Process Fragment Based Process Complexity with Workflow Management Tables

Masaki Obana[1] , Noriko Hanakawa[2] , Norihiro Yoshida[1] , Hajimu Iida[1]
[1]Nara Institute Science and Technology
[2]Hannan University
masaki-o@is.naist.jp, hanakawa@hannan-u.ac.jp, yoshida@is.naist.jp, iida@itc.naist.jp

ABSTRACT
The actual software development processes deviate from initial model planned at early stage. One of the reasons is that additional processes get triggered by urgent changes of software specifications. Such additional processes increase the complexity of the whole development process, and possibly decrease the product quality. In this paper, we propose a novel complexity measure of software process, which based on the number of process fragments, the number of simultaneous process fragments, and the number of developers' groups. Proposed process complexity is applied to two industrial projects in order to show the usefulness of this measure. As a result, we found that the higher value of process complexity indicates higher risk of products' faults.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management: software process models

General Terms
Management, Measurement.

Keywords
Process complexity, risk management, a workflow management table, change of customers' demand.

1. Introduction
In software development projects, large gaps between planned development processes and actual executed development processes exist. Even if a development team has originally selected the waterfall model unplanned small processes are often triggered as shown in the following examples.

i. One activity of waterfall-based process is changed into new iterative process because of urgent specification changes.

ii. Developers design GUI using a new prototype development process at design phase.

iii. In an incremental development process, developers correct defects in the previous release while developers are implementing new functions in current release.

That is, the waterfall-based process at planning time is often gradually transformed into the combination of the original waterfall-based process and several unplanned small processes (hereinafter referred to as process fragments). In consequence, actual development processes are more complicated than planned one (see also Figure 1).

In this paper, we firstly assume that complicated process decreases product quality, and then propose a new metric for process complexity based on the number of unplanned process fragments, the number of simultaneous execution processes, and the number of developers' groups. It can be used to visualize how process complexity increases as actual development process proceeds.

An aim of measuring process complexity is to prevent software from becoming low quality product. A process complexity is derived from a base process and process fragments. A base process means an original process that was planned at the beginning of a project. Process fragments mean additional and piecemeal processes that are added to the base process on the way of a project. Process fragment occurs by urgent changes of customers' requirement, or sudden occurrence of debugging faults. Process fragments can be extracted from actual workflow management tables which are popular in Japanese industrial projects. We especially focus on simultaneous execution of multiple processes to model the process complexity. Simultaneous execution of multiple processes is caused by adding process fragments on the way of a project. Finally, we perform an industrial case study in order to show the usefulness of process complexity. In this case study, we found that process complexity indicated the degree of risk of post-release faults.

Section 2 shows related work about process metrics, and risk management. The process complexity is proposed in section 3. Section 3 also describes how the proposed complexity would be used in industrial projects. Case studies of two projects are shown in section 4. In section 5, we discuss a way of risk management using the process complexity. Summary and future works are described in Section 6.

2. Related Work
Many software development process measurement techniques have been proposed. CMM [1] is a process maturity model by Humphrey. Five maturity levels of organizations have been proposed in CMM. When a maturity level is determined, various values of parameters (faults rate in total test, test density, review density) are collected. In addition, Sakamoto et al. proposed a metrics for measuring process improvement levels [2]. The metrics were applied to a project based on a waterfall process model. Theses process measurement metrics' parameters include the number of times of review execution and the number of faults in the reviewed documents. The aim of these process measurement techniques is improvement of process maturity of an organization, while our research aims to measure process complexity of a project, not organization. Especially, changes of process complexity in a project clearly are presented by our process complexity.

Many researches of process modeling techniques have been ever proposed. Cugola et al. proposed a process modeling language that describes easily additional tasks [3]. Extra tasks are easily added to a normal development process model using the modeling language. Fuggetta et al. proposed investigated problems about software development environments and tools oriented on various process models [4]. These process modeling techniques are useful to simulate process models in order to manage projects. However, these process modeling techniques make no mention of process complexity in a project.

In a field of industrial practices, Rational Unified Process (RUP) has been proposed [5]. The RUP has evolved in integrating several practical development processes. The RUP includes Spiral process model, use-case oriented process model, and risk oriented process model. Moreover, the RUP can correspond to the latest development techniques such as agile software development, and .NET framework development. Although problems of management and scalability exist, the RUP is an efficient integrated process for practical fields. The concept of the various processes integration is similar to our process fragments integration, while the RUP is a pre-planned integration processes. The concept of our process complexity is based on more flexible model considering changes of development processes during a project execution. Our process complexity focuses on changes of an original development process by process fragments, and regarded as a development processes change.

