A Software Process Management System Considering Workers' Workload

Atsuo HAZEYAMA

NEC Corporation
NEC Igarashi BLDG.
2-11-5 Shibaura Minato-ku
Tokyo 108 Japan

Seiichi KOMIYA

Information-technology Promotion Agency(IPA)
Shuwa-Shibakoen 3-chome BLDG.
3-1-38 Shibakoen Minato-ku
Tokyo 105 Japan

Abstract

Workers in software projects are usually engaged in plural works, not only the main development works but also various other works, concurrently. Such other works might put pressure on the schedule of the whole project. Therefore in order to manage the whole project appropriately, not only the main development works but also various other works should be dealt with as management objects. This paper clarifies a framework to support various type of works and different granularity of processes in an integrated manner and shows the behavior of the system using an example. This framework enables appropriate process management of the whole project by taking into consideration workers' workload (who is doing what activities at how much workload at what time).

1 Introduction

With advances in the informatization of society, the need for software development has been increasing. Furthermore, along with the trends toward increasing the scale and complexity in the software developed, it has become difficult to complete the development of such software in accordance with the required quality, the deadline, and within budgetary limits.

Under these circumstances, research and development of software processes have been actively undertaken (for example [1], [2], [3], [4], [5]). In spite of the keynote speech of Professor Dr. K. Ochimizu in the 1st Japanese Software Process Symposium [5] that 'Software process is a tool for software project management', there is little literature with respect to management planning and control in software process research[8]. Some of the literature is described below:

Dr. M. Penedo and Dr. D. Stuckle proposed a PMDB (Project Master DataBase) model which aimed at constructing an integrated software engineering database (including both development support and management support) [10]. PMDB represents objects and their relationships in software engineering environments (SEEs) with ER (Entity-Relationship)-model. PMDB is a generic model for SEEs, however, it does not provide various aspects in SEEs.

Dr. L. Liu and Dr. E. Horowitz proposed DesignNet model [7]. DesignNet represented the WBS (Work Breakdown Structure) [9] using AND/OR operators and the dependency/concurrency relationships in software projects using the PetriNet model. DesignNet included some important characteristics of software projects, however, from their papers they focus on the only activities of traditional WBS level. They do not deal with one of the most important characteristics in software process that a worker might be engaged in several activities concurrently.

Dr. M. Kellner did quantitative simulation (considering resource constraints) for management activities of planning and control using STATEMATE [8]. This simulation supports only activities of WBS level. As Kellner stated, STATEMATE can not reflect actual resources' information specified in the organizational perspective on the simulation directly.

In the field of IE (Industrial Engineering), PERT (Program Evaluation and Review Technique) is used for process management technique. This is a method to point out critical paths in the whole process by the dependency relationships of activities. However, it does not consider the constraints on resources which are assigned to the activities.

Although the above literature enables us to manage the whole project from the viewpoint of activities, they do not support to grasp workers' workload. However as we pointed out in [6], it is also very important to monitor workers' workload in software project management, because workers are usually engaged in several activities concurrently in most software development projects.

And management objects in traditional project management is the main development work like design, coding, test, and so on, which take several weeks or months. However, we have experiences in real projects that software developers perform not only the main development works but also other works such as users' support, business trips, training, machine setup, reports writing, and so on. Furthermore they participate in fine-grained processes which do not appear in the schedule table, such as problem resolution processes, document authorization processes, etc. Such activities might put pressure on the schedule of the main development works. In project management of large software projects, a project manager can not usually grasp all of the activities and each person's schedule in detail. But such information is necessary for project management by all means. And such information is also useful for each worker because he/she might forget his/her assigned works. Therefore a software process management system should deal with not only the main development works but also other works and/or more fine-grained processes which do not appear in the schedule table. In this paper, based on the above experiences, we propose a framework of a software process management system to support management of various type of works and/or different granularity of works in an integrated manner.

