Vol.2013-SE-182 No.31
2013/10/25

ソフトウェアプロダクト間での Logical Coupling 検出に向けた予備的な調査

中村 高士[†1,a)]　早瀬 康裕[†2,b)]　北川 博之[†2,c)]

概要：2 つのソフトウェアモジュールが，同時に変更されやすい関係にあることを表す指標 Logical Coupling は，プログラム理解や変更支援において有用な指標と考えられている．Logical Coupling はバージョン管理システムのリポジトリの中で同時変更の関係を見付ける手法であったため，単一のソフトウェアプロダクトでのみ検出することが可能であった．しかし，異なるプロダクトに属するモジュールであっても，そのモジュールの組に関係があれば，同時に変更されやすい関係にあるのではないかと考えられる．そこで本稿では，異なるソフトウェアプロダクトに含まれるモジュール間の Logical Coupling を検出する方法を考案し，その方法を用いて，オープンソースソフトウェアのリポジトリ集合に対して予備的な調査を行う．調査の結果，複数のソフトウェアプロダクトについて，同時に変更されやすいモジュールが存在することが確認できた．

Exploratory Study for Logical Coupling Detection among Software Products

Nakamura Takashi[†1,a)]　Hayase Yasuhiro[†2,b)]　Kitagawa Hiroyuki[†2,c)]

Abstract: Logical Coupling which shows that two software modules tend to change together is a useful indicator for program understanding and change recommending. As Logical Coupling is a method for detecting co-change relationship in a repository of version control system, it can be detected in single software product. Though the modules belong to different products, if there is relationship among the modules, the modules tend to change together. Then, we propose the mothod for detecting Logical Coupling among modules in defferent software products and do an exploratory investigation of OSS repositories using the method. After the investigation, we confirm that there are modules which tend to change together in multiple software products.

1. はじめに

ソフトウェア開発では，複数の開発者が協力して開発が行われていることが多い．特に OSS（Open Source Software）の開発では，世界中から多くの開発者がソフトウェア開発プロジェクトに参加し，開発を行っている．多くの OSS 開発プロジェクトでは，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している開発者の管理やソースコードなどのリソースの共有を行っている．そして，プロジェクトに参加する開発者で作業を分担することで，効率的に開発を行っている [1]．

一方で，プロジェクトに参加する開発者といえども，開発するプロダクトのモジュール間の関係を把握することは難しい．モジュール間の関係を見逃してしまい，モジュール同士の関連する箇所を変更し忘れてしまった場合，バグの原因となってしまうことがある．また，プロダクトの構造が複雑になるとソフトウェアを理解することが困難になり，開発効率が低下してしまう可能性がある．

†1 現在，筑波大学システム情報工学研究科
Presently with Graduate School of Systems and Information Engineering, University of Tsukuba
†2 現在，筑波大学システム情報系
Presently with Faculty of Engineering, Information and Systems, University of Tsukuba
a) tnakamura@kde.cs.tsukuba.ac.jp
b) hayase@cs.tsukuba.ac.jp
c) kitagawa@cs.tsukuba.ac.jp

このような問題を解消するために，ソフトウェアプロダクト内のモジュール同士の関連を表す指標である Logical Coupling が提案されている [2]．Logical Coupling は「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高い割合で変更されている」というモジュール間の関係を表す．Logical Coupling が表すモジュール間の関係は，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，Logical Coupling はモジュールの変更をアイテムとした相関ルールによって表現されることがある．相関ルールの計算に必要なモジュールの変更に関するトランザクションの情報は，ソフトウェアの開発履歴等を参照して導出されていることが多い．例として，Zimmermann ら [3] はソフトウェアの開発履歴から得られる情報を用いてモジュールの変更に関する相関ルールを計算し，Logical Coupling の検出を行った．

このように，単一のプロダクト内でのモジュール間の関係について，いくつもの研究が行われている．一方で，ソフトウェアの開発は単一のプロジェクト内で完結せず，他のプロジェクトで開発されている別のプロダクトと影響し合う場合がある．例えば，ある開発者が複数の OSS の開発プロジェクトに参加しているという状況は少なくない [4], [5], [6]．これら複数のプロジェクトが管理する各プロダクトで類似する問題が見つかった場合，開発者はプロダクト間でコードの再利用を行い，問題に対処すると考えられる．このような場合，プロダクト同士に直接的な関わりがないとしても，それらのプロダクト間には互いに影響し合う関係が存在すると言える．別の状況として，あるプロダクトが他のプロダクトを利用している場合には，他のプロダクトの更新にともなって，プロダクトのソースコードを変更する必要がある場合がある．例として，ライブラリ等のソフトウェア開発の基盤となるプロダクトのインターフェースが更新された際に，それらを利用するプロダクトが影響を受けるといった状況が考えられる．

