Generalized Software Reliability Model Considering Uncertainty and Dynamics: Theoretical Foundations and Empirical Applications

不確実性と時間変化を考慮した一般化ソフトウェア信頼性モデル：理論的基礎と実証的応用

February, 2017

Waseda University
Graduate School of Fundamental Science and Engineering
Department of Computer Science and Engineering
Research on Reliable Software Engineering

Kiyoshi HONDA
本田 澄

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Software reliability is a critical component of computer system availability. Software reliability growth models (SRGMs) such as the Times Between Failures Model and the Failure Count Model can indicate whether a sufficient number of faults have been removed to release the software. The Failure Count Model is based on counting failures and probability methods. Representatives of this type of model include the Goel-Okumoto non-homogeneous Poisson process (NHPP) Model and the Musa Execution Time Model.

Recent studies by Tamura [47], Yamada [55], Zhang [59], Cai [4], Kamei [23], Schneidewind [43] and Nguyen [33] have attempted to describe the dynamics of developments using a stochastic process. However, they have not evaluated the dynamics with actual datasets. Existing methods assume that each parameter is independent of time, which leads to the inability to account for dynamics. Although this assumption limits the models, it makes the models solvable by mathematical methods. For example, the NHPP model has two parameters (i.e., the total number of faults and the fault detection rate), which are independent of time because the NHPP model equations cannot be solved if these values have time dependencies. Although Okamura et al. proposed a multi-factor software reliability model framework that can deal with the metrics observed in the testing phase such as test coverage, the number of test workers, etc. [38], their method employs a logistic regression to analyze the relationships between the probability that an event occurs and environmental factors, including the person hours. They evaluated their method with five datasets, including the number of test cases, cumulative test cases, and increment of code coverage, etc., but they did not evaluate their method with person hours. On the other hand, I hypothesize that if

several developers suddenly join a project in which a SRGM is applied, the development environment suddenly changes, impacting the parameters, especially the SRGM parameters, since sudden changes should be treated with time-dependent parameters.

1.1 Background

Software reliability is important to release software. Several approaches have been proposed to measure reliability. One is to model fault growth, which is a type of SRGM. Because software development includes numerous uncertainties and dynamics regarding development processes and circumstances, this section explains SRGMs, their uncertainties and dynamics as well as provides a motivating example.

1.2 Software Reliability Growth Model (SRGM)

SRGMs are used to predict the numbers of faults. Because it is important to estimate the costs to develop software, several methods exist in software engineering. One method is called the function point method, which can estimate the person hours to analyze functions software development. The SRGM is one such method to estimate the person hours to release software by quantifying the person hours by the numbers of faults and the numbers of predicted faults. However, existing SRGMs often inaccurately estimate the numbers of faults. In my experience, existing SRGMs estimate between 0.5 to 10 times the actual numbers of faults.

I hypothesized that existing models estimate the wrong number of faults because they are unable to model the fault detection process. In this thesis, I propose a new model. Similar to SRGMs that assess software reliability quantitatively from fault data observed in the testing phase, my approach is also based on the fault counting [8] model.

Many software reliability models have been proposed, but the most popular is the NHPP model. In this study, NHPP models are compared with my method, which I named the Generalized Software Reliability Model (GSRM) using development data containing the number of faults detected in a given time. My GSRM is formulated by counting the number of faults detected in a given time assuming that fault detection is based on a stochastic process,

whereas the NHPP model assumes that the stochastic process governing the relationship between fault detection and a given time interval is a Poisson process. In actual developments, fault counting predicts the number of remaining faults.

In the NHPP model, the probability of detecting n faults is described by

$$Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!} \exp\{-H(t)\} \tag{1.1}$$

$N(t)$ is the number of faults detected by time t and $H(t)$ is the expected cumulative number of detected faults [54]. Assuming that the total number of faults is constant at $N_{\max}$, the number of detected faults at a unit time is proportional to the number of remaining faults. These assumptions yield the following equation

$$\frac{dH(t)}{dt} = c(N_{\max} - H(t)) \tag{1.2}$$

where c is a proportionality constant. The solution of this equation is

$$H(t) = N_{\max}(1 - \exp(-ct)) \tag{1.3}$$

This model is called the exponential software reliability growth model, and was originally proposed by Goel and Okumoto [9]. Yamada et al. derived the delayed S-shaped software reliability growth (S-shaped) model from equation (1.3) with a fault isolation process [57]. Equation (1.3) can be rewritten into the delayed S-shaped SRGM using $H_{\mathrm{S}}(t)$, which is the expected cumulative number of faults detected in the S-shaped model, as

$$H_{\mathrm{S}}(t) = N_{\max}\{1 - (1 + ct)\exp(-ct)\} \tag{1.4}$$

Researchers have proposed other software reliability growth models by applying several failure time distributions in NHPP models. Table 1.1 presents the standard SRGMs and their corresponding failure time distributions.

In early works, researchers employed the maximum likelihood estimation to estimate the model parameters. However, today the least square method is also employed to estimate model parameters [40]. In this thesis, I only take the trend curves of these SRGMs and estimate these curves in terms of the non-linear least squares method.

Table 1.1: Standard SRGMs and their failure time distributions.

Model name	Failure time distribution	References
Exponential model	Exponential distribution	[9]
S-Shape model	Gamma distribution	[57]
Logistic model	Truncated logistic distribution	[34]
Gompertz model	Truncated extreme value max distribution	[36]
Weibull model	Log extreme value min distribution	[8] [36]

Chapter 2

Overview of This Thesis

I consider that developers and researchers have tried to remove uncertainties in software engineering, for a long time. For example, in requirement engineering several researchers proposed methods to model the uncertainties and risks. In implementation, Kernighan et al. surveyed the elements of good style about programming, taking best practices, and mentioned that "Write clearly - don't be too clever" in "The elements of programming style"[25]. This means that Kernighan et al. were aware of the problem that bad programming style gave the readers of programming code a difficulty to understand. If there is a difficulty to understand in the programming code, it makes other programmers who try to improve the program to fail to understand or to write error code. Thanks to these studies, nowadays almost all companies employ the naming convention, which is one of the programming style, in its development team because a lot of developers develop a product by cooperating with other developers and needs rules to understand the code easily and clearly for other developers. In short, such naming convention and coding standards are ones of the challenges to remove the uncertainties between developers in implementation.

Figure 2.1 shows the over view of related work about uncertainty of software developments as far as I know. The X axis represents phases of software development. The Y axis represents modeling uncertainty and controlling uncertainty. The areas represent the research areas of several researchers or research group. The area (3) is my research area and the theme of this thesis.

About (1) in figure 2.1, several researchers tried to treat the uncertainties in requirements and operations. Wallace et al. studied the risk [49] and analyzed the software project risks to reduce the incidence of failure [49]. They

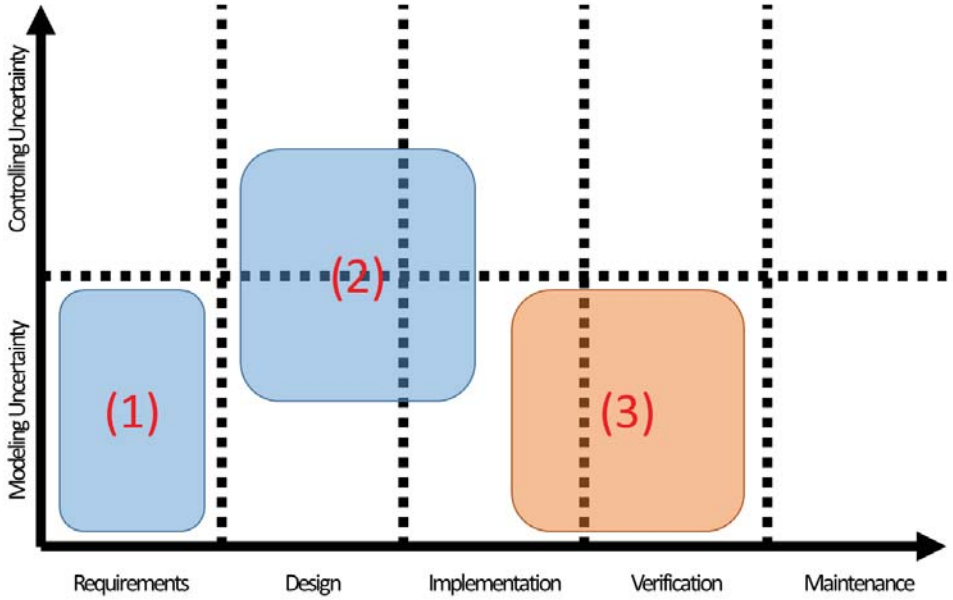


Figure 2.1: Overview of related work.

mentioned that software projects have six dimensions: Team Risk, Organizational Environment Risk, Requirements Risk, Planning and Control Risk, User Risk, and Complexity Risk. They emphasized that Organizational Environment Risk and Requirements Risk are due to risks and uncertainties. Goseva-Popstojanova and Kamavaram studied the uncertainties in requirements and operations by component-based software engineering [22] [10]. In [22], they analyzed the uncertainties of operational profiles and component reliability by calculating the conditional entropy of each component. In [10], they analyzed the uncertainties of the operational profiles and the component reliability by Monte Carlo simulations. These analytical methods focus on the requirements and operations.

About (2) in figure 2.1, N. Ubayashi et al. proposed a method to describe uncertainty in design models or programs by specifying uncertain architectural points [48]. They modeled architectural uncertainty in design and coding phases. Additionally, T. Fukamachi et al. proposed a modularization mechanism for uncertainty [7]. Their approach to deal with uncertainty modeled uncertainty about requirement by describing in design and code as a method and tried to control uncertainty by adding or deleting the

uncertainty method.

About (3) in figure 2.1 , I proposed a method to quantify uncertainty through the software reliability growth model [16] and evaluated my method by using actual data sets [31]. Additionally, I proposed a method to detect unexpected situations in actual situations [14]. My research area is focusing on the process of detecting faults in implementation and verification. And I was able to quantify the uncertainties through the fault detection.

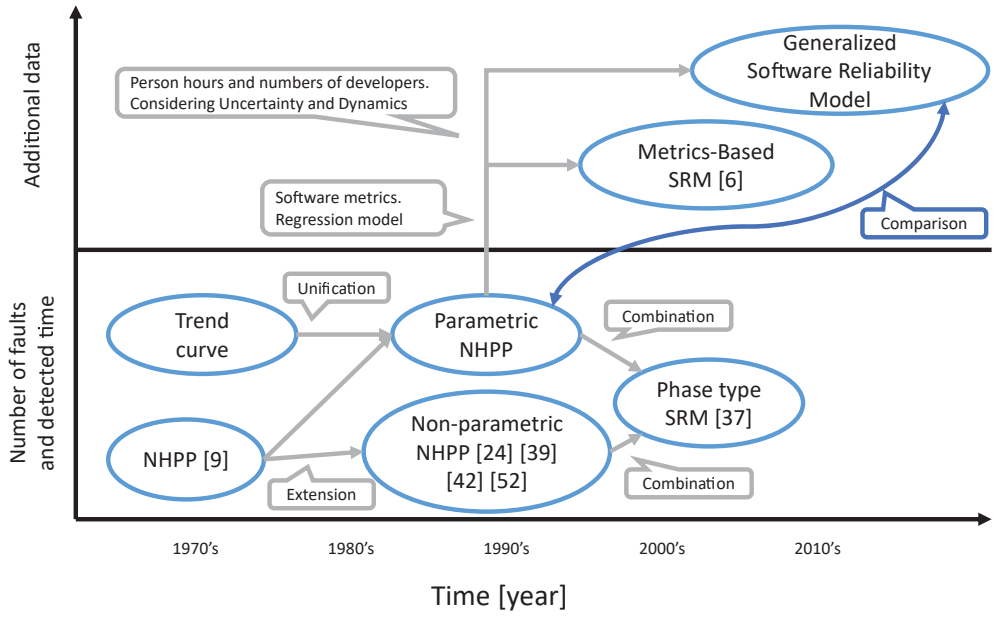


Figure 2.2: Overview of software reliability models.

Additionally, I summarized the relations between several famous software reliability models and my proposed model, which is named as a Generalized Software Reliability Model (GSRM) in figure 2.2. The X axis represents the time when the models had been proposed. The Y axis represents the amount of data which each model can treat.

At the beginning of the studies of the software reliability models, there were trend curve model and Non-Homogeneous Poisson Process model in 1970's. NHPP model was proposed by Goel et al. [9]. The famous models of trend curve models are Logistic model and Gompertz model.

As the researches of software reliability models made progress, it became clear that NHPP based models could describe the same model as trend curve

models. Moreover, several researchers extended NHPP based models to non-parametric NHPP models which employed neural network [24] [42] and support vector machine (SVM) [52] [39]. The existing NHPP based models were classified as parametric NHPP models.

In 2000's, Okamura et al. proposed phase type software reliability models which combined non-parametric NHPP models and parametric NHPP model [37]. Fujii et al. proposed metrics-based software reliability models which used both the number of faults and software metrics and employed regression models to treat the software metrics [6].

The GSRM is based on the parametric NHPP model and can directory treat the person hours and numbers of developers without regression models. Moreover, the GSRM can model the uncertainties and dynamics of developments.

In this thesis, I compared the GSRM and existing parametric NHPP models by using actual data sets. Additionally, I did not compare the GSRM with the metrics-based SRM because I did not have the datasets which contains the faults data and metrics data. In order to compare the GSRM with the metrics-based SRM, I should prepare the dataset which contains the faults data and metrics data and the numbers of developers.

Chapter 3

Generalized Software Reliability Model Considering Uncertainty and Dynamics: Theoretical Foundations

3.1 Introduction to This Chapter

Previous studies only use linear stochastic differential equations, but my research indicates that nonlinear stochastic differential equations more realistically model actual situations. These studies and SRGMs treat and test several datasets only in the given situation (e.g., within the same company or organization). In short, existing models are tested and applied to the situation from which the datasets are obtained, and are evaluated from different domains. For example, one work evaluated several SRGMs with automotive software datasets, while another assessed two SRGMs with an army system dataset. Rana et al. studied four software projects from the automotive sector and concluded two statistic SRGMs perform better than other SRGMs [40]. On the other hand, Goel et al. investigated two stochastic SRGMs with a U.S. Navy project dataset and concluded their model provides a plausible description [9]. These studies did not evaluate existing SRGMs with other domains.

Herein I propose a model called the Generalized Software Reliability Model (GSRM) [16] to describe several development situations that involve

random factors (e.g., team skills and development environment) to estimate the time that a development will end [15]. Additionally, I have predicted the release times of open source software (OSS) using GSRM [18] and agile development [50]. Moreover, I have applied GSRM and SRGMs to company's datasets [15] [14] [13] [12]. Due to random factors, the GSRM in each situation has an upper and lower limit, suggesting that a GSRM can predict the maximum and minimum number of faults. I formulate the upper and lower limit equations for three development situations with approximations in order to treat these equations easily and predict the number of faults in several ranges. I evaluate and test GSRM and other models using datasets from different organizations and circumstances. GSRM can quantify uncertainties that are influenced by random factors (e.g., team skills and development environments), which is important to more accurately model the growth of software reliability and to optimize development teams or environments.

This study aims to answer the following research questions:

1. RQ1: Can GSRM be applied to several development situations?
2. RQ2: Is GSRM an improvement over other models (e.g., NHPP) in describing the growth of software reliability under different situations?

My contributions are as follows:

1. I propose a software reliability model applicable to nine development situations, which is 12% more precise than existing models for recent datasets.
2. Three approximate equations about three uncertainty types in my software reliability model.
3. An evaluation with actual datasets confirms that my software reliability model can classify development situations.

To evaluate my model, I simulate nine types of development situations using the Monte Carlo method. To simplify the application of my model, I divide the situations related to uncertainty situations into three types and derive an approximation equation for each. Finally, I apply the approximation equations to four projects from two different organizations and classify these projects into three uncertainty types.

The rest of this Chapter is organized as follows. Section 2 describes my model as a generalized software reliability model and summarizes the types

of developments depending on dynamics and uncertainties. In addition, I derive three equations, which depend on the uncertainties. Section 3 applies GSRM to four projects using two datasets and evaluates whether GSRM can accommodate different situations. Moreover, I compare GSRM with NHPP models by focusing on how closely the models can simulate actual data. Section 4 discusses related work, while Section 5 provides the conclusion.

3.2 Background

Software reliability is important to release software. Several approaches have been proposed to measure reliability. One is to model faults growth, which is a type of SRGM. Because software development includes numerous uncertainties and dynamics regarding development processes and circumstances, this section explains SRGM, its uncertainties and dynamics as well as provides a motivating example.

In this Chapter, I compare GSRM with these two models. Equation (1.3) results in an exponentially shaped software reliability graph. However, a real software reliability graph typically follows a logistic curve or a Gompertz curve [56], which is more complex. Therefore, I propose a new model that can express either a logistic curve or an exponentially shaped curve for use in actual developments.

3.2.1 Uncertainty and Dynamics

Software development projects have uncertainties and risks. Wallace et al. analyzed the software project risks to reduce the incidence of failure [49]. They mentioned that software projects have six dimensions: Team Risk, Organizational Environment Risk, Requirements Risk, Planning and Control Risk, User Risk, and Complexity Risk. They emphasized that Organizational Environment Risk and Requirements Risk are due to risks and uncertainties. However, existing software reliability growth models do not contain these uncertainty elements. On the other hand, several SRGMs treat limited time-dependent assumptions. Yamada et al. proposed an extend NHPP model related to the test-domain dependence [58]. The test-domain dependent model includes the notion that a tester's skills improve in degrees. Although growth of skills is a time-dependent additional assumption to the NHPP model, this

model does not correspond to dynamic changes (e.g., changes in or the number of team members).

3.2.2 Motivating Example

Existing studies describe usages and evaluate models, and almost all compare models and try to suggest an improvement. Existing models can be applied to similar situations, but they cannot be applied different situations like other domains or scales. For example, Figure. 3.1 shows the impact of applying two existing models to different datasets in references [9] and [45], which belong to different organizations. The crosses for "Dataset 1" represent the actual data in reference [9] and the crosses for "Dataset 2 release 2" represent one project of the actual data in reference [45]. The dashed lines represent the exponential model fitted to the actual data. The dotted-dashed lines represent the S-shaped model fitted to the actual data. These results indicate that the fitness of the S-shaped model is greater than the exponential model in "Dataset 1." However, the fitness of the exponential model is greater than the S-shaped model in "Dataset 2 release 2." In other words, the fitness of a model depends on the situation and is not universal. Therefore, I propose a model that is suitable for different situations.

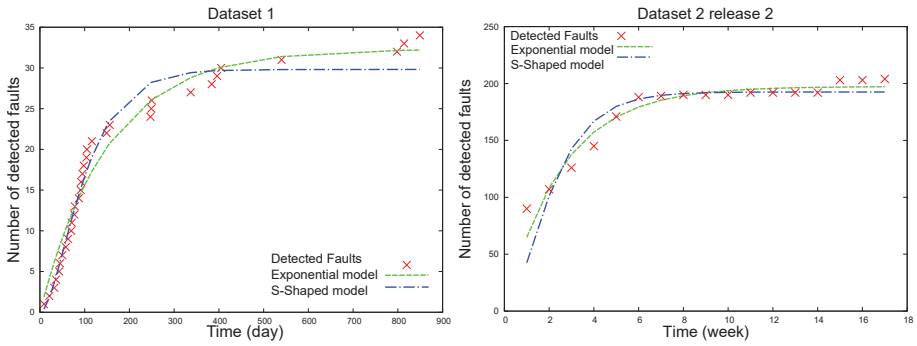


Figure 3.1: Cumulative number of detected faults for Dataset 1 (left) and Dataset 2 release 2 (right) as a function of elapsed time. In the legends, Detected faults, exponential model, and S-shaped model represent the actual data, the fit using the exponential model, and the fit using the S-shaped model, respectively.

3.3 Generalized Software Reliability Model (GSRM)

For my software reliability model [16], I extend a nonlinear differential equation that describes the fault content as a logistic curve as an Ito type stochastic differential equation. In this Chapter I derive three equations corresponding to three types of uncertainties [16]. Moreover, I apply and evaluate the three equations by using actual datasets.

In my previous paper [16], I proposed a simple GSRM equation, which only simulated several development situations that involved random factors. In [16], I could not analyze actual datasets. In this Chapter, I extend the simple GSRM equation to apply it to actual datasets. The equation is divided into three uncertainty types: late, constant, and early types. Using clearly defined equations, I apply GSRMs to actual datasets and obtain the upper and lower limits in each situation due to random factors, which mean that the GSRM can predict the maximum and minimum number of faults. Additionally, I evaluate and test my GSRMs and other models using datasets from different organizations and circumstances. The GSRM can quantify uncertainties that are influenced by random factors, which is important to more accurately model the growth of software reliability and to optimize development teams or environments.

I start with the logistic differential equation, which is expressed as

$$\frac{dN(t)}{dt} = N(t)(a + bN(t)) \tag{3.1}$$

$N(t)$ is the number of detected faults by time t, a defines the growth rate, and b is the carrying capacity. If $b = 0$, then the solutions are exponential functions. Because the numerous uncertainties and dynamic changes prevent actual developments from correctly obeying equation (3.1), it should be extended into a stochastic differential equation. I assume that such dynamic elements are time dependent and contain uncertainties. These elements are expressed using a. The time dependence of a can be used to describe situations such as an improved development skills and increased growth rate. The uncertainty of a can describe parameters such as the variability of development members and the environment. The growth of software is analyzed with an emphasis on the testing phase by simulating the number of detected faults. I assume that software development has the following properties:

1. The total number of faults is constant.

2. The number of faults that can be found depends on time.

3. The number of faults that can be found contains uncertainty, which can be simulated with Gaussian white noise.

The first assumption means that the total number of faults is finite and can be treated as a boundary condition. This assumption implies that correcting faults does not create new ones. Indeed, many SRGMs assume that the total number of faults is constant [28]. Several researchers have proposed SRGMs that can treat infinite faults [30]. I assume that the debugging process creates new faults. I suppose that I can use one model for fault creation and fault detection in the debugging process.

The second assumption means that the ability to detect the faults varies; the ability depends on time because the number of developers changes in recent developments, and the number of developers affects the ability to detect the faults. Researchers have proposed several time dependent models whose situations are limited such as the error detection per time increases with the progress of software testing [57]. Hou et al. proposed a SRGM upon considering two learning curves (the exponential learning curve and the S-shaped learning curve) [19]. These existing models assume that the number of developers does not change and the time dependent parameters are a specific model like the exponential learning curve. In contrast, my model does not depend on a specific model.

The third assumption means that uncertainties in the development process affect the ability to detect faults. This assumption is due to the fact that actual datasets have non-constant detection rates or seem to be independent of time. Analyzing actual faults in datasets, I observed several sudden increases in the number of detected faults. Then I modeled the uncertainty, which affects the ability to detect faults as Gaussian white noise that is a simple but commonly used noise. To the best of my knowledge, other SRGMs do not treat such uncertainties that occur in the development. By analyzing the uncertainties for each development, I can understand how a development progresses and predict the progress with a concrete tolerance.

3.3.1 Modeling Uncertainties and Dynamics

Considering these properties, equation (3.1) can be extended to an Ito type stochastic differential equation with $a(t) = \alpha(t) + \sigma(t)dw(t)$, which is expressed as

$$dN(t) = (\alpha(t) + \beta N(t))N(t)dt + N(t)\sigma(t)dw(t) \tag{3.2}$$

$N(t)$ is the number of detected faults by time t, $\alpha(t)+\sigma(t)dw(t)$ is the differential of the number of detected faults per unit time, $\gamma(t) = N(t)\sigma(t)dw(t)$ is the uncertainty term, $\sigma(t)$ is the dispersion, and β is the nonlinear carrying capacity term. This equation has two significant terms, $\alpha(t)$ and $\sigma(t)dw(t)$; $\alpha(t)$ affects the end point of development and $\sigma(t)dw(t)$ affects the growth curve through uncertainties. Thus, my model treats both uncertainties and dynamics. However, uncertainties cannot be treated directly because they depend on the cause. My approach treats the uncertainties through a fault detecting process.

3.3.2 Uncertainties

In particular, equation (3.2) indicates that the stochastic term is dependent on $N(t)$, which means that the uncertainties depend on the number of detected faults. According to equation (3.2), as the number of detected faults increases, the stochastic term has a greater effect on the number of detected faults. Such a situation corresponds to software development with late uncertainty.

I compare three different types of dependencies of $\gamma(t)$ on $N(t)$:

- The late uncertainty type is where $\gamma(t) = N(t)\sigma dw(t)$.
- The constant uncertainty type is where $\gamma(t)$ is independent of $N(t)$: $\gamma(t) = \sigma dw(t)$.
- The early uncertainty type is where $\gamma(t)$ depends on the inverse of$N(t)$: $\gamma(t) = 1/N(t)\sigma dw(t)$.

As $\alpha(t)$ and the coefficient of $dw(t)$ are varied, models are simulated using equation (3.2). Table 3.1 summarizes the types of $\alpha(t)$, the coefficient of $dw(t)$, and the corresponding situations. Using GSRM, the type must be chosen for Table 3.1 to calculate the parameters using past data. In development, faults are detected and debugged. The detected faults are counted

and used to predict when the project will end. Projects contain a lot of uncertainty elements, and the predicted development period is almost never long enough. GSRM can describe the uncertainty of applied development and calculate the uncertainty of fault detection.

I describe the uncertainty as $\sigma(t)dw(t)$, which is basically Gaussian white noise obtained from past data. The uncertainty is difficult to calculate using equation (3.1), so I assume some limits and obtain $\sigma(t)dw(t)$ quantitatively. I start by defining $a(t)$ in terms of

$$a(t) = \alpha(t) + \sigma(t)dw(t) \tag{3.3}$$

Equation (3.1) cannot be solved due to the time dependence of a, as shown in equation (3.3). Therefore, I assume that a is time independent with an added term δ, which is small. This assumption allows equation (3.1) to be solved. These three uncertainty types can be rewritten as

$$dN(t) = (\alpha(t) + \beta N(t))N(t)dt + N(t)\delta dw(t) \tag{3.4}$$

$$dN(t) = (\alpha(t) + \beta N(t))N(t)dt + \delta dw(t) \tag{3.5}$$

$$dN(t) = (\alpha(t) + \beta N(t))N(t)dt + \frac{1}{N(t)}\delta dw(t) \tag{3.6}$$

Each GSRM model is derived from one of these three types of uncertainty.

3.3.3 Simulations

Using these equations for GSRM, I simulated these nine cases. Figure 3.2 models and plots these nine cases. For each column in Table 3.1, the difference between each model is the parameter $\alpha(t)$. In **Model 1-1**, **Model 2-1** and **Model 3-1**, which are based on **Model 1-1**, $a_2 = a_1$, $a_3 = 2a_1$ and $t_1 = t_{max}/2$ in **Model 2-1**, and $\alpha_3(t) = a_1 t$ in **Model 3-1**. $\alpha(t)$'s are set in the same manner along all columns (i.e., $\alpha(t)$ is the same along each row in Table 3.1). For **Model 1-1**, **Model 2-1** and **Model 3-1**, the effect of uncertainty over time, $\gamma(t) = N(t)\sigma dw(t)$ increases. The situation in **Model 2-1** corresponds to the number of development team members doubling at time t_1. The situation corresponding to **Model 3-1** is that the members' skills improvement over time, effectively doubling the manpower by the time t_{max}. For Model 1-2, Model 2-2, and Model 3-2, the effect of uncertainty $\gamma(t) = \sigma dw(t)$ is constant. For **Model 1-3**, **Model 2-3** and **Model 3-3**, the effect of uncertainty $\gamma(t) = \sigma dw(t)/N(t)$ decreases over time.

Table 3.1: Combinations of dynamics as characterized by $\alpha(t)$ and $\gamma(t)$. $\alpha(t)$ and $\gamma(t)$ indicate the number of detected faults per unit time and the uncertainty term, respectively.

	$\gamma(t) = N(t)\sigma dw(t)$	$\gamma(t) = \sigma dw(t)$	$\gamma(t) = \sigma dw(t)/N(t)$
$\alpha_1(t) = a_1$(const.)	The number of detected faults per unit time is constant, but the uncertainty increases near the end. This model is similar to a logistic curve. (**Model 1-1**)	The number of detected faults per unit time and uncertainty are constant. (**Model 1-2)**	The number of detected faults per unit time is constant, but the uncertainty decreases over time (e.g., the team matures over time). (**Model 1-3**)
$\alpha_2(t) = a_2$ $(t < t_1)$ $\alpha_2(t) = a_3$ $(t \geq t_1)$	The number of detected faults per unit time changes at t_1, and the uncertainty increases near the end (e.g., new members join the project at time t_1). (**Model 2-1)**	The number of detected faults per unit time changes at t_1, but the uncertainty is constant. (**Model 2-2)**	The number of detected faults per unit time changes at t_1, but the uncertainty decreases over time. (**Model 2-3)**
$\alpha_3(t) \propto t$	The number of detected faults per unit time and the uncertainty increase near the end (e.g., increasing manpower with time). (**Model 3-1)**	The number of detected faults per unit time increases, but the uncertainty is constant. (**Model 3-2)**	The number of detected faults per unit time increases, but the uncertainty decreases over time. (**Model 3-3)**

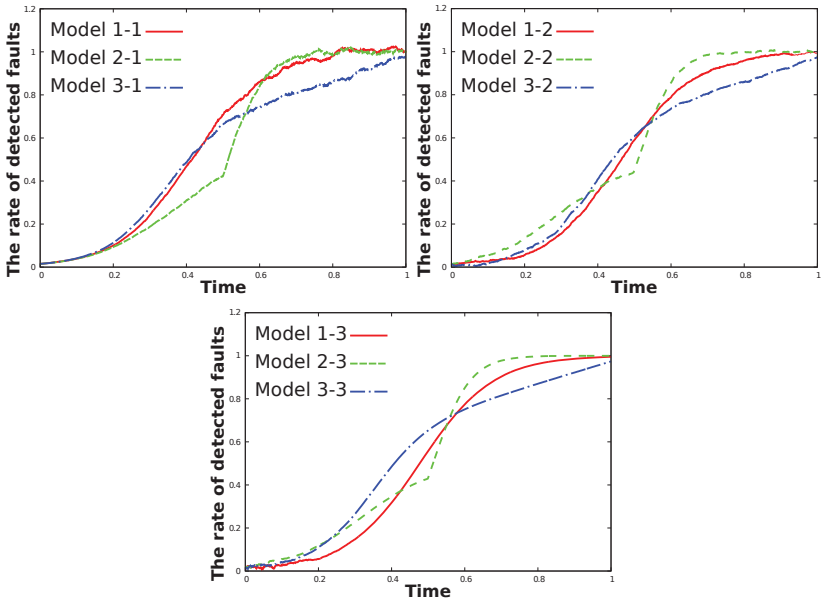


Figure 3.2: Ratio of the cumulative number of detected faults at time t versus the total number of detected faults for the entire project where the. x-axis represents time in arbitrary units. 1 corresponds to t_{max} and $0:5$ to t_1. In **Model 1-1**, **Model 1-2** and **Model 1-3**, the number of detected faults per unit time is constant. In **Model 2-1**, **Model 2-2** and **Model 2-3**, the number of detected faults per unit time changes at t_1. In **Model 3-1**, **Model 3-2** and **Model 3-3**, the number of detected faults per unit time increases.