This paper is organized as follows: Section 2 categorizes the works in relation to software processes. Section 3 proposes a support model for a software process management system. Section 4 picks up one case study and shows the behavior using an example. Finally we summarize this paper.

2 Categorization of Works to Be Managed in Software Projects

In this section, we categorize works to be managed in software projects.

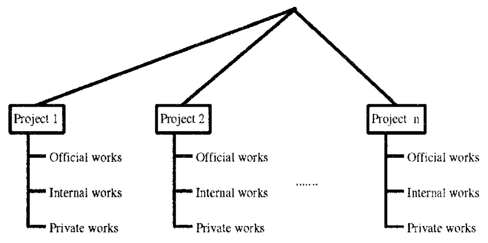


Figure 1: Categorization of works to be managed in software projects

The following three type of works are performed in software projects (Figure 1):

(1) Official works

These are works whose schedules are to be open outside the concerned project.

(2) Internal works

These are works which are performed according to the official works. Not only the main development works but also other accompanying works such as users' support, training, business trips, machine setup, etc. should be included in this category.

(3) Private works

These are works which workers perform for themselves privately in order to perform their assigned works more smoothly.

(1) and (2) are objects of project management but (3) is beyond the scope of project management.

We regard works respectively as coarse-grained processes or fine-grained processes depending on whether the unit of process management is longer or shorter than a management cycle. Unless the progress of a software project is critical, we usually adopt one week as a unit of management cycle. If the progress of a software project is critical, we usually adopt one day as a unit of management cycle.

The schedule table for official works is described in terms of coarse-grained processes. Therefore, the processes of official works are divided into more fine-grained processes in order to manage a software project more appropriately. Therefore the schedule table for internal works is required.

3 A Support Model

This section proposes a model for managing several type of works categorized in the previous section.

3.1 A Process Model for Project Management

This section describes a (coarse-grained) process model for software project management.

In general, software development projects are performed as follows: a project consists of a lot of activities which are performed in order to develop the final target system, and during this process a number of intermediate products are created by project members using tools and machines. Project members are usually engaged in several activities playing various organizational roles. We represent these situations as a process model using a representation like SADT [11] (Figure 2).

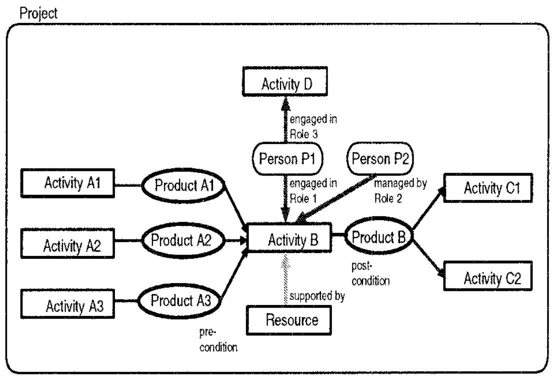


Figure 2: A process model for software project management

We consider five major classes of objects for software project management. These are **Activity**, **Product**, **Resource**, **Role**, and **Project**. Each class of object can, however, be further subdivided. An actual project consists of the instances of these objects.

