ソフトウェア開発プロジェクト管理支援のための構成管理・障害管理データの活用

松村知子† 門田暁人†

†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
E-mail: {tomoko-m, akito-m}@is.naist.jp

1．はじめに

奈良先端科学技術大学院大学（NAIST）および大阪大学が進めている産学連携の EASE(Empirical Approach to Software Engineering)プロジェクト[1]では，日本のソフトウェア開発企業に対して，データ収集，分析，およびフィードバックという実証的なアプローチによる開発プロセスの改善の枠組みの導入を支援している．本プロジェクトは 2003 年 4 月より文部科学省の 5 年プロジェクトとして開始された．

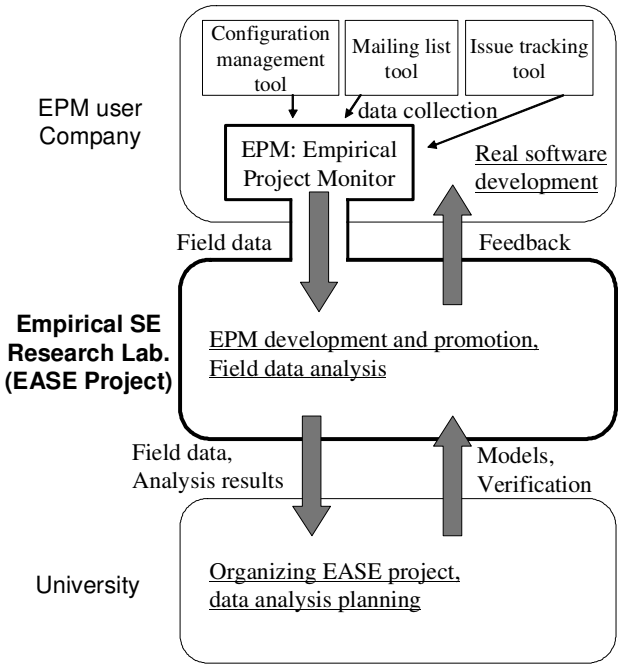

図 1 EASE プロジェクトの体制

[1] http://www.empirical.jp/

図 1 に EASE プロジェクトの体制を示す．同プロジェクトでは，実際のソフトウェア開発企業がプロジェクトを遂行している現場で発生する問題を，現場において解決するための体制を整備している．EPM(Empirical Project Monitor)は現場の開発技術者に負担をかけることなく，プロジェクト管理に必要なデータ収集を行うために開発された[1]．EPM は，開発で一般的に利用される開発支援ツール(以降，CASE ツールと呼ぶ)，すなわち構成管理ツール，障害解析ツール，メーリングリスト管理ツールなどから，自動的に時系列のデータを収集する．産学双方の研究者は，ソフトウェア工学ラボラトリ（大阪・千里中央）[2]において，EPM の開発・普及，および収集データの分析や企業・社会へのフィードバックを目的に活動している．

本稿では，EPM 収集データから加工して得られる定量的データ(メトリクス)を用いた実プロジェクトの計測事例について報告する．まず，2 章で EPM のアーキテクチャやその機能について説明する．3 章では適用事例について説明する．まず，適用対象プロジェクトについて説明し，次に適用したメトリクスについて述べる．次に，実際に計測した中から 2 つの組織について紹介し，それぞれから検出できた事象やその原因について述べる．最後にこの適用結果を考察し，今後の課題について述べる．

