Doctoral Dissertation

Stability Assessment Methodology for
Open Source Projects Considering Uncertainty
（不確実性を考慮したオープンソースプロジェクトに対する
安定性評価手法）

September, 2023

Hironobu Sone
Graduate School of Sciences and Technology for Innovation,
Yamaguchi University

Abstract

Open source software (OSS) are adopted as embedded systems, server usage, and so on because of quick delivery, cost reduction, and standardization of systems. Therefore, OSS is often used not only for the personal use but also for the commercial use. Many OSS have been developed under the peculiar development style known as the bazaar method. According to this method, many faults are detected and fixed by developers around the world, and the fixed result will be reflected in the next release. Also, many OSS are developed and managed by using the fault big data recorded on the bug tracking systems. Then, many OSS are developed and maintained by several developers with many OSS users.

According to the results of the 2022 Open Source Security and Risk Analysis (OSSRA), OSS is an essential part of proprietary software, e.g., the source code containing OSS is 97%, all source code using OSS is 78%.

On the other hand, OSS has issues from various perspectives. Therefore, OSS users need to decide on whether they should use OSS with consideration of each issue. In addition, the managers of open source projects need to manage their projects appropriately because OSS has a large impact on software around the world.

This thesis focuses on the following three issues among many ones. We examine a method for OSS users and open source project managers to evaluate the stability of open source projects.

1. Selection evaluation and licensing: Methods for OSS users to make selections from the many OSS available situation,

2. Vulnerability support: Predicted fault fix priority for the reported OSS,

3. Maintenance and quality assurance: Prediction of appropriate OSS version upgrade timing, considering the development effort required after OSS upgrade by OSS users.

In “1. Selection evaluation and licensing,” we attempt to derive the OSS-oriented EVM by applying the earned value management (EVM) to several open source projects. The EVM is one

of the project management methodologies for measuring the project performance and progress. In order to derive the OSS-oriented EVM, we apply the stochastic models based on software reliability growth model (SRGM) considering the uncertainty for the development environment in open source projects. We also improve the method of deriving effort in open source projects. In case of applying the existing method of deriving effort in open source projects, it is not possible to derive some indices in the OSS-oriented EVM. Thus, we resolve this issue. The derived OSS-oriented EVM helps OSS users and open source project managers to evaluate the stability of their current projects. It is an important to use the decision-making tool regarding their decisions and projects of OSS. From a different perspective, we also evaluate the stability of the project in terms of the speed of fault fixing by predicting the time transition of fixing the OSS faults reported in the future.

2. In "Vulnerability support", in terms of open source project managers, we create metrics to detect faults with a high fix priority and predicted a long time for fixing. In addition, we try to improve the detection accuracy of the proposed metrics by learning not only the specific version but also the bug report data of the past version by using the random forest considering the characteristic similarities of bugs fix among different versions. This allows the project managers to identify the faults that should be prioritized for fixing when a large number of faults are reported, and facilitates project operations.

In "3. Maintenance and quality assurance", as an optimum maintenance problem, we predict the appropriate OSS version-up timing considering the maintenance effort required by OSS users after upgrading the OSS. It is dangerous in terms of the vulnerability to continue using the specified version of OSS ignoring the End of Life. Therefore, we should upgrade the version periodically. However, the maintenance cost increase with the version upgrade frequently. Then, we find the optimum maintenance time by minimizing the total expected software maintenance effort in terms of OSS users. In particular, we attempt to reflect the progress of open source projects by using the OSS-oriented EVM in deriving the optimal maintenance time.

In conclusion, we found that there is the applicability as the stability evaluation of open source projects from three perspectives. Particularly, the OSS-oriented EVM discussed in "1. Selection

evaluation and licensing" can contribute to the visualization of maintenance effort in open source projects. The proposed method will potentially contribute to the development of OSS in the future.

Acknowledgments

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 Yuji Wakasa, Professor Toshihiko Tanaka, Professor Kei Kawamura, and Associate Professor Masashi Hotta, the members of the dissertation reviewing committee, for reading the manuscript and making helpful comments.

The author wishes to particularly thank Emeritus 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.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The source code of open source software (OSS) is freely available for use, reuse, fixing, and redistribution by the users. OSS are adopted as the embedded systems, the server usage, and so on because of the quick delivery, the cost reduction and the standardization of systems. Therefore, OSS is often used not only for the personal use but also for the commercial use. Many OSS are developed under the peculiar development style known as the bazaar method [1]. According to this method, many faults are detected and fixed by the developers around the world, and the fixed result will be reflected in the next release. Also, many OSS are developed and managed by using the fault big data recorded on the bug tracking systems. Then, many OSS are developed and maintained by several developers with many OSS users.

Although OSS is useful as the reusable software, it also has several issues. The examples of issues are as follows [2].

1. Selection evaluation and licensing
2. Vulnerability support
3. Maintenance and quality assurance
4. Supply chain management
5. Personal capacity and education
6. Organizational structure

7. Community activities

Examples of the issues described above are listed in Table 1.1. Table 1.1 shows the issues from various perspectives in terms of OSS. Therefore, many companies and individuals need to prepare well before using OSS.

Various researches have been conducted on each of the issues related to OSS. In this thesis, we consider the three issues listed in Table 1.1. The issues are 1. Selection evaluation and licensing, 2. Vulnerability support, and 3. Maintenance and quality assurance.

1.2 Purpose

In this thesis, we focus on the three issues listed in Chapter 1.1.

In "1. Selection evaluation and licensing,"we attempt to derive OSS-oriented EVM by applying the earned value management (EVM) to several open source projects. The EVM is one of the project management methodologies for measuring the project performance and progress. In order to derive the OSS-oriented EVM, we apply the stochastic models based on software reliability growth model (SRGM) considering the uncertainty for the development environment in open source projects. Then, we can apply it to bullseye charts by deriving the OSS-oriented EVM. The bullseye chart provides a means of visualizing the simultaneous progress toward each goal. In this thesis, we also discuss the application of the OSS-oriented EVM to the bullseye chart. We also improve the method of deriving effort in open source projects. In case of applying the existing method of deriving effort in open source projects, it is not possible to derive some indices in OSS-oriented EVM, thus we resolve this issue. The derived OSS-oriented EVM helps OSS users and open source project managers to evaluate the stability of their current projects. It is an important decision-making tool for them regarding their OSS usage decisions and projects. From a different perspective, we also evaluate the stability of the project in terms of the speed of fault fixing by predicting the time transition of fixing reported OSS faults in the future.

In "2. Vulnerability support,"in terms of open source project managers, we create the metrics to detect faults. The reported faults with a high priority is speedy fix, and a long time to fix, and predict. In addition, we try to improve the detection accuracy of the proposed metrics by learning not only the specific version but also the bug report data of the past version by using random

Table 1.1: Examples of issues in using OSS.

#	Issues in OSS	Examples of issues in using OSS
1	Selection evaluation and licensing	• Difficulty in safe OSS selection • Unable to decide whether to entrust the selection of OSS to a subcontractor • Violation of license by modification of OSS • Unable to evaluate OSS license • Balance between functionality, safety, and reliability
2	Vulnerability Support	• Granularity and variability in software management • Difficulty in understanding product configuration (components, OSS, etc.) and vulnerabilities • Inadequate vulnerability notification, application methods, and support • Legacy devices and systems, IoT devices • Cost-effectiveness (effort and cost)
3	Maintenance and quality assurance	• Abandonment of unsupported OSS • Difficulty in planning operations and maintenance • Necessity of long term support • Indemnification in case of failure • Setting of disclaimer of liability • Disposal and termination of use
4	Supply chain management	• Difficult to coordinate comprehensively • Trading practices not based on OSS assumptions • Variation in responsiveness among companies • OSS management within all supply chains • Ensure vulnerability constitution • Necessity of contractual clarification
5	Personal capacity and education	• Insufficient risk management by executives • Insufficient criteria for judgment and response • Insufficient literacy, insufficient knowledge • Dependence on others, dependence on community
6	Organizational structure	• Lack of internal system to support OSS utilization • Unstable response to vulnerabilities • Many manufacturers do not have PSIRTs up and running
7	Community activities	• Community Assessment • Internal recognition and understanding of activities • Activation, increased voice

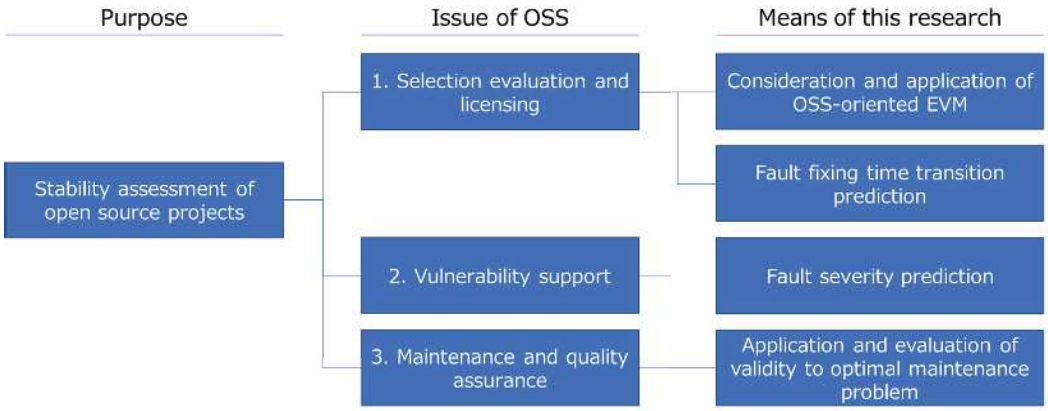

Fig. 1.1: Summary of this research.

forest considering the characteristic similarities of bugs fix among different versions. This allows the project managers to identify faults that should be prioritized for fixing when a large number of faults are reported, and facilitates project operations.

In "3. Maintenance and quality assurance,"as an optimum maintenance problem, we predict the appropriate OSS version-up timing considering the maintenance effort required by OSS users after upgrading the OSS. It is dangerous in terms of the vulnerability to continue using the specified version of OSS ignoring the End of Life (EOL). Therefore, we should upgrade the version periodically. However, the maintenance cost increase with the version upgrade frequently. Then, we find the optimum maintenance time by minimizing the total expected software maintenance effort in terms of OSS users. In particular, we attempt to reflect the progress of open source projects by using OSS-oriented EVM in deriving the optimal maintenance time.

The scope of this research is summarized in Fig. 1.1.

1.3 Structure

This thesis is structured as follows:

Chapter 1 describes the background and purpose of this research as Introduction.

Chapter 2 compares the proprietary software and the OSS. Then, we summarize the issues and conventional approaches to OSS and OSS development.

Chapter 3 discusses the method used in this research to derive the maintenance effort for open source projects. There are several issues in the conventional method of deriving maintenance effort for open source projects. In particular, it is difficult to derive the OSS-oriented EVM, as described in Chapter 4, using the conventional method of deriving maintenance effort. Therefore, it is necessary to improve the derivation method in terms of the maintenance effort.

Chapter 4 discusses how to derive OSS-oriented EVM applying EVM to open source projects in terms of "1. Selection evaluation and licensing" of OSS issues. In addition, we derive and discuss bullseye chart by applying OSS-oriented EVM.

Chapter 5 also discusses the issues in OSS in terms of "1. Selection evaluation and licensing." We evaluate the stability of the project in terms of the speed of fault fixing by predicting the time transition of fixing OSS faults reported in the future.

Chapter 6 discusses the issue in terms of "2. Vulnerability support" in OSS. We create several metrics to detect faults that reported faults have a high priority to fix and a long time to fix, and predict.

Chapter 7 discusses the application of OSS-oriented EVM to the optimal maintenance problem in terms of "3. Maintenance and quality assurance" of OSS issues. In other words, we predict the appropriate OSS version-up timing considering the maintenance effort required by OSS users after upgrading the OSS by using OSS-oriented EVM. We also verify whether the appropriate OSS version-up timing is appropriate in terms of the number of remaining faults.

Finally, Chapter 8 describes summary of this research, future issues, and future prospects.

In particular, this thesis focuses on several data sets. Actually, many stochastic models for software reliability assessment have been proposed and development by many researchers in the past. There are many kinds of OSS, e.g., application, server, embedded system, cloud software, etc. Then, the developers have to select the appropriate method for the specified OSS. Therefore, we use the appropriate data set for the proposed method. Then, the embedded OSS, the application OSS, and the cloud OSS are used as the data sets of this thesis. Thereby, we will be able to assess appropriately from various standpoints.

Appendix discusses the status transition diagram of details in Bugzilla. Also, we describe the maximum likelihood estimation method we use in this thesis to estimate the parameters of the SRGMs. In addition, Appendix contains the results of analyses by various proposed models not

described in individual chapters.

Chapter 2

Background of Proprietary Software and Open Source Software

There are two main types of software: proprietary software and OSS.

Proprietary software is software licensed under the exclusive legal rights of the copyright holder, and the users of proprietary software are permitted to use it under certain conditions. In general, proprietary software users must enter into an end-user license agreement (EULA) between the user and the software author in order to install the application on their hardware. By accepting the EULA, the user agrees not to modify the software, derive source code, or redistribute the software.

OSS is software that is usually provided free of charge under a license, most commonly a "copyleft" license such as the GNU General Public License (GPL), giving legal permission to copy, distribute, and modify the software. Therefore, OSS are used under the various situations, because OSS are useful for many users to make cost reduction, standardization, and quick delivery. Many OSS programs are known as high performance and reliability, even though many OSS are free of charge. Furthermore, many IT companies often develop OSS for the commercial use.

In this chapter, we explain the differences between proprietary software and OSS development. Then, by explaining the development styles and project management methods of each software, we will summarize the differences between proprietary software and OSS development, as well as the issues involved in OSS development.

2.1 Examples of development style in proprietary software

Proprietary software are developed in a variety of ways. The Waterfall model, the V-Shaped model, the Agile model, the Extreme Programming, the Spiral model, and the Prototype model

are described as examples of major development methods.

The Waterfall model [3] consists of the following phases:

- Requirements specification (Requirements analysis)
- Software design
- Implementation and Integration
- Testing (or Validation)
- Deployment (or Installation)
- Maintenance

Traditionally with the Waterfall model, the developers can only start on the next phase when the previous phase is finished. Therefore, it is called the Waterfall method. This method is a linear method in which there is a big emphasis on collecting requirements and designing the software architecture before doing development and testing. The advantage of this method is that projects are well planned, mid-project costs for changing requirements are minimized, and these projects tend to be well documented. This usually results in a major version release with a significant number of new features every few years. The disadvantage is that it is very hard to adjust the feature set in the middle of development. It often happens as problems are uncovered in development or changing business environments change what is needed in the software. For this reason, many companies have "feature freezes" in which they delay changes to features that should be included in a given version of software once development of the software begins. As a result, the needed features are pushed to later major versions, and software users end up waiting years for those features.

The V-Shaped model is similar to the Waterfall model. This model is executed sequentially [4]. Each phase must be completed before the next phase can begin. In this model, the testing is more important than in the Waterfall model. The testing procedures are developed early in the lifecycle, before coding takes place, in each phase prior to implementation. The requirements begin the lifecycle model as in the Waterfall model. Before the development begins, the

system test plan is developed. The test plan focuses on meeting the functionality specified in the requirements gathering.

Agile model are meant to adapt to changing requirements, minimize development costs, and still give reasonable quality software [5, 6]. The agile project is characterized by many incremental releases each generated in a very short period of time. Also, the agile model focuses on the collaboration between customers and developers and encourages development teams to be self-organizing [7]. Typically, all members of the team are involved in all aspects of planning, implementation, and testing. The agile project is done in small teams of no more than nine people, who meet daily. The team may include customer representatives. The emphasis is on testing the software as it is written. The downside of agile methods is that they do not work well for projects with hundreds of developers, projects that last for decades, or where requirements emphasize strict documentation, well-documented design and testing.

Extreme Programming (XP) is a frequent release development methodology in which developers work in pairs for continuous code review [8]. This gives very robust, high quality software, at the expense of twice the development cost. There is a strong emphasis on test driven development.

The Spiral model, similar to the waterfall model, is one of the life cycle models of software development [9]. The spiral shape is a way to visualize the way development should occur under this methodology; it should loop until the end of the project when all of the client's requirements are addressed. Spiral projects start small, first investigating the highest risk issues then slowly expand the project once those key components are functioning. Two or more phases of prototyping are done before the final implementation. The spiral model is considered to be better than the waterfall for large, expensive, complicated projects. The spiral development is generally considered the inappropriate for small projects.

2.2 Cost estimation methodology for proprietary software development

2.2.1 COCOMO/COCOMO II model

COCOMO [10] is known as an effort estimation and development period estimation model. Specifically, its models the relationship between software size (e.g., lines of code), cost (e.g., development effort), effort, and development time. There is also a method of refining the model using project performance data. All methods for creating model equations are disclosed, and the models themselves can be calibrated or expanded according to the needs of the organization. The model equations include effort variation due to system and project characteristics as a scale factor and cost driver. Therefore, COCOMO can be used not only for estimation, but also for investments that require effort for software, trade-offs in cost, development time, functionality, performance, and quality factors, decisions in risk management for cost and development time, decisions on development, purchase, reuse, etc., and organizational improvement, and has a wide range of applications. The basic relationship between software scale and effort in the COCOMO model is expressed by the following equation.

$$Effort = a \times Scale^{b}, \tag{2.1}$$

where, $Effort$ is the development effort, $Scale$ is a measure of software scale, and a and b are constants. It is also assumed that the relationship between $Effort$ and $Scale$ is different from a simple linear proportional relationship. Depending on the value of b, the exponent of $Scale$, the relationship between $Effort$ and $Scale$ changes. If $b < 1$, the productivity increases as $Scale$ increases. If $b = 1$, productivity does not change as $Scale$ changes. If $b > 1$, the productivity decreases as $Scale$ increases. In the COCOMO model, $Scale$ was defined as the number of lines in the source program. However, since this makes it difficult to estimate the analysis and design process, the object points and function points were added as elements of $Scale$ in the COCOMO II model [11].

2.2.2 Function point method

The function point method is a technique to estimate the scale of software in terms of the number and complexity of functions to be implemented [12]. This method estimates the number and difficulty of functions from the user's point of view, such as program input screens, output screens, output forms, and files, by calculating the number of functions and applying a correction factor.

Based on the interaction of the system components internally and with external users, applications, etc. They are categorized into five types:

- Transaction functions
  - External Inputs: Data input function to update internal logical files
  - External Outputs: Data output function to external parties
  - Inquiries: Data processing function that combines inputs and outputs
- Data Functions
  - Internal Logic File: File managed by the software being developed
  - External Logic File: File referenced by the software being developed

2.2.3 Lines of Code method

The Lines of Code (LOC) method is a method for estimating system development costs based on the number of program steps. If the program does not exist, the number of steps is estimated by referring to past development cases, etc. In the LOC method, the estimate may be affected by the way the program is written and the skill of the programmer. The LOC method can be used as a guide for estimating effort. However, its estimation accuracy is lower than that of the function point method and COCOMO model.

2.2.4 Bottom-up estimating method

Bottom-up estimating is a method of estimating and accumulating the amount of resources for each component by breaking down deliverables and tasks. There are two methods: one is to

structure the software and estimate it in units of functions, and the other is to break down the work to be performed into a WBS (Work Breakdown Structure) and estimate the effort for each WBS.

2.2.5 Software reliability growth model

In the testing process of software development, the number of potential faults in the software decreases with the progress of testing time, because a lot of resources are spent on fault detection and fix. Therefore, the probability of software fault occurrence decreases with the testing time. Then, the software reliability and the interval of software fault occurrence time increase. Such software reliability model describes the software fault phenomenon. This is called SRGM.

There are two types of SRGM proposed in the past: (1) exponential and (2) S-shaped models.

(1) makes the assumption that a finite source code has a finite number of defects. The number of defects possibly increases as new defects are introduced by fixes or by the implementation of new functionality. Some models explicitly account for the introduction of new defects during testing, while others assume that they can be ignored or are handled by a statistical fit to the SRGM data.

(2) assumes that the fault detection rate is proportional to the number of faults in the code. Since the total number of faults in the code decreases each time a fault is repaired, the fault detection rate is assumed to decrease as the number of faults detected (and fixed) increases.

SRGM has been applied not only to the number of potential faults in software, but also to the prediction of development effort [13]. Therefore, in this thesis, we use SRGMs in development effort estimating. In addition, this thesis uses the exponential model [14], the delayed S-shaped model [15], and the infection S-shaped model [16], one of the major SRGM. The exponential model does not consider the time it takes to find a fault or recognize a failure. The delayed S-shaped model considers that there is a delay between the detection of a fault and the recognition of it as a defect. The infection S-shaped mode is a model in which the fault detection rate increases with proficiency as the test is conducted. The distribution is shown in Figs. 2.1-2.3.

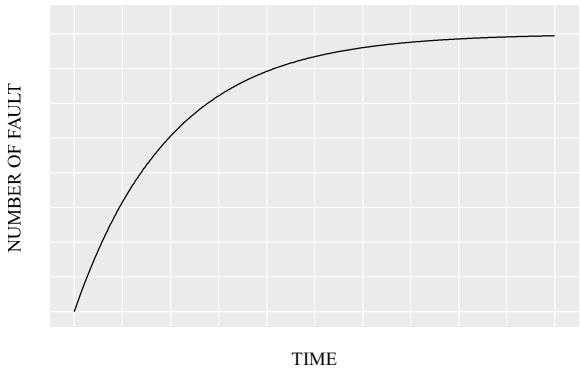

Fig. 2.1: Example of exponential model.

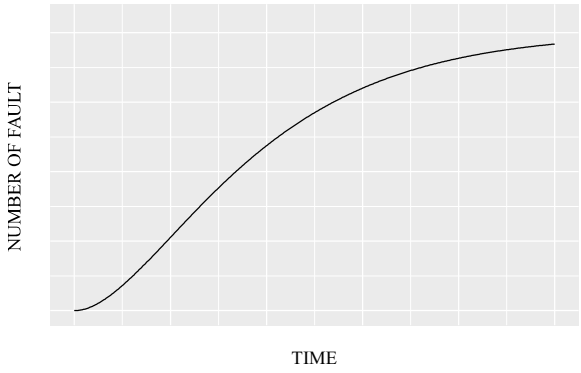

Fig. 2.2: Example of delayed S-shaped model.

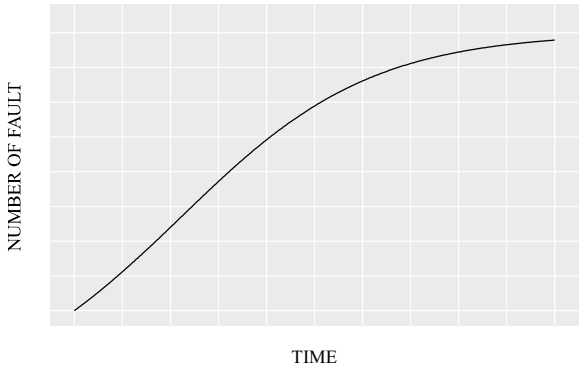

Fig. 2.3: Example of infection S-shaped model.

2.3 Example of cost management method for proprietary software development

2.3.1 Earned Value Management

EVM [17] is a typical cost management method. EVM is a method to grasp and manage the progress of a project, the performance of tasks, and future forecasts by using earned value, and is used in many industries regardless of IT companies. EVM was first applied to the U.S. missile development program in the 1960s. The basic concept of EVM was conceived with an emphasis on efficient cost management, and the Cost/Schedule Control System Criteria (C/SCSC), the original source of EVM, was established in 1967. In 1998, the American National Standards Institute (ANSI) announced the Earned Value Management System (EVMS) for use in the private industry. This is the current standard for EVM [18]. EVM is able to manage not only the progress of activities, but also the costs, on the same graph. EVM has four basic elements and several indicators:

- Planned Value (PV): PV represents the budget for tasks or work to be completed by a specific point in time. PV is a baseline indicator for EVM, as it allows for the determination and identification of cost overruns.
- Earned Value (EV): Earned Value (EV): EV is the amount of work completed to a specific point in time.
- Actual Cost (AC): Actual Cost (AC): AC is the total actual cost up to a certain point in time. We can determine that a project is on schedule when AC is the same as PV.
- Budget at Completion (BAC): BAC, like PV, is set at the project planning stage. In EVM, when the total actual cost is within the BAC, the project is considered on-budget; otherwise, the project is considered over-budget.

EVM is represented by a graph with time on the horizontal axis and cost on the vertical axis. The three indicators displayed are EV, PV, and AC, and an example is shown in Fig. 2.4.

Various values can also be calculated using each indicator, and these are shown in Table 4.1.

Table 2.1: Several examples of the indices used in EVM.

EVM Elements	Explanation
Planned Value (PV)	total budget estimated at the planning phase up to a certain point
Earned Value (EV)	total budgeted cost resources of processes completed at a certain point
Actual Cost (AC)	total actual cost resources invested
Budget at Completion (BAC)	total budget to completion defined at the time of planning
Cost Variance (CV)	shows whether a project is under or over budget CV=EV-AC
Cost Performance Index (CPI)	evaluates how efficiently the project is using its resources. CPI=EV/AC
Schedure Variance (SV)	determines whether a project is ahead of or behind schedule. SV=EV-PV
Schedule Performance Index (SPI)	evaluates how efficiently the project team is using its time. SPI=EV/PV
Estimate at Completion (EAC)	final cost of the project in case of continuing current performance trend. EAC=BAC/CPI
Estimate to Complete (ETC)	shows what the remaining work will cost. ETC=(BAC-EV)/CPI

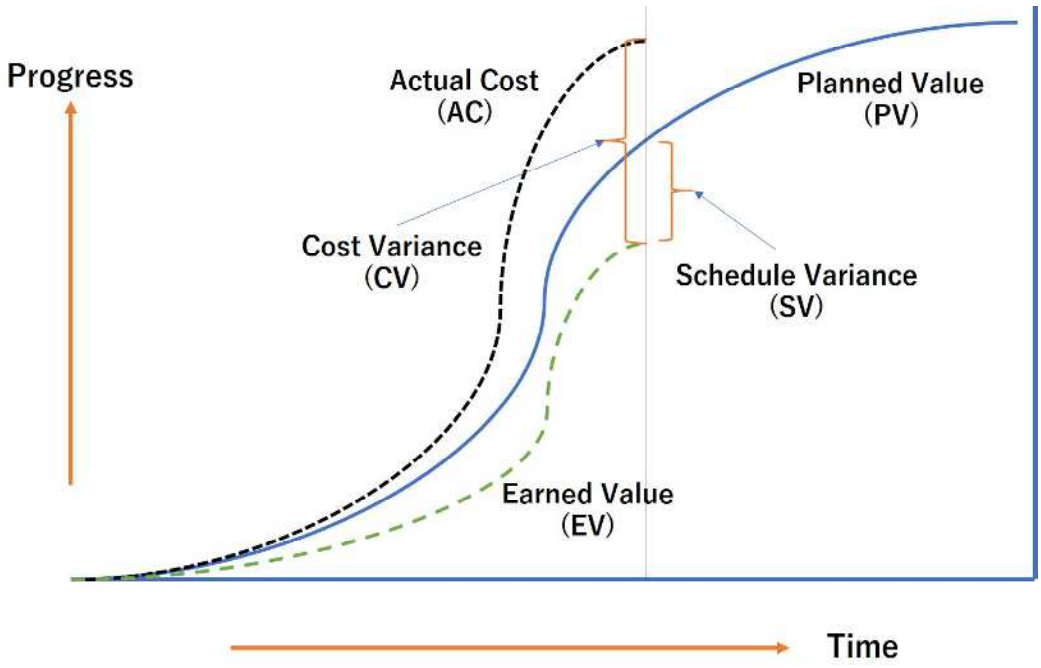

Fig. 2.4: An example of EVM.

2.3.2 Bullseye chart

The bullseye chart is a chart that integrates SPI and CPI. By plotting the SPI results on the horizontal axis and the CPI results on the vertical axis, we can evaluate the project status from two perspectives: schedule and cost. Fig. 2.5 shows how the bullseye chart is presented.

