Regular Paper

Quantitative Analysis of Information Leakage in Security-Sensitive Software Processes

Yuichiro Kanzaki,† Hiroshi Igaki,† Masahide Nakamura,† Akito Monden† and Ken-ichi Matsumoto†

This paper presents a method to evaluate the risk of information leakage in software processes for security-sensitive applications. A software process is modeled as a series of subprocesses, each of which produces new work products from input products. Since a process is conducted usually by multiple developers, knowledge of work products is shared among the developers. Through the collaboration, a developer may share with others the knowledge of products that are not related to the process. We capture the transfer of such irrelevant product knowledge as information leakage in a software process. In this paper, we first formulate the problem of information leakage by introducing a formal software process model. Then, we propose a method to derive the probability that each developer d knows each work product p at a given process of software development. The probability reflects the possibility that someone leaked the knowledge of p to d. We also conduct three case studies to show the applicability of leakage to practical settings. In the case studies, we evaluate how the risk of information leakage is influenced by the collaboration among developers, the optimal developer assignment and the structure of the software process. As a result, we show that the proposed method provides a simple yet powerful means to perform quantitative analysis on information leakage in a security-sensitive software process.

1. Introduction

Information leakage is a serious problem in today's advanced information society. Many incidents have been reported recently, including leakage of user accounts and passwords [11], medical records [13], and product source codes [4]. Because of its impact on trust and security, minimizing the *risk* of information leakage is now a social responsibility for every organization.

Although information leakage occurs in many domains, this paper focuses especially on the leakage in a software development process. The development of a complex and large-scale software system requires the collaborative effort of many people with an elaborate *software process* [8]. A (whole) software process is composed of partially-ordered *sub-processes* (simply called *processes*) such as design, coding and testing. Each process has *work products* (simply *products*) as either the input or output of the process.

Typically, a number of developers including manages, designers and programmers, participate in a common process. Given input products, the developers collaborate with each other, and produce output products. Through the collaboration, they usually share their knowledge of the products in order to achieve the process efficiently. Thus, when multiple developers participate in a process, certain *knowledge transfer* may occur in the process. From the viewpoint of efficiency, knowledge transfer should be improved, since it helps developers to acquire a similar understanding of the process [9].

However, in the development of security-sensitive software such as DRM applications [3],[10], the transfer of product knowledge is not always encouraged. To understand this better, we introduce a simple example.

Figure 1 shows a software process consisting of two processes P1 and P2. P1 is a design process conducted by developer Alice, where Alice produces a product *Specification* from two given products, *Requirement* and *SecretInfo*. Here we assume that *SecretInfo* is confidential information (e.g., device keys or S-BOX of CPRM systems [1]) that only Alice is authorized to see, and that must appear in *Specification* in a certain encoded form. P2 is a coding process in which Alice, Bob, and Chris participate. The three developers collaborate with each other and produce *Code* from *Specification*. Since Alice knows *Specification*, Alice may explain it to Bob and Chris in the collaboration. This

† Graduate School of Information Science, Nara Institute of Science and Technology

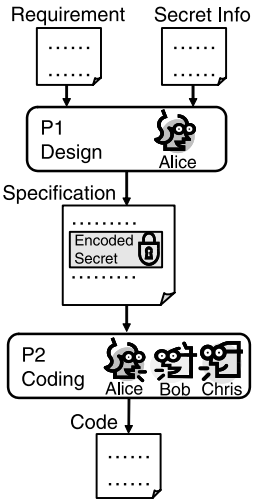


Fig. 1 Security-sensitive software process.

knowledge transfer is reasonable, since *Specification* is necessary for Bob and Chris to perform P2 together. Even if Alice does not give any explanation, Bob and Chris must know *Specification*. On the other hand, both *Requirement* and *SecretInfo* are *irrelevant* to P2, since they are not directly connected to P2. However, what happens if Alice tells Bob or Chris the knowledge about *SecretInfo* in P2? In this case, Bob or Chris gets to know *SecretInfo*, which ruins the security scheme.

From the above observation, we believe that in a security-sensitive software process, the knowledge transfer with any such irrelevant products should carry a warning such as *information leakage*. Note that the risk of information leakage varies, depending on the structure of the software process and the assignment of developers to each process. For example, in Fig. 1 if Alice is not assigned to P2, no leakage occurs.

The goal of this paper is to propose a framework to evaluate quantitatively the risk of information leakage for a given software process. To achieve this, we first formulate the problem of information leakage by introducing a formal software process model. The model is based on the conventional *process-centered software engineering environment*[5),7)]. Our contribution is to formulate *product knowledge* of each developer on top of the model, focusing on the process structure and the developer assignment.

Next, assuming that the knowledge of the irrelevant products can be transferred (i.e., leaked), we present a method to compute the probability of each developer knowing each product. The probability reflects the risk that someone leaked the product to the developer. We derive the probability from the given software process model using a recurrence formula.

To show applicability to practical or actual settings, we conduct three case studies. The first case study demonstrates how the information leakage varies depending on the assignment of developers. In the second case study, we present an application to find an optimal assignment of developers for a constrained software process. In the last case study, we investigate the relationship between the process structure and the information leakage. The proposed method provides a simple but powerful means to perform quantitative analysis on information leakage in a security-sensitive software process.

The rest of this paper is organized as follows: In Section 2, we give definitions of the software process model. Section 3 describes the proposed method for characterizing the dynamics of information leakage. In Section 4, we conduct the case studies. In Section 5, we have two discussions: setting probability of leakage among developers, and leakage in a deterministic manner. We also review the related work in the section. Finally, Section 6 concludes the paper and presents directions for future work.