The purpose of the simulations is to confirm that my approach can assess software reliability under dynamic changes and uncertainties in development as well as adapt the models to produce appropriate results. I used a Monte Carlo method to examine these models. Figure 3.2 shows the effects of uncertainties and dynamics.

3.3.4 Formulation

In sections 3.3.2 and 3.3.3, I simulated several stochastic differential equations. The results show that random factors affect the number of predicted faults. I assumed that the effects of random factors obey Gaussian white noise, which follows the dispersion of equation (3.2). Thus, the number of detected faults has an upper and a lower limit. In this section, I discuss the three types of uncertainty (late uncertainty, constant uncertainty, and early uncertainty) and formulate the stochastic differential equations to approximate differential equations without stochastic variables in order to treat these stochastic differential equations easily and predict the number of faults in several ranges. These approximated differential equations have simple solutions, which contain uncertainty values and will give the upper and lower limits.

Late uncertainty type

The late uncertainty type means that the uncertainties in the development circumstances increase as the development progresses. Consider the following equation

$$dN(t) = (\alpha + \beta N(t))N(t)dt + N(t)\delta dw(t) \tag{3.4}$$

If $\delta dw(t)$ is a constant value (i.e., time independent), equation (3.4) can be written as

$$dN(t) = (\alpha + \delta + bN(t))N(t)dt \tag{3.7}$$

The solution to equation (3.7) is given by

$$N(t) = \frac{N_{\max}}{1 + b\exp\{-(\alpha + \delta)t\}} \tag{3.8}$$

This equation is a logistic equation where δ is the origin of the uncertainty. $\alpha+\delta$ is the gradient. The sign of δ can be positive or negative. If δ is negative (positive), the gradient of the equation is small (large). The sign of δ gives

the limitation of the uncertainty. If δ is negative (positive), the growth of the graph provides a lower (upper) limit. I calculate the upper and lower limits of some projects and plot them in the next section. δ is determined as

$$\delta_i = -\frac{1}{t_i} \ln\left\{ \frac{1}{b}\left(\frac{N_{\max}}{N_i} - 1 \right) \right\} - \alpha \tag{3.9}$$

The subscript i indicates the data is for the ith fault detected at t_i. i differs from the approximate value at t_i. Finally, I obtain the average and variance of δ. I can construct the equation of SRGM from the average and variance of δ to simulate the projects and to predict when they will end by using δ and its distribution, which is Gaussian white noise [15]. I assume that the detected faults obey equation (3.8) and that the detection rate has a time-independent uncertainty δ. This assumption yields the following upper and lower limits

$$N_+(t) = \frac{N_{\max}}{1 + b\exp\{-(\alpha+\delta)t\}} \tag{3.10}$$

$$N_-(t) = \frac{N_{\max}}{1 + b\exp\{-(\alpha-\delta)t\}} \tag{3.11}$$

$N_+(t)$ means the upper limit about $N(t)$. If the development proceeds via a favorable situation, the number of detected faults will obey equation (**??**). $N_-(t)$ denotes the lower limit about $N(t)$. If the development proceeds via an unfavorable situation, the number of detected faults will obey equation (3.11).

Constant uncertainty type

In this section, the constant uncertainty type where the development circumstances have the uncertainties that are independent of time is discussed. Consider the following equation

$$dN(t) = (\alpha + \beta N(t))N(t)dt + \delta dw(t) \tag{3.5}$$

If $\delta \to 0$, equation (3.5) is derived as follows

$$dN(t) = (\alpha + \beta N(t))N(t)dt \tag{3.12}$$

This equation can be solved as

$$N(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} \tag{3.13}$$

This is the base equation for the other types. However if $\delta << 1$ and $\delta \neq 0$, I should consider the δ term. The entire term, which is related with δ, is δdw. Because dw is Gaussian white noise, the value of δ can be calculated from the actual data and base equation (3.13). I write the number of the actual detected faults as N_i and the number of predicted faults as $N(t)$. d_i, which is the difference between the actual and predicted data, is defined using these two terms as

$$d_i = N_i - N(t_i) \tag{3.14}$$

δ is expressed as

$$\delta = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(d_i - \bar{d})^2} \tag{3.15}$$

δ means the standard deviation. $\bar{d}$ is the average of d_i. m represents the total number of detected faults. Finally, the upper limits and lower limits of the constant uncertainty type equations are expressed as

$$N_+(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} + \delta \tag{3.16}$$

$$N_-(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} - \delta \tag{3.17}$$

$N_+(t)$ is the upper limits about $N(t)$, while $N_-(t)$ means the lower limits about $N(t)$. These limit equations mean that when the number of detected faults becomes large, they are not affected by the δ uncertainty term.

Early uncertainty type

In this section, I discuss the early uncertainty type, which means that the uncertainties in the development circumstances decrease in the late stage of the development. Consider equation (3.6), which is given by

$$dN(t) = (\alpha(t) + \beta N(t))N(t)dt + \delta\frac{1}{N(t)}dw(t) \tag{3.6}$$

If $\delta \to 0$, then it can be rewritten as

$$dN(t) = (\alpha + \beta N(t))N(t)dt \tag{3.18}$$

This equation can be solved as

$$N(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} \tag{3.13}$$

This is the base equation of the other types. However if $\delta << 1$ and $\delta \neq 0$, then this should be regarded as the δ term. The whole of the uncertainty term is $\delta dw(t)/N(t)$. Because $dw(t)$ is Gaussian white noise, the value of $\delta dw(t)/N(t)$ can be calculated from the actual data and base equation (3.13). The concrete equation is obtained from equation (3.13) as

$$\frac{\delta\{1 + b\exp(-\alpha t)\}}{N_{\max}} \tag{3.19}$$

N_i is the number of actual faults and $N(t)$ is the number of predicted faults. d_i, which is the difference between the actual data and predicted data, can be expressed using these two terms as

$$d_i = N_i - N(t_i) \tag{3.20}$$

Equation (24) shows the difference between the actual data and the predicted data. Using this equation and the relation as $\delta dw/N(t)$, δ is expressed as

$$\delta = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\{N(t_i)d_i\}^2} \tag{3.21}$$

This value δ denotes the standard deviation. Additionally, dw means Gaussian white noise. Finally the equation for the early type uncertainty is written by adding equation (3.19) to equation (3.13)

$$N_+(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} + \delta\frac{1 + b\exp(-\alpha t)}{N_{\max}} \tag{3.22}$$

$$N_-(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} - \delta\frac{1 + b\exp(-\alpha t)}{N_{\max}} \tag{3.23}$$

$N_+(t)$ means the upper limits about $N(t)$, and $N_-(t)$ means the lower limits about $N(t)$. These equations indicate that when the number of detected faults is large, the uncertainty term has a negligible influence on the number of detected faults.

3.4 Evaluation

In this section, I apply the three types of uncertainties to four projects from two datasets to determine which uncertainty types are suitable for each dataset. The first development dataset is from reference [9]. The second development dataset is from reference [45]. Moreover, I compared GSRM with other SRGMs using two datasets.

3.4.1 Evaluation design and results

I evaluate GSRM by applying to two datasets [9] [45] and comparing my model with other models. First, I applied the three uncertainty types of GSRM to these datasets to determine which uncertainty type is suitable for the development situation. To analyze the suitability, I evaluated the coverages between their upper and lower limits. The areas between these limits contain several actual data points. If the number of contained actual data points of one type is larger than that of the other type, I considered the uncertainty type is suitable for the dataset. Second, I compared GSRM with other models with respect to the accuracy of applying the models to these datasets. I adopted the residual sum of square (RSS) and Akaike Information Criteria (AIC). RSS means the differences between the model and the data. AIC means the differences between the model and the data considering the number of parameters in the model. If the RSS and AIC values are small, the model fits with the data. Because these values indicate model fitness, the fitness of different models can be compared. My evaluation uses the datasets in Table 3.4. DS 1 [9] was obtained in 1979. DS 2-1, 2-2, and 2-3 [45] were obtained in 2000. DS 2-1, 2-2, and 2-3 were developed with several testing guidelines, including the fault-prone method, and using previous projects' data [46].

Design

This development dataset is from reference [9] and is written by Goel and Okukmoto. The data are originally from the U.S. Navy Fleet Computer Programming Center and consist of the errors in software development (Table 3.2). The second development dataset is from reference [45] and is written by Stringfellow et al. The data come from three releases of a large medical record system, which consists of 188 software components (Table 3.3). The

Table 3.2: Dataset 1 (DS 1). Each row contains the total number of faults detected over the corresponding number of days.

Error No.	1	2	3	4	5	6	7	8	9	10	11	12
Time (days)	9	21	32	36	43	45	50	58	63	70	71	77
Error No.	13	14	15	16	17	18	19	20	21	22	23	24
Time (days)	78	87	91	92	95	98	104	105	116	149	156	247
Error No.	25	26	27	28	29	30	31	32	33	34		
Time (days)	249	250	337	384	396	405	540	798	814	849		

Table 3.3: Dataset 2(DS 2-1, 2-2, 2-3). Each row contains the total number of faults detected over the corresponding number of weeks.

Weeks	Release 1 (DS 2-1)	Release 2 (DS 2-2)	Release 3 (DS 2-3)
1	28	90	9
2	29	107	14
3	29	126	21
4	29	145	28
5	29	171	53
6	37	188	56
7	63	189	58
8	92	190	63
9	116	190	70
10	125	190	75
11	139	192	76
12	152	192	76
13	164	192	77
14	164	192	
15	165	203	
16	168	203	
17	170	204	
18	176		

Table 3.4: Details of each dataset.

Dataset	Year	Term	Faults	Citation	Explain
DS 1	1979	849 days	34	[9]	US Navy
DS 2-1	2000	18 weeks	176	[45]	Pharmacy (release 1)
DS 2-2	2000	17 weeks	204	[45]	Pharmacy (release 2)
DS 2-3	2000	13 weeks	77	[45]	Pharmacy (release 3)

data contain the cumulative number of faults and their detected times for the three different releases of the software program. I applied GSRM to these datasets and classified the datasets based on the uncertainty type by comparing the covered actual data points between the upper and lower limits. To compare with NHPP models (NHPP and S-shaped), I adopted the time independent model of GSRM because these datasets do not have person month data. The models were evaluated using RSS and AIC where small values indicate that the model has a sufficient fit.

Table 3.4 shows that these datasets contain data from the 1970s and the 2000s, which were produced by waterfall developments. The 2000s projects were developed with the fault prone method, which is a modern approach [46]. Therefore, the 2000s projects were developed via more modern processes than the 1970s project.

Results

I evaluated the adoptions for the three uncertainty types and compared the GSRMs with NHPP models through the two datasets. For each uncertainty type, I calculated the upper limit and the lower limit as well as its suitability. Figures 3.3 – 3.6 and Table 3.5 show the results. I compared the GSRMs with the NHPP models (e.g., the exponential model and S-shaped model) through RSS and AIC. As mentioned above, small RSS and AIC values indicate a good model fitness. Figure 3.7 and Table 3.6 show the results.

Type Selection

I calculated δ and the upper and lower limits. Figures 3.3 – 3.6 plot the results where the x-axis represents time and the y-axis represents the number of detected faults. Solid, dashed, and dotted-dashed lines represent values

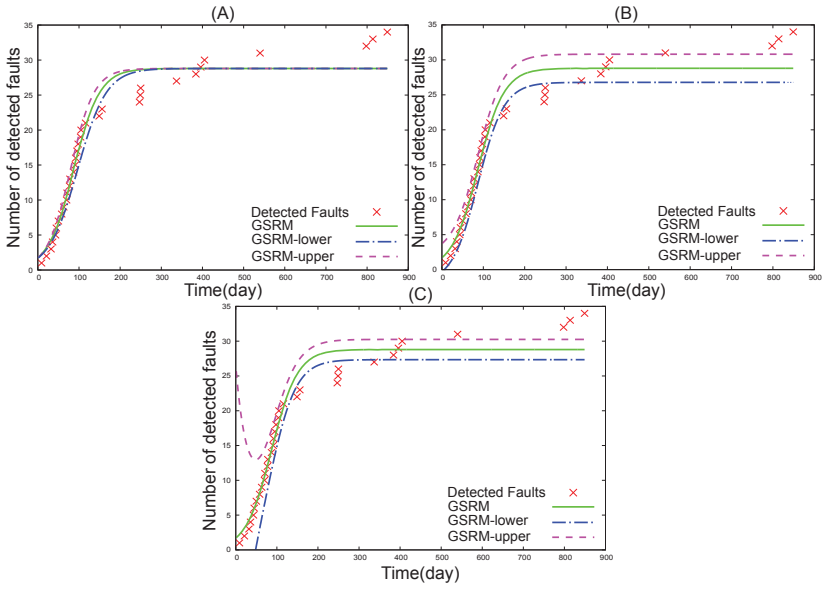


Figure 3.3: Cumulative number of detected faults for the entire project of DS 1 plotted against the elapsed number of days. In the legend, Detected faults, GSRM, GSRM-upper, and GSRM-lower represent the actual data, the fit using GSRM, the predicted upper limit, and the predicted lower limit, respectively. (A) the late uncertainty type, (B) the constant uncertainty type, and (C) the early uncertainty type.

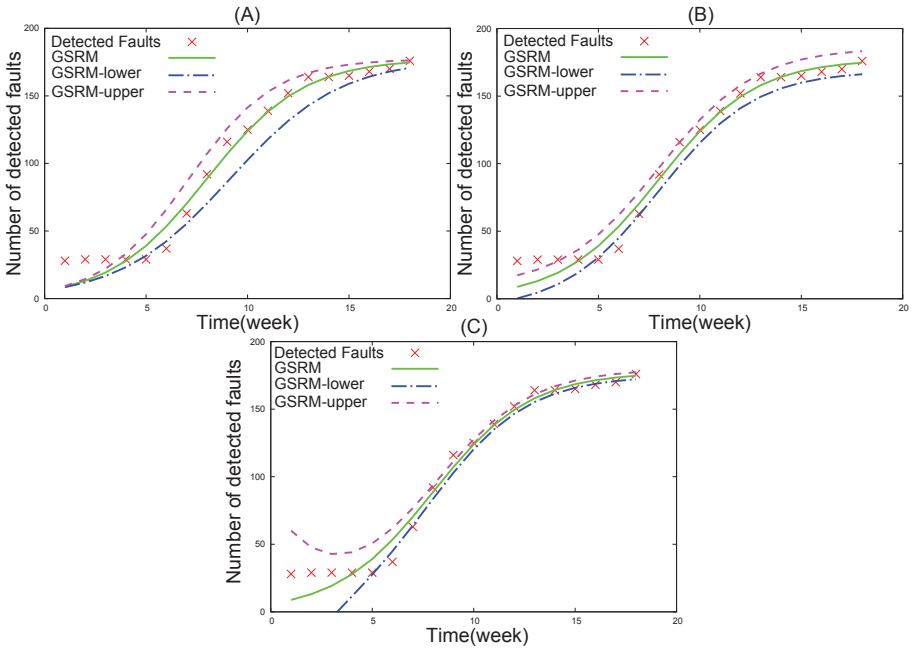


Figure 3.4: Cumulative number of detected faults for the entire project of DS 2-1 plotted against the elapsed number of weeks. Legends and the titles are the same as Fig. 3.3.

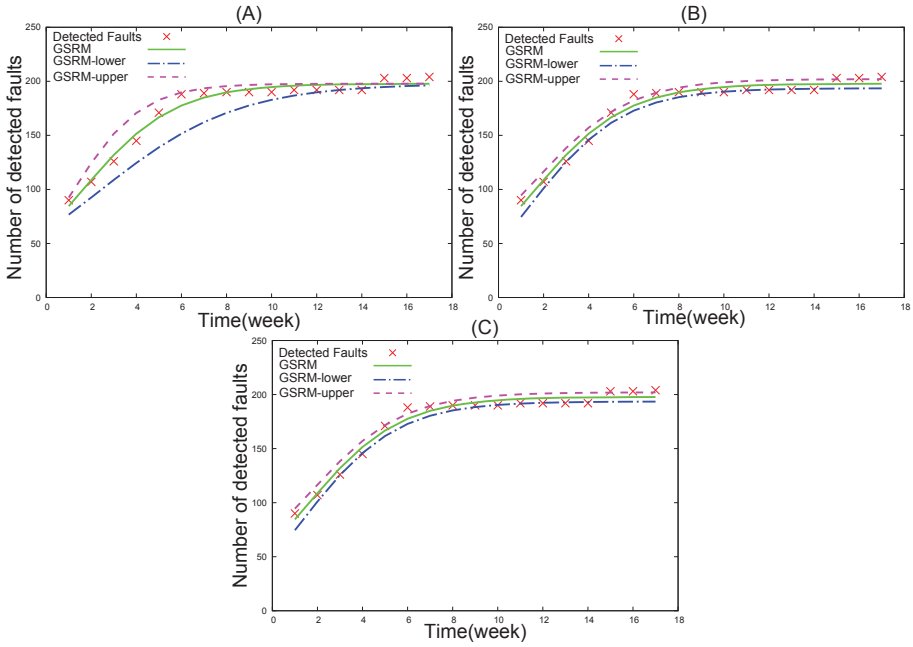


Figure 3.5: Cumulative number of detected faults for the entire project of DS 2-2 plotted against the elapsed number of weeks. Legends and the titles are the same as Fig. 3.3.

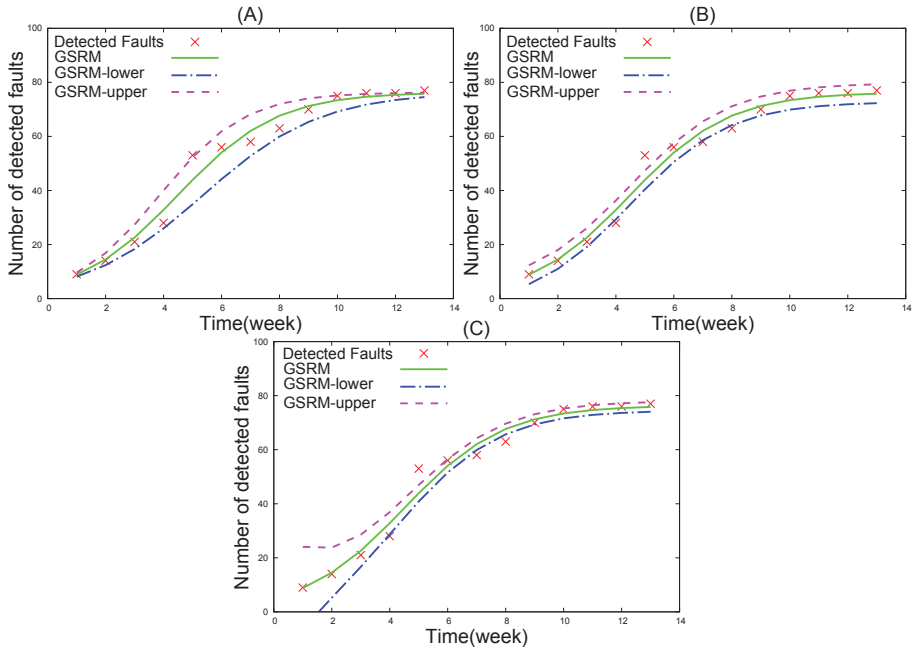


Figure 3.6: Cumulative number of detected faults for the entire project of DS 2-3 plotted against the elapsed number of weeks. Legends and titles are the same as Fig. 3.3.

calculated with GSRM, upper limits, and lower limits, respectively. The crosses indicate the actual data in reference [9] and [45]. These results confirm that almost all of the real data points are contained within the calculated upper and lower limits. I quantitatively evaluated which type is suitable for

Table 3.5: Selection of three uncertainty types for Datasets 1 and 2.

	Late type			Constant type			Early type		
	Coverage	Area	Rate	Coverage	Area	Rate	Coverage	Area	Rate
DS 1	16	62.9	0.254	25	136.9	0.183	24	271.3	0.088
DS 2-1	13	350.6	0.037	12	307.1	0.039	11	374.9	0.029
DS 2-2	13	343.5	0.038	9	179.5	0.050	8	174.3	0.046
DS 2-3	10	114.3	0.087	9	90.9	0.099	9	106.6	0.084

each datasets by calculating the Coverage, Area, and Rate for each dataset. Coverage means the number of actual data points between the upper and lower limits. Area, which denotes the area surrounded by the upper and lower models, is calculated by integrating the upper and lower models. Rate, which indicates the coverage rate of the type, represents the Coverage divided by Area. Coverage is the most important value since a model with a large Coverage can cover a lot of actual data points. If one type has a larger Coverage than the other types, it is more suitable for the dataset. However, if the types have equal Coverage, then the type with the largest Rate is most suitable for the dataset.

Table 3.5 shows the results. The constant type of uncertainty is suitable for dataset 1 because it has the largest coverage (Figure. 3.3 and Table 3.5). This is reasonable since the interval of detecting faults is random as the development proceeds in dataset 1, indicating that uncertainty events randomly occur. The results of release 1, 2, and 3 indicate that the late type is suitable for dataset 2 because it has the largest coverage (Figures. 3.4 – 3.6). This is consistent with Tables 3.3 and 3.5 where more faults are detected at the end of development for dataset 2, indicating that the development has issues up until each release.

In these datasets, I could not verify the results by interviewing the development teams that produced these datasets due to the age of the datasets. Therefore, I requested that a Japanese IT company, which employs about 5000 people, use GSRMs and evaluate the results, including the uncertainties, from 2013 to 2015. I collected the fault datasets from the two projects in the company. Additionally, the two managers of the two development teams

evaluated the results. Then I asked the two managers about the GSRM results and whether they agreed with the uncertainty type according to the GSRMs. They responded that the uncertainty type indicated by the results is consistent with their thoughts.

Comparison

The NHPP models are well known reliability models. In this section, I discuss the differences between GSRM and NHPP models using the actual development data in a given situation when the growth rate is time independent. The reason for this limitation is because the NHPP model cannot be applied to time-dependent situations. Figure 3.7 shows the results, where the crosses represent the actual data in reference [9] and [45]. The solid, dashed, and dotted-dashed lines represent the GSRM fitted to the actual data, the exponential model fitted to the actual data, and the S-shaped model fitted to the actual data, respectively. The parameters are calculated by R [1], which is a language and environment for statistical computing and graphics. DS 1 and DS 2-2 are the same projects as shown Figure. 3.1, which is the motivating example. To verify that GSRM is a better model than the other models, I calculate the RSSs and the AICs from these models and the development data (Table 3.6).

The RSSs and AICs of the GSRM are lower in DS 2-1, DS 2-2 and DS 2-3 than those of the exponential and S-shaped models, suggesting that GSRM is a better approximation than the NHPP models because GSRM has more flexibility due to the nonlinear term. Especially, the GSRM has the breaking term such as β, which causes the model to converge on the given values near the actual data points.

3.4.2 Discussion

I compared GSRM with other NHPP models using two datasets in order to assess whether GSRM can describe the data precisely. Additionally, I applied the three uncertainty types to the datasets to verify whether GSRM can adapt to different situations.

[1] The R Project for Statistical Computing `http://www.r-project.org/`

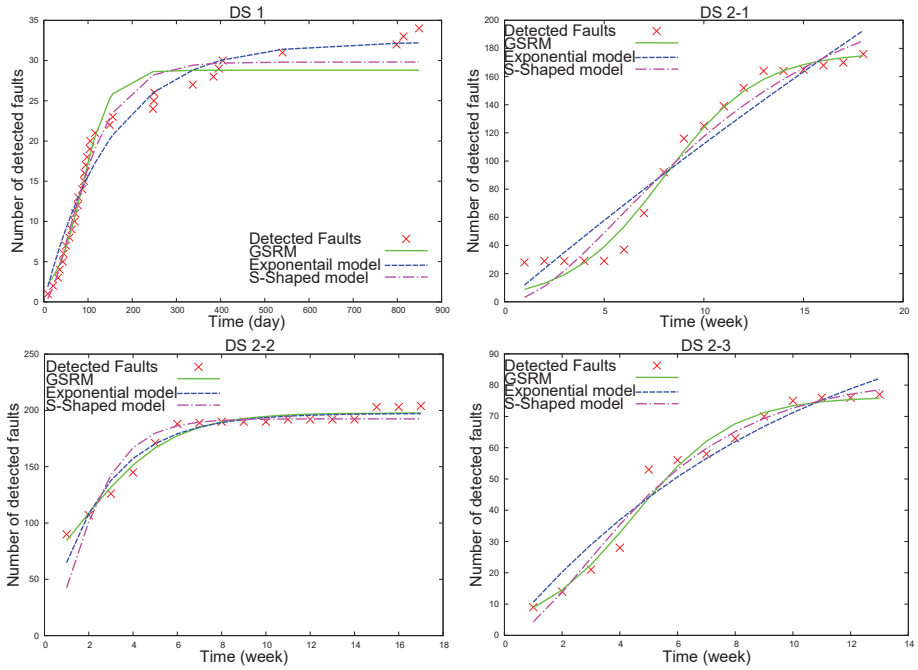


Figure 3.7: In "DS 1," the cumulative number of detected faults for DS 1 is plotted against the elapsed number of days. In the legend, Detected faults, GSRM, exponential model, and S-shaped model represent the actual data, the fit using GSRM, the fit using the exponential model, and the S-shaped exponential model, respectively. In the other graphs, the cumulative number of detected faults for 1 project of Dataset 2 is plotted against the elapsed number of weeks. Legends are the same as "DS 1."

Table 3.6: Comparison of GSRM with the NHPP models using datasets 1 and 2.

		exponential model	S-Shaped model	GSRM
DS 1	RSS	131.3	100.1	137.8
	AIC	148.4	139.2	152.1
DS 2-1	RSS	4612	3246	1310
	AIC	158.9	152.6	136.3
DS 2-2	RSS	696.1	3489	473.7
	AIC	119.4	145.8	112.8
DS 2-3	RSS	264.8	181.1	158.8
	AIC	84.07	79.14	77.43

Wide applicability (RQ1)

In my simulations, I applied the reliability growth models to nine types of development situations, which are characterized by two uncertainty elements related to the detection of faults.

I successfully simulated the three types of dynamics of the development situation: the number of detected faults per unit time is constant (e.g., the number of members is constant), the number of detected faults per unit time changes at the given time (e.g., new members join the project a given time) and the number of detected faults per unit time increases (e.g., new members join the project gradually). Especially, for the type where the number of detected faults per unit time increases, the number of detected faults is smaller than the other types in the late development, but it is greater than the other types in early development. This means that if new members join the project gradually, the number of detected faults will be smaller than other types at the late of the development. If managers decided to increase the members at the early development, ideally they would add new members all at once and not gradually.

Existing models can describe only one of these situations with additional limitations, but GSRM can describe several of these situations primarily because existing models cannot handle time-dependent growth rates without limitations, whereas GSRM can handle time-dependent rates as long as the appropriate uncertainty situation is inputted. Additionally, GSRM has a scheme for development uncertainties and can construct a model involving uncertainties.

GSRM has a good fit for almost all datasets. The S-shape model has a good fit for several datasets, but a worse fit for other datasets (e.g., dataset 2, release 2), indicating that the appropriateness of the model depends on the situation. GSRM shows that dataset 2, release 2 is the late uncertainty type, while the other releases have similar coverages for the different uncertainty types. The results suggest that dataset 2, release 2 may be complicated since many faults are detected in the first week. Regardless, these results indicate that GSRM can treat several situations.

The old dataset (e.g., dataset 1) is a constant type of GSRM, while the new datasets (e.g., dataset 2) are late types of GSRM. This means that the old development has uncertainty events throughout development, while the new development has uncertainty events at the later stage of development. Nowadays, many researchers and developers recommend that faults or bugs

be removed at the earlier stages of development. I assume that the new development would be well controlled since modern developers may have more skills and knowledge than traditional developers. On the other hand, I assume that the old development would be always exposed to uncertainty events since developers may not have sophisticated skills and specific knowledge.

Comparison with other models (RQ2)

Given a situation where the growth rate is time independent, I used two actual datasets to compare to GSRM with the NHPP models. The results show a high-precision convergence of the numbers of faults and appropriate development terms with GSRM. The precision of convergence is at least 12% higher for GSRM than for the NHPP models, confirming that GSRM can describe software growth more realistically than previously proposed models based on the NHPP models. Thus, using GSRM may help developers devise a more accurate plan for releasing software.

3.4.3 Limitations

I derive three uncertainty types by considering development situations artificially. However, the three types are intuitive.

To verify the probability of the types (late type, constant type, and early type), I interviewed several developers about whether these three types are suitable for actual developments. Almost all developers responded affirmatively and answered that their experiments seem to correspond to these types. Although one developer answered that there are other uncertainty types, the framework that the product employed in his experiment was forced to suddenly change to another framework mid-development. Although I propose three types of uncertainties in development as an initial study, it is true that there are other types of uncertainties.

Threats to Internal Validity

I treated the growth of the number of faults as a time dependent function. The growth of the number of faults may be related to other factors (e.g., test efforts). In this research, I could not collect and evaluate the test efforts and other data. The test efforts should influence the number of the detected

faults because if the test efforts are not constant, the number of detected faults changes.

I chose three types of uncertainty: late, constant, and early. It is not necessary to divide the uncertainty types by time categories. Although the responses vary by individual, I asked several developers whether their assessment agreed with the uncertainty type of the GSRM. Overall, the developers' thoughts were consistent with the uncertainty types.

In comparing the models, I used two datasets, both of which were obtained by one organization or company. It is possible that the data contains mistakes or other false elements. Several studies have focused on the accuracy of the faults datasets. An early paper on bug reports in open software indicates that the bug reports often contain information that does not describe bugs [11]. However, the datasets I used were collected from industries. In general, industries try to collect exact bug reports because they do not want to release software with bugs. If bugs remain in software released to other companies, the bugs may have a detrimental effect. Thus, products developed by industries tend to contain few faults. In [45], products related to dataset 2 have few faults in each release. Hence, the number of mistakes remaining in the dataset does not greatly impact my research results.

Additionally, the data were too old to compare with recent developments. However, recent studies have also employed these datasets, which should protect the validity of the results of this study.

Threats to External Validity

I only tested GSRM with two datasets, which is insufficient to make generalizations about GSRM. Moreover, the datasets are old and the scales of their systems are smaller than recent systems. Although a lot of factors (e.g., development styles, development scales, organizations, ages, etc.) should greatly affect the growth of the number of faults, I could not evaluate such factors. I evaluated two datasets belonging to a different organization published in different timeframes.

