ソフトウェア工学 98－6
（1994．5．25）

ソフトウェアプロセスの動的な変更部分の分析による プロジェクト運営実践技術のモデル化とプロセス改善

阪井 誠[†‡]　松本 健一[†]　鳥居 宏次[†§]

[†] 奈良先端科学技術大学院大学 情報科学研究科
[‡] 株式会社 SRA
[§] 大阪大学 基礎工学部

本報告では，ソフトウェアプロセスに対して行われている動的変更で用いられる“プロジェクト運営の実践技術”のモデルを提案し，その再利用を容易にする．また，モデル化された実践技術に基づくソフトウェアプロセスの改善方法を提案をする．ケーススタディの結果，提案するモデルとプロセスの改善方法が実際のソフトウェアプロセスに適用可能であること，及び知識と経験の少ないプロジェクトリーダであっても，熟練者の実践技術の再利用が容易となることが分かった．

Modeling Software Project Operation Skills and Improving Software Processes based on Software Process Change Analysis

Makoto Sakai[†‡]　Ken-ichi Matsumoto[†]　Koji Torii[†§]

[†] Graduate School of Information Science, Nara Institute of Science and Technology
[‡] Software Research Associates, Inc.
[§] Faculty of Engineering Science, Osaka University

[†] 8916-5 Takayama, Ikoma, Nara 630-01, Japan
[‡] 1-1-1 Hirakawa, Chiyoda, Tokyo 102, Japan
[§] 1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

This paper proposes a software project operation skill model and a software process improvement method that based on software process change analysis. Results of case study show that the proposed model and method can be applied to some typical software processes, and can transfer expert skills of software operation to novice software project operators.

1 まえがき

Osterweil が提案する “プロセスプログラミング” の考え方に基づいて，ソフトウェアの開発過程（ソフトウェアプロセス）を形式的に記述する研究が盛んに行われている [1]．ソフトウェアプロセスの形式的記述（以下では，プロセス記述と略す）の目的は，(1) 理解とコミュニケーションの容易化，(2) プロセスの改善，(3) プロセスの管理，(4) プロセス実行時の支援，(5) 自動実行環境の実現，である [2][3]．

これまでにも数多くのプロセス記述法が提案され，記述の試みも行われている [4]．しかし，ソフトウェア開発において起こり得る様々な事象について，その詳細な記述を予め（プロセスの実行前に）獲得することは，現時点では，困難とされている [5]．特に，開発作業の遅れ，仕様の変更，開発環境の変化といった予定外の事象の発生やその対処の過程を予め全て記述することは現実的ではない．

そこで，プロセス実行前には，プロセスの概略のみを記述しておき，プロセスの実行に従い，必要に応じて記述の変更や詳細化を動的に行うというアプローチが研究されている [5]．文献 [5] では，プロセスの骨組みが開発者に渡され，その詳細化は各開発者が必要に応じて行う，という方法が述べられている．各開発者により詳細化されたプロセスは統合され，プロセスの予実管理に用いられる．また，ソフトウェアプロセスに対する動的な変更を記述する方法も提案されている．HFSP は，属性文法を拡張したプロセス記述言語である [6]．Redoing/Refrection と呼ばれる概念に基づいて，プロセスの動的な変更の記述が可能である．

実際のソフトウェア開発プロジェクトにおいても，プロセスの動的な変更や詳細化が行われることは多い．例えば，プロジェクトリーダ（管理者）は，記述の詳細レベルは高くないが比較的適用範囲の広いプロセス（基本プロセス）に従ってプロジェクトを進める．基本プロセスにその詳細が記述されていない “予定外の事象” が発生すると，プロジェクトリーダは，基本プロセスの実行を一旦停止し，発生した事象による被害ができるだけ小さくなるように，基本プロセスを変更し，その実行を再開する．プロジェクトリーダによるこうしたプロセスの動的変更に関わる技術を，ここでは，“プロジェクト運営の実践技術” と呼ぶこととする．