- A: Both progress and cost progressing within the planned progress and cost
- B: Progress is slow, but cost overruns are not occurring
- C: Slow progress, excessive costs
- D: Progress is on schedule, but cost overruns are occurring

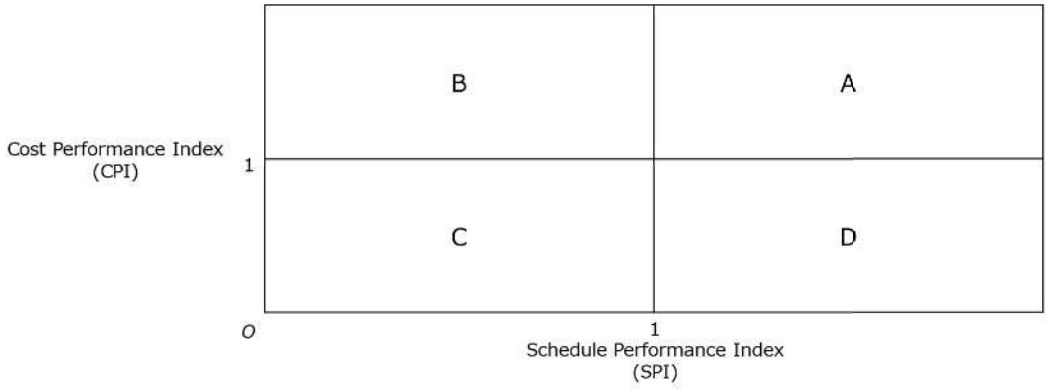

Fig. 2.5: An example of Bullseye chart.

2.4 Concept of open source software development

The Bazaar method is known as the method of OSS development. The Bazaar method is a method in which multiple participants develop without any special restrictions, and the designer brings them together to create a single product. The Bazaar method was originally introduced in Eric Steven Raymond's article on OSS [1]. In the paper, it is used as a contrast to the Cathedral method.

The Cathedral method is a method in which one person or a small group of people lead the development of software, like a "cathedral," which is a building such as a temple or a hall of fame. In contrast, the Bazaar method is a method in which participants develop software freely in an environment where their originality is respected, and the results are brought together like a "bazaar" to create a single piece of software while retaining the best. The Bazaar method is not strict about releases, leaving to others what can be entrusted to them, and releasing the results early and often. In fact, the Bazaar method has been adopted as a development method, especially for Linux [19], a UNIX-compatible OS.

One of the fundamental differences between the two methods is their approach to bugs. In the Cathedral method, the bugs should not exist. In other words, a small number of developers search for bugs (and fix them) for a long period of time before release in order to ensure a bug-free release. On the other hand, the Bazaar method, bugs are fixed when the bug found. After release, bugs are searched for by users all over the world. When a bug is found, it is fixed by the

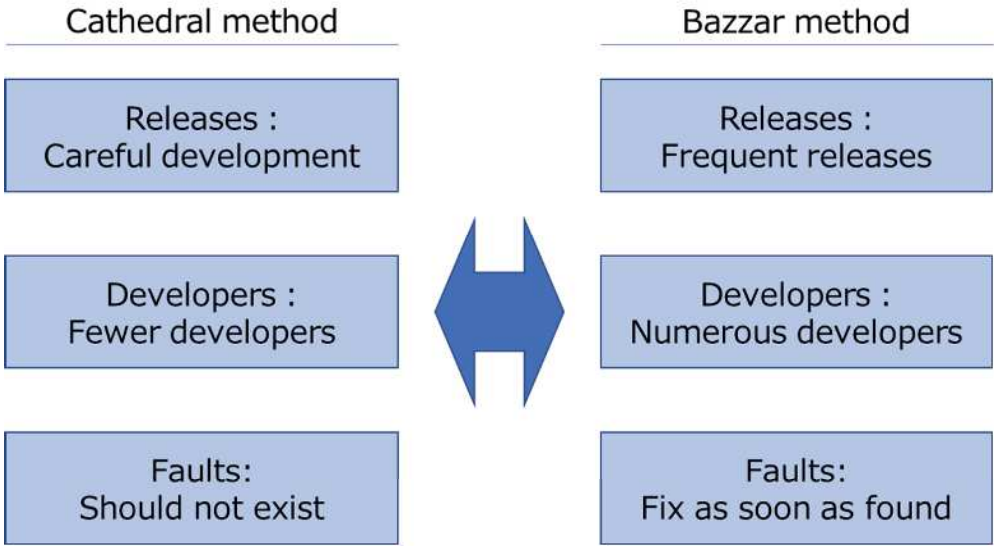

Fig. 2.6: Differences between Cathedral method and Bazzar method.

developers around the world and reflected in the next release. In the bazaar method, both users and developers may be the majority of the world's population, as shown in Fig. 2.6, which allows for quicker bug finding and fixing, contributing to shorter release cycles.

2.5 Examples of development style in open source software

Many OSS development projects focus on development tools and platforms, and developers themselves often participate in the development, simultaneously participating in the roles of customer and developer. The Open Source Initiative [20] keeps track of and grants licenses for software that complies with the OSS definitions. Several researchers [21, 22] suggest that the OSS project represents in fact a virtual organization. In many respects, OSS follows the same approach as other agile methodologies. For example, the OSS development process starts with early and frequent releases and lacks many of the traditional mechanisms used to coordinate software development, such as planning, system-level design, scheduling, and defined processes. In general, OSS projects consist of the following tangible phases [23]:

- Identification of the problem

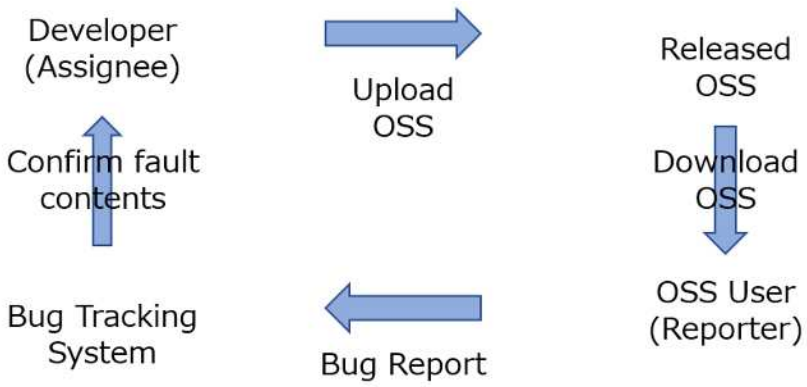

Fig. 2.7: Example of OSS development style.

- Identification of volunteers
- Proposing a solution
- Code development and testing
- Code change review
- Code commitment and documentation

OSS are promoted by an unspecified number of users and developers. The bug tracking system is also one of the systems used to develop OSS. The bug tracking system registers a lot of defect information, such as the status of fixes, the details of fixes, and the priority of fixes. The information is used to manage the open source project. The development style of OSS using the bug tracking system is shown in 2.7. Then, a person is assigned to respond to the reported fault or improvement request, and the correction is made. The fixed contents are reflected in the new version of the OSS. In addition, details of the development style in Bugzilla [24], is known as one of the bug tracking systems, are described in the Appendix.

In proprietary software development, development begins after the resources (personnel and equipment) required for software development have been acquired. On the other hand, OSS development depends heavily on the voluntary participation of new development collaborators after the start of development. If developers do not have the knowledge necessary to solve technical

problems that arise during development, existing developers are forced to acquire new knowledge. Thus, this can easily lead to stagnation and failure of development projects. The majority of projects in large OSS development communities (e.g., more than 80% of all projects in SourceForge.net [34]) have only three or fewer developers, a serious obstacle to active OSS development [35].

2.5.1 Examples of cost estimation and management methods in open source software development

Considering the development style of OSS, estimating effort for open source projects as a whole is difficult because various factors need to be considered. Therefore, several methods have been proposed to estimate effort to correct each reported fault and to estimate effort within a specific period of time [26–29]. In addition, a small number of time-series effort estimation methods have also been proposed [32]. In open source projects, it is difficult to estimate effort for the entire project because a fixer is assigned to each reported fault or improvement request. Therefore, there are few studies on time-series effort estimation methods.

As a research on cost management methods in open source projects, there are researches to predict the duplication of faults reported to the bug tracking system [30, 31]. There are also optimal maintenance problem that predict when OSS users will upgrade their OSS based on the number of effort expenditure on the open source project [32, 33]. There are several types of cost management methods in open source projects, but it is difficult to apply the same cost management methods to open source projects as to proprietary software development, especially EVM. Although there is research in open source projects that attempts to apply EVM, it has been incomplete because it has not been possible to derive PV and BAC, important indices for EVM.

2.5.2 Applicability of existing cost estimation and management methods to open source projects

Typical cost estimation and management methods for proprietary software development are described above. However, it is difficult to apply these methods to OSS development because the development style is very different. Open source projects have the characteristic that anyone can

participate in the project. Also, not all fault reporters and fixers have sufficient skill sets, and there is often turnover in project members. In addition, the project participants' work time and development environment may not satisfy the project manager's expectations. Therefore, there is a large degree of uncertainty and variability in cost estimates. This is characteristic of open source projects, and uncertainty must be considered when properly estimating and managing costs.

In addition, in an open source project with a large uncertainty, the stability of the project is often a requirement for using OSS as an OSS user. Therefore, "project stability" in an open source project is defined as the state in which the creation provided by the open source project can be provided in a stable state.

In subsequent chapters of this thesis, we define the uncertainty as the cause of variation in cost estimates that is unique to open source projects. Then, we will examine how to approach the issues in open source projects described in chapter 1. Also, the evaluation of project activities in terms of effort is regarded as an evaluation of project stability.

In this thesis, we also define a large project as a project that has enough developers and focus on those projects, referring to chapter . Therefore, open source projects that satisfy the following criteria will be considered in this thesis.

- Well-known OSS: Many users, i.e., testers
- Enough developers (more than 5 fault fixers): Problems due to lack of developers are less likely to occur
- More than 1,000 reported faults: Active open source community

In this thesis, we examine approaches to OSS issues from five perspectives, as shown in Fig. 2.8. In particular, we consider uncertainty in three of the approaches.

To summarize, the uncertainty and stability are defined as follows in this thesis.

- The effort has the fluctuation in the large-scale OSS. Then, the fluctuation of effort can be represented as the uncertainty.
- The uncertainty as effort can illustrate as the noise of Wiener process.
- Then, the size of noise by Wiener process can define as the stability.

Fig. 2.8: research approaches in this thesis.

The above mentioned points, we assess the uncertainty by using the size of noise by Wiener process in this thesis. Then, the stability is the assessment measurement of uncertainty, e.g., the stability can show by using the white noise parameter.

Chapter 3

Maintenance Effort Derivation Method

Software faults continuously occur in the software development process. The undetected faults potentially lead to significant problems to the individuals and companies. The purpose of software maintenance is not only to improve performance, but also to fix faults and enhance software functionality, leading to higher reliable software.

In recent years, the usage of OSS has become increasingly popular. The OSS is code-designed to be accessible to everyone. Because OSS is developed by open source communities rather than by specific companies, it is often more inexpensive, more flexible, and longer-lasting than the proprietary software. However, because anyone can join an open source community, there is a high degree of uncertainty in project progress due to differences in the project members' skills, the development environments, and the length of activity. Therefore, it is difficult to predict the progress of open source projects, and it is important for many users and companies to understand the development and operational status of open source projects in order to make decisions about upgrading or installing the OSS.

In large open source projects, the project teams typically use a bug tracking system to record reported software fault reports in order to coordinate and avoid duplication of tasks. Many fault reports recorded by OSS users and developers are stored in the bug tracking system. Then, the faults are assigned in order to fix by the member of OSS developers. Developers on the project team rely on the reports to manage and fix reported faults, and fault reports are an important source of information for developers to resolve faults, especially during the maintenance process of large software.

EVM is one of the famous project management methods. EVM is a project management technique to quantitatively evaluate project execution in terms of budget and schedule, and to grasp cost efficiency and progress rate at once. Applying project effort data to EVM, we need a time series of effort data. However, the open source projects manage the information on individual

fault data, but do not manage effort for the project as a whole.

COCOMO II model [11] is a typical method for estimating the effort in software development. The COCOMO II model can calculate the estimated values of development effort, the number of developers, and the productivity from the estimated value of software scale, e.g., the number of lines of source code. However, it is difficult to apply the COCOMO II model in order to predict the effort in time series, because the open source projects are very different from proprietary software development methods.

There are several methods for estimating the effort for open source projects in previous researches [25–28]. Robles et al. [25] presented a novel approach to estimate the effort of large scale OSS by considering the data from source code management repositories. In addition, for estimating the effort, they use the survey data answered by over 100 developers. Kula et al. [26] estimated the maintenance effort from the complexity and duration of micro processes per an issue. Rakha et al. [27] survey the effort needed for manually identifying duplicate reports in four open source projects, i.e., Firefox, SeaMonkey, Bugzilla, and Eclipse-Platform. Mishra et al. [27] have proposed a metric for computing the effort and contribution of a patch reviewer based on modified file size, patch size and program complexity variables. Qi et al. [29] have proposed a metric for estimating the effort required to develop a project by dividing it into two categories: full-time and part-time developers. These researches estimate the total effort required by the project and do not estimate time-series effort. On the other hand, there are only a few researches that perform time-series effort estimation [32]. Tamura et al. [32] created a time series of effort data using the period from the date of report to the last update of fault data obtained from a bug tracking system and the total number of people involved in the fault report. By creating effort data and applying an effort estimation model such as in this previous research, we can estimate the EVM values in the future. However, Tamura et al.'s research is a simplified method of deriving effort, accounting for all effort required to fix a fault at the time the fault is reported. In practice, the application of project management methods such as EVM is less effective unless the effort data is prorated within the fault fixing period.

Since an unspecified number of people are involved in the development of an open source project, it is difficult to see the entire project and the people involved, and it is not easy to evaluate the progress and stability of the project. Therefore, it is significant for future development that

there are more options for project evaluation methods in OSS development.

In this thesis, we improve the time-series effort data creation model of the previous research and examine its applicability to the SRGMs.

3.1 Effort derivation

In order to derive the time-series effort data for the project management methods such as EVM, it is necessary to derive the time contributed by project participants to open source projects. Tamura et al. [32] assumes that there are two types of contributors to open source projects: fault reporters and fault assignees. In the previous research, the effort data has been calculated using the time required for fault reporting and correction by fault reporters and fault assignees, respectively. In this thesis, a similar approach is used to calculate effort.

3.1.1 Effort calculation method

Participants in open source project have very different levels of the contribution to the project due to differences in their skills and level of participation in the project. Therefore, considering their contribution to the project, the effort of fault reporters and fault assignees for the i-th fault response are as follows:

$$effort_{ri} = T_{ri} N_r C_{ri}, \tag{3.1}$$

$$effort_{ai} = T_{ai} N_a C_{ai}, \tag{3.2}$$

where T_{ri} is the fault reporting interval time, N_r is the number of reporters, C_{ri} is the contribution rate of the fault reporter, T_{ai} is the fault fixing time, N_a is the number of assignees and C_{ai} is the assignee's contribution rate. In particular, C_{ri} and C_{ai} are the number of times of the particular reporter or assignee appears in the project divided by the total number of reporters and assignees participating in the project. Thus, the total effort required to fix faults up to the k-th fault in the open source project is as follows:

$$effort_k = \sum_{n=1}^{k} (effort_{ri} + effort_{ai}). \tag{3.3}$$

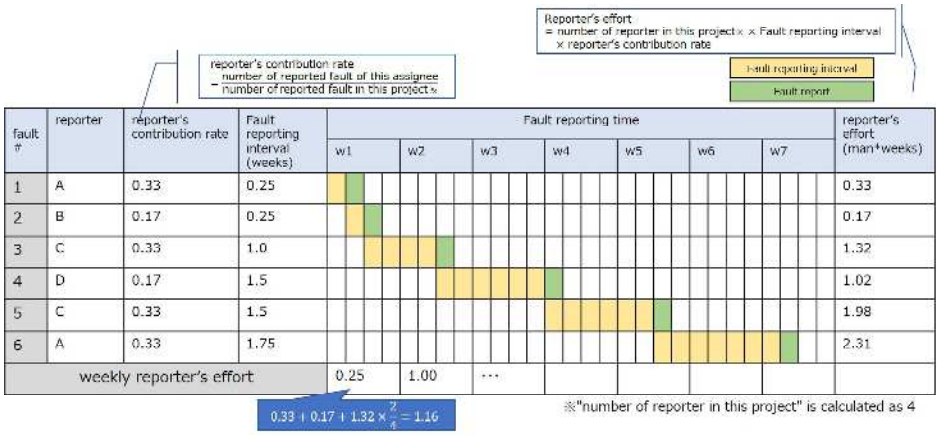

Fig. 3.1: The method of prorating reporter's effort in open source projects.

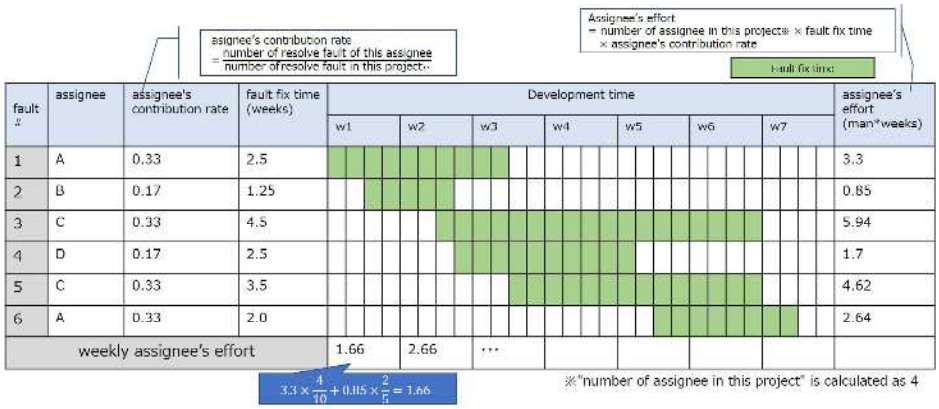

Fig. 3.2: The method of prorating assignee's effort in open source projects.

3.1.2 Data preprocessing

In previous research, reporter and assignee effort expenditure was defined as occurring at the timing of the fault report. However, in terms of the reporter's effort, some effort expenditure is required, before the reporter reports a fault, so it is necessary to account for some effort before the timing of the report. Therefore, in this thesis, the time required for reporting is prorated on a weekly basis in case of creating weekly data, as shown in Fig. 3.1. In addition, as shown in Fig. 3.2, the efforts of the assignee are also prorated on a weekly basis for the fault fixing time.

The data used to calculate the effort is obtained from the bug tracking systems. In this thesis, we use the fault data registered in Bugzilla. In order to calculate the effort, we use the data

Table 3.1: Data items used to calculate the effort.

Data item	Item details
Opened	The date the fault was reported.
Changed	The last update date of the fault. The time difference between Changed and Opened is used for the effort calculation.
Reporter	The reporter of the fault. The reporter is used to calculate the number of reporters and the reporter's contribution rate.
Assignee	The assignee in charge of handling the fault. The assignee is used to calculate the number of assignees and the assignee's contribution rate.

registered in the fault data as shown in Table 3.1.

3.2 Numerical examples

3.2.1 Used data set

In this thesis, we use the data set of open source project for applying proposed method. For applying the proposed method to actual project effort data set, we use the data of Red Hat Enterprise Linux (RHEL) [36] and LibreOffice [37] obtained from Bugzilla. RHEL is a commercial Linux distribution developed and marketed by Red Hat. It is mainly developed for use in data centers and high-end server applications where stability and security are required. LibreOffice is an office suite OSS provided by The Document Foundation. It consists of word processing software, spreadsheet one, presentation one, drawing one, database management system, and formula editing one.

3.2.2 Effort calculation result

Figs. 3.3 and 3.4 show the results of the effort calculation using our proposed model and Tamura model [32]. Figs. 3.3 and 3.4 show that the proposed model is more similar to the S-shape, the shape that can occur in proprietary software development. The reason our proposed model is similar to the S-shape is that effort variation in open source projects tends to be based on the S-shape. Therefore, the effort change of open source projects tends to be similar to the proprietary software development, thus SRGM is applicable to open source projects.

Tamura model [32] is a simplified effort derivation method that accounts for all the efforts required to correct a fault when a fault is reported. Therefore, effect increases significantly at the timing of fault reporting and the start of fault fixing. Figs. 3.3 and 3.4 also show similar results, and the development effort derivation method proposed in this research is more suitable when applied to effect evaluation indices such as EVM.

Previous research has confirmed that the SRGMs can be applied to effort in proprietary software development [13]. The application of SRGMs to maintenance effort in OSS development has been confirmed to be possible when using the simple maintenance effort derivation results [32], but not when deriving maintenance effort in detail. The proposed method for deriving maintenance efforts in this thesis shows S-shaped trends, indicating that the SRGMs can be used to predict the maintenance effort.

3.3 Conclusion in this chapter

In this thesis, we have examined the time-series effort data creation model for open source projects. We have also examined the applicability of the proposed effort calculation model based on SRGMs using actual project data. As a result, the effort data of open source projects show similar trends to proprietary software development and shows its applicability to SRGMs. Therefore, by using our proposed method to calculate the effort of projects, it is possible to apply the proposed method to project management methods such as EVM. It is not easy to evaluate the progress and stability of open source projects because of the large scale and lack of visibility of the entire group of people involved in the development of OSS. Therefore, we believe that project evaluation methods used in conventional software development will increase the number

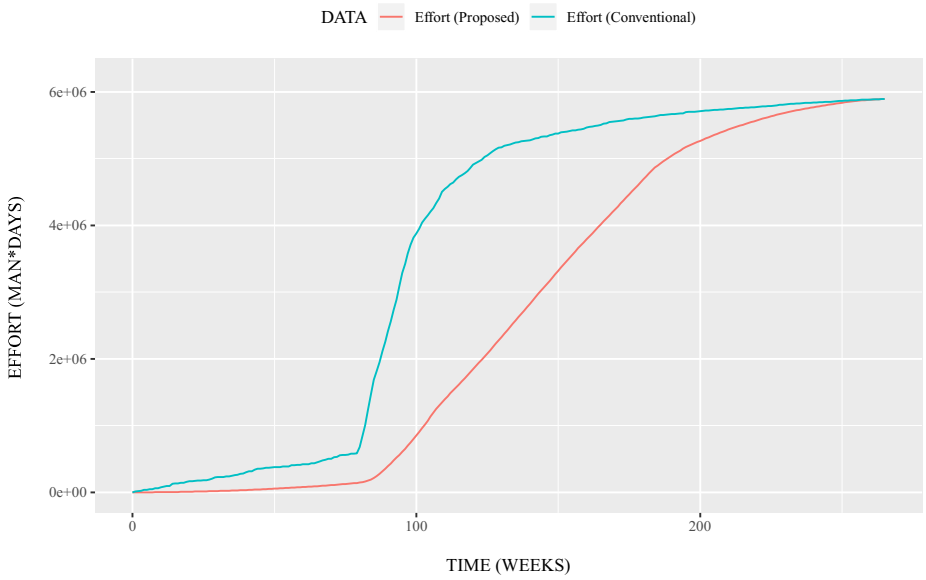

Fig. 3.3: Comparison of effort calculation results in RHEL project.

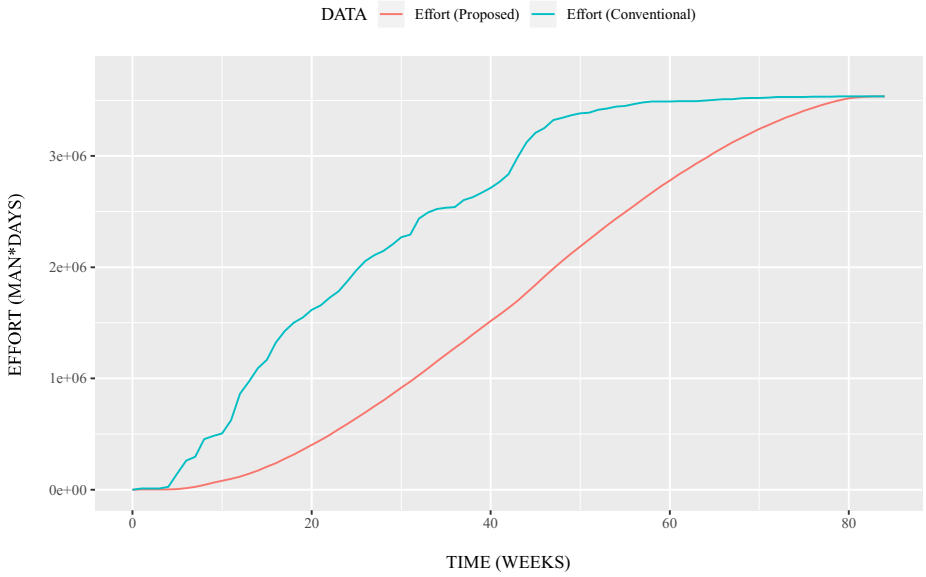

Fig. 3.4: Comparison of effort calculation results in LibreOffice project.

of evaluation methods for OSS development and contribute greatly to the development of OSS. On the other hand, we have used only two project data in this research. Therefore, it is necessary to apply the proposed method to more project data to investigate patterns applicable to SRGMs

for open source projects.

Related Paper in this Chapter

1. H. Sone, Y. Tamura, and S. Yamada, “Study of effort calculation and estimation in open source projects,” International Journal of Reliability, Quality and Safety Engineering, Vol. 30, No. 3, World Scientific, pp. 2350011-1-2350011-13, 2023.

Chapter 4

OSS-orientd EVM

Recently, EVM [17] is applied to the actual software projects under various IT companies. The EVM is one of the project management techniques for measuring the project performance and progress. The EVM basically measures the project performance and progress using three indices: Planned Value (PV), Earned Value (EV), and Actual Cost (AC), respectively. Also, we can quantitatively grasp the current status of the project by comparing three indices as shown in Fig. 2.4 described in chapter2. In addition, we can derive the schedule forecast, cost one, productivity, etc. as shown Table 4.1, by using the three indicators, PV, EV and AC, respectively.

From the characteristics of OSS development, many OSS are developed and maintained by several developers with many OSS users. The methods of OSS reliability assessment have been proposed [25, 28, 39, 40]. However, it is difficult to apply EVM directly in terms of the characteristic of the open source project. There is the research papers on deriving EVM measurements by using the development efforts of open source projects [39, 40]. However, the references [39, 40] has the problems that PV and BAC could not be derived due to the characteristics of the data used.

In this thesis, we examine the method of deriving EVM indices and propose a method that can derive all EVM values as an OSS-oriented EVM, considering the deriving method for open source project maintenance effort in chapter 3.

4.1 Effort estimation model based on stochastic differential equation

Generally, the EVM is applied to commonly software development projects. However, it is difficult to directly apply the EVM to the actual open source projects, because the development

Table 4.1: Several examples of the indicators used in EVM.

EVM Elements	Explanation
Planned Value (PV)	PV is a supposed work value at any given point in the project schedule
Earned Value (EV)	EV is a value of work progress at a given point in time
Actual Cost (AC)	AC is an amount of resources that have been expended to date
Budget at Completion (BAC)	BAC represents the total PV for the project
Cost Variance (CV)	CV shows whether a project is under or over budget. CV=EV-AC
Cost Performance Index (CPI)	CPI evaluates how efficiently the project is using its resources. CPI=EV/AC
Schedure Variance (SV)	SV determines whether a project is ahead of or behind schedule. SV=EV-PV
Schedule Performance Index (SPI)	SPI evaluates how efficiently the project team is using its time. SPI=EV/PV
Estimate at Completion (EAC)	EAC shows the final cost of the project in case of continuing current performance trend. EAC=BAC/CPI
Estimate to Complete (ETC)	ETC shows what the remaining work will cost. ETC=(BAC-EV)/CPI

cycle of open source project is different from the traditional software development paradigm. As the characteristics of OSS, the OSS development project is managed by using the bug tracking systems. In order to apply EVM in open source projects, we consider deriving AC, EV, PV and BAC, respectively. The open source projects involve an indefinite number of people from all over the world. Then, the progress of the project becomes the irregular status. Therefore, we consider the irregularities for the effort in open source projects. In this thesis, we discuss the irregularities of the effort in open source projects by using the stochastic model of Wiener process.