In the future, I plan to use datasets related to large-scale systems. Moreover, I plan to use datasets with different development styles or development scales. Additionally, I only compared GSRM with NHPP models. However, other models exist. Although these models have similar origins as the NHPP model, GSRM should be compared to other models besides NHPP models.

Threats to Construct Validity

I supposed that the uncertainty types of developments can be categorized as late type, constant type, and early type, and can be evaluated by applying GSRM to actual datasets. The types of uncertainties were artificial and may not be applicable to actual datasets because it is possible that the datasets belong to other uncertainty types.

Additionally, it is unclear whether I evaluated the correct uncertainty values by applying GSRMs to the datasets quantitatively since my definitions of the uncertainty values were built from only the viewpoints of the number of faults and time series. However, several developers who understood the results of GSRM concurred with the types determined by GSRM.

3.5 Related work

3.5.1 Software Reliability Growth Models

Many kinds of software reliability growth models exist. Several researchers proposed time-dependent models whose situations are limited such as the error detection per time increase with the progress of software testing [57]. Yamada et al. proposed an extend NHPP model, which is related with test-domain dependence [58]. The test-domain dependent model includes the idea that the tester's skills should improve by degrees (e.g., the growth of skills is time dependent). Hou et al. proposed a SRGM considering two learning curves, which are the exponential learning curve and the S-shaped learning curve [19]. These models assume that the number of developers does not change and the time dependent parameters are model specific like the exponential learning curve. My model does not depend on such specific models and situations.

Although software reliability models have not been used on waterfall development, Fujii et al. developed a quantitative software reliability assessment method in incremental development processes, which is an agile software development, based on the familiar non-homogeneous Poisson processes [6]. They used not only the number of faults but also software metrics, which are the number of software modules, the number of design reviews, the number of test cases performed, the size of software, and the development effort. They showed a software reliability prediction through a case study. They developed a SRGM for the incremental development processes and adopted

other metrics to their SRGM, but they did not directly treat the number of developers. My model can directly treat the number of developers, which should be suitable to actual and recent development styles.

Kuo et al. proposed a framework for modeling the software reliability model using various testing efforts and fault detection rates [27]. The testing efforts mean the resource expenditures spent on software testing (e.g., test cases, human resource, CPU time) applied into SRGMs in [53]. Kuo et al. applied their framework to several NHPP models by employing three types of testing effort functions (constant testing effort consumption, Weibull-type testing effort function, and Logistic testing effort functions) and two types of fault detection rates (constant proportionality and time-variable fault detection rates).

Additionally, Ahmad et al. proposed an S-shaped NHPP model containing testing-effort [1]. Ahmad et al. assumed that the testing effort expenditures are described by the Log-logistic function and integrated the Log-logistic function into an S-shaped NHPP model. Their model requires the test effort expenditures such as the test cases and human resource, but my model only needs the number of developers.

Huang et al. proposed five SRGMs with multiple changing points, which could be treated as the timing of introducing new tools or techniques [20]. They prepared datasets with and without multiple changing points, and evaluated their SRGMs with the changing points and the estimated changing points. Their idea about multiple changing points in development is similar to my assumption about dynamic changes in development. However, they only target the timing of the changes of development, whereas my model targets the timing and the concrete values such as the increased numbers of developers. Since my model can treat concrete values such as the numbers developers, I am able to simulate the development in section 3.3.

Singh et al. proposed a SRGM using a feed forward neural network approach, which is a machine learning method [44]. Their approach uses part of the data (60% to 80% from the beginning) for several datasets to train each neural network model. I supposed that if several development changes occur after learning data their model would not be able to treat such situations since it was constructed with the learning data using a machine learning technique. My model can treat a situation such as changes in developers, which I simulate in section 3.3.

3.5.2 Uncertainties

Several researchers tried to treat the uncertainties in requirements and operations. Wallace et al. studied the risk [49] and analyzed the software project risks to reduce the incidence of failure [49]. They mentioned that software projects have six dimensions: Team Risk, Organizational Environment Risk, Requirements Risk, Planning and Control Risk, User Risk, and Complexity Risk. They emphasized that Organizational Environment Risk and Requirements Risk are due to risks and uncertainties. My studies focus on the uncertainties and try to evaluate the uncertainties in a quantitative manner. In contrast, Wallace et al. did not focus on the uncertainties and did not evaluate the uncertainties in a quantitative way.

Goseva-Popstojanova and Kamavaram studied the uncertainties in requirements and operations by component-based software engineering [22] [10]. In [22], they analyzed the uncertainties of operational profiles and component reliability by calculating the conditional entropy of each component. In [10], they analyzed the uncertainties of the operational profiles and the component reliability by Monte Carlo simulations. These analytic methods focus on the requirements and operations. However, my methods focus on the uncertainties in development phases, including requirements and operations. In the future, I plan to analyze the effects of the uncertainties in requirements and operations since I suppose that the uncertainties in the requirement phases affect the development process.

3.6 Conclusion

Using GSRM, I successfully simulated developments containing uncertainties and dynamic elements. I obtained the time-dependent logistic curve and growth curve, which is not possible using other models, as well as simulated and analyzed nine types of developments with GSRM. Additionally, I formulated equations for three types of uncertainty that are related to actual development situations. I also defined uncertainty values from actual data containing information on faults during development and applied GSRM to datasets to calculate the fitness of the models. These results demonstrate that GSRM can calculate uncertainties using past data and predict how long a project will take.

In the future, I plan to evaluate teams or team members using quantita-

tive methods while considering uncertainties to optimize teams for a particular project using GSRM.

Chapter 4

Predicting Release Time of Open Source Software Based on GSRM

4.1 Introduction to This Chapter

I propose a model (GSRM) that can describe several development situations that involve random factors, such as the skills of teams and development environments, to provide a time range in which the development will end. Earlier studies only use linear stochastic differential equations; however, my research indicates that non-linear stochastic differential equations lead to more elaborate equations that can model situations more realistically. Moreover, I aim to reveal the development of open source software (OSS).

4.2 Proposal method

I focus on the determination when OSS will be released in the point of view of issue growth. Especially I use two methods which are below.

A. Separating development time periods into each version.

B. Using GSRM and predicting the number of issues and release dates.

4.2.1 Separating time periods

The upper side graph of figure 4.1 indicate the growth of issues about "foundation,"[1] which is a front-end framework, divided by each version. The shapes of curves are significantly sharped when the new version released. Therefore I scope the changing points of versions and separate them into each version, and apply my model (GSRM). The reason for separating them is to approximate GSRM more precisely and treat them more naturally.

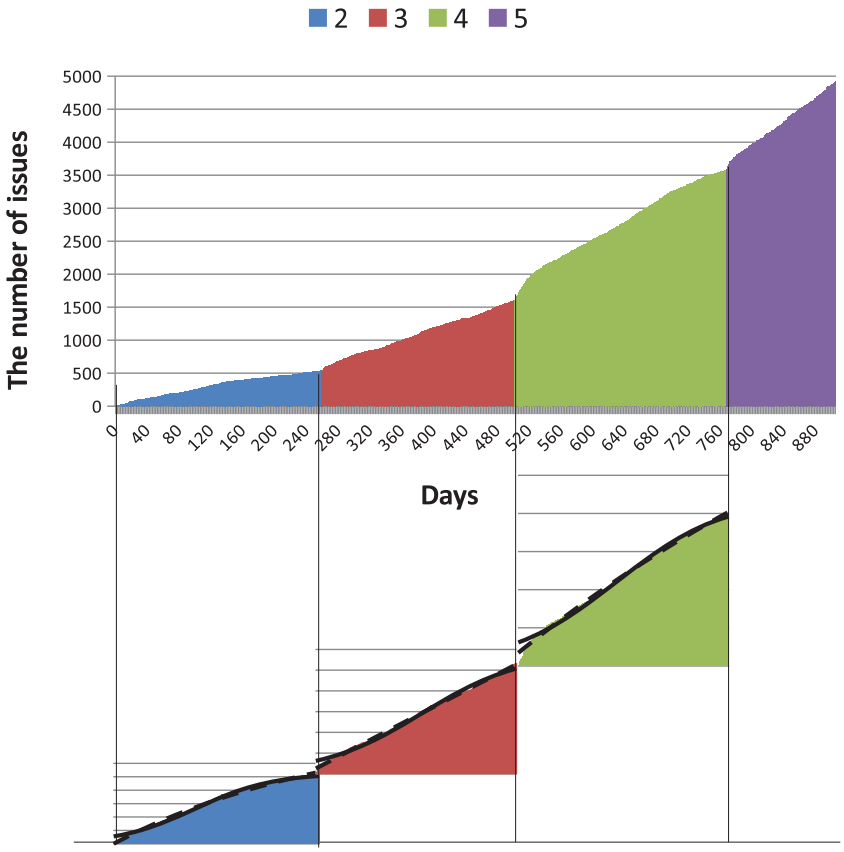


Figure 4.1: The number of issues and development days about "foundation."

[1] http://foundation.zurb.com/

4.2.2 Prediction of Release Time

I defined the methods to predict release times by using a GSRM and an NHPP model (Exponential model). In general, the time when the development will end is planned as when the number of detected faults is getting around 95% of the predicted number of faults. SRGM can calculate the predicted number of faults and the time when the development will end.

In the case of a GSRM, by using the equation (3.13), I can obtain the time when 95% of the predicted number of faults will be found.

$$N(t) = \frac{N_{\max}}{1 + b\exp(-\alpha t)} \tag{2.13}$$

I substitute $N(t) = 0.95 \cdot N_{\max}$ for the equation (3.13) and solve it in terms of t.

$$t = \frac{1}{\alpha}\ln(19 \cdot b) \tag{4.1}$$

In the case of an NHPP (Exponential model), by using the equation (1.3), I can obtain the time when 95% of the predicted number of faults will be found.

$$H(t) = N_{\max}(1 - \exp(-ct)) \tag{1.3}$$

I substitute $H(t) = 0.95 \cdot N_{\max}$ for the equation (1.3) and solve it in terms of t.

$$t = \frac{1}{c}\ln(20) \tag{4.2}$$

I used these two equation (4.1) and (4.2) and evaluated the predicted times.

4.3 Application to OSS

I discuss the differences between GSRM and the Non-Homogeneous Poisson Process (NHPP) models using actual development data of OSS named as "foundation" in a given situation as the growth rate is time-independent. The reason for this limitation is because the NHPP model cannot be applied to time-dependent situations. I compare GSRM with a general NHPP model on data sets obtained from Github sites[2]. In Table 4.1, I show the numbers of issues and the days each version and the residual sum of squares (RSS) and the Akaike's Information Criterion (AIC) about models. This results show

[2]https://github.com/zurb/foundation

GSRM is better than Exponential model in the view point of predictions. Predicted numbers of issues and days by GSRM are more precisely than those of Exponential model.

Table 4.1: Comparison of GSRM with NHPP model (Exponential model).

		Actual Data	GSRM	Exponential model
Version 2	Issue	536	526	899
	Days	258	245	854
	RSS	-	50388	25929
	AIC	-	2108	1936
Version 3	Issue	1066	1170	32555
	Days	242	306	23102
	RSS	-	182119	44708
	AIC	-	2306	1965
Version 4	Issue	1974	2203	5897
	Days	265	323	2017
	RSS	-	720089	302405
	AIC	-	2865	2634

4.4 Related work

The software reliability models had ever been used on water fall development, however Fujii et al. developed a quantitative software reliability assessment method in incremental development processes, which is one of agile software developments, based on the familiar non-homogeneous Poisson processes.[6] Fujii et al. did not use only the number of faults but also the software metrics and showed software reliability prediction through a case study.

4.5 Conclusion

Using GSRM, I was able to successfully predict the release dates and the number of issues about OSS. However NHPP can more precisely approximate the growth of issues than GSRM. For future work I will adjust the time-dependence of models. In this Chapter, I limited the development situation as time-independence situation, in order to compare GSRM with NHPP and

the lack of data about time-dependent elements, thus GSRM could more precisely approximate the growth of issues.

Chapter 5

Predicting Time Range of Development Based on GSRM

5.1 Introduction to This Chapter

Software reliability is a critical component of computer system availability. Especially, quality managers try to control software reliability and project managers try to estimate the end of development for planing developing term and distribute the manpower to other developments. In a estimating method, a manager collects faults data, which contains the number of faults and the time when faults are detected, and estimates how many faults remained using the faults data applying a prediction model, which is called software reliability growth model. Thus, software reliability growth models have been developed to indicate whether enough faults have been removed to release the software. Although the logistic curve and Gompertz curve [56] are well-known software reliability growth curves, they cannot account for the dynamics of software development, which are affected by various development environment elements, such as the skills of the development team, changing requirements, etc. However, these curves cannot account for the dynamics of software development. Developments are affected by various elements of the development environment, such as the skills of the development team and changing requirements.

Many current models only give the number of faults that will be found within some time range. Here, I propose a model called the generalized software reliability model (GSRM), which can describe several development situ-

ations that include random factors (e.g., skills of the development team and development environment), to provide a time range in which development will end. Although earlier studies use linear stochastic differential equations, my research indicates that non-linear stochastic differential equations lead to more elaborate equations that can model situations more realistically. Furthermore, GSRM can quantify uncertainties influenced by random factors. To more accurately predict the time required to complete development and to optimize development teams and environments, uncertainties must be quantified. Thus, this study aims to answer the following three research questions.

RQ1: How much better is GSRM at describing the growth of software reliability in different situations compared to other models (e.g., NHPP)?

RQ2: How accurately does GSRM describe the convergence of the number of faults and the appropriate development term in a given situation compared to other models?

RQ3: How does GSRM predict when a development will end, considering the dispersion due to uncertainty?

My contributions are as follows:

- A generalized software reliability model, which is 12% more precise than existing models.
- A new method to predict when development will end.

5.2 Motivating example

Existing software reliability growth models give us the number of faults will be found with some ranges of raults, however the models cannot precisely indicate the time when the development will end. Figure 5.1 shows an example dataset from reference [45] written by C. Stringfellow et al. , which is drawn with three models: the normal NHPP (Exponentail model), its upper limit, and its lower limit, which indicate a greater deal of faults than Exponentail model and a less than Exponentail model, whose values are calculated with confidence interval as 70%. For example, the end for the “Lower Limit Time” is twice that of the “Upper Limit Time,” indicating that the predictions are

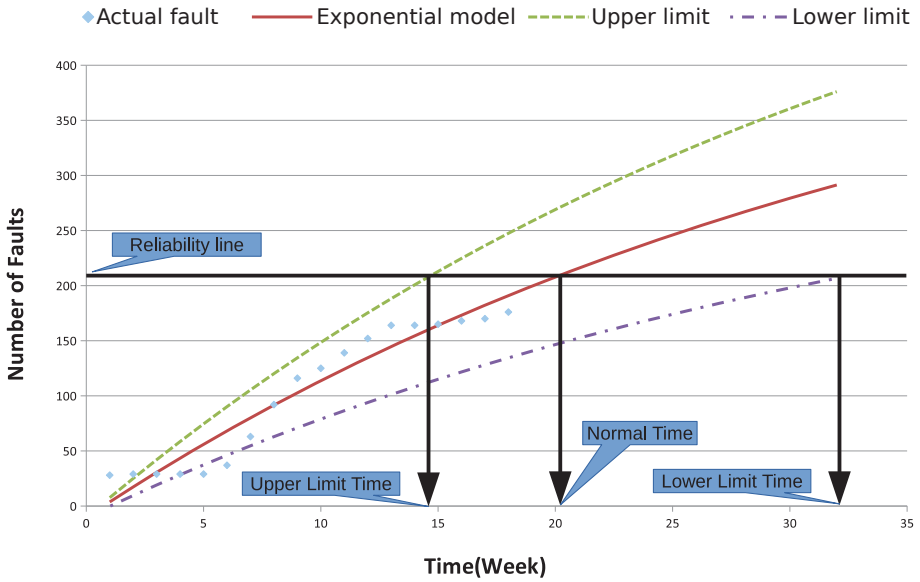


Figure 5.1: Time ranges based on NHPP model (Exponential model).

not meaningful. Hence, I try to construct a new method which can define the time ranges of development at section 5.3.2.

In this chapter, I compare GSRM with these models. Equation (1.3) results in an exponentially shaped software reliability graph. However, actual software reliability graphs typically follow a logistic curve or a Gompertz curve [56], which are more complex. Therefore, I propose a new model, GSRM, which can fit either a logistic curve or an exponentially-shaped curve for use in actual developments.

5.3 Generalized Software Reliability Model

For my software reliability model, I extend a nonlinear differential equation that describes the fault content as a logistic curve into an Ito-type stochastic differential equation. I start with equation. (5.1), which is called the logistic differential equation.

$$\frac{dN(t)}{dt} = N(t)(a + bN(t)) \tag{5.1}$$

$N(t)$ is the number of detected faults by time t, a defines the growth rate, and b is the carrying capacity. If $b = 0$, then the solutions of this equation become exponential functions. Equation (5.1) can be extended into a stochastic differential equation because actual developments do not strictly obey equation (5.1) due to the numerous uncertainties and dynamic changes. Such dynamic elements are considered time-dependent and to contain uncertainty; these factors are expressed in a. The time-dependence of a can be used to describe situations such as skill improvement of development members and increases in the growth rate, while the uncertainty of a can describe parameters such as the variability of development members and the environment. I analyze the growth of software with an emphasis on the testing phase by simulating the number of detected faults. Software development is assumed to have the following properties:

1. The total number of faults is constant.

2. The number of faults that can be found varies depending on time.

3. The number of faults that can be found contains uncertainty, which can be simulated with Gaussian white noise.

Considering these properties, equation (5.1) can be extended to an Ito-type stochastic differential equation with $a(t) = \alpha(t) + \sigma dw(t)$ as

$$dN(t) = (\alpha(t) + \frac{\sigma^2}{2} + \beta N(t))N(t)dt + N(t)\sigma dw(t) \tag{5.2}$$

$N(t)$ is the number of detected faults by time$t$, $\alpha(t) + \sigma^2/2 + \sigma dw(t)$ is the differential of the number of detected faults per unit time, $\gamma(t) = N(t)\sigma dw(t)$ is the uncertainty term, σ is the dispersion, and β is the non-linear carrying capacity term.

5.3.1 Uncertainty and Time-dependence

In development, faults are detected and debugged. The detected faults are counted and used to predict when the project will end. A project has a lot of uncertain elements, and the predicted development period is almost never long enough. GSRM can describe the uncertainty of the applied development and calculate the uncertainty of fault detection.

I describe the uncertainty as σdw, which is basically Gaussian white noise and can be obtained from past data. Because the uncertainty is difficult to calculate from equation (5.1), I assume there are some limits to obtain σdw in quantitative manner. The result can be useful. I start by defining a in terms of σdw from equation (5.1) as

$$a = \alpha(t) + \sigma dw(t) \tag{5.3}$$

However, equation (5.1) cannot be solved due to the time-dependence of a as shown in equation (5.3). Therefore, I assume that a is time-independent with an added term δ, which is small. This assumption allows equation (5.1) to be solved, and can be rewritten as

$$\frac{dN(t)}{dt} = N(t)(\alpha + \delta + bN(t)) \tag{5.4}$$

Equation (5.4) can be solved as

$$N = \frac{N_{max}}{1 + b\exp\{-(\alpha + \delta)t\}} \tag{5.5}$$

This equation is a logistic equation where δ is the origin of the uncertainty. $\alpha + \delta$ is the gradient. The sign of δ can be positive or negative. If δ is negative, the gradient of the equation is small, whereas if it is positive, the gradient of the equation is large.

The sign of δ provides the limitation of the uncertainty. If δ is negative (positive), the growth of the graph provides the lower (upper) limit. The lower and upper limits are calculated in the next section. δ is calculated as

$$\delta_i = -\frac{1}{t_i}\ln\left\{\frac{1}{b}\left(\frac{N_{max}}{N_i} - 1\right)\right\} - \alpha \tag{5.6}$$

The subscript i indicates that the data is for the ith fault detected at t_i. i differs from the approximate value at t_i. Finally, the average and variance of δ are obtained, which are used to construct an equation for the software reliability growth model.

By using δ and its distribution, which are Gaussian white noise, I can predict the range of the required development period. The range due to uncertainties is obtained using the equations below.

I assume that the detected faults obey equation (5.4) and that the detection rate has an uncertainty δ that is time independent, which leads to

$$N_+(t) = \frac{N_{max}}{1 + b\exp\{-(\alpha + \delta)t\}} \tag{5.7}$$

$$N_-(t) = \frac{N_{max}}{1 + b\exp\{-(\alpha - \delta)t\}} \tag{5.8}$$

For $N_+(t)$, the rate of development is faster than for $N(t)$. For $N_-(t)$, the rate of development is slower than for $N(t)$. Using these equations, I can establish the range from the shortest development period to the longest. The development periods are expressed as

$$t_\pm(t) = -\frac{1}{\alpha \pm \delta}\left[\ln\left\{\frac{1}{b}\left(\frac{N_{max}}{N} - 1\right)\right\}\right] \tag{5.9}$$

5.3.2 Time Range of Development

The development period usually ends when a certain percentage of expected faults (typically 95%) are detected and removed. Using equation (5.9), the range of the development period can be calculated before the development period actually ends. The range is defined as $\Delta t = t_- - t_+$, and is expressed as

$$\Delta t = \left(\frac{-2\delta}{\alpha^2 - \delta^2}\right)\left[\ln\left\{\frac{1}{b}\left(\frac{N_{max}}{N} - 1\right)\right\}\right] \tag{5.10}$$

By calculating the development period range in the development, the delay risk can be predicted as well as the delay range. Figure 5.2 depicts the relations among these equations.

Figure 5.2 shows the time range of the same data with Figure 5.1. In figure 5.1, the time range is 26.30 weeks, however in figure 5.2 the time range is 6.40 weeks. These results show my method is more meaningful to predict the end of the development than that of NHPP.

5.4 Evaluation and Discussion

5.4.1 Comparison with the NHPP models

The exponential model is one of many proposed reliability models. In this section, I discuss the differences between GSRM and exponential model us-

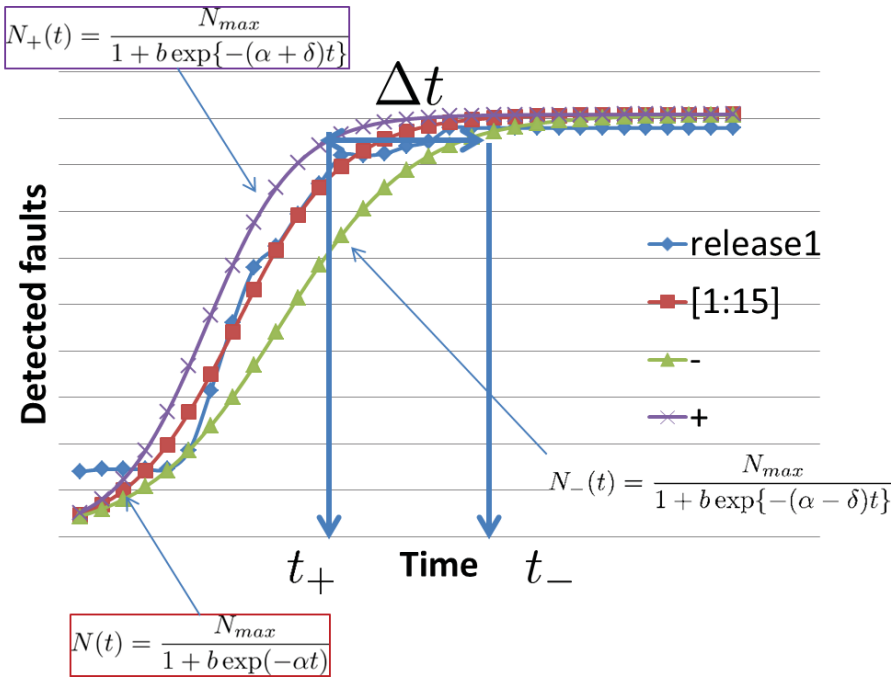


Figure 5.2: Relationship between the equations for $N(t)$, $N_+(t)$ and $N_-(t)$, and Δt, t_+ and t_-.

ing actual development data for a given situation when the growth rate is time-independent. This limitation is imposed because the exponential model cannot be applied to other time-dependent situations.

Dataset 1

This development dataset is from reference [9] written by Goel and Okukmoto. The data are originally from the U.S. Navy Fleet Computer Programming Center, and consist of the errors in software development. Figure 5.3 plots the results using the exponential model and GSRM. The parameters for both GSRM and the exponential model are calculated by R, which is a language and environment for statistical computing and graphics. The residual sum of squares(RSS) and Akaike's Information Criterion (AIC) are calculated from these models and the development data (Table 5.1). These results show that GSRM provides a better approximation than the NHPP models because GSRM is more flexible due to the non-linear term.

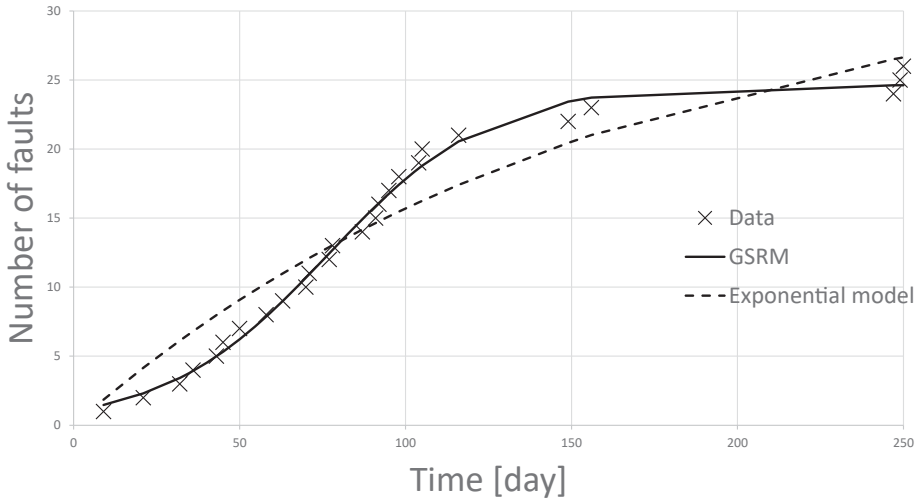


Figure 5.3: Comparison of GSRM and the exponential model.

Table 5.1: Comparison of GSRM and NHPP models using dataset 1.

	Exponential model	S-Shaped model	GSRM
RSS	67.21	35.14	11.2
AIC	106.5	87.62	59.90

Dataset 2

The second development dataset is from reference [45] written by C. Stringfellow et al. The data come from three releases of a large medical record system, which consists of 188 software components (Table 5.2). The data contain the cumulative number of faults and their detected times for the three different releases of a software program. Figures 5.4-5.6 plot the results for each release using the exponential model and GSRM. The parameters are calculated by R for both GSRM and the exponential model. Then these equations and developmental data are used to calculate RSS and the AIC (Table 5.3). Furthermore, δ is calculated, and the upper and lower limits are simulated and calculated. Almost all of the real data points are contained within the calculated upper and lower limits. GSRM produces a good fit for release 1 (Figure 5.4) as the curve for the lower limit corresponds to the worst-case scenario, indicating that if the development is continued until 95% of the 176 faults are detected, five more weeks are necessary than it actually took to complete the development. However, the upper and lower limits are almost the same for the release 2 (Figure 5.5), suggesting that the development does not have critical uncertainties. Additionally, the GSRM results realize a good fit for release 3 (Figure. 5.6), and although most of the data points are within the curves for the upper and lower limits, a few are above the upper curve.

Table 5.2: Dataset 2. Number of weeks for development and the number of faults for the three different releases of a large medical record system.

	Weeks	Number of faults
Release 1	18	176
Release 2	17	204
Release 3	13	77

Dataset 3

I collected the third development dataset from Yahoo Japan Corporation in 2013. The data comes from a platform of a search engine. A platform consists of seven major modules: messaging, storage, UI, common, consumer, control-api, and data-api. The modules manage development using Jenkins

Table 5.3: Comparison of GSRM with the NHPP models using dataset 2.

		Exponential model	S-Shaped model	GSRM
Release 1	RSS	4612	3246	1310
	AIC	158.9	152.6	136.3
Release 2	RSS	696.1	3489	473.7
	AIC	119.4	145.8	112.8
Release 3	RSS	264.8	181.1	158.8
	AIC	84.07	79.14	77.43

and track faults with it. Table 5.4 shows each module's faults. Figures 5.7-5.9 plot the actual number of faults and the predicted number of faults using GSRM for messaging, common and consumer. Table 5.5 compares GSRM to the NHPP models. The results demonstrate that GSRM more accurately.

Table 5.4: Number of faults in dataset 3.

Module Name	Days	Number of faults	Predicted faults
messaging	206	240	232.88
storage	194	50	54.63
UI	187	148	144.09
common	184	134	126.72
consumer	157	63	58.50
control-api	190	73	68.08
data-api	183	147	144.60

5.4.2 Prediction of time ranges

The time range (Δt) when a development is predicted to end is calculated using GSRM as well as by the exponential model. In the exponential model, the range is determined by using the upper and lower boundaries to define a confidence interval of 90%.

Dataset 2

In dataset 2, Δt is determined by GSRM when the upper and lower boundaries cross the 85% mark of the total number of predicted faults. Table 5.6

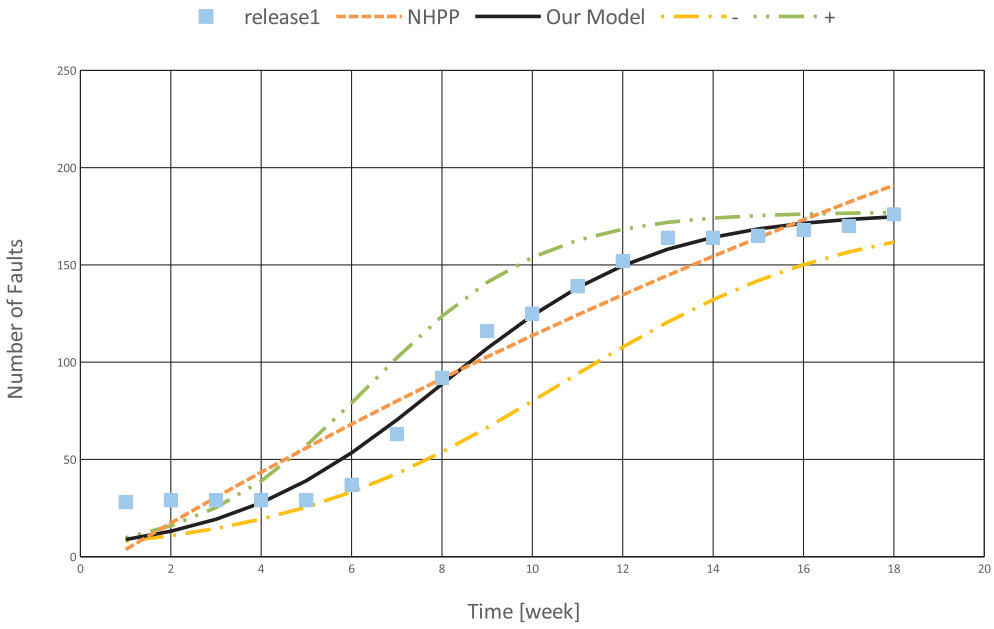


Figure 5.4: Cumulative number of detected faults for the entire project of release 1 versus the elapsed number of weeks. release1, Exponential, My Model, +, and − represent the actual data, the fit using Exponential model, the fit using GSRM, the predicted upper limit, and the predicted lower limit, respectively.

