# A proposal for a low-cost model of software process evaluation

Nobuhiro Kataoka *, Tsukasa Kudou **

* School of Information and Telecommunication Eng., Tokai University, Japan
kataoka9@tokai.ac.jp
**Mitsubishi Electric Information Systems Corporation, Japan
kudou-tsukasa@mdis.co.jp

***Abstract*** - Well-known process level evaluation methods used by software companies include CMMI certification and ISO/IEC TR 15504 process assessment, both of which are hearing methods with evidence base. These methods create a heavy workload for organizations that perform process evaluations as well as for the organizations being evaluated. However, since software companies collect various data in their everyday development work, we believe that the process level of such companies can be easily evaluated from these data, thus reducing the financial and other burdens of such evaluations. In this paper, we propose a low-cost model that uses these data to evaluate the process. The data used in the model include the specifications review man-hour rate, the test specifications review man-hour rate, and the software reuse rate. The model calculates the partial correlation coefficient in which the objective functions are productivity and number of bugs, and the explanatory variables are the three data items described above. Using this model, we can evaluate the amount of correlation from the results, and the strengths and weaknesses of an organization's process are revealed. We applied this model to an actual company's data and were able to determine the strengths and weaknesses of the company's software process from an evaluation of the results. Thus, the effectiveness of the proposed model in evaluating the software process at low cost was confirmed.

***Keywords***: software process, process evaluation, low cost, assessment, productivity

## 1 INTRODUCTION

Some well-known process level evaluation methods used by software companies include CMMI (Capability Maturity Model Integration)[1] certification and ISO/IEC TR 15504[2] process assessment. Both are hearing methods with evidence base. These create a heavy load for organizations that perform process evaluations as well as for the organizations being evaluated.

However, software companies in their everyday development work collect various data and we believe that the process level of software companies can be easily evaluated from these data, thus reducing the financial burden of such evaluations.

In this paper, we propose a low-cost model that uses these data to evaluate the process, specifically the effect of software reuse, design specifications review, and test specifications review, at low cost. In this way, we raise awareness of these activities within the organization and also tie such awareness to making further improvements in the process.

Evaluation is performed by using the value of the partial correlation coefficient in which the objective functions are productivity and number of bugs present, and the explanatory variables are the specifications review man-hour rate, the test specifications review man-hour rate, and the software reuse rate. When there is an area that does not produce a good effect, this indicates where in the software development process a problem exists.

In this way, the weak points in the software development process can be determined using our model. As a next step, more detailed analysis is required to pinpoint the exact areas for process improvement.

To determine whether the proposed model was effective for an initial general evaluation of the process at low cost, we used actual system data to evaluate the model. As a result, we were able to determine the strengths and weaknesses of the organization's process, thereby confirming the effectiveness of our model.

The remainder of this paper is organized as follows. Section 2 presents the research situation and current challenges in process evaluation, and Section 3 introduces the proposed model. Section 4 describes the results when actual system data were applied to the proposed model, Section 5 presents the overall evaluation results, Section 6 discusses the significance of our model, and Section 7 provides concluding remarks.

## 2 METHOD OF SOFTWARE PROCESS EVALUATION

Simply acquiring ISO 9001[3] certification does not improve the software development process of an organization. Rather, process improvement is carried out on a daily basis using a quality system that has been developed over time. In addition, to acquire CMMI certification, the process level is certified according to the level of the certification acquired, but great expense and effort is necessary for this. The software development process can otherwise be evaluated by the process assessment of ISO/IEC TR 15504, however this also requires much effort.

It is well known that upstream review plays a considerable role in improving the quality of software [4].
A method has been proposed to improve the efficiency of review methods [5], and there is also a proposal on how to zero-in on a process that has become a problem [6]. In addition, Komuro et al. collected three data items-review

speed, indication density, and review efficiency-and, using a statistical technique, identified problems and made improvements, confirming the improvement of the review effect and cost reduction [7].

All these improvement methods are for organizations that have processes of a certain level, and only focus on reviews that have a strong relationship to quality. However, there are many organizations worldwide that do not reach this level. In such organizations, advancing to the next step comes only after analyzing the strengths and weaknesses of the existing processes of the organization. Thus, a low-cost process diagnosis model requiring only collected data is necessary.

