A New Challenge for Applying Time Series Metrics Data to Software Quality Estimation

Sousuke AMASAKI (amasaki@ist.osaka-u.ac.jp), Takashi YOSHITOMI, Osamu MIZUNO (o-mizuno@ist.osaka-u.ac.jp), Yasunari TAKAGI and Tohru KIKUNO

Graduate School of Information Science and Technology, Osaka University, Japan

Abstract. In typical software development, a software reliability growth model (SRGM) is applied in each testing activity to determine the time to finish the testing. However, there are some cases in which the SRGM does not work correctly. That is, the SRGM sometimes mistakes quality for poor quality products. In order to tackle this problem, we apply time series data collected from development to quality estimation. First, we investigate the characteristics of the time series data on the detected faults by observing the change of the number of detected faults. Using the rank correlation coefficient, the data are classified into four kinds of trends. Next, with the intention of estimating software quality, we investigate the relationship between the trends of the time series data and software quality. Here, software quality is defined by the number of faults detected during six months after shipment. Finally, we find a relationship between the trends and metrics data collected in the software design phase. Using logistic regression, we statistically show that two review metrics in the design & coding phase can determine the trend.

Keywords: Software testing, Software quality, Time series data, Statistical analysis

1. Introduction

Software quality assurance is important for satisfying not only the users' requirements but also for reducing the cost of software maintenance. As the life cycle of software becomes longer and software is updated many times according to new requirements, software quality becomes more difficult to assure.

One of the ways to assure software quality is to reduce the number of faults remaining in software products by sufficient testing (Marick, 1995). Here, the software quality is considered as the number of remaining faults. For this purpose, estimating the number of remaining faults has become important. Therefore, various methods have been proposed. Generally speaking, these methods are classified into two types: estimation using the product metrics and estimation using the process metrics.

The methods using the product metrics estimate the number of remaining faults (that is, software quality) by using metrics such as the code size, the complexity, and so on, which are obtained or observed from software products. For instance, Halstead proposed an equation that calculates the number of faults from the complexity of software, which is derived from the source code(Halstead, 1977). Compton et al. proposed an equation that calculates the number of faults from the

LOC, which is the well-known size metrics (Compton and Withrow, 1990).

Although various methods have been proposed, methods based on product metrics usually estimate the software quality less accurately, whereas methods based on process metrics estimate more acculately, especially in the case of large size projects. The reason for this difference seems that the effect of human factors and the quality of organization, which can be observed by process metrics, are more significant than that of product characteristics, which can be measured by product metrics in large software projects. For example, in CMM (Paulk et al., 1993), the importance of quality of organization, which can be guessed from process metrics but not from product metrics, is especially emphasized.

Therefore, methods based on process metrics, which estimate the number of remaining faults, have also been proposed. Process metrics are recorded in the progress of the development process. Methods estimating the number of remaining faults (that is, software quality), such as capture-recapture model (Basin, 1973) and the software reliability growth model (SRGM) (Goel, 1985; Musa et al., 1987) are well-known. Both methods are designed for software testing activity.

In the capture-recapture model, pseudo defects are seeded in software products before testing. During testing, such pseudo defects and

real defects are found. By using the number of both defects, the number of real defects is estimated (Basin, 1973). However, an application for practical use is difficult because of the difficulty and the cost of the implantation of defects, especially for large products.

In the SRGM, the change in the number of faults detected in a testing activity is represented by a successive curve function that aims to fit the actual number of detected faults. The software release time, which is the time to finish testing, can be decided using the ratio of the time to be spent and the number of faults expected to be detected, both of which are known from the curve. Thus, developers in actual companies have extensively applied the SRGM to decide release time in order to manage testing efficiently.

In a certain company, the SRGM has been used to decide the release time of a testing activity. In most projects of the company, the final quality of software products has been successfully assured to be good by testing based on the SRGM. However, cases remain where the final quality of software products is still poor even after decision making by the SRGM. Therefore, the software engineering process group (SEPG) in the company believes that finding a reason for the problem becomes an important problem to be solved.

