情報漏えいのリスクと生産性を考慮した ソフトウェアプロセスの開発者割当て探索

神崎 雄一郎
熊本電波工業高等専門学校
情報工学科
kanzaki@ieee.org

井垣 宏
南山大学
数理情報学部 情報通信学科
igaki@acm.org

中村 匡秀
奈良先端科学技術大学院大学
情報科学研究科
masa-n@is.naist.jp

門田 暁人
奈良先端科学技術大学院大学
情報科学研究科
akito-m@is.naist.jp

松本 健一
奈良先端科学技術大学院大学
情報科学研究科
matumoto@is.naist.jp

要旨

ソフトウェア開発において，各作業 (プロセス) にどの開発者を割り当てるかは重要な問題である．本稿では，生産性と情報漏えいのリスクとを同時に考慮した開発者割当てを探索する新たな枠組みを提案する．まず，ソフトウェアプロセスを定義した後,「あるプロセスが実行されるとき，そのプロセスを共に行う開発者間で，プロダクトに関する知識が共有される」という仮定のもと，開発者間の知識伝達の機構を定式化する．さらに，信頼度の低い開発者に秘密プロダクトの知識が伝播することを，ソフトウェアプロセスにおける情報漏えいと定義し，そのリスクを見積もる方法を示す．ケーススタディでは，情報漏えいのリスクが低く，かつ，生産性が高い開発者割当てを探索する．

1. はじめに

複数の作業 (**プロセス**) によって構成されるソフトウェア開発を実施するときには，**開発者割当て**，すなわち，各プロセスにどの開発者を割り当てるかを決定する作業が必要となる．一般的に開発者割当ては，ソフトウェア開発が円滑に効率よく進み，決められた納期に間に合うように決定される [7].

しかし，開発するソフトウェアによっては，成果物 (**プロダクト**，開発途中で生成される中間成果物も含む) に秘密情報が含まれる場合がある．例えば，DRM システムの秘密鍵の定数 [3,11] が含まれたドキュメントや，商業的価値のあるアルゴリズムが含まれたソースコードなどを扱うソフトウェアプロセスの場合，プロダクト中の秘密情報が漏えいするリスクを十分考慮して開発者割当てを行うことが求められる [8, 9]．仮に，生産性のみを重視してあるプロセスに多人数の開発者を割り当てた場合，秘密情報を知りうる開発者が多くなるため情報漏えいのリスクが高くなる．すなわち，生産性と情報漏えいのリスクはトレードオフの関係にある．

情報漏えいのリスクを考慮して開発者割当てを行うためには，開発者間の**知識伝達** (knowledge transfer) を把握することが重要である．例を用いて説明する．図 1 は，Alice, Bob, Chiris, Dan の 4 人の開発者によって行われる，ソフトウェア開発 (の一部) を示す．この例では，仕様とソースコードからテストデータを作る「テスト設計」プロセス，テストデータをレビューする「テストレビュー」プロセス，レビューされたテストデータとバイナリコードを用いて行われる「テスト」プロセスがある．「テスト設計」は社内の人間である Alice が一人で行い，続く「テストレビュー」は社外の人間 Bob, Chiris も参加,「テスト」プロセスは，Bob, Chiris, Dan という完全に社外の人間が行うという開発者割当てを想定してい

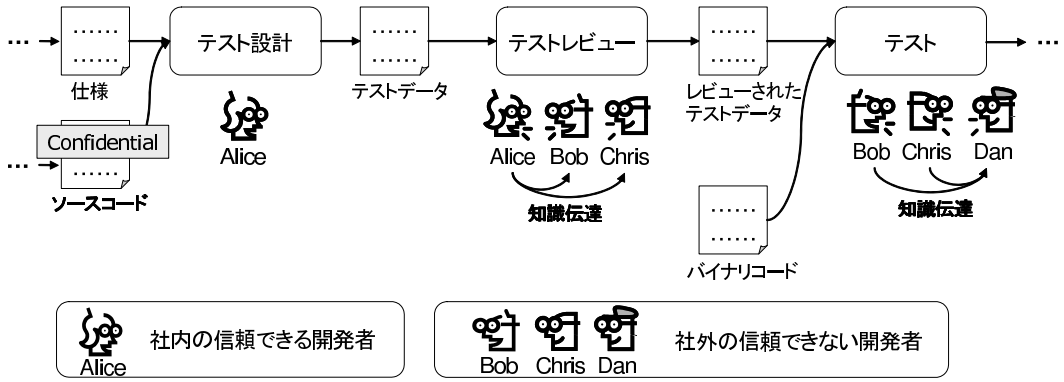


図 1. ソフトウェアプロセスの開発者と知識伝達

る．さらに，ここではソースコードが秘密情報を含むプロダクト (**秘密プロダクト**と呼ぶ) と仮定し，社内の人間である Alice のみが知る権利があるとする．