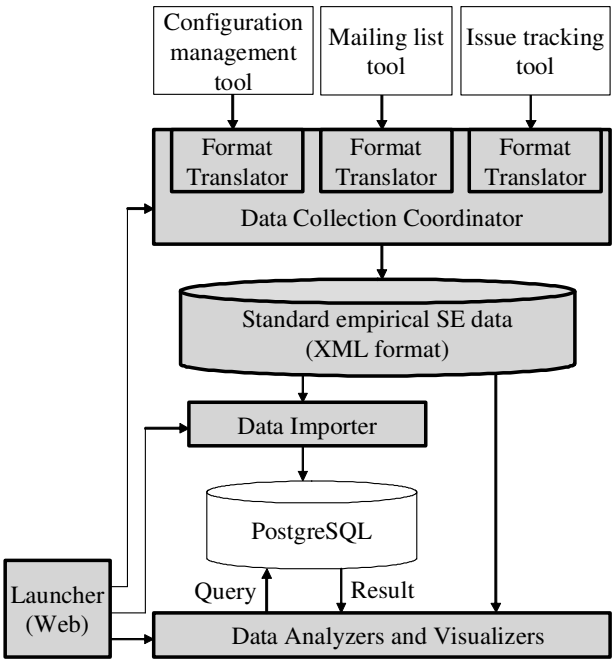

図 2　EPM アーキテクチャ

[2] http://www.empirical.jp/j_access.html

２．EPM(Empirical Project Monitor)

図 2 に EPM のアーキテクチャを示す．EPM は，3 つのタイプの CASE ツール，すなわち，構成管理ツール（CVS など），障害管理ツール（GNATS など），メーリングリスト管理ツール（Mailman など）からデータを収集する．これらの CASE ツールは，日常的に開発プロジェクトの作業中に利用され，開発者や管理者がこれらのツールからデータを収集するにあたり，追加的な作業を必要としない．

収集されたデータは，XML 形式に変換された後，PostgreSQL のデータベースに変換され，ここから様々な分析・加工データを取得，可視化することができる．これらすべての作業は，Web ベースのランチャー経由で実行できる．

EPM は，障害管理ツールに登録されるすべての障害データ，プログラムコードの追加・修正・削除などの履歴データ，メールの送受信データを随時収集することができる．障害データには，障害の内容や重要度，混入原因のほか，発見日や登録日，対応完了日が含まれる[2]．構成管理データからは，履歴データのほかプログラムのサイズを時系列で計測したデータを出力する．分析・可視化の技術として，SRGM（Software Reliability Growth Model）やパレート図，各種メトリクス（規模，バグ数，メール数など）の時系列グラフに対応している．一方で，現在のところ，対象となる CASE ツールでは収集できないコスト管理や予定－実績管理などには対応していない．

本稿では，EPM をデータ収集ツールとして用い，出力される XML 形式の中間データを他のアプリケーションを使って分析する．分析のために，EASE プロジェクトでは C によるデータ加工プログラムを開発し，さらに Microsoft-Access や Excel の計算機能・グラフ化機能などを用いた．これらの処理は，本稿での事例研究後，機能を改善・追加し，１つのツールとしてまとめられた[3]．

３．適用事例

3.1　適用プロジェクト

本稿で適用対象としたプロジェクトは，今日の日本における情報システム開発の典型とも言える「マルチベンダ開発」である．具体的には，経済産業省の支援を受けて進められた中規模の情報システム開発プロジェクト「プローブ情報システム開発プロジェクト」を対象とした．本プロジェクトは，2 年にわたって新規開発(一年目)，処理の共通化と機能追加（二年目）が行われたが，本稿の事例は一年目のデータを用いて行う．

このプロジェクトでは，EASE プロジェクトとソフトウェアエンジニアリング

技術研究組合(以降，COSE と呼ぶ)[3]，情報処理推進機構（IPA）下に設置されたソフトウェアエンジニアリングセンター（以降，SEC と呼ぶ）[4]の 3 者が連携して，データ収集体制の構築とデータ分析を行った．COSE はユーザ的立場の企業とプロジェクト管理担当企業，開発担当５社から成り，ユーザ的立場の企業が要求を出し，開発担当５社が分担してサブシステムの開発を行い，プロジェクト管理担当会社が 5 社を統括した全体のプロジェクト管理を行う．開発はウォーターフォールプロセスにより進められ，具体的には，要件定義を受けて基本設計を全社の協働で行い，以降各社で詳細設計，プログラム設計，製造，単体試験，社内結合試験が行われた．その後，各社のプログラムを集め，組合全体として社間結合試験，総合試験が行われた．

