Software Quality Journal, 13, 177–193, 2005

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
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 focussed on the trend of time series data of software defects among successive testing phases and tried to estimate software quality using the trend. 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 and coding phase can determine the trend.

Keywords: software testing, software quality, time series data, statistical analysis

1. Introduction

Software quality assurance is important for not only satisfying 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 (Broekman and Notenboom, 2002; Horch, 2003; Marick, 1995; Marks, 1992). 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). Munson and Khoshgoftaar used several metrics related with complexity of program to predict whether the module is fault-prone or not by discriminant analysis in Munson and Khoshgoftaar (1992).

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, several methods based on process metrics have been proposed in order to predict remaining faults. Process metrics are recorded in the progress of the development process. There are several methods using regression and artificial intelligence (Khoshgoftaar and Seliya, 2002; Morgan and Knafl, 1996; Padberg et al., 2004; Yokoyama and Kodaira, 1998). In Khoshgoftaar and Seliya (2002), classification tree was used to predict the number of faults in a module based on design related metrics. In Morgan and Knafl (1996), a regression model is proposed that uses testing process metrics and product metrics as explanation variables and predicts defect rate. In Yokoyama and Kodaira (1998), the number of defects in inspection process is predicted by using 19 quality factors. In Padberg et al. (2004), neural network is applied to predict the number of faults at inspection activity by using statistics derived from inspection activity. These methods are usually applied at the start of testing phase. Thus, continuous data obtained according to the progress of testing activity are not used for prediction.

On the other hand, since there is a thought that observation of such continuous data during testing phase is important to ascertain the number of residual faults, the other methods shown in below were proposed with intention to use after a project goes into testing phase. As methods based on detected faults, there are capture-recapture model (Basin, 1973), Schneidewind's Software reliability model (Schneidewind, 1997), and the software reliability growth model (Goel, 1985; Musa et al., 1987; Lyu, 1996).

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.

We have also performed similar research so far. In Mizuno et al. (2002), a method is proposed to estimate cost satisfying a required quality (here, quality means the number of residual faults) by using metrics of effort of design and review phases. While the previous paper proposed a method estimating effort of testing, our current paper proposed the method predicting fault-proneness of product.

In the SRGM (Goel, 1985; Musa et al., 1987; Lyu, 1996), the number of failures in a unit period in a testing activity is assumed to follow a stochastic process. The stochastic process model specified by a project can be obtained by fitting this stochastic process with a record of an actual number of failures. The software release timing, which is the time to

finish testing, can be predicted based on time to failure and test time required to achieve a given number of remaining failures.

In company A, which is a certain company cooporated with us, the SRGM has been used to decide the release time of a testing activity. In most projects of company A, 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 company A 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. (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. By using the result of analysis investigating the relationship between a shape of the time series data among testing phases and software quality, we intend to supplement such a weak point of the SRGM that the SRGM cannot be applied to the data obtained from plural testing phases. Thus, datasets used are collected in projects which pass the SRGM validation.

Thus, we investigate the relationship between a shape of the time series data and software quality, and utilize this relationship in software quality prediction.

In our paper, we first classify projects by the trend of fault detection defined with quantitative method. Next, we clarify relationship between the trend and field quality of products. Finally, we try to show relationship between metrics obtained from design and coding phases and the trend, and thus, we try to estimate the quality of software product in an early phase of development.

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 company A. 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 company A. 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 and debug phase exceeds 10. This condition implies that the size of target project is relatively large. Second, the size of product exceeds 4Kstep. This condition implies that the size of product is also relatively large. 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 and coding phase and the test and debug phase. The design and 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 and debug phase consists of four activities: Unit test and debug (*UT*), Integration test and debug (*IT*), Function test and debug (*FT*), and System test and debug (*ST*).

One characteristic of the design and coding phase is that review activity is introduced after each design activity. A review activity not only improves the quality of the artifacts but also helps software development organizations reduce their cost of producing software (Fagan, 1986). In the review activity, peer review (Bisant and Lyle, 1989) is performed. The SEPG in company A establishs several guidelines for the review activity. One guideline