We focus on the fact that although the SRGM was applied to each of the four testing activities, which are a part of the software testing phase

(usually, consisting of four activities: unit testing, integration testing, function testing, and system testing), the relationship among the data collected from more than one testing activity in the testing phase has not been investigated. In order to check this possibility, we investigate the time series data on the number of faults collected from more than one test activity and focus on the trends of the time series data.

In previous research, an investigation using time series data is performed by Smidts et al. (Smidts et al., 1996). In order to develop a software reliability prediction model (SRPM), the authors introduced a requirement failure data histogram over the software development process. In the histogram, the x-axis represents a life-cycle effort and the y-axis represents a percentage of requirement failures (that is: change, addition, and deletion of requirements). Next, the authors found that the shape of the histogram is bimodal, and they developed the SRPM using this characteristic. However, empirical evaluation has not been performed.

On the other hand, we focus on the time series data of the number of detected faults. Based on the observation that projects have several specific kinds of shapes, we assume the shape itself plays an important role. That is, the shapes tend to show software quality indirectly. Thus, we investigate the relationship between a shape of the time series data

and software quality, and utilize this relationship in software quality prediction.

First, we present the characteristic of the time series data on the numbers of detected faults by investigating the changes in the values. In this study, we use actual data collected from development projects in a certain company. By applying the rank correlation coefficient, the data are classified into four types of trends: strictly increasing (T_{SI}), almost increasing (T_{AI}), almost decreasing (T_{AD}), and strictly decreasing (T_{SD}).

Secondly, we show the relationship between the trends of the time series data on the numbers of detected faults and software quality. In this study, we take the number of faults detected during six months after shipment as the software quality. As a result of the statistical analysis, we find that the quality of software products developed by projects with trends T_{AD} and T_{SD} can be relatively high.

Finally, with the intention of controlling the trend of detected faults in the early stage of a project (if we can do so before the testing phase, the quality of products will become better), we investigate a relationship between the trends of detected faults in the test phase and metrics data collected in the design and the coding phases. From this study, we find that two review metrics, the efficiency of the design review and the amount of effort in the code review, can determine

the trends. Furthermore, we find that a logistic regression model using these metrics can estimate the trends of detected faults successfully.

2. Target Projects

2.1. Characteristics of Projects

The projects targeted in this paper are the development of computer control systems with embedded software in a certain company. The software products developed by the projects have the following common characteristics. The systems are related to retail systems, and thus embedded software implements rather complex functions dealing with many sensors, actuators, and control signals including various kinds of interrupts. Furthermore, since the software products are delivered in the form of LSI chips, modification of faults after delivery is very expensive. Thus, high quality is especially required for the embedded software.

We use actual project data of 111 projects, which have already finished their development. Each project was carried out between 1995 and 1998. Additionally, we select these projects under certain conditions. First, the total number of faults detected during the test & debug phase

exceeds a certain number. This condition implies that the size of target project is relatively large. Second, the size of product exceeds a certain size. This condition implies that the size of product is also relatively large. In this research, we assume the threshold is about 4Kstep. Third, as in any targeted project, the code review must be performed.

2.2. Process Model

In the target projects, the products are developed under a development process as shown in Figure 1. The development process is an ordinal waterfall model. This process consists of two successive phases, namely the design & coding phase and the test & debug phase. The design & coding phase is divided into five activities: Concept design (CD), Function design (FD), Structure design (SD), Module design (MD), and Coding (CO). Each activitiy has a review activity, such as Concept design review (CDR), Function design review (FDR), Structure design review (SDR), Module design review (MDR), and Coding review (CR), to assure software quality. The test & debug phase consists of four activities: Unit test & debug (UT), Integration test & debug (IT), Function test & debug (FT), and System test & debug (ST).