## 3 PROPOSED MODEL

We used the following three data items in our proposed model: 1) design specifications review man-hour rate; 2) test specifications review man-hour rate; and 3) source code reuse rate. The data we used as evaluation result data were the development productivity and the bug rate (number of bugs per line of source code). Our reasons for choosing these data were as follows. The importance of review in software development is well known. Software development is human intellectual work, and there are many elements in which we may make errors, for example, through a lack of technical skills, incorrect beliefs, or discrepancy with the work of other people.

In such a situation, 7.3.1 b for ISO 9001-2000 requires that a review suited for each stage be performed, and the inspection propriety confirmed. In addition, CMMI requires a review known as "a pure review" in the third level key process area.

On the other hand, the reuse of source code has been said to be the trump card of productivity improvement. It is natural that productivity will improve with code reuse, but the impact on productivity changes greatly depending on the type of code reuse. In other words, the impact on productivity and quality changes greatly depend on whether it is reuse of code at source level or whether it is reuse of modules. Recently, a number of tools that automatically generate source code have been developed. However, only the skeleton of the source code is generated by design specifications and the details must be added by the user. In addition, due to the popularization of Java, the use of modules is spreading with the use of JDK and Java Beans, but application logic for domains are limited to reuse among sections. Moreover, in certain domains the product is used as a framework after system development is complete, and reuse is planned by changing some part of the system, thus improving the productivity.

### 3.1 The development model

Figure 1 presents the development model which is the premise of the proposed model. This development model is a general waterfall model, but we will describe it in order to eliminate any possible misunderstanding. In the figure, we show the flow of program development, the stage where the program reuse process is performed, and the stage where the review is done.

In addition, we clarify the meaning of productivity of the overall software, productivity of program development, total bugs, combinatorial testing bugs, and system testing bugs.

### 3.2 An evaluation model

(1) In the evaluation model, we assume that the following data are collected at each project.

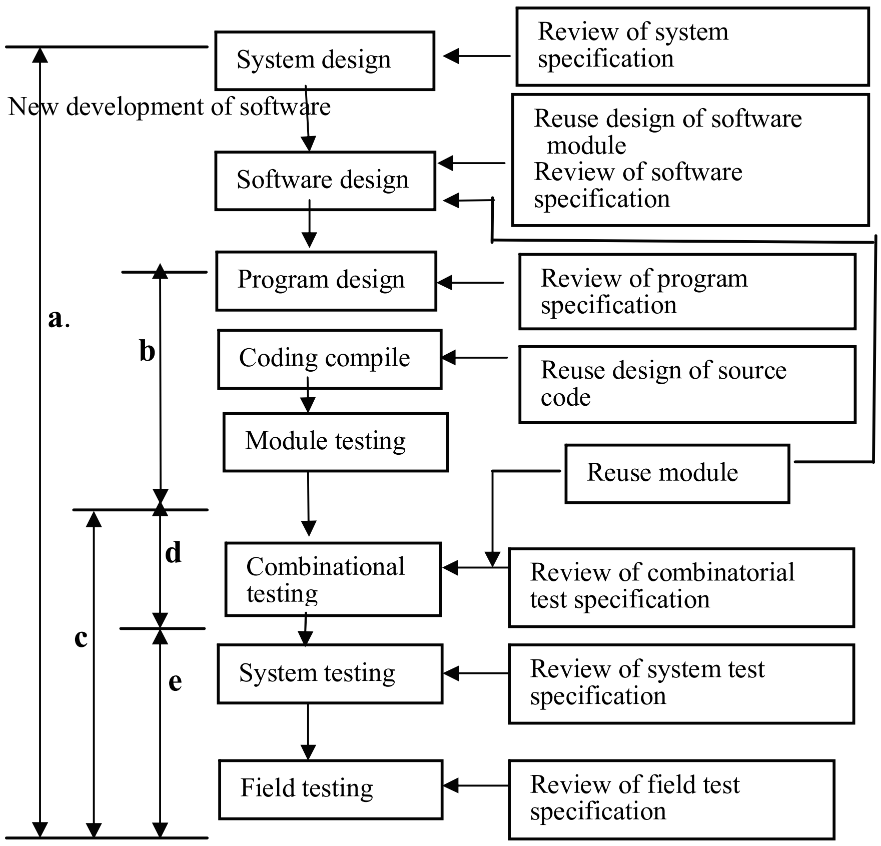
- All of the software development lines of code
- New production lines of code
- Reused lines of code
- Man-hours throughout the entire development
- Man-hours creating program specifications + man-hours creating the program + man-hours testing modules (hereafter, program development man-hours)
- Source code reuse rate
- Man-hours used for the specifications review
- Man-hours used for the test specifications review
- Number of bugs in combinatorial testing
- Number of bugs in system testing (including field testing)
- Man-hours for bug repair from combinatorial testing to field testing (bug repair man-hours)

We selected these data items assuming that many software companies collect them. Of importance in our model is that it can perform analysis even if some data are absent without impacting the overall situation, and it can also watch over wider overall tendencies. The reason we did not include review man-hours of the source code in the model is that we determined that the numbers of cases in which the source was reviewed were few, with the exception of specific parts such as important parts of the system or parts made by new employees.

