情報処理学会論文誌 Vol.56 No.1 59–71 (Jan. 2015)

推薦論文

# OSSシステムとコミュニティの共進化の理解を目的としたデータマイニング手法

山谷 陽亮[1] 大平 雅雄[1,a)] パサコーン パンナチッタ[2] 伊原 彰紀[2]

受付日 2014年3月13日, 採録日 2014年10月8日

**概要**：オープンソースソフトウェア（OSS）を活用したシステム開発が一般的になりつつある一方，「サポートが得られるかどうか分からない」などの理由から，依然としてOSSの活用に躊躇するシステム開発企業は少なくない．本研究では，OSSシステムとコミュニティの共進化のプロセスを定量的に分析するためのデータマイニング手法を提案する．本手法は，時間的順序関係を考慮した相関分析を行うためのものであり，一方の系統の進化が一定時間後に他方の系統の進化に影響を与えるという関係の抽出を支援する．Eclipse Platformプロジェクトを対象に131種類のメトリクスを用いてケーススタディを行った結果，従来の相関分析では抽出できない31件の関係を提案手法により抽出できた．また，抽出された関係を追加分析することで，Eclipse Platformにおける共進化のプロセスをより正確に観察できることを確かめた．

**キーワード**：遅延相関分析，リポジトリマイニング，オープンソースソフトウェア開発，システムとコミュニティの共進化，ソフトウェアメトリクス

# A Data Mining Method for Understanding Co-evolution of OSS Systems and Communities

Yosuke Yamatani[1] Masao Ohira[1,a)] Passakorn Phannachitta[2] Akinori Ihara[2]

Received: March 13, 2014, Accepted: October 8, 2014

***Abstract:*** While using open source software (OSS) has been becoming common as a means for developing software systems, many companies still have a reluctance to utilize OSS products. This paper proposes a data mining method to analyze the co-evolution process of OSS systems and communities quantitatively. The method includes time-delayed correlation analysis to investigate whether changes in the amount of an explanatory variable have an impact on changes in the amount of an objective variable after a certain period of time. Through a case study of the Eclipse Platform project using 131 software metrics and our method, we found that our method extracted 31 correlations which cannot be extracted by the common correlation analysis. We also found that our method can support additional analyses to observe the co-evolution process in the Eclipse Platform project more precisely.

***Keywords:*** time-delayed correlation analysis, repository mining, open source software development, co-evolution of OSS systems and communities, software metrics

## 1. はじめに

近年，システム開発の低価格化・短納期化に対応するために，Linuxをはじめとするオープンソースソフトウェア

[1] 和歌山大学システム工学部情報通信システム学科
Department of Computer and Communication Sciences, Faculty of Systems Engineering, Wakayama University, Wakayama 640–8510, Japan
[2] 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, Ikoma, Nara 630–0192, Japan
[a)] masao@sys.wakayama-u.ac.jp

本論文の内容は2013年7月のマルチメディア，分散，協調とモバイル（DICOMO2013）シンポジウム2013にて報告され，グループウェアとネットワークサービス研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

（以下 OSS）を活用したシステム開発が一般的になりつつある．一方，システム開発を請け負うベンダー企業から見ると，OSS を活用したシステム開発には依然として懸念材料が数多く残されている．独立行政法人情報処理推進機構（IPA）の「第 3 回オープンソースソフトウェア活用ビジネス実態調査」（有効回答数 700 社以上，2009 年度の調査結果）によると，システム開発への OSS 導入に関する企業の懸念事項として，58.8%の企業が「利用している OSS がいつまで存続するか分からない」ことをあげている．

OSS は一般的に，世界中に分散しているボランティアの開発者らの共同作業によって開発が進められている [15]．開発や保守を義務付けられたり強制されることがないため，Netscape プロジェクトのようにコミュニティが崩壊したり，OpenOffice プロジェクトのように分裂したりすることは珍しくない [4]．「利用している OSS がいつまで存続するか分からない」という懸念が生じるのは当然ともいえる．

そのような懸念を払拭するために，OSS 開発の進化に関する研究がさかんに行われている [17]．先行研究の多くは，OSS システムあるいはコミュニティがどのように発展（あるいは衰退）するのかを理解することで，OSS を利用したソフトウェア開発の信頼性に関して有用な知見を導くことを目的としている．ただし，OSS システムあるいはコミュニティを別々の系統としてとらえ，それぞれの系統を個別に分析したものがほとんどである [2], [5], [10]．

一方，Ye らは，OSS システムとコミュニティはともに相互作用しながら進化するものととらえ，共進化のモデル [24] を提案している．共進化とは，複数の進化の系統が互いに影響を与えながら進化していく過程のことである．Ye らのモデルでは，アーティファクト（ソースコードなどのプロダクトのこと），開発者（コミュニティに参加する開発者個人のこと），コミュニティという，3 つの系統が相互に関係し合い，ともに進化することを前提としている．しかし，Ye らの研究 [24] では具体的なコミュニティを対象に定量的に共進化のプロセスを分析していない．

そこで本研究は，Ye らのモデルをベースとして OSS システムとコミュニティの共進化のプロセスを定量的に示すことを目指している．多くの OSS プロジェクトに共通する一般的な知見だけではなく，個々のコミュニティに特有の知見も導出することができれば，OSS 開発に関する懸念の払拭につながると考えている．本論文ではその第一歩として，OSS システムとコミュニティの共進化を定量的に分析するためのデータマイニング手法を提案する．本手法は，時間的順序関係を考慮した相関分析を行うためのものであり，一方の系統の進化が一定時間後に他方の系統の進化に影響を与えるという関係の抽出を支援する．本論文では，提案手法の有用性を確認するために行ったケーススタディについて報告する．Eclipse Platrom プロジェクトを対象に 131 種類のメトリクスを用いてケーススタディを行った結果，従来の相関分析では抽出できない 31 件の関係を提案手法により抽出できた．また，抽出された関係を追加分析することで，Eclipse Platform における共進化のプロセスをより正確に観察できることを確かめた．

本論文の構成は次のとおりである．続く 2 章では，関連研究について述べる．3 章では，提案手法について説明し，4 章で提案手法の有用性を確かめるために行ったケーススタディの結果を示す．5 章ではケーススタディの結果を考察し，最後に 6 章においてまとめと今後の課題について述べる．

## 2. 関連研究

本章ではまず，OSS の進化に関する研究を紹介する．次に，本研究が着目する共進化プロセスのモデルについて述べる．

### 2.1 OSS の進化に関する研究

OSS 開発では，ソースコードだけでなく，これまで報告された不具合とその修正過程，メーリングリスト上での開発者間の議論など，第三者が OSS 開発を理解するために必要となる情報がほぼすべて公開されている．そのため，OSS システムおよびコミュニティの進化について，様々な観点から分析が行われてきた．

OSS の進化に関する既存研究は，以下の 3 つの観点から大別することができる．

- **アーティファクト**：ソースコードの規模推移，モジュールの結合度の経時的変化，パッチ件数の変化 [14] など
- **開発者**：コミッタへの昇格，役割の変化など
- **コミュニティ**：参加開発者の増減，組織構造の変化，活動量（コミット回数やメール流量）の変化など

既存研究の多くは，これらの 3 つの観点を別系統ととらえ，それぞれの進化のプロセスを単体で分析している．

#### 2.1.1 アーティファクトの進化に関する研究

アーティファクトの進化に関する研究は，ソースコードを主に分析対象としている．システム開発ベンダが，利用する OSS を選定するにあたって，安定して開発・保守が続けられているか，複雑になりすぎていないかなどを知る手掛かりとするためである．たとえば，Godfrey らは，Linux カーネルの規模推移をサブシステムごとに調査し，どのサブシステムが最も発展しているかなどを明らかにしている [5]．また，Thomas らは，ソースコードの変更履歴に対して LDA（Latent Dirichlet Allocation）を用いることで，過去から現在に至るまでにどのような機能がコミュニティにおける開発の主流であったかを明らかにしている [21]．その他，OSS 開発に利用されるプログラミング言語の進化について調査した Karus らの研究 [9] や，OSS のライセンス変更を追跡調査した Penta らの研究 [13] などがある．さらに，OSS の信頼性を評価する研究も行われている．た

