情報処理学会論文誌 Vol.58 No.4 807–817 (Apr. 2017)

改版履歴の分析に基づく変更支援手法における時間的近接性の考慮と同一作業コミットの統合による影響

森 達也[1,a)] アンダース ハグワード[1,2] 小林 隆志[3,b)]

受付日 2016年8月10日, 採録日 2017年1月10日

概要：改版履歴の分析によってファイルの共変更ルールを抽出し，必要な変更箇所を推薦することで開発者への支援を行う研究が進められている．既存手法による推薦は正確であるが，多くの場合に変更漏れを推薦できないという問題点がある．本研究ではより多くの変更漏れを開発者に推薦するため，改版履歴を分析して共変更ルールを抽出するうえで考慮すべき 2 つの特性に着目した．1 つは時間的近接性である．ソフトウェアの進化にともないソフトウェアの依存関係は変化するため，そこから得られる共変更ルールも変化しうる．全コミットではなく最近のコミットのみを分析の対象とすることで，共変更ルールの質の向上が期待できる．もう 1 つは同一作業コミットの統合である．同一作業に関するコミットを統合することで，コミットの粒度が統一され，共変更ルールの質向上につながると考える．我々は，この 2 つの特性が共変更ルールの質にどのような影響を及ぼすかを調査した．代表的な OSS を用いた評価実験により，共変更ルールが時間とともに変化すること，および，時間的近接性の考慮によってより多くの変更漏れを推薦できることを明らかにした．特に Eclipse においては，*Recall* が最大で 0.11 から 0.28 まで上昇した．また，同一作業コミットを統合することが，推薦性能の向上に有益であることを明らかにした．

キーワード：変更支援，ソフトウェアリポジトリマイニング，改版履歴，ソフトウェア保守

An Empirical Study of the Effects of Recency-aware History Analysis and Commits Aggregation on Change Guide

Tatsuya Mori[1,a)] Anders Hagward[1,2] Takashi Kobayashi[3,b)]

Received: August 10, 2016, Accepted: January 10, 2017

Abstract: Many studies on change guide, which suggest necessary code changes with co-change rules extracted from a change history, have been performed so far. Recommendations by existing approaches are adequately accurate, however, those approaches often fail to detect candidates of overlooked changes. In this study, we focus on two characteristics to recommend more overlooked changes. One is *recency*. Some of software dependencies used for change guide become obsolete along with long-term evolution. We use only recent commits for extracting co-change rules to avoid incorrect suggestions stemming from such obsolete dependencies. The other is *commits aggregation*. The granularity of commits depends on the nature of developers and projects. We aggregate commits for the same task to capture actual co-change relations, which expects to improve the quality of co-change rules. We investigate the effects of the two characteristics on the quality of co-change rules. Empirical results using typical OSS show that co-change rules vary over time and we can detect more overlooked changes by focusing on recency. Particularly, in the Eclipse case, the maximum *Recall* improved up to 0.28 by recency-aware history analysis whereas the one of the baseline is 0.11. The results also show that commits aggregation for the same task can improve the recommendation performance.

Keywords: change guide, mining software repository, commit history, software maintenance

1. はじめに

ソフトウェアの大規模化にともない，保守工程の労力が増大している．不具合修正や機能追加の際には，各種の依存関係により，変更の影響がソフトウェアの各所へ波及するため，漏れなく変更が必要な箇所を特定することは困難である．これに対し，プログラムを静的解析することで依存関係を検出し，変更の波及範囲を特定する方法が提案されてきた [1]．しかし，検出される依存関係は非常に膨大であり，また，すべての依存関係が変更の伝播を引き起こすわけではない [2]．加えて，ソースファイルとソースファイル以外のファイル（設定ファイル・外部ライブラリなど）の間の依存関係を検出できないという問題がある [3]．

静的解析による変更波及解析・変更支援手法の限界に対処するために，Git や Subversion などのバージョン管理システムの改版履歴を分析の対象とする研究が行われてきた．改版履歴に対してデータマイニングの手法を適用することで，Logical Coupling [4] という静的解析では検出できない隠れた依存関係を検出することができる．改版履歴に相関ルールマイニングを適用することで，頻繁に同時変更されているファイル集合を発見し，あるファイルが変更された際に高い確率で別のファイルも同時変更されることを示す**共変更ルール**という形で Logical Coupling を抽出することができる．本論文における共変更とは，バグ修正に限定したものではなく，機能追加やリファクタリングなどを含むあらゆる目的の同時変更を対象とする．

Zimmermann らはこの共変更ルールをフィールドやメソッドといったクラス要素単位で抽出し，開発者に変更推薦を行うツール eROSE [5] を提案した．eROSE では “同時に変更しなければならなかった” という must の関係だけでなく，“同時に変更した方が適切だった” や “たまたま同時に変更した” という should/may のような関係も対象にして開発者に対する推薦を行う．Zimmermann らは Navigation と Prevention という 2 つのシチュエーションを想定して開発者に対する変更推薦を行った．Navigation は開発者が IDE 上で開発を行っている際に現在変更しているファイルから次に変更すべきファイルをリアルタイムに推薦するものであり，Prevention は開発者がバージョン管理システムに対してコミットを行った際に変更し忘れている可能性の高いファイルを推薦するというものである．本研究では Prevention のシチュエーションを研究対象としており，以後本論文では，must の関係だけでなく should/may の関係も含めて Prevention のシチュエーションで推薦の対象となるファイルを “変更漏れ” であると定義する．Zimmermann らの評価実験の結果から eROSE による推薦は非常に正確であり，共変更ルールを変更支援に利用することの有効性が示されている．しかし，eROSE による推薦には，推薦された結果の精度は非常に高いが多くの場合に変更漏れに対して推薦機能が実行されないという問題点が存在する．

我々は，変更支援においてはより多くの変更漏れを開発者に推薦することが重要であると考える．共変更ルールに基づく変更推薦において推薦できる変更漏れが少ないという問題を解決するために，本研究では共変更ルールの質を向上させるべく改版履歴の 2 つの特徴 [6] に着目する．1 つは時間的近接性であり，もう 1 つは同一作業コミットの統合である．

対象プロジェクトの開発期間が長期化すると，ある変更が波及する範囲は変化する．開発初期には同時変更されていたファイル群でも，現在では依存関係が失われ，同時変更の関係ではなくなっているものが存在する．共変更ルールの抽出は，過去に何回同時変更されたかという頻度に基づいて行われるため，過去の全改版履歴をマイニングの対象にすると開発初期の履歴がノイズとなり，有益な依存関係が特定できなくなる可能性がある．この問題に対し我々は，時間的近接性を考慮することで現在の変更に関連の強い共変更ルールを抽出できるのではないかと考える．すべてのコミットではなく最近のコミットのみをマイニングの対象とすることで，現在の状況により則した共変更ルールの抽出を行う．