プロジェクト運営の実践技術は実際に利用され，その重要性も認識されている．しかし，実践技術の再利用，及び，ソフトウェアプロセスの改善への利用に関する研究はほとんど行われていない．そのため，実践技術の具体的な内容やその適用結果は，それを用いた（あるいは，開発した）プロジェクトリーダの知識や経験としてのみ残り，広く利用されることはない．結果として，プロセスの動的変更の妥当性や有効性は，それを行うプロジェクトリーダの知識や経験に大きく依存することになり，個人差も大きくなる．特に，知識や経験の少ないプロジェクトリーダの場合，プロセスを具体的に変更することができないだけでなく，予定外の事象の発生を早期に検知することすら困難となる．

本報告では，ソフトウェアプロセスの動的変更で用いられる “プロジェクト運営の実践技術” のモデルを提案し，その再利用を容易にする．また，モデル化された実践技術に基づくソフトウェアプロセスの改善方法の提案を行う．更に，提案するモデルとプロセスの改善方法が，実際のソフトウェアプロセス（の動的変更）に適用可能であることを確認するために行ったケーススタディについて述べる．

2 プロジェクト運営モデル

熟練したプロジェクトリーダによるプロジェクト運営のモデルを図 1に示す．このモデルでは，プロジェクトリーダは次の 3 つの作業を行う．

作業 1: 基本プロセスの実行と監視

基本プロセスに従ってプロジェクトを実行する．但し，基本プロセスには含まれていないが，プロジェクトにおいて起こり得る予定外の事象 (Unscheduled event) の発生を検知するため，監視プロセス (Moni-

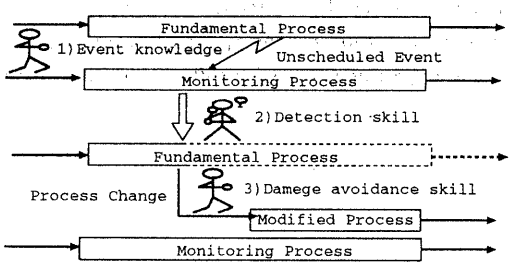

図 1: プロジェクト運営モデル

toring process) を基本プロセスと並行して実行する．

作業2: 基本プロセスの停止と予定外の事象の分析

監視プロセスにより得られた状況や，予定外の事象を早期に検出実践技術 (Detection skill) により予定外の事象が検知されると，基本プロセスの実行を一旦停止し，発生した予定外の事象の詳細を分析する．

作業3: 基本プロセスの変更と再実行

予定外の事象の分析結果に基づいて，被害が出来るだけ小さくなるように基本プロセスを変更し，その実行を再開する．

これら3つの作業に用いられる技術を "プロジェクト運営の実践技術" と呼ぶ．なお，作業2と作業3は，予定外の事象が検出されるたびに行われる．基本プロセスはそのたびに変更されることになる．

こうした方式でプロジェクト運営が行えるのは，熟練したプロジェクトリーダ (Expert) が業務に精通し，作業スタイルも確定しており，同時に複数の作業を高速に実行できるからである．基本プロセスを頻繁にチェックしているので，予定外の事象の検出が早く，その事象による被害も小さくできる．

一方，知識や経験の少ないプロジェクトリーダ (Novice) は，業務に不慣れなため作業の速度は遅い．作業スタイルが確定していないため，基本プロセスの実行に集中しがちとなる．予定外の事象の検出も遅れがちとなり，検出できてもプロセスの変更をうまく行えない．

両者をコンピュータシステムに例えるとその差がより明確になる（表1参照）．知識や経験の少ないプロジェクトリーダ (Novice) は，処理能力が低く命令セットも貧弱な CPU で，疑似マルチタスクを限界に近い能力でこなしている状況に似ている．熟練したプロジェクトリーダ (Expert) は，処理能力が高く，命令セットが豊富な CPU で，プリエンプティブ（予測可能）なマルチタスク処理を高速にこなしている状況に似ている．

こうした表を考察すると，知識や経験の少ないプロジェクトリーダに，熟練したプロジェクトリーダと同じ方式でプロジェクト運営をさせるのは難しいと考えられる．実践技術を単に伝達するだけでなく，知識や経験の少ないプロジェクトリーダに適した運営方式を開発し，リーダの習熟を待つのが現実的なアプローチである．

ここでは，割り込みによる疑似マルチタスク方式に替わるものとして，予定外の事象を検出するための一連の作業 (Detection Process) を予め基本プロセスに組み込む方式を提案する（図2参照）．