Considering the characteristic of the operation phase in OSS projects, the time-dependent expenditure phenomenon of maintenance effort keeps an irregular state in the operation phase, because there is variability among the levels of developers' skill. Then, the time-dependent effort expenditure phenomenon of maintenance phase becomes unstable.

The operation phases of many OSS projects are influenced from the external factors by the triggers such as the difference of skill, time lag of development and maintenance activities. Considering the above points, we apply the stochastic differential equation modeling to managing the open source project. Then, let $\Omega(t)$ be the cumulative maintenance effort, such as finding software faults and improving functionality up to operational time $t\,(t \geq 0)$ in the open source project. Suppose that $\Omega(t)$ takes on continuous real values. Since the estimated maintenance effort are observed during the operational phase of the open source project, $\Omega(t)$ gradually increases as the operational procedures go on. Based on SRGM approach [41–44], the following linear differential equation in terms of the maintenance effort can be formulated:

$$\frac{d\Omega(t)}{dt} = \beta(t)\{\alpha - \Omega(t)\}, \tag{4.1}$$

where $\beta(t)$ is the increase rate of maintenance effort at operational time t and a non-negative function, and α means the estimated maintenance effort expenditures required until the end of operation.

Therefore, we extend Eq. (4.1) to the following stochastic differential equation with Brownian motion [45]:

$$\frac{d\Omega(t)}{dt} = \{\beta(t) + \sigma\nu(t)\}\{\alpha - \Omega(t)\}, \tag{4.2}$$

where σ is a positive constant representing a magnitude of the irregular fluctuation, and $\nu(t)$ a standardized Gaussian white noise. By using Itô's formula [46], we can obtain the solution of

Eq. (4.2) under the initial condition $\Omega(0) = 0$ as follows:

$$\Omega(t) = \alpha \left[1 - \exp\left\{ - \int_0^t \beta(s) ds - \sigma\omega(t) \right\} \right], \tag{4.3}$$

where $\omega(t)$ is one-dimensional Wiener process which is formally defined as an integration of the white noise $\nu(t)$ with respect to time t. Moreover, we define the increase rate of maintenance effort in case of $\beta(t)$ defined as [47]:

$$\int_0^t \beta(s) ds \doteq \frac{\frac{dF_*(t)}{dt}}{\alpha - F_*(t)}. \tag{4.4}$$

In this thesis, we assume the following equations based on software reliability models $F_*(t)$ as the cumulative maintenance effort expenditures function of the proposed model:

$$F_e(t) \equiv \alpha \left(1 - e^{-\beta t} \right), \tag{4.5}$$

$$F_s(t) \equiv \alpha \left\{ 1 - (1 + \beta t) e^{-\beta t} \right\}, \tag{4.6}$$

$$F_i(t) \equiv \frac{\alpha \{1 - \exp(-\beta t)\}}{1 + c \cdot \exp(-\beta t)}, \tag{4.7}$$

where $\Omega_e(t)$ means the cumulative maintenance effort expenditures for the exponential SRGM with $F_e(t)$. Similarly, $\Omega_s(t)$ is the cumulative maintenance effort expenditures for the delayed S-shaped SRGM with $F_s(t)$. Also, $\Omega_i(t)$ means the cumulative maintenance effort expenditures for the inflection S-shaped SRGM with $F_i(t)$, respectively. The reason why this thesis use the three models is that the exponential model, the delayed S-shaped model and the infection S-shaped models are one of the famous software reliability growth models [48].

Therefore, the cumulative maintenance effort, Ω_* up to time t are obtained as follows:

$$\Omega_e(t) = \alpha [1 - \exp\{-\beta t - \sigma\omega(t)\}], \tag{4.8}$$

$$\Omega_s(t) = \alpha [1 - (1 + \beta t) \exp\{-\beta t - \sigma\omega(t)\}], \tag{4.9}$$

$$\Omega_i(t) = \alpha \left[1 - \frac{1 + c}{1 + c \cdot \exp(-\beta t)} \exp\{-\beta t - \sigma\omega(t)\} \right]. \tag{4.10}$$

In these models, we assume that the parameter σ depends on several noises by external factors from several triggers in open source projects. Then, the expected cumulative maintenance effort

expenditures spent up to time t are respectively obtained as follows:

$$\mathrm{E}[\Omega_e(t)] = \alpha\left[1-\exp\left\{-\beta t+\frac{\sigma^2}{2}t\right\}\right], \tag{4.11}$$

$$\mathrm{E}[\Omega_s(t)] = \alpha\left[1-(1+\beta t)\exp\left\{-\beta t+\frac{\sigma^2}{2}t\right\}\right], \tag{4.12}$$

$$\mathrm{E}[\Omega_i(t)] = \alpha\left[1-\frac{1+c}{1+c\cdot\exp(-\beta t)}\exp\left\{-\beta t+\frac{\sigma^2}{2}t\right\}\right]. \tag{4.13}$$

Similarly, we consider the sample path of maintenance effort expenditures required for OSS maintenance, e.g., the needed remaining maintenance effort expenditures from time t to the end of the project, Ω_{r*} are obtained as follows:

$$\Omega_{re}(t) = \alpha\exp\{-\beta t-\sigma\omega(t)\}, \tag{4.14}$$

$$\Omega_{rs}(t) = \alpha(1+\beta t)\exp\{-\beta t-\sigma\omega(t)\}, \tag{4.15}$$

$$\Omega_{ri}(t) = \alpha\frac{1+c}{1+c\cdot\exp(-\beta t)}\exp\{-\beta t-\sigma\omega(t)\}. \tag{4.16}$$

Then, the expected maintenance effort expenditures required for OSS maintenance until the end of operation time t are respectively obtained as follows:

$$\mathrm{E}[\Omega_{re}(t)] = \alpha\exp\left\{-\beta t+\frac{\sigma^2}{2}t\right\}, \tag{4.17}$$

$$\mathrm{E}[\Omega_{rs}(t)] = \alpha(1+\beta t)\exp\left\{-\beta t+\frac{\sigma^2}{2}t\right\}, \tag{4.18}$$

$$\mathrm{E}[\Omega_{ri}(t)] = \alpha\frac{1+c}{1+c\cdot\exp(-\beta t)}\exp\left\{-\beta t+\frac{\sigma^2}{2}t\right\}. \tag{4.19}$$

4.2 Assessment measures for OSS-oriented EVM

In OSS-oriented EVM, the period of data used for Planned Value (PV) and Actual Cost (AC) have the different values. Both PV and AC use the data obtained from the bug tracking systems and required by the fault reporters and the fault fixers. For the prediction of PV, we use Eqs. (4.8)-(4.13) and maintenance effort data up to OSS's release. In particular, the parameter α in Eqs. (4.8)-(4.13) can be regarded as the estimated maintenance effort at the time OSS is released. Therefore, the parameter α can be rephrased as Budget at Completion (BAC) in OSS-oriented EVM. AC uses the maintenance effort data obtained from the bug tracking systems, including

after the OSS release required by the fault reporter and the fault fixer. Therefore, the start time of the data used to derive PV and AC is the same.

Earned Value (EV) is the cumulative maintenance effort viewed on the same scale as the project budget (BAC). Therefore, if the OSS development effort increase but the fault is not fixed, the value of EV becomes small and it is regarded as an inefficient open source project. In the derivation of EV value, the number of potential faults predicted from the fault data reported up to the time of OSS release is used. We use Eqs. (4.8)-(4.13) to predict the number of potential faults. We derive the "fault resolving cost", the value obtained by dividing the number of potential faults from the BAC, as follows:

$$\gamma = \frac{BAC}{p}. \tag{4.20}$$

Then, γ means fault resolving cost, and p means potential faults at OSS release. We can derive the EV in case of $F_e(t)$, $F_s(t)$ and $F_i(t)$ by using the fault resolving cost γ and the cumulative number of fixed faults up to the operating time t.

$$EV_e(t) = \gamma\left[\alpha_f\left[1-\exp\left\{-\beta_f t-\sigma_f\omega(t)\right\}\right]\right], \tag{4.21}$$

$$EV_s(t) = \gamma\left[\alpha_f\left[1-(1+\beta_f t)\exp\left\{-\beta_f t-\sigma_f\omega(t)\right\}\right]\right], \tag{4.22}$$

$$EV_i(t) = \gamma\left[\alpha_f\left[1-\frac{1+c_f}{1+c_f\cdot\exp\left(-\beta_f t\right)}\exp\left\{-\beta_f t-\sigma_f\omega(t)\right\}\right]\right]. \tag{4.23}$$

Then, α_f, β_f, c_f, and σ_f are parameters used to predict the cumulative number of fixed faults at time t. Therefore, the expected EV required for OSS maintenance until the end of operation time t are respectively obtained as follows:

$$\mathrm{E}\left[EV_e(t)\right] = \gamma\left[\alpha_f\left[1-\exp\left\{-\beta_f t+\frac{\sigma_f^2}{2}t\right\}\right]\right], \tag{4.24}$$

$$\mathrm{E}\left[EV_s(t)\right] = \gamma\left[\alpha_f\left[1-(1+\beta_f t)\exp\left\{-\beta_f t+\frac{\sigma_f^2}{2}t\right\}\right]\right], \tag{4.25}$$

$$\mathrm{E}\left[EV_i(t)\right] = \gamma\left[\alpha\left[1-\frac{1+c_f}{1+c_f\cdot\exp\left(-\beta_f t\right)}\exp\left\{-\beta_f t+\frac{\sigma_f^2}{2}t\right\}\right]\right]. \tag{4.26}$$

Then, the fixed cumulative number of faults is counted when the fault status is "Closed" in the bug tracking systems.

4.3 How to derive OSS-oriented EVM value

Generally, the EVM is applied to commonly software development projects. However, it is difficult to directly apply the EVM to the actual open source project, because the development cycle of open source project is different from the traditional software development paradigm. As the characteristics of OSS, the OSS development project is managed by using the bug tracking systems. This chapter shows the method of earned value analysis for OSS projects by using the data sets obtained from bug tracking systems.

Considering the deriving AC, PV and EV proposed in this chapter, we assume the following terms shown Table 4.2 as the OSS-oriented EVM for OSS development:

Then, the expected Cost Variance (CV) for OSS maintenance up to operational time t in case of $\Omega_e(t)$ and $\Omega_s(t)$ can be formulated as:

$$\mathrm{E}[CV_e(t)] = \mathrm{E}[EV_e(t)] - \mathrm{E}[\Omega^a_{re}(t)], \tag{4.27}$$

$$\mathrm{E}[CV_s(t)] = \mathrm{E}[EV_s(t)] - \mathrm{E}[\Omega^a_{rs}(t)], \tag{4.28}$$

$$\mathrm{E}[CV_i(t)] = \mathrm{E}[EV_i(t)] - \mathrm{E}[\Omega^a_{ri}(t)], \tag{4.29}$$

where $\mathrm{E}[EV_e(t)]$, $\mathrm{E}[EV_s(t)]$ and $\mathrm{E}[EV_i(t)]$ are the expected maintenance effort expenditures considering EV until the end of operation. Also, $\mathrm{E}[\Omega^a_{re}(t)]$, $\mathrm{E}[\Omega^a_{rs}(t)]$ and $\mathrm{E}[\Omega^a_{ri}(t)]$ mean the maintenance effort expenditures considering AC until the end of operation. Especially, a in case of $\mathrm{E}[\Omega^a_{re}(t)]$, $\mathrm{E}[\Omega^a_{rs}(t)]$ and $\mathrm{E}[\Omega^a_{ri}(t)]$ comes from 'actual'. Similarly, the sample path of CV for open source project maintenance up to operational time t in case of $\Omega_e(t)$, $\Omega_s(t)$ and $\Omega_i(t)$ are given by

$$CV_e(t) = EV_e(t) - \Omega^a_{re}(t), \tag{4.30}$$

$$CV_s(t) = EV_s(t) - \Omega^a_{rs}(t), \tag{4.31}$$

$$CV_i(t) = EV_i(t) - \Omega^a_{ri}(t). \tag{4.32}$$

The zero point of $\mathrm{E}[CV_e(t)]$, $\mathrm{E}[CV_s(t)]$ and $\mathrm{E}[CV_i(t)]$ mean the starting point of surplus effort. Therefore, the open source project managers will be able to judge the necessity of maintenance effort and stability of OSS from the starting point of surplus effort. Moreover, we can obtain the

Table 4.2: Explanation for OSS-oriented EVM.

OSS-oriented EVM elements	Explanatory
Planned Value (PV)	Cumulative maintenance effort as planned value up to operational time t considering the fault reporter and fault fixer
Earned Value (EV)	Cumulative maintenance effort up to operational time t viewed on the same scale as BAC
Actual Cost (AC)	Cumulative maintenance effort up to operational time t considering the fault reporter and fault fixer
Budget at Completion (BAC)	Total budget in the end point as the specified goal of open source project
Cost Variance (CV)	$\mathrm{E}[CV_e(t)]$ and $\mathrm{E}[CV_s(t)]$ obtained from EV-AC
Cost Performance Index (CPI)	$\mathrm{E}[CPI_e(t)]$ and $\mathrm{E}[CPI_s(t)]$ obtained from EV/AC
Schedure Variance (SV)	SV obtained from EV-PV
Schedule Performance Index (SPI)	SPI obtained from EV/PV
Estimate at Completion (EAC)	$\mathrm{E}[EAC_e(t)]$ and $\mathrm{E}[EAC_s(t)]$ obtained from BAC/CPI
Estimate to Complete (ETC)	$\mathrm{E}[ETC_e(t)]$ and $\mathrm{E}[ETC_s(t)]$ obtained from (BAC-EV)/CPI

Cost Performance Index (CPI) by using the following equations:

$$\mathrm{E}[CPI_e(t)] = \frac{\mathrm{E}[EV_e(t)]}{\mathrm{E}[\Omega^a_{re}(t)]}, \tag{4.33}$$

$$\mathrm{E}[CPI_s(t)] = \frac{\mathrm{E}[EV_s(t)]}{\mathrm{E}[\Omega^a_{rs}(t)]}, \tag{4.34}$$

$$\mathrm{E}[CPI_i(t)] = \frac{\mathrm{E}[EV_i(t)]}{\mathrm{E}[\Omega^a_{ri}(t)]}. \tag{4.35}$$

The value of CPI is derived by $\frac{EV}{AC}$. Similarly, the sample path of CPI in case of Ω_e, Ω_s and Ω_i are given by

$$CPI_e(t) = \frac{EV_e(t)}{\Omega^a_{re}(t)}, \tag{4.36}$$

$$CPI_s(t) = \frac{EV_s(t)}{\Omega^a_{rs}(t)}, \tag{4.37}$$

$$CPI_i(t) = \frac{EV_i(t)}{\Omega^a_{ri}(t)}. \tag{4.38}$$

Furthermore, we can obtain the Estimate at Completion (EAC) by using the following equations:

$$\mathrm{E}[EAC_e(t)] = \mathrm{E}[\Omega^a_{re}(t)] + \frac{(BAC - \mathrm{E}[EV_e(t)])}{\mathrm{E}[CPI_e(t)]}, \tag{4.39}$$

$$\mathrm{E}[EAC_s(t)] = \mathrm{E}[\Omega^a_{rs}(t)] + \frac{(BAC - \mathrm{E}[EV_s(t)])}{\mathrm{E}[CPI_s(t)]}, \tag{4.40}$$

$$\mathrm{E}[EAC_i(t)] = \mathrm{E}[\Omega^a_{ri}(t)] + \frac{(BAC - \mathrm{E}[EV_i(t)])}{\mathrm{E}[CPI_i(t)]}. \tag{4.41}$$

The value of EAC is derived by AC+ETC. Similarly, the sample path of EAC in case of Ω_e, Ω_s and Ω_i are given by

$$EAC_e(t) = \Omega^a_{re}(t) + \frac{(BAC - EV_e(t))}{CPI_e(t)}, \tag{4.42}$$

$$EAC_s(t) = \Omega^a_{rs}(t) + \frac{(BAC - EV_s(t))}{CPI_s(t)}, \tag{4.43}$$

$$EAC_i(t) = \Omega^a_{ri}(t) + \frac{(BAC - EV_i(t))}{CPI_i(t)}. \tag{4.44}$$

Finally, we can obtain the Estimate to Complete (ETC) by using the following equations:

$$\mathrm{E}[ETC_e(t)] = \frac{(BAC - \mathrm{E}[EV_e(t)])}{\mathrm{E}[CPI_e(t)]}, \tag{4.45}$$

$$\mathrm{E}[ETC_s(t)] = \frac{(BAC - \mathrm{E}[EV_s(t)])}{\mathrm{E}[CPI_s(t)]}, \tag{4.46}$$

$$\mathrm{E}[ETC_i(t)] = \frac{(BAC - \mathrm{E}[EV_i(t)])}{\mathrm{E}[CPI_i(t)]}. \tag{4.47}$$

The value of ETC is derived by $\frac{BAC-EV}{CPI}$. Similarly, the sample path of ETC in case of Ω_e, Ω_s and Ω_i are given by

$$ETC_e(t) = \frac{(BAC - EV_e(t))}{CPI_e(t)}, \tag{4.48}$$

$$ETC_s(t) = \frac{(BAC - EV_s(t))}{CPI_s(t)}, \tag{4.49}$$

$$ETC_i(t) = \frac{(BAC - EV_i(t))}{CPI_i(t)}. \tag{4.50}$$

In particular, Eqs. (4.33) - (4.50) are based on the EVM derivation method [17]. Also, the CPI is very important for open source project managers to assess the stability of open source project.

4.4 Numerical examples

In this thesis, we use the data set of open source project for deriving EVM indices. For applying the proposed model to actual project data set, we use the data of LibreOffice [37] obtained from Bugzilla. LibreOffice is an office suite OSS provided by The Document Foundation. In particular, the effort and fault data have been obtained from Bugzilla are version 7.2 for estimating PV and AC. In this thesis, the cumulative number of reported faults are 298 and 878, respectively. In particular, we use the project data for about 39 weeks, before LibreOffice was released for estimating PV. For estimating AC, we also use project data for about 112 weeks. Also, each unit data is weekly.

4.4.1 Estimation of EVM Indices

In this chapter, we estimate the model parameters of the three SRGM models for estimating the maintenance effort and the number of faults in case of LibreOffice version 7.2 project.

Table 4.3 shows the results of parameter estimation of maintenance effort, and AIC (Akaike's Information Criterion) [38] for comparison of model equations. Also, the parameter α in the PV data can be rephrased as BAC. In terms of AIC, the delayed S-shaped model is the best one for PV estimation. Fig. 8 shows the results of applying the delayed S-shaped model to the open source project data.

Table 4.3: Parameter estimation of maintenance effort in terms of PV.

		Planned Value		
		exponential model	delayed S-shaped model	infection S-shaped model
parameter	α	2.867×10^{5}	9.283×10^{5}	1.859×10^{6}
	β	3.327×10^{-2}	2.293×10^{-2}	3.352×10^{-2}
	c	-	-	1.984×10^{1}
	σ	2.361×10^{-2}	7.504×10^{-4}	7.577×10^{-4}
AIC		798.191	**647.075**	709.147

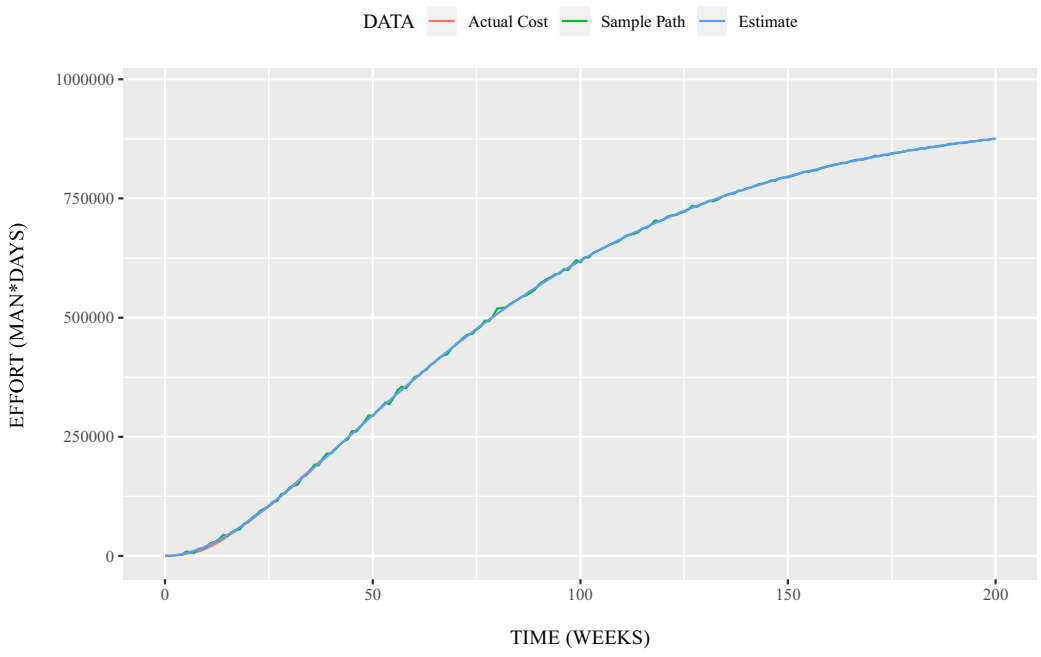

Fig. 4.1: The cumulative maintenance effort expenditures as PV in LibreOffice Ver. 7.2 project by using Eqs. (4.9) and (4.12).

Table 4.4: Parameter estimation of maintenance effort in terms of AC.

		Actual Cost		
		exponential model	delayed S-shaped model	infection S-shaped model
parameter	α	1.325×10^{6}	1.317×10^{6}	2.728×10^{6}
	β	9.286×10^{-3}	2.387×10^{-2}	1.253×10^{-2}
	c	-	-	3.283×10^{1}
	σ	4.840×10^{-3}	2.762×10^{-3}	1.011×10^{-3}
AIC		1229.007	**1163.817**	1205.231

Next, Table 4.4 shows the results of parameter estimation of AC and AIC. Also, the parameter α can be rephrased as the project's estimated AC. In terms of AIC, the delayed S-shaped model is the best one for AC estimation. Fig. 8.14 shows the results of applying the delayed S-shaped model to the open source project data.

Also, Table 4.5 shows the results of parameter estimation of the estimated number of potential faults at OSS release, and AIC. We use the parameter α for deriving fault resolving cost. There is no significant difference in AIC values among all the model equations used in this thesis. Therefore, it is difficult for us to identify a suitable model for the data used in this thesis. For convenience, we assume that the exponential model with the smallest AIC is the appropriate model. Fig. 8.15 shows the results of applying the exponential model to the open source project data.

Finally, Table 4.6 shows the results of parameter estimation of the estimated number of resolved faults at present, and AIC. In terms of AIC, the infection S-shaped model is the best method for the number of resolved faults estimation. Fig. 8.16 shows the results of applying the delayed S-shaped model to the open source project data.

A comparison of the AIC values during parameter estimation in the three model equations showed that the delayed S-shaped model and the infection S-shaped model are appropriate. In the LibreOffice version 7.2 project data, the increase rate of maintenance effort and number of faults at the start of the maintenance phase is small. We find that the delayed S-shaped model and infection S-shaped model are appropriate for such project data.

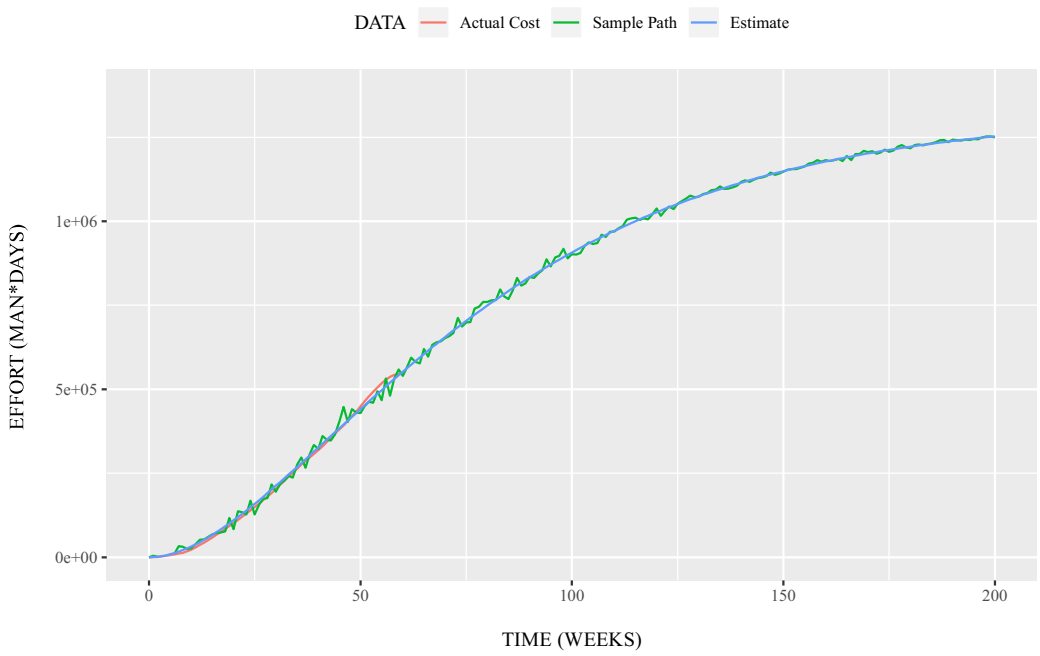

Fig. 4.2: The cumulative maintenance effort expenditures as AC in LibreOffice Ver. 7.2 project by using Eqs. (4.9) and (4.12).

Table 4.5: Parameter estimation of number of potential faults in case of LibreOffice.

		Estimated number of potential faults at OSS release		
		exponential model	delayed S-shaped model	infection S-shaped model
parameter	α	1.401×10^{3}	5.291×10^{2}	4.792×10^{2}
	β	6.274×10^{-3}	4.818×10^{-2}	4.718×10^{-2}
	c	-	-	2.083×10^{1}
	σ	3.165×10^{-3}	1.113×10^{-2}	1.535×10^{-2}
AIC		**238.479**	**238.613**	**238.961**

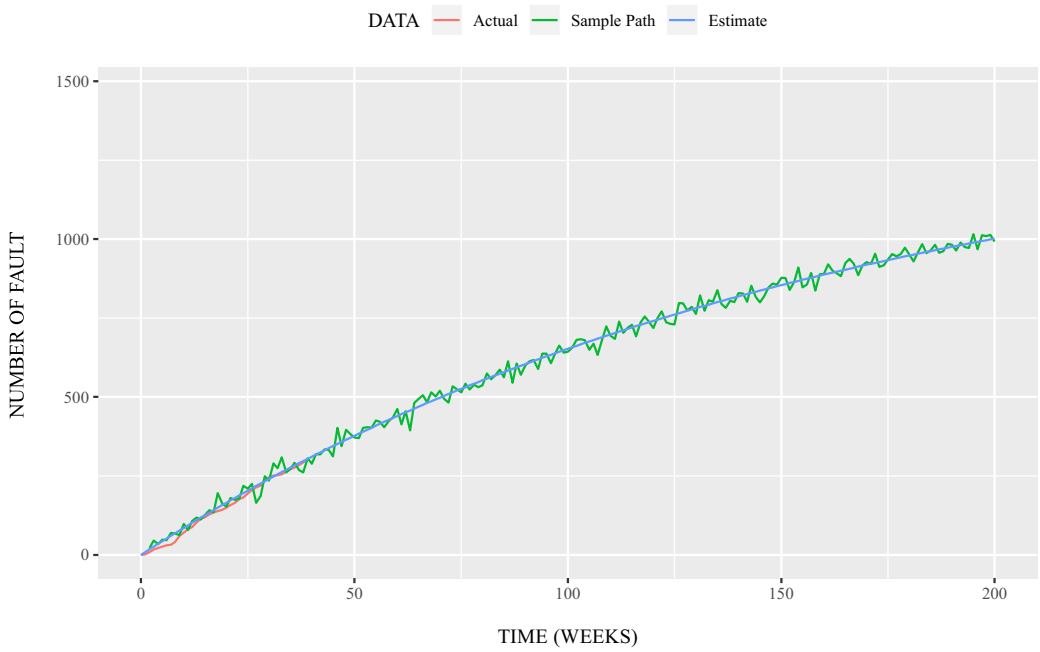

Fig. 4.3: The cumulative estimated number of potential faults in LibreOffice Ver. 7.2 project by using Eqs. (4.8) and (4.11).

