x_3, y_3) between them is calculated as:

$$y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_3 - x_1) + y_1. \tag{4.1}$$



(a) P-DS1

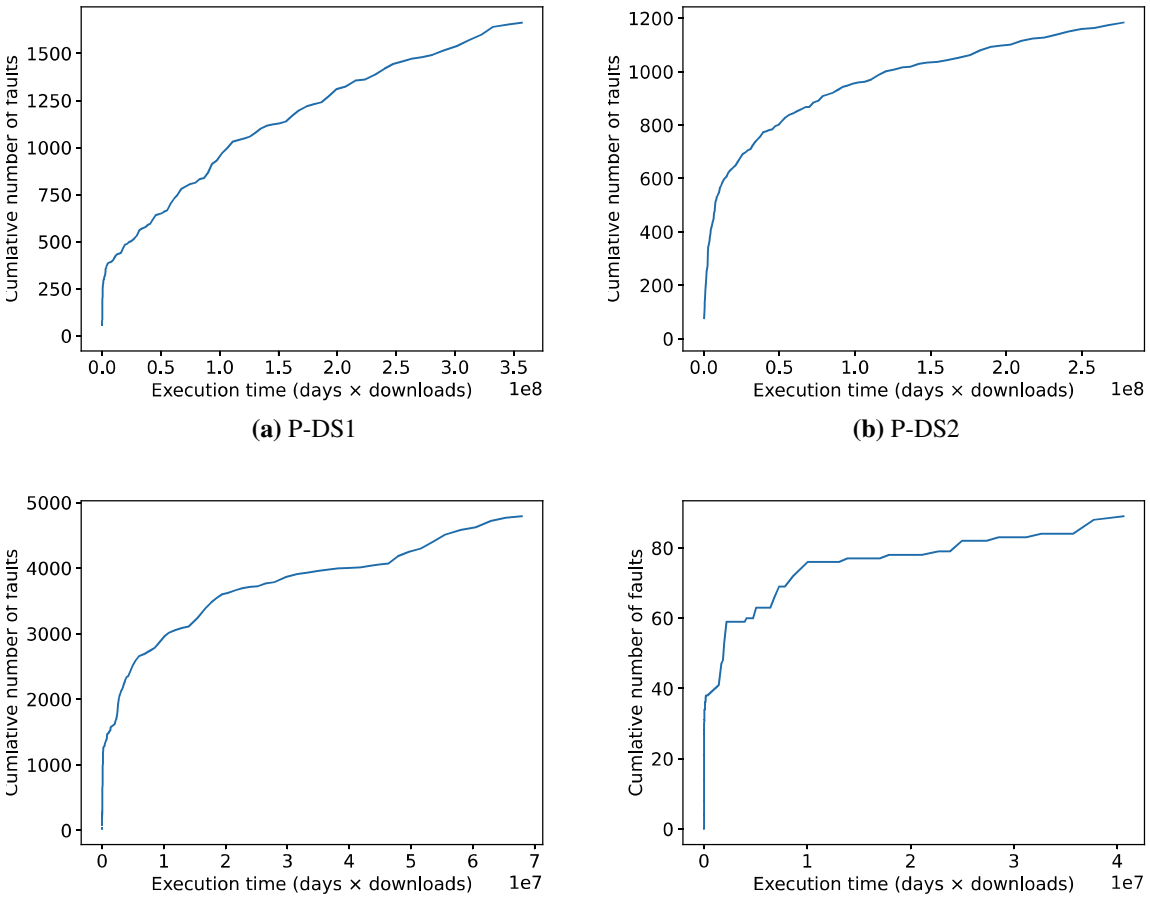
(b) P-DS2

(c) P-DS3

(d) P-DS4

Figure 4.1: The relationship between cumulative the number of faults and time.

From the linearly interpolated function, the average error is obtained by dividing the execution time into 100 equal parts. Table 4.2 shows the MAE of normalized SRGM based on NHPP. From Table 4.2, it can be seen that there is no significant difference between the curves of reported fault SRGM based on NHPP and confirmed fault SRGM based on NHPP. Also, it can be seen that there is no significant difference in each error rate depending on the number of downloads. These results show that the curves of the reported and confirmed fault SRGM based on NHPP are similar. It is also shown that there is the possibility to estimate the curve of certified fault SRGM based on NHPP from reported fault SRGM based on NHPP.



(a) P-DS1 **(b)** P-DS2

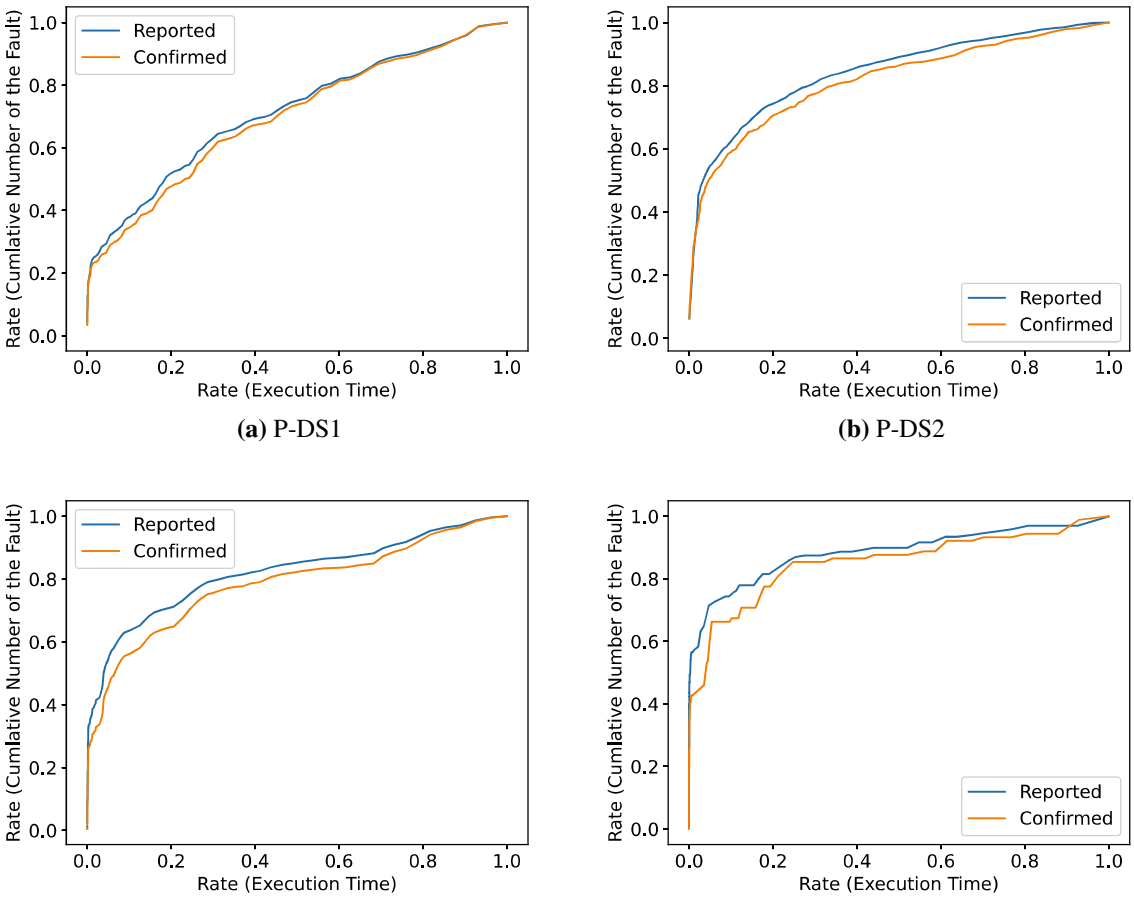
(c) P-DS3 **(d)** P-DS4

Figure 4.2: The relationship between cumulative the number of faults and execution time.

4.3.3 Relationship between confirmed fault and rejected fault

The relationship between confirmed faults and rejected faults was investigated for the SRGM curve based on NHPP. The number of reported faults consists of the number of confirmed fault and the number of rejected fault. Then, this thesis analyzed the relationship between confirmed fault and rejected fault. Figure 4.4 shows the scatter plot and regression line for confirmed fault and rejected fault. The regression line was derived using the following equation as follows:

$$\min_{w} ||Xw - y||_2^2, \tag{4.2}$$



(a) P-DS1 **(b)** P-DS2

(c) P-DS3 **(d)** P-DS4

Figure 4.3: The normalized SRGM.

where w is the coefficients of liner function. A no correlation test was also performed on each OSS. The significance level is set at 5% and the population correlation coefficient is assumed to be zero. Table 4.3 shows the correlation coefficients and p-values for confirmed and rejected faults for each OSS. From Table 4.3, the p-value is well below the significance level and the hypothesis is rejected for each OSS. Therefore, it can be seen that each OSS is not uncorrelated. Moreover, correlation coefficients exceeding 0.3 are observed for P-DS1, P-DS2, and P-DS3. It was found that when the number of reported faults is large, the correlation coefficient between rejected faults and confirmed ones tends to be proportional. On the other hand, when the number of reported faults is small, the results of P-DS4show that there is no proportional relationship.

Table 4.2: The evaluation result by MAE for normalized SRGM based on NHPP.

Total rank	OSS	MAE
3	P-DS1	1.702×10^{-2}
5	P-DS2	2.728×10^{-2}
63	P-DS3	3.339×10^{-2}
99	P-DS4	1.057×10^{-2}

4.3.4 Distribution of each type of the faults

It is known that the distribution of the number of faults occurrences follows Poisson distribution. Since the software fault detection process is NHPP, it does not show the perfect Poisson distribution, although it does show close form. On the other hand, reported fault includes rejected fault that didn't confirm. Therefore, the distribution of each type of fault is analyzed.

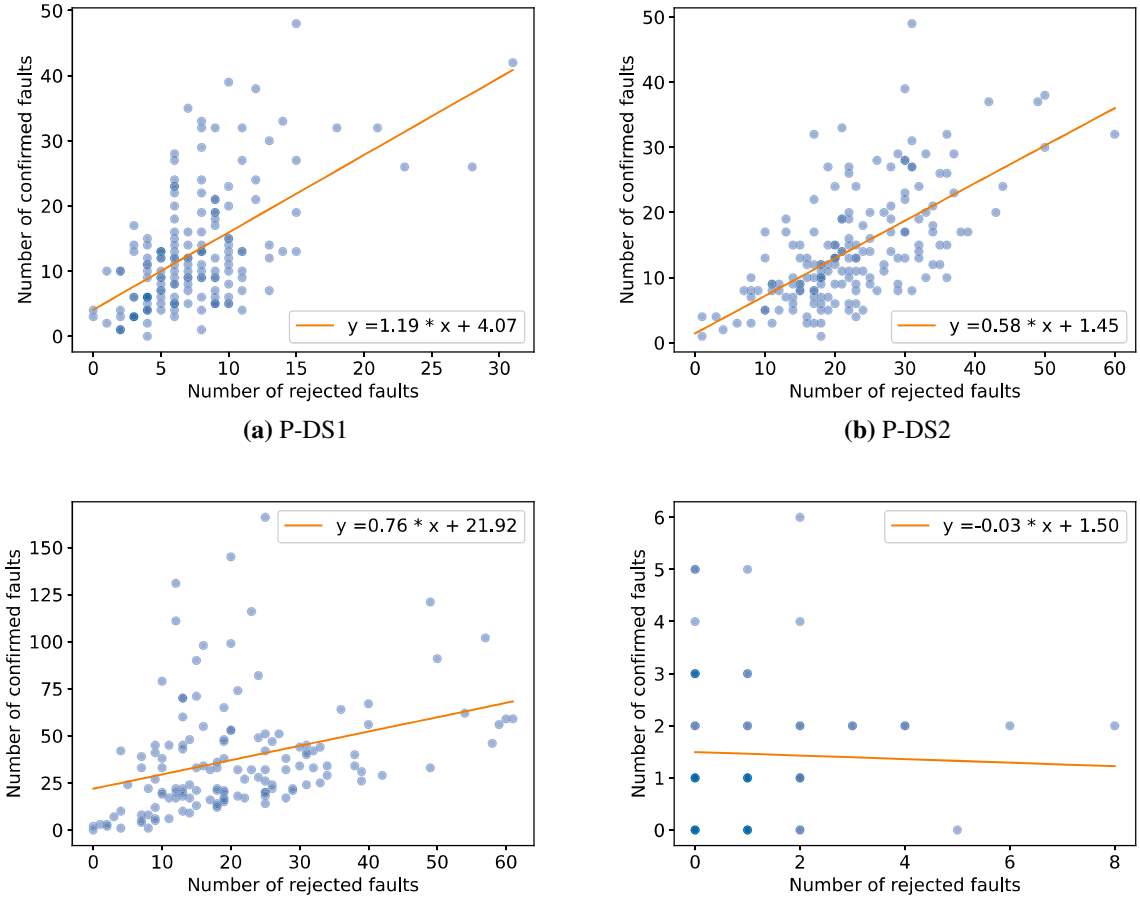
Figure 4.5 shows the distribution of fault types in each OSS.

Figure 4.5 shows that the confirmed and rejected fault results are closer to the strong Poisson distribution than the reported result for each OSS. In addition, the reported faults do not show the distribution that deviates significantly from the Poisson distribution. Therefore, the number of reported faults can be treated as an NHPP as well as the number of confirmed faults.

In SRGM based on NHPP, the mean value function is $H(t)$ in Eq. (2.1). Furthermore, it fluctuates continuously. Therefore, if the original measurements are data from NHPP, the trend is removed and the data show Poisson distribution centered at about zero. Then, moving average is subtracted from the measured values to check whether each data is NHPP. Figure 4.6 shows the distribution of the number of faults detrended by moving average for each OSS. Figure 4.6 shows the normal distribution for all fault types. Therefore, it can be seen that each fault follows NHPP.

