INFORMATION
SCIENCE
TECHNICAL
REPORT

NAIST-IS-TR2008004
ISSN 0919-9527

構成管理情報の定量的計測による異常モジュール検知手法の提案

松村知子，勝又敏次，森崎修司，玉田春昭，
角田雅照，門田暁人，松本健一

March 2008

NAIST

〒 630-0192
奈良県生駒市高山町 8916-5
奈良先端科学技術大学院大学
情報科学研究科

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan

構成管理情報の定量的計測による異常モジュール検知手法の提案

松村知子[i]　勝又敏次[ii]　森崎修司[i]　玉田春昭[i]　角田雅照[i]
門田暁人[i]　松本健一[i]

概要

プロジェクト管理のためのデータ収集と分析，それに基づくプロセス改善などの活動は，ソフトウェアの品質向上や開発の効率化のための重要な活動である．文部科学省の「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ている EASE(Empirical Approach to Software Engineering)プロジェクトでは，現場の開発者やプロジェクト管理者の負担を最小限にして，プロセス改善に有効なデータを収集・分析する手法の研究を行っている．データを半自動的に収集・分析するツールの開発と，そのツールに基づくプロジェクト管理およびプロセス改善のフレームワークの確立を目的として，2005 年度から 2 年間にわたり経済産業省の支援の受けて進められた情報システム開発プロジェクトにおいて，実践的な産学連携の活動を行ってきた．

本稿は，EASE プロジェクトで開発したツール群により計測・分析した結果を用いて，プロジェクト中の自動的な問題検出の試みについて報告する．用いたツール群は，構成管理システムや障害管理システムから自動的にデータを収集し，分析することを目的に開発された．しかし，実際に開発プロジェクトへの有用な情報のフィードバックには，技術的知識的な背景を持つエキスパートによる手作業が必要とされていた．本稿では，エキスパートの作業を一部肩代わりし，定量的な分析によって有用な情報，ここでは注意が必要な異常の検出を行う方法を提案する．また，実プロジェクトでのデータに基づいて，プロジェクト進行中に人手によって異常や問題が検出されたモジュールと，同手法を用いて異常が検出されたモジュールの比較による評価を行う．

[i] 奈良先端科学技術大学院大学　情報科学研究科
[ii] 株式会社 NTT データ

1．背景

1．1　エンピリカルソフトウェア工学研究における EASE プロジェクト

奈良先端科学技術大学院大学および大阪大学が進めている EASE(Empirical Approach to Software Engineering)プロジェクト[1]では，進行中のソフトウェア開発プロジェクトを定量的に計測することによるプロジェクトの進捗管理やリスク管理を支援する研究を行っている．特に，EASE プロジェクトで開発した EPM(Empirical Project Monitor)ツール[1]は，構成管理システム，障害管理システム，メーリングリスト管理システムからの自動的なデータ収集によって，プロジェクトのリアルタイム計測を支援する．

さらに，同プロジェクトでは，構成管理システムデータや障害管理データから有用と考えられる定量的なデータ(メトリクス)の加工と可視化を支援する EPM－ProStar[5]を開発した．メトリクス群は「プロジェクト遅延」や「障害対応遅延」に繋がる問題発生を検出するという目的の下，GQM パラダイム[4]に従って抽出された．GQM パラダイム[4]とは，分析目的を明確にした上でトップダウンに計測を行うフレームワークで，メリーランド大学の Basili 教授らによって提唱された．Goal(目的)は，多くの現場関係者にアンケートを行った結果，「プロジェクトの遅延の主な要因」として指摘された「要求の不安定」「設計の不完全」「劣悪な品質」「不適切な要員配置」を検出することとした．2004 年 11 月 Basili 教授を招いて，構成管理ツールと障害管理ツールからの収集データを用いて，上記の目的に従って GQM モデル構築ワークショップを行った．

2005 年度，同ワークショップで検討したメトリクスは，いくつかの商用開発プロジェクトに適用実験を行い，現場開発者からのフィードバックなどを得て改善を行い，現場の視点から有用と考えられるメトリクスが抽出された．そのメトリクスと適用結果について，[7]に報告されている．さらに，2006 年度，よりプロジェクトマネージャや開発者の実感に合った実用的なメトリクスを抽出した上，それらを適用してプロジェクト管理支援を行った[6]．

1．2　数社適用後の課題

1．1 節で述べたメトリクスの実プロジェクトへの適用結果，我々はいくつかの同メトリクスを用いたプロジェクト管理支援について，いくつかの課題を抽出した．

- モジュール別分析の必要性：プロジェクトは多くのサブシステムやモジュールから構成され，多人数よって並行して開発されるため，開発プロジェクトを 1 単位とした計測では，問題の発生を捉える計測値の変化や異常値が不明瞭になる．また，異常の発生をメトリクス上で検知しても，どのサブシステム・モジュールで発生しているか特定できないと，実用的ではない．
- 累計値による微小な変化の見逃し：規模，変更頻度，変更行数など，時系列の累計値

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

で見た場合，グラフ上では累計値が大きいほど変化量が正規化されることによって，実際の変化量を把握できなくなる．1000 行のモジュールにおける 100 行の変更と，500 行のモジュールにおける 100 行では，変化量の重大さが前者は小さく評価されるが，実際には 100 行の変更はモジュールサイズにかかわらず重大な問題が存在する場合が多い．

- 工程・フェーズによる計測値の変化：プログラムの変更回数や規模の変化量など構成管理システムから計測できる値は，開発のフェーズによって変化する．コーディングフェーズとシステム試験フェーズでは，変更量の異常と思われる値も変化する．したがって，一律に固定の閾値による異常検出を行うことは実情に合わない．さらに，コーディングフェーズでも変更行数は，後半になるほど収束すると思われる．
- 開発期間のずれ：並行開発を行う場合でも，一人の開発者が複数のモジュールを担当する場合，それぞれのモジュールの開発期間がずれる．そのため，標準的な開発作業が進行していても，作業間に生じる空白期間によって，グラフ化したときに同じような変動のパターンを示すとは限らない．

1．3　エキスパートによるプロジェクト管理での適用の流れ

我々は，メトリクスをプロジェクト中の進捗管理（問題検出を含めた）に適用するに当たり，長年プロジェクトマネージャの経験を持つ人と検討を行い，エキスパートの経験に基づくメトリクスの活用の流れを次のようにまとめた．

まず，通常の進捗管理では，各開発者（請負企業）からの進捗報告に基づく．進捗報告は，計画値（最終値）と，それに対する進捗率（N%完了），完了予定日などが含まれる．さらに，フェーズによって報告する値は異なる．コーディング工程ならば規模（LOC 等），テスト工程ならばテスト項目数（消化数）となる．また，モジュールの特性（優先度，複雑度，難易度，再利用率等）を加味して，進捗率や残作業量から進捗の遅れや作業の停滞を問題として対策を取ったり，経過観察したりする．このとき，対策の要否判定には，エキスパートの長年の経験から蓄積された問題ケースの知識が用いられる（図 1 参照）．

一方，我々が提示する構成管理情報から抽出されるメトリクスの活用方法としては，図 2 のようなフローで見るのが効果的であるとの意見を得た．まず，規模(LOC 等)を見るが，変化が大きい場合は，なんらかの異常発生を疑う．もし変化が微少あるいは無しの場合は，変更回数を確認する．変更回数が大きい場合は，異常を疑う．さらに変更回数が少ない場合でも，追加行数や削除行数が大規模な場合，異常を疑わなければならない．この異常の疑いに対して，図 1 のようなさまざまな情報を加味して，実際の異常を検出することができる．