また，バグ修正時に変更漏れが発生してしまった場合，開発者がそのことに後から気づき，必要な変更を追加コミットするという状況が考えられる．改版履歴の分析を行ううえでは，本来同時に変更されるべきファイル群の関係を抽出するために，このような分割されたコミットを 1 つのコミットとして扱うべきである．この問題はコミットの粒度という問題に一般化することが可能である．前述の例のような意図しない分割に限らず，コミットの粒度は開発者やプロジェクトの特性に依存する．たとえば，同一の作業内容でも，ある開発者はすべての変更を 1 度にコミットするのに対し，別の開発者は何度かに分けてコミットするという場合があげられる．このようなコミットの粒度の違いは，改版履歴を分析するうえでノイズとなってしまう．この問題に対し我々は，改版時の作業内容に着目し，同一作業のコミットを統合することで，共変更ルールの質を向上させることができるのではないかと考える．

本研究では，時間的近接性の考慮と，同一作業コミット

[1] 東京工業大学大学院情報理工学研究科計算工学専攻
Department of Computer Science Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Meguro, Tokyo 152–8552, Japan
[2] The School of Computer Science and Communication KTH Royal Institute of Technology 100 44 Stockholm Sweden
[3] 東京工業大学情報理工学院情報工学系
Department of Computer Science School of Computing, Tokyo Institute of Technology, Meguro, Tokyo 152–8552, Japan
[a)] tmori@sa.cs.titech.ac.jp
[b)] tkobaya@cs.titech.ac.jp

の統合が，改版履歴の分析に基づく変更支援に及ぼす影響を明らかにする．具体的には，以下の Research Question に回答する．

RQ1 ソフトウェア進化にともないファイル間の依存関係は変化するか．

RQ2 時間的近接性の考慮により推薦性能は向上するか．

RQ3 同一作業コミットの統合により推薦性能は向上するか．

本論文の貢献は以下のとおりである．

- 大規模な実用ソフトウェアを対象とした調査により，ソフトウェア進化にともないファイル間の依存関係が実際に変化することを明らかにした．
- 評価実験を通して，時間的近接性を考慮することで，より多くの変更漏れファイルを推薦できるようになることを実証した．
- 評価実験を通して，改版時の作業内容に着目し，同一作業のコミットを統合することで，共変更ルールの質向上につなげられることを実証した．

以降，2 章では関連研究について，3 章では実験の設定について，4 章では実験の結果とその考察について，5 章では妥当性への脅威について，6 章では結論と今後の課題について述べる．

2. 関連研究

2.1 Logical Coupling の抽出と利用

Gall ら [4] は，CVS などのバージョン管理システムに蓄積された改版履歴をもとに，同時に変更される可能性の高いファイル間に張られる依存関係を求め，これを Logical Coupling と名づけた．Logical Coupling ではプログラムの静的な情報を解析したのでは明らかにすることのできない依存関係を示すことが可能である．D'Ambros らはファイル単位とモジュール単位の Logical Coupling を統合し視覚化する Evolution Radar [7] を提案した．Alali らは，Logical Coupling における時間的局所性と空間的局所性について調査を行った [8]．松村らは，改版履歴をマイニングし，その変更内容（差分）の類似度も考慮することによって Logical Coupling の精度を向上させる手法を提案している [9]．Wetzlmaier らは商用ソフトウェアの改版履歴から Logical Coupling を抽出し，その有用性を示した [10]．Logical Coupling は進化分析や波及解析，変更支援に応用されており，ファイル単位 [11] だけでなく構文要素単位の Logical Coupling を抽出し分析する試み [12] が行われている．

Zimmermann らが提案した eROSE [5] は，この Logical Coupling をフィールドやメソッドの単位で特定し，開発者があるメソッドを変更した際に，次に変更するメソッドを推薦するツールである．eROSE では Logical Coupling を相関ルールの分析方法を用いて共変更ルールという形で抽出しており，このルールに基づいて推薦を行う．Zimmermann らは eROSE を用いて，開発者がバージョン管理システムにコミットを行う際に，変更漏れを推薦することが可能かという実験を行っている．評価尺度としては，推薦が正しいかどうかの精度を表す *Precision* と，推薦できた変更漏れの割合を表す *Recall* が用いられている．実験の結果，eROSE による推薦の *Precision* は 0.69 であった．これは推薦の 69%が正しかったことを意味しており，eROSE による推薦が正確であることを示している．一方で，*Recall* は 0.023 であった[*1]．これは，わずか 2.3%の変更漏れしか推薦できなかったことを意味している．eROSE による推薦は正確であるが，我々はより多くの変更漏れを推薦する必要があると考える．

Rolfsnes らは，分析対象となる値間の分類概念（Taxonomy）を導入することにより有益な相関ルールを導出する Srikant らの相関ルールマイニング手法 [13] を利用する TARMAQ [14] を提案している．TARMAQ は推薦の順位も加味した推薦の精度を表す *AveragePrecision* において eROSE よりも優れているが，*Recall* については評価がなされていない．

2.2 改版履歴の分析に基づく変更支援手法

Kagdi らが提案した sqminer [15] は，Logical Coupling では考慮されていなかった変更の順序を擬似的に求めて，変更支援の誤検出を減少させている．Canfora らは改版履歴に対してグレンジャー因果性検定を適用し，相関ルールマイニングで検出できる Logical Coupling と組み合わせることでそれぞれの手法を単独で用いるよりも高い *Recall* を示す推薦を可能とした [3]．山森らは，改版履歴に Mylyn [16] の操作履歴を組み合わせることで，変更推薦の性能が向上することを明らかにした [17]．

2.3 時間的近接性の考慮

購買データなどのデータマイニングの分野では，顧客のセグメンテーションのために RFM 分析が用いられている [18], [19]．RFM とは，Recency（時間的近接性），Frequency（頻度），Monetary（累計金額）からなり，それぞれ，ある顧客がどのくらい最近購入したか，どの程度頻繁に購入するか，いくら支払ったのかを意味する．一方で，改版履歴の分析に基づく変更支援では，相関ルール解析を用いることが一般的であり，同時変更の回数である Frequency のみしか考慮されない．

また，プロジェクトが長期間開発されると変更波及を引き起こす依存関係が変化するため，開発初期の改版履歴から得られた共変更ルールが現在の開発においても変わらず

[*1] 彼らの論文の中では *Recall* は 0.75 だと述べられているが，これは eROSE が変更漏れに対して推薦候補を 1 つもあげなかった場合を除いて計算された値である．我々は，全体の試行回数に対して eROSE が推薦を行った割合を表す *Feedback* という値と報告されている *Recall* の積として実際の *Recall* を再計算した．

有用であるとは限らない．Lehman のソフトウェアの進化に関する研究によれば，ソフトウェアは 3 種類に分類され [20]，このうち実世界の問題を扱う E-type に分類されるソフトウェアは，使用者の要求の変化に対応しながら進化し続けるという特徴を持つ [21]．このようなソフトウェアにおいては特に，開発が進む中でファイル間の依存関係も変化することを考慮する必要がある．

我々は，改版履歴に相関ルールマイニングを適用する際に時間的近接性も考慮することで，現在の変更と関連の強い共変更ルールが得られるのではないかと考える．

2.4 同一作業に関連したコミットの統合

McIntosh らは，ソースコードとビルドファイル間の依存関係について調査している．彼らは相関ルールマイニングを適用するうえで，開発者によるコミットの粒度の違いから生まれるノイズを削減するために，同一の作業に関連したコミットを統合している [22]．彼らは課題管理システムから抽出した情報に基づいて同一作業に関連したコミットを統合し，“work item” と呼んでいる．彼らは，単一のコミットよりも work item の方が，同時変更されるソフトウェア・エンティティを特定するのに適した粒度であると述べている．我々は，変更推薦の性能を向上させるうえで，McIntosh らの work item の考え方が有効であると考えている．