Table 4.6: Parameter estimation of number of resolved faults in case of LibreOffice.

		Estimated number of resolved faults at present		
		exponential model	delayed S-shaped model	infection S-shaped model
parameter	α	1.019×10^{4}	7.887×10^{4}	6.054×10^{3}
	β	1.501×10^{-3}	2.767×10^{-3}	7.143×10^{-2}
	c	-	-	4.350×10^{2}
	σ	1.697×10^{-3}	1.395×10^{-4}	1.595×10^{-3}
AIC		534.055	488.803	**460.932**

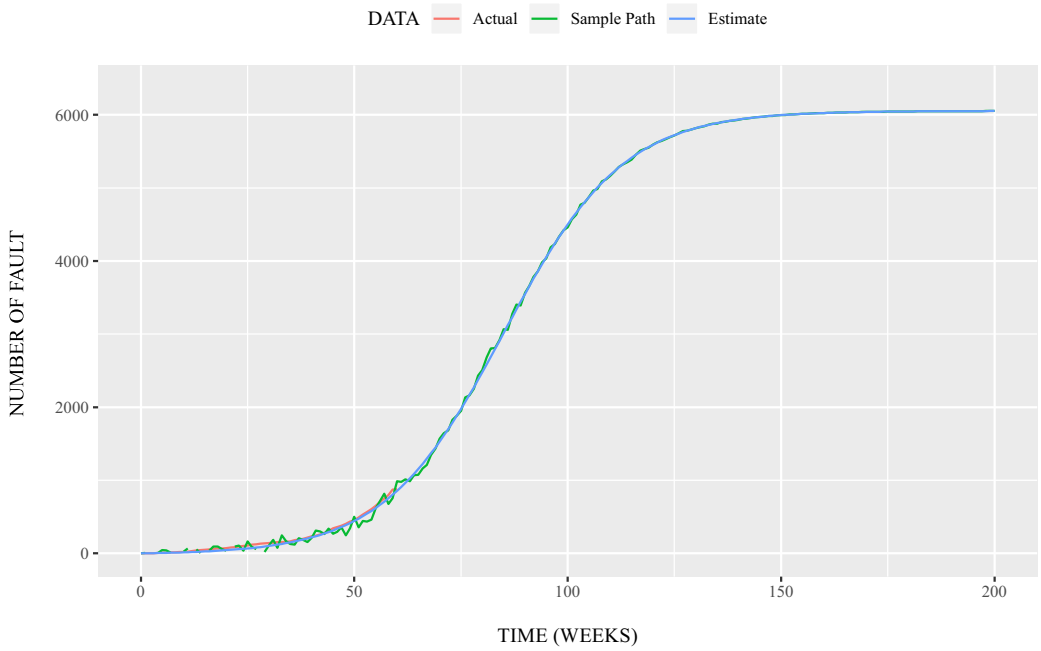

Fig. 4.4: The cumulative estimated number of resolved faults in LibreOffice Ver. 7.2 project by using Eqs. (4.10) and (4.13).

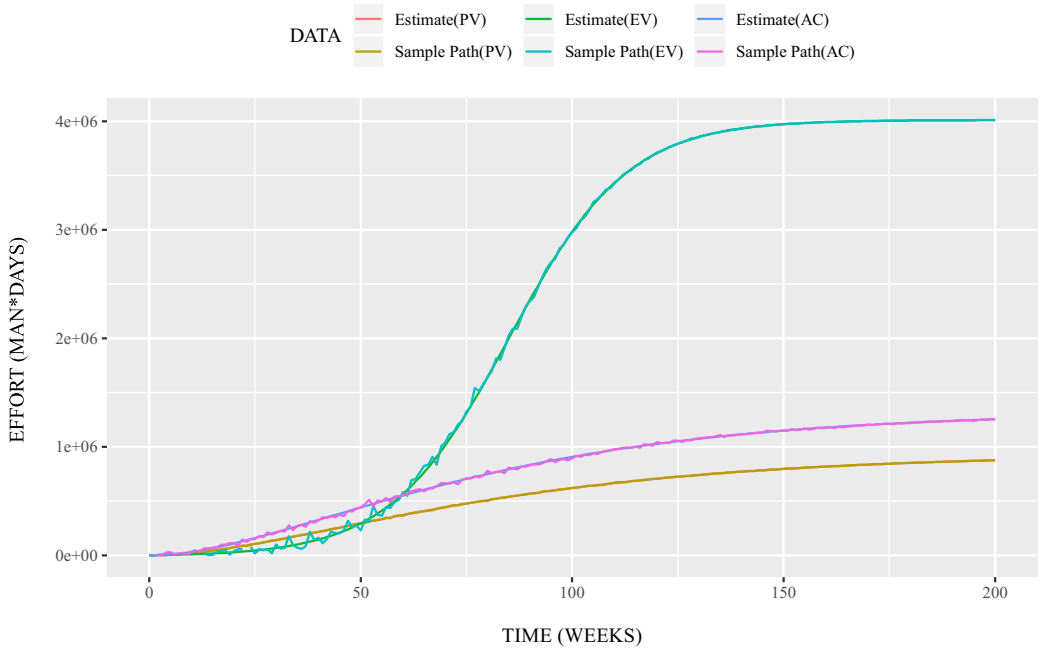

Fig. 4.5: EVM estimation results in LibreOffice project.

In the open source projects, the number of fault reports increases as the number of OSS users increases after the release of a particular version. As a result, the effort required for fault maintenance increases. Therefore, the appropriate model equation for many open source project data would be the same as in this thesis.

In this thesis, we derive EVM indices by using the best-fit model equation for each data set. The fault resolving cost $\gamma = 662.419(man \cdot days)$, one of the EVM indices, is necessary for the derivation of EV. Fig. 4.5 shows the results of EV, AC, and PV estimations.

Fig. 4.5 shows that both EV and AC are larger than PV. In particular, the EV value is very large. This is because the number of resolved faults is estimated to be higher than the number of potential faults. On the other hand, the EV value is lower than the EV and AC values around 50 weeks, the time of the version 7.2 release, showing the project is in a delayed state. In other words, after version 7.2 was released, we found that the project became more active as the number of users of that version increased.

4.4.2 Output result of bullseye chart

We used the EVM indices derived in Fig. 4.5 to draw the Fig. 4.6 bullseye chart.

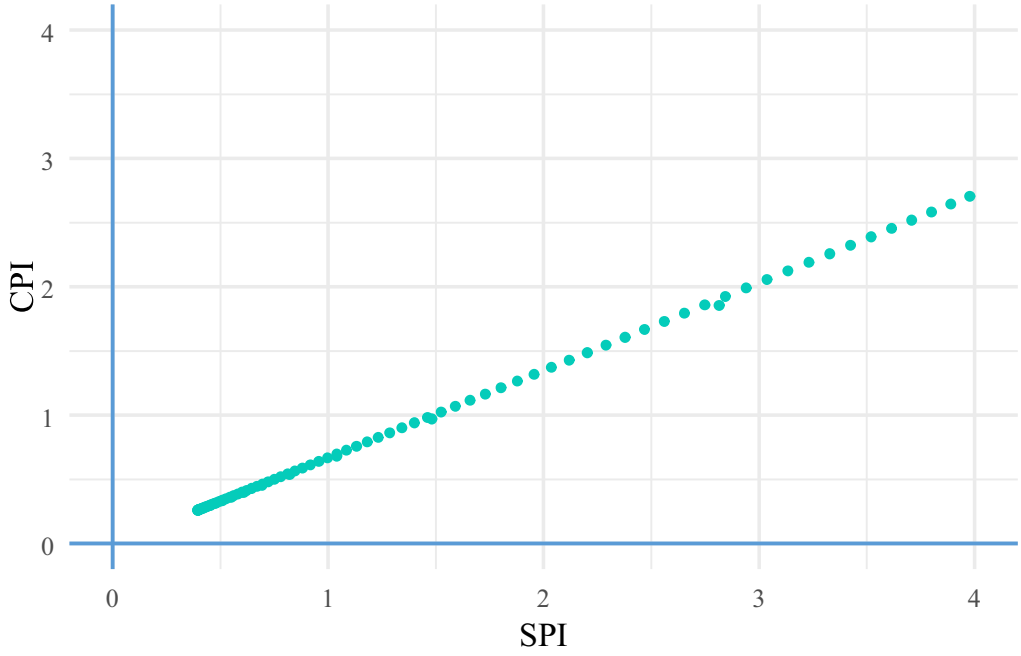

Fig. 4.6: The bullseye chart in LibreOffice project.

The bullseye chart was initially delayed in terms of schedule and cost because both the SPI and CPI were less than 1. However, as the value of EV gradually increased, both values also increased, and they will be above 1 in the future. Therefore, the bullseye chart shows that the LibreOffice project will be stable in terms of schedule and cost in the future.

4.5 Conclusion in this chapter

In this thesis, we have examined the method for evaluating project stability based on SRGM's in open source projects. In terms of AIC, we have identified the appropriateness models in the LibreOffice project. Then, we have found that the delayed S-shaped model and the infection S-shaped model are the best fitted models because of the increase in the number of users of the version over time. We have considered that the results are the same as other open source project,

because of the characteristic of the number of OSS users' transitions. Also, we have derived the OSS-oriented EVM and bullseye chart by using the appropriate SRGM models. As the result, we have found that the trigger for activating open source projects is after the release of a particular version.

Researches on stability evaluation methods for open source projects have often focused on the resolving of individual faults. Therefore, the practical application of EVM for evaluating the stability of open source projects as a whole will contribute to the future development of OSS. On the other hand, since the proposed method evaluates the stability based on the cost of the entire open source projects, it is difficult to evaluate the causes of project stability in fault units. Therefore, we consider that the use of not only the proposed method but also individual fault-based project evaluation methods will provide a better project stability evaluation tool.

As only one open source project data set has been used in this thesis, it is necessary to verify the characteristics of the trends in maintenance effort and number of faults by using multiple project data sets in the future.

Figures not shown in this chapter are shown in Appendix.

Related Paper in this Chapter

1. 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.

2. 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, May 2022.

3. H. Sone, Y. Tamura, and S. Yamada, "Statistical maintenance time estimation based on stochastic differential equation models in OSS development project," Computer Reviews Journal, PURKH, Vol. 5, pp. 126-140, December 2019.

4. H. Sone, Y. Tamura, and S. Yamada, "Optimal maintenance problem with earned value requirement for OSS project," Proceedings of the Fourteenth International Conference on

Industrial Management, Hangzhou, China, September 12-14, 2018, pp. 253-258.

5. H. Sone, Y. Tamura, and S. Yamada, “Stochastic differential equation modeling for development effort estimation and its application in open source project,” Proceedings of 32th National Conference of the Society of Project Management, Doshisha University, August 30-31, 2018, pp. 439-443 (in Japanese).

Chapter 5

Fault Fixing Time Transition Prediction

Although OSS programs have been actively developed and used in recent years, there are problems with promoting open source projects. Massive open source projects have many fault reports: there are over 100 faults reported per day in massive open source projects [30]. Due to this, it becomes difficult to fix the faults quickly [49, 50].

From the perspective of the fault fixing speed of OSS, OSS users usually use the OSS under the conditions of stable development. There are many studies in terms of the stability degree assessment of open source projects [25, 27, 51–56]. There are many researches that predict the development effort in open source projects [25, 27, 51, 52] and studies that predict the fix time of each reported fault [53–55]. Furthermore, the previous research showed that the large open source projects tend to have a shorter fixing time as the project progresses [56].

Generally, as the development of an OSS progresses, the number of unfixed faults and faults with a long fixing time decreases. Then, the OSS becomes stable. Several previous studies predicted the fault fixing time. However, these research works could not predict the software's stability, because the previous research did not consider the fixing time for all faults in software development.

Therefore, in this thesis, we evaluate the state of open source projects by predicting the fix time transition of many faults that occur during the development of OSS. The steps are as follows.

1. Examine the amount of data that can be learned.

2. Consider the model equations using the SRGMs.

3. Apply the model equation to the measured data and check the prediction results of the fixing time transition.

The main contributions of the proposed method are as follows:

- By using the proposed method, the OSS manager can assess the stability of the open source project considering the external factors of open source projects.

- Considering the Wiener process, the OSS manager can treat many noisy cases during the fault fixing time.

- The proposed method will lead to the assessment of the stability of OSS systems considering the convergence of the fault fixing time in various open source projects, because the proposed method can rapidly assess the stability and reliability of future projects by using fault fixing time data.

5.1 Confirmation of the amount of training data

In predicting the fault fixing time transition from open source project data, we examine how much fault data can be learned. In particular, the data used in this thesis is Red Hat Enterprise Linux (RHEL) [36] version 8.0. RHEL is a commercial open-source Linux distribution developed by Red Hat, and the data used in this thesis is obtained from Bugzilla [24]. The number of faults used in this thesis is 5068.

Considering the number of data that can be trained, since some of the most recently reported fault data have not yet been closed or have been repeatedly closed and reopened, it is ideal to treat completely closed fault data as training data. The average time when a fault becomes CLOSE is shown in Fig. 5.1. Fig. 5.1 shows that there are many faults with a long fixing time, although the variation is large, up to about 100 weeks, and it is highly probable that the status repeats between CLOSE and REOPEN. In other words, many faults are considered to require numerous effort to fix as a result. On the other hand, since the fixing time gradually decreases after 200 weeks, there may be several faults that will become REOPEN faults in the future, even if their status is CLOSE. Therefore, in this thesis, we consider the possibility that the most recent fault will be REOPEN, and use the training data up to the time when the fix is complete and there is little possibility of REOPEN (190 weeks).

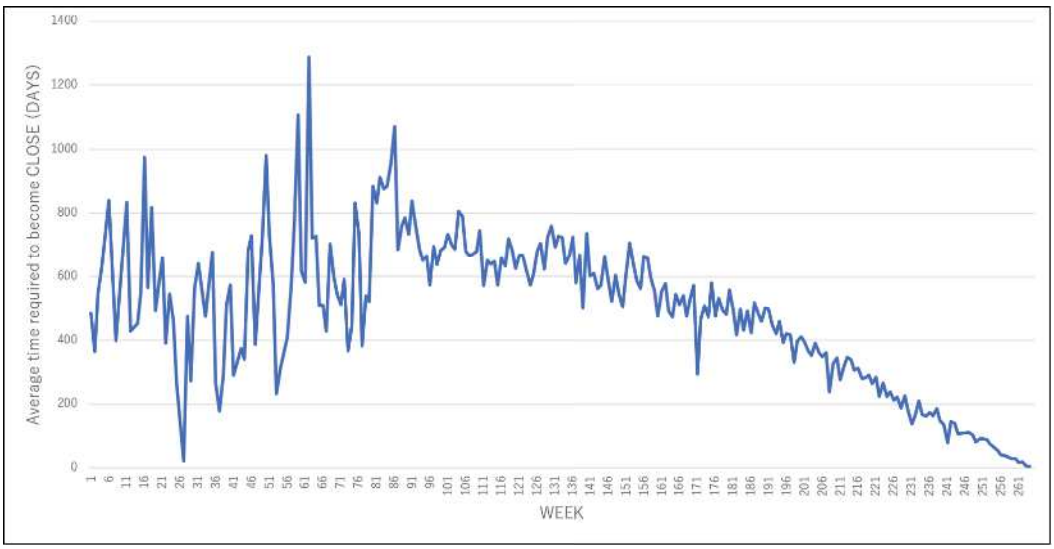

Fig. 5.1: Average time required to become CLOSE.

5.2 Method for predicting the fixing time transition

In this thesis, we predict that the time required for fault fixing in OSS development decreases and converges with the transition of the fix time. By predicting the transition of the fixing time, we can obtain one indicator to know when the project becomes stable. Specifically, we focus on the OSS development using a bug tracking system in the operational phase, as shown in Fig. 5.2. In this thesis, we aim to predict the transition of the fix time using the exponential model, the delayed S-shaped model and the infection S-shaped model [15] derived from the SRGMs [41–44]. In particular, we predict the transition of the fixing time considering the characteristics of open source projects. Then, we assume that the number of developers and users changes irregularly. We can easily discover the irregularity from various factors in open source projects and apply mathematical models with multiple parameters. However, it is difficult to use these models actually in terms of parameter estimation. In this thesis, we apply a stochastic differential equation model with noise based on the Wiener process considering the specific circumstances of open source projects. The proposed model will be able to evaluate the project quantitatively considering external factors indirectly in open source projects.

In particular, the equations used in this thesis are based on Eqs. (4.8)∼(4.13) of Chapter 4.1.

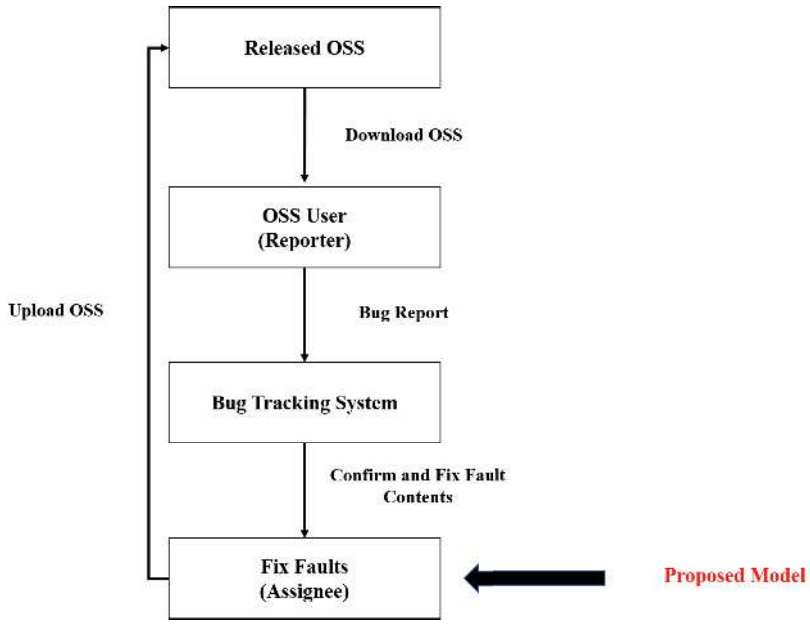

Fig. 5.2: Open source software (OSS) development using the bug-tracking system.

Also, the meaning of some parameters changes in this chapter. $\beta(t)$ is the increase rate of the fault fixing time at operational time t and a non-negative function and α means the estimated fault fixing time required until the end of the operation.

In this thesis, the cumulative fixing time $\Omega_*(t)$ is differentiated in order to understand the fixing time transition.

$$\frac{d\Omega_e(t)}{dt} = \alpha\{\beta(t)+\sigma\nu(t)\}\exp\{-\beta t-\sigma\omega(t)\}, \tag{5.1}$$

$$\frac{d\Omega_s(t)}{dt} = \alpha(1+\beta t)\{\frac{\beta^2 t}{1+\beta t}+\sigma\nu(t)\}\exp\{-\beta t-\sigma\omega(t)\}, \tag{5.2}$$

$$\frac{d\Omega_i(t)}{dt} = \alpha\frac{(1+c)[\{-\beta t-\sigma\omega(t)\}\{1+c\exp(\beta t)\}+\beta c\exp(-\beta t)]}{\{1+c\exp(-\beta t)\}^2} \cdot \exp\{-\beta t-\sigma\omega(t)\}. \tag{5.3}$$

5.3 Application of proposed method to actual data

Table 5.1 shows the parameters of the RHEL fault data used in this thesis when applied to the (5.1)∼(5.3). Using the maximum likelihood estimation method for parameter estimation, we find that there is no significant difference among any of the models in terms of AIC [38]. Therefore,

Table 5.1: Parameter estimation results in RHEL 8.0.

		exponential model	delayed S-shaped model	infection S-shaped model
parameter	α	1.102×10^5	1.044×10^5	1.037×10^5
	β	1.184×10^{-2}	2.494×10^{-2}	2.351×10^{-2}
	c	-	-	3.283
	σ	4.826×10^{-3}	6.394×10^{-3}	6.420×10^{-3}
AIC		2515.564	2514.653	2512.884

Fig. 5.3 shows the results of the prediction of the fixing time transition by the exponential model. The noise in the exponential model appears larger than the actual. Therefore, we can see that the magnitude of the noise decreases with time elapse, similar to the change of the noise over time in the actual fixing time transition.

Fig. 5.4 shows only the expected and actual values of Fig. 5.3. The predictions in the exponential model predicted a longer time than the actual revision time trend. Therefore, there is a possibility that the most recently fixed fault will be REOPEN and the fixing time will be longer. In other words, it is necessary to consider the training data period when predicting the trends of fault fixing time.

5.4 Conclusion in this chapter

In general, the prediction of development effort and fixing time for individual faults can be assessed by using the conventional OSS reliability evaluation methods. However, there is no research in terms of the transition of the fault fixing time for a long time, i.e., there is no research for predicting the fault fixing time. It is difficult to use the conventional SRGMs for the fault fixing time, because the conventional SRGMs mainly evaluate the number of faults in software development. We can easily control open source projects if we can rapidly assess the stability and reliability of future projects by using fault fixing time data. Thereby, the proposed method will lead to assessing the stability of OSS systems developed in terms of the convergence of the fault fixing time under various open source projects. Furthermore, the appropriate control of manage-

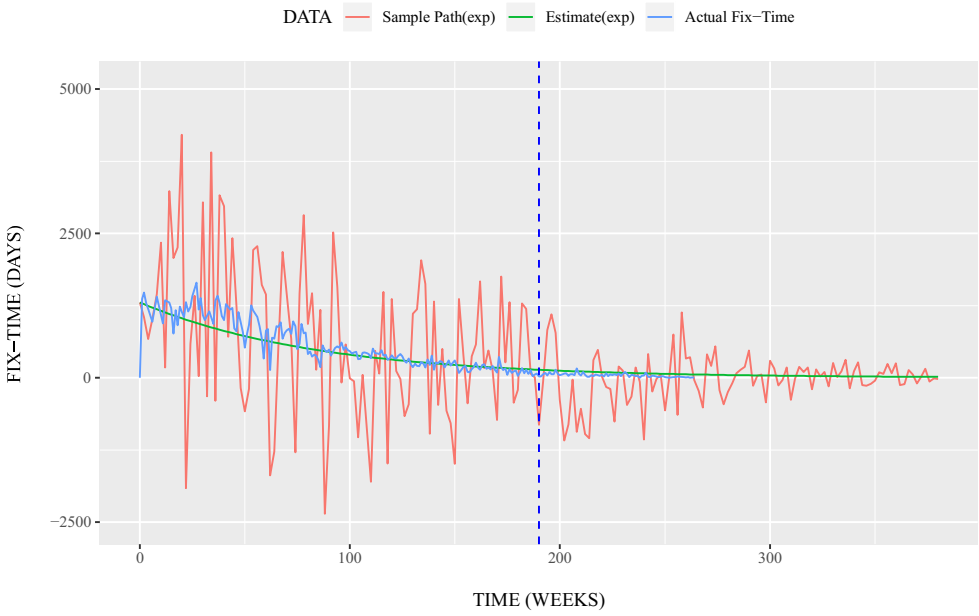

Fig. 5.3: Prediction result of fault fixing time transition in the exponential model by using Eq. (5.1).

ment effort for OSS will indirectly link to the quality, safety, reliability, and cost reduction of OSS if the manager knows the future of the project's progress.

In this thesis, we discussed the transition analysis method of fault fixing time based on a stochastic differential equation model in an open source project. In particular, considering the characteristics of changes in the fault fixing time in large-scale open source projects and the complexity in OSS development, we predicted the transition of fault fixing time based on the Wiener process. Furthermore, we considered the possibility that the fault status could be REOPEN and examined the number of training data. As a result, there is a possibility that the last fault fixing time can be considered as the final fault fixing time by excluding the most recent fault data. The proposed method can help for the project managers and OSS users as an evaluation method of open source project progress in the operation phase. In the future, we need to consider the appropriate models for each project by applying them to the other open source project.

Figures not shown in this chapter are shown in Appendix.

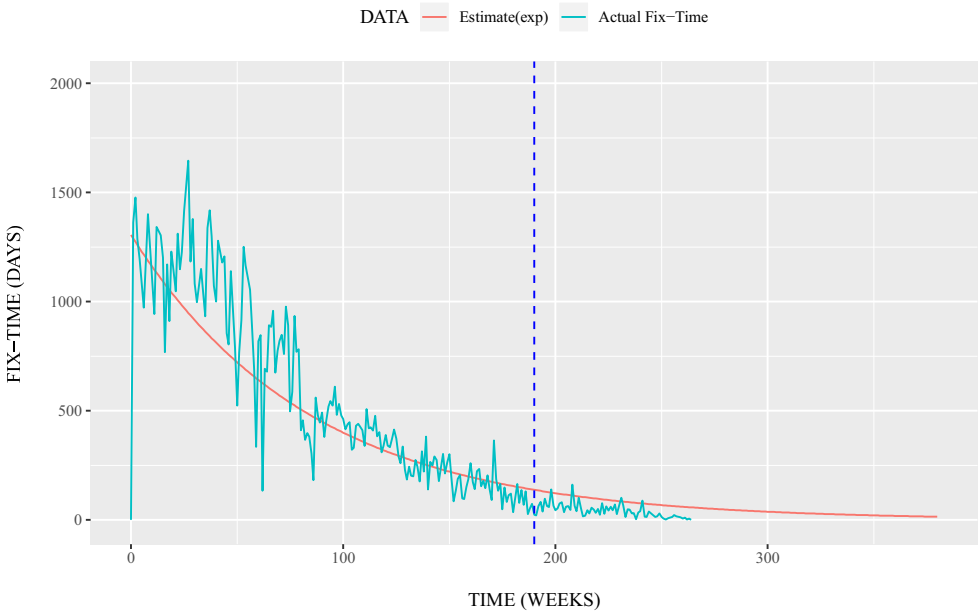

Fig. 5.4: Comparison of the measured and expected fault fixing time transition in the exponential model.

Related Paper in this Chapter

1. H. Sone, Y. Tamura, and S. Yamada, "Prediction of fault fixing time transition and practical feasibility study for open source projects," Kyoto University Reserach Institute for Mathematical Sciences Kôkyûroku "Mathematical Decision Making Under Uncertainty and Related Topics," No. 2242, pp. 140-145, January 2023 (in Japanese).

2. H. Sone, Y. Tamura, and S. Yamada, "Prediction of fault fix time transition in large-scale open source project data," Data, Multidisciplinary Digital Publishing Institute, Switzerland, Vol. 4, No. 3, Multidisciplinary Digital Publishing Institute, Switzerland, DOI: 10.3390/data4030109, pp. 1-12, July, 2019.

Chapter 6

Fault Severity Prediction

In open source projects, many fault information such as fix status, their details, and fix priorities are registered through the bug tracking systems. Although OSS has been actively developed and used in recent years, there are problems in promoting open source projects. There are over 100 faults reported per a day in massive open source projects [57] Massive open source projects have many fault reports. Then, it is difficult to quickly fix the faults [49, 50]. From the above problems, several researchers have been published papers to predict the fault fixing time for open source projects. For example, there are several papers in terms of the prediction whether the fault fixing time will end within a specific time [55, 58, 59]. However, there are no research papers in terms of the fault identification method considering the change of fault fixing environment in the open source project, e.g., the fault fixing priority, the number of fault occurrences, and the number of faults fixing assignees. In this thesis, we discuss the prediction methods of faults that should be fixed preferentially in newly reported faults in open source projects by using the random forest and available data obtained from the bug tracking system. In addition, we discuss the improvement of prediction accuracy and use logistic regression analysis for the comparison. This thesis also shows several numerical examples by using the real fault big data obtained from the bug tracking system.