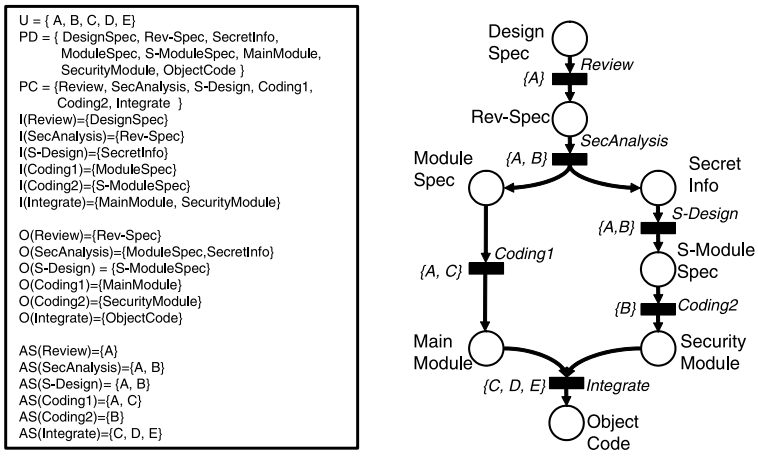
2. Preliminaries

2.1 Software Process Model

We start with a definition of a software process model. The model is based on the conventional *process-centered software engineering environment*[5),7)], where the software development is modeled by partially-ordered activities (*processes*) operating with given or intermediate *working products*. In addition to the conventional model, our model involves the *assignment of developers* to specify explicitly who participates in each process.

Definition 1 (Software Process Model) A software process model is defined by $P = (U, WP, PC, I, O, AS)$, where:

- U is a set of all *developers* participating in the development.
- WP is a set of all *work products*.
- PC is a set of all *processes*.
- I is an *input* function $PC \rightarrow 2^{WP}$ that maps each process $p \in PC$ onto a set $IP(\subseteq WP)$ of *input products* of p.
- O is an *output* function $PC \rightarrow 2^{WP}$ that maps each process $p \in PC$ onto a set $OP(\subseteq$



(a) Software Process Model (b) Petri-Net Representation

Fig. 2 Process model example.

WP) of *output products* of p.

- AS is an *developer assignment function* $PC \rightarrow 2^U$ that maps each process $p \in PC$ onto a set of developers participating in the process p.

Figure 2 (a) shows an example of the software process model, which simplifies an implementation stage of a security-sensitive application. The model contains five developers, eight work products, and six processes. The scenario assumed in the model is briefly explained as follows:

Example Scenario: The implementation stage produces an object code from a given design specification. In this stage, the design specification is revised by a review process. Next, by applying a security analysis to the reviewed specification, all security-sensitive information is isolated from the specification. The rest of the specification is called *ModuleSpec*. From the security information, authorized developers design a specification, called a *S-ModuleSpec* for an independent security module in which the raw security information is encoded. A main module and the security module are coded respectively from *ModuleSpec* and the *S-ModuleSpec*. Finally, these two modules are integrated as the object code.

In Fig. 2 (a), let us consider the process *Review*. This process models the review of the design specification. Review takes *DesignSpec* as an input product, and outputs a reviewed specification (*Rev-Spec*). In this example, only developer A is responsible for conducting the process. Next, we consider the process *SecAnalysis*. This process takes *Rev-Spec* as an input, and outputs *ModuleSpec* and *SecretInfo*. Two developers A and B participate in the process. Through a similar discussion, we can see how the process model achieves the example scenario.

By definition, each process has a set of input products and a set of output products. This definition allows us to draw a given process model by a *Petri net*[12], by associating WP with places, PC with transitions, and by connecting a place and a transition with an arc according to I and O. Figure 2 (b) shows a schematic representation of the example process with Petri net. Also, we associate a set of developers with each corresponding transition based on AS, as depicted in the left of the transition. Note that the use of the Petri net is just for better comprehension of the overview of the process structure, but is not essential to our methodology.

2.2 Order among Processes

Suppose that $P = (U, WP, PC, I, O, AS)$ is given. For $p \in PC$, $w \in I(p)$ and $w' \in O(p)$, we use a triple (w, p, w') to represent a *product dependency* of process p, where a work product w' is produced from w via p. The product dependencies implicitly specify a *partial order* between processes, since a process needs input products that have been previously generated by other processes.

Definition 2 (Order of Processes) For processes p and p', we say that p is *executed before* p' (denoted by $p < p'$) iff there exists a sequence $(w_0, p, w_1)\ (w_1, p_1, w_2)\ ...\ (w_{n-1}, p', w_n)$ of product dependencies. For processes q and

q', if any $<$ is not defined between q and q', we say that q and q' are *independent*.

Let us consider the previous example. As depicted in Fig. 2 (b), we can see the order among the six processes, i.e., *Review* $<$ *SecAnalysis* $<$ *Coding1* $<$ *Integrate*, and *Review* $<$ *SecAnalysis* $<$ *S-Design* $<$ *Coding2* $<$ *Integrate*. Note that the order is *partial* at this moment. Indeed, no order between *Coding1* and *S-Design* (or *Coding2*) is defined, thus they are independent. The independent processes can be executed in any order, even concurrently.

2.3 Assumption on Software Process Model

In this paper, we use the following two assumptions for a given process model $P = (U, WP, PC, I, O, AS)$.

Assumption A1: There exists no sequence $(w_0, p_0, w_1)\ (w_1, p_1, w_2)\ ...\ (w_{n-1}, p_n, w_n)$ of product dependencies such that $w_0 = w_n$.

Assumption A2: For any pair of independent processes p and p', if $AS(p) \cap AS(p') \neq \phi$, then an order between p and p' must be given.

Assumption A1 states that the product dependencies never form a loop. This is quite reasonable for general software processes. Indeed, it is unrealistic to assume that a work product newly obtained can be used as the input of the processes that have been completed previously. By this assumption, we have a consistent partial order among processes for a given sequence of product dependencies.

Assumption A2 states that independent processes p and p' must be ordered when the same developer is assigned to both p and p'. This assumption is based on the observation that a developer cannot engage in more than one process simultaneously. Let us consider the process model in Fig. 2. In this example, processes *Coding1* and *S-Design* are independent. However, they cannot be executed simultaneously, since the same developer A is assigned to both processes (i.e., $AS(\textit{S-Design}) \cap AS(Coding1) = \{A\}$). Hence, we need to give an order between these processes, for instance, *S-Design* $<$ *Coding1*, so that A conducts *S-Design* first.

By these assumptions, if we fix a developer u, then the processes in which u participates are totally-ordered.