3. 実験の設定

本章では，実験の設定について述べる．3.1 節で実験に用いるツールの説明をし，3.2 節でツールを用いた実験の手順を説明する．3.3 節では実験に用いたデータについて記述する．3.4 節では RQ1 に，3.5 節では RQ2 に，3.6 節では RQ3 に固有の実験設定をそれぞれ説明する．3.7 節では RQ2 と RQ3 において議論する推薦性能の評価尺度について説明する．

3.1 実験に用いる変更推薦ツール

本研究では，アプリオリ・アルゴリズム [23] を用いた相関ルールマイニングによって Git リポジトリから抽出した共変更ルールに基づいて推薦を行うツール **LCExtractor** [24] を実装し，実験に使用する．共変更ルールは “A ⇒ B” という形式で表され，“もしファイル A が変更されたならば，ファイル B も同時に変更される可能性が高い” ことを意味する．A，B はそれぞれルールの左辺，右辺と呼ばれる．本研究では，左辺はファイルの集合であり，右辺は単一のファイルである．

LCExtractor は eROSE と同様の機能を有しているが，いくつかの違いが存在する．eROSE がフィールドやメソッドといったクラスの要素単位の共変更ルールの抽出が可能であるのに対し，LCExtractor はファイル間における共変更ルールのみを対象としている．また，eROSE が Eclipse のプラグインであるのに対し，LCExtractor は過去のコミットにおいて変更すべきファイルが変更されなかったという状況を仮想的に作り出し，変更漏れを推薦できたかどうかを評価するツールである．

LCExtractor には，eROSE よりも優れている点がいくつか存在する．まず，改版履歴の中で改名されたファイルを追跡することを可能とした．この改良により，LCExtractor では改名前と改名後のファイルを同一のものとして扱えるため，改名前のファイルから得られた共変更ルールを用いて改名後のファイルを推薦することが可能となった．改名されたファイルだけでなく削除されたファイルの追跡も可能であり，削除されたファイルが推薦の候補となった場合は推薦を行わないようになっている．また，eROSE が解析の対象とするバージョン管理システムが CVS であるのに対し，LCExtractor は現在主流となっている Git や Subversion を対象としている．

本研究では，iMac Retina 5k Late 2014，CPU 4 GHz Intel Core i7，メモリ 32 GB の環境上で LCExtractor を利用し実験を行う．

3.2 実験手順：共変更ルールの抽出と変更推薦

LCExtractor を用いた共変更ルールの抽出と変更推薦および推薦結果の評価方法について説明する．**図 1** は実験手順の概要を示したものである．まず，“最新の 1,000 コミット”，のように改版履歴の中で評価の対象として扱うコミットの範囲を設定する．LCExtractor は設定された範

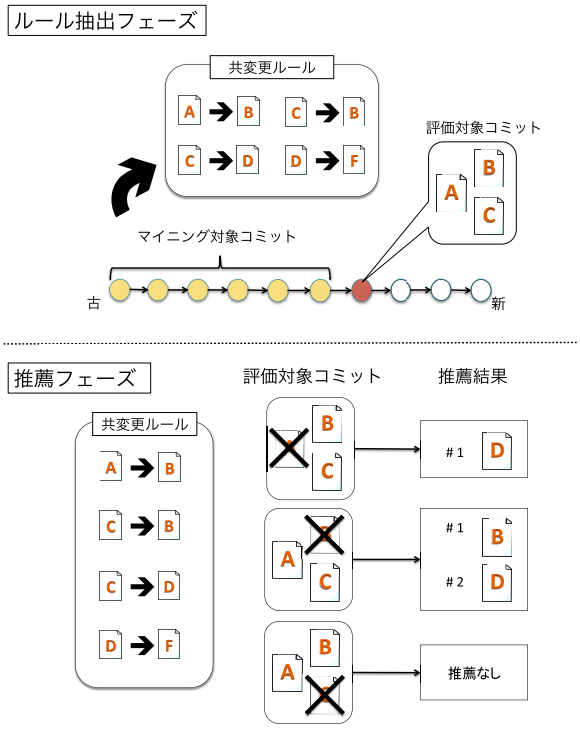

図 1 実験手順
Fig. 1 Experimental procedure.

囲の中で最も古いコミットを最初の評価対象とし，図 1 のルール抽出フェーズのように評価対象のコミットより過去のコミットに対して相関ルールマイニングを適用することで共変更ルールを抽出する．

次に，leave-one-out 交差検証によって評価対象のコミットを評価する．ファイルを 1 つ欠損させることによって，そのファイルをコミットし忘れたという状況を仮想的に作り出す．抽出した共変更ルールの中で左辺のファイル集合が評価対象コミットに含まれているものがあった場合に，LCExtractor はその共変更ルールの右辺を推薦の候補とし，候補となったファイルをランキング形式で出力する．このとき，変更漏れとして扱っているファイル以外で評価対象コミットに含まれているものは推薦する必要がないため，ランキングからは除外する．

ファイルを 1 つ欠損させて変更漏れとして扱い，推薦を行うという上記の手順を，評価対象コミットに含まれるすべてのファイルに対して行う．図 1 の推薦フェーズの例では，評価対象コミットに含まれるファイルが A，B，C の 3 つであるため，各ファイルを変更漏れとした場合についてそれぞれ推薦を行っている．評価対象コミット内のすべてのファイルに対して推薦が終了したら，評価対象コミットを次のコミット，すなわち時系列順で 1 つ新しいコミットに変更し，同様の手順で推薦を行う．これを最初に設定した範囲のすべてのコミットに対して繰り返し行う．留意すべき点として，評価対象として扱われたコミットも，その後のサイクルでは過去のコミットとして共変更ルールを抽出する際にマイニングの対象として利用される．

3.3 データセット

実験対象としては，**表 1** に示す 6 つの大規模 OSS を用いる．GitHub[*2]から各プロジェクトのすべての改版履歴をクローン[*3]し，実験に使用する．

3.4 ソフトウェア進化にともなうファイル間の依存関係変化の調査

Eclipse は，Lehman の分類 [21] では継続的な変更が行われるソフトウェアに分類される．また，Eclipse は実際に継続的な機能追加が行われており，開発期間全体を通してファイルが追加，変更，削除され続けていることが Mens らによって報告されている [25]．上記の理由から，継続的な変更が行われるソフトウェアの代表として Eclipse における依存関係の変化を調査する．この調査では，すべての改版履歴を分析して共変更ルールを抽出した場合と，推薦対象コミットの直近の 5,000 コミットのみを分析して共変更ルールを抽出した場合における，推薦成功数の違いについて考察する．LCExtractor の中で用いられているアプリオリ・アルゴリズムでは，最小支持度（以下，*minsup*）と最小確信度（以下，*minconf*）という 2 つのパラメータが必要である．本実験では *minsup* として 0.0025，*minconf* として 0.1 を用いる．また，3.2 節で述べたように，LCExtractor が改版履歴の中で評価の対象として扱うコミットの範囲を設定する必要がある．本実験では最新の 2,000 コミットを評価対象とする．

表 1 対象プロジェクト
Table 1 Target projects.