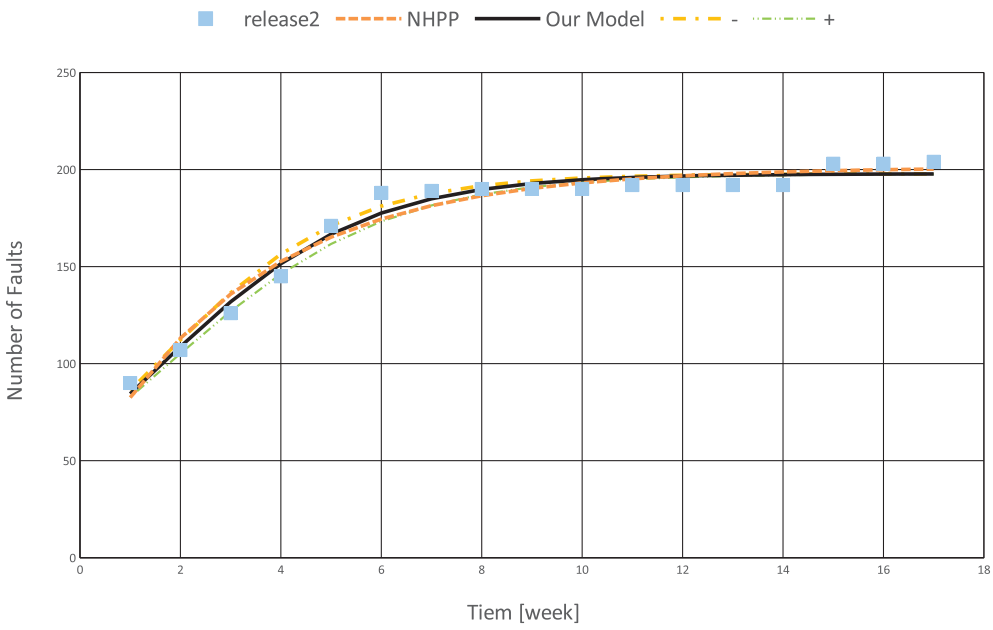


Figure 5.5: Cumulative number of detected faults for the entire project of release 2 versus the elapsed number of weeks. Legend is the same as Figure 5.4.

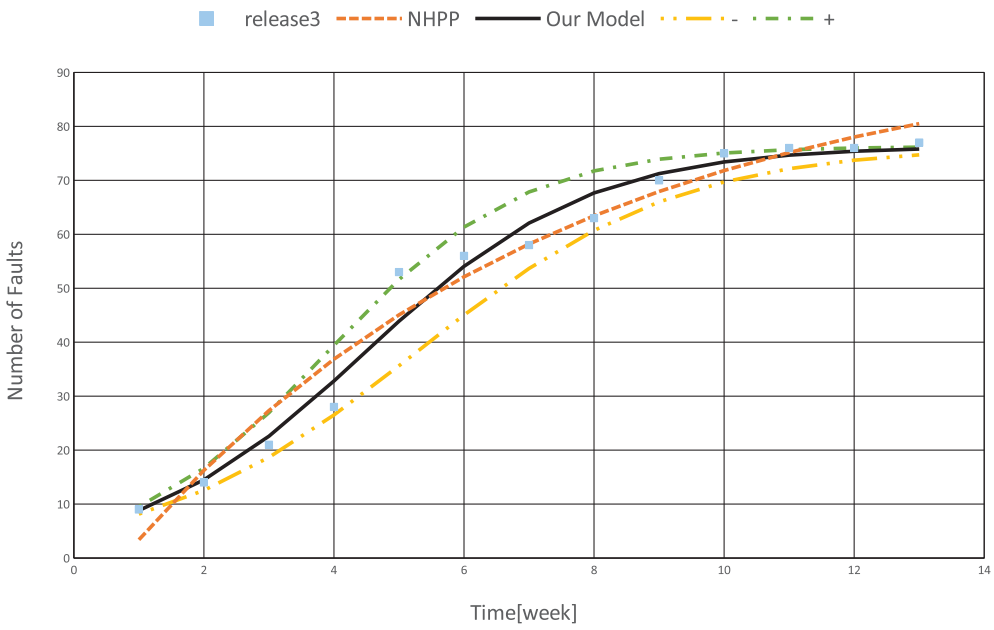


Figure 5.6: Cumulative number of detected faults for the entire project of release 3 versus the elapsed number of weeks. Legend is the same as Figure 5.4.

Table 5.5: Comparison of GSRM and the NHPP models using dataset 3.

Module Name		Exponential	S-Shaped	GSRM
messaging	RSS	50626	31510	12240
	AIC	1971	1857	1632
storage	RSS	409	592	232
	AIC	253	592	226
UI	RSS	47250	8160	2416
	AIC	1279	1019	841
common	RSS	357852	6824	7124
	AIC	1443	912	920
consumer	RSS	13168	622	514
	AIC	521	329	319
control-api	RSS	1913	1500	784
	AIC	451	433	388
data-api	RSS	9560	3359	1001
	AIC	1036	883	707

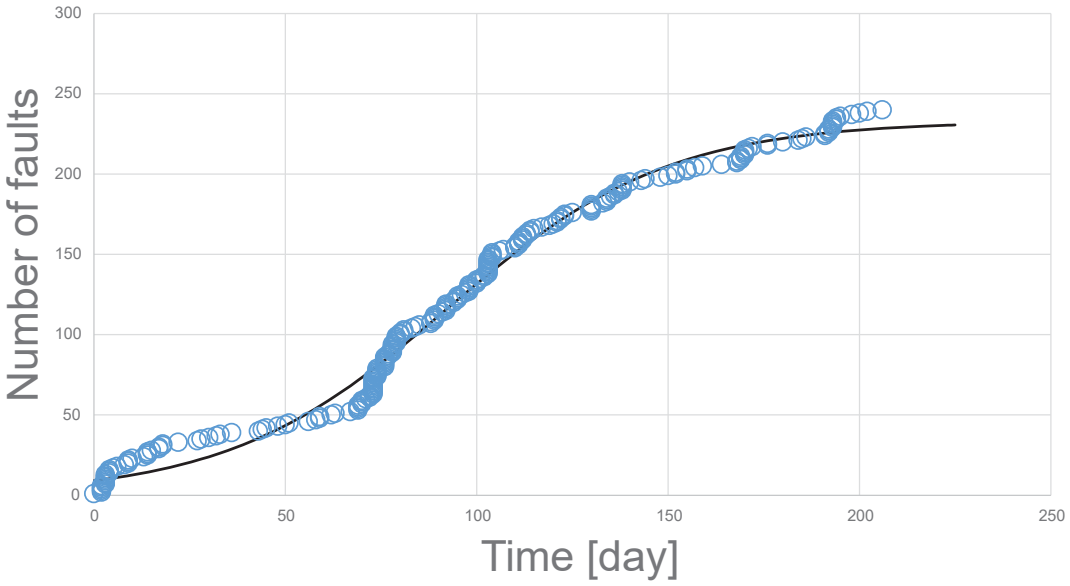


Figure 5.7: Plot of the number of faults over time for the messaging module. Circles and solid line indicate the actual faults and predicted faults by GSRM, respectively.

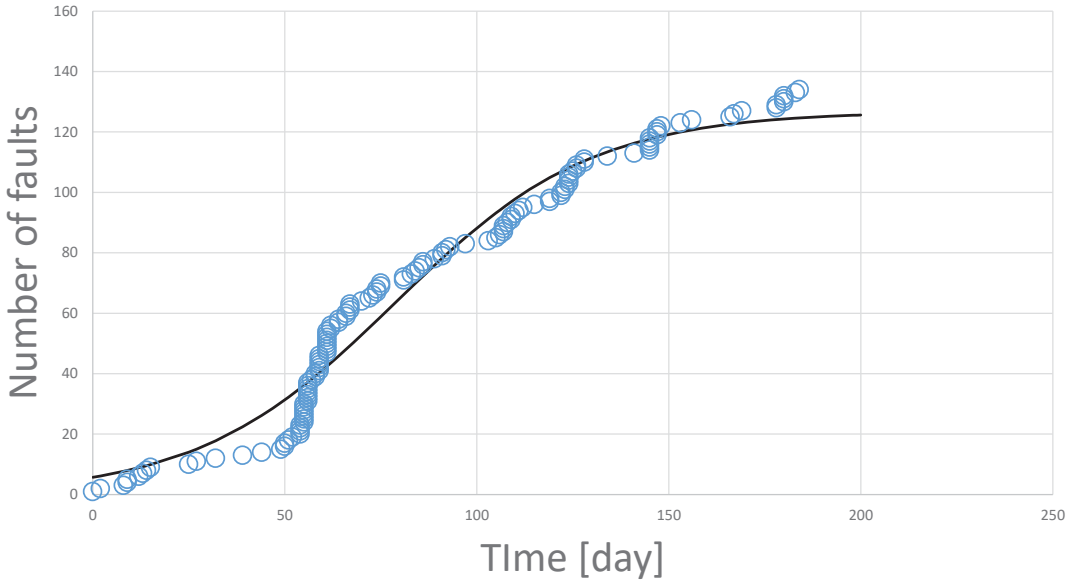


Figure 5.8: Plot of the number of faults over time for the common module. Circles and solid line indicate the actual faults and predicted faults by GSRM, respectively.

shows Δt for GSRM and the exponential model. Δt's obtained by GSRM have a 40% narrower width than those obtained by the exponential model, and each dataset lies almost entirely within the time ranges obtained by GSRM. Table 5.7 shows Δt predicted based on GSRM using a partial dataset. The values of Δt decrease as the development proceeds and the amount of data used for the predictions increases. The results indicate that GSRM can be used to predict delays in development.

Table 5.6: Predicted ranges of the exponential model and GSRM for each dataset. Unit time is weeks.

	Exponential model	GSRM
Release 1	31.002	3.739
Release 2	2.890	1.051
Release 3	9.601	3.739

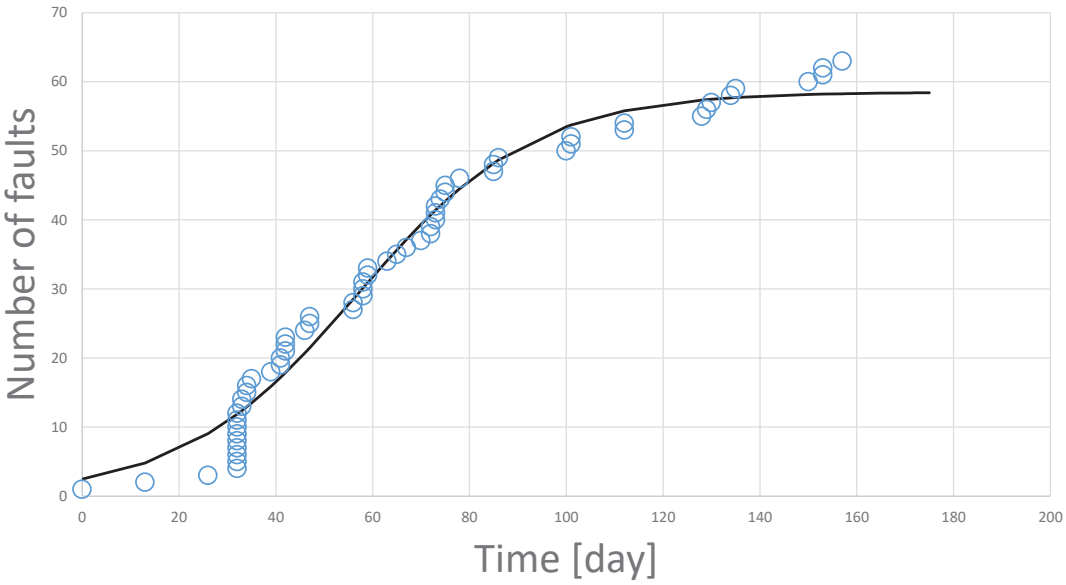


Figure 5.9: Plot of the number of faults over time for the consumer module. Circles and solid line indicate the actual faults and predicted faults by GSRM, respectively.

Table 5.7: Δt for each releases. Unit time is weeks.

	All	[1:15]	[1:12]	[1:9]	[1:7]
Release 1	5.314	6.395	8.436	-	-
Release 2	0.771	0.727	0.679	-	0.263
Release 3	2.287	-	4.075	2.754	3.254

Dataset 3

In dataset 3, Δt for GSRM or the exponential model is determined when the upper and lower boundaries cross the 70% mark of the total number of predicted faults (Table 5.8).

Table 5.8: Ranges of the exponential model and GSRM in dataset 3.

Module Name	Exponential model	GSRM
messaging	1309	58.13
storage	219	86.92
UI	3308	17.01
common	58162	-
consumer	NaN	-
control-api	421	14.62
data-api	444	72.41

All of the ranges from GRSM are less than those of exponential. At least one of the exponential ranges is 2.5 times greater than those of GRSM, demonstrating that the exponential model's range is not meaningful for predicting when development will end.

Table 5.9 shows Δt of each module's faults and that predicted using GSRM and a partial dataset. By comparing to the data in Table 5.4, some modules are well predicted during development, and almost all the GSRM results are well fitted and meaningful to predict when development will end. However, the common and consumer modules cannot be predicted because the predictions using partial data are inaccurate. The prediction using the partial dataset for the common (consumer) module is 101.74 (98.03) days, while the actual development is 184 (157) days.

5.4.3 Summary

Wide applicability (RQ1)

My simulations applied the reliability growth models to nine types of development situations, which are characterized by two uncertainty elements related to fault detection. Although existing models can describe only one of these situations with additional limitations, GSRM can describe several of these situations. This is primarily because existing models cannot handle

Table 5.9: Number of faults in dataset 3.

Module Name	Predicted end days	Predicted left days	Number of faults
messaging	173.33	25	198
storage	201.36	60	41
UI	184.69	57	117
common	101.74	-23	107
consumer	98.03	-3	52
control-api	199.85	76	58
data-api	156.39	43	124
Module Name	Predicted faults	Predicted range	
messaging	230	58.13	
storage	49	86.92	
UI	173	17.01	
common	96	-	
consumer	53	-	
control-api	97	14.62	
data-api	145	72.41	

time-dependent growth rates without limitations. In contrast, GSRM can handle time-dependence, and only the appropriate type of situation must be inputted. Additionally, GSRM has a scheme for development uncertainties and can construct a model involving uncertainties.

Comparison with NHPP model (RQ2)

Given a situation where the growth rate is time-independent, I used two actual datasets and compared GSRM to the NHPP model. The results show a precise convergence of the numbers of faults and the appropriate development terms with GSRM. The convergence precision is at least 12% higher for GSRM than for the NHPP model, demonstrating that GSRM can describe software growth more realistically than previously proposed models based on the NHPP model. Thus, GSRM may provide developers with a more accurate plan for releasing software.

Predictions involving uncertainties (RQ3)

For two datasets, GSRM is able to model the uncertainties, and calculate Δt to predict not only the total development time, but also how long development will be delayed due to uncertainties. Δt cannot be obtained with other models. Therefore, existing models can only predict the day when the development will be end, but not the length of a delay.

Internal validity threats

In comparing models, I use two datasets, both of which were obtained by one organization or company. Therefore, the data could contain mistakes or some other false elements. Moreover, the data were too old to compare with recent developments. However, recent studies also use these data so the validity should be protected.

External validity threats

We only tested GSRM with two datasets, which is insufficient to make generalizations about GSRM. Moreover, the datasets are old and the scales of their systems are smaller than recent systems. A future project will use datasets related to large-scale systems. Additionally, I only compared GSRM with the NHPP model (exponential model). There are a lot of other proposed and applied models. Although these other models have similar origins as the NHPP model, GSRM should also be compared to them.

5.5 Related work

Many different types of software reliability growth models exist. Yamada et al. proposed an extend NHPP model, which is related to test-domain dependence [58]. The test-domain dependent model includes the notion that the tester's skills should improve by degrees; thus, skills grow over time. The test-domain dependent model adds additional assumptions to the NHPP model.

Although water fall development has not been applied to software reliability models, Fujii et al. developed a quantitative software reliability assessment method via incremental development processes, which is a type

of agile software development based on the familiar non-homogeneous Poisson processes [6]. Fujii et al. used both the number of faults and software metrics to demonstrate software reliability predictions via a case study.

M. Xie et al. studied about the predictions and estimations of software release time with the exponential model [51]. They changed the values of parameters which were obtained from the software development in progress and estimated the optimal software release time. Their method can estimate a later release time and earlier release time, however, their method needs the concrete values to change, like the 10% of the parameters. My method does not need such concrete values to change because my method predicts the release time by evaluating the differences between the model and actual values.

S. Inoue et al. proposed a cost-optimal software release planning by using a bootstrap method [21]. They focused on the cost optimizations for software developments and employed the software reliability growth model and a bootstrap method. By using a cost condition, they evaluated the optimal software release time intervals with a bootstrap confidence intervals. On the other hand, my method did not treat cost optimizations. In the future work, I should compare my method with their method by adding the cost optimization to my method.

5.6 Conclusion

GSRM can be used to predict the length of the development when the team composition drastically changes during development as well as to simulate and analyze nine types of developments. Furthermore, I was able to define uncertainty values from actual data containing information on the faults during development, and apply GSRM to three datasets to calculate Δt, including the range of possible development times considering the uncertainty values. I also examined how well GSRM can predict the required development time using actual datasets. By using past data, GSRM can calculate the uncertainties with and predict how long a project will take. In the future, I aim to find methods to quantitatively evaluate teams or team members considering uncertainties and to optimize teams to suit particular projects using GSRM.

Chapter 6

Detection of Unexpected Situations by Applying Software Reliability Growth Models to Test Phases

6.1 Introduction to This Chapter

I apply SRGMs to the datasets of two projects developed by Fujitsu Labs Ltd. in order to determine when SRGMs provide ill-fitted or unexpected results. I assume that the detected faults differ by test phase, and this difference is the source of misfit between the model and actual data. To investigate the source of unexpected results, two different SRGMs are used. In the first case, SRGM is applied to the entire dataset. In the second case, the dataset is divided into test phases, SRGM is applied to each phase separately, and the results are summed. Separating the faults into test phases and combining the results provides a better fitting model.

6.1.1 Motivating example

I found two problems when applying SRGMs to an actual dataset. One is an ill-fit between the model and the dataset. The other is when I applied SRGM to a dataset during the middle of development, the values were overestimated compared to the anticipated ones.

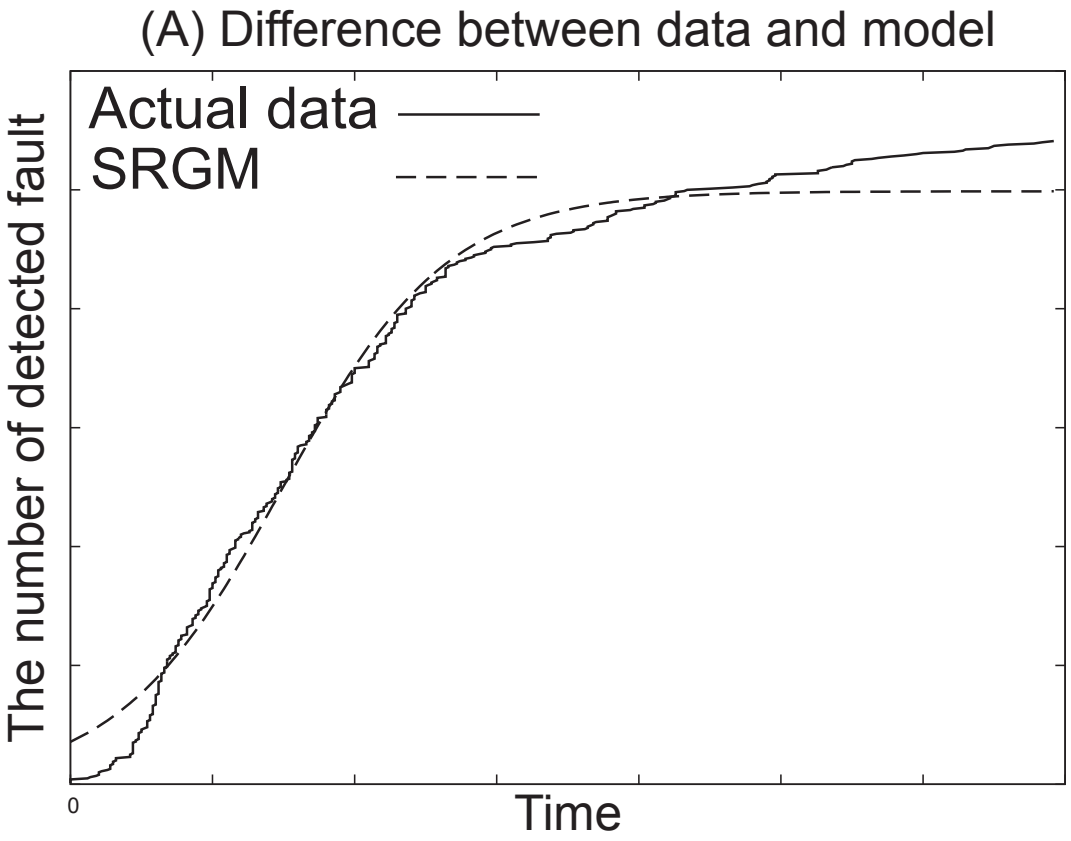


Figure 6.1: (A) Difference between the actual data and the model. Solid and dashed lines represent the actual data and SRGM, respectively. Cumulative number of detected faults for all of Project 1 as a function of elapsed time.

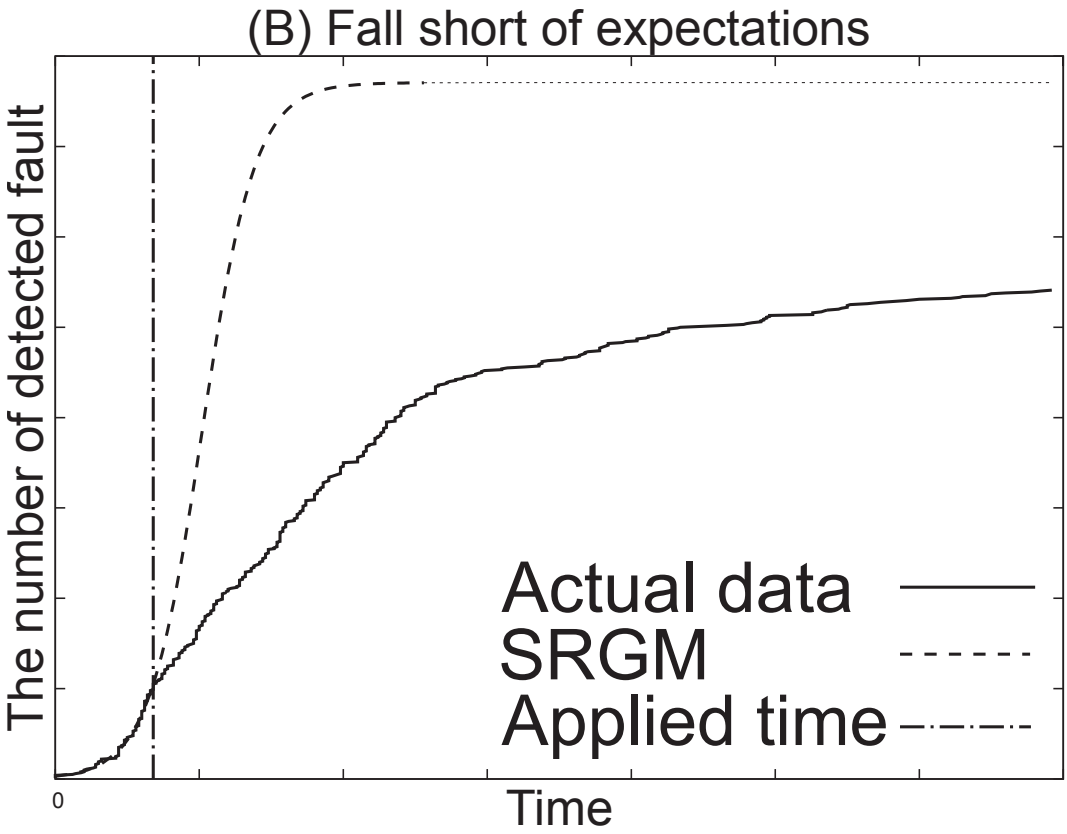


Figure 6.2: (B) Case where SRGM overestimates expectations. Solid, dashed, and dotted-dashed lines represent the actual data, SRGM, and the time I applied SRGM, respectively. Cumulative number of detected faults for all of Project 1 as a function of elapsed time.

Figure 6.1 indicates that the model and the actual data do not fit well during the early, middle, and end of development for the cumulative project. If the model does not fit the actual data, developers and managers cannot decide development plans and release times. Moreover, the misfit leads to difficulties determining when testing is complete. Herein I assume that the misfit is due to the difference between test phases. Test processes are separated by purpose (e.g., unit test, integration test, system test). Therefore, the detected faults depend on the test phase and related modules or features.

Figure 6.2 indicates that when I applied SRGM to the middle of development (at the dashed-dotted line) the model overestimates the actual data. This means that developers and managers will not believe the model because it indicates that too many faults will be found. It is assumed that when developers and managers apply SRGM depends on the development situation.

This study aims to answer the following research questions:

1. RQ1: How precise is the SRGM model when faults are separated by test phase?

2. RQ2: Can continuous fault predictions and monitoring detect unexpected situations?

6.2 Background

Software reliability is important when releasing software. Several approaches have been proposed to measure reliability. One is to model fault growth, which is a type of SRGM. Because software development includes numerous uncertainties and dynamics regarding development processes and circumstances, this section explains SRGM, its uncertainties, and dynamics as well as provides a motivating example.

I have proposed a model called the Generalized Software Reliability Model (GSRM) to treat the uncertainties and dynamics regarding development processes and circumstances [17] and studies about predicting release time. I have proposed a method to predict the release time of open source software (OSS) by using GSRM [18] and agile development [50]. Additionally, I have implemented applying GSRM to company's datasets [15].

6.2.1 Software Reliability Growth Model (SRGM)

Although many software reliability models have been proposed, the most popular is the non-homogeneous Poisson process (NHPP) model. However, a recent study has suggested the Logistic model is the best model followed by the Gompertz model with regard to fitness [40]. In my study, I employ the Logistic model and Gompertz model using development data containing the number of faults detected for a given time. These models are common in Japan.

The equation of the Logistic model is given by

$$N_{\mathrm{L}}(t) = \frac{N_{\max}}{1 + \exp\{-A_{\mathrm{L}}(t - B_{\mathrm{L}})\}} \tag{6.1}$$

where $N_{\mathrm{L}}(t)$ is the number of faults detected by time t. If $t \to \infty$, $N_{\mathrm{L}}(t)$ becomes $N_{\max}$. The parameters, $N_{\max}$, A_{L} and B_{L} can be calculated using R, which is a language and environment for statistical computing and graphics. The equation of the Gompertz model is given as

$$N_{\mathrm{G}}(t) = N_{\max} \exp(-A_{\mathrm{G}} {B_{\mathrm{G}}}^{t}) \tag{6.2}$$

where $N_{\mathrm{G}}(t)$ is the number of faults detected by time t. If $t \to \infty$, $N_{\mathrm{G}}(t)$ becomes $N_{\max}$. The parameters, $N_{\max}$, A_{G} and B_{G} can be calculated using R.

6.2.2 Project monitoring

Although several methods exist to monitor projects, there are several concerns in software development. The Engineering Project Management using the Engineering Cockpit is one method to manage and monitor project situations [29]. It provides developers and managers with the project specific information.

Nakai et al. studied how to identify the state of a project and the quality of a project based on GQM [3] and project monitoring [31]. They employed Jenkins, which is a continuous integration tool to visualize and collect fault data, lines of codes, test coverage, etc. They tried to judge the status of the project from the collected data based on the GQM method.

Ohira et al. developed the Empirical Project Monitor (EPM), which automatically collects and analyzes data that are versioning histories, mail

archives, and issue tracking records from multiple software repositories [35]. EPM provides graphs of collected and analyzed data to help developers and managers. However, EPM is not applicable to analyze SRGMs or to visualize the results.

6.3 Proposal to detect unexpected situations

I propose the following method to detect unexpected situations using software reliability growth models:

1. Separate faults into the phases that they are detected.
2. Apply SRGM daily to each fault.
3. Detect any unusual situations regarding the predicted number of faults.

The first step clarifies the fault data because the faults detected depend on the phase. For example, faults detected in a unit test relate to a specific module or feature, whereas faults detected in an integration test relate two modules or features. Consequently, the fault level depends on the phase that the fault is detected. Additionally, phases progress differently.

The second step applies the SRGM to the faults by phase to provide the detailed situation of each phase. Moreover, to monitor the behavior of SRGM, I apply it to each fault individually. In this Chapter, I apply SRGM daily to the data of two projects. I focused on the behavior about the predicted total numbers of faults, which is the model's parameter $N_{\max}$.

The third step monitors the behavior of the SRGM to detect unusual situations. In the motivating example, I mentioned that SRGM sometimes overestimates the expected results. It is assumed that unexpected situations occur in developments. In this Chapter, I assess when SRGM behaves unexpectedly in developments.

6.4 Evaluation and Results

I show the results for two projects on large-scale embedded software developed by Fujitsu Ltd. Herein these projects are identified as Project 1 and Project 2. These projects' qualities have been guaranteed by quality assurance divisions and sufficiently tested. Figures 6.3 and 6.4 show the number

of faults separated by the test phase when there are eight phases. Although the actual data of Project 1 and Project 2 contain more than eight phases, I treated only eight because the other phases do not have enough faults to model by SRGM.