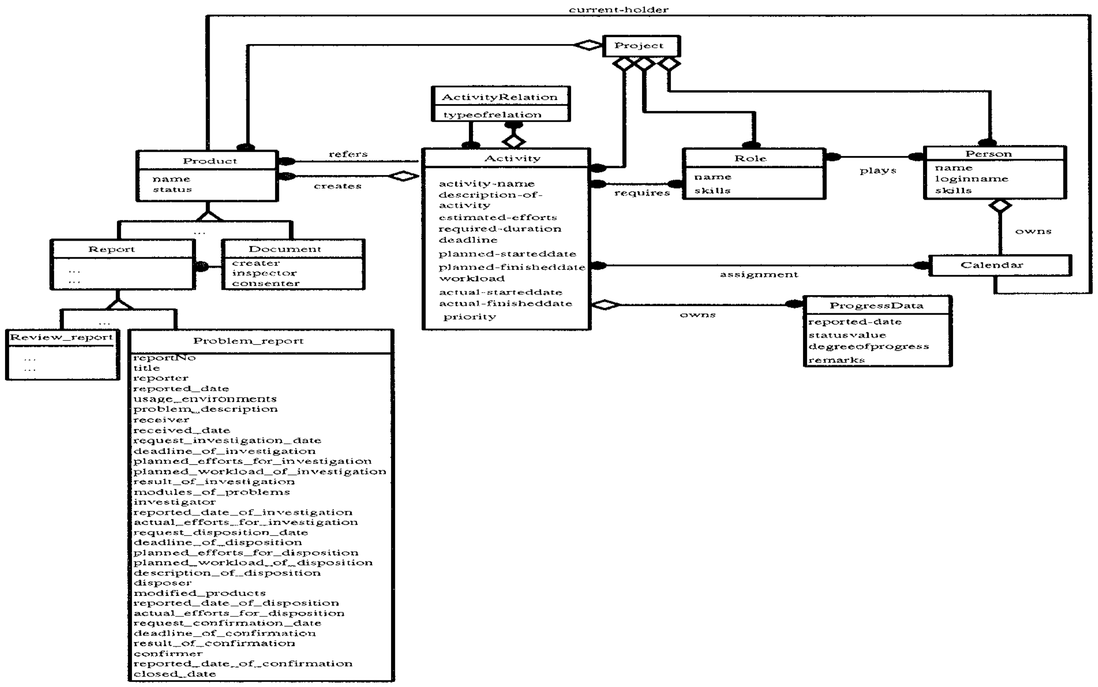


Figure 3: Object model for integrated process management

These objects are mutually related by means of various relationships. These are represented by the objects Activity, Product, Person (subclass of Resource), Role, Project and their relationships in Figure 3.

3.2 An Object Model for Integrated Process Management

In this section, we propose a model for managing various type of works and/or different granularity of works in an integrated manner.

We manage coarse-grained official works and internal works as instances of class 'Activity'. This type of work can be managed by the framework mentioned above. On the other hand, we manage fine-grained cooperative internal works in which several workers play different roles as state transitions of Product objects.

Therefore, we introduce an attribute 'status' into each product object and represent state transitions. We also introduce the relationship 'current-holder' between a product object and a person object so that we can grasp who holds the product object at each state. Furthermore we introduce the attributes of 'efforts estimation' and 'work assignment period' to grasp the workload of each process step of a fine-grained process and the workload of each worker assigned to each process step.

Figure 3 shows an object model for integrated process management. We used the object model of the OMT method for the notation [12].

The workload of each worker is calculated by both the relationship between Calendar and Activity and the relationship between Calendar and Product. Calendar object manages person's schedule information.

4 A Case Study

In this section, we show how the integrated process management is done using an example. We pick up problem resolution process in a system test phase as an example of fine-grained processes.

4.1 Problem Resolution Process

Problem resolution process is the process to detect software faults (i.e., bugs) when a software failure occurred in a software system, then to remove them, and to restore the system to normal state. In this process several persons who play various roles participate in. They are detectors of problems, the QA manager who receives problem reports, requests investigation/disposition and manages the progress, persons who investigate problems, persons who dispose problems, and persons who confirm the resolution of problems, and so on. Each person does his/her assigned work, fills the result in the corresponding problem reports, and passes them to the next person (Figure 4).

In the problem resolution process in a system test phase, information on the assignment of workers to each activity and on the results which workers have performed are usually recorded in the form of a problem report. Therefore we regard a problem report as a subclass of a product. And we treat items which should be described within the problem resolution process as attributes of this object. A problem report has attributes which identify each object, for example, a problem identification number and a title, and attributes which hold descriptions of the problem/result and schedule/assignment information. Furthermore each problem report has relationships with workers who participate in the process steps of its resolution process and with a product for which the problem report is created.