対象プロジェクト	総コミット数	開発期間
Eclipse JDT	21,378	2001–2014
Firefox	395,466	1998–2014
Tomcat	13,824	2006–2014
OpenLDAP	22,068	1998–2016
OpenSSL	17,370	1998–2016
GCC	148,080	1988–2016

3.5 時間的近接性の考慮

時間的近接性の考慮が推薦の性能に及ぼす影響を調査するため，図 1 のルール抽出フェーズにおいて相関ルールマイニングの対象とするコミットの範囲を，推薦対象コミットより古い直近の 5,000 コミットとした場合と，推薦対象コミットより古いすべてのコミットとした場合で比較を行う．前者を時間的近接性を考慮した場合として扱い，後者をベースラインと呼ぶ．

本実験では，*minsup* を Eclipse と Firefox では 0.0025，Tomcat と GCC では 0.001，OpenLDAP と OpenSSL では 0.002 に設定する．*minconf* についてはすべてのプロジェクトで 0.1 から 0.9 まで 0.1 刻みで変化させて実験を行う．評価対象として扱うコミットの範囲は 3.4 節の設定に合わせ，すべてのプロジェクトで最新の 2,000 コミットを評価対象とする．

3.6 同一作業コミットの統合

同一作業に関連したコミットを統合することが推薦の性能に及ぼす影響を調査するため，同一作業コミットを統合した改版履歴を用いた場合と，コミットの統合を行わない場合とで比較を行う．後者をベースラインと呼ぶ．

本実験では，同一作業に関係するコミットを統合するため，コミットメッセージに含まれるバグ ID に着目する．これは課題管理システムやバグ管理システムにおいてバグに割り振られる番号である．事前に，対象プロジェクトの改版履歴を目視で確認したところ，Eclipse，Firefox，Tomcat，OpenLDAP の 4 プロジェクトについては下記の規則でバグ ID が記述されていることが分かった．

- bug[# \t]*[0-9]+

*2 https://github.com/
*3 Eclipse，Firefox，Tomcat は 2014 年 12 月 2 日時点，OpenLDAP，OpenSSL は 2016 年 7 月 27 日時点，GCC は 2016 年 8 月 5 日時点．

- pr[# \t]*[0-9]+
- show_bug\.cgi\?id=[0-9]+
- its[# \t]*[0-9]+

改版履歴中のコミットメッセージが上記の正規表現に部分一致した場合に，そのコミットとバグ ID を対応付ける．連続するコミットで同一のバグ ID が割り当てられているものがあれば，それらを統合して 1 つのコミットとして扱うようにする．統合の際はコミットメッセージに記述されたバグ ID の情報のみを用いており，コミットを行った開発者の名前や，コミットされた時間の間隔などの情報は用いない．また，機能追加やリファクタリングに関するコミットに関しては統合を行っていない．

コミットメッセージ内のバグ ID の記述に規則性を見つけることのできなかった OpenSSL と GCC は対象外とし，その他の 4 つのプロジェクトを用いて評価を行う．また，共変更ルールの抽出時には評価対象コミットより古いすべての改版履歴を使用し，時間的近接性は考慮しない．*minsup* と *minconf* の設定，評価対象として扱うコミットの範囲は 3.5 節と同様である．

3.7 評価尺度

本研究の目的は，バグの原因となる変更漏れの防止である．我々は推薦性能を評価するため，文献 [5] と同様にコミット時の変更漏れに対する推薦という実験設定を用いている．文献 [5] では評価尺度として，主に *Precision* と *Recall* を用いているが，*Precision* を用いることにはいくつかの問題点が存在する．上記の実験設定では，評価対象コミットの中からファイルを 1 つだけ欠損させて変更漏れファイルとし，推薦の正解として扱うため，正解となる推薦はつねに 1 ファイルである．このため，正解が高い順位で推薦されていたとしても，推薦数が多いと多数の擬陽性（false positives）のせいで *Precision* が不当に低くなってしまう．また，正解が推薦された順位が異なる場合でも，推薦された数が同じであれば *Precision* は等しくなってしまうため，推薦の正確さを正しく測ることができない．そこで，本研究では推薦の正確さを表す指標として，平均逆順位（*MRR*）を用いる．*MRR* が高いと，正解が平均して高い順位で推薦されていることを表す．たとえば，平均して 3 位に正解が推薦されていた場合に，*MRR* は 0.33 となる．*MRR* は文献 [5] の中では用いられていない評価指標であるが，正解の順位まで推薦を行った場合の *Precision* と同義であるため，推薦の正確さを表す指標として妥当であると考える．推薦できる変更漏れの割合を表す指標としては，文献 [5] と同様に *Recall* を用いた．*MRR* と *Recall* はトレードオフの関係にあるが，我々はより多くの変更漏れを推薦することが重要であると考えているため，*MRR* より *Recall* を重視する．

LCExtractor における *MRR*，*Recall* の算出方法について説明する．共変更ルールの集合を $Rule = \{(l_1, r_1), (l_2, r_2), ..., (l_m, r_m)\}$ とする．l_i，r_i はそれぞれ 3.1 節で述べた左辺と右辺である．改版履歴を $Commit = \{com_1, com_2, ..., com_n\}$ とする．com_i は各コミットにおいて変更されたファイル集合を表す．ファイル $c \in com_i$ を com_i から取り除くことで変更漏れとして扱い，c を取り除いた com_i に対して推薦されるファイル集合 $recom_{i,c}$ を下記の式で定義する．

$$recom_{i,c} = \bigcup_{(l_j, r_j) \in Rule} \begin{cases} r_j & (\text{if } l_j \subseteq (com_i - \{c\})) \\ \emptyset & (\text{else}) \end{cases}$$

$rank_{i,c}$ は確信度によって順位付けされて推薦された $recom_{i,c}$ における c の順位を表す．確信度とは相関ルールの質を表す尺度の 1 つである [26]．$recom_{i,c}$ の中に c が含まれていなかった場合は，$rank_{i,c}$ は 0 となる．

次に，各 com_i に対して，$feedback_i$，$recall_i$，mrr_i を定義する．

$$feedback_i = \frac{1}{|com_i|} \sum_{c \in com_i} \begin{cases} 1 & (\text{if } recom_{i,c} \neq \emptyset) \\ 0 & (\text{else}) \end{cases}$$

$$recall_i = \frac{1}{|com_i|} \sum_{c \in com_i} \begin{cases} 1 & (\text{if } c \in recom_{i,c}) \\ 0 & (\text{else}) \end{cases}$$

$$mrr_i = \frac{1}{feedback_i \cdot |com_i|} \sum_{c \in com_i} \frac{1}{rank_{i,c}}$$

