ソフトウェア開発プロセスにおける情報漏えいのリスク評価

神崎 雄一郎　井垣 宏　中村 匡秀　門田 暁人　松本 健一

奈良先端科学技術大学院大学 情報科学研究科
630-0192 奈良県生駒市高山町 8916-5

{yuichi-k, hiro-iga, masa-n, akito-m, matumoto}@is.naist.jp

あらまし ソフトウェアプロセスにおいて，情報漏えいのリスクを評価するための方法を提案する．本論文では，入力されたワークプロダクトをもとに新たなワークプロダクトを生成するサブプロセスの集合をソフトウェアプロセスとみなす．また，実行中のサブプロセスとは無関係なワークプロダクトが，ある開発者から共同作業を行っている別の開発者に渡ることを，情報漏えい (information leakage) とする．まず，形式的なソフトウェアプロセスモデルの導入を通して情報漏えいの定式化を行う．その上で，任意の開発者が任意のワークプロダクトに関する知識を持つ確率を計算する方法を提案する．

Evaluating the Risk of Information Leakage in Software Process

Yuichiro Kanzaki　Hiroshi Igaki　Masahide Nakamura　Akito Monden
Ken'ichi Matsumoto

Graduate School of Information Science, Nara Institute of Science and Technology,
8916-5, Takayama, Ikoma, Nara, 630-0192 Japan

Abstract This paper presents a method to evaluate the risk of information leakage in software development process. A software process is modeled as a series of sub-processes, each of which produces new work products based on input products. Since a (sub-)process is conducted usually by multiple developers together, information of work products is shared among the developers. For this, a developer might tell others information of some products that are not related to the process. We regard the flow of such unrelated information in the shared process as "leakage".

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.

1 Introduction

Software development companies often work with other organizations to develop large-scale software. Cooperative developments, however, can cause some security problems, such as theft of source code, and leakage of private information contained in work products.

An important factor of the security problems is redundant access to work products during a development process. To control access to work products, there has been proposed a number of methods [1, 2]. Although the access control methods are of great value in reducing redundant accesses, we usually can not ensure enough security only by the methods since each information can be passed on from person to person.

We focus in this paper on the *person-to-person transmission* of information. We assume that when a developer works with other ones, there is a possibility that information that he/she knows is passed to the other. Figure 1 illustrates an example of passing information. When *SubDesign*1 is performed, Al-

ice may pass on $UseCase$ that she obtained when $RequirementAnalysis$ is performed, to Dan who is working together. If Dan receives $UseCase$, he may pass on it to Bob when $Coding$ is performed. Although both Bob and Dan don't handle $RequirementAnalysis$, they might obtain $UseCase$. As seen in this example, we suppose that information appeared in a process can be passed on to those who do not directly handle works that require the information. We regard the flow of such unrelated information in the shared process as "leakage".

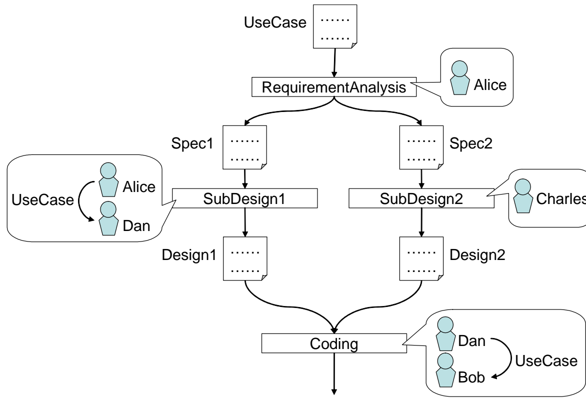


Figure 1: Passing Information

The goal of this paper is to propose a method that can be used to evaluate the risk of information leakage in software development process. 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 rest of this paper is organized as follows: In Section 2, we give definitions of software process model. Section 3 describes the proposed method for characterizing dynamics of information leakage. Finally, Section 4 concludes the paper and future work.

2 Preliminaries

2.1 Software Process Model

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

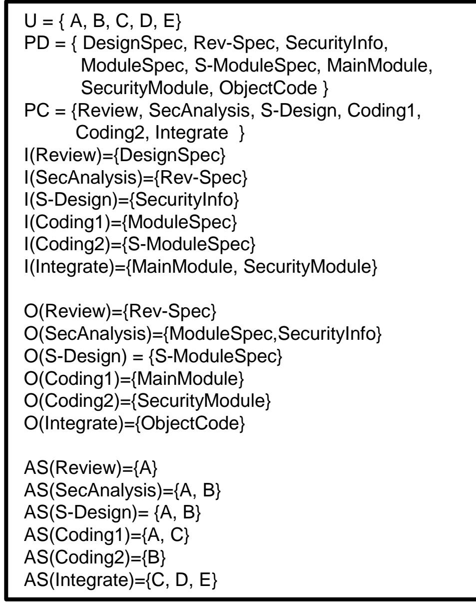
- U is a set of all *developers* participating the process.
- WP is a set of all *work products* developed in the process.
- PC is a set of all *(sub-)processes* performed in the process.
- I is an *input* function $PC \to 2^{WP}$ that maps each process $p \in PC$ onto a set $IP(\subseteq WP)$ of *input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