システムのサイズは，約 330K ステップで，主に C/C++のプログラムで構成されているが，一部スクリプト，バッチファイルなどが含まれる．ファイル数は，約 1400 で，開発期間は基本設計から総合試験まで約 10 ヶ月である．

3.2　計測メトリクス

本事例で用いたメトリクスは表 1 のとおりである．

メトリクス名は，本稿で計測したデータの日本語名である．以降，本稿の事例中では，略称を用いる．計測方法は，計測元になるデータリポジトリやデータの取得・計算方法などについて記述している．計測理由は，そのメトリクスがプロジェクト管理に有用である，と考えた理由である．参考文献には，同様のメトリクスをプロジェクト管理に用いた既存の研究や事例について述べている．計測の目的や活用方法などから本稿で扱うメトリクスと同等に扱うことができる，もしくは代用できると推定されるものについても挙げている．

「21 世紀へのソフトウェア品質保証技術：日科技連ソフトウェア品質管理研究会 10 年の成果」[4]は，日本企業での品質管理，プロジェクト管理の事例をまとめており，事例中で具体的に用いられたメトリクスについて紹介している．「CMM によるプロセス改善入門」[8]は，CMM に基づいたソフトウェア開発プロセス管理を行うメトリクスについて，具体的に様々な例を上げ，キープロセスエリアとの関連や可視化方法などを示している．

---

[3] ソフトウェアエンジニアリング技術研究組合（COSE）：COnsortium for Software Engineering

[4] http://sec.ipa.go.jp/index.php

表 1　計測メトリクス一覧

メトリクス名	本稿中の略称	計測方法	計測理由	参考文献
プログラム規模	KSLOC	プログラムのステップ数．Unix(Linux)の wc[5]コマンドで計測．コメントや空白行を含む.	コーディングの進捗を管理する基本的なメトリクス.	計画，追跡(進捗)，見積もりのベース[5] 進捗管理，予実管理（機能漏れ，見積もり・設計ミス，後工程圧迫予想，リソース不足の発見支援）[4](p.408)[7] 規模，ソフトウェア拡大度指標[8]
変更回数	FCtotal	ファイル更新回数．EPM の更新履歴から，ファイルの登録・更新(修正)の回数を集計．ファイルの削除は含まない.	ファイル更新回数は，作業の密度を示す．通常，プログラム規模は開発後半では安定し，あまり変化が見られないが，一方でコーディングミスやバグ修正などで小規模な修正が頻繁に行われている場合もある．このメトリクスを用いて，作業の頻度を把握することができる.	ソフトウェア成熟度，将来工数予測(保守性)[8]

[5] wc コマンドは指定されたファイルのファイルの行数(-l)、ワード数(-w)、バイト数(-c)を計算し、表示する.

メトリクス名	本稿中の略称	計測方法	計測理由	参考文献
一定規模以上行削除された変更回数	FCdel	一定規模（例えば10行以上）の行削除を伴うファイル更新回数. EPMの更新履歴から，削除行数が一定規模以上の更新(修正)の回数を集計．ファイルの削除を含む.（ファイルを削除した場合，ファイルサイズ分の行が削除されたものとみなす)	一定規模以上のファイル削除を伴う変更からは，手戻りの頻度を推定できる．手戻りの中には，コーディングミスのほか，仕様変更，設計変更などが考えられる．一方，一定規模未満の小規模な削除は，重要度の低い修正と考えられるため，除外する．	手戻り作業数[6]
追加行数	FLadd	EPM の更新履歴から，追加行数を集計．ファイルを登録した場合，登録されたファイルサイズを計上する．	追加行数は，実際にコーディングを行った実生産量，と認識とする．この量が多い場合，作業は生産(コーディング)フェーズにある，と考えられる．また，実規模(KSLOC)に対して，この値が大きい場合，生産効率が悪いと考えられる．	開発作業量[7] 新規開発量（生産性）[4](p.408)
削除行数	FLdel	EPM の更新履歴から，削除行数を集計．ファイルを削除した場合，削除されたファイルサイズを計上する．	削除行数は，手戻り量，と認識とする．一度追加したコードを削除する場合，生産効率が悪いと考えられる．また，削除行数が多い場合，仕様変更・設計変更など重大な問題に対応したとも考えられる．	手戻り作業量（ソフトウェア成熟度(誤り)/ソフトウェア変更性(改善)）[6]