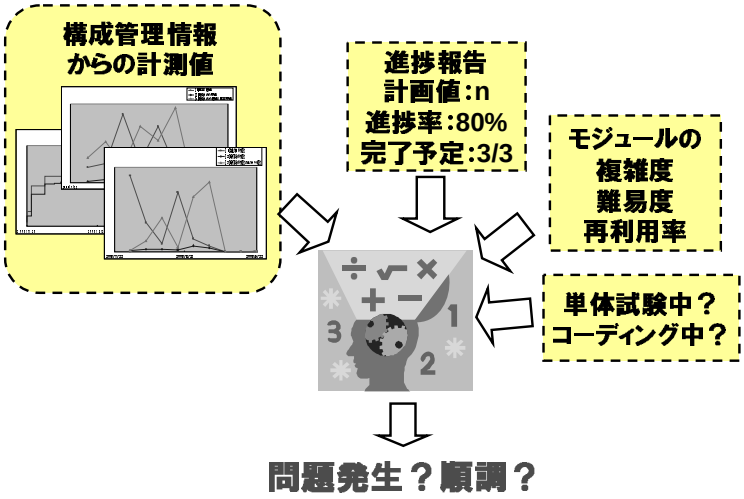


図 1　エキスパートによる進捗状況判断

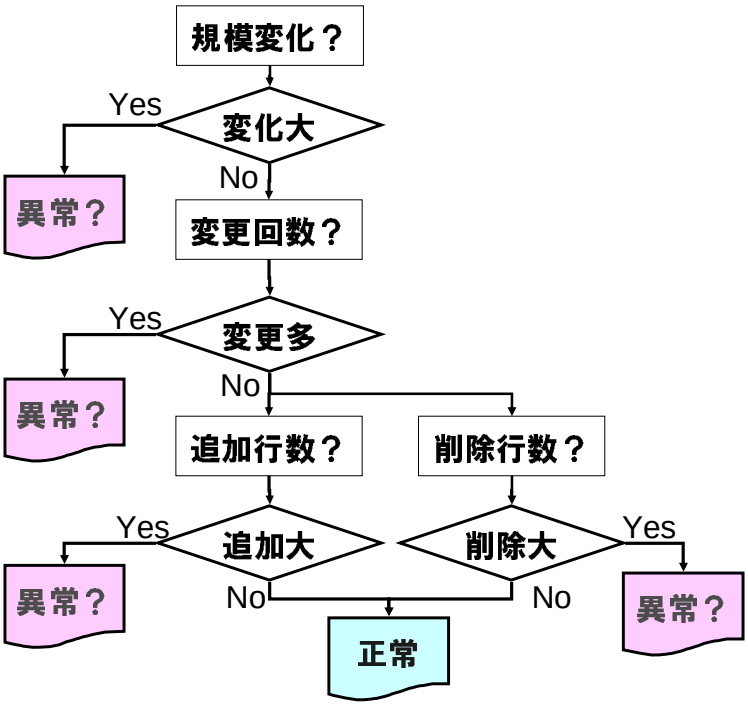


図 2　構成管理情報からのメトリクスの判定フロー図

1．4　プロジェクトの透明性の確保

近年のソフトウェア開発プロジェクトは，開発体制の複雑さ（アウトソースやオフショアの増大）によってユーザや一次請負のベンダからも開発プロセスが不透明になってきている．また，ソフトウェアの開発コストの低減や品質の向上のため，既存ソフトウェアの再利用も進んでいる．既存ソフトウェアを他社から購入したりする場合，通常そのソフトウェアの開発プロセスは全く触れることはできない．

「次世代IT基盤構築のための研究開発」の委託によるSTAGEプロジェクト[2][1][2]では，「エンピリカルデータに基づくソフトウェアタグ技術の開発と普及」を目的として研究活動を行っている．「ソフトウェアタグ」とは，ソフトウェアの品質情報や開発プロセスの要約データを包含しソフトウェア製品に付加されるもので，発注者や一般ユーザにデータを開示することで，安心してソフトウェアを購入・利用することを目的に考案された[8]．この場合，ソフトウェアタグとして付加されるデータは，エキスパートのプロジェクトマネージャが見るだけではなく，発注者（顧客）や一般ユーザである場合もある．そのため，データの評価も専門知識を必要としないような手法が必要となる．

２．本研究の目的

本研究では，構成管理システム情報から得られる定量的なデータ(メトリクス)から，開発過程で異常が発生していると思われるモジュールを抽出する方法を提案する．我々は，すでにいくつかのメトリクスがエキスパートによって有効に活用できる点について，実プロジェクトで確認したが，さらにこの研究ではエキスパートの知識やノウハウを用いずに，自動的に異常モジュールを検出する方法を提案する．

この手法によって，以下のようなことが期待できる．

- ソフトウェア開発やプロジェクト管理についての専門的な知識や特殊な技術を用いずに，開発過程で異常が発生しているモジュールを検出できる
- 構成管理情報から半自動的に抽出することで，プロジェクト管理者(第３者)がプログラムコードや開発要員に関する情報など管理に不必要な情報に触れる必要がなく，社間（ユーザ・ベンダ間，元請ベンダ・下請け企業）の機密が保持された上での運用が可能である
- プロジェクトの管理コストの増大なしに，新たなメトリクスの導入できる

３．アプローチ

3．1　手法の流れ

本手法の概要を図 3に示す．

まず，本手法で用いる各メトリクス（規模変更率，変更回数，追加行数，削除行数）について，時系列で値の変化を示すシーケンスデータを作成する(3．2 節)．このシーケンスデータについて，Gamma Analysis[9][10][11]を適用し，モジュール毎にメトリクスの時間に伴う変化パターンを作成する(3．3節と 3．4節)．全モジュールのパターンを用いて，標準的パターンを抽出し(3.5節)，これと各モジュールのパターンを比較することによって(3.6節)，異常な変化パターンで開発されたモジュールを検出することができる．

---

[2] STAGE：Software Traceability and Accountability of Global software Engineering

以降，各手順の詳細について，説明する．

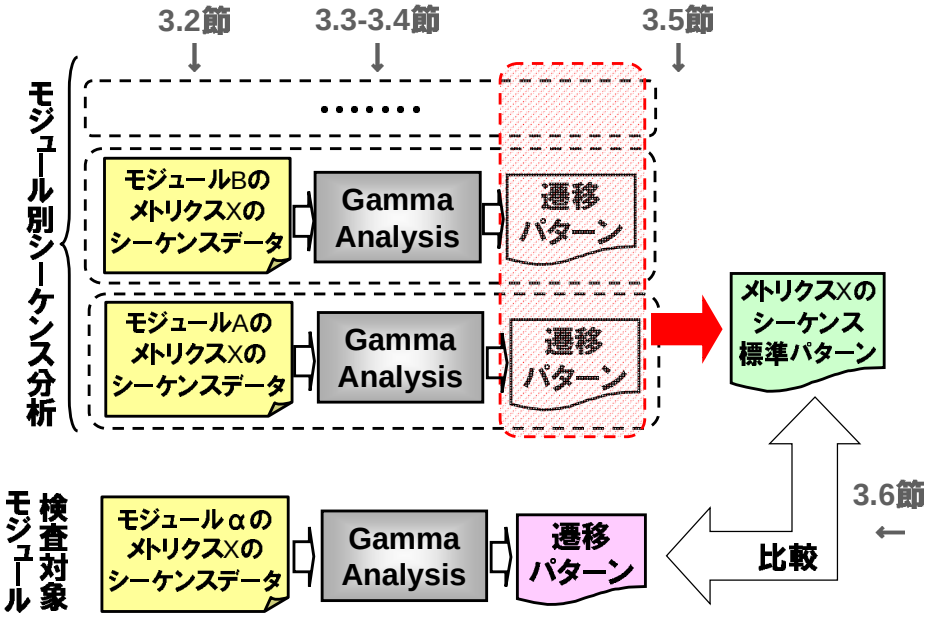


図 3　提案する手法の概要

3．2　シーケンスデータ作成

シーケンスデータは，モジュール毎，メトリクスごとに作成する．メトリクスは，規模変更率，変更回数，追加行数，削除行数を用いるが，それぞれのデータ作成方法について個別に説明する．(括弧内は，略称．以降の文中で用いる)

- 規模変更率(LChg)：ある日の変更量の前日までの全体規模に対する割合を示す．1 月 2 日の規模が 100 行で，1 月 3 日に 100 行増加した場合，変更率は 100%となる．
- 変更回数(Freq)：一定期間に構成管理上更新された回数を示す．例えば，3 つのファイルが 2 回ずつ更新されると，6 回となる．また，ファイルの削除は，この回数に含まないが，ファイルの追加は 1 ファイル 1 回で計上される．
- 追加行数(Ladd)：一定期間行われた構成管理上の更新の追加行数の合計行数とする．ファイルを追加した場合は，そのファイルの総行数を計上する．
- 削除行数(Ldel)：一定期間に行われた構成管理上の更新の削除行数の合計行数とする．ファイルを削除した場合は，そのファイルの総行数を計上する．

それぞれシーケンスデータは，日次・週次など一定の期間ごとに作成することが可能である．プロジェクトの規模，管理プロセス，管理粒度によって，調整することが必要である．

3．3　フェーズ(作業)分類(Phase Mapping)

シーケンスデータをフェーズ分類する．ここでいうフェーズとは，作業の内容を示す．例えば，追加行数について，以下のような分類を行う．

- 全く作業が行われていないフェーズ
- 作業が行われているがプログラムコードの追加が行われていないフェーズ
- 新規のプログラム作成時のように大規模な追加が行われるフェーズ
- コーディングの終了期などの中規模の追加が行われるフェーズ
- バグの修正や小規模な調整など小規模の追加が行われるフェーズ

3．4　Gamma Analysis

Gamma Analysis は，連続性のあるデータの一般的なフェーズの順序を推定する順序相関分析のノンパラメトリック分析手法の 1 つで，フェーズが明瞭に区切られていても，オーバーラップしていても使えるという特徴がある．「グッドマン・クラスカルの順序連関係数（ガンマ)」も呼ばれる[3].

分析は 2 つの事象（ここでいうフェーズ）のペアに対して行われる．事象 A，B について，「P :B に先立って，A が発生する数」，「Q :　A に先立って，B が発生する数」を計測し，Gamma Score＝(P-Q)÷(P+Q)を計算する．-1　<Gamma Score　<+1 の範囲の値が得られ，Gamma Score が-1 に近い場合は，P が Q に対して小さいことを示し，A は B の後に発生する頻度が高いフェーズ，Gamma Score が+1 に近い場合，P が Q に対して大きいことを示し，A は B より前に発生する頻度が高いフェーズと解釈される．0 に近い値は，明確な順序関係が認められない(相関がない)関係である．

3．5　標準パターン抽出(Gamma Mapping)

3. 4 節で算出した Gamma Score からフェーズの前後関係を推定することができる. 図 4 は, 3. 2 節から 3. 5 節までの処理の流れをサンプルのデータで示している. 図 4 の Gamma Table(Gamma Score 表)から，Phase4 は Phase2 や Phase3 に対して前に発生する頻度が高いことがわかる．Phase1 に対しては若干後に発生する可能性が高いが，あまり関係は明瞭ではない. Phase2 と Phase3 には明瞭な順序関係が認められない. このような解釈から，図 4 の Gamma Map を作成することができる．このデータからは，Phase1，Phase4 がシーケンスの初期に発生し，Phase2，Phase3 がその後に発生する頻度が高い.

