ソフトウェアプロダクト間に存在する Logical Coupling を用いた変更箇所の推薦

中村　高士†　　早瀬　康裕††　　北川　博之††

† 筑波大学大学院システム情報工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学大学院システム情報系情報工学域　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †tnakamura@kde.cs.tsukuba.ac.jp, ††{hayase,kitagawa}@cs.tsukuba.ac.jp

あらまし　ソフトウェア開発において，単一ソフトウェアプロダクト内のモジュール間の関連性を表す指標の一つに，Logical Coupling という関係が存在する．Logical Coupling は，二つのモジュールが同時に変更されやすいことを表す関係であり，変更すべきモジュールの推薦などに利用できる．一方で，一つのソフトウェアプロダクトの開発に他のソフトウェアプロダクトのモジュールが関連している場合，これらのソフトウェアプロダクト間にも Logical Coupling が存在すると考えられる．この関係を検出することで，複数のソフトウェアに跨った変更すべきモジュールの推薦が可能になる．そこで本稿では，ソフトウェアプロダクト間に存在する Logical Coupling を検出する手法を提案する．そして，検出された Logical Coupling を用いた変更すべきモジュール推薦の可能性について，予備的検討を行う．

キーワード　Logical Coupling，相関ルール，ソフトウェアリポジトリ

Takashi NAKAMURA†, Yasuhiro HAYASE††, and Hiroyuki KITAGAWA††

† Graduate School of Systems and Information Engineering, University of Tsukuba
1–1–1 Tennodai, Tsukuba, Ibaraki, 305–8573 Japan
†† Faculty of Engineering, Information and Systems, University of Tsukuba
1–1–1 Tennodai, Tsukuba, Ibaraki, 305–8573 Japan
E-mail: †tnakamura@kde.cs.tsukuba.ac.jp, ††{hayase,kitagawa}@cs.tsukuba.ac.jp

1. はじめに

近年，多くのソフトウェアが開発され，様々な場面で利用されている．ソフトウェア開発者は世界中に数多く存在し，ソフトウェアの開発を行なっている．最近では，OSS（Open Source Software）の開発も盛んに行われており，多くの開発者が OSS 開発プロジェクトに参加している．OSS 開発プロジェクトの多くは，プロジェクトホスティングサービスを利用することで，プロジェクトに参加する開発者を管理し，ソースコードなどのリソースを開発者間で共有しながら開発を行なっている．

数多くのソフトウェアが開発され，ソフトウェアの構造も複雑になるにつれて，複数のプロジェクトに跨って開発が行われる状況も増えてきている．OSS 開発では，一人の開発者が複数の OSS 開発プロジェクトに参加していることも珍しくなく，相互に関連する複数のプロジェクトに参加している開発者も多い [11], [12], [14]．また，あるプロジェクトのリソースに変更を加える際，他のプロジェクトとの関連性が深く，複数のプロジェクトのリソースを一緒に変更しなければならない場面も存在する．例として，あるプロジェクトが別のプロジェクトで開発されているライブラリを利用しており，途中でライブラリの仕様が変更されてしまった場合に，ライブラリを呼び出す関数側も修正しなければならない場合などが挙げられる．

一方，単一ソフトウェアプロダクト内の関連するモジュールを見つけるため，モジュール間の結合度を評価する指標である Logical Coupling が提案されている．Logical Coupling とは，Gall ら [9] によって提案された指標で，「あるモジュールが変更された際には，ある別のモジュールも変更されている」というモジュール同士の関係を表したものである．Logical Coupling の利用方法としては，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を評価することによるソフトウェアの構造化などが挙げられる．Logical Coupling の計算をするために必要な，複数のモジュールが同時に変更されたという情報は，ソフトウェアの開発履歴を参照することで取得することができる．Zimmermann ら [13] は，Logical Coupling とそのモジュールの変更内容との関連について調査し，複数のモジュールが共通の目的で同時に変更されていることを示した．Fluri