とえば，Lavazza らは，OSS の信頼性を客観的尺度や知覚品質の点から評価するために，OSS が ECA（Elementary Code Assessment）規約に違反しているかどうか分析を行い，ECA を用いることによって OSS の信頼性を静的に予測可能であることを示唆している [11]．また，Sato らは，汎用の欠陥モジュール予測モデルを構築することを目的に，メトリクスに基づく信頼性予測モデルの適用可能性について実験を行い，コード行数および最大ネストの深さを用いたモデルのみが一定の汎用性を持つことを明らかにしている [16]．

#### 2.1.2 開発者の進化に関する研究

OSS 開発では，コミッタと呼ばれるソースコードを直接書き換えることのできる特別な権限を持つ開発者が存在する．コミッタや管理者は，開発の品質や生産性を左右する重要な役割であるため，コミッタへの昇格プロセスや，開発者の役割変化に関する研究がこれまでさかんに行われてきている．Jensen らのコミッタ昇格プロセスに関する分析では，一般開発者がコミッタになるための方法として，現コミッタからの推薦や投票が存在することを明らかにしている [7]．また，Sinha らはこの開発者がコミッタに昇格する要因をソースコードの書き換え履歴や，報告された不具合の修正履歴などの情報を用いて分析している [20]．コミュニティ内での開発者の役割の変化は，オニオンモデル [23] や，ピラミッドモデル [7] としてモデル化されている．これらのモデルでは，コミュニティ内には階層的に複数の役割が存在することを表している．新規参加者が時間とともに徐々にコミュニティ内で重要な役割を与えられる（役割が変わる）ことにより，開発者がコミュニティへ貢献し続けるための動機付けとなっていることを示すものであり，役割の変化はコミュニティの進化とも関連して重要な要件であるとされている．

#### 2.1.3 コミュニティの進化に関する研究

一般的な OSS 開発におけるプロジェクト参加者の多くは，開発の義務や責任を負わないボランティア開発者である．開発者の自由意思でプロジェクトの加入・脱退を決めることができるため，プロジェクト管理者はコミュニティを維持するために工夫が必要となる．そのため，OSS コミュニティの発展または衰退する要因を明らかにしようとする研究が多数行われてきた．たとえば，Bird らは，生存分析（survival analysis）を用いて，OSS プロジェクトに参加した開発者がプロジェクトを脱退するまでのモデルを構築している [1]．PostgreSQL，Apache，Python プロジェクトを分析した結果，開発者は平均約 1.5 年の「生存期間（プロジェクトに参加する平均期間）」が存在するため，プロジェクトにより長く貢献してもらうためには，1.5 年以内にコミッタ権限を与えるなど，プロジェクト内でより重要な役割をになってもらうことが必要であるとしている．そのほかには，ソーシャルネットワーク分析を用いて OSS コミュニティの組織構造の特徴を明らかにすることで，成功・失敗要因を特定した研究 [2], [8] や，コミュニティを維持するうえで重要な「社会的」ポジションを担う開発者を明らかにした研究 [25] などがある．

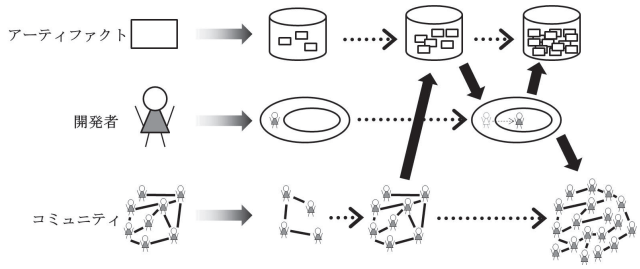



**図 1** OSS 共進化の概念（文献 [24] を参考に改編）
**Fig. 1** The architecture of OSS co-evolution.

### 2.2 OSS システムとコミュニティの共進化

前述の関連研究が OSS の進化の系列を別系統としてとらえたに対して，Ye ら [24] は，OSS システムとコミュニティはともに相互作用しながら進化するものとして，そのプロセスをモデル化している．**図 1** は，Ye らのモデル [24] を模式的に示したものである．図 1 での上段は，ソースコードなどのアーティファクの発展を表している．中段は，OSS 開発に貢献して行くうちに開発者がコミュニティの中心人物へと役割を変化させるなどの開発者の発展を表している．下段は，コミュニティの規模が大きくなっていくなどのコミュニティの発展を示している．

本モデルでは，各系統がそれぞれが独立に進化するのではなく，ある程度の時間的遅延をともなってお互いに影響し合いながら共進化していくことを示唆している．OSS 開発の進化のプロセスを俯瞰的に理解するためには，関連研究のように進化の系統を独立して扱うのではなく，Ye らのモデルのように系統どうしの相互作用をふまえた分析を行うことが合理的である．しかしながら，Ye らの研究で，客観的な定量的データとして共進化のプロセスを示すまでは行っておらず，実証的な調査が必要となる．そこで我々は，OSS 開発における共進化のプロセスを定量的に分析するためのデータマイニング手法を構築するという発想を得た．

### 2.3 共進化モデルに基づく定量的分析

Ye らの共進化のモデルを定量的に分析するためには，3 種類の進化の系統がお互いにどのような影響を与えているかを調べる方法が必要となる．以下に，本研究で構築するデータマイニング手法が備えるべき要件と本研究における方針をまとめる．

- 時系列データを扱えること：各系統進化のプロセスは基本的に時系列データ（たとえば，ソースコードの規模推移など）として計測されるため，時系列データを扱えることがまず必要となる．本研究では，長期にわ

たる共進化のプロセスを対象としているため，一定区間（たとえば1カ月間）ごとのデータをまとめたデータポイントとして測定することで時系列データを取り扱う．

- 各系統間の関係を定量化できること：Yeらの共進化のモデルに基づいて共進化のプロセスを分析するためには，各系統間の関係を定量化する必要がある．信号処理などの分野では，2種類の時系列データの類似性を評価することで時系列データ間の関係を調べる相互相関分析（cross-correlation analysis）[19] を用いるのが一般的である．対象となる時系列データは，定常的な確率過程に基づいていることが前提となる．しかし，OSS開発では，開発メンバの不規則な増減 [26]，予期しない不具合への対応 [18]，不具合報告の急増によるリリース計画の見直し [22] など，突発的な事象に対応しながら各系統が進化する．定期的あるいは周期的に発生するイベントは稀であるため，計測される時系列データを定常的な確率過程に基づくものとして取り扱うことは困難である．したがって，OSS開発データでは，2つの変数の値を時間的に同期させながら計測し，2変数間の関係をノンパラメトリックな手法で調べる必要がある．本研究では，スピアマンの順位相関係数を用いて系統間の関係を調査できるようにする．
- 時間的遅延をともなう系統間の関係を分析できること：前節で述べたように，OSS開発の進化のプロセスは，各系統がある程度の時間的遅延をともなってお互いに影響し合いながら共進化することが考えられるため，時間的遅延を考慮して2変数間の関係を調べる必要がある．スピアマンの順位相関係数を用いた通常の相関分析では時間的遅延を考慮できないため，後述する遅延相関分析を用いる．
- 各系統を複数のメトリクス（変数）で定義し，系統間の関係を探索的に分析できること：Yeらの共進化プロセスモデルでは，各系統を表す具体的なメトリクスが定義されていない．どのメトリクスがどのメトリクスに影響を与えるのかは未知であるため，各系統を表すメトリクスをできるだけ多く用いて，関連の深いメトリクスペアを探索的に発見する必要がある．本研究では，ソフトウェアリポジトリから機械的に取得可能なメトリクスを対象とする．

