社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2003-48 (2004-03)

Empirical Project Monitor: プロセス改善支援を目的とした定量的開発データ自動収集・分析システムの試作

大平 雅雄† 横森 励士†† 阪井 誠††† 松本 健一† 井上 克郎††
鳥居 宏次†

† 奈良先端科学技術大学院大学 〒630-0192 奈良県生駒市高山町 8916-5
†† 大阪大学 〒560-8531 大阪府豊中市待兼山町 1-3
††† 株式会社 SRA 先端技術研究所 〒164-0004 東京都渋谷区四谷 3-12
E-mail: †ohira@empirical.jp, {matumoto, torii} @is.aist-nara.ac.jp, ††{yokomori,inoue}@ist.osaka-u.ac.jp, †††sakai@sra.co.jp

あらまし ソフトウェアの生産性や信頼性向上を目的とするプロセス改善は急務の課題となっているが，実際のソフトウェア開発現場におけるプロセス改善の実施は困難を伴う場合が多い．その原因の一つとして，一貫性のある定量的なデータを継続的に測定することが難しいという問題が挙げられる．本稿では，ソフトウェア開発におけるプロセスデータを自動的に収集・分析する Empirical Project Monitor (EPM) を紹介する．EPM は，開発者に特別な負担をかけることなく開発プロセスに関する定量的データを自動的に収集し，プロセス改善のために有益な分析結果を提示するシステムである．

キーワード ソフトウェアプロセス改善，エンピリカルソフトウェア工学，データ自動収集，定量的データ分析

Empirical Project Monitor: A System that Automatically Collects and Analyzes Quantitative Development Data toward Process Improvement

Masao OHIRA†, Reishi YOKOMORI††, Makoto SAKAI†††, Ken-ichi MATSUMOTO†, Katsuro INOUE††, and Koji TORII†

† Nara Institute of Science and Technology, 8916-5, Takayama, Ikoma, Nara 630-0192, Japan
†† Osaka University, 1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
††† SRA Key Technology Laboratory, Inc. 3-12, Yotsuya, Shinjuku, Tokyo, 164-0004, Japan
E-mail: †ohira@empirical.jp, {matumoto, torii} @is.aist-nara.ac.jp, ††{yokomori,inoue}@ist.osaka-u.ac.jp, †††sakai@sra.co.jp

Abstract In recent years, improvement of software process is increasingly gaining attention. However, its practice is not easy because of difficulties of data collection and utilization of collected data. In this paper, we introduce Empirical Project Monitor (EPM), which is an automatic data collection and analysis system to support software process improvement. Collecting data from common tools used in software development, EPM analyzes the data automatically and provides graphical results. EPM assists and facilitates coherent data collection which is a difficult task in practice.

Key words Software Process Improvement, Empirical Software Engineering, Automatic Data Collection, Quantitative Data Analysis

1. はじめに

近年，ソフトウェアシステムの社会的役割が多様化し，かつその重要性が増す中で，ソフトウェアの生産性および信頼性の向上を目的としたソフトウェアプロセスの改善は急務の課題となっている．一般的にソフトウェアプロセスの改善は，ソフト

ウェアプロセスの現状分析，新しいソフトウェアプロセスの設計，新しいソフトウェアプロセスの評価と展開の3つの段階を順次，継続的に行うことで達成されるものである[1].

プロセスの現状分析では，現状のプロセスを理解し問題点の洗い出し等を行い，次に，プロセスの設計では，現状プロセスの分析結果に基づいて改善策の立案等を行う．そしてプロセスの評価では，新しく設計されたプロセスが以前のプロセスに比べてどの部分が，どのように，どれくらい改善されたか等について評価を行う．これらのステップをより適切かつ効果的に行う，すなわち，より良いプロセス改善を進めるためには，客観的で一貫性のある定量的データに基づいてプロセスの分析・設計・評価がなされることが望ましい．

