Annals of Software Engineering 14, 383–406, 2002

A Knowledge-Based Software Process Simulation Model

NORIKO HANAKAWA [a,b], KEN-ICHI MATSUMOTO [b] and KOJI TORII [b] hanakawa@hannan-u.ac.jp
[a] *Hannan University, 5-4-33 Amami, Higashi, Matsubara, Osaka, 580-8502 Japan*
[b] *Nara Institute of Science and Technology, 8916-5, Takayama, Ikoma, Nara, 630-0101 Japan*

Abstract. In this paper we propose a new software development process simulation model. The model can predict variations of productivity based on dynamic changes in the developer's knowledge structure. An important concept of the model is that a developer's productivity is influenced by the developer's knowledge. Moreover, a developer can acquire new knowledge by executing activities of a project. In other words, the developer's knowledge structure changes during the project. The knowledge structure is defined using a cognitive map that consists of knowledge elements and prerequisite relationships among the knowledge elements. By adding the specific developer's knowledge and the specific project workload to the knowledge structure, an increment of the developer's knowledge and the project progress are calculated into the model. The simulation results are useful for making project plans including technical reviews, which are an efficient technique for acquiring new knowledge. The simulation model can predict what knowledge should be discussed in the technical review, when the review should be held, and who the members of the review should be. The simulation results help managers make the most appropriate and executable project plan.

Keywords: process simulation, cognitive map, knowledge structure, technical review

1. Introduction

In today's fast Information Technology society, changes are made almost daily in software development projects. For example, development methodology has changed from a conventional structured development method to an object-oriented development method. Reuse of classes in an object-oriented methodology help developers achieve higher productivity than in a conventional methodology [Hanakawa *et al.* 1999a]. Because customers want more and higher quality software, developers must use new technologies such as object-oriented development methods, even though the acquisition and application of new technologies is often time consuming.

In such situations, software development project managers need to consider the developers' knowledge in project planning and control [Bochenskim 1994]. A developer's knowledge influences the productivity and progress of software development projects [Yourdon 1994]. However, the developers' knowledge structures vary greatly because the knowledge structure depends on the individual developer's experience, motivation, and education. Moreover, such knowledge is dynamically changed during a project. For example, even if a developer has no initial knowledge about Network, the developer can acquire Network knowledge bit by bit in developing a network system. The vari-

ous knowledge structures and the dynamic changes of the knowledge structures make predicting the progress of projects difficult.

For this reason, we propose a new process simulation model based on the dynamic changes in a developer's knowledge structure. The developer's knowledge structure is defined using a cognitive map that consists of nodes and links. The nodes represent knowledge elements, and the links represent prerequisite relationships among the knowledge elements. The dynamic changes of knowledge and the progress of the project in the knowledge structure are calculated using a simulation model that has been already proposed [Hanakawa *et al.* 1998, 1999b].

The proposed model can also make clear what a developer cannot understand, and when a developer can no longer acquire knowledge. The prerequisite relationship can show a prerequisite knowledge shortage that prevents a developer from acquiring new knowledge. Moreover, because two the attributes: the developer's knowledge and the project workload, are added to each knowledge element respectively, the knowledge structure may show an imbalance in the individual developer's knowledge and the specific project workload. For example, if no workload requiring prerequisite knowledge for a new technology exists, a developer will not efficiently acquire the new technology because of this shortage in prerequisite knowledge.

In section 2 of this paper, a knowledge structure in a cognitive map is shown and two attributes added to the knowledge element are explained. In section 3, a simulation model is described. First, the original model already proposed by Hanakawa *et al.* [1998] is described. Next, a new process simulation model and the simulation procedure are explained. In section 4, case studies using a simulator based on the new process simulation model are presented. A discussion of the model's efficiency is shown in section 5. Related works are listed in section 6. In section 7, the conclusion is presented.

2. Knowledge structure

Knowledge is a diffuse concept. However, we define knowledge by focusing on fairly well-defined and contained skills under mathematical models. A key point of our proposed model is that our simulation model includes the dynamic changes of a developer's knowledge. Therefore, we first show a cognitive map that clarifies general elements of knowledge and the prerequisite relationships among the elements. Next, the general cognitive map is transformed into a knowledge structure including the developer's knowledge and the project workload. We explain a way of transforming the cognitive map into the knowledge structure.

2.1. A cognitive map

A developer's knowledge structure is described in a cognitive map [Shavelson 1972] that is popular in cognitive science fields. Figure 1 shows an example of a cognitive map used for developing Application software. To make discussion simple, the range of the cognitive map of figure 1 is limited. A real cognitive map for developing software

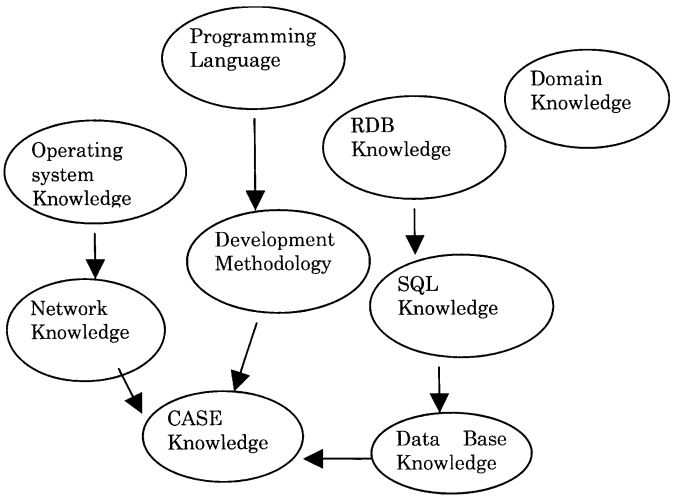

Figure 1. A cognitive map in software development.

will be broader and more complicated. The cognitive map in figure 1 is a graph that consists of nodes and links. The nodes of the graph represent the knowledge required in developing the system. For example, "*Data Base*" knowledge, "*SQL*" knowledge, and "*Network*" knowledge are set to the nodes. We call these "*knowledge elements.*" The links of the graph represent a prerequisite relationship among the knowledge elements. In figure 1, the knowledge element "*RDB*" is a prerequisite for the knowledge element "*SQL.*" That is, if a developer has enough "*RDB*" knowledge, the developer will be able to easily acquire "*SQL*" knowledge. In addition, when a developer executes an activity that requires "*SQL* " knowledge, the developer will acquire not only "*SQL*" knowledge, but also "*RDB*" knowledge.

Here we indicate a way of generating the knowledge elements and the prerequisite relationships among the knowledge elements. We try to generate the knowledge elements and the prerequisite relationships using a textbook. A textbook usually consists of chapters, sections, and subsections. The chapter, section, and subsection discuss a specific topic. In the case of a *C language* textbook, a section shows the details of a concept and examples about a "*Pointer technique.*" Therefore, we use a section topic as a knowledge element "*Pointer.*" Moreover, if the "*Pointer*" concept refers to a section of "*Listing technique*," the knowledge element "*Pointer*" must be a prerequisite for the knowledge element "*Listing.*" The prerequisite relationship between "*Pointer*" and "*Listing*" can be generated using a reference relationship between the section "*Pointer*" and the section "*Listing.*" If we use chapters of a textbook to generate a cognitive map, we will have made a rough cognitive map. In contrast, if we use subsections, we will have made a detailed cognitive map.

In addition, a way of generating the cognitive map has been proposed [Takemura 2002]. The prerequisite relationships have been derived using a statistical technique derived from the results of examinations about knowledge. If there are causal relationships

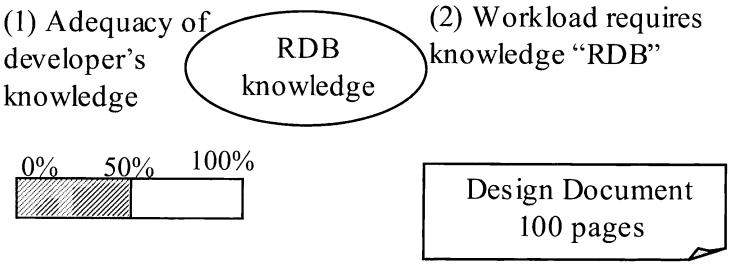

Figure 2. Two attributes added to the knowledge element.

between the right answers of examinations about knowledge "*A*" and the incorrect answers of examinations about knowledge "*B*," then the prerequisite relationship of the cognitive map will be set between knowledge "*A*" and knowledge "*B*."