以上の要件に基づいて，本研究では時間的順序関係を考慮したデータマイニング手法を構築した．次章では，手法の詳細について説明する．

## 3. 時間的順序関係を考慮した相関分析

### 3.1 概要

OSS開発における共進化のプロセスを定量的に分析するためのデータマイニング手法を提案する．提案手法は，ソフトウェア開発プロセスおよびプロダクトから計測される様々なメトリクスを入力として，相関がみられるメトリクスのペアのみを自動的に抽出するためのものである．特に，竹内らの時系列データ解析手法 [29] を参考に，ペアとなるメトリクス間の時間的順序関係を考慮した相関分析（以降では，遅延相関分析と呼ぶ）を行う点に特徴がある．遅延相関分析を用いることにより，説明変数となる一方のメトリクスの一定期間の平均値と，目的変数となるもう片方のメトリクスの変化量が，時間的な遅延をともなって関係するかどうかを確かめることができる．また，提案手法は，遅延相関分析における各種パラメータ（遅延係数，加算係数）の最適値を，遅延相関係数が最大になるよう自動的に求める処理が含まれる．本手法を用いて計測した多数のメトリクスを解析することで，分析者は遅延の大きさや分析窓の大きさを気にせず，最も相関係数が大きくなるメトリクスの組合せのみを分析対象とすることができる．以降では，提案手法の各処理（前処理，遅延相関分析，後処理）の詳細について説明する．

### 3.2 前処理

提案手法で入力となる時系列データは，ソフトウェアやその開発プロセスから計測することができる．ただし，ソフトウェア開発データは外れ値を含む場合が多い．ソフトウェア開発では，先ほど述べたようにイベントによる影響を受けやすく，イベントの影響によってソフトウェア開発データの値が著しく大きくなる場合がある．たとえば，重大な不具合が見つかることによって，著しくソフトウェアの変更回数が多くなることなどがある．このような場合，ソフトウェアの変更回数を計測すると，ある時点にのみ著しく値が大きい点（外れ値）が発生する．分析結果の信頼性を確保するためには，外れ値をあらかじめ除去しておく必要がある．また，外れ値を除去した時系列データの適切な補完が必要となる．さらに，計測されるメトリクスは，単位やスケールの異なるデータ（たとえば，コード行数とメソッド数）であるため，任意の2つのメトリクスの相関をすべて算出する前にデータを正規化しておく必要がある．

#### 3.2.1 外れ値除去

時系列データに対する外れ値検出では，2次元空間内の距離の基づく外れ値検出が必要になる [6]．本研究では，マハラノビス汎距離を用いて，データの分布を考慮した多次元データに対する外れ値検出し除去する．

#### 3.2.2 データ補完

外れ値除去によって削除した外れ値の点を補完する．本研究では，与えられた複数の点を通る曲線を描くことで補間を行うスプライン補間を用いる．

#### 3.2.3 正規化

データの単位やスケールを揃えるために，式 (1) によりすべてのデータを0から1の間の値で正規化する．

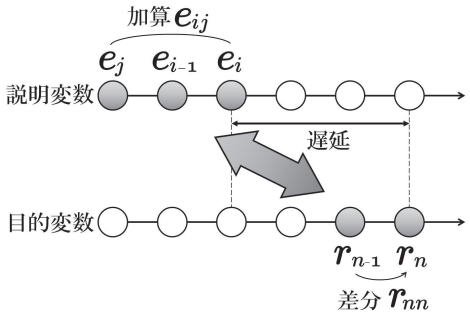



図 2 遅延相関分析の概念図（文献 [28] を参考に改編）

**Fig. 2** A concept of time-delayed correlation analysis (revised based on Ref. [28]).

$$x_{i_{nor}} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{1}$$

$x$ は入力される時系列データを示し，$x_i$ は時刻 $i$ における $x$ の値を示す．また，$x_{i_{nor}}$ は時刻 $i$ における正規化された値を示す．$max(x)$ は時系列データ $x$ の最大値，$min(x)$ は時系列データ $x$ の最小値を表現している．

### 3.3 遅延相関分析

前処理後，すべてのメトリクスの組合せに対して相関分析を行う．ただし，従来の相関分析は，時間的なずれのともなう 2 種類の時系列データの相関を求めることができないため，本研究では，時間的順序関係を考慮した相関分析である遅延相関分析を用いる．

本研究における遅延相関分析は，竹内らによって提案されている時系列データ解析手法 [29] を参考に，説明変数の一定期間の平均値が一定期間後に目的変数の変化量に影響を及ぼす，という考え方に基づいている（**図 2**）．竹内らの手法は，消費エネルギーなどの生活習慣データと体脂肪率などの健康データの因果関係を明らかにするために，生活習慣データ（説明変数）の蓄積と一定期間後の健康データ（目的変数）の変化との関係を分析するためのものである．たとえば，消費エネルギーが多いと体脂肪率が減少することは明らかであるが，1 日消費エネルギーが多いからといって体脂肪率が減少するとは限らない．そのため，竹内らの手法では，一定期間の生活習慣データの蓄積を表現するために説明変数の値を加算し，健康データの変化を表現するために目的変数の値の差分をとっている．本研究が対象とする OSS 開発における共進化のプロセスにおいても，同様の事象を観測できると考えられる．たとえば，プロジェクト途中での開発者の追加投入は，短期的には生産性の向上に寄与せず（場合によっては生産性が低下する），効果が現れるまでには一定の時間を要する [3] ことはソフトウェア工学分野では広く知られている事象であり，本研究が竹内らの手法に依拠する理由となっている．

遅延相関分析では，説明変数の値を平均する期間を加算係数（$i-j$），目的変数の値が変化するまでの期間を遅延係数（$n-i$）とするパラメータ（**表 1**）を，任意の期間（たとえば，0 から 12 カ月）に対して設定する．竹内らの手法とは異なり，本研究では，説明変数を移動平均として表現する．ソフトウェアリポジトリから取得されるデータからは，状態量を表すメトリクス（たとえば，コード行数）や時間的に計測される量（たとえば，不具合修正時間）など様々な種類の「量」を計測することになる．したがって，竹内らの手法と同様にこれら異なる種類の量を単純に累積するのではなく，一定期間でのメトリクス値の平均としてとらえるのが妥当であると考えた．

表 1 パラメータ一覧

**Table 1** Parameters for time-delayed correlation analysis.

| パラメータ名 | 定義式 |
|---|---|
| 加算係数 | $i-j$ |
| 遅延係数 | $n-i$ |

提案手法は，すべてのメトリクスの組合せに対して，各パラメータのすべての値を組み合わせて遅延相関係数を算出する．たとえば，計測されたメトリクスが $N$ 種類存在し，各パラメータを 1 から 12 カ月の期間として設定した場合，$N(N-1)\times 12^2$ 種類の遅延相関係数が算出される．ただし，メトリクスの組合せとパラメータの組合せから，計算結果が膨大な数になることが予想される．そのため，分析者の労力を考慮して，$N(N-1)$ 種類のペアそれぞれに対して，最大の遅延相関係数とその場合の各パラメータの値のみを出力するようにしている．以降では，説明変数および目的変数の変化量の求め方と，遅延相関係数の求め方について述べる．

#### 3.3.1 説明変数および目的変数の変化量の算出

遅延相関分析では，説明変数の一定期間の平均値と目的変数の変化量の間の相関を求める．説明変数の一定期間の平均は，加算係数の値に従って説明変数の値の平均を求めることによって表現する．たとえば図 2 において，$e_i$ は時刻 $i$ における説明変数の値，$e_j$ は時刻 $j$ における説明変数の値である．時刻 $i$ における，説明変数の値の平均を $e_{ij}$ とすると，$e_{ij}$ は式 (2) で表される．

$$e_{ij} = \frac{e_i + e_{i-1} + \cdots + e_j}{i-j+1} \tag{2}$$

目的変数の変化量は，目的変数の値の差分を求めることによって表現する．図 2 において，$r_n$ は時刻 $n$ における目的変数の値とする．時刻 $n$ における，目的変数の変化量を $r_{nn}$ とすると，$r_{nn}$ は式 (3) で表される．

$$r_{nn} = r_n - r_{n-1} \tag{3}$$

式 (3) の定義上，目的変数の値は，1 時点前の目的変数の値との差分をとった値のみを考慮している．本来は目的変数も説明変数と同様にある程度の時間幅を調整可能にするべきであるが，説明変数および遅延係数のパラメータの

組合せがすでに膨大になることが予想されるため，得られる結果の解釈と提案手法の有用性を確認するために，本論文では目的変数の値を最小の差分をとって表現することとした．目的変数に時間幅を持たせた分析については今後の課題としたい．