Garcia et al. evaluated maintainability and modifiability of process models using new metrics based on GQM [6]. They focus on additional task as modifiability. The focus of additional task is similar to our concept of process complexity. Although their research target is theoretical process models, and our research target is practical development processes. In this way , there was no studies for measuring complexity of process during a project as long as we examined it. Therefore, the originality of our proposed complexity may be high.

3. Proposed Metrics Based on Process fragment

3.1 Process fragment

In software development, a manager makes a plan of a single development process like a waterfall process model at the beginning of a project. However, the planned single process usually continues to change until the end of the project. For example, at the beginning of a project, a manager makes a plan based on a waterfall process model. However the original process is changed to an incremental process model because several functions' development is shifted to next version's development. Moreover, at the requirement analysis phase, prototyping process may be added to an original process in order to satisfy customers' demands. If multiple releases like an incremental process model exist, developers have to implement new functions while developers correct detects that were caused in the previous version's development. In this paper, we call the original process "a base process", we call the additional process "a process fragment". While original process is a process that was planned at the beginning of a project, a process fragment is a process that is added to the original process on the way of a project. "Fragment" means piecemeal process. Process fragments are simultaneously executed with a base process. Process fragment

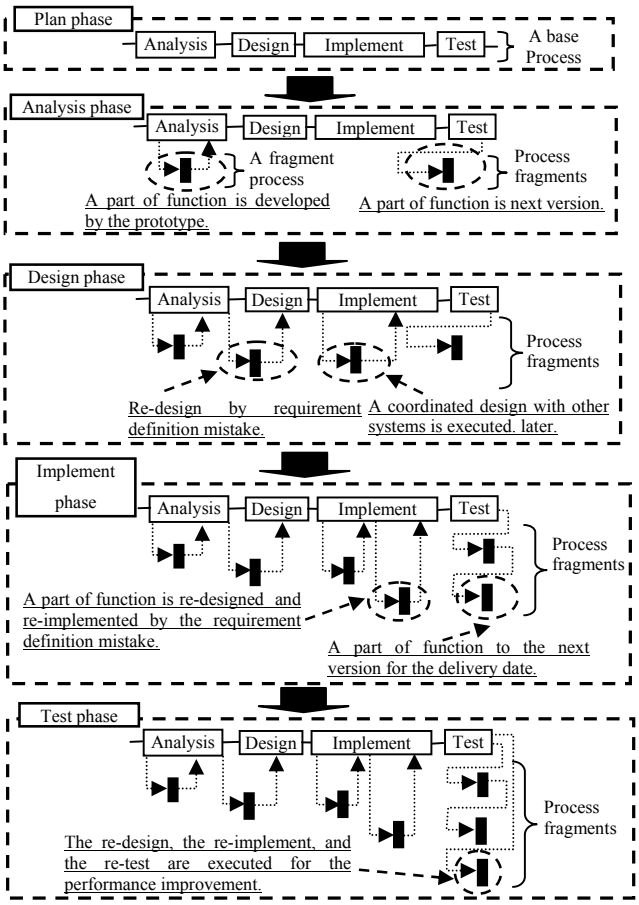

Figure 1 A concept of process fragments

does not mean simple refinement of a base process, but rather may separately executes from a base process execution.

Figure1 shows an example of process fragment. At the planning phase, it is a simple development process that consists of analysis activity, design activity, implementation activity, and testing activity. In many cases, because of insufficient information, a manager often makes a rough simple process (macro process) rather than a detailed process (micro process) [7]. However, information about software increases as a project progresses, and the original simple process changes to more complicated processes. In the case of Figure 1, at the analysis phase, an unplanned prototype process was added to the original process because of customers' requests. As a result of analysis phase, implementations of some functions were shifted to next version development because of constraints of resources such as time, cost, and human. The process fragments were shown at the Figure1 as small black boxes. In the design phase, because customers detected miss-definitions of system specifications that were determined in the previous analysis phase, a process for reworking of requirement analysis was added to the development process. Moreover, the manager shifted the development of a combination function with the outside system when the outside system was completed. During the implementation phase, several reworks of designs occurred. In the test phase, reworks of designs occurred because of low performance of several functions.