(2) Data classification and collection

Here, we describe how to create the evaluation data from the above-mentioned data

- Total productivity: Lines of source code from the entire software development / man-hours used throughout the entire development
- New total productivity: Lines of source code from the new production / man-hours used throughout the entire development
- Program development productivity: Lines of source code from the entire software development/ man-hours used throughout the program development
- New program productivity: Lines of source code from the new development/ man-hours used in new partial program development
- Specifications review man-hour rate: (system specifications + software specifications + program specifications) man-hours used in review / man-hours used throughout the entire development
- Total man-hours used for bug repair: Bugs repair man-hours used in combination testing, system testing and field testing
- Number of bugs in combinational testing: Number of bugs in combinational testing
- Number of bugs in system testing: Number of bugs in system and field testing.


System design
Review of system specification
New development of software
Software design
Reuse design of software module
Review of software specification
Program design
Review of program specification
a.
b
Coding compile
Reuse design of source code
Module testing
Reuse module
d
Combinational testing
Review of combinatorial test specification
c
e
System testing
Review of system test specification
Field testing
Review of field test specification

a. whole software productivity b. program development productivity
c. total bugs d. combination testing bugs e. system testing bugs

Fig.1 Process of software development

## 3.3 Desirable value for the model

The proposed model calculates the partial correlation coefficient in which the objective functions are productivity and number of bugs present, and the explanatory variables are specifications review man-hour rate, test specifications review man-hour rate, and software reuse rate. The number of bugs is normalized by dividing it by a development line of source code. It is necessary to use partial correlation analysis because in simple correlation other elements are impacted, and it is not possible to perform evaluation of the exact degree of impact. As a result, we believe that the desirable tendency is as shown in Table 1. The table is explained as follows.

(1) Source code reuse rate:

It was natural that the total productivity and source code reuse rate would show a high correlation, so we set this value to 1.0. For the productivity of a new development part, there is a system framework, and it shows high correlation if we have a style change or addition for the framework. However, because all development is not done in this style, we cannot expect it to be 1.0. Therefore, in Table 1 we set this value slightly lower than 1.0, at 0.8 as a temporary value.

It is natural that entire program productivity show a high correlation, so we set this to 1.0 We set the correlation value to 0 for real new parts which will have no correlation.

Because bugs are not included if the reuse of source code is at the program level or module level, bug repair man-hours might be in high negative correlation. In addition, the number of bugs in combinatorial testing and system testing may have a high negative correlation for similar reasons. In the case of reuse of source level code, it is very likely that bugs will be created in change stages and insertion stages even if bugs were not in the original source code, so there will not always be a high negative correlation. Therefore, we temporarily set a low value of -0.8 rather than a value of -1.0 here.

(2) Specifications review man-hour rate:

When the review man-hour rate is high, an improvement in productivity and a reduction effect in the number of bugs are recognized for the following reasons.

Here, we are assuming an organization with a 5% or lower review man-hour rate, not organizations with more than a 10% man-hour rate.

**Table 1 Desired value of this model**

| | Total development productivity | | Program productivity | | Bug repair total man-hours | Number of bugs | |
|---|---|---|---|---|---|---|---|
| | Total | New part | Total | New part | | Combinational testing | System testing |
| Software reuse rate | 1.0 | 0.8 | 1.0 | 0 | −0.8 | −0.8 | −0.8 |
| Specifications review man-hour rate | 1.0 | 1.0 | 1.0 | 1.0 | −0.2* | −0.2* | −0.2* |
| Test specifications review man-hour rate | 0.8** | 0.8** | 0.8** | 0.8** | −1.0 | 1.0 | −1.0 |

* We expected some negative correlation and so used - 0.2.
** We expected considerable positive correlation and so used 0.8.