また，本論文では，説明変数を移動平均として，目的変数を1時点前の目的変数の値との差分をとったものとして定義しているが，一定期間内の変数の値の和や差，あるいは値そのものを用いて算出することも理論上可能であり提案手法の有用性を大きく向上させる可能性がある．本論文では誌面の都合上，それらすべてを考慮した分析を行うことはできないが，提案手法の改良点として今後取り組む必要がある．

#### 3.3.2 遅延相関係数の算出

遅延相関係数は，図2に示すとおり，説明変数を加算係数の値に基づいて平均値をとった $e_{ij}$ と，遅延係数の値 $n-i$ に基づいてシフトさせた目的変数の変化量 $r_{nn}$ でペアを作り相関係数として算出する．ソフトウェア開発データは，正規分布に従うことが少ないため [27]，相関係数の算出には，スピアマンの順位相関を用いる．

なお，目的変数の変化量を表現する $r_{nn}$ の値が小さすぎる場合，高い相関が観察されたとしても結果的に間違った解釈になってしまうことも考えられる．特に活動が長期にわたるプロジェクトを対象として分析を行う場合，プロジェクトの活動が停滞している時期が含まれることがある．プロジェクトの停滞時期に計測されるデータを含めて相関係数を求めた場合，多くのメトリクスペアの間で相関がほとんど見られないという結果になる可能性が高い．そのため，提案手法では，目的変数の変化量（$r_{nn}$ の値）が小さいデータを除去する処理が必要であると考えられる．提案手法では，目的変数となるメトリクスの値が1時点前の値と比べて5%以下の変化率しかない場合，目的変数の変化量 $r_{nn}$ と，$r_{nn}$ とペアとなる説明変数の累積値 $e_{ij}$ を除去する．

### 3.4 後処理

提案手法に含まれる後処理では，遅延相関分析によって得られた相関に有意差があるかを統計的に検証する．また，各パラメータの値の最適な組合せを求めるために，最大相関係数の更新を行う．

#### 3.4.1 相関係数の検定

遅延相関分析では，相関係数のみを求めるため，得られた相関に有意差があることを統計学的に確かめる必要がある．提案手法では，ノンパラメトリック検定法として知られているマン・ホイットニーのU検定を行う．p値が0.05より小さければ，時系列データ間に有意差があると判定し，それ以外の場合は，結果を出力しない．

#### 3.4.2 最大相関係数の更新

相関係数の検定の結果，有意差がみられたすべてのメトリクスペアに対して最大相関係数の更新を行う．最大相関係数の更新を行うことによって，算出された相関係数の絶対値が最大となるメトリクスペアの各パラメータの値およびその相関係数のみを出力する．

#### 3.4.3 分析結果の出力

以上の処理により，計測したメトリクスのすべての組合せ（すべての説明変数と目的変数の組合せ）に対して，指定する最大相関係数の絶対値（閾値）を超える場合のみ，最大相関係数および各パラメータの値を出力する．提案手法の後処理によって出力結果が絞られるため，分析者は，どのメトリクスがどれくらいの期間（$i-j$）変化すると，どのメトリクスにどれくらい遅延（$n-i$）して影響を与えているかについて，効率的に分析することができる．

## 4. ケーススタディ

### 4.1 目的

山谷らは先行研究 [30] において遅延相関分析手法を提案し，3種類のメトリクス（パッチ数，コミッタ昇格数，総開発者数）を用いて2つのOSSプロジェクト（Apache HTTPDおよびEclipse Platform）を対象にケーススタディを行っている．ケーススタディの結果，時間的な順序関係を考慮しない従来の相関分析に比べ，遅延相関分析はOSS開発における共進化のプロセスをより正確に観察するための手段として有用であることを示した．ただし，文献 [30] では，(1) 分析期間が短いこと（4年間のプロジェクトデータ），(2) メトリクスの種類が少なすぎること，(3) 最大相関係数とその際の各パラメータに関する議論にとどまっていること（なぜメトリクス間に時間的に遅延した相関が見られるのかについての議論が少ないこと），(4) 本論文における前処理および後処理の一部（相関係数の検定）を含まない手法であったことなど，ケーススタディによって得られた知見にはいくつかの制約があった．

本論文では，提案手法の実践的な適用を想定して，Eclipse Platformプロジェクトを対象としたケーススタディを行う．本ケーススタディの目的は，通常の相関分析に加え，遅延相関分析を併用した分析を行い，相関分析のみを用いた従来の分析方法よりも有用性の高い分析が行えることを示すことである．具体的には，2003年7月から2012年6月までのプロジェクトデータから取得可能な131種類のメトリクスに対して提案手法および相関分析手法を適用する．提案手法および相関分析手法それぞれが系統内・系統間の関係をどの程度抽出できるのかを明らかにするとともに，提案手法のみによって抽出可能な関係を詳細に分析することで，パラメータ間の有意な関係を広く探索的にマイニング可能な提案手法の有用性を示す．

表 3 対象とする Git リポジトリ
**Table 3** Target Git repositories.

| Git リポジトリ名 | ファイル数 | Java ファイル数 | コード行数 |
|---|---|---|---|
| eclipse.platform | 1,905 件 | 943 件 | 97,358 |
| eclipse.platform.common | 3,750 件 | 0 件 | 0 |
| eclipse.platform.debug | 1.780 件 | 1,325 件 | 122,049 |
| eclipse.platform.releng | 524 件 | 128 件 | 14,380 |
| eclipse.platform.releng.eclipsebuilder | 311 件 | 1 件 | 327 |
| eclipse.platform.resources | 1,008 件 | 701 件 | 96,258 |
| eclipse.platform.runtime | 649 件 | 478 件 | 44,618 |
| eclipse.platform.swt | 3,565 件 | 2,792 件 | 538,622 |
| eclipse.platform.team | 2,587 件 | 1,609 件 | 192,882 |
| eclipse.platform.text | 1,475 件 | 1,075 件 | 123,664 |
| eclipse.platform.ua | 2,337 件 | 969 件 | 92,377 |
| eclipse.platform.ui | 2,337 件 | 6,134 件 | 601,282 |
| 合計 | 28,640 件 | 16,155 件 | 1,923,817 |

表 2 対象とするデータソース
**Table 2** Target data sources.

| データソース | 件数 |
|---|---|
| ソースコード | 28,640 件 |
| ソースコードの変更履歴 | 90,190 件 |
| 不具合修正履歴 | 96,387 件 |
| Eclipse News Forums | 185,181 件 |

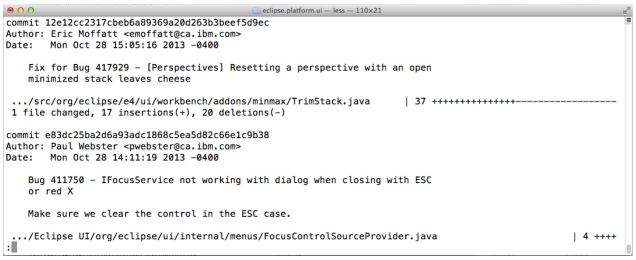



図 3 ソースコードの変更履歴の例
**Fig. 3** An example of change history of source code.