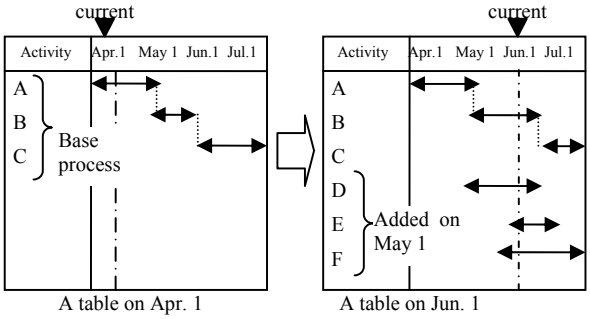

Figure 2 Extracting process fragments from configuration of a workflow management table

Process fragments are caused by urgent customers' requests, design errors, combination with outside systems on the way of development. Various sizes of process fragments exist. A small process fragment includes only an action such as document revision. A large process fragment may include more activities for example, a series of developing activities; design activity, implementation activity, and test activity.

3.2 Calculation of process complexity

3.2.1 Extracting process fragments

Process complexity is calculated based on process fragments. Process fragments are identified from a series of workflow management table. That is a continuator revised along the project. Figure 2 shows two versions of a workflow management table. Each column means a date, each row means an activity. A series of A, B, C activities is a base process. D, E, F activities are added to the base process on May 1. Therefore, D, E, F activities are process fragments. On Jun. 1, the base process and three process fragments are simultaneously executed. In this way, process fragments can be identified from configuration management of a workflow management table. Difference between current version and previous version of a workflow management table means process fragments. Of course, the proposed complexity is available in various development processes such as an agile process as long as managers manage process fragments in various management charts.

3.2.2 Definition of process complexity

Process complexity is defined by the following equation.

$$PC_{(t)} = \sum_{i=1}^{N_{(t)}} (Num\_dev_{(t)i} \times L_{(t)i} \times term_{(t)i}) \quad \text{.....(1)}$$

$PC_{(t)}$: process complexity on time t

$N_{(t)}$: the total number of process on time t

$Num\_dev_{(t)i}$: the number of group of developers of the *i-th* process fragment on time t

$L_{(t)i}$: the number of simultaneous processes of the *i-th* process fragment on time t. But the i-th fragment is eliminated from these multiplications in L(t).

$term_{(t)i}$: ratio of an executing period of the *i-th* process fragment for the whole period of the project on time t, that is, if $term_{(t)i}$ is near 1, a executing period of the process fragment is near the whole period of the project.

Basically, proposed process complexity is the accumulation of all process fragments including finished processes. The reason of the accumulation is that the proposed complexity's target is a whole project, not a spot timing of a project. For example, many process fragments occur at the first half of a project. Even if the process fragments have finished, the fragments' executions may harmfully influence products and process at the latter half of the project. Therefore, the proposed complexity is accumulation of all process fragments.

The process complexity basically depends on three elements; the number of group of the *i-th* process fragment on time *t*: $Num\_dev_{(t)i}$, the number of simultaneous processes of the *i-th* process fragment on time *t*: $L_{(t)i}$, and ratio of an executing period of the *i-th* process fragment for the whole period of project time *t*: $term_{(t)i}$. Granularity of group of developer($Num\_dev_{(t)i}$) depends on the scale of process fragments. If a process fragment is in detail of every hour, a group simply correspond to a person. If process fragment is large such as design phase, a group unit will be an organization such as SE group and programmer group. The group of developers will be carefully discussed in future research. The ratio of an executing period of the *i-th* process fragment for the whole period of project time *t* ($term_{(t)i}$) means impact scale of a process fragment. If a process fragment is very large, for example an executing period of the process fragment is almost same as the whole period of a project, the process fragment will influence largely the project. In contrast, if a process fragment is very small, for example an executing period is only one day, the process fragment will not influence a project so much.

In short, when more and larger scale process fragments are simultaneously executed, a value of process complexity becomes larger. When fewer and smaller scale process fragments simultaneously are executed, a value of process complexity becomes smaller. Values of the parameters of equation (1) are easily extracted from configuration management data of a workflow management table.