ソフトウェア開発に関する諸計測技術は，ソフトウェアプロセスおよびソフトウェアプロダクトの理解，管理，制御，予測等を行う際の有効な手段としてこれまで数多く提案されてきた[2]．一般に，ソフトウェア計測のためのデータ収集に要するコストは決して低いものではないが，多くの企業では多大な労力を払いデータの収集を行っている．

しかしながら，ソフトウェア計測に関する十分な経験がなければ，プロセス改善のための目標を明確に設定できない，分析に必要なソフトウェアやメトリクスが何かわからない，分析結果を次の改善活動に結びつける道筋が定かではない等，提案されているメトリクスや収集データを活用し効果的にプロセス改善を進めることは難しい[3]．また，関係者の配置転換等，人員の流動によって組織内にデータ収集に関する十分な経験が蓄積されにくいという場合も少なくないため，一貫性のないデータ収集方法（方針）が継続的なプロセス改善を困難にする原因ともなっている．

このような問題に対処するためのアプローチとして，GQM (Goal-Question-Metric) 法がある[4],[5]．GQM 法は洗練された計測技法の一つである．GQM 法では，計測のための目標 (Goal) を設定し，その目標を達成するために必要な要件を明確にするための質問 (Question) を挙げる．そして，それらの質問に基づき計測モデルと計測手順を決定する．GQM 法を利用したソフトウェアプロセスの改善は，論理的かつ合理的な手法である．一方，その実践においては，関係者が計測プロセスの詳細について関与する必要があるため，たゆまぬ努力と高いコストが要求される．

上述の問題を要約すると，計測のための努力がソフトウェアプロセスの効果的な改善に結びつかない大きな要因は，

- 計測に関する経験の欠如
- 計測にかかるコストの負荷

の2点に集約されると捉えることができる．

これらを克服するためのアプローチとして，我々はエンピリカルソフトウェア工学[6]に取り組んでいる．定量的なデータに基づいて様々な手法，技術，ツール等を評価する一方で，産業界との協力による実データの収集を図ることで，より実践的なプロセス改善のための支援環境（エンピリカルソフトウェア工学環境 (Empirical Software Engineering Environment: ESEE) と呼ぶ.）の構築を目指す．本稿では，ESEE 構築の一環として開発中の Empirical Project Monitor (EPM) を紹介する．EPM は，開発者に特別な負担をかけることなく開発プロセスに関する定量的データを収集し，プロセス改善のために有益な分析結果を提示するシステムである．

以下，本稿の構成は次の通りである．2章では，ソフトウェアプロセス改善に関する諸問題に対処するための我々のアプローチと ESEE の概要を述べる．続く3章では，EPM を紹介し，4章では EPM をプロセス改善のために利用する利点について説明する．最後に，5章においてまとめと今後の課題を述べ本稿を結ぶ．

2. エンピリカルソフトウェア工学環境

本章では，前章で述べた諸問題に対処するための我々のアプローチについて述べる．

2.1 改善のためのサイクリックプロセス

これまで，ソフトウェアプロセス改善を支援するために，数多くのモデルが提案されてきた（CMMI[7], IDEAL MODEL[8] 等）．これらのモデルは共通して，段階的かつ継続的なプロセス改善を通じてソフトウェアの生産性や信頼性を向上させるための一連のガイドラインを提供している．

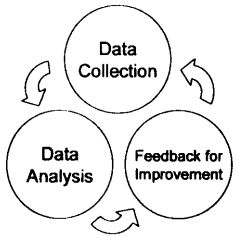

図1 プロセス改善のための基本モデル

図1は，プロセス改善に対する我々の基本モデルである．本モデルは，ソフトウェア開発データの大規模収集，集約的データ分析，プロセス改善のためのフィードバック，というサイクリックなプロセス改善活動を想定している．特に，他の既存モデルのように厳格な手順を規定するものではなく，ソフトウェア開発プロジェクトおよび組織の問題に沿って柔軟な改善活動を行うためのモデルである．このような柔軟な改善モデルに基づく支援環境は，

