エンピリカルデータを対象としたマイクロプロセス分析

森崎 修司 松村 知子 大蔵 君治 伏田 享平 川口 真司 飯田 元

奈良先端科学技術大学院大学 情報科学研究科

ソフトウェア開発時にプロダクトの管理情報やツールの実行履歴から自動収集できる時系列データ(エンピリカルデータ)をもとに，プロセス品質を分析する手法(マイクロプロセス分析)を提案する．提案手法ではエンピリカルデータにエンピリカルデータと同程度の粒度の手順を示したプロセスモデルを与えることにより，マイクロプロセスメトリクスを得る．得られたマイクロプロセスメトリクスはプロセス遵守の度合い，作業の所要時間，繰返し回数などがあり，これらのマイクロプロセスメトリクスからプロセス品質の定量的な分析が期待される．本稿では，商用開発で利用した構成管理ツール，障害管理ツールから得たエンピリカルデータに対し，不具合修正に関する構成管理ツール，障害管理ツールの運用手順に基づくプロセスモデルを与え，マイクロプロセスメトリクスが得られることを示す．

Micro Process Analysis for Empirical Software Engineering Data

Shuji Morisaki Tomoko Matsumura Kimiharu Ookura Kyohei Fushida
Shinji Kawaguchi Hajimu Iida

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

This paper proposes microprocess analysis to analyze process quality from empirical software engineering data. The empirical data can be collected automatically from management information of products and execution histories of tools during a software development. In the proposed method, microprocess metrics are obtained from the empirical data with the process model defined by the same granularity of the given empirical data. Microprocess metrics includes degree of followed process, duration of a process, and the number of repetition of a process. Microprocess metrics leads quantitative analysis of process. In this paper, we conducted a microprocess analysis to empirical data with a process model. The empirical data consists of execution histories of configuration management system and issue tracking system during coding and testing phase of a commercial software development.

1. はじめに

ソフトウェア品質が与える社会的影響の増大，及び，市場競争におけるソフトウェアの役割の増大に伴い，納期の短縮，高信頼化に対する要求は高まるばかりである．それらの要求に応えるべく，品質計画や品質管理に関する手法がいくつも提案されている[2][7][8]．品質計画や品質管理には，客観的な品質測定が不可欠であり，品質測定は大別してプロダクトの品質測定とプロセスの品質測定がある．プロダクトの品質測定には，ソースコードメトリクスをはじめ収集，解析/分析の先行研究が多くあり，自動化による支援も進んでいるといえる[8][10]．しかしながら，プロセスの品質測定を目的としたデータの収集や解析/分析の自動化に関する研究はそれほど進んでいるとはいえない．

プロセスデータの収集，分析の自動化への障壁となっている最大の要因は，ソフトウェアプロセスにおける活動の主体が人間であるため，その振る舞いに関するデータ(＝プロセスデータ)のみを自動収集することが難しいことである．たとえば Humphry の提唱する PSP (Personal Software Process)では，開発者が個人プロセスの計画・計測・評価を行うよう訓練付けることで，個人レベルでのプロセスの品質を明快に評価できる枠組みを提供している．作業の記録自体は(ツールによる補助は期待できるものの)手動を前提としており[3]，プロセスデータの収集には，開発者の PSP 実践のための訓練，及び，開発の始終で開発者の手動による記録が必要となる．

したがって，プロセスを評価する取り組みの多くは，

CMMI[1]のように組織のプロセス実施能力を定性的に評価するものか，あるいは，成果物の管理情報として記録されたデータからプロセスに関する情報を取り出し，作業効率や生産性についての定量的な評価を行うものである．後者の取り組みの多くは，作業報告書に類するドキュメント類をデータ源としており，分析対象となるプロセスの粒度は，工程やそれを何段階かに分割した作業に対応したものである．

一方，近年では，開発プロジェクトの作業記録を自動収集する研究が鳥居らのグループ[9]や Johnson らのグループ[4]により進められている．これらの手法では，ツールの実行履歴や成果物の管理情報などプロセスデータを含むデータ収集が提案されている．これらの手法を前提とすることで，より細かな粒度での定量的なプロセス分析が期待できる．

本研究では，時系列に沿って自動的に収集された開発作業の記録（エンピリカルデータ）からプロセスの振る舞いに関して従来よりも細かい視点で分析を行うことを提案する．エンピリカルデータとは，時系列にそった作業実行イベントの集合であり，これに作業実行順序の規範としてのプロセスモデルを当てはめ解釈することで，形式的で定量的な分析が可能となる．