3.3 Setting a base process and extracting process fragments

At the beginning of a project, a base process is determined. If a planned schedule is based on a typical waterfall process model such as the base process in Figure 1, the parameters' values of process complexity are $t=0$, $N_{(0)}=1$, $Num\_dev_{(0)}=3$ (SE group developer group, customer group), $L_{(0)}=1$, *and* $term_{(0)}=1$. Therefore the value of process complexity $PC_{(0)}=3$.

As a project progresses, unplanned process fragments are occasionally added to the base process at time *t1*. A manager registers the process fragments as new activities to the workflow management table. The manager also assigns developers to the new activities. Here, we assume that the planned period of a base process is 180 days. A manager adds two activities to the workflow management table. The period of each additional activity is planned as 10 days. Therefore the total number of process $N_{(t1)} = 3$, and the process complexity is calculated as follows;

(1) for $i = 1$ (a base process)
- $Num\_dev_{(t1)} = 3$
- $L_{(t1)1} = 3$
- $term_{(t1)1} = 180/180 = 1.0$

(2) for $i = 2$ (the first process fragment)
- $Num\_dev_{(t1)2} = 1$
- $L_{(t1)2} = 3$
- $term_{(t1)2} = 10/180 = 0.056$

(3) for $i = 3$ (the second process fragment)
- $Num\_dev_{(t1)3} = 1$
- $L_{(t1)3} = 3$
- $term_{(t1)3} = 10/180 = 0.056$

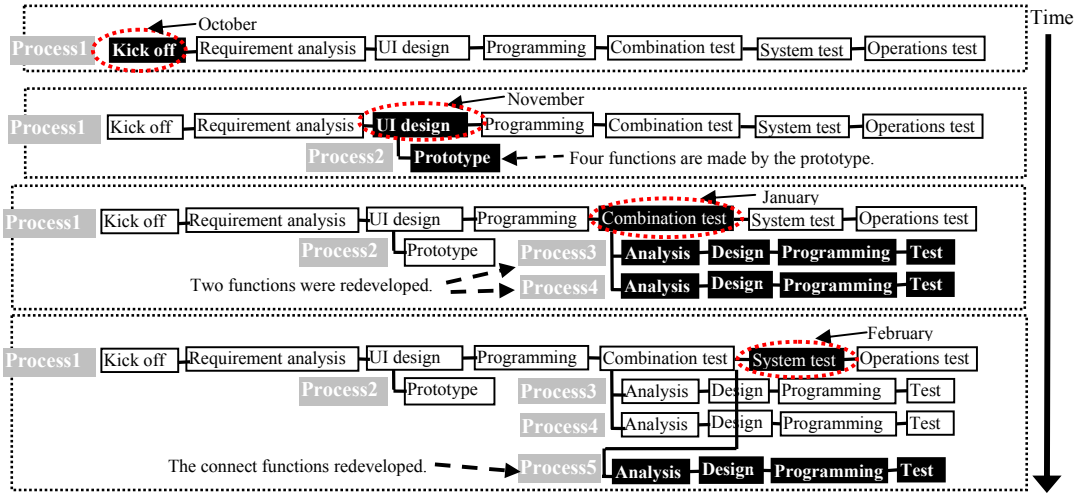

Figure 3 A variation of development process of the HInT project

Finally, a value of process complexity at $t=t1$ can be calculated as $PC_{(t1)} = 9.000 + 0.168 + 0.168 = 9.336$.

In this way, the value of $PC_{(t)}$ at any value of time t can be calculated based on workflow management table.

4. Application to Two Industrial Projects

The process complexity has been applied to two practical projects.

4.1 The HInT project

The first project's name is HInT (Hannan Internet communication Tool). The HInT project developed a web-based educational portal system. The development began from October 2007, the release of the HInT was April 2008. Because the workflow management table was updated every week, 20 versions of the workflow management table are obtained. At the beginning of the project, the number of activities in the workflow table was 20. At the end of the project, the number of activities reached to 123. Each activity had a planned schedule and a practice result of execution. Figure 3 shows a rough variation of development process of the project. At the beginning of the project, the shape of the development process was completely a waterfall process. However, at the UI design phase, a prototype process was added to the waterfall process. In the prototype process, four trial versions were presented to customers. At the combination test phase, developers and customers found significant errors of specifications and two process fragments were added to the development process in haste. Reworks such as re-design, re-implement, and re-test for the specification errors continued until the operation test phase. At the system test phase, an error of a function connecting with an outside system occurred and a new process fragment was introduced to the development process. The introduced process fragment consists of various activities such as investigating network environments, investigating specification of the outside system, and revising the programs. These activities continued until the delivery timing.