- 個人の経験に依存しない定量的な計測
- 特別な作業負荷やコストを必要としない計測

を行えるように設計・構築することが重要であると考える．これらの2点は，前章で挙げた問題を解決するための我々の計測活動支援のための基本方針でもある．

2.2 ESEE の設計方針

現在我々は，柔軟な改善モデルを実現するために，以下に述べる3つの設計方針に沿って支援環境 ESEE を構築中である．

データ収集のための方針：

- 改善目標の設定優先ではなく，大規模データ収集を優先

(理想的には改善のための目標設定が重要であるが，あらかじめ明確な改善目標を設定すること自体が困難な場合が少なくないため)

- プロダクトデータを中心に収集（データ収集のための管理者・開発者の余分な負担を最小限に抑えるため，プロセスデータはプロダクトデータから可能な限り抽出する）
- 人為的操作のない加工されていないデータを収集（ドキュメント等，主観的なデータが混入しやすいものを排除するため）
- リアルタイムでのデータ収集（現状を即座に把握し開発プロセスの管理・改善に連動させるため）
- 広く利用されている開発支援システムからデータを収集（開発形態に依存しない多様なプロジェクトも支援の対象とするため）

データ分析のための方針：本研究では，以下に挙げる順序で段階的に実データを分析しつつ，その有効性を探る予定である．

（1） 単一のプロジェクトを対象とするプロセス・プロダクトメトリクス（簡易，容易）

（2） 複数のプロジェクト間のメトリクス

（3） プロジェクトの分類・進化予測

（4） コンポーネント・専門知識の再利用

（5） さらなる複雑・高度な分析

本研究では，定量的かつ大規模なデータを分析処理する手法および技術を開発し，組織の利益に直結するようなフィードバックを提供する仕組みを構築する．個々の管理者・開発者あるいは単一のプロジェクトを支援の対象にした研究やツールは数多く提案されてきたが，数千規模のプロジェクトおよび組織を支援の対象にする研究はほとんど見当たらない．大規模データから有用な知識を抽出する方法を探るために，上記のように簡易なメトリクスから順次，複雑・高度な分析手法を開発し，実データに適用する方針である．

フィードバック提供のための方針：我々は，プロジェクトおよび組織のそれぞれの目的に応じたフィードバックを提供する柔軟なシステム構築を目指しており，現在様々なツールや手法の評価を行いながら，ESEE を構築中である．3 章で紹介する Empirical Project Monitor (EPM) は，ESEE 構築の一環として開発したシステムである．EPM は，開発者に特別な負担をかけることなく定量的な開発データを収集し，プロセス改善のために有益な分析結果を提示するシステムである．

2.3 アーキテクチャ

ESEE は図 1 のすべてのステップを支援する単一の巨大な支援環境ではなく，分析の目的や方法に応じて交換可能なプラグイン形式の様々なツール群から構成される柔軟なシステムである．ESEE の本質的な機能は，(1) 数千規模のプロジェクトの開発データを収集し，(2) その膨大な収集データをプロセス改善および組織の利益に直結させるために分析結果（フィードバック）を提供するという 2 点にある．

図 2 は ESEE の概略図である．ESEE は，データ収集，フォーマット変換，データ蓄積，データ分析・可視化の 4 つのコンポーネントから構成されている．以下では，それぞれの機能について簡単に説明する．

データ収集：構成管理システムやメーリングリストマネージャ等，広く普及している開発支援システムから数千規模の開発プロジェクトデータを自動的に収集する．ソフトウェア開発に一般的に利用されているシステムからデータを収集することで，データ収集専用の新たなシステムの導入や，不慣れなシステムの利用にかかる負担を避ける狙いがある．

フォーマット変換：収集データをプロダクトデータと（プロダクトデータから抽出した）プロセスデータに分類する．プロセスデータは XML 形式のデータに変換する．プロダクトデータを直接利用するのではなく，一度中間形式に変換するので，分析目的に応じた様々なデータにも対応が可能である．