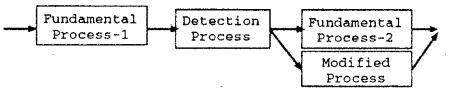

図 2: Embedded-type operation

表 1: Expert と Novice の比較

	Expert	Novice
Speed	excellent	poor
Command set	rich	poor
Concurrency	premptive	event driven

この方式では，予定外の事象の検出プロセス (Detection process)，及び，その事象による被害が最小となるように予め設計された対処プロセス (Modified process) が，図2に示すような形で基本プロセスに埋め込まれている．検出プロセスは，熟練したプロジェクトリーダが持つ事象を早期に発見するプロセスに基づいて作成されており，予定外の事象による被害が大きくなる前にその検出が可能な場所に埋め込まれている．一方，対処プロセスは，熟練したプロジェクトリーダが持つ対処プロセス（図 refskill 中の Modified process）に基づいて作成されており，対応する基本プロセスの部分と並行する形で埋め込まれている．検出プロセスによって予定外の事象が検出されると，対処プロセスが実行され，検出されないと，基本プロセスが実行される．

予定外の事象の検出を予め埋め込んだ検知プロセスによって行うことより，この方式を Embedded-type operation と呼ぶ．この方式は，知識や経験の少ないプロジェクトリーダ向けの運営方式である．また，図1で示した，熟練したプロジェクトリーダ向けの運営方式は，監視プロセスによって予定外の事象の検出していることから，Monitoring-type operation と呼ぶ．

3 ペトリネットによる記述

本報告では，図1および図2で示した各種のプロセスをペトリネットを用いて表現する．

ペトリネットは容易に理解が可能で，システムにおけるさまざまな事象の生起および動作を記述するモデルとして用いられる [7]．事象 (events) と条件

(preconditions / postconditions) はトランジション，プレース，トークン，及び，アークにより記述される．図3において，四角がトランジション，白丸がプレース，黒丸がトークン，矢印がアークである．

トランジションが動作を表し，プレースは動作の待ち状態を表す．入力側のトークンは動作への入力（仕様，処理の計画など）や前提条件を表す．出力側のプレースのトークンは動作による出力や後提条件を表す．トランジションに1つのアークを介して接するプレースの中の1つにでもトークンが置かれていない場合，トランジションの前提条件は満たされず，トランジションは発火できない．トランジションが発火しないと後提条件は満たされず，トランジションに1つのアークを介して接するプレースにトークンは置かれない．

図3のおけるペトリネットの動作を説明する．a-1 と b-1 の動作は，どちらか一方のみ実行される（分岐）．a-2 と a-3 の動作は，並行して行われる（並列動作）．a-4 は a-2 と a-3 の動作が終了し，直前の2つのプレースにトークンが揃うまで，発火せず，同期をとる役目を果たす．b-2 は b-1 が終了し，直前のプレースにトークンが置かれるまで発火しないので，b-1,b-2 は逐次動作である．

ペトリネットは，もともとプロセス中の処理の実行順序など時間的な振舞い，すなわち動作的側面 (Behavioral Perspective) が表現可能なモデルである [8]．最近では，階層化や資源表現等の拡張が行われている．本報告ではカラードペトリネットに文献 [9] の SPADE における時間的制約に準じた拡張を行ったものを用いる．

レビューの基本プロセスを表現したものを図4に示す．図4において，アーク上に書かれているのはトークンのカラーである．トランジション t2 （レビュー）に対しては，仕様書が前提条件，障害報告書とレビュー後の仕様書が後提条件である．右下に

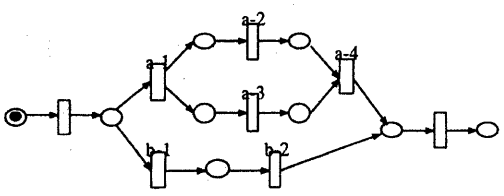

図 3: ペトリネットの記述例

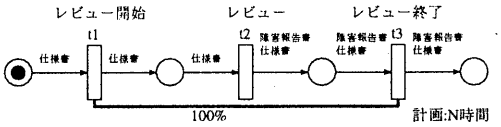

図 4: レビューの基本プロセス

書かれた時間は，このプロセスの計画時の作業時間である．一方，太線とその下の数字は作業時間の計画と実際の比率を表す．即ち，モデルで表されたプロセスが実際に行われた場合，太線で結ばれたトランジション間の一連の作業に要した時間が，計画された時間の何パーセントであったのかを表す*．