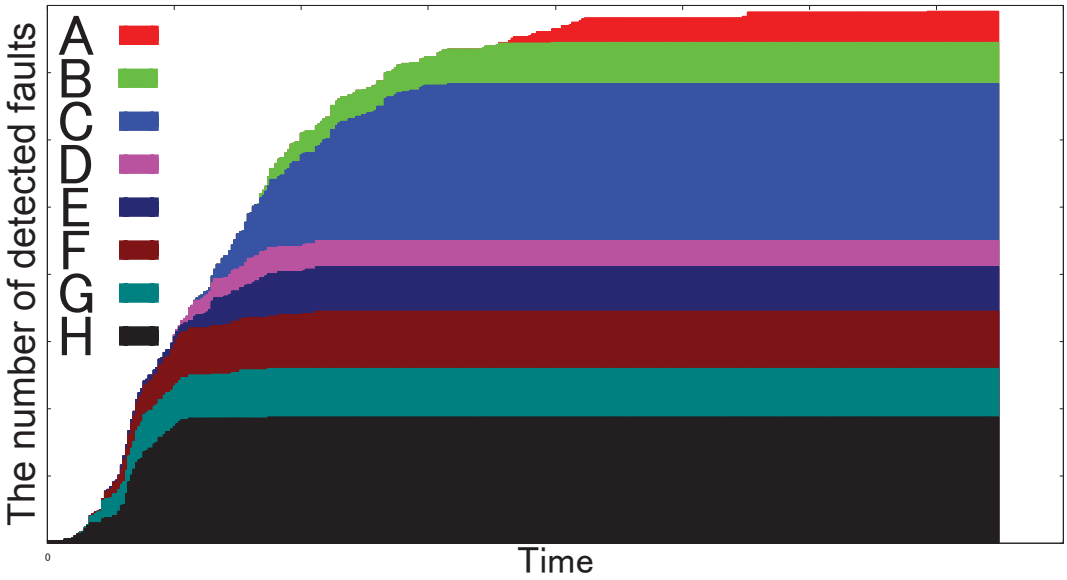


Figure 6.3: Cumulative number of detected faults for all of Project 1 represented as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

6.4.1 Fitness of model (RQ 1)

I evaluated the fitness of SRGMs in two cases. Case 1 applies SRGM to all the faults in the model simultaneously. Case 2 separates the faults the into eight test phases, applies SRGM to each phase, and then sums the results to treat as one model.

I show the results of Project 1 and 2 in Figs. 6.5 and 6.6, respectively. It is should be noted that these figures do not indicate actual values because the information is confidential. The separated faults model (case 2) provides a better fitness than the simultaneous model (case 1). Table 6.1 shows the residual sum of squares (RSS) ratio for each model.

These values are divided by the RSS value of case 1. Therefore, the separated model (case 2) for both projects indicates a good fitness because

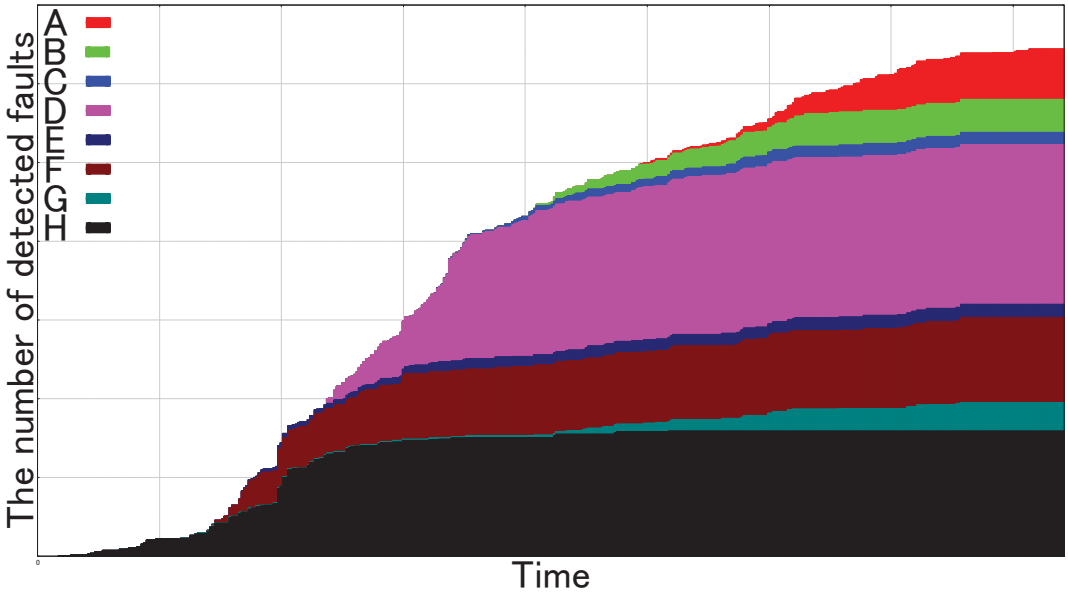


Figure 6.4: Cumulative number of detected faults for all of Project 2 represented as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

Table 6.1: Comparison of the simultaneous model (CASE 1) with the separated model (CASE 2) using RSS ratio datasets.

	Case 1	Case 2
Project 1	1.000	0.345
Project 2	1.000	0.190

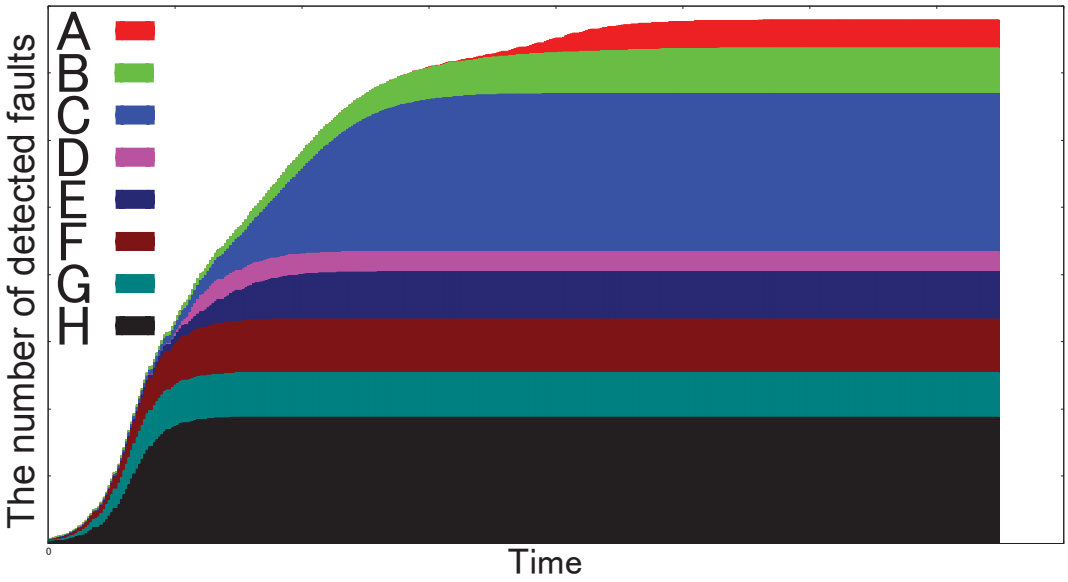


Figure 6.5: Cumulative number of predicted faults by SRGMs for all of Project 1 represented as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

the RSS values of the separated models (case 2) are smaller than those of the combined models (case 1).

6.4.2 Monitoring Predicted Faults (RQ 2)

I monitored the results of SRGMs by applying them daily to detect unexpected values. Figures 6.7 and 6.8 show the results for monitoring the maximum predicted number of faults for Project 1 and Project 2, respectively. Figure 6.7 has two irregular points when the maximum predicted number of faults is too large, whereas Fig. 6.8 has five irregular points when the maximum predicted number of faults is too large.

I interviewed the project manager about the situations when the graph indicates an irregular point. In figure 6.7, the first irregular point coincides with the time that developers thought it was difficult to continue on schedule because several problems remained. The second irregular point is when developers tried to reschedule the release plan because several problems reoccurred.

In figure 6.8, from the first irregular point to the third, several prob-

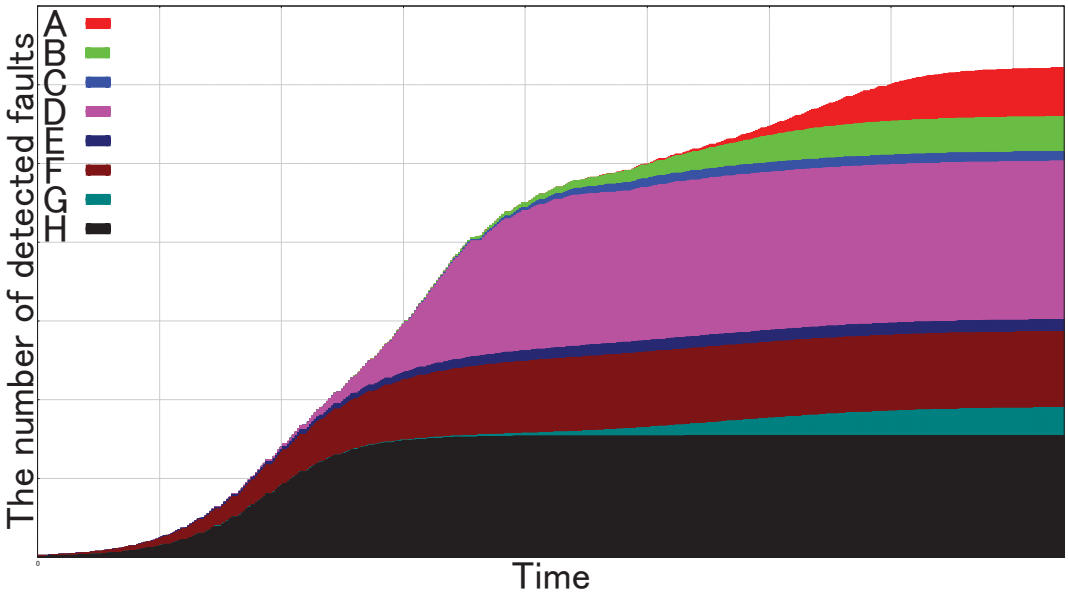


Figure 6.6: Cumulative number of predicted faults by SRGMs for all of Project 2 as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

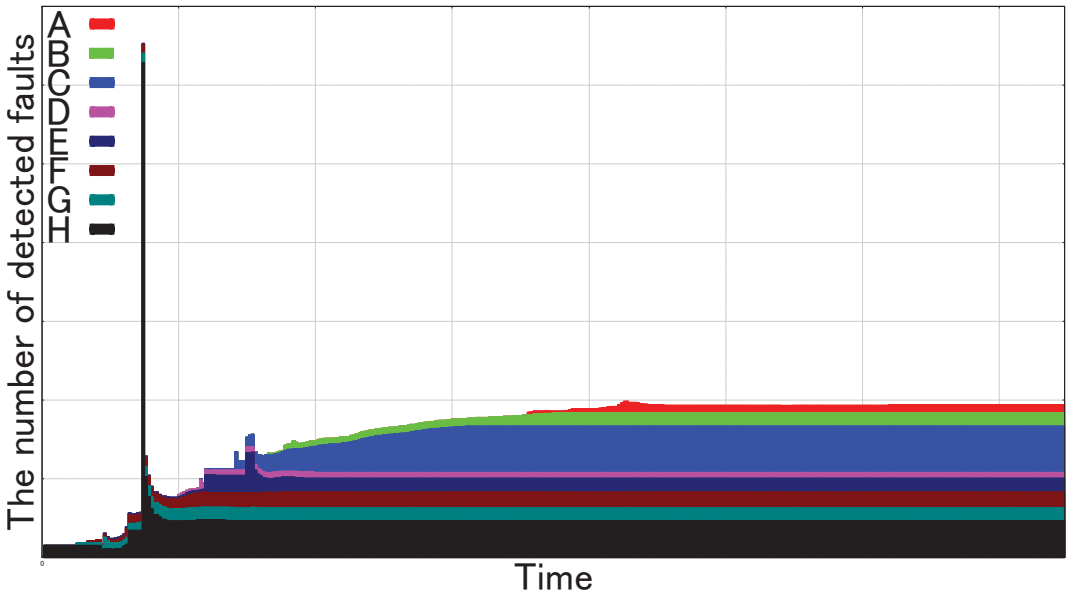


Figure 6.7: Cumulative maximum predicted number of faults by SRGMs for all of Project 1 as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

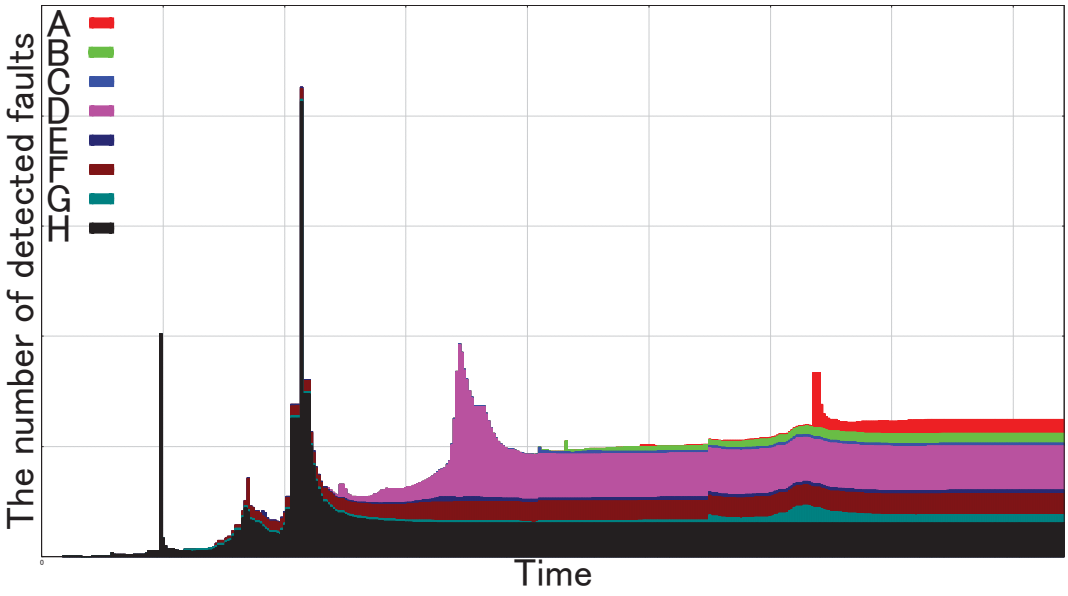


Figure 6.8: Cumulative maximum predicted number of faults by SRGMs for all of Project 2 as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

lems occurred intermittently. At the fourth irregular point, several problems reoccurred and developers stopped tests and restarted other tests. At the fifth irregular point, developers detected more faults than ever because they refined and created new test cases.

In the two projects, the irregular points are the same as the unexpected situations. Hence, the results show that monitoring the behavior should detect unexpected situations early.

6.4.3 Thread to validity

In this Chapter, I only treat two similar projects from the same organization. However, other researches have treated similar projects in the same organization. Additionally, my results found unexpected situations in the two projects, which I assumed are coincidental and are not related to this research.

6.5 Conclusion

I successfully obtained a good fitness model by separating faults by test phase and applying SRGM. Moreover, I found unexpected situations in development by monitoring the faults and the behavior of the SRGM. These results demonstrate that if developers and managers monitor the behavior of the SRGM results from the beginning of development, they can detect several unexpected situations earlier than ever.

To provide insight to developers and managers who have trouble with development, I plan to evaluate my method by applying it to ongoing projects and other datasets belonging to other domains or organizations.

In the Chapter 3, I assumed that the numbers of detected faults per unit time contained uncertainty and proposed GSRM which can treat uncertainty. In this Chapter, the unexpected situations are almost related to sudden increases of detected faults. This is a piece of evidence that the numbers of detected faults per unit time contain uncertainty. In the future work, I plan to analyze the relations between the unexpected situations and the uncertainties in developments.

Chapter 7

Project Management Using Cross Project Software Reliability Growth Model

7.1 Introduction to This Chapter

Over the past few decades, several companies have employed software reliability growth models (SRGMs) to evaluate reliability, which is an important component of software [8] [56] [55] [54] [4]. However, SRGMs have several issues. SRGMs sometimes misfit the actual data in ongoing developments. In addition, the results do not always match the developers' expectations.

If SRGMs misfit the actual data, the managers and developers will decide wrong plans, for example stopping testing early or release software which has not been tested enough. On the other hand, if the results of SRGMs indicate that the faults are enough detected or not enough detected contrary to the managers and developers' expectations, they will decide wrong plans too. If software is released with several faults left, the company which have released it will take time to debug it or cause damage or affect negatively to its users.

In order to avoid such misfitting and mismatching the developers' expectations, I propose new good accuracy SRGMs using person hours. I assume that SRGMs based on calendar time cannot realize accurate predictions, because many kinds of SRGMs treat calendar time, which includes holidays and non-testing time and does not reflect the actual efforts by developers.

7.1.1 Research Questions

This study aims to answer the following research questions:

1. RQ1: Do the results from SRGMs based on person hours differ from those based on calendar time?
2. RQ2: If the results differ, which model more precisely describes the relation between faults and detection time?
3. RQ3: Are there any metrics that can compare the progress against other developments?
4. RQ4: If such metrics exist, can they compare the progress between different developments?

My contributions are as follows:

- SRGMs based on person hours and calendar time are compared in nine empirical projects.
- A method to compare SRGMs is derived.
- A method to monitor the progress of a project in person hours is developed and implemented in test cases.

In this Chapter, I compared the SRGMs based on person hours and calender time in nine empirical projects. The results indicated the SRGMs based on person hours tend to be good fitting, so using SRGMs based on person hours would make precise plan.

7.2 Background

Several approaches have been proposed to measure reliability due to its importance when releasing software. Software development includes numerous uncertainties and dynamics regarding development processes and circumstances. This section explains SRGMs, their uncertainties and dynamics as well as provides a motivating example.

I have proposed a model called the Generalized Software Reliability Model (GSRM) to treat the uncertainties and dynamics regarding development processes and circumstances[17]. Previously, I have predicted the release times of open source software (OSS) using GSRM [18] and agile development [50]. Additionally, I have applied GSRM to company's datasets [15].

7.2.1 Software Reliability Growth Model (SRGM)

This section treats some example software reliability models, while the next section explains my model. Although numerous models have been proposed, the most famous is the non-homogeneous Poisson process (NHPP) model.

Some methods quantitatively assess software reliability from fault data observed in the software-testing phase using a software reliability model based on the counting faults [8] model. Similarly, my approach is also based on the counting faults model. By counting the faults and measuring the detection time, a software reliability model is formulated assuming that fault detection is based on a stochastic process. The NHPP model assumes that the stochastic process for the relationship between fault detection and detection time is a Poisson process. In actual developments, counting faults predicts the remaining faults and provides an indication about the end of the development.

First consider the general NHPP model, where the probability of detecting n faults is described as

$$Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!} \exp\{-H(t)\} \tag{7.1}$$

where $N(t)$ is the number of faults detected by time t, $H(t)$ is the expected cumulative number of faults detected [5]. Assuming that the total number of faults before testing is constant, N_{max}, the number of detected faults at a unit of time is assumed to be proportional to the remaining faults. These assumptions are expressed as

$$\frac{dH(t)}{dt} = c(N_{max} - H(t)) \tag{7.2}$$

where c is a proportionality constant. The solution to the above equation is

$$H(t) = N_{max}(1 - \exp(-ct)) \tag{7.3}$$

This model, which is called an exponential software reliability growth model, was originally proposed by Goel and Okumoto [9]. In this Chapter, I compare my model to this model.

Equation (7.3) provides an exponential shaped graph. However, in actual developments the number of faults curve does not fit the exponential shaped; it usually fits a logistic curve or Gompertz curve [2], which are more complex than an exponential shaped graph. Consequently, I propose a new model

that can fit a logistic curve or an exponential shaped curve for use in actual developments.

Although many software reliability models have been proposed, the most popular is the non-homogeneous Poisson process (NHPP) model, but a recent study has suggested that the Logistic model followed by the Gompertz [36] model are the most suitable with respect to fitness [40]. In this study, I employ the Logistic model and the Gompertz model using development data containing the number of faults detected for a given time. These models are common in Japan.

The Logistic model is expressed by

$$N_{\mathrm{L}}(t) = \frac{N_{\max}}{1 + \exp\{-A_{\mathrm{L}}(t - B_{\mathrm{L}})\}} \tag{7.4}$$

where $N_{\mathrm{L}}(t)$ is the number of faults detected by time t. If $t \to \infty$, $N_{\mathrm{L}}(t)$ becomes $N_{\max}$. The parameters, $N_{\max}$, A_{L} and B_{L} can be calculated using Levenberg-Marquardt method with R, which is a language and environment for statistical computing and graphics.

The Gompertz model is given by

$$N'_{\mathrm{G}}(t) = N'_{\max} \exp(-A'_{\mathrm{G}} {B'_{\mathrm{G}}}^{t'}) \tag{7.5}$$

where $N_{\mathrm{G}}(t)$ is the number of faults detected by time t. If $t \to \infty$, $N_{\mathrm{G}}(t)$ becomes $N_{\max}$ $(0 < B_{\mathrm{G}} < 1)$. The parameters, $N_{\max}$, A_{G} and B_{G} can be calculated using Levenberg-Marquardt method with R.

7.2.2 Project monitoring

Although multiple methods exist to monitor projects, there are several concerns in software development. The Engineering Project Management using the Engineering Cockpit is one method to manage and monitor project situations [29]. It provides developers and managers with project specific information.

Nakai *et al.* studied how to identify the state and the quality of a project based on goal, question, metric (GQM) [3] and project monitoring [31]. They employed Jenkins, which is a continuous integration tool to visualize and collect fault data, lines of codes, test coverage, etc. Then they evaluated the project status using the collected data based on the GQM method.

Ohira *et al.* developed the Empirical Project Monitor (EPM), which automatically collects and analyzes data from versioning histories, mail archives, and issue tracking records from multiple software repositories [35]. EPM provides graphs of the collected and analyzed data to help developers and managers. However, EPM is not applicable to analyze SRGMs or to visualize the results.

7.2.3 Motivating example

Figures 7.1 and 7.2 show my motivating examples. Figure 7.1 indicates that the number of faults based on calendar time, which includes holidays and non-testing time, does not reflect the actual efforts by developers. I hypothesize that SRGMs based on calendar time cannot realize accurate predictions. Herein the accuracy of a model based on calendar time is compared to that based on person hours by evaluating nine projects developed by Sumitomo Electric Industries, Ltd.

Figure 7.2 shows an example of two fault datasets, where the x-axis represents the calendar time and the y-axis represents the number of faults. For comparison, the scales of the x- and y-axes are the same for Project P2 and Project P5. Therefore, the total time is longer and more faults are detected for Project P2 than Project P5.

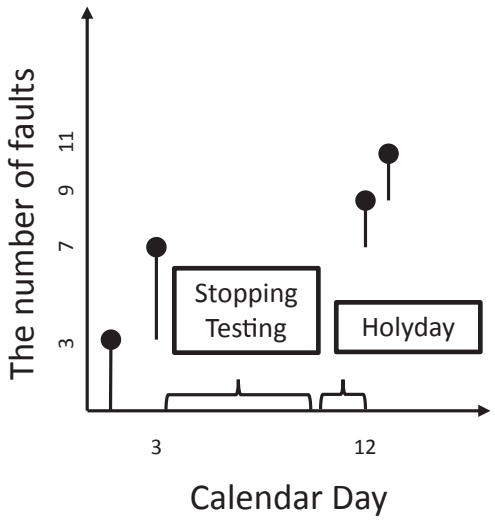
However, criteria to compare these projects do not exist because each project depends on its lines of code, domain, developers, budget, etc. In this Chapter, I evaluate the fault density as functions of calendar time, person hours, and number of tested cases. This evaluation realizes a method to compare different projects.

7.3 Proposal to compare SRGM between projects

I propose an extension of SRGM to cover fault densities as well as a method to apply the person hours to SRGMs.

Date	Fault	Day
8/May/2015	3	1
9/May/2015	0	2
10/May/2015	4	3
11/May/2015	0	4
12/May/2015	0	5
13/May/2015	0	6
14/May/2015	0	7
15/May/2015	0	8
16/May/2015	0	9
17/May/2015	0	10
18/May/2015	0	11
19/May/2015	2	12
10/May/2015	2	13

Plot



Date	Fault	PH
8/May/2015	3	4.25
9/May/2015	0	0
10/May/2015	4	5
11/May/2015	0	0
12/May/2015	0	0
13/May/2015	0	0
14/May/2015	0	0
15/May/2015	0	0
16/May/2015	0	0
17/May/2015	0	0
18/May/2015	0	0
19/May/2015	2	9.75
10/May/2015	2	12.25

Plot

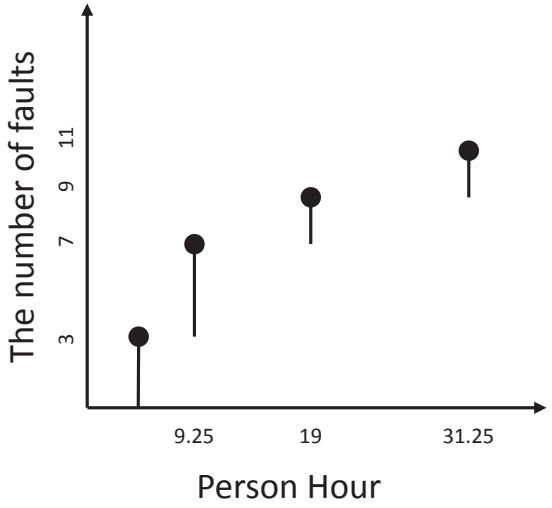


Figure 7.1: Examples of calendar time and person hours.

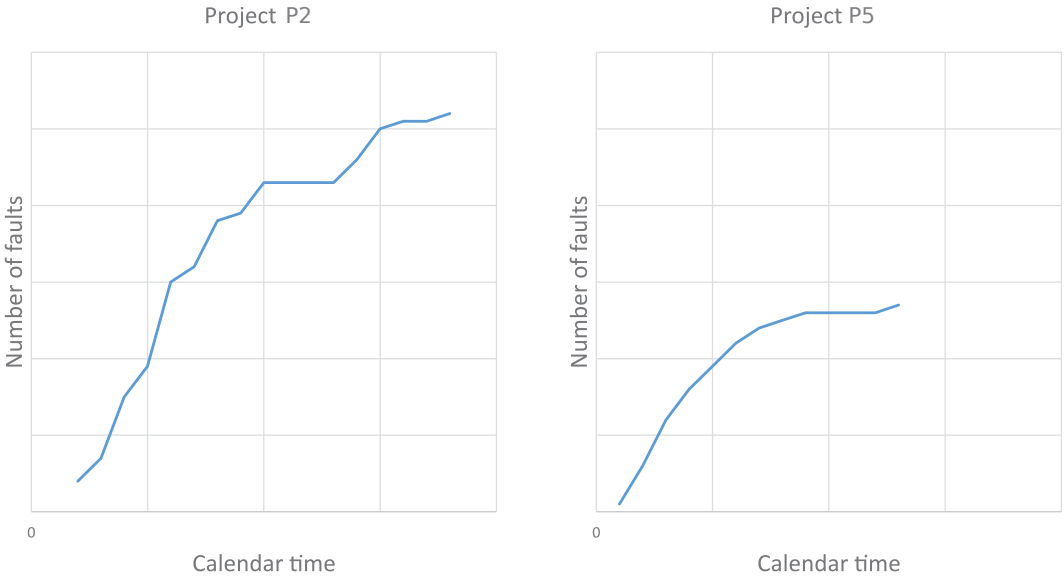


Figure 7.2: Example of a comparison between projects.

7.3.1 Extension of SRGM to fault densities

The equation of the Logistic model for fault densities and rates of used person hours is given by

$$D_{\mathrm{L}}(t') = \frac{D_{\max}}{1 + \exp\{-A'_{\mathrm{L}}(t' - B'_{\mathrm{L}})\}} \tag{7.6}$$

where $D_{\mathrm{L}}(t)$ is the fault density by the rate of used person hours t'. If $t' \to \infty$, $D_{\mathrm{L}}(t)$ becomes $D_{\max}$. The parameters, $D_{\max}$, A'_{L} and B'_{L} can be calculated using Levenberg-Marquardt method with R. The equation of the Gompertz model for fault densities and rates of used person hours is given as

$$D_{\mathrm{G}}(t') = D_{\max} \exp(-A'_{\mathrm{G}} {B'_{\mathrm{G}}}^{t'}) \tag{7.7}$$

where $D_{\mathrm{G}}(t')$ is the number of faults detected by the rate of used person hours t'. If $t' \to \infty$, $D_{\mathrm{G}}(t')$ becomes $D_{\max}$ $(0 < B'_{\mathrm{G}} < 1)$. The parameters, $D_{\max}$, A'_{G} and B'_{G} can be calculated using Levenberg-Marquardt method with R.

7.3.2 Comparison of projects

Figure 7.3 overviews my method, which compares the results of SRGMs between projects with different lines of code, numbers of test cases, total

person hours, and numbers of faults. My method has three steps:

1. Divide the number of detected faults by the created lines of code for all data. Convert the person hours to the rate of used person hours.
2. Merge all data into one dataset. Rearrange it into chronological order.
3. Apply SRGM to the new dataset.

I consider the SRGM from the new dataset as a leveled SRGM of all datasets.

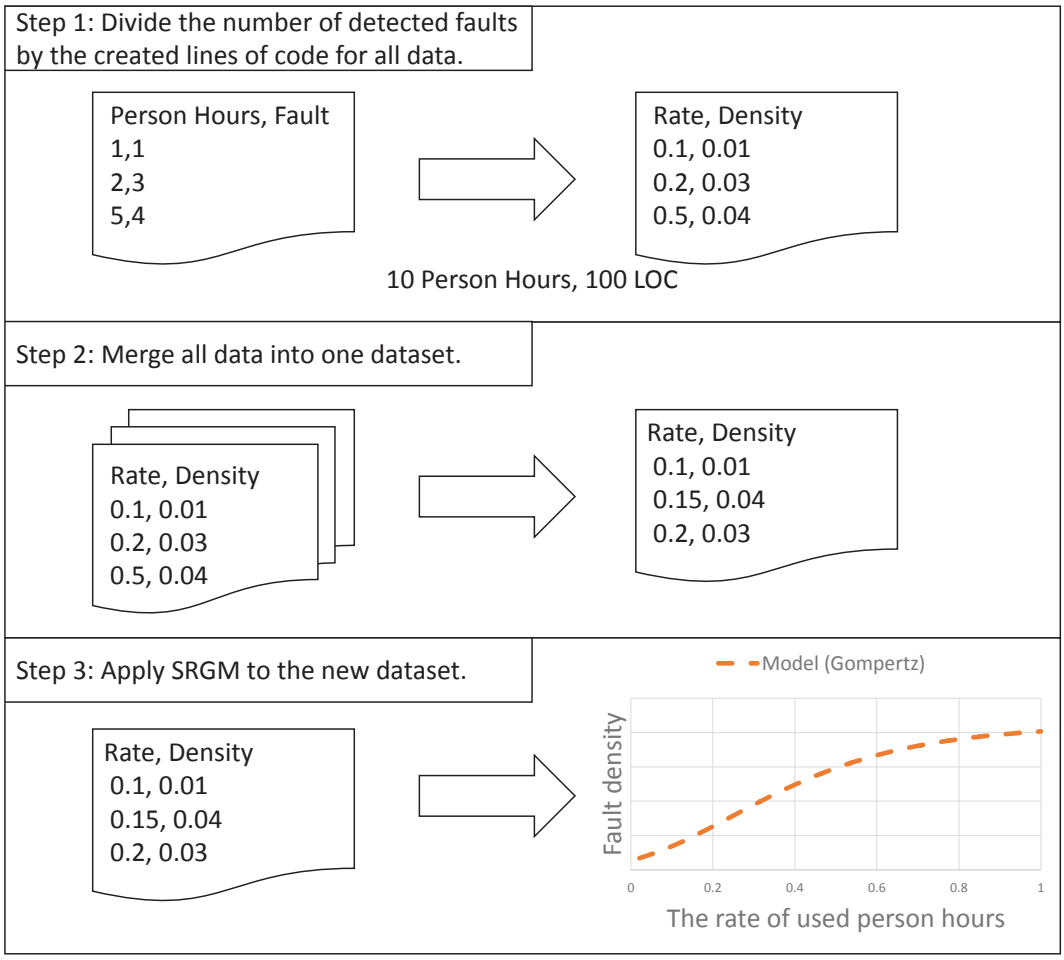


Figure 7.3: Overview to compare the results of SRGM between projects.