通常，複数人が参加するプロセス (図 1 では「テストレビュー」と「テスト」) では，ミーティング等を通して情報の交換が行われるのが一般的である [1].「テストレビュー」を実施する時点で，Alice は「テスト設計」を既に行っているので，仕様，ソースコード，テストデータに関する知識を持っている．したがって,「テストレビュー」時の情報交換において，Alice が仕様やテストデータに関する知識を Bob, Chiris に伝えるとで開発の生産性を上げることができる [1,10]．しかしながら，秘密プロダクトであるソースコードに関する知識まで伝えてしまうと，情報漏えいが発生してしまうことになる．この場合，さらに続く「テスト」で，Bob や Chris から Dan にソースコードに関する知識が伝達されてしまう可能性もある．このように，秘密プロダクトに直接アクセスする権利がなくても，作業実行時における開発者間の知識伝達によって，信頼度の低い開発者に秘密情報が伝わってしまうことがある．これが繰り返されると，結果的に大きな情報漏えいの問題に結びつく [8]．情報漏えいのリスクは，ソフトウェアプロセスの構造，および，開発者割当ての方法に大きく依存する．

本稿の目的は，与えられたソフトウェアプロセスに対して，生産性と情報漏えいのリスクとを同時に考慮した，最適な開発者割当てを導出する手法を提案することである．まず，2 章においてソフトウェアプロセスを定義する．次に，3 章で開発者間の知識伝達をモデル化し,「あるプロダクトに関する情報がある開発者に知られる確率」を計算する方法を導く．また，知識伝達の特殊なケースとして，情報漏えいの定義を行う．続く 4 章で情報漏えいのリスクと生産性が考慮された開発者割当て問題について整理し，5 章で，ケーススタディとして，最適な開発者割当ての探索の例を示す．最後に 6 章において，まとめと今後の課題について述べる．

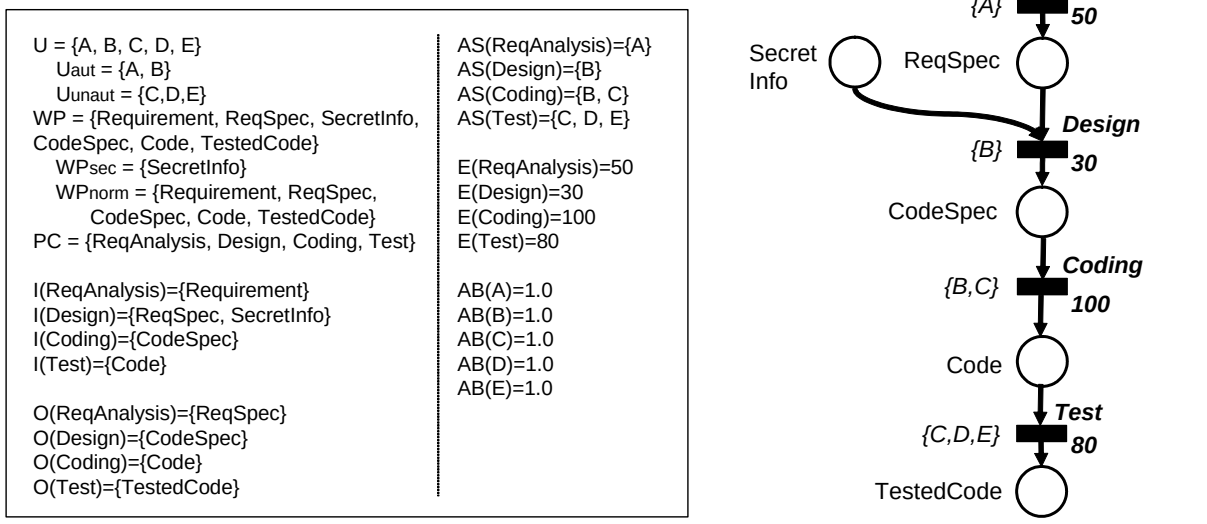
2. ソフトウェアプロセスモデル

2.1. 定義

一般にソフトウェア開発は，複数のプロセスから構成される．各プロセスはあるプロダクト (の集合) を入力とし，別のプロダクト (の集合) を生成する作業として捉えられる．本稿で採用するモデルは，従来のプロセス中心型ソフトウェア開発環境モデル (PSEE: Process-centered Software Engineering Environment) [4–6] を拡張し，開発者割当て，開発者の信頼度，秘密プロダクトに関する概念を陽に導入している．