本稿ではこのようなプロセス分析の手法を特に「マイクロプロセス分析」と呼ぶ．マイクロプロセス分析では，定量的な指標（マイクロプロセスメトリクス）を直接測定するので，プロセス品質の，より詳細な評価が可能となる．具体的に測定可能なマイクロプロセスメトリクスには，プロセス遵守の度合い，作業の遅延時間，繰返し回数などがあり，これを元に，プロセスの実行の誤り，作業の遅延，不必要な繰返し，などの視点からプロセスの品質を評価することができる．

本稿では，商用開発の下流工程で収集されたエンピリカルデータとそこで想定されているプロセスモデルとを対象に，マイクロプロセス分析を試行した結果について報告する．対象となるエンピリカルデータには構成管理システム CVS とバグ管理システム GNATS の実行記録が含まれている．

以降，2 章で対象とする時系列のエンピリカルデータ，プロセスモデルについて述べ， 3 章で，プロセスモデルによるエンピリカルデータの解釈方法と解釈から得られるプロセスメトリクスについて述べる．4 章では，その適用例として，構成管理システムとバグ管理システムから得られたエンピリカルデータ，及び，プロセスモデルから得たプロセスメトリクスを述べ，5 章でまとめる．

2. プロセスの記録と表現方法

2.1. 作業記録

本節ではマイクロプロセス分析が対象とするエンピリカルデータの内容について定義を与える．エンピリカルデータとは，ソフトウェア開発プロセス実行中に実際に観測された様々な時系列イベントの集合 E として定義される．個々のイベント $e \in E$ は，イベントの発生時刻 t，イベントの種類 s，イベント属性の集合 $a(\alpha_1, \ldots \alpha_n)$から成る．属性の集合 a の要素数 n はイベントの種類に応じて異なる．

イベントには，ソフトウェア開発の中間成果物，及び，最終成果物の管理情報として得られるものや，ツールの実行履歴や，会議体の記録など，様々な種類のものが存在する．イベントの種類や粒度について特に制約はないが，ここでは，記録ツール等により自動的に収集可能なものを想定する．

一つのイベント e は〈t, s, a〉の 3 字組形式で記述される．たとえば，2006/12/2 13:07 にファイル「基本設計書.doc」を更新者「木村」が変更し，更新内容が「レビュー指摘事項 No. 0004 を反映した修正」であるイベントは，以下のように記述する．

〈2006/12/2 13:07, 更新, {ファイル名=”基本設計書.doc” 更新者=木村, 更新内容=”レビュー指摘事項 No. 0004 を反映した修正”〉

これらのイベント列は，中間成果物を含む成果物の管理ツール（たとえば CVS）や変更管理/不具合情報管理ツール（たとえば GNATS）の実行履歴などから抽出が可能である．また，EPM(Empirical Project Monitor)システムを導入することで，これらのイベント列を統一的に自動収集可能である[5].

成果物の管理情報には，イベント日付時刻として，構成管理ツールに入力のあった日付時刻，イベント種類として，構成管理ツールへの入力内容，イベント属性として，変更サイズや更新者などの付随情報，及び，更新者が入力した属性情報，が含まれる．

変更管理/不具合管理情報は，イベント日付時刻として，入力のあった日付時刻，イベント種類として，新規登録や更新や削除などの入力種別，イベント属性として，登録者，不具合の内容，再現方法に関する記述，対応責任者が含まれる．

2.2. ソフトウェアプロセスモデル

ソフトウェアプロセスとは，狭義にはソフトウェア開発における作業間の順序関係や包含関係など意味するが，さらに，対象成果物や開発組織の構造，開発や管理のための方法論までを含めてソフトウェアプロセスと称することもある．ソフトウェアプロセスを一定の記法に基づき記述したものをソフトウェアプロセスモデルと呼ぶ．本稿では，狭義の意味によるプロセスに着目する．つまり，プロセスの持つ振る舞い的な側面を記述したものをプロセスモデルとする．

これまでに，プログラミング言語に基づいたものなどを含め，多くのプロセスモデル記述言語が提案されているが，WBS（Work Breakdown Structure）形式に基づいた表や文章などが，現実的な記法として広く用いられ