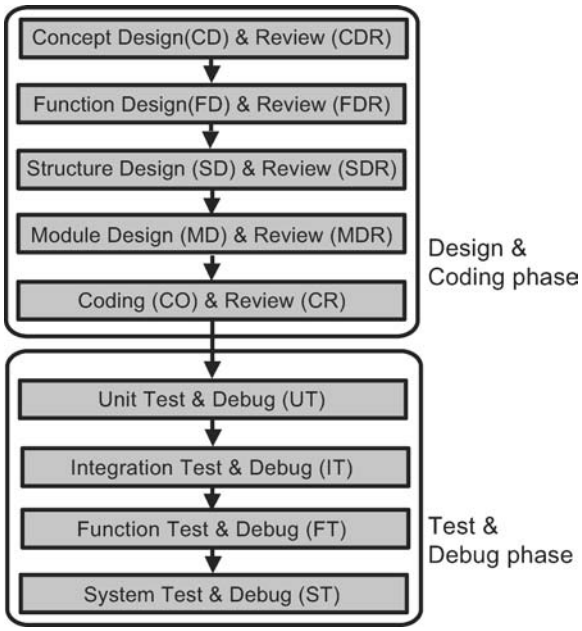


Figure 1. Development process.

directs at least 15% of the total effort of design and coding phase to be assigned to review activities (Takagi et al., 1995).

The test and 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_α

DF_α is the number of detected faults normalized by the effort needed for an activity α in Figure 1, which is perfomed 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, which represent the number of detected faults in an activity α and the effort need for an activity α (measured by person-day),

respectively. By using these metrics, DF for activity α is defined as follows:

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

where, α denotes one of activities CR, UT, IT, FT, and ST.

(2) Field fault density: FFD
In company A, the field quality of the final product is measured by the metrics named FFD. This metric is defined as follows:

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

Here, S_{final} represents the product size and is represented by the unit $KLOC$ and D_{field} is defined as the number of faults detected during six months after delivery in company A.

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 without FFD because we cannot show them due to contract obligations with company A. Here, since the neighboring testing activities are sometimes combined and performed as one testing, some metric data of several projects are actually 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

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

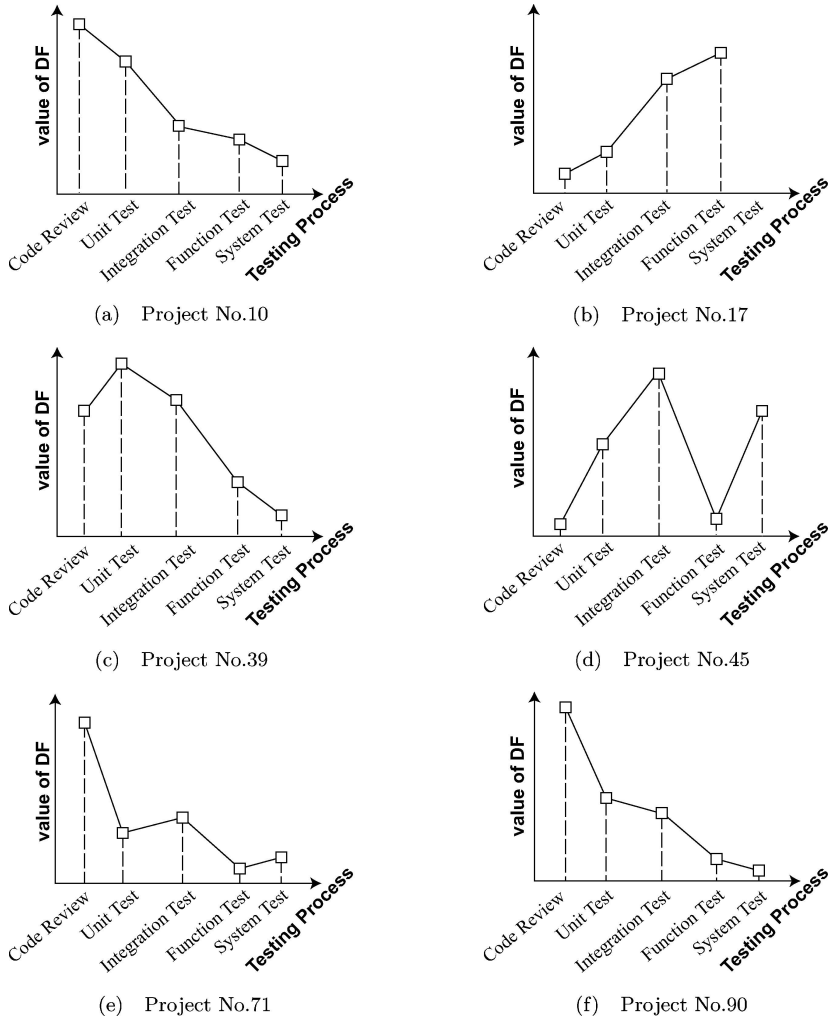


Figure 2. Trend of actual data.

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 (f), respectively, are classified into type T_{SD}. Project No. 39 and project No. 71 with $\tau = -0.6$ in Figure 2(c) and (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.

Table 2. Classification by fault detection trend.