定義 1 (ソフトウェアプロセスモデル) ソフトウェアプロセスは，$P = (U, WP, PC, I, O, AS, E, AB)$ で定義される．ここで，

- U は開発に参加しているすべての**開発者**の集合である．ここで $U = U_{aut} \cup U_{unaut}$，$U_{aut} \cap U_{unaut} = \phi$ であり，U_{aut} は信頼度の高い開発者の集合，U_{unaut} は信頼度の低い開発者の集合を示す．
- WP はすべての**プロダクト**の集合である．ここで $WP = WP_{sec} \cup WP_{norm}$，$WP_{sec} \cap WP_{norm} = \phi$ であり，WP_{sec} は秘密プロダクトの集合，WP_{norm} は通常プロダクトの集合を示す．



(a) ソフトウェアプロセスモデル　　(b) ペトリネット表現

図 2. ソフトウェアプロセスモデルの例

- PC はすべての**プロセス**の集合である．
- I は，各プロセス $p \in PC$ と p の**入力プロダクト**の集合 $IP(\subseteq WP)$ を対応付ける**入力関数** $(PC \to 2^{WP})$ である．
- O は，各プロセス $p \in PC$ と p の**出力プロダクト**の集合 $OP(\subseteq WP)$ を対応付ける**出力関数** $(PC \to 2^{WP})$ である．
- AS は，各プロセス $p \in PC$ と p に参加する開発者の集合を対応付ける**開発者割当て関数** $(PC \to 2^{U})$ である．
- E は，各プロセス $p \in PC$ と p の (見積もり) 工数 (単位:人時) を対応付ける**工数関数** $(PC \to N)$ である．ここで N は，自然数の集合を表す．
- AB は，各開発者 $u \in U$ と，u が単位時間あたりにこなせる工数 (単位:人時/時) を対応付ける**能力関数** $(U \to R^{+})$ である．ここで R^{+} は，正の実数の集合を表す．

図 2(a) に，ソフトウェアプロセスモデルの例を示す．この例は，秘密情報を含むアプリケーションを開発するためのソフトウェアプロセスで，5 人の開発者，6 つのプロダクト，および，4 つのプロセスを含んでいる．また，信頼度の高い開発者は A および B の 2 人で，秘密のプロダクトは *SecretInfo* である．このプロセスモデルにおけるシナリオの例を，簡単に示す．

シナリオの例: 要求仕様 *ReqSpec* からテストが完了したコード *TestedCode* を得ることを目的とする．まず，要求分析のプロセス *ReqAnalysis* により，要求 *Requirement* から要求仕様 *ReqSpec* が作成される．次に，設計のプロセス *Design* により，要求仕様 *ReqSpec* および秘密情報 *SecretInfo* からコード仕様 *CodeSpec* が得られる．続いて *Coding* プロセスにより *Code* が作成され，最後に *Test* プロセスによって *TestedCode* が得られる．

図 2(b) は，このシナリオの例を**ペトリネット** [12] で表現したものである．定義 1 における WP の 1 要素を 1 つのプレース，PC の 1 要素を 1 つのトランジションとみなし，I と O に応じてそれぞれを結んでいる．さらに，AS に応じて開発者の集合が各トランジションの左側に，E に応じて工数が各トランジションの右側に示されている．例えば，*Coding* のプロセスの見積もり工数は 100 人時 $(E(Coding) = 100)$ で，開発者は B, C が割当てられている $(AS(Coding) = \{B, C\})$.

2.2. ソフトウェア開発時間

プロセスモデル (開発者割当ても含む) が与えられたとき，各プロセスの実行時間，および，全プロセスの実行に必要な時間 (**ソフトウェア開発時間**) を以下のように定義する．

定義 2 (プロセス実行時間) プロセス p の実行に要する**プロセス実行時間** $t(p)$ は，次式で定義される．

$$t(p) = \frac{E(p)}{\sum_{u \in AS(p)} |AB(u)|}$$

p における**生産性**，すなわち，p に割り当てられた各開発者の開発能力の総和が大きくなるほど，$t(p)$ の値は小さくなる[1]．

ソフトウェア開発時間 T は，定義 2 とソフトウェアプロセスの構造により決定される．例えば，すべてのプロセスが直列に実行されるようなソフトウェアプロセス P の場合，次式のようにプロセス実行時間の単純な総和で表される．

$$T = \sum_{p \in PC} t(p)$$

図 2 の例では，$t(ReqAnalysis) = 50$[時]，$t(Design) = 30$[時]，$t(Coding) = 50$[時] および $t(Test) = 26.7$[時] となり，ソフトウェア開発時間 T は，各値の総和をとった 156.7[時] となる．

3. プロダクト知識の伝達と情報漏えい

3.1. 開発者のプロダクト知識

開発者はプロセスにおいて，入力プロダクトをもとに，出力プロダクトを作成する．プロセス p を実行するためには，p に割当てられた開発者は p の入力プロダクトについての知識が求められる．また，p を終えたとき，開発者は p の出力プロダクトについても知ることになる．