Figure 4 shows process complexity of the HInT project. At the beginning of the project, the value of process complexity was very low while the value of the process complexity increased as the project progresses. In particular, after the system test phase, four processes were simultaneously executed. Then process complexity increased to 44.167.

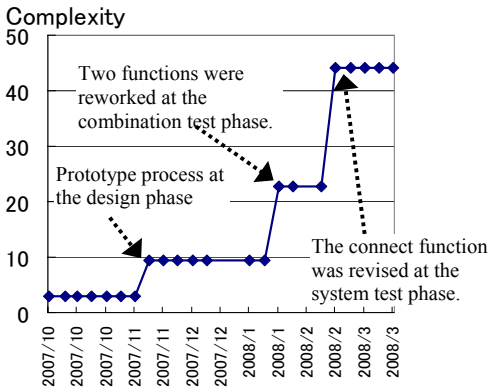

Figure 4 Process complexity of the HInT project

4.2 The p-HInT project

The p-HInT is a lecture support system for large-scale lectures with mobile terminals [8][9]. A base process of the p-HInT project was an incremental development process. We released the p-HInT product four times from April 2008 through April 2010. A development process of the each release included analysis phase, design phase, implement phase, and test phase. At the first version development, basic functions and infrastructure implementation were planned in detail. However, the manager determined only rough design of next versions' functions. At the second version development, new four functions were developed while the developers fixed defects that had been introduced at the previous version's development. Of course, the debugging works were not planned in the original schedule. At the third version development, product refactoring was executed because the design of the product became too complicated for introduction of new functions and also for debugging works. At the fourth version development, customers requested new functions that were not discussed at the requirement analysis phase. The customers said that the requested new functions were more important than the functions that were originally planned at the beginning of the project. At same time, performance improvement work for some functions in the previous releases was made at the fourth version development.

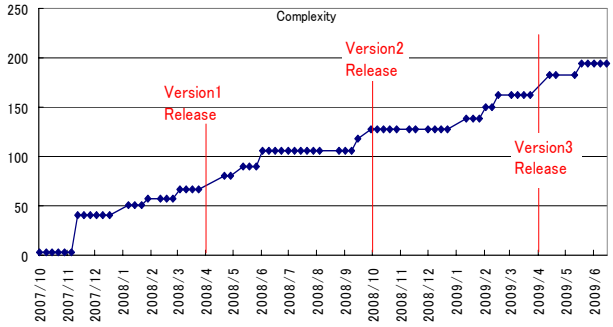

Figure 5 Process complexity of the p-HInT project

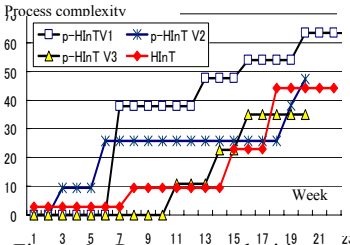

Figure 6 Process complexities of the four project

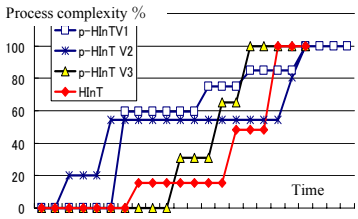

Figure 7 Patterns of growth of process complexity

The process complexity of the p-HInT project was measured until June 2009. The number of items listed in the workflow table was 31. The items were divided into activities that were executable on each week. Figure 5 shows a result of the measurement of process complexity.

At the begging of the project, the value of process complexity was low (3.0) because the original development process is a simple incremental process. At November 2007, because the original process was divided into each process for each function, the value of the process complexity increases to about 45.0.

At the release time of the first version, the value was 63.48. In the second version development, processes of improving performance for the released functions were added to the original process. That is, developers had to revise the released functions while the developers had to implement new functions as originally planned at the beginning of the project. The value of the process complexity increased to 127.56.

At the test phase of the second version, some specification errors came to light. For fixing the specification errors, several process fragments were added and the value of process complexity increased again.

On the first half of the third version development, the value of process complexity did not increase because of a product refactoring activity. Developers concentrated to the product refactoring work. However, on the latter half of the third version, problems such as low performance of the released functions and miss data connection with outside systems occurred. Because process fragments for fixing the problems were added, the value of process complexity increased to 182.26.