Types	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%)

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,

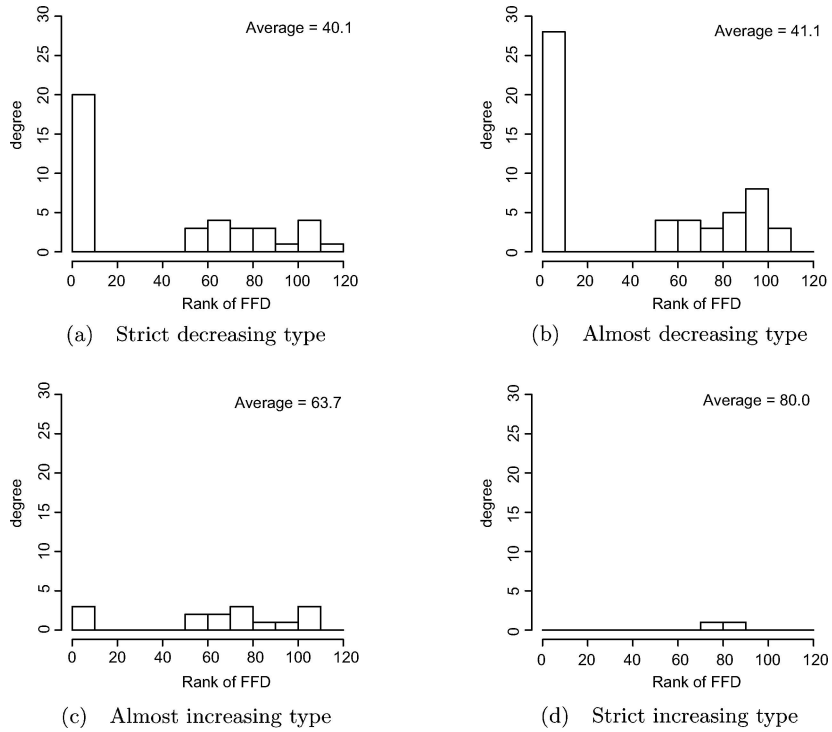


Figure 3. Histogram of *FFD* for each type.

rank = 1 implies the highest quality (that is, $FFD = 0$ in actual data) and rank = 111 implies the lowest quality. Figure 3(a) and (b) show that about 50% of projects in each type have the rank = 1 (because the degree of bar in left-side is equal to the number of projects producing fault-free product). Therefore, we can assume that projects with these types tend to produce very high quality products. On the other hand, Figure 3(c) and (d) shows that a few projects have 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.

5. Analysis for trend metrics

5.1. Relationship between trend and field quality

5.1.1. Outline of analysis 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[1] 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. First, we check whether there is a significant difference among θ_{SD}, θ_{AD}, and θ_I. That is, the following null hypothesis must be rejected:

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

Next, as shown in Equation (1), θ_{SD}, θ_{AD}, and θ_I must have ordered relation especially. That is, alternative hypothesis deals with a significant ordered differences. Thus, alternative hypothesis becomes the following equation:

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

Jonckheere test is one that is appropriate for such situation.[2]

Analysis 2. (Multiple comparison)

If the order exists among θ_{SD}, θ_{AD}, and θ_I, we perform Ryan's pairwise comparison (Hochberg and Tamhane, 1987) to show more detailed relations among them. In this procedure, the overall null hypothesis H_0, alternative hypothesis H_1 against H_0, and significance level α must be defined. In our analysis, we defined $\alpha = 0.1$, and H_0 and H_1 are defined as follows:

$$H_0 : \ \theta_{SD} = \theta_{AD} = \theta_I$$
$$H_1 : \ \theta_{SD} \leq \theta_{AD} \leq \theta_I$$

Then, subset hypotheses consisting of all possible pairs of groups are defined. In our analysis, the number of subset hypotheses is ${}_3C_2 = 3$ and alternative hypotheses are $\theta_{SD} \leq \theta_I, \theta_{SD} \leq \theta_{AD}$, and $\theta_{AD} \leq \theta_I$. Next, for each hypothesis, the nominal significance level α' that is used instead of the significance level α is defined. By using the overall significance level α, the nominal significance level α' is given as follows:

$$\alpha' = \frac{2\alpha}{k(m-1)}$$

where, k is the number of groups (in our case, 3) and m is a distance defined as the number of group X_p satisfying $X_i \leq X_p \leq X_j$ in H_1. Here, X_i and X_j are pair in a concerned hypothesis. Thus, the distance of θ_{SD} and θ_I is 3, in our analysis.