ら [8] も同様に，Logical Coupling とそのモジュールの変更内容に関連があることを確認している．

OSS は他の OSS に依存したり関連した形で開発されているため，モジュール間の Logical Coupling は複数のプロジェクトに跨って存在しているはずである．しかし，既存の Logical Coupling の検出手法は単一のリポジトリを対象として検出を行うため，異なるプロジェクトのモジュール間に存在する Logical Coupling の検出を行うことはできない．異なるプロジェクトのモジュール間に存在する Logical Coupling を検出するためには，それらのモジュールの変更履歴を確認し，ログに含まれる情報を踏まえた上で，どの変更が同時に行われたものであるかを判別する必要がある．

また，あるプロジェクトと関連する他のプロジェクトのモジュールを変更する際，変更が必要なモジュールを特定するのは困難である．複数のソフトウェアプロダクトは独立して開発・保守されており，それらに含まれるモジュール間の繋がりを確認することはできないためである．この時，Logical Coupling を利用することで，あるモジュールの変更に際して，同時に変更する必要があるモジュールを推薦することができる．

そこで，本稿では異なるソフトウェアプロダクト間に存在する Logical Coupling を検出する手法を提案する．まず，複数のプロジェクトのモジュールに対して同時に行われた変更を推定し，Logical Coupling の検出を行う．そして，検出された Logical Coupling を用いてモジュールの推薦を行い，それらモジュールの間に同時に変更すべき関係が存在するかを確認する．

本稿の構成は以下の通りである．まず，2. 節で本研究に関連する知識と，ベースとなる Logical Coupling の検出手法について紹介する．次に，3. 節で提案する Logical Coupling の検出手法について説明し，4. 節でにそれを用いたモジュールの推薦ついて述べる．最後に，5. 節でまとめと今後の課題について述べる．

2. 関 連 知 識

本節では，本研究を説明する上で必要な関連知識について述べる．まず，2.1 節でバージョン管理システムとその変更履歴から得られる情報について述べた後，2.3 節で Logical Coupling の検出手法を提案した先行研究について紹介する．

2.1 バージョン管理システム

バージョン管理システム (Version Control System または VCS) とは，ソフトウェアのリソースを管理し，開発履歴を記録するシステムである（図 1）．管理対象としたファイルのスナップショットを保存することで，変更履歴の管理を行う．VCS によって管理されるソフトウェアのリソースとその変更履歴が保存されたデータベースをリポジトリと呼び，リポジトリにファイルのスナップショットを保存する作業をコミットと呼ぶ．この時，コミットに伴う情報もログとしてリポジトリに保存されるため，ログを参照することで後からコミットの詳細を確認することができる．ログには，コミットを行った開発者のユーザ名とメールアドレス，コミット日時，コミットについてのメモ，変更されたファイル名，変更内容などが含まれる．

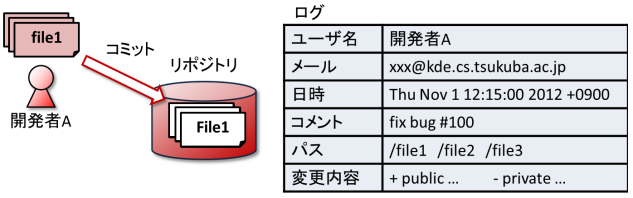

図 1 バージョン管理システム

また，VCS では複数箇所への変更を一度のコミットでまとめてリポジトリへ格納することができる．この複数箇所への変更をまとめた一連の変更のかたまりを，コミットトランザクションと呼ぶ．一般的に，論理的な繋がりをもった一連の変更を，一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．今日の VCS はコミットトランザクションを明示的に管理しているものがほとんどであり，ログを参照することであるコミットトランザクションに含まれる一連の変更を確認することもできる．

2.2 プロジェクトホスティングサービス

プロジェクトホスティングサービスとは，VCS によって作られたソフトウェアリポジトリを複数の開発者で共有して，ソフトウェアの開発を行うためのサービスである．プロジェクトホスティングサービスを利用することで，異なる場所にいる複数の開発者が，共通のソフトウェアの開発に継続的に参加することができるようになる．オープンソースソフトウェア（OSS）開発では，異なる場所にいる複数の開発者で作業を分担して行う必要があるため，プロジェクトホスティングサービスが頻繁に利用されている．プロジェクトホスティングサービスの代表的な例として，GitHub [2] や SourceForge [3]，Google Code [4] や CodePlex [5] などがある．