Table 4.3: The evaluation result by correlation coefficient.

OSS	Correlation	P-value
P-DS1	5.680×10^{-1}	2.599×10^{-15}
P-DS2	6.405×10^{-1}	2.988×10^{-21}
P-DS3	3.415×10^{-1}	4.416×10^{-5}
P-DS4	2.177×10^{-1}	2.713×10^{-2}



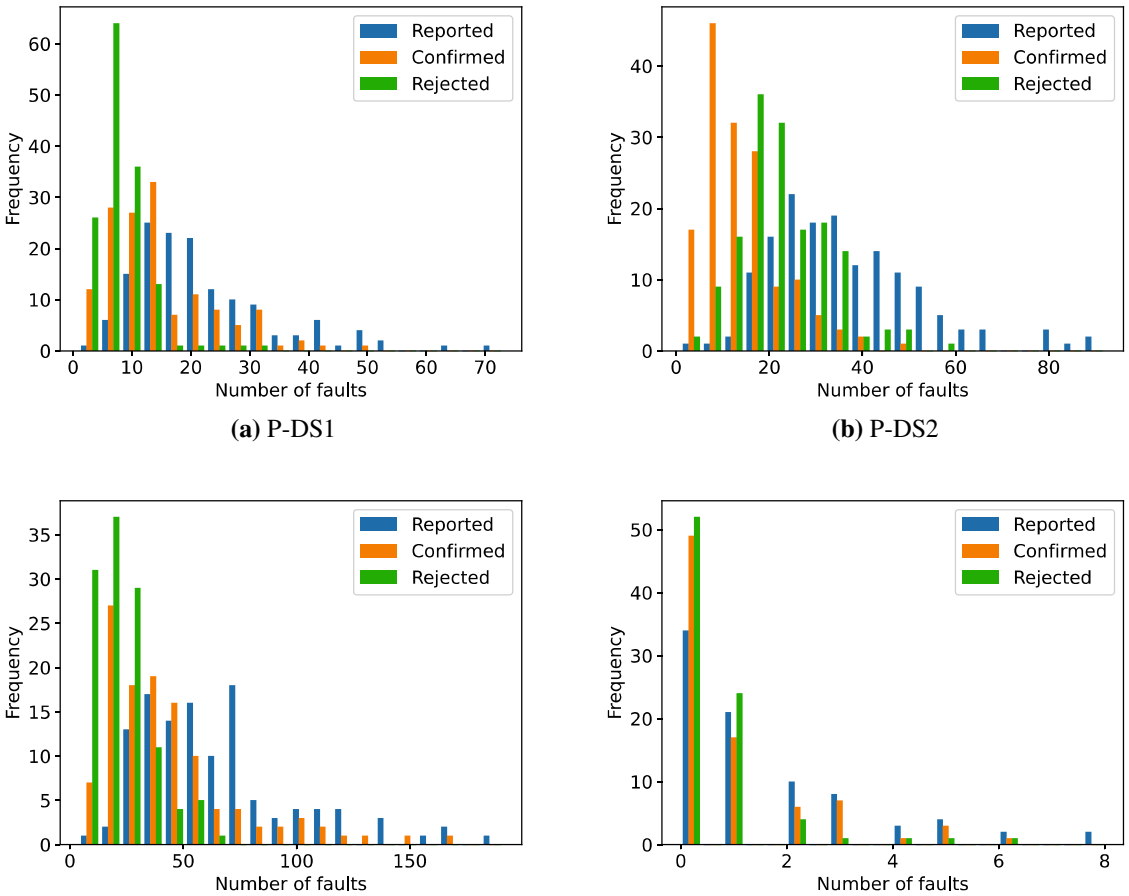
(a) P-DS1 **(b)** P-DS2

(c) P-DS3 **(d)** P-DS4

Figure 4.4: The scatter plot of the number of rejected faults and the number of confirmed ones.

4.4 Numerical examples

From Section 4.3, there was no significant difference between the curves of SRGM based on NHPP constructed with reported faults and SRGM based on NHPP constructed with confirmed faults. The analysis also showed that the proportionality relationship where the number of rejected faults increases as the number of confirmed faults increases, and that SRGM based on NHPP curves equivalent to those of confirmed faults can be obtained even if only reported faults are constructed. Therefore, each SRGM based on NHPP was estimated. The errors were then compared. Here, it is necessary to correct the SRGM based on NHPP for the number of reported faults and the curves for Confirmed faults. Therefore, the SRGM was normalized for each SRGM based on NHPP. This



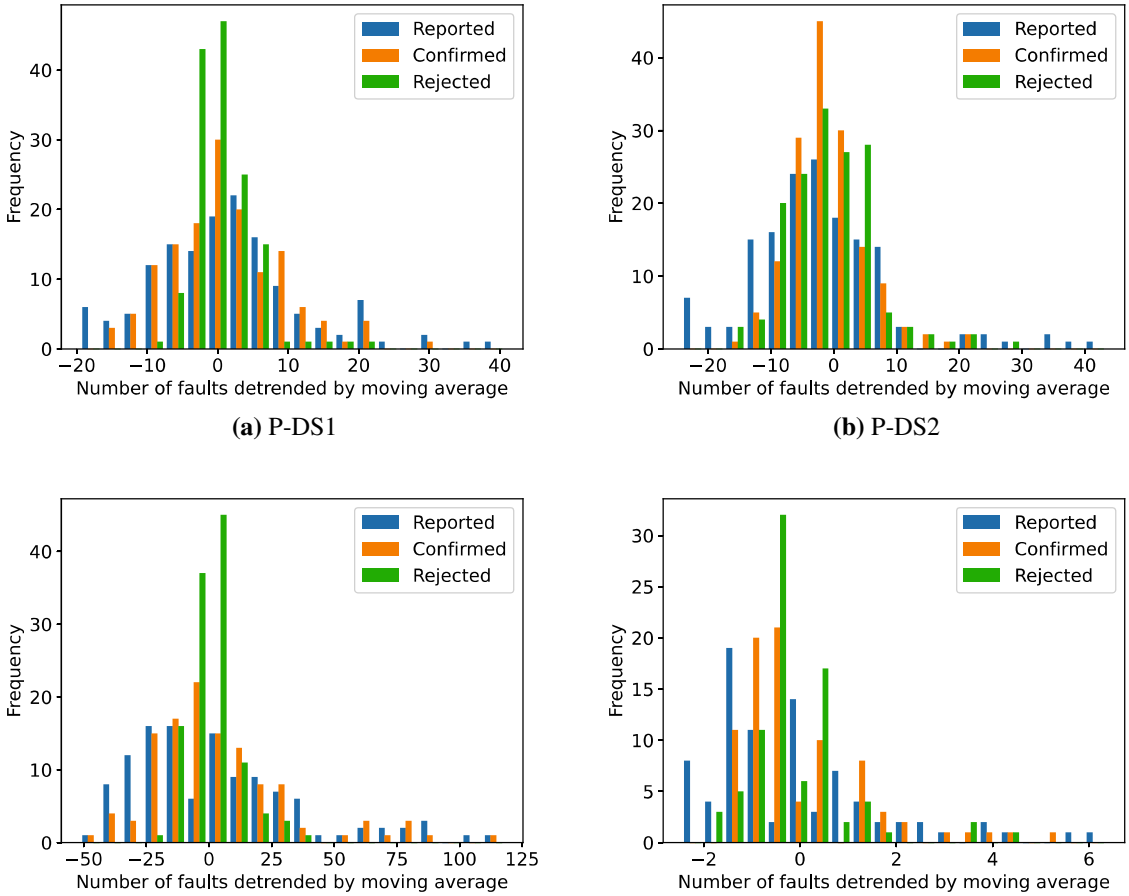
(a) P-DS1 **(b)** P-DS2

(c) P-DS3 **(d)** P-DS4

Figure 4.5: The distribution for each types of faults.

thesis applies the exponential model, the delayed S-shaped model, the proficiency S-shaped model, and the logarithmic poison execution time model proficiency model as the mean value function [8–11]. This thesis demonstrated estimation based on the maximum likelihood estimation method using each mean value function and compared the differences in curves. Figure 4.7 shows the results of SRGM based on NHPP estimation in each OSS. Table 4.4 shows evaluation results by MAPE.

From Figure 4.7, the MAPE of the SRGM based on NHPP curves is about within 5%. In particular, the logarithmic type shows the high accurate MAPE except for the P-DS1. It can be seen that the reported fault SRGM based on NHPP and confirmed SRGM based on NHPP curves are very similar. Furthermore, no significant difference in estimation accuracy is observed even for P-DS4,



(a) P-DS1

(b) P-DS2

(c) P-DS3

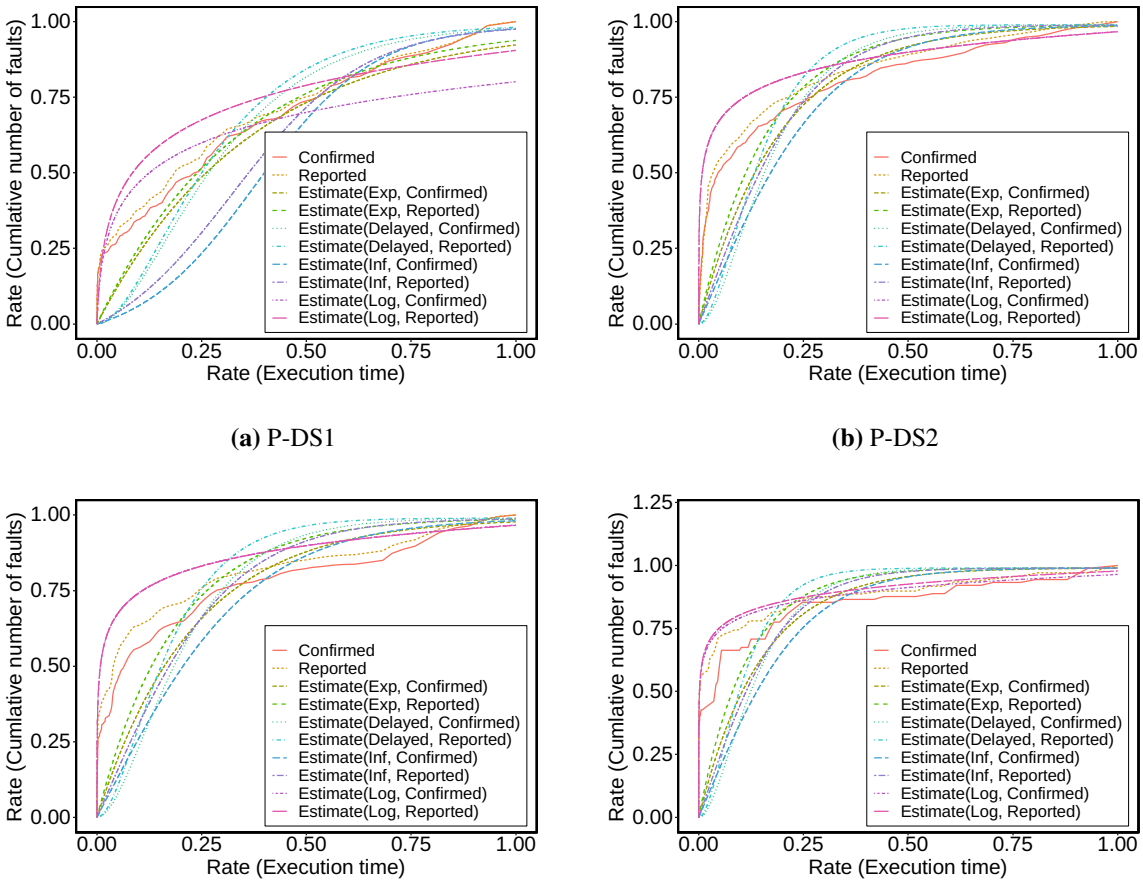
(d) P-DS4

Figure 4.6: The fault distribution based on the moving average for each types of faults.

Table 4.4: The evaluation results by MAPE of the reported fault and the actual SRGM based on NHPP in each OSS.

OSS	Exp	Delayed	Inf	Log
P-DS1	4.259×10^{-2}	4.226×10^{-2}	1.173×10^{-1}	1.123×10^{-1}
P-DS2	4.810×10^{-2}	5.734×10^{-2}	5.617×10^{-2}	3.190×10^{-4}
P-DS3	6.598×10^{-2}	7.423×10^{-2}	6.840×10^{-2}	9.447×10^{-4}
P-DS4	5.147×10^{-2}	6.468×10^{-2}	5.364×10^{-2}	1.505×10^{-2}

which has the fewest reported faults among the four. This thesis compared the models using MAPE and found that the number of reported faults can be used to predict the confirmed fault SRGM based on NHPP, even when the number of reported faults is small.



(a) P-DS1

(b) P-DS2

(c) P-DS3

(d) P-DS4

Figure 4.7: The estimated SRGM.

4.5 Summary

In this chapter, the relationship between reported faults and confirmed faults on the open BTS for popular HTTP libraries on PyPI was analyzed. Additionally, the distribution of each type of fault for OSS was examined. The applicability of the reported fault SRGM based on NHPP was considered for estimating the confirmed fault SRGM based on the NHPP curve by comparing the curves of the estimated mean value functions. As a result, it was found that the SRGM curves based on NHPP for confirmed and reported faults are very similar. It was also observed that confirmed and rejected faults are not uncorrelated. Furthermore, it was demonstrated that each type of fault follows an NHPP. The errors in the SRGM curves based on NHPP estimated using reported faults

were found to be small.

In future work, the applicability of this research will be further examined by analyzing fault data of OSS initial stage of development.