以上のことから，Logical Coupling は単一のプロダクト内だけでなく，異なるプロダクト間にも存在すると考えられる．プロダクト間での Logical Coupling を検出することで，他のプロダクトの変更によって受ける影響に対応しやすくなり，影響を受ける箇所の修正やリファクタリングをより効率的に行うことができるようになる．

しかし，従来の Logical Coupling 検出では単一のプロダクトを対象としており，異なるプロダクト間での Logical Coupling の検出を行うことができない．異なるプロダクト間での Logical Coupling を検出するためには，複数のプロダクトから抽出した情報をどのように統合し，利用するかを考える必要があるためである．

そこで本研究では，ソフトウェアプロダクト間での Logical Coupling 検出の方針を提案する．異なるプロダクト間での Logical Coupling を検出するためには，複数のプロダクトの開発履歴から得られる情報を統合した上で，どのモジュールとどのモジュールが頻繁に一緒に変更されているかを判別する必要がある．開発履歴からモジュールの変更に関する情報から，どの変更とどの変更が一緒に行われたものかを推測することで，Logical Coupling の検出に利用する．

更に，現在の手法でどのような結果が得られるかを確認するため，提案した手法に基づいて考案した手法を OSS のリポジトリに対して適用する実験を行う．どのモジュールが，どのような理由で変更された結果，異なるプロダクト間ではどのような Logical Coupling が検出されるのかを確認する．そして，得られた結果を考察し，より有用な Logical Coupling を検出するための方法を検討する．

本稿の構成は以下の通りである．まず，2 節で提案する Logical Coupling の検出方針について説明する．次に，3 節で提案方針に基づいて行った調査について説明する．そして，4 節で関連研究を紹介し，最後に 5 節でまとめと今後の課題について述べる．

2. ソフトウェアプロダクト間での Logical Coupling 検出の方針

本節では，ソフトウェアプロダクト間での Logical Coupling 検出方針について述べる．ソフトウェアプロダクト間での Logical Coupling にはどのような状況が考えられるかを挙げ，その状況に則した検出手法を提案する．

ソフトウェアプロダクト間での Logical Coupling 検出方針を述べるにあたって，まず従来の単一プロダクト内での Logical Coupling を検出する場合との違いを明確にしておく．従来手法では，単一のソフトウェアプロダクトの開発履歴の入力を前提として，単一のプロダクトに含まれるモジュールの変更に関する相関ルールを出力していた．一方提案手法では，複数のソフトウェアプロダクトの開発履歴の入力を前提として，異なるプロダクト内のモジュールの変更に関する相関ルールを出力する．この違いから，プロダクト間での Logical Coupling を検出するためには，単一プロダクト内の Logical Coupling を検出する手法とは違った，新しい手法を提案する必要がある．

続いて，検出方法の詳細について説明していく．まず，2.1 節で異なるリポジトリのモジュールが同時に変更される状況にはどういった状況があるかを考えておく．次に，2.2 節でプロダクト間での Logical Coupling 検出の手順について述べる．そして，2.3 節で IRCT を推定する方法について説明し，2.4 節で Logical Coupling の検出について説明する．

2.1 リポジトリを跨いだ同時変更トランザクション

手法を提案する前に，リポジトリ間での Logical Coupling 検出を行うにはどのような情報が必要かを明らかにしておく必要がある．リポジトリ間での Logical Coupling 検出では，単一リポジトリ内での Logical Coupling 検出と異なり，モジュールが同時に変更されたという情報をリポジトリから抽出することができない．そのために，異なるリポジトリに含まれるモジュールが同時に変更される状況にはどういった状況があるかを考えておき，同時に行われた変更を推定する必要がある．

まず一つ目に，異なるプロジェクトによって管理されるソフトウェアウェアプロダクトに対してであっても，ある開発者によって同様の内容の変更が行われるという状況がある．開発者があるプロダクトに対して何らかの変更を行った際，同様の内容の変更を他のプロダクトに対しても行う場合があると考えられる．例として，複数のプロダクトにコピーアンドペーストなどによって作られた類似したコードが存在している場合，それらのコードに対して行われる変更は，同様の内容の変更となるだろう．

