Vol.2015-SE-190 No.17
2015/12/16

共変更の新しさと開発者の貢献度合いに着目したロジカルカップリングの評価 —ソースコード修正漏れの自動検出に向けて—

尾田 祐之介[1,a)]　阿萬 裕久[2]　川原 稔[2]

概要：ソフトウェアの開発・保守においては複数のソースコードが同時に変更される（共変更が起こる）ことが多い．その際，一部のコードに修正漏れがあると，システムに不具合が発生し，再修正が必要になってくる．そこで修正漏れを未然に防ぐため，ソースコード間のロジカルカップリングを検出するという手法が知られている．本稿では，従来からロジカルカップリングの検出に使われていた共変更の新しさに加え，別の視点として開発者の貢献度合いの類似性にも着目している．二つのモジュールが同じ開発者群から開発・保守されていた場合，過去に共変更の実績がない場合でもロジカルカップリングが存在している可能性があるというのが，開発者に着目している理由である．そして，これら二つの異なる視点から得られる評価を合成した新たなロジカルカップリングの評価方法（メトリクス）を提案している．三つのオープンソースソフトウェアを対象とした評価実験では，“共変更の新しさ”と“開発者による貢献度の類似性”の二つを合成した評価値を用いることで，共変更発生予測の精度向上が確認されている．

1. はじめに

一般にソフトウェアシステムは多数のモジュールで構成され，モジュール間での協調的な動作によってさまざまな機能が実現されている．つまり，モジュール一つ一つが独立して動作するだけではシステムとして成り立たず，システムによって程度の差はあるものの，モジュール間では一種の結合関係（カップリング）が必要になってくる．そのため，システムに新たなモジュールを追加したり，既存のモジュールを修正したりした場合，それと関係のある他のモジュールも修正しなければならないということが珍しくない．その際に修正すべき箇所を見落とすと，モジュール間である種の不整合が発生してしまい，システム障害を招いてしまう恐れがある．

修正漏れを未然に防ぐため，一つの修正が他の部分に及ぼす影響を分析する影響波及分析が従来から研究されている [1]．しかしながら，影響を及ぼす可能性を持った結合関係にはさまざまなものがあり [2]，可能性のあるものを全て考慮すると候補が膨大になってしまい有用な情報を提供できないという問題がある．さらには，二つのモジュール間に直接的な結合関係が無い場合であっても，両者の間にコードクローン [3] が存在し，結果的に両者は一緒に修正されなければならない場合もある [4]．この場合も従来から知られている影響波及分析法では検出が難しい．

これに対し，プログラムの内容を解析するのではなく，プログラムの変更履歴情報を使ってモジュール間の結合関係を評価するロジカルカップリングという考え方がある [5]．端的に言えば，過去に二つのモジュールが同時に修正されたとすると，それらの間には何らかの関連性があったものと推察される．その実績データをもとにして，モジュール間の結合度合いを評価しようというものである．この評価法は近年，リポジトリマイニング分野 [6] において盛んに研究されているテーマの一つであり [7]，ある程度の変更履歴情報が蓄積した後でなければ使えないという問題はあるが現実的には修正漏れを未然に防ぐ有効な手法の一つである [8]．また近年，リポジトリマイニングの分野では，プログラムのオーナーシップに着目した研究も行われるようになってきている [9]．即ち，各モジュールの開発・保守において，誰がどれだけ貢献しているかという観点に立った研究である．筆者らの知る限り，これまでロジカルカップリングと開発者の貢献度の関係については十分に議論されていない．そこで本稿では，過去に二つのモジュールが同時

1 愛媛大学大学院理工学研究科
Graduate School of Science and Engineering, Ehime University, 3, Bunkyo-cho, Matsuyama 790–8577, Japan
2 愛媛大学総合情報メディアセンター
Center for Information Technology, Ehime University, 3, Bunkyo-cho, Matsuyama 790–8577, Japan
a) yunosuke@se.cite.ehime-u.ac.jp

ⓒ 2015 Information Processing Society of Japan

に修正された（共変更が起こった）という情報だけでなく，各モジュールにおける各開発者の貢献度も考慮し，二つの情報を組み合わせた新たなロジカルカップリングのメトリクスを提案する．

本稿の構成は以下の通りである．まず，2 章でロジカルカップリングと開発者の貢献度について簡単に紹介し，その上で 3 章にてロジカルカップリングの評価法について説明・提案を行う．そして，4 章で評価実験を行い，最後に 5 章で本稿のまとめと今後の課題について述べる．

2. ロジカルカップリングと開発者の貢献度

本稿では，二つのモジュールが同時に変更されるという事象を “共変更” と呼ぶことにし，その発生を二つの異なる視点から予測することを考える．本章では手法の提案に先立って，関連する先行研究について簡単に紹介する．