Total productivity, new part productivity, the entire program productivity, and new program productivity should show a high correlation due to the reduction of rework man-hours by the review man-hour rate of system specifications, software specifications, and program specifications.

Therefore, we set this value to 1.0. Likewise, program productivity should show a strong positive correlation with the review man-hours rate. This is similar for the entire program and new parts. Therefore, we set this value to 1.0.

Because errors are extracted by the specifications review, the bug repair man-hours will show a negative correlation. In the specifications review it is difficult to perform a review with enough detail that enable program errors to be extracted, so we cannot expect a large value as negative correlation. Therefore, this is temporarily set to - 0.2. We temporarily set combinatorial testing and the number of bugs of system testing to - 0.2 for similar reasons.
(3) Test specifications review man-hour rate:
If the test specifications review man-hour rate is high, the bugs in combinatorial testing increase exponentially and will show a strong positive correlation with combinatorial testing. Therefore, we set this value to 1.0. On the other hand, by detecting a bug by combinatorial testing early, the number of bugs can be reduced in the system testing plus field testing (hereafter, "system testing"), so we expect a strong negative correlation. Therefore, we set this value to -1.0. In addition, the repair costs for a bug are large in system testing for combinatorial testing, so a strong negative correlation was expected for the bug repair cost. Therefore, we set this correlative value to -1.0. Because the bug repair cost decreases, a strong correlation with total productivity can be expected.

However, this correlation is not, in actuality as strong as anticipated. In addition, it is similar to the productivity of total and new development parts. Therefore, we set this value to 0.8. Because we can say that the productivity of program development will be similar, also we set this value to 0.8.

## 3.4 Contribution rate for total productivity

A characteristic of this model is the fact that we can evaluate a single year of data. However, it can be used to evaluate the contributing causes for improvements in productivity.

For example, we assume that α is the productivity improvement rate for the current year over the previous year. "A" is the growth of the source code reuse rate, "B" is the growth of the design specifications review man-hour rate, and "C" is the growth of the test specifications review man-hour rate.

In addition, we set each correlation value for total. productivity as a, b, and c, and set each contribution rate for each productivity as βa, βb, and βc. First we find βa', βb', and βc', which are the apparent contribution rates for each element. This can be expressed by Equation 1:

$$\beta a' = A \times \frac{a}{a+b+c}$$
$$\beta b' = B \times \frac{b}{a+b+c} \qquad (1)$$
$$\beta c' = C \times \frac{c}{a+b+c}$$

Next, we distribute the total contribution rate using the apparent contribution rate, and calculate the actual contribution rate. This can be expressed by Equation 2:

$$\beta a = \frac{\alpha \times \beta a'}{\beta a' + \beta b' + \beta c'}$$
$$\beta b = \frac{\alpha \times \beta b'}{\beta a' + \beta b' + \beta c'} \qquad (2)$$
$$\beta c = \frac{\alpha \times \beta c'}{\beta a' + \beta b' + \beta c'}$$

The following shows a calculation result using a data sample.

$\alpha$ =20%   A=30%   B=10%   C=15%
a=0.8   b= 0.9   c=0.85
$\beta$ a'=30×0.8/2.55= 9.4 %
$\beta$ b'=10×0.9/2.55= 3.5 %
$\beta$ c'=15×0.85/2.55= 5.0 %

$\beta$ a =20%×9.4/(9.4+3.5+5.0)=10.5 %
$\beta$ b =20%×3.5/(9.4+3.5+5.0)=3.9 %
$\beta$ c =20%×5.0/(9.4+3.5+5.0)=5.6 %

## 4 EVALUATION OF MODEL USING DATA FROM AN ACTUAL COMPANY

We evaluated the model using data from an actual company's projects. The data was derived from 30 projects carried out during a certain year, and the development number of source code lines varied from 5KL to 700KL, with an average of 120KL. In addition, reuse rate in the domain concerned was considerably high at about 50% on average, and the highest projects were at 90%. Of course, there were also projects at 0%.

The results from an evaluation of these data using our model are shown in Table 2. Next, we will describe the analysis results. In addition, we show the desired value of the model and the real data in a radar chart in Figure 2. However, we reversed the positives and negatives in this chart because a decrease in the number of bugs is desirable. However, since an increase in the specifications review man-hour rate is desirable, we do not reverse the positives and negatives here.

