情報処理学会論文誌 Vol.56 No.2 657–668 (Feb. 2015)

トランザクション推定に基づく リポジトリをまたいだロジカルカップリングの検出

早瀬 康裕[2,a)] 中村 高士[1,b)] 天笠 俊之[2,c)] 北川 博之[2,d)]

受付日 2014年5月12日, 採録日 2014年11月10日

概要：ソフトウェアモジュール間の関係を表す指標ロジカルカップリングは，あるモジュールが変更されたときに別のモジュールもまた一緒に変更されやすいという関係を表しており，プログラム理解やソフトウェアの変更支援などに利用されている．既存のロジカルカップリング検出手法は，バージョン管理システムの１つのリポジトリの中で一緒に変更された関係に基いていたため，単一のソフトウェアプロダクトの中でしかロジカルカップリングを見つけることができなかった．しかし今日では，ライブラリやフレームワークといった他のプロダクトを利用してソフトウェア開発を行うのが一般的であるため，変更の影響は複数のプロダクトに及ぶと考えられる．そこで本論文では，異なるリポジトリに記録されたモジュール間に存在するロジカルカップリングを検出する手法を提案する．具体的には，複数のリポジトリで一緒に行われたコミットをグループ化することでリポジトリをまたいだトランザクションを推定し，このトランザクション集合に対する相関ルールを求めることでロジカルカップリングを検出する．評価実験により，リポジトリをまたいだロジカルカップリングが存在することと，提案手法によって一緒に変更すべきモジュールを推薦できることが確認された．

キーワード：ロジカルカップリング，相関ルール，ソフトウェアリポジトリ

Detection of Inter-repository Logical Coupling Based on Transaction Estimation

Yasuhiro Hayase[2,a)] Takashi Nakamura[1,b)] Toshiyuki Amagasa[2,c)] Hiroyuki Kitagawa[2,d)]

Received: May 12, 2014, Accepted: November 10, 2014

Abstract: Logical Coupling, which represents how software modules were changed together during development, is utilized in program comprehension and development support. Existing techniques target detection of logical coupling in a single product, since these techniques use commit transactions recorded in one repository of a version control system. However, changes of modules propagate across product boundaries, because it is common to use external products (e.g., libraries or frameworks) for developing a software product in modern software development. This paper proposes a method to detect logical couplings between software modules stored in different repositories Inter-repository commit transactions, i.e., groups of commits which were performed together, are estimated, then association rules are mined as logical couplings. In an evaluation experiment, we confirmed that inter-repository logical couplings have a potential to correctly recommend modules to be changed together.

Keywords: Logical Coupling, Association Rule, Software Repository

[1] 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 筑波大学システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan

a) hayase@cs.tsukuba.ac.jp
b) tnakamura@kde.cs.tsukuba.ac.jp
c) amagasa@cs.tsukuba.ac.jp
d) kitagawa@cs.tsukuba.ac.jp

1. はじめに

ソフトウェアの開発過程で蓄積される記録には，開発者間でのやりとりや，ソースコードに代表される成果物の変更履歴，欠陥の発見から修正までの履歴などがあり，これらの情報は様々な方法で分析され，ソフトウェア開発支援に利用されている [1], [2]．ソフトウェアの開発履歴には，過去のソフトウェア開発で積み重ねられてきた経験的な知識が含まれているため，有用な知識を抽出し開発者に提供することで，ソフトウェアの開発過程で遭遇する問題に対処しやすくなる．

大規模なソフトウェアプロダクトを開発する際の困難の 1 つに，モジュール間の関係の把握が難しいという問題があり，これを支援する方法が研究されている．ソフトウェアプロダクトのあるモジュールを変更した場合には，関連する別のモジュールに対しても変更やテストの必要が生まれるが，この際に見落しがあると，不具合による損失につながりうる．このようなモジュール間の強い結合はソフトウェアの保守性を低下させるため，強い結び付きの存在を把握することが必要と考えられている [3]．モジュール間の関連を発見する方法は数多く提案されており，ソースコードの構造的な情報を利用するもの [4] や，コメントなどの意味的な情報を利用するもの [5]，プログラムの実行ログを利用するもの [6], [7] などがある．

モジュール間の関連を評価する方法の 1 つに，プログラムの変更履歴に基づいてモジュール間の関連を表す指標ロジカルカップリング [8] がある．ロジカルカップリングは「あるモジュールが変更されたときには，別のあるモジュールもまた一緒に変更されやすい」という関係を表す (図 1)．ここで，「モジュールを一緒に変更する」というフレーズは，「ある 1 つの目的を達成するために，2 つ以上のモジュールを変更する」ことを意味する．ロジカルカップリングは，ソースコードの変更漏れの防止や，ソフトウェアのモジュラリティの評価に利用することができる．また，ロジカルカップリングはモジュールの変更にともなう他のモジュールへの影響を分析する Change Impact Analysis にも利用されている [9], [10]．

一方，ソフトウェア開発におけるモジュール間の影響は，他のソフトウェアプロダクトに及ぶことがある．たとえば，ライブラリやフレームワークを利用することで効率的にソフトウェア開発を行うことは一般的な方法となっているが，ライブラリなどに仕様の変更があった場合には，それを利用しているプロダクトが影響を受ける．別の事例として，特にオープンソースソフトウェア（OSS）において，開発者間で開発目的に違いが大きくなった場合に，プロダクトの開発記録を複製し，新しいプロダクトとして開発を継続する場合（プロダクトの派生）がある．派生後の 2 つのプロダクトは派生前の開発履歴から影響を受けることになる．たとえば，派生前から存在する欠陥は，派生前後の両方のプロダクトで修正されなければならない．さらに別の観点として，1 人の開発者が複数の OSS プロダクトの開発に参加する場合 [11], [12], [13], [14] にも，他のプロダクトからの影響が発生しうる．開発者は，他のプロダクトの開発で得た知識を用いてソフトウェアを開発したり，プロダクト間でコードの再利用を行ったりすることがあるためである [15]．

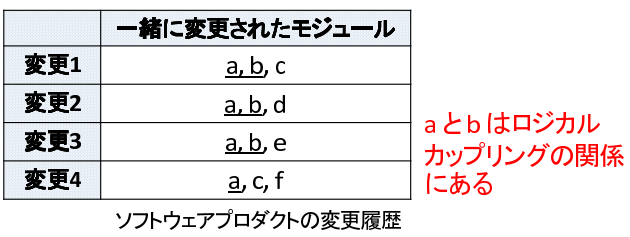

図 1 ロジカルカップリングの例
Fig. 1 Example of logical coupling.

このように，ソフトウェアプロダクト間での影響は様々な理由によって発生するため，異なるプロダクトに含まれるモジュール間にもロジカルカップリングの関係が存在していると推測される．もし異なるプロダクトに含まれるモジュール間でのロジカルカップリングを検出できれば，変更が他のプロダクトに与える影響を推測することができ，保守性をより正確に評価できるようになると期待される．また，他のプロダクトが変更された場合に，開発中のプロダクトが受ける影響を速やかに知ることもできる．しかし，従来のロジカルカップリング検出手法は，単一のプロダクトの開発履歴を分析するものであったため，異なるプロダクト間に存在するロジカルカップリングを検出することはできなかった．

そこで本研究では，異なるプロダクトに含まれるモジュール間に存在するロジカルカップリングを検出する手法を提案する．検出のために，複数のリポジトリで一緒に行われたコミットをグループ化することでリポジトリをまたいだトランザクションを推定し，このトランザクション集合に対する相関ルールを求めることでロジカルカップリングを検出する．さらに評価実験を行い，リポジトリをまたいだロジカルカップリングが実際に存在すること，さらにロジカルカップリングが一緒に変更すべきモジュールの推薦に役立つことを示した．