The first step converts the fault data of each project into the fault density and the rate of used person hours because the numbers of faults and the terms

depend on the project. For example, consider a scenario where 100 person hours are required to detect 20 faults in Project 1, but 50 person hours are necessary to detect 10 faults in Project 2. If only the number of faults and person hours are treated, the effort of the developers and the difficulty of project cannot be evaluated. Additionally, I assume that the fault densities and the rates of used person hours are values that can be used to compare and monitor projects because the fault densities values are the same and the rate of used person hours converge.

The second step merges the converted dataset into one dataset to create an averaged SRGM. Moreover, to model the merged dataset, the data is rearranged in chronological order. This study models the dataset to SRGM by a nonlinear least-squares method through R.

The third step applies the merged dataset to SRGM based on the fault densities and the rate of used person hours. The results indicate the leveled line of development, which can be used to help managers and developers assess the progress of a development. If the dataset for a development strays from the leveled line, it means that the development is not going well at that time.

7.4 Evaluation and Results

I evaluated my method via case studies. Then I applied my proposed method to datasets from nine projects developed by Sumitomo Electric Industries, Ltd. using the same framework.

7.4.1 Evaluation design and result

To answer RQ1 (Do the results from SRGMs based on person hours differ from those based on calendar time?) and RQ2 (If the results differ, which model more precisely describes the relation between faults and detection time?), I compared the differences between models based on calendar time and person hours. Specifically, I applied the Logistic model and the Gompertz model to nine project datasets using calendar time and person hours. Then I calculated the residual sums of square (RSS) for each model and compared the results. RSS indicates the differences between actual data and a model. A small RSS value indicates the model is a good fit for the actual data.

To answer RQ3 (Are there any metrics that can compare the progress against other developments?) and RQ4 (If such metrics exist, can they compare the progress between different developments?), I compared the correlations between the metrics in the collected datasets, the lines of code that only developers created, the numbers of faults, the number of test cases estimated by developers, the number of test cases that developers tested, calendar time, and person hours. Specifically, I evaluated the correlations between the metrics and then applied the Logistic model and the Gompertz models, which are based on the fault density, person hours, and test cases, to the calculated RSS for each model. Finally, I compared the correlation to answer RQ3. Then I compared the RSS results and interviewed the managers to quantitatively and qualitatively answer RQ4.

In this evaluation, I collect nine projects data from Sumitomo Electric Industries, Ltd. including function points, lines of code, number of fault, a number of estimated test cases and the time series of detected fault days, a number of implemented test cases, and the person hours. These projects are developed for business applications throgh web from 2013 to 2015 with a same framework which has been developed by Sumitomo Electric Industries, Ltd.

Comparison between calendar time and person hours

I compared SRGMs based on calendar time (Figure 7.4) to those based on person hours (7.5). In Figure 7.4 (7.5), the x-axis represents the calendar time (person hours) and the y-axis represents the number of faults. In Figure 7.6, the x-axis represents the number of implemented test cases and the y-axis represents the number of faults. The legends, which are the same in Figs. 7.4 – 7.6, indicate the nine project datasets, which are labeled P1 to P9.

Figures 7.4 and 7.5 suggest that there is a difference in understanding the growth of faults between calendar time and person hours. To clearly understand this difference, I applied SRGMs (the Logistic model and the Gompertz model) to the calendar datasets and person hour datasets, and evaluated the residual sum of squares (RSS), which is typically used to evaluate the differences between the data and the model (Table 7.1). The model with the best fit has the smallest RSS. For all datasets, SRGMs based on calendar time produce larger RSSs than SRGMs based on person hours.

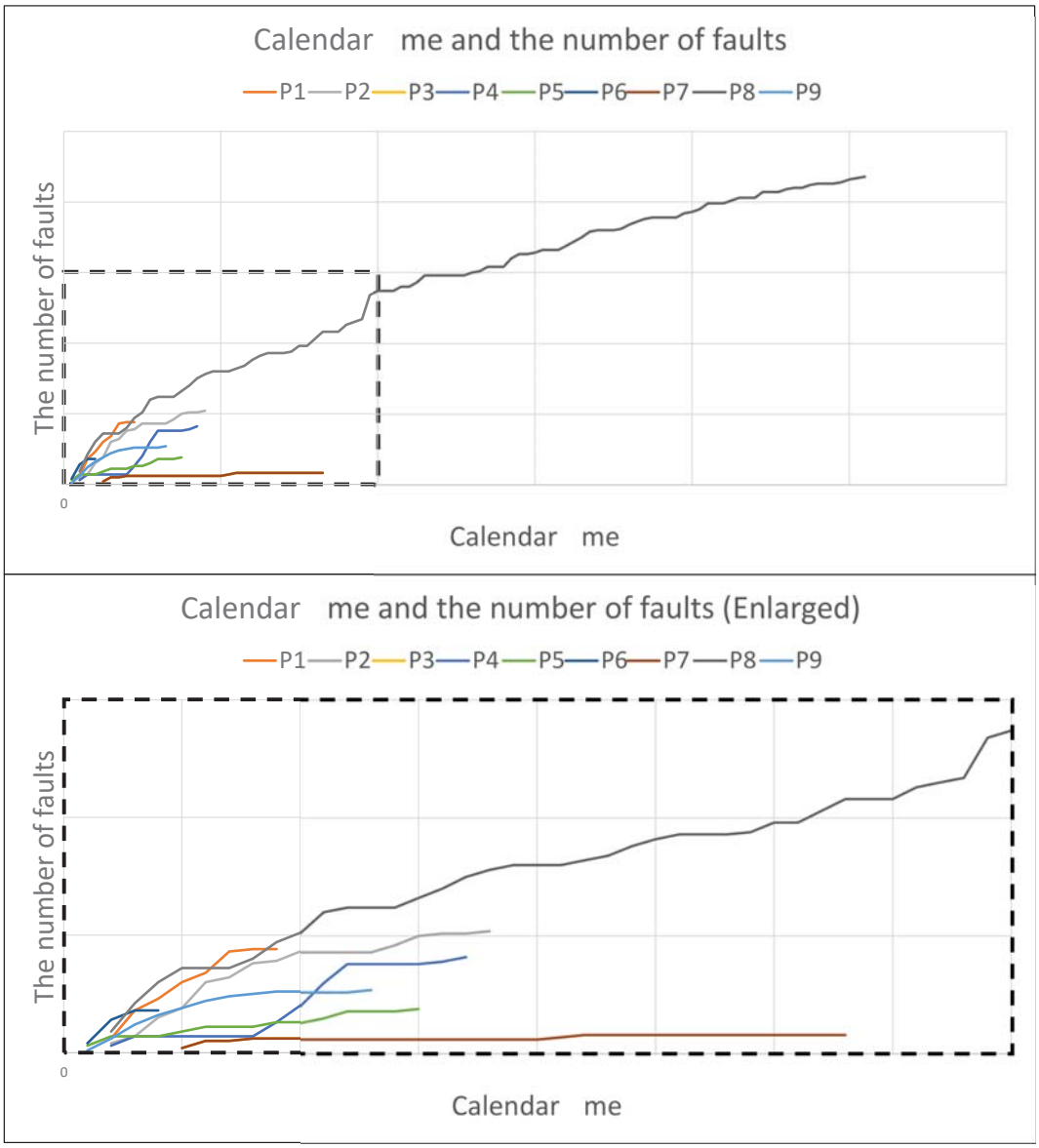


Figure 7.4: Relation of the number of faults and calendar time.

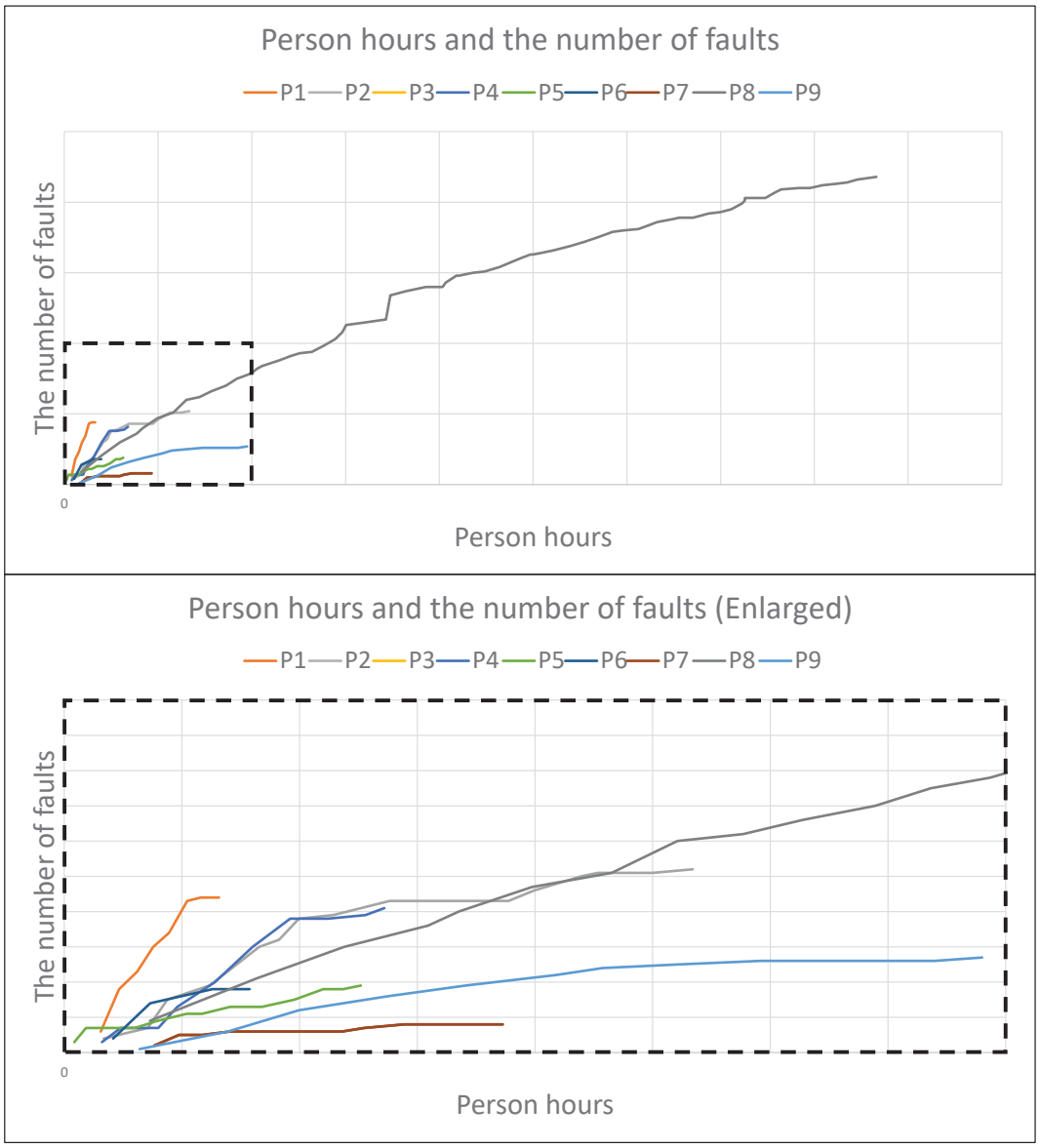


Figure 7.5: Relation of the number of faults and person hours.

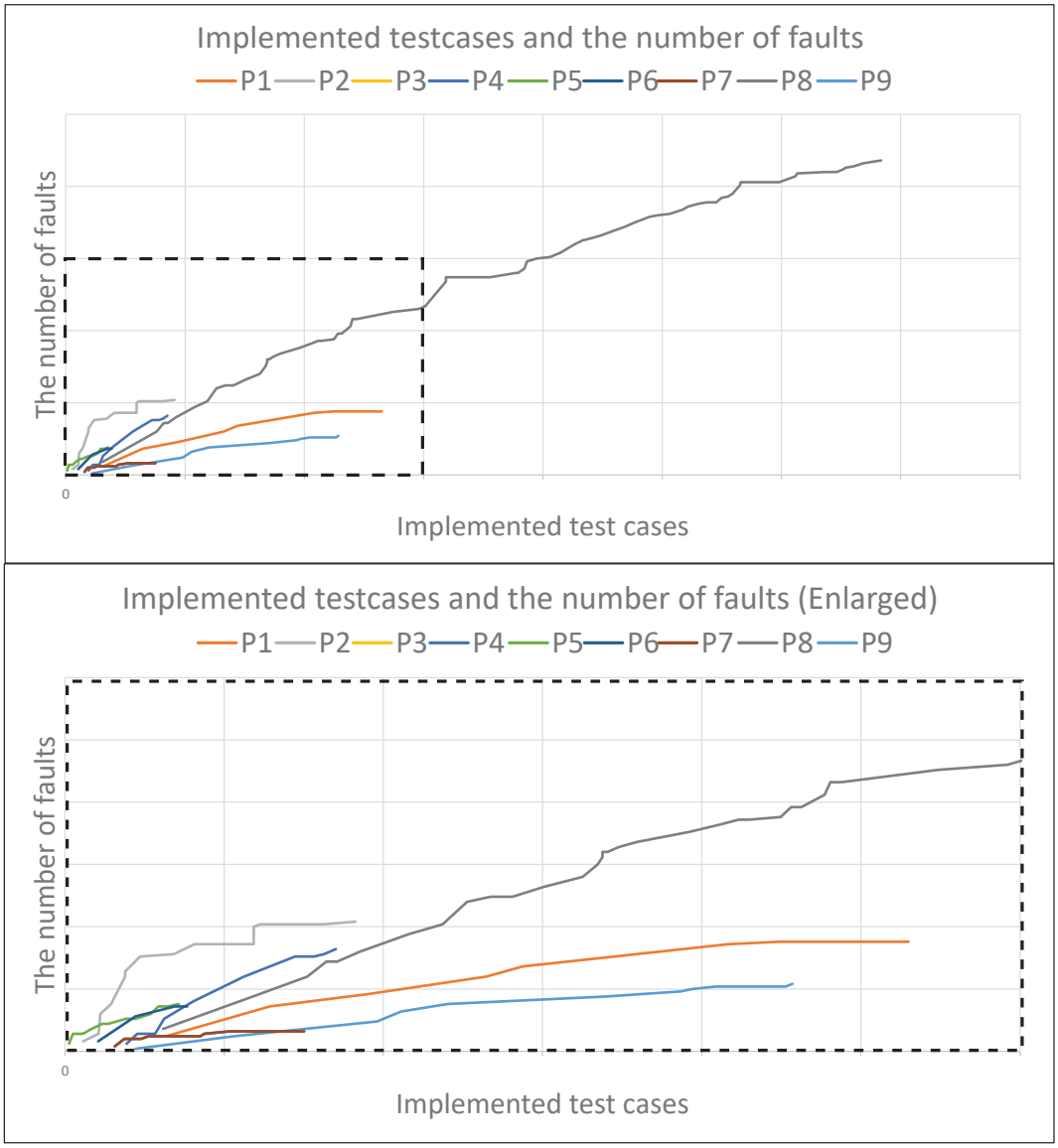


Figure 7.6: Relation of the number of faults and implemented test cases.

Table 7.1: Comparison of SRGMs based on calendar time and person hours

Project	RSS (Calendar Time)		RSS (Person Hours)	
	Logistic	Gompertz	Logistic	Gompertz
P1	122.99	115.07	26.99	23.06
P2	130.38	100.66	113.5	85.75
P3	12.82	12.65	6.911	6.317
P4	177.1	NaN	35.26	68.97
P5	13.4	12.91	11.57	10.95
P6	0.08373	0.2568	0.002522	0.01944
P7	12.82	12.65	6.911	6.317
P8	3993	2460	2936	1408
P9	34.19	28.75	11.13	4.284

Table 7.2: Comparison of SRGMs based on calendar time and person hours

Project	Calendar Time / Person Hours	
	Logistic	Gompertz
P1	0.2194	0.2004
P2	0.8705	0.8519
P3	0.5391	0.4994
P4	0.1991	NaN
P5	0.8634	0.8482
P6	0.0301	0.0757
P7	0.5391	0.4994
P8	0.7353	0.5724
P9	0.3255	0.1490

Comparison of values

To evaluate the correlations between each value, I compared the function points (FP), lines of code (LOC), number of fault (Fault), fault densities of LOC (Fault/LOC), fault densities of FP (Fault/FP), number of estimated test cases (TC), number of implemented test cases (TC), total calendar time, and total person hours (Table 7.3). All the values are from the end of the project. Because the datasets are developed within the same framework, the FP represents the function points targeting the extended function. Similar to FP, the developers created the LOC. In Table II, the Fault correlation values of the person hours, Implemented TC, and Calendar time are large (0.95582, 0.93433, and 0.92996, respectively), indicating that the number of faults is strongly related to person hours, number of implemented test cases, and calendar time.

Compare the projects

I compared the results of SRGMs based on person hours to SRGMs based on the implemented test cases for the nine projects (Table 7.4).

Rates of used person hours

Figure 7.7 shows the results of the model using fault densities and the rate of used person hours, where the x-axis represents the rate of used person hours and the y-axis represents the fault density. The legend indicates the nine project datasets, which are labeled as P1 to P9.

Rates of tested cases

Figure 7.8 shows the results of the model using the fault densities and the rate of tested cases, where the x-axis represents the rate of implemented test case and the y-axis represents the fault density. The legend is the same as Figure 7.7.

Table 7.3: Correlations between each value

	FP	LOC	Fault
FP	1	0.9008	0.93883
LOC	0.9008	1	0.77178
Fault	0.93883	0.77178	1
Fault/LOC	0.12692	-0.12702	0.42154
Fault/FP	-0.1931	-0.30505	0.14561
Estimated TC	0.7648	0.63311	0.77432
Implemented TC	0.91526	0.72619	0.93433
Calendar time	0.93732	0.81877	0.92996
Person hours	0.98159	0.83476	0.95582
	Fault/LOC	Fault/FP	Estimated TC
FP	0.12692	-0.1931	0.7648
LOC	-0.12702	-0.30505	0.63311
Fault	0.42154	0.14561	0.77432
Fault/LOC	1	0.83274	0.43197
Fault/FP	0.83274	1	0.02498
Estimated TC	0.43197	0.02498	1
Implemented TC	0.39555	0.01007	0.93057
Calendar time	0.22485	-0.05423	0.76607
Person hours	0.20124	-0.1113	0.7607
	Implemented TC	Calendar time	Person Hours
FP	0.91526	0.93732	0.98159
LOC	0.72619	0.81877	0.83476
Fault	0.93433	0.92996	0.95582
Fault/LOC	0.39555	0.22485	0.20124
Fault/FP	0.01007	-0.05423	-0.1113
Estimated TC	0.93057	0.76607	0.7607
Implemented TC	1	0.90194	0.92417
Calendar time	0.90194	1	0.9478
Person hours	0.92417	0.9478	1

Table 7.4: Comparison of SRGMs based on person hours and implemented test cases

Project	RSS (Person Hours)		RSS (implemented test cases)	
	Logistic	Gompertz	Logistic	Gompertz
P1	26.99	23.06	20.59	15.73
P2	113.5	85.75	200.6	176.9
P3	6.911	6.317	6.852	6.200
P4	35.26	68.97	29.52	25.31
P5	11.57	10.95	15.33	13.94
P6	0.002522	0.01944	0.02495	0.06991
P7	6.911	6.317	6.852	6.200
P8	2936	1408	4095	2469
P9	11.13	4.284	13.27	9.947

Table 7.5: Comparison of SRGMs based on person hours and implemented test cases

Project	Person Hours / implemented test cases	
	Logistic	Gompertz
P1	1.170	1.120
P2	1.324	0.427
P3	1.094	0.922
P4	0.511	2.336
P5	1.056	0.714
P6	0.1297	0.7792
P7	1.094	0.9219
P8	2.085	0.3438
P9	2.598	0.3228

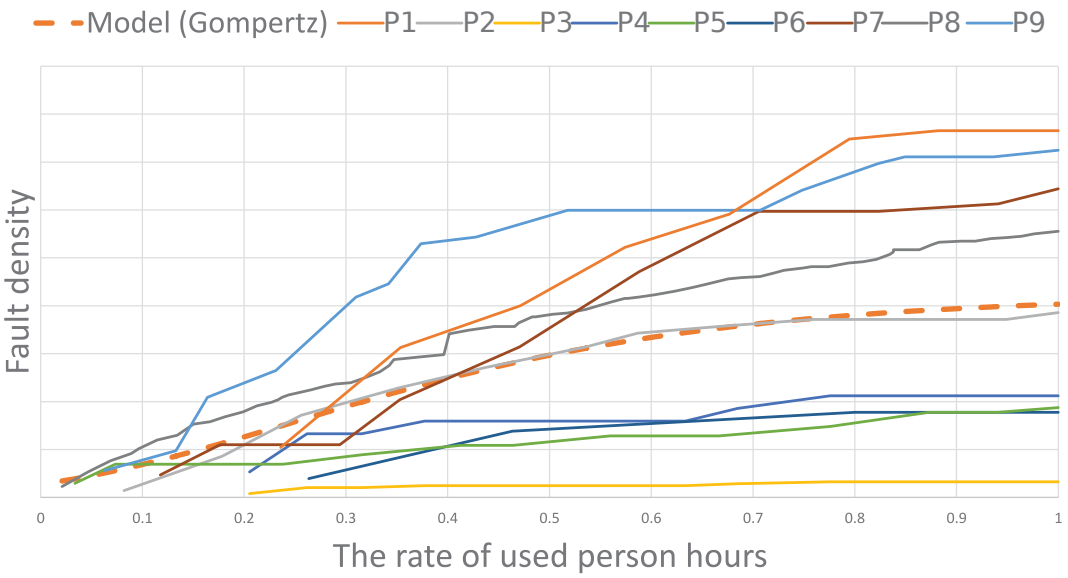


Figure 7.7: Results of the fault densities and the rates of used person hours.

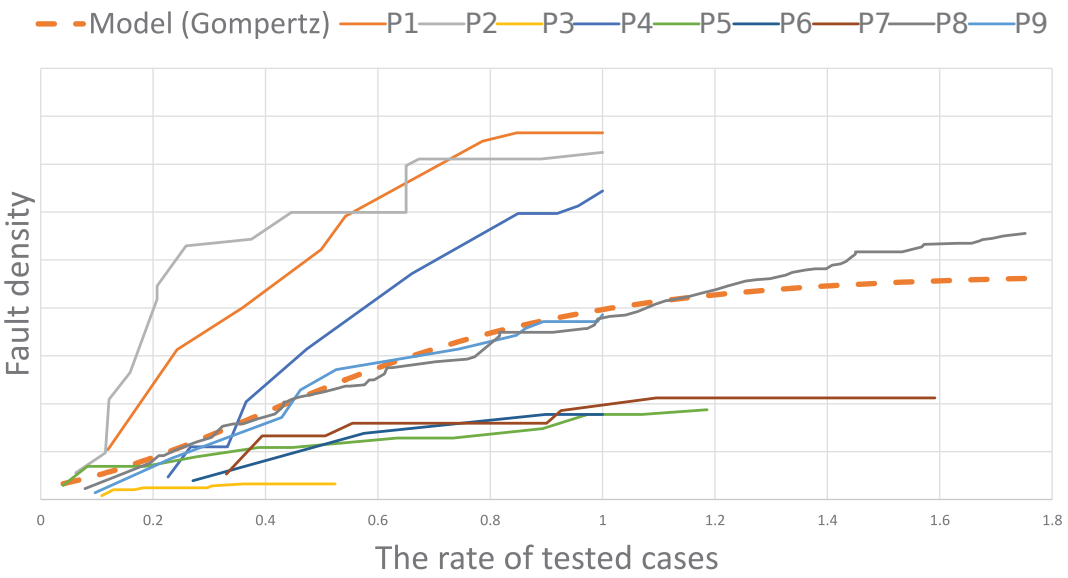


Figure 7.8: Results of the fault densities and the rates of used person hours.

7.4.2 Discussion

RQ1 (Do the results of SRGMs using person hours differ from those using calendar time?)

SRGMs based on calendar time and those based on person hours produce different results, implying that SRGMs based on person hours only reflect actual efforts and do not consider non-working time. Table 7.1 shows that all the RSSs for all projects differ based on person hours and calendar times, demonstrating that SRGMs based on person hours and calendar time have unique features (RQ1). However, several projects have similar RSSs between calendar time and person hours, indicating that developers worked on the project during holidays and did not stop testing.

RQ2 (How do they differ?)

SRGMs based on person hours are more precise than SRGMs based on calendar time. For all dataset, the RSSs based on person hours are lower than those based on calendar time. Table 7.1 indicates that Project P6 (P2) has the greatest (smallest) decrease of about 97% (13%) when using person hours. For Project P6, the RSS of the Logistic model in calendar time is 0.08373, and the RSS of the Logistic model in person hours is 0.002522. The value of RSS of person hours divided by calendar time is around 3%. On the other hand, the RSS of the Logistic model in calendar time for Project P2 is 130.38 and the RSS of the Logistic model in person hours is 113.5. The value of RSS of person hours divided by calendar time is around 87%. These results confirm that there is difference between SRGMs based on calendar time and person hours (RQ2).

RQ3 (Do specific metrics evaluate progress?)

The number of faults is significantly related to person hours and tested cases in the nine project datasets, suggesting that the person hours required to determine the number of faults can be modeled. The largest correlation coefficient with the number of faults is Person Hours, which is 0.95582 followed by Function Point (0.93883), Implemented test cases (0.93433), and Calendar time (0.92996). Except for Function Point, I monitored the values as a time series. Because the Function Point occurs at the beginning of development, estimating the number of faults is useful using Function Points.

Thus, specific metrics can be used to evaluate progress (RQ3).Because the fault density and rates of used person hours are related, SRGMs based on the faults density, rate of used person hours, and rate of used tested cases provide better fitting models than those based on calendar time. Moreover, the nine managers that I interviewed indicated that the leveled lines should assist in confirming progress. Table 7.4 shows that the fitness of SRGMs based on person hours and implemented test cases depends on the project. For Project P9, the RSS of the Logistic model in person hours is 11.13, and the RSS of the Logistic model in implemented test cases is 13.27. The value of the RSS of person hours divided by implemented test cases is around 260%, which is the largest rate in Table 7.4.

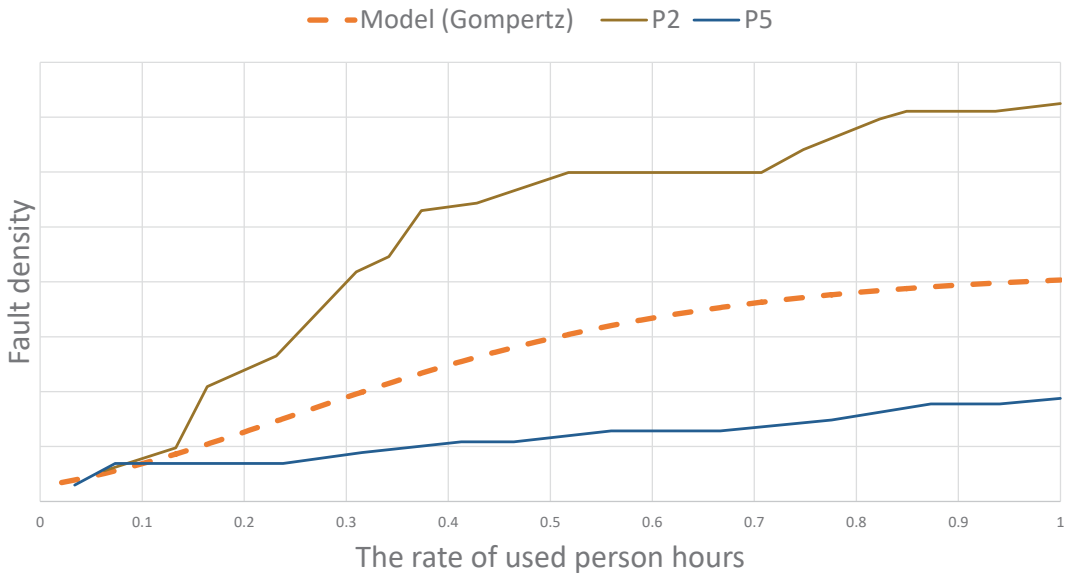


Figure 7.9: Fault densities and rates of used person hours for project B and E and the leveled Gompertz model

RQ4 (Can progress of projects be directly compared?)

Figure 7.2 shows an example of the fault densities and the rates of person hours for Projects P2 and P5 as well as the leveled lines of the Gompertz model through nine projects. Although the two projects cannot be compared in Figure 7.9 due to differences in the x- and y-axes, the axes in Figure 7.2 are the same. Additionally, leveled lines can be prepared for the projects,

demonstrating that the progress of different projects can be directly compared (RQ4).

Moreover, I have introduced a system for using my method in Sumitomo Electric Industries, Ltd. and made a request to several managers and developers for using my method. After several months, I had interviewed the managers and developers who had used my method for about 3 hours. My interviews about the models indicate that leveled SRGMs provide useful information about the progress of the projects.

7.4.3 Limitations

Internal validity threats

I treated only three relations between the numbers of faults and calendar time, between the numbers of faults and person hours, and between the numbers of faults and implemented. The number of faults can be related to other factors, for example, the skills of developers. In this research, I was not able to collect and evaluate the skills of developers and other data. The skills of developers influence the speeds of detections of faults because experts can detect the fault earlier than beginners detect. In the comparison, I used nine datasets from the same company. Therefore, the data may contain mistakes or other false elements. Moreover, the data contains several domains.

External validity threats

I only tested SRGMs based on person hours with nine datasets, which is insufficient to make generalizations about SRGMs based on person hours. Additionally the nine datasets which I collected are developed by a waterfall model. I could not evaluate the differences of development styles.

7.5 Related work

Many different types of software reliability growth models exist. Yamada *et al.* proposed an extend NHPP model, which is related to the test-domain dependence [58]. Test-domain dependent models include the notion that the tester's skills should improve by degrees; thus, skills grow over time. The test-domain-dependent model adds additional assumptions to the NHPP model.