4 実践技術によるプロセス改善

実践技術をモデル化する基本アイデアはソフトウェアの変更で用いられる patch-list と同じである．即ち，要求が満たされていないソフトウェアに対しては，部分的な修正 (patch) が行われる．修正された新しいソフトウェアと元のソフトウェアとの差分 (difference) は patch-list として保存される．他のソフトウェアの修正に patch-list を用いる場合は patch-list を抽象化し，手法として利用する．

プロセスもソフトウェアである [1]．プロセスというソフトウェアにおける部分的な修正は，プロセスの動的な変更であり，手法は抽象化された実践技術に相当する．従って，プロセスの動的な変更を記述し，その記述から実践技術を抽出し，抽象化することにより，ソフトウェアプロセスの改善が容易になると考える．具体的な手順を以下に示す．

1. プロジェクト実行中に，予定外の事象により生じたプロセスの動的な変更の部分をペトリネットで記述する．記述対象は事象が検出されたプロセスと，その事象による被害を小さくするために実行されたプロセスである．予定外の事象は文章で記述する．

2. プロセスの動的な変更部分の記述に含まれる事象を検出した部分が，事象を早期に検出する

*文献 [9] では，太線とその下に書かれた時間は作業間の時間制約を表しているが，ここでは実際に費やされた作業時間（作業実績）を表す．また，ケーススタディにおいて規模の異なる2つのプロジェクト間で比較を行うため，作業実績は当初の計画との比率で表す．図4の例では予定の N 時間の 100 %でレビュー作業が終了したことを示している．

実践技術を含んだプロセスであれば，それを抽出する．

3. プロセスの動的な変更部分の記述に含まれる，事象による被害を小さくする部分が，事象による被害を小さくする実践技術を含んだプロセスであれば，それを抽出する．

4. 2,3 で抽出したプロセス記述の前提条件，後提条件，動作名を工程に依存しない名称に変更する．例えばレビュー工程であれば，“仕様書” は “検査対象”，“障害報告書” は “障害記録”，“レビュー” は “検査” の様に変更する．これに伴い事象の表現も変更する．また，事象が複数ある場合は直接変更の原因となった事象以外は削除する．この作業により実践技術は，汎用的な改善技術として蓄積される．

5. 予定外の事象の再現性を確認する．具体的には，他のプロジェクトの同じ工程で，同一事象が起こり，プロセスの変更が行われているかを調べる．再現性が確認できない場合は，収集された実践技術は蓄積するが，基本プロセスの改善は行わない．

6. 動的なプロセスの変更が行われた工程に，2,3,4 で得られた実践技術を組み込む．事象を早期に検出する実践技術と事象による被害を小さくする実践技術のうち，最も効果のあった実践技術を 1 つずつ組み込む．具体的には，事象が検出された位置に，事象を早期に検出する実践技術のプロセスを挿入する．その直後に分岐処理を加え，事象が検出された場合には事象による被害を小さくする実践技術のプロセスが実行されるようにする．

7. 進捗管理を容易にするために，実践技術の組み込みにより追加されたトランジションの前提条件および後提条件をプロダクトに変更する．また，そのプロダクトを生成可能なトランジション（動作）の直後までのトークンのカラーにそのプロダクトを付加し，到達可能なプロセス記述にする．

5 ケーススタディ

ソフトウェアプロジェクトの運営において，レビュー時に一部のプロダクトのみに多くの障害（バグ）が発見された場合，レビューの計画を変更することがある．障害の多く発生したプロダクトを重点的にレビューをし，障害をできるだけ多く取り除く努力を行うのである．ここでは，このようなプロセスの動的な変更の行われた 2 つのプロセスを用い，前述の方法によるプロセス改善のケーススタディを行う．以下に示すプロジェクト A はデータベースアプリケーションを開発するプロジェクトである．工数は 1 人× 6 カ月である．プロジェクト B は文書処理アプリケーションを開発するプロジェクトである．工数はピーク時 10 人× 11 カ月である．プロセス記述はプロジェクトの開発要員が行った．