本論文の構成は次のとおりである．まず，2 章で関連研究を紹介し，本研究で必要となる重要な概念について説明する．次に，3 章でロジカルカップリングの検出手法について説明する．続いて，4 章で評価実験について述べる．そして最後に，5 章で本研究の結論を述べる．

2. 関連研究

2.1 分散型バージョン管理システムと派生プロダクト

バージョン管理システム（VCS）は，ファイルの変更を個別に管理する SCCS [16] や RCS [17] から始まり，次にプロダクトのディレクトリ階層を記録する CVS [18] が生まれ，現在主流となっている分散型 VCS へとつながった．分散型 VCS は複数の開発者がネットワーク経由で変更をやりとりするソフトウェア開発に適合したシステムであり，代表的なものに BitKeeper [19] や Git [20]，mercurial [21] がある．CVS 以前のバージョン管理システムは，変更をファイル単位でのみ記録しており，複数のファイルに対する同時に行われた変更（**コミットトランザクション**）を管理していなかった．本論文では，特に断わらない限りコミットと書いたときにはコミットトランザクションを意味するものとする．

次に，最も広く用いられている分散型 VCS である Git について説明する．Git を用いたソフトウェア開発では，個々の開発者は変更を複製（クローン）されたリポジトリの上で行い，変更が完了すると複製元のリポジトリに変更内容を反映する．Git リポジトリをクローンした場合，派生元のリポジトリに記録されていた過去の変更（コミット）は派生リポジトリにも共通して記録されていることになる（図 **2** (a)）．

Git リポジトリのクローンは，プロダクトの派生版を作る際にも用いられる．プロダクトの派生が行われた後で開発が進むと，それぞれのリポジトリには別々のコミットが蓄積され，履歴として保存される．派生先と派生元のプロダクトの間では，派生が起こった後にも変更内容が共有されることがあるが，その場合には Git の機能を用いてコミットを取り込むことができる（図 2 (b)）．取り込まれたコミットは，2 つのリポジトリに共通して含まれることになる．

2.2 プロダクト内のロジカルカップリング検出手法

ロジカルカップリングは，プロダクトに含まれるモジュール間の結合度を評価する指標として，1998 年に Gall らによって提唱された [8]．ロジカルカップリングはソフトウェア開発の実績に基づく結合度であり，ソフトウェアの開発履歴を分析し，一緒に変更されやすいモジュールの組合せを見つけることで得られる．この研究で Gall らは，プロダクトの変更履歴からモジュール間の結合度を評価するために CAESAR という手法を提案し，ロジカルカップリングの検出を行った．CAESAR は，2 つのプロセスからなる手法である．まず CSA（Change Sequence Analysis）によって，各モジュールの変更履歴を比較し，共通する変更履歴に基づいて，ロジカルカップリングの関係にありそうなモジュールを決定する．次に，CRA（Change Report Analysis）では，ロジカルカップリングの関係にありそうなモジュールの変更時のメッセージを確認する．このようにして確認した変更の動機が同じであることをもって，ロジカルカップリングの関係にあることを確認する．

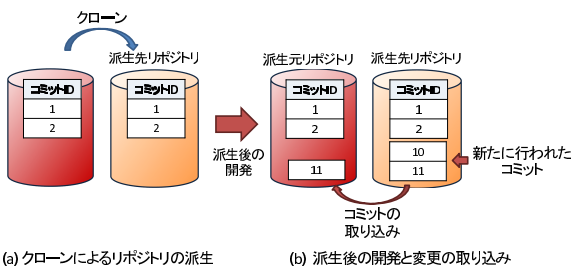

図 **2** クローンによるプロダクトの派生
Fig. 2 Product derivation using repository cloning.

Gall らは続く研究 [22] で，VCS のコミット記録を用いることで細粒度な変更に基づいてロジカルカップリングを検出する方法を提案した．この研究の特徴は，ロジカルカップリング検出における複数のモジュールが一緒に変更された事実として，VCS におけるコミットトランザクションを利用したことにある．VCS に対してコミットを行う際，意味的なつながりを持つ複数のファイルに対する変更を，1 つのコミットトランザクションに含めることが推奨される．この性質を利用し，同一のコミットトランザクションに含まれる変更を一緒に行われた変更と仮定することで，コミットごとという細かいスパンでの変更に基づくロジカルカップリングの検出が可能となった．

この研究では CVS リポジトリを対象としたが，前節で述べたとおり CVS はコミットトランザクションを記録しないため，Gall らは**コミットトランザクションの推定**を行った．推定では，個々のファイルへのコミットを行った開発者の名前が一致しており，コミットが行われた日時が 4 分以内であることを基準として，各ファイルへのコミットをグループ化してまとめた．そして，推定したコミットトランザクション集合の中で，頻繁に同じコミットトランザクションに含まれるモジュールの間にロジカルカップリングがあると判定した．

Zimmermann ら [23] は，モジュールの変更の予測や推薦に利用することを目的として，ロジカルカップリングを相関ルール [24] の形で表現する手法を提案した．相関ルールとは，アイテム集合（トランザクション）を集めたデータベースにおいて，アイテムの出現傾向を表現する規則である．2 つのアイテム集合 X，Y の間に，「X のアイテムすべてがトランザクションに出現するときには，Y のアイテムすべてが同じトランザクションに高い確率で出現する」という関係であり，この関係を $X \Rightarrow Y$ と表記する．左辺は前提部（または条件部）と呼ばれ，右側は帰結部（または結論部）と呼ばれる．相関ルールの性質を表す代表的な指標に，データベース内でルールに含まれるアイテムすべて

が出現する頻度を表す支持度と，前提部が成立したときに帰結部が成立する確率を表す確信度がある．ロジカルカップリングを表現する相関ルールは，「前提部のモジュールが変更されたときには，帰結部のモジュールが高い確率で変更される」ということを意味しているため，前提部のモジュールのみを開発者が変更されたときに，「帰結部のモジュールを変更すべきではないか」という推薦を行うことができる．Zimmermann らは，帰結部のモジュールが 1 つのみに制約されたルールを検出したが，この制約があっても変更すべきモジュールの予測の精度が落ちないことが示されている．

また，この研究の特徴の 1 つに，非常に多くのモジュールにいっせいに行われたコミットを相関ルールの計算に利用しないようにするために，**巨大なコミットトランザクションの除去**を行ったことがある．VCS のリポジトリには，大量の変更を一括で取り入れた場合などに，関係の薄い大量のモジュールにいっせいに変更が行われることがある．ロジカルカップリングの検出はモジュール間の関係を取得することが目的であり，上記のような大規模な変更は目的の妨げとなる．そこで，30 個以上のモジュールを同時に変更したコミットトランザクションを取り除き，相関ルールの検出に用いないようにすることで，ロジカルカップリングの精度を向上させた．

さらにロジカルカップリングの評価として，リポジトリに記録されたコミットを新旧に 2 分割し，旧いコミットから得たロジカルカップリングによって，新しいコミットにおける変更を予測できるかという実験を行った．結果として，予測対象とした変更のうち，最大で 15%の変更を予測することができたと述べている．

2.3 ロジカルカップリングの応用と有用性の評価

D'Ambros ら [25], [26] は，システム内の問題のある箇所を特定するためにロジカルカップリングを利用した．あるモジュールと他のモジュールのロジカルカップリングの関係の強さを視覚的に確認できるように，The Evolution Rader というロジカルカップリングの可視化ツールを作成した．これによって，あるモジュールと強い関連がある他のモジュールを特定することができ，モジュール間の依存関係を少なくする手助けを行った．

また，Wong ら [27] は，ソフトウェアのモジュール構造とロジカルカップリングを比較することで，モジュール構造の違反を検出する研究を行った．モジュール構造的に関係のないモジュールどうしがロジカルカップリングの関係にある場合に，それらのモジュールを構造違反として検出する．そして，2 種類のソフトウェアプロダクトを対象に評価実験を行い，提案手法を用いることで，開発者がリファクタリングを行うよりも早くモジュール構造の違反を検出できることを示した．