データ蓄積：プロダクトデータとプロセスデータをそれぞれデータベースに格納する．プロダクトデータはコードクローン検出 [9] のようにソースコード等の生データを必要とする分析に利用する．プロセスデータは作業進捗，作業効率，生産性等，開発作業全体の特性を分析するために利用する．

データ分析・可視化：コードクローン検出，ソフトウェアコンポーネント検索 [10]，協調フィルタリングによる類似プロジェクト同定 [11]，プロジェクト分類 [12] 等，様々な分析・可視化ツールを提供予定である．

このように，ESEE は膨大なデータを数千規模のソフトウェア開発プロジェクトから自動的に収集し，分析結果を提供するシステムである．プラグイン形式でシステムへ機能を追加することができる仕組みによって ESEE を拡張性の高い柔軟なシステムにする．プロジェクトや組織によってプロセス改善のためのデータ分析の目的は異なる場合が多いため，様々なプロジェクトおよび組織を支援する上で，この仕組みは非常に重要であると思われる．このことが，柔軟性のない（追加的な分析要求に対応しにくい）完全なシステムを構築するよりも，定量的データ収集基盤というべき基本的な枠組みを提供することを重視する理由である．

3. Empirical Project Monitor (EPM)

本章では，ESEE 構築の一環として実装中の Empirical Project Monitor (EPM) を紹介する．

3.1 EPM の概要

EPM はプロセス改善の支援を目的とした，開発データの自動収集・分析システムである．EPM は，ソフトウェア開発において一般的に利用されているシステムから開発活動に関するデータを収集し，プロセス改善のために有益な分析結果をユーザに提供する．EPM は，実際の開発現場では困難がつきまとう一貫性のある定量的データ収集とデータ利用を容易にする．

EPM は ESEE の基本コンポーネントから構成されている（図 2 中の灰色部）．ESEE の枠組みに沿って，EPM は構成管理システム，メーリングリストマネージャ，障害管理システムという，ソフトウェア開発支援として広く普及し利用されているシステムからデータを収集する．

EPM は Ruby で記述されたスクリプトによって分析対象のデータを XML 形式に変換する．XML 形式のデータは各コンポーネントとのインターフェースとして公開されている．現在 EPM は，構成管理システムとして CVS，メーリングリストマネージャとして mailman, Majordomo, fml，障害管理システム

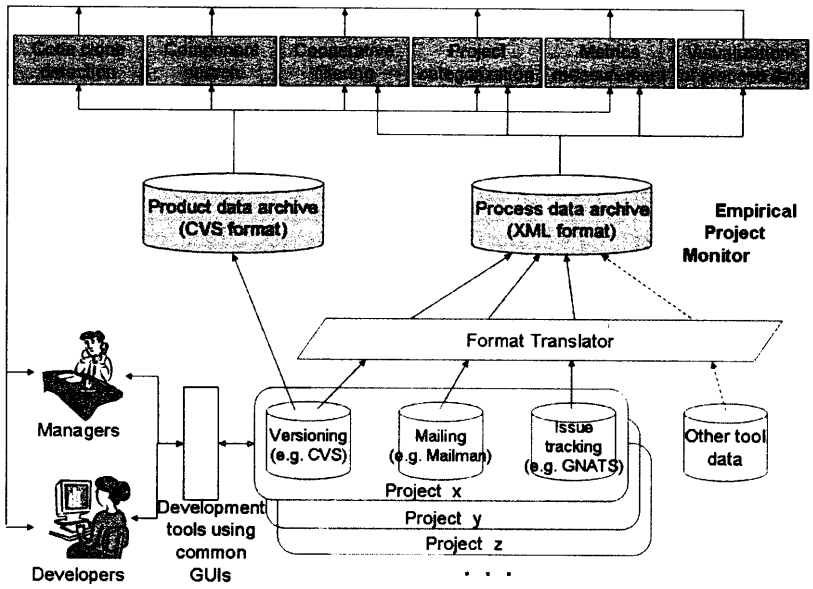

図 2　ESEE のアーキテクチャと EPM

として GNATS, Bugzilla のデータを XML に変換可能である．これらは（特にオープンソース）ソフトウェア開発プロジェクトで一般的に利用されているシステムであるため，データ収集のために管理者・開発者に特別な作業や導入コストをかけない．これらのシステムは改造せず，コマンドの実行結果を出力したログの解析や，明確に定義された保存ファイル・DB からの抽出によってデータを得ており，最新バージョンへの対応が比較的容易な構造になっている．これら以外のシステムのデータについては，現在のフォーマットとのデータ項目の対応付けを行い XML 形式に変換することで利用可能である．