メトリクス名	本稿中の略称	計測方法	計測理由	参考文献
コーディングバグ数	Cbug	コーディングミスによるバグ数．障害追跡ツールのバグの混入工程（バグの原因となった工程）によって分類する．コーディング工程以降（テスト工程を含む）に混入したバグを集計.	コーディングの品質を計測する．コードレビューや単体試験で多くなると考えられるが，この数が異常に多い場合は，開発者のコーディング技術に問題があると思われる．また，コーディング工程以降の混入は，修正時の混入，デグレードなどが考えられる．	コーディング品質[4]
設計バグ数	Dbug	設計ミスによるバグ数．障害追跡ツールのバグの混入工程（バグの原因となった工程）によって分類する．コーディング工程より前に混入したバグを集計.	設計の品質を計測する．本来，設計レビューなどにより各工程内で発見されることが望ましい．テスト工程で見つかる場合は，早期に見つかるほど，バグの重要度・影響度も小さいと考えられる．	設計工程への手戻り件数[4](p.410)
平均ファイル変更者数	Ave-FOwners	1 つのファイルを変更した開発者数の平均．ファイル毎に，変更履歴のファイル変更者のアカウント名から異なる変更者の数を計測し，その平均値を求める．	通常，要員の配置や分担が明確で作業が順調に行われていれば，1 ファイルの変更者は一人であると考えられる．この値が大きい場合，不適切な要員配置や，システム構造が複雑でバグの修正や設計の変更の為に他人のファイルを修正する必要が生じたことなどが考えられる．また，要員の変更なども影響する．	なし

メトリクス名	本稿中の略称	計測方法	計測理由	参考文献
2名以上で変更されたファイル数	Fmul	二人以上の開発者に変更されたファイル数．変更履歴のファイル変更者のアカウント名から異なる変更者の数を計測し，二人以上で変更されたファイル数を計測．	Ave-FOwnersと同じ．ただし，ファイル数を見ることで，実際の問題が発生している範囲を知ることができる．この値が大きい場合，要員配置に関する問題がシステムの広範囲に発生していると考えられる．	なし
累積バグ数	Iss	全発見バグ数．障害追跡ツールに登録されたバグの総数．	バグの収束状況を見る．これが単体試験工程で，急激に増加しない，結合試験など後工程でいつまでも増加する場合，問題が存在すると考えられる．	ソフトウェア信頼性[8] 累積件数の収束状況[4](p.366,p.369)
残存バグ数	Iss-Unres	未修正のバグ数．障害追跡ツールに登録されたバグで，状態(Status)が完了・無効以外のもの．これから対応が必要，もしくは対応中のバグ数．	障害の修正作業の進捗状況を見る．増加し続けたり，バグ発生が収束したのに，0にならない場合，要員不足等でバグ対応が遅れている，重大な問題が発生して対応に時間がかかっているなどの要因が考えられる．	(修正されたバグ数)品質の改善[8] 未完了手戻り作業(ソフトウェア虚弱性(誤り)/スケジュールリスク(改善)) [6]
平均滞留時間	Ave-Iss-Duration	未修正バグの発見からの経過時間の平均値．障害追跡ツールに登録されている残存バグ(Iss-Unres)の発見日時からの経過時間の平均値．	残存バグの対応状況を詳細に見る．増加し続ける場合，バグの滞留が長期化し，バグの対応に問題が発生していると考えられる．一方，残存バグがあっても，滞留時間が増加しない場合，バグは発見されたものから順次処理されている，と推定できる．	問題は迅速に解決されているか？[4](p.404) バグ修正の組織的能力[8]

3.3　データ収集準備

表 1 のメトリクスを計測するに当たり，対象プロジェクトの各開発組織に構成管理ツール，および障害管理ツールの運用について，以下のような依頼を行った.