Papers Related in this Chapter

1. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “Estimation method based on SRGM using uncertain fault information on open BTS,” Recent Advances in Reliability and Maintenance Modeling, CRC Press, pp. 140–147, 2024.

Chapter 5

Major Version Model

5.1 Introduction

The internal structure of software changes with the updates. It is known that if the number of undiscovered latent faults in the software increases significantly, the probability of the detected faults increases and the number of reported faults increases. At this time, the phenomenon of software fault occurrence deviates from the assumptions of the model. Then, the estimation accuracy decreases. Therefore, several researchers have proposed the models considering the multiple releases [75–77]. Also, many models have been proposed that treat new releases as the change-points. Several researchers have proposed the models considering the change-points other than the software updates [78–81]. On the other hand, a large number of releases have been made for OSS in recent years. In particular, the developments of many OSS cause to many OSS releases. In this case, it is difficult to develop the model in case of many releases of new version.

In addition, when a new OSS is released, there are the users who continue to use the old OSS out of concern about the impact on existing systems. At this time, it is considered that the debugging ability of the new version of OSS will decrease. However, no research has been clarified the effect on the debugging ability.

This thesis collects an actual usage data of OSS from the software repository by repository mining. Moreover, this thesis analyzes the effect of software updates for SRGM from the perspective

of the semantic versioning. This thesis also compares the version-specific download data with the status recorded on the software repositories. Furthermore, this thesis proposes the SRGM based on the analysis results.

5.2 Software version

5.2.1 Versioning

Until recently, there was no standard for the format of the software versioning. As a result, the developers used their own versioning styles, utilizing numbers or letters in various ways. These formats were inconsistent in terms of the scope of changes and the formatting of version numbers. It makes difficult to grasp from version number in case that the changes had been made in the new version. It was especially problematic in the systems with many dependencies. Then, version management required significant effort. As simplify version management, most software uses Semantic Versioning today [82]. It was proposed in 2010. The semantic versioning standardized the versioning formats. It had become customary in software development at the time. The version numbers in semantic versioning are expressed in the form $X.Y.Z$. The version numbers in the semantic versioning are changed as follows:

- Major version (X): When making changes that are not backward compatible.
- Minor version (Y): When making improvements or deprecation while maintaining backward compatibility.
- Patch version (Z): When fixing faults while maintaining backward compatibility.

When the higher version number is changed, the lower numbers are reset to 0. Additionally, when the Major version is 0, it indicates that the software is under development. It does not guarantee the backward compatibility. The developers can utilize these version numbers to implement flexible version management. For example, it is possible to immediately upgrade only the patch version updates.

5.2.2 Version control

Since OSS have multiple versions, it is necessary to specify the version when adopting them in the software. In this case, the version specifiers are used for the version control. In the python, the version specifiers standardized in python enhancement proposals (PEP) 508 [83] are utilized. The specifiers standardized in PEP 508 are shown in Table 5.1. Figure 5.1 shows the example of configuration file of the everyonecancode [84] that is released by Microsoft.

Table 5.1: The specifiers available for version specification.

Specifier	Behavior
==	Install the specified version
!=	Install any version except the specified one
>=	Install versions greater than or equal to the specified version
<=	Install versions less than or equal to the specified version
>	Install versions newer than the specified version
<	Install versions older than the specified version
~=	Install the specified major and minor version
,	Specify multiple conditions
===	Equivalent to ==

```
...
ujson==5.9.0
urllib3==2.2.1
uvicorn==0.29.0
uvloop==0.19.0
...
```

Figure 5.1: The configuration file in everyonecancode.

As shown in Figure 5.1, the external dependency packages and their versions used by the package are recorded. By tracking the change history of these configuration files in the software repository, it is possible to understand the adoption status of the software. This thesis utilizes this feature to make a repository mining on the software repository and collect data regarding the adoption status.

5.3 Data collection from repositories

5.3.1 Analysis target

To investigate the relationship between the number of downloads and the adoption ratio, the adoption status was obtained from public repositories on GitHub under the following conditions [43].

To ensure non-biased data, the OSS with the highest number of downloads was selected. The top five downloads from February 22nd to February 29th, 2024, are shown in Table 5.2.

Table 5.2: Top 5 most downloaded software on PyPI from February 22nd to February 29th, 2024.

Rank	Software name	Number of downloads
1	boto3	294967601
2	botocore	150476752
3	urllib3 (P-DS1)	49799383
4	wheel	43085892
5	requests	42860438

This thesis focuses on the effect of version specification of developer. The boto3 and the botocore are developed for cloud service [85,86]. It may be forced to be updated depending on the cloud service. Therefore, this thesis selected the P-DS1.

5.3.2 Retrieve data from software repository using repository mining

Data collection was focused on major version 2, as it has a large amount of data available on the software repository. The following steps were taken to collect data on the update status of version 2 of P-DS1 from public repositories. Versions 2.0.2 to 2.2.1 were targeted, excluding any versions that had been retracted. The data was collected from the software repository as follows:

- Search for files named “requirements.txt” that specify version 2 of P-DS1 using specifiers in their content.
- Based on the obtained repository, project name, and file path, search for commits and extract the date and version.

This thesis obtained the number of data points resulting from query executions for each specifier on the repository to investigate whether each specifier has a sufficient amount of data for analysis. The results are shown in Table 5.3.

Table 5.3: The amount of data for each specifier in the query execution.

Specifier	==	>=	<=	≈=	===
Count	68.4k	402	32	288	35

As shown in Table 5.3, the results for each search, except for the specifier "==," were less than 1% of the search results for the specifier "==." Therefore, this thesis collected the data from repository mining using "==." This thesis analyzed the extracted data from the perspective of major, minor, and patch updates, respectively. The software was categorized into two groups: software adopting P-DS1 for the first time and software that had undergone one or more version updates. The adoption trends of new adoptions and update adoptions were then analyzed. Furthermore, this thesis analyzed the relationship between the software repository data and the number of downloads by the version.

5.4 The effect of software update

5.4.1 Major version

This thesis analyzed the trends in major version adoptions. Among the retrieved update histories, 3,401 repositories included version 1. When these repositories first transitioned to version 2, it can be considered that they make a major version upgrade. The results of the first updated version are shown in Table 5.4. Figure 5.2 shows the number of days obtained for the major version to be updated after the release date.

It is evident that version 2.1.0 is the most common in the updated OSS. Additionally, many users take the considerable amount of time to update to the new major version while some users update immediately after release. Therefore, it is generally understood that it takes time for major versions to be updated. The version first adopted in major version 2 was analyzed. Table 5.5 shows the first adopted version. Additionally, the number of days from the release of the major version to the

Table 5.4: The updated major version.

Version	Count
2.0.1	6
2.0.2	16
2.0.2	454
2.0.3	351
2.0.4	459
2.0.5	72
2.0.6	200
2.0.7	404
2.1.0	607
2.2.0	140
2.2.1	692

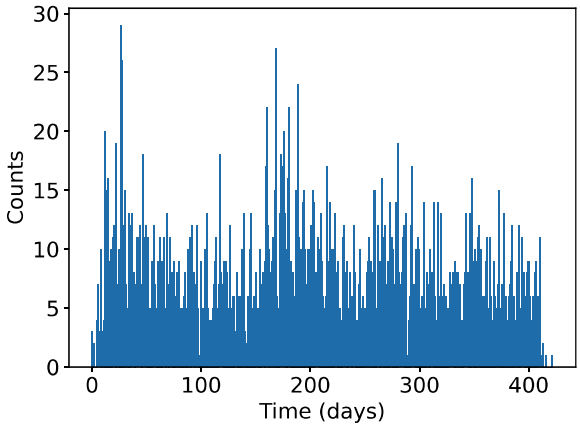


Figure 5.2: The number of new adoptions for each version.

adoption of major version 2 is shown in Figure 5.3.

These results show that it takes time for the new major version to be widely adopted by users. This thesis analyzed the similar trend in the number of downloads. Figure 5.4 shows the trend in the number of downloads for major versions. Even after version 2 was adopted, the number of downloads for version 1 has not decreased. Furthermore, the number of downloads for version 2 is also increasing such as version 1. The download numbers also shows that it takes time for the new major version to be widely adopted by users.

Table 5.5: The updated major version.

Version	Count
2.0.0	6
2.0.1	19
2.0.2	3187
2.0.3	4096
2.0.4	5498
2.0.5	1516
2.0.6	2184
2.0.7	6153
2.1.0	11851
2.2.0	2812
2.2.1	5593
2.0	1
2.1	1

Table 5.6: The number of adoptions for each version.

Version	Release Date	New	Updated
2.0.2	May 4, 2023	1596	578
2.0.3	June 7, 2023	2529	704
2.0.4	July 20, 2023	3481	1161
2.0.5	September 20, 2023	733	379
2.0.6	October 3, 2023	1064	1003
2.0.7	October 18, 2023	2872	1408
2.1.0	November 13, 2023	9432	1833
2.2.0	January 31, 2024	1748	762
2.2.1	February 18, 2024	16360	2346

5.4.2 Minor version

This thesis analyzed the trends in minor version adoptions. Table 5.6 shows the release date of each version, as well as the number of new adoptions and update adoptions made during the latest release period for each version.

From Table 5.6, it can be seen that the ratio of newly adopted software to the updated software is not constant. In particular, the ratio of updated software fluctuated significantly from about one-quarter to one-half when minor updates were made from version 2.0.7 to 2.1.0. Thus, the difference in trend can be seen between new and updated adoptions. The focus is on the average number of

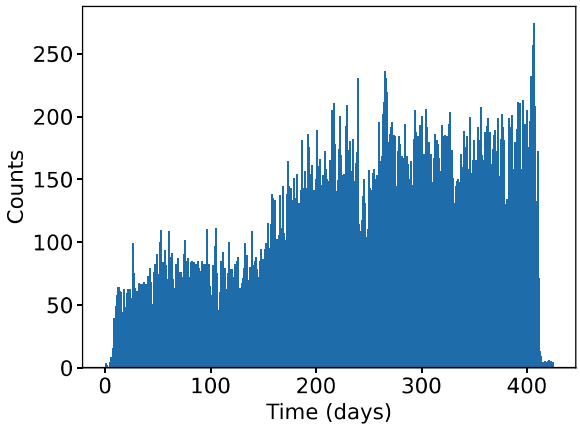


Figure 5.3: The number of new adoptions for each version.

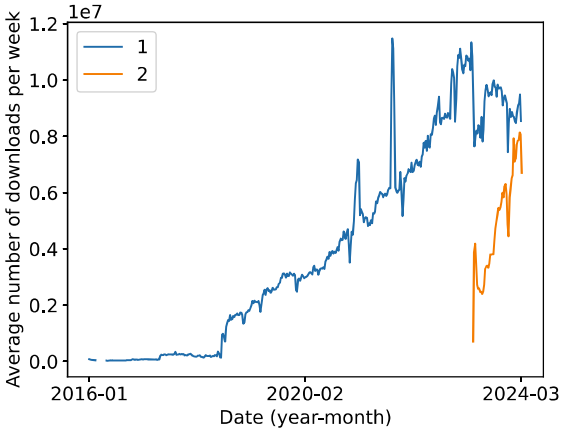


Figure 5.4: The number of downloads for each version.

adoptions. The average number of adoptions is calculated as follows:

$$n = \frac{a}{t}, \tag{5.1}$$

where n is the average number of adoptions, a is the number of adoptions, and t is the latest release period. Table 5.7 shows the update contents and the average number of adoptions for each version.

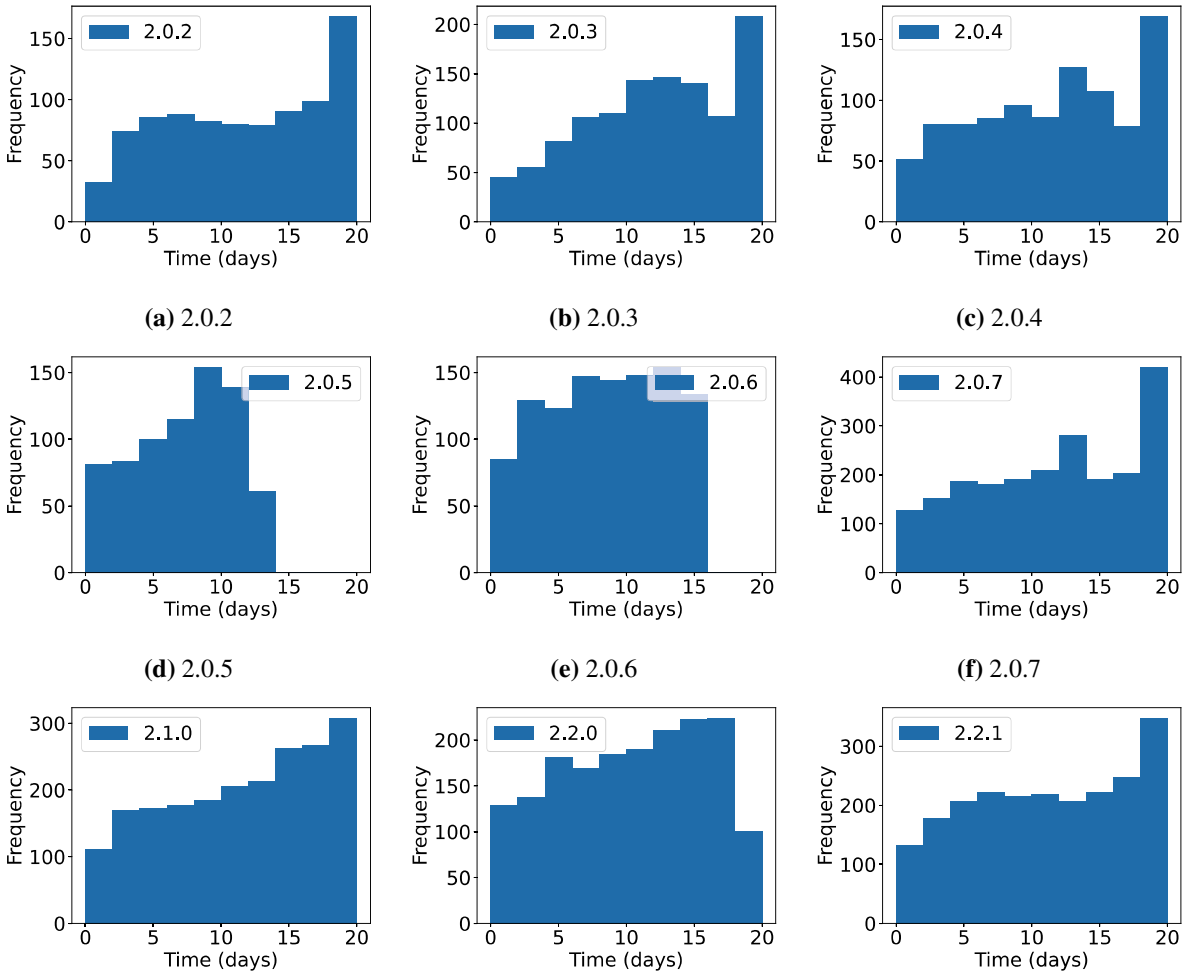
From Table 5.7, it can be seen that the average number of adoptions for newly adopted software tends to increase with each version. Additionally, the average number of adoptions for the updated software tends to decrease as the release period becomes longer.