XML 形式のデータは EPM の内部では PostgreSQL データベースに格納される．格納されたプロセスデータはユーザ（管理者・開発者）の要求に応じて分析結果を提示する．データは随時収集され，リアルタイムで分析されるため，ユーザは容易に現在のプロジェクトの状況を理解することができる．

例えば，構成管理システムからは，ソースコードの修正，チェックイン，チェックアウト等のイベント情報を抽出し，プロセスデータとして XML 形式のデータに変換される．同時に，CVS リポジトリ中のコンポーネントのサイズ，更新時期，バージョン等の変遷を抽出する．メーリングリストマネージャあるいは障害管理システムからは，メール，障害報告や障害修正報告等を元に，投稿時間，件名，差出人等の情報を抽出し，XML 形式に変換する．同時に，話題ごとに集計した一覧表も作成する．

分析は Java サーブレットにより行われ，結果をウェブブラウザから閲覧することができ，管理者・開発者間の分析結果の共有を促進する．分析結果共有の簡便化は，収集データを理解するために行う議論の場に有効である．

このように，EPM は定量的な開発データを低いコストで収集し，開発プロジェクトのリアルタイムでの管理を支援する．基本的な 3 種類のシステムから人為的操作のされにくいデータを収集することによって，一貫性のある分析結果を提供することができる．また，オープンソースで利用され，かつ一般的なシステムを利用するため，作業負荷・導入コストを最小限に抑える事ができる．

3.2　EPM の可視化結果

EPM は収集データから主に，ソースコードの規模 (Lines of Code)，チェックイン/チェックアウトの数，メール/障害報告の数等を計測する．それらのメトリクスを使用したデータ分析を行い，現在，EPM は 5 種類の分析結果を提示することができる．以下では，我々のプロジェクト（EASE プロジェクト [13]）を例として利用し，EPM が可視化する分析結果について述べる．

ソースコードの規模推移：図 3 は，は，我々のプロジェクトにおけるソースコード規模の推移（図中，黒の折線）を示したものである．図中，灰色の垂線は開発者が CVS リポジトリを更新（チェックイン）した時期を示しており，リポジトリ中のあるソースコードに対して追加/修正/削除等の操作を行ったことを示している．このグラフはプロジェクトの現在の進捗を把握するために役立つ．また，過去の同様なプロジェクトと比較することによりある程度の予測が可能となる．

チェックインとチェックアウトの関連：図 4 は，チェックインの時期（図中，灰色の垂線）とチェックアウト回数（図中，黒の垂線）との関係を示している．CVS リポジトリに対して開発者が更新操作（チェックイン）を行った場合，通常他の開発者はローカルにある自らのファイルを更新するためにチェックアウトの操作を行う．チェックイン後にチェックアウト数が少ない場合は，ファイルが更新されているにもかかわらず他の開発者がそれを参照していない，あるいは，その更新が他の開発者には重要ではなかったことを意味する．したがってこのグラフから，開発者がチェックイン後に CVS リポジトリを参照したかどうか，参照が少ない場合に他の開発者に知らせるべき更