プロジェクトホスティングサービスにホストされたリポジトリは，一部のプライベートなリポジトリを除いて，ソースコードなどのリソースが公開されている．そのため，ユーザは自由にソースコードを確認したり，リポジトリをダウンロードしてソフトウェアを利用することができる．また，リポジトリに対して機能の追加やバグ修正などを施した場合，元のリポジトリに対して変更を取り込んでもらえる場合もある．

また，プロジェクトホスティングサービスでは，ソフトウェア開発に関わるやり取りなども公開している．プロジェクトホスティングサービスはソフトウェア開発を円滑に行うための様々なサービスを提供しているが，それらの履歴等も公開されているのである．GitHub というプロジェクトホスティングサービスでは，ユーザがリポジトリに対してバグ報告や機能追加の要求を行うことができたり，ダウンロードしたリポジトリに対して行った修正をダウンロード元のリポジトリに取り込んでもらうようリクエストを送ったりすることができる．ユーザはリポジトリに含まれるリソースだけでなく，こういった開発の履歴についても確認することができる．

2.3 Logical Coupling の検出

まず，VCS から抽出したログを用いて Logical Coupling の検出を行った例として，Gall ら [10] の研究を紹介する．Gall

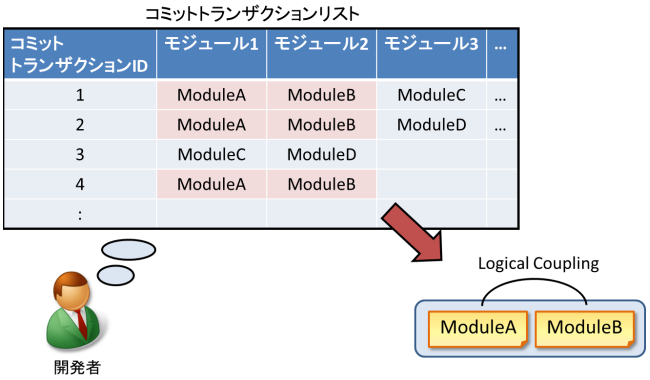

図 2　Gall らによる Logical Coupling の検出手法

らは，VCS によって管理されるソフトウェアの開発履歴を分析することで，Logical Coupling の検出を行う手法を提案した．また，Gall らは代表的な VCS の一つである CVS [6] のリポジトリを対象としている．

Gall らの手法では，どのファイルが同時に変更されたかという情報を必要とする．しかし，CVS はコミットトランザクションを明示的に管理しない VCS である．そのため，Gall らは各ファイルに対して行われた変更が一つのコミットトランザクションに属するものであるかを推定している．

Gall らは，コミットトランザクションを推定するために，以下の二点を同時に満たすことを基準として用いた（図 2）．

（1） 著者の名前が一致していること．

（2） 4 分以内に行われた変更であること．

このように，同一の開発者によってとても近い時間に行われた複数の変更は，同一のコミットトランザクションに属すると仮定して，コミットトランザクションの推定を行った．

続いて，Gall らの手法の手順を説明する．まず，VCS のログを参照し，モジュールが変更された時間と変更者の名前を取得する．そして，上記の条件を満たすコミット群を一つのコミットトランザクションとみなし，それらのコミットによって変更されたモジュールから二つを選択することにっよって，同時に変更されたモジュールのペアを生成する．最後に，モジュールのペアごとに出現回数をカウントし，その数が定めたしきい値を上回っている場合，そのモジュールのペアを Logical Coupling と判定する．このようにして，Gall らは実際のソフトウェア開発記録を用いて，単一リポジトリ内でのモジュール間の結合度を調査した．

