Vol.2014-SE-185 No.8
2014/7/10

# ソフトウェア進化の理解を目的とした遅延相関分析手法：OSSプロジェクトデータへの適用

山谷 陽亮[1,a)]　大平 雅雄[2,b)]

**概要**：大規模化かつ複雑化した近年のソフトウェア開発では，ソフトウェアに加えた変更がシステム全体あるいはシステムの個々の機能の信頼性や保守性にどのような影響を及ぼすのかを正確に把握することは非常に困難である．そのため，ソフトウェア進化の理解を支援することを目的とした研究が近年盛んに行われているものの，実務者がソフトウェア進化を自身で分析するための手法やツールの開発は未だ十分ではない．特に既存の手法やツールは，時間的に遅延して観察される 2 変数間の関係 (例えば，実装工程でのコード行数の増加と出荷後に検出される欠陥数との関係など）を考慮していないため，実務者のソフトウェア進化に対する理解を十分に支援することはできないことが問題であった．既存手法の問題点を解決するために，山谷は [12]，時間的遅延を考慮してソフトウェア進化を理解することを支援する分析手法を提案している．文献 [12] では，提案手法を Eclipse Platform に適用した結果，不具合修正時間が長期化すると不具合の優先度を決定し，先に取り組むべき不具合を決定している傾向があることを明らかにしている．本稿では，3 つのオープンソースプロジェクト（Eclipse Platform, Apache HTTP Server, Gimp）に提案手法を適用し，すべてのプロジェクトに共通してみられる相関関係および個々のプロジェクト内でみられる相関関係の特徴を分析する．また，時間的遅延を考慮しない従来の相関分析との結果と比較することで，提案手法の有用性を明らかにする．

## 1. はじめに

ソフトウェア進化 [2], [5], [9] は，多様化するユーザのニーズやソフトウェアシステムを取り巻く環境の変化に対応するためにソフトウェアに対しておこなわれる様々な変更を指す．近年のソフトウェアシステムは，我々の社会活動を支える上で必要不可欠な存在となっているため，適宜ソフトウェアに変更を加え，機能追加したり品質を改善しながら長期間保守し続ける必要がある [4]．そのため，ソフトウェアに対して行われる変更（すなわち，ソフトウェア進化）がソフトウェアの品質やユーザの満足度に与える影響を慎重に見極める必要がある．

しかしながら，大規模化かつ複雑化した近年のソフトウェア開発では，ソフトウェアに加えた変更がシステム全体あるいはシステムの個々の機能の信頼性や保守性にどのような影響を及ぼすのかを正確に把握することは非常に困難である．そのため，ソフトウェア進化の理解を支援することを目的とした研究が近年盛んに行われている．例えば Zimmermann らは，ソースコードの変更履歴データに対してアソシエーションルールマイニングを行うことによって，開発者がファイルを変更した際に，同時に変更されやすいファイルを推薦するツールを開発した [8]．このように，ソフトウェア進化の理解は，（管理者・開発者）がソフトウェアの開発・保守を行う際の有益な知見となり得る．

ソフトウェア進化に関する研究は盛んにおこなわれているものの，実務者がソフトウェア進化を自身で分析するための手法やツールの開発は未だ十分ではない．例えば，ソフトウェアに対しておこなわれた変更がソフトウェアの品質にどのような影響を与えるかを調べるためには，実装工程から時間的に遅延する後の工程（例えば，試験工程や製品の出荷後）において検出される欠陥の数や欠陥密度を測定し，変更と品質との関係を調べる必要がある．既存の相関分析では，時間的遅延をせずに，あるいは，特定の期間のデータに対して特定の遅延時間を設定した後に，変更と品質との相関関係を調べることしかできないため，長期間保守され続けるようなソフトウェアシステムを対象とした分析手法としては適切ではない．

また，長期間保守され続けるようなソフトウェアシステムでは，分析をおこなう実務者がこれまでの開発過程のす

[1] 和歌山大学 システム工学研究科
和歌山県和歌山市栄谷 930 番地
[2] 和歌山大学 システム工学部
和歌山県和歌山市栄谷 930 番地
[a)] s151049@center.wakayama-u.ac.jp
[b)] masao@center.wakayama-u.ac.jp