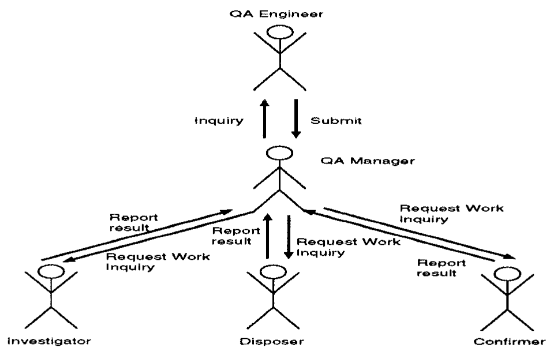


Figure 4: Problem resolution process

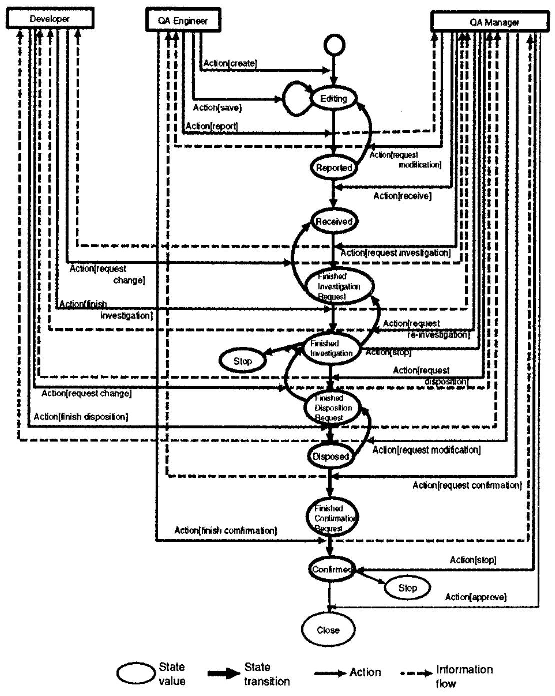

Developer
QA Engineer
QA Manager
Action[create]
Editing
Action[save]
Action[report]
Action[request modification]
Reported
Action[receive]
Received
Action[request investigation]
Action[request change]
Finished Investigation Request
Action[finish investigation]
Action[request re-investigation]
Finished Investigation
Stop
Action[stop]
Action[request disposition]
Action[request change]
Finished Disposition Request
Action[finish disposition]
Action[request modification]
Disposed
Action[request confirmation]
Finished Confirmation Request
Action[finish comfirmation]
Action[stop]
Confirmed
Stop
Action[approve]
Close
State value
State transition
Action
Information flow

Figure 5: State transition of a problem report

A problem report also has states as an attribute (we call this attribute status). It represents the progress situation from the creation of a problem report until the completion of the disposition as a finite state machine model (Figure 5). By describing to whom a problem report is sent when an action which changes the status value is invoked as a finite state machine model, work induction is achieved.

The progress of each problem report can be grasped by its status value and the whole progress of system test can be grasped by the trend of the number of problem reports.

We show some meaning of this finite state machine model: for example, a problem report which has already been received by the QA manager is notified of a selected person as a work instruction, after the manager takes an action of 'request investigation work' of the problem to an appropriate person. At this time, work instructions from the QA manager to workers should be issued taking into account the workers' *workload*. Workload means the load of work of each worker. We define workload as follows: it divides estimated efforts into assigned efforts. For example, in case that an activity which is estimated to take 3 days will be planned for 5 days, the workload of the assigned worker becomes 60%. To calculate the workload, in issuing work instructions, information with respect to deadline and required efforts is needed. Based on this information, a support system checks the validation of each work instruction. In addition, a support system also checks the total workload of each worker because it might happen to become over-workload by assigning several activities to one worker, even though each work instruction is valid. To check this, a support system must be able to refer the schedules of each worker. This schedule information is held in the process model (Figure 3).