2.2. *Adding two attributes to the knowledge element*

To transform the general cognitive map into a knowledge structure that is based on an individual developer and a specific project, two attributes are added to the knowledge element (see figure 2). The first attribute is concerned with the individual developer's knowledge, which we term the "*adequacy of knowledge*." The second attribute is concerned with the specific project workload, which we term the "*workload*." The adequacy of knowledge of the knowledge element is represented as the percentage of achievement in acquiring the knowledge element for an individual developer. Figure 3a represents a developer who has sufficient knowledge about the "*Programming Language*" and the "*Development Methodology*," but is not good at "*RDB*," "*SQL*," "*Network*," or "*OS*." The advantages and disadvantages of the individual developer's knowledge are clarified by the "*adequacy of knowledge*" attributes. The workload is represented using an estimation value for the activity that requires the knowledge element to complete the activity. For example, when a developer makes a design document in a project, the estimation values are shown using the number of pages of the design document (see figure 3b). In figure 3b, the designing activity needs "*RDB*," "*SQL*," and "*Data Base*" knowledge, however "*Programming Language*," "*OS*," "*Network*," and "*Development Methodology*" knowledge are not required in the designing activity.

Next we explain a way of getting the values for the two attributes. In the "*adequacy of knowledge*" attribute, the percentage for acquiring the knowledge element is decided by the result of an examination about the knowledge element. For example, if a developer's result of the examination about "*CASE*" is 50%, then the value of the adequacy of the knowledge element "*CASE*" will be set at 50%. On the other hand, the percentage will be extracted from an experience of a developer if the examinations about all knowledge elements cannot be executed. In practice, we investigated the relationship between periods of an individual developer's experience and the "*adequacy of knowledge*" [Hanakawa *et al.* 1999b]. In the result of the investigation using questionnaires sent to managers, when a developer has 10 years or more experience for *C Program-*

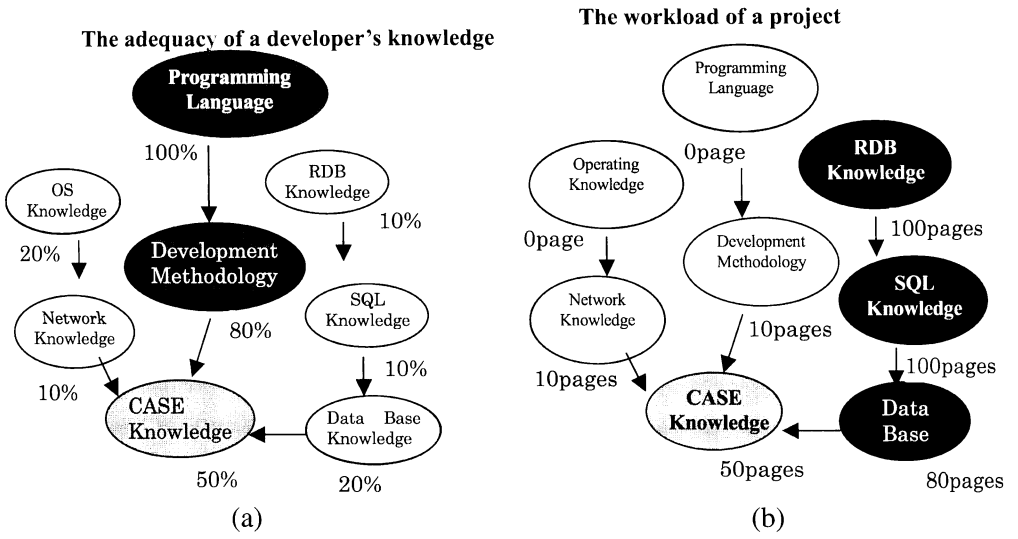

Figure 3. (a) An individual developer's knowledge. (b) The project workload.

ming Language, the percentage of achievement in acquiring knowledge of the *C Programming Language* is almost 100%. If he/she has 5 years experience, the percentage of achievement in acquiring knowledge of the *C Programming Language* will be around 70–80%. If the experience is only 2–3 years, the percentage of achievement in acquiring knowledge of the *C Programming Language* will be around 50%. The result of the investigation, however, is a rough standard.

In the "*workload*" attribute, the estimation values are decided by analyzing documents that have already been completed in other similar projects. In the case of making a design document, first, the total number of pages of the design document that has been completed in the other similar project is counted. Next, the knowledge required in the making of each page of the document is clarified. For example, if in project "*ABC*," a design document is 500 pages, then the number of pages that require knowledge "*RDB*" will be 50; likewise, the number of pages that require knowledge "*SQL*" will be 100. In other words, 10% of the design document required "*RDB*" knowledge, and 20% of the design document required "*SQL*." Using the percentages of the analysis results in a project "*ABC*," the estimation values for the "*workload* " attributes are decided in a new project. If the requirement (e.g., Function Point) in the new project is a half scale of the requirement in the project "*ABC*," we will be able to predict that the total number of pages for the design document in the new project will be 250. Moreover, we can predict that the number of pages requiring knowledge "*RDB*" will be 25, and the number of pages requiring "*SQL*" knowledge will be 50. Therefore, the value 25 is added to the knowledge element "*RDB*" as the "*workload*" attribute and the value 50 is added to the knowledge element "*SQL*" as the "*workload*" attribute in the new project's knowledge structure. If we analyze many design documents in similar projects, the estimation values of the "*workload*" attribute will be more exact.

3. The simulation model

Our new model is generated from combining the new knowledge structure and the original simulation model that has already been proposed [Hanakawa *et al.* 1998]. In this section, first, the original simulation model is explained. Next, we show how the new model combines the original model with the new knowledge structure.

3.1. The original model

The original model consists of three sub models: the Activity Model, the Productivity Model, and the Knowledge Model. The Activity Model illustrates the characteristics of activities that are executed by an individual developer in a project. The Knowledge Model shows the characteristics of an individual developer. The Productivity Model shows the relationships between the activity and the individual developer. One of the key concepts of the original model is the "knowledge level" [Hanakawa *et al.* 1998]. The knowledge level shows how much a developer knows and how difficult an activity is. In other words, a developer with a high level of knowledge is an expert, and an activity that requires a high level of knowledge is a difficult activity. The knowledge level shows not only the developers' skill, but also the difficulty of the activity. Because the knowledge level is concerned with the characteristics of an activity and the developer's characteristics as a common measurement, we call this measurement the "knowledge level" in this paper.

3.1.1. Figure 4. Activity Model

In the original model, a development activity is composed of a set of simple activities called "primitive activities." For example, a design activity is composed of a set of primitive activities: architectural design, interface design component design, and algorithm design. A primitive activity has a knowledge level that is needed to execute this primitive activity. The Activity Model is based on the following assumptions:

(i) an activity is composed of a set of primitive activities,

(ii) the primitive activities are put in order of the level of the required knowledge,

(iii) a high level of knowledge requires more knowledge than a low level of knowledge, and

(iv) the knowledge needed to execute a primitive activity of higher rank includes all the knowledge needed to perform a primitive activity of lower rank.