べてを熟知しているとは考えにくく，開発コンテキストを十分には共有してない実務者が調査したい対象や計測すべきメトリクスをあらかじめ絞り込むことは非常に困難であると想定される．そのような状況下では，GQM 法のようなトップダウンな計測アプローチは適用しにくいため，(ある程度) 網羅的にメトリクスを計測し散布図行列や平行座標プロットのようなボトムアップな分析ツールを用いて，全体の傾向や着目すべき特徴を調べるという方法を用いる必要がある．ただし，ボトムアップな分析手法・ツールも時間的な遅延を考慮していないため，ソフトウェア進化を分析するには不十分であるといえる．

既存手法の問題点を解決するために，山谷は [12]，時間的遅延を考慮してソフトウェア進化を理解することを支援する分析手法を提案している．文献 [12] では，提案手法を Eclipse Platform に適用した結果，不具合修正時間が長期化すると不具合の優先度を決定し，先に取り組むべき不具合を決定している傾向があることを明らかにしている．本稿では，3 つのオープンソースプロジェクト (Eclipse Platform, Apache HTTP Server, Gimp) に提案手法を適用し，すべてのプロジェクトに共通してみられる相関関係および個々のプロジェクト内でみられる相関関係の特徴を分析する．また，時間的遅延を考慮しない従来の相関分析との結果と比較することで，提案手法の有用性を明らかにする．

続く 2 章では，上記の問題を解決するためのソフトウェア進化を理解するための手法を提案する．そして 3 章では，提案手法の有用性を確かめるために Eclipse Platform, Apache HTTP Server, Gimp の 3 つのプロジェクトを対象として，ケーススタディを行う．4 章で，ケーススタディの結果について説明し，5 章で考察を行う．6 章で関連研究について述べ，7 章で本論文のまとめと今後の課題について述べる．

## 2. アプローチ

本章では，本研究のアプローチを述べる．まず，1 章で述べた問題点を解決するための要件，すなわち，ソフトウェア進化の理解支援を目的とする分析手法に求められる要件について述べ，山谷 [12] が提案する遅延相関分析手法の詳細について説明する．

### 2.1 分析手法に求められる要件

長期間に渡り開発・保守され続ける大規模ソフトウェアシステムの進化を分析するためには，以下の要件を満たす必要がある．

- 時間的な遅延を伴うメトリクス間（変数間）の関係を明らかにできること
- 開発コンテキストを十分に共有していない実務者でも分析が容易に行えること

1 つ目は，従来の相関分析手法やツールが考慮しないメトリクス間の時間的遅延に対応するためのものである．ソフトウェアプロダクトあるいはプロセスに与えた変更がある程度の時間的遅延を伴って，ソフトウェアプロダクトあるいはプロセスに影響を与えること（相関がある）を分析できるようにする．時刻 $t$ に計測される各種メトリクス $m_i(i = 1, 2, 3..., j)$ と，$k$ か月後に計測される各種メトリクス $m_n(n = 1, 2, 3..., m)$ とに関係があることを調べるために，$k$ の値を 1 ヶ月から 12 か月まで自動的に変化させて相関分析が行える（本研究における遅延相関分析）ようにする．

2 つ目は，長期間に渡るソフトウェアシステムの開発・保守過程のすべてを熟知していない実務者に対応するためのものである．近年では，ソフトウェア開発過程の多くがリポジトリに記録されているため，リポジトリから計測可能なメトリクスをできるだけ多く取得し，網羅的にメトリクス間の関係を調べることを可能にする．また，遅延相関分析の結果を相関係数の高いものから順に提示することにより，着目すべき関係を発見するのを手助けする．

次では，山谷が提案する時間的順序関係を考慮した相関分析手法 [12] について述べる．

### 2.2 時間的順序関係を考慮した相関分析手法

探索的アプローチにより，メトリクス間の関係を明らかにするために，提案手法は，さまざまな時系列データで表現されるメトリクスを入力すると，出力として相関がみられるメトリクスのペアのみを抽出する．また，提案手法は