$feedback_i$ は，com_i に対して評価を行う際の推薦の発生率を表す．mrr_i の計算には，文献 [5] の中で *Precision* を計算する際と同様に，$feedback_i$ の値を用いている．これは推薦が行われた場合のみ推薦の正確さを測るためである．$feedback_i$ が 0 であった場合，すなわち変更漏れに対して推薦が行われなかった場合は mrr_i の計算を行わず，後述の *MRR* の計算時に $feedback_i$ が 0 となったコミット com_i を取り除いている．$recall_i$ の計算には，文献 [5] とは異なり $feedback_i$ の値を用いていない．これは推薦の有無にかかわらず，すべての変更漏れに対して推薦することのできた割合を測るためである．$feedback_i$ が 0 であった場合，すなわち変更漏れに対して推薦が行われなかった場合は，$recall_i$ は 0 となる．最後に，*Recall* と *MRR* をそれぞれ $recall_i$ の平均，mrr_i の平均として算出する．また，*Recall* と *MRR* はトレードオフの関係にあるため，*Recall* と *MRR* の調和平均として F_{MRR} を定義し，推薦性能の総合的な評価に使用する．

$$Commit^* = \{com_i | com_i \in Commit, feedback_i \neq 0\}$$

$$Recall = \frac{1}{|Commit|} \sum_{com_i \in Commit} recall_i$$

$$MRR = \frac{1}{|Commit^*|} \sum_{com_i \in Commit^*} mrr_i$$

$$F_{MRR} = 2 \cdot \frac{MRR \cdot Recall}{MRR + Recall}$$

4. 実験結果

4.1 RQ1：ソフトウェア進化にともないファイル間の依存関係は変化するか

3.4 節で設定した評価対象コミット内の各ファイルを欠損させて変更漏れとして扱うことで，計 9,201 回の推薦試行を行った．**表 2** に共変更ルールの抽出時に直近の 5,000 コミットのみを分析した場合と，全改版履歴を分析した場合における推薦の成功数の違いを示す．9,201 回の推薦試行のうち，全改版履歴を分析した場合には推薦を行うことができず，直近の 5,000 コミットのみを分析対象とすることで推薦が行えた回数は 1,537 回であった．推薦に成功した 1,537 個の変更漏れを調べた結果，857 個は直近の 5,000 コミットよりも昔から存在するファイルであることが分かった．これは，直近の 5,000 コミットより以前ではあまり同時変更されていなかったが，直近の 5,000 コミットでは頻繁に同時変更されるようになったファイル群が多数存在していることを意味している．また，全改版履歴を分析した場合のみ推薦に成功した回数はわずか 127 回であった．これは，古い改版履歴の中でのみ頻繁に同時変更されていたファイル間の共変更ルールが，現在の開発に対する推薦において大きく貢献しないことを意味している．これらのことから，ファイル間の依存関係がソフトウェア進化にともない確かに変化することが分かった．

RQ1 に対する回答： ソフトウェア進化にともない，実際にファイル間の依存関係が大きく変化していくことが明らかになった．

表 2 推薦に成功した回数
Table 2 # of successful recommendation.

	推薦成功数
直近の 5,000 コミットから学習した場合のみ成功	1,537 回
両方で成功	1,426 回
全改版履歴から学習した場合のみ成功	127 回

4.2 RQ2：時間的近接性の考慮により推薦性能は向上するか

RQ1 から実際にファイル間の依存関係が変化することが分かったため，時間的近接性を考慮することで現在の変更に関係のある質の高い共変更ルールが得られることが期待できる．本節では，3.5 節の設定に従って実験を行った結果を示し，時間的近接性を考慮した場合の推薦性能の変化について議論する．

図 2 はそれぞれのプロジェクトにおける *MRR* と *Recall* の関係を，*minconf* の値を変化させてプロットしたグラフである．赤が時間的近接を考慮したときのグラフであり，青がベースラインのグラフである．時間的近接性を考慮することですべてのプロジェクトにおいて *Recall* が向上した．一方で，Tomcat 以外の 5 プロジェクトでは *MRR* の最大値が低下した．我々は開発者に対してより多くの変更漏れを推薦することを目的としているため，すべてのプロジェクトにおいて *Recall* が向上したことは良い結果であると考える．特に，Eclipse については *Recall* が大きく向上した．図 2 から分かるように，ベースラインでは *Recall* の最大値が 0.11 であるのに対し，時間的近接性を考慮することで *Recall* が 0.28 に向上した．これは，時間的近接性を考慮した結果，ベースラインに比べて 2.5 倍の変更漏れを推薦できたことを意味している．*MRR* に関しては，最

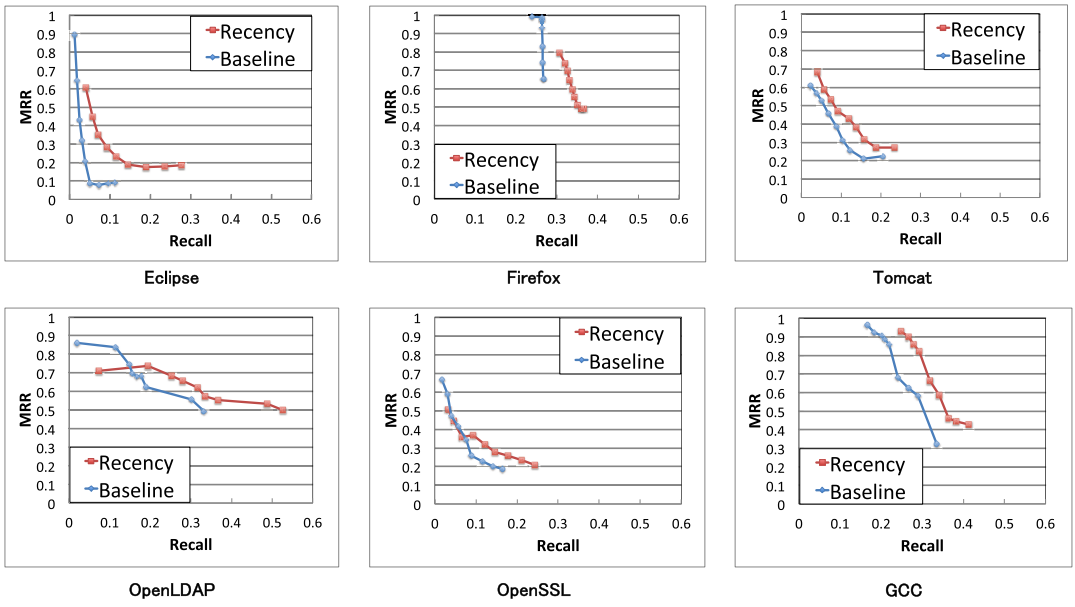

図 2 *minconf* を変化させたときの推薦性能．縦軸：*MRR*，横軸：*Recall*
Fig. 2 Performance of recommendations with varying *minconf*. x-axis: *Recall*, y-axis: *MRR*.

表 3 時間的近接性の考慮による F_{MRR} の最大値の変化
Table 3 Maximum F_{MRR} before/after consideration of recency.