Fujii *et al.* developed a quantitative software reliability assessment method via incremental development processes, which is a type of agile software development based on familiar non-homogeneous Poisson processes [6]. Fujii *et al.* used both the number of faults and software metrics to demonstrate software reliability predictions via a case study.

Rana *et al.* study about improving SRGMs predictive accuracy using historical projects data [41]. They collected defect report data from three companies. They applied several SRGMs to a past project's defect report data from one company and applied the SRGMs to another data from the same company with the parameters which were obtained from the past project. They concluded that the SRGMs using the past project were more accurate than making asymptote predictions without using such information.

Several researchers study about the cross project fault prediction and management. Zimmermann *et al.* studied cross-project defect prediction models in points of lines of codes and bugs and other factors [60]. For 12 real-world applications including open source software and enterprise software, they ran cross-project predictions using finished projects datasets. In this Chapter, I focus on the time series of projects and monitoring the projects in points of person hours and implemented test cases.

Kuo *et al.* proposed a new scheme for constructing software reliability growth models based on NHPP [26]. Their scheme provides a model which considers testing efforts and fault detection rates. They estimate the testing efforts and predict the trends of fault detection rates which can be obtained from historical records of previous releases or other similar software projects. My method did not need the estimation of testing efforts and used the projects developed by the same company and using the same framework.

7.6 Conclusion

Using SRGMs based on person hours, I successfully modeled nine actual datasets. SRGMs based on person hours can more precise model compared to SRGMs based on calendar time. SRGMs based on person hours are between 13% and 97% more precise than those based on calendar time.

Moreover, I propose leveled SRGMs based on the fault density and the rates of person hours as well as the rates of used test cases. My interviews of managers about the models indicate that leveled SRGMs provide useful information about the progress of the projects.

In the future work, I plan to apply this method to GSRM. To apply this method to GSRM makes it possible to compare the uncertainties of projects.

Chapter 8

Project Management Using Cross Project Software Reliability Growth Model Considering System Scale

8.1 Introduction to This Chapter

Several researchers have proposed software reliability growth models (SRGMs), which have been used to assess and predict software reliability. These models, which are applied to one project dataset, predict the number of faults that will be detected. Prior to developing such a model, several faults must be identified. In industrial studies, managers often want to predict the number of faults in a current project based on previous projects in the same domain and scale. However, previous models do not always predict the new project. Moreover, if a project does not have the same domain and scale as a past project, a previous model cannot be applied. In such situations, managers and developers cannot determine when to end the test phases or release a project.

I proposed a cross project SRGM to monitor a project by comparing it other past projects [12]. My method creates a leveled SRGM from old project datasets and helps managers and developers decide when to end the test phases or release a project by comparing the situation of the new project and a leveled SRGM. Since the leveled SRGM contains all kinds of projects,

not all projects can be compared with the leveled SRGM. For example, since one project is always under the leveled SRGM, the managers of the project cannot decide to end test phases.

In this Chapter, I extend my method by classifying the projects contained within the leveled SRGM. Prior to the test phases, I selected system scale parameters such as the LOC, number of test cases, and test density (which is defined as the number of test cases divided by LOC) as classification parameters to create a leveled SRGM.

This study aims to answer the following research questions:

1. RQ1: Do the results from the classified leveled SRGMs differ from those of the unclassified SRGMs?

2. RQ2: If the results differ, which classification more precisely describes the results?

My contributions are as follows:

- Three types of classified SRGMs are compared in nine empirical projects.

- A method to monitor the progress of a project is derived.

In this Chapter, I classify and compare three leveled SRGMs in nine empirical projects. The results indicate that the leveled SRGMs classified by test density tend to be a good fit. Thus, employing leveled SRGMs classified by test density can help managers and developers determine when to each the test phases or release a project.

8.2 Motivating Example

Figure 8.1 shows the results of my method, which were obtained by a leveled SRGM from the datasets for nine projects developed by Sumitomo Electric Industries, Ltd. Leveled SRGMs do not seem appropriate for projects P2 and P5 because these projects are far from the leveled SRGM line.

In Figure 8.1, I show the results of my method. The results are obtained a leveled SRGM from the datasets for nine projects developed by Sumitomo Electric Industries, Ltd. It would seem to be not good for project P2 and P5 to use the leveled SRGM's since project B and E are far from the leveled SRGM's line.

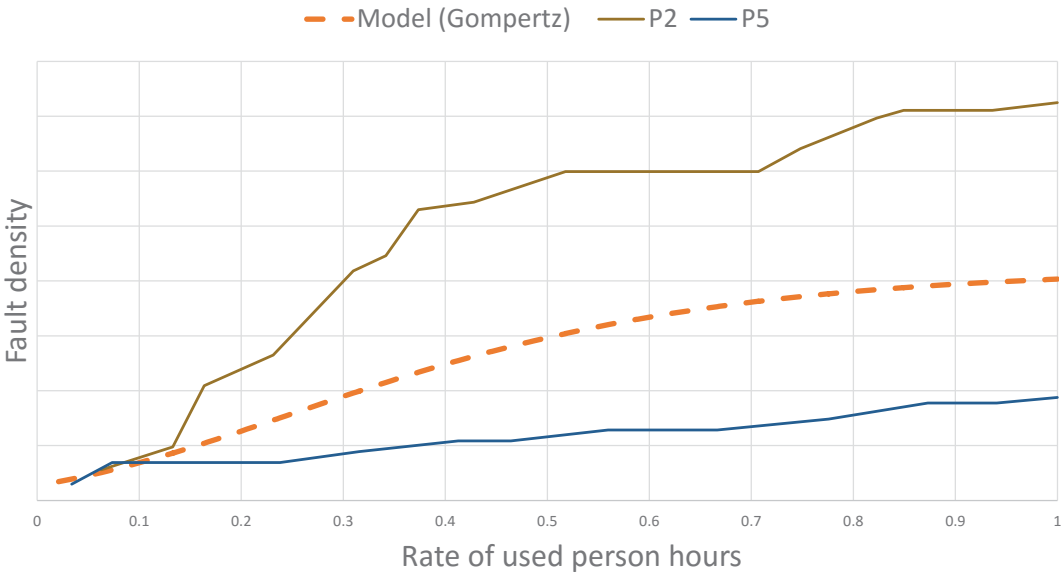


Figure 8.1: Fault densities and rates of used person hours for projects P2 and P5 and the leveled Gompertz model

8.3 Proposal of classified leveled SRGM considering system scale

I propose that a classified leveled SRGM considering the system scale should resolve the project dependency.

8.3.1 Comparison of projects

Figure 7.3 overviews my method, which compares the results of SRGMs between projects with different lines of code, numbers of test cases, total person hours, and number of faults. My method has three steps:

1. Divide the number of detected faults by the created lines of code for all data. Convert the person hours to the rate of used person hours.

2. Merge all the data into one dataset. Rearrange the data in chronological order.

3. Apply a SRGM to the new dataset.

I consider the SRGM from the new dataset to be a leveled SRGM of all datasets.

In order to consider the system scales, I classified projects into two groups by the median of lines of code, the number of estimated test cases, and test density before the first step.

8.4 Evaluation and Results

I evaluated my method via case studies. Then I applied my proposed method to the datasets from nine projects developed by Sumitomo Electric Industries, Ltd. using the same framework. It is should be noted that figures and tables do not indicate actual values because the information is confidential.

8.4.1 Evaluation design and result

To answer RQ1 (Do the results from the classified leveled SRGMs differ from those of the unclassified SRGMs?) and RQ2 (If the results differ, which classification more precisely describes the results?), I compared the differences between models classified by lines of code (LOC), the number of estimated test cases (test case), and test density. Specifically, I applied the Gompertz model to nine project datasets and classified them into two groups by the median of each value. Table 8.1 shows the details of projects. Then I calculated the residual sums of square (RSS) for each model and compared the results. RSS indicates the differences between the actual data and a model, where a small value indicates a good model fit.

In this evaluation, I collected data from nine projects from Sumitomo Electric Industries, Ltd., including lines of code, number of fault, number of estimated test cases, and the time series of detected fault in days and person hours. I compared the unclassified SRGM (Figure 3) to the SRGMs classified by LOC (Figure 8.3), test case (Figure 8.4), and test density (Figure 8.5). In Figures Figure 8.2 – 8.5, the x-axis represents the rate of used person hours, while the y-axis indicates the fault density. The legends, which are the same in Figs. 8.2 – 8.5, denote the nine project datasets, which are labeled P1 to P9.

Table 8.1: Details of projects.

Project	LOC	Number of test case	Test density
P1	Small	Large	Large
P2	Small	Small	Large
P3	Large	Large	Small
P4	Small	Small	Large
P5	Large	Small	Small
P6	Large	Small	Small
P7	Small	Small	Small
P8	Large	Large	Small
P9	Small	Large	Large

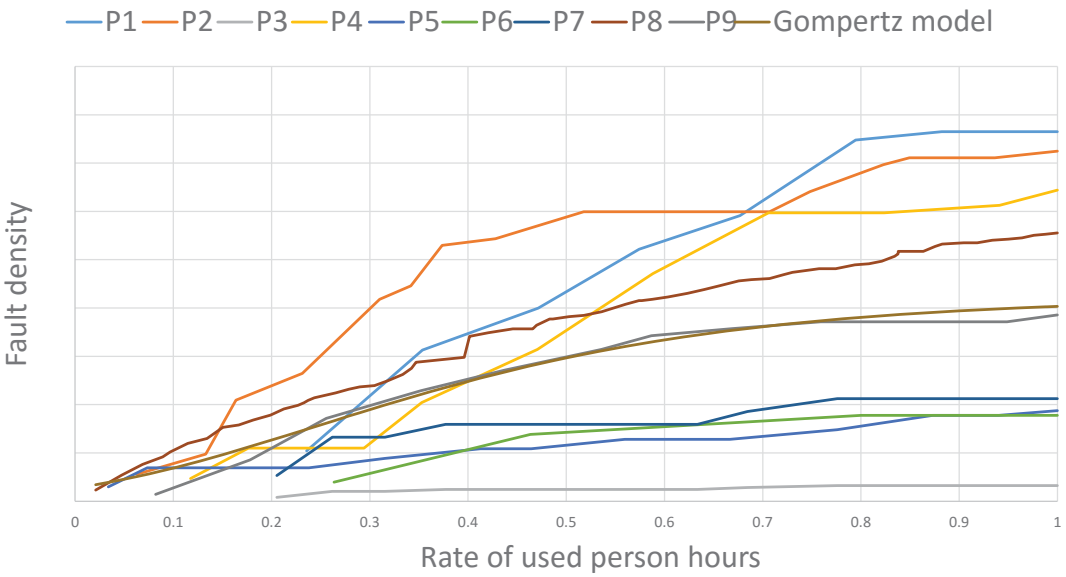


Figure 8.2: Results of the unclassified SRGM and the projects.

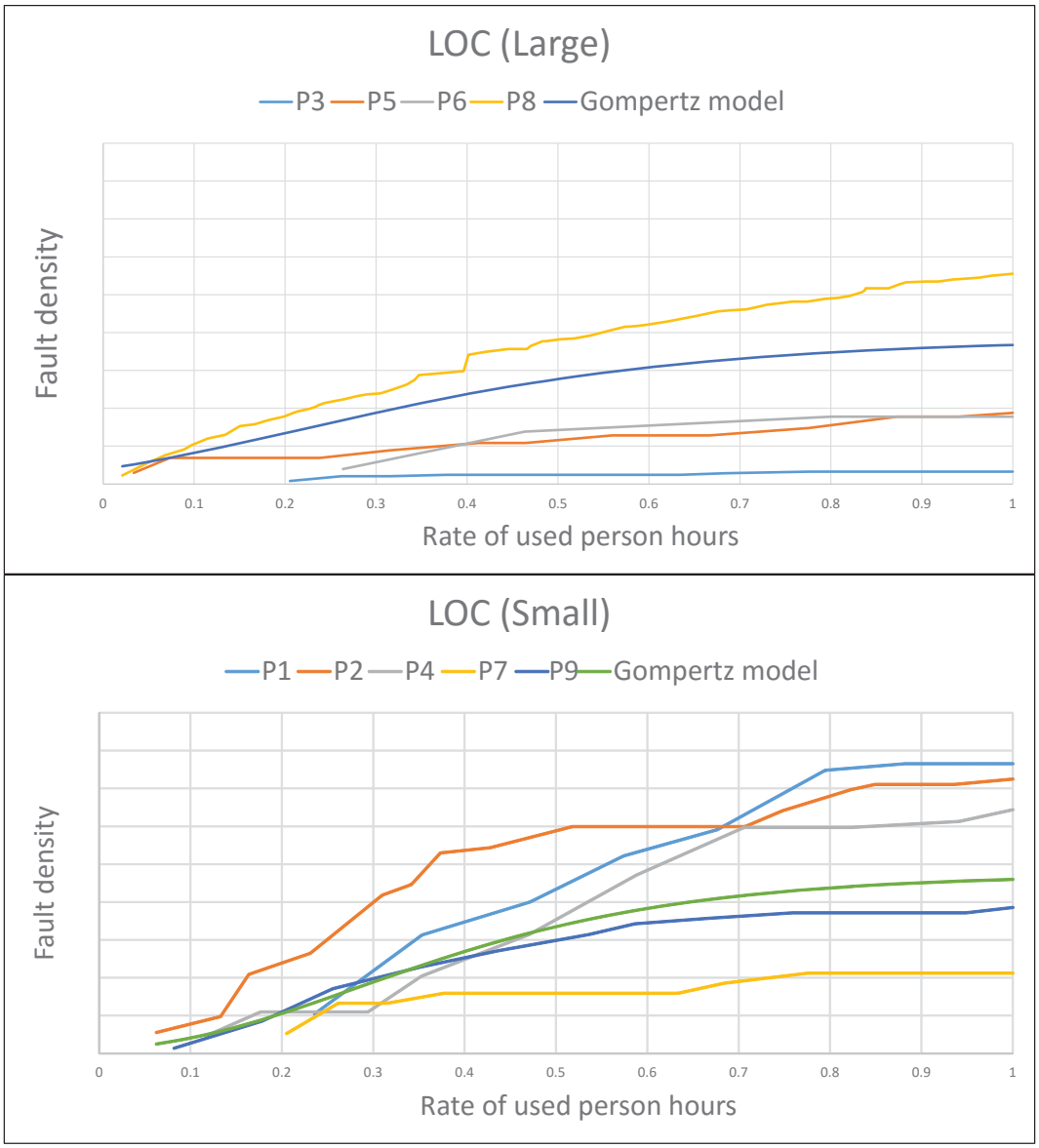


Figure 8.3: Results of the SRGM model classified by LOC and the projects.

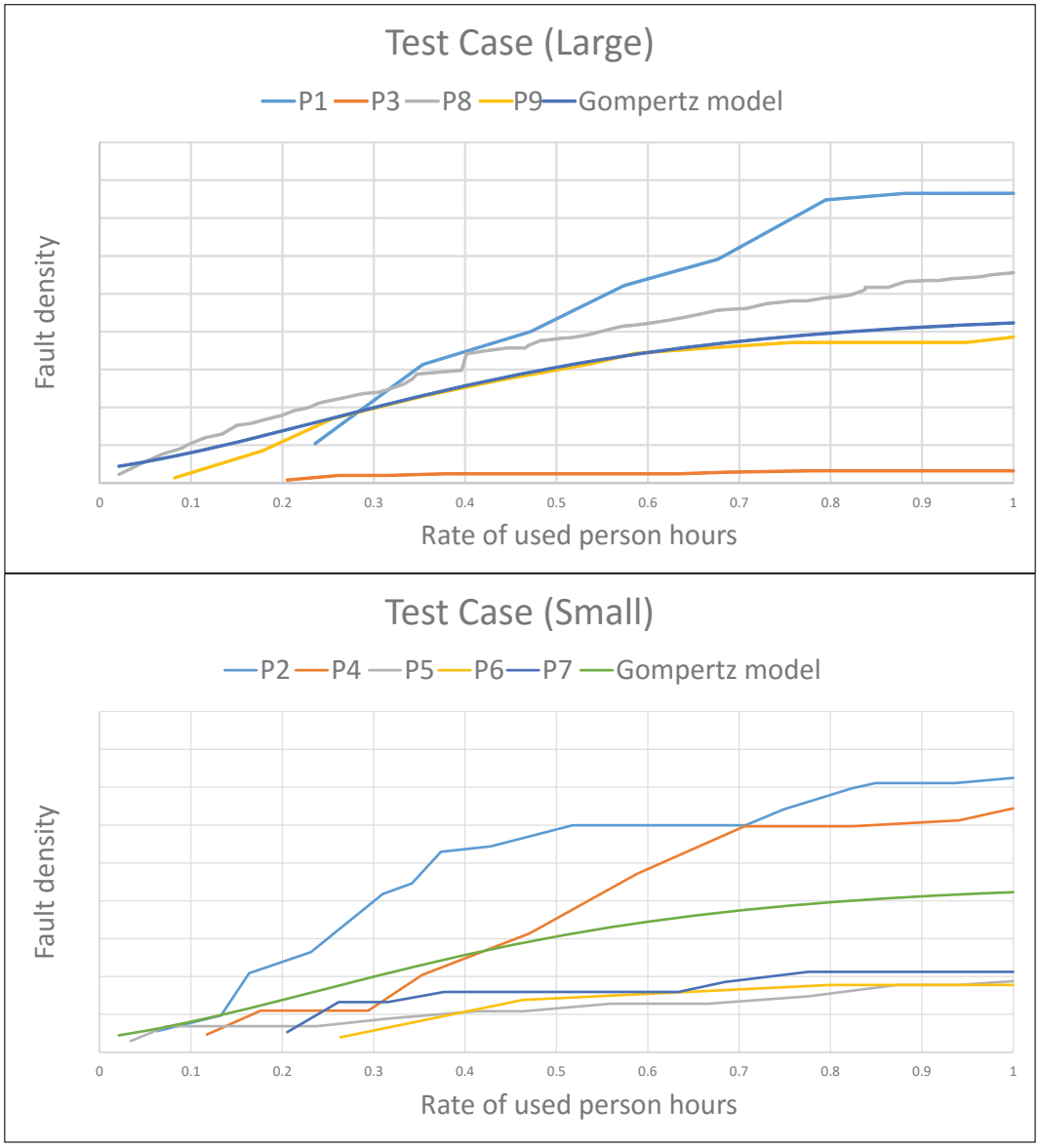


Figure 8.4: Results of the SRGM model classified by the test case and the projects.

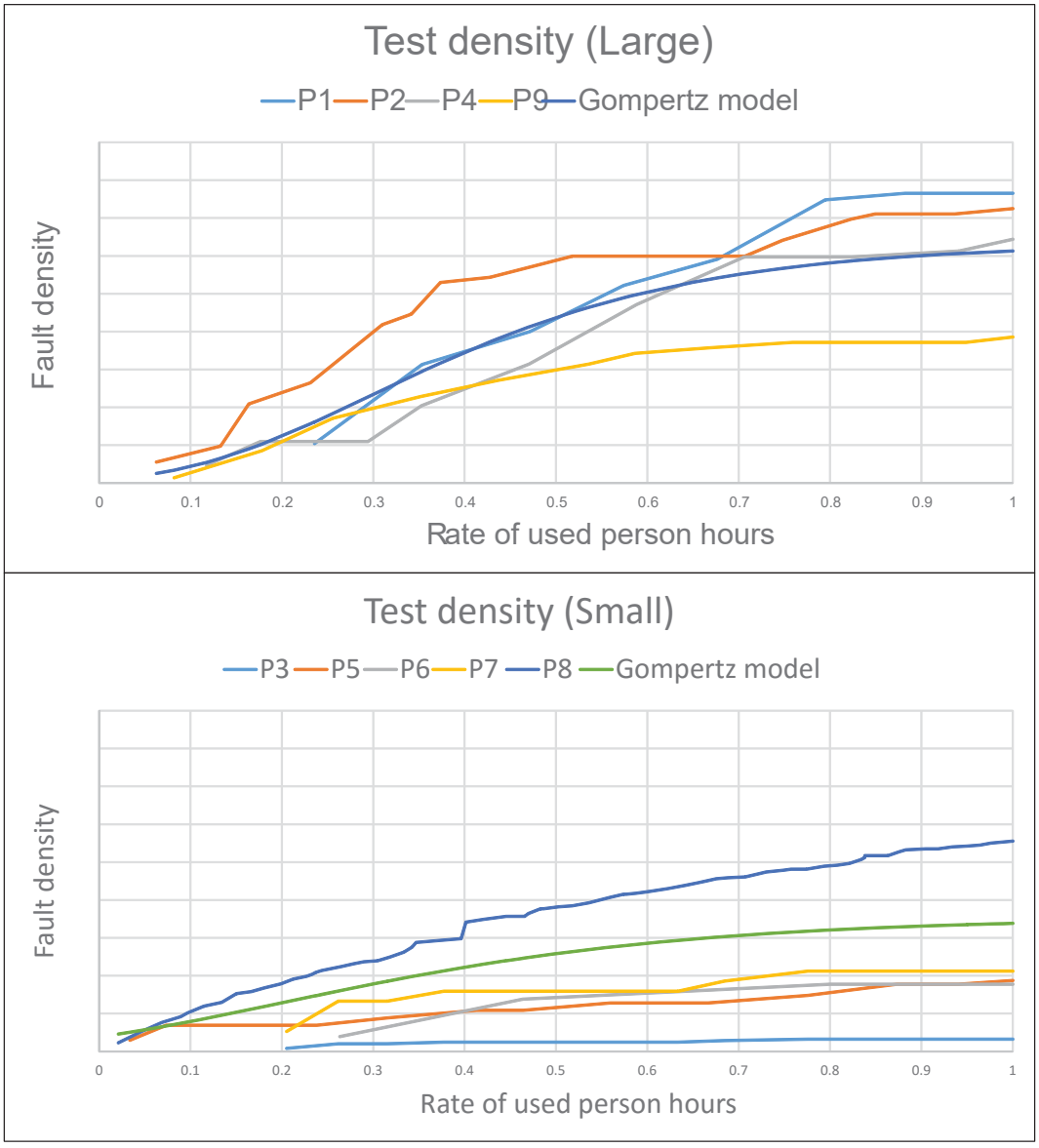


Figure 8.5: Results of the SRGM model classified by the test density and the projects.

8.4.2 Discussion

RQ1 (Do the results from the classified leveled SRGMs differ from those of the unclassified SRGMs?)

Table 8.2 shows the RSS of the classified and unclassified leveled SRGMs. Each value indicates the RSS of the model. The sum is the total of the values of the large group and the values of the small group. The results of the SRGM do differ based on the classification.

Table 8.2: Comparison of the RSS of the classified and unclassified leveled SRGMs.

Classification	Large	Small	Sum
None	-	-	161.80
Case	97.15	52.56	149.71
LOC	96.29	59.74	156.03
Density	19.15	104.7	123.85

RQ2 (If the results differ, which classification more precisely describes the results?)

Table 8.2 indicates that the most precise model in the large group is the classification by test density, but this is the worst model in the small group. However, for the total optimization, the classification by test density gives the most precise model. In the large and the small groups, the classification by LOC and test case yield almost the same value. In the total optimization, the unclassified SRGM provides the worst model.

Figure 8.6 shows the Gompertz model classified by the test density and P2, and P5. The leveled SRGMs more precisely describe the data than the unclassified leveled SRGM in Figure 8.1. Thus, the leveled SRGM classified by the test density has the smallest RSS in these models, implying that the classification by test density gives the most precise model.

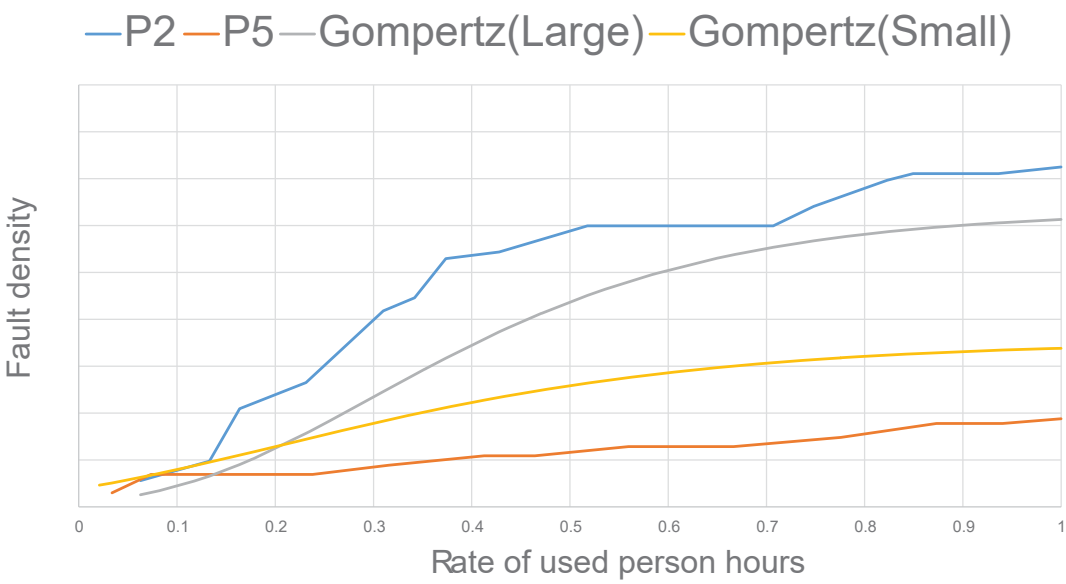


Figure 8.6: Fault densities and rates of used person hours for P2 and P5 and the leveled Gompertz models classified by test density.

8.5 Conclusion

I proposed a leveled SRGM which treated cross project datasets by classifying system scales of projects to compare software products developed by the same company in the same domain. I successfully modeled nine actual datasets by classifying with system scale parameters. The SRGM classified by test density can more precisely model the data than other classifications, including no classification.

In the future, I plan to use other dividing methods such as the k-means clustering since this work divided nine projects into two group by the median. Additionally, I plan to apply this method to GSRM. To apply this method to GSRM makes it possible to compare the uncertainties of projects.

Chapter 9

Conclusion

9.1 Summary of This Thesis

In this thesis, I present a Generalized Software Reliability Model (GSRM) and its empirical applications. Chapter 3 presents the GSRM, a new software reliability growth model. Nine types of developments are simulated and analyzed with the GSRM. Additionally, three types of uncertainties, which are related to actual development situations, are formulated.

Chapter 4 shows a technique to predict the release time based on the GSRM. Using the GSRM, I successfully predict the release dates and the number of issues about OSS.

Chapter 5 demonstrates a technique to predict the ranges of release development time, Δt, based on the GSRM. The uncertainty values are defined from actual data containing information on the faults during development. I apply the GSRM to three datasets to calculate Δt, including the range of possible development times considering the uncertainty values.

Chapter 6 describes a technique to detect unexpected situations in a development by separating faults by the test phase and applying the SRGM. I found unexpected situations in a development by monitoring the faults and the behavior of the SRGM.

Chapter 7 discusses a technique to compare projects by extending SRMGs with the fault density and the person hours. This technique is named a leveled SRGM. I employ a leveled SRGM to successfully compared nine actual datasets.

Finally, Chapter 8 demonstrates a technique to compare projects by ex-

tending the leveled SRGM by classifying datasets considering system scales.

9.2 Future Work

There are several additional issues in terms of the GSRM (Chapter 3), the detection of unexpected situations (Chapter 6), and the project management technique (Chapter 7 and 8). To address the issues with the GSRM, I plan to evaluate teams or team members using quantitative methods while considering uncertainties to optimize teams for a particular project using the GSRM. To resolve the unexpected situations and to provide insight to developers and managers who have trouble with development, I plan to evaluate my method by applying it to ongoing projects and other datasets belonging to other domains or organizations. Additionally, I plan to use other dividing methods (e.g., such as k-means clustering) since this work divided the nine projects into two group by the median.

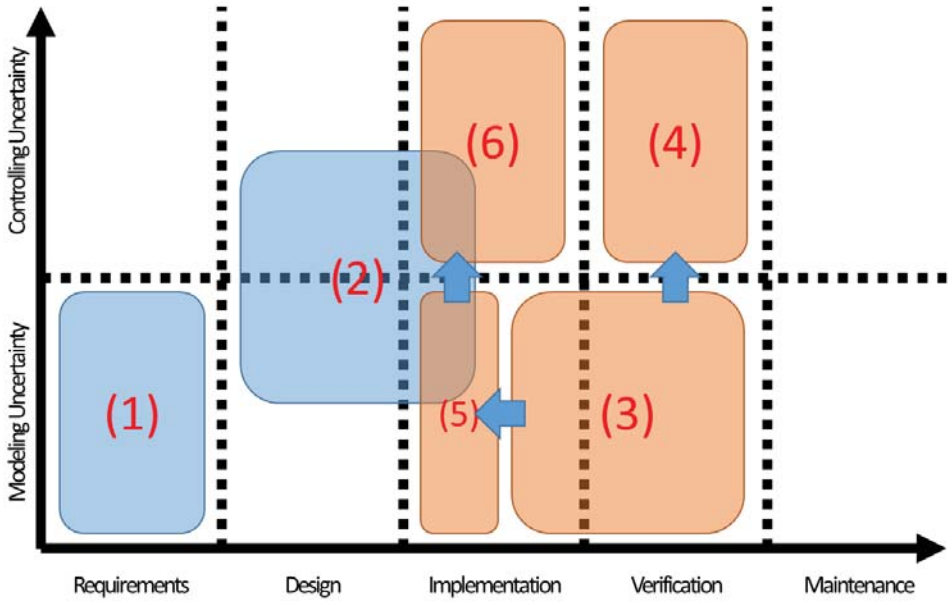


Figure 9.1: Overview of future work.

Figure 9.1 overviews my future work with regard to model uncertainty and the software development process. For area (4), I intend to analyze the

uncertainties about the verification process and specify the elements of uncertainties. In my opinion, the detection of faults is related to the dispersion of developers' skills. I plan to propose a method to evaluate developers' skills and analyze the relation between their dispersions and uncertainties.

On the other hand, for areas (5) and (6), my intent is to extend the GSRM to include the implementation phase. I plan to model elements of uncertainties such as violations of the coding convention and analyze the relation between their dispersions and uncertainties. In my opinion, numerous coding convention violations is an indicator that the developers misunderstood what other developers coded. Such a misunderstanding creates uncertain situations in developments.

Finally, I plan to propose a comprehensive model, which includes uncertain elements in implementation and verification that helps developers and managers to control their developments.