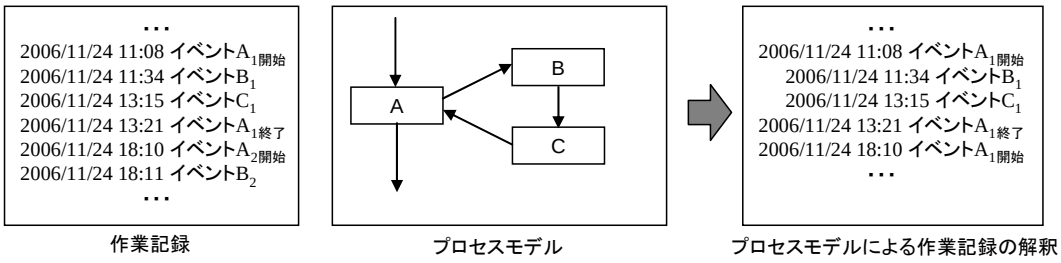

図 1 プロセスモデルによる作業記録の解釈の模式図

ている. より先進的な記述言語としては, OMG 標準となった SPEM（Software Process Engineering Metamodel）がある. SPEM ではプロセス記述の持つ意味がメタモデルにより細かく規定され, ソフトウェアプロセスの持つ様々な記述要件を満たすものとなっているが, 一方であまりに多くの記述要素や記述形態を含むため, 開発現場におけるプロセスの振る舞い的な性質の記述には, UML のアクティビティ図や BPMN（Business Process Modeling Notation）など, より単純な記法が用いられることが多い.

本稿では, 特定のプロセスモデル記述言語を前提としていない. 振る舞いの例示には便宜上 UML のアクティビティ図を用いる.

3. マイクロプロセス分析の提案

3.1. プロセスモデルに基づく作業記録の解釈

エンピリカルデータのイベント列をプロセスモデル中で定義された作業群に対応付けることにより, プロセスモデルに基づいた解釈が成立する. 具体的には, エンピリカルデータ中の1イベント e, 1 イベント中の属性情報 a_k をプロセスモデル中の1作業, もしくは, 1作業の開始または終了に対応付けを行う.

図 1 は単純な階層関係を持つプロセスモデルを用いて作業記録の解釈を行う例である. 作業 A は作業 B および C の逐次実行に分解されている. この場合, 各イベントについて,「作業 A の開始」,「作業 A の終了」,「作業 B の実行」,「作業 C の実行」の 4 種類に対応づけた解釈を行っている. このように, プロセスモデルに対して実際の作業記録を対応づけて解釈したものを本稿ではプロセスインスタンスと呼ぶこととする.

プロセスインスタンスに対しては作業時間の計測をはじめとして様々な定量的評価が可能である. また, プロセスインスタンスとプロセスモデルとの照合の度合いから, 実際の開発作業がどの程度, 標準プロセスモデルに準拠したものであったかを評価することも可能となる.

3.2. マイクロプロセスメトリクスの測定

一般なプロセスメトリクスとは作業効率や生産性といった, 抽象的でより粒度の荒い概念である. 我々は, マイクロプロセスに対するメトリクス（マイクロプロセスメトリクス）として以下のものを提案する. マイクロプロセスメトリクスには, 大別して, 手順, 回数, 時間の 3 種類が考えられる.

手順的メトリクスの例としては以下のものがある.

- 手順漏れ
  プロセスモデルで定義されている手順を実行しているかどうか
- 遵守度合い
  プロセスモデルで定義されている手順どおりに実施できているかどうか（手順の入れ替わりなど）
- 登録情報漏れ（手順の完遂）
  イベント属性 a_k の抜け, 誤り
- 並列度
  並列して実行している作業の数

回数的メトリクスの例としては以下のものがある.

- 繰返し回数
  作業の繰返し回数

時間メトリクスの例としては以下のものがある.

- 所要時間
  個々の作業の所要時間, その平均や分散
- 全時間対やり直し時間比
- 全時間対待ち時間比
- 全時間対同期/合流待ち時間比
- 遅延（ロスタイム）

これらのメトリクス自体は, 比較的単純なものが数多く含まれるが, いずれも従来の作業報告ベースでのプロセス記録からは計測困難であり, エンピリカルデータを対象とすることではじめて現実的な計測・評価が可能となる.