6.1 Related research

In massive open source projects, there is a problem that the fault fixing time is prolonging. It is necessary to prevent the fault fixing time from prolonging [60, 61], because the OSS is socially important software. There are several previous researches focused on the fault fixing and reporting times by using the bug tracking system. In the software development using a bug tracking system, it is necessary to consider the priority and severity of fault in order to report the fault.

However, there is research reports that the high priority faults do not have a significant effect on fault fixing time, while several high severity faults tend to contribute to shortening fault fixing time [50, 56]. In this thesis, we focus on a new index called *fix priority* by considering the features in terms of the priority and importance of the fault. We also evaluate newly reported faults associated with the previously fixed faults with *fix priority*.

6.2 Fault identification method considering high fix priority

The purpose of this thesis is to identify the serious fault. This serious fault is more serious than newly reported fault with fix priority. Therefore, the fault data for identification is used as the test data. Then, the learning data is the fault data used for comparison. The process of identification of *fix priority* is follows:

1. We make an evaluation index by using the fault fixing time and the fault severity obtained from the bug tracking system.

2. We make label "High" or "Low" to the evaluation index in terms of the value of evaluation index.

3. We use the random forest to predict High and Low in test data. The using data for prediction is obtained from the bug tracking systems.

6.2.1 Evaluation index

We propose the measure as fix priority using the fault fixing time and the severity of learning and test data as follows:

$$T_i = t_{standard_i} \times severity_i, \tag{6.1}$$

where i is the fault number, and $severity$ is the fault severity. Also, the i-th fault data in Eq. (6.1) is the test data, and the others are learning data. In the other words, the *fix priority* of one data is predicted using i-1 data. In particular, $severity$ is registered when a fault is registered in the bug tracking system, and the range of possible values is 1 to 5. Table 6.1 shows the values of the

Table 6.1: The scored fault severity.

Severity	Score
Blocker	5
Critical	4
Major	3
Normal	2.5
Minor, Enhancement	2
Trivial	1

severity changed from qualitative index for deriving T_i in this thesis. Also, t is a normalized value of the fault fixing time for i-th fault data. Then, the standardized value of t is given as follows:

$$t_{standard_i} = \frac{t_i - min(t)}{max(t) - min(t)}, \tag{6.2}$$

where t_i is the fixing time for i-th bug. T_i is calculated by using the learning data. Then, the maximum and minimum values at $t_{standard_i}$ are the values used in the learning data.

6.2.2 Labelling with evaluation index

We can label the learning and test data in terms of High or Low based on the value of the evaluation index T_i created with i fault data. In this thesis, we assume the following three patterns in terms of the labeled learning and test data:

1. if $\mu \geq T_i$ then $High$ else Low
2. if $\mu + \sigma \geq T_i$ then $High$ else Low
3. if $\mu + 2\sigma \geq T_i$ then $High$ else Low

where μ is the mean value of i-th fault data, and σ the standard deviation. In particular, $High$ means that the fixing time of fault is long, and its severity is high in the i-th fault data. Also, we can judge that the faults recognized as $High$ under condition 3. are serious condition in all of the fixing time and severity, because the criteria for $High$ is the most severe of the three condition.

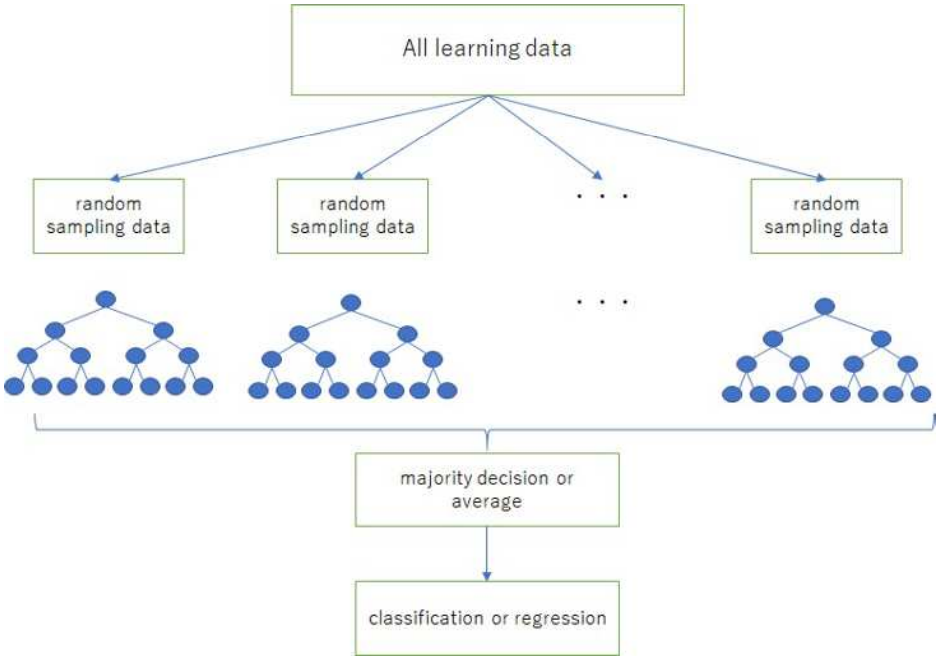

Fig. 6.1: Scheme of random forest.

Then, we should fix the fault identified as *High* as soon as possible. Then, we predict these labels by using the random forest and the logistic regression.

6.2.3 Prediction method with random forest

In this thesis, we use the random forest to predict *High* and *Low* in the test data. The random forest is a method of group learning by using the regression trees [62]. We show the scheme of random forest in Fig. 6.1. The repeated random sampling (reconstruction extraction) is performed on the data set for the model construction. Then, the multiple regression trees are created from the obtained sample group. The final prediction result is obtained by the majority of the output in each regression tree. In the conventional group learning, all explanatory variables are used in the model construction. Then, the selected randomly explanatory variables are used in the method of random forest.

In this thesis, we use two types of learning data and compare their accuracy. One is the case that only the version containing the test data (the current version) is used as learning data. The other case is that both the past and current versions are used as learning data, because the

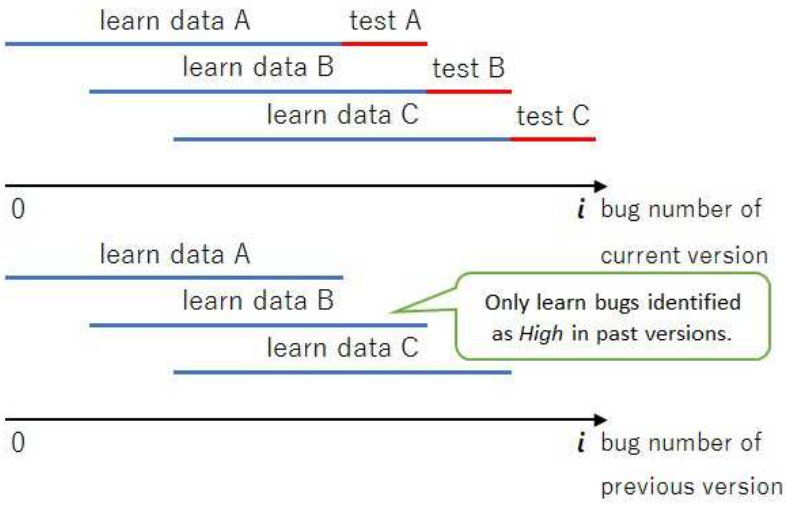

Fig. 6.2: Learning data with past version.

distribution of the value of index T_i is similar to the other versions.

In learning case of the past versions, we only include the faults that have been identified to be *High* in the learning data. As shown in Fig. 6.2, the number of data to be learned equals to the number of learning data in the current version.

6.2.4 Input data as the explanatory variable

As shown in Table 6.2, we have used the data obtained from the bug tracking system as explanatory variables. In particular, the variables of Component, OS, Reporter, Hardware, and Assignee are converted to the appearance rates in order to use as the learning data. Also, the range of possible values for Priority is 1 to 5.

6.3 Numerical examples

In order to evaluate the performance of the proposed method, we focus on version 17 of OpenStack [63] and versions 4.7 of Eclips [64] in this thesis. Each data set is obtained from a bug tracking system, Bugzilla [24] operated by the Apache Software Foundation. Also, we use 2284 fault report data for each version in OpenStack, and 1800 fault report data for each version in Eclipse. In this thesis, 200 learning data are used for preparing one test data. In order to predict

Table 6.2: Input data as explanatory variable.

Explanatory Variable	Scale	Description
Component	ratio scale	Component name where the bug occurred
Words Count of Summary	ratio scale	Number of words of detailed information in the reported bug
OS	ratio scale	Operating Sytem where the bug occurred
Weekday	nominal scale	Bug report day (weekday, weekend)
Reporter	ratio scale	Developer name who reported the bug
Type	nominal scale	Content of the bug report(Bug,Enhancement)
Opened Interval	interval scale	Elapsed days since last bug report
Opened Month	interval scale	Month the bug was reported
Hardware	ratio scale	Hardware on which the bug occurred
Opened(day)	interval scale	Elapsed days since the first bug was reported
Assignee	ratio scale	Developer name who fixed the bug

the value of T_i after the first 200 fault data, we move the learning data and the test data one by one as shown in Fig. 6.2.

6.3.1 Model evaluation criteria

In this thesis, we use *Recall*, *Precision*, and $F_1 measure$ [65] as the evaluation criteria. *Recall* means the proportion of correctly predicted faults among the faults identified as $High$. Also, *Precision* is the proportion of correctly predicted faults among the $High$ predicted faults. In this thesis, we use the $F_1 measure$, the harmonic mean of *Recall* and *Precision* as the evaluation criterion. In particular, we can use $F_1 measure$ to consider the balance between *Recall* and *Precision*. The $F_1 measure$ is defined as

$$F_1 measure = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{6.3}$$

All three evaluation criteria take the range [0,1]. The prediction accuracy is high if this value becomes large, and the prediction accuracy is low if this value becomes small.

6.3.2 Prediction results

Table 6.3 shows the accuracy of the prediction model using the random forest and the logistic regression for the Eclipse and OpenStack projects. There are two types of learning data in case of using the random forest, i.e., the first case is using the current version and the other case is using the current and past versions. Moreover, these results show the mean value in 10 times calculations.

Table 6.3 shows that the value of *Recall* is almost improved by learning both the current version and the past version in both OpenStack and Eclipse. On the other hand, in both cases, the value of *Precision* is almost becoming worse, but the value of $F_1 measure$ is improved. In conclusion, we have found that it is better to use the past version as well as the current version for learning data.

Also, we list the variable importance of the prediction model using random forest with the highest $F_1 measure$ and the threshold μ, $\mu + \sigma$, and $\mu + 2\sigma$ in Tables 6.4, 6.5, and 6.6, respectively. Moreover, we list in Tables 6.4, 6.5, and 6.6 as the variable importance, respectively.

Table 6.3: Accuracy of the prediction model for the Eclipse and OpenStack.

	threshold	OpenStack			Eclipse		
		Recall	Precision	F1 measure	Recall	Precision	F1 measure
A) Random forest using only current version	μ	0.583	0.467	0.519	0.386	0.320	0.350
	$\mu+\sigma$	0.165	0.235	0.194	0.160	0.203	0.179
	$\mu+2\sigma$	0.048	0.065	0.055	0.037	0.062	0.046
B) Logistic regression	μ	0.558	0.430	0.486	0.325	0.288	0.305
	$\mu+\sigma$	0.182	0.149	0.164	0.162	0.167	0.164
	$\mu+2\sigma$	0.193	0.035	0.060	0.070	0.035	0.047
C) Random forest using current and past version	μ	0.724	0.438	0.546	0.593	0.283	0.383
	$\mu+\sigma$	0.456	0.176	0.254	0.294	0.163	0.210
	$\mu+2\sigma$	0.198	0.036	0.060	0.064	0.047	0.054
Improvement rate from A) to C)	μ	24.15%	-6.30%	5.17%	53.80%	-11.50%	9.60%
	$\mu+\sigma$	176.62%	-25.06%	31.11%	84.18%	-19.45%	17.59%
	$\mu+2\sigma$	310.00%	-44.93%	9.31%	72.97%	-24.52%	16.53%
Improvement rate from B) to C)	μ	29.73%	1.83%	12.34%	82.46%	-1.60%	25.51%
	$\mu+\sigma$	150.86%	17.78%	54.85%	81.47%	-1.93%	27.88%
	$\mu+2\sigma$	2.50%	0.47%	0.78%	-8.57%	32.99%	15.49%

Then, we have learned 200 each fault from the first fault, the 501st fault, the 1001st fault, and the 1501st fault. In particular, the bold indicate that variable importance is the top five highest in prediction model.

Tables 6.4, 6.5, and 6.6 show that "Opened Interval" and "Opened (day)" have high variable importance in both Open Stack and Eclipse. In other words, we can understand that "time" is greatly influenced in terms of the prediction of T_i. Also, variable "Opened Month" has consistently high variable importance in OpenStack. On the other hand, Eclipse depends on the learning data.

6.4 Conclusion in this chapter

In many open source projects, bug tracking system is an important system for developing the OSS. OSS has been actively developed and used by a lot of people in recent years, so many faults in OSS have been reported. As a result, the developers sometimes take a long time to fix a fault, and sometimes miss a fix for faults. In this thesis, we aim to identify the fix priority of newly registered faults in the bug tracking system, and we have proposed the fix priority index considering the faults fixing time and severity. Also, we have compared with the most recently fixed faults to predict if the new reported fault's fix priority is high or not. In addition,

Table 6.4: Variable importance in threshold μ.

explanetory variable	scale	OpenStack				Eclipse			
		1-	501-	1001-	1501-	1-	501-	1001-	1501-
Component	ratio scale	9.413	11.329	**12.637**	9.389	**8.212**	**10.299**	**14.504**	9.993
Words Count of Summary	ratio scale	10.123	**11.795**	10.474	**14.735**	4.605	9.037	9.858	**11.180**
OS	ratio scale	**17.637**	4.463	7.476	6.498	7.210	8.562	**10.203**	6.304
Weekday	nominal scale	0.155	0.964	1.029	0.598	0.261	0.717	0.976	1.101
Reporter	ratio scale	8.672	**13.379**	8.709	11.196	**8.930**	**9.719**	9.783	**11.872**
Type	nominal scale	0.565	0.026	0.133	3.595	1.198	0.605	0.635	0.568
Opened Interval	interval scale	**15.942**	**18.848**	**17.232**	**11.773**	**13.371**	**14.670**	**14.149**	10.210
Opened Month	interval scale	**15.659**	**16.356**	**16.808**	**11.473**	5.789	5.615	3.541	**15.776**
Hardware	ratio scale	**15.072**	10.083	7.504	8.180	**8.071**	**12.108**	**13.239**	8.838
Opened (day)	interval scale	**30.698**	**32.835**	**38.634**	**29.332**	**47.035**	**25.780**	**35.889**	**33.477**
Priority	nominal scale	5.777	8.373	5.577	7.809	0.056	1.475	0.447	0.400
Assignee	ratio scale	9.413	9.967	**11.500**	**11.292**	5.761	9.187	8.863	**12.229**

Table 6.5: Variable importance in threshold $\mu + \sigma$.

explanetory variable	scale	OpenStack				Eclipse			
		1-	501-	1001-	1501-	1-	501-	1001-	1501-
Component	ratio scale	4.570	7.794	**6.791**	5.539	**4.981**	4.247	**8.239**	**9.897**
Words Count of Summary	ratio scale	4.172	**7.942**	**7.836**	**8.320**	1.141	5.758	5.753	7.296
OS	ratio scale	**10.235**	4.199	3.831	6.455	**5.000**	**6.258**	**9.225**	4.116
Weekday	nominal scale	0.424	0.616	0.829	1.508	0.312	0.417	0.473	1.457
Reporter	ratio scale	4.698	**8.098**	6.082	6.359	2.375	6.025	5.689	**10.477**
Type	nominal scale	0.772	0.011	0.029	1.703	0.392	0.246	0.797	0.120
Opened Interval	interval scale	**9.638**	**11.945**	**11.787**	**8.913**	**7.348**	**11.066**	**10.978**	**8.490**
Opened Month	interval scale	**8.983**	**14.950**	**15.361**	**12.550**	3.544	4.605	2.447	**7.227**
Hardware	ratio scale	**10.246**	7.928	4.247	**7.907**	**5.091**	**8.844**	**12.736**	4.449
Opened (day)	interval scale	**21.965**	**28.540**	**29.372**	**24.311**	**38.016**	**18.756**	**28.154**	**20.589**
Priority	nominal scale	2.725	3.713	3.748	5.195	0.098	1.789	0.798	0.325
Assignee	ratio scale	4.335	5.912	5.737	6.057	2.556	**6.834**	6.059	6.552

Table 6.6: Variable importance in threshold $\mu + 2\sigma$.

explanetory variable	scale	OpenStack				Eclipse			
		1-	501-	1001-	1501-	1-	501-	1001-	1501-
Component	ratio scale	1.615	2.131	1.906	1.295	**3.736**	2.467	**5.089**	**5.035**
Words Count of Summary	ratio scale	1.317	2.152	**3.503**	1.674	0.915	3.916	3.803	**5.239**
OS	ratio scale	2.583	0.830	1.333	**4.168**	**3.149**	**5.061**	**5.986**	2.351
Weekday	nominal scale	0.086	**0.252**	0.045	0.289	0.104	0.194	0.244	0.547
Reporter	ratio scale	1.652	2.076	**2.782**	1.512	1.455	3.604	3.659	3.804
Type	nominal scale	**2.812**	0.000	0.001	0.152	0.260	0.089	0.023	0.046
Opened Interval	interval scale	**4.290**	**4.118**	**3.369**	**3.361**	**4.755**	**6.743**	**5.370**	**5.333**
Opened Month	interval scale	**4.137**	**5.632**	**8.171**	**9.340**	**2.439**	**5.015**	1.295	**5.048**
Hardware	ratio scale	**3.517**	**3.994**	1.094	**4.021**	1.675	**8.486**	**10.075**	3.380
Opened (day)	interval scale	**7.562**	**10.821**	**12.875**	**12.263**	**27.241**	**13.347**	**13.859**	**15.654**
Priority	nominal scale	1.616	0.979	1.186	0.972	0.103	1.015	0.098	0.089
Assignee	ratio scale	1.399	1.277	1.767	1.802	1.821	4.350	3.238	2.481

we have found that the transition of the value of fix priority T_i shows the same tendency as different versions for the same OSS. In addition, we have found that it is possible to improve the prediction accuracy by learning not only the version of the fault for prediction but also the past version. From the above, we have considered that the proposed method using past version data is a practical method in order to identify the high fix priority of large open source projects.

Related Paper in this Chapter

1. H. Sone, Y. Tamura, and S. Yamada, “A method of fault identification considering high fix priority in open source project,” Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, Macau, China, December 15-18, 2019, CD-ROM (Project Management 1).

2. H. Sone, Y. Tamura, and S. Yamada, “A method of fault fix priority identification for open source project,” International Journal of Recent Technology and Engineering, Blue Eyes Intelligence Engineering & Sciences Publication, Vol. 8, No. 4, pp. 2396-2400, November 2019.

3. H. Sone, Y. Tamura, and S. Yamada, “Fault identification method considering high priority in open source project,” Proceedings of Forum on Information Technology 2019, Okayama

University, September 3-5, 2019, pp.141-142 (in Japanese).

4. H. Sone, Y. Tamura, and S. Yamada, “Instant fault identification analysis based on fixing priority for open source project,” Proceedings of 34th National Conference of the Society of Project Management, Hokkaido Citizens Actives Center, August 29-30, 2019, pp. 102-108 (in Japanese).

5. H. Sone, Y. Tamura, and S. Yamada, “A method of fault identification considering fault severity in OSS development,” Proceedings of the 12th Japan-Korea Software Management Symposium, Pusan, Korea, November 29-30, 2018, pp. 1-4.

Chapter 7

Optimum Maintenance Problem

OSS has a support period for each version, is called EOL. It is dangerous in terms of vulnerability to continue using the specified old version of OSS considering the EOL. Then we should upgrade the version. However, the maintenance cost increase with the version upgrade frequently. Therefore, it is necessary to update the OSS at a timing of the cost reduction. The timing of the optimal maintenance problem is shown Fig.7.1. Then, in this thesis, we find the optimum maintenance time by minimizing the total expected software maintenance effort. In addition, we have verified the appropriateness of the optimum maintenance time in terms of the cumulative number of reported faults, because the proper management of maintenance effort affects the software quality. Furthermore, several numerical examples of the earned value analysis and the effort optimization based on the proposed method are shown by using the effort data under actual open source project.

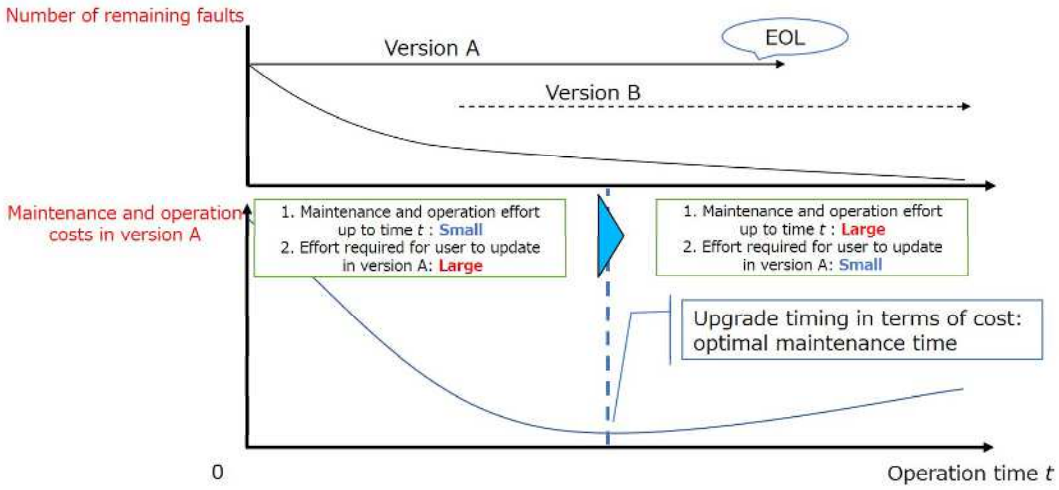

Fig. 7.1: Overview of the optimum maintenance problem.

7.1 Related research

In this thesis, we focus on the development effort of OSS. In particular, we focus on the optimal maintenance problem based on Tamura et al.'s research using the SRGM [66] with Wiener process for predicting the development effort. The SRGM is mainly used for software reliability assessment. Zhou et al. [67] collected bug tracking data from a few popular open source projects and investigated the time related bug reporting patterns from them. This research shows that the trends of reliability growth are the same cases between the open source project and the other projects. Therefore, the research shows that the SGRM is appropriate to use for the open source projects. In addition, several previous researches have shown in terms of the reliability evaluation of OSS and proprietary software, the model formula using SRGM with Wiener process is better than SRGM based on non-homogeneous Poisson process (NHPP) [68].

The optimal maintenance problem is an application of the optimal software release problem [69, 70] proposed by Yamada et al. The optimum software release problem means the deriving a time when to stop the software testing by minimizing the expected total software cost [69,70]. On the other hands, the optimum maintenance problem means the deriving the optimum maintenance time by minimizing the total expected software maintenance effort in OSS development.

In the past, Tamura et al. have proposed several solutions to the optimal maintenance problem in OSS development [39, 71]. In these research papers, various prediction models have been applied to the development effort. Also, Tamura et al. have proposed an OSS-oriented EVM. However, they did not apply it to the optimal maintenance problem. Therefore, we focus on the optimal maintenance problem considering OSS-oriented EVM in OSS development. In addition, we verify the appropriateness of the optimum maintenance time in terms of OSS quality.

7.2 Optimum maintenance time based on wiener process models

This chapter discusses the optimal maintenance time problem by minimizing the maintenance effort expenditures for the operation of OSS. Then, using the derivation method of the optimal release problem [69, 70], one of the software reliability evaluation methods, we define the fol-

lowing effort rate parameters:

e_1 : the maintenance effort per effort needed to operate OSS,

e_2 : the operation effort per unit time during the operation,

e_3 : the maintenance effort per effort after the upgrade task such as major version upgrade.

Then, the expected maintenance effort expenditures in the operation of OSS can be formulated as:

$$E_1(t) = e_1 \mathrm{E}[\Omega_*^a(t)] + e_2 t. \tag{7.1}$$

Also, the expected software maintenance effort expenditures after the maintenance of OSS is represented as follows:

$$E_2(t) = e_3 \mathrm{E}[ETC_*(t)]. \tag{7.2}$$

In particular, we consider ETC obtained from OSS-oriented EVM as the software maintenance effort after the maintenance of OSS.

Consequently, from Eqs. (7.1) and (7.2), the total expected software maintenance effort expenditures during the specified period such as the specified version is given by

$$E(t) = E_1(t) + E_2(t). \tag{7.3}$$

The optimum maintenance time t^* is obtained by minimizing $E(t)$ in Eq. (7.3).

7.3 Consideration of optimum maintenance time for software quality

Although the optimal maintenance time can be derived using Eq. (7.3), there is no research on verifying that the maintenance time is appropriate. In this thesis, we judge whether the optimal maintenance time is an appropriate one in terms of the transition probability distribution of the number of reported faults. Specifically, we predict the cumulative number of faults reported using the equation in chapter 4. Then, the transition probability distribution [47] of the cumulative number of reported faults is derived. Finally, we discuss the comparison of goodness-of-fit for the optimum maintenance time.

In the following Eq. (7.4), (7.5), and (7.6), $\Omega(t)$ is the cumulative number of fault reported up to operational time t ($t \geq 0$) in the open source project. Also, $\beta(t)$ is the increase rate of the number of reported faults at operational time t and a non-negative function, and α is the estimated number of reported faults required until the end of operation, and σ is a positive constant representing a magnitude of the irregular fluctuation, c is a positive constant by using the infection S-shaped model.

Since the Wiener process $\omega(t)$ is a Gaussian process, $\log\{\alpha - \Omega(t)\}$ is also a Gaussian process. The mean values of $\log\{\alpha - \Omega(t)\}$ are derived as follows:

$$\begin{aligned}
\mathrm{E}[\log\{\alpha - \Omega_e(t)\}] &= \log\alpha - \beta t, && (7.4)\\
\mathrm{E}[\log\{\alpha - \Omega_s(t)\}] &= \log\alpha - \beta t + \log(1+\beta t), && (7.5)\\
\mathrm{E}[\log\{\alpha - \Omega_i(t)\}] &= \log\alpha - \beta t + \log\frac{1+c\cdot\exp(-\beta t)}{1+c}. && (7.6)
\end{aligned}$$

Also, the variances of $\log\{\alpha - \Omega(t)\}$ are derived as follows:

$$\begin{aligned}
\mathrm{Var}[\log\{\alpha - \Omega_e(t)\}] &= \sigma^2 t, && (7.7)\\
\mathrm{Var}[\log\{\alpha - \Omega_s(t)\}] &= \sigma^2 t, && (7.8)\\
\mathrm{Var}[\log\{\alpha - \Omega_i(t)\}] &= \sigma^2 t. && (7.9)
\end{aligned}$$

Thus, the following equations are derived:

$$\begin{aligned}
\Pr[\log\{\alpha - \Omega_e(t)\} \leq x] &= \Phi\left(\frac{x - \log\alpha + \beta t}{\sigma\sqrt{t}}\right), && (7.10)\\
\Pr[\log\{\alpha - \Omega_s(t)\} \leq x] &= \Phi\left(\frac{x - \log\alpha + \beta t - \log(1+\beta t)}{\sigma\sqrt{t}}\right), && (7.11)\\
\Pr[\log\{\alpha - \Omega_i(t)\} \leq x] &= \Phi\left(\frac{x - \log\alpha + \beta t + \log\frac{1+c\cdot\exp(-\beta t)}{1+c}}{\sigma\sqrt{t}}\right), && (7.12)
\end{aligned}$$

where x is the cumulative number of reported faults at time t. Also, Φ means standard normal distribution and is defined as follows:

$$\Phi(x) = \frac{1}{\sqrt{2\sigma}}\int_{-\infty}^{x}\exp(-\frac{y^2}{2})dy. \quad (7.13)$$

Considering the above points, the transition probability distributions of $\Omega_e(t)$, Ω_s, and (t) are

obtained as:

$$\Pr[\Omega(t) \leq n | \Omega_e(0) = 0] = \Phi\left(\frac{\log\frac{\alpha}{\alpha-n} - \beta t}{\sigma\sqrt{t}}\right), \tag{7.14}$$

$$\Pr[\Omega(t) \leq n | \Omega_s(0) = 0] = \Phi\left(\frac{\log\frac{\alpha}{\alpha-n} - \beta t + \log(1+\beta t)}{\sigma\sqrt{t}}\right), \tag{7.15}$$

$$\Pr[\Omega(t) \leq n | \Omega_i(0) = 0] = \Phi\left(\frac{\log\frac{\alpha}{\alpha-n} - \beta t - \log\frac{1+c\cdot\exp(-\beta t)}{1+c}}{\sigma\sqrt{t}}\right). \tag{7.16}$$

7.4 Application of proposed method to actual data

7.4.1 Used data set

In this thesis, we use the data of open source project to derive the OSS-oriented EVM and the optimum maintenance time. For applying the proposed model to actual project data, we use the data of OpenStack [63] obtained from Bugzilla. The OpenStack is OSS for cloud computing. This project uses the Bugzilla [24] as open source bug tracking system. The data about the reported faults is freely available from the bug tracking system. In particular, the effort and fault data were obtained from Bugzilla are version 16(Pike). For estimating PV and AC, in this thesis, the cumulative number of reported faults is 655 and 2249. In particular, we use the project data for about 8 months before OpenStack was released to predict PV. For prediction AC, we also use the project data for about 16 months after OpenStack released. Also, each data is weekly unit data.