また，複数のリポジトリに対して Logical Coupling の検出を行った例として，Fischer ら [7] の研究がある．Fischer らは，同じ製品グループに属するリポジトリ群に対して，Logical Coupling の検出手法を適用した．複数のリポジトリの開発履歴から得られる情報を統合し，分析を行なっている．これによって Fischer らは，異なるリポジトリに含まれる複数のモジュールが同時に変更されていることを示した．

3. 提案する Logical Coupling の検出手法

本節では，複数のリポジトリの開発履歴を用いて，複数の

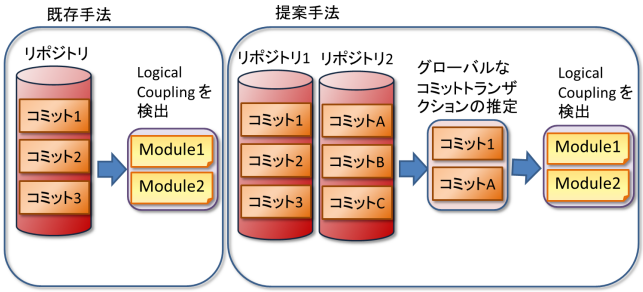

図 3　既存手法と提案手法の比較

リポジトリに対して同時に行われる変更を推定し，ソフトウェアプロダクト間に存在する Logical Coupling を検出スル手法を提案する．ソフトウェアプロダクト間に存在する Logical Coupling を検出することによって，異なるプロジェクトのモジュールの変更に関する相関ルールを抽出する．こうして得られた相関ルールを利用することで，開発者があるモジュールを変更した際，次に変更すべきモジュールをプロジェクトの枠を越えて推薦することができる．

また，独立して管理されている複数のリポジトリについて，それらに対して行われたコミットが同時に行われたものであるかを推定する必要がある（図 3）．これは，複数のリポジトリに跨って複数の変更が同時に行われたという，コミットトランザクションにあたる情報が存在しないため，同時に行われた変更を明示的に確認することができないためである．

本研究では，複数のリポジトリに対して同時に行われた変更を，インターリポジトリコミットトランザクションと呼ぶことにする．インターリポジトリコミットトランザクションを定義することで，複数のリポジトリに対して同時に行われた変更を一つのまとまった変更と見なし，相関ルールの抽出を行うことができるようになる．

また，本手法では代表的な VCS の一つである Git [1] により作成された，Git リポジトリを対象としている．異なるプロジェクトにより管理される複数の Git リポジトリを入力として，別々のリポジトリに含まれる同時に変更すべきモジュール（Logical Coupling）を出力する．

続いて，手法全体の流れを説明する（図 4）．まず，プロジェクトホスティングサイトにホストされているリポジトリに対して Git コマンドを用いることで，リポジトリのログを取得する．ここで，取得したログはリレーショナルデータ分析する際に扱いづらいため，必要な情報を整理しデータベースへ格納する．対象とする複数のリポジトリに対して同様の操作を実行することで，複数のデータベースを構築する．

次に，データベースに含まれる提案する条件を満たすコミットを，インターリポジトリコミットトランザクションとして推定する．インターリポジトリコミットトランザクションの推定にあたって，通常のコミットトランザクションを推定する場合と比較して，次のようなことを考慮する必要がある．まず，開発者は複数のプロジェクトに参加しているため，開発者を一意に識別する ID の役割を持つものを用意する．また，一人の開

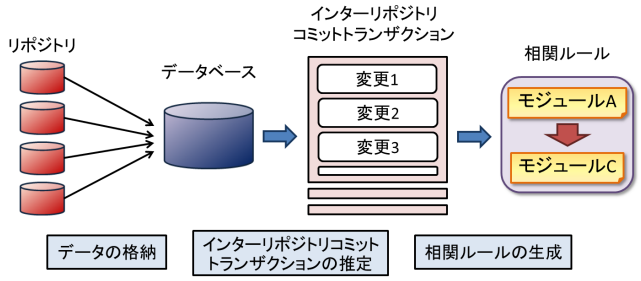

図 4　提案手法の流れ

発者によって同時期に為された変更であっても，ある程度離れた時間間隔でコミットされることが考えられるため，適切な時間範囲を設定する必要がある．次に，リポジトリ同士に利用関係が存在する場合，利用されている側のリポジトリに含まれるモジュールが変更された場合，それを利用している側のリポジトリのリポジトリも同時に変更しなければならない場合がある．この時，ある開発者が双方のリポジトリを管理するプロジェクトに参加しているとは限らないため，コミットを行う開発者が異なる状況も考えられる．