したがって，あるプロセスが実行されるとき，開発者はプロセスに関連するプロダクトの知識を獲得する．この知識は，与えられる I，O および AS に依存する．

[1]開発者数を増加させても，プロセスの生産性が必ずしも加算的に上昇しない事例が報告されている [2]．しかし，ここではモデルが複雑になりすぎることを避けるため，開発者数が多いほど生産性が上昇し，開発に要する時間が短くなるという考えに基づく．

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

u	Req	ReqSpec	SecInfo	CdSpec	Code	TCode
A	1	1	0	0	0	0
B	0	1	1	1	1	0
C	0	0	0	1	1	1
D	0	0	0	0	1	1
E	0	0	0	0	1	1

例えば，図 2 において，開発者 A は *ReqAnalysis* に割当てられている．したがって，*ReqAnalysis* が実行されるとき，A は *Requirement* と *ReqSpec* に関する知識を得る．続く *Design* が B によって行われたとき，B は *ReqSpec*，*SecretInfo* および *CodeSpec* に関する知識を得る．

定義 3 (プロダクト知識) $P = (U, WP, PC, I, O, AS, E, AB)$ を与えられたソフトウェアプロセスモデルとする．それぞれの $u \in U$ および $p \in PC$ について，次に示すようなプロダクトの集合 $Know(u, p)(\subseteq WP)$ を定義する．

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

ここで，$\leq$ はプロセスの順序関係であり，上では p の前に行われたすべてのプロセス p' に関する集合和を計算している．$Know(u, p)$ をプロセス p 実行時における，開発者 u の**プロダクト知識**と呼ぶ．プロダクト知識とは，プロダクトに関するアイデアやメカニズム，アクセス方法やプロダクトそのものを指す概念として考える．

図 2 の例において，$Know(B, Coding)$ を計算してみる．*Coding* が行われる前に，B は *Design* に参加している．したがって，B はこの 2 つのプロセスに対する入力プロダクトおよび出力プロダクトに関する知識を得ることになり，$Know(B, Coding) = \{ReqSpec, SecretInfo, CodeSpec, Code\}$ となる．便宜上，$Know(u, p)$ を二値 (1 か 0) のベクトルとして考える．$w_1, w_2, ..., w_n$ を，WP のすべてのプロダクトとすると，$Know(u, p) = [wp_1, wp_2, ..., wp_n]$ となる．ただし，$w_i \in Know(u, p)$ の場合にのみ $wp_i = 1$，それ以外の場合 $wp_i = 0$ である．最後のプロセスである *Test* が実行された後における，すべての開発者のプロダクト知識を表 1 に示す．

3.2. プロダクト知識の伝達

続いて，同じプロセスに割当てられた開発者どうしが，プロダクト知識を共有する状況について考える．例えば，図 2 の *Coding* では，B と C が割当てられている．*Coding* が実行された後の B と C のプロダクト知識は次のようになる．

$$\begin{array}{rcl} & & \begin{array}{ccccccc} \text{Rq} & \text{RqS} & \text{SI} & \text{CdS} & \text{Cd} & \text{TCd} & \end{array} \\ Know(B, Coding) & = & [\ \ 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0\ \ ] \\ Know(C, Coding) & = & [\ \ 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0\ \ ] \end{array}$$

Coding は C が参加する最初のプロセスである．したがって，*Coding* の実行を終えた時点では，C は *CodeSpec* と *Code* に関する知識のみ持っている．一方，以前に *Design* に参加した B は，C よりも多くの知識を持つ．

ただし，*Coding* が実行されたときに B が C に，C の知らないプロダクト知識 (たとえば *SecretInfo*) を伝達した場合，C は以前に直接接する機会のなかった *SecretInfo* に関する知識を持つことになる．一度 C が *SecretInfo* についての知識を持つと，C が D および E と共同で行うプロセス *Test* において，*SecretInfo* の知識が C から D や E に伝播されるかもしれない．

このように，複数の開発者が同じプロセスで作業するとき，プロダクト知識は，プロダクトを知る開発者から知らない開発者へ拡散する可能性がある．この知識の波及を，**知識伝達**としてとらえ，具体的に次にように定義する．

定義 4 (プロダクト知識の伝達) 各々の開発者 $u,\ u' \in U$，プロダクト $w \in WP$，プロセス $p \in PC$ について，$\{u, u'\} \subseteq AS(p)$，かつ，$w \in Know(u, p)$ と $w \notin Know(u', p)$ が両方成り立つとき，w に関する知識は，p において u から u' へ伝達する可能性がある．その伝達を $KT(p, u, w, u')$ と表す．

3.3. 知識所有確率

定義上，知識伝達の対象となるプロダクトは，元来当該プロセスに関係ない (入力でも出力でもない) プロダクトであり，当該プロセスの参加者は必ずしも全員がそれらを知らなくてもプロセスの実行に支障はない．したがって，知識伝達は確率的に発生すると考えることが自然である．そこで，定義 4 に基づき，確率的なプロダクト知識を定義することを考える．まず，プロセスモデル $P = (U, WP, PC, I, O, AS, E, AB)$ に次のような仮定を導入する．