7.4.2 Numerical examples for optimum maintenance time

Tables 7.1 and 7.2 show the results of parameter estimation of maintenance effort, and AIC [38]. In terms of AIC, the delayed S-shaped model fits better than the exponential model. Also, the parameter α in the PV data can be rephrased as BAC.

In addition, we used Eqs. (4.8)-(4.13) to derive the parameters for the cumulative number of faults. Tables 7.3 and 7.4 show the results of parameter estimation of number of fault, and AIC. In terms of AIC, the infection S-shaped model and the delayed S-shaped model fit better. Also, the parameter α in the PV data can be rephrased as potential faults at OSS release. In other

Table 7.1: Parameter estimation of maintenance effort in terms of PV.

		Planned Value		
parameter		exponential	delayed S-shaped	infection S-shaped
	α	2.315×10^{6}	2.004×10^{6}	9.847×10^{5}
	β	2.879×10^{-3}	1.640×10^{-2}	1.257×10^{-2}
	c	-	-	9.422×10^{-1}
	σ	2.315×10^{-3}	1.617×10^{-3}	5.982×10^{-3}
AIC		724.297	**687.640**	728.550

Table 7.2: Parameter estimation of maintenance effort in terms of AC.

		Actual Cost		
parameter		exponential	delayed S-shaped	infection S-shaped
	α	1.436×10^{7}	4.070×10^{6}	4.044×10^{6}
	β	1.012×10^{-3}	1.179×10^{-2}	1.833×10^{-2}
	c	-	-	9.675
	σ	4.864×10^{-4}	1.343×10^{-3}	1.712×10^{-3}
AIC		2112.663	**2035.109**	2089.214

Table 7.3: Parameter estimation of number of potential faults in case of OpenStack.

		Estimated number of potential faults at OSS release		
		exponential	delayed S-shaped	infection S-shaped
parameter	α	1.961×10^{3}	4.083×10^{3}	2.728×10^{3}
	β	1.160×10^{-2}	1.963×10^{-2}	8.577×10^{-2}
	c	-	-	6.781×10^{1}
	σ	1.110×10^{-2}	4.121×10^{-3}	5.670×10^{-3}
AIC		319.766	306.047	**300.255**

Table 7.4: Parameter estimation of number of resolved faults in case of OpenStack.

		Estimated number of resolved faults at present		
		exponential	delayed S-shaped	infection S-shaped
parameter	α	1.923×10^{4}	5.796×10^{3}	1.126×10^{4}
	β	9.978×10^{-4}	1.159×10^{-2}	2.633×10^{-3}
	c	-	-	5.359×10^{-1}
	σ	2.202×10^{-3}	8.082×10^{-3}	3.786×10^{-3}
AIC		1064.888	**1062.983**	1066.369

words, from Eq. (4.20), we can calculate the fault resolving cost $\gamma \coloneqq 361$ (man $\cdot$ days).

Fig. 7.2 shows the result of deriving PV, AC, and EV. In particular, the most appropriate model equation in terms of AIC is used to derive each indicator of EVM.

Figs. 7.3 and 7.4 show the result of deriving CPI and ETC. From the results of Fig. 7.4, we found that the value of ETC is negative. This is because the number of faults actually corrected exceeds the number of potential faults initially assumed.

Fig. 7.5 shows the estimated total software effort by using Eq. (7.3). We found that the optimum maintenance time is derived as $t^* = 6.886$ years 359.3 (weeks).

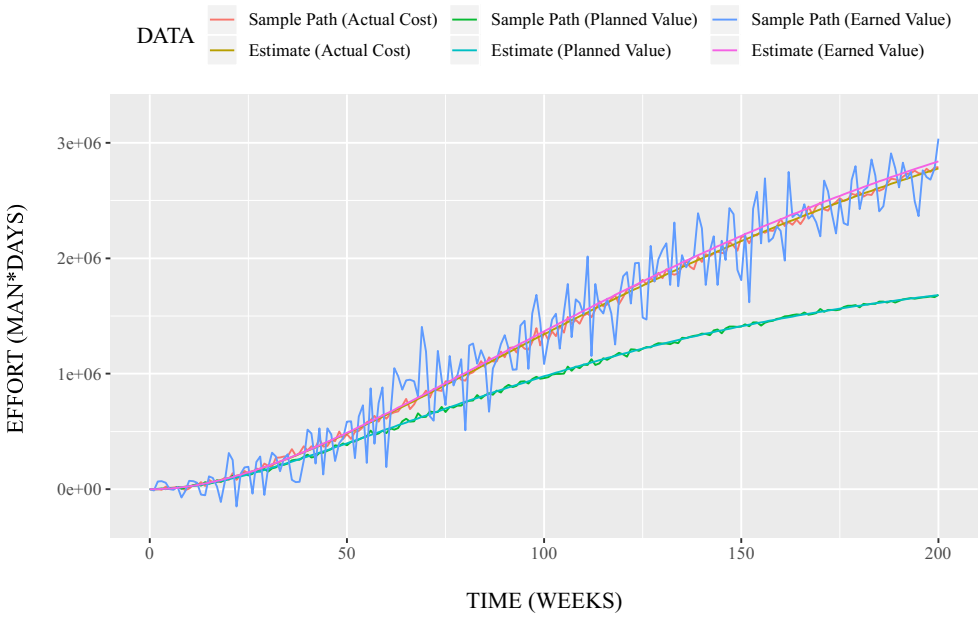

Fig. 7.2: EVM estimation results in OpenStack project.

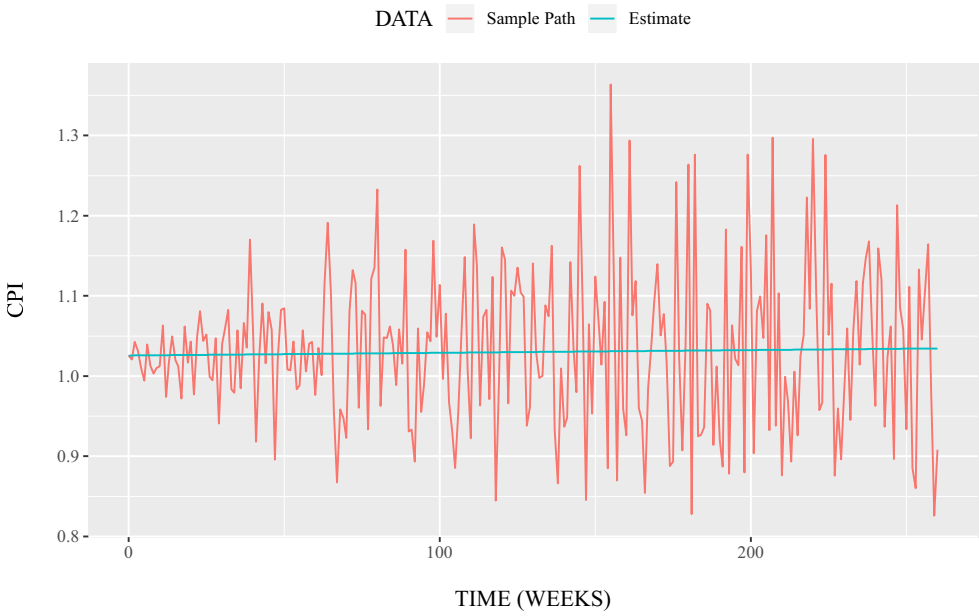

Fig. 7.3: The result of CPI in OpenStack Ver. 16 project.

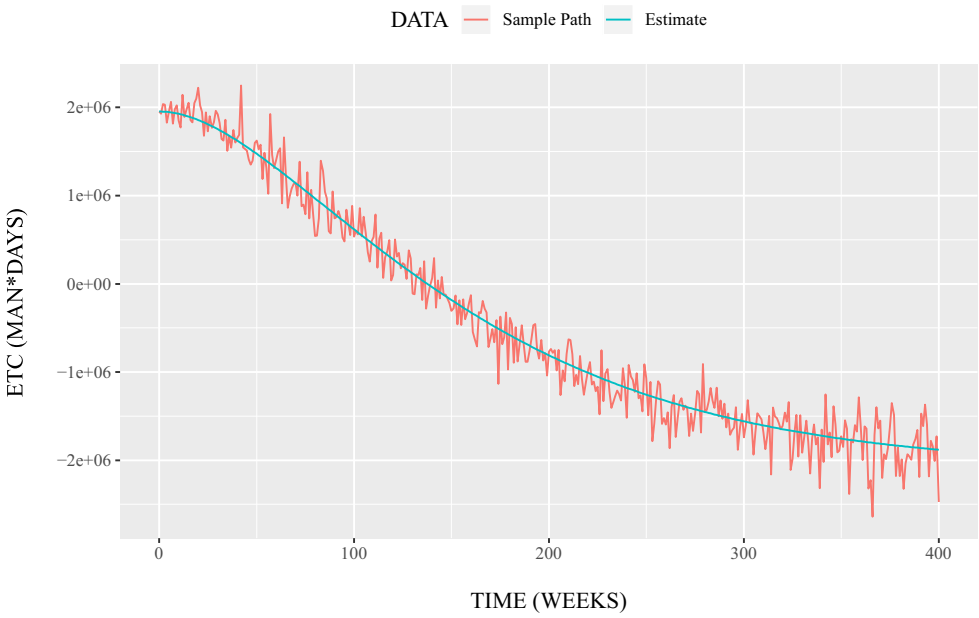

Fig. 7.4: The result of ETC in OpenStack Ver. 16 project.

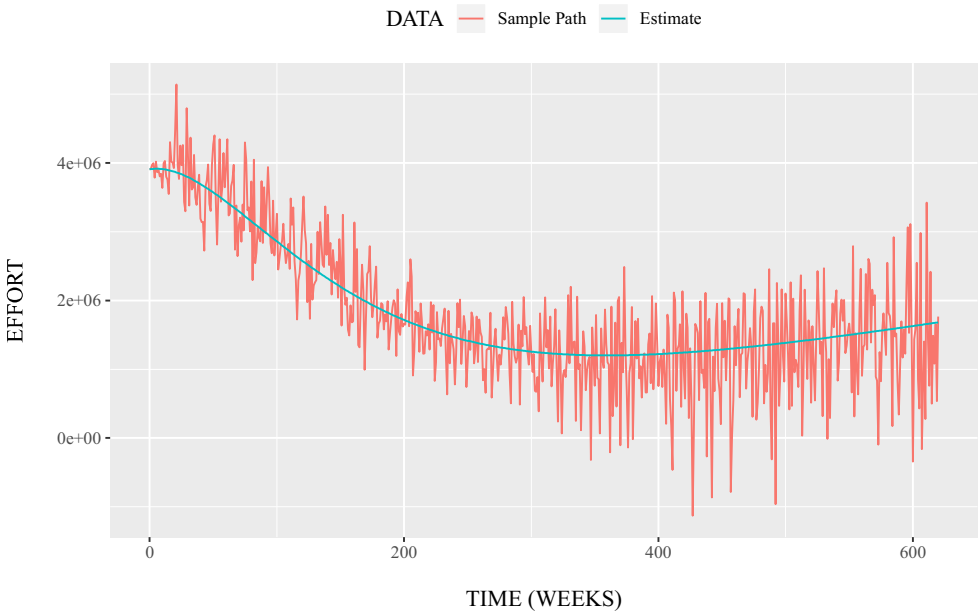

Fig. 7.5: The estimated total software effort in OpenStack Ver. 16 project.

Table 7.5: Fault detection rate and probability in OpenStack project.

Fault detection rate	70%	85%	90%	95%
Probability	1	0.992	0.254	2.0×10^{-9}

7.4.3 Numerical examples considering the software quality of optimum maintenance time

In order to evaluate the performance of the result of the optimum maintenance time, we use the result of estimating the cumulative number of revised faults shown in Table 4.6. In terms of AIC, the delayed S-shaped model fits better than the other models for a suitable model.

For predicting the time when reported faults reach 50%, 70%, 90%, and 99%, transition probability distributions are shown in Figs. 7.6 by using Eq. (7.16).

For performance assessment of the optimum maintenance time, we use the estimated total software effort and transition probability. Fig. 7.7 shows the result of the estimated cumulative maintenance effort expenditures. Simultaneously, from Fig. 7.7, the optimum maintenance time is roughly appropriate prediction, because the optimum maintenance time is near the time when the transition probability of cumulative revised faults reaches 90%. In other words, the OSS maintains high quality at the optimum maintenance time. Also, Table 7.5 shows the total fault detection rate at the optimal maintenance time. From this result, we can see that 85% of potential faults in this OSS are reported with a probability of 99.2%, and maintaining the high quality in terms of the fault fixing.

From the above results, it is possible to evaluate the optimum maintenance time considering the risk of system failure due to unrevised faults by predicting the revised status of faults.

7.5 Conclusion in this chapter

It is important for OSS users to decide the optimal length of version upgrade duration and maintenance time considering the operation status of open source project management. The optimal maintenance time problem based on maintenance effort for OSS have been proposed in this the-

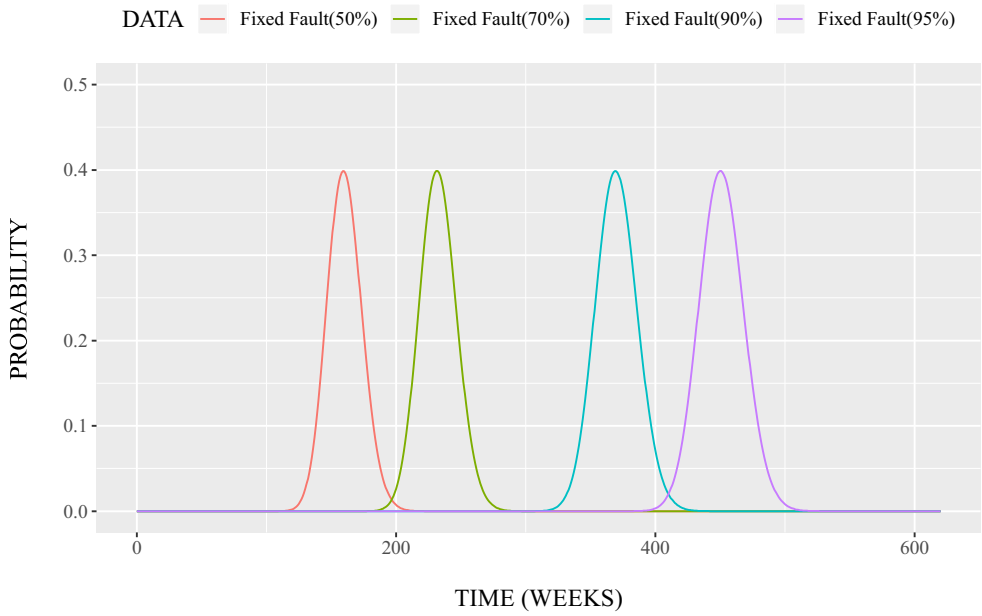

Fig. 7.6: The transition probability distribution of the number of faults revised in case of Eq. (7.16)

sis. In particular, we have defined the optimal maintenance time problems considering the CPI as the stability. In the past, it is difficult to control the OSS quality and manage the project under the traditional method based on fault data for software reliability assessment. By focusing on the reporter and assignee on the fault big data recorded on the bug tracking system of open source project, we have proposed a new approach based on the earned value and optimal maintenance time problem considering the project stability by using OSS effort data. In particular, this thesis has proposed the method for deriving optimum maintenance time of open source projects considering the irregular fluctuation from the characteristics of OSS development and management by using OSS-oriented EVM. In addition, we have judged whether the optimal maintenance time is an appropriate time in terms of the transition probability distribution of the cumulative number of reported faults, because the proper maintenance management affects the software quality. As the result, we have derived the optimum maintenance time from the proposed models. The proposed method will be helpful as the assessment method of the progress of the open source projects in operation phase. Also, we have found that our method can assess the stability and effort control

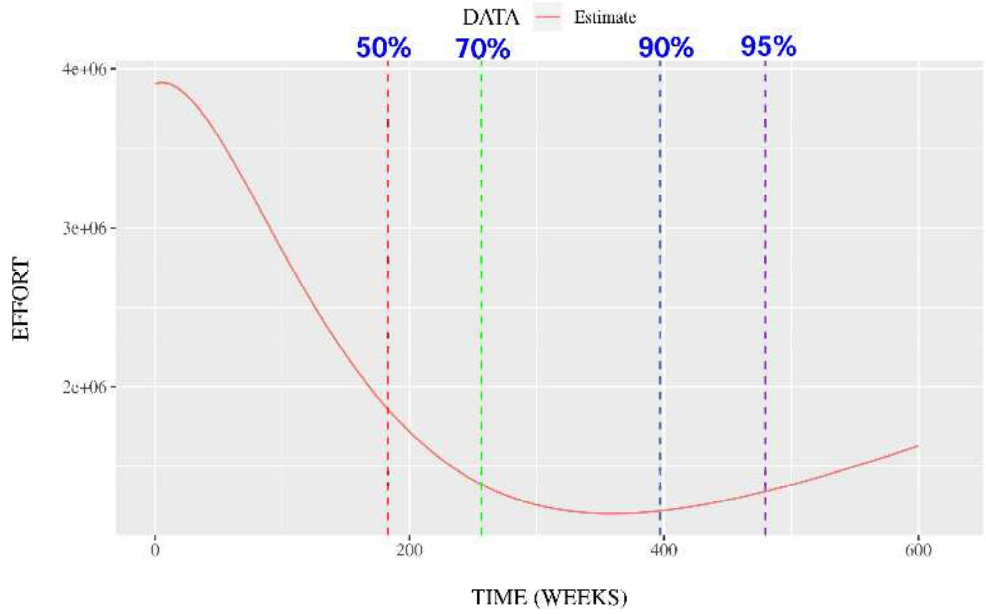

Fig. 7.7: The estimated total software effort and the transition probability of cumulative revised faults.

considering the operational environment of OSS. Furthermore, the data set of actual OSS effort has been analyzed to show several numerical examples of the progress analysis and optimum maintenance time considering the stability for an open source project.

In the future, we will propose the generalized method for deriving the validity of the optimum maintenance time.

Figures not shown in this chapter are shown in Appendix.

Related Paper in this Chapter

1. 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.

2. H. Sone, Y. Tamura, and S. Yamada, "Statistical maintenance time estimation based on stochastic differential equation models in OSS development project," Computer Reviews

Journal, PURKH, Vol. 5, pp. 126-140, December 2019.

3. H. Sone, Y. Tamura, and S. Yamada, “Optimal maintenance problem with earned value requirement for OSS project,” Proceedings of the Fourteenth International Conference on Industrial Management, Hangzhou, China, September 12-14, 2018, pp. 253-258.

4. H. Sone, Y. Tamura, and S. Yamada, “Stochastic differential equation modeling for development effort estimation and its application in open source project,” Proceedings of 32th National Conference of the Society of Project Management, Doshisha University, August 30-31, 2018, pp. 439-443 (in Japanese).

Chapter 8

Conclusion

This thesis has focused on the following three issues among many ones. We have examined the method for OSS users and open source project managers to evaluate the stability of open source projects.

1. Selection evaluation and licensing: Methods for OSS users to make selections from the many OSS available situation,

2. Vulnerability support: Predicted fault fix priority for the reported OSS,

3. Maintenance and quality assurance: Prediction of appropriate OSS version upgrade timing, considering the development effort required after OSS upgrade by OSS users.

In "1. Selection evaluation and licensing," we have attempted to derive the OSS-oriented EVM by applying the EVM to several open source projects. In order to derive the OSS-oriented EVM, we have applied the stochastic models based on SRGM considering the uncertainty for the development environment in open source projects. We have also improved the method of deriving effort in open source projects. As a result, in terms of the method for deriving effort, the effort data of open source project has shown similar trends to proprietary software development and has shown its applicability to SRGM. Therefore, by using our proposed method to calculate the effort of projects, it is possible to apply the proposed method to actual project management methods such as OSS-oriented EVM. In case of applying the existing method of deriving effort in open source projects, it is not possible to derive some indices in the OSS-oriented EVM. Thus, we have resolved this issue. The derived OSS-oriented EVM helps OSS users and open source project managers to evaluate the stability of their current projects. It is an important to use the decision-making tool regarding their decisions and projects of OSS. From a different perspective, we have also evaluated the stability of the project in terms of the speed of fault fixing by predicting

the time transition of fixing the OSS faults reported in the future. It is not easy to evaluate the progress and stability of open source projects because of the large scale and the lack of visibility in the entire group of people involved in the open source software development. Therefore, we believe that our project evaluation methods used in conventional software development will increase the number of evaluation methods for OSS development and contribute greatly to the development of OSS.

From the different perspective, we have also evaluated the stability of the project in terms of the speed of fault fixing by predicting the time transition of fixing the OSS faults reported in the future. In particular, considering the characteristics of changes in the fault fixing time under the large-scale open source projects and the complexity in OSS development, we predicted the transition of fault fixing time based on the Wiener process. Furthermore, we have considered the possibility that the fault status could be REOPEN and examined the number of training data. As the results, there is a possibility that the last fault fixing time can be considered as the final fault fixing time by excluding the most recent fault data. The proposed method can be helpful for the project managers and OSS users as the evaluation method of open source project progress in the operation phase.

In "2. Vulnerability support," in terms of the open source project managers, we have created several metrics to detect faults. The reported faults with a high priority is speedy fix, and a long time to fix, and predict. In addition, we have tried to improve the detection accuracy of the proposed metrics by learning not only the specific version but also the bug report data of the past version by using the random forest considering the characteristic similarities of bugs fix among different versions. This allows the project managers to identify the faults that should be prioritized for fixing when the large number of faults are reported, and facilitates project operations. In "2. Vulnerability support," we have aimed to identify the fix priority of newly registered faults in the bug tracking system, and we have proposed the fix priority index considering the faults fixing time and severity. Also, we have compared with the most recently fixed faults to predict if the new reported fault's fix priority is high or not. In addition, we have found that the transition of the value of fix priority T_i shows the same tendency as different versions for the same OSS. In addition, we have found that it is possible to improve the prediction accuracy by learning not only the version of the fault for prediction but also the past version. From the above results, we have

considered that the proposed method using the past version data is a practical method in order to identify the high fix priority of large open source project. This allows the project managers to identify the faults that should be prioritized for fixing when the large number of faults are reported, and facilitates the project operations.

In "3. Maintenance and quality assurance," as the optimum maintenance problem, we have predicted the appropriate OSS version-up timing considering the maintenance effort required by OSS users after upgrading the OSS. It is dangerous in terms of the vulnerability to continue using the specified version of OSS ignoring the EOL. Therefore, we should upgrade the version periodically. However, the maintenance cost increase with the version upgrade frequently. Then, we have found the optimum maintenance time by minimizing the total expected software maintenance effort in terms of OSS users. In particular, this thesis has proposed the method for deriving optimum maintenance time of open source projects considering the irregular fluctuation from the characteristics of OSS development and management by using the OSS-oriented EVM. In addition, we have judged whether the optimal maintenance time is an appropriate time in terms of the transition probability distribution of the cumulative number of reported faults, because the proper maintenance management affects the software quality. As the result, we have derived the optimum maintenance time from the proposed models. The proposed method will be helpful as the assessment method of the progress for the open source project in operation phase.

In conclusion, we have found that there is the applicability as the stability evaluation of open source projects from three perspectives. In particular, the OSS-oriented EVM discussed in "1. Selection evaluation and licensing" can contribute to the visualization of maintenance effort in open source projects. The proposed method will potentially contribute to the development of OSS in the future.

References

[1] S.E. Raymond, “The cathedral and the bazzar: Musings on Linux and open source by an accidental revolutionary,” O’Reilly and Associates, Sebastopol, California, 1999.

[2] Cybersecurity Division, Commerce and Information Policy Bureau, “Collection of use case examples compiled regarding management methods for utilizing open source software and ensuring its security,” Ministry of Economy, Trade and Industry, 2021 (in Japanese).

[3] W.W. Royce, “Managing the development of large software systems: Concepts and techniques,” Proceedings of IEEE WESCON, August 1970, pp.1-9.

[4] N.M.A. Munassar and A. Govardhan, “A Comparison between Five Models of Software Engineering,” International Journal of Computer Science Issues (IJCSI), Vol. 7, No. 5, pp. 94-101, 2010.

[5] agilemanifesto.org, “Principles Behind the Agile Manifesto,” https://agilemanifesto.org/

[6] D.Dönmez, G. Grote, and S. Brusoni, “Routine interdependencies as a source of stability and flexibility. A study of agile software development teams,” Information and Organization, Vol. 7, Issue 3, 2016, pp. 63-83.

[7] N. Ozkan, M.S. Gök, and B.ö.Köse, “Towards a better understanding of agile mindset by using principles of agile methods,” Proceedings of the 2020 15th Conference on Computer Science and Information Systems (FedCSIS), Sofia, Bulgaria, 2020, pp. 721-730.

[8] K. Beck, “Embracing change with extreme programming,” Computer, Vol. 32, no. 10, pp. 70-77, 1999.

[9] B.W. Boehm, “A spiral model of software development and enhancement,” Computer, Vol. 21, pp. 61-72, 1986.

[10] B.W. Boehm, “Software engineering economics,” Prentice Hall, New Jersey, 1981.

[11] B.W. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. Steecem and B.K. Clark, “Software cost estimation with COCOMO II,” Prentice Hall, New Jersey, 2000.

[12] C. Jones, “Software reliability: measurement, prediction, application,” McGraw-Hill, New York, NY, USA, 2008.

[13] P.N. Misra, “Software reliability analysis,” IBM Systems Journal, Vol. 22, No. 3, pp. 262-270, 1983.

[14] A.L. Goel and K. Okumoto, “A time dependent error detection model for software reliability and other performance measures,” IEEE Trans. Reliability, Vol. R-28, pp. 206-211, 1979.

[15] S. Yamada, M. Ohba, and S. Osaki, “S-shaped reliability growth modeling for software error detection,” IEEE Trans. Reliability, Vol. R-32, pp. 475-484, 1983.

[16] M. Ohba, “Inflection S-shaped software reliability growth model,” Stochastic Models in Reliability Theory, vol. 235, Springer, Berlin, Heidelberg, pp. 144-162, 1984.

[17] Q.E. Fleming, J.M. Koppelman, “Earned value project management (4th Ed.),” PMI, Newton Square, U.S.A., 2010.

[18] W.F. Abba, “The evolution of earned nalue management,” The Measurable News, Issue 2 pp. 9-12, 2017.