One characteristic of the design & coding phase is that review activity is introduced after each design activity. Review activity enables

the detection and correction of faults in software artifacts as soon as these artifacts are created. Furthermore, the review activity not only improves the quality of the artifacts but also helps software development organizations reduce their cost of producing software (Bisant and Lyle, 1989; Fagan, 1986). In the review activity, the documents are distributed to the persons concerned in the company, and then review results are returned to developers via a manager (this review activity is called peer review (Bisant and Lyle, 1989)). The SEPG in the company establishs several guidelines for the review activity. One guideline directs at least 15% of the total effort of design & coding phase to be assigned to review activities (Takagi et al., 1995).

The test & debug phase consists of the repetition of a pair of test activity and debug activity. Testing activity is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software items. The SRGM has been used to decide the release time of each testing activity. Debug activity is the process used to detect, locate, and correct faults (International Standard Organization, 1990). The persons engaged in the test and debug phase are directed to record all faults that are detected by the test activity and removed by the debug activity (Tanaka et al., 1995). In order to improve and to assure the quality of the software product, the product needs to be sufficiently tested.

3. Time Series Data of Detected Faults

3.1. Software Metrics

(1) Frequency of detected faults: DF

The frequency of detected faults (DF) is the number of detected faults normalized by the effort needed for an activity for detection. This metric indirectly represents the ability of developers and the density of faults. In order to define DF, we introduce the following two parameters D_α and E_α where α denotes an activity out of five activities: CR, UT, IT, FT and ST in Figure 1.

D_α**:** the number of detected faults in an activity α.

E_α**:** the effort needed for an activity α (measured by person-day).

For example, the number of detected faults in the Unit test & debug activity (UT) is described as D_{UT}. Thus, DF for activity α is defined as follows:

$$DF_\alpha = \frac{D_\alpha}{E_\alpha}$$

According to this definition, we define the frequency of detected faults DF_{CR}, DF_{UT}, DF_{IT}, DF_{FT}, DF_{ST} for activities CR, UT, IT, FT, and ST, respectively.

(2) Field fault density: FFD

In the target company, the field quality of the final product is measured by the metrics named field fault density FFD. This density is defined as follows:

$$FFD = \frac{D_{field}}{S_{final}}$$

Here, S_{final} represents the product size and is represented by the unit $KLOC$.

In the company, D_{field} is defined as the number of faults detected during six months after delivery.

3.2. Collected Data

The effort data and faults data are recorded manually, and are stored in workstations by each developer. Next, the data are collected by the project leader, and validated by the manager. On the other hand, field faults data are reported by a quality assurance staff, translated into a fault-based number by the project leader, and validated by the manager. All validated data are sent to the SEPG, who analyzes such data and reports the analysis report back to the project team and development organization(Takagi et al., 1995).

Table 1 shows the actual project data. This data includes quality data for DF_{CR}, DF_{UT}, DF_{IT}, DF_{FT}, and DF_{ST} for 111 projects. Regarding FFD, we have actual data, but cannot show such data here due to contract obligations with the company. In Table 1, for projects No.1, No.2, No.110, and No.111, all metrics data are shown. However, since the neighboring testing activities are sometimes combined and performed as one testing, some metric data of several projects are missing.

Table 1 also summarizes the average and the median of DF_{α} in each activity. As observed, the average and the median values of DF_{α} decrease as the testing activities proceed from CR to ST. Figure 2 shows the actual values of DF's for six projects. From this figure, we can ascertain a certain trend in the successive values of DF_{CR}, DF_{UT}, DF_{IT}, DF_{FT}, and DF_{ST}. Intuitively speaking, projects No.10, No.39, and No.90 show decreasing trend in these successive DF's, and project No.17 shows increasing trend in these successive DF's.

4. Classification by Trend in Faults Detection

4.1. Key Idea

As shown in Table 1 and in Figure 2, the projects are classified into several groups by the faults detection trend (for example, the increasing or decreasing trend in the successive DF values). In order to identify the trend, we introduce a quantitative measure of the trend. Although there are many metrics for measuring trend, the rank correlation coefficient is one of the most popular metrics for a monotonic increasing or decreasing trend(Muto, 1995). In this paper, we adopt Kendall's rank correlation coefficient τ(Kendall and Gibbons, 1990).