仮定 A1: 各 $u, u' \in U$，$w \in WP$ について，u と u' があるプロセス $p \in PC$ を共有した時，ある確率 $trans(u, w, u')$ で $KT(p, u, w, u')$ 発生する．ここで，$trans(u, w, u')$ は与えられるものと仮定する．

この仮定のもと，プロセス p の終了時において開発者 u がプロダクト w について知っている確率を $Pkn(u, p, w)$ として，$Pkn(u, p, w)$ を計算する方法について考える．

p の終了時に u が w について知っているとき，次の 2 つのケースが考えられえる．

Case C1: $w \in Know(u, p)$，あるいは，

Case C2: $w \notin Know(u, p)$，かつ，他の開発者が w を u に伝達する (あるいは以前に伝達した).

Case C1 は，w がすでに u のプロダクト知識に含まれていることを意味する．この場合，$Pkn(u, p, w) = 1.0$ と考える．Case C2 は，さらに次の 2 通りに分割される．

Case C2a: u は p が実行される前に (ある別の開発者を通して) w を知っていた．あるいは，

Case C2b: $[u \in AS(p)]$，かつ，[p 以前に u が w を知らなかった]，かつ，[p において，u とともに p に割当てられたある開発者が w に関する知識を u に伝達した].

Case C2a が起きる確率は，

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

ここで，$pred_u(p)$ は，p の直前に行われた u の参加するプロセスである．すなわち，「u が w を直前のプロセス ($pred_u(p)$) で知っていた」可能性となる．

Case C2b が起きる可能性は，次にように定式化される．

$$\begin{aligned} P(C2b) \quad = \quad & C(u, p) * \ (1 - Pkn(u, pred_u(p) \\ & , w)) * \ P_{trans} \end{aligned}$$

ここで，$C : U \times PC \to \{0, 1\}$ は，$u \in AS(p)$ の場合にのみ $C(u, p) = 1$，そうでなければ $C(u, p) = 0$ となる関数である．また，P_{trans} は，p を実行するある開発者が u に w の知識を伝達する確率である．

次に，P_{trans} を定式化する．$u_1, u_2, ..., u_j$ を，u とともに p を実行する開発者とする．(すなわち，

$\{u_1, u_2, ..., u_j\} = AS(p) - \{u\}$)．$u_i$ が p において w を伝達するには，次の 2 つの条件が求められる．(1) p 以前に u_i は w を知っている，かつ，(2) u_i が w を u に伝達する．ゆえに，u_i が p において w を伝達する確率は，仮定 A1 より，

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

となる．さらに，少なくとも $u_1, u_2, ..., u_j$ のうちの 1 人が p において w を u に伝達するとき，u が w を知ることになる．これが生じる確率は,「p において，$u_1, u_2, ..., u_j$ が誰も w を u に伝達しない」の余事象の確率となるので,

$$\begin{aligned} P_{trans} &= 1 - \prod_{u_i \in AS(p)-\{u\}} \{1 - Pkn(u_i, \\ &\quad pred_{u_i}(p), w) * trans(u_i, w, u)\} \end{aligned}$$

となる．

すべての式を組み合わせると，p において u が w を知る確率 $Pkn(u, p, w)$ は最終的に次のようになる．

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

ここで $Pkn(u, p, w)$ を**確率的プロダクト知識**として，次のように定義する．

定義 5 (確率的プロダクト知識) $P = (\ U, WP, PC, I, O, AS, E, AB)$ を，仮定 A1 を満たすソフトウェアプロセスモデルとする．

$w_1, w_2, ..., w_n$ を，WP に含まれるすべてのプロダクトとする．それぞれの $u \in U$，$p \in PC$ について，次のようなベクトル $PKnow(u, p)$ を定義する．

$$\begin{aligned} PKnow(u,p) &= [Pkn(u,p,w_1), Pkn(u,p,w_2), \\ &\quad ..., Pkn(u,p,w_n)] \end{aligned}$$

$PKnow(u, p)$ は，p の実行終了時における u の**確率的プロダクト知識**と呼ぶ．

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

u	Req	ReqSpec	SecInfo	CodeSpec	Code	TCode
A	1.0	1.0	0	0	0	0
B	0	1.0	1.0	1.0	1.0	0
C	0	0.01	0.01	1.0	1.0	1.0
D	0	0.0001	0.0001	0.01	1.0	1.0
E	0	0.0001	0.0001	0.0101	1.0	1.0

図 2 の例について，最後のプロセス (すなわち *Test*) が終了した後のすべてのユーザの確率的プロダクト知識を表 2 に示す．簡単のために，すべての $u, u' \in U$ および $w \in WP$ において，$trans(u, w, u') = 0.01$ としている．