プロジェクト	時間的近接性を考慮	ベースライン
Eclipse JDT	0.221 (*minconf*: 0.1)	0.100 (*minconf*: 0.1)
Firefox	0.447 (*minconf*: 0.8)	0.416 (*minconf*: 0.8)
Tomcat	0.252 (*minconf*: 0.1)	0.214 (*minconf*: 0.1)
OpenLDAP	0.512 (*minconf*: 0.1)	0.397 (*minconf*: 0.1)
OpenSSL	0.225 (*minconf*: 0.1)	0.175 (*minconf*: 0.1)
GCC	0.431 (*minconf*: 0.6)	0.386 (*minconf*: 0.2)

大値で見るとベースラインよりも低下したプロジェクトの方が多かったが，*MRR* の値が同一であるときの *Recall* を比べると時間的近接性を考慮したときのほうが良い結果であることが分かる．これは実際に開発者に対して推薦を行う場合に，開発者が同じ順位まで推薦を見たときに時間的近接性を考慮した方がより多くの変更漏れを指摘できることを意味している．

表 3 に各プロジェクトの F_{MRR} の最大値を，時間的近接性を考慮した場合とベースラインの場合でそれぞれ示す．すべてのプロジェクトにおいて，時間的近接性を考慮した場合のほうが，ベースラインよりも F_{MRR} の最大値が高い結果となった．これは，時間的近接性を考慮することで，ベースラインに比べて総合的な推薦の性能が向上したことを意味している．

支持度は共変更ルール抽出時に分析の対象となったコミットの中で，共変更ルールが表すファイルの同時変更が起こった割合である．同時変更された回数が同じでもベースラインでは時間的近接性を考慮した場合に比べ分析対象となるコミット数が多いため，支持度が相対的に下がり，*minsup* を超えず共変更ルールが得られない場合が存在する．時間的近接性を考慮した場合に分析対象となる改版履歴はベースラインにおける分析対象の一部でもあるため，ベースラインでも *minsup* を下げることで，時間的近接性を考慮した場合にしか得られなかった共変更ルールを抽出することができるようになる．時間的近接性を考慮する方法と，*minsup* を下げる方法とで，有効性の違いを分析するため，最も *Recall* の変化率の大きかった Eclipse に対し，ベースラインの設定において *minsup* を 0.001 まで下げて追加の実験を行った．

図 3 は Eclipse における *MRR* と *Recall* の関係を，*minconf* の値を変化させてプロットしたグラフである．赤が時間的近接性を考慮したときのグラフ，青がベースラインのグラフ，緑が *minsup* を 0.001 としたベースラインのグラフである．ベースラインの設定で *minsup* を 0.001 にした結果，*Recall* が時間的近接性を考慮した場合と同等となったが，*MRR* の最大値は低下した．支持度は共変更ルールの質を表す指標なので，一般的に *minsup* を下げるとより多くの共変更ルールを得られる代わりに，ノイズとなるものも多数抽出されてしまう．時間的近接性を考慮すれば，*minsup* を下げなくても推薦に必要な共変更ルールの抽出が行えることが明らかになった．

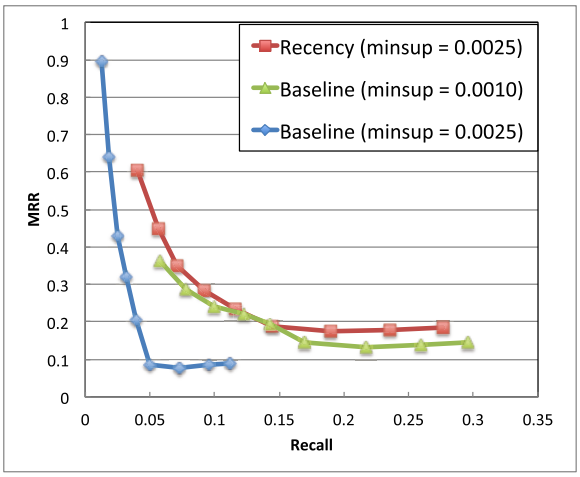

図 3 *minconf* を変化させたときの推薦性能
縦軸：*MRR*，横軸：*Recall*．
Fig. 3 Performance of recommendations with varying *minconf*. x-axis: *Recall*, y-axis: *MRR*.

RQ2 に対する回答： 時間的近接性を考慮することで，ベースラインよりも質の高い共変更ルールを抽出できることが明らかになった．全体の傾向として，時間的近接性の考慮が *Recall* の向上につながることが分かった．

4.3 RQ3：同一作業コミットの統合により推薦性能は向上するか

本節では，3.6 節の設定に従って実験を行った結果を示し，同一作業コミットの統合による推薦性能の向上について議論する．**図 4** はそれぞれのプロジェクトにおける *MRR* と *Recall* の関係を，*minconf* の値を変化させてプロットしたグラフである．黄色が同一作業コミットを統合した場合のグラフであり，青がベースラインのグラフである．

図 4 から，Firefox 以外の 3 プロジェクトでは *MRR* も *Recall* も向上していないことが分かる．一方で，Firefox に関しては同一作業コミットを統合することで *Recall* が向上した．これは，同一作業に関連しているのに分割されてコミットされていたためベースラインでは得ることのできなかった共変更ルールを抽出できたことを意味している．Firefox 以外のプロジェクトではベースラインとの差があまり見られなかった．この原因を調べるため，詳細な調査を行った．

表 4 に各プロジェクトの，同一作業コミットを統合する前後のコミット数の変化を示す．Firefox では統合によって全体のコミット数が大きく減少した一方で，その他の 3 プロジェクトでは統合できたコミットが 1〜3%と少なかっ

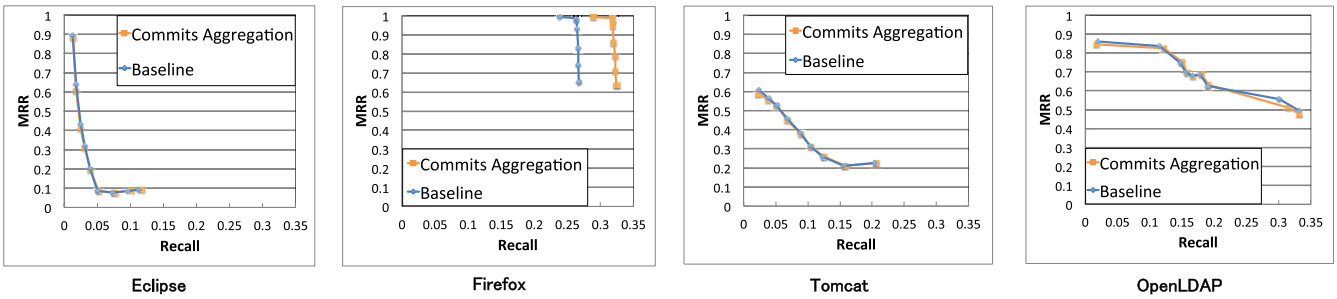

図 4 *minconf* を変化させたときの推薦性能．縦軸：*MRR*，横軸：*Recall*
Fig. 4 Performance of recommendations with varying *minconf*. x-axis: *Recall*, y-axis: *MRR*.

表 4 コミットの統合によるコミット数の変化
Table 4 # of commits before/after commits aggregation.

プロジェクト	統合前	統合後（統合できた割合）
Eclipse JDT	21,378	21,092 (1%)
Firefox	395,466	358,264 (9%)
Tomcat	13,824	13,661 (1%)
OpenLDAP	22,068	21,358 (3%)

表 5 コミットの統合による F_{MRR} の最大値の変化
Table 5 Maximum F_{MRR} before/after commits aggregation.