By using Kendall's τ, we can quantitatively measure the trend of faults detection. Therefore, we define the following four types of trends based on the value of τ.

(a) Strict decreasing type (T_{SD}) ($\tau = -1$)

(b) Almost decreasing type (T_{AD}) ($-1 < \tau < 0$)

(c) Almost increasing type (T_{AI}) ($0 \leq \tau < 1$)

(d) Strict increasing type (T_{SI}) ($\tau = 1$)

Projects No.10 and No.90 with $\tau = -1$ in Figures 2(a) and 2(f), respectively, are classified into type T_{SD}. Project No.39 and project

No.71 with $\tau = -0.6$ in Figure 2(c) and Figure 2(e), respectively, are classified into type T_{AD}. Project No.45 with $\tau = 0.21$ in Figure 2(d) is type T_{AI}. Finally, project No.17 with $\tau = 1$ in Figure 2(b) is type T_{SI}. This result implies that the value of Kendall's τ is adequate for distinguishing these trends.

Here, we explain the motivation of the classification for τ. We firstly try to define the two trends: decreasing ($-1 \leq \tau < 0$) and increasing ($0 \leq \tau \leq 1$). However, based on the experience of the actual developers, we know empirically that there may be a significant difference between the projects with $\tau = -1$ (or $\tau = 1$) and $-1 < \tau < 0$ (or $0 \leq \tau < 1$). The followings present interpretations of each trend:

(a) Strict decreasing: Many faults are detected and removed in the early stage, and then a few faults are detected and removed in the later stage. Thus, only a very few faults remain in the final software product. This type project is most desirable.

(b) Almost decreasing: This type is similar to the "strict decreasing type". Thus, a few remaining faults may be found. However, an increase of detected faults seems caused by a flaw of design, by testing in previous activity, or by process management. Then, the almost decreasing type is to be a good trend type, but is not the best one.

(c) Strict increasing: A few faults are detected and removed in the early stage, and then many faults are detected and removed in the later stage. Thus, many faults still remain in the final software product. This type project is most undesirable.

(d) Almost increasing: This type is similar to the "strict increasing type". Thus many faults may remain. However, a decreasing of the trend is caused by an activity trying to recover the quality of product and process. Therefore, the almost increasing type seems better than a strict one.

Using these trends for classification, the various effects of factors such as the skill level of the development team, the kind of product, etc. are mitigated and included in the trend.

4.2. Result of Classification

Table 2 shows the result of classification by applying Kendall's rank correlation coefficient τ to 111 projects. We can see that almost 35% of projects are in type T_{SD}, and almost 50% of projects are in type T_{AD}. Thus, 85% of projects have decreasing trends. However, the remaining 15% of projects have increasing trends (that is, types T_{AI} and T_{SI}).

In order to find the relationship between the FFD and the four types, we present histograms of the FFD in Figure 3. Figure 3 also

shows the average rank of the field quality for each type. Please note that the y-axis of Figure 3 denotes the ranks of the values of FFD rather than the values themselves. Here, the rank of each project takes a value from 1 to 111. Thus, rank= 1 implies the highest quality (that is, $FFD = 0$ in actual data) and rank= 111 implies the lowest quality (for this case, we cannot show actual value of FFD by the contract). Figure 3(a) with type T_{SD} shows that 50% of projects have the rank= 1. Similarly, 3(b) with T_{AD} shows that 51.2% of projects have the rank= 1. Therefore, we can assume that projects with these types tend to produce very high quality products. On the other hand, Figure 3(d) with type T_{AI} shows that 20% of projects have the rank= 1, and Figure 3(c) with type T_{SI} shows that no project has the rank= 1. Thus, we can assume that projects with these types tend to produce lower quality products.