最後に，推定されたインターリポジトリコミットトランザクションを用いて，モジュールの変更に関する相関ルールを生成する．この時，単一リポジトリに含まれるモジュール同士に関する相関ルールは，本研究の検出対象ではないため除外する．そして，生成された相関ルールについて出現回数と確信度を計算し，その両方が定めたしきい値を超えているルールに含まれるモジュールを Logical Coupling と判定する．

まず，3.1 節でログをデータベースへ格納する方法について説明する．次に，3.2 節でインターリポジトリコミットトランザクションを推定するための新たな条件を提案する．最後に，3.3 節で相関ルールの生成について説明する．

3.1　データベースへのログの格納

Git では，リポジトリに対してログ取得のコマンドを実行することで，リポジトリに対して行われたコミットのログを時系列順に取得することができる．取得したログはテキスト形式となっているため，ログの形式に沿って必要なデータを抽出し，用意したデータベースのスキーマに沿って整理する．

そして，それぞれのリポジトリに対して行われたコミットの一覧と，それらのコミットにより変更されたモジュールの一覧を関連付けて，データベースへ格納する．これによって，それぞれのコミットがどのリポジトリに対して行われたか，どのモジュールがどのコミットにより変更されたかを確認することができる．

3.2　インターリポジトリコミットトランザクションの推定

ここでは，複数のリポジトリに対して同時に行われた変更，即ちインターリポジトリコミットトランザクションを推定する条件について述べる．インターリポジトリコミットトランザクションが発生すると考えられる状況に合わせて，本手法では三種類の条件を提案する．

まず，一人の開発者が複数のプロジェクトに参加している状況において，複数のプロジェクトに対して同時期に行った変更を検出するために，以下の条件を提案する．

- コミット同士が近い時間に行われており，また，同一著者による変更であること．

この時，コミットを行うのはあくまで一人の開発者であるため，コミット同士の時間間隔は比較的短いものでよい．また，同一著者であるかの判定には，コミットのログから得られる開発者の情報を利用する．

次に，リポジトリ間に利用関係がある場合，利用されている側のリポジトリに含まれるモジュールが変更され，利用している側のリポジトリに含まれるモジュールも変更する必要がある，という状況がある．このような変更を検出するために，以下の条件を提案する．

- コミット同士が近い時間に行われており，また，変更されたモジュールが利用関係の近傍にあるリポジトリに含まれていること．

リポジトリの利用関係は，ソースコードを調べることで確認できることがある．ソースコードから確認できるリポジトリ同士の利用関係の例として，Java における import 文や，C における include 文などがある．また，VCS の機能で他のリポジトリを利用指定をしている場合，VCS から利用関係の情報を得ることができる．この時，変更が伝搬するのに多少の時間がかかるため，コミット同士の時間間隔は比較的長い時間を想定する．

最後に，その他の状況において何らかの共通の要因によって，異なるプロジェクトに含まれるモジュールが同時行われた変更を検出するために，以下の条件を提案する．

- コミット同士が近い時間に行われており，また，コミットコメントの内容が類似していること．

コミットコメントの内容を比較する際，コメントの量や，使用されている語句の専門性を考慮する．膨大な量のコメントにおいてその多くの単語が類似している場合，そのコメント同士は非常に類似していると考えられる．同様に，あまり一般的に使用されない専門的な語句がコメント内に共通して含まれている場合，そのコメント同士は類似していると考えられる．