プロジェクト	コミットを統合	ベースライン
Eclipse JDT	0.103 (*minconf*: 0.1)	0.100 (*minconf*: 0.1)
Firefox	0.481 (*minconf*: 0.8)	0.416 (*minconf*: 0.8)
Tomcat	0.215 (*minconf*: 0.1)	0.214 (*minconf*: 0.1)
OpenLDAP	0.392 (*minconf*: 0.1)	0.397 (*minconf*: 0.1)

たことが分かった．このことから，同一作業コミットの統合が共変更ルールの質に及ぼす影響は，プロジェクトの特性やコミットポリシに依存するという知見を得た．

表 5 に各プロジェクトの F_{MRR} の最大値を，同一作業コミットを統合した場合とベースラインの場合でそれぞれ示す．Firefox に関しては，同一作業コミットを統合した場合の方がベースラインよりも F_{MRR} の最大値が高い結果となった．そのほかの 3 つのプロジェクトについては，前述のとおり *MRR* と *Recall* がともにあまり変化しなかったため，F_{MRR} の最大値も同等となった．これは，プロジェクトによっては同一作業コミットを統合することで質の高い共変更ルールが得られ，推薦の性能が向上することを意味している．また，コミットの統合によって推薦性能が向上する場合はあっても，性能が低下することはないという知見を得た．

同一作業コミットを統合することで，ノイズとなるルールも同時に得られてしまうことが懸念される．同一作業コミットの統合によるノイズの増加率を明らかにするため，同一作業コミットの統合による影響が大きかった Firefox を対象に，統合前に得られた共変更ルールと統合後に新たに得られた共変更ルールについて，それぞれノイズ含有量の分析を行った．

表 6 Firefox における統合前，統合後のルールのノイズ含有量
Table 6 Noise content before/after commits aggregation on Firefox.

ルール	推薦に貢献	推薦に悪影響	無関係
統合前から存在	2,102 (52.8%)	31 (0.8%)	1,851 (46.4%)
統合後新たに出現	108 (22.6%)	9 (1.9%)	361 (75.5%)

表 6 に Firefox において統合前から存在していた共変更ルール 3,984 個と統合後に新たに得られた共変更ルール 478 個の，推薦に貢献した割合，推薦に悪影響を与えた割合，推薦に使用されなかった割合をそれぞれ示す．この分析では，推薦の結果に悪影響を与えたルール，つまり 1 度も正解の推薦に貢献せず誤った推薦を引き起こしたルールをノイズと定義する．統合前から存在したルールに含まれるノイズの割合が 0.8%であったのに対し，統合後に新たに得られたルールに含まれるノイズの割合は 1.9%であった．これは，統合前から存在したルールと統合後に新たに得られたルールのノイズ含有率に大きな差がないことを表しており，本手法によって *MRR* があまり低下しないことの裏付けにもなっている．このことから，コミットを統合することで新たに得られるルールによって推薦できる正解の数が増え，推薦の精度にもさほど影響を与えないことを明らかにできた．

RQ3 に対する回答： 同一作業コミットを統合することで，プロジェクトによっては質の高い共変更ルールが抽出することができることが明らかになった．また，抽出できる有益なルールが増えなかった場合においても，同一作業コミットの統合による悪影響はないということが明らかになった．

5. 妥当性への脅威

LCExtractor の実装と，実験時のパラメータの設定に関して内的妥当性への脅威が存在する．我々は LCExtractor の実装を注意深く行ったが，認識できていない誤りが存在している可能性がある．実験の際には，共変更ルールの抽出時に必要なパラメータである *minsup* を Eclipse と Firefox

では 0.0025，Tomcat と GCC では 0.001，OpenLDAP と OpenSSL では 0.002 に設定した．本手法では，得られる共変更ルールの数が少ないと変更漏れに対して推薦がほとんど発生せず，推薦性能の適切な比較が行えない．現在のところ適切な *minsup* の値を一意に定める手法がないため，今回の実験ではベースラインの設定で *minconf* を 0.1 としたときに，変更漏れに対して推薦が発生する割合を表す $feedback_i$ の平均値が 0.33 以上となるように（少なくとも 3 回に 1 回は推薦が発生するように）minsup の値を定めた．*minsup* の決定方法が一意ではないが，個々のプロジェクトにおいて手法を比較する際の *minsup* を統一しているため，各プロジェクトごとの *minsup* が異なっていても，本手法がベースラインよりも優れているという評価には影響を与えないと考える．

実験結果の一般性に関して外的妥当性への脅威が存在する．本研究では実験の対象として 6 つの異なるプロジェクトを用いた．CLI アプリケーション，GUI アプリケーション，Web コンテナなど様々な種類のプロジェクトを対象とし結果の一般性を高めるようにしてはいるが，今後より多くのプロジェクト・他ドメインを対象とした実験を行い，同様の結果が得られるかさらなる調査が必要と考える．

実験の設定に関して，構造的妥当性への脅威が存在する．今回の実験では，共変更ルールの抽出時にすべての改版履歴を使用するのではなく，推薦対象コミットの直近の 5,000 コミットのみを分析の対象とすることを時間的近接性の考慮と位置づけた．しかし，時間的近接性を考慮するうえで分析の対象とするコミット数を変化させた実験を行っていない．また RQ3 に対する実験において，コミットメッセージに記述されたバグ ID の情報のみに基づいてコミットの統合を行った．本研究ではバグ修正だけでなく，機能追加やリファクタリングなどを含むあらゆる目的の共変更を対象としているが，機能追加やリファクタリングに関するコミットを適切に統合する方法がなく，バグ修正に関するコミットのみを統合して評価を行った．手法の有効性をより適切に評価するためには，機能追加やリファクタリングに関するコミットも統合する必要があると考える．バグ修正に関するコミットの統合については，課題管理システムやバグ管理システムの詳細な情報も利用する手法 [27] や，変更内容から自動生成されたコミットメッセージの情報も加味する手法 [28] を用いることでより高い精度でコミットの統合が行えると考える．

6. おわりに

改版履歴の分析から得られた共変更ルールに基づいて次に変更すべき箇所を推薦することで，開発者への支援を行う研究が進められている．しかし，既存手法では推薦することのできる変更漏れの数が少ないという問題がある．本研究では時間的近接性の考慮と，同一作業コミットの統合により質の高い共変更ルールを抽出できると考え，これらが改版履歴に基づく変更推薦にどのような影響を及ぼすかを調査した．

我々はまず，継続的な変更が行われるソフトウェアの代表である Eclipse に対して調査を行い，ソフトウェア進化にともなって確かにファイル間の依存関係が変化することを明らかにした．その後，6 つの代表的かつ大規模な OSS を用いた評価実験により，時間的近接性を考慮することでより多くの変更漏れを推薦できることを実証した．また，同一作業コミットの統合については，推薦の性能に悪影響を与えることはなく，プロジェクトによってはより多くの変更漏れが推薦できることを実証した．