本手法では，モジュール毎に計測した Gamma Score の平均を取り，その平均値から Gamma Mapping を行うことで，メトリクス毎に標準的なシーケンスを抽出する.

---

[3] http://www2.econ.osaka-u.ac.jp/~takeuchi/lec06/ssm/pdf/ssm06-11.pdf／
http://risya.hus.osaka-u.ac.jp/taroh/SocialStatistics/statistics.pdf

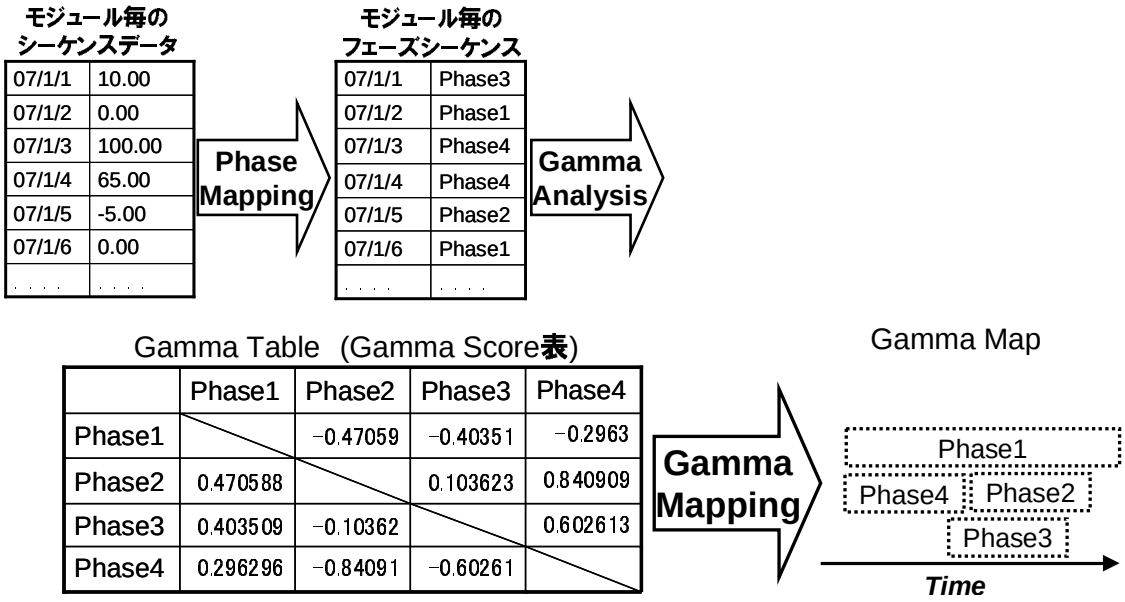


図 4　Gamma Mapping の例

3．6　異常パターンの抽出

本手法では，3．5節で計測した Gamma Table を用いて，異常パターンを抽出する方法として，以下の 2 つを用いる．

- 前後関係の逆転：標準的なパターンに対して，ペアの前後関係が完全に入れ替わる．例えば，図 4 では，Phase4 と Phase2 の前後関係が入れ替わる，など．
- 曖昧な関係と明確な関係の変化度合い：標準的なパターンに対して，順序の相関が曖昧だった関係が，はっきりした前後関係に変わる，あるいははっきりしていた前後関係が曖昧になる，などの変化が発生する度合いが多いもの．例えば，図 4 では Phase2 と Phase3 が明らかに前後関係を持つようになる，あるいは，Phase4 と Phase 3 の関係が曖昧になる，など．その変化した関係の数が多い場合，そのモジュールを異常とする．

このステップでは，具体的に以下のような手順で異常モジュールを検出する．

① 各関係を 3 つの分類する：Gamma Score が 0.5 以上の場合 Forward（対象フェーズが先に出現），Gamma Score が-0.5 以下の場合 Behind（対象フェーズが後に出現），-0.5< Gamma Score<0.5 は Middle（順序関係が曖昧）とする

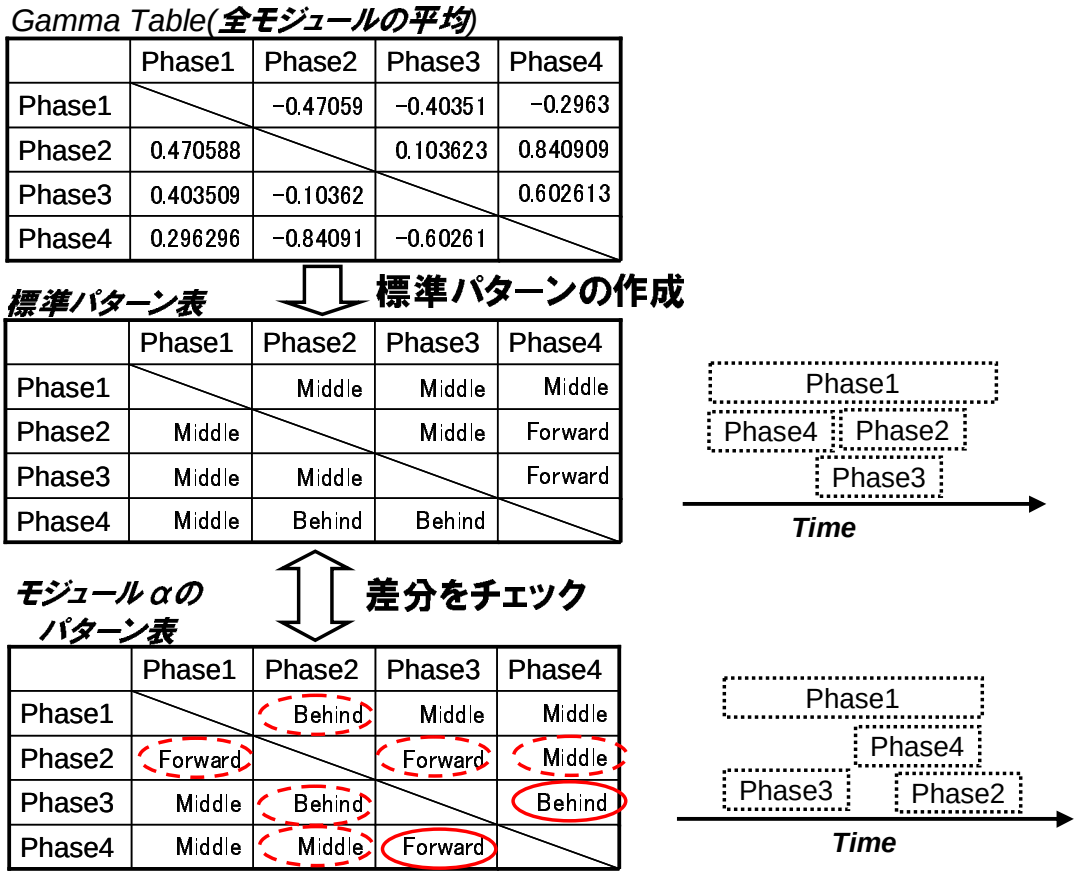
② 関係の逆転を検出し，異常モジュール(High　Alert)として抽出する：標準で Forward のペアが Behind になる，あるいはその逆を示すモジュール

③ 関係の変化数を算出する：標準で Forward/Behind のペアが Middle になる，あるいはその逆を示すペアの数を算出する

④　閾値以上の変化数のモジュールを異常モジュール(Low　Alert)として抽出する：上記の変化数がある閾値 N 以上のモジュール

この手順は，メトリクス毎に行い，1 つでも上記の②④の条件にあてはまるモジュールは，異常モジュールとして抽出する．

図 5 で，モジュールαのパターンは，標準パターンに対して，Phase3 と Phase 4 の関係が逆転しているので，High Alert の異常モジュールとなる．一方，曖昧な関係の変化が 6 箇所で見られる（組み合わせでは 3 つ）．Phase3 と Phase 4 の関係に逆転がなくても，閾値 N=6 ならば，このモジュールは Low Alert の異常モジュールとして抽出される．閾値 N は，Phase 数に依存する．



	Phase1	Phase2	Phase3	Phase4
Phase1		-0.47059	-0.40351	-0.2963
Phase2	0.470588		0.103623	0.840909
Phase3	0.403509	-0.10362		0.602613
Phase4	0.296296	-0.84091	-0.60261	

	Phase1	Phase2	Phase3	Phase4
Phase1		Middle	Middle	Middle
Phase2	Middle		Middle	Forward
Phase3	Middle	Middle		Forward
Phase4	Middle	Behind	Behind	

	Phase1	Phase2	Phase3	Phase4
Phase1		Behind	Middle	Middle
Phase2	Forward		Forward	Middle
Phase3	Middle	Behind		Behind
Phase4	Middle	Middle	Forward	

図 5　異常モジュールの抽出手順

4．事例研究

4．1　対象プロジェクト

本研究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，2006 年度はその二年目にあたる．

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)[4]，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）[5]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．各社で分担して作成した基本設計書については，全社の担当者によるレビューが行われたが，以降社内結合試験までは週次の進捗報告(進捗率)と各工程完了時の品質評価報告のみがプロジェクト管理担当企業に提示される．このような限定されたプロジェクト管理方法となった要因の１つには，開発対象が協調（公開）領域と競争（非公開）領域が混在したものになっており，社間のみならずプロジェクト管理担当会社に対してもプロダクトや社内情報に関する機密保持が求められたからである．