次に，多くのソフトウェアプロダクトには，ある共通の要因によって，一斉に変更されるという状況がある．プロダクトごとに変更が行われる時期に多少の差はあったとしても，保守されているプロダクトであれば比較的近い時間間隔で変更が行われるものと予想される．このような状況の例として，類似するプロダクトで用いられている定数の値が変更される場合等には，その定数を用いている多くのプロダクトが影響を受けることになる．

また，あるモジュールが他のプロダクトを利用している場合，他のプロダクトの更新に影響を受けて変更が行われる状況がある．これは，利用しているプロダクトの更新が原因でバグが発生すると，プロダクトのソースコードを修正しなければならなくなるということである．他のプロジェクトで開発されているライブラリ等を利用しているモジュールがあった場合，ライブラリのインターフェースが変更になると，ライブラリを利用するモジュールのインターフェースも変更する必要が出てくる．このような，プロダクト同士が直接的な繋がりを持つような状況も存在すると考えられる．

本稿では，このような状況で行われたコミットの集合をインターリポジトリコミットトランザクション（IRCT）と定義する．そして，IRCT を推測することでモジュールに対して行われた変更をまとめ，Logical Coupling の検出に利用する．

2.2 ソフトウェアプロダクト間での Logical Coupling の検出方法

本節では，2 節の冒頭で述べた方針に沿って考案した，プロダクト間での Logical Coupling の検出方法ついて説明する．尚，異なるソフトウェアプロダクトは，それぞれが別々のリポジトリで管理されることが一般的である．そのため，異なるリポジトリ間での Logical Coupling を検出することで，ソフトウェアプロダクト間での Logical Coupling 検出を行うことができると考えられる．

それでは，プロダクト間での Logical Coupling 検出の手順を説明していく．まず，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から，コミットログを取得する．ここで取得したログは扱いやすいように整形されているわけではないため，必要な情報を整理し，リレーショナル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処理を行い，複数のリポジトリのコミットログの情報を格納したデータベースを作成する．

続いて，コミットログから得られる情報を参照して，IRCT の推定を行う．データベースに格納されたコミットログを参照し，定めた基準を満たすコミットの集合を IRCT として推定する．この時，異なるリポジトリに対して行われたコミットについて，同じ目的で行われたコミットを判別するため，次のようなことを考える必要がある．まず，同じ開発者が行ったコミットを判別する場合，開発者を一意に識別するために，開発者の ID の役割を持つものを考える必要がある．更に，同じ開発者によって同じ目的で行われたコミットはどの程度の時間以内にコミットされているのかを考え，適切な時間間隔を設定する必要がある．また，あるモジュールが他のプロダクトを利用している場合等では，同じ目的で行われたコミットが一人の開発者によって行われたものとは限らない．このような状況についても，状況に則した IRCT の推定基準を定める必要がある．

最後に，推定された IRCT を用いて，異なるリポジトリ間での Logical Coupling を検出する．本研究では Logical Coupling を相関ルールの形で表現する．また，単一のリポジトリ内での Logical Coupling は，本研究の検出対象ではないため検出を行わない．そして，相関ルールの生成を行い，support count と confidence の値が定めた条件を満たすルールのみを，Logical Coupling として検出する．

2.3 IRCT の推定方法

ここでは，コミットから IRCT を推定する方法について述べる．2.1 節で述べた通り，異なるリポジトリに含まれるモジュールであっても，一緒に変更される状況は考えられる．このような，同じ目的で行われたコミットを IRCT として推定する．

本手法では，コミットが同じ目的で行われたものだと推測される状況に合わせて，三種類の推定基準を提案する．これらの推定基準はそれぞれ異なる状況で行われたコミットを IRCT として推定するために定めた基準である．そのため，これらの基準に沿って推定された三種類の IRCT

はそれぞれ独立のものであり，どの IRCT を利用するかによって異なる Logical Coupling が検出できるものと考えられる．それでは，提案する推定基準と，その基準を定めた理由について順に説明していく．

推定基準 1.