Table 5.7: Updates and average number of adoptions for each version.

	Average number of adoptions		
Version	New	Updated	Latest release days
2.0.2	46.24	16.74	34.51
2.0.3	59.92	16.68	42.20
2.0.4	55.56	18.53	62.65
2.0.5	59.05	30.53	12.41
2.0.6	70.85	66.79	15.01
2.0.7	107.24	52.57	26.77
2.1.0	120.74	23.46	78.11
2.2.0	92.35	41.13	18.52
2.2.1	135.87	19.48	120.40

The distribution of version adoption status was analyzed. Figure 5.6 shows the adoption status for new adoptions. Figure 5.6 shows the adoption status for updates. The number of adoptions is low immediately after release and tends to increase over time in new adoptions. In contrast, regardless of minor or patch versions for updated adoptions, the highest number of adoptions is observed immediately after release according to the decrease trend. Moreover, the number of users who update immediately after release is smaller compared to patch versions in case of the minor updates to 2.1.0 and 2.2.0. The users refrained from updating immediately due to consideration of the impact of the update. Furthermore, there were users who updated to older minor versions even after minor updates. As with major versions, there were users who refrained from adopting minor version updates immediately after the update.

Figure 5.7 shows the trend of the adoption status of each version for new adoptions. Figure 5.8 shows the trend of the adoption status of each version for update adoptions. Figure 5.9 shows the daily totals for new adoptions and update adoptions. Even after a minor update, there are new and updating users who continue to use the old minor version. Additionally, new minor versions are more adopted earlier than major versions. Figure 5.10 shows the daily download counts for each version. The same trend in the number of downloads as in the adoption situation on the repository. From the perspective of semantic versioning, minor updates are generally guaranteed to be backward compatible. Therefore, it is thought to be easier for users to adopt than major versions.

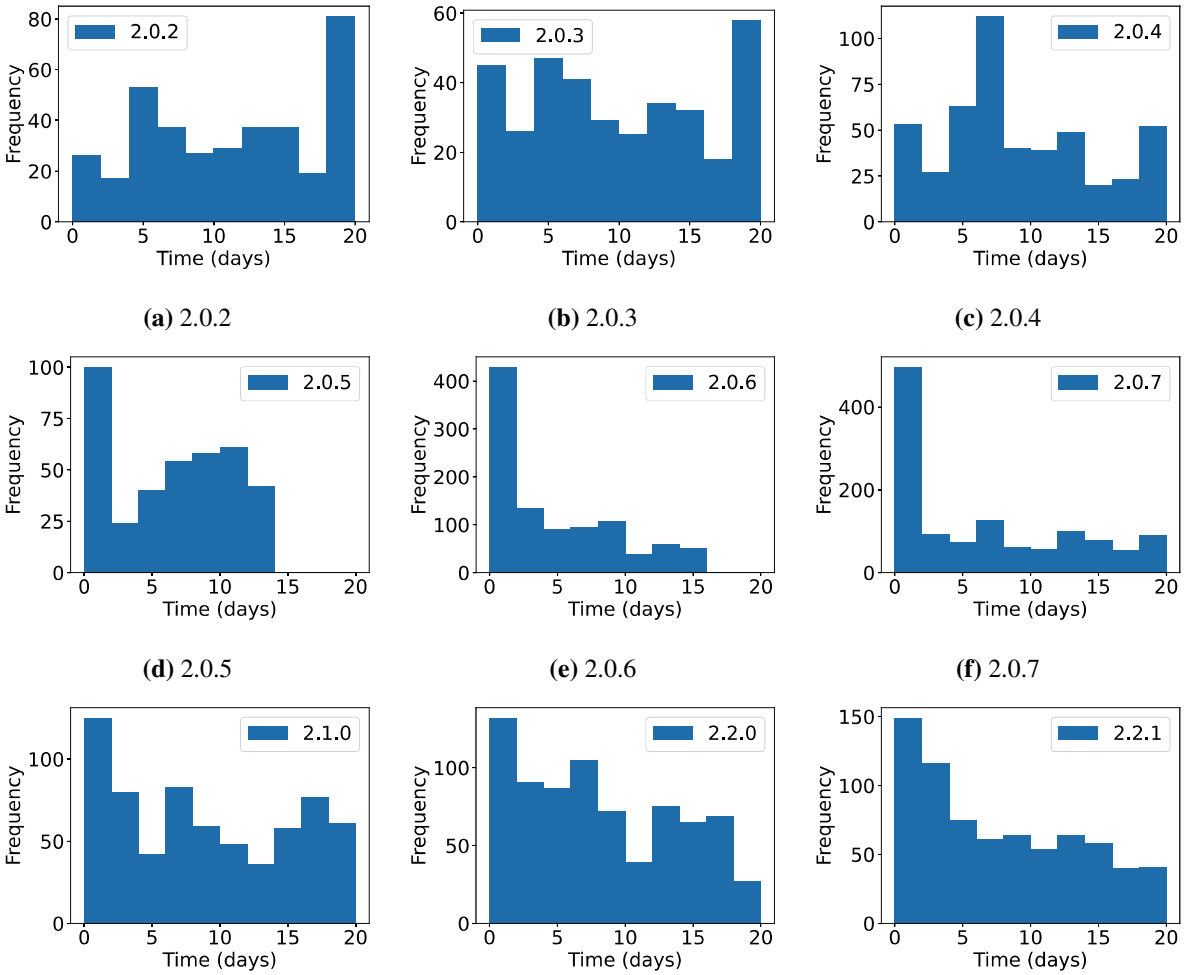


(a) 2.0.2 **(b)** 2.0.3 **(c)** 2.0.4

(d) 2.0.5 **(e)** 2.0.6 **(f)** 2.0.7

(g) 2.1.0 **(h)** 2.2.0 **(i)** 2.2.1

Figure 5.5: New adoption.



(a) 2.0.2 **(b)** 2.0.3 **(c)** 2.0.4

(d) 2.0.5 **(e)** 2.0.6 **(f)** 2.0.7

(g) 2.1.0 **(h)** 2.2.0 **(i)** 2.2.1

Figure 5.6: Updated adoption.

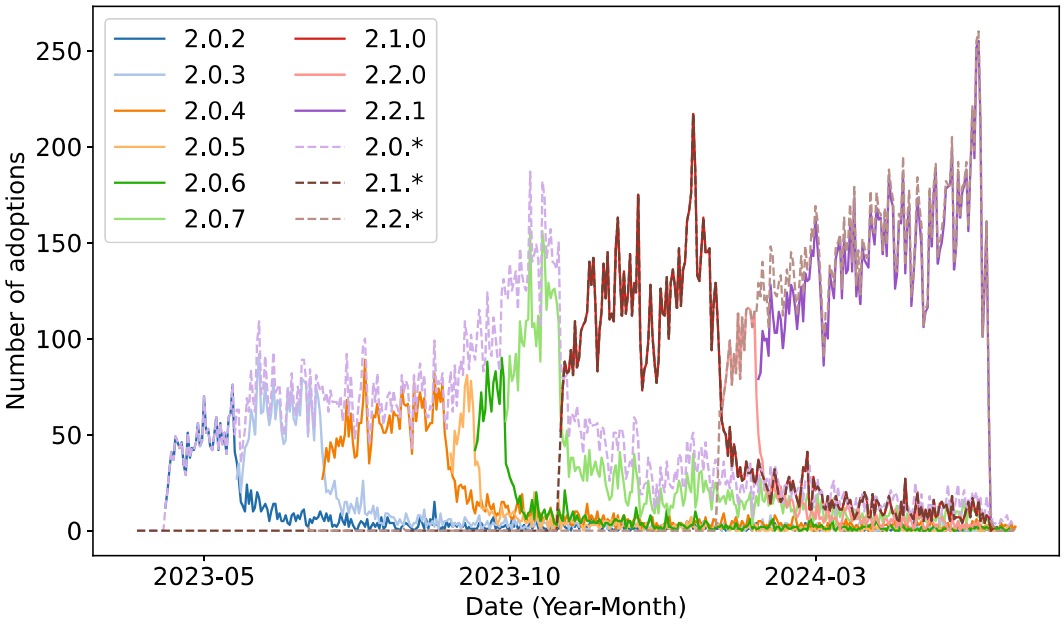


Figure 5.7: The number of new adoptions for each version.

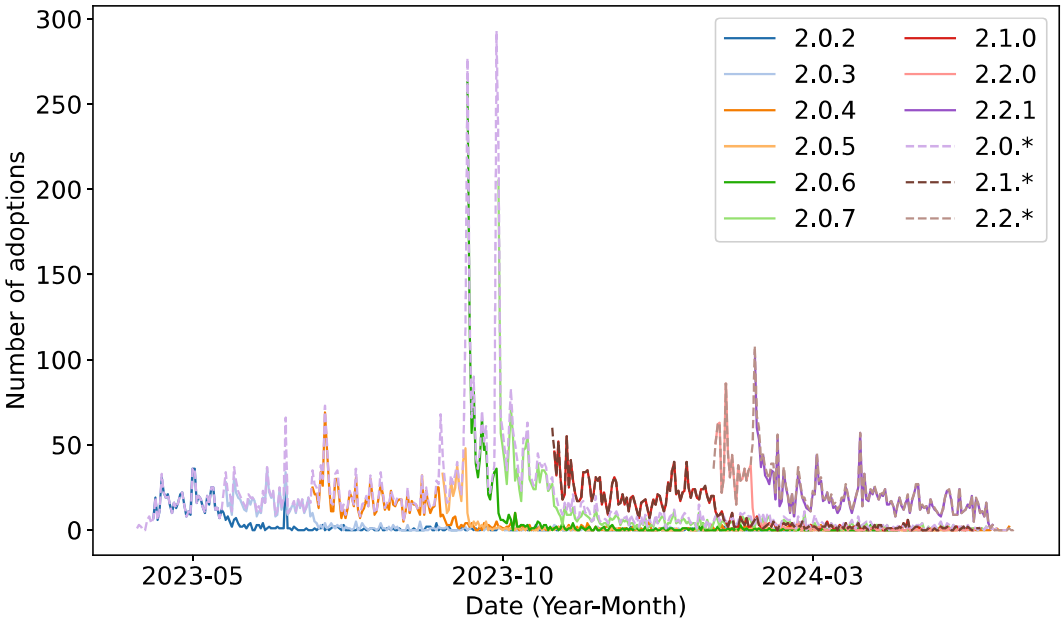


Figure 5.8: The number of updated adoptions for each version.

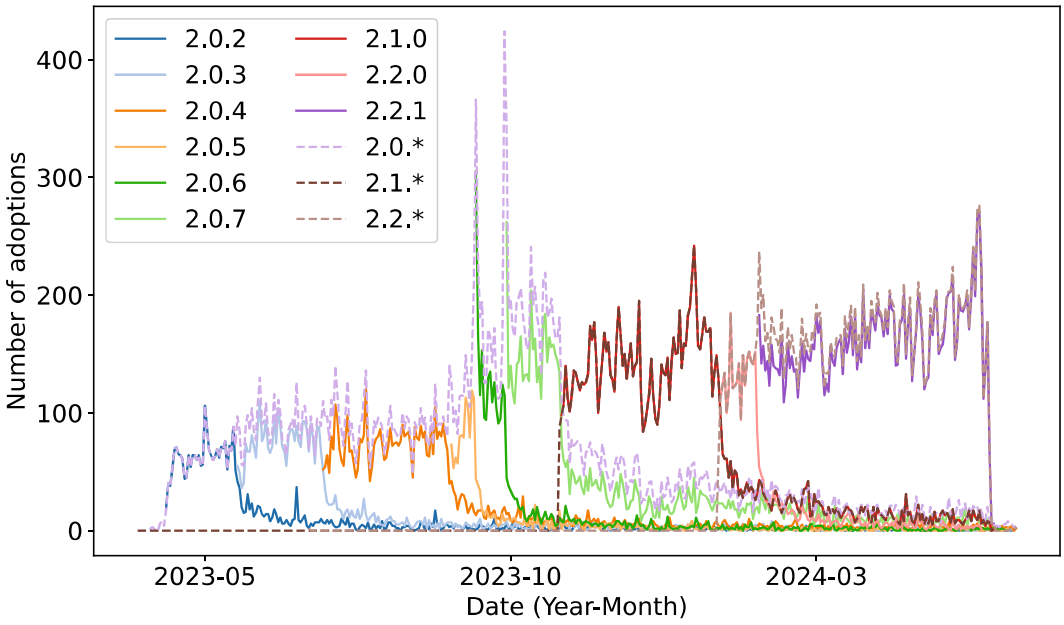


Figure 5.9: The number of adoptions for each version.