4.2 An Example

In this section, we show the behavior of the integrated process management.

We consider a project where a system test of three modules α, β, γ and manual writing of the three modules are performed in parallel. Module α, β, γ are developed by worker A, B, C, respectively. We assume worker D and E are engaged in the system test, who are quality assurance engineers and do not participate in the development of the modules. During the system test, the workers who developed the modules are supposed to be engaged in manual writing of the corresponding module. Such situations can be represented as a process model shown in Figure 6. This is a figure which represents the subset process of the overall development project. Figure 7 shows the assignment of activities to workers. This figure, for example, shows the activity 'manual writing of module α' has been assigned to worker A for 7 days during March 7 through March 15. As this activity is estimated to take 5 days,

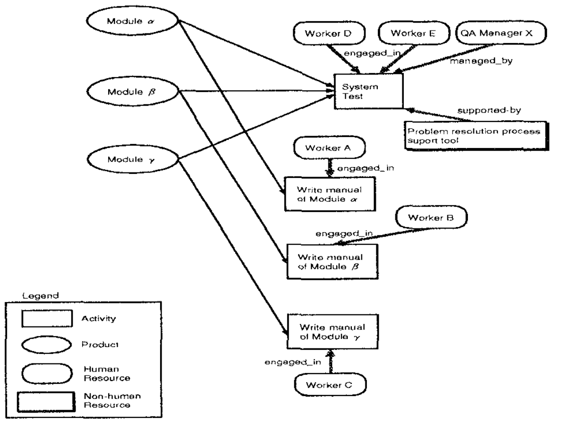


Figure 6: A process model of the example project

worker A is assigned to this activity at the workload of 71% during this period.

We assume this planning is made up on 7 March based on the deadline and estimated efforts information as shown in Figure 7, and has been already notified to workers. Let us consider a situation where a problem occurs after the system test started. A problem report which was created by a quality assurance engineer, is submitted to the QA manager. The QA manager assigns the investigation and/or disposition work to appropriate persons. For example, we will consider a case that a problem (its issued ID is No.1) occurs on 8 March and that the QA manager assigns the investigation work to Worker A. We assume that the QA manager estimates the investigation work to take a half day and sets the deadline of the reply to 9 March. The workload of this assignment is 25% per day because the worker does a half day's work in two days (during 8 March through 9 March). Therefore the total workload of Worker A (including this investigation work) is shown in Figure 8.

We will consider another case that a problem (its issued ID is No.2) occurs on 8 March and that the QA manager assigns the investigation work to Worker C. We also assume that the QA manager estimates this investigation work to take a half day and sets the deadline of the reply to 9 March. However, Worker C is scheduled to participate in a seminar during 9-11 March. Therefore the QA manager should be notified that this assignment (to set the deadline to 9 March) is invalid. Thereafter we assume the QA manager sets the deadline to 8 March (to resolve the problem before the worker goes to a seminar). However, at this time, worker C must perform manual-writing work of module γ at the workload of 100% on 8 March (shown in Figure 9). In this situation, if the problem investigation work is assigned to worker C, his/her workload on 8 March becomes 150%. Such a situation is over-assigned. A system notifies the QA manager of such