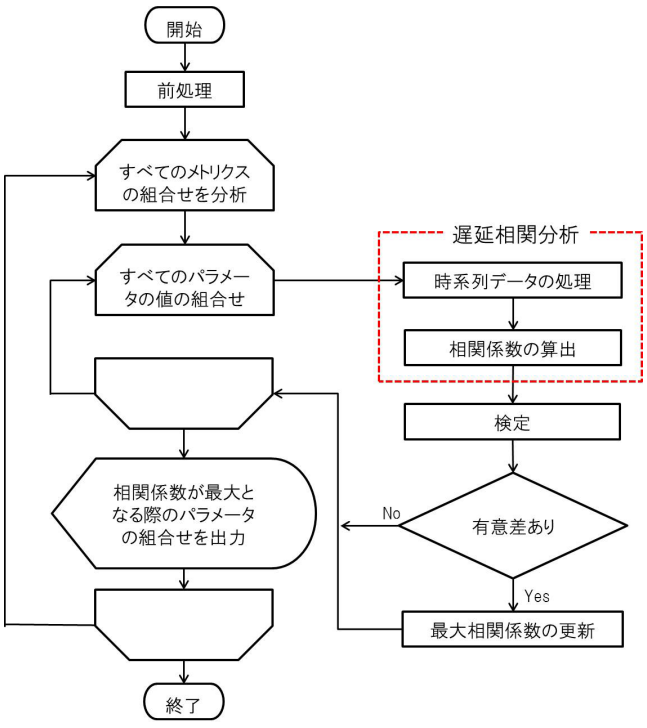


図 1 提案手法のフローチャート

竹内らの時系列データ解析手法 [Takeuchi] を参考にし，時間的順序関係を考慮した相関分析（以降，遅延相関分析）を行う．遅延相関分析を行うことにより，説明変数となるメトリクスの一定期間の変化量と，目的変数となるメトリクスの一定期間の変化量が，時間的な遅延を伴って関係するかどうかを確かめることができる．

提案手法のフローチャートを図に示す．図に示すように，提案手法は遅延相関分析における各種パラメータ（説明変数の加算係数，目的変数の加算係数，遅延係数）のすべての組み合わせに対して，遅延相関分析を行う．そのため，遅延相関係数（遅延相関分析によって求めた相関係数）が最大となる各種パラメータの値の組み合わせを自動的に求めることができる．また，すべてのメトリクスの組み合わせについて遅延相関分析を行うため，膨大なメトリクスの組み合わせの中から，時間的順序関係がみられるメトリクスのペアのみを抽出することができる．したがって，分析者は遅延の大きさや分析窓の大きさを気にすることなく，最も遅延相関係数が大きくなるメトリクスの組み合わ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．

#### 2.2.1 前処理

提案手法で入力となるメトリクスは，ソフトウェア開発データから計測されるデータである．一般的に，ソフトウェア開発データは外れ値を含む場合が多いことが知られている．外れ値の存在により遅延相関分析の結果に多大なる影響を与えてしまうことが予想されるため，外れ値を除去する必要がある．時系列データに対する外れ値検出では，2 次元空間内の距離に基づく外れ値検出が必要となる [3]．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を行い，除去する．

また，提案手法で入力となるデータは時系列データであるため，外れ値を除去すると欠損値が発生する．欠損値を含む時系列データに対して相関分析を行うことはできないため，欠損値を適切に補完する必要がある．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欠損値補完の方法であるスプライン補間を用いることとした．

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．本研究では，すべてのデータを 0 から 1 の間の値で表現する．正規化を行う式を式 (1) に示す

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

$x$ 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．

表 1　パラメータ一覧

| パラメータ名 | 定義式 |
|---|---|
| 説明変数の加算係数 | $i-j$ |
| 目的変数の加算係数 | $m-n$ |
| 遅延係数 | $n-i$ |

#### 2.2.2 遅延相関分析

図 2.2 に示すように，前処理を行った後，時間的順序関係を持つメトリクスの組み合わせを抽出するために，すべてのメトリクスの組み合わせに対して遅延相関分析を行う．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 [11] を参考に，一定期間の説明変数の変化量が一定期間後に一定期間の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）．

遅延相関分析では，説明変数の値の変化が続く期間を説明変数の加算係数（$i-j$），目的変数の値の変化が続く期間を目的変数の加算係数（$m-n$），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（例えば，1 から 12 ヶ月）に対して設定する．提案手法は，すべてのメトリクスの組み合わせに対して，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．例えば，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 ヶ月の期間として設定した場合，$N(N-1) \times 12^3$ 種類の遅延相関係数が算出される．ただし，メトリクスの組み合わせとパラメータの組み合わせから，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るために行う時系列データの処理と，遅延相関係数の求め方について述べる．

**時系列データの処理**

遅延相関分析では，説明変数と目的変数の一定期間での変化量の相関を求める．一定期間での説明変数および目的変数の変化量は，それぞれの加算係数の値に従い，説明変数および目的変数の値の累積値を求めることによって算出