Ying ら [28] はロジカルカップリングの検出を行うとともに，出力されたロジカルカップリングを用いた予測が開発者にとって価値のあるものであるかについても検討した．まず，あるモジュールの変更に際して行われると考えられる，他のモジュールの変更を予測した．さらに，精度や再現度といった量的なスコアを測るだけでなく，それらのモジュール間に構文的なつながりがあるかどうかについても調査し，構文的な情報からは分からないモジュール間の関係があることを示した．

Bavota ら [29] は，ロジカルカップリングを含む 4 種類の結合度の指標の有用性を評価するため，2 つの観点で調査を行った．まず，4 種類の結合度に基づくモジュール間の関係がどのような相補関係にあるのかを調査した．結果として，ロジカルカップリングに基づいて検出されるモジュールの多くは，他の指標でも検出されるという結果となった．次に，4 種類の指標に対して，その指標が開発者の直感に則したものであるかどうかを調査した．結果として，ロジカルカップリングは開発者の直感に則したモジュールが検出できる場合があることが確認された．

また D'Ambros ら [30] は，ロジカルカップリングとソフトウェア欠陥との関係について調査を行い，それらの間に関連があることを示した．また，ロジカルカップリングを用いたバグ予測精度の改善についても検討を行った．

2.4 複数のリポジトリをまたいだ変更波及に関する研究

Fischer らは，複数のリポジトリを対象として変更の波及を調査した [31]．この調査では，共通の祖先を持つ 3 つのプロダクト（FreeBSD，NetBSD，OpenBSD）に対して，あるプロダクトの変更が他のプロダクトにどう影響を与えているかを吟味している．まず，各プロダクトの名称をキーワードとし，各モジュールごとに，モジュールを変更したコミットのコメントにキーワードが含まれている件数をカウントした．そして，コメントにキーワードが含まれているコミットが多かったモジュールを，他のプロダクトと関連の深いモジュールとして検出した．この調査により，あるリポジトリで行われた変更が，別のあるリポジトリに対して影響を与えていることが定量的に明らかになった．

3. 提案手法

本章では，分散型 VCS である Git のリポジトリ集合を対象として，異なるリポジトリをまたいだロジカルカップリングを検出し，相関ルールの形で検出する手法を提案する．1 章と 2.4 節で述べたように，ロジカルカップリングは単一のプロダクトに含まれるモジュール間に存在するだけでなく，異なるプロダクトに含まれるモジュールの間にも存在する（リポジトリをまたいだロジカルカップリングがある）と考えられる．しかし，多くの場合，プロダクトごとに VCS のリポジトリは異なるため，既存手法では単

一のプロダクト内のモジュール間でしかロジカルカップリングを検出できなかった．提案手法を用いることで，ロジカルカップリングを利用した様々な開発支援手法を，複数のプロダクト間で適用できるようになると期待される．

異なるリポジトリをまたいだロジカルカップリングを検出するには，異なるプロダクトに含まれるモジュールが一緒に変更された事実を集める必要がある．現在の多くの VCS はどのモジュールが同時に変更されたかをコミットトランザクションとして記録しているため，単一プロダクト内のロジカルカップリング検出に必要な情報は容易に得ることができる．一方，異なるプロダクトに含まれるモジュールは異なるリポジトリに記録されているため，これらのモジュールがいつ同時に変更されたかを容易に知ることはできない．

この課題を解決するために，一緒に行われたコミットトランザクションのグループである**インターリポジトリコミットトランザクション（IRCT）**を導入し，これを相関ルール検出におけるトランザクションとして利用する．異なるリポジトリに含まれるモジュール間では，一緒に変更されたという情報はリポジトリなどに明示的には記録されていないので，推定を行う必要がある．提案手法では，Gall らによるコミットトランザクションの推定方法 [22] を参考にして，コミットを行った開発者と日時に基づいた IRCT の推定方法を策定する．この推定方法には解決すべき問題が 1 つある．2.1 節で述べたとおり，分散型 VCS のリポジトリには同一のコミットが複製された結果，重複して記録されている場合がある．3.2 節で詳説するが，重複したコミットはリポジトリをまたいだロジカルカップリングの計算に悪影響を与えるため，この影響を取り除くために相関ルール検出の手順に工夫を加える．

提案手法の全体像を**図 3** に示す．まず，複数のリポジトリから，各リポジトリで行われたコミットを収集する．次に，収集したコミット集合から IRCT を推定する．この推定の際に，Zimmermann ら [23] がコミットトランザクションに対して行ったのと同様に，多くのモジュールが含まれる巨大な IRCT を除去する．最後に，得られた IRCT 集合をトランザクションデータとして相関ルール検出を行うことで，異なるリポジトリに含まれる 2 つのモジュール間でのロジカルカップリングを計算する．

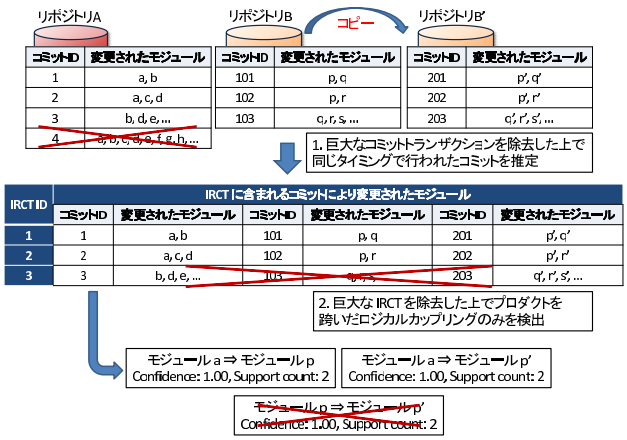

図 3 手法の流れ

Fig. 3 Overview of proposing method.

以下，3.1 節で IRCT の推定の方法を，3.2 節で推定によって得られた IRCT からロジカルカップリングを検出する方法について詳説する．

3.1 IRCT の推定

IRCT の推定では，複数の Git リポジトリから収集した大量のコミットトランザクションを入力として，一緒に行われたと思われるコミットの集合を出力する．推定方法の基本的なアイデアは「同一の開発者によって近い日時に行われた変更は，同じ目的のために行われた可能性が高い」というものであり，この考えは Gall らのコミットの推定方法 [22] に基づいている．非分散型の VCS とは異なり，Git リポジトリには，プロダクト開発の核となる開発者以外による変更であってもコミットが細粒度かつ正確に記録されるという特徴があるため，複数のリポジトリをまたいだ変更の推定に必要な情報が入手できる．また，大量のモジュールが含まれる巨大な IRCT は，後段の相関ルール検出の邪魔となるため，これを除去する．

IRCT の推定基準を検討する．Gall らは単一のリポジトリ内で行われた細粒度のコミットを取り出すことを目的としていたため，日時の近さの基準に 4 分という閾値を用い，閾値よりも短い時間に同一の開発者によって行われたコミットを推移的にグループ化することでトランザクションを推定していた．IRCT は複数のリポジトリをまたいだ変更であるため，一連の変更作業には 4 分より長い時間を要する場合が多いと想像される．一方で，長い閾値で推移的にコミットをグループ化したとすると，連鎖的にコミットがグループ化され，日時が大きく離れた関連の薄いコミットまでが，1 つの IRCT に含まれると推定される恐れがある．

そこで，同一開発者によるコミットを，一定時間幅のウィンドウによって切り分け，同一のウィンドウに属したコミットを IRCT として推定することとする．具体的には，コミットに記録されたユーザ名とメールアドレスが同一であり，UTC で表したコミット日時の日付部分が同一であることを基準とした（**図 4**）．