- 時間間隔 Δ_1 以内に行われたコミットである．
- 同一の開発者によって行われたコミットである．

推定基準 1 は，複数のプロジェクトが管理する異なるプロダクトに対して，同じ目的で行われたコミットの推定を行うために提案したものである．同じ開発者によって同じ目的でコミットが行われている場合，近い時間間隔でコミットが行われることが予想されるため，この基準で推定することができるのではないかと考えた．

推定基準 2.

- 時間間隔 Δ_2 以内に行われたコミットである．
- コミットのコメントの内容が類似している．

推定基準 2 は，共通の要因によって複数のプロダクトに含まれるモジュールに対して，同じ目的で行われたコミットの推定を行うために提案したものである．共通の要因による変更であれば，コミットを行った開発者が別の人物だったとしても，同様の内容のコミットコメントを残すことが予想される．そして，類似するコミットコメントを持ち，近い時間間隔で行われたコミットのは，何か共通の要因によって引き起こされたものである可能性が高いと考えた．

推定基準 3.

- 時間間隔 Δ_3 以内に行われたコミットである．
- あるコミットにより変更されたモジュールが，別のコミットにより変更されたモジュールを含むプロダクトを利用している．

推定基準 3 は，あるモジュールが他のプロダクトを利用しており，他のプロダクトの更新に伴いモジュールを変更する必要が発生した場合に行われたコミットの推定を行うために提案したものである．利用関係が存在するようなプロダクトに対して，近い時間にコミットが行われている場合，一方のプロダクトに対する変更が他方のプロダクトに影響を与え，コミットが行われたという可能性が考えられるため，このような基準を提案した．

2.4 Logical Coupling の検出

ここでは，推定した IRCT を同時に行われた変更のトランザクションとして扱うことで，Logical Coupling の検出を行う方法を説明する．異なるプロダクトに含まれるモジュールについて，変更に関する相関ルールを計算し，定めた条件を満たすルールのみを Logical Coupling として検出する．

まず，同一 IRCT に属するコミットによって変更されたすべてのモジュールが一つのアイテムトランザクションに含まれるとみなして，相関ルールの生成を行う．最初に，各モジュール集合が何個の IRCT に出現したかを意味する，support count を求める．そして，ルールの前提部のモジュールの support count で，ルールの前提部及び帰結部のモジュールを含むモジュール集合の support count を割った値を求め，各ルールの confidence とする．こうして，同一 IRCT に一度以上含まれた，異なるリポジトリに含まれるすべてのモジュールの組み合わせについて，相関ルールを生成する．

こうして生成された相関ルールの中から，条件を満たすものを Logical Coupling として検出する．まず，相関ルールの support count と confidence の値に条件を設定する．そして，support count と confidence が定めた条件を満たしていた場合，それらのモジュール間には Logical Coupling の関係があると判定する．

3. Logical Coupling 検出のための予備調査

本節では，有用な Logical Coupling の検出に向けて行った予備調査について述べる．まず，3.1 節で調査の目的について述べ，3.2 節で調査の方法について説明する．次に，3.3 節で調査における各種条件について説明する．そして，3.4 節で調査の結果について述べ，3.5 節でその結果について考察する．

3.1 調査目的

本調査は，ソフトウェアプロダクト間での有用な Logical Coupling を検出するための予備的な調査である．そのためにまず，現在の手法を実際のソフトウェアリポジトリに適用した場合，どのようなモジュールが，どのような理由でコミットされ，Logical Coupling として検出されるのかを確認する．更に，それらの情報を元に，より有用な Logical Coupling を検出するための方法を検討する．

予備調査の目的を達成するため，Research Question を以下のように定めた．

RQ1. 検出された **Logical Coupling** について，ルールに含まれる二つのモジュールは本当に同じ目的で変更されるような関係にあるのか．

RQ2.Logical Coupling 検出の際，**confidence** のパラメタを変化させることによって，結果はどう変化したか．

3.2 調査方法

現在の手法を用いて検出された Logical Coupling について，ルールに含まれるモジュールに対して行われたコミットを確認することで，どのような目的でコミットが行われたのかを調査する．今回は IRCT の推定基準 1 についてのみ実装を行ったため，推定基準 1 を用いて推定した IRCT を用いて調査を行う．

まず，ルールに含まれる各モジュールに対して行われた，

同一 IRCT に含まれるコミットを確認し，それらがどのような目的で一緒に変更されたのかを確認する．同一 IRCT に含まれるコミットについて，そのコミットが行われた時間，コミットの内容，コミットコメントを確認し，それらのコミットがどのような理由で行われたものなのかを調べる．