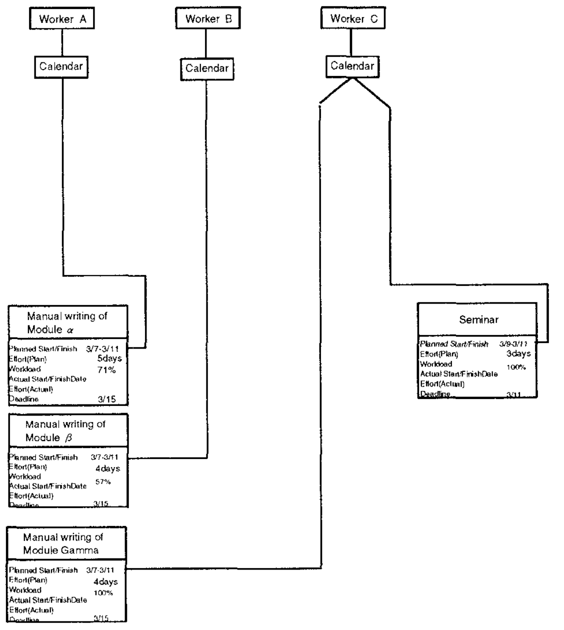


Figure 7: Instances of this example (planning before system test)

over-assignment. Based on this information, the QA manager decides whether he or she continues to assign the worker to the work or changes the assignment.

Once work (investigation, disposition, confirmation) is assigned to a worker by the QA manager, a message which means request of work is displayed on the assigned worker's user interface. Figure 10 is an example user interface for each worker. If worker A pushes the 'Problem List' button on his/her own user interface, problem reports which are assigned to him or her are displayed in the form of Figure 11. This figure shows the investigation work of a problem is assigned to worker A. When a user selects each item in this list, the corresponding problem report is shown (Figure 12). After finishing his/her assigned work, the worker returns the problem report to the QA manager, that is, when the worker fills in the result of his/her work and takes a 'Finish investigation' action (Figure 12), then the item is erased from the list of worker A (Figure 13), and a message that the investigation work has finished is notified to the QA manager (Figure 14).

5 Conclusion

In this paper we described that in software process management not only the activities but also the workload of workers should be managed because workers of software projects are usually assigned to plural activities concurrently and this might cause delay the whole project. In order to manage processes more accurately, we investigated works performed in

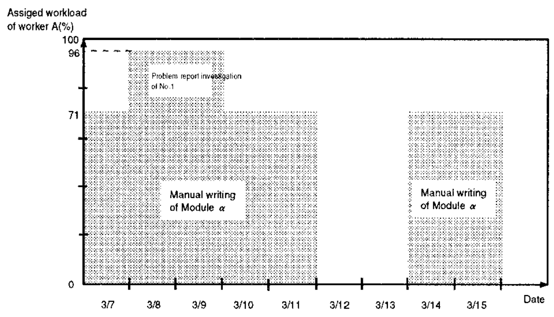


Figure 8: Workload of Worker A

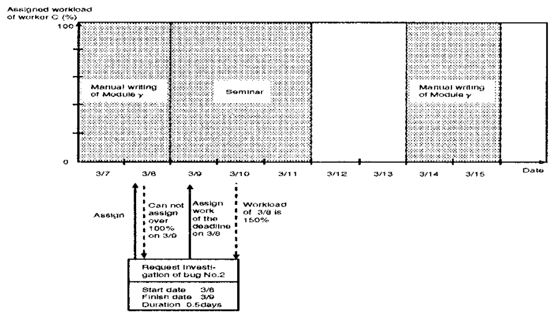
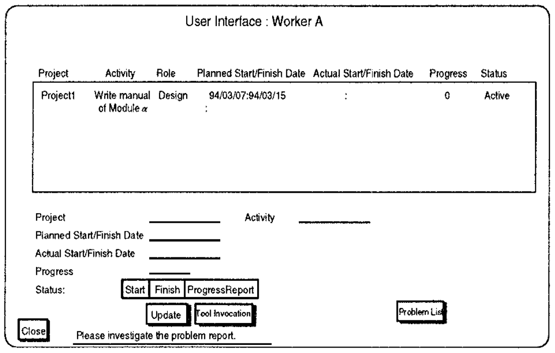