Figure 2(a) shows an example of software process model, which simplifies an implementation stage of a security software development. The whole process consists of five developers, eight work products, and six sub-processes. The scenario is briefly explained as follows:

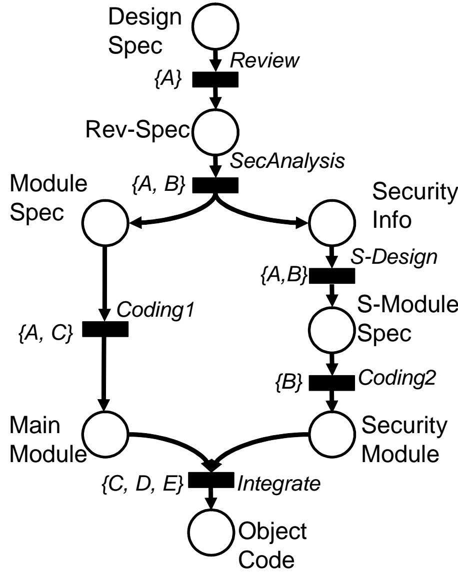
Example Scenario: The process produces an object code from a given design specification. The design specification is reviewed first. From the reviewed specification, the security information is separated for an independent security module. The rest of part is used to code a main module. The main module and the security module are finally integrated to obtain the object code.

In Figure 2(a), let us take the process *Review*. This models the review of the design specification. It takes *DesignSpec*) as an input product, and outputs the reviewed specification (*Rev-Spec*). In this example, only developer A is responsible to conduct the process. Next, consider the process *SecAnalysis*. This takes *Rev-Spec* as an input, and outputs *ModuleSpec* and *SecurityInfo*. Two developers A and B participate the process. By the similar discussion, we can see the process model achieves the example scenario.

By definition, each process has a set of input products and a set of output products. This allows us to draw a given process model by a Petri net, by associating WP with places, PC with transitions, 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, depicted in the left of the transition. Note that the use



(a) Software Process Model



(b) Schematic Representation

Figure 2: Process Model Example

of 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 by p from w. The product dependencies implicitly specify a *partial order* between processes, since a process needs input products that have been 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 Figure 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 put 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 that a work product newly obtained is 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 says that independent processes p and p' must be ordered in case that the same developer is assigned to both p and p'. This is based on the observation that a developer cannot engage in more than one processes simultaneously. Let us consider the process

model in Figure 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., $AS(S{-}Design) \cap AS(Coding1) = \{A\}$). Hence, we need to give an order between them, 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 Figure 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, they 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 Figure 2. Developer A participates process *Review*. Hence, when *Review* is finished, A must know products *DesignSpec* and *Rev-Spec*. Similarly, the completion of *SecAnalysis* provides the knowledge of *Rev-Spec*, *ModuleSpec* and *SecurityInfo* for both A and B. Thus, when A completes *SecAnalysis*, A knows four products; *DesignSpec, Rev-Spec, ModuleSpec, SecurityInfo*.

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 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 *product knowledge* of developer u at the completion of process p.

We use the term "knowledge" in some 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 Figure 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, SecurityInfo, 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.

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

u	DSpc	RSpc	SInfo	MSpc	SSpc	MMo	SMo	OCod
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

3.2 Leakage of Product Knowledge

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

As an example, consider *Coding1* in Figure 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:

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 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, say *SecurityInfo*, with some probability. Then, C becomes to know *SecurityInfo* although C has never directly touched it before. Once C knows *SecurityInfo*, 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 ones 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 $w \in Know(u,p)$ and $w \notin Know(u',p)$.

The above definition of leakage might be *broad* a bit. Indeed, it covers a case 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. 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 a 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 the 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 developers leak (or leaked) w to u.

Case C1 means that w is already count 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 has already known w before p, 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 $C : U \times PC \to \{0,1\}$ such that $C(u,p) = 1$ iff $u \in AS(p)$, otherwise $C(u,p) = 0$, and P_{leak} is a probability that some developers sharing p leaks 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

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

Now, u knows w iff at least one of $u_1, u_2, ..., u_j$ leaks w to u in p, which is the complement

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

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

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 together, we finally derive $Pkn(u,p,w)$, which is a probability that u knows w at the completion of p, in Figure 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 Figure 2 with *S-Design* < *Coding1*. For just simplicity, let us assume a fixed probability $leak(u,w,u') = 0.01$ for all $u, u' \in U$ and $w \in WP$. Then, 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	OCod
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	1.0	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 Conclusion

In this paper, we have presented a method to evaluate the risk of information leakage in software development process. We formulated the problem of information leakage by introducing a formal software process model, and we then proposed a method to derive the probability that each developer knows each work product at a given process of software development. We suppose that our method can be used for not only software development process, but also for many kinds of processes (e.g, business processes, medical processes) that have security-sensitive products.

Finally, we summarize our future work. We are going to implement a prototype system that automates the calculation of information leakage. Then we plan to conduct a quantitative case study to demonstrate how the information leakage varies depending on the assignment of developers.

References

[1] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[2] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Symposium on Operating Systems Principles*, pages 129–142, 1997.