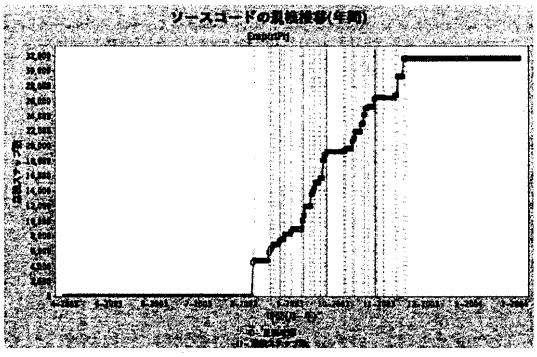

図 3　ソースコードの規模推移

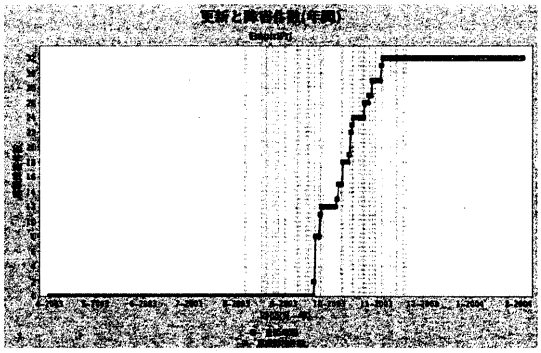

図 6　チェックインと障害件数との関連

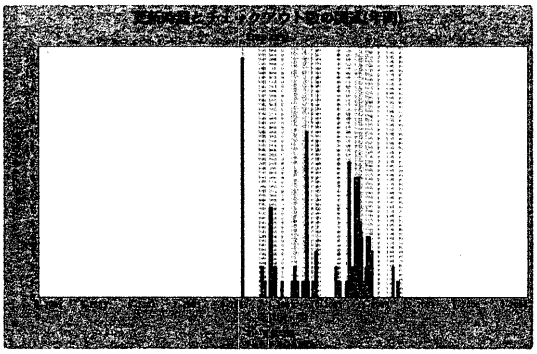

図 4　チェックインとチェックアウトとの関連

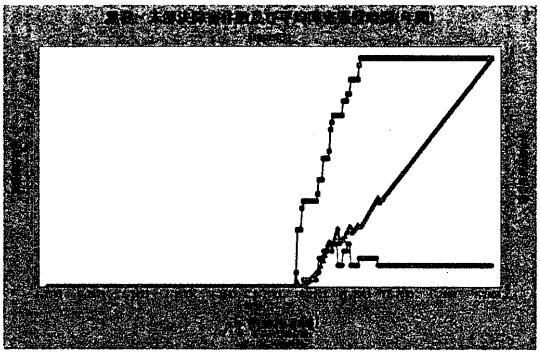

図 7　累積・未解決障害件数と平均障害滞留時間との関連

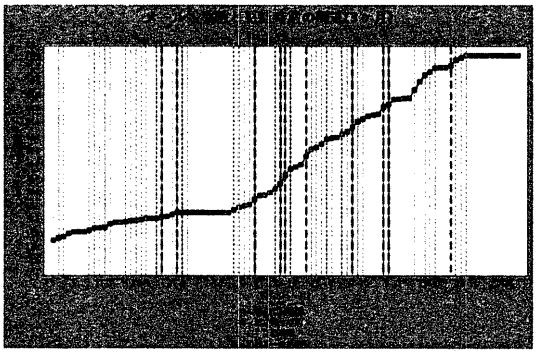

図 5　メール投稿数の推移とチェックイン/障害発生/解決時期との関連

新が存在するかどうか等の確認が行える．

メールの投稿数の推移とチェックイン/障害発生/障害解決時期との関連：図 5 は，開発者用のメーリングリストを使用して投稿されたメールの累積の推移（図中，太い黒の折線），障害発生時期（図中，細かい点線），障害解決時期（図中，粗い点線），チェックインの時期（図中，灰色の垂線）との関係を表したものである．このグラフから，障害発生/解決時期とコミュニケーションの関係を知ることができる．障害管理システムへ障害報告がなされた場合，一般的に，開発者はその障害についてメーリングリストを利用して活発に議論をはじめる場合が多い．障害が数多く報告されているにもかかわらず，開発者が議論を行っていないならば，開発者間のコミュニケーションに問題がある可能性が高い．また，このグラフの障害発生時期，(バグフィックスを行った）チェックイン時期，障害解決時期の対応から，プロジェクトの障害対応の状況も確認することができる．