3.3　相関ルールの生成

3.2 節で推定したインターリポジトリコミットトランザクションを元に，相関ルールを生成する．本手法では異なるリポジトリのモジュール間に存在する Logical Coupling を検出対象としているため，相関ルールに含まれるモジュールが同一のリポジトリに含まれていないルールのみを生成対象とする．

本手法では，インターリポジトリコミットトランザクションを一つのアイテムトランザクションと見なして，相関ルールの生成を行う．インターリポジトリコミットトランザクション中における，各モジュール集合の出現頻度を計算する．更に，ルールの前提部のモジュール集合の出現頻度とルール全体のモジュール集合の出現頻度から，各相関ルールの確信度を計算する．この時，計算された相関ルールの出現回数と確信度が定めたしきい値を上回っていた場合，ルールに含まれるモジュールを Logical Coupling と判定する（図 5）．

表 1　推薦・評価の結果

推薦対象モジュール		推薦されたモジュール		出現回数	確信度	関連性
リポジトリ名	モジュール名	リポジトリ名	モジュール名			
senchalabs/connect	lib/patch.js	visionmedia/express	lib/response.js	4	0.57	×
senchalabs/connect	lib/middleware/router.js	visionmedia/express	package.json	4	0.57	×
senchalabs/connect	lib/patch.js	visionmedia/jade	Readme.md	4	0.57	×
rails/rails	railties/html/javascripts /dragdrop.js	madrobby/scriptaculous	CHANGELOG	14	0.54	○
visionmedia/express	.pomo	visionmedia/jade	test/jade.test.js	4	0.50	×
ajaxorg/ace	src/ace/conf/keybindings /default_mac.js	ajaxorg/cloud9	client/ext/tree/tree.xml	4	0.50	×
visionmedia/express	.pomo	visionmedia/jade	lib/jade.js	4	0.50	×

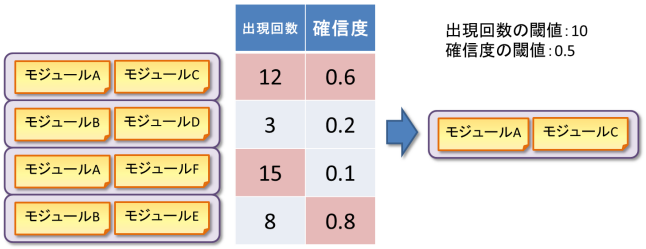

図 5　Logical Coupling の判定

4. 変更箇所の推薦

本節では，提案手法によって検出した Logical Coupling を用いた変更箇所の推薦の結果について議論する．検出した Logical Coupling を用いてモジュールの推薦を行い，推薦されたモジュールが推薦対象のモジュールと一緒に変更すべきモジュールであるかどうかを確認する．

まず，複数のリポジトリを対象として，提案手法を用いて検出した Logical Coupling を用いて変更箇所の推薦を行う手法を紹介する．これによって，あるモジュールの変更に伴って他のプロジェクトのモジュールを変更する場合に，必要な変更をより容易に行うことができると考えられる．

変更箇所の推薦には，3. 節の手法によって抽出した，モジュールの変更に関する相関ルールを利用する．まず，相関ルールの出現回数と確信度のしきい値を設定し，出現回数・確信度の両方が定めたしきい値を超えているルールを抽出する．そして，抽出されたルールについて，ルールの前提部と推薦対象のモジュールが一致している場合，ルールの結論部のモジュールを推薦する．

次に，推薦結果の評価基準について説明する．推薦結果の相関ルールに含まれるモジュールが，本当に同時に変更すべきモジュールであるかどうかを確認する必要がある．例として，二つのモジュールの間にコードクローンが存在したり，モジュールが含まれるリポジトリ間に利用関係などが存在する場合，それらは同時に変更すべきモジュールであると考えることができる．

二つのモジュールが同時に変更すべきかどうかを客観的に判断するため，今回は以下の基準のいずれかを満たすことを条件とした．

- 同一のインターリポジトリコミットトランザクションに含まれる，二つのモジュールに対して行われたコミットの diff に類似した行が存在する．
- 同一のインターリポジトリコミットトランザクションに含まれる，二つのモジュールに対して行われたコミットのコメントの内容が類似している．
- 一方のモジュールが含まれるプロダクトが，もう一方のモジュールが含まれるプロダクトを利用している．