3.4. 情報漏えいのリスク

情報漏えいは，前節で定式化した知識伝達の特別なケースと捉えることができる．すなわち，秘密プロダクトに関する知識が信頼度の低い開発者に伝達された場合である．

定義 6 (ソフトウェアプロセスにおける情報漏えい) 各々の開発者 u, $u' \in U$，プロダクト $w \in WP$，プロセス $p \in PC$ について，$w \in WP_{sec} \wedge u' \in U_{unaut} \wedge KT(p, u, w, u')$ が成り立つ時，ソフトウェアプロセスにおける**情報漏えいが発生する**という．

情報漏えいのリスクは,「全プロセスが終了した時点での，信頼度の低い開発者が持つ確率的プロダクト知識」として定式化される．具体的には，最終のプロセスが p_{final} であるソフトウェアプロセス P において，ある秘密プロダクト $w \in WP_{sec}$ が信頼度の低い開発者 $u \in U_{unaut}$ に漏えいする確率は,

$$Pkn(u, p_{final}, w)$$

で求めることができる．さらに，P においてある秘密のプロダクト $w \in WP_sec$ が，信頼度の低い開発者に漏えいする期待値 $Risk(w)$ は，次式で表される．

$$Risk(w) = \sum_{u \in U_{unaut}} Pkn(u, p_{final}, w)$$

4. 情報漏えいのリスクと生産性を考慮する開発者割当て問題

定義 2 によると，各プロセスに割り当てられる開発者数を増やすことで生産性が増し，プロセス実行時間を減

らすことができる．一方，定義 6 によると，信頼度の低い開発者数の増加は，情報漏えいのリスクを増加させることになる．ソフトウェアプロセスの開発者割当ては，人的資源や納期などに関する制約のもと，生産性の高さと情報漏えいの低さの両方を満たすように，うまく割り当てることが求められる．

ここで,「**最適**な開発者割当て」を「ソフトウェア開発時間が最も小さく，かつ，全プロセス終了時の情報漏えいのリスクが最も低い開発者割当て」と定義する．情報漏えいのリスクと生産性が考慮されたソフトウェアプロセス P の最適な開発者割当て問題の入力および出力は，次のようにまとめることができる．

入力：

1. ソフトウェアプロセス P を構成する，U, WP, PC, I, O，E および AB(AS を除く各要素)
2. 各々の $u, u' \in U$，$w \in WP$ についての $trans(u, w, u')$
3. P に対する制約条件．例えば，ソフトウェアプロセス全体の延べ開発者人数，1 つのプロセスに割り当てることができる開発者数の最大値など

出力： T が最小，かつ，$\sum_{w \in WP_{sec}} Risk(w)$ が最小である開発者割当て AS

5. ケーススタディ

本章では，4 章に基づき，最適な開発者割当てを探索する例を示す．ここでの開発者割当て問題の入力は次のとおりである．

1. 図 2 に示されるソフトウェアプロセス $P = (U, WP, PC, I, O, AS, E, AB)$ を対象とする．ただし，U, WP, PC, I, O，E および AB は与えられているが，AS は決められていないものとする．
2. すべての $u, u' \in U$ および $w \in WP$ について，$trans(u, w, u') = 0.01$ とする．
3. 次のような制約条件を与える．
   - P は，のべ最大 10 人の開発者によって実行される．
   - 各々のプロセス $p \in PC$ は，最大 4 人の開発者によって実行される．
   - 秘密のプロダクト $SecretInfo$ を直接扱うプロセス $Design$ は，信頼度の低い開発者 (C，D および E) を割り当てることはできない．

上の入力の条件を満たす開発者割当てを全探索し，各々の割当てについて $Risk(SecretInfo)$(以下，単純に $Risk$ と表記する) を計算した．

図 3 は，$Risk$ と T の関係を示すグラフである．縦軸は $Risk$ を示し，横軸は T(単位は時間) を示す．同じ T の値を持つ複数の割当てが見つかった場合，$Risk$ の最大値と $Risk$ の最小値を直線で結んでいる．この図より，T が大きくなるにしたがって，$Risk$ の最大値が減少する傾向にあることがわかる．

図 3 中に AS_{opt} で示される点は，最適な割当て，すなわち，$Risk$ および T がともに最小となった割当て ($Risk = 0.000, T = 100$) を表す．一方，比較のために，$Risk$ が最大となった割当て AS_{risky}($Risk = 0.061, T = 125$)，および，T が最大となった割当て AS_{long}($Risk = 0.000, T = 260$) を図中に示している．AS_{opt}，AS_{risky} および AS_{long} の開発者割当てを，それぞれ図 4 の (a), (b) および (c) に示す．