チェックインと障害件数との関係：図 6 は障害報告の累積の推移（図中，黒の折線グラフ）とチェックインの時期（図中，灰色の垂線）との関係を示している．障害報告がなされた後，開発者が障害修正の結果を反映するためにチェックインが行われる場合が多い．したがって，このグラフは各バージョンごとの障害対応の状況を確認するのに役立てることができる．

累積・未解決障害件数と平均障害滞留時間との関係：図 7 は，障害報告の累積（図中，上の折線），未解決障害件数の推移（図中，下の折線），平均障害滞留時間（図中，真中の折線）との関係を示したものである．このグラフは，プロジェクト管理者が現在プロジェクトが抱えている障害件数や障害が解決されるまでにかかる平均時間を把握するのに役にたつ．また，プロジェクトが抱えている障害状況が一目できるため，開発者がバグを減らすための動機付けとしても利用することができる．

このように，EPM はプロジェクト管理のために有益な分析結果を提供し，プロジェクトのプロセス改善に利用することができる．現在の実装では，上記の 5 種類のグラフおよびスケール（週/月/年）を変更したものを提供するのみであるが，データ分析のための方針 (2) の複数のプロジェクト間の関係を分析・可視化する機能を実装予定である．また，ユーザ自身がデータベースを直接操作し，希望に応じた分析結果を提供できるようなカスタマイズ機能も実装中である．

4. EPM の利用シナリオ

本章では，プロジェクト管理に EPM を利用する利点を，ソフトウェア開発の現場でしばしば言及される以下のような問題をシナリオとして用いながら説明する．

- データを収集する際の管理者・開発者の負荷の増加

- プロジェクトの進捗状況に関連した情報交換の遅れ
- データの人手による操作

一点目は，管理者・開発者がデータ収集のために付加的な作業を求められるときに問題となる．例えば，データ収集が目的の進捗報告やプロダクト報告は，作業量の増加を敬遠する管理者・開発者にとっては負担となることは少なくない．

EPM は，一般的なソフトウェア開発において広く普及している開発支援システムからデータを収集するため，EPM を導入するにあたって付加的な作業やコストはほとんど生じない．また，たとえ CVS 等のシステムを利用していないプロジェクトであっても，フリソフトウェアを基本データ収集用システムとして使用しているため，導入コストは極めて低い．

二点目の問題は，コミュニケーションに関連したものである．多くの場合，開発者間のコミュニケーション不足は，生産性を引き落とす原因である [14]．特に，近年盛んに行われている分散開発では，対面でのコミュニケーションを行う機会が非常に少ない．遠隔地の他者とコミュニケーションをとる手段としては電子メールが日常的に利用されているが，電子メールを介したコミュニケーションでは適切なタイミングで適切な人物から正しい情報を得ることは極めて難しい [15]．

EPM の特徴の一つは，データをリアルタイムに収集し，結果を一般的なブラウザを利用して閲覧可能な点である．このことは開発者が現在のプロジェクトの状況を把握し，他の開発者と現状理解を共有するのに役立つ．特に，前章図 5 で紹介したグラフのような，リアルタイムで確認できるコミュニケーションの履歴は，コミュニケーション不足等を示す一種のアラートとして利用することができる．

最後の問題は，ドキュメント中心のプロジェクト管理においてしばしば言及される問題である．人手によるドキュメントの質は，個々人の経験の違いや人事評価を意識したデータの加工等によって客観性および一貫性が失われる場合がある．

EPM を利用したプロジェクト管理では，人手による恣意的なデータ操作は基本的に不可能なため，上述の問題は生じにくい．この特徴が，客観的なデータに基づいた一貫性のある継続的なプロセス改善活動を支援することにつながるものと考える．