また，検出された Logical Coupling が妥当なものであるかどうかを判断するため，コミットが同じ目的で行われたものであるかを評価する．この時，単一のリポジトリに対して行われた複数のコミットが同一の IRCT に含まれる場合，他方のリポジトリに対して行われたコミットと順番に比較することで，全てのコミットの組み合わせを確認する．これによって，現在の手法を用いて検出された Logical Coupling について，ルールに含まれるモジュールが同じ目的で変更されるような関係にあるのかを確認する．

同じ目的で行われたコミットであるかを判別するために，どのような条件を満たせば，二つのコミットが同じ目的で行われたものであるとみなすのかを判別する基準を決める必要がある．客観的な評価を行うためには，明確な基準を設定し，基準に沿って評価を行う必要があるためである．本調査では，同じ目的で行われたコミットを推定する基準として，以下のような基準を定めた．

- コミットの差分中に，もう一方のプロダクトに関する記述が存在すること．
- コミットの差分において，同様の内容と思われる変更が行われていること．
- コミットコメントが同様の意図で書かれたと思われるコメントが書かれていること．

上記の基準を一つ以上満たしている場合，二つのコミットは同じ目的で行われたものであると判断する．これらの基準では，明確に同じ目的で行われたコミットを判別することはできない．しかし，コミットの差分とコミットコメントを比較し，二つのコミットの関連する箇所が見つかった場合，二つのコミットが全く無関係なものではないことを確認することができると考え，このような基準を定めた．

3.3 調査条件

本節では，実験における各種条件について説明する．まず，今回の実験では GitHub にホストされた 10,000 件の Git[*1] リポジトリを対象とした．GitHub では，watch という機能を用いてリポジトリをブックマークすることができる．今回調査の対象としたリポジトリは，その watch 数の多い順に 10,000 件を選択した．また，10,000 件のリポジトリに対して行われた総コミット回数は 7,484,874 件であり，それらのコミットによって一度でも変更が行われた総ファイル数は 8,091,772 個であった．また，Git リポジトリからコミットログを取得する際，リポジトリのマスターブランチに対して行われたコミットログのみを取得している．

また，同一 IRCT に含まれるコミットであると推定する時間間隔を 24 時間とした．今回の調査では同じ開発者によって行われたコミットを IRCT として推定する．同じ開発者による変更であれば，変更の間に数週間といった時間は発生しないのではないかと考えたため，このような時間間隔を設定した．

最後に，相関ルール生成に関するパラメタについて述べる．相関ルール生成に関するパラメタとして，support count の閾値を 10 で固定し，confidence の値の条件を三つに分けて結果を確認した．confidence の値が 0.1 以上かつ 0.2 未満と低いもの, 0.5 以上かつ 0.6 未満と真ん中くらいのもの, 0.9 以上のものの三条件である．これらの confidence の値を持つルールを結果として生成し，ルールの confidence の値による違いを確認する．

3.4 結果の概要

調査対象として，相関ルール生成の際の各条件において検出された Logical Coupling のから 10 件ずつをランダムに選択した．support count の値が 10 以上で confidence の値が 0.9 以上のルールを表 1, support count の値が 10 以上で confidence の値が 0.5 以上かつ 0.6 未満のルールを表 2, support count の値が 10 以上で confidence の値が 0.1 以上かつ 0.2 未満のルールを表 3 に示す．一列目はランダムに選ばれた 10 件ルールの ID を表している．二列目は評価したコミットの組み合わせの数を示しており，三列目はそのコミットの組み合わせの内，同じ目的行われたと判定されたコミットの組み合わせの数を示している．四列目は同じ目的行われたと判定されたコミットの割合を示しており，五列目は 10 件のルールにおける同じ目的行われたと判定されたコミットの割合の平均値を表示してある．

今回，confidence を三つの領域に分けてそれぞれの confidence を持つルールを調査したが，評価の結果に大きな差は見られなかった．同じ目的で行われたと判定されたコミットの割合は，高い confidence を持つルールの方が若干高いという結果であった．

また，同一 IRCT に含まれるコミットについて，コメント及び差分が非常に類似しているものが多く見つかった．これらのコミットは，確認した全ての場合についてコミットされた時間が一致していた．このことから，二つのリポジトリが同じコミットを取り込んだか，元々同じリポジトリから派生したリポジトリであると推測される．

3.5 調査結果

RQ1. 検出された **Logical Coupling** について，ルールに含まれる二つのモジュールは本当に同じ目的で変更さ

[*1] Git. http://git-scm.com/

れるような関係にあるのか.