Proposition 1 Let $P = (U, WP, PC, I, O, AS)$ be a given process model with Assumptions A1 and A2. For a developer $u \in U$, let $PC_u = \{p | p \in PC \wedge u \in AS(p)\}$ be a set of all processes to which u is assigned. Then, PC_u is *totally-ordered*.

Consider the process model in Fig. 2 with *S-Design* $<$ *Coding1*. Then, the processes to be conducted by each user are ordered as follows:

PC_A : *Review* $<$ *SecAnalysis* $<$ *S-Design* $<$ *Coding1*
PC_B : *SecAnalysis* $<$ *S-Design* $<$ *Coding2*
PC_C : *Coding1* $<$ *Integrate*
PC_D : *Integrate*
PC_E : *Integrate*

Since PC_u are totally-ordered, any process in PC_u has at most one *immediate predecessor*.

Definition 3 (Predecessor of Process) Let $p_{u_1}, p_{u_2}, ..., p_{u_k}$ be all processes in PC_u such that $p_{u_1} < p_{u_2} < ... < p_{u_k}$. For $p_{u_i} \in PC_u$, we call $p_{u_{i-1}}$ *immediate predecessor* of p_{u_i} with respect to u, which is denoted by $pred_u(p_{u_i})$. Also, we define $pred_u(p_{u_1})$ to be ϵ (empty).

In the above example, we have $pred_A(Coding1)$ = *S-design*, which means that A participates in *S-design* immediately before $Coding1$. Also, we have $pred_C(Coding1) = \epsilon$ meaning that $Coding1$ is the first process that C engages in.

3. Characterizing Dynamics of Information Leakage

3.1 Product Knowledge of Developers

To perform a process p, developers engaging in p must know all the input products of p. Based on the input products, they develop the output products. Hence, when finishing p, developers should be acquainted with the output products as well. Thus, when a process is performed, the developers acquire knowledge about the related (i.e., input/output) products. For each developer, the knowledge is accumulated in the sequence of completed processes. This dynamics depends on the given process model, specifically, I, O, and AS.

For example, consider the example in Fig. 2. Developer A participates in process *Review*. Hence, when *Review* is finished, A must know the products *DesignSpec* and *Rev-Spec*. Similarly, the completion of *SecAnalysis* provides the knowledge of *Rev-Spec*, *ModuleSpec*, and *SecretInfo* for both A and B. Thus, when A completes *SecAnalysis*, A knows four products; *DesignSpec, Rev-Spec, ModuleSpec, SecretInfo*.

Definition 4 (Product Knowledge) Let $P = (U, WP, PC, I, O, AS)$ be a given software process model. For $u \in U$ and $p \in PC$, we

Table 1 $Know(u, Integrate)$ $(u \in \{A, B, C, D, E\})$.

u	DSpc	RSpc	SInfo	MSpc	SSpc	MMo	SMo	OCd
A	1	1	1	1	1	1	0	0
B	0	1	1	1	1	0	1	0
C	0	0	0	1	0	1	1	1
D	0	0	0	0	0	1	1	1
E	0	0	0	0	0	1	1	1

define a set of working products $Know(u, p)$ $(\subseteq WP)$ s.t.

$$Know(u, p) = \bigcup_{u \in AS(p') \wedge p' \leq p} (I(p') \cup O(p'))$$

$Know(u, p)$ is called the *product knowledge* of developer u at the completion of process p. We use the term "knowledge" in an abstract sense, which can be refined in terms of, for instance, the essential idea or mechanism, the product's document itself, or the access method to the product.

Let us compute $Know(B, Coding2)$ with Fig. 2. Before *Coding2*, B has participated in *SecAnalysis* and *S-Design*. Hence, accumulating the input/output products of these three processes, we have $Know(B, Coding2)$ = { *Rev-Spec, SecretInfo, ModuleSpec, S-ModuleSpec, SecurityModule* }.

For convenience, we represent $Know(u, p)$ with a *binary vector*. Let $w_1, w_2, ..., w_n$ be all work products in WP. Then, we denote $Know(u, p) = [wp_1, wp_2, ..., wp_n]$, where $wp_i = 1$ iff $w_i \in Know(u, p)$, otherwise $wp_i = 0$. Then, the product knowledge of all users at the completion of the last process (i.e., *Integrate*) can be represented in **Table 1**.

3.2 Leakage of Product Knowledge

Now suppose a situation such that a developer may *share* his/her product knowledge to other developers sharing the same process.

As an example, consider *Coding1* in Fig. 2. This process is shared by A and C. Assuming an order *S-Design* < *Coding1*, the product knowledge of A and C at *Coding1* are computed as follows:

	DS	RS	SI	MS	SS	MM	SM	OC
Know(A,Coding1) = [	1	1	1	1	1	1	0	0]
Know(C,Coding1) = [	0	0	0	1	0	1	0	0]

Coding1 is the first process that C participates in. Hence, at this moment, C is supposed to know only *ModuleSpec* and *MainModule*. C does not need to know all the rest of the products. On the other hand, A has more product knowledge than C, because A has previously participated in three other processes.

Assume now that during *Coding1*, A tells C the product knowledge that C does not know, for example *SecretInfo*, with some probability. As a result, C learns *SecretInfo* although C has never directly touched it before. Once C knows *SecretInfo*, the knowledge would be *propagated* to D and E, since C shares the subsequent process, *Integrate*, with D and E. As a result, the isolation of security information would be in vain.

Thus, when multiple developers work in the same process, the product knowledge can be spread from the developer who knows the product to developers who do not know. We regard this as *information leakage in the software process*, which is specifically defined as follows.

Definition 5 (Leakage) For developers u, $u' \in D$, a work product $w \in WP$ and a process $p \in PC$, we say that u *may leak* w to u' at p iff $\{u, u'\} \subseteq AS(p)$ and both $w \in Know(u, p)$ and $w \notin Know(u', p)$.

The above definition of leakage might be a bit broad. Indeed, this definition covers a case such that a security product w is known to an unauthorized developer u'. On the other hand, someone may say that it is not leakage if w is not a security-sensitive product, or if u and u' work for the same company. However, for simplicity and generality of the model, we keep this broad definition. A more detailed criteria of the leakage should be tuned depending on the target software process.