5. Discussion

We discuss usefulness of process complexity. Process complexity has a role of capturing dynamic processes as a project progresses. If the values of process complexity for the beginning of a project and the end of a project don't differ so much, we can interpret that the project was a stable because the project may have been executed smoothly without large changes. In contrast, if a value of process complexity largely increased, we can interpret that the project was unstable due to large changes. Figure 6 shows changes of the values of process complexity of the two projects. We regard each version of the p-HInT as one project. The fourth version development of the p-HInT is excluded because all data extracting the workflow management table was not prepared.

It became clear change of process complexity of each project in Figure 6. Projects with large increase of process complexity are version 1 and version 2 of the p-HInT project, and the HInT project. Each increase value is 63.48, 47.28, and 41.17. In contrast, in the version 3 of the p-HInT, the value of process complexity was 35.10.

In addition, each project has a feature of a process complexity growth pattern. Figure 7 shows the patterns of the projects. A maximum value of process complexity of each project is set to 100%. The value of process complexity of the version 1 of the p-HInT increased suddenly on the first half of the project. In version 2 of the p-HInT, the value of complexity largely increased at two timings; at the first half of the project, and at the end of the project. In addition, in version 3 of the p-HInT, the value of complexity gradually increased. The value of complexity of the HInT project increased at the latter half of the project. We call these patterns as "growth patter on early stage type", "growth patter on early and late stage type, "growth patter smoothly type", and "growth patter on late stage type". The "growth patter on early stage type" often occurs when a concept of development and development methods are changed on the first half of a project. In the version 2 of the p-HInT, debugging activities for the previous version's faults were set to highest priority activities on the first half of the project. The process of the version 2 largely changed. Moreover, the unreasonable executions of the debugging activities caused occurrences of new faults on the latter half of the project. Therefore, addition of the debugging activities for the new faults caused the increase of the value of process complexity on the latter half of the project. On the other hand, in the HInT project, developers presented the product demonstration to customers at the system test phase. As a result, several specification errors and customers' demand changes were clarified. Therefore, many activities such as re-design, re-implement, re-test were added to the development process. The process fragments such as the re-work activities caused the increase of process complexity on the latter half of the project.

Therefore, we propose a way of evaluating project risk using the changes of process complexity. The project risk is as follows;

$$Risk = Rank \times Variation \qquad \text{.....(2)}$$

Risk: Project Risk
Rank: growth patter patterns of process complexity:
"growth patter smoothly type": 1,
"growth patter on early stage type" : 2,
"growth patter on late stage type" : 3,
"growth patter on early and late stage type" : 4.
Variation: gap of values of process complexity between at the beginning of a project and at the end of a project.

If a value of *Risk* is large, we can judge the project high risk. *Rank* shows growth patter patterns of process complexity. The pattern of "growth patter smoothly type" is most stable. The pattern of "growth patter on early stage type" is relatively stable. The reason is that developers can cope with the process changes on the first half of a project. Because the changes occur on the early stage,

developers have time for arranging their works. In contrast, changes on the latter half of a project such as "growth patter on late stage type" have high risk. Because developers have little time for arranging their works, developers are confused in simultaneous executions of processes for the changes and an original process. The pattern of "growth patter on early and late stage type" is worst. Because the additional works on the first half of a project influence the works on the latter half of a project. Managers may add the additional works in unplanned on the first half of a project. In addition, *Variation* means a gap value of process complexity between at the beginning of a project and at the end of a project. Of course, a small value of *Variation* is better. In this way, by calculating a value of *Risk*, we evaluate whether a project is complicated or not. A detailed way of determining a value of *Rank* will be discussed in future.

Figure 8 shows results of project risks of the four projects. The project risk of the version 2 of the p-HInT is highest. The version 3 of the p-HInT has lowest risk. Figure 9 shows specific gravity of failures occurred after release. The specific gravity failure is calculated by the number of failure and importance level of failure. The importance level means strength of impact on a product. As Table 1, we classified the failures into five important levels; SS, S, A, B, C. SS means most strong impact like system-down. C means weakest impact like GUI's improvement. The specific gravity of the failures is a value that multiplied the number of failures and the importance level. In the calculation, we set up that a constant of SS is 5, a constant of S is 4, a constant of A is 3, a constant of B is 2, and a constant of C is 1. For example, a value of specific gravity of the p-HInT version 1 is calculated by "5*2 + 4*3 + 3*19 +2*1 + 1*1". The value is 82. In the same way, the specific gravity of failure of the p-HInT version 2 is 88, one of p-HInT version 3 is 37, and one of HInT is 156.