また，今回は提案手法で述べた三種類の条件のうち，「二つの変更が近い時間に行われており，また，同一著者による変更であること」という条件についてのみ，実験を行った．実験に用いた具体的な基準はそれぞれ次の通りである．

- 著者のメールアドレスが一致していること．
- 24 時間以内に行われた変更であること．

実験によって検出するモジュールの粒度をファイル単位として，Logical Coupling の検出を行う．モジュールの粒度をファイルとした理由は，ファイルの内容や使用言語に影響を受けない範囲で，最も粒度の細かい単位であるためである．また，今回は全てのルールのうち，相関ルールの前提部，結論部の要素が一つのルールのみを抽出している．

続いて，今回行った変更箇所の推薦の実験に使用したデータと，実験結果について述べる．まず，4.1 節で使用したデータについて説明し，4.2 節で実験結果について述べる．そして，4.3 節でその結果について考察する．

4.1 実験に使用したデータ

GitHub にホストされた 100 件のリポジトリを，データセットとして使用した．これらのリポジトリは，GitHub の watch という機能を用いることで注目しているユーザ数の多いリポジトリを特定し，その数の多い順に 100 件のリポジトリを選択している．また，100 件のリポジトリに対して行われた総コミット数は 527,317 件であり，ファイルが変更された回数の合計は 1,691,746 回であった．

4.2 結　　果

100 件のリポジトリから取得したログを用いて，条件を満たすインターリポジトリコミットトランザクションを推定し，モジュールの変更に関する相関ルールを生成した．この時生成された相関ルール数は合計 78,306 件であった．これらのルールのうち，確信度が 0.5 以上であり，出現回数が 4 回以上のルー

ルを，モジュールの推薦に利用した．

結果として，定めたしきい値を満たす相関ルール数は 7 件であった．表 1 はそれらの相関ルールを，確信度の高い順に上位 10 件を並べたものである．この表では，それぞれの行が一つの相関ルールを表している．一列目，二列目が推薦対象モジュールと，それに対して推薦されたモジュールを表しており，三列目，四列目がルールの出現回数と確信度を示している．

前述の条件にしたがって，モジュール間に同時に変更すべき関係があるかを判断した結果を，表 1 の五列目に示した．総評として，モジュール間に関連性が見られたものは 7 件中 1 件であり，あまり良い推薦結果は得られなかった．

今回の実験で生成されたルールでは，どのようなモジュールの変更に対してどのようなモジュールが推薦されていたのかを説明する．まず最初に，関連がないと判断した，表 1 の二行目のルールについて言及する．推薦対象モジュールは sen-chalabs/connect リポジトリに含まれる lib/patch.js，推薦されたモジュールは visionmedia/express リポジトリに含まれる lib/response.js である．推薦対象モジュール，推薦されたモジュールともに，javascript のソースファイルである．それぞれのモジュールに対して行われたコミットの diff，コミットのコメントを確認したが，類似した内容は見られなかった．また，それぞれのモジュールが含まれるプロダクト同士に利用関係が無いか探したが，そのような関係を見つけることはできなかった．そのため，今回はこのルールに含まれるモジュール同士には関連がないと判断した．また，表 1 の三，四，六，七，八行目のルールについても，同様に内容を確認したが上記の基準を満たさなかったため，ルールに含まれるモジュール同士には関連がないと判断した．

続いて，関連があると判断した 1 件のルールについて言及する．推薦対象モジュール rails/rails リポジトリに含まれる railties/html/javascripts/dragdrop.js，推薦されたモジュール madrobby/scriptaculous に含まれる CHANGELOG である．推薦対象モジュールは javascript のソースファイルで，推薦されたモジュールは変更ログファイルである．それぞれのモジュールに対して行われたコミットのコメントを確認したところ，双方のコメントに共通の内容が確認できたため，このルールに含まれるモジュール同士には関連があると判断した．ただし，このルールでは変更ログファイルが推薦されていたため，あまり有用な推薦結果とは言えないだろう．