The Activity Model shows the relationship between the quantity of the primitive activity and the required level of knowledge needed to execute the primitive activity. The Activity Model illustrates a distribution of the level of knowledge needed to execute the primitive activities (see figure 4). The X-axis of the Activity Model represents the level of knowledge needed to execute the primitive activities. In figure 4, the knowledge levels of the Activity Model are divided into 100 levels. The Y-axis represents the total

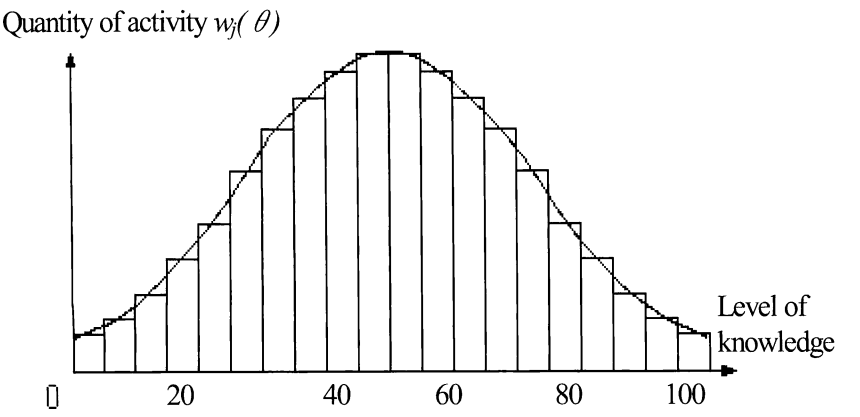

Figure 4. Activity Model.

amount of primitive activities that require the same level of knowledge. If we assume that the distribution of the level of knowledge is normal distribution, then the Activity Model is defined as follows:

$$w_j(\theta) = W_j \frac{1}{\sqrt{2\pi}s} e^{[-(\theta-\bar{\theta})^2/(2s^2)]}. \quad (1)$$

$w_j(\theta)$: total amount of primitive activities required for the level of knowledge θ under an activity j,
θ: required level of knowledge needed to execute a primitive activity of activity j,
W_j: total amount of activity j,
$\bar{\theta}$: average of θ,
s: standard deviation of θ.

3.1.2. Figure 5. Knowledge Model

The Knowledge Model shows the quantity of gain to a developer's knowledge by executing an activity. Quantity of gain to the developer's knowledge is derived from the relationship between b_{ij} and θ. b_{ij} is the developer's level of knowledge, and θ is the required level of knowledge needed to execute the activity. This model is based on the following assumptions:

(i) if b_{ij} is more than θ, developer i will not gain new knowledge by executing activity j, and the developer's level of knowledge will not changed,

(ii) if b_{ij} is less than θ, developer i will gain new knowledge by executing activity j, and the developer's level of knowledge increases. If the gap between b_{ij} and θ is small, developer i gains more new knowledge; if the gap is large, developer i gains limited new knowledge. In short, a developer must perform a difficult activity to acquire knowledge. However, if the activity is too difficult, then he/she will gain little knowledge.

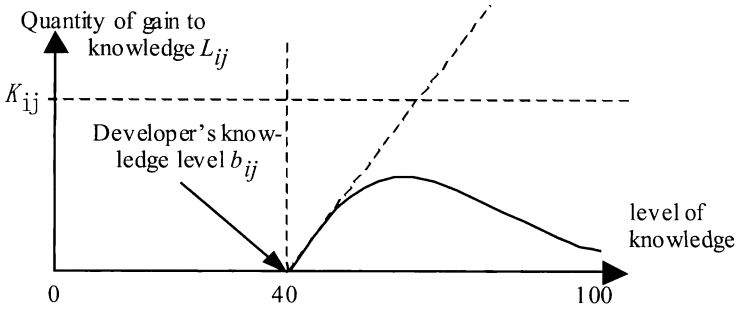

Figure 5. Knowledge Model.

The Knowledge Model is defined as follows (see figure 5):

$$L_{ij}(\theta) = W_j \begin{cases} K_{ij} e^{-E_{ij}(\theta - b_{ij})}, & b_{ij} \leqslant \theta, \\ 0, & b_{ij} > \theta, \end{cases} \tag{2}$$

$L_{ij}(\theta)$: quantity of gains to knowledge of developer i by executing a primitive activity of activity j, which has the level of knowledge θ,
K_{ij}: maximum quantity of gains to knowledge of developer i by executing activity j,
b_{ij}: developer i's knowledge level about activity j,
E_{ij}: developer i's downward rate of gain to knowledge by executing activity j,
θ: required level of knowledge needed to execute the primitive activity of activity j,
W_j: total amount of activity j.

3.1.3. Figure 6. Productivity Model

The Productivity Model shows a developer's productivity in the execution of an activity. When a developer does primitive activity A, the developer's productivity is derived by processing the developer's level of knowledge and the required level of knowledge needed to execute the primitive activity A. The Productivity Model using a cumulative normal model *Ogive model* [Lord 1968] is defined as follows (see figure 6):

$$P_{ij}(\theta) = C_{ij} \int_{-\infty}^{a_j(b_{ij}-\theta)} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt, \tag{3}$$

$P_{ij}(\theta)$: productivity of a developer i under a primitive activity of an activity j, which has the level of knowledge θ,
C_{ij}: maximum of the productivity of developer i under activity j,
a_j: level of accuracy needed to execute activity j (>0),
b_{ij}: level of knowledge of developer i about activity j,
θ: required level of knowledge needed to execute the primitive activity of activity j.

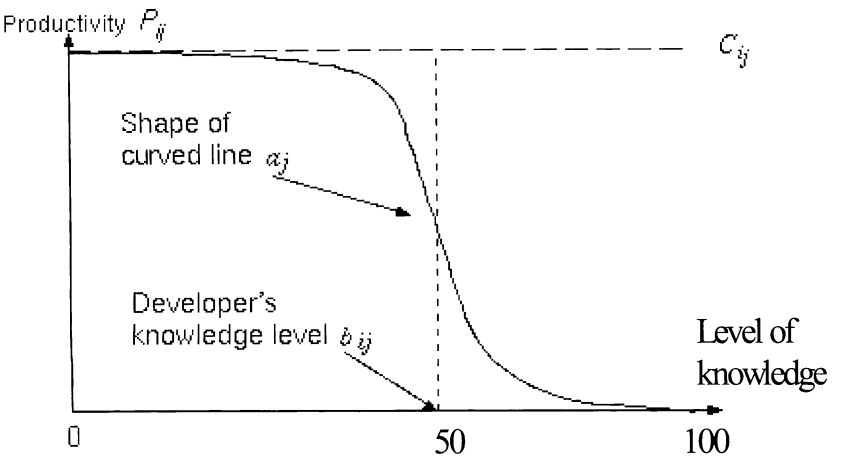

Figure 6. Productivity Model.

In figure 6, the X-axis of the Productivity Model represents the levels of knowledge needed to execute the primitive activities. The level of knowledge of the Productivity Model is divided into 100 levels. The Y-axis represents a developer's productivity. Variation of the productivity is shown as a curve in figure 6. Figure 6 shows that if the developer's level of knowledge b_{ij} is higher than the required level of knowledge θ, then the productivity is high. In contrast, if b_{ij} is less than θ, the productivity is low. Productivity especially decreases within narrow limits. The range is around one point where θ is almost equal to b_{ij}. If θ is equal to b_{ij}, the productivity is half of the productivity's maximum C_{ij}. The parameter a_j is important for determining the shape of the curved line in the Productivity model [Lord 1968]. If a_j is big, then the curved line declines sharply within narrow limits, and a small gap in the developer's knowledge can greatly change the productivity. On the other hand, if a_j is small, then the curved line goes down loosely within wide limits. In particular, if a_j is equal to zero, the productivity is always half of the maximum productivity of C_{ij} regardless of the value of θ; a large gap in the developer's knowledge will only change the productivity slightly.

3.1.4. Simulation procedure

The proposed model can be used to simulate a development progress. The following formula has been applied to compute the value of progress.

$$\text{Progress} = \frac{W_j - \sum_{\theta=1}^{100} w_j(\theta)}{W_j}, \tag{4}$$

$w_j(\theta)$: remaining quantity of primitive activities that require a level of knowledge θ under an activity j,

θ: required level of knowledge needed to execute a primitive activity of activity j,

W_j: total amount of activity j.