4. 適用例

4.1. 対象データ

実際のソフトウェア開発の際に収集した, 構成管理ツールのイベントとバグ管理票のイベントを実際の運用フローから得たプロセスモデルにあてはめ, 手順漏れ, 遵守度合い, 登録情報漏れ, 所要時間を測定した. 対象データは, 経済産業省の支援を受けた COSE（COnsortium for Software Engineering）の参加企業に

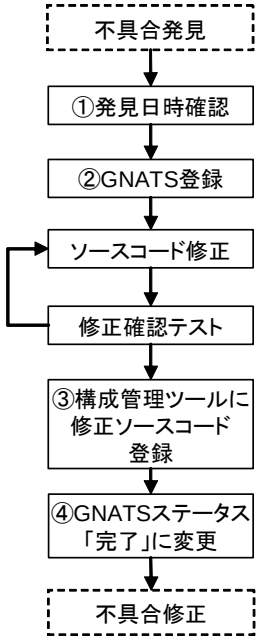

図 2 不具合修正手順

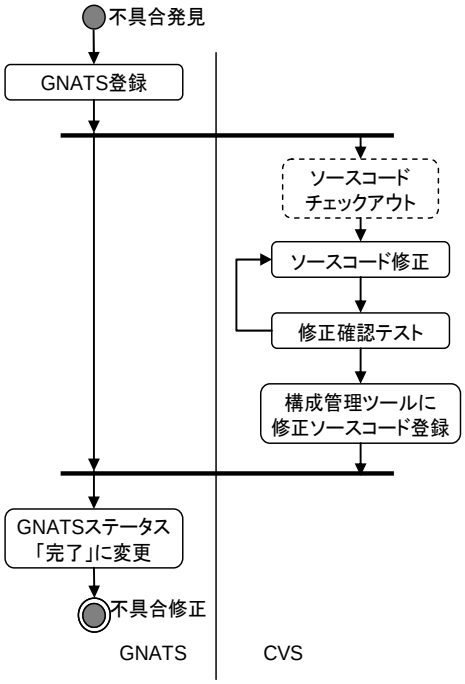

図 3 修正手順のプロセスモデル

よって実施されたプローブ情報システム開発プロジェクトの実施中に収集したものである. データの収集には, EASE プロジェクトで開発した EPM(Empirical Project Monitor)[3]が利用されており, 構成管理ツール CVS のログ, 障害管理ツール GNATS の障害票のステータス変更の履歴が含まれている.

対象プロジェクトは複数ベンダによるウォータフォール型の分担開発である. CVS と GNATS のデータはコーディング工程の後半から単体テスト, ベンダ内結合テスト工程終了までの 1 ヶ月程度の期間に, 複数ベンダのうちの 1 社から収集されたものである.

開発ベンダは, 開発したソースコードを図 2 に示す手順でデバッグするよう, あらかじめ依頼されている. 図 2 の手順をコード修正のための部分プロセス(CVS の操作に対応)不具合の登録と解決のための部分プロセス(GNATS の操作と修正作業に対応)にわけて記述したアクティビティ図が図 3 である.

表 1 にそれぞれの手順で, 開発担当者が入力, 確認すべき情報を示す. バグが発見されたと判断したら, 図 3 の「GNATS 登録」において, 発見者は GNATS にバグ情報を登録する. バグ情報には, 発見日時, 対応者, バグの内容, 再現方法, 優先度合いが含まれる. バグを登録するとバグ番号が自動的に割り振られ, 「着手」状態になる. 対応者は必要に応じて該当部分のソースコードをチェックアウトする(図 3「ソースコードチェックアウト」). 対応者は, 原因を特定し, 該当するバグをソースコードから除去し, 必要に応じて回帰テストを実施しながら, 修正を確認する(図 3 の「ソースコード修正」, 「修正確認テスト」). 修正できたと判断したら, 修正したソースコードを構成管理ツールに登録する(図 3 中「構成管理ツールに修正ソースコード登録」). ソースコードを登録する際に図 3 の「GNATS 登録」の際に割振られたバグ番号を登録時のコメントとして手動で登録する. 最後に 該当するバグの状態を「完了」に変更する(図 3 の「GNATS ステータス「完了」に変更」).

4.2. 適用結果

収集されたエンピリカルデータに対して, 手作業によるマイクロプロセス分析を試行した. まず, イベント列と図 3 で規定された手順との対応関係を取り, バグ登録件数と同数の 31 個のプロセスインスタンスを同定した. 今回の例ではプロセスの規模が小さいこともあり, 特に問題なく対応付けを行うことができた.