By using above subset hypotheses and nominal significance level α', multiple comparison is performed. The order of comparisons is determined according to descending order of m. For each pairwise comparison, Mann-Whitney test is performed. In our analysis, the subset null hypothesis $\theta_{SD} = \theta_I$ is firstly tested. If the subset null hypotheses are rejected, then test all of subset hypotheses that have larger α' (except those are that have pair X_p and X_q satisfying $X_i \leq X_{p,q} \leq X_j$, where X_i and X_j is of hypothesis retained). Continue in this manner until no subset remains to be tested.

By repeating such comparison step by step, we finally test whether there is a significant difference between each neighboring pair.

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.

5.1.2. 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

Table 3. Result of the Ryan's pairwise comparison.

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

relation:

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

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

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

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 Sections 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.

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.

Table 4. Fisher's exact test.

	FFD	
Types	LQ	HQ
T_D	29	65
T_I	10	7

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 company A to discriminate poor projects, as the threshold. Unfortunately we cannot show the value itself here because of the contract with company A. However we can say that the value is quite close to the average value of *FFD*s 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.

5.1.3. 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.

5.2. Relationship between trend and early phase activity

In order to control software quality, we show the relationship between trend metrics and metrics obtained from design and coding phase.

Table 5. Logistic regression model vs. rank correlation coefficient.

		Classification by logistic model	
		# of Type T_D	# of Type T_I
Classification by τ	# of Type T_D	91	3
	# of Type T_I	13	4

With the trend metric as the dependent variable (this variable may take T_D or T_I), we first performed the logistic regression analysis that applies the step-wise method to metrics recorded at design and coding phase in company A. Next, by applying Fisher's exact test, we verify the correlation between the classification result using the logistic model and the classification using the rank correlation coefficient.

As a result of logistic regression analysis, only two metrics, DF_{review} which is frequency of detected faults through all review activities, and CRR which is percentage of code review effort in effort of coding phase, are shown to be significant. A logistic regression model using these two metrics is constructed as follows:

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

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

For Fisher's exact test, we show Table 5. The sum of numbers in each row represents the number of projects classified by the rank correlation coefficient. The sum of numbers in each column represents the number of projects classified by logistic regression model. As a result of the Fisher's exact test, the null hypothesis is rejected by the significance level $\alpha = 0.01$ (<0.1). Thus, we can say that two metrics, DF_{review} and CRR, are the characterizing factor of the trend of detected faults.

6. 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.

Appendix

Appendix A1: Jonckheere test

Let the data consist of $N = \sum_{j=1}^{k} n_j$ observations, with n_j observations from the jth treatment, $j = 1, \ldots, k$.
To test H_0 of the form

$$H_0 : \ \theta_1 = \theta_2 = \cdots = \theta_k,$$

against an alternative H_1 of the form

$$H_1 : \ \theta_1 \leq \theta_2 \leq \cdots \leq \theta_k,$$

where at least one of the inequalities is strict,

1. Compute $k(k-1)/2$ Mann-Whitney counts U_{uv}, $u < v$, where

$$U_{uv} = \sum_{i=1}^{n_u} \sum_{i'=1}^{n_v} \phi(X_{iu}, X_{i'v})$$

   and $\phi(a, b) = 1$ if $a < b$, $1/2$ if $a = b$, 0 otherwise. That is, U_{uv} is the number of sample u before sample v precedences.
2. Let

$$J = \sum_{u<v}^{k} U_{uv} = \sum_{u=1}^{k-1} \sum_{v=u+1}^{k} U_{uv}$$

   be the sum of these $k(k-1)/2$ Mann-Whitney counts.
3. At the α level of significance,

$$\begin{aligned} &\text{reject} \quad H_0 \quad \text{if} \quad J \geq j(\alpha, k, (n_1, \ldots, n_k)), \\ &\text{accept} \quad H_0 \quad \text{if} \quad J < j(\alpha, k, (n_1, \ldots, n_k)), \end{aligned}$$

   where the constant $j(\alpha, k, (n_1, \ldots, n_k))$, which satisfies the equation $P_0\{J \geq j(\alpha, k, (n_1, \ldots, n_k))\} = \alpha$.

Notes

1. Here, the parameter of the location indicates the median of population since all statistical tests in our research are nonparametric ones.
2. For more concrete procedure of the Jonckheere test, please consult Appendix A1.

References

Basin, S.L. 1973. Estimation of software error rates via captrue-recapture sampling, Technical report, Science Application, Inc.

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

Broekman, B. and Notenboom, E. 2002. *Testing Embedded Software*, Addison-wesley.

Compton, T. and Withrow, C. 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*, 1411–1423.