From Table 2 and Figure 3, we can expect that the average field quality of the projects with the types T_{SD} and T_{AD} is rather good. On the other hand, the field quality is not so good for types T_{SI} and T_{AI}. This property will be further investigated in the next section.

Figure 1. Development process

Table 1. Actual project data

No.	DF_{CR}	DF_{UT}	DF_{IT}	DF_{FT}	DF_{ST}
1	2.1	0.6	0.7	1.8	0.5
2	6.7	0.0	1.7	1.2	0.0
...	...	...	...	...	...
110	11.6	1.8	0.2	1.0	2.9
111	3.3	5.3	15.2	3.3	6.3
Average	7.3	2.2	1.4	0.7	0.8
Median	5.2	1.1	1.1	0.5	0.3

Table 2. Classification by fault detection trend

Type s	Number of projects (%)
Strict decreasing (T_{SD})	39 (35.1%)
Almost decreasing (T_{AD})	55 (49.6%)
Almost increasing (T_{AI})	15 (13.5%)
Strict increasing (T_{SI})	2 (1.8%)

5. Relationship between Trend and Field Quality

5.1. Outline of Analysis

In this section, we try to clarify the relationship between the trend and the field quality of products.

As shown in Table 2, the number of projects in T_{SI} is too few to perform statistical analysis. Therefore, we integrate T_{SI} and T_{AI} into T_I. Since our purpose is to predict whether or not the final quality becomes low, we can integrate the projects whose quality is low. Then, in order to compare the field quality of projects having three types of trends, we introduce θ, which is the parameter of the location on the rank of the FFD. In this paper, we define θ_{SD}, for projects with type T_{SD}, to be the average rank of FFD. Similarly, we define θ_{AD} and θ_I for projects with types T_{AD} and T_I, respectively. For simplicity, we call θ_{SD}, θ_{AD} and θ_I the average FFD of projects with types T_{SD}, T_{AD} and T_I, respectively. Then, from the average and the interpretations, we naturally derive the following relationship.

$$\theta_{SD} \leq \theta_{AD} \leq \theta_I \tag{1}$$

In this paper, we planned three steps of analysis for the average FFD, θ_{SD}, θ_{AD}, and θ_I of projects.

Analysis 1. (Jonckheere test(Lehmann, 1975))

By using the Jonckheere test, we can check if there is a significant ordered difference shown in Equation 1 among θ_{SD}, θ_{AD}, and θ_I.

Analysis 2. (Multiple comparison)

If the order exists among θ_{SD}, θ_{AD}, and θ_I, we perform Ryan's pairwise comparison(Iwahara, 1964) to show more detailed relations among them. Ryan's procedure consists of the repetition of the following procedure: Compare the farthermost pair on the order. When there is significant difference, the next farthermost pairs are compared.

By repeating such comparison step by step, we finally analyze a significant difference between neighboring pairs.

Analysis 3. (Fisher's exact test)

In this step, we investigate whether or not there is a significant difference among the trend groups of projects with respect to the final quality of products. We first define the criterion for the quality, and classify the project data into high quality projects and low quality projects, according to the criterion. Next, we investigate the distribution of the number of high quality projects and the number of low quality projects in the same type. If the trend is useful as the predictor, then the distribution of each type

should have significant difference. In order to evaluate, we perform Fisher's exact test.

5.2. Experimental Evaluation

Analysis 1. (Jonckheere test)

We define the following two hypotheses H_0 and H_1 for the average FFD of projects in types, θ_{SD}, θ_{AD}, and θ_I. The null hypothesis H_0 is the following relation:

$$H_0 : \quad \theta_{SD} = \theta_{AD} = \theta_I$$

On the other hand, the alternative hypothesis H_1 is the following relationship with at least one of the inequalities being strict.

$$H_1 : \quad \theta_{SD} \leq \theta_{AD} \leq \theta_I$$

In this analysis, the level of significance α is chosen as 0.1.