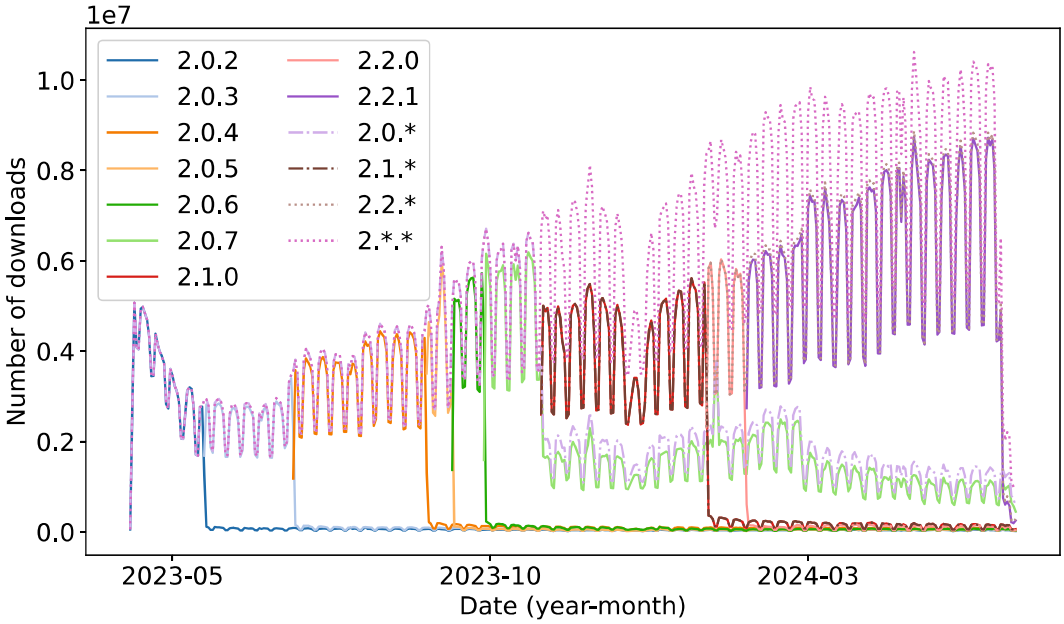


Figure 5.10: The number of downloads for each version.

On the other hand, there are some users who are temporarily refraining from updating because it takes time to update the version.

5.4.3 Patch version

This thesis analyzed the trends in patch version adoptions. Figures 5.5 and 5.6 shows that patch versions are used by many users soon after release. Moreover, Figures 5.7-5.9 shows that old patch versions are no longer used after a patch update. Figure 5.10 shows a similar trend in the number of downloads. From the perspective of semantic versioning, patch version updates are intended only to fix faults. Therefore, there are no concerns about backward compatibility. It is likely why many users are quick to adopt them.

5.5 Proposed method

From Section 5.4, the following versions can be understood for each type of update:

- Major version: Users decrease in the early stages of the release, and a certain number of users continue to use the older major version.
- Minor version: After the release, users update to the new minor version, but a certain number of users continue to use the older minor version.
- Patch version: After the release, users immediately adopt the new patch version.

In particular, it is confirmed that many users tend to refrain from using a major version immediately after its release. This is likely due to the fact that, according to the rules of Semantic Versioning, the backward compatibility is not guaranteed during a major version upgrade, leading some users to continue using the older major version. At this time, as the number of users adopting the latest major version decreases upon its release, the debugging activity on the OSS source code also decreases. On the other hand, since new major version includes changes that do not guarantee backward compatibility, as per the rules of Semantic Versioning, the number of latent faults significantly increases. In this case, a mixed process of reduced debugging capability and increased faults occurs during the release of a major version.

Regarding debugging capability, the specific debugging capability of a version can be calculated by applying Eq. (3.1) to the number of downloads of the new major version. Additionally, when the number of latent faults increases significantly, the probability of fault detection also increases markedly, similar to when new software is released. Therefore, this thesis proposes the method to divide the software reliability growth curve at major versions and use the number of downloads of the major version to calculate execution time. By using the number of downloads for a specific major version, the proposed method is expected to improve the estimation accuracy.

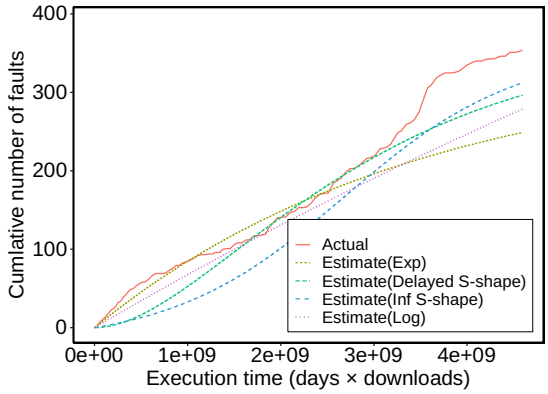
5.6 Numerical examples

The proposed method was applied to version 2 of P-DS1, and an SRGM. The mean value functions are used the exponential, delayed S-shaped, inflection S-shaped, and logarithmic Poisson execution time models, respectively. Figure 5.11 shows the estimation results using the execution time for all versions, only major versions, and only the download counts of the latest version. Table 5.8 shows the evaluation result by MAE, MAPE, and AIC. In most cases, the download counts of major versions shows better results. In particular, the performance of the inflection S-shaped SRGM when using major versions is remarkably high. Additionally, it is confirmed that the estimation results for the model using the latest version's download counts and the model using the major version's download counts are the same result.

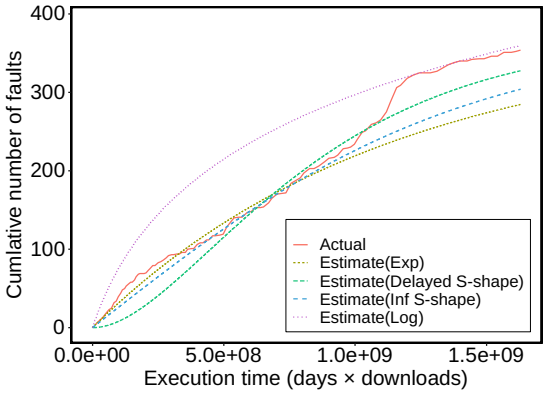
5.7 Summary

In this chapter, the usage trends of P-DS1 were investigated using data from software repositories. Moreover, this thesis analyzed the relationship with the number of OSS download. The analysis results are as follows:

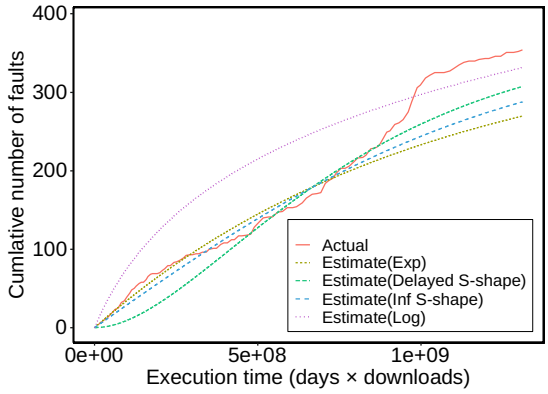
- There is the proactive trend in updating and using the patch versions.
- There is also a positive trend observed in the updating and use of minor versions. The trend was weaker than for the patch versions.



(a) The number of downloads for the all version.



(b) The number of downloads for each major version.



(c) The number of downloads for the latest version.

Figure 5.11: Comparison of downloads for different versions.

- The similar proactive trend in patch version updates was observed in the software repositories, comparable to download numbers.
- The passive trend was observed in adopting new major versions in the software repositories.

Additionally, this thesis proposed the version-specific reliability growth curves based on the analysis results. Furthermore, this thesis analyzed the range of versions used in calculating the execution time for the version-specific reliability growth curves. As the result, the proposed models estimated using download counts of major versions showed better estimation results in many existing models compared to models using all versions or only the latest version for execution time. Moreover, the learning S-curve model provided the best estimation results. The proposed model enables

Table 5.8: Comparison of Results.

(a) The result of all versions.

	Exp	Delay	Inf	Log
MAPE	0.19	1.47	1.15	0.29
MAE	31.33	25.69	38.31	31.31
AIC	616.24	608.10	562.81	542.34

(b) The result of the latest major version.

	Exp	Delay	Inf	Log
MAPE	0.14	1.23	0.18	0.31
MAE	22.16	22.40	19.35	52.02
AIC	545.45	570.65	523.44	606.12

(c) The result of the latest version.

	Exp	Delay	Inf	Log
MAPE	0.14	1.06	0.16	0.30
MAE	25.77	25.13	22.47	50.88
AIC	590.74	587.27	559.63	654.22

more accurate reliability evaluations in OSS where frequent updates occur. Additionally, the proposed method, by constructing reliability growth models for each major version, could evolve into a method for selecting major versions in OSS. On the other hand, the accuracy may decrease when the software has few users because this study was applied to OSS with numerous downloads on PyPI. In addition, the estimation accuracy may decrease for OSS when the OSS does not follow to the semantic versioning.

In future work, the method will be applied to OSS in the early stages of development to analyze its effect. Additionally, the impact on estimation accuracy will be investigated when OSS does not adhere to semantic versioning.

Papers Related in this Chapter

1. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “A Study on the Relationship Between OSS Adoption and Download Numbers Based on Repository Mining,” IECIE Technical Report[Reliability], Shogai Gakushu Center Kiran (Muroran, Hokkaido), Vol. 124, No. 135, pp. 7–12, July 27, 2024 (in Japanese).

2. S. Miyamoto, Y. Tamura and S. Yamada, “A method of reliability assessment based on trend analysis for open source software,” International Journal of Reliability, Quality and Safety Engineering, Vol. 31, No. 4, World Scientific, pp. 2450004-1–2450004-15, 2024.

3. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada,“Prediction of the OSS Number of Faults Based on a Reliability Growth Model Considering User Usage Patterns,” IECIE Technical Report[Reliability], Matsue Terrsa, Vol. 123, No. 399, pp. 1–5, February 29, 2024 (in Japanese).

4. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “A Study on the Impact of Software Updates on OSS Reliability Growth Curves,” Proceedings of the 2023 Autumn Research Presentation Conference of the Project Management Society, Yamaguchi University Tokiwa Campus, pp. 1–6, August 31 – September 1, 2023 (in Japanese).

Chapter 6

Share Ratio Model

6.1 Introduction

In chapter 5, this thesis focuses on the improving of estimation accuracy by applying SRGM to download numbers by major version. By utilizing version-specific download data, it becomes possible to estimate the reliability of OSS with higher precision. However, in some cases, such as OSS on node.js, the version-specific download numbers cannot be obtained. In such OSS, the method proposed in chapter 5 cannot be applied. On the other hand, several researchers have studied modeling the number of shares and users in the past. Then, the method proposed in chapter 5 can be replicated through modeling by using the share ratio to convert the total download numbers into version-specific download numbers. Therefore, this thesis focuses on the share ratio of major versions in this chapter. The method to construct an SRGM with high accuracy by utilizing the download numbers of all versions is proposed. In this chapter, the analysis of P-DS1 continues, as presented in chapter 5.

6.2 Prediction of the share ratio

Predicting the number of users is effective in estimating future business prospects in product development and marketing. For example, businesses can determine strategies based on forecasts of how many users will be needed in the future once the number of product users begins to increase.

It is known that the number of users tends to show a growth ratio that gradually decreases and converges to a certain value, similar to market size. Many studies have been done to use this feature to predict the number of users, market size, and technology life cycles [87–92]. The SRGM has similar concept in estimating fault numbers. In these researches, Gompertz function (Gomp) and exponential function have been used factually.

Exponential curves are models used in SRGM, where the value increasing per unit time is proportional to the remaining quantity. It can be modeled by replacing a with the saturation value of Eq. (2.2).

The Gompertz curve is a formula that shows that the amount of increase decreases exponentially over time [93]. It has the characteristic of initially increasing gradually. Finally, it converges to a saturation point. It is expressed as follows:

$$f(t) = Kb^{\exp\{-ct\}}, \tag{6.1}$$

where K is the saturation point, b is the displacement, c is the growth ratio. Similar to SRGM, it is possible to estimate the parameters of the curve by applying data to the model and using the maximum likelihood estimation method. In this chapter, this thesis constructs a model that takes into account the increase in user share rates by utilizing these models.

6.3 Share ratio of major version

In order to examine a model that takes market share into account, changes in market share for P-DS1 are analyzed. The data used for the analysis is the same as that used in chapter 5. The number of downloads for each major version is tallied, and the share ratio is calculated. P-DS1 has major versions 0, 1, and 2. Version 0 is excluded from the statistics from the perspective of semantic versioning. The changes in market share are shown in Figure 6.1.