3.3 Stochastic Product Knowledge

Now, let us take the leakage of product knowledge into account in our model. Specifically, we introduce the following assumption for a given process model $P = (U, WP, PC, I, O, AS)$:

Assumption A3: For $u, u' \in U$ and $w \in WP$, let $leak(u, w, u')$ be the probability that u leaks w to u'. We assume that $leak(u, w, u')$ is given for any u, u' and w.

Then, in a process p, a developer u may happen to know a product w such that $w \notin Know(u, p)$, since someone could leak w to u with a certain probability. This motivates us to deal with product knowledge in a *stochastic* manner.

Let us consider a probability that a developer u knows a work product w at the completion of process p, which we denote $Pkn(u, p, w)$. When u knows w at the completion of p, two cases can be considered.

Case C1: $w \in Know(u, p)$, or

Case C2: $w \notin Know(u, p)$ and some devel-

$$Pkn(u,p,w) = \begin{cases} 1.0 \quad (\cdots \textbf{if} \quad w \in Know(u,p)) \\ \\ Pkn(u, pred_u(p), w) + C(u,p) * (1 - Pkn(u, pred_u(p), w)) \\ *[1 - \prod_{u_i \in AS(p)-\{u\}} \{1 - Pkn(u_i, pred_{u_i}(p), w) * leak(u_i, w, u)\}] \\ \quad (\cdots \textbf{if} \quad w \notin Know(u,p)) \end{cases}$$

Fig. 3 Probability that a developer u knows a work product w at the completion of a process p.

opers leak (or leaked) w to u.

Case C1 means that w is already counted in u's product knowledge. For this case, we have $Pkn(u,p,w) = 1.0$. Case C2 can be further divided into two sub-cases.

Case C2a: u knew w before p (via someone else), or

Case C2b: $[u \in AS(p)]$ and [u did not know w before p] and [in p some developers sharing p with u leak w to u].

The probability that Case C2a holds is

$$P(C2a) = Pkn(u, pred_u(p), w)$$

which means that u knew w in the predecessor process. Next, the probability for Case C2b can be formulated by

$$P(C2b) = C(u,p) * \ (1 - Pkn(u, pred_u(p), w)) * \ P_{leak}$$

where $C : U \times PC \rightarrow \{0,1\}$ such that $C(u,p) = 1$ iff $u \in AS(p)$; otherwise $C(u,p) = 0$, and P_{leak} is the probability that some developers sharing p leak w to u.

Next, we formulate P_{leak}. Let $u_1, u_2, ..., u_j$ be developers who share p with u (i.e., $\{u_1, u_2, ..., u_j\} = AS(p) - \{u\}$). In order for u_i to leak w in p, two conditions are required: (1) u_i needs to have known w before p, and (2) u_i leaks w to u. Therefore, the probability that u_i leaks w to u in p is

$$Pkn(u_i, pred_{u_i}(p), w) * leak(u_i, w, u)$$

Moreover, u knows w iff at least one of $u_1, u_2, ..., u_j$ leaks w to u in p, which is the complement of "none of $u_1, u_2, ..., u_j$ leaks w to u in p". Hence,

$$P_{leak} = 1 - \prod_{u_i \in AS(p)-\{u\}} \{1 - Pkn(u_i, pred_{u_i}(p), w) * \ leak(u_i, w, u)\}$$

Combining all formulas together, we finally derive $Pkn(u,p,w)$, which is the probability that u knows w at the completion of p, in **Fig. 3**.

Note that $Pkn(u,p,w)$ is specified as a recurrence formula with respect to the process p. According to Assumptions A1 and A2, the set of processes that u participates in is totally-ordered. Hence, $pred_u(p)$ is uniquely obtained. Also, by Assumption A3, $leak(u_i, w, u)$ is given. Therefore, the value of $Pkn(u,p,w)$ can be calculated deterministically.

$Pkn(u,p,w)$ is now defined as *stochastic product knowledge*.

Definition 6 (Stochastic Product Knowledge) Let $P = (U, WP, PC, I, O, AS)$ be a given software process model with Assumptions A1, A2 and A3. Let $w_1, w_2, ..., w_n$ be all work products in WP. For $u \in U$, $p \in PC$, we define a vector $PKnow(u,p)$ s.t.

$$PKnow(u,p) = [Pkn(u,p,w_1), Pkn(u,p,w_2), ..., Pkn(u,p,w_n)]$$

$PKnow(u,p)$ is called *stochastic product knowledge* of u at the completion of p.

Consider the example in Fig. 2 with *S-Design* $<$ *Coding1*. For the sake of simplicity, let us assume a fixed probability $leak(u,w,u') = 0.01$ for all $u, u' \in U$ and $w \in WP$. Therefore, the stochastic product knowledge of all users at the completion of the last process (i.e., *Integrate*) can be obtained as shown in **Table 2**.

Table 2 $PKnow(u, Integrate)$ $(u \in \{A,B,C,D,E\})$.

u	DSpc	RSpc	SInfo	MSpc	SSpc	MMo	SMo	OCd
A	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0
B	0.0199	1.0	1.0	1.0	1.0	0.0	1.0	0.0
C	0.01	0.01	0.01	1.0	0.01	1.0	1.0	1.0
D	0.0001	0.0001	0.0001	0.01	0.0001	1.0	1.0	1.0
E	0.0001	0.0001	0.0001	0.01	0.0001	1.0	1.0	1.0

4. Case Studies

To show the applicability of the proposed method to practical software processes, this section conducts three case studies.

We have implemented a software tool which automatically derives the stochastic product knowledge. The tool is written in C++, comprising about 600 lines of codes. The tool computes the stochastic product knowledge from a given text file, which includes software process model and values of $leak(u,w,u')$ in a text for-

mat. As for the performance, the tool required 0.07 seconds to compute the result for our example process shown in Fig. 2, on a Pentium 4 PC (3.60 GHz).

4.1 Case Study 1: Impact of Collaboration among Developers

The aim of this case study is to demonstrate how the *collaboration* among developers influences the *risk* of information leakage. Here, we further investigate the software process model shown in Section 2.1 (see Fig. 2). We compute the stochastic product knowledge with varying AS on some processes.