Figure 9: Workload of Worker C



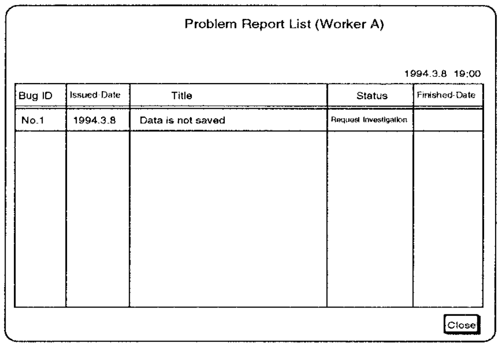
User Interface : Worker A

Project	Activity	Role	Planned Start/Finish Date	Actual Start/Finish Date	Progress	Status
Project1	Write manual of Module α	Design	94/03/07:94/03/15 :	:	0	Active

Project ______ Activity ______
Planned Start/Finish Date ______
Actual Start/Finish Date ______
Progress ______

Please investigate the problem report.

Figure 10: Request work to Worker A



Problem Report List (Worker A)

1994.3.8 19:00

Bug ID	Issued Date	Title	Status	Finished Date
No.1	1994.3.8	Data is not saved	Request Investigation	

Figure 11: Problem reports assigned to Worker A

software projects and categorized them. To manage various type of works and/or different granularity of works in an integrated manner, we proposed a support model. The base of the integrated process management is a process model we have already proposed for coarse-grained process management. Fine-grained processes are represented in product objects of the process model. Each product object having 'current_holder' relationship with resource object and attributes for assignment information, the workload of workers for fine-grained processes can be also calculated. By integrating all level of works, progress data that project managers want to know are calculated by accumulating the progress data of the lowest works. Our framework in this paper is effective in process management and resource management (who is doing what activities at how much workload at what time).

Acknowledgement

The authors would like to thank the anonymous referees for comments that contributed to improvements of this paper. And first author of this paper would like to thank Mr. H. Oka and Mr. G. Yamazaki for encouraging this work.

References

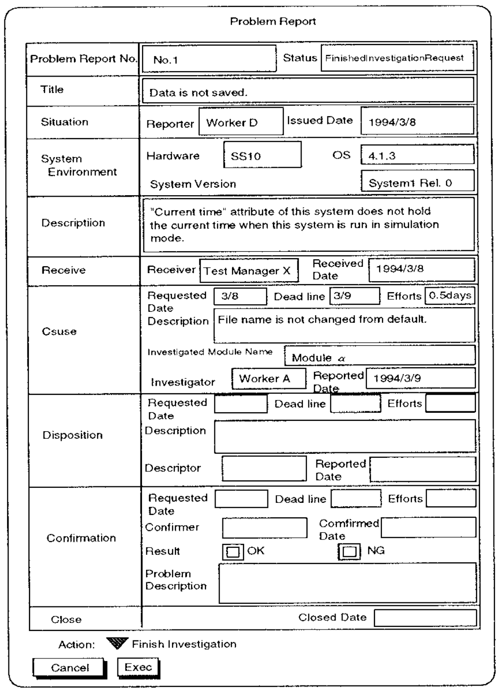
[1] Shaefer W. (ed.), Proceedings of the 4th European Workshop on the Software Process Technology, Lecture Notes in Computer Science 913, Springer-Verlag, Noordwijkerhout, the Netherland, 3-5 April 1995.

[2] Notkin D. (ed.), Proceedings of the 17th International Conference on Software Engineering, IEEE Computer Society Press, Seattle, Washington, USA, 23-30 April 1995.

[3] Perry D. (ed.), Proceedings of the Third International Conference on the Software Process -Applying the Software Process-, IEEE Computer Society Press, Reston, Virginia, USA, 10-11 October 1994.

[4] Ghezzi C. (ed.), Proceedings of the 9th International Software Process Workshop –, IEEE Computer Society Press, Airlie, Virginia, USA, 5-7 October 1994.