It can be seen that it has had the certain share since the initial release of version 2. The PyPI returns the latest version when the user does not specify a version. It may be one of the factors for the existence of the initial share ratio. There are also observed points where the initial share

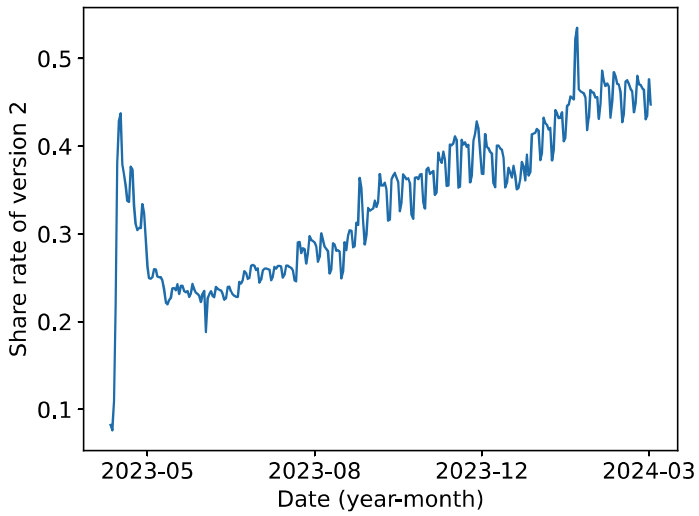


Figure 6.1: The share ratio trend of P-DS1 version2.

ratio temporarily declines. The Version 1.2.6.16 was released on May 23, 2023. It is thought to be due to the increase in downloads of version 1. As time goes on, the share of version 2 is gradually increasing.

Figure 6.2 shows the change in share ratio over time. It can be seen that there are no major differences in the curves since the share ratio increases gradually. From these data, it can be seen that modeling is not difficult since there is no extreme fluctuation in the share ratio in version 2 of P-DS1.

6.4 Proposed method

6.4.1 Share ratio function

In order to analyze the overall trend of the share ratio, the maximum likelihood estimation method was used to derive the estimated curve for the increasing trend. It is known that the number of software users follows an exponential or S-shaped curve.

- Certain the number of users already exist when new version is released.
- Finally, the share ratio reaches saturation value.

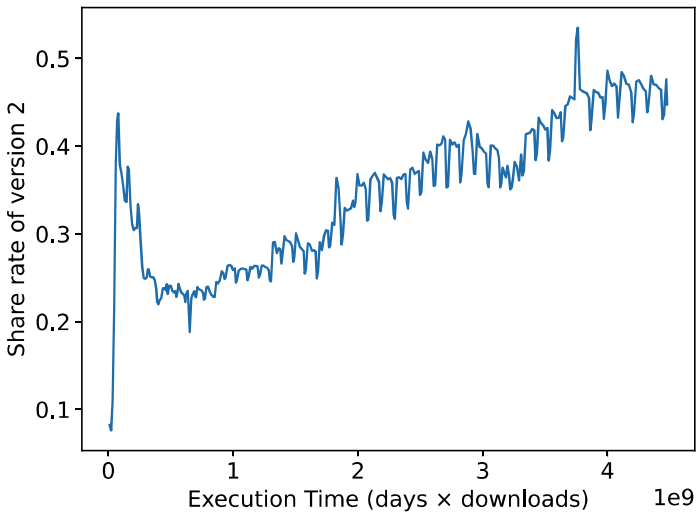


Figure 6.2: The estimated result using the corrected execution time.

Therefore, the share ratio trend function is derived, considering the offset of the initial and final share ratios. Considering the offset and the final share ratio in the exponential function, it is as follows:

$$e(t) = a(1 - \exp\{-b(t + t_{off})\}), \tag{6.2}$$

where a is the final share ratio, b is the ratio parameters, and t_{off} is the offset of initial share ratio. If the latest version eventually gets all the shares, the exponential share ratio function is as follows:

$$e(t) = 1 - \exp\{-b(t + t_{off})\}. \tag{6.3}$$

Considering the offset and the final share ratio in the Gompertz function, it is as follows:

$$f(t) = Kb^{\exp\{-c(t+t_{off})\}}, \tag{6.4}$$

where K is the final share ratio, b is the displacement, c is the growth ratio, and t_{off} is the offset of initial share ratio. If the latest version eventually gets all the shares, the Gompertz share ratio

function is as follows:

$$f(t) = b^{\exp\{-c(t+t_{off})\}}. \tag{6.5}$$

6.4.2 The correction functions

This thesis proposes the model that enables highly accurate estimation even when using data from those distributors. This thesis corrects the execution time by considering the percentage for users of the latest major version. The corrected execution time t_{mi} using the usage ratio of the latest version can express as follows:

$$t_{mi} = t_i r(t_i), \tag{6.6}$$

where $r(t_i)$ is the share ratio of the latest version at time t_i, t_i is the execution time for all version at t, and i is the number of observations, respectively. The share of the latest major version is expected to increase and reach 1 if new major version wasn't released. Therefore, this thesis introduces the share ratio trend that reach to 1 finally. Substituting Eqs. (6.3) and (6.6) into Eq. (6.6), the corrected execution time based on exponential function t_{mi} is as follows:

$$t_{mi} = \sum_{k=1}^{i} N_k(1 - \exp\{-b(t+t_{off})\})\Delta T_k. \tag{6.7}$$

Substituting Eqs. (6.5) and (6.6) into Eq. (6.6), the corrected execution time based on exponential function t_{mi} is as follows:

$$t_{mi} = \sum_{k=1}^{i} N_k(b^{\exp\{-c(t+t_{off})\}})\Delta T_k. \tag{6.8}$$

6.5 Numerical examples

The proposed method is applied to the version 2 fault data of P-DS1 and compared with the previous method. Furthermore, data from April 27th, 2023 to February 29th, 2024 are used. 90& of the dataset is employed to estimate the parameters of the mean value function using the maximum likelihood estimation method. The exponential model, delayed S-shaped model, inflection S-shaped model, and logarithmic Poisson execution time model are applied as the mean value function [8–11].

Figure 6.3 and Figure 6.4 show the estimation results when using the exponential system correction model and the Gompertz correction model, respectively. Table 6.1 shows the results of model evaluation using MAE, MAPE, and AIC. The results in Table 6.1 show that the function with the highest accuracy varies depending on the mean value function. For this reason, when actually applying the method, it is necessary to use both models and select the one with the highest accuracy.

The evaluation indices of the estimated results are compared with those in Table 5.8. When compared with Table 5.8a, it can be seen that the proposed method is able to estimate with high accuracy, except for the delayed S-shaped model. Furthermore, when compared with Table 5.8b, the results obtained are found to be close to the accuracy of the model in Table 5.8b.

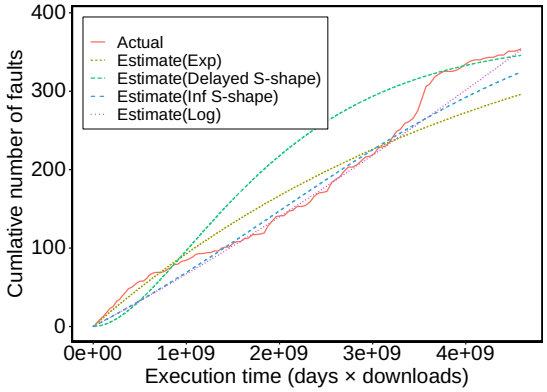


Figure 6.3: The estimated result using the exponential function.

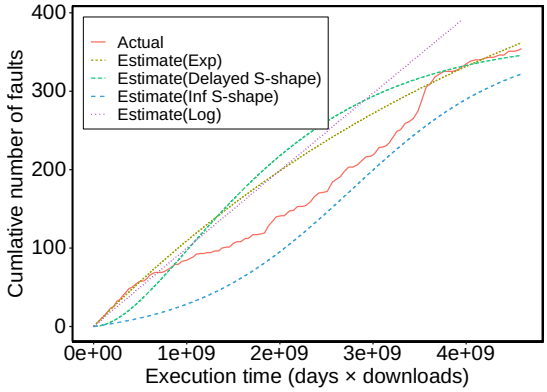


Figure 6.4: The estimated result using the Gompertz function.

Table 6.1: The evaluation result of the models.

	Exp		Delayed		Inf		Log	
	Exp	Gomp	Exp	Gomp	Exp	Gomp	Exp	Gomp
MAPE	0.15	0.16	0.67	0.67	0.24	1.42	0.22	0.22
MAE	23.74	29.35	39.53	39.54	16.93	38.85	13.28	47.20
AIC	573.36	555.14	677.38	679.38	530.66	582.64	509.58	526.16

6.6 Summary

This thesis proposed the SRGM considering the growth of the major version share ratio. As a result, a model has been structured that can be applied even when the number of downloads by version cannot be retrieved. chapter 5's model requires the number of downloads by version. Therefore, the number of applicable OSS is limited. By applying the proposed method, the model proposed in chapter 5 can be used with more OSS. In addition, the proposed model showed an estimation accuracy similar to that when the number of downloads by version is used. The proposed method is expected to be useful for the reliability assessment of more OSS. Furthermore, by improving the share ratio function in the future, more accurate estimation can be expected.

On the other hand, since this model does not assume that the share ratio of the major version will decrease, if the quality of the released major version is low, the share ratio may not increase and the estimation accuracy may decrease. In addition, the share ratio of OSS with the small number of downloads may fluctuate drastically. It may reduce the accuracy of the proposed method.

In the future, data on the trends of the share ratio will be collected from additional OSS projects to study the optimal share ratio function.

Papers Related in this Chapter

1. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Software Reliability Growth Model Considering Package Manager Behavior," Proceedings of the 29th ISSAT International Conference on Reliability and Quality in Design, August 8–10, 2024, pp. 244–248 (Virtual mode).

2. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "A Study on the Relationship Between OSS Adoption and Download Numbers Based on Repository Mining," IECIE Technical Report[Reliability], Shogai Gakushu Center Kiran (Muroran, Hokkaido), Vol. 124, No. 135, pp. 7–12, July 27, 2024 (in Japanese).

3. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada,"Prediction of the OSS Number of Faults Based on a Reliability Growth Model Considering User Usage Patterns," IECIE Technical Re-

port[Reliability], Matsue Terrsa, Vol. 123, No. 399, pp. 1–5, February 29, 2024 (in Japanese).

Chapter 7

Conclusion

This thesis focuses on the three issues listed in Section 1.1 for the developing SRGM for OSS. The following points are discussed in this thesis:

1. Variability in the number of users: Methods to build the SRGM considering the number of users.

2. Differences in fault reporting process in BTS: Methods to utilize the fault data quickly in OSS BTS.

3. Software updates: Methods to build the SRGM considering the software update.

In "Variability in the number of users," this thesis analyzed the publicly available data on the number of OSS downloads and the number of faults. Moreover, this thesis proposed the new method to calculate the execution time considering the number of users for standardizing the test time for OSS. Furthermore, this thesis evaluated the compatibility of the proposed method with the existing SRGM. The results showed that the proposed method showed the trends similar to the results observed in the traditional proprietary software SRGM. It indicates that the proposed method could potentially be applied for the reliable software evaluations.

It has been previously known that the traditional SRGM can be applied to OSS, it tends to show the linear increase in the faults. However, even with highly reliable OSS, SRGM tends to evaluate

its reliability low. There is a discrepancy between the evaluation results and the reality. By using the proposed method to standardize the test time based on the number of users, it becomes possible to evaluate similarly to the traditional SRGM. It enables the developers to apply the same reliability assessments for OSS like the proprietary software. Additionally, it can be applied directly to any SRGM that developers wish to use since the proposed method does not interfere with parameters other than time in existing SRGM. It allows for the flexibility in performing reliability evaluations, considering various scenarios as necessary.

The proposed method showed the higher degree of model fit compared to previous methods from the perspective of model evaluation metrics. In the software reliability assessments, models with a high degree of fit are essential for accurate evaluations. Traditional SRGM have shown tendencies to overfit certain aspects of the model, reducing overall accuracy. By adopting the proposed method, it becomes possible to conduct reliability assessments with higher precision.

In "Differences in fault reporting process in BTS," this thesis proposed the method to accelerate the acquisition of the fault data necessary for the SRM in OSS. Since OSS is often developed by the volunteer developers, it typically takes longer to confirm the software faults compared to the proprietary software. Additionally, the popular software with a large user base may experience a high volume of the fault reports. It can delay the process of confirming the fault report. As a result, it took time to use the fault data from the BTS in the reliability evaluations.

This thesis focused on the trend of SRGM curves that reflect both the unconfirmed and the confirmed fault data. This thesis proposed the method for SRGM using the unconfirmed fault data. As the result, this thesis proposed the SRGM that enables highly reliable evaluations. It close to the evaluations based on the confirmed faults even when using the unconfirmed failure data. In highly active OSS projects, the source code is frequently updated. The reliability assessments using old fault data may estimate inaccurate results. By applying the proposed method, the fault data can be used for real-time reliability evaluations at the point when faults are reported. OSS project developers can quickly assess the reliability of OSS and accurately estimate the necessary testing period before releasing the software. Additionally, OSS users can perform real-time reliability

assessments for OSS. It can be possible to help to select high-reliability OSS for their purposes.

In "Software updates," this thesis proposed the SRGM considering software updates in OSS. Unlike the proprietary software, OSS undergoes updates not only for debugging but also for enhancing the functionality. It leads to the variations in the number of latent faults in the source code. It can affect the accuracy of SRGM. This thesis analyzed the OSS updates from the perspectives of major updates, minor updates, and patch updates. Moreover, this thesis proposed the models considering the software major update. As a result, it was found that OSS users tend to be more proactive in updating the minor and patch versions. It is thought to be due in part to the fact that the effect of changes associated with updates is small. On the other hand, major updates were less frequently executed. Generally, the software updates are recommended from the security and the performance perspectives. However, concerns regarding compatibility with the existing systems and the risk of introducing unknown faults often lead to a delay in implementing updates, necessitating further investigation. However, the effect on software reliability evaluation was not understood. This thesis shows that the effect of the software updates can be reduced by dividing the fault data in the major updates. Furthermore, this thesis proposed the method for SRGM for each major version based on the analysis. It achieved more accurate estimates compared to the models that do not differentiate between the versions. By applying reliability evaluations for each major version, this method may help to the better version selection strategies.