表 1, 2, 3 に，各 confidence 値を持つルールの評価結果を示した．同じ目的で変更されたと判定されたコミットの割合は全体的に高く，その平均値は confidence 値の低いルールであっても 7 割以上という結果が得られた.

実際にコミットコメントやコミットの差分を調査したところ，同じ目的で行われたと思われるコミットが多かった．同じ意図で書かれたと思われるコメントを持つコミット，差分の一部が共通するコミットの例を以下に示す.

同じ意図で書かれたと思われるコメントの例

- `Another pgindent run. Sorry folks.`
- `pgindent run over code.`

共通している差分の例

- `- printf("CREATED relation %s with OID %d\n",`
  `+ printf("CREATED relation %s with OID %u\n",`
- `- printf("CREATED relation %s with OID %d\n",`
  `+ printf("CREATED relation %s with OID %u\n",`

同じ意図で書かれたと思われるコメントの例では，pgindent と呼ばれる SQL のコードを整形するプログラムが動作したことが書かれている．このコメントから，別のプロダクトに対して同じ処理が行われたことが考えられる．また，共通している差分の例は，%d を %u に修正する内容であった．この差分から，これらのコミットによってプログラムの変数値の範囲の変更や表記の統一などが行われたと予想される.

このように，多くのコミットが同じ目的で行われており，検出された Logical Coupling について，ルールに含まれるモジュールは同じ目的で変更されていると考えられる．今回検出した Logical Coupling のルールに含まれるモジュールは，それぞれ別のコミットによって変更されていたとしても，それらのコミットの多くで内容が関連していることを確認した.

しかし，Logical Coupling が機能的に関連する有用なものであるかどうかを評価する際には，現在のままでは不十分であると言える．その理由として，現在の方法は同一の IRCT に含まれるコミットに，機能的な関連があるかどうかを考慮していないということがある．機能的に関連する有用な Logical Coupling を検出するためには，機能的に関連するコミットをまとめた IRCT を作成する方法を考える必要があると言える.

RQ2.Logical Coupling 検出の際，confidence のパラメタを変化させることによって，結果はどう変化したか.

今回の調査では，Confidence の条件を三種類に分け，各条件を用いて Logical Coupling の検出を行った．confidence 値が高い場合，低い場合，その中間の場合を用意し，それぞれどのような結果が出力されるかを確認した.

結果として，条件によって多少の差はあったが，モジュール同士が一緒に変更される関係にあるという判断に貢献したコミットの割合はそれぞれ，0.85, 0.81, 0.73 であった．一応，confidence 値が高いルールのほうが同じ目的で行われたと判定されたコミットの割合が高いが，あまり大きな差は見られなかった.

表 1 conf ≧ 0.9, #sup ≧ 10 を満たすルールの評価結果

rule ID	#commits	#valid	valid rate	avg valid rate
1	10	10	1.00	0.85
2	36	32	0.89	
3	47	41	0.87	
4	78	72	0.92	
5	16	14	0.88	
6	45	34	0.76	
7	16	14	0.88	
8	15	12	0.80	
9	10	10	1.00	
10	47	24	0.51	

表 2 0.6 ≧ conf ＞ 0.5, #sup ≧ 10 を満たすルールの評価結果

rule ID	#commits	#valid	valid rate	avg valid rate
1	13	10	0.77	0.81
2	14	13	0.93	
3	14	10	0.71	
4	10	10	1.00	
5	21	19	0.90	
6	12	10	0.83	
7	10	9	0.90	
8	19	15	0.79	
9	14	11	0.79	
10	21	9	0.43	

また，低い confidence 値を持つルールでは，ライセンスの変更やコードの整理のために行われたコミットが多く見受けられた．表 3 は confidence が 0.1 以上 0.2 未満のルールの評価結果を示しており，同表の三列目は同じ目的行われたと判定されたコミット数を示している．表 3 に示した 10 件のルールにおける同じ目的で行われたと判定されたコミットの合計は 145 件であったが，うち 116 件のコミットでは，ライセンスの変更かコード整形が主な変更内容であることが確認できた．今回の基準では同じ目的で行われたコミットと判定されたものの，あまり機能的に関連がないコミットの影響を受けて Logical Coupling が検出された結果ではないかと考えられる.