この推定基準によって起こりうる誤りは，本来は一緒に行ったコミットを別の IRCT に分類してしまう場合（第 2 種過誤）と，関係しないコミットを同一の IRCT に含めてしまう場合（第 1 種過誤）の 2 種類がある．第 1 種過誤が起こる原因は，一緒に行ったコミットが日付をまたいでしまうことである．第 2 種過誤によってあるロジカルカップリングを検出することができなくなるのは，ロジカルカッ

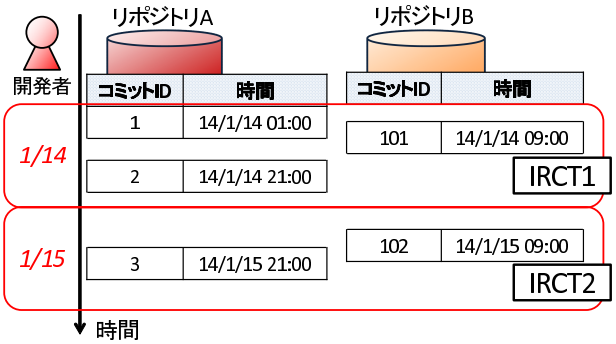

図 4 IRCT の推定
Fig. 4 Inferring IRCT.

プリングの関係にある 2 つのモジュールが一緒に変更される際に，それぞれへの変更が UTC の午前 0 時を越えて 2 つのコミットに分かれる強い傾向を持っている場合に限られる．しかし，現実にはあるモジュールの組への変更が，特定の時刻またぐ理由を持つことは稀であると予想される．また，もしそのような場合があるのであれば，ウィンドウ切り分けの条件を変更する（たとえば切り分けの時刻を午前 1 時にする）ことで見落としをなくすことができる．第 1 種過誤は，1 人の開発者が，近い時刻に関係のない複数の目的で開発を行い，変更をコミットすることによって起こる．この複数の目的間には関係がないため，関係しない変更が同一の IRCT に含まれるかどうかは，偶然によって決まることになる．相関ルール検出は多くの偶然に発生する事象の中から，関連して発生する事象を見出すように設計されているため，第 1 種過誤の影響は相関ルール検出とその閾値設定により緩和することができる．

大量のモジュールが含まれる IRCT は相関ルール計算に悪影響をおよぼすため，巨大な IRCT を除去する．ただし，巨大な IRCT には 2 種類のものが存在すると考えられる．1 種類目は巨大なコミットを含む IRCT であり，2 種類目は大量のコミットを含む IRCT である．前者については，IRCT を推定する前に，大量のモジュールを含むコミットを削除することで対処する [23]．後者については，IRCT を推定した後に，各 IRCT のすべてのコミットに含まれるモジュールを重複なく数え上げ，その数が閾値以上である IRCT を削除する．

3.2 IRCT を利用したロジカルカップリング検出

3.1 節で推定した IRCT をトランザクションとして，複製されたコミットを考慮した相関ルール検出によってロジカルカップリングを検出する．2.1 節で述べたとおり，分散型 VCS では，プロダクトの派生や変更の取り込みが起こると，複数のリポジトリに同一のコミットが複製された結果として，重複して記録される．重複したコミットは，3.1 節で定義した IRCT 推定基準によって同一の IRCT としてグループ化され，本手順の入力となる．重複したコミットは一見すると，2 つ（またはそれ以上）のリポジトリで同一開発者が同一時刻に同名のモジュールを変更したように見える．そのため，もし素朴に相関ルール検出を行うと，この同名のモジュール間に強いロジカルカップリングが検出されてしまう．しかし，実際には開発者は 1 つのプロダクトに対して変更を行ったのみであり，「異なるプロダクト（リポジトリ）に含まれるモジュールを，一緒に変更した」とはいえない．

コミットが重複したものかどうかの判別には，Git が管理するコミットのハッシュ値を比較すればよい．このハッシュ値は暗号的ハッシュ関数によって計算されるため，偶然に一致する確率は無視できるほど低い．そのため，2 つのリポジトリに同一ハッシュ値のコミットトランザクションが記録されている場合，どちらかのリポジトリ，もしくはどちらでもない別のリポジトリで行われた 1 つのコミットが，複製により重複したものであると断定して実用上差し支えない．

重複したコミットへの対処として考えられる単純な方法は，重複のうち 1 つだけを残し他を削除する方法であるが，これには問題がある．たとえばプロダクトの派生を考えると，派生後の両プロダクトは個別に開発されてゆくためそれぞれロジカルカップリングを求める必要があるが，その開発は派生よりも前の開発からも影響を受ける．変更の取り込みの場合でも，その変更が 2 つのプロダクトの個々のモジュールに反映されたことは事実である．どちらの場合でも，複製されたコミットの 1 つだけを残す場合，どちらかのリポジトリでの変更記録が失われ，ロジカルカップリングを見逃すことにつながる．

そこで，この問題を解決するために，IRCT で 2 つのモジュールが同時に変更されたという関係から，重複したコミットで変更されたという関係のみを削除して，2 つのモジュール間での相関ルールを求める方法を提案する．得られる相関ルールは，前提部と帰結部にそれぞれ 1 つずつモジュールを含むものに制限されるが，制限されたルールであってもモジュール間の結合度を評価するのに役立てることができる．

重複したコミットの影響を排除するために，ある IRCT t の中で変更された 2 つのモジュール x, y について，「IRCT t の中でモジュール x がモジュール y から独立して変更された」という関係を導入する．この関係は，IRCT t_1 の中で，モジュール x を変更したすべてのコミットのハッシュ値の集合から，モジュール y を変更したすべてのコミットのハッシュ値の集合を除いたときに，空集合にならない場合と定義される．具体例を，**図 5** に示す．IRCT t_1 は 4 つのコミットからなるが，そのうち左に描かれた 2 つは重複したコミットである．ここでリポジトリ A のモジュール p とリポジトリ B のモジュール q（それぞれ A.p，B.q と表記する）に着目する．A.p を変更したコミッ

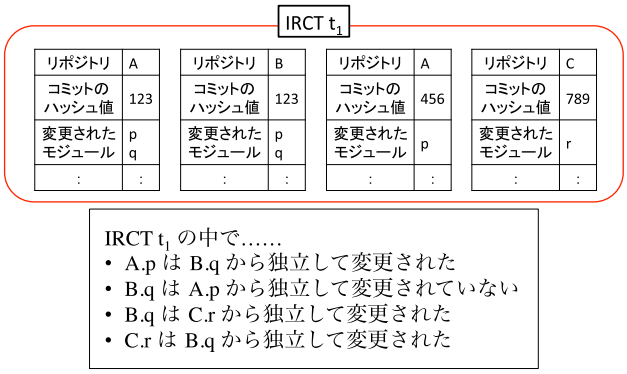

図 5 独立して変更された例と，そうでない変更の例
Fig. 5 Examples of independent or dependent changes.