今後の研究としては，次のような内容を明らかにしていく．今回，時間的近接性の考慮が有効であることを示したが，どの程度最近の改版履歴を分析の対象とすればよいかは明らかになっていない．時間的近接性としてどの程度最近の改版履歴を分析対象とすることが最適か，定量的な調査を行うことで特性を明らかにしていく予定である．

謝辞 本研究の一部は JSPS 科研費 JP24300006，JP26280021，JP15H02683 の助成を受けた．

参考文献

[1] Briand, L.C., Wust, J. and Lounis, H.: Using coupling measurement for impact analysis in object-oriented systems, *Proc. ICSM*, pp.475–482 (1999).
[2] Geipel, M.M. and Schweitzer, F.: Software change dynamics: evidence from 35 java projects, *Proc. FSE*, pp.269–272 (2009).
[3] Canfora, G., Ceccarelli, M., Cerulo, L. and Di Penta, M.: Using multivariate time series and association rules to detect logical change coupling: an empirical study, *Proc. ICSM*, pp.1–10 (2010).
[4] Gall, H., Hajek, K. and Jazayeri, M.: Detection of logical coupling based on product release history, *Proc. ICSM*, pp.190–198 (1998).
[5] Zimmermann, T., Zeller, A., Weissgerber, P. and Diehl, S.: Mining version histories to guide software changes, *IEEE TSE*, Vol.31, No.6, pp.429–445 (2005).
[6] Mori, T., Hagward, A.M. and Kobayashi, T.: Effects of Recency and Commits Aggregation on Change Guide Method Based on Change History Analysis, *Proc. ICSEA*, pp.96–101 (2015).
[7] D'Ambros, M., Lanza, M. and Lungu, M.: The evolution radar: Visualizing integrated logical coupling information, *Proc. MSR*, pp.26–32 (2006).
[8] Alali, A., Bartman, B., Newman, C.D. and Maletic, J.I.: A preliminary investigation of using age and distance measures in the detection of evolutionary couplings, *Proc. MSR*, pp.169–172 (2013).
[9] 松村知子，横森励士，大杉直樹，川口真司，松下 誠：ファイルの同時変更パターンと変更差分の分析による論理的結合関係の自動抽出，ソフトウェアシンポジウム論文集，pp.104–112 (2005).
[10] Wetzlmaier, T., Klammer, C. and Ramler, R.: Extracting dependencies from software changes: an industry experience report, *Proc. IWSM-MENSURA*, pp.163–168 (2014).

[11] Ying, A., Murphy, G., Ng, R. and Chu-Carroll, M.: Predicting source code changes by mining change history, Vol.30, No.9, pp.573–586 (2004).
[12] Robbes, R., Pollet, D. and Lanza, M.: Logical Coupling Based on Fine-Grained Change Information, *Proc. WCRE*, pp.42–46 (2008).
[13] Srikant, R., Vu, Q. and Agrawal, R.: Mining Association Rules with Item Constraints, *Proc. SIGKDD*, pp.67–73 (1997).
[14] Rolfsnes, T., Alesio, S.D., Behjati, R., Moonen, L. and Binkley, D.W.: Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis, *Proc. SANER*, pp.201–212 (2016).
[15] Kagdi, H., Yusuf, S. and Maletic, J.I.: Mining sequences of changed-files from version histories, *Proc. MSR*, pp.47–53 (2006).
[16] Kersten, M. and Murphy, G.C.: Mylar: A degree-of-interest model for IDEs, *Proc. AOSD*, pp.159–168 (2005).
[17] 山森章弘，Hagward, A.，小林隆志：改版履歴を用いた変更支援手法における操作履歴の活用に向けて，信学技報（ソフトウェアサイエンス），SS2014-54, pp.127–132 (2015).
[18] Verhoef, P.C., Spring, P.N., Hoekstra, J.C. and Leeflang, P.S.: The commercial use of segmentation and predictive modeling techniques for database marketing in the Netherlands, *Decision Support Systems*, Vol.34, No.4, pp.471–481 (2003).
[19] McCarty, J.A. and Hastak, M.: Segmentation approaches in data-mining: A comparison of RFM, CHAID, and logistic regression, *Journal of business research*, Vol.60, No.6, pp.656–662 (2007).
[20] Lehman, M.M.: Programs, life cycles, and laws of software evolution, *Proc. IEEE*, Vol.68, No.9, pp.1060–1076 (1980).
[21] Lehman, M.M.: Laws of Software Evolution Revisited, *EWSPT*, Springer-Verlag, pp.108–124 (1996).
[22] McIntosh, S., Adams, B., Nguyen, T.H., Kamei, Y. and Hassan, A.E.: An empirical study of build maintenance effort, *Proc. ICSE*, pp.141–150 (2011).
[23] Agrawal, R., Srikant, R. et al.: Fast algorithms for mining association rules, *Proc. VLDB*, Vol.1215, pp.487–499 (1994).
[24] Hagward, A.M.: Using Git Commit History for Change Prediction: An empirical study on the predictive potential of file-level logical coupling, Master's thesis, KTH, School of Computer Science and Communication (CSC) (2015).
[25] Mens, T., Fernandez-Ramil, J. and Degrandsart, S.: The evolution of Eclipse, *Proc. ICSM*, pp.386–395 (2008).
[26] Agrawal, R., Imieliński, T. and Swami, A.: Mining association rules between sets of items in large databases, *ACM SIGMOD Record*, Vol.22, No.2, pp.207–216 (1993).
[27] Wu, R., Zhang, H., Kim, S. and Cheung, S.-C.: Relink: recovering links between bugs and changes, *Proc. FSE*, pp.15–25 (2011).
[28] Le, T.-D.B., Linares-Vásquez, M., Lo, D. and Poshyvanyk, D.: RCLinker: automated linking of issue reports and commits leveraging rich contextual information, *Proc. ICPC*, pp.36–47 (2015).

森 達也

2015 年東京工業大学工学部情報工学科卒業．2017 年同大学大学院情報理工学研究科計算工学専攻修士課程修了．同年アマゾンウェブサービスジャパン株式会社に入社．ソフトウェア保守，リポジトリマイニング，プログラム変更支援などの研究に従事．

アンダース ハグワード

2012 年スウェーデン王立工科大学卒業．2015 年同大学大学院修士課程修了．2013～2014 年 Academic Cooperation Agreement Program（ACAP）の交換留学生として東京工業大学大学院に在籍．現在，スウェーデンヨーデボリの IT 企業にソフトウェアエンジニアとして勤務．ソフトウェア保守，リポジトリマイニング等の研究に従事．

小林 隆志 （正会員）

1997 年東京工業大学工学部情報工学科卒業．1999 年同大学大学院情報理工学研究科計算工学専攻修士課程修了．2004 年同専攻博士課程修了．博士（工学）．2002 年同大学学術国際情報センター助手．2007 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター特任准教授．2009 年同研究科准教授．2012 年東京工業大学大学院情報理工学研究科計算工学専攻准教授．現在，同大学情報理工学院准教授．ソフトウェア再利用技術，プログラム理解，リポジトリマイニング，ソフトウェア設計，複合メディアコンテンツの管理・検索，Web サービス連携等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，日本データベース学会，ACM，IEEE，IEEE-CS 各会員．