同プロジェクトで EPM[3]や EPM-ProStar[5]を用いた分析手法および適用フレームワーク[6]は，対象プロジェクトのような研究組合型マルチベンダ開発の特徴である限定された情報によるプロジェクト管理に対して，有効な支援を行う方法として導入された．

4．2　データ収集

シーケンスデータを構成管理システムから作成するために，EASE プロジェクトで開発した EPM[1]及び EPM-ProStar[5]を利用する．図 6 は，シーケンスデータを作成する手順を示している．EPM のトランスレータ機能は構成管理ツールから構成管理履歴データと各バージョンのファイルのサイズを計測して時系列のデータとして XML 形式のファイルで出力する．EPM-ProStar は XML 形式のファイルを入力とし，データ作成機能は時系列の上記のメトリクスを含む様々なデータを加工し，CSV ファイルで出力する．グラフ作成機能は，各メトリクスやそれぞれの一定期間ごとの差分データを作成してグラフとして出力するほか，そのデータを Excel や CSV 形式で取り出すことができる．これらのツールは，分析単位(モジュール)や期間，差分データの区間を指定してデータを加工する機能を持ち，本研究に必要なデータのほとんどは，この EPM-ProStar を用いて作成した．LOC の変更率のみ，LOC 遷移データから Excel のマクロ機能を使って作成した．今回の分析のデータ

---

[4] 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

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

は，2006 年度実際に 4．1 節でプロジェクト管理のために週次で作成し，プロジェクト管理者や各社開発メンバにフィードバックしたデータである．2005 年度のデータは，本研究のために，過去の構成管理リポジトリから EPM, EPM-ProStar を用いて後から作成した．

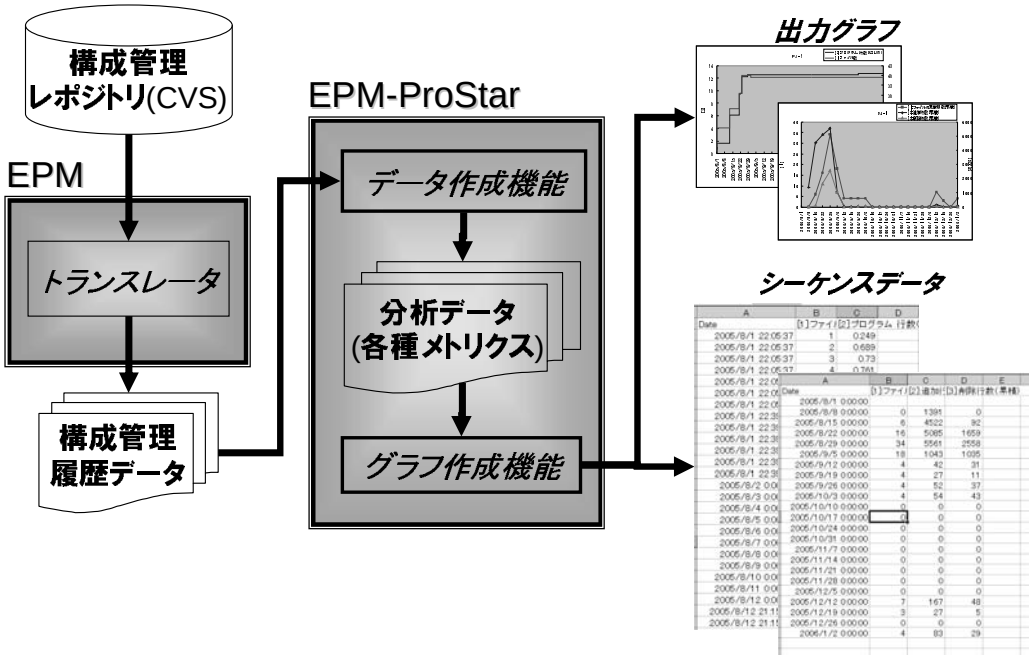


図 6　シーケンスデータ作成手順

収集したデータは，2005 年度のプロジェクトの第 1 フェーズで 54 モジュール，2006 年度の第 2 フェーズで 36 モジュールである．2005 年度のこのプロジェクトとしては新規の開発となるが，各社が自社内の資産である技術・システムを流用していることもあり，完全なスクラッチからの開発ではないモジュールも混在する．しかし，この点については，プロジェクト全体では把握されていない．各社内の開発の形態（新規・流用・改造）は，社間秘とされている．2006 年度は，2005 年度開発のシステムに対して，新規の機能を追加したり，各社で共通の機能を共通モジュール化したりしている．そのため，多くのモジュールは改造レベルで，新規開発モジュールでも，既存のモジュールをコピーし改造したものが多い．

4．3　データ前処理と前提条件

分析対象となる構成管理ツールの履歴データは，開発者や管理者の運用方法に大きく依存するため，今回のプロジェクトでは各社のデータ粒度や分析のタイミングなどについて，各社との議論を経て運用ルールを定め，開発現場への導入を依頼した．すでに社内の規定の運用ルールを持っている組織や，従来開発者にその運用方法を任せている組織など様々なケースがあったが，極力各社で統一した運用が行われるように協力をいただいた．

- 構成管理システムへのファイルの登録は，単体試験開始前とし，コーディング中から運用することも可能.
- 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受容する．その場合でも，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．これは，すべての更新理由について把握し，分析・評価に用いることを目的としている．
- CVS でブランチ機能や Export・Import 機能などは，履歴に明確に表現されないため極力用いない.

シーケンスデータの開始日と終了日は，各モジュールで異なる．組織によって開発の開始時期は異なり，さらに，一人の開発者が複数モジュールを担当するため，コーディングの開始時期も単体試験の期間もずれている．ただし，社間結合試験以降は，全社同一スケジュールで実施されたため，社間結合試験～総合試験の開始・終了時期は一致する．本研究では，モジュールの開発シーケンスの開始時期を対象ファイルの最初の登録日とし，モジュール毎に決定した．一方，終了日は総合試験完了日であり，構成管理への更新の有無に関らず，2005 年 10 月 31 日，2006 年 10 月 31 日とした.

4．4　フェーズ分類のための閾値の決定

本研究では，各メトリクスを以下の基準によって 5 つのフェーズに分類した.

- 0 : 作業なし（更新が無い）
- 0 : 作業あり（更新されているが，メトリクスに変化が無い）
- 0 以外の値 : 値の大小による 3 分割

0 の場合の作業有無の分類方法は, 各メトリクスによって異なり, 以下の基準で分類した.

- 規模変更率(LChg) : 構成管理上更新・登録等のイベントが履歴にあるが，行数に変化がない場合，作業がある 0 値とする．履歴上のイベントがなく，行数に変化もない場合，作業無しの 0 値とする
- 変更回数(Freq), 追加行数(Ladd), 削除行数(Ldel) : それぞれ対象のメトリクスが 0 で, 他の 2 つが 0 以外の場合, 作業ありの 0 値, 全部のメトリクスが 0 の場合作業無しの 0 値とする.
- 規模変更率(LChg)のみ, 最初の登録時に分母が 0 になるため, そのケースは除外する. (登録日のデータは，上記のどの分類にも計上されない)

0 以外の値を 3 分割する方法としては，以下の 4 つの方法を用いて分析し，その結果の妥当性を考察した．

- 等数分割：データの数が等数になるように分割する値を閾値とする
- 4 分位：データを値順に整列し，データ数が 25%以下，25%以上 75%以下，75%以上になる値を閾値とする
- 等値：データの最大値，最小値の差分を 3 分割した値を閾値とする
- LChg のみ 0 未満，0 より大を等数(2 つ)分割とするケースを追加し，等値分割はデータ数が著しく偏るため，用いない．

表 1 に各分類方法を用いた場合の Gamma Analysis の結果を示す．

表 1　各分類方法による Gamma Score の平均値

	2005				2006			
	LChg	Freq	Ladd	Ldel	LChg	Freq	Ladd	Ldel
等数分割	0.416062	0.634267	0.635471	0.522314	0.364865	0.679936	0.776837	0.554568
4 分位 25%/75%	0.413924	0.608262	0.641492	0.589907	0.399812	0.700898	0.829105	0.480882
等値分割		0.572439	0.728298	0.645843		0.781308	0.786877	0.743139
0 未満＋等数分割	0.409812				0.332398			

それぞれの Gamma Score は全モジュールのすべてのペア（２０ペア＝5 phase×４）の Gamma Score の絶対値の平均である．値が大きい場合，明確なシーケンスが検出できたことを示す．全体的に等値分割のスコアが高めだが，等値分割では，各分類のデータ数が偏り， 1，-1 が多い（各分類のデータが１つずつだと，スコアが１か-1 になってしまう）ためと思われる．また，分類に当てはまるデータがないため，計算できないペアも多かったため，これは用いないこととする．その他を比較の上，比較的シーケンスの検出が明確であった 4 分位を使う事とする．

4 分位に四つ分割を行った場合の各メトリクスの閾値は以下のとおりとなる．

表 2　各メトリクスの Phase 分割のための閾値

		規模変更率 (LChg)	変更回数 (Freq)	追加行数 (Ladd)	削除行数 (Ldel)
2005 年度	閾値 1	0.16	1	34	1
	閾値 2	4.99	8	942	104
2006 年度	閾値 1	0.05	1	34	4
	閾値 2	1.85	11	1054	163