次に, 作業手順の漏れ, 作業手順の遵守度合い, 登録情報漏れ, 所要時間に関するプロセスメトリクスを測定した. 結果を表 2, 図 4 に示す. 要点は以下の通りである:

- GNATS 登録の手順漏れは発見されなかった.
- 修正したソースコードの CVS 登録の手順漏れが 2 件あった.
- 一時的に CVS が最新の状態に保たれていない状況が起きていた.
- バグ修正の確認とともに GNATS のステータスを「着手」状態から「完了」状態に変更していないものが 3 件あった.

表 1 登録情報

手順	登録内容	手順を守らなかったときのリスク
①発見日時確認	なし(日時の目視のみ)	正確な発見日がわからない
②GNATS 登録	①発見日時 バグの説明, 再現方法, 修正担当者	バグ検出が開発者間で周知できず, 同一のバグに対して複数の担当者が修正する
③構成管理ツールに修正ソースコード登録	②で得られるバグ番号, 修正担当者	構成管理ツールからチェックアウトしたソースコードが最新でなくなる. ソースコード修正部分が不明になる.
④GNATS ステータス「完了」に変更	なし	バグが修正されていることを周知できない.

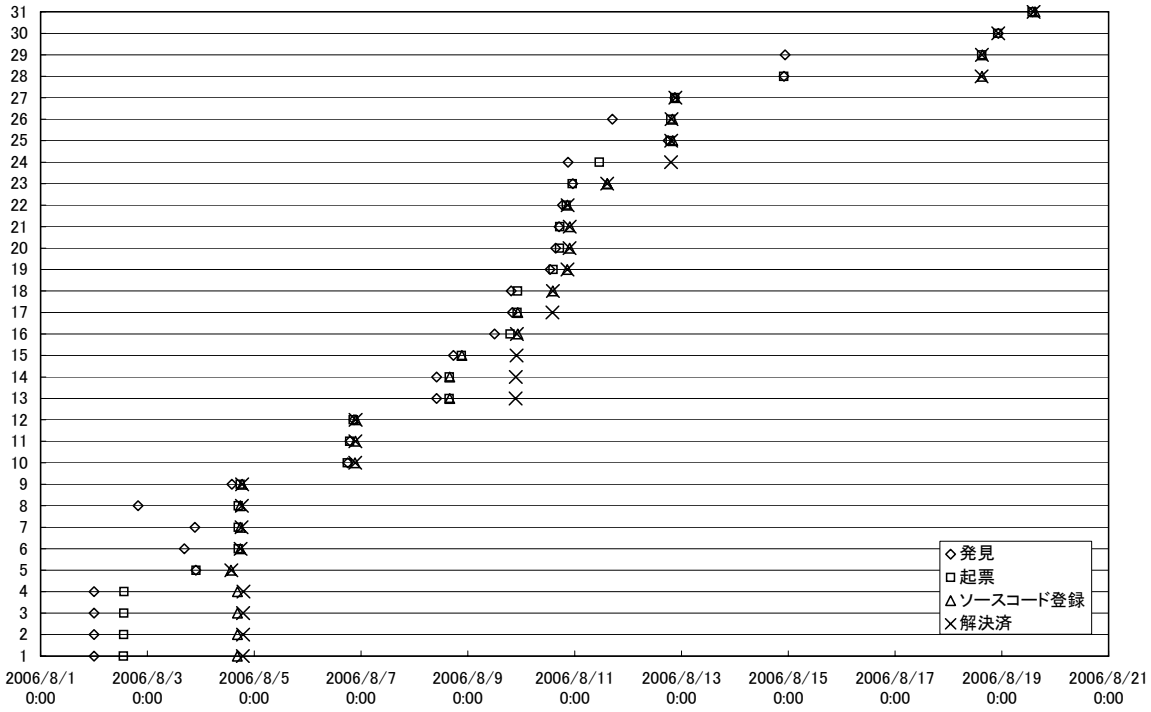

図 4 手順の時系列プロット

- 実際にはバグ修正が完了しているにもかかわらず, GNATS 上では完了していないことになっているバグが 3 件あった.
- 発見日時が登録日時よりも後になっているものが 7 件あった.
- GNATS 登録よりも先にソースコードを CVS に登録しているものはなかった.
- GNATS のステータスをソースコードの登録よりも先に「完了」にしているものが 7 件あった.
- GNATS の登録情報の誤り, 漏れが 7 件あり, 全て発見日時の誤りであった.
- ソースコードを CVS に登録する際の情報に漏れや誤りはなかった.