(1) Correlation with source code reuse rate
The correlation between total productivity and source code reuse rate is close to the desired value. In addition, the correlation of the productivity of a new development part with the source code reuse rate is also close to the desired value. This means that when there is a framework for the entire system, repetitive development is done, such as adding new functionality. As a reuse method, we can determine that this is a desirable development process.

The program production correlation is lower than the impact of total productivity. As for this, reuse of the source code for reuse of the design specifications might require extra work. We believe this is because it is reuse of the source code itself, and not reuse of parts of the source code. If parts of the source code are reused, the correlation should be of the same degree as the correlation with total productivity. There is no correlation with the productivity of new parts of program productivity. This is in accordance with the expected value.

Bug repair man-hours have a value close to the expected value. We believe this to be caused by the reduction in the number of system test bugs. However, in regards to the number of bugs in combinatorial testing and system testing, they were considerably different from the expected values. We believe that this is because the source code was reused, and it was not just reuse involving most of the system or parts of the system.

(2) Correlation with specifications review rate
In total development productivity, the total productivity and new partial productivity both had a correlative value close to the expected value. We believe that a sound design review was performed for the system specifications and software specifications. We believe that the correlation of new parts was particularly high because many items are pointed out for design review because they are new, so the review effect is high.

For program productivity, total productivity displays a considerably high correlation. For new parts, the correlation is considerably lower than the expected value. We believe that a sound review could not be done unless someone who understands the contents of the program specifications performs the review.

There is no clear correlation with number of bugs, and the results do not accord with the expected values at all. This means that the review of the program specifications was not done to the error extraction level.

This suggests that, in order to extract a program bug, someone who understands the program must do the review of the program specifications, or a review of the source code must be done.

(3) Effect of the test specifications review
The review rate of specifications has some correlation with total productivity. We believe that it has the effect of reducing the number of bugs repair man-hours. However, no correlation is observed with new production parts.

Table 2: Example of estimate of partial correlation coefficient using actual company data

| | Total development productivity | | Program productivity | | Bug repair total man-hours | Number of bugs | |
|---|---|---|---|---|---|---|---|
| | Total | New part | Total | New part | | Combinational testing | System testing |
| Software reuse rate | 0.94* | 0.87* | 0.90* | 0.09 | −0.82* | −0.44 | −0.54 |
| Specifications review man-hour rate | 0.92* | 0.95* | 0.88* | 0.69 | 0.20 | 0.14 | 0.30 |
| Test specifications review man-hour rate | 0.71 | 0.32 | 0.73 | 0.53 | −0.81* | 0.84 | −0.35 |