This simulation consists of the following steps:

Step 0. Initialization.

Parameters (W_j, b_{ij}, $\bar{\theta}$, s, a_j, C_{ij}, K_{ij}, E_{ij}) of the three submodels are initialized to the values determined by a user (e.g., a project manager). The time t is initialized to zero.

Step 1. Choice of a primitive activity.

A primitive activity that is executed at time t is chosen randomly out of the primitive activities with non-zero quantities in the Activity Model. Parameter θ is determined by the chosen primitive activity.

Step 2. Calculation of productivity $P_{ij}(\theta)$.

The value of productivity $P_{ij}(\theta)$ is calculated by equation (3) in the Productivity Model which is given four parameters: C_{ij}, which is determined in step 0, a_j, which is determined in step 0, b_{ij} (the developer's level of knowledge), which is determined in step 0 (only in the first cycle) or step 5, and θ (required level of knowledge needed to execute the primitive activity) which is determined in step 1.

Step 3. Renewal of the quantity of a primitive activity $w_j(\theta)$ in the Activity Model.

$P_{ij}(\theta)$, which has already been calculated in step 2 is subtracted from the amount of the primitive activity that has the θ. The Activity Model is reset to the result of this subtraction. Using mathematical terms, $w_j(\theta)$ of the Activity Model can be expressed as

$$w_j(\theta)(t+1) = w_j(\theta)(t) - P_{ij}(\theta). \tag{5}$$

Step 4. Calculation of the quantity of gain to knowledge $L_{ij}(\theta)$.

The quantity of gain to knowledge $L_{ij}(\theta)$ is calculated by equation (2) in the Knowledge Model which is given four parameters: K_{ij}, which is determined in step 0, E_{ij}, which is determined in step 0, b_{ij}, developer's level of knowledge, which is determined in step 0 (only in first cycle) or step 5, and θ (required level of knowledge needed to execute the primitive activity), which is determined in step 1.

Step 5. Renewal of the developer's knowledge level b_{ij}.

The quantity of gain to knowledge $L_{ij}(\theta)$ that has already been calculated in step 4 is added to b_{ij} (the developer's level of knowledge). The level of knowledge is reset to the developer's new knowledge level b_{ij}:

$$b_{ij}(t+1) = b_{ij}(t) + L_{ij}(\theta)(t). \tag{6}$$

Step 6. Renewal of the value of productivity in the Productivity Model.

The value of productivity is reset to the developer's new b_{ij} (the developer's level of knowledge) determined in step 5.

Step 7. Go to step 1.

In the Activity Model, if the total amount of remained activities $\sum_{\theta=1}^{100} w_j(\theta)$ is equal to zero, the simulation is finished; if $\sum_{\theta=1}^{100} w_j(\theta)$ is greater than 0, set the value of time t to $(t+1)$, then go back to step 1.

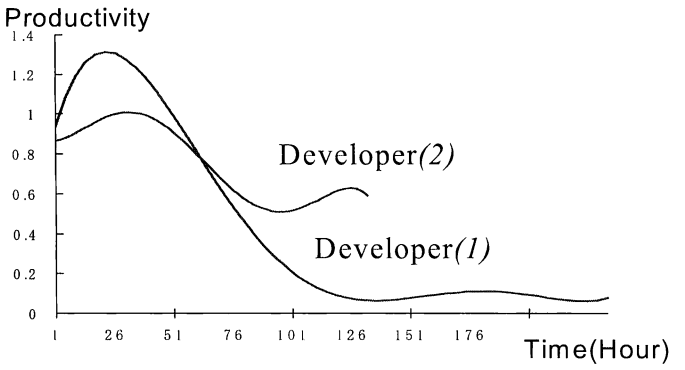

Figure 7. Variations of productivities.

In the simulation, the developer's level of knowledge b_{ij} is increased in step 5. The growth of b_{ij} during the execution of activity j shows the developer's learning. By resetting the Productivity Model in the new b_{ij}, the curve of productivity in figure 6 shifts to the right side of the Productivity Model; that is, productivity becomes higher than the productivity in the former Productivity Model that is not reset yet. Therefore, the model can show the variation of productivity based on changes in the developer's level of knowledge.

3.1.5. Case studies of the original model

In order to evaluate the usefulness and characteristics of the original model, we apply the model to a case in which the development progress is influenced by the developer's knowledge. In this case, we assume that two developers execute the same activity using a new technology. The first developer*(1)* has sufficient experience and knowledge in conventional technology ($b_{ij} = 75$), but he/she is not good at acquiring new ideas of the new technologies effectively ($K_{ij} = 1$, $E_{ij} = 0.01$). The second developer*(2)* has little experience and knowledge ($b_{ij} = 25$) in software development, but he/she easily learns new ideas of the new technologies ($K_{ij} = 10$, $E_{ij} = 0.1$). The other parameters' (W_j, b_{ij}, $\bar{\theta}$, s, a_j, C_{ij}) values are same. The simulation results are shown in figures 7, 8. Figure 7 shows variations of the developers' productivities, and figure 8 shows the progress of the developers in executing the activity. The gap between the two developers is clarified. In the early stage of the project, developer*(1)* achieves a higher productivity than developer*(2)*'s productivity, and developer*(1)* makes better progress than developer*(2)*'s. However, in the middle stage of the project, developer*(1)*'s productivity decreases suddenly because developer*(1)* is not good at acquiring new skills. It takes long time for the developer*(1)* to execute the remaining difficult activities. In contrast, the reduction of developer*(2)*'s productivity is less than the reduction of developer*(1)*'s productivity because developer*(2)* is good at acquiring new technologies. Executing the remaining difficult activities does not take much time. As a result, to complete the activity, developer*(1)* takes 229 hours, while developer*(2)* takes 133 hours. A simulation result will help a project manager decide who should be a member of a critical project.

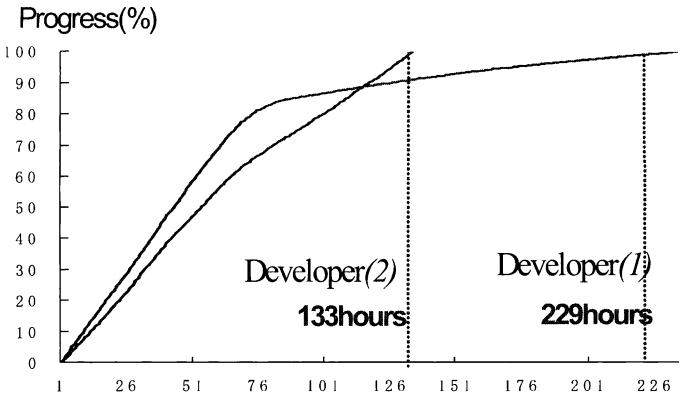

Figure 8. Progress of the developers.

3.1.6. Parameter values

The values of the model's parameters are determined based on industrial project managers' intuitions and experience [Hanakawa *et al.* 1999b]. Using questionnaires about virtual projects, experienced managers estimate the workload for the activity's parameters (W_j, $\bar{\theta}$, s, a_j), the parameters for the developers' abilities (b_{ij}, C_{ij}, K_{ij}, E_{ij}) and the development periods for the simulation results. Formulas converting the value of the estimations to the values of the parameters are derived, as all estimations by the managers are valid. Using the formulas of the parameters and the manager's answers to the questionnaires about the projects, the values of the parameters of the original model are determined. In addition, the simulation results of the original model have already been evaluated. The simulation results (development periods) fit experienced managers' estimations at a 5% level of statistical significance.

3.2. *The new model*

To apply the concept of the knowledge structure (see section 2) to the original model, the three sub models of the original model are allocated to each knowledge element in the knowledge structure, respectively (see figure 9). The developer's knowledge level b_{ij} in the Productivity Model and the Knowledge Model are equal to the first attribute, "*adequacy of knowledge*," which is set to each knowledge element. The total amount of activity W_j in the Activity Model is equal to the second attribute, "*workload*," which is set to each knowledge element.