図 4 は横軸を日時, 縦軸をバグ番号とし, 発見(①発見日時), 起票(②GNATS 登録), ソースコード登録(③構成管理ツールに修正ソースコード登録), 完了(④GNATS ステータス「完了」に変更)をプロットしたものである. 図 4 からわかるように, 8 月上旬には一括登録, 一括修正, 一括完了が多い. 中旬以降では, 一括登録は少ない. 下旬には,「ソースコード登録」と「完了」の手順が入れ替わるものが多かった. また, バグ番号 1～4 まではほぼ同時に発見, 登録, ソースコードの登録, 完了されており, このうち, 3, 4 のソースコード登録は一括登録(一度の CVS Commit 操作)である. バグ番号 6, 7, 8, 9 は発見日時はそれぞれ異なるが, 起票はほぼ同時であり, 6, 7, 8 のソースコード登録は一括登録であ

表 2 対象データから得たマイクロプロセスメトリクス

項目	該当件数	割合
手順漏れ(②)	0	0%
手順漏れ(③)	2	6.5%
手順漏れ(④)	3	9.7%
手順の不遵守(①→②)	7	22.6%
手順の不遵守(②→③)	0	0%
手順の不遵守(③→④)	7	22.6%
登録情報漏れ, 誤り(②)	7	22.6%
登録情報漏れ, 誤り(③)	0	0%

る. 10, 11, 12 は互いに近い日時で登録されているものの, 一括登録されているものではない. バグ番号 18 以降は, 発見, 起票, ソースコード登録が比較的短時間で実施されている. ただし, 28, 29 は対応中に夏季休暇を含んでいるため, 間隔があいている.

4.3. 考察

対象プロセスインスタンスの数は 31 個であり, ここから一般的な結論を導くことは困難であると考えるが, マイクロプロセス分析により, 手順漏れ, 手順の不遵守, 登録情報漏れ, 登録情報の内容誤り, それぞれの所要時間を確認することができた. また, 修正ソースコードの登録忘れ, GNATS ステータスの「完了」状態への変更忘れを指摘することができた. 対象プロジェクトでは, ダブルチェックや開発担当者の臨機応変な対応により実際の問題は起きなかったが, マイクロプロセス分析による指摘は潜在的に大幅な作業やり直しや手戻りにつながるものである.

表 2 のマイクロプロセスメトリクスと図 4 に示した各手順間の所要時間を計測することにより, 手順忘れの候補を指摘することができる. たとえば, 「③構成管理ツールに修正ソースコード登録」のイベントが発生してから 24 時間以内に GNATS ステータスが「完了」になっていなければ, それを指摘することにより, 手順遵守を支援することができる. 過剰支援が作業の妨げとならないように, 支援は表 1 に示す手順を守らなかったときのリスクの大きなものを対象とすべきである.

マイクロプロセス分析は, 開発中に実施したプロセス(プロセスインスタンス)の品質だけでなく, マイクロプロセスモデル自体の改善や評価に用いることもできる. たとえば, 適用例では発見→起票の手順の不遵守が多く発見された. この手順が守られていない場合のリスクはそれほど大きくないので, これら 2 つの手順を 1 つに集約したり, 短時間であれば手順が入れ替わっていても許容したりするなど, マイクロプロセスモデルを改善する際の参考とすることができる.

適用例では, 複数のバグに対する独立した複数のソースコード修正の登録を 1 回のソースコード登録(1 度の CVS Commit 操作)で済ませた手順が観察された(バグ番号 3, 4). このような一括登録は時間間隔が短ければ大きな問題にはつながらない上に, 作業時間の上でも効率的である. しかし, 時間間隔が長い場合には, 大幅な作業のやり直しや繰り返し, デグレードを招く場合があるので考慮が必要となる.

最後に本手法における課題について考察する.

第一に, 分析コストの問題が挙げられる. 今回の適用例では, マイクロプロセスモデルによる作業記録の解釈を手動で実施した. 対象データが工程の一部, かつ, 開発機能の一部であり, 小規模であったため, 特に問題なく一人の分析者が数時間で解釈を行えたが, さらに大規模で複雑なプロセスを含むエンピリカルデータを対象としたときの作業量は飛躍的に増大する. 今後実用的レベルでマイクロプロセス分析を実施するには, 自動的なプロセス解釈ツールやメトリクス計測ツールの開発が必須である.