Let us recall the scenario of the process model. In the scenario, a work product *SecretInfo* is assumed to be confidential. We also suppose that only developers A and B are authorized to access *SecretInfo*. When *S-Design* is completed, A and B are the only developers that know *SecretInfo*.

Our interest is to evaluate the risk that *SecretInfo* is leaked to unauthorized developers C, D or E. For this evaluation, we vary the developers assignment AS in the subsequent three processes *Coding1*, *Coding2* and *Integrate*.

For convenience, we first define the following parameters:

- $U_{aut} = \{A, B\}$: authorized developers.
- $U_{uaut} = \{C, D, E\}$: unauthorized developers.
- $PC_{tgt} = \{Coding1, Coding2, Integrate\}$: target processes where the developers assignment is varied.

The risk of the leakage depends heavily on how the authorized developers (A, B) collaborate with the unauthorized ones (C, D, E) in the target processes (PC_{tgt}). To characterize the collaboration, we define the following parameter for a process p:

$$Col(p) = |U_{aut} \cap AS(p)| * |U_{uaut} \cap AS(p)|$$

Also, we define

$$Col = \sum_{p \in PC_{tgt}} Col(p)$$

$Col(p)$ represents the number of combinations of authorized and unauthorized developers in a process p. This intuitively characterizes the *degree of collaboration* where an authorized developer *interacts* with an unauthorized one in p. For example, if $AS(p) = \{A, B, C, D\}$ with $U_{aut} = \{A, B\}$, $U_{uaut} = \{C, D\}$, then $Col(p) = 4$, which implies that there are 4 patterns where an authorized A or B interacts with an unauthorized C or D. Col is the total number of such interactions in the target processes. Hence, with the greater value of Col, the more the authorized developers can collaborate with the unauthorized ones.

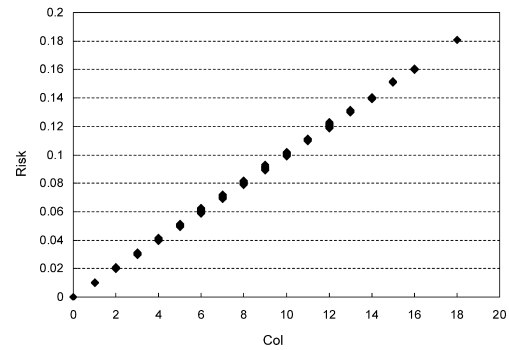


Fig. 4 Computation result of $Risk$.

For a fixed developers assignment $as = [AS(Coding1), AS(Coding2), AS(Integrate)]$, the risk that *SecretInfo* is leaked to unauthorized developers is formulated by:

$$Risk_{as} = \sum_{u \in U_{uaut}} Pkn(u, integrate, SecretInfo)$$

$Risk_{as}$ is characterized as the expected number of unauthorized developers knowing $SecretInfo$.

Using the developed tool, we have computed $Risk_{as}$ for all possible assignments $as \in 2^U \times 2^U \times 2^U$. In the computation, we assume a fixed probability $leak(u, w, u') = 0.01$ for every $u, u' \in U$ and $w \in WP$.

Figure 4 depicts the result. In this scattered plot, the horizontal axis represents Col, while the vertical axis plots $Risk_{as}$. **Table 3** shows the average value of $Risk_{as}$ with respect to Col [☆].

As seen in the result, the risk that *SecretInfo* is leaked grows as Col increases. This increase implies that more collaboration among authorized and unauthorized developers causes the higher risk of information leakage. In this case study, each probability that a developer u leaks a product w to another u' is relatively small (i.e., $leak(u, w, u') = 0.01 = 1\%$). However, if the developers share many processes, the total probability of the leakage becomes significantly large. For $Col = 18$ where all of five developers are assigned to every target process, the risk

☆ Due to the structure of the given process model, there is no developer assignment such that $Col = 17$.

Table 3 Average of *Risk* w.r.t. *Col*.

Col	Risk
0	0.000000000
1	0.010040352
2	0.020082816
3	0.030094976
4	0.040112381
5	0.050125019
6	0.060119186
7	0.070156853
8	0.080101327
9	0.090217868
10	0.100071167
11	0.110321914
12	0.120011146
13	0.130566804
14	0.139962521
15	0.151329923
16	0.160040993
17	—
18	0.180650505

becomes as large as 18%.

4.2 Case Study 2: Optimal Developers Assignment

This case study demonstrates how the proposed method can be used to find an optimal assignment of developers. For a software process model (with certain constraints), we say that an assignment of developers AS is *optimal* if the risk of the leakage is minimized by AS.

In this case study, we use a software process model $P = (U, WP, PC, I, O, AS)$, where U, WP, PC, I and O are the same as those in Fig. 2 and AS is not determined yet. We also assume the same security scheme as the one in the previous case study. In general, a software process has certain constraints with respect to human resources, developing time, etc. As a typical example, we impose the following constraints on P:

- All processes in WP must be performed by the effort of the total 10 developers (with overlaps).
- At most two developers can conduct each process.
- Both *SecAnalysis* and *S-Design* must be done only by the authorized developers ($U_{aut} = \{A, B\}$).
- $leak(u, w, u') = 0.01$ for all $u, u' \in U$, and $w \in WP$.

With the above constraints, we compute the optimal assignment, where the risk that *SecretInfo* is leaked is minimized.

The computation is rather straightforward. For every possible assignment as that satisfies the constraints, we compute $Risk_{as}$ (see Case Study 1). The optimal assignment is shown in **Fig. 5** (a). In this assignment, there is no risk that *SecretInfo* is leaked (i.e., $Risk_{as} = 0.0000$). For a just comparison, we also compute the assignment where $Risk_{as}$ is maximized within the constraints. Figure 5 (b) shows one of such cases, where $Risk_{as}$ is as large as 0.0300).

We can see in the optimal assignment (Fig. 5 (a)) that after *S-design* there is no process involving authorized (A, B) and unauthorized developers (C, D, E), simultaneously. In contrast, as for the risky assignment (Fig. 5 (b)), in every process after *S-design* either A or B shares the process with an unauthorized developer.