3.2.1. Advanced Knowledge Model

Formula (2) of the Knowledge Model changes as follows:

$$L_{ij}(\theta)' = W_j \begin{cases} K'\mathrm{e}^{-E_{ij}(\theta - b_{ij})}, & b_{ij} \leqslant \theta, \\ 0, & b_{ij} > \theta, \end{cases} \qquad K' = K_{ij} \times \frac{\sum_{l=1}^{all} b_{ij}(l)}{100 \times all}, \tag{7}$$

$L_{ij}(\theta)'$: quantity of gains to knowledge of a developer i by executing a primitive activity of activity j, which has knowledge level θ,
b_{ij}: developer i's level of knowledge about activity j,
E_{ij}: developer i's downward rate of gain to knowledge by executing activity j,
θ: required level of knowledge needed to execute the primitive activity of activity j,
W_j: total amount of activity j,
K_{ij}: maximum quantity of gains to knowledge of developer i by executing activity j,
all: the number of the prerequisite knowledge elements for knowledge element "A."

We call formula (7) the "Advanced Knowledge model." Parameter K_{ij} of the original model changes to K' of the Advanced Knowledge Model. In K', K_{ij} is defined based on the ratio of achievement in acquiring the prerequisite knowledge elements. That is, if a developer has sufficient prerequisite knowledge, there will be a high possibility that the developer will gain knowledge element "A" by executing an activity j because the value of $(\sum_{l=1}^{all} b_{ij}(l))/(100 \times all)$ is almost 1.0. In contrast, if a developer has little prerequisite knowledge, where the value of $(\sum_{l=1}^{all} b_{ij}(l))/(100 \times all)$ is almost 0.0, the developer cannot efficiently acquire knowledge element "A" by executing an activity j, in spite of his/her efforts.

For example, in figure 9, the workload of the activity that requires "*SQL*" knowledge is 50 pages of a design document. The developer, however, has only 20% knowledge about "*SQL*." Moreover, the developer has only 10% of the prerequisite knowledge "*RDB*" for knowledge "*SQL*." Therefore, the developer has to make the design document by learning not only "*SQL*" but also "*RDB*." He/she will need much time to make a design document with "*SQL*" knowledge because the developer needs to learn "*SQL*" and "*RDB*." In contrast, if the developer has sufficient knowledge of "*RDB*," the developer will finish making the document in a shorter time than in the above case. All in all, we may assume that a shortage of prerequisite knowledge prevents a developer from acquiring new knowledge.

3.2.2. Simulation procedure for the new model

A simulation procedure consists of six steps including the procedure of the original model (see section 3.1).

Step A. Creating a cognitive map.
Determining a cognitive map in a cognitive science field is difficult. However, our purpose in using the cognitive map model is only to make clear the knowledge elements and the prerequisite relationships among the knowledge elements. A way of generating a cognitive map has already been shown in section 2.1.

Step B. Initializing the cognitive map into a knowledge structure.
The two attributes: adequacy of knowledge (b_{ij}) and *workload* (W_j), are added to each knowledge element. A way of determining the values of the attributes has al-

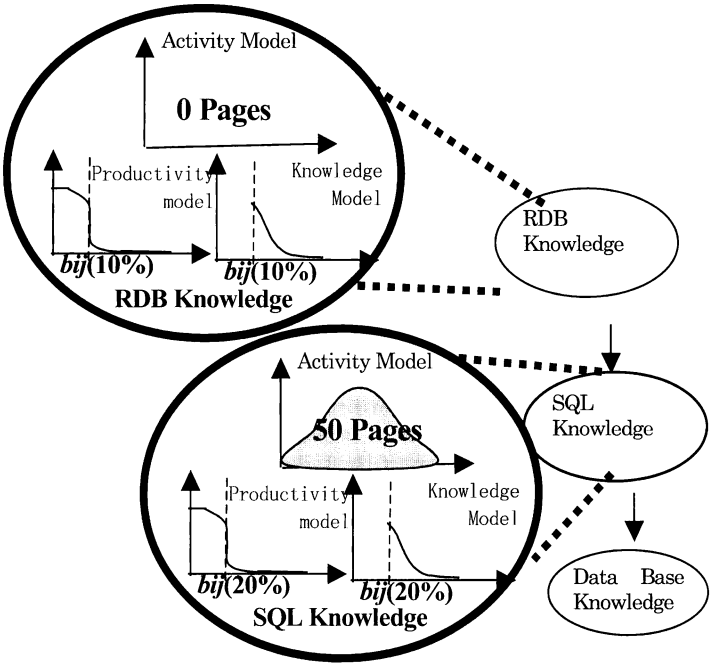

Figure 9. Setting the original model to the knowledge element.

ready been shown in section 2.2. In addition, the three submodels: the Activity Model, the Advanced Knowledge Model, and the Productivity Model, are initialized (see step 0 of the original model) in each knowledge element of the knowledge structure. Time t is set to zero.

Step C. Choice of a knowledge element.

First, a knowledge element "l" in the knowledge structure is chosen randomly at time t. Next, a primitive activity is chosen randomly out of the primitive activities in the Activity Model of the chosen knowledge element. Parameter θ at time t is determined by the chosen primitive activity (see step 1 of the original model).

Step D. Calculation in the model of the knowledge element.

The procedure from step 2 to step 6 of the original model (see section 3.1) is executed here. However, the Advanced Knowledge Model is used instead of the Knowledge Model of the original model. The productivity $P_{ij}(\theta)$, quantity of gain to knowledge $L_{ij}(\theta)'$ and the developer's knowledge level b_{ij} of the knowledge element "l" are calculated in the Productivity Model and the Advanced Knowledge Model of knowledge element "l." In addition, the Productivity Model and the Advanced Knowledge Model of knowledge element "l" are reset using a revised b_{ij}, and the Activity Model is reset by $P_{ij}(\theta)$.

Step E. Acquisition of prerequisite knowledge.

By executing the chosen primitive activity $w_j(\theta)$ of knowledge element "l," a developer's knowledge level b_{ij}'s of all prerequisite knowledge elements for the knowl-

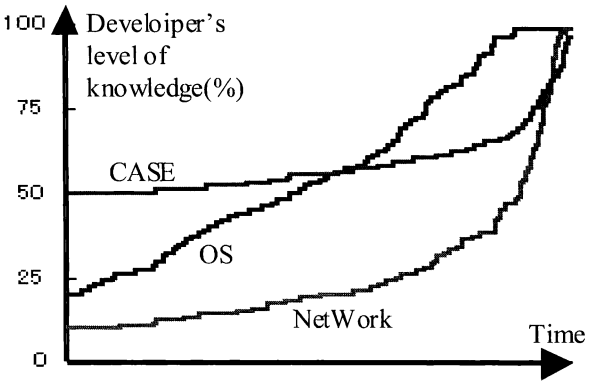

Figure 10. Increments of the developer's level of knowledge of the knowledge elements.

edge element "l" also increases. Step 4, step 5 and step 6 of the original model (see section 3.1) are executed with all prerequisite knowledge elements for the knowledge element "l." Like step D, the Advanced Knowledge Model is used instead of the Knowledge Model of the original model. In step 4 the quantities of gain to knowledge $L_{ij}(\theta)'$ of the prerequisite knowledge elements are calculated. In step 5 the developer's knowledge level b_{ij}'s of the prerequisite knowledge elements are renewed based on each $L_{ij}(\theta)'$, and the Advanced Knowledge models of the prerequisite knowledge elements are reset by the revised each b_{ij}. In step 6 the Productivity models of the prerequisite knowledge elements are reset based on the revised each b_{ij}. However, the Activity models of the prerequisite knowledge elements are not renewed because there is no execution of activities requires the prerequisite knowledge elements, in other words, workloads of the prerequisite knowledge elements do not decrease.