[19] Linux.org, “Linux.org,” https://www.linux.org/

[20] The Open Source Initiative, “To simplify software development,” https://opensource.org/

[21] M. Gallivan, “Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies,” Information Systems Journal, Vol. 11, Issue 4, pp. 277-304, 2001.

[22] K. Crowston and B. Scizzi, “Open source software projects as virtual organizations: Competency rallying for software development,” IEE Proceedings Software, Vol. 149, Issue 1, No. 1, pp. 3-17, 2001.

[23] S. Sharma, V. Sugumaran, and B. Rajagopalan, "A framework for creating hybrid-open source software communities," Information Systems Journal, Vol. 12, Issue 1, pp. 7-25, 2002.

[24] Bugzilla, "The software solution designed to drive software development," https://www.bugzilla.org/

[25] G. Robles, M.J. González-Barahona, C. Cervigön, A. Capiluppi, and D. Izquierdo-Cortäzar, "Estimating development effort in Free/OSS projects by mining software repositories: a case study of OpenStack," Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad. India. 31 May-1 June, 2014, pp. 222-231.

[26] R.G. Kula, K. Fushida, N. Yoshida, and H. Iida, "Micro process analysis of maintenance effort: an open source software case study using metrics based on program slicing," Journal of Software: Evolution and Process, Vol. 25, Issue 9, pp. 935-955, 2013.

[27] S.M. Rakha, W. Shang, and E.A. Hassan, "Studying the needed effort for identifying duplicate issues," Empirical Software Engineering, Vol. 21, Issue 5, pp. 1960-1989, 2016.

[28] R. Mishra and A. Sureka, "Mining peer code review system for computing effort and contribution metrics for patch reviewers," Proceedings of the 2014 IEEE 4th Workshop on Mining Unstructured Data, Victoria, Canada, 30 September, 2014, pp. 11-15.

[29] M.D. Koulla, A. Alain, and Kolyang, "Duration estimation models for open source software projects," International Journal of Information Technology and Computer Science, Vol. 13 Issue 1, pp. 1-17, 2021.

[30] C. Sun, D. Lo, X. Wang, J. Jiang, and S.A. Khoo, "Discriminative model approach for accurate duplicate bug report retrieval," Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), Cape Town, South Africa, 2-8 May, 2010, pp.45-54.

[31] S. M. Rakha, W. Shang, and E. A. Hassan, "Studying the needed effort for identifying duplicate issues," Empirical Software Engineering, Springer Science+Business Media: Berlin, Germany, 2015.

[32] Y. Tamura and S. Yamada, “Maintenance effort management based on double jump diffusion model for OSS project,” Annals of Operations Research, Vol. 312 Issue 1, pp. 411-426, 2022.

[33] K. Sugisaki, Y. Tamura, and S. Yamada, “OSS Effort expense optimization based on Wiener process model and GA,” Journal of Software Engineering and Applications, Vol. 14, No. 1, pp. 11-25, 2021.

[34] sourceforge.net, “The Complete Open-Source and Business Software Platform,” https://sourceforge.net/

[35] M. Ohira, N. Ohsugi, T. Ohoka, and K. Matsumoto, “Accelerating cross-project knowledge collaboration using collaborative filtering and social networks,” Proceedings of 2nd International Workshop on Mining Software Repositories (MSR2005), St. Louis, Missouri, 17 May, 2005, pp. 111-115.

[36] Red Hat, “Red Hat Enterprise Linux,” https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux

[37] The Document Foundation, “LibreOffce,” https://libreoffce.org/

[38] H. Akaike, “A new look at the statistical model identification,”IEEE Transactions on Automatic Control, vol. 19, no. 6, pp. 716-723, 1974.

[39] H. Sone, Y. Tamura, and S. Yamada, “Optimal maintenance problem with earned value requirement for OSS project,” Proceedings of the Fourteenth International Conference on Industrial Management, Hangzhou, China, 12-14 September, 2018, pp. 253-258.

[40] H. Sone, Y. Tamura, and S. Yamada, “Statistical maintenance time estimation based on stochastic differential equation models in OSS development project,” Computer Reviews Journal, PURKH, Vol. 5, pp. 126-140, 2019.

[41] S. Yamada, “Software reliability modeling: Fundamentals and applications,” Springer-Verlag, Tokyo/Heidelberg, 2014.

[42] M.R. Lyu, Ed., "Handbook of software reliability engineering," IEEE Computer Society Press, Los Alamitos, CA, U.S.A., 1996.

[43] J.D. Musa, A. Iannino, K. Okumoto, "Software reliability: measurement, prediction," Application. McGraw-Hill, New York, 1987.

[44] P.K. Kapur, H. Pham, A. Gupta, and P.C. Jha, "Software reliability assessment with OR applications," Springer-Verlag, London, 2011.

[45] E. Wong, "Stochastic processes in information and systems," McGraw-Hill, New York, 1971.

[46] L. Arnold, "Stochastic differential equations-theory and applications," John Wiley & Sons, New York, 1971.

[47] S. Yamada, M. Kimura, H. Tanaka, and S. Osaki, "Software reliability measurement and assessment with stochastic differential equations," IEICE Transactions on Fundamentals, Vol. E77-A, No. 1, 109-116, 1994.

[48] K. Ranjan, K. Subhash, and K.T. Sanjay, "A study of software reliability on big data open source software," International Journal of System Assurance Engineering and Management, Vol. 10, No. 2, pp. 242-250, 2019.

[49] P. Hooimeijer and W. Weimer, "Modeling bug report quality," Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering(ASE '07), Georgia, USA, 5-9 November, 2010, pp. 34-43.

[50] M. Nurolahzade, S.M. Nasehi, S.H. Khandkar, and S. Rawal, "The role of patch review in software evolution: an analysis of the mozilla firefox," Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops, Amsterdam, The Netherlands, 24-25 August, 2009, pp. 9-18.

[51] Y. Tamura, H. Sone, and S. Yamada, "OSS project stability assessment support tool considering EVM based on wiener process models," Applied System Innovation, Vol. 2, No. 1, pp. 1-12, 2019.

[52] R. Mishra and A. Sureka, "Mining peer code review system for computing effort and contribution metrics for patch reviewers," Proceedings of the 2014 IEEE 4th Workshop on Mining Unstructured Data, Victoria, BC, Canada, 30 September, 2014, pp. 11-15.

[53] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, Cape Town, South Africa, 4 May, 2010, pp.52-56.

[54] G. Bougie, C. Treude, M. D. German, and A.M. Storey, "A comparative exploration of freeBSD bug lifetimes," Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa, 2-3 May, 2010, pp.106-109.

[55] S. Akbarinasaji, B. Caglayan, and A. Bener, "Predicting bug-fixing time: A replication study using an OSS project," Journal of Systems and Software, Vol. 136, pp. 173-186, 2018.

[56] L. Marks, Y. Zou, and E.A. Hassan, "Studying the fix-time for bugs in large open source projects," Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise'11), Alberta, Canada, 20-21 September, 2011, pp.11:1-11:8.

[57] C. Sun, D. Lo, X. Wang, J. Jiang, and S.C. Khoo, "A discriminative model approach for accurate duplicate," Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Cape Town, South Africa, 1-8 May, 2010, pp. 45-54.

[58] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?," Proceedings of the 8th Working Conference on Mining Software Repositories, Hawaii, USA, 21-22 May, 2011, pp. 207-210.

[59] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," Proceedings of the International Conference on Software Engineering, Cape Town, South Africa, 1-8 May, 2010, pp. 1-10.

[60] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," Proceedings of the 16th International Symposium on Foundations of Software Engineering, Atlanta, Georgia, 9-14 November, 2008, pp. 308-318.

[61] S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," Proceedings of the Symposium on Visual Languages and Human-Centric Computing, Bayern, Germany, 15-19 September, 2008, pp. 82-85.

[62] L. Breiman, "Random forests," Machine Learning, Springer, vol.45, pp. 5-32, 2001.

[63] The OpenStack Foundation, "The OpenStack project," http://www.openstack.org/

[64] The Eclipse Foundation, "The Eclipse Project," http://www.eclipse.org/

[65] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl, "Evaluating collaborative filtering recommender systems," Transactions on Information Systems, vol.22, No. 1, pp. 5-53, 2004.

[66] S. Yamada, Software Reliability Models: Fundamentals and Applications (in Japanese), JUSE Press, Tokyo, 1994.

[67] Y. Zhou and J. Davis, "OSS reliability model: an empirical approach," Proceedings of the Fifth Workshop on OSS Engineering, Missouri, USA, 17 May, 2005, pp. 67-72.

[68] Y. Shigeru, N. Akio, and K. Mitsuhiro, "A stochastic Differential equation model for software reliability assessment and its goodness-of-fit," International Journal of Reliability and Application, vol. 4, no. 1, pp. 1-11, 2003.

[69] S. Yamada and S. Osaki, "Cost-reliability optimal software release policies for software systems, IEEE Transactions on Reliability," vol. R-34, no. 5, pp. 422-424, 1985.

[70] S. Yamada and S. Osaki, "Optimal software release policies with simultaneous cost and reliability requirements," European Journal of Operational Research, vol. 31, no. 1, pp. 46-51, 1987.

[71] Y. Tamura and S. Yamada, “Optimal Maintenance Problem Based on Maintenance Effort for OSS System Development Project,” Proceedings of the 31st Workshop on Circuits and Systems, Fukuoka, Japan, 17-18 May, 2018, pp. 31-35 (in Japanese).

Publication List of the Author

Books

1. 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.

2. Y. Tamura, H. Sone, and S. Yamada, "Reliability assessment model based on Wiener process considering network environment for edge computing," Reliability and Maintenance Modeling with Optimization, CRC Press Taylor & Francis Group, pp. 215-225, 2023.

3. 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, 2022.

4. Y. Tamura, H. Sone, and S. Yamada, "Stochastic effort estimation for open source project," Recent Advancements in Software Reliability Assurance, CRC Focus Series, Advances in Mathematics and Engineering, CRC Press Taylor & Francis Group, pp. 15-28, 2019.

Journal Papers (Peer-Reviewed)

1. H. Sone, Y. Tamura, and S. Yamada, "Study of effort calculation and estimation in open source projects," International Journal of Reliability, Quality and Safety Engineering, Vol. 30, No. 3, World Scientific, pp. 2350011-1-2350011-13, 2023.

2. 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, May 2022.

3. Y. Tamura, H. Sone, and S. Yamada, "Flexible jump diffusion process models for open source project with application to the optimal maintenance problem," International Journal of Reliability, Quality and Safety Engineering, Vol. 27, No. 6, World Scientific, pp. 2050020-1-2050020-18, 2020.

4. H. Sone, Y. Tamura, and S. Yamada, "Stability assessment method considering fault fixing time in open source project, International Journal of Mathematical," Engineering and Management Sciences, Vol. 5, No. 4, pp. 591-601, 2020.

5. H. Sone, Y. Tamura, and S. Yamada, "Statistical maintenance time estimation based on stochastic differential equation models in OSS development project," Computer Reviews Journal, PURKH, Vol. 5, pp. 126-140, 2019.

6. H. Sone, Y. Tamura, and S. Yamada, "A method of fault fix priority identification for open source project, International Journal of Recent Technology and Engineering," Blue Eyes Intelligence Engineering & Sciences Publication, Vol. 8, No. 4, pp. 2396-2400, 2019.

7. H. Sone, Y. Tamura, and S. Yamada, "Prediction of fault fix time transition in large-scale open source project data," Data, Multidisciplinary Digital Publishing Institute, Switzerland, Vol. 4, No. 3, Multidisciplinary Digital Publishing Institute, Switzerland, DOI: 10.3390/data4030109, pp. 1-12, 2019.

8. Y. Tamura, H. Sone, and S. Yamada, "Productivity assessment based on jump diffusion model considering the effort management for OSS project," International Journal of Reliability, Quality and Safety Engineering, Vol. 26, No. 5, World Scientific, pp. 1950022-1–1950022-22, 2019.

9. Y. Tamura, H. Sone, and S. Yamada, "OSS project stability assessment support tool considering EVM based on Wiener process models," Applied System Innovation, Vol. 2, No. 1, Multidisciplinary Digital Publishing Institute, Switzerland, DOI: 10.3390/asi2010001, pp. 1-12, 2019.

International Conference Papers (Peer-Reviewed)

1. H. Sone, Y. Tamura, and S. Yamada, "Open source project evaluation methodology considering OSS-oriented EVM and number of potential faults," Proceedings of the 28th ISSAT International Conference on Reliability and Quality in Design, August 3-5, 2023, pp. 270-272 (Virtual mode).

2. Y. Tamura, H. Sone, and S. Yamada, “A maintenance effort dependent two dimensional Wiener process model considering OSS network for edge computing,” Proceedings of the 27th ISSAT International Conference on Reliability and Quality in Design, August 4-6, 2022, pp. 91-95 (Virtual mode).

3. H. Sone, Y. Tamura, and S. Yamada, “Quantitative progress evaluation for open source project with application to bullseye chart,” Proceedings of the 24th International Conference on Human-Computer Interaction, 26 June-1 July, 2022, pp. 398-409, (Virtual mode).

4. Y. Tamura, H. Sone, A. Anand, and S. Yamada, “A method of vulnerability assessment based on deep learning and OSS fault big data,” Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, Singapore, December 13-16, 2021, CD-ROM (Reliability and Maintenance Engineering 2, Virtual mode).

5. H. Sone, Y. Tamura, and S. Yamada, “Comparison of stabilities for open source project,” Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, Macau, China, December 13-16, 2021, CD-ROM (Project Management 2, Virtual mode).

6. Y. Tamura, H. Sone, and S. Yamada, “A method of reliability assessment based on Wiener process model for edge computing,” Proceedings of the Reliability and Maintenance Engineering Summit 2021, Nantong, China, September 11-13, 2021, pp. 17-23, (Virtual mode).

7. H. Sone, Y. Tamura, and S. Yamada, “Examination of OSS-oriented EVM considering the progress of fault resolving,” Proceedings of the Reliability and Maintenance Engineering Summit 2021, Nantong, China, September 11-13, 2021, pp. 1-8, (Virtual mode).

8. Y. Tamura, H. Sone, R. Ueki, and S. Yamada, “Optimal effort allocation problem based on the jump diffusion process model for edge computing,” Proceedings of the 26th ISSAT International Conference on Reliability and Quality in Design, August 5-7, 2021, pp. 87-91 (Virtual mode).

9. Y. Tamura, H. Sone, K. Sugisaki, and S. Yamada, “A method of parameter estimation in flexible jump diffusion process models for open source maintenance effort management,”

Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, Macau, China, December 15-18, 2019, CD-ROM (Reliability and Maintenance Engineering 2).

10. H. Sone, Y. Tamura, and S. Yamada, "A method of fault identification considering high fix priority in open source project," Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, Macau, China, December 15-18, 2019, CD-ROM (Project Management 1).

11. Y. Tamura, H. Sone, K. Sugisaki, and S. Yamada, "Optimum maintenance problem based on stochastic differential equations considering 3V model," Proceedings of the 12th Japan-Korea Software Management Symposium, Pusan, Korea, November 29-30, 2018, pp. 1-5.

12. H. Sone, Y. Tamura, and S. Yamada, "A method of fault identification considering fault severity in OSS development," Proceedings of the 12th Japan-Korea Software Management Symposium, Pusan, Korea, November 29-30, 2018, pp. 1-4.

13. H. Sone, Y. Tamura, and S. Yamada, "A method of OSS effort assessment based on deep learning considering GUI design," Proceedings of the 11th Japan-Korea Software Management Symposium, Kansai University, Japan, December 1, 2018, pp. 12-16.

14. Y. Tamura, H. Sone, and S. Yamada, "Earned value analysis tool based on Wiener process model for OSS project," Proceedings of the 11th Japan-Korea Software Management Symposium, Kansai University, Japan, December 1, 2018, pp. 1-5.

15. H. Sone, Y. Tamura, and S. Yamada, "Optimal maintenance problem with earned value requirement for OSS project," Proceedings of the Fourteenth International Conference on Industrial Management, Hangzhou, China, September 12-14, 2018, pp. 253-258.

16. Y. Tamura, H. Sone, K. Sugisaki, and S. Yamada, "Effort analysis of OSS project based on deep learning considering UI/UX design," Proceedings of the IEEE International Conference on Reliability, Infocom Technology and Optimization, Amity University, Uttar Pradesh, Noida, India, pp. 36-41, August 29-31, 2018.

17. Y. Tamura, H. Sone, and S. Yamada, “Earned value analysis and effort optimization based on Wiener process model for OSS project,” Proceedings of 2018 Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modeling (APARM 2018) and 2018 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE 2018), Qingdao, Shandong, China, August 21-24, 2018, pp. 373-378.

Research Reports

1. H. Sone, Y. Tamura, and S. Yamada, “Prediction of fault fixing time transition and practical feasibility study for open source projects,” Kyoto University Reserach Institute for Mathematical Sciences Kôkyûroku “Mathematical Decision Making Under Uncertainty and Related Topics,” No. 2242, pp. 140-145, January 2023 (in Japanese).

2. Y. Tamura, H. Sone, R. Ueki, and S. Yamada, ” Jump diffusion process model considering the cyclic change of communication environ- ment for edge computing,” Kyoto University Reserach Institute for Mathematical Sciences Kôkyûroku “New Developments on Mathematical Decision Making Under Uncertainty,” No. 2220, pp. 184-190, May 2022 (in Japanese).

3. Y. Tamura, H. Sone, K. Sugisaki, and S. Yamada, ” A method of parameter estimation based on deep learning for jump diffusion process model,” Kyoto University Research Institute for Mathematical Sciences Kôkyûroku “Theory and Its Application of Mathematical Decision Making under Uncertainty and Ambiguity,” No. 2158, pp. 47-53, June 2020 (in Japanese).

4. Y. Tamura, H. Sone, and S. Yamada, ” Optimization approach of maintenance effort based on jump diffusion process model for OSS,” Kyoto University Research Institute for Mathematical Sciences Kôkyûroku “Mathematics of Decision Making under Uncertainty and Related Topics,” No. 2126, pp. 174-180, August 2019 (in Japanese).

Research Presentations

1. H. Sone, “Consideration of balancing work and graduate school activities,” Proceedings of 40th National Conference of the Society of Project Management, Hotel Kyocera, March 9-10, 2023, pp. 859-861 (in Japanese).

2. H. Sone, Y. Tamura, and S. Yamada, “Derivation and verification of optimal maintenance time using OSS-oriented EVM,” Proceedings of 34th National Conference of the Society of Project Management, March 11-12, 2023, pp. 107-114, (Virtual mode, in Japanese).

3. H. Sone, Y. Tamura, and S. Yamada, “A study on fault fix time prediction in open source software development,” Proceedings of the 2019 Student Paper Presentation Conference of the Kanto Branch of Japan Industrial Management Association, Chuo University, February 28, 2020, C-1-4, pp. 75-76 (in Japanese).

4. H. Sone, and Y. Tamura, “A study on the recognition propagation in new products using highly sensitive consumer characteristics,” Marketing Analysis Contest 2019, Nomura Research Institute, Ltd., 2019 (in Japanese).

5. H. Sone, and Y. Tamura, “A study on propagation of recognition level for new products,” Proceedings of 2019 National Conference of the Japan Society for Management Information, Shizuoka University, October 19-20, 2019, 1P1-10, pp.112-115 (in Japanese).

6. K. Ohno, and H. Sone, “Study on customer visit prediction at a hairdresser considering the beauty awareness,” Proceedings of 2019 National Conference of the Japan Society for Management Information, Shizuoka University, October 19-20, 2019, 1P1-14, pp.128-131 (in Japanese).

7. H. Sone, Y. Tamura, and S. Yamada, “Fault identification method considering high priority in open source project,” Proceedings of Forum on Information Technology 2019, Okayama University, September 3-5, 2019, pp.141-142 (in Japanese).

8. H. Sone, Y. Tamura, and S. Yamada, “Instant fault identification analysis based on fixing priority for open source project,” Proceedings of 34th National Conference of the Society of Project Management, Hokkaido Citizens Actives Center, August 29-30, 2019, pp. 102-108 (in Japanese).

9. Y. Tamura, H. Sone, K. Sugisaki, and S. Yamada, “Analysis of jump characteristic in flexible jump diffusion process model for OSS project effort estimation,” Proceedings of 34th

National Conference of the Society of Project Management, Hokkaido Citizens Actives Center, August 29-30, 2019, pp. 432-435 (in Japanese).

10. H. Sone, Y. Tamura, and S. Yamada, "Stochastic differential equation modeling for fault fix time prediction in open source projects and its application," Tokai University-Tokyo City University Joint Symposium, Tokyo City University, August 23, 2019 (in Japanese).

11. H. Sone, Y. Tamura, and S. Yamada, "An estimation method of imperfect effort expenditure rate based on stochastic differential equation model for open source project," Proceedings of 33th National Conference of the Society of Project Management, Toyo University, March 14-15, 2019, pp. 325-328 (in Japanese).

12. H. Sone, K. Ohno, K. Sugisaki, and Y. Tamura, "A study on time-shift viewing prediction considering TV viewing attitudes," 2008 Joint Council of Management Science Research Divisions Data Analysis Competition, The University of Tokyo, February 10, 2019 (in Japanese).

13. H. Sone, Y. Tamura, and S. Yamada, "Development of a tool for predicting development effort for open source projects," Proceedings of 2018 Autumn National Conference of the Japan Industrial Management Association, Tokai University, October 27-28, 2018, pp. 22-23 (in Japanese).

14. H. Sone, Y. Tamura, and S. Yamada, "Stochastic differential equation modeling for development effort estimation and its application in open source project," Proceedings of 32th National Conference of the Society of Project Management, Doshisha University, August 30-31, 2018, pp. 439-443 (in Japanese).

Software Tools

1. H. Sone, Y. Tamura, and S. Yamada, EVA Based on SDE Model for OSS Project (Earned Value Analysis Tool Based on Stochastic Differential Equation Model for Open Source Software Project), http://www.tam.eee.yamaguchi-u.ac.jp/

Awards

1. FIT Encouragement Award

   Hironobu Sone, “Fault identification method considering high priority in open source project,” Proceedings of Forum on Information Technology 2019, Okayama University, September 3-5, 2019, pp.141-142 (in Japanese).

2. Student Presentation Award Encouragement Award

   Hironobu Sone, “Instant fault identification analysis based on fixing priority for open source project,” Proceedings of 34th National Conference of the Society of Project Management, Hokkaido Citizens Actives Center, August 29-30, 2019, pp. 102-108 (in Japanese).

3. Student Presentation Award Outstanding Performance Award

   Hironobu Sone, “An estimation method of imperfect effort expenditure rate based on stochastic differential equation model for open source project,” Proceedings of 33th National Conference of the Society of Project Management, Toyo University, March 14-15, 2019, pp. 325-328 (in Japanese).

4. Student Presentation Award Outstanding Performance Award

   Hironobu Sone, “Stochastic differential equation modeling for development effort estimation and its application in open source project,” Proceedings of 32th National Conference of the Society of Project Management, Doshisha University, August 30-31, 2018, pp. 439-443 (in Japanese).

Appendix

Example of development style in Bugzilla

In Chapter 2.5.2, we have discussed bug tracking system. The status transition diagram of details in Bugzilla is shown in Fig. 8.1.

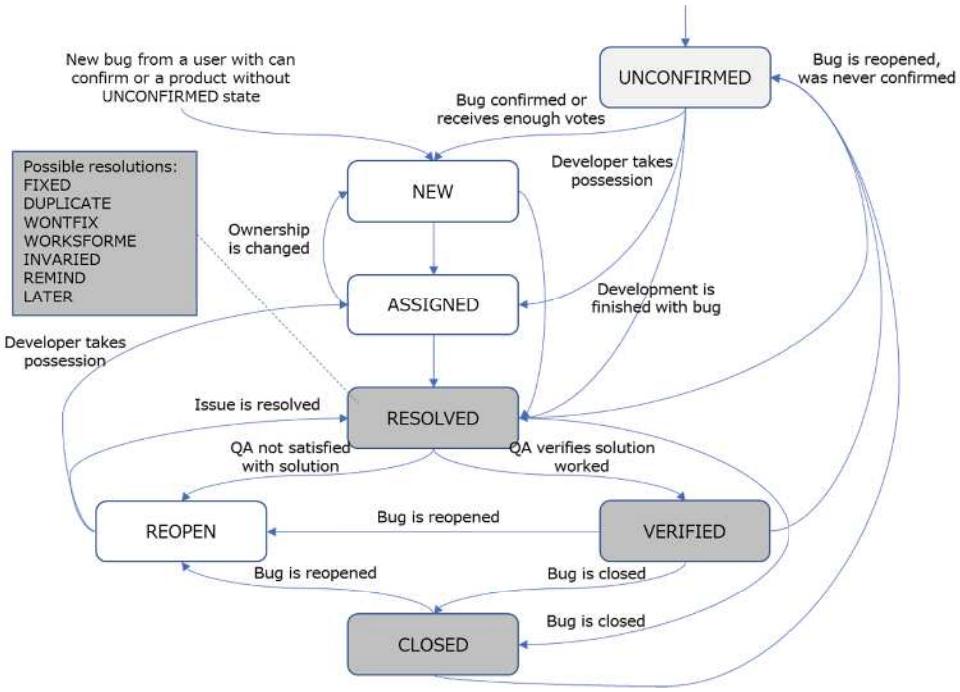

Fig. 8.1: Example of development style in Bugzilla.

Application of the proposed method

For using SRGM, we need to estimate parameters, α , β, c and σ. In this thesis, we have estimated the parameters by the method of maximum likelihood.

The estimation method of unknown parameters α, β, c and σ in Eqs. (4.11) - (4.13) is presented. The joint probability distribution function of the process $\Omega(t)$ as

$$\begin{aligned} &P(t_1,\, y_1;\, t_2,\, y_2;\, \cdots;\, t_K,\, y_K) \\ &\qquad \equiv \Pr[\Omega(t_1) \leq y_1,\, \cdots,\, \Omega(t_K) \leq y_K | \Omega(t_0) = 0]. \end{aligned} \tag{8.1}$$

The probability density of Eq. (8.1) is denoted as

$$p(t_1,\, y_1;\, t_2,\, y_2;\, \cdots;\, t_K,\, y_K) \quad \equiv \quad \frac{\partial^K P(t_1,\, y_1;\, t_2, y_2;\, \cdots;\, t_K,\, y_K)}{\partial y_1 \partial y_2 \cdots \partial y_K}. \tag{8.2}$$

Since $E(t)$ takes on continuous values, the likelihood function, l, for the observed data $(t_k, y_k)(k = 1, 2, \cdots, K)$ is constructed as follows:

$$l = p(t_1,\, y_1;\, t_2,\, y_2;\, \cdots;\, t_K,\, y_K). \tag{8.3}$$

For convenience in mathematical manipulations, the following logarithmic likelihood function is used:

$$L = \log l. \tag{8.4}$$

The maximum-likelihood estimates α^*, β^*, c^*, and σ^* are the values making L in Eq. (8.4) maximize. These can be obtained as the solutions of the following simultaneous likelihood equations [43]:

$$\frac{\partial L}{\partial \alpha} = \frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial c} = \frac{\partial L}{\partial \sigma} = 0. \tag{8.5}$$

Results of Analyses of Various Proposed Models not Described in Individual Chapters

This Appendix contains the figures not included in the main chapters.

OSS-oriented EVM in Chapter 4

In Chapter 4, the parameters maintenance effort, the number of potential faults and number of resolved faults were estimated. In particular, we have not shown all of the model equations in Chapter 4, so the remaining figures are shown below.