In conclusion, the proposed methods successfully addressed three challenges in SRGM for OSS, enabling the construction of an SRGM that provides reliable evaluations. These methods hold potential as tools for selecting high-reliability OSS and improving software reliability when utilizing OSS. Additionally, OSS projects that adopt the proposed methods may contribute to the development of more reliable OSS.

References

[1] Synopsys, “2022 Open source security and risk analysis report,” https://www.synopsys.com/, 2022.

[2] U.S. Department of Defense, “Software Development and Open Source Software,” 2022

[3] ISO/IEC, “ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models,” Organization for Standardization, 2011.

[4] Y. Tamura and S. Yamada, “A component-oriented reliability assessment method for open source software,” *International Journal of Reliability, Quality and Safety Engineering*, Vol. 15, No. 1, pp. 33–53, 2008.

[5] M.R. Lyu, “*Handbook of software reliability engineering*,” IEEE computer society press Los Alamitos, Vol. 222, 1996.

[6] Y. Tamura, S. Miyamoto, L. Zhou, A. Anand, P.K. Kapur, and S. Yamada, “OSS reliability assessment method based on deep learning and independent Wiener data preprocessing,” *Journal International Journal of System Assurance Engineering and Management*, Springer, pp. 1–9, 2024.

[7] H. Pham, “*System software reliability*,” Springer Science & Business Media, 2007.

[8] A.L. Goel and K. Okumoto, “Time-dependent error-detection rate model for software reliability and other performance measures,” *IEEE transactions on Reliability*, Vol. 28, No. 3, pp. 206–211, 1979.

[9] S. Yamada, M. Ohba, and S. Osaki, “S-shaped software reliability growth models and their applications,” *IEEE Transactions on Reliability*, Vol. 33, No. 4, pp. 289–292, 1984.

[10] M. Ohba, “Inflection S-shaped software reliability growth model,” *Stochastic Models in Reliability Theory: Proceedings of a Symposium Held in Nagoya, Japan, April 23–24, 1984*, pp. 144–162, Springer, 1984.

[11] J. D. Musa, A. Iannino, and K. Okumoto, “*Software reliability: measurement, prediction, application*,” McGraw-Hill, Inc., 1987.

[12] B.W. Boehm, “Managing Software Productivity and Reuse,” *Computer*, Vol. 32, pp. 111–113, 1999.

[13] A.B. Lliteras, D. Torres, C.A. Collazos, and A. Fernandez, “Development, reuse, and repurposing of software artifacts in Digital Citizen Science. Are we reinventing the wheel,” The VI Iberoamerican Conference of Computer Human Interaction, pp. 16–18, 2020.

[14] S.A. Ajila and D. Wu, “Empirical study of the effects of open source adoption on software development economics,” *Journal of Systems and Software*, Vol. 80, No. 9, pp. 1517–1529, 2007.

[15] Open Source Initiative, “The Open Source Definition,” https://opensource.org/docs/osd, 2006.

[16] Open Source Initiative, “The MIT License,” https://opensource.org/license/MIT, 2007.

[17] Free Software Foundation, “GNU General Public License,” https://www.gnu.org/licenses/gpl-3.0.html, 2007.

[18] Japan Ministry of Economy, Trade and Industry, “Case Studies on the Utilization of OSS and Management Methods for Ensuring Its Security,” 2022.

[19] The Linux Foundation, “Open Source Program Offices in 2023: Key Insights and Gap Exploration, ” https://www.linuxfoundation.org/, 2023.

[20] Heartbleed, “Heartbleed Bug,” https://heartbleed.com/, 2014.

[21] The OpenSSL Project, “OpenSSL: The Open Source toolkit for SSL/TLS,” https://openssl.org/.

[22] M. Williams, M. Tüxen, and R. Seggelmann, “Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension,” No. 6520, RFC Editor, 2012.

[23] M. Jain, T. Manjula, and T.R. Gulati, “Software reliability growth model (SRGM) with imperfect debugging, fault reduction factor and multiple change-point,” Proceedings of the International Conference on Soft Computing for Problem Solving, pp. 1027–1037, Springer India, 2012.

[24] P. Kapur, S. Bhushan, and S. Younes, “An exponential srgm with a bound on the number of failures,” *Microelectronics Reliability*, Vol. 33, No. 9, pp. 1245–1249, 1993.

[25] Q. Li and H. Pham, “NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage,” *Applied Mathematical Modelling*, Vol. 51, pp. 68–85, 2017.

[26] R. Micko, S. Chren, and B. Rossi, “Applicability of software reliability growth models to open source software,” Proceedings of 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), (Los Alamitos, CA, USA), pp. 255–262, IEEE Computer Society, 2022.

[27] J.A. Nelder and R. Mead, “A simplex method for function minimization,” *The computer journal*, Vol. 7, No. 4, pp. 308–313, 1965.

[28] H. Sone, Y. Tamura and S. Yamada, “Optimal maintenance problem with OSS-oriented EVM for OSS project,” *Reliability and Maintenance Modeling with Optimization*, CRC Press Taylor & Francis Group, pp. 197–213, 2023.

[29] H. Sone, Y. Tamura, and S. Yamada, “A study of quantitative progress evaluation models for open source projects,” *Journal of Software Engineering and Applications*, Vol. 15, No. 5, pp. 183–196, 2022.

[30] T.M. Khoshgoftaar and T.G. Woodcock, “Software reliability model selection,” *Quality and Reliability Engineering International*, Vol. 8, pp. 457–469, 1992.

[31] H. Akaike, “Factor analysis and AIC,” *Psychometrika*, Vol. 52, pp. 317–332, 1987.

[32] Linux Kernel Organization, “The Linux Kernel Archives,” https://kernel.org/.

[33] The CVS Team, “CVS - Concurrent Versions System,” https://www.nongnu.org/cvs/.

[34] Git, “Git, ” https://git-scm.com/.

[35] CollabNet and Apache, “Apache Subversion,” https://subversion.apache.org/.

[36] W.F. Tichy, “An Interview with Prof. Andreas Zeller: Mining your way to software reliability,” *Ubiquity*, Vol. 2010, p. 3, 2010.

[37] L. Wu, B. Xie, G.E. Kaiser, and R. Passonneau, “BUGMINER: Software Reliability Analysis Via Data Mining of Bug Reports,” Proceedings of International Conference on Software Engineering and Knowledge Engineering, pp. 95–100, 2011.

[38] J. Liang and O. Mizuno, “Analyzing Involvements of Reviewers through Mining a Code Review Repository,” Proceedings of 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement, pp. 126–132, 2011.

[39] R. Paramitha and Y.D.W. Asnar, “Mining Software Repository for Security Smell Code Review,” Proceedings of 2021 International Conference on Data and Software Engineering, pp. 1–6, 2021.

[40] M.A.d.F. Farias, R.L. Novais, M.C.R. Junior, L.P.d.S. Carvalho, M.G. Mendonça, and R.O. Spínola, “A systematic mapping study on mining software repositories,” Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016.

[41] Bugzilla, “The software solution designed to drive software development,” https://www.bugzilla.org/.

[42] Redmine, “Redmine: Overview,” https://www.redmine.org/.

[43] GitHub, “GitHub,” https://github.com/.

[44] GitLab Inc, “GitLab,” https://about.gitlab.com/.

[45] Y. Tamura and S. Yamada, “Reliability analysis based on deep learning for fault big data on bug tracking system,” Proceedings of 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 31–36, 2016.

[46] V.B. Singh and K.K. Chaturvedi, “Bug Tracking and Reliability Assessment System (BTRAS),” *International Journal of Software Engineering and Its Applications*, Vol. 5 No. 4, pp.17–30, 2011.

[47] T.F. Bissyande, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Reveillere, “Empirical Evaluation of Bug Linking,” Proceedings of 2013 17th European Conference on Software Maintenance and Reengineering, pp. 89–98, 2013.

[48] M. Kumari, A. Misra, S. Misra, L.F. Sanz, R. Damasevicius, and V.B. Singh, “Quantitative Quality Evaluation of Software Products by Considering Summary and Comments Entropy of a Reported Bug,” *Entropy*, Vol. 21, 2019.

[49] Y. Lee, S. Lee, C.G. Lee, I. Yeom, and H. Woo, "Continual Prediction of Bug-Fix Time Using Deep Learning-Based Activity Stream Embedding," *IEEE Access*, Vol. 8, pp. 10503–10515, 2020.

[50] P. Ardimento and C. Mele, "Using BERT to Predict Bug-Fixing Time," Proceedings of 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS), pp. 1–7, 2020.

[51] Y. Noyori, H. Washizaki, Y. Fukazawa, K. Ooshima, H. Kanuka, S. Nojiri, and R. Tsuchiya, "Extracting features related to bug fixing time of bug reports by deep learning and gradient-based visualization," Proceedings of 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), pp. 402–407, 2021.

[52] W.Y. Ramay, Q. Umer, X.C. Yin, C. Zhu, and I. Illahi, "Deep Neural Network-Based Severity Prediction of Bug Reports," *IEEE Access*, Vol. 7, pp. 46846–46857, 2019.

[53] R. Malhotra, A. Dabas, H.A. S, and M. Pant, "A Study on Machine Learning Applied to Software Bug Priority Prediction," Proceedings of 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 965–970, 2021.

[54] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging," Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, pp. 171–179, 2019.

[55] W.W. Royce, "Managing the development of large software systems: Concepts and techniques," Proceedings of IEEE WESCON, pp. 1–9, 1970.

[56] P.C. Pendharkar and J.A. Rodger, "The relationship between software development team size and software development cost," *Commun. ACM*, Vol. 52, No. 1, pp. 141–144, 2009.

[57] N. Li, Q. Han, Y. Zhang, C. Li, Y. He, H. Liu, and Z. Mao, “Standardization Workflow Technology of Software Testing Processes and its Application to SRGM on RSA Timing Attack Tasks,” *IEEE Access*, Vol. 10, pp. 82540–82559, 2022.

[58] S. Chatterjee and A. Shukla, “Effect of Test Coverage and Change Point on Software Reliability Growth Based on Time Variable Fault Detection Probability,” *Journal of Software*, Vol. 11, pp. 110–117, 2016.

[59] npm, “npm,” https://www.npmjs.com/.

[60] Nx, “Smart, fast extensible build system,” https://nx.dev/.

[61] SWC, “Rust-based platform for the web,” https://swc.rs/.

[62] webpack, “webpack,” https://webpack.js.org/.

[63] Vite, “Vite — next generation frontend tooling,” https://vitejs.dev/.

[64] H. Pan, A. Sheng, Z. Wang, and X. Han, “Analysis of MTBF evaluation methods for small sample sizes,” Proceedings of 2016 11th International Conference on Reliability, Maintainability and Safety, pp. 1–7, 2016.

[65] P.A. Kullstam, “Availability, MTBF and MTTR for Repairable M out of N System,” *IEEE Transactions on Reliability*, Vol. R-30, pp. 393–394, 1981.

[66] M. Kimura, S. Yamada, and S. Osaki, “Statistical software reliability prediction and its applicability based on mean time between failures,” *Mathematical and Computer Modelling*, Vol. 22, pp. 149–155, 1995.

[67] T. Warns, C. Storm, and W. Hasselbring, “Availability of Globally Distributed Nodes: An Empirical Evaluation,” Proceedings of the 2008 Symposium on Reliable Distributed Systems, pp. 279–284, 2008.

[68] Python Software Foundation, “PyPI,” https://pypi.org/.

[69] urllib3, “urllib3,” https://urllib3.readthedocs.io/en/stable/.

[70] Python Software Foundation, “Requests: HTTP for Humans,” https://requests.readthedocs.io/en/latest/.

[71] aio-libs, “Welcome to AIOHTTP,” https://docs.aiohttp.org/en/stable/.

[72] Hyper, “h11: A pure-Python HTTP/1.1 protocol library,” https://h11.readthedocs.io/en/latest/.

[73] Google, “BigQuery,” https://cloud.google.com/bigquery.

[74] Linehaul, “Linehaul-cloud-function,” https://github.com/pypi/linehaul-cloud-function.

[75] A. Tandon, A.G. Aggarwal, and N. Nijhawan, “An NHPP SRGM with Change Point and Multiple Releases,” *International Journal of Information Systems in the Service Sector*, Vol. 8, pp. 57–68, 2016.

[76] N. Nijhawan and A.G. Aggarwal, “On development of change point based generalized SRGM for software with multiple releases,” Proceedings of 2015 4th International Conference on Reliability, pp. 1–6, 2015.

[77] A.G. Aggarwal, V. Dhaka, N. Nijhawan, and A. Tandon, “*Reliability Growth Analysis for Multi-release Open Source Software Systems with Change Point*,” System Performance and Management Analytics, pp. 125–137, 2018.

[78] J. Zhao, H.W. Liu, G. Cui, and X.Z. Yang, “Software reliability growth model with change-point and environmental function,” *Journal of Systems and Software*, Vol. 79, No. 11, pp. 1578–1587, 2006.

[79] N. Zhang, G. Cui, and H. Liu, “A stochastic software reliability growth model with learning and change-point,” Proceedings of 2012 World Automation Congress, pp. 399–403, 2012.

[80] Z. Mengmeng and P. Hoang, “A multi-release software reliability modeling for open source software incorporating dependent fault detection process,” *Annals of Operations Research*, Vol. 269, No. 1–2, pp. 773–790, 2017.

[81] S. Inoue and S. Yamada, “Change-point modeling for software reliability assessment depending on two-types of reliability growth factors,” Proceedings of 2010 IEEE International Conference on Industrial Engineering and Engineering Management, pp. 616–620, 2010.

[82] T.P. Werner, “Semantic Versioning 1.0.0-beta,” https://semver.org/spec/v1.0.0-beta.html, 2009.

[83] Python Enhancement Proposals, “PEP 508 - Dependency specification for Python Software Packages,” https://peps.python.org/pep-0508/, 2015.