By the Jonckheere test, the null hypothesis H_0 is rejected at significance level $\alpha = 0.1$ level because the probability that H_0 cannot be rejected becomes 0.068. Thus, the alternative hypothesis H_1 is accepted. This result implies that there is a statistically significant difference among θ_{SD}, θ_{AD}, and θ_I of target projects as the whole, and the field quality becomes worse according to the order of T_{SD}, T_{AD},

and T_I. However, at this stage we cannot know which inequalities in the hypothesis H_1 hold.

Analysis 2. (Ryan's pairwise comparison)

Next, the result of the Ryan's pairwise comparison is summarized in Table 3. Here, the significance level α is chosen as 0.1. In Step 1, comparison of the farthermost pair, θ_{SD} and θ_I, is performed. As a result, there is somewhat significant difference between θ_{SD} and θ_I. Similarly, Step 2 shows a significant difference between θ_{AD} and θ_I, but no significant difference between θ_{SD} and θ_{AD}.

From these facts, the following equation holds for the average FFD of projects.

$$\theta_{SD} \leq \theta_{AD} < \theta_I$$

This equation implies that the average rank of the field fault density FFD becomes larger according to the order of θ_{SD}, θ_{AD}, and θ_I, and that an especially large difference exists between θ_{AD} and θ_I.

In subsections 4.1 and 4.2, we estimated that the two types, T_{SD} and T_{AD}, behave in the same way with respect to (the rank of) FFD. Similarly, two types T_{SI} and T_{AI} behave in the same way. Furthermore, the analysis result implies that there is a large difference between two types T_I (that is, T_{SI} and T_{AI}), and T_{AD}. Therefore, the estimation agrees with this result.

(a) Project No.10

(b) Project No.17

(c) Project No.39

(d) Project No.45

(e) Project No.71

(f) Project No.90

Figure 2. Trend of actual data

Table 3. Result of the Ryan's pairwise comparison

Step	Nominal significance level α'	Pair of types	p-value
1	α'=0.033	(θ_{SD}, θ_{I})	0.026
2	α'=0.067	(θ_{SD}, θ_{AD})	0.452
		(θ_{AD}, θ_{I})	0.034

Analysis 3. (Fisher's exact test)

As a result of Analysis 1 and 2, we cannot say that there is a significant difference between θ_{SD} and θ_{AD} but we can see that the trend affects the rank of FFD, that is, the final quality of product.

In this analysis, we analyze whether the value of FFD affects the trend conversely in order to show that the trend is a factor that affects the final quality. Here, we integrate T_{SD} and T_{AD} into T_D, and use them for analysis.

First, we classify the projects into two types according to the value of FFD. Here, in order to divide the projects, we introduce a threshold to the values of FFD. In the analysis we decided to use a value, which has been used in the company to discriminate poor projects, as the threshold. Unfortunately we cannot show the value itself here because of the contract with the company. However we can say that the value is quite close to the average value of FFDs of 111 projects.

Classified groups are called "High Quality (HQ)" (FFD is larger than the average) and "Low Quality (LQ)" (FFD is smaller than the average). Next, by using Fisher's exact test, we try to show the difference of the distribution of the trend between classified groups.

Table 4 shows the result of classification. As the result of Fisher's exact test using Table 4, the p-value becomes $p = 0.050$. Therefore, the null hypothesis can be rejected at the significance level $\alpha = 0.1$.

This implies that the ratios of the number of projects in the two groups (that is, "HQ" and "LQ") have significant difference between that of θ_D and that of θ_I.

5.3. Conclusion of Evaluation

In Analysis 1, we showed that there is an order among θ_{SD}, θ_{AD}, and θ_I, as a whole. The order is as follows:

$$\theta_{SD} \leq \theta_{AD} \leq \theta_I$$

Next, in Analysis 2, we showed that a significant difference only exists between θ_{AD} and θ_I. This relation is as follows:

$$\theta_{SD} \leq \theta_{AD} < \theta_I$$

From Analysis 1 and 2, we can see that the trend affects the rank of FFD. In Analysis 3, we showed conversely that the value of FFD affects the trend.