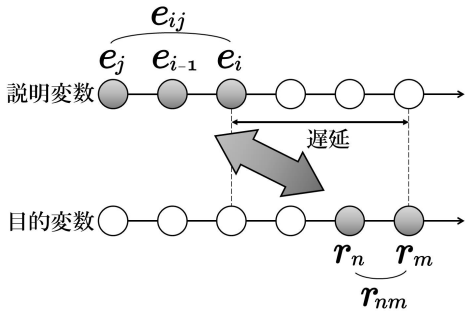


図 2　遅延相関分析の概念図（[10] を参考に改編）

表 2　対象とするデータ

| プロジェクト名 | 言語 | 期間 | ファイル数 | コードの変更履歴 | 不具合修正履歴 | コメント数 |
|---|---|---|---|---|---|---|
| Platform | Java | 2003/7–2012/6 | 940 | 5,240 件 | 74,202 件 | 401,609 |
| Apache | C | 2002/6–2011/5 | 258 | 12,556 件 | 6,174 件 | 23,066 |
| Gimp | C | 2001/1–2011/12 | 1596 | 25,515 件 | 10,609 件 | 54,245 |

する．例えば図 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，説明変数の変化量を $e_{ij}$ とすると，$e_{ij}$ は式 (2) で表される．

$$e_{ij} = e_i + e_{i-1} + \cdots + e_j \quad (2)$$

同様に，時刻 $n$ における目的変数の変化量を $r_{nm}$ とすると，$r_{nm}$ は式（3）で表される．また，$r_n, r_m$ は時刻 $n$ および時刻 $m$ の目的変数の値である．

$$r_{nm} = r_n + r_{n+1} + \cdots + r_m \quad (3)$$

**遅延相関係数の算出**

説明変数および目的変数の変化量の一定期間後の関係を明らかにするために，遅延相関係数を求める．本研究では，目的変数の変化量のペアを遅延係数の値 $n-i$ に基づいてシフトさせることで遅延相関係数を求める．例えば，図 2 に示す通り，時刻 $i$ における説明変数の変化量 $e_{ij}$ 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 $r_{nm}$ でペアを作り遅延相関係数を算出する．このように目的変数の変化量をシフトさせることによって，説明変数および目的変数の変化量の一定期間後の関係を明らかにしている．

また，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，ノンパラメトリックなデータの相関関係を明らかにすることのできるスピアマンの順位相関を用いて遅延相関係数を求める．

#### 2.2.3　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み合わせを求めるために，最大相関係数の更新を行う．

**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．

**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の組み合わせ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ことができる．

**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み合わせ（すべての説明変数と目的変数の組み合わせ）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（$n-i$）してどれくらいの期間（$m-n$）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．

## 3.　ケーススタディ

本章では，提案手法の有用性を確かめるために行うケーススタディについて述べる．

ケーススタディの目的は，提案手法を大規模ソフトウェアプロジェクトに適用し，従来手法（遅延を考慮しない相関分析）の結果と提案手法の結果を比較することにより，提案手法が従来手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ことである．

以降では，ケーススタディを行う際に用いるデータセットおよび，メトリクスについて述べる．また，分析を行う方法についても述べる．

### 3.1　データセット

ケーススタディは，Eclipse Platform プロジェクト [*1]（以降，Platfrom），Apache HTTP Server プロジェクト [*2]（以降，Apache），Gimp プロジェクト [*3] の三つのプロジェクトを対象に行う．Platform, Apache, Gimp プロジェクトを対象にした理由は，長期間開発・保守が行われていること，そして，ソースコードリポジトリや不具合管理リポジトリが公開されており，開発履歴データが入手可能であることである．Platform プロジェクトは 2003 年 7 月から 2012 年 6 月までの 9 年間，Apache プロジェクトは 2002 年 6 月から 2011 年 5 月までの 9 年間，Gimp プロジェクトは 2001 年 1 月から 2011 年 12 月までの 10 年間のデータをデータセットとして用いる．

また，ケーススタディで対象とするデータは，ソース

[*1] Eclipse Plaform: http://projects.eclipse.org/projects/eclipse.platform
[*2] Apache HTTP Server Project : http://httpd.apache.org/
[*3] Gimp: Project http://www.gimp.org/