Step G. Judgment of finish.

In all Activity models of all knowledge elements, if the total amount of remained activities $\sum_{\theta=1}^{100} w_j(\theta)$ of all Activity models is equal to zero, the simulation is finished; if $\sum_{\theta=1}^{100} w_j(\theta)$ of all Activity models is greater than zero, set the value of time t to $(t+1)$, then go back to step C. Time t at the finish represents a development period.

3.2.3. Increment of prerequisite knowledge

In figure 10, a simulation result of a knowledge structure in the example case of figure 3 is shown. The knowledge structure of figure 3 consists of eight knowledge elements. We chose the knowledge elements "*OS*," "*Network*," and "*CASE*" from the knowledge structure in figure 3. The workload of an activity that requires knowledge element "*CASE*" is 50 pages; in contrast, the workload of an activity that requires knowledge elements "*OS*" and "*Network*" is zero. Therefore, it appears that a developer does not need knowledge "*OS*" and "*Network*." Nonetheless, the knowledge elements "*OS*" and "*Network*" are the prerequisite knowledge for the knowledge element "*CASE*" which means that the developer has to acquire not only the knowledge element "*CASE*," but also the knowledge

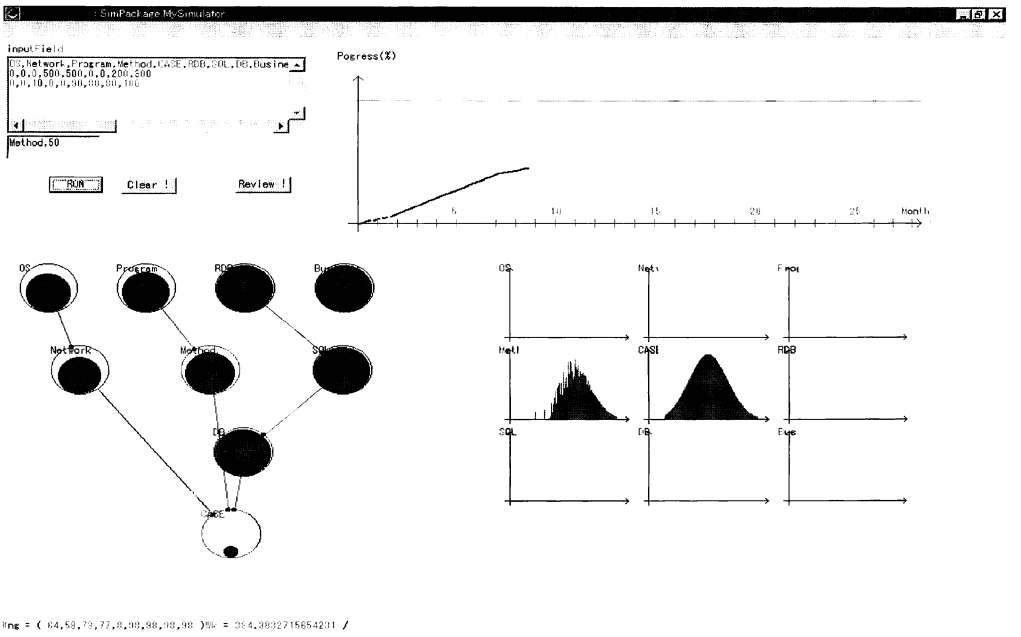

Figure 11. A simulator based on the proposed model.

elements "*OS*" and "*Network*." Therefore, in figure 10, the developer's levels of knowledge of "*OS*" and "*Network*" also increase while the developer executes the activity that requires the knowledge element "*CASE*."

4. Case studies

Some cases are simulated using the model in order to illustrate our idea. The potential ability of the model is shown in the simulation results of the cases studies.

4.1. A simulator based on the proposed model

A simulator was implemented based on a simulation model similar to figure 11. The simulator consists of four parts. The first part is an input area at the upper-left side of figure 11. Users input information of a case of simulation including the knowledge elements, the prerequisite relationships, the workload of activities, and the adequacy of knowledge elements. The second part is a knowledge structure with the adequacy of knowledge. The knowledge structure is shown at the lower-left side of the simulator. The size of the circle filled with black color represents the adequacy of the knowledge element. If the blackened circle is large, the adequacy of the knowledge element is great; likewise, if the blackened circle is small, the developer has little knowledge. The blackened circles grow larger and larger during a simulation.

Table 1
Values of two attributes of knowledge elements.

Knowledge element	Case 1		Case 2	
	Adequacy of the knowledge element	Number of pages of the design document	Adequacy of the knowledge element	Number of pages of the design document
OS	10%	0	0%	0
Network	10%	0	0%	0
Program	80%	150	10%	0
Method	90%	300	0%	500
CASE	90%	150	0%	500
RDB	0%	0	90%	0
SQL	0%	0	80%	0
DB	20%	300	80%	200
Domain	40%	450	100%	400

At the upper-right side of figure 11 is the third part of the simulator. This part is a progress graph. During a simulation, increments of progress are shown in this part. The fourth part is an area of Activity models. An Activity model is allocated to each knowledge element, respectively. These Activity models are shown at the lower-right side of figure 11. The Activity model shows the quantity of the remaining activity that requires the knowledge element. The shapes of the Activity models are also changed during a simulation.

4.2. Two typical cases

We show two typical cases of a developer making a design document. We assume that a developer completes a design document in 10 months with full knowledge of knowledge elements "*OS*," "*Network*," "*Program language*," "*Development methodology*," "*CASE tool*," "*RDB*," "*SQL*," "*DB*" and "*Domain*" (see figure 11). The prerequisite relations among the knowledge elements are set as the links of the cognitive map in figure 11. In case 1, the developer has to make a design document that requires "*DB*" knowledge, however, he/she has little knowledge of "*DB*." In case 2, the developer has to make a design document in a new development environment; a new OS, a new CASE tool, or in a new development methodology.

4.2.1. Case 1

A developer with good skills in "*Programming*" is assigned to case 1 (see column "Case 1" of table 1). Although the developer is good at "*Programming*," he/she has only a little knowledge of "*OS*" (10%), "*Network*" (10%), "*RDB*" (0%), "*SQL*" (0%) and "*DB*" (20%). Making the design document requires knowledge of "*Programming*," "*Method*," "*CASE*" and "*DB*." The number of pages of the design document are 150 (requiring "*Program*"), 300 (requiring "*Method*"), 150 (requiring "*CASE*") and 300 (requiring "*DB*"), respectively. In other words, the developer has to acquire "*DB*" knowledge to make the design document.

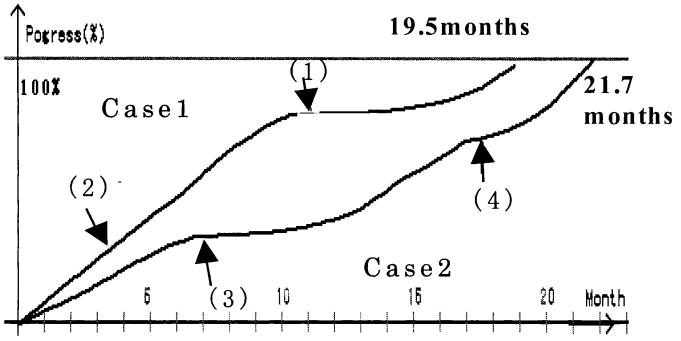

Figure 12. Simulation results of the case studies.

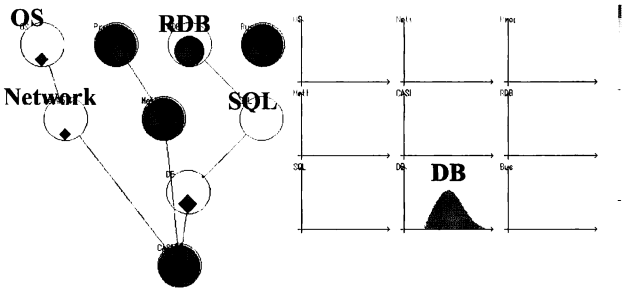

Figure 13. The cognitive map and Activity models at (1) of the case 1 simulation.