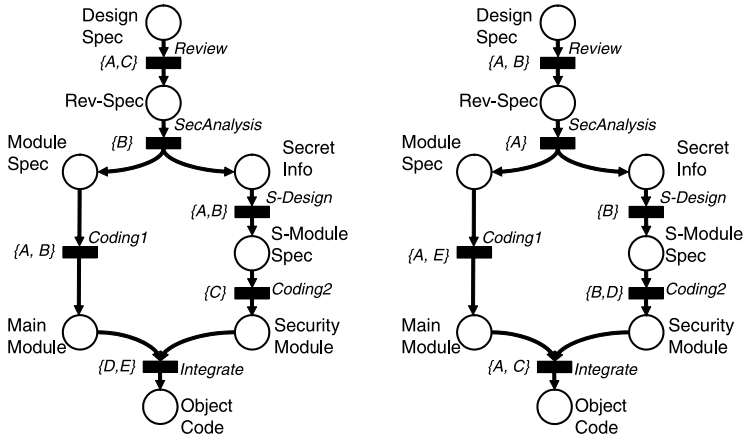
Thus, the proposed method can be also used as a powerful means to perform optimal tunings of the process configuration.

4.3 Case Study 3: Influence of Process Structure

The previous two case studies showed that the assignment of developers to each process is an essential factor for controlling the risk of the information leakage. Our next interest is to examine the influence of the *process structure* (i.e., the shape of the Petri Net, intuitively) on information leakage.

For this, we introduce two software process models $S1$ and $S2$ shown in **Fig. 6**. The scenario of $S1$ is summarized as follows:

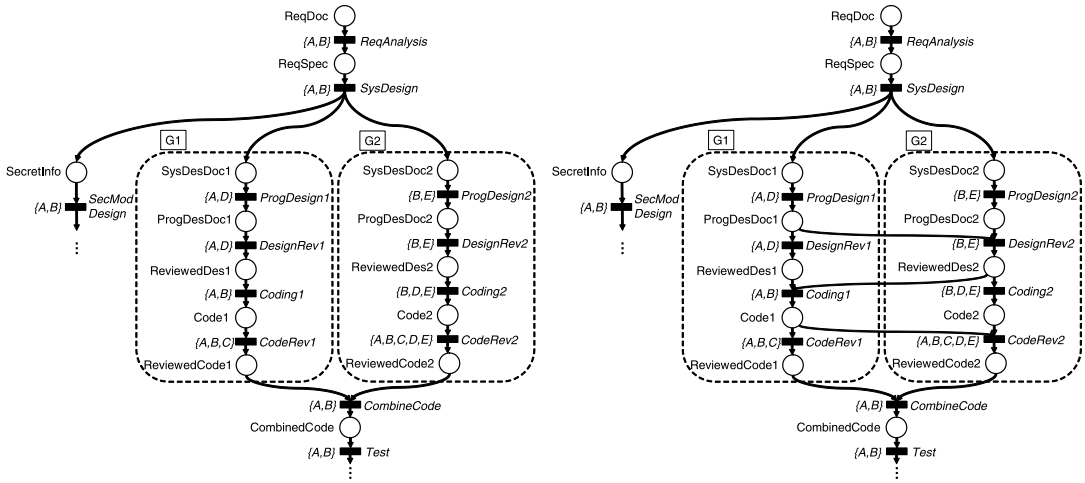
- Five developers A, B, C, D, and E participate in the software process.
- A and B first make the requirement specification (*ReqSpec*) from the given requirement document (*ReqDoc*) by the requirement analysis (*ReqAnalysis*).
- A and B conduct the system design (*SysDesign*) to divide the whole system into three sub-systems: a security module and two sub-modules.
- In *SysDesign*, A and B carefully isolate the confidential information (*SecretInfo*) from the rest of the system, to design the security module. For this isolation, we assume that only A and B are authorized to access *SecretInfo*. That is, we have $U_{aut} = \{A, B\}$ and $U_{uaut} = \{C, D, E\}$.
- The two sub-modules are developed in two concurrent processes $G1$ and $G2$ (shown as dotted boxes in Fig. 6), each of which consists of four processes (program design, design review, coding, and code review).
- The reviewed codes of the two sub-modules are combined into one. The resultant code



(a) Assignment that $Risk_{as}$ is ($Risk_{as} = 0.0000$)

(b) Assignment that $Risk_{as}$ is maximized ($Risk_{as} = 0.0300$)

Fig. 5 Optimal assignment and risky assignment.



(a) Structure S1

(b) Structure S2

Fig. 6 Target software process model.

is applied to the subsequent test process. Model $S2$ is almost the same as $S1$, but is somewhat ill structured. $S1$ and $S2$ have the same sets of products and processes. The developers assignment for $S1$ is also equal to the one for $S2$. However, for $S2$, some processes in G1 require some products in G2 (vice versa), which is represented by the three extra arcs between G1 and G2. These arcs suppress the execution order of some processes. For instance, *DesignRev2* can be performed only after *ProgDesign1* is completed.

For both $S1$ and $S2$, authorized developers A and B share some processes with the unauthorized ones C, D and E. Therefore, the knowledge of *SecretInfo* may leak. Our interest is to see how the structural difference between $S1$ and $S2$ (i.e., the three extra arcs) impacts the risk of the leakage. As in a similar discussion in the previous case studies, the risk is formulated by:

$$Risk_{str} = \sum_{u \in \{C,D,E\}} Pkn(u, Test, SecretInfo)$$

Figure 7 illustrates the result, where the horizontal axis represents the name of the model, while the vertical axis plots $Risk_{str}$ for all possible execution orders of the processes. As seen in the result, $S2$ tends to have a higher risk of information leakage.