* shaded areas denote statistically significant results

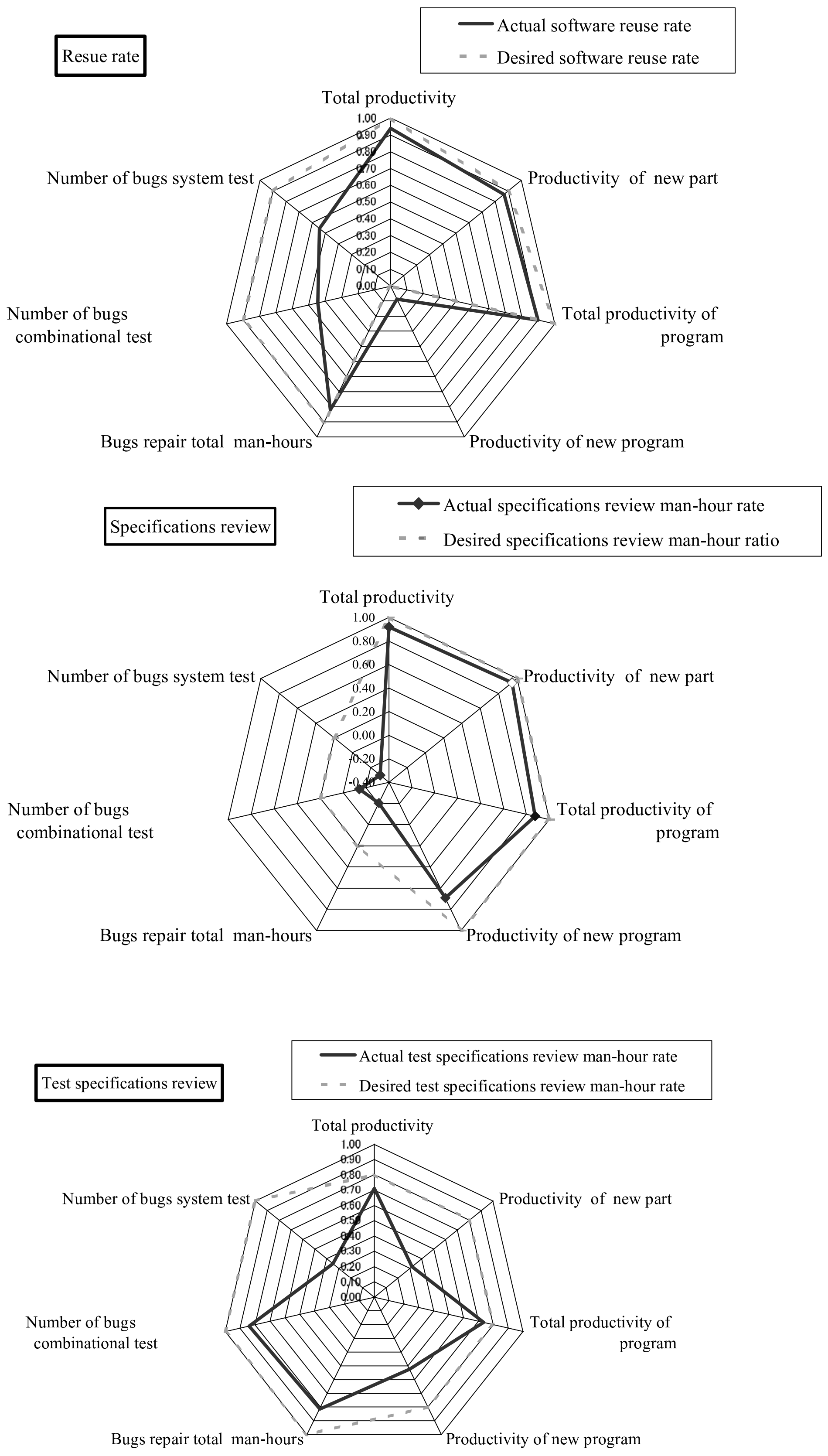


Fig. 2 Radar chart of desired and actual partial correlation coefficient

It is easy to carry out a review of specifications for existing parts with a fixed architecture, whereas for new parts, it is difficult to do a review of test specifications. It is similar in the case of the correlation with program productivity.

In regards to the correlation with bugs, man-hours used to repair all bugs shows a considerable negative tendency and, in this aspect, is close to the expected value. In addition, the number in combinatorial testing shows a high positive correlation, and this is in accordance with the tendency of the expected value. However, the number of bugs in system testing shows a negative correlation tendency, but the value is insufficient. This shows that the test specifications review is not yet sufficient, and as a result, the bugs could not be completely caught in combinational testing.

(4) Contribution rate for total productivity

In this case there was no data on the productivity improvement rate from the previous year so were not able to perform an evaluation. However, in Section 3.4 we showed that a, b, and c for the reuse rate, specifications review man-hour rate, and test specifications review man-hour rate were 0.94, 0.92, and 0.71, respectively. Therefore, we presume that entire productivity is probably considerably dependent on the first two.

## 5. EVALUATION

From the analysis we could analyze the strengths, weaknesses, and areas for improvement for this organization, thereby confirming the effectiveness of our model. The results can be summarized as follows.

(1) Strengths in the organization’s process

Reuse of software depends on the design specifications framework, and this is reused for system designs and software designs. By further promoting this in the future, we can expect higher productivity and quality improvements from software reuse. Specifications reviews are largely being done in the upstream process. Test specifications reviews have a considerable effect on reducing the number of bug repair man-hours

(2) Weaknesses in the organization’s process

We believe the following are necessary process improvement items. In software reuse, the reuse must be approached not in terms of source code, but in terms of parts, or as a framework. Downstream process reviews such as program specifications reviews do not show a large enough effect. Specifically, they do not have an effect on bug extraction. We believe that there are a few review effects for new programs in particular and the following items are necessary to efficiently review a new program: a review checklist, setting the numerical quota of bug search items, and an efficient explanation of the specifications by the creator. A checklist is also necessary for an efficient specifications explanation, and performing a prior check based on this checklist is also one method.

In test specifications reviews, a more thorough review of newly produced parts is required. This also requires a review checklist, setting numerical quota for bug search items, and an efficient explanation of the specifications by the creator.