The simulation result is a curved line called "*Case 1*" in figure 12. The developer needs 19.5 months to complete the design document. The productivity is not constant; the developer's productivity decreases suddenly, l 1.5 months later. This point is marked with (1) in figure 12. The reason for the decrease in productivity is that the developer needs to acquire the "*DB*" knowledge because the remaining activity requires only "*DB*" knowledge. Figure 13 shows the cognitive map and the Activity models at (1) in figure 12. In the cognitive map of figure 13, the adequacies of "*DB*," "*SQL*," "*OS*" and "*Network*" knowledge are very small. On the other hand, in the Activity models of figure 13, only the activity that requires the "*DB*" knowledge remains. The quantities of the other activities are zero. Therefore, we can assume that the developer is deadlocked because of the shortage of "*DB*" knowledge at (1) in figure 12.

In addition, the acquisition of "*DB*" knowledge requires "*SQL*" knowledge. However, the "*SQL*" knowledge is zero. We can guess that the shortage of "*SQL*" knowledge prevents the developer from acquiring "*DB*" knowledge. If the developer gains "*SQL*" knowledge from an expert in a technical review at (1) in figure 12, the productivity of the developer will increase. The technical review is discussed in section 5.

4.2.2. Case 2

A developer with knowledge about "*DB*" is assigned to case 2 (see column "*Case 2*" of table 1). The developer has to make a design document with little knowledge about

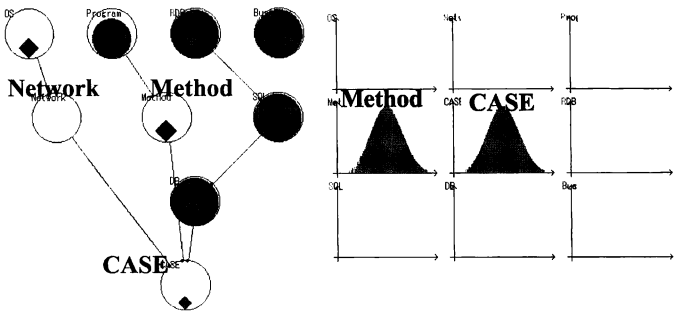

Figure 14. The cognitive map and Activity models at (3) of the case 2 simulation.

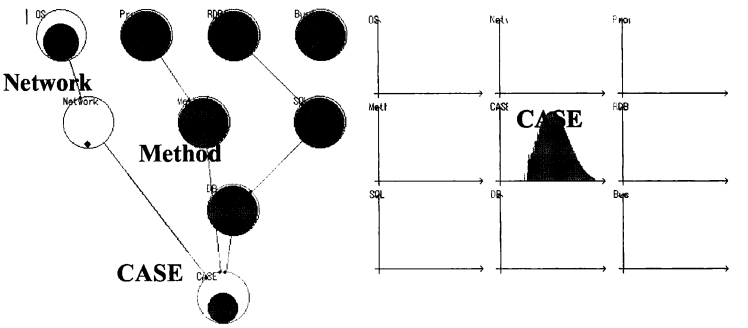

Figure 15. The cognitive map and Activity models at (4) of the case 2 simulation.

"*OS*" (0%), "*Network*" (0%), "*Program*" (10%), "*Method*" (0%) and "*CASE*" (0%). In other words, the developer has to design a system in a new development environment. Making the design document requires knowledge of "*Method*," "*CASE*" and "*DB*." The number of pages of the design document are 500 (requiring "*Method*"), 500 (requiring "*CASE*") and 200 (requiring "*DB*"), respectively. The developer must acquire the "*Method*" and "*CASE*" knowledge to make the design document.

The result of the simulation is a curved line called "*Case 2*" in figure 12. The developer needs 21.7 months to make the design document. There are two points where the developer's productivity decreases suddenly in case 2. These two points are marked as (3) and (4) in figure 12. Figure 14 shows the cognitive map and the Activity models at (3) of case 2. Because the developer has little knowledge of "*Method*" at (3), the activity that requires "*Method*" knowledge remains open. Also, a shortage of "*OS*," "*Network*," and "*Method*" knowledge prevents the developer from acquiring "*CASE*" knowledge, the activity that requires the "*CASE*" knowledge is not reduced. In the period from (3) to (4) of case 2, the developer has acquired "*Method*" knowledge (compare the cognitive map of figure 14 and the cognitive map of figure 15). Therefore, the developer's productivity is low during the first half of the period from (3) to (4) in figure 12. After (4) of case 2 in figure 12, because the developer makes the remaining document by acquiring "*CASE*" and "*Network*" knowledge, the productivity decreases again around (4) in figure 12 (see the cognitive map of figure 15).

In short, the simulation model can calculate the dynamic changes of the developer's knowledge structure (see cognitive maps of figures 13–15). The cognitive map during the simulation shows how the adequacy of the developer's knowledge increases, and what knowledge a developer cannot acquire any more. The developer's productivity also varies as the knowledge structure changes. In addition, the quantities of the remaining activities are also calculated into the simulation. By comparing the remaining activities with the knowledge structure, a project manager will be able to get answers to the questions: "what knowledge should the developer acquire?" and "when should that knowledge be acquired?".

5. Discussion

In this section, we discuss the usefulness of the results of the simulation. In this section, an efficient way of acquiring knowledge as a set of technical reviews [Hanakawa *et al.* 2000] is discussed. In a technical review, an expert can solve the problems of a developer; in other words, an expert can provide knowledge that a developer cannot acquire by himself/herself. We use the results of the simulation to make project plans including the technical reviews. Again, we assume that a technical review requires 30 hours of work.

5.1. Case 1 with a technical review

In the result of the simulation of case 1 in figure 12, the developer's productivity decreases suddenly. As mentioned in section 4, the developer cannot acquire the "*DB*" knowledge because of the shortage of "*SQL*" knowledge. If the developer can gain the "*SQL*" knowledge at (1) in case 1, then the developer's productivity will not decrease. Figure 16 shows the result of a simulation including a technical review (Review 1) where the developer is given the "*SQL*" knowledge from an expert at (1) in figure 12. The developer completes the design document within 14.4 months (see figure 16). Review 1 reduces the development period of making the document to 5.1 months.

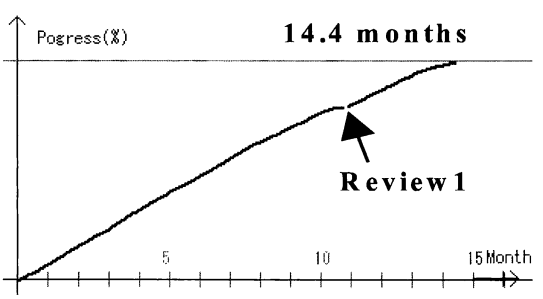

Figure 16. The simulation result of the case 1 with Review 1.

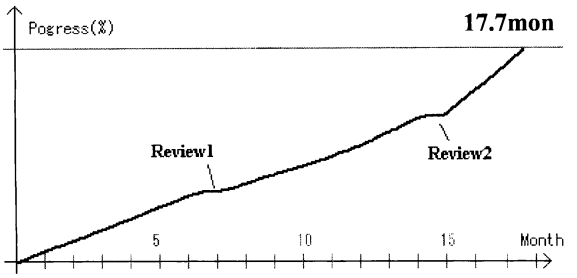

Figure 17. The simulation result of the case 2 with two reviews, Review 1 and Review 2.

5.2. *Case 2 with technical reviews*

In the result of the simulation of case 2 in section 4, the developer's productivity decreases suddenly at two points: (3) and (4) of figure 12. Technical reviews can keep the productivity from decreasing. Figure 17 shows the result of a simulation with two technical reviews: Review 1 and Review 2. In Review 1 held at (3) of case 2, an expert provides "*Method*" knowledge to the developer. In Review 2 held at (4) of case 2, the developer acquires "*CASE*" knowledge from an expert. As a result, the developer completes the design document within 17.7 months. The two reviews reduce 4 months of the development period.