トは 2 つで，そのハッシュ値の集合は $\{123, 456\}$ となる．一方 B.q を変更したコミットの集合は $\{123\}$ となる．ここで，$\{123, 456\} \setminus \{123\} = \{456\}$ であるので，IRCT t_1 の中で A.p は B.q から独立して変更されたといえる．一方，$\{123\} \setminus \{123, 456\} = \emptyset$ であるので，IRCT t_1 の中で B.q は A.p から独立して変更されていないことになる．

次に，異なるリポジトリの属する 2 つのモジュール x，y に対して，x が y とは独立して変更された IRCT の集合を $\mathrm{isup}(x, y)$，モジュール x，y が相互に独立して変更された IRCT の集合を $\mathrm{ssup}(x, y)$ と定義する集合 $\mathrm{ssup}(x, y)$ の意図は，リポジトリをまたいで x と y が一緒に変更されたという事実の集合を近似することである．$\mathrm{isup}(x, y)$，$\mathrm{ssup}(x, y)$ は，IRCT 全体の集合を T，コミットを c_n，コミットのハッシュ値を $c_n.\mathrm{id}$ とした場合，次の式で表される．

$$\mathrm{isup}(x, y) = \{t \in T \mid \exists c_1 \in t\ \forall c_2 \in t \\ (x \in c_1 \wedge y \in c_2 \wedge c_1.\mathrm{id} \neq c_2.\mathrm{id})\}$$

$$\mathrm{ssup}(x, y) = \mathrm{isup}(x, y) \cap \mathrm{isup}(y, x)$$

図 5 の例で，ssup の条件を満たす場合と満たさない場合を考えてみる．IRCT t_1 で，A.p は B.q から独立して変更されているものの，B.q は A.p から独立して変更されていないため，$X \notin \mathrm{ssup}(\mathrm{A.p}, \mathrm{B.q})$ となる．一方で，B.q は C.r から独立して変更されており，C.r もまた B.q から独立して変更されているため，$X \in \mathrm{ssup}(\mathrm{B.q}, \mathrm{C.r})$ となる．

最後に，isup と ssup を用いて，2 つのモジュール間の相関ルールに対する支持度と確信度を再定義する．これらの支持度 support_count と，確信度 confidence が定めた閾値を超える相関ルールをロジカルカップリングとして検出する．support_count と confidence 定義は下のとおりである．

$$\mathrm{support\_count}(x \Rightarrow y) = |\mathrm{ssup}(x, y)|$$

$$\mathrm{confidence}(x \Rightarrow y) = \frac{|\mathrm{ssup}(x, y)|}{|\mathrm{isup}(x, y)|}$$

この定義では，支持度は「2 つのモジュール x，y が双方から独立に変更された事例の数」となっている．同様に，確信度は「あるモジュール x が別のモジュール y と独立に変更されたときに，y もまた x とは独立に変更される確率」となっている．

4. 評価実験

提案手法によって得られるロジカルカップリングが，リポジトリをまたいで一緒に行われる変更を予測できるかを評価するために，実験を行った．実験では，モジュールの単位としてファイルを採用する．また，検出されたロジカルカップリングのいくつかの事例について調査する．4.1 節で評価基準の定義とその目的を述べ，4.2 節で利用したデータを，4.3 節で実験における各種条件について説明する．評価結果を 4.4 節で示す．ロジカルカップリングの事例調査について 4.5 節で述べ，最後に 4.6 節で結果を考察する．

4.1 評価基準

実験対象のコミットが行われた日時はある期間にわたっているが，ある日時を境界として期間を前半（**学習期間**）と後半（**評価期間**）の 2 つに分割し，学習期間のコミットから得たロジカルカップリングによって，評価期間の IRCT に含まれる変更を予測できるかを評価する．ただし，IRCT は推定によって得るものであり，IRCT 内に含まれるファイル変更は，必ずしも 1 つの目的で行われたとは限らない．（3.1 節で述べた第 1 種過誤）そこで，相関ルール検出における支持度および確信度に対する閾値と類似した考え方に基づいて，偶然に行われた可能性の高い変更を除去する．

まず，ロジカルカップリングによって，後半の期間に 1 度でも変更されたファイルに対して，同時に変更すべき可能性のあるファイル集合（予測集合）を算出する．最初に，学習期間のコミットから，Support Count と Confidence が設定したある閾値を超える相関ルールを求める．次に，評価期間のすべてのコミットから，1 回以上変更されたファイル（クエリファイル）を取り出す．各クエリファイルに対して，前提部がクエリファイルとなるルールをすべて取り出し，取り出したルール集合の帰結部を集めた集合が，クエリファイルに対する予測集合となる．

次に，各クエリファイルに対する正解を定義する．まず，評価期間のコミット集合から，IRCT を算出する．そして，クエリファイルが変更されている IRCT を取り出し，それぞれの IRCT でクエリファイル以外の変更されているファイルを取り出し，重複を排除して集合とする．クエリに対応する各 IRCT から変更されたファイルの集合は 1 つずつ得られるので，これらを重複を含めて和を求めて多重集合を作る．この多重集合から，重複度が閾値未満の要素を除去したものが，クエリファイルに対する正解多重集合となる．この閾値を変化させながら，評価結果がどのように変化するかを確認する．

予測集合と正解多重集合がどの程度一致しているのかを評価する評価基準には適合率（Precision）と，再現率（Recall）を用いる．適合率は，予測集合のうち正解多重集合に含まれていた要素（ファイル）の割合であり，これは一般的な定義と同様である．一方，再現率は，正解多重集合のうち予測結果に含まれていた要素を，重複を含めて求めた割合と定義する．具体例として，予測集合が $\{a, b\}$，正解多重集合が $\{a, a, c\}$ であった場合を考える．予測集合のうち，正解多重集合に含まれるのは a であり，含まれないのは b であるので，適合率は 1/2 となる．一方，正解多重集合のうち，予測集合に含まれるのは a であり，含まれないのは c であるが，a は正解多重集合に 2 つ重複しているため，再現率は 2/3 となる．この再現率の定義は，重複度の閾値が 1 である（すなわち，要素を除去しない）場合には，多重集合を構成する元となった各 IRCT から構成した集合に対して予測を行い，その再現率をマイクロ平均した値と同一になるように設計されている．

最後に，すべてのクエリファイルに対して適合率と再現率を算出し，それぞれのマクロ平均を求める．この適合率と再現率のマクロ平均を，手法を評価する指標とする．ただし，予測集合もしくは正解多重集合の要素数が 0 の場合には，それぞれ適合率と再現率が定義できないため，平均の算出からは除く．

4.2 利用したデータ

GitHub において watch と呼ばれる機能でリポジトリの動向を注目しているユーザが多い順に，10,000 件の Git リポジトリを収集した．10,000 件のリポジトリを決定した時期は 2011 年 9 月であり，収集は 2011 年 9 月から同年 12 月までの期間で行った．収集の対象は各リポジトリの master ブランチであり，収集を行った時点での master ブランチのすべてのコミットを取得した．なお，10,000 件のリポジトリから得られたコミットは 7,484,874 件であり，コミットによって 1 度以上変更されたファイルの数は 8,091,772 個であった．期間全体で 1 つ以上のコミットを行った開発者は 87,359 人であった．**表 1** に，開発者のコミットが含まれていたリポジトリの数を集計した結果を示す．2 つ以上のリポジトリにコミットが含まれていた開発者は 30,472 人であり，開発者全体の約 35%であった．評価実験における学習期間は 2010 年末まで，評価期間は 2011 年初頭以降とする．

4.3 各種条件

ロジカルカップリング検出に用いたパラメータについて説明する．まず単一の巨大なコミットの除去では，11 個以上のファイルを変更したコミットを削除した．そのうえで，IRCT 中で 31 個以上のファイルを変更した IRCT を除去した．除去された単一のコミットと IRCT の割合は，それぞれ全体の 5%と 2%にあたる．**図 6** と**図 7** に，コミットおよび IRCT によって変更されたファイル数の分布を示す．なお，参考として，研究 [23] では，単一のコミットが 30 個以上のファイルを変更した場合に，そのコミットを削除していた．

また，学習期間における相関ルール検出の際の条件として，最小支持度（以下，minsup）を 1，3，5，10 と変化させ，最小確信度（以下，minconf）を 0 から 1 まで 0.1 刻

表 1 開発者のコミットが含まれていたリポジトリ数の集計
Table 1 #developers whose commits are included in repositories.