図 5-a-1 はプロジェクト A のプロセスの動的な変更部分の記述である．直前に行われた打合せの際に，説明用としてドキュメントの一部を発注元の担当者に見せたことで，実質的に事前レビューが行われ，計画の 7%のレビューが実施された．この作業により，障害の多いプロダクトが存在しているという障害情報が得られた．障害の多いプロダクトが存在するという予定外の事象を検出し，プロセスは動的に変更された．具体的には，障害が多いと考えられるプロダクトからレビューされた．また，障害が多いと考えられるプロダクトにレビュー時間が多く配分された．事前レビューは，打合せ作業の一部として行われたため，レビュー工数は増えているが，スケジュールの遅れは生じなかった．

プロジェクト A からは 2 つの実践技術が収集される．図 5-a-2 に示す実践技術 a は，検査対象を部分検査し，障害の多いプロダクトが存在するという事象を早期に発見する実践技術である．図 5-a-3 に示す実践技術 b は，得られている障害情報の多い検査対象により多くの検査時間を配分し，障害の多い検査対象が存在するという事象による被害を小さくする実践技術である．

図 5-b-1 は同様のレビュー工程でプロセスの動的な変更が行われたプロジェクト B の変更部分の記述である．設計の遅れているプロダクトのレビューは最後に行われた．この最後にレビューの行われたプロダクトに多くの障害が集中していたため，障害の多いプロダクトの発見はレビュー作業として計画さ