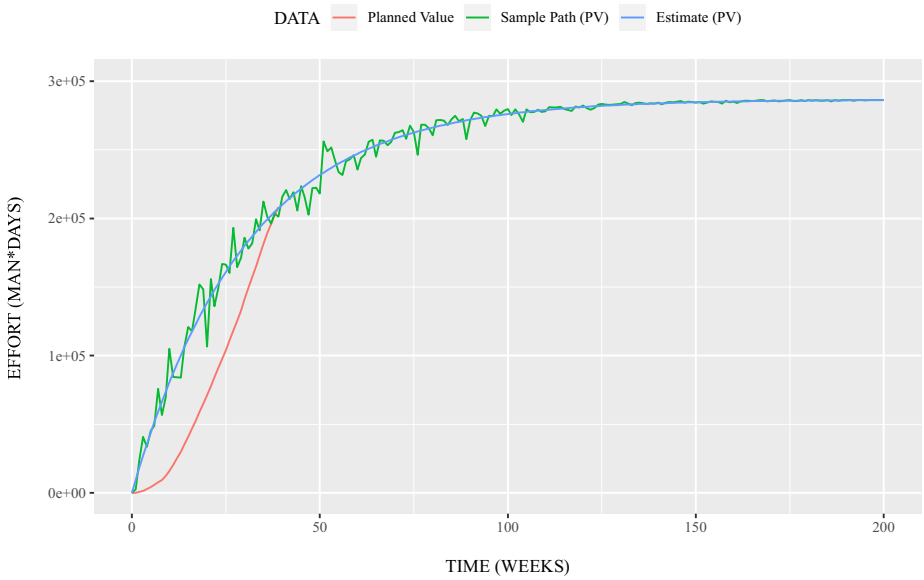

Fig. 8.2: The cumulative maintenance effort expenditures as PV in LibreOffice Ver. 7.2 project by using Eqs. (4.8) and (4.11).

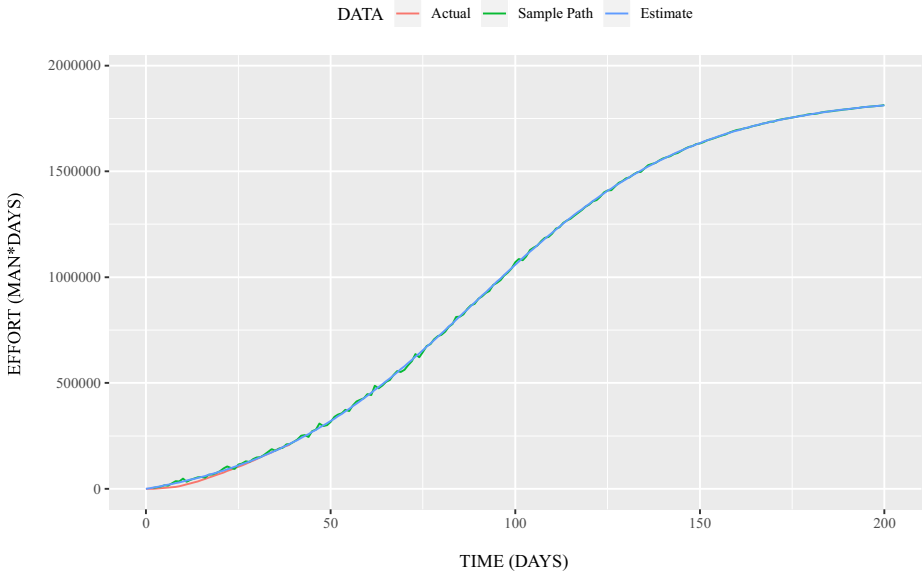

Fig. 8.3: The cumulative maintenance effort expenditures as PV in LibreOffice Ver. 7.2 project by using Eqs. (4.10) and (4.13).

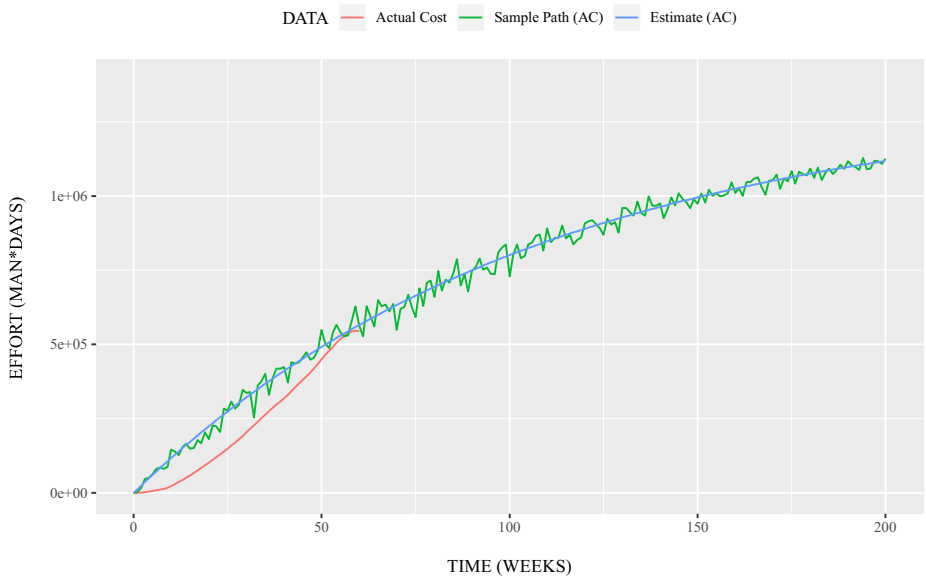

Fig. 8.4: The cumulative maintenance effort expenditures as AC in LibreOffice Ver. 7.2 project by using Eqs. (4.8) and (4.11).

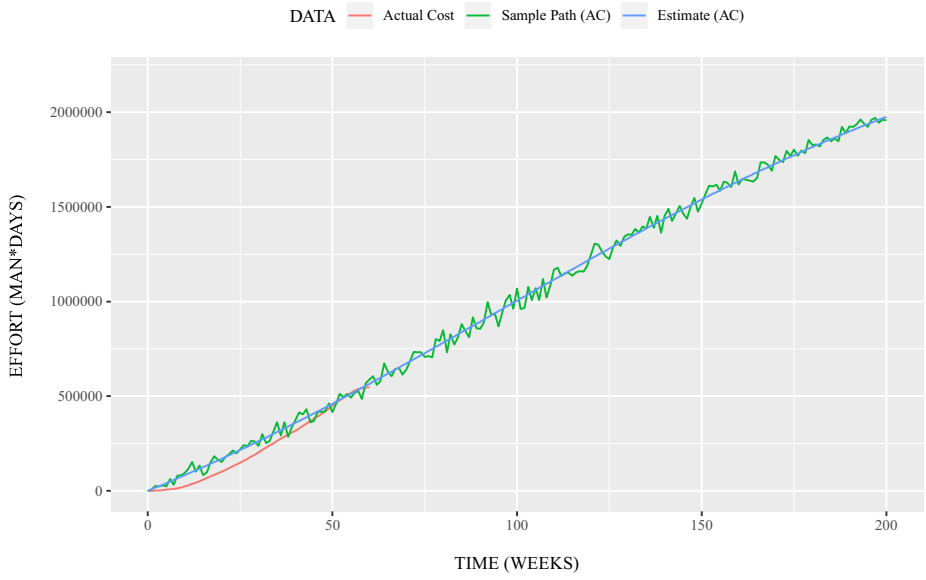

Fig. 8.5: The cumulative maintenance effort expenditures as AC in LibreOffice Ver. 7.2 project by using Eqs. (4.10) and (4.13).

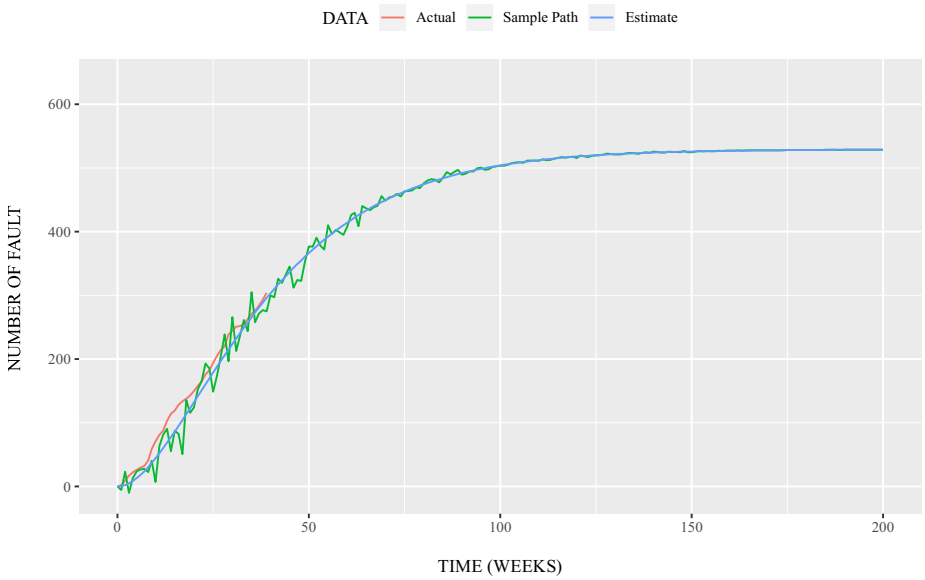

Fig. 8.6: The cumulative estimated number of potential faults in LibreOffice Ver. 7.2 project by usingEqs. (4.9) and (4.12).

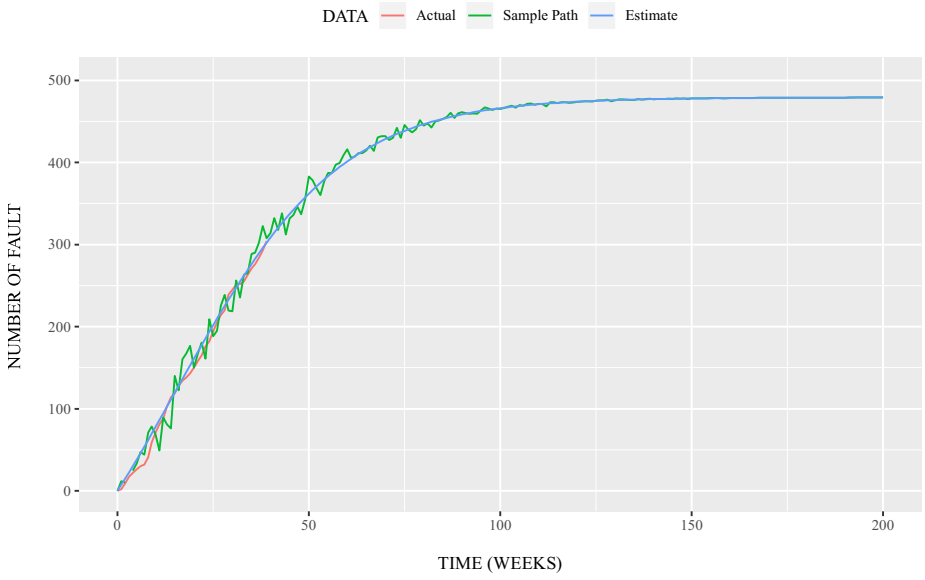

Fig. 8.7: The cumulative estimated number of potential faults in LibreOffice Ver. 7.2 project by using Eqs. (4.10) and (4.13).

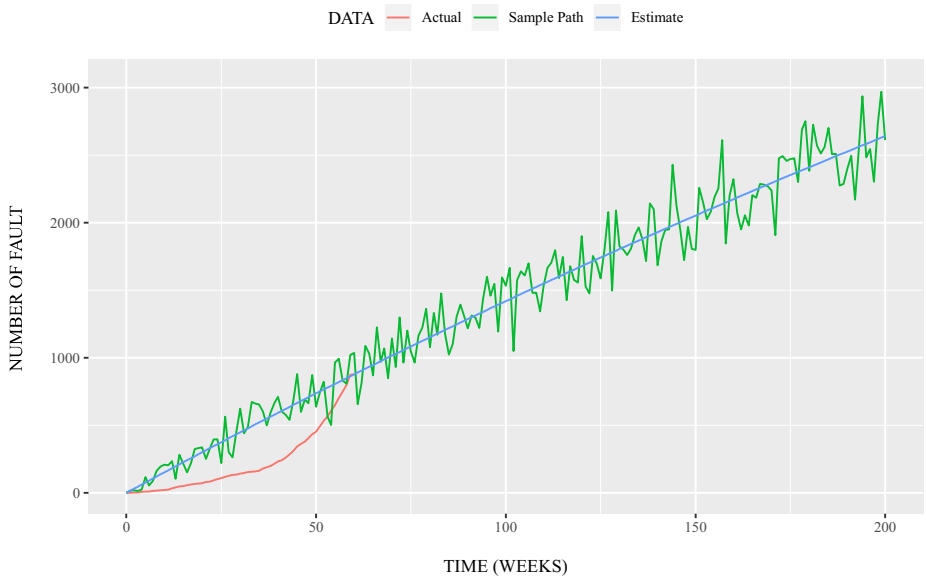

Fig. 8.8: The cumulative estimated number of resolved faults in LibreOffice Ver. 7.2 project by using Eqs. (4.8) and (4.11).

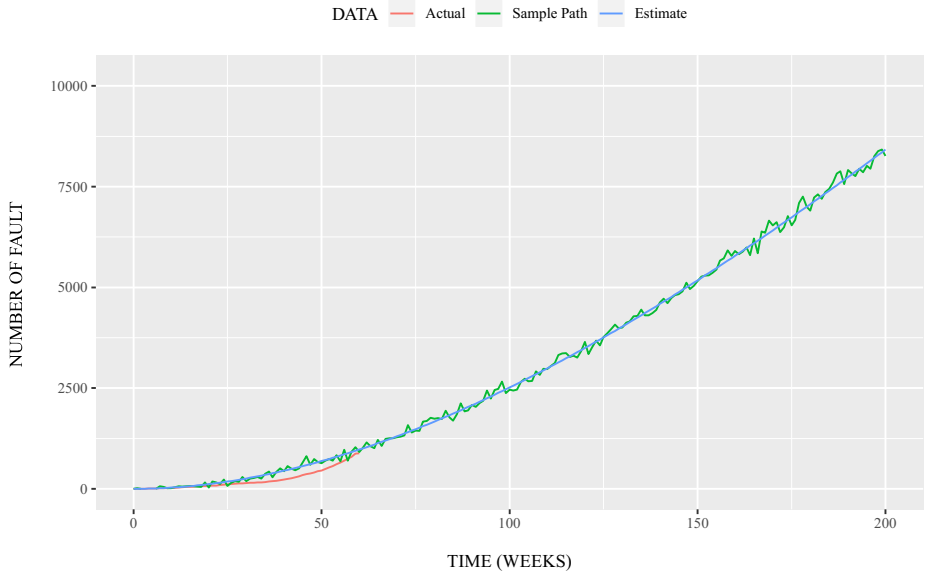

Fig. 8.9: The cumulative estimated number of resolved faults in LibreOffice Ver. 7.2 project by using Eqs. (4.9) and (4.12).

Fault fixing time transition prediction in Chapter 5

In Chapter 5, the parameters maintenance effort, number of potential faults and number of resolved faults were estimated. In particular, we have not shown all of the model equations in Chapter 5, so the remaining figures are shown below.

In Chapter 5, we estimated parameters using three SRGMs to predict the fault fixing time transition. In particular, we have not shown the prediction results for all model equations in Chapter 5. Therefore, the remaining figures are presented below.

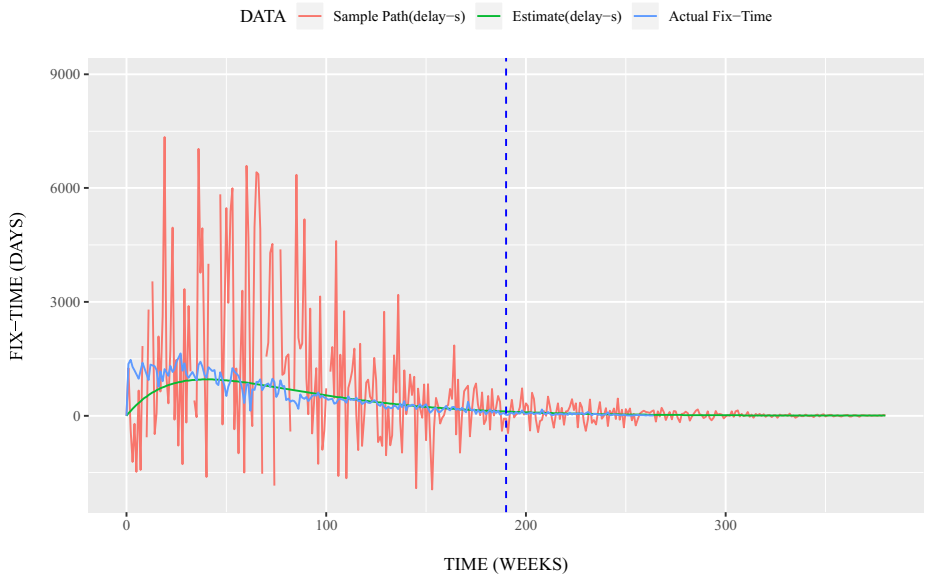

Fig. 8.10: Prediction result of fault fixing time transition in the delayed S-shaped model by using Eq. (5.2).

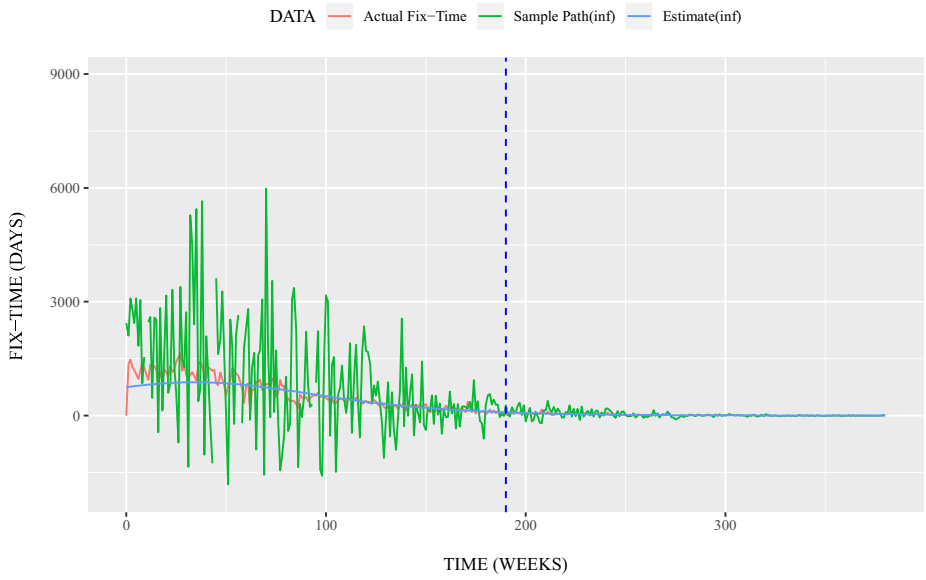

Fig. 8.11: Prediction result of fault fixing time transition in the infection S-shaped model by using Eq. (5.3).

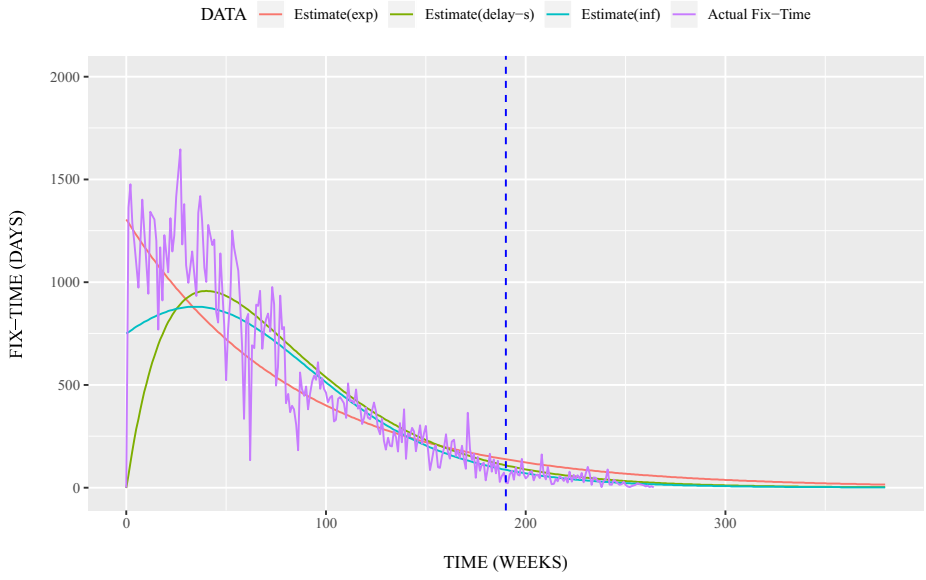

Fig. 8.12: Comparison of measured and expected fault fixing time transition in the three SRGMs.

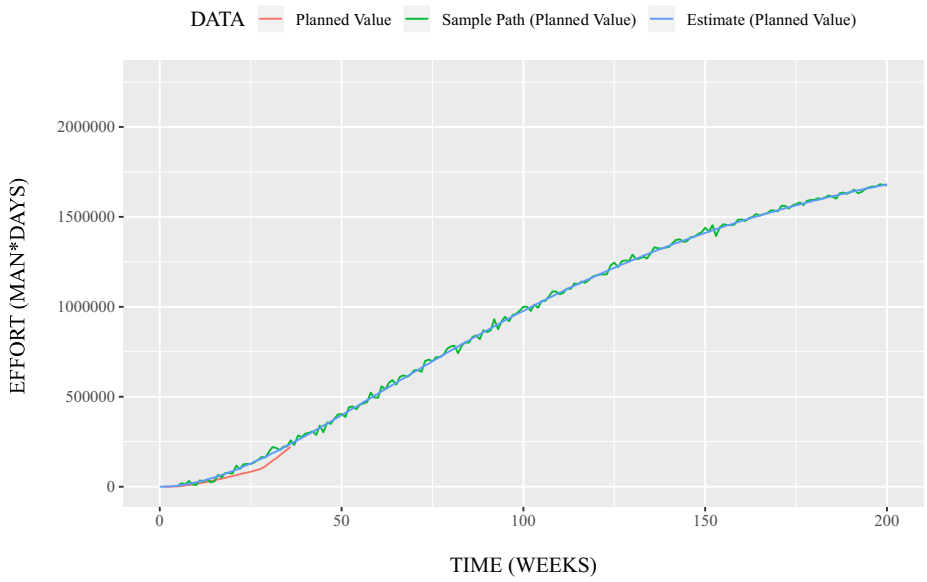

Fig. 8.13: The cumulative maintenance effort expeditures as PV in OpenStack Ver. 16 project by using Eqs. (4.9) and (4.12).

Optimum maintenance problem in Chapter 7

In Chapter 7, we have derived the OSS-oriented EVM for the OpenStack version 16 project. However, we have not shown the prediction results when the appropriate SRGMs are applied for maintenance effort and the number of faults prediction, respectively. In this chapter, we show the prediction results using the SRGMs that were not shown in Chapter 7.

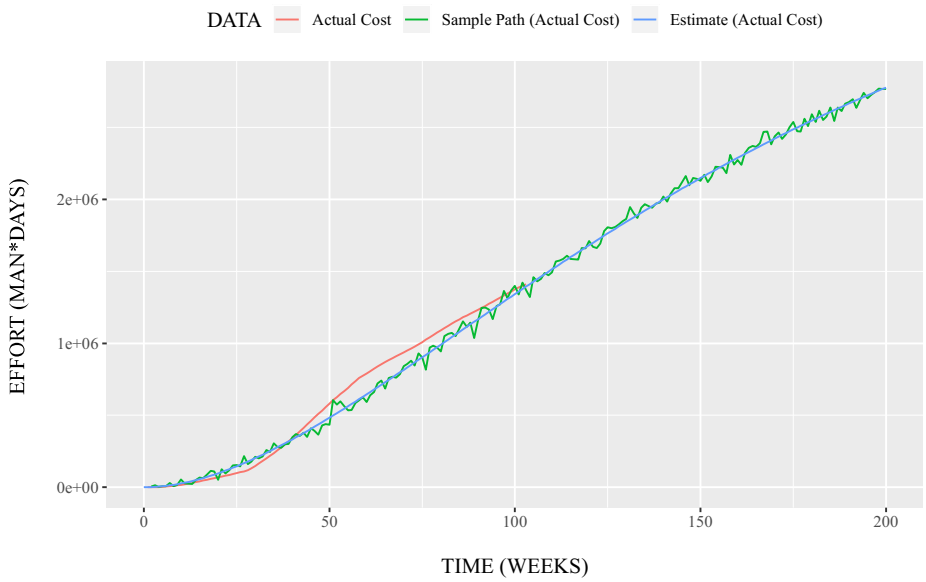

Fig. 8.14: The cumulative maintenance effort expeditures as AC in OpenStack Ver. 16 project by using Eqs. (4.9) and (4.12).

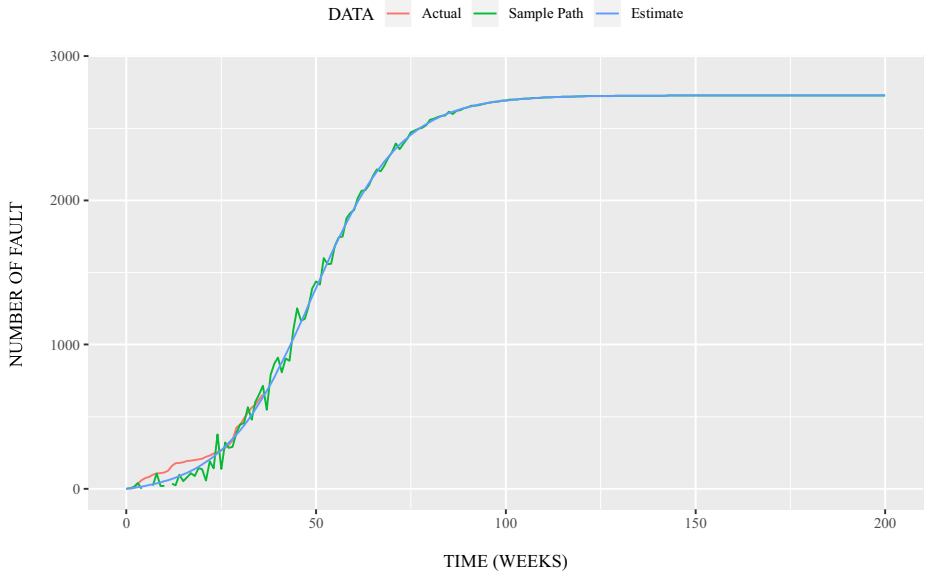

Fig. 8.15: The cumulative estimated number of potential faults in OpenStack Ver. 16 project by using Eqs. (4.10) and (4.13).

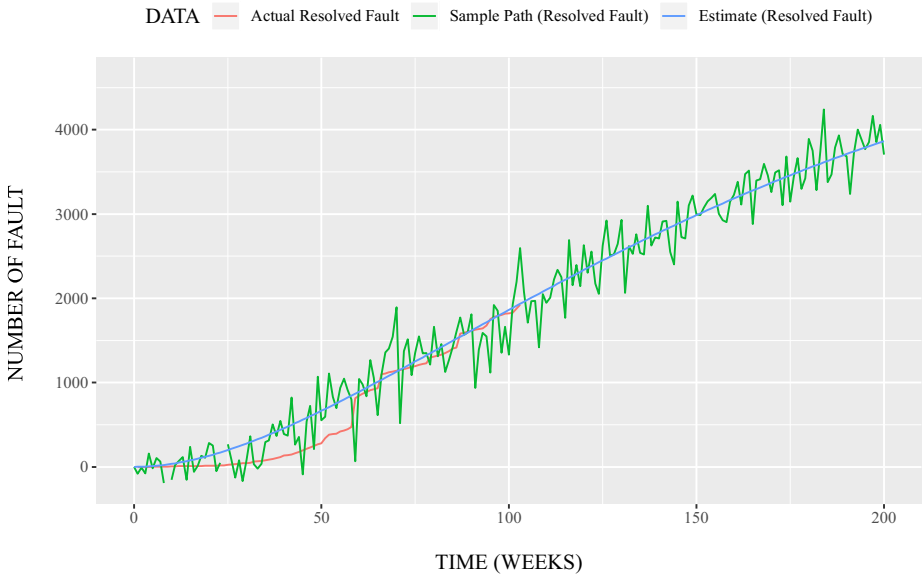

Fig. 8.16: The cumulative estimated number of resolved faults in OpenStack Ver. 16 project by using Eqs. (4.9) and (4.12).