リポジトリ数	開発者数
1	56,887
2〜3	14,939
4〜7	13,990
8〜15	1,202
16〜31	291
32〜63	43
64〜83	7
計	87,359

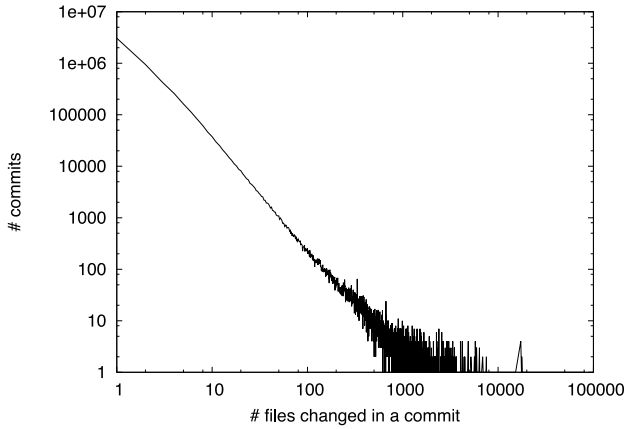

図 6 コミットによって変更されたファイル数とその出現回数の分布
Fig. 6 Distribution of #files changed in a commit.

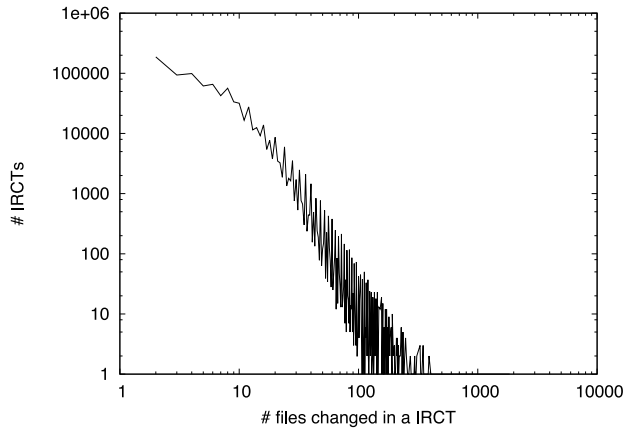

図 7 IRCT に含まれるコミットによって変更されたファイル数とその出現回数の分布
Fig. 7 Distribution of #files changed in a IRCT.

みで 11 段階に変化させて，相関ルールを算出した．支持度と確信度の閾値を変化させると，得られるロジカルカップリングの集合も異なるものになる．正解多重集合の作成時における多重度の閾値（4.1 節で定義．以下，minmult）も，1，3，5，10 と変化させ，複数の正解多重集合を得る．

4.4 結果

定義した評価基準と，前述のデータセットから得た結果を用いて，評価を行った結果について述べる．パラメータ minsup と minconf が変化することで検出されるロジカルカップリング集合が変化し，minmult が変化することで正解多重集合が変化するため，このロジカルカップリング集合と正解多重集合の組合せに対して，適合率と再現率のマクロ平均を求める．以下，適合率と再現率のマクロ平均を，それぞれ単に適合率と再現率と記す．さらに，これらのパラメータが変化することによって，評価指標がどのように変化するのかについて議論する．

図 8，**図 9**，**図 10**，**図 11** に，minmult がそれぞれ 1，3，5，10 である場合の評価結果を示す．グラフの縦軸は適合率を，横軸は再現率を表す．グラフ中の点は，minsup と minconf がある値であったときの適合率と再現率に対応しており，折れ線は同じ minsup の値の点を，minconf 値の順につないだものである．折れ線を見ることで，minconf が 0.0 から 1.0 まで変化するときに指標値がどう変化するかが分かる．

4 つのグラフ全体に共通する傾向を説明する．まず，minconf が低いほど再現率が高く，さらに minconf が 0.2 から 0.4 の範囲で適合率が最大となる．一方で，minsup は低いほど再現率が高くなるが，minconf の値が低い一部の領域を除いて minsup が低くても適合率はあまり低下しない．minmult は高い値であるほど，すなわち，評価期間に頻繁に一緒に変更されたファイルに限定すればするほど，適合率と再現率がともに高まることが分かる．

適合率が最も高かったパラメータは，minmult を 10 として正解多重集合を作り，minsup を 1，minconf を 0.4 とした場合であり，適合率が 0.49，再現率が 0.40 となった．このときの F1 スコアは 0.44 である．

最後に，今回の結果を既存研究と比較する．しかし，複数リポジトリをまたいだロジカルカップリングを検出した

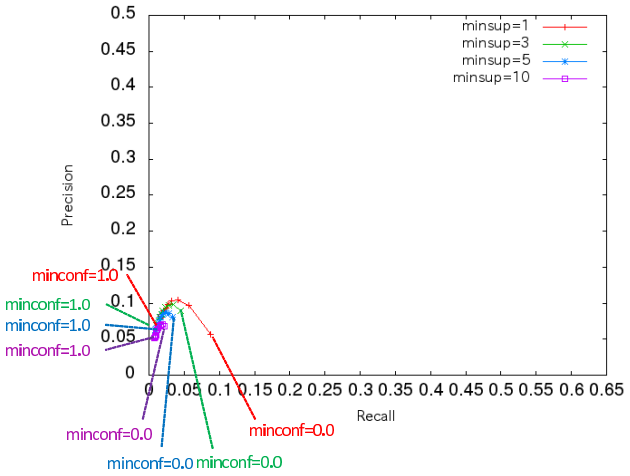

図 8 minmult = 1 の正解多重集合に対する適合率–再現率グラフ
Fig. 8 Precision-recall graph when minmult = 1.

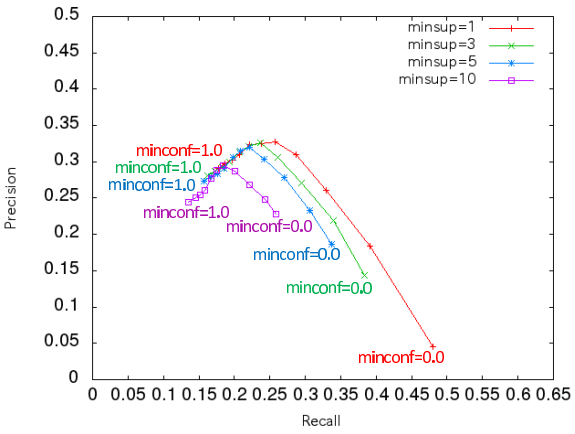

図 10 minmult = 5 の正解多重集合に対する適合率–再現率グラフ
Fig. 10 Precision-recall graph when minmult = 5.

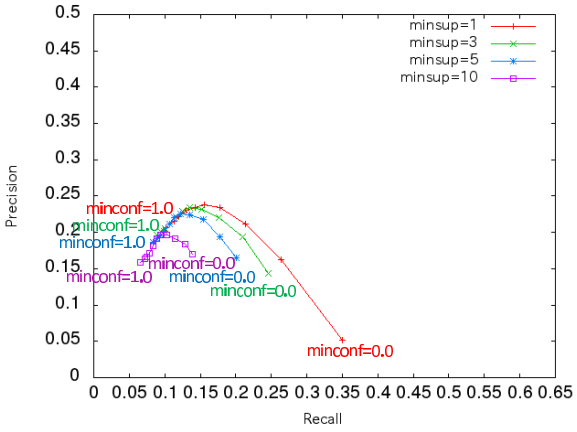

図 9 minmult = 3 の正解多重集合に対する適合率–再現率グラフ
Fig. 9 Precision-recall graph when minmult = 3.

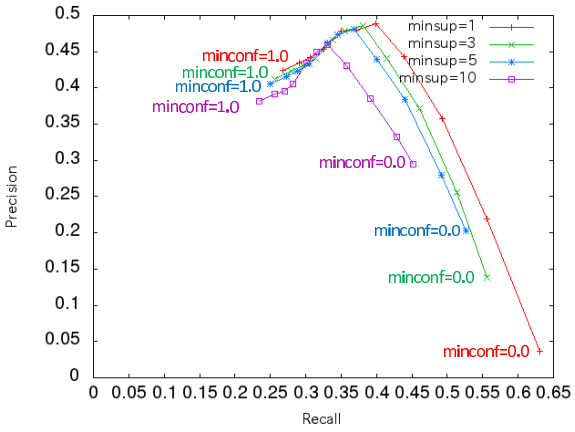

図 11 minmult = 10 の正解多重集合に対する適合率–再現率グラフ
Fig. 11 Precision-recall graph when minmult = 10.