- 構成管理システムへのファイルの登録は，単体試験開始前とする．コーディング中から運用することも可.
- 単体試験以降は，1 バグ修正単位でファイルの更新を行うことを基本とする．ただし，単体試験初期で多数のバグが発生する場合，複数バグを一括修正し登録することを許容する．その場合でも，修正後 1 週間以内に更新する.
- バグ修正時は,構成管理システムへの登録時に修正したバグ番号を入力する．バグ修正以外の修正についても，その理由を明記する.
- CVS でブランチ機能や Export・Import 機能などは極力用いない．履歴に明確に表現されないため.
- 障害は，極力発見直後に登録するが，それが不可能な場合，発見日に発見された日時を記入する.
- バグの完了状態への更新は，構成管理システムへの修正ファイル登録後とし，構成管理と同様，複数バグの一括更新は極力しない.

3.4　分析結果の可視化とその解釈

図 3 図 4 に 3.1 節で述べた適用対象プロジェクトの 2 組織における各メトリクスの計測結果を示す．以降，1 つ目の組織をプロジェクト A，他方をプロジェクト B とする．横軸は，開発開始からの経過時間を示し，縦の挿入線は大雑把な工程の区切りを示す．Unit test は単体試験，Intgr. Test は社内結合試験，Cross-company intg. Test は社間結合試験，System test は社間総合試験を示す．　1 組織内では複数コンポーネントの作業が並行して行われているため，工程区切りはあまり明確ではない．KSLOC や Iss は累計値である．FC，FL，bug，Fmul などは，1 週間に発生した作業量を集計している(累計ではない).

以降，各組織の分析結果を詳細に説明する.

3.4.1　プロジェクト A の分析

プロジェクト A の分析結果を図 3 に示す．プロジェクト A では，Dbug も少なく，FCtotal や FCdel，FLadd，FLdel はコーディングや単体試験工程では大きい値を示すが，社内結合試験以降(42 日目以降)とても小さく，特に大きな問題が発見されず，開発が順調に収束していることを示している.

一方，コーディング作業中の FCtotal に対する FCdel の割合や FLadd に対する FLdel の割合が，約 3 分の 1 と高くなっている．これは，開発担当者へのインタビューにより，コードの静的解析ツールの導入により，頻繁にコードの改善を行っていたため，ということが判明している．

Ave-FOwners や Fmul は社間結合試験以降で増加している．これは，設計や仕様変更による 1 開発者による広範囲な修正が発生した，と考えられたが，開発者へのインタビューでは，要員の変更が発生したため，と判明した．また，Fmul の 84 日目あたりの増加は，ある開発者によりコメント等の見直しが行われ，システム全体に対して一人のプログラマによる修正が行われたためである．

Iss や Iss-Unres は，56 日目あたりまで同じ値を示す．これは，すべての発見されたバグが修正されずに一ヶ月以上放置されているように見られる．しかし，インタビューによりほとんどのバグは発見直後に修正されたが，障害管理ツールへの状態更新を 56 日目あたりで一括して行ったため，と判明した．また，Ave-IssDuration が，結合試験以降，増加し続けているが，これは 1 つのバグが未完了状態でプロジェクトの終了時まで残っていたためである．これについても，修正されていたが，障害管理システム上の更新忘れということがわかっている．