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

Hochberg, Y. and Tamhane, A.C. 1987, *Multiple Comparison Procedures*, Wiley.

Horch, J.W. 2003, *Practical Guide to Software Quality Management*, Artech House Publishers.

International Standard Organization. 1990. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12–1990.

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

Khoshgoftaar, T.M. and Seliya, N. 2002. Tree-based software quality estimation models for fault prediction, In *Proc. 8th IEEE International Symposium on Software Metrics*, pp. 203–214.

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

Lyu, M.R. (ed.). 1996. *Handbook of Software Reliability Engineering*, McGraw Hill.

Marick, B. 1995. *The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing*, Prentice-Hall, NJ.

Marks, D.M. 1992. *Testing Very Big Systems*, McGraw-Hill.

Mizuno, O., Shigematsu, E., Takagi, Y., and Kikuno, T. 2002. On Estimating testing effort needed to assure field quality in software development, In *Proc. of 13th International Symposium on Software Relaibility Engineering (ISSRE2002)*, Annapolis, MD, USA, pp. 139–146.

Morgan, J.A. and Knafl, G.J. 1996. Residual fault density prediction using regression methods, In *Proc. 7th International Symposium on Software Reliability Engineering*, pp. 87–92.

Munson, J.C. and Khoshgoftaar, T.M. 1992. The detection of fault–prone programs, *IEEE Trans. on Software Engineering* 18(5): 423–433.

Musa, J.D., Iannino, A., and Okumoto, K. 1987. *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill.

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

Padberg, F., Ragg, T., and Schoknecht, R. 2004. Using machine learning for estimating the defect content after an inspection, *IEEE Trans. on Software Engineering* 30(1): 17–28.

Paulk, M.C., Curtis, B., and Weber, C. 1993. Capability maturity model, version 1.1, *IEEE Software* 10(4): 18–27.

Schnieidewind, N.F. 1997. Software metrics model for integrating quality control and prediction, In *Proc. 8th International Symposium on Software Reliability Engineering*, pp. 402–415.

Smidts, C., Stoddard, R.W., and Stutzke, M. 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., Tanaka, T., Niihara, N., Sakamoto, K., Kusumoto, S., and Kikuno, T. 1995. Analysis of review's effectiveness based on software metrics, In *Proc. of 5th International Symposium on Software Reliability Engineering*, pp. 34–39.

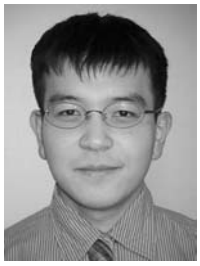
Tanaka, T., Sakamoto, K., Kusumoto, S., Matsumoto, K., and Kikuno, T. 1995. Improvement of software process by process description and benefit estimation, In *Proc. of 17th International Conference on Software Engineering*, pp. 123–132.

Yokoyama, Y. and Kodaira, M. 1998. Software cost and quality analysis by statistical approaches, In *Proc. 20th International Conference on Software Engineering*, pp. 465–467.

Sousuke Amasaki received the B.E. degree in Information and Computer Sciences from Okayama Prefectural University, Japan, in 2000 and the M.E. degree in Information and Computer Sciences from Graduate School of Information Science and Technology, Osaka University, Japan, in 2003. He has been in Ph.D. course of Graduate School of Information Science and Technology at Osaka University. His interests include the software process and the software quality assurance technique. He is a student member of IEEE and ACM.

No photo available

Takashi Yoshitomi received the B.E. degree in Information and Computer Sciences from Osaka University, Japan, in 2002. He has been working for Hitachi Software Engineering Co., Ltd.



Osamu Mizuno received the B.E., M.E., and Ph.D. degrees in Information and Computer Sciences from Osaka University, Japan, in 1996, 1998, and 2001, respectively. He is an Assistant Professor of the Graduate School of Information Science and Technology at Osaka University. His research interests include the improvement technique of the software process and the software risk management technique. He is a member of IEEE.

Yasunari Takagi received the B.E. degree in Information and Computer Science, from Nagoya Institute of Technology, Japan, in 1985. He has been working for OMRON Corporation. He has been also in Ph.D. course of Graduate School of Information Science and Technology at Osaka University since 2002.

Tohru Kikuno received the B.E., M.Sc., and Ph.D. degrees in Electrical Engineering from Osaka University, Japan, in 1970, 1972, and 1975, respectively. He joined Hiroshima University from 1975 to 1987. Since 1990, he has been a Professor of the Department of Information and Computer Sciences at Osaka University. His research interests include the analysis and design of fault-tolerant systems, the quantitative evaluation of software development processes, and the design of procedures for testing communication protocols. He is a member of IEEE and ACM.