4. 関連研究

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 [7] の研究を紹介する．Gall らは，VCS のリポジトリに記録された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.

表 3　0.2 ≧ conf ＞ 0.1, #sup ≧ 10 を満たすルールの評価結果

rule ID	#commits	#valid	valid rate	avg valid rate
1	15	13	0.87	0.73
2	13	9	0.69	
3	11	11	1.00	
4	10	7	0.70	
5	29	24	0.83	
6	20	0	0.00	
7	28	20	0.71	
8	29	22	0.76	
9	34	26	0.76	
10	13	13	1.00	

Gall らの手法は，VCS に頻繁に一緒にコミットされるファイル間の関係を，Logical Coupling として検出するというものである．リポジトリから取得できるコミットトランザクションを一緒に変更されたモジュールと考え，頻繁に一緒に変更されるファイル間の関係を Logical Coupling として検出する．

しかし，Gall らが対象とした VCS である CVS[*2]は，コミットトランザクションを明示的に管理しない VCS であった．これは，複数のファイルを一斉にコミットしたとしても，それら複数のファイルに対するコミットが別々のものとして扱われるということを意味している．そのため，Gall らは各ファイルに対して行われた変更について，一緒に行われた変更を推測することで，コミットトランザクションの推定を行った．

Gall らは，以下の二つの基準を同時に満たす変更が同一のコミットトランザクションに属するとして，コミットトランザクションの推定を行った．

(1) 変更を行った開発者のユーザ名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．

続いて，Gall らの手法の手順は以下の通りである．まず，対象リポジトリのコミットログを確認し，各モジュールが変更された時間と変更者のユーザ名を取得する．そして，上記の基準を同時に満たすコミットの集合を同一のコミットトランザクションと見なして，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の全ての組み合わせを生成する．最後に，モジュールの組み合わせごとに出現回数をカウントし，その数が定めた閾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 らは実際のリポジトリに記録された開発記録を用いて，単一リポジトリ内でのモジュール間の関連を調査した．

また，Logical Coupling を利用した研究に，Wong ら [8] の研究がある．Wong らは，ソフトウェアのモジュール構造と Logical Coupling 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．UML から導出したモジュール構造を元に一緒に変更されると思われる関係を抽出し，モジュール構造的に関係のないモジュール同士が Logical Coupling である場合に，モジュール構造の違反として検出する．Wong らは二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．

最後に，複数のリポジトリを対象に Logical Coupling の検出を行った研究として，Fischer ら [9] の研究がある．Fischer らは，同じ製品グループに属する複数のリポジトリに対して，Logical Coupling の検出を行った．Fischer らは，各リポジトリの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，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．これにより Fischer らは，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を示した．

5. まとめと今後の課題

本稿では，ソフトウェアプロダクト間での Logical Coupling 検出するための方針を示した．異なるプロダクト間での Logical Coupling にはどのような状況が考えられるかを挙げた上で，それらの状況に合った検出手法を提案した．

そして，提案した方針に基づいて考案した手法をオープンソースソフトウェアのリポジトリに対して適用することで，有用な Logical Coupling 検出のための予備的な調査を行った．複数の OSS リポジトリから抽出したコミットログに手法を適用し，いくつかの Research Question に答えることで，現在の手法によって得られる結果とその問題点を確認した．

今後の課題は，本稿で説明した Logical Coupling の検出方法を見直し，より有用な Logical Coupling を検出できるようになることである．リポジトリとそのバックアップリポジトリとの間で見つかるような Logical Coupling や，あまり機能的に関連のないコミットによる繋がりによって検出される Logical Coupling は，あまり有用なものとは言えない．このような，有用ではない Logical Coupling を検出結果から除外するため，現在の手法を改良する．

謝辞　本研究は，若手（B）25730036 による．

参考文献

[1] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation open source software developers. In

[*2] CVS. http://www.nongnu.org/cvs/

Proc. Int'l Conf. Software Engineering, 2003.

[2] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proc. Int'l Conf. Software Maintenance*, 1998.

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

[4] 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*, 2005.

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

[6] 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.

[7] 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*, 2003.

[8] Sunny Wong, Yuanfang Cai, Miryung Kim, and Michael Dalton. Detecting software modularity violations. In *Proc. Int'l Conf. Software Engineering*, 2011.

[9] 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*, 2005.