同組織においては，コーディング品質，設計品質，要員配置等の問題は発生していないが，特に障害管理ツールの運用に関して問題が認められた．

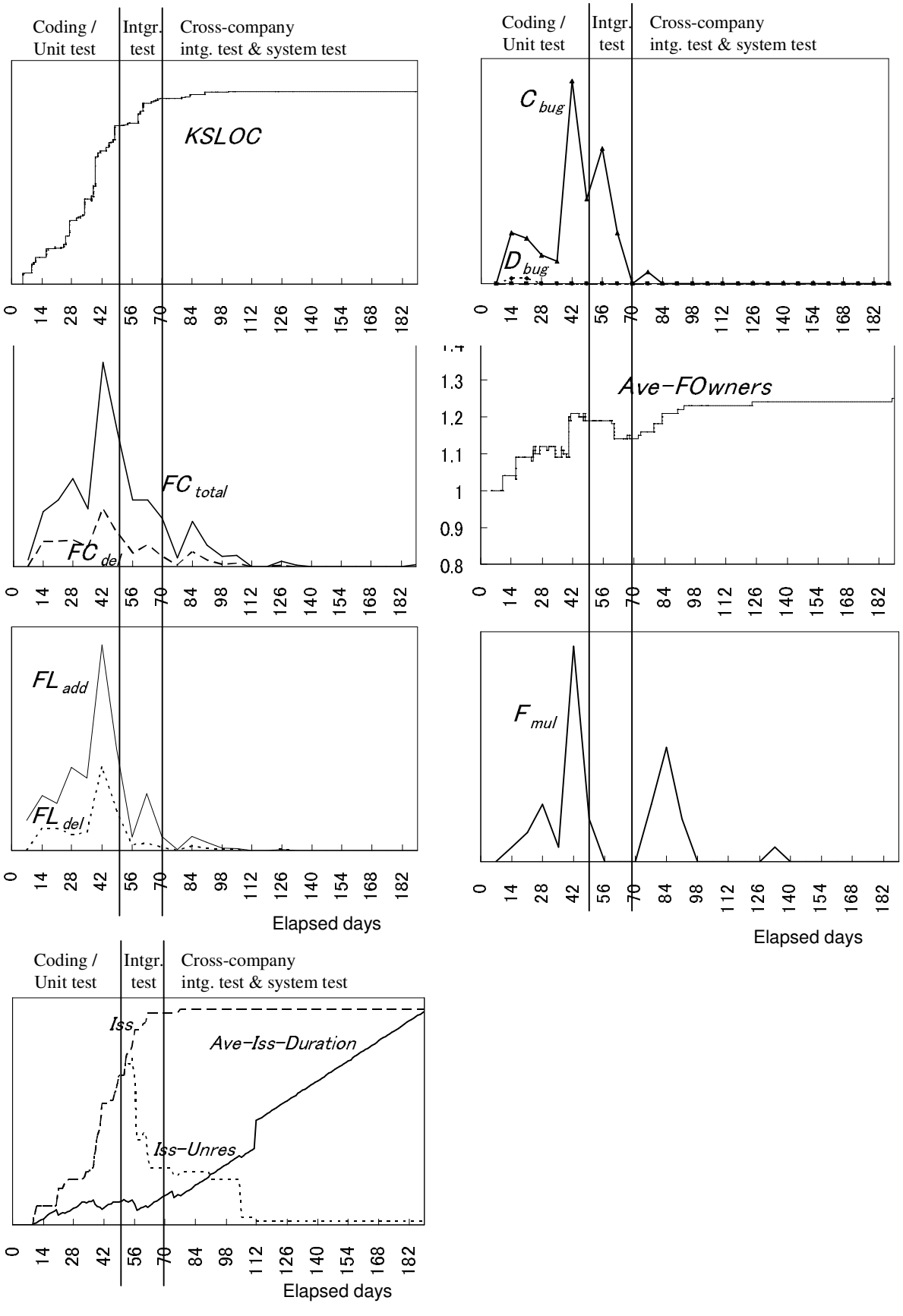

図 3　プロジェクト A の分析メトリクス

3.4.2 プロジェクトBの分析

図 4 にプロジェクト B の分析結果を示す．プロジェクト B では，FCdel が大きな特徴を示している．単体試験工程より社内結合試験，社間結合・総合試験で大きい．これは，広い範囲で大きな変更が行われたことを示し，仕様や設計の変更が疑われる．また,Dbug が社内結合試験で発見されている．また，いったん収束したコーディングバグ数も社内結合試験で再度増加している．このことから，コーディング品質 や設計品質に問題があったことが推定できる．実際に，インタビューにより同組織においては設計レビューが不十分であったことがわかっている．また，社内結合試験で見つかったバグのいくつかは，バグ修正時に混入した障害であることも，障害データの混入工程情報から判明している．単体試験の前半だけで設計バグが見つかっているプロジェクト A と比較すると，対照的である．また，120 日目あたりで設計バグが見つかっているが，これは性能に関する問題である．