4.3 考　　察

今回の実験では，異なるプロダクト間に存在するモジュールの変更に関する相関ルールを生成したが，ルールに含まれるモジュール間の関連はあまり見られなかった．有用な推薦が行えなかった原因として，推薦されるモジュールの変更回数を考慮していなかったことが考えられる．相関ルールの性質上，とても頻繁に変更されるモジュールはルールの結論部に出現しやすくなるためである．

また，パラメタについて十分な検討ができていなかったことが考えられる．まず，インターリポジトリコミットトランザクションの推定のための時間間隔を一律で 24 時間と設定した．しかし，より近い時間に行われた変更は，同時に行われた変更である可能性が高いと考えられるため，その点を考慮した推定基準を検討する必要がある．また，相関ルールの生成時のパラメタについても，より良いパラメタを検討する必要がある．

最後に，今回対象としたリポジトリ数が 100 件と少なく，データセット中に深く関連するプロジェクトがあまり含まれていなかったことが考えられる．調査するリポジトリ数が少ない場合，データセット中に深く関連するプロジェクトが含まれる可能性が低くなるためである．

5. まとめと今後の課題

本稿では，複数のリポジトリに対して Logical Coupling の検出を行う手法を提案した．複数のリポジトリに対して同時に行われるコミットとはどのようなものかを考え，それに則した検出条件を作成した．そして，提案した手法のうち一つを実装し，Logical Coupling の検出を行った．また，提案手法を用いて変更すべきモジュールの推薦を行い，その結果の有用性について議論した．

今後の課題として，まずモジュールの変更回数を考慮するなどして，より有用な推薦を行える手法について検討する．また，今回行った実験は対象としているリポジトリ数が少なかったため，より多くのリポジトリを対象として Logical Coupling の検出を行う必要がある．多くのリポジトリを対象とすることで関連があるリポジトリの絶対数が増加するため，同時に変更すべきモジュールも増加すると考えられる．最後に，今回実験を行うことが出来なかった条件についても Logical Coupling の検出を行い，それを用いた場合の推薦結果を評価する．

謝辞　本研究は，産総研基盤 A（24240015A）による．

文　　献

[1] Git. `http://git-scm.com/`. accessed on 01/14, 2013.
[2] GitHub. `https://github.com/`. accessed on 01/14, 2013.
[3] SourceForge. `http://sourceforge.net/`. accessed on 01/14, 2013.
[4] Google Code. `http://code.google.com/`. accessed on 01/14, 2013.
[5] CodePlex. `http://www.codeplex.com/`. accessed on 01/14, 2013.
[6] CVS. `http://www.nongnu.org/cvs/`. accessed on 01/14, 2013.
[7] Michael Fischer, Johann Oberleitner, Jacek Ratzinger, and Harald Gall. Mining evolution data of a product family. In *Proc. Int'l Workshop on Mining Software Repositories (MSR)*, 2005.
[8] Beat Fluri, Harald C. Gall, and Martin Pinzger. Fine grained analysis of change couplings. In *Proc. Int'l Workshop on Source Code Analysis and Manipulation (SCAM)*, 2005.
[9] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proc. Int'l Conf. Software Maintenance (ICSM)*, 1998.
[10] Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. CVS release history data for detecting logical couplings. In *Proc. Int'l Workshop on Principles of Software Evolution (IWPSE)*, 2003.

[11] Masao Ohira, Naoki Ohsugi, Tetsuya Ohoka, and Ken-ichi Matsumoto. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. In *Proc. Int'l Workshop on Mining software repositories (MSR)*, 2005.

[12] Didi Surian, Nian Liu, David Lo, Hanghang Tong, Ee-Peng Lim, and Christos Faloutsos. Recommending people in developersćollaboration network. In *Proc. Working Conference on Reverse Engineering (WCRE)*, 2011.

[13] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proc. Int'l Conf. Software Engineering (ICSE)*, 2004.

[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.