### 4.2 データセット

#### 4.2.1 対象プロジェクト

ケーススタディは，Eclipse Platform プロジェクトを対象に行う．Eclipse Platform は，主に IBM 社の開発者の貢献によって開発・保守が長期間行われてきたプロジェクトであり，提案手法の実践的な適用を想定することができる．また，他の OSS プロジェクトと同様，開発履歴が各種リポジトリに保存されているため，様々なメトリクスを計測することができる．さらに，数多くの研究で分析対象とされているため，提案手法を用いて得られた分析結果や知見に新規性があるかどうかを判断しやすい．

#### 4.2.2 対象データ

ケーススタディで対象とするデータソースと件数を**表 2** に示す．表 2 の件数は，2003 年 7 月から 2012 年 6 月までのものである．以下では，各データソースについて説明する．

**ソースコード** **表 3** は，ソースコードの収集対象とした Git リポジトリとソースコードの統計量（2012 年 6 月時点でのファイル数および Java ファイル数，ソースコードの行数）を示している．Eclipse Platform プロジェクトにおけるソースコードは，版管理システムである Git で管理されている．本ケーススタディで対象とした Eclipse Platfrom に関連する Git リポジトリは，2013 年 9 月に Ohloh[*1]の Code Locations に設定されているリポジトリに限定した．ケーススタディの対象となったソースコードは，合計 28,640 件あった．

**ソースコードの変更履歴** Eclipse Platform プロジェクトにおけるソースコードの変更履歴も，版管理システムである Git にコミットログとして記録されている．**図 3** は，Git コマンドを入力として得られたソースコードの変更履歴の例である．コミットログには，コミット ID，コミッタのメールアドレス，コミット日時，コミットに対するコメント，変更したファイル名とその変更行数などが含まれる．ケーススタディの対象となったコミットログは，合計 90,190 件あった．

**不具合修正履歴** Eclipse Platform プロジェクトにおける不具合修正履歴は，不具合管理システムである Bugzilla[*2]で管理されている．Bugzilla では，1 つの不具合報告に対して，不具合情報が記載されたページと不具合の修正履歴ページが作成される．**図 4** に示すように，不具合情報が記載されたページには，固有の ID 番号や，不具合が発生したプロダクト，不具合の再現方法などの情報が記録されており，利用者または開発者が投稿した不具合に対する意見や質疑なども不具合情報が記載されたページで行われている．また，変更履歴ページには，不具合の状態や不具合担当

[*1] Ohloh, the open source network, http://www.ohloh.net/

[*2] Eclipse Bugzilla, https://bugs.eclipse.org/bugs/

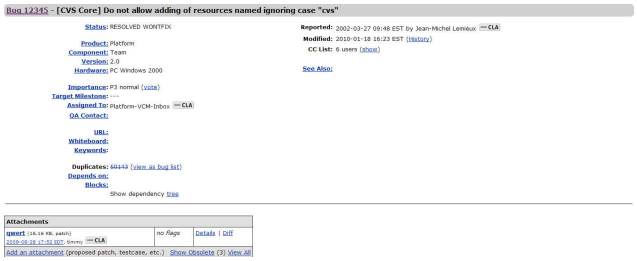



図 4 不具合情報が記載されているページの例
**Fig. 4** An example of a bug report in Bugzilla.

表 4 対象とするニュースグループ
**Table 4** Target news groups.

| ニュースグループ名 | 投稿数 |
|---|---|
| eclipse.platform | 84,227 件 |
| eclipse.platform.jface | 2,147 件 |
| eclipse.platform.pde | 4,703 件 |
| eclipse.platform.rcp | 44,770 件 |
| eclipse.platform.swt | 47,625 件 |
| eclipse.platform.ua | 1,682 件 |
| eclipse.financial-platform | 31 件 |
| 合計 | 185,181 件 |

者，不具合の重要度・優先度などの不具合に関する情報が「いつ，だれによって，どのように変更されたのか」が記録されている．ケーススタディでは，プロダクト（Product）として Platform が指定されている合計 96,387 件の不具合報告を対象とする．

**Eclipse News Forums** Eclipse Platform プロジェクトにおける開発間のコミュニケーションは Eclipse News Forums [*3]で行われる．Eclipse News Forums は，主に開発者が技術的な問題を議論する場として用いられている．Eclipse News Forums では，サブプロジェクトごとや，コンポーネントごとに，ニュースグループが設けられている．ケーススタディでは，グループ名に Platform を含むものをのみ対象とした．対象とするニュースグループおよびその投稿数を**表 4** に示す．

### 4.3 対象メトリクス

ケーススタディでは，対象データソースから抽出可能な各系統を定義する 131 種類のメトリクスを対象とする．提案手法で扱うデータは時系列データである必要があるため，抽出するメトリクスは 1 カ月ごとに集計したものを用いる．アーティファクトを定義するメトリクスとして 67 種類のメトリクスを抽出する．アーティファクトはソースコードなどのプロダクトを指しているため，ソースコードから抽出するメトリクスや，不具合の数などのメトリクスをアーティファクトの進化を示すメトリクスとして抽出する．また，開発者の系統を定義するメトリクスとして 8 種類のメトリクスを抽出する．開発者の進化とは，開発者の役割の変化やコミッタへの昇格を指していることから，新しいコミッタ数などの各役割を初めて行った開発者の数を開発者の進化を示すメトリクスとして抽出する．さらに，コミュニティの系統を定義するメトリクスとして 56 種類のメトリクスを抽出する．コミュニティの規模を示すメトリクスとして開発者の数，コミュニティの活動を示すメトリクスとしてコミット数などのメトリクスを抽出する．

ソースコードから抽出するメトリクスは，ソースコード解析ツールである Understand [*4]を用いる．Eclipse Platform プロジェクトは主に Java によって開発されているため，解析対象とするファイルは Java ファイルに限定した．ソースコードメトリクス以外のメトリクスは，自作のスクリプトを用いて抽出した．分析に用いたメトリクスの一覧は，Web 上[*5]で公開しているのでそちらを参照していただきたい．

### 4.4 分析手順

ケーススタディでは，従来の相関分析に加え，遅延相関分析を併用した分析を行い，相関分析のみを用いた従来の分析方法よりも有用性の高い分析が行えることを示す．そのため，まず，131 種類のメトリクスのすべての組合せについて，通常の相関分析手法および提案手法を用いた分析を行う．なお，提案手法における各パラメータがとり得る範囲は，加算係数（$0 \leq i-j \leq 11$），遅延係数（$0 \leq n-i \leq 12$）とする．時系列データの時間間隔が 1 カ月であることから，パラメータの単位は 1 カ月となる．たとえば，遅延係数が 1 であるときは，メトリクス間の関係に 1 カ月の遅延が発生することを指している．また，本研究では 131 種類のメトリクスを用いているため，説明変数および目的変数に割り当てるメトリクスの組合せが 17,030 通り（${}_{131}\mathrm{P}_2$）存在し，各組合せに対して 12 種類の加算係数と 13 種類の遅延係数を用いて，相関係数を算出する．出力結果が膨大な数になるため，本ケーススタディでは，相関係数の絶対値が 0.4 以上の場合にメトリクス間に関係があると判断する．

次に，遅延相関分析の場合のみ相関が強くみられた関係について，従来の相関分析によって得られる結果と比較しながら詳細に分析し，相関分析のみを用いた従来の分析方法では発見できないメトリクス間の因果関係を解明するのに役立つことを示す．

### 4.5 分析結果

#### 4.5.1 遅延をともなう相関関係

時間的順序関係を考慮しない通常の相関分析で 131 種類

*3 Eclipse News Forums, http://www.eclipse.org/forums/
*4 Understand, Source Code Analysis & Metrics, http://www.scitools.com/
*5 http://oss.sys.wakayama-u.ac.jp/yamatani/ipsj2014

表 5 提案手法のみで抽出することのできた相関関係
Table 5 Correlations extracted only from the proposed method.