第二に, 収集できるエンピリカルデータの質(作業記録の質)が, 管理ツール(今回の場合, CVS や GNATS)の運用方法により左右されるため, マイクロプロセスモデルの定義や計測ツールの運用方法を十分に考慮する必要がある. たとえば, 障害登録やソースコード変更の登録作業をある程度の量ためておいて, 一時にまとめて行うような運用では, 開発者の実作業の振る舞いがエンピリカルデータに反映されず, マイクロプロセス分析の適用自体が困難となる. このようなことを避けるには, 障害発生やソースコード更新など作業の実施タイミングと管理ツールの使用タイミングを一致させることが必要である. 将来的には, 開発者に負担をかけずに, 計測の粒度や精度をより向上させることのできる計測環境を整えることも望まれる.

5. まとめ

本稿では, エンピリカルデータをもとに, プロセスの振る舞いに関して従来よりも細かい視点で分析を行う「マイクロプロセス分析」アプローチと, それによって測定可能となる定量的なプロセスメトリクスの概念を提案した. 時系列データからプロセスの振る舞いに関するメトリクスを測定することでプロセスに対してより直接的で詳細な分析・評価が可能になると期待される.

本提案を実践する試みとして COSE プロジェクトで収集されたエンピリカルデータの一部に対して, 予備的な分析を行った. 分析対象の規模や期間が限られるため, 複雑なプロセス構造に対する評価は行えなかったが, 手順の不遵守や作業遅延を表す定量値が実際に測定可能であることが確認できた.

今後は, 先に挙げた課題の解決法を検討するとともに, より大規模なエンピリカルデータを対象としたマイクロプロセスメトリクスの計測やそれらのメトリクスの妥当性の検証などを行っていきたい.

謝辞

本研究の一部は, 文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた. 本研

究の進行にあたり COSE 参加企業, IPA ソフトウェアエンジニアリングセンタ 神谷芳樹氏, 樋口登氏, 奈良先端科学技術大学院大学松本健一教授をはじめ EASE プロジェクトの関係諸氏に多くのご協力を頂いた.

参考文献

[1] CMMI Product Team: CMMI for Systems Engineering / Software Engineering / Integrated Product and Process Development / Supplier Sourcing. Version 1.2. CMU / SEI-2006-TR-008 (2006)

[2] Fenton N., and Neil M.: A Critique of Software Defect Prediction Models. IEEE Transaction on Software Engineering, Vol.26, Issue 5, pp. 675-689 (1999).

[3] Hayes, and Will: "The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers," CMU/SEI-97-TR-001. (1997)

[4] Johnson P.M., Kou H., Agustin J.M., Zhang Q., Kagawa A. and Yamashita T.: "Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH", Proceedings of International Symposium on Empirical Software Engineering (ISESE2004), pp. 136-144 (2004).

[5] 大平 雅雄, 横森 励士, 阪井 誠, 岩村 聡, 小野 英治, 新海 平, 横川 智教, "ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム," 電子情報通信学会論文誌 D-I, Vol.J88-D-I, No.2, pp.228-239, (2005).

[6] Shepperd M., Schofiled C., : Estimating Software Project Effort Using Analogies, IEEE Transaction on Software Engineering Vol. 23, No. 12, pp. 736-743 (1997).

[7] Srinvasan K., Fischer D., : Machine Learning Approaches to Estimating Software Development Effort, IEEE Transaction on Software Engineering, Vol.21, No.2, pp. 126-137 (1995).

[8] Takabayashi, S., Monden A., Sato S., Matsumoto K., Inoue K., and Torii K.: The Detection of Fault-Prone Program Using a Neural Network. Proceedings of the International Symposium on Future Software Technology '99 pp. 81-86 (1999).

[9] Torii K., Matsumoto K., Nakakoji K., Takada Y., Takada S., and Shima K.: "Ginger 2: An environment for CAESE (Computer-Aided Empirical Software Engineering), IEEE Transaction on Software Engineering, vol. 25, no. 4, pp. 474-492 (1999).

[10] Xing F., Guo P., and Lyu M. R.: A Novel Method for Early Software Quality Prediction based on Support Vector Machine. Proceedings of the 16th International Symposium on Software Reliability Engineering, pp. 213-222 (2005)