(3) Effectiveness of the model

From the previously mentioned analysis results, we were able to make an analysis of the overall strengths and weaknesses of the organization. In addition, as a result of a hearing undertaken with the development section, we heard the opinion from many that some of the items were vague. Presenting items with a clear number makes it easy for managers to choose the necessary action, and it increases their sensitivity towards developers. This shows the effectiveness of this mode

## 6 DISCUSSION

Detailed data was collected on the organization chosen for evaluation and the various data were consolidated for model evaluation. We believe that there are a number of organizations that collect these data but do not use it in this way, and our model is likely to be beneficial for such organizations.

As a next step, it is necessary for them to take measures to improve their weak points. While various methods have been proposed, it is necessary to adopt an improvement method according to the organization’s specific situation [8][9][10][11]. In addition, it is necessary to extract problem processes from upstream specification review results. This will cause the effectiveness of the review to increase.

We have previously proposed such a method [6]. With this method, when upstream process management is insufficient, we focus our attention on the fact that the effect could appear in the each bug reports of the test process in the downstream processes. We can analyze the bug reports and identify groups that we believe to have characteristics similar to the process impact. Then, by following the process characteristics noted in ISO/IEC 15504, and the combinatorial table from the work output, we can determine the particular upstream processes that were the cause of most of the bugs occurring in the downstream tests and then make improvements.

Also, in this example, because the man-hours used in the upstream process review were comparatively low (less than 5%), the review man-hour rate and productivity had an extremely positive correlation. If the review man-hour rate becomes high, the challenge will be how to conduct reviews efficiently. We have also previously suggested a method to improve review efficiency [5] in which a quality coordinate [7] is initially used to set a start and end standard point of review. Following on form this, we have proposed here a method to continuously improve the review checklist using the statistics technique of a management diagram. As a result of applying this method to an actual organization, we were able to confirm the tolerance of the quality coordinate, the review cost, and a shift in the test man-hours to the upstream process for the entire lifecycle. In addition, the product quality is gradually improving. Improvements were confirmed for each criterion, implying the effectiveness of the proposed method.

## 7 CONCLUSIONS

The characteristics of this model are, firstly, it is able to make a process evaluation using the data collected every day within an organization and, secondly, even a single year of data can be used to evaluate the process, multiple years are not needed. Although this is only analyzing general tendencies, there is much value in using the model because analysis can be done using data that is collected daily, and thus it requires less labor and time than assessments done by outside companies or quality management organizations.

Using our model, we can find the weaknesses of organizations and improve them, and we can connect this to actions to make the strengths even stronger, clarifying the direction for such organizations. For future work, we will apply this method to other projects in order verify whether it has a similar effect.

**REFERENCES**

[1] http://www.sei.cmu.edu/cmmi/index.html (2009/8/20)
[2] http://www.isospice.com/ (2009/8/20)
[3] ISO 9001:2000 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=46486 (2009/8/20)
[4] Rogger S.Pressman, "SOFTWARE ENGINEERING /sixth edition" McGraw-Hill (2005)
[5] A. Hayashi, N. Kataoka, "A Method to Identify Critical Software Process Improvement Area using Quality Function Deployment". Proceeding International Conference on Innovation in Software Engineering ISE, pp1143-1148 (2008)
[6] A. Hayashi, N. Kataoka , "A Method to Identify Critical Software Process Improvement Area using Quality Function Deployment", Journal C of IEEJ , Vol.128, No7 , pp1231-1241 (2008)
[7] Komuro et al. "Peer review technique improvement and quantitative analysis of improvement effect on the basis of the reality of the development work " ,SEC Journal, Vol.1, No.4, PP.6-15, Nov. 4(2005)
[8] Tom Gilb, D. Graham, "Software Inspection", Addison-Wesley Professional (1993)
[9] Peter C. Rigby, Daniel M. German, Margaret-Anne Storey, "Open source software peer review practices: a case study of the apache server", ICSE '08: Proceedings of the 30th international conference on Software engineering, May, pp541-550 (2008)
[10] Dewayne Perry, Adam Porter, Michael Wade, "Reducing inspection interval in large-scale software development", IEEE Transactions on Software Engineering archive, Vol. 28, Issue 7, PP.695-705, July (2002)
[11] KOMIYAMA Toshihiro, "Development of Foundation for Effective and Efficient Software Process Improvement (<Special Features> Study of The Latest Trend of Software Management Technology)", IPSJ Magazine, Vol. 44, No4, Apr., pp341-347 (2003)