Comparing Figure 8 with Figure 9, Spearman rank correlation coefficient is 0.4. Therefore, we confirmed a weak relationship between the values of project risk and the value of specific gravity of failures. The version 3 of the p-HInT is not only lowest risk but also lowest specific gravity of failure. In addition, if a value of the project risk is high, the specific gravity of failure will be high. That is, possibility of occurrence of strong impact failures becomes high as a value of project risk is more. In the HInT project, because customers suddenly requested new functions, many process fragments were added to the process on the latter half of the project. A value of the project risk increased at the latter half of the project. As a result, many significant failures with SS rank occurred after release.

In this way, by the project risk we can predict possibility of failure occurrences after release. A most useful feature of the project risk is early prediction before start of executions of processes. The prediction is possible when a workflow management table is revised by adding process fragments. The possibility of the early prediction in the project risk is most different from product metrics. Product metrics is calculated by the already complicated product. However, the project risk can prevent a project from becoming high risk. It is useful for managers to judge whether new process fragments are added.

Table 1 Failures after releases of the four projects

	SS	S	A	B	C
p-HInT Version1	2	3	19	1	1
p-HInT Version2	10	6	4	1	0
p-HInT Version3	3	3	2	1	2
HInT	14	12	12	1	0

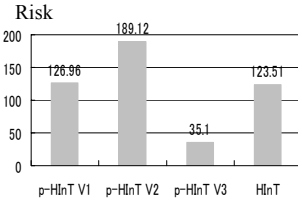

Figure 8 Project risks of the four projects

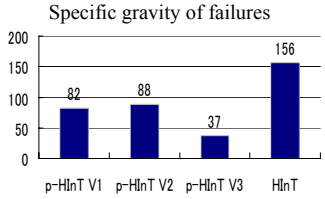

Figure 9 Specific gravity of failures of the four projects

6. Summary

We proposed process complexity model for risk management in software development. The process complexity is calculated by process fragments derived from a workflow management table. In addition, project risk metric has been also proposed. The project risk is calculated by changes of process complexity during a project. We applied the process complexity and the project risk metrics to two industrial projects. As a result, the project risk values have a weak relation with specific gravity of failures. Therefore, managers can predict the quality of released product when the workflow management table is revised.

In future, we will apply the process complexity and the project risk metrics to many industrial projects. We will clarify the relationships between project risk and specific gravity of failures. Then, we will determine value of thresholds of the project risk.

7. REFERENCES

[1] Humphrey Watts S.1989. *Managing the software process.* Addison-Wesley Professional, USA.

[2] Sakamoto, K., Tanaka, T., Kusumoto, S., Matsumoto, K., and Kikuno T. 2000. *An Improvement of Software Process Based on Benefit Estimation.* IEICE Trans. Inf. & Syst.(in Japanese), J83-D-I(7), pp.740-748.

[3] Cugola, G. 1998. *Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models*, IEEE Transaction of Software Engineering, Vol. 24, No. 11, pp.982-1001.

[4] Fuggetta, A. and Ghezzi, C. 1994. *State of the art and open issues in process-centered software engineering environments*, Journal of Systems and Software, Vol. 26, No. 1, pp.53-60.

[5] Kruchten, R. 2000. *The Rational Unified Process,* Addison-Wesley Professional, USA.

[6] Garcia, F., Ruiz, F., Piattini, M. 2004. *Definition and empirical validation of metrics for software process models,* Proceedings of the 5th International Conference Product Focused Software Process Improvement (PROFES'2004), pp.146-158.

[7] Obana, M., Hanakawa, N., Iida, H. 2010. Process complexity metrics based on fragment process on workflow management tables, Proceeding of the Software Engineering Symposium SES2010 (in Japanese), pp.89-96.

[8] Hanakawa, N., Yamamoto, G., Tashiro, K., Tagami, H., Hamada, S. 2008. *p-HInT: Interactive Educational environment for improving large-scale lecture with mobile game terminals*, Proceedings of the16th International Conference on Computers in Education (ICCE'2008), pp.629-634.

[9] Hanakawa, N., Obana, M. 2010. *Mobile game terminal based interactive education environment for large-scale lectures*, Proceeding of the Eighth IASTED International Conference on Web-based Education (WBE2010)