先行研究は存在しないため，比較対象として Zimmermann らによる単一リポジトリ内でのロジカルカップリング検出手法 [23] を選んだ．**表 2** に Zimmermann らの手法の評価と，本評価実験の結果とを示す．既存手法の評価は適合率の定義が大きく異なるため値を比較できないが，再現率は近い定義を用いているため，再現率が最も良かった場合に着目して比較することにする．提案手法を minmult=1 の正解多重集合に対して用いた場合には，提案手法は既存手法よりも劣る結果となった．しかし，minmult=10 として，正解多重集合を同時に変更されることが多いファイルの関係に絞ったところ，再現率は 0.63 となり既存手法を大きく上回る．

4.5 ロジカルカップリングの事例調査

本節では，検出されたロジカルカップリングを目視で調査し，筆者が興味深いと考えた事例を報告する．まず，任意の 2 つのリポジトリ r_1，r_2 について，r_1 と r_2 を所有する github のユーザが異なり，かつ，r_1 に属するファイルから r_2 に属するファイルへ，確信度が 0.5 より大きい相関ルール（ロジカルカップリング）が 10 個以上存在する事例を抽出した．条件を満たすリポジトリ (r_1, r_2) の 2 個組は 1,664 個存在した．ここから，同一プロジェクトのミラーのためロジカルカップリングが存在する理由が自明である組を除く．さらに残った組から，著者がロジカルカップリングが存在する理由を理解できたリポジトリを 7 組抽出した．**表 3** に，抽出したリポジトリの組と，その間に存在するロジカルカップリングの数を示す．以下で，4 つのグループに分類し，その意味を説明する．

グループ 1：分岐したプロジェクト SingularityCore と TrinityCore はオンラインゲームの一種である MMORPG を開発するためのフレームワークである．SingularityCore は TrinityCore から分岐・派生した．この 2 つのプロジェクトはソースコードの多く共有しながら開発を進めているため，ロジカルカップリングが多数存在する結果となった．

グループ 2：移植されたソフトウェア mac-zfs は Mac OS X 上でファイルシステム ZFS を利用するためのソフトウェアである．また，illumos-gate は OpenSolaris の kernel から派生した実質的な後継である．ZFS は Solaris を起源としており，illumos-gate の ZFS に関するソースコードと，mac-zfs とが連携して開発を進めていることが分かる．

グループ 3：基盤とそれを利用するソフトウェア グループ 3 には，リポジトリの組が 3 つある．

parrot は動的言語を実行するための汎用仮想マシンであり，rakudo と cardinal は parrot 上でそれぞれ Perl 言語と Ruby 言語を動作させるための実装である．

一方，OAuth2Client は Mac OS X や iOS 上で，OAuth 2.0 を実現するためのライブラリである．一方 soundcloud ユーザの cocoa-api-wrapper リポジトリは，クラウドアプリケーション SoundCloud を Mac OS X や iOS 上で利用するためのライブラリである．SoundCloud の利用に OAuth 2.0 が必要になるため，cocoa-api-wrapper は OAuth2Client を利用している．

3 つの組のどれもが，基盤となるソフトウェア（仮想マシンやライブラリ）と，それを利用するソフトウェアという関連であり，その結び付きに起因するロジカルカップリングが発見された．

グループ 4：仕様とそれに対応する複数の実装 rubyspec は，Ruby 言語の仕様の曖昧な部分を，テストコードの形で書き下すことで明確化することを目的としたプロジェクトである．一方，rubinius と jruby はどちらも Ruby 言語の処理系であり，rubyspec を用いてテストされている．仕様と実装とが，互いに影響しながら修正されているため，ここにロジカルカップリングが存在したものと考えられる．

表 2 既存研究との再現率の比較
Table 2 Comparision of recall.

	既存手法 [23]	提案手法 minmult = 1	提案手法 minmult = 10
再現率の最大値	0.15	0.09	0.63
適合率の最大値（参考）	0.01 以下	0.04	0.40

表 3 ロジカルカップリングが多数存在したリポジトリの事例
Table 3 Cases of repositories which involve many IRCTs.

リポジトリ 1		リポジトリ 2		ロジカルカップリングの数		
所有者名	リポジトリ名	所有者名	リポジトリ名	1 → 2	2 → 1	グループ
TrinityCore	TrinityCore	SingularityCore	Singularity	13,569	13,502	1
illumos	illumos-gate	dustin	mac-zfs	552	719	2
parrot	parrot	rakudo	rakudo	652	409	3
parrot	parrot	cardinal	cardinal	695	720	3
nxtbgthng	OAuth2Client	soundcloud	cocoa-api-wrapper	47	70	3
rubyspec	rubyspec	jruby	jruby	844	727	4
rubyspec	rubyspec	rubinius	rubinius	3,022	3,011	4

4.6 考察

実験では，頻繁に一緒に変更される minmult が 10 の場合のファイルの関係については，相関ルール検出のパラメータを調整することで，適合率が 0.49 再現率が 0.40 と，高い精度を変更を予測することができた．このことから，提案手法による変更予測は，開発者による変更の支援として利用できる可能性が十分にあるものと考える．

また，このことは，異なるプロダクトに含まれるファイル間に，ロジカルカップリングが存在することを示すことができたと解釈できる．今回の実験では，ロジカルカップリングがなぜ生じているのかまでは調べていないため，今後はプロダクトをまたいだロジカルカップリングが生じる理由を明らかにする必要がある．

評価において，minconf を 1.0 に近い高い値とした場合に，適合率と再現率がともに下がる現象が確認された．一般的には適合率と再現率はトレードオフの関係にあり，片方が下がればもう一方が高まることが多い．本実験で適合率と再現率の両方が低くなってしまった原因としては，minconf を高くしすぎたために，クエリファイルが前提部となる相関ルールが 1 つも存在しなくなる場合が増えてしまい，予測集合が空になってしまった場合があるのではないかと考えられる．また，minsup が高い場合に適合率と再現率がともに下がることも同じ原因で説明が可能である．どのような minsup と minconf が良い予測につながるかを，今後，さらに調査する必要がある．

5. まとめと今後の課題

本論文では，ソフトウェアリポジトリをまたいだロジカルカップリングを検出する手法を提案した．検出のために，一緒に行われたコミットトランザクションのグループ IRCT という概念を提案し，複数のリポジトリに分散して記録されているコミットを統合することで，IRCT の推定する手法を示した．推定した IRCT に基づいて，相関ルールの形でロジカルカップリングの検出を行った．また，提案手法の評価として，提案手法によって過去のデータから得たロジカルカップリングにより，未来の変更を予測できるかを評価した．結果として，頻繁に一緒に変更されるような関係にあるモジュールについては，高い精度で変更を予測できることが分かった．

今後の課題としては，変更の予測に役立ったロジカルカップリングと，役立たなかったものと違いを吟味して，予測精度をより改善することが考えられる．そのためには，ロジカルカップリングが生じたソフトウェア開発上の理由や原因を調査することが有用であると考えている．研究の発展として，IRCT 推定の方法をより洗練させることも考えられる．本論文では，日付と開発者名によって IRCT を推定したが，1 日という間隔や境界の設定には工夫の余地がある．さらに，一緒に行われる変更は，必ずしも 1 人の開発者によって行われるとは限らない．これについては，開発者名以外の情報，たとえばコミットのコメントや，変更差分の内容を利用することで，複数の開発者による IRCT を推定することが考えられる．

謝辞 本研究は JSPS 科研費 25730036 の助成を受けたものです．

参考文献

[1] Fischer, M., Pinzger, M. and Gall, H.: Populating a Release History Database from Version Control and Bug Tracking Systems, *Proc. International Conference on Software Maintenance* (2003).