Finally, we can say that trend type, T_D and T_I, is an important factor that may estimate software quality.

6. Estimating Trend by Metrics in Design & Coding Phase

In Section 5, we showed the relationship between the trend of detected faults and the field quality. However, in order to assure the quality of software products in an earlier stage of the development, we need to analyze the factor that affects the trend and is obtained from the design & coding phase. In the practical development process, such factors are used to predict the trend in the test & debug phase. Thus, according to the result, a plan for the test & debug phase can be managed to control the final quality of the product.

6.1. Outline of analysis

In Section 5, we show the relationship between the trend of detected faults and the final software quality. Since the target is the test & debug phase, which is the last phase of the development process, we cannot use the trend directly for software quality improvement. Therefore, in order to control software quality, we use the data collected from the design and coding phase. In brief, by using such data obtained from the earlier phase, we try to estimate the trend in the test & debug phase.

Therefore, we first analyze the factor that affects the trend from the data obtained from the design & coding phase.

For analysis, we select the following four metrics that will affect the trend of detected faults in the test & debug phase based on the experience of the developers in the company. (As space is limited, we present the exact or formal definitions in subsection 6.2 only for the metrics to be utilized.)

- Frequency of detected faults in the review activities: DF_{review}
- Productivity of the team in a coding activity: P
- Review effort ratio: RR
- Coding review effort ratio: CRR

The metrics related to the review effort ratio are regarded as especially important in the company(Takagi et al., 1995).

Using the trend type T_D and T_I as the dependent variable, we first performed the logistic regression analysis that applies the stepwise method to the four metrics listed above. By performing logistic regression analysis on these four metrics, we verify which metrics are a meaningful factor of the logistic regression model. Logistic regression is a standard classification technique based on maximum likelihood estimation and has been used in software engineering.

Next, by applying Fisher's exact test, we verify the correlation between the classification using the logistic model and the classification using the rank correlation coefficient.

6.2. Result of Statistical Test

As a result of logistic regression analysis, only two metrics, DF_{review} and CRR, are shown to be significant, but neither P nor RR are serious.

Thus, we explain the definition and the meaning of the two metrics, DF_{review} and CRR, as follows:

– Frequency of detected faults in the review activities: DF_{review}

  DF_{review} is calculated from the total number of faults detected in review activities and the total time of effort of review activities at the design phase. Here, we define the following two parameters:

  D_{review}**:** the total number of detected faults in the four review activities, CDR, FDR, SDR, and MDR in the design phase.

  E_{review}**:** the total effort needed for review activities, CDR, FDR, SDR, and MDR in the design phase (measured by person-day).

  Then, DF_{review} is formulated as follows:

$$DF_{review} = \frac{D_{review}}{E_{review}}$$

– Coding review effort ratio: CRR

Coding review is the review activity performed after the coding activity. We define CRR as follows:

$$CRR = \frac{E_{CR}}{E_{CO} + E_{CR}}$$

We can assume the metrics as review effort ratio at coding activity.

A logistic regression model using these two metrics is constructed as follows:

$$p = \frac{e^{-22.391CRR - 0.149DF_{review} + 1.261}}{1 + e^{-22.391CRR - 0.149DF_{review} + 1.261}}$$

Here, p represents the probability that a project is classified into type T_I.

Table 5 shows the result of classification using the logistic regression model p and the rank correlation coefficient τ. As a result of the Fisher's exact test using Table 5, the null hypothesis is rejected by the significance level $\alpha = 0.01 (< 0.1)$. Overall, the model p tends to predict the project as T_D. However, the accuracy of prediction about T_I is 57.1%, and the one about T_D is more than 90%. That is, by using model p, we can find at the average 57.1% of projects with low quality products at the end of the design & coding phase. Thus, we can say that two metrics, DF_{review} and CRR, are the characterizing factor of the trend of detected faults. Furthermore, we can also say that model p

classifies the projects successfully so that the trends of detected faults can be estimated.