最適な開発者割当て (図 4(a)) においては，($Design$ より後のプロセスにおいて) 信頼度の高い開発者 (A，B) と信頼度の低い開発者 (C，D，E) は同じプロセスに割当てられていない．一方，$Risk$ が最大となった開発者割当て (図 4(b)) においては，$Design$ より後のすべてのプロセスにおいて信頼度の高い開発者と多数の信頼度の低い開発者 (A と C，D，E) が共同の開発者として割当てられている．また，T が最大となった開発者割当て (図 4(c)) では，すべてのプロセスが一人の開発者 (A) のみで行われる．共同開発者がいないため，情報漏えいのリスクは 0 となるが，ソフトウェア開発時間 T は最も高くなり，生産性は最も低くなる．

このように，提案手法は，情報漏えいのリスクとソフトウェア開発時間の両面において最適な開発者割当てを探索するのに役立てることができる．

6. おわりに

本稿では，情報漏えいのリスクと生産性とを同時に考慮して，ソフトウェアプロセスの開発者割当てを探索す

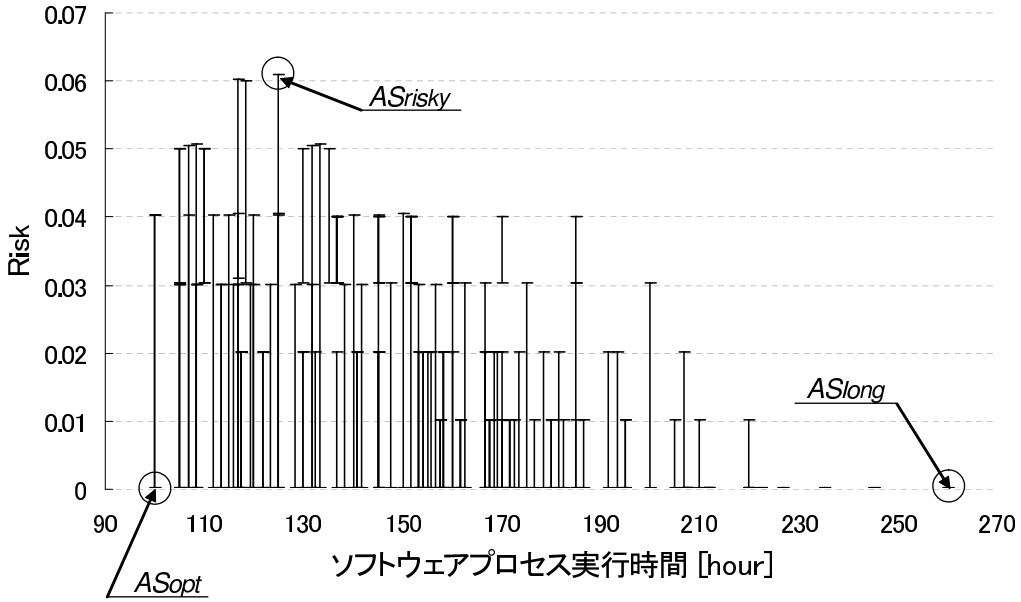


図 3. 情報漏えいのリスク $Risk$ とソフトウェア開発時間 T

る枠組みを提案した．まず，ソフトウェアプロセスを定義した後,「ある開発プロセスが実行されるとき，そのプロセスを実行した開発者の知識が互いに伝播し得る」という仮定のもと，開発者間の知識伝達のメカニズムを定式化した．また，定式化された式を応用し，ソフトウェアプロセスにおける情報漏えいのリスク，すなわち，秘密のプロダクトが信頼度の低い開発者に知られる確率を見積もる方法を示した．ケーススタディでは，情報漏えいのリスクが低く，生産性が高い開発者割当てを探索する例を通し，提案方法が情報漏えいを考慮したソフトウェアプロセスの開発者割当ての探索に役立つことを示した．

最後に，今後の課題を述べる．まず，知識伝達と生産性の関係をモデル化することを考えている．例えば，入力プロダクトに関する知識を持つ開発者が誰もいない場合よりも，入力プロダクトに関する知識を持つ開発者がいて，その知識を共有できる場合の方が，プロセスの生産性が高くなると考えられる．このように，知識伝達が開発者ごとの生産性に与える影響について考慮することで，より現実的な開発者割当ての探索が可能になると期待できる．

また，提案方法はソフトウェアプロセスに限らず，秘密情報を含む一般のワークフローにも適用可能である．例えば，医療ワークフロー [13] における作業者の割当て等にも応用が可能である．新しい適用分野の調査も今後の課題の 1 つである．