[5] Inoue K. (ed.), Proceedings of the 1st Japanese Software Process Symposium, Information Processing Society of Japan, Tokyo, Japan, 26-27 May 1994.


Problem Report

Problem Report No.	No.1 Status FinishedInvestigationRequest
Title	Data is not saved.
Situation	Reporter Worker D Issued Date 1994/3/8
System Environment	Hardware SS10 OS 4.1.3 System Version System1 Rel. 0
Descriptiion	"Current time" attribute of this system does not hold the current time when this system is run in simulation mode.
Receive	Receiver Test Manager X Received Date 1994/3/8
Csuse	Requested Date 3/8 Dead line 3/9 Efforts 0.5days Description File name is not changed from default. Investigated Module Name Module α Investigator Worker A Reported Date 1994/3/9
Disposition	Requested Date Dead line Efforts Description Descriptor Reported Date
Confirmation	Requested Date Dead line Efforts Confirmer Comfirmed Date Result OK NG Problem Description
Close	Closed Date

Action: Finish Investigation

Cancel Exec

Figure 12: Finish to fill in the result of investigation of a problem report

[6] Hazeyama A., and Komiya S., A Process Model for Software Process Management, Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE'92), pp.582-589, IEEE Computer Society Press, Capri, Italy, June 15-20, 1992.

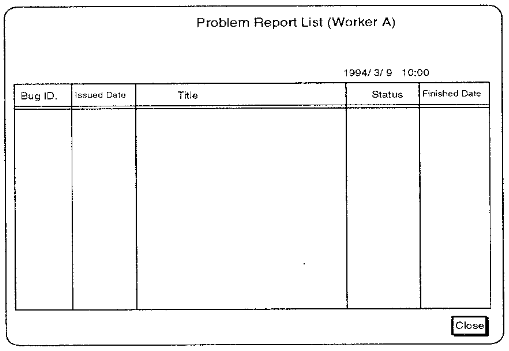
[7] Liu L. and Horowitz E., Object Database Support for a Software Project Management Environment, ACM SIGSOFT Software Engineering Notes, Vol.13 No.5, November 1988, pp.85-96.

[8] Kellner M. I., Software Process Modeling Support for Management Planning and Control, Proceedings of the 1st International Conference on Software Process, pp.8-28.

[9] Ould M. A., Strategy for Software Engineering, John Wiley & Sons, Ltd., 1990.

[10] Penedo M. H. and Stuckle D., PMDB - A Project Master Database for Software Engineering Environments, Proceedings of the 8th International Conference on Software Engineering, 1985, pp.150-157.

[11] Ross D.T., Structured Analysis (SA) for Communicating Ideas, IEEE Transactions on Software Engineering, Vol.3 No.1, pp.16-34, 1977.

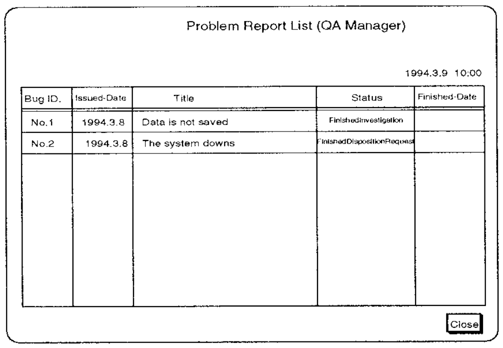

Problem Report List (Worker A)

1994/ 3/ 9 10:00

Bug ID.	Issued Date	Title	Status	Finished Date

Close

Figure 13: Erase a problem report which has already been investigated from the list


Problem Report List (QA Manager)

1994.3.9 10:00

Bug ID.	Issued-Date	Title	Status	Finished-Date
No.1	1994.3.8	Data is not saved	FinishedInvestigation	
No.2	1994.3.8	The system downs	FinishedDispositionRequest	

Close

Figure 14: Notify completion of investigation work of the manager

[12] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., Object-Oriented Modeling and Design, Prentice Hall, 1991.