4．5　Gamma Score 計測結果

表 3～表 10 に，メトリクスごとに計測した Gamma Score を示す．モジュール毎に計測後，その平均を示している．各分類名の意味は以下のとおりである．

- Phase1：作業なし 0 値
- Phase2：作業あり 0 値
- Phase3：閾値 1 以下
- Phase4：閾値 1 より大で，閾値 2 以下
- Phase5：閾値 2 より大

表 3　2005 年度データの規模変更率の Gamma Score

LChg(2005)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	…	0.416	0.477	0.578	0.647
Phase2	-0.416	…	0.188	0.114	0.654
Phase3	-0.477	-0.188	…	0.055	0.567
Phase4	-0.578	-0.114	-0.055	…	0.444
Phase5	-0.647	-0.654	-0.567	-0.444	…

表 4　2006 年度データの規模変更率の Gamma Score

LChg(2006)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	…	0.530	0.472	0.474	0.674
Phase2	-0.530	…	-0.037	-0.130	0.288
Phase3	-0.472	0.037	…	0.125	0.644
Phase4	-0.474	0.130	-0.125	…	0.624
Phase5	-0.674	-0.288	-0.644	-0.624	…

表 5　2005 年度データの変更回数の Gamma Score

Freq(2005)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	---	0.804	0.591	0.627	0.559
Phase2	-0.804	---	-0.667	-0.724	-0.636
Phase3	-0.591	0.667	---	0.351	0.767
Phase4	-0.627	0.724	-0.351	---	0.356
Phase5	-0.559	0.636	-0.767	-0.356	---

表 6　2006 年度データの変更回数の Gamma Score

Freq(2006)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	---	0.848	0.518	0.697	0.787
Phase2	-0.848	---	-0.889	-0.875	-0.933
Phase3	-0.518	0.889	---	0.402	0.600
Phase4	-0.697	0.875	-0.402	---	0.459
Phase5	-0.787	0.933	-0.600	-0.459	---

表 7　2005 年度データの追加行数の Gamma Score

Ladd(2005)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	---	0.125	0.617	0.650	0.726
Phase2	-0.125	---	#DIV/0!	#DIV/0!	1.000
Phase3	-0.617	#DIV/0!	---	0.613	0.765
Phase4	-0.650	#DIV/0!	-0.613	---	0.636
Phase5	-0.726	-1.000	-0.765	-0.636	---

表 8　2006 年度データの追加行数の Gamma Score

Ladd(2006)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	---	0.875	0.584	0.760	0.843
Phase2	-0.875	---	-1.000	-1.000	1.000
Phase3	-0.584	1.000	---	0.557	0.961
Phase4	-0.760	1.000	-0.557	---	0.712
Phase5	-0.843	-1.000	-0.961	-0.712	---

表 9　2005 年度データの削除行数の Gamma Score

Ldel(2005)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	…	0.797	0.435	0.628	0.580
Phase2	-0.797	…	-1.000	-0.592	-0.351
Phase3	-0.435	1.000	…	0.336	0.722
Phase4	-0.628	0.592	-0.336	…	0.457
Phase5	-0.580	0.351	-0.722	-0.457	…

表 10　2006 年度データの削除行数の Gamma Score

Ldel(2006)	Phase1	Phase2	Phase3	Phase4	Phase5
Phase1	…	0.752	0.608	0.696	0.772
Phase2	-0.752	…	-0.471	-0.404	-0.296
Phase3	-0.608	0.471	…	0.104	0.841
Phase4	-0.696	0.404	-0.104	…	0.603
Phase5	-0.772	0.296	-0.841	-0.603	…

この Gamma Score から標準的なパターンを取得する．解釈例として，表 9 表 10 の削除行数について説明する．

- Phase1 は全体的に－1 に近い値をとる→Phase1 はフェーズの後半に多い⇒作業は，フェーズの後半ほとんど発生しない，また，開発中に作業無しの期間は少ない
- Phase2 は特に 2005 年度のデータで 1 に近い値をとる→Phase2 はフェーズの前半に多い⇒削除を伴わない作業（追加作業）は，フェーズの前半に発生する
- Phase5 は Phase2 以外には全体的に 1 に近い値をとる→Phase5 はフェーズの前半に多い⇒大規模な削除は，フェーズの前半に発生する
- Phase3，Phase 4 はやや曖昧だが，2005 年度では Phase4 の後に Phase3 が発生するという順序関係が見られる⇒開発進行に伴い，削除行数は減少していく傾向にある

これらの結果から，標準的なパターンを示すモジュールを以下に図示する．標準的なパターンとは，3．6 節の①の Gamma Score 解釈に従って，Behind，Middle，Forward に分類したときに，すべての分類が一致するモジュールである．