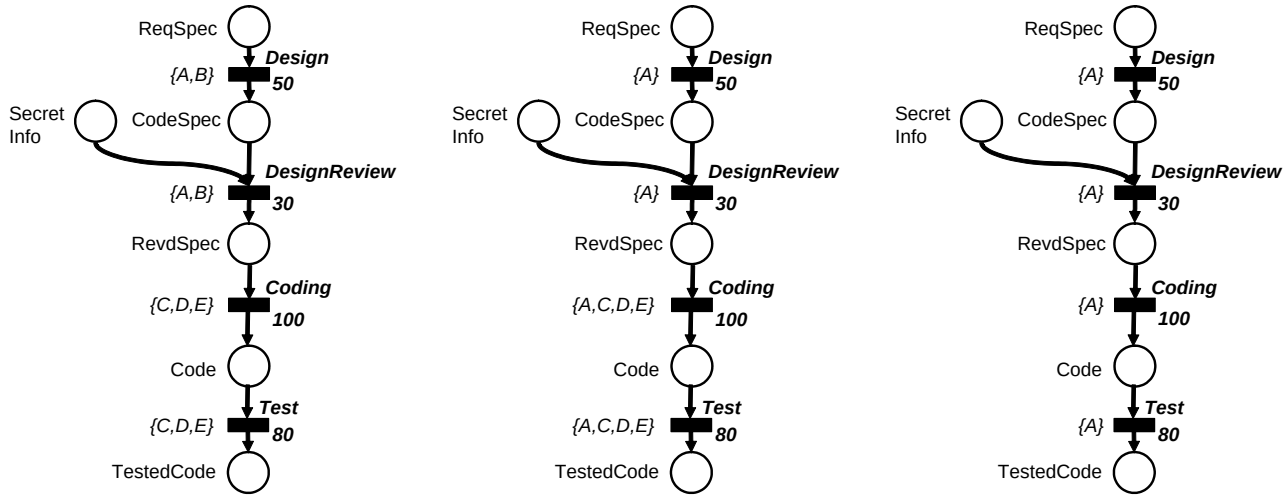
参考文献

[1] 有沢誠 “ソフトウェア工学”, 岩波コンピュータサイエンス, 岩波書店, 東京, 1988.

[2] Brooks, F. P. “The Mythical man-month: Essays in software engineering”, Addison-Wesley Professional, 1975.

[3] Chow, S., P. Eisen, H. Johnson, and van P. Oorschot “A white-box DES implementation for DRM applications”, in *Proc. 2nd ACM Workshop on Digital Rights Management*, pp. 1–15, 2002.

[4] Feiler, P. H. and W. S. Humphrey “Software Process Development and Enactment: Concepts and Definitions”, in *Proc. 2nd International Conference on Software Process*, pp. 28–40, 1993.



(a) 最適な開発者割当て
($Risk = 0.000, T = 100$)

(b) $Risk$ が最も大きい開発者割当て
($Risk = 0.061, T = 125$)

(c) T が最も大きい開発者割当て
($Risk = 0.000, T = 260$)

図 4. 最適な割当て，$Risk$ が最大となる割当ておよび T が最大となる割当て

[5] Garg, P. K. and M. Jazayeri “Process-Centered Software Engineering Environments”, IEEE Computer Society Press, 1995.

[6] 井上克郎, 松本健一, 飯田元 “ソフトウェアプロセス”, ソフトウェアテクノロジーシリーズ, No. 8, 共立出版, 東京, 2000.

[7] Jacobson, I., G. Booch, and J. Rumbaugh “The unified software development process”, Addison-Wesley Longman Publishing Co., Inc. , 1999.

[8] Kanzaki, Y. “Protecting Secret Information in Software Processes and Products”, PhD thesis, Nara Institute of Science and Technology, 2006.

[9] Kanzaki, Y., H. Igaki, M. Nakamura, A. Monden, and K. Matsumoto “Quantitative Analysis of Information Leakage in Security- Sensitive Software Processes”, *IPSJ Journal, Special Issue on Research on Computer Security Characterized in the Context of Social Responsibilities*, Vol. 46, No. 8, pp. 2129–2141, 2005.

[10] Keller, F., P. Tabeling, R. Apfelbacher, B. Grone, A. Knopfel, R. Kugel, and O. Schmidt “Improving Knowledge Transfer at the Architectural Level: Concepts and Notations”, in *Proc. The 2002 International Conference on Software Engineering Research and Practice*, 2002.

[11] Liong, Y.-L. and S. Dixit “Digital Rights Management for the Mobile Internet”, *Wireless Personal Communications*, Vol. 29, No. 1-2, pp. 109–119, 2004.

[12] Marsan, M. A., G. Balbo, G. Conte, S. Donatelli and G. Franceschinis “Modelling with Generalized Stochastic Petri Nets”, John Wiley, 1995.

[13] Quaglini, S., C. Mossa, C. Fassino, M. Stefanelli, A. Cavallini and G. Micieli “Guidelines-Based Workflow Systems”, Vol. 1620/1999 of *Lecture Notes in Computer Science*, pp. 65–75, Springer-Verlag, 1999.