| 説明変数 | 目的変数 | 加算係数 | 遅延係数 | 相関係数 | p 値 |
|---|---|---|---|---|---|
| 新しいコミッタによる説明文の文字数 | Duplicate になった不具合件数 | 0 | 7 | −0.42 | 0.00 |
| 新しいコミッタによる説明文の文字数 | Duplicate になった不具合件数 | 0 | 7 | −0.42 | 0.00 |
| 1 コミットに対する変更したファイル数 | スレッドの平均文字数 | 0 | 7 | −0.48 | 0.00 |
| 1 コミットに対する変更したファイル数 | Priority が 1（高）である不具合件数 | 1 | 3 | −0.44 | 0.00 |
| 1 コミットに対する変更したファイル数 | Wontfix になった不具合件数 | 0 | 11 | −0.45 | 0.00 |
| 1 コミットに対する変更したファイル数 | 不具合を修正した人数 | 0 | 0 | −0.49 | 0.00 |
| コードオーナが削除した行数 | 新しいコミッタによる説明文の文字数 | 0 | 4 | −0.42 | 0.00 |
| コードオーナが削除した行数 | 関数・メソッドのコード行数の平均 | 0 | 5 | −0.47 | 0.00 |
| コードオーナが追加した行数 | 新しいコミッタが削除した行数 | 1 | 8 | −0.43 | 0.00 |
| 新しいコミッタが削除した行数 | 不具合管理に関係があった件数 | 1 | 0 | 0.41 | 0.02 |
| 新しいコミッタが追加した行数 | 不具合の情報が変更された件数 | 1 | 0 | 0.42 | 0.03 |
| 新しいコミッタが追加した行数 | 不具合管理に関係があった件数 | 1 | 0 | 0.47 | 0.04 |
| 新しいコミッタが変更したファイル数 | コミッタ数 | 1 | 9 | −0.43 | 0.00 |
| 新しいコミッタが変更したファイル数 | Priority が 1（高）である不具合件数 | 3 | 1 | −0.45 | 0.00 |
| コードオーナが行数を削除したコミット回数 | Priority が 5（低）である不具合件数 | 10 | 12 | −0.40 | 0.00 |
| コードオーナが行数を追加したコミット回数 | Priority が 5（低）である不具合件数 | 10 | 12 | −0.4 | 0.00 |
| コードオーナのコミット回数 | Priority が 5（低）である不具合件数 | 11 | 12 | −0.40 | 0.00 |
| リプライの平均行数 | スレッドの平均行数 | 0 | 2 | −0.42 | 0.00 |
| スレッドの平均文字数 | Wontfix になった不具合件数 | 0 | 4 | −0.44 | 0.00 |
| 平均文字数 | スレッドの平均行数 | 0 | 2 | −0.42 | 0.00 |
| 1 スレッドに対するリプライ数 | Wontfix になった不具合件数 | 1 | 10 | −0.42 | 0.00 |
| 報告されてから修正されるまでの時間 | Priority が 1（高）である不具合の数 | 0 | 3 | 0.41 | 0.00 |
| 不具合修正にかかった時間 | スレッドの平均行数 | 0 | 3 | −0.41 | 0.00 |
| **不具合修正にかかった時間** | **Priority が 1（高）である不具合の数** | **0** | **3** | **0.50** | **0.00** |
| 不具合 1 件あたりの説明文の文字数 | 関数・メソッドのコード行数の平均 | 0 | 6 | −0.48 | 0.00 |
| 不具合 1 件あたりの説明文の文字数 | Wontfix になった不具合件数 | 0 | 11 | −0.46 | 0.00 |
| 再割当てが発生した割合 | Priority が 5（低）である不具合の数 | 11 | 11 | −0.40 | 0.00 |
| Severity が Trivial である不具合件数 | スレッドの平均行数 | 0 | 11 | −0.44 | 0.00 |
| WorksForMe になった不具合件数 | スレッドの平均行数 | 0 | 10 | −0.47 | 0.00 |
| Invalid になった不具合件数 | スレッドの平均行数 | 0 | 10 | −0.43 | 0.00 |
| Verified になった不具合件数 | スレッドの平均行数 | 0 | 12 | −0.41 | 0.00 |
| 新しく不具合の情報を変更した人数 | スレッドの平均行数 | 0 | 5 | −0.41 | 0.00 |

のメトリクス間の関係を分析した結果，751 件の相関を抽出できた．通常の相関分析の結果とは，提案手法の前処理を行った時系列データに対して加算や差分をとるなどといった処理を行わず 131 種類のメトリクスの組合せに対して相関分析を行った結果のうち相関係数の絶対値が 0.4 以上であった関係を指している．通常の相関分析では，時間的順序関係を考慮しないため，8,515 通り（$131 \times 130 \div 2$）のメトリクスの組合せで相関分析を行った．一方，提案手法により抽出できた相関は 171 件であった．また，171 件のうち，提案手法によってのみ抽出できた相関は 31 件あった（**表 5**）．

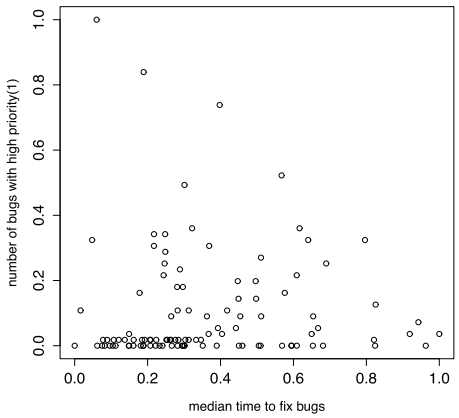
表 5 に，抽出することのできた関係およびその関係の強さ，各パラメータの値，p 値を示す．たとえば，表 5 中に太字で示した相関係数が最大となった関係は，「加算係数の値が 0，遅延係数の値が 3 の際に，不具合修正時間と優先度が P1 である不具合の数に中程度の相関がみられる（相関係数 0.50）」というものであり，「不具合の修正に多くの時間を要する状態が 1 カ月続くと，3 カ月後に優先度の高い（P1）の不具合数が増加する傾向がある」ことを示唆している．

#### 4.5.2 詳細分析の結果

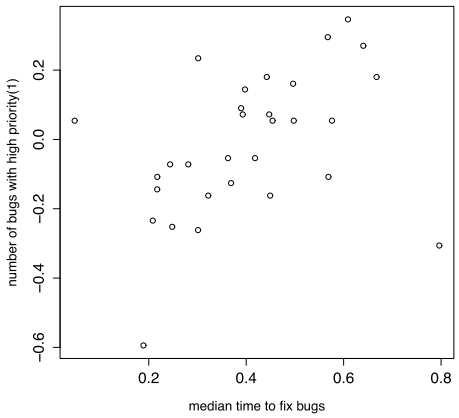
ケーススタディにおいて相関が最も強くみられた関係について，詳細に分析した結果について述べる．

表 5 から，「不具合修正時間」と「優先度が 1 である不具合の数」に最も強い相関（相関係数 0.50）が見られた．**図 5** (a) に従来の相関分析を用いた場合の散布図，図 5 (b) に提案手法を用いた場合の散布図を示す．データ数が従来の相関分析で 108，遅延相関分析で 30 となっている理由は，3.3.2 項で述べたように，目的変数の変化量が少ない（0.05 以下）場合にデータを除去する処理を提案手法が行っているためである．図 5 より，従来の相関分析では 2 つのメトリクス間には相関がみられないことが分かる．一方，図 5 (b) から，提案手法を用いた相関分析では，2 つのメトリクス間の相関を見て取ることができる．

遅延相関分析を用いることで明らかになった「不具合の修正に多くの時間を要するようになると，3 カ月後に優先度の高い不具合数が増加する」関係が成り立つ理由を明らかにするために，Eclipse Platform プロジェクトに報告さ




(a) 遅延を考慮しない場合（データ数：108，相関係数：0.20）




(b) 遅延を考慮した場合（加算係数：1，遅延係数：3，データ数：30，相関係数：0.50）

**図 5** 不具合修正時間と優先度 1 の不具合数の散布図
**Fig. 5** Scatter plots for bug-fix time and P1 bugs.

**表 6** 優先度ごとの不具合の割合
**Table 6** Ratio of bugs by priority.

| 優先度 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 割合 [%] | 2.14 | 5.28 | 87.48 | 2.34 | 2.75 |

れたすべての不具合の優先度の分布を調べた．**表 6** が示すように，87.48%の不具合が優先度が 3 となっており，他の OSS プロジェクトに対して行われた分析 [12] と同様，管理者は不具合修正の優先度を厳密に決定しているわけではないことが分かる．Eclipse プロジェクトでは，マイルストーンが定められており毎年 6 月にメジャーバージョンがリリースされる．メジャーバージョンがリリースされてからしばらくは，報告された不具合を優先度 3 で管理するため，次のメジャーバージョンのリリースに向けたテストが実施されるまでは不具合修正時間が徐々に大きくなると考えられる．テスト実施後は，次のリリースまでにどの不具合を修正しどの不具合を修正しないかを決定する必要があることから，確実にリリースに間に合うように，優先度 3 として管理されていた不具合の中から修正すべき不具合を優先度 1 に変更して対処しているものと思われる．