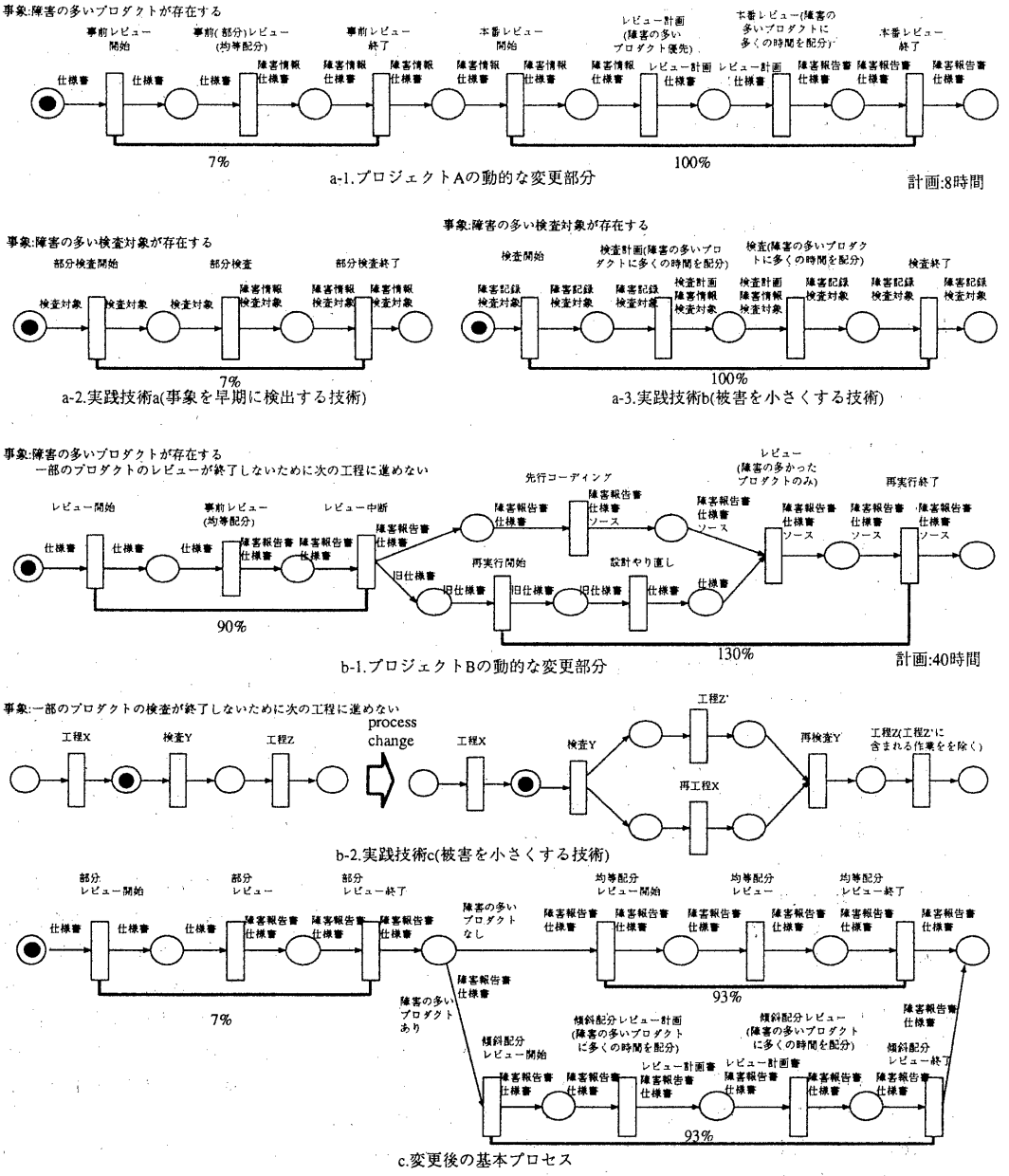

図 5: ケーススタディにおける実践技術とプロセス改善

れた時間の90%が経過してからであった．プロジェクトBでもプロジェクトAと同様に障害の多いプロダクトが存在するという事象が生じており，これが再現性のある事象であることが確認できた．プロジェクトBではこの事象から，一部プロダクトのレビューが完了しないため次の工程に進めないという予定外の事象が生じた．この事象の被害を小さくするため，レビューの終了したプロダクトは先行してコーディングを行った．また，障害の多かったプロダクトは再設計および再レビューを行った．再設計と再レビューはレビュー期間内に終らず，レビューの終了は遅れた．

プロジェクトBからは実践技術cが収集される．図5-b-2に示す実践技術cは，検査の遅れているプロダクト以外は，次の工程を行うことにより，一部プロダクトの検査が終了しないために次の工程に進めないという事象による被害を，小さくする実践技術である．

こうして得られた実践技術を用いてプロセスの改善を試みる．具体的には，再現性が確認された“障害の多いプロダクトが存在する”という事象の早期検出と，事象の被害を小さくするために，実践技術aおよび実践技術bを図4に示す基本プロセスに組み込み，知識や経験の少ないプロジェクトリーダにも再利用が容易なプロセスを記述する．

障害の多いプロダクトの存在を早期に検出するため，部分レビューを組み込む(実践技術a)．障害の多いプロダクトの存在という事象を検出した場合は，障害の多かったプロダクトに，多くのレビュー時間を配分するというプロセスを組み込む(実践技術b)．

実践技術aでは部分レビューの障害情報を後提条件としていたが，進捗の管理を容易にするため，プロダクトである障害報告書を部分レビューの後提条件とする．傾斜配分レビューの前提条件の障害情報はこのままでは生成されないので，障害情報を障害報告書に置き換える．この置き換えにより，生成可能な動作の直後までのトークンのカラーに前提条件が含まれている．

以上の手順で得られた新たな基本プロセスを図5-cに示す．

6 考察

プロセスの動的な変更部分を分析し，プロジェクト運営の実践技術をモデル化し，知識や経験の少ないプロジェクトリーダにも再利用が可能なプロセスを得るケーススタディを行った．ケーススタディで得られた実践技術aは，予定外の事象を早期に検出する技術である．実践技術b,cは，予定外の事象による被害を小さくする技術である．

ケーススタディで得られたプロセス（図5-c参照）は，以下の点が改善されている．

(1) 予定外の事象の生じる可能性のある工程に，事象の発生を判定する具体的な基準が記述されている．

(2) 予定外の事象を早期に検出するためのプロセスは，部分レビューとして組み込まれている．

(3) 問題による被害を少なくするプロセスは，分岐処理により組み込まれている．

従って，ケーススタディで得られたプロセスは，図2で示したEmbeded-type operationのためのプロセスになっている．プロジェクトA,Bで得られた実践技術を，実践技術を持たない非熟練者に伝達し，再利用することが容易になったと考えられる．

また，追加された動作の前提条件を，後提条件とすることが可能な動作までのトークンのカラーに追加している．このため，得られた基本プロセスは可達であり，プロセスが計画通り実施されれば終了可能である．