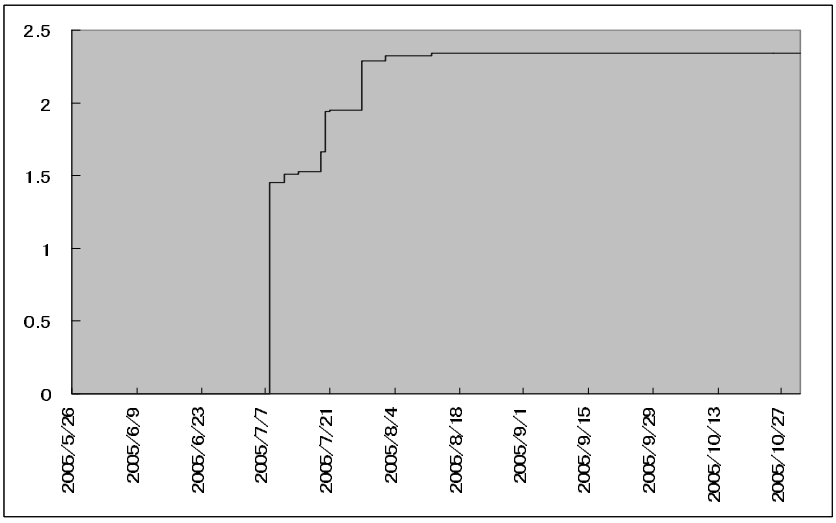


図 7　2005 年度規模遷移(SLOC)標準パターン

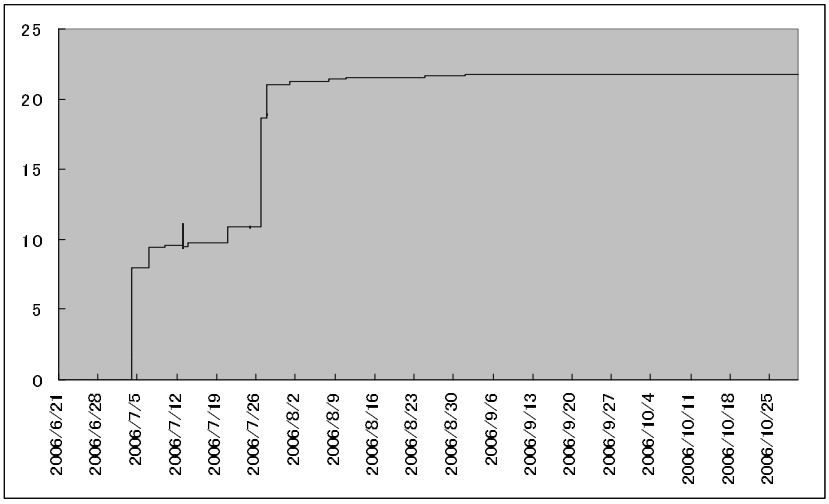


図 8　2006 年度規模遷移(SLOC)標準パターン

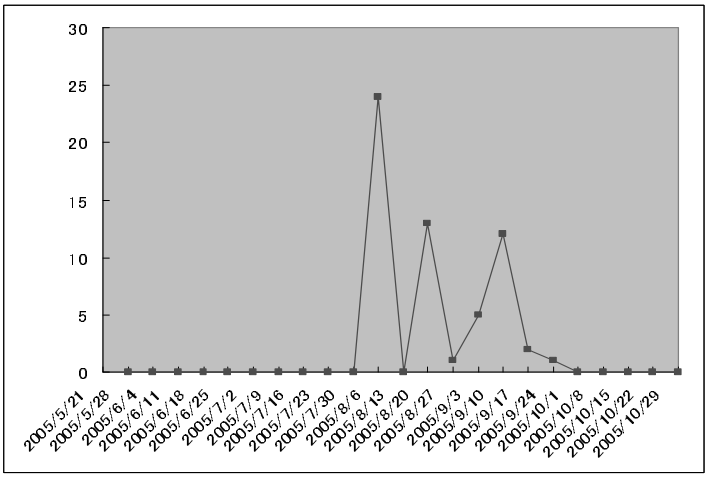


図 9　2005 年度変更回数標準パターン

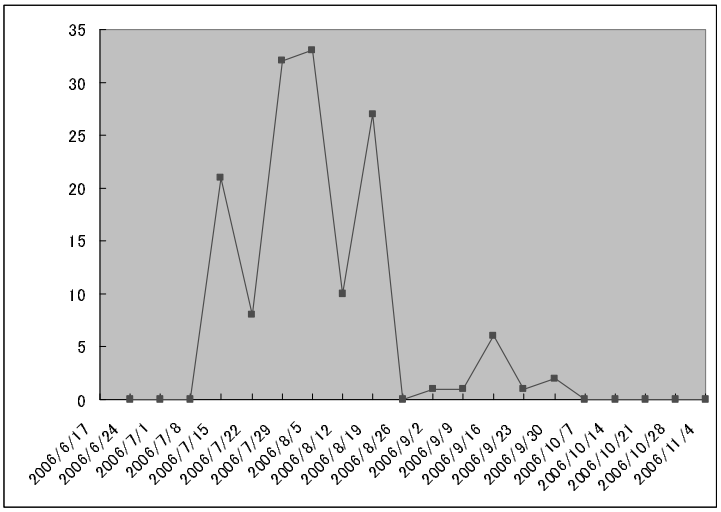


図 10　2006 年度変更回数標準パターン

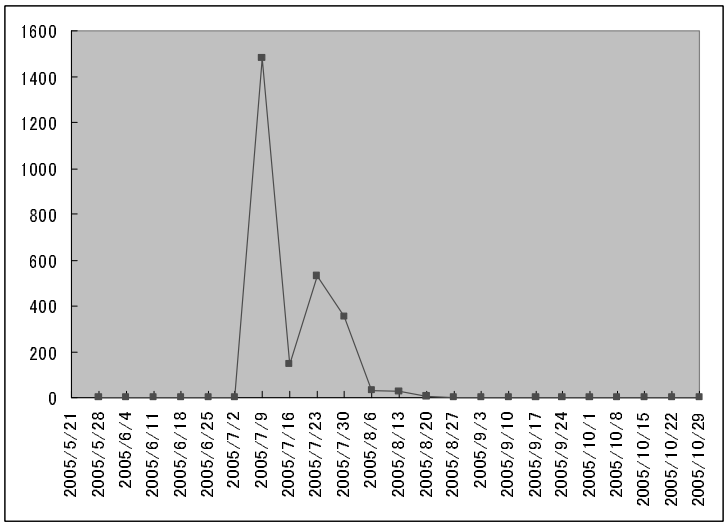


図 11　2005 年度追加行数標準パターン

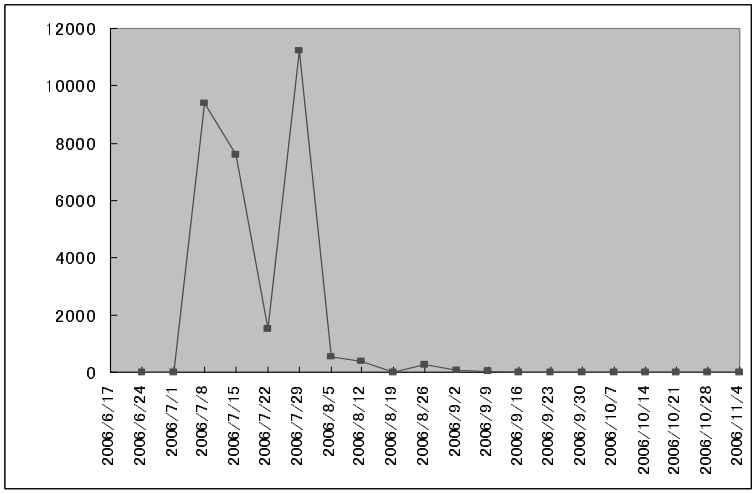


図 12　2006 年度追加行数標準パターン

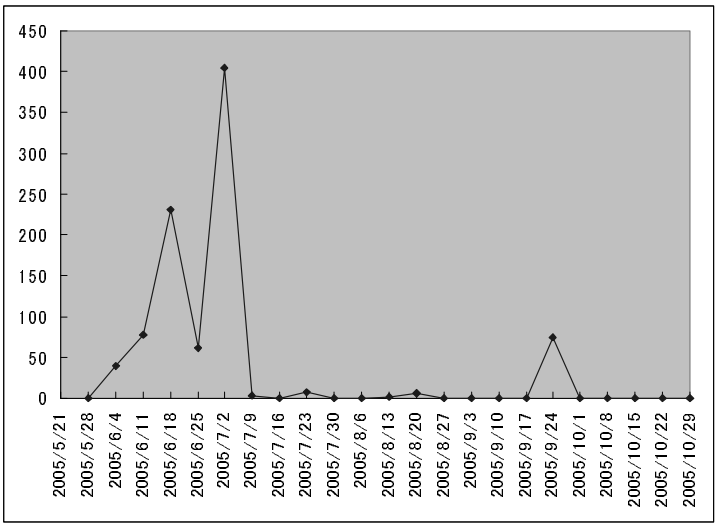


図 13　2005 年度削除行数標準パターン

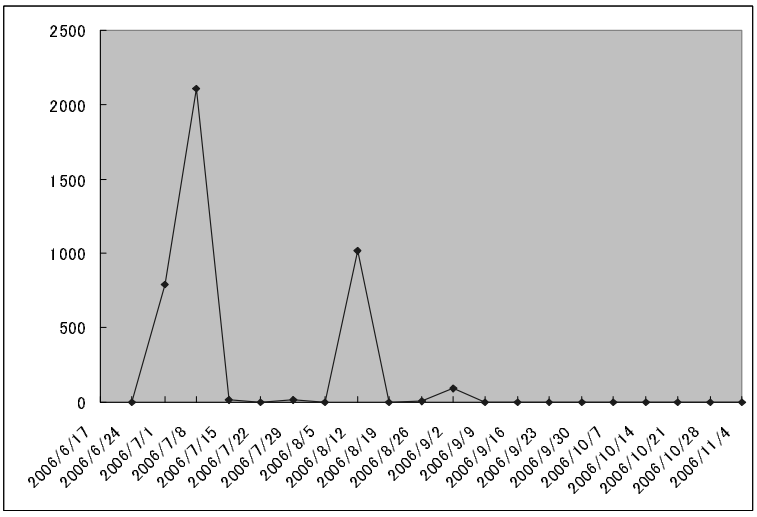


図 14　2006 年度削除行数標準パターン

5．評価

5．1　評価手順

本手法の有効性を検証するために，我々は 2006 年度のデータを用いて，本手法で抽出された異常モジュールと実際にプロジェクト進行中にエキスパートによって指摘された異常モジュールと比較し，適合率と再現率を求める．適合率からは，本手法で検出される異常のうちどのくらいが実際に実プロジェクト進行中で指摘されていたかを示し，本手法による誤検出の割合がわかる．一方，再現率からは，エキスパートによって検出された異常モジュールのうちいくつが本手法で検出されたかを示し，本手法の見逃しの割合がわかる．

以降，2 つの異常モジュール検出方法の詳細について説明する．

[本手法による異常モジュール]　3．6 節の方法で検出するが，　Low Alert を抽出する相

違ペア数の閾値は，8 とする．すなわち，8 以上のペア(A→B，B→A を別個にカウントするので，実質は 4 つ以上のペア)が標準パターンと異なる場合，異常とみなす．

[エキスパート指摘による異常モジュール] エキスパートによる異常モジュールの検出過程は，[6]に詳細に報告されているが，ここでは概要について説明する．

対象プロジェクトでは，構成管理ツールや障害管理ツールを用いたインプロセス分析によるプロジェクト管理支援手法の適用を実施し，本研究で用いたメトリクスも用いてプロジェクト進行中に納期遅延や品質低下につながる問題の検出を試みた．具体的なプロセスを図 15 に示す．週次で各社から提供されたデータ(構成管理リポジトリや障害管理リポジトリ)に対して EPM[1]・EPM-ProStar[5]などのツールを用いて分析を行ったうえで，可視化したデータを各社にフィードバックし，同時にそこから検出される異常と思われる点について指摘し，状況について質問をした．その回答などから，実際に異常が発生していたモジュールを抽出した．このサイクルは，コーディング工程開始時期に始められ，社間総合試験完了までほぼ毎週行われた．この間に作成されたフィードバック資料やそれに基づく指摘・質問，および各社からの回答が，評価のための情報として記録された．さらに，プロジェクト完了時には，各指摘に対する社内での認識状況や対応についてインタビューを行った．本稿の研究は，EPM・EPM-ProStar などのツールによるフィードバックデータを用いて，開発完了後に開始された．

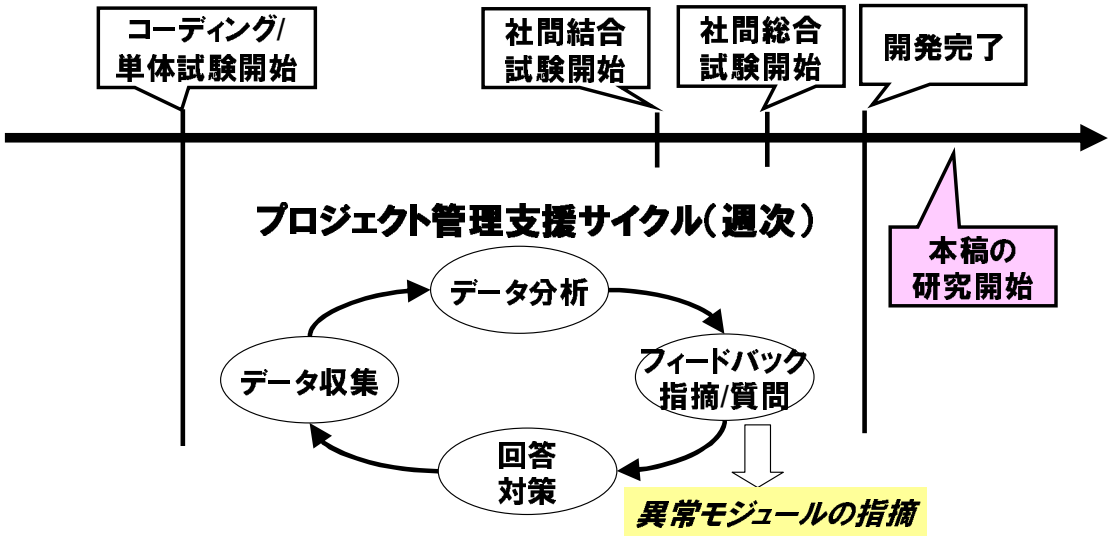


図 15　プロジェクト進行中の異常モジュール検出

本稿では，図 15 におけるプロジェクト進行中の異常モジュール指摘を，「エキスパート指摘」と呼ぶ．エキスパート指摘には，ツールによる分析データも用いるが，その他にプロジェクトに関する様々な明示された情報及び暗黙の情報を用いる．例えば，週次で各社から提出される進捗情報（モジュール毎の進捗率，テスト項目消化数など），工程情報（コーディング工程の後半，結合試験中など），CVS のコメント（変更理由など），バグ情報（対