上記のような関係は，通常の相関分析では抽出することはできず，また，プロジェクトの当事者でなければその背後にある因果関係を理解することは困難である．提案手法を用いて抽出される関係は，時間的な遅延をともなうため意外性の高い関係が多く含まれるものの，上記のような追加の分析や既知の知見を組み合わせることで，外部の者であってもある程度因果関係を推測することができる．本ケーススタディのような大規模プロジェクトを対象とした分析は，他のプロジェクト管理者にとってもプロジェクトマネジメントを考える上で重要な指針になると考えられる．本節で示した結果は，提案手法が共進化のプロセスの解明にとどまらず，実用的に役立つ知見を導くのに役立つ可能性があることを示唆するものと考えることができる．

## 5. 考察

### 5.1 相関分析との比較

提案手法では，パラメータの値のすべての組合せについて時系列データの処理を行い，メトリクス間の相関を分析しているため，従来の相関分析よりも抽出することのできる関係は多くなると予想していた．しかし，ケーススタディの結果から，従来の相関分析で抽出することのできた関係が 751 件，提案手法により関係が抽出することができた件数が 171 件と，従来の相関分析手法の方が抽出することのできた関係が多いという結果になった．

このような結果になった理由は，今回用いた 131 種類のメトリクスの中に，類似したメトリクスのペアが多く含まれていたことが原因であると考えられる．たとえば，ソースコードの規模を測るメトリクスとして，メソッドの平均行数や，メソッドの平均コメント行数などを用いているが，これらのメトリクスの値は時間的に類似して推移するため，従来の相関分析では，類似するメトリクスのペアに対しても高い相関が見られることになる．

一方，提案手法は，説明変数の状態が一定期間後の目的変数の変化にどのように影響を与えるのかという考えに基づいた遅延相関分析を行い，説明変数の変化量と目的変数の変化量について時間的順序関係を考慮した相関分析を行っている．時間的にも類似して推移する類似メトリクスのペアは，提案手法により除去されるため，結果的に従来の相関分析の結果よりも少ない関係のみが抽出されたものと考えられる．

### 5.2 ソフトウェア開発現場への適用

本研究では，Eclipse Platform プロジェクトの 9 年間のソフトウェア開発データから計測されたメトリクスを用いてケーススタディを行った．4.5.2 項で示したように，抽出した関係に対して追加の分析を行うことで，不具合修正時間が長期化すると不具合の優先度が変更されやすいといった，Eclipse Platform プロジェクト特有の傾向を明ら

表 7 系統内・系統間でみられた関係の数
Table 7 Number of correlations among/between evolution systems.

| | 系統内 | 系統間 | 合計 |
|---|---|---|---|
| 従来の相関分析でみられた関係 | 440 | 311 | 751 |
| 提案手法のみでみられた関係 | 10 | 21 | 31 |

かにすることができた．プロジェクトの傾向を掴むことは，OSS 開発だけでなく，長期間にわたる保守運用が必要となる企業のソフトウェア開発の現場において役立つものであると考えられる．

長期間にわたる保守運用が必要となる開発プロジェクトにおいては，保守運用の担当者がすべての期間において不変であることは考えにくく，保守運用の担当者の変更が起きるものと考えられる．保守運用の担当者の変更が起きた際，新しく保守運用の担当になった者は，これまでのプロジェクトの傾向を詳細を知らない可能性が高く，保守運用の作業に影響を及ぼすことも考えられる．提案手法は，プロジェクト固有の傾向を知るための手段として，企業におけるソフトウェア開発の現場においても有用である．

### 5.3 OSS 開発における共進化の関係

**表 7** に，系統内，系統間それぞれの遅延を考慮しない従来の相関分析で抽出することのできた結果および，遅延相関分析のみで抽出することのできた結果を示している．従来の相関分析で抽出することのできた結果 751 件のうち，440 件が系統内の関係（たとえば，ネスト値が増加すると複雑度が増加するなどのアーティファクトに関するメトリクス間の関係）であった．同様に系統間の関係（たとえば，不具合が増加すると Forum の投稿数が増加するなどのアーティファクトに関するメトリクスとコミュニティに関するメトリクスとの関係）は 311 件であった．また，遅延相関分析のみで抽出することのできた結果 31 件のうち，系統内の関係は 9 件，系統間での関係は 22 件であった．従来の相関分析で抽出することのできた結果は，系統間の関係よりも系統内の関係の方が多いことに対し，遅延相関分析のみで抽出することのできた結果は，系統内の関係よりも系統間の関係の方が多い結果となった．このように，従来の相関分析だけでなく，遅延相関分析を行うことで，OSS 開発における共進化の関係をより正確に理解することができると考えられる．

また，遅延相関分析のみみられた系統間の関係 22 件のうち，開発者 → コミュニティの関係（説明変数に開発者のメトリクス，目的変数にコミュニティのメトリクスを割り当てた関係）が 1 件，アーティファクト → コミュニティの関係が 4 件，コミュニティ → アーティファクトの関係が 17 件という結果であった．コミュニティ → アーティファクトの関係が最も多くみられたことから，コミュニティの進化がアーティファクトの進化に影響を与える傾向があることが明らかになった．

### 5.4 本論文の制約

#### 5.4.1 各パラメータの値と相関関係

表 5 に示す提案手法のみで抽出することのできた結果のパラメータの値に着目すると，加算係数の値が 0 や 1 以外に 10 や 11 が多くみられる．加算係数の値が 11 であるということは，12 カ月間の説明変数の平均値が目的変数に影響を与えるということを指している．このような結果になったのは，Eclipse プロジェクト特有のリリース間隔が影響を与えていると考えられる．Eclipse プロジェクトでは，2004 年より，毎年 6 月に定期的にリリースを行っている．このように，1 年ごとに定期的にリリースされることによって，各パラメータの値に 11 が多くなったと考えられる．

#### 5.4.2 各パラメータの値域に関する妥当性

本研究では，加算係数および遅延係数の各パラメータが取りうる値の上限を，それぞれ 11 および 12 としてケーススタディを行った．各パラメータの値の上限を 12 とした理由は，1 カ月という時間間隔では説明することのできなかった関係が，1 年という時間間隔で関係を説明することによって明らかになる，ということがあると考えたためである．実際に，Eclipse プロジェクトでは，1 年間隔でリリースが行われているなど，1 年という期間で分析をすることによって初めて明らかになる関係が存在すると考えられる．しかし，今回用いた上限よりも大きな値を用いる方がより有用な知見を導けた可能性もあり，パラメータの適切な上限値を決定する方法を考案することを今後の課題としたい．

## 6. おわりに

本論文では，OSS システムとコミュニティの共進化のプロセスを定量的に分析することを目的とした相関分析手法を提案した．提案手法の有用性を確かめるために，Eclipse Platform プロジェクトを対象とするケーススタディを行った．4 種類のデータソースから計測された 131 種類のメトリクスを用いて，合計 17,030 件の組合せに対して提案手法を適用した結果，従来の相関分析では抽出することができなかった関係を 31 件抽出することができた．また，提案手法により抽出した関係を追加分析することで，メトリクス間の因果関係をより詳細に解明するのに役立つことを示した．本研究の今後の課題は，得られた 31 件の相関関係をさらに分析することや，他の OSS プロジェクトに対しても提案手法を適用し，OSS システムとコミュニティの共進化において共通する重要な関係を導出して行くことなどがあげられる．

**謝辞** 本研究の一部は，文部科学省科学研究補助金（基

盤（C）：24500041）および（若手（B）：25730045）による助成を受けた．また，独立行政法人情報処理推進機構が実施した「2013 年度ソフトウェア工学分野の先導的研究支援事業」の支援を受けた．

## 参考文献


[1] Bird, C., Gourley, A., Devanbu, P., Swaminathan, A. and Hsu, G.: Open Borders? Immigration in Open Source Projects, *Proc. 4th International Workshop on Mining Software Repositories* (*MSR'07*), No.6 (2007).

[2] Bird, C., Pattison, D., D'Souza, R., Filkov, V. and Devanbu, P.: Latent Social Structure in Open Source Projects, *Proc. 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*FSE'08*), pp.24–35 (2008).

[3] Brooks, F.P.: *The Mythical Man-Month: Essays on Software Engineering*, Addison Wesley, MA (1975).

[4] DiBona, C., Stone, M. and Cooper, D.: *Open Sources 2.0–The Continuing Evolution*, O'Reilly Media, Sebastopol, CA (2005).