35 日目と 70 日目あたりで Ave-FOwners や Fmul が増加しているが，これは要員変更によるものである．この点では，Fmul のほうが要員変更を検出するには有用である，と言える．

Iss-Unres は社間試験工程に入っても減少せず，それに伴って Ave-IssDuration も増加し続けている．これは，修正が困難な性能に関する問題がなかなか解決されずに残存していたためであった．また，70 日目あたりで要員が変更され，バグの滞留時間が増加しているが，他の残存バグのためあまり明確に現れていない．前述したとおり，Fmul のほうが明確に検出できる．

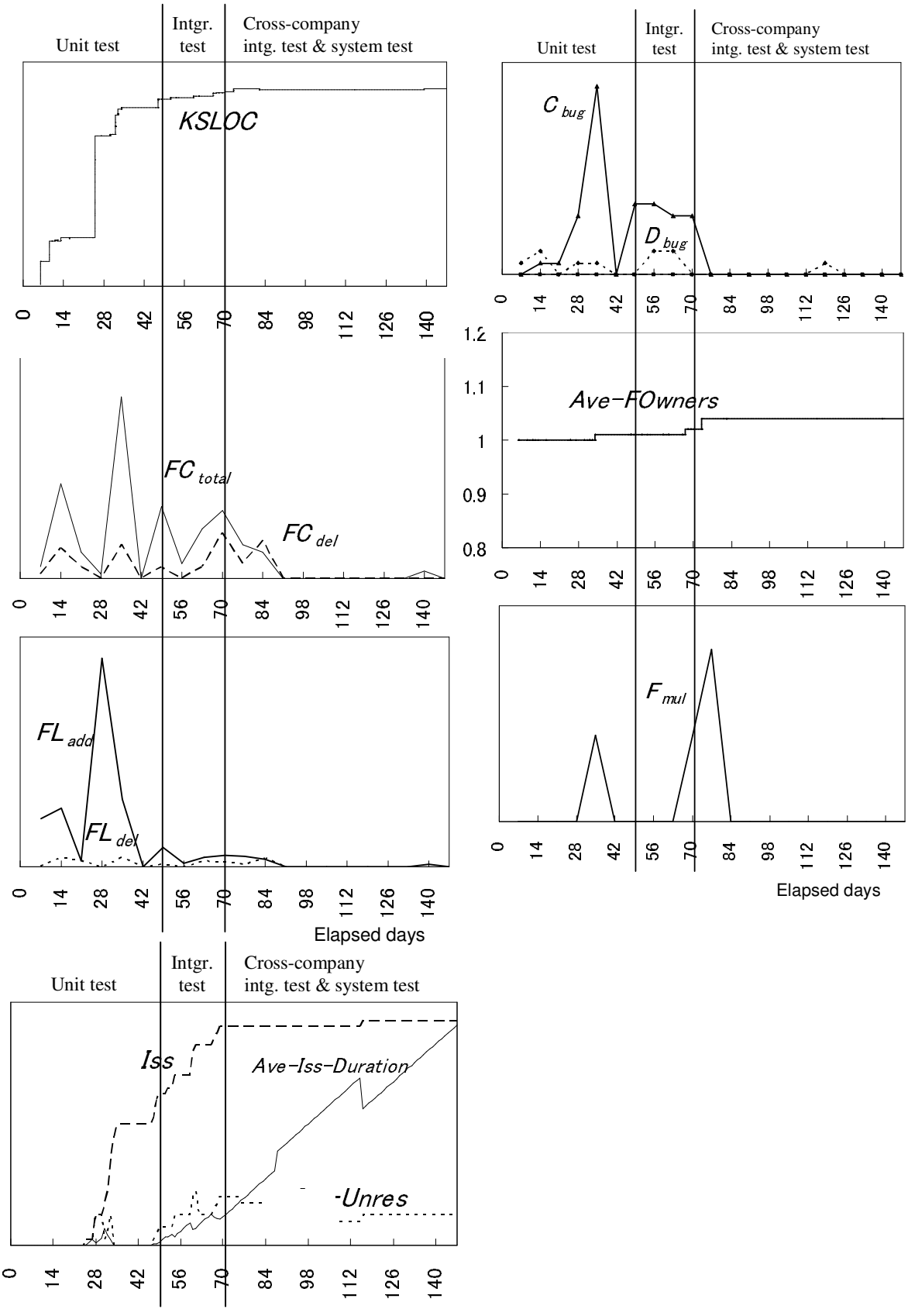

図 4　プロジェクト B の分析メトリクス

４．考察

3章の適用事例から，発見できたプロジェクト管理上の問題症状は次のとおりである．

- プロジェクト B で，Dbug(設計バグ数)，FCdel（一定規模以上行削除された変更回数）から設計品質に問題があることが推定できた
- プロジェクト B で，Ave-IssDuration，Iss-Unres，Fmul から要員配置の変動が判明した
- プロジェクト B で，Cbug からコーディング品質に問題があることが推定できた
- プロジェクト A では，Ave-IssDuration，Iss-Unres，から要員配置に関する問題があると推定されたが，これは不適切な障害管理ツールの運用が原因であることがわかった

適用実験では，本プロジェクト管理者および開発者に対して，組織毎に 2～3回フィードバックミーティングを行い，ここで指摘した事象について，各チーム内でいずれもすでに把握されていたことを確認した．しかし，プロジェクトマネージャやプロジェクトリーダは，問題が定量的に確認できるのは大変有益である，と評価している．また，プロジェクトマネージャからは，問題の発生箇所を特定するため，もっと細かい粒度（例えば，モジュール単位）の分析をすることが必要である，という意見を得た．

５．まとめ

本稿では，EPM ツールにより構成管理ツールや障害管理ツールから自動的に収集・計測できるデータを用いたソフトウェア開発作業の可視化と，そこからの問題発見の試みについて報告した．利用したメトリクスは，いずれもプロジェクトで発生する諸問題を定量的に確認するのに有効であった．具体的には，1)設計品質の問題，2)要員配置の問題，3)コーディング品質の問題，4)不適切な障害管理ツール運用の問題を検出することができた．