応中のバグ，残バグ数など)，その他開発会社の体制(下請け体制)，各社の既存の管理手法，システム構造や改造・流用・複雑さなどの情報に基づいて，実際のプロジェクト経験などから異常を感じるポイントについて指摘が行われた．各メトリクスの視覚化された情報(グラフなど)は，上記の様々な情報と関連付けて用いられた．

以下に，同プロジェクトでプロジェクト進行中に開発者側へ指摘した内容例を示す．下線部は，指摘する際に質問文には含まれていないが，実際に指摘者が質問の前提として用いている情報である．

- ［質問］7/15～7/22 の週で，更新回数，変更規模が落ち込んでいますが，この週の作業に何か問題（出戻り等)が発生していなかったでしょうか？(コーディング工程の初期における指摘)
  →［回答］7/8 の週までに初期投入した流用元ソースの中で，間違って投入したものがあり，7/8～7/15の間にそのソースを削除して，正しいソースを投入しなおしたため，変更規模が大きくなっております．7/15～7/22 は通常の開発です．7/22～7/29 の週は，未投入だった流用元ソースをすべて投入したため，変更規模が大きくなっております．

- ［質問］xxx コンポーネントで，8/9 あたりに大規模な変更が行われています．GNATS の 20，22，23 あたりですが，GNATS 上の工数は特に大きくありません．変更量に対して，修正工数が低い理由が何か考えられますでしょうか？(単体試験工程で，バグの修正工数や修正バグ数などと対応させて指摘)
  →［回答］上記修正番号で修正が大きかったのは 23 です．C++の string クラスに関して使い方が間違っていたことが分かり使用している部分を全て見直しため，変更量が大きくなりました．修正工数が少ないのは，関連見直しのため複雑な修正ではなかったためです．

5．2　評価結果

表 11 に異常モジュールの検出結果を示す．モジュール名の“Module_”の後の頭の 1 文字が組織を示している．変更回数は，各モジュールで構成管理履歴上の更新回数を示す．ここには，ファイル新規登録を含むがファイルの削除，Import（CVS の機能）は含まれない．異常メトリクスは，3．6 節で説明した方法で異常が検出されたメトリクスに◎（High Alert）○(Low Alert)が記入されている．High Alert が検出された場合，Low Alert は無視する．エキスパート指摘は 5．1 節で述べたプロジェクトの進行中に各社に対して異常および異常の疑いを指摘したもので，一回でも指摘を受けたモジュールは●が記入されている．評価では変更回数が１０回以下のモジュールは，網掛けされたモジュールで，評価対象としない．これらは流用モジュールやテスト用スクリプトなどで，開発の重要な要素ではない，と判断したからである．

Module_A1 は，エキスパートによって指摘されたが，本手法では検出できなかったモジュールである．また，Module_B8 や Module_F3 では，High Alert のメトリクスが検出されたが，実際のプロジェクト中には異常を指摘されなかった．

異常を示した 2 つのモジュールのグラフを図 16 と図 17 に示し，CVS 上の情報と，プロジェクト進行中およびプロジェクト完了時に各社に対して行ったインタビューにより，判明した詳細な状況を説明する．

表 11　本手法とエキスパートによる異常指摘の一覧

モジュール名	変更回数	異常メトリクス				エキスパート指摘
		規模変更率	変更回数	追加行数	削除行数	
Module_A1	108					●
Module_B1	77	○				●
Module_B2	5	○				
Module_B3	21		○		○	●
Module_B4	57	○		◎		●
Module_B5	112	○			○	●
Module_B6	43	○				
Module_B7	7				○	●
Module_B8	28	○	◎	◎		
Module_B9	18				○	
Module_C1	6					●
Module_C2	15		◎		◎	●
Module_D1	290				○	●
Module_E1	13				○	●
Module_E2	30	○			○	
Module_E3	27	○			○	
Module_E4	74	○			○	
Module_F1	5				○	
Module_F2	23	○				
Module_F3	145	○		◎		
Module_F4	114				◎	●

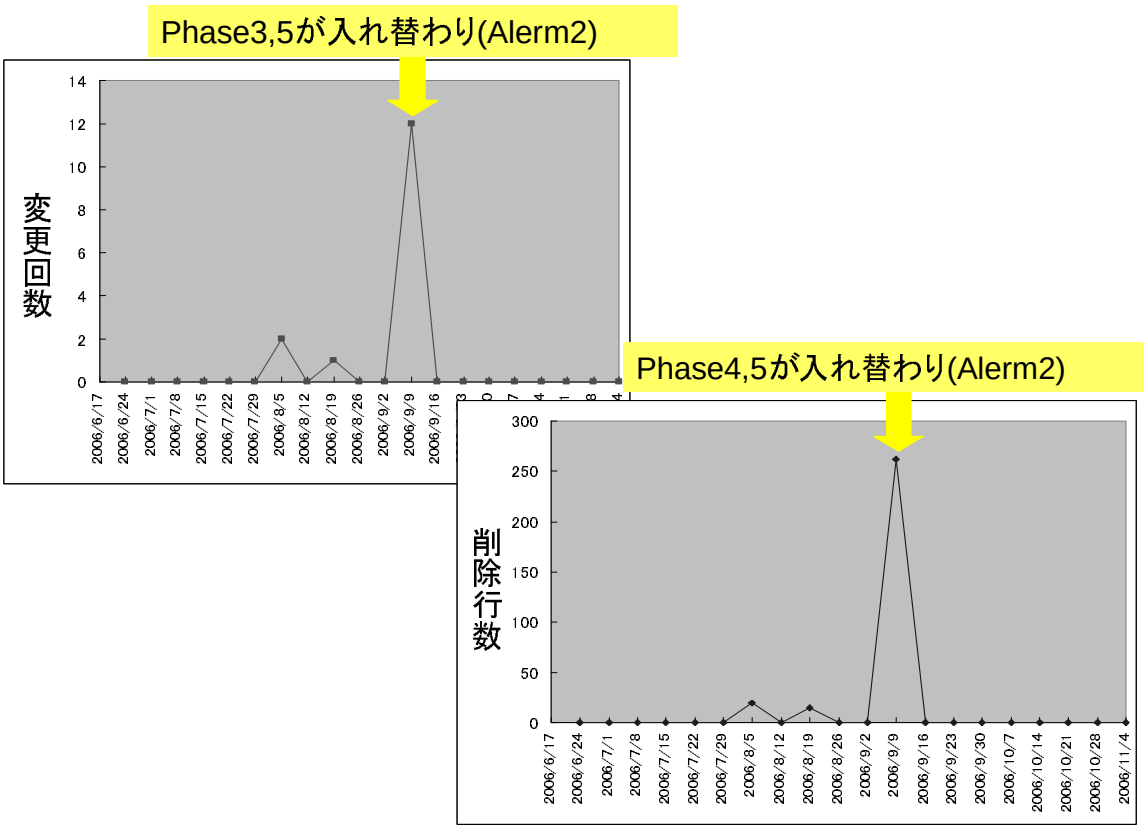


図 16　Module_C2 の変更回数と削除行数の異常

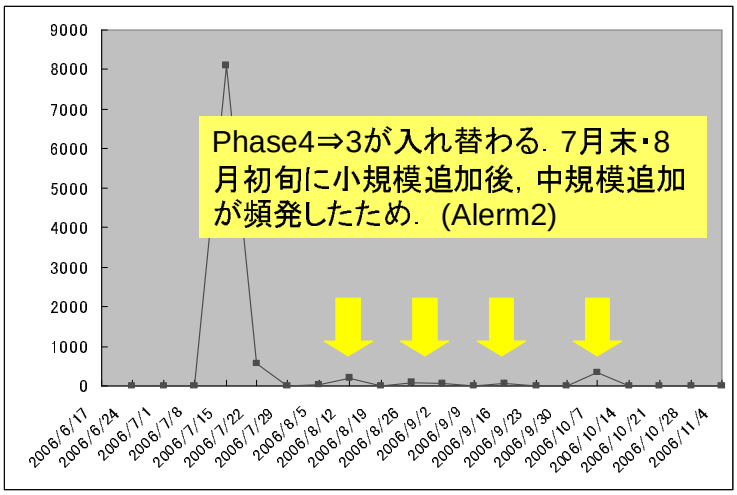


図 17　Module_B4 の追加行数の異常

Module_C2 の場合，社間結合試験の開始時期にヘッダーファイルが構成管理システム上に大量に追加されたことと，「コーディングフェーズの変更」（CVS のコメント）が一括でこの時期に更新されていることが原因で変更回数，削除行数共に大きくなっている．これらについては，担当開発リーダに調査を依頼したが，回答はなく理由は不明である．開発