[5] Godfrey, M.W. and Tu, Q.: Evolution in Open Source Software: A Case Study, *Proc. International Conference on Software Maintenance* (*ICSM'00*), pp.131–142 (2000).

[6] Ishida, K. and Kitagawa, H.: Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams, *Proc. 19th International Conference on Database and Expert Systems Applications* (*DEXA 2008*), pp.255–268 (2008).

[7] Jensen, C. and Scacchi, W.: Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study, *Proc. 29th International Conference on Software Engineering* (*ICSE'07*), pp.364–374 (2007).

[8] Kamei, Y., Matsumoto, S., Maeshima, H., Onishi, Y., Ohira, M. and Matsumoto, K.: Analysis of Coordination between Developers and Users in the Apache Community, *The 4th International Conference on Open Source Systems* (*OSS2008*), pp.81–92 (2008).

[9] Karus, S. and Gall, H.: A Study of Language Usage Evolution in Open Source Software, *Proc. 8th Working Conference on Mining Software Repositories* (*MSR'11*), pp.13–22 (2011).

[10] Kumar, A. and Gupta, A.: Evolution of Developer Social Network and Its Impact on Bug Fixing Process, *Proc. 6th India Software Engineering Conference* (*ISEC'13*), pp.63–72 (2013).

[11] Lavazza, L., Morasca, S., Taibi, D. and Tosi, D.: Predicting OSS trustworthiness on the basis of elementary code assessment, *Proc. 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (*ESEM'10*), No.36 (2010).

[12] Mockus, A., Fielding, R.T. and Herbsleb, J.D.: Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Engineering and Methodology*, Vol.11, No.3, pp.309–346 (2002).

[13] Penta, M.D., German, D.M., Gueheneuc, Y.-G. and Antoniol, G.: An Exploratory Study of the Evolution of Software Licensing, *Proc. 32nd International Conference on Software Engineering* (*ICSE'10*) (2010).

[14] Phannachitta, P., Ihara, A., Jirapiwong, P., Ohira, M. and Matsumoto, K.: An Algorithm for Gradual Patch Acceptance Detection in Open Source Software Repository Mining, *IEICE Trans. Information and Systems*, Vol.E95-A, No.9, pp.1478–1489 (2012).

[15] Raymond, E.S.: *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media, Sebastopol, CA (1999).

[16] Sato, S., Monden, A. and Matsumoto, K.: Evaluating the applicability of reliability prediction models between different software, *Proc. International Workshop on Principles of Software Evolution* (*IWPSE'02*), pp.97–102 (2002).

[17] Scacchi, W.: Understanding Open Source Software Evolution, *Software Evolution and Feedback*, Madhavji, N.H., Fernandez-Ramil, J.C. and Perry, D.E. (Eds.), John Wiley & Sons, Ltd., chapter 9, pp.181–205 (2006).

[18] Shihab, E., Mockus, A., Kamei, Y., Adams, B. and Hassan, A.E.: High-impact Defects: A Study of Breakage and Surprise Defects, *Proc. 13th European Software Engineering Conference and the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering* (*ESEC/FSE '11*), pp.300–310 (2011).

[19] Shumway, R.H. and Stoffer, D.S.: *Time Series Analysis and Its Applications With R Examples*, Springer, 3rd edition (2010).

[20] Sinha, V.S., Mani, S. and Sinha, S.: Entering the Circle of Trust: Developer Initiation as Committers in Open-Source Project, *Proc. 8th Working Conference on Mining Software Repositories* (*MSR'11*), pp.133–142 (2011).

[21] Thomas, S.W., Adams, B., Hassan, A.E. and Blostein, D.: Modeling the Evolution of Topics in Source Code Histories, *Proc. 8th Working Conference on Mining Software Repositories* (*MSR'11*), pp.173–182 (2011).

[22] Weiss, C., Premraj, R., Zimmermann, T. and Zeller, A.: How Long Will It Take to Fix This Bug?, *Proc. 4th International Workshop on Mining Software Repositories* (*MSR '07*), No.1 (2007).

[23] Ye, Y. and Kishida, K.: Toward an understanding of the motivation Open Source Software developers, *Proc. 25th International Conference on Software Engineering* (*ICSE'03*), pp.419–429 (2003).

[24] Ye, Y., Nakakoji, K., Yamamoto, Y. and Kishida, K.: The Co-Evolution of Systems and Communities in Free and Open Source Software Development, *Free/Open Source Software Development*, Koch, S. (Ed.), Idea Group Publishing, Hershey, PA., chapter 3, pp.59–82 (2004).

[25] Zhou, M. and Mockus, A.: Developer fluency: Achieving true mastery in software projects, *Proc. 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*FSE'10*), pp.137–146 (2010).

[26] Zhou, M. and Mockus, A.: What Make Long Term Contributors: Willingness and Opportunity in OSS Community, *Proc. 34th International Conference on Software Engineering* (*ICSE '12*), pp.518–528 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2337223.2337284⟩ (2012).

[27] 古山恒夫：プロジェクトデータ分析の指針と分析事例，*SEC Journal*, Vol.1, No.3, pp.6–13 (2005).

[28] 黛　勇気，竹内裕之，児玉直樹：生活習慣と健康状態の時系列データ解析における重み付けの検討 (I)—日毎の任意係数による重みづけ，*Proc. 3rd Forum on Data Engineering and Information Management* (*DEIM'11*), D7-5 (2011).

[29] 竹内裕之，児玉直樹：生活習慣と健康状態に関する時系列データ解析手法の開発，*Proc. 3rd Forum on Data Engineering and Information Management* (*DEIM'08*), E1-

5 (2008).
[30] 山谷陽亮，大平雅雄，Phannachitta, P.，伊原彰紀：OSSシステムとコミュニティの共進化の理解を目的としたデータマイニング手法，情報処理学会マルチメディア，分散，協調とモバイル（DICOMO）シンポジウム論文集，情報処理学会，情報処理学会，pp.1695–1703 (2013).

## 推薦文

本研究発表では，オープンソースソフトウェア（OSS）システムとコミュニティの共進化のプロセスを定量的に分析するためのデータマイニング手法を提案している．提案手法は，遅延相関分析の考え方に基づいて，一方の進化の系列が他方の進化の系列に与える影響を一定時間後に観察できることを考慮したものである．また，遅延相関分析を容易に行うために，遅延相関係数が最も高くなる際の各種パラメータを自動的に求める点に特徴がある．Apache および Eclipse コミュニティを対象としたケーススタディの結果，遅延相関を考慮しない従来の分析手法に比べ，提案手法が共進化のプロセスをより正確に観察できることを示している．OSS を活用したシステム開発が一般的になりつつある昨今，「サポートが得られるかどうか分からない」などの理由から，依然として OSS の活用に躊躇するシステム開発企業は少なくないため，提案手法を用いた分析により OSS システムとコミュニティの共進化が「見える化」されることは，OSS を用いたシステム開発に大きな貢献となる．

（グループウェアとネットワークサービス研究会主査　市村　哲）

**山谷 陽亮**　（学生会員）

2014 年和歌山大学システム工学部情報通信システム学科卒業．現在，同大学大学院システム工学研究科博士前期課程在学中．学士（工学）．オープンソースソフトウェア工学，リポジトリマイニングに興味を持つ．

**大平 雅雄**　（正会員）

1998 年京都工芸繊維大学工芸学部電子情報工学科卒業，2003 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同大学情報科学研究科助教を経て，2012 年和歌山大学システム工学部准教授．博士（工学）．ソフトウェア工学，特にリポジトリマイニング，ソフトウェア開発における知的協調作業支援の研究に従事．電子情報通信学会，ヒューマンインタフェース学会，ACM 各会員．

**パサコーン パンナチッタ**

2011 年 Kasetsart 大学卒業．2013 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学博士後期課程在籍．修士（工学）．データマイニング技術，高性能計算処理技術を用いたエンピリカルソフトウェア工学の研究に従事．

**伊原 彰紀**　（正会員）

2007 年龍谷大学理工学部卒業．2009 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．2012 年同大学博士課程修了．同年同大学情報科学研究科助教．博士（工学）．ソフトウェア工学，特にオープンソースソフトウェア開発・利用支援の研究に従事．ソフトウェア科学会，電子情報通信学会，IEEE 各会員．