ソフトウェアの開発・保守において，あるモジュールに対して修正を加えた場合に別のモジュールにも同時に修正を加えなければならないことがある．一般にそのような関係は，モジュール間の結合関係（カップリング）と呼ばれる．結合関係には，モジュール間での呼び出し関係やデータの授受といった直接的な関係だけでなく，両者が同じデータ領域にアクセスしていたり，プログラムの一部がコードクローン関係にあったりするといった別の観点での関係も存在する．したがって，一般には “プログラムを解析するだけではすべての結合関係を検出できない” という難しさが結合関係の検出にはある．

そこで，プログラムの解析とは別の視点に立った研究がいくつか行われている．代表的なものとして，プログラムの変更過程において “二つのモジュールが同じタイミングで変更される（共変更される）” という事象を着目し，これをロジカルカップリングと呼んで予測（検出）の対象とする研究がリポジトリマイニング分野を中心として活発に行われている [5], [7], [8], [10]．ロジカルカップリングの基本的な評価法としては，モジュール対において過去に発生した共変更の回数が用いられる．つまり，過去に何度も同じ組のモジュールが共変更されてきたならば，その後もまた共変更されやすいであろうというものである．例えば，Gall ら [5] は，変更履歴における共変更の発生とそこでの変更ログ（メッセージ）に着目してロジカルカップリングを検出しようとしている．Mondal ら [10] は，ソースファイルの中で使われている識別子の一致度を用いてロジカルカップリングを検出する手法を提案している．そこでは，二つのメソッドの変更箇所において，同じか似た意味の単語が識別子に使われているかどうかでもってロジカルカップリングを評価している．そのような識別子の一致度を用いることで，過去に発生した共変更の回数が少ない場合に対応しようとしている．

また近年，モジュールに対するオーナーシップに注目した研究も報告されている．Posnett ら [9] は，エントロピーの概念を使って各ソースファイルにおける各開発者のオーナーシップ（貢献度）を評価し，品質評価に役立てようとしている．実際，誰がどのモジュールの開発・保守にどれだけの貢献をしているかという情報は，その後の保守においても重要になってくると思われる．そのことはロジカルカップリングの検出にも有用であると期待される．例えば，二つのモジュールがそれまで同一人物によって開発・保守されてきたものであれば，過去に共変更の発生実績がなくとも，また，識別子やコードクローンといった別の視点で関連性が見られなくとも，両者の間にロジカルカップリングが存在している可能性は否定できない．筆者らの知る限り，これまでロジカルカップリングと開発者の貢献度の関係について，定量的・実証的な研究はほとんど行われていない．そこで本稿では，従来からロジカルカップリングの研究で使われている共変更の概念だけでなく，開発者のモジュール開発・保守に対する貢献度についても考慮した新たなロジカルカップリングの評価法（メトリクス）について提案していく．

3. 提案手法

本章では，共変更の新しさと開発者の貢献度の二つを考慮してロジカルカップリングを検出する手法を提案する．まず準備段階として共変更の新しさと開発者の貢献度について説明し，これら二つを考慮したロジカルカップリングの検出方法を提案する．

3.1 準備

3.1.1 共変更の新しさに着目したロジカルカップリングの検出

ロジカルカップリングの検出では，“過去に共変更が何回発生したか” という変更回数が基本となっており [5]，さらにはそこに “いつ共変更が発生したか” という時間の概念も採り入れた手法が広く知られている．例えば，（ロジカルカップリングを直接評価するものではないが，）Google Bugspots[11] では，バグ修正の回数に修正時刻の重みをつけ，バグ修正が最近になって頻繁にされているモジュールに対して高いスコアをつけている．

本稿でも，同様の概念に基づいて共変更の新しさに着目したロジカルカップリングの検出方法を考える．まず，二つのモジュール m_1, m_2 が現在までに $N(\geq 0)$ 回一緒に変更されたとする．そして，その中の古い方から数えて k 番目の変更が起こった正規化時刻を t_k とする（$k = 1, \ldots, N$）．ただし，正規化時刻とは，モジュール m_i と m_j の中で後から作られた方の新規作成時刻を 0 とし，現在の時刻を 1 とした時刻である．本稿では，共変更の新しさを正規化時刻 t_k の 1 への近さ（t_k の値の大きさ）で表す．そして，

表 1 2 個のモジュール m_1, m_2 を 3 人の開発者 d_1, d_2, d_3 で開発・保守していた場合の例

時刻	イベント	開発者	モジュール	変更行数
2015/4/14 15:30:00	新規開発	d_1	m_1	40
2015/4/16 10:30:00	新規開発	d_1	m_2	30
2015/4/18 09:00:00	変更	$d1$	m_1	15
2015/4/18 14:00:00	共変更	$d1$	m_1	10
	共変更	$d1$	m_2	20
2015/4/19 11:30:00	変更	$d2$	m_1	30
2015/5/10 16:30:00	変更	$d1$	m_2	5
2015/5/14 13:00:00	共変更	$d2$	m_1	10
	共変更	$d2$	m_2	15
2015/7/3 15:30:00	変更	$d3$	m_1	10
2015/7/6 14:00:00	変更	$d3$	m_1	5
2015/7/7 09:30:00	変更	$d3$	m_1	10
2015/7/10 16:30:00	共変更	$d2$	m_1	5
	共変更	$d2$	m_2	10
2015/7/14 14:00:00	共変更	$d2$	m_1	10
	共変更	$d2$	m_2	20