According to Assumption $A2$, we have to

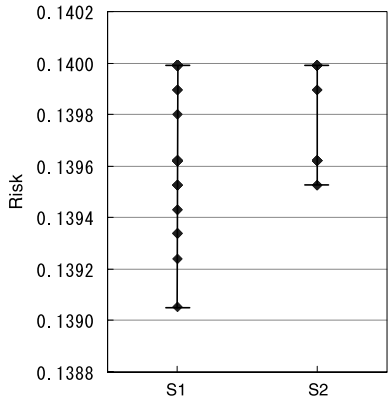


Fig. 7 Influence of process structure.

give an order for between any independent processes. A simple solution is to give an order so that $Risk_{str}$ is minimized. For $S1$, the optimal order of processes is:

$ProgDesign2 < DesignRev2 < Coding2 < CodeRev2 < ProgDesign1 < DesignRev1 < Coding1 < CodeRev1$
where $Risk_{str} = 0.139052$. On the other hand, for $S2$, the optimal execution order of processes is:

$ProgDesign1 < ProgDesign2 < DesignRev2 < Coding2 < DesignRev1 < Coding1 < CodeRev2 < CodeRev1$
where $Risk_{str} = 0.139524$.

In $S1$, each process of $G1$ is completely independent of another process in $G2$. Therefore, the processes in $G1$ can be executed without concern for the progress of $G2$. On the other hand, for $S2$, the extra arcs suppress some execution order of the processes (for example, we cannot assume the order $Coding1 < DesignRev2$). Therefore, $S2$ cannot take a wide range of execution orders, which results in a higher risk of leakage in this experiment.

Thus, the structure of the process controls the execution order of processes, which significantly influences the risk of information leakage.

5. Discussion

5.1 Setting Value of $leak(u, w, u')$

The proposed framework requires the user to give absolute probability $leak(u, w, u')$ for every u, w and u'. Although $leak(u, w, u')$ is assumed to be given (see Assumption A3), we here discuss the idea how to determine the value in a practical setting.

By definition, the value $leak(u, w, u')$ characterizes the probability that a developer u leaks the product knowledge w to another developer u'. In reality, since this action of the leakage involves many *human factors*, it would be difficult to estimate an *exact* value of $leak(u, w, u')$ for each individual developer.

However, as we demonstrated in Section 4, even if the user simply determines a *uniform value* of $leak(u, w, u')$ for all u, w and u', the user can analyze extensively the security aspects of the given software process. In this case, the user assumes that all developers are *equally likely* to leak their own product knowledge. Since varying the uniform value is easy, we believe that this type of analysis is quite useful to examine the process structure and developer assignment in the process planning stage.

On the other hand, if the user desires to estimate a more *realistic* value of $leak(u, w, u')$ for each individual developer, we consider that the user should refer to the *profile* information of developers, products, and working environment, which are supposed to be available in the organization. The profile information is used to derive objective attributes for each developer and product, involving; the age, the position, working experience, and security awareness of the developer, as well as organizational policies for confidential products and the trust of companies in collaboration. Based on the derived attributes, the user would be able to estimate $leak(u, w, u')$ in a more credible way.

A practically reasonable solution would be to introduce a *multi-grade* system with respect to the risk of leakage. For instance, the user evaluates each developer according to three-grade system: dangerous, moderate, safe. Then, the user assigns 0.1, 0.01 or 0.001 to $leak(u, w, u')$ for the dangerous, moderate or safe developer u. Applying these settings to the proposed framework, the user can simulate more realistic situation of information leakage. More sophisticated methods and their evaluation are left as a challenging issue in our future work.

5.2 Deterministic Model

We have so far characterized the information leakage with the stochastic model. However, we might able to consider the leakage in a *deterministic* manner, assuming that any *potential* leakage actually occurs. Such a deterministic model could be used to determine the *safest* way to avoid the leakage.

Indeed, this deterministic model can be constructed within the proposed framework by setting every parameter $leak(u, w, u')$ to be either

0.0 or 1.0.

The deterministic model has the great advantage of simplicity in determining the value of $leak(u, w, u')$, which can be used to analyze some *special* cases. For example, suppose that a software process in a company X is performed by the collaboration with another external company Y. For every pair of developers x and y from X and Y, respectively, we can construct a deterministic model, assuming that $leak(x, w, y) = 1.0$, and that no leakage occurs between developers within the same company. Then, the model can compute the set of product knowledge that could be transfered (or leaked) from the company X to Y.

Note, however, that the deterministic model can capture only some special situations according to the "always or never" basis. In the practical software process, we believe that *every* developer has a possibility of leaking his product knowledge. The deterministic model cannot deal with the *degree* of the potential leakage that is significantly characterized by the process structure and the developer assignment. In the above example, as long as x and y shares at least a certain process, the deterministic model always concludes that the leakage occurs. This conclusion is just by the worst-case analysis, and is *independent* of the process structure and the number of collaborations among x and y. Thus, the deterministic model tends to omit the detailed characteristics of the given software process model itself.

On the other hand, based on the assumption that every developer has a potential of leakage, the proposed stochastic model can quantitatively derive the degree of leakage of product knowledge, taking both the process structure and the developer assignment into account, as illustrated in Section 4. Although the stochastic model has a difficulty in justification of the probability setting on $leak(x, w, y)$, it is expected to provide a reasonable and useful metric for many process improvement tasks, such as constructing optimal software process under a constraint, and examining differences among multiple software process models.

Finally, note again that the proposed framework can deal with both the deterministic and stochastic models. So, the user can choose either model at his discretion.

5.3 Related Work

To our knowledge, no research study on a software process involving the leakage of product knowledge from one person to another exists. Chou, et al. [2] presented a model for access control named *WfACL*, which aims to prevent information leakage within work flows that may execute among competing organizations. Chou, et al. address issues related to the management of dynamic role change and access control. However, the model includes no concrete method to *evaluate* the risk of leakage quantitatively.

A numerical approach to compute information leakage might be to use *Generalized Stochastic Petri Net* (*GSPN*) [12] extensively. We first examined this approach. To do this, however, both the structure of process and the dynamics of leakage must be modeled in one GSPN. This complicates the net structure, and the state space becomes so large that the GSPN solver cannot compute the probability within a reasonable time. Therefore, we decided to treat the process description and the leakage computation separately.

In addition, much research has been focused on different kinds of access control methods, such as *role-based access control* [6),15], and *task-based access control* [16]. The goal of access control is to ensure that only authorized people are given access to certain resources (i.e., products in our problem). However, the aim of the proposed method is not to control the access authority, but to evaluate the risk of leakage as unexpected knowledge transfer among developers.