[2] German, D.M., Rigby, P.C. and Storey, M.-A.: Using Evolutionary Annotations from Change Logs to Enhance Program Comprehension, *Proc. International Workshop on Mining Software Repositories* (2006).

[3] Du Bois, B., Demeyer, S. and Verelst, J.: Refactoring — Improving Coupling and Cohesion of Existing Code, *Proc. 11th Working Conference on Reverse Engineering*, *WCRE '04*, Washington, DC, USA, IEEE Computer Society, pp.144–151 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1038267.1039046⟩ (2004).

[4] Briand, L.C., Daly, J.W. and Wüst, J.K.: A Unified Framework for Coupling Measurement in Object-Oriented Systems, *IEEE Trans. Softw. Eng.* (1999).

[5] Poshyvanyk, D. and Marcus, A.: The Conceptual Coupling Metrics for Object-Oriented Systems, *Proc. International Conference on Software Maintenance* (2006).

[6] Arisholm, E., Briand, L.C. and F., A.: Dynamic Coupling Measurement for Object-Oriented Software, *IEEE Trans. Softw. Eng.* (2004).

[7] Livshits, B. and Zimmermann, T.: DynaMine: Finding Common Error Patterns by Mining Software Revision Histories, *Proc. European Software Engineering Conference Held Jointly with ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2005).

[8] Gall, H., Hajek, K. and Jazayeri, M.: Detection of Logical Coupling Based on Product Release History, *Proc. International Conference on Software Maintenance* (1998).

[9] Wong, S. and Cai, Y.: Generalizing Evolutionary Coupling with Stochastic Dependencies, *Proc. International Conference on Automated Software Engineering* (2011).

[10] Gethers, M., Dit, B., Kagdi, H. and Poshyvanyk, D.: Integrated Impact Analysis for Managing Software Changes, *Proc. International Conference on Software Engineering* (2012).

[11] Ohira, M., Ohsugi, N., Ohoka, T. and Matsumoto, K.-I.: Accelerating Cross-project Knowledge Collaboration Using Collaborative Filtering and Social Networks, *Proc. International Workshop on Mining Software Repositories* (2005).

[12] Nakakoji, K., Yamamoto, Y. and Ye, Y.: Supporting Software Development as Knowledge Community Evolution, *Proc. ACM CSCW Workshop Supporting the Social Side of Large-Scale Software Development* (2006).

[13] Surian, D., Liu, N., Lo, D., Tong, H., Lim, E.-P. and Faloutsos, C.: Recommending People in Developers' Collaboration Network, *Proc. Working Conference on Reverse Engineering* (2011).

[14] 中村高士，早瀬康裕，北川博之：プロジェクト横断的なオープンソースソフトウェア開発記録の分析手法，情報処理学会 第 74 回全国大会 (2012).
[15] Kashima, Y., Hayase, Y., Yoshida, N., Manabe, Y. and Inoue, K.: An Investigation into the Impact of Software Licenses on Copy-and-paste Reuse among OSS Projects, *WCRE*, Pinzger, M., Poshyvanyk, D. and Buckley, J., (Eds.), pp.28–32, IEEE Computer Society (2011).
[16] Rochkind, M.J.: The source code control system, *IEEE Trans. Softw. Eng.*, Vol.SE-1, No.4, pp.364–370 (1975).
[17] Tichy, W.F.: RCS&Mdash;a System for Version Control, *Softw. Pract. Exper.*, Vol.15, No.7, pp.637–654 (online), DOI: 10.1002/spe.4380150703 (1985).
[18] CVS, available from ⟨http://www.nongnu.org/cvs/⟩, (accessed 2014-01-12).
[19] BitKeeper, available from ⟨http://www.bitkeeper.com/⟩.
[20] Git, available from ⟨http://git-scm.com/⟩ (accessed 2014-01-12).
[21] Mercurial SCM, available from ⟨http://mercurial.selenic.com/⟩. (accessed 2014-01-12).
[22] Gall, H., Jazayeri, M. and Krajewski, J.: CVS Release History Data for Detecting Logical Couplings, *Proc. International Workshop on Principles of Software Evolution* (2003).
[23] Zimmermann, T., Weißgerber, P., Diehl, S. and Zeller, A.: Mining Version Histories to Guide Software Changes, *Proc. International Conference on Software Engineering* (2004).
[24] Agrawal, R., Imieliński, T. and Swami, A.: Mining Association Rules Between Sets of Items in Large Databases, *Proc. ACM SIGMOD International Conference on Management of Data* (1993).
[25] D'Ambros, M. and Lanza, M.: Reverse Engineering with Logical Coupling, *Proc. Working Conference on Reverse Engineering* (2006).
[26] D'Ambros, M., Lanza, M. and Lungu, M.: Visualizing Co-Change Information with the Evolution Radar, *IEEE Trans. Softw. Eng.* (2009).
[27] Wong, S., Cai, Y., Kim, M. and Dalton, M.: Detecting Software Modularity Violations, *Proc. International Conference on Software Engineering* (2011).
[28] Ying, A.T.T., Murphy, G.C., Ng, R. and Chu-Carroll, M.C.: Predicting Source Code Changes by Mining Change History, *IEEE Trans. Softw. Eng.* (2004).
[29] Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D. and De Lucia, A.: An Empirical Study on the Developers' Perception of Software Coupling, *Proc. International Conference on Software Engineering* (2013).
[30] D'Ambros, M., Lanza, M. and Robbes, R.: On the Relationship Between Change Coupling and Software Defects, *Proc. Working Conference on Reverse Engineering* (2009).
[31] Fischer, M., Oberleitner, J., Ratzinger, J. and Gall, H.: Mining Evolution Data of a Product Family, *Proc. International Workshop on Mining Software Repositories* (2005).

早瀬 康裕 （正会員）

平成 14 年大阪大学基礎工学部情報科学科卒業．平成 19 年同大学大学院博士後期課程修了．同年同大学特任助教．平成 22 年東洋大学総合情報学部助教．平成 23 年筑波大学システム情報系助教．博士（情報科学）．ソフトウェア開発支援，オープンソースソフトウェア開発分析の研究に従事．IEEE 会員．

中村 高士

平成 24 年筑波大学情報学群情報科学類卒業．平成 26 年同大学大学院システム情報工学研究科博士前期課程修了．修士（工学）．リポジトリマイニングに関する研究に従事．

天笠 俊之 （正会員）

平成 11 年群馬大学大学院工学研究科修了．博士（工学）．奈良先端科学技術大学院大学情報科学研究科助手，筑波大学大学院システム情報工学研究科講師，同准教授を経て，現在，筑波大学システム情報系准教授．データベース，データマイニング等の研究に従事．電子情報通信学会，IEEE 各シニア会員．日本データベース学会，ACM 各会員．

北川 博之 （フェロー）

昭和 53 年東京大学理学部物理学科卒業．昭和 55 年同大学大学院理学系研究科修士課程修了．日本電気（株）勤務の後，昭和 63 年筑波大学電子・情報工学系講師．同助教授を経て，現在，筑波大学システム情報系教授，ならびに計算科学研究センター教授．理学博士（東京大学）．データベース，情報源統合，データマイニング，情報検索等の研究に従事．著書『データベースシステム』（オーム社），『The Unnormalized Relational Data Model』（共著，Springer-Verlag）等．情報処理学会フェロー，電子情報通信学会フェロー，日本データベース学会会長，ACM，IEEE-CS，日本ソフトウェア科学会各会員．