[84] Microsoft, “everyonecancode,” https://github.com/microsoft/everyonecancode.

[85] Amazon Web Services, “boto3,” https://github.com/boto/boto3/, 2024.

[86] Amazon Web Services, “botocore,” https://github.com/boto/botocore/, 2024.

[87] T. Sudtasan and H. Mitomo, “Comparison of Diffusion Models for Forecasting the Growth of Broadband Markets in Thailand,” 14th ITS Asia-Pacific Regional Conference, Kyoto 2017: Mapping ICT into Transformation for the Next Information Society, International Telecommunications Society (ITS), 2017.

[88] Z. Geng and C. Yu, “An integral explanation to the S-shape growth figure in network effect product market,” Proceedings of 2008 Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing, pp. 154–158, 2008.

[89] C. Zhan and C.K.M. Tse, “A universal model for growth of user population of products and services,” *Network Science*, Vol. 4, pp. 491–507, 2016.

[90] C. Zhan and C.K.M. Tse, "A Model for Growth of Markets of Products or Services Having Hierarchical Dependence," *IEEE Transactions on Network Science and Engineering*, Vol. 6, pp. 198–209, 2019.

[91] C. Zhan, X. Zhong, Q. Zhang, M. Zhao, and Y. Wang, "Modelling for Dynamic Growth of User Population of Products and Services," Proceedings of 2019 IEEE 17th International Conference on Industrial Informatics, Vol. 1, pp. 1478–1482, 2019.

[92] T. Jin, "Research on gompertz curve model used for mobile user growth," Proceedings of 2010 International Conference on Educational and Network Technology, pp. 557–560, 2010.

[93] B. Gompertz, "XXIV. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. In a letter to Francis Baily, Esq. FRS &c," *Philosophical transactions of the Royal Society of London*, No. 115, pp. 513–583, 1825.

Acknowledgment

The author would like to express his gratitude to Professor Yoshinobu Tamura, the supervisor of the author's study and the chairman of this dissertation reviewing committee, for his introduction to research on software reliability, valuable advice, continuous support, and warm guidance.

The author is indebted to Professor Hidekazu Murata, Professor Kei Kawamura, Associate Professor Masashi Hotta, and Associate Professor Hiroaki Yamada the members of the dissertation reviewing committee, for reading the manuscript and making helpful comments.

The author wishes to particularly thank Emeritus Professor/Research Professor Shigeru Yamada of Tottori University for invariable encouragement and support.

The author has also received priceless cooperation and suggestions from many people for the achievement of this work.

The author would also like to acknowledge the kind hospitality and encouragement of the past and present members of Professor Tamura's laboratory.

Finally, the author deeply grateful to all those who have influenced my life.

Publication List of the Author

Books

1. Y. Tamura, S. Miyamoto, L. Zhou, and S. Yamada, "Comparison of OSS reliability assessment methods by using Wiener data preprocessing based on deep learning," Reliability Engineering for Industrial Processes, Springer Series in Reliability Engineering, Springer, pp. 1–17, April 23, 2023.

2. H. Sone, S. Miyamoto, Y Kashihara, Y. Tamura and S. Yamada, "Deep learning approach based on fault correction time for reliability assessment of cloud and edge open source software," Predictive Analytics in System Reliability, Springer, pp. 1–18, September 9, 2022.

Journal Papers

1. H. Takeda, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Sensitivity analysis for deep learning data sets considering the human immune system," Journal of Graphic Era University, Vol. 12 Issue 2, pp. 309–328, November 8, 2024.

2. S. Miyamoto, Y. Tamura and S. Yamada, "A method of reliability assessment based on trend analysis for open source software," International Journal of Reliability, Quality and Safety Engineering, Vol. 31, No. 4, World Scientific, pp. 2450004-1–2450004-15, April 30, 2024.

3. L. Zhou, S. Miyamoto, Y. Tamura, and H. Yamamoto, "Reliability evaluation of multi-state linear Consecutive-k-out-of-n:G systems and its application to maintenance problems," Inter-

national Journal of Reliability, Quality and Safety Engineering, Vol. 31, No. 4, World Scientific, pp. 2450006-1–2450006-17, May 4, 2024.

4. Y. Tamura, S. Miyamoto, L. Zhou, A. Anand, P.K. Kapur, and S. Yamada, "OSS reliability assessment method based on deep learning and independent Wiener data preprocessing," Journal International Journal of System Assurance Engineering and Management, Springer, pp. 1–9, February 2024.

5. Y. Tamura, S. Miyamoto, L. Zhou, and S. Yamada, "OSS sustainability assessment based on the deep learning considering effort Wiener process data," International Journal of Reliability, Quality and Safety Engineering, Vol. 31, No. 1, World Scientific, pp. 2350032-1–2350032-15, November 4, 2023.

6. X. Dong, Y. Liang, S. Miyamoto, and S. Yamaguchi, "Ensemble learning based software defect prediction," Journal of Engineering Research, Vol. 11, No. 4, pp. 377–391, November 3, 2023.

7. S. Miyamoto, Y. Tamura, and S. Yamada, "A method of OSS reliability assessment based on public repository analysis," International Journal of Reliability, Quality and Safety Engineering, Vol. 30, No. 5, World Scientific, pp. 2350021-1–2350021-12, August 9, 2023.

8. S. Miyamoto, Y. Tamura, and S. Yamada, "Reliability assessment tool based on deep learning and data preprocessing for OSS," American Journal of Operations Research, Vol. 12, No. 3, pp. 111–125, May, 2022.

International Conference Papers (Peer-reviewed)

1. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Estimation method based on SRGM using uncertain fault information on open BTS," Recent Advances in Reliability and Maintenance Modeling, CRC Press, pp. 140–147, November 15, 2024.

2. Y. Tamura, H. Takeda, S. Heima, S. Miyamoto, L. Zhou, A. Anand, P.K. Kapur, and S. Yamada, “Performability assessment measure based on deep learning for open source software,” Recent Advances in Reliability and Maintenance Modeling, CRC Press, pp. 217–224, November 15, 2024.

3. Y. Tamura, S. Miyamoto, L. Zhou, and S. Yamada, “OSS reliability assessment method based on deep learning and two dimensional noisy processes,” Proceedings of the 29th ISSAT International Conference on Reliability and Quality in Design, August 8–10, pp. 219–223, 2024 (Virtual mode).

4. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “Software Reliability Growth Model Considering Package Manager Behavior,” Proceedings of the 29th ISSAT International Conference on Reliability and Quality in Design, August 8–10, pp. 244–248, 2024 (Virtual mode).

5. L. Zhou, S. Miyamoto, Y. Tamura, and H. Yamamoto, “A note on the improvement of the reliability evaluation of linear consecutive-k-out-of-n: G systems under the inspection,” Proceedings of the 29th ISSAT International Conference on Reliability and Quality in Design, August 8–10, pp. 259–263, 2024 (Virtual mode).

6. Y. Tamura, S. Miyamoto, L. Zhou, A. Anand, P.K. Kapur, and S. Yamada, “OSS fault removal system based on deep learning inspired by immune system,” Proceedings of the IEEE/ACIS 26th International Conference on Computer and Information Science (ICIS 2024-Summer III), Kitakyushu, Japan, July 16–18, 2024, to be published.

7. L. Zhou, S. Miyamoto, Y. Tamura, and H. Yamamoto, “A note on reliability computation for linear consecutive-k-out-of-n : G systems using domination,” Proceedings of the 7th Asian Conference of Management Science and Applications, Okinawa, Japan, December 15–17, pp. 1–6, 2023.

8. S. Miyamoto, Y. Tamura, and S. Yamada, “A method of reliability assessment based on stochastic model for open source software,” Proceedings of the 28th ISSAT International

Conference on Reliability and Quality in Design, August 3–5, pp. 274–278, 2023 (Virtual mode).

9. S. Miyamoto, Y. Tamura, and S. Yamada, “A method of software reliability assessment based on natural language processing for OSS,” Proceedings of the 1st International Conference on Mathematical, Engineering and Management Sciences, June 25–26, pp. 410–420, 2022 (Virtual mode).

Reports

1. S. Miyamoto, Y. Tamura, and S. Yamada, “A Reliability Assessment Method for Public Software Repository Based on Deep Learning, ”RIMS kokyuroku “Mathematical Decision Making Under Uncertainty and Related Topics,” No. 2242, pp. 69–80, January, 2023 (in Japanese).

Presentations

1. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “A Study on the Relationship Between OSS Adoption and Download Numbers Based on Repository Mining,” IECIE Technical Report[Reliability], Shogai Gakushu Center Kiran (Muroran, Hokkaido), Vol. 124, No. 135, pp. 7–12, July 27, 2024 (in Japanese).

2. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “Prediction of the OSS Number of Faults Based on a Reliability Growth Model Considering User Usage Patterns,” IECIE Technical Report[Reliability], Matsue Terrsa, Vol. 123, No. 399, pp. 1–5, February 29, 2024 (in Japanese).

3. Y. Tamura, S. Miyamoto, L. Zhou, and Y. Tamura, “A Comparative Study on the Suitability of OSS Reliability Evaluation Methods Based on Deep Learning Considering Wiener Processes and Jump Diffusion Processes,” IECIE Technical Report[Reliability], Matsue Terrsa, Vol. 123, No. 399, pp. 46–51, February 29, 2024 (in Japanese).

4. T. Kagiyama, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, “Reliability Evaluation of Cloud OSS Based on Deep Learning with RNN and LSTM,” The 74th Joint Convention

Record of Chugoku Section of Institutes of Electrical and Information Engineers, Okayama University, R23–24–01, pp. 1–2, October 28, 2023 (in Japanese).

5. S. Heima, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Development of 3D Software for Reliability Evaluation of OSS and Its Applications," The 74th Joint Convention Record of Chugoku Section of Institutes of Electrical and Information Engineers, Okayama University, R23–25–18, pp. 1–2, October 28, 2023 (in Japanese).

6. R. Masugi, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Reliability Evaluation Based on Transfer Learning Algorithms Using Multiple OSS," The 74th Joint Convention Record of Chugoku Section of Institutes of Electrical and Information Engineers, Okayama University, R23–25–19, pp. 1–2, October 28, 2023 (in Japanese).

7. K. Mihara, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Data Preprocessing and Reliability Evaluation Method Based on Deep Learning Considering Wiener Processes for OSS, and its Suitability Assessment," The 74th Joint Convention Record of Chugoku Section of Institutes of Electrical and Information Engineers, Okayama University, R23–25–20, pp. 1–2, October 28, 2023 (in Japanese).

8. L. Zhou, S. Miyamoto, Y. Tamura, and H. Yamamoto, "A reliability model for street lighting systems subject to random shock under dynamic environments," Proceedings of the 23rd Asia Pacific Industrial Engineering & Management Systems Conference (APIEMS 2023), Kuala Lumpur, Malaysia, October 22–26, 2023.

9. S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "A Study on the Impact of Software Updates on OSS Reliability Growth Curves," Proceedings of the 2023 Autumn Research Presentation Conference of the Project Management Society, Yamaguchi University Tokiwa Campus, pp. 1–6, August 31–September 1, 2023 (in Japanese).

10. K. Fujuta, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "Sensitivity Analysis and Applications of Edge Server Optimization Based on a Composite Stochastic Process Model,"

Proceedings of the 2023 Autumn Research Presentation Conference of the Project Management Society, Yamaguchi University Tokiwa Campus, pp. 137–144, August 31–September 1, 2023 (in Japanese).

11. Y. Gotan, S. Miyamoto, L. Zhou, Y. Tamura, and S. Yamada, "A Development Effort Prediction Model Based on Jump Diffusion Stochastic Processes Considering Edge Environments," Proceedings of the 2023 Autumn Research Presentation Conference of the Project Management Society, Yamaguchi University Tokiwa Campus, pp. 345–352, August 31–September 1, 2023 (in Japanese).

12. S. Miyamoto, Y. Tamura, and S. Yamada, "A Reliability Evaluation Tool for Open Source Software Based on Deep Learning and MTBF," Proceedings of the 23rd IEEE Hiroshima Section Student Symposium, Yamaguchi University Tokiwa Campus, Online, pp. 191–194, November 27–28, 2021 (in Japanese).

13. S. Miyamoto, Y. Tamura, and S. Yamada, "A method of cloud OSS reliability assessment considering the fault level based on deep learning," Proceedings of the 2nd Joint Seminar–University Tun Hussein Onn Malaysia and Yamaguchi University, pp. 28–29, November 19, 2021 (Virtual mode).

14. S. Miyamoto, Y. Tamura, and S. Yamada, "A Study on MTBF Estimation for OSS Reliability Evaluation Using Deep Learning," Abstracts of the 2021 Autumun National Conference of the ORSJ, Kyushu University, No. 2–D–3, September 16–17, 2021 (Virtual mode, in Japanese).

Software Tools

1. S. Miyamoto, Y. Tamura and S. Yamada, RAT Based on DL and DP for OSS (Reliability Assessment Tool Based on Deep Learning and Data Preprocessing for OSS), http://www.tam.eee.yamaguchi-u.ac.jp/

2. Y. Tamura, S. Miyamoto, and S. Yamada, ET of FMT Based on OSS Fault Severity Levels

for EC (Estimation Tool of Fault Modification Time Based on OSS Fault Severity Levels for Edge Computing), http://www.tam.eee.yamaguchi-u.ac.jp/

Awards

1. The Operations Research Society of Japan Chugoku-Shikoku Section Chief Award, The Operations Research Society of Japan, Japan, March 31, 2023 (in Japanese).

2. The Operations Research Society of Japan Chugoku-Shikoku Section Chief Award, The Operations Research Society of Japan, Japan, March 31, 2022 (in Japanese).