7. Conclusion

In this paper, we analyzed time series data on the number of faults detected by successive coding review and testing activities. First, by applying the rank correlation coefficient to actual project data, we have successfully classified the data into four types of trends: T_{SI}, T_{AI}, T_{AD} and T_{SD}. Next, we have investigated the relationships between trends and field quality, and showed that the software project having trend T_{AD} or T_{SD} would produce high quality products.

Moreover, we have investigated the relationships between trends and metrics collected at the design phase, and showed that DF_{review} and CRR are related to the trends of detected faults in the test phase.

As future work, since these results are obtained from data of one particular company, we need to analyze more data from various software development organizations and to generalize the results.

(a) Strict decreasing type

(b) Almost decreasing type

(c) Almost increasing type

(d) Strict increasing type

Figure 3. Histogram of FFD for each type

Table 4. Fisher's exact test

Table 5. Logistic regression model vs. rank correlation coefficient

		Classification by p	
		# of type T_D	# of type T_I
Classification by τ	# of type T_D	91	3
	# of type T_I	13	4

References

Basin, S. L.: 1973, 'Estimation of Software Error Rates via Capture-Recapture Sampling'. Technical report, Science Applications, Inc.

Bisant, D. B. and J. R. Lyle: 1989, 'A two-person inspection method to improve programming productivity'. *IEEE Trans. on Software Engineering* **15**(10), 1294–1304.

Compton, T. and C. Withrow: 1990, 'Prediction and Control of Ada Software Defects'. *Journal of Systems and Software* **12**, 199–207.

Fagan, M. E.: 1986, 'Advances in software inspections'. *IEEE Trans. on Software Engineering* **12**(7), 744–751.

Goel, A. L.: 1985, 'Software Reliability Models: Assumptions, Limitations, and Applicability'. *IEEE Trans. on Software Engineering* pp. 1411–1423.

Halstead, M. H.: 1977, *Elements of Software Science*. Elsevier.

International Standard Organization: 1990, *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990.

Iwahara, S.: 1964, *Psychological Statistics: Non-parametric Method*. Nihon Bunka Kagakusha Co.,Ltd. (in Japanese), 2nd edition.

Kendall, M. and J. D. Gibbons: 1990, *Rank Correlation Methods*. Edward Arnold, 5th edition.

Lehmann, E. L.: 1975, *Nonparametrics: Statistical Methods Based on Ranks*. Holden-Day, Inc.

Marick, B.: 1995, *The craft of software testing: subsystem testing including object-based and object-oriented testing*. NJ: Prentice-Hall.

Musa, J. D., A. Iannino, and K. Okumoto: 1987, *Software reliability: measurement, prediction, application.* McGraw-Hill.

Muto, S.: 1995, *Statistical Analysis Handbook.* Asakura Books (in Japanese), 1st edition.

Paulk, M. C., B. Curtis, and C. Weber: 1993, 'Capability Maturity Model, Version 1.1'. *IEEE Software* **10**(4), 18–27.

Smidts, C., R. W. Stoddard, and M. Stutzke: 1996, 'Software Reliability Models: An Approach to Early Reliability Prediction'. In: *Proc. of 7th International Symposium on Software Reliability Engineering.* pp. 132–141.

Takagi, Y., T. Tanaka, N. Niihara, K. Sakamoto, S. Kusumoto, and T. Kikuno: 1995, 'Analysis of review's effectiveness based on software metrics'. In: *Proc. of 5th International Symposium on Software Reliability Engineering.* pp. 34–39.

Tanaka, T., K. Sakamoto, S. Kusumoto, K. Matsumoto, and T. Kikuno: 1995, 'Improvement of software process by process description and benefit estimation'. In: *Proc. of 17th International Conference on Software Engineering.* pp. 123–132.