我々は，さらにここで利用したメトリクスを精選し，プロジェクトマネージメント上での有効な利用のフレームワークを確立するために，さらに適用実験やツールの改善，機能追加を行っている．

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．本研究にあたり多くのご協力を頂いたソフトウェアエンジニアリング技術研究組合参加企業，ソフトウェアエンジニアリングセンター，EASE プロジェクトの関係諸氏に感謝します．

引用文献

[1] 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, VolJ88-D-I, No.2, pp228-239, 2005.

[2] 松村 知子, 門田 暁人, 森崎 修司, 松本 健一, “マルチベンダ情報システム開発における障害修正工数の要因分析,” 情報処理学会論文誌, Vol.48, No.5, pp.1926-1935, May 2007.

[3] 玉田 春昭, 松村 知子, 森崎 修司, 松本 健一, “プロジェクト遅延リスク検出を目的とするソフトウェア開発プロセス可視化ツール ProStar,” NAIST-IS-TR2007002, February 2007

[4] 菅野文友, 吉澤正監修, 日科技連ソフトウェア品質管理研究会編, “21 世紀へのソフトウェア品質保証技術：日科技連ソフトウェア品質管理研究会 10 年の成果, ” 日科技連出版社, 1994.

[5] Park, R. ″Software Size Measurement: A Framework for Counting Source Statements, CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh, 1992.

[6] Royce, W. “Pragmatic quality metrics for evolutionary software development models. “ In Proceedings of the Conference on TRI-ADA ’90 (Baltimore, Maryland, United States, December 03 - 06, 1990), pp.551-565,1990.

[7] 会澤実　白井保隆　大木雅彦　杉山昭洋　吉崎浩二, “構成管理システムを活用したソフトウェア開発プロセス改善への取り組み事例, ” 情報処理学会第 66 回全国大会論文集, pp.181--182, 2001.

[8] Joseph Raynus, 富野壽監訳, “CMM によるプロセス改善入門, ” 共立出版, 2001.