表 3　メトリクス一覧

| データソース | メトリクス |
|---|---|
| ソースコード | （行数・空白行数・コード行数・コメント行数）の（平均・合計），関数・メソッドの数，（宣言部・実行部）のコード行数，セミコロン数，（全体・宣言部・実行部）のステートメント数，（Cyclomatic・Cyclomatic Modified・Cyclomatic Strict・Essential）複雑度の（平均・最大・合計），コメント率 |
| コード変更履歴 | コミッタ数，新規コミッタ数，コードオーナのコミット率，（全体・新規コミッタ・コードオーナ）の（コミット回数，追加・削除した行数，行数を追加・削除したコミット回数，変更したファイル数，平均コミットコメント文字数） |
| 不具合修正履歴 | （報告・割当・修正）した不具合数，（報告・割当・修正）した開発者数，（割当・修正）時間，修正総時間，優先度が（P1・P2・P3・P4・P5）の不具合数，重要度が（Blocer・Critical・Major・Normal・Minor・Trivial・Enhancement）の不具合数，（CCList 数・説明文の文字数）の平均．（再発・再割当）が発生した（不具合数・回数・割合），（Duplicate・Invalid・Resolved・Verified・Wontfix）になった不具合数 |
| 不具合コメント | コメントした開発者数，新しくコメントした開発者数，コメント数の（平均・合計），平均コメント文字数，ドメイン数，最多（コメントした開発者数・ドメイン）が占める割合，新しくコメントした開発者が占める割合，平均リプライ時間 |

コード，ソースコードの変更履歴，不具合修正履歴，不具合に関するコメントである．表 2 にケーススタディで用いるデータソースについて，プロジェクトごとに示す．ソースコードおよびソースコードの変更履歴は，各プロジェクトの版管理システムである Git リポジトリにアクセスすることによって入手することができる．また，各プロジェクトは，不具合管理システムとして Bugzilla を使用しており，Bugzilla 上では，開発者は不具合を報告するだけではなく，不具合に関する議論なども行っている．そのため本研究では，Bugzilla で行われる不具合に関する議論（コメント）を開発者間のコミュニケーションとして捉え分析する．

### 3.2　用いるメトリクス

ケーススタディでは，対象データソースから計測可能な 100 種類のメトリクスを用いて分析を行う．表 3 にケーススタディで用いるメトリクスを示す．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，計測するメトリクスは 1 か月ごとに集計したものを用いる．

ソースコードからは 28 種類のメトリクスを計測する．ソースコードから計測するメトリクスは，ソースコード解析ツールである Understand[*4] を用いて計測する．Platform プロジェクトでは，Java ファイルを，Apache および Gimp プロジェクトでは C ファイルを解析対象とした．解析対象としたファイル数は表 2 に示している．また，ソースコードの変更履歴から 25 種類，不具合修正履歴から 37 種類，不具合管理システム上でのコメントから 10 種類のメトリクスを計測する．これらのメトリクスは，スクリプトを作成することにより計測される．

### 3.3　分析方法

ケーススタディでは，100 種類のメトリクスを入力とし，提案手法を適用する．提案手法を適用させた結果，3 つの

[*4] Understand Source Code Analysis & Metrics http://ww.scitools.com/

表 4　相関分析および提案手法で抽出することのできた関係

| | $C$（件） | $DC$（件） | $\overline{C} \wedge DC$（件） |
|---|---|---|---|
| Eclipse | 1,012 | 7,749 | 5,447 |
| Apache | 434 | 4,885 | 2,278 |
| Gimp | 660 | 5,679 | 5,447 |
| 共通 | 28 | 783 | 127 |

プロジェクトで共通して捉えることのできた関係を明らかにする．本研究では，相関係数の絶対値が 0.7 以上のメトリクス間の関係を相関関係があると判断する．なお，提案手法における各パラメータがとり得る値の範囲は，説明および目的変数の加算係数（$1 \leq i-j, m-n \leq 12$），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 か月であることから，パラメータの単位は 1 か月となる．例えば，遅延係数が 1 であるときは，メトリクス間の関係に 1 か月の遅延が発生していることを示している．

また，100 種類のメトリクスのすべての組み合わせについて通常の相関分析を行うことで，時間的順序関係を考慮しない通常の相関分析手法と提案手法との比較を行う．具体的には，提案手法を用いて抽出することのできた関係（相関係数の絶対値が 0.7 を超えるメトリクスのペア）のうち，通常の相関分析では相関関係をみつけることのできなかった関係（相関係数の絶対値が 0.7 未満であるメトリクスのペア）を明らかにすることで，提案手法がソフトウェア開発データを分析する上で有用であることを確認する．