5. まとめと今後の課題

本稿では，プロセス改善へ向けたの我々のアプローチとエンピリカルソフトウェア工学環境 (ESEE) の概要について述べた．また，ESEE の部分的実装として構築した Empirical Project Monitor (EPM) を紹介した．EPM は，開発者に特別な負担をかけることなく，開発プロセスに関する定量的データを自動的に収集し，プロセス改善のために有益な分析結果を提示することができる．現在我々は，様々な分析要求に対応可能なカスタマイズ機能，複数のプロジェクトデータの比較分析機能を実装中である．また，EPM の評価を目的として企業の開発現場に EPM を導入することも計画中である．

これまでわが国では，多くの課題を抱えている産業界のソフトウェア開発に関するデータが，大学等の研究者に届くことは少なく，この領域の技術進歩の大きな阻害要因となってきた．本研究ではソフトウェア産業からの協力を得てこの壁を打破し，産業界の実データと実践的な手法によって問題解決のための研究を進めたいと考えている．

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．ESEE および EPM の実装では，岩村聡氏，小野英治氏，新海平氏にご協力頂いた．深く感謝いたします．

文　献

[1] 大場充, 堀田勝美, 松浦建司, ソフトウェアプロセス改善と組織学習 CMM を毒にするか？薬にするか？−, ソフト・リサーチ・センター, 2003.

[2] 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, プロセスと環境トラック, 第 8 巻, 共立出版, 2000.

[3] L. Briand, C. Differding, and D. Rombach, Practical guidelines for measurement-based process improvement, Technical Report ISERN-96-05, Department of Computer Science, University of Kaiserslautern, Germany, 1996.

[4] V. Basili and D. Weiss, A methodology for collecting valid Software Engineering Data, IEEE Transactions on Software Engineering, Vol.10, No.6, pp.728–738, 1984.

[5] V. Basili, Applying the Goal/Question/Metric Paradigm in the Experience Factory, Presented at the 10th Annual CSR Workshop in Amsterdam, 1993.

[6] V. Basili, The Experimental Software Engineering Group: A Perspective, ICSE'00 award presentation, Limerick, Ireland, June 5–10, 2000.

[7] CMMI Product Team, Capability Maturity Model Integration (CMMI) Version 1.1 CMMISM for Systems Engineering and Software Engineering (CMMI–SE/SW, v1.1), Continuous Representation, CMU/SEI–2002–TR–001, ESC–TR–2002–001, 2001.

[8] J. Gremba and C. Myers, The IDEAL Model: A Practical Guide for Improvement, appeared in the Software Engineering Institute (SEI) publication, Bridge, issue 3, 1997.

[9] T. Kamiya, S. Kusumoto, and K. Inoue, CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, IEEE Transactions on Software Engineering, Vol.28, No.7, pp. 654–670, 2002.

[10] R. Yokomori, T. Ishio, T. Yamamoto, M. Matsushita, S. Kusumoto, and K. Inoue, Java Program Analysis Projects in Osaka University: Aspect-Based Slicing System ADAS and Ranked-Component Search System SPARS-J, Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pp.828–829, Portland, Oregon, 2003.

[11] N. Ohsugi, A. Monden, and S. Morisaki, Collaborative Filtering Approach for Software Function Discovery, Proceedings of 2002 International Symposium on Empirical Software Engineering (ISESE 2002), Vol.2, pp.45–46, Nara, Japan, 2002.

[12] S. Kawaguchi, P. K. Garg, M. Matsushita and K. Inoue, Automatic Categorization for Evolvable Software Archive, International Workshop on Principles of Software Evolution (IWPSE'03), pp.195–200, Helsinki, Finland, 2003.

[13] EASE (Empirical Approach to Software Engineering) project, http://www.empirical.jp/index-e.html

[14] A. H. Dutoit and B. Bruegge, Communication Metrics for Software Development, IEEE Transactions on Software Engineering, Vol.24, No.8, pp.615–628, 1998.

[15] J. D. Herbsleb, A. Mockus, T. A. Finholt and R. E. Grinter, An Empirical Study of Global Software Development: Distance and Speed, Proceedings of the 23rd international conference on Software engineering (ICSE'01), pp.81–90, Toronto, Canada, 2001.