5.3. *Various technical reviews*

Holding technical reviews not only wastes a developer's time and efforts but also wastes the valuable time and effort of the experts. Obviously, holding too many reviews is time consuming. Too many reviews prevent developers from executing a project efficiently. Reducing the development period using the fewest technical reviews when making a project plan is important. Also, although a project manager may find and use the most efficient review, the manager may not be able to make a project plan that contains the most efficient review. For example, an expert may not be assigned to the review because an expert is busy with his or her own tasks.

In such cases, project managers should consider plans that include various technical reviews. Therefore, a simulation is useful in embedding various reviews into various plans. By comparing the results of simulations and the possibility of assigning experts, the manager will be able to indicate the most appropriate and executable plans.

Table 2 shows six simulation results of the case 1 example with various reviews. The shortest period of the simulation result is 14 months in No. 1. If a manager wants to complete the project as soon as possible, the manager should make a plan embedding a review of No. 1. However, if the manager cannot assign the experts who have "*SQL*" and "*RDB*" knowledge, then the manager should choose the review in No. 6. The review in No. 6 is the second shortest period for a case. In such a way a manager can choose the most appropriate and executable plan.

Table 2
Simulation results with various reviews.

No.	Contents of review	When is the review held?	Simulation result
1	SQL (2 times), RDB	11 months	14.0 months
2	SQL (2 times)	11 months	14.6 months
3	SQL	11 months	16.7 months
4	SQL	5 months	15.8 months
5	DB	5 months	18.4 months
6	DB	11 months	14.4 months

In addition, a project manager can choose the most efficient review using the results of the simulation. By comparing No. 1 and No. 2 of table 2, the result (14.0 months) of No. 1 is not so different from the result (14.6 months) of No. 2. However, three reviews (2 times SQL and RDB, or 90 hours) are held in No. 1; in contrast, two reviews (2 times SQL, or 60 hours) are held in No. 2. In other words, in case 1, the simulation results indicate that holding the two technical reviews of No. 2 is more efficient than holding the three technical reviews of No. 1. Next, by comparing No. 5 (holding the review 5 months later) and No. 6 (holding the review 11 months later), the developer in No. 5 takes 4 months more than in the case of No. 6, in spite of the same review about "*DB*." The review in No. 5 is held too early for the developer to acquire the "*DB*" knowledge. In other words, because the prerequisite knowledge for the "*DB*" knowledge is not sufficient, the developer cannot understand what the expert explains in the technical review. Obviously, the manager will choose review No. 6.

6. Related works

Several methods have already been proposed for modeling and evaluating the software development process. Kellner proposed a method for modeling the software development process through three separate points of view: structural, functional, and behavioral [Kellner 1991]. The process simulation is mainly based on the behavioral model, and the model executes STATEMATE, which is one of the CASE tools used to design software for communication networks. STATEMATE can establish the schedule for software development and can estimate work effort. Since STATEMATE is based on the behavioral model, productivity and changes of knowledge are calculated from the probability of a state-transition. However, STATEMATE does not take variations of productivity and changes of knowledge into account.

Kusumoto *et al.* proposed a development model using an extended Generalized Stochastic Petri-net [Kusumoto *et al.* 1997]. In this model, development period, work effort, and quality of software are estimated by assigning developers to activities. The model has a parameter including the experience level of the developer. This parameter is important in determining the probability of the injection and the removal of a fault and the firing rate of a Petri-net. Although the experience level parameter is a constant for each developer, the parameter does not change during the simulation.

Eden *et al.* proposed a model of competencies using a cognitive mapping [Eden *et al.* 2000]. The model shows competencies as patterns as well as the way in which the patterns often express the distinctiveness of competencies. The relationship between the patterns of competencies and the goals of an organization are explored as the basis for establishing core distinctive competencies. Using this relationship, a business model that will inform strategic direction is explored. Although the model is useful in planning a project's strategy, productivity, which is an important factor in the planning phase, is not discussed.

A conventional cost estimation model COCOMO takes account of a developer's skills and experience [Boehm 1981]. COCOMO is a statistical model that is derived from the analysis of 63 software projects. The developer's skills and experience influence the results of estimations in COCOMO. However, a concept of the dynamic change of a developer's skill and knowledge during a project are not incorporated into COCOMO.

7. Conclusion

We propose a new simulation model based on an individual developer's knowledge structure. The model consists of a knowledge structure and a simulation model that has already proposed. Since two attributes (workload of an activity and adequacy of a developer's knowledge) are added to all knowledge elements in the knowledge structure, the simulation results include the dynamic changes made to a developer's knowledge structure. Based on these results, managers can make plans for projects that include the most appropriate and the most executable technical reviews. The reviews help a developer acquire new knowledge efficiently.

The basic concept of the model is concerned with an individual developer's knowledge structure although there are actually many developers in a software development project. Obviously, the developers influence each other in the collaboration of tasks, and an individual developer's knowledge is influenced by other developers' knowledge, workloads, communications, and motivation. In the future, we will investigate how the developers' knowledge structures changes in the collaboration of tasks, and how the developers' productivities vary in activities divided into developers.

References

Bochenskim, B. (1994), *Implementing Production-Quality Client/Server System*, Wiley, New York.

Boehm, B.W. (1981), *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.

Eden, C. and F. Ackermann (2000), "Mapping Distinctive Competencies: A Systemic Approach," *International Journal of the Operational Research Society 51*, 1, 12–20.

Hanakawa, N., H. Iida, K. Matsumoto, and K. Torii (1999a), "Generation of Object-Oriented Software Process Using Milestones," *International Journal of Software Engineering and Knowledge Engineering 9*, 4, 445–466.

Hanakawa, N., K. Matsumoto, and K. Torii (1999b), "Application of Learning Curve Based Simulation Model for Software Development to Industry," In *Proceedings of the 11th International Conference on Software Engineering and Knowledge*, Kaiserslautern, Germany, World Scientific Publishing, pp. 283–289.

Hanakawa, N., S. Morisaki, and K. Matsumoto (1998), "A Learning Curve Based Simulation Model for Software Development," In *Proceedings of the 20th International Conference on Software Engineering*, Kyoto, Japan, IEEE Computer Society Press, pp. 350–359.

Hanakawa, N. and H. Nogi (2000), "Human Factor-Based Quality Control with Technical Reviews," In *Proceedings of the Second International Conference on Software Quality*, Yokohama, Japan, Union of Japanese Scientists and Engineers, pp. 563–568.

Kellner, M.I. (1991), "Software Process Modeling Support for Management Planning and Control," In *Proceedings of the 1st International Conference on Software Process*, Redondo Beach, CA, pp. 8–28.

Kusumoto, S., O. Mizuno, T. Kikuno, Y. Hirayama, Y. Takagi, and K. Sakamoto (1997), "A New Software Project Simulator Based on Generalized Stochastic," In *Proceedings of 19th International Conference on Software Engineering*, Boston, MA, IEEE Computer Society Press, pp. 293–302.

Lord, F.M. and M.R. Novick (1968), *Statistical Theory to Mental Test Score with Contributions by A-Birnbaum*, Addison-Wesley.

Shavelson, R.J. (1972), "Some Aspects of the Correspondence Between Content Structure and Cognitive Structure in Physics Instruction," *International Journal of Educational Psychology 33*, 225–234.

Takemura, Y. (2002) "A Framework for Supporting Software Engineer Education and an Effective Process for Learning Programming," PhD Dissertation (NAIST-IS-DT-9961017), Department of Information System, Nara Institute of Science and Technology, Nara, Japan (in Japanese).

Yourdon, E. (1994), *Object-Oriented System Design*, Prentice-Hall.