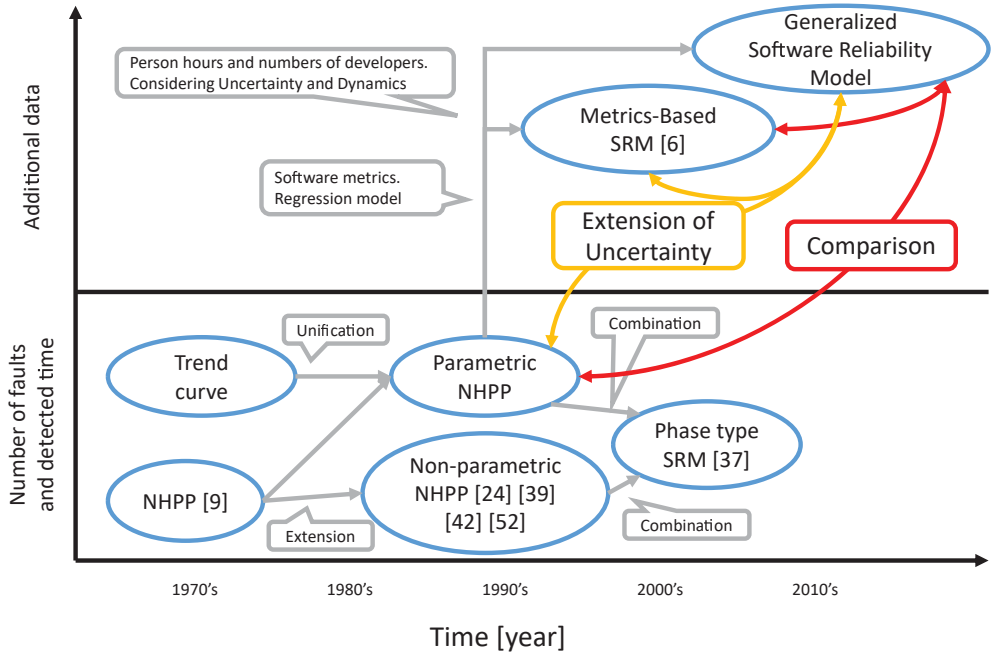


Figure 9.2: Overview of the future work in software reliability models.

Figure 9.2 overviews my future work on software reliability models. I will compare the GSRM with metrics-based software reliability models and phase-type software reliability models. To compare the GSRM with other software

reliability models, I plan to collect datasets containing faults data, metrics data, and the numbers of developers. In this thesis, I could not compare the GSRM with the metrics-based SRM because the datasets lacked metrics data and the number of developers. To compare these two, I need to devise a system that can collect the time series of metrics data or analyze open source repositories with the source codes.

About the extensions of the uncertainties, I will try to extend other models to treat the uncertainties. My GSRM approach can be applied to other SRGMs based on the NHPP model because I just extended an existing NHPP model by adding a stochastic variable to its parameters. Of course, I intend to survey the possibility of similar extensions because it is unclear whether such extensions are suitable for other NHPP models.

Finally, I would like to mention the application targets of software reliability models. Software reliability models are applied in a lot of domains such as military systems [9], medical record systems [45], automotive systems [40], etc. However, to the best of my knowledge, the relations between software reliability models and application domains have not been studied. To analyze the relations between them, I participated in the research team about software quality evaluation based on ISO/IEC 25022 and ISO/IEC 25023. The research team aims to evaluate a lot of software products and analyze the relations between product quality and quality in use [32]. I plan to analyze the relations between the software reliability models and the application domains.

Acknowledgments

To finish the work presented in this thesis, I owe much to a lot of persons. First, I would like to thank Prof. Hironori Washizaki for his supervision and support on my research. This thesis was accomplished with the help of Prof. Yoshiaki Fukazawa, Prof. Kazunori Ueda, Assoc. Prof. Hiroyuki Okamura as sub-examiners, and other professors of the Department of Computer Science and Engineering, Waseda University.

I also show sincere thanks to Assist. Prof. Kazunori Sakamoto (National Institute of Informatics), Prof. Kenichi Matsumoto (Graduate School of Information Science, Nara Institute of Science and Technology), Assist. Prof. Akinori Ihara (Graduate School of Information Science, Nara Institute of Science and Technology), Mr. Ken Asoh (Yahoo Japan Corporation), Mr. Kazuyoshi Takahashi (Yahoo Japan Corporation), Mr. Kentarou Ogawa (Yahoo Japan Corporation), Mr. Maki Mori (Yahoo Japan Corporation), Mr. Takashi Hino (Yahoo Japan Corporation), Mr. Yosuke Hayakawa (Yahoo Japan Corporation), Mr. Yasuyuki Tanaka (Yahoo Japan Corporation), Mr. Shinichi Yamada (Yahoo Japan Corporation), Mr. Daisuke Miyazaki (Yahoo Japan Corporation), Mr. Teppei Yamaguchi (Yahoo Japan Corporation), Mr. Tomoaki Yagi (Yahoo Japan Corporation), Mr. Hiroyuki Shibata (Yahoo Japan Corporation), Ms. Mikako Ishigaki (Yahoo Japan Corporation), Mr. Kazuki Munakata (Fujitsu Labs Ltd.), Ms. Sumie Morita (Fujitsu Labs Ltd.), Mr. Tadahiro Uehara (Fujitsu Labs Ltd.), Ms. Rieko Yamamoto (Fujitsu Labs Ltd.), and Mr. Nobuhiro Nakamura (Sumitomo Electric Industries, Ltd.). Plenty of discussions and advice have simulated and deepened my research analysis.

A part of this thesis was accomplished with the help of the members in Washizaki Laboratory, Waseda University. I would like to thank Mr. Hidenori Nakai, Ms. Chihiro Uchida, Mr. Toru Yagishita, Mr. Chi Jieming, Mr. Watanabe Yasuhiro, Mr. Masaki Hosono, and Mr. Yuki Noyori. I

am also thankful to all staffs and members in Washizaki Laboratory and Fukazawa Laboratory. Also, many (anonymous) referees contributed to the improvement of this thesis via conferences and paper submissions. Finally, I would like to thank my family for their support.

Bibliography

[1] Nesar Ahmad, Mohammed GM Khan, and Loriza S Rafi. Analysis of an inflection s-shaped software reliability model considering log-logistic testing-effort and imperfect debugging. *International Journal of Computer Science and Network Security*, 11(1):161–171, 2011.

[2] Mohd Anjum, Md Asraful Haque, and Nesar Ahmad. Analysis and ranking of software reliability models based on weighted criteria value. *International Journal of Information Technology and Computer Science (IJITCS)*, 5(2):1, 2013.

[3] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley & Sons, Inc, 1994.

[4] X. Cai and M. R. Lyu. Software reliability modeling with test coverage: Experimentation and measurement with a fault-tolerant software project. In *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*, pages 17–26, Nov 2007.

[5] Tadashi Dohi and Toshio Nakagawa. *Stochastic Reliability and Maintenance Modeling: Essays in Honor of Professor Shunji Osaki on His 70th Birthday*. Springer Publishing Company, Incorporated, 2013.

[6] Toshiya Fujii, Tadashi Dohi, and Takaji Fujiwara. Towards quantitative software reliability assessment in incremental development processes. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 41–50, New York, NY, USA, 2011. ACM.

[7] Takuya Fukamachi, Naoyasu Ubayashi, Shintaro Hosoai, and Yasutaka Kamei. Modularity for uncertainty. In *Proceedings of the Seventh In-*

ternational Workshop on Modeling in Software Engineering, MiSE '15, pages 7–12, Piscataway, NJ, USA, 2015. IEEE Press.

[8] A.L. Goel. Software reliability models: Assumptions, limitations, and applicability. *Software Engineering, IEEE Transactions on*, SE-11(12):1411–1423, Dec 1985.

[9] Amrit L Goel and Kazu Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE transactions on Reliability*, 3:206–211, 1979.

[10] K. Goseva-Popstojanova and S. Kamavaram. Assessing uncertainty in reliability of component-based software systems. In *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*, pages 307–320, Nov 2003.

[11] Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 392–401, Piscataway, NJ, USA, 2013. IEEE Press.

[12] K. Honda, N. Nakamura, H. Washizaki, and Y. Fukazawa. Case study: Project management using cross project software reliability growth model. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 39–46, Aug 2016.

[13] K. Honda, N. Nakamura, H. Washizaki, and Y. Fukazawa. Case study: Project management using cross project software reliability growth model considering system scale (to appear in). In *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*, Oct 2016.

[14] K. Honda, H. Washizaki, Y. Fukazawa, K. Munakata, S. Morita, T. Uehara, and R. Yamamoto. Detection of unexpected situations by applying software reliability growth models to test phases. In *Software Reliability Engineering Workshops (ISSREW), 2015 IEEE International Symposium on*, pages 2–5, Nov 2015.

[15] Kiyoshi Honda, Hidenori Nakai, Hironori Washizaki, Yoshiaki Fukazawa, Ken Asoh, Kaz Takahashi, Kentarou Ogawa, Maki Mori, Takashi Hino,

Yosuke HAYAKAWA, et al. Predicting time range of development based on generalized software reliability model. In *21st Asia-Pacific Software Engineering Conference (APSEC 2014)*, 2014.

[16] Kiyoshi Honda, Hironori Washizaki, and Yoshiaki Fukazawa. *A Generalized Software Reliability Model Considering Uncertainty and Dynamics in Development*, pages 342–346. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[17] Kiyoshi Honda, Hironori Washizaki, and Yoshiaki Fukazawa. A generalized software reliability model considering uncertainty and dynamics in development. In Jens Heidrich, Markku Oivo, Andreas Jedlitschka, and MariaTeresa Baldassarre, editors, *Product-Focused Software Process Improvement*, volume 7983 of *Lecture Notes in Computer Science*, pages 342–346. Springer Berlin Heidelberg, 2013.

[18] Kiyoshi Honda, Hironori Washizaki, and Yoshiaki Fukazawa. Predicting release time based on generalized software reliability model (gsrm). In *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pages 604–605. IEEE, 2014.

[19] Rong-Huei Hou, Sy-Yen Kuo, and Yi-Ping Chang. Applying various learning curves to hyper-geometric distribution software reliability growth model. In *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, pages 8–17, Nov 1994.

[20] Chin-Yu Huang and Michael R Lyu. Estimation and analysis of some generalized multiple change-point software reliability models. *Reliability, IEEE Transactions on*, 60(2):498–514, 2011.

[21] S. Inoue and S. Yamada. Bootstrap interval estimation methods for cost-optimal software release planning. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 621–626, Oct 2013.

[22] Sunil Kamavaram and Katerina Goseva-Popstojanova. Entropy as a measure of uncertainty in software reliability. In *13th Int?l Symp. Software Reliability Engineering*, pages 209–210, 2002.

[23] Yasutaka Kamei, Akito Monden, and Ken-ichi Matsumoto. Empirical evaluation of svm-based software reliability model. In *Proc. Fifth*

ACM/IEEE Int'l Symp. Empirical Software Eng, volume 2, pages 39–41, 2006.

[24] N. Karunanithi, D. Whitley, and Y. K. Malaiya. Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, 18(7):563–574, Jul 1992.

[25] Brian W Kernighan and Phillip James Plauger. The elements of programming style. *The elements of programming style, by Kernighan, Brian W.; Plauger, PJ New York: McGraw-Hill, c1978.*, 1, 1978.

[26] Sy-Yen Kuo, Chin-Yu Huang, and M. R. Lyu. Framework for modeling software reliability, using various testing-efforts and fault-detection rates. *IEEE Transactions on Reliability*, 50(3):310–320, Sep 2001.

[27] Sy-Yen Kuo, Chin-Yu Huang, and Michael R Lyu. Framework for modeling software reliability, using various testing-efforts and fault-detection rates. *Reliability, IEEE Transactions on*, 50(3):310–320, 2001.

[28] Richard Lai and Mohit Garg. A detailed study of nhpp software reliability models. *Journal of Software*, 7(6):1296–1306, 2012.

[29] T. Moser, R. Mordinyi, D. Winkler, and S. Biffl. Engineering project management using the engineering cockpit: A collaboration platform for project managers and engineers. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 579–584, July 2011.

[30] J. D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. In *Proceedings of the 7th International Conference on Software Engineering*, ICSE '84, pages 230–238, Piscataway, NJ, USA, 1984. IEEE Press.

[31] H. Nakai, K. Honda, H. Washizaki, Y. Fukazawa, K. Asoh, K. Takahashi, K. Ogawa, M. Mori, T. Hino, Y. Hayakawa, Y. Tanaka, S. Yamada, and D. Miyazaki. Initial industrial experience of gqm-based product-focused project monitoring with trend patterns. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, volume 2, pages 43–46, Dec 2014.

[32] H. Nakai, N. Tsuda, K. Honda, H. Washizaki, and Y. Fukazawa. Initial framework for software quality evaluation based on iso/iec 25022

and iso/iec 25023. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 410–411, Aug 2016.

[33] E.A. Nguyen, C.F. Rexach, D.P. Thorpe, and A.E. Walther. The importance of data quality in software reliability modeling. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 220–228, Nov 2010.

[34] Mitsuru Ohba. *Inflection S-Shaped Software Reliability Growth Model*, pages 144–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 1984.

[35] Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, and Koji Torii. Empirical project monitor: A tool for mining multiple project data. In *International Workshop on Mining Software Repositories (MSR2004)*, pages 42–46. IET, 2004.

[36] Koji Ohishi, Hiroyuki Okamura, and Tadashi Dohi. Gompertz software reliability model: Estimation algorithm and empirical validation. *Journal of Systems and Software*, 82(3):535 – 543, 2009.

[37] H. Okamura and T. Dohi. Building phase-type software reliability models. In *2006 17th International Symposium on Software Reliability Engineering*, pages 289–298, Nov 2006.

[38] H. Okamura, Y. Etani, and T. Dohi. A multi-factor software reliability model based on logistic regression. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 31–40, Nov 2010.

[39] Ping-Feng Pai and Wei-Chiang Hong. Software reliability forecasting by support vector machines with simulated annealing algorithms. *Journal of Systems and Software*, 79(6):747 – 755, 2006.

[40] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Torner. Evaluating long-term predictive power of standard reliability growth models on automotive systems. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 228–237, Nov 2013.

[41] Rakesh Rana, Miroslaw Staron, Christian Berger, Jrgen Hansson, Martin Nilsson, Fredrik Trner, Wilhelm Meding, and Christoffer Hglund. Selecting software reliability growth models and improving their predictive accuracy using historical projects data. *Journal of Systems and Software*, 98:59 – 78, 2014.

[42] Y. s. Su, C. y. Huang, Y. s. Chen, and J. x. Chen. An artificial neural-network-based approach to software reliability assessment. In *TENCON 2005 - 2005 IEEE Region 10 Conference*, pages 1–6, Nov 2005.

[43] N. Schneidewind and M. Hinchey. A complexity reliability model. In *Software Reliability Engineering, 2009. ISSRE '09. 20th International Symposium on*, pages 1–10, Nov 2009.

[44] Yogesh Singh and Pradeep Kumar. Prediction of software reliability using feed forward neural networks. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–5. IEEE, 2010.

[45] C. Stringfellow and A.Amschler Andrews. An empirical method for selecting software reliability growth models. *Empirical Software Engineering*, 7(4):319–343, 2002.

[46] Catherine V Stringfellow. An integrated method for improving testing effectiveness and efficiency. 2007.

[47] Yoshinobu Tamura and Shigeru Yamada. A flexible stochastic differential equation model in distributed development environment. *European Journal of Operational Research*, 168(1):143–152, 2006.

[48] Naoyasu Ubayashi, Di Ai, Peiyuan Li, Yu Ning Li, Shintaro Hosoai, and Yasutaka Kamei. Uncertainty-aware architectural interface. In *Proceedings of the 9th International Workshop on Advanced Modularization Techniques*, AOAsia 2014, pages 4–6, New York, NY, USA, 2014. ACM.

[49] Linda Wallace, Mark Keil, and Arun Rai. Understanding software project risk: a cluster analysis. *Information & Management*, 42(1):115–125, 2004.

[50] H. Washizaki, K. Honda, and Y. Fukazawa. Predicting release time for open source software based on the generalized software reliability model. In *Agile Conference (AGILE), 2015*, pages 76–81, Aug 2015.

[51] M. Xie and G.Y. Hong. A study of the sensitivity of software release time. *Journal of Systems and Software*, 44(2):163 – 168, 1998.

[52] Fei Xing and Ping Guo. *Support Vector Regression for Software Reliability Growth Modeling and Prediction*, pages 925–930. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[53] S. Yamada, J. Hishitani, and S. Osaki. Software-reliability growth with a weibull test-effort: a model and application. *IEEE Transactions on Reliability*, 42(1):100–106, Mar 1993.

[54] Shigeru Yamada. Recent developments in software reliability modeling and its applications. In *Stochastic Reliability and Maintenance Modeling*, pages 251–284. Springer, 2013.

[55] Shigeru Yamada, Mitsuhiro Kimura, Hiroaki Tanaka, and Shunji Osaki. Software reliability measurement and assessment with stochastic differential equations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 77(1):109–116, 1994.

[56] Shigeru Yamada, Mitsuru Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. *Reliability, IEEE Transactions on*, R-32(5):475–484, Dec 1983.

[57] Shigeru Yamada, Mitsuru Ohba, and S. Osaki. s-shaped software reliability growth models and their applications. *Reliability, IEEE Transactions on*, R-33(4):289–292, 1984.

[58] Shigeru Yamada, Hiroshi Ohtera, and Mitsuru Ohba. Testing-domain dependent software reliability models. *Computers and Mathematics with Applications*, 24(12):79 – 86, 1992.

[59] Nan Zhang, Gang Cui, and Hongwei Liu. A stochastic software reliability growth model with learning and change-point. In *World Automation Congress (WAC), 2012*, pages 399–403, June 2012.

[60] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 91–100, New York, NY, USA, 2009. ACM.

List of Publications

Journal Papers

- ◎ **Generalized Software Reliability Model Considering Uncertainty and Dynamics: Model and Applications**
  Kiyoshi Honda (first author), Hironori Washizaki and Yoshiaki Fukazawa
  International Journal of Software Engineering and Knowledge Engineering (IJSEKE), World Scientific, pp.1–28, Sep. 2017 (to appear), (reviewed).
  (related to Chapter 3)
- まねっこダンス：真似て覚えるプログラミング学習ツール
  坂本 一憲，本田 澄，音森 一輝，山崎 頌平，服部 真智子，松浦 由真，高野 孝一，鷲崎 弘宜，深澤 良彰
  コンピュータソフトウェア，32(4), 日本ソフトウェア科学会, pp.103 – 114, Nov. 2015, (reviewed).

International Conference Presentations

- **Identifying Potential Problems and Risks in GQM+Strategies Models Using Metamodel and Design Principles**
  Chimaki Shimura, Hironori Washizaki, Takanobu Kobori, Yohei Aoki, Kiyoshi Honda, Yoshiaki Fukazawa, Katsutoshi Shintani and Takuto Nonomura
  50th Annual Hawaii International Conference on System Sciences (HICSS 50), IEEE, pp.1–10, Jan. 2017, (reviewed).
- **Evaluating Software Product Quality based on SQuaRE Series**

Hidenori Nakai, Naohiko Tsuda, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa
2016 IEEE Region 10 Conference (TENCON), pp.3708–3711, Nov. 2016, (reviewed).

◎ **Case Study: Project Management Using Cross Project Software Reliability Growth Model Considering System Scale**
Kiyoshi Honda (first author), Nobuhiro Nakamura, Hironori Washizaki and Yoshiaki Fukazawa
2016 IEEE International Symposium on Software Reliability Engineering Workshops(ISSREW), IEEE, pp.1–4, Oct. 2016, (reviewed).
(related to Chapter 8)

- **Initial Framework for a Software Quality Evaluation based on ISO/IEC 25022 and ISO/IEC 25023**
Hidenori Nakai, Naohiko Tsuda, Kiyoshi Honda, Hironori Washizaki, and Yoshiaki Fukazawa
The 2016 IEEE International Conference on Software Quality, Reliability & Security (QRS 2016), IEEE, pp. 410–411, Aug. 2016, (reviewed).

◎ **Case Study: Project Management Using Cross Project Software Reliability Growth Model**
Kiyoshi Honda (first author), Nobuhiro Nakamuray, Hironori Washizaki and Yoshiaki Fukazawa
The 2016 IEEE International Conference on Software Quality, Reliability & Security Companion, IEEE, pp.39-46, Aug. 2016, (reviewed).
(related to Chapter 7)

- **GO-MUC: A Strategy Design Method Considering Requirements of User and Business by Goal-Oriented Measurement**
Chihiro Uchida, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, Kentaro Ogawa, Tomoaki Yagi, Mikako Ishigaki, Masashi Nakagawa
9th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2016), IEEE, pp. 93-96, May 2016, (reviewed).

◎ **Software Reliability Growth Model Considering Uncertainty and Dynamics in Development**
Kiyoshi Honda (first author), Hironori Washizaki, Yoshiaki Fukazawa
23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016), IEEE, pp.1–1, Mar. 2016, (reviewed).

- **Toward selecting a reliable version of OSS library based on bug-fixing curve**
Keisuke Fujino, Akinori Ihara, Kiyoshi Honda, Hironori Washizaki, Kenichi Matsumoto
23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016), Poster, Mar. 2016, (reviewed).

◎ **Case Study: Software Reliability Growth Model Based on Person Hours**
Kiyoshi Honda (first author), Nobuhiro Nakamura, Hironori Washizaki and Yoshiaki Fukazawa
7th IEEE International Workshop on Empirical Software Engineering in Practice (IWESEP), Poster, Mar. 2016, (reviewed).

◎ **Detection of Unexpected Situations by Applying Software Reliability Growth Models to Test Phases**
Kiyoshi Honda (first author), Hironori Washizaki, Yoshiaki Fukazawa, Kazuki Munakatay, Sumie Moritay, Tadahiro Ueharay, and Rieko Yamamoto
2016 IEEE International Symposium on Software Reliability Engineering Workshops(ISSREW), IEEE, pp.2-5, Nov. 2015, (reviewed).
(related to Chapter 6)

- **Predicting Release Time for Open Source Software based on the Generalized Software Reliability Model**
Hironori Washizaki, Kiyoshi Honda, Yoshiaki Fukazawa
Proceedings of Agile Conference 2015, IEEE, pp.76–81, Aug. 2015 (reviewed).

- **Comparative Study on Programmable Robots as Programming Educational Tools**
Shohei Yamazaki, Kazunori Sakamoto, Kiyoshi Honda, Hironori

Washizaki, Yoshiaki Fukazawa
Proceedings of the 17th Australasian Computing Education Conference (ACE 2015), Australian Computer Society Inc., pp.155–164, Jan. 2015, (reviewed).

◎ **Predicting Time Range of Development Based on Generalized Software Reliability Model**
Kiyoshi Honda (first author), Hidenori Nakai, Hironori Washizaki, Yoshiaki Fukazawa (Waseda University), Ken Asoh, Kaz Takahashi, Kentarou Ogawa, Maki Mori, Takashi Hino, Yosuke Hayakawa, Yasuyuki Tanaka, Shinichi Yamada, Daisuke Miyazaki
21st Asia-Pacific Software Engineering Conference (APSEC 2014), IEEE, pp.351–358, Dec. 2014, (reviewed).
(related to Chapter 5)

- **Initial Industrial Experience of GQM-based Product-Focused Project Monitoring with Trend Patterns**
Hidenori Nakai, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, Ken Asoh, Kaz Takahashi, Kentarou Ogawa, Maki Mori, Takashi Hino, Yosuke Hayakawa, Yasuyuki Tanaka, Shinichi Yamada, Daisuke Miyazaki
21st Asia-Pacific Software Engineering Conference (APSEC 2014), pp.43–46, Dec. 2014, (reviewed).

- **Continuous Product-Focused Project Monitoring with Trend Patterns and GQM**
Hidenori Nakai, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, Ken Asoh, Kaz Takahashi, Kentarou Ogawa, Maki Mori, Takashi Hino, Yosuke Hayakawa, Yasuyuki Tanaka, Shinichi Yamada, Daisuke Miyazaki
Proceedings of the 2nd International Workshop on Quantitative Approaches to Software Quality (QuASoC 2014), IEEE, pp. 69–74, Dec. 2014, (reviewed).

- **Toward Monitoring Bugs-fixing Process after the Releases in Open Source Software**
Keisuke Fujino, Akinori Ihara, Kiyoshi Honda, Hironori Washizaki and Kenichi Matsumoto
6th International Workshop on Empirical Software Engineering in Practice (IWESEP 2014), Poster, Nov. 2014, (reviewed).

◎ **Predicting the Release Time Based on a Generalized Software Reliability Model (GSRM)**
Kiyoshi Honda (first author), Hironori Washizaki, Yoshiaki Fukazawa
Proceedings of the 38th Annual IEEE International Computers, Software, and Applications Conference (COMPSAC), IEEE, pp.604–605, Jul. 2014, (reviewed).
(related to Chapter 4)

◎ **A Generalized Software Reliability Model Considering Uncertainty and Dynamics in Development**
Kiyoshi Honda (first author), Hironori Washizaki, YoshiakiFukazawa
Proceedings of 14th International Conference of Product Focused Software Development and Process Improvement (PROFES 2013), Springer, pp.342–346, Jun. 2013, (reviewed).

Domestic Conference Presentations

- **GO-MUC (Goal-Oriented Measurement for Usability and Conflict): ゴール指向によるユーザ・ビジネス要求を満たす戦略立案支援**
内田 ちひろ，本田 澄，渡邊　泰宏，鷲崎 弘宜，深澤良彰，小川 健太郎，八木 智章，石垣 光香子，中川　雅史
HCD-Net フォーラム 2016, Poster, Jun. 2016, (reviewed).
- 欠陥とソースコードの変更回数の関係分析
本田 澄 (first author), 坂口 英司, 伊原 彰紀, 鷲崎 弘宜, 深澤 良彰
ソフトウェア工学研究会ウィンターワークショップ 2016・イン・逗子 論文集, 情報処理学会, pp.57 – 58, Jan. 2016, (reviewed).
- オープンソースソフトウェアに関するソースコードの変更回数とバグ修正の関係分析に向けて
本田 澄 (first author)，伊原 彰紀，鷲崎 弘宜，深澤 良彰
日本ソフトウェア科学会 第 22 回 ソフトウェア工学の基礎ワークショップ (FOSE 2015), Poster, Nov. 2015, (reviewed).
- 開発者行動を考慮したソフトウェア信頼性モデル
本田 澄 (first author), 鷲崎 弘宜, 深澤 良彰
ソフトウェア工学研究会ウィンターワークショップ 2015・イン・宜野湾, 情報処理学会, pp.37 – 38, Jan. 2015, (reviewed).

- **OSS の不具合修正曲線に基づく残存未修正不具合数の予測の試み**
  藤野 啓輔，伊原 彰紀，本田 澄，鷲崎 弘宜，松本 健一
  日本ソフトウェア科学会 第 21 回 ソフトウェア工学の基礎ワークショップ (FOSE 2014), 近代科学社, pp.57–62, Dec. 2014, (reviewed).
- **開発者数の変動を含むソフトウェア信頼性モデルを用いた欠陥数予測**
  本田 澄 (first author), 中井 秀矩, 鷲崎 弘宜, 深澤 良彰
  日本ソフトウェア科学会 第 21 回 ソフトウェア工学の基礎ワークショップ (FOSE 2014), Poster, Dec. 2014, (reviewed).

◎ **不確実性を含む信頼性成長モデル**
本田 澄 (first author) , 鷲崎 弘宜 , 深澤 良彰
ソフトウェア工学研究会ウィンターワークショップ 2014・イン・大洗 論文集, 情報処理学会, pp.15–16, Jan. 2014, (reviewed).

- **GQM を用いた改善プロセスサポートツールの開発**
  中井 秀矩 , 本田 澄 , 鷲崎 弘宜 , 深澤 良彰
  ソフトウェア工学研究会ウィンターワークショップ 2014・イン・大洗 論文集, 情報処理学会, pp.105–106, Jan. 2014, (reviewed).
- **CI ツールとリポジトリシステムを用いた欠陥数予測**
  本田 澄 (first author)，中井 秀矩，鷲崎 弘宜，深澤 良彰，森牧，小川 健太郎，高橋 一貴
  SQiP シンポジウム 2014, Oral presentation, Sep. 2014, (reviewed).
- **ホワイトボックス単体テストにおけるペアテスティング**
  坂本 一憲，本田 澄，鷲崎 弘宜，深澤 良彰
  日本ソフトウェア科学会 第 20 回 ソフトウェア工学の基礎ワークショップ (FOSE 2013), 近代科学社, pp.227–232, Nov. 2013, (reviewed).

Annual Convention Presentation

- **ソフトウェアメトリクス測定値を用いた開発時のサブシステムにおける品質改善効率低迷状態検出**
  細野 将揮, 鷲崎 弘宜, 深澤 良彰, 本田 澄, 宗像 一樹, 森田 純恵, 上原 忠弘, 山本 里枝子
  日本ソフトウェア科学会第 33 回大会, 日本ソフトウェア科学会, pp.1–8, Sep. 2016, (reviewed).

- オープンソースソフトウェア開発における変更行数とリリースの関係分析に向けて
  本田 澄 (first author)，伊原 彰紀，鷲崎 弘宜，深澤 良彰
  日本ソフトウェア科学会第 33 回大会, Poster, Sep. 2016, (reviewed).
- ソースコードの変更回数と不具合修正の関係分析に向けて
  本田 澄 (first author)，伊原 彰紀，鷲崎 弘宜，深澤 良彰
  日本ソフトウェア科学会第 32 回大会, Poster, Sep. 2015, (reviewed).
- プログラミング教育：ロボットの導入による効果についての比較研究
  山崎 頌平，坂本 一憲，本田 澄，鷲崎 弘宜，深澤 良彰
  教育システム情報学会研究報告, 教育システム情報学会, 29(5), pp. 105–110, Jan. 2014, (reviewed).
- リポジトリシステムとソフトウェア信頼性モデルを用いた欠陥数予測
  本田 澄 (first author)，鷲崎 弘宜，深澤 良彰
  電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, 一般社団法人電子情報通信学会, 114(271), pp.13–16, Oct. 2014, (reviewed).
- まねっこダンス：真似て覚えるプログラミング学習ツール
  坂本 一憲，高野 孝一，本田 澄，音森 一輝，山崎 頌平，鷲崎 弘宜，深澤 良彰
  日本ソフトウェア科学会第 31 回大会, 日本ソフトウェア科学会, pp.1–8, Sep. 2014, (reviewed).

◎ 開発における不確定性と時間変化を考慮した一般化信頼性モデル
本田 澄 (first author)，鷲崎 弘宜，深澤 良彰
研究報告ソフトウェア工学（SE）, 2013-SE-180(8), 情報処理学会, pp.1–8, May 2013, (reviewed).

Other Publications and Presentations

- **Software Reliability Growth Model with Uncertainties and Dynamics in Development**
  Kiyoshi Honda (first author)

5th Asian Workshop of Advanced Software Engineering (AWASE2016), Oral presentation, Mar. 2016.

- メトリクス公団 **Vol.1 (不確定性と時間変化を含む一般化信頼性モデル, pp.8–15)**
  鷲崎 弘宜，本田 澄，深澤 良彰, Edited by TEF 東海メトリクス勉強会, Self-publishing, Sep. 2013.