ケースタディで得られた（改善された）基本プロセスは文献[10]の方法と類似している．文献[10]では，抜き取り検査の結果から障害量を予測し，障害量に比例したレビュー時間を割り当てている．評価実験では開発中に発見された障害量のうち，レビュー時に検出された障害量比率が向上したと報告されている．ケースタディで得られた基本プロセスは文献[10]と比較すると曖昧な点がいくつか認められるが，ほぼ同等と見なすことができ，プロセスは改善されたと考えられる．

ケーススタディで得られた実践技術cは，現実のプロジェクトにおいてプロジェクトリーダが，全ての工程で常に気を配っている点である．検出工程を組み

込むと基本プロセスが複雑になるので，Monitoring-type operation の技術として記述を蓄積することが適当と考えられる．

ケーススタディからわかるように，プロセスの動的な変更部分の分析に必要となるのは，部分的なプロセス記述であり，記述量，及び，記述のための作業量は少ない．データ量が少ないため，粒度が細かくプロダクトを生成しないプロセスであっても，その記述は比較的容易であると考えられる．

7 まとめ

本報告では，ソフトウェアプロセスに対して行われている動的変更で用いられる“プロジェクト運営の実践技術”のモデルを提案し，モデル化された実践技術に基づくソフトウェアプロセスの改善方法を提案した．更に，提案したモデルとプロセスの改善方法が，ソフトウェアプロセスに対して行われた実際の動的変更に適用可能であることを確認するために行ったケーススタディについて述べ，その結果の考察を行った．

ケーススタディの結果，熟練したプロジェクトリーダが持つ実践技術を基本プロセスに組み込むことにより，その再利用が容易になることがわかった．実践技術を基本プロセスに組み込み，基本プロセスを徐々に改善していくという点から，提案した方法はプロトタイププロセスモデルに基づくものと言える．

提案する方法は，ケーススタディからもわかるように人間に任されているところが多い．例えば，提案した方法では，プロジェクト運営の方式には，Monitoring-type operation と Embedded-type operation の2つがあるとしている．しかし，プロセスの動的変更の分析によって得られた個々の実践技術を，どちらの方式において利用すべきか，その判断基準は明確になっていない．

プロセスモデル構築の知識獲得と利用に関しては文献[11]があり，学習機構を用いたインタビューシステムにより知識を抽出し，プロセスモデルを構築する研究が行われている．こうした研究の成果を利用し，より形式化された構成方法とする必要がある．

参考文献

[1] L. Osterweil: “Software process are software too”, Proc. of 9th International Conference on Software Engineering, pp. 2–13 (1987).

[2] B. Curtis, M. I. Kellner and J. Over: “Process modeling”, Commun. ACM, **35**, 9, pp. 75–90 (1992).

[3] 井上：“最近のソフトウェアプロセスの研究動向”, ソフトウェア・ツール・シンポジウム'93, pp. 5–13 (1993).

[4] 鈴木：“代表的なプロセス記述言語の特徴”, ソフトウェア科学会研究会報告, **SP93-1-2**, pp. 13–24 (1993).

[5] 石若, 元治, 荻原, 井上：“動的詳細化が可能なプロセス記述の表記法とその開発現場への応用について”, 情報処理学会研究会報告, **92**, 88, pp. 88–5 (1992).

[6] T. Katayama: “Mechanisms for software process dynamics”, Proc. of 5th International Software Process Workshop (1989).

[7] J. L. Peterson: “Petri Net Theory and The Modeling of Systems”, Prentice-Hall (1981).

[8] 佐伯：“ソフトウェアプロセスのモデル化へのネットの応用”, 情報処理, **34**, 6, pp. 718–730 (1993).

[9] S. Bandinelli, A. Fuggetta, C. Ghezzi and S. Grigolli: “Process enactment in SPADE”, Proc. of 2nd European Workshop Software Process Technology, pp. 67–83 (1992).

[10] 地村, 阿部, 楠本, 松本, 菊野：“技術レビューにおけるレビュー時間割り当て方法の実験的評価”, 第14回ソフトウェア信頼性シンポジウム, pp. 39–44 (1993).

[11] 山口, 落水：“ソフトウェアプロセスモデル構築における知識獲得と利用方式”, 人工知能学会研究会資料, **SIG-FAI-9002-3**, pp. 19–28 (1991).