さらに，提案手法にのみ相関がみられるメトリクス間の関係について詳細な分析を行う．

## 4.　分析結果

本章では，Platform, Apache, Gimp プロジェクトから計測された 100 種類のメトリクスに対して，提案手法を適用した結果について述べる．表 4 に通常の相関分析手法（$C$）および提案手法 ($DC$) で抽出することのできた関係の件数を示す．表 4 に示すように，遅延相関係数の絶対値が 0.7

表 5　分析結果：報告された不具合数と検証された不具合数の関係

| プロジェクト | 説明変数の加算係数 | 目的変数の加算係数 | 遅延係数 | 遅延相関係数 | 相関係数 |
|---|---|---|---|---|---|
| Eclipse | 7 | 9 | 4 | 0.97 | 0.62 |
| Apache | 12 | 12 | 12 | 0.70 | 0.22 |
| Gimp | 8 | 12 | 5 | 0.82 | 0.21 |

表 6　分析結果：不具合再割当回数と平均空白行数の関係

| プロジェクト | 説明変数の加算係数 | 目的変数の加算係数 | 遅延係数 | 遅延相関係数 | 相関係数 |
|---|---|---|---|---|---|
| Eclipse | 9 | 11 | 1 | -0.75 | -0.21 |
| Apache | 12 | 12 | 6 | -0.87 | -0.07 |
| Gimp | 12 | 12 | 6 | -0.72 | -0.23 |

以上であり，相関関係に有意差がみられたメトリクスの組み合わせが，Platform プロジェクトでは 7,749 件，Apache プロジェクトでは 4,885 件，Gimp プロジェクトでは 5,679 件抽出された．また，このうち 3 つのプロジェクトで共通して相関がみられた関係は 783 件存在した．

本研究では，この 3 つのプロジェクトで共通して相関関係がみられたメトリクスの組み合わせのうち，通常の相関分析では明らかにすることのできなかった関係（$\overline{C} \wedge DC$）に着目する．3 つのプロジェクトで共通して相関がみられた関係のうち，通常の相関分析で明らかにすることのできなかった関係は 127 件存在した．以降では，提案手法にのみ相関がみられた関係である，「報告された不具合数」と「検証された不具合数」の関係，および，「不具合再割当回数」と「平均空白行数」の関係について説明する．

ケーススタディの結果，「報告された不具合数」と「検証された不具合数」に時間的順序関係がみられた．表 5 に相関係数が最大となった際の各パラメータの値および最大相関係数，通常の相関分析を行った際の相関係数をプロジェクトごとに示す．表 5 に示すように，報告された不具合数が増加すると，Eclipse プロジェクトでは 4 か月後に，Apache プロジェクトでは 12 か月後に，Gimp プロジェクトでは 5 か月後に検証された不具合数が増加する傾向にあることがわかる．このように，通常の相関分析では相関関係があると判断することのできなかった関係が，時間的順序関係を考慮することによって，相関関係を明らかにすることができている．

ケーススタディの結果，「不具合再割当回数」と「平均空白行数」に時間的順序関係がみられた．「不具合再割当回数」とは，不具合修正プロセスにおいて不具合修正の担当者が変更される回数を指しており，「平均空白行数」は，1 ファイルに含まれる空白行数を意味している．表 6 に相関係数が最大となった際の各パラメータの値および最大相関係数，通常の相関分析を行った際の相関係数をプロジェクトごとに示す．表 6 より，不具合再割当回数が増加すると，Eclipse プロジェクトでは 1 か月後に，Apache, Gimp プロジェクトでは 6 か月後に平均空白行数が減少する傾向があることがわかる．

表 7　不具合が報告されてから検証されるまでの時間

| プロジェクト | 中央値（日） | 平均値（日） |
|---|---|---|
| Eclipse | 29.60 | 111.37 |
| Apache | 77.41 | 381.68 |
| Gimp | 2732.58 | 2515.62 |