6. Conclusion

In this paper, we have presented a method to evaluate the risk of information leakage in the software development process. We formulated the leakage as an unexpected transfer of product knowledge among developers sharing the same process. We then proposed a method to derive the probability that each developer will know each work product at any process of software development. We also conducted three case studies. The result of the first case study quantitatively showed that; more collaboration among authorized and unauthorized developers causes a higher risk of information leakage. In the second case study, we showed that the proposed method can be also used as a powerful means to perform optimal tunings of the process configuration. Finally, in the third case study, we showed that the structure of the process controls the execution order of processes,

which in turn, significantly influences the risk of information leakage.

In addition, we discussed how to determine the value $leak(u, w, u')$, which characterizes the probability that a developer u leaks the product knowledge w to another developer u', in a practical setting. More sophisticated methods for calculating $leak(u, w, u')$ are left as a challenging issue in our future work. We also discussed the information leakage with a deterministic model, which could be used to determine the safest way to avoid the leakage.

The proposed method is simple and generic; therefore, it should not be limited to the security-sensitive software process. We expect that the proposed method is highly feasible for other workflow-based applications, such as *medical work flows*[14] where private information must be protected. Investigation of the emerging application domain is also an interesting issue for future research.

References

1) 4C-Entity: *Policy statement on use of content protection for recordable media, (CPRM) in certain applications* (2001). (Available online August 2001).
2) Chou, S.-C., Liu, A.-F. and Wu, C.-J.: Preventing information leakage within workflows that execute among competing organizations, *The Journal of Systems and Software* (2004). (Available online 4 February 2004).
3) Chow, S., Eisen, P., Johnson, H. and van Oorschot, P.: A white-box DES implementation for DRM applications, *Proc. 2nd ACM Workshop on Digital Rights Management*, pp.1–15 (2002).
4) Conrante Tech News: Microsoft's code leakage. http://www.corante.com/openmind/archives/001884.php
5) Feiler, P.H. and Humphrey, W.S.: Software Process Development and Enactment: Concepts and Definitions, *Proc. 2nd International Conference on Software Process*, pp.28–40 (1993).
6) Ferraiolo, D. and Kuhn, R.: Role-Based Access Controls, *15th NIST-NCSC National Computer Security Conference*, pp.554–563 (1992).
7) Garg, P.K. and Jazayeri, M.: *Process-Centered Software Engineering Environments*, IEEE Computer Society Press (1995).
8) Jacobson, I., Booch, G. and Rumbaugh, J.: *The unified software development process*, Addison-Wesley Longman Publishing Co., Inc. (1999).
9) Keller, F., Tabeling, P., Apfelbacher, R., Grone, B., Knopfel, A., Kugel, R. and Schmidt, O.: Improving Knowledge Transfer at the Architectural Level: Concepts and Notations, *Proc. 2002 International Conference on Software Engineering Research and Practice* (2002).
10) Liong, Y.-L. and Dixit, S.: Digital Rights Management for the Mobile Internet, *Wireless Personal Communications*, Vol.29, No.1-2, pp.109–119 (2004).
11) Mainichi Shimbun: Firms struggling to plug customer information leaks (2004). Mainichi Shimbun, March 2.
12) Marsan, M.A., Balbo, G., Conte, G., Donatelli, S. and Franceschinis, G.: *Modelling with Generalized Stochastic Petri Nets*, John Wiley (1995).
13) Monthly Information Security: Database of Information Leakage Incidents. http://www.monthlysec.net/ (in Japanese).
14) Quaglini, S., Mossa, C., Fassino, C., Stefanelli, M., Cavallini, A. and Micieli, G.: *Guidelines-Based Workflow Systems*, Lecture Notes in Computer Science, Vol.1620/1999, pp.65–75, Springer-Verlag (1999).
15) Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E.: Role-Based Access Control Models, *IEEE Computer*, Vol.29, No.2, pp.38–47 (1996).
16) Thomas, R.K. and Sandhu, R.S.: Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Autorization Management, *Proc. IFIP Workshop on Database Security*, pp.166–181 (1997).

(Received December 1, 2004)
(Accepted June 9, 2005)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.1, pp.322–334.)

Yuichiro Kanzaki received his B.E. degree in computer and systems engineering from Kobe University, Japan in 2001, and the M.E. degree in information science from Nara Institute of Science and Technology, Japan in 2003. He is currently a doctoral student of the Graduate School of Information Science at Nara Institute of Science and Technology. His research interests include software protection, program comprehension and software process security. He is a student member of the IEEE and IEICE.

Hiroshi Igaki received the B.E. degree (2000) in Department of Electrical and Electronics Engineering from Kobe University, Japan, and the M.E. degree (2002) and D.E. degree (2005) in information science from Nara Institute of Science and Technology, Japan. He is currently a post-doctoral fellow of Graduate School of Information Science at Nara Institute of Science and Technology. His research interests include communication support in software development, web services and service-oriented architecture. He is a member of the IEEE and a member of the IEICE.

Masahide Nakamura received the B.E., M.E., and Ph.D. degree in information and computer science from Osaka University, Japan, in 1994, 1996, 1999, respectively. From 1999 to 2000, he has been a post-doctoral fellow in SITE at University of Ottawa, Canada. He joined Cybermedia Center at Osaka University from 2000 to 2002. He is currently Assistant Professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include the service-oriented computing, feature interaction problem in network services, software validation and verification, and software metrics and security. He is a member of the IEEE and a member of the IEICE.

Akito Monden received the B.E. degree (1994) in electrical engineering from Nagoya University, Japan, and the M.E. degree (1996) and D.E. degree (1998) in information science from Nara Institute of Science and Technology, Japan. He was honorary research fellow at the University of Auckland, New Zealand, from June 2003 to March 2004. He is currently Associate Professor of Graduate School of Information Science at Nara Institute of Science and Technology. His research interests include software security, software measurement, and human-computer interaction. He is a member of the IEEE, ACM, IEICE, JSSST and JSiSE.

Ken-ichi Matsumoto received the B.E., M.E., and Ph.D. degrees in information and computer science from Osaka University, Japan, in 1985, 1987, 1990, respectively. He is currently Professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include software measurement and software user process. He is a senior member of the IEEE, and a member of the ACM and IEICE.