者が，単体試験完了時に登録漏れのファイルを一括登録したり更新忘れの修正ファイルを一括更新したと考えられる．

Module_B4 では，社間結合で他社の共通モジュールの機能について認識違いが発覚し，設計変更を行い，大幅な修正が発生した．そのため，中規模(34 行より大で，1054 行以下)程度の追加行数が開発の最終段階（10 月上旬）に発生した．

以下，表 11 から得られた結果をまとめる．

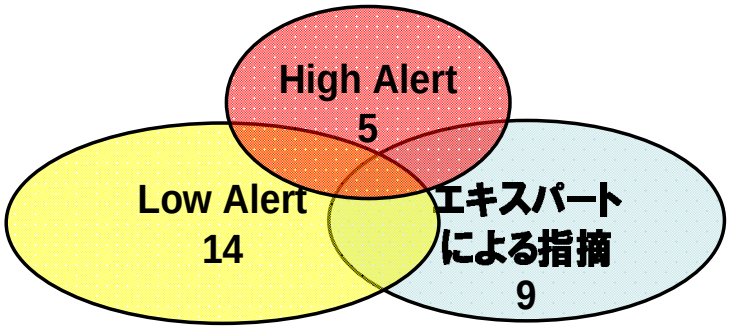


図 18　検出された異常モジュール数

プロジェクト進行中に，エキスパートによって指摘された異なるモジュール数は全部で 9 個あった．それに対して，本手法で検出された異常モジュールは，High Alert のモジュールが 5 個，Low Alert のモジュール数が 14 個だった．High/Low の両方で検出されたモジュールは，表 11 のとおり，3 個あった．

エキスパートによる指摘モジュールに対する同手法の再現率と適合率を表 12 に示す．High Alert のみを用いた場合，エキスパートが指摘した異常モジュールのうち，3 個を検出できた．High/Low の両方の Alert を用いると，8 個，約 88.8%を検出できた．一方，High Alert で検出された 5 個のうち 3 個はエキスパートでも指摘していたモジュールで，High/Low の両方の Alert で指摘された 16 モジュールのうち 8 個はエキスパートでも指摘していた．

表 12　エキスパートによる指摘に対する再現率と適合率

	High Alert のみ	Low Alert + High Alert
再現率	3/9＝33.3%	8/9＝88.8%
適合率	3/5＝60.0%	8/16＝50.0%

6．考察

5 章の結果から，得られた知見を以下にまとめる．

- 【High Alert の再現率】High Alert を示したメトリクスは，目でも確認できる大きなグラフ上の異常が検知できている．しかし，エキスパートによって指摘されたモジュ

ールの 33.3%であることから，メトリクスのグラフだけから異常の検出では，見落としが多くなる，と思われる．エキスパートによる指摘でも，グラフを用いているが，それ以上に有効な情報をエキスパートは用いている，と考えられる．

- 【High Alert の適合率】High Alert による検出モジュールのうち，60%がエキスパートでも指摘していることから，本手法で用いたメトリクスのグラフで見られる異常は，半分以上実際になんらかの異常の発生を示している，と思われる．従って，High Alert を示すモジュールは，注意が必要である．
- 【High Alert＋Low Alert の再現率】Low Alert の場合，グラフの見た目だけで大きな変化が見られないため，見落とされる可能性が高いが，本手法によって異常を検出したモジュールの大半がエキスパートにより指摘されていたモジュールである．したがって，より微細なシーケンス分析により，エキスパートに近い異常モジュールの検出が可能になる，と考えられる．標準シーケンスに対する大きな相違だけでなく，微妙な相違も異常モジュールの検出に貢献する，ということがわかる．
- 【High Alert＋Low Alert の適合率】High Alert だけの場合より，若干適合率が下がることから，検出されたモジュールが異常でないケースも増加する．しかし，High Alert のみの場合と大きな相違ではないことから，再現率を考慮すると Low Alert を用いた異常検出も用いるべきと考えられる．

実際に検出されるべき異常モジュールが，エキスパートによってすべて検出されていたかどうかは，確認されていない．また，エキスパートでも，指摘のすべてが本来検出すべき異常を指し示していたわけでない．そのため，有効性の評価には，別の観点からの分析も必要と考えている．

各メトリクスについて，個別に Gamma Analysis の結果や異常検出への貢献度から，以下のようなことがわかった．

- 削除行数がもっとも異常の検出に貢献している．対象プロジェクトへのメトリクス導入検討時に，プロジェクトマネージメントの長年の経験者から，削除行数に最も着目している，という意見があった．本稿の結果は，この意見を裏付けている．
- 削除行数は，Gamma Score から 2005 年度と 2006 年度で相違が大きい．これは，新規開発と改造開発による相違と思われる．削除行数を用いた分析においては，開発形態（新規，改造，再利用）が大きく影響する，と考えられる．
- 規模変更率は，明確な異常の検出，すなわち High Alert の検出にあまり貢献しない．Gamma Score も全体的に非常に低く，明確なパターンが抽出できなかった．モジュールの規模で正規化することで，実際の作業規模が曖昧になっていると思われる．一定期間での作業量を見る場合には，異常の検出は対象モジュールのサイズには依存しない，と思われる．
- 変更回数や追加行数は，High Alert を示すモジュールが多い．この 2 つのメトリクス

は，標準のパターンが非常に明確であり，特殊な作業が混入することによって，顕著な相違を検出することができる，と考えられる．しかし，必ずしも，それが異常を示すものではない．

今回の対象プロジェクトは，5 社の共同開発で，データにもばらつきが多いと予想されたが，各メトリクスについて，値の変動傾向にある一定のパターンが抽出できることがわかった．削除行数以外は，新規開発でも改造開発でもあまり大きな相違がない．このことから，使用したメトリクスは汎用的に用いることができる指標である，と言える．

ただし，ある程度の構成管理ツールの運用ルールを規定した上での開発だったので，プロセスが個人に依存するアジャイルプロセスやオープンソースの開発などでは，違うパターンを示す，もしくは一定のパターンが取り出せないことが予想される．

また，本分析では，5 つの Phase に分類したが，これが妥当であるかはわからない．メトリクスの特徴や開発の特徴 (規模, 期間, プロセスなど)，あるいは計測の単位期間(日次，週次，月次など)によって Phase の分類方法も調整が必要である可能性がある．各 Phase のケース数が少ないと，Gamma Score の絶対値は 1 に近くなるので，ケース数などに応じた調整が必要と思われる．

7．まとめ

本稿では，構成管理情報から抽出できる定量的な時系列データの分析によって，異常な作業が発生しているモジュールを検出する方法を提案し，実際のプロジェクトに適用してエキスパートによる他の情報や経験に基づく異常検出に近い検出結果を得ることができた．

今後は，異なるプロセスや分野のプロジェクトデータでの有効性を評価するため，他のプロジェクトデータでの検証を行う予定である．また，今回は対象プロジェクトの特徴に応じて決定した Phase 分割の閾値や Low Alert 検出の閾値，データ集計期間などの決定方法を明確にした上で，EPM-ProStar に実装することにより，一般ユーザ(非技能者)でも活用できる環境を構築することが必要と考えている．

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」および「次世代 IT 基盤構築のための研究開発」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] Katsuro Inoue, “Software Tag: Empirical Software Engineering Data for Traceability and Transparency of Software Project,” Workshop on Accountability and Traceability in Global Software Engineering(ATGSE2007) Proceedings, pp.35-36, Nagoya, Japan, 2007.

[2] Ken-ichi Matsumoto, “A Purchaser-Centered Approach for Empirical Software Engineering,” Workshop on Accountability and Traceability in Global Software Engineering(ATGSE2007) Proceedings, pp.33-34, Nagoya, Japan, 2007.

[3] 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.

[4] 井上克郎, 松本健一, 飯田元著, 「ソフトウェアプロセス」pp.112-117 共立出版, 2000.

[5] 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, “ 奈良先端科学技術大学院大学テクニカルレポート, NAIST-IS-TR2007002, February 2007.

[6] 松村 知子, 勝又敏次, 森崎 修司, 玉田 春昭, 大杉 直樹, 門田 暁人, 楠本 真二, 松本 健一, 自動データ収集・可視化ツールを用いたリアルタイムフィードバックシステムの構築と試行, 奈良先端科学技術大学院大学テクニカルレポート, NAIST-IS-2007001, February 2007.

[7] 松村 知子, 門田 暁人, “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,” 奈良先端科学技術大学院大学テクニカルレポート, NAIST-IS-2008002, February 2008.

[8] 松村 知子, 森崎 修司, 門田 暁人, 楠本 真二, 飯田 元, 松本 健一, 久保 浩三, 井上 克郎, “ユーザ視点に立ったソフトウェア開発データ活用技術の提案,” 情報処理学会ソフトウェア工学研究会研究報告, March, 2008.

[9] D.C.Pelz “Innovation Complexity and the Sequence of Innovating Stages”, Science Communication, vol.6, pp261-291, 1985.

[10] C.F.Kemerer, S.Slaughter, “An Empirical Approach to Studying Software Evolution”, Trans. On Software Engineering, Vol.25, No.4, pp.493-509, 1999.

[11] Z.Xing, E.Stroulia, “Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software”, Trans. On Software Engineering, Vol.31, No.10, pp.850-868, 2005.