## 5. 考察

### 5.1 時間的順序関係がみられた背景

**「報告された不具合数」と「検証された不具合数」の関係**

「報告された不具合数」と「検証された不具合数」に時間的順序関係がみられた理由を説明する．一般的に OSS 開発では，開発者やエンドユーザーは不具合を発見すると，Bugzilla などの不具合管理システムに不具合の内容などを記述し報告を行う．不具合が報告されると，開発者は不具合の内容を確認し，修正パッチなどを作成することなどによって，不具合を修正する．そして，修正された不具合は，修正を行った開発者以外の開発者によって，修正内容が正しいのかどうか検証される．このように，報告された不具合は，修正，検証というステップを踏み，解決する．表 7 に不具合が報告されてから検証が行われるまでの時間の中央値および平均値をプロジェクトごとに示す．表 7 に示すように，Platform, Apache プロジェクトでは不具合が報告されてから検証されるまでの平均時間がそれぞれ 111.37 日（約 4 か月），381.68 日（約 12 か月）となっている．この結果と提案手法によって算出した遅延の大きさである遅延係数の値を比較することにより，提案手法によって正しい時間間隔でメトリクス間の時間的順序関係を抽出すること

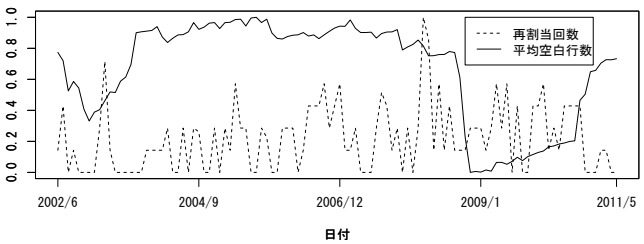


図 3　再割当回数と平均空白行数の経時的変化

ができていることがわかる．

「不具合再割当回数」と「平均空白行数」の関係

「不具合再割当回数」と「平均空白行数」に時間的順序関係がみられた理由を Apache プロジェクトを例に説明する．図 3 に前処理を行った後の不具合再割当回数と平均空白行数の経時的変化を示す．図 3 から，2008 年 12 月あたりに平均空白行数が著しく減少していることがわかる．平均空白行数が著しく減少した理由は，2008 年 12 月から PCRE ライブラリを Apache 内に保持し，配布することをやめたためである．PCRE（Perl Compatible Regular Expressions）ライブラリは，Perl 互換の正規表現を扱うために用いられるが，PCRE が Apache に同梱されているライブラリと実行時に衝突する問題点が明らかとなった．そのため，2008 年 12 月に Apache 上から PCRE ライブラリが削除されることとなった．PCRE ライブラリには，*pcre.c* という空白行数を 1,267 行もつ C ファイルが含まれており（2008 年 12 月の平均空白行数は 88.59），そのような著しく空白行数の多いファイルが削除されることにより，平均空白行数が減少した．また，平均空白行数が減少する 6 か月ほど前に，不具合再割当回数が多かったことから，不具合再割当回数と平均空白行数に時間的順序関係がみられたと結論づけられる．

### 5.2 提案手法の適用範囲

本研究では，3 つのプロジェクトの 9 年間以上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．ケーススタディの結果に基づき分析を行うことによって，前節で示したように，再割当回数と平均空白行数の関係から，同梱ライブラリが削除されるなどの突発的なイベントを明らかにすることができた．相関分析では，このような突発的なイベントにより，結果が影響されやすいが，このようなイベントを検知し，そのイベントがプロダクトやプロセスにどのような影響を与えてきたかを理解することは，ソフトウェアの変更を理解する上では有用である．

ソフトウェアの変更を当事者以外の人間が理解するためには，その当時の開発コンテキストを理解する必要がある．当事者以外の人間とは，研究者のような外部の人間であることもあれば，プロジェクトを引き継いだ現在の管理者や責任者であることもあり，ソフトウェアの変更の理解を必要としているのはむしろ当事者以外の人間である．しかし，当事者以外の人間にとって，プロジェクトで過去に起こったすべてのイベントを把握することは難しく，また，それらのイベントがプロダクトやプロセスにどのような影響を与えてきたかを具体的に知ることはさらに難しい．したがって，提案手法のように，当時の開発コンテキストを理解することを助けることのできる手法は当事者以外の人間にとって，有用なものになると考えられる．そのため，定期的にプロジェクトの管理者が変更されると予想される長期的に開発・保守が行われているプロジェクトの管理者や，そのようなプロジェクトを分析しようとしている研究者に提案手法を利用してもらうことを期待している．

## 6. 関連研究

本章では，本研究の位置づけを明らかにするために，ソフトウェア開発データからさまざまな関係を抽出するために行われている研究について紹介する．

Crowston らは，OSS プロジェクトの成功例から，分散型チーム開発がうまく機能する要因を明らかにするために，122 プロジェクトを対象に，不具合修正時間やコミュニティのアクティビティレベル，ソフトウェアのダウンロード数，開発チームの規模などのメトリクスを用いて，各メトリクス間の相関を分析している [1]．Crowston らの研究では，ある特定の時期のメトリクスの値しか計測していない．本研究では，ソフトウェアの変更をより正確に理解するためには，時間的概念は無視できないものであると考えているため，時系列データを分析している．

また，ソフトウェア開発データに対して，アソシエーションルールマイニングを適用する研究も行われている [7][8]．Ying らは，Eclipse および Mozilla プロジェクトのソースコードの変更履歴に対してアソシエーションルールマイニングを適用することによって，同時に変更されやすいファイルを分析した．その結果，同時に変更されやすいファイルを開発者に推薦することは，ソフトウェアの品質を向上することに寄与することが明らかとなった．アソシエーションマイニングは，同時に変更するファイル名などの，定性的なデータが入力となるが，本研究では，メトリクスの値などの定量的なデータを入力として扱うことができる．

さらに，Xie らは，開発者が API の使用状況を理解し，効率的にコードを書くことを支援するために，ライブラリの API の呼び出しパターンを検出する MAPO と呼ばれるツールを開発している [6]．MAPO は，シーケンシャルパターンマイニングの考え方を基にライブラリの API の呼び出しパターンを検出している．シーケンシャルパターンマイニングでは，順序関係を考慮しているものの，関係がどれくらい後にみられるのか，といった事象間の遅延の大きさを特定することはできない．本研究で提案した時間的順序関係を考慮した相関分析手法では，事象間の順序関係だけではなく，遅延の大きさも特定することができる．

## 7. おわりに

本研究では，3 つのオープンソースプロジェクト（Eclipse Platform, Apache HTTP Server, Gimp プロジェクト）から計測される 100 種類のメトリクスを入力として，ソフトウェア進化の理解を目的とした提案手法を適用した．ケー

スタディの結果，報告された不具合数が増加すると検証された不具合数が一定期間後に増加するなどの，3 つのプロジェクトに共通してみられる相関関係をみつけることができた．また，時間的遅延を考慮しない従来の相関分析との結果の比較を行うことで，提案手法の有用性を明らかにした．今後は，提案手法で抽出することのできた関係のうち，本稿で説明していない関係について詳細に分析を行うつもりである．

**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）による助成を受けた．

## 参考文献

[1] Crowston, K., Annabi, H., Howison, J. and Masango, C.: Towards a portfolio of FLOSS project success measures, *Workshop on Open Source Software Engineering, 26th International Conference on Software Engineering*, pp. 29–33 (2004).
[2] Fernandez-Ramil, J. and Perry, D.: *Software Evolution and Feedback: Theory and Practice*, Wiley (2006).
[3] Ishida, K. and Kitagawa, H.: Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams, *Proceedings of 19th International Conference on Database and Expert Systems Applications (DEXA 2008)*, pp. 255–268 (2008).
[4] Lehman, M. M.: Software Engineering, the Software Process and their Support, *Software Engineering Journal*, Vol. 6, pp. 243–258 (1991).
[5] Mens, T. and Demeyer, S.: *Software Evolution*, Springer (2008).
[6] Xie, T. and Pei, J.: MAPO: Mining API Usages from Open Source Repositories, *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pp. 54–57 (2006).
[7] Ying, A. T., Murphy, G. C., Ng, R. and Chu-Carroll, M. C.: Predicting Source Code Changes by Mining Change History, *IEEE Transactions on Software Engineering*, Vol. 30, pp. 574–586 (2004).
[8] Zimmermann, T., Weissgerber, P., Diehl, S. and Zeller, A.: Mining Version Histories to Guide Software Changes, *IEEE Transactions on Software Engineering*, Vol. 31, pp. 429–445 (2005).
[9] 大森隆行，丸山勝久， 林晋平，沢田篤史：ソフトウェア進化研究の分類と動向，コンピュータソフトウェア，Vol. 29, No. 3, pp. 168–173 (2012).
[10] 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-，*Proceedings of the 3th Forum on Data Engineering and Information Management (DEIM'11)*, pp. D7–5 (2011).
[11] 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，*Proceedings of the 3th Forum on Data Engineering and Information Management (DEIM'08)*, pp. E1–5 (2008).
[12] 山谷陽亮：時間的順序関係を考慮したメトリクス間の関係抽出手法，2013 年度卒業論文，和歌山大学システム工学部 (2014).