次式で表される $S_{co}(m_i, m_j)$ をモジュール m_i と m_j の間での関連性の強さと定義する．

$$S_{co}(m_i, m_j) = \frac{1}{M}\sum_{k=1}^{N} t_k \ , \qquad (1)$$

ただし，M は m_i と m_j のそれまでリポジトリでの登場回数の大きい方とする．$S_{co}(m_i, m_j)$ は，m_i（または m_j）のこれまでの開発・変更の中で共変更であったものの重み（正規化時刻）の平均を表している．この値が 1 に近いほど，そのモジュールで起こった共変更は “新しい” といえる．

表 1 に 2 個のモジュール m_1, m_2 を 3 人の開発者 d_1, d_2, d_3 で開発・保守していた場合の例を示す．この例では 4 回の共変更が発生しており，モジュール対が出そろった “2015/4/16 10:30:00” を時刻 0，“2015/7/14 14:00:00” を時刻 1 として正規化すると，それぞれの共変更の発生時刻は次のように正規化時刻で表される：

- “2015/4/18 14:00:00” は 0.024071
- “2015/5/14 13:00:00” は 0.315261
- “2015/7/10 16:30:00” は 0.956298
- “2015/7/14 14:00:00” は 1.0

ここで, モジュール m_1 の総登場回数は 10 回, m_2 の総登場回数は 6 回であることから $M = 10$ として，（1）式より $S_{co}(m_1, m_2) = (0.024071+0.315261+0.956298+1.0)/10 = 0.229563$ が得られる．

3.1.2 開発者の貢献度に着目したロジカルカップリングの検出

3.1.1 節では二つのモジュール m_i，m_j 間で共変更が起こった時刻に着目し，新しい共変更が多く起こっているほどモジュール間の結びつき，即ちロジカルカップリングの度合いは強いと考え，その強さを $S_{co}(m_i, m_j)$ として定義した．ここでは次に，別の視点に立ったロジカルカップリングの検出方法についても考える．

一般に，各モジュールの作成・変更にはさまざまな開発者が携わっている．その中で，ロジカルカップリングとなっているモジュール間では，同一開発者によって短期間または同時に変更が施されることが多いという報告がある [12]．そこで，同一の開発者が開発に携わっているモジュール間には（前述した）共変更の新しさとは違った属性の関連性があると考えられる．本稿では，各開発者の各モジュールに対する貢献度合いに着目することにする．つまり，共変更の回数が多いわけではないが，過去の変更履歴において二つのモジュールで開発者ごとの貢献度合いが “似ている” ならば，それもまた二つのモジュール間の関係が深いものと考える．

ここでは，各開発者の貢献度を当該モジュールに対する変更行数で定量化し，貢献度合いの類似性をコサイン類似度で評価する．コサイン類似度は，二つのベクトルに対してコサインを求めることで，ベクトルの方向，即ちベクトルで表現される二つの対象物の特徴がどれだけ近いのかを表す指標となる [13]．例えば，自然言語処理の分野では，文章に出現する単語の特徴をベクトルで表し，そのコサイン類似度を使って文章の類似性を評価するといったことが行われている [14]．

各モジュールにおける開発者の貢献度をベクトルで表現し，それらのコサイン類似度を使ってモジュール間のロジカルカップリングの強さを定義する．いま，分析対象のソフトウェア開発プロジェクトに $D(\geq 1)$ 人の開発者 d_p がいるものとする（$p = 1, \ldots, D$）．そして，モジュール m_i がこれまでにリポジトリへ N 回コミットされたとす

表 2 調査対象

ソフトウェア名	ファイル数	LOC	開発者数	開発言語	調査期間
Angry IP Scanner	444	10K	7	Java	2006-07 ∼ 2014-07
MediathekView	399	40K	2	Java	2012-02 ∼ 2014-07
Nakloid	41	4K	2	C++	2012-07 ∼ 2014-03

る．ここで，開発者 d_p のモジュール m_i に対する貢献度 $c(m_i, d_p)$ を次式で定義する：

$$c(m_i, d_p) = \sum_{k=1}^{N} l_k(m_i, d_p) \ , \qquad (2)$$

ただし，$l_k(m_k, d_p)$ はモジュール m_i について行われた（古い方から数えて） k 番目のコミットにおいて，開発者 d_p が施した変更の行数である．

モジュール m_i に対して D 人すべての開発者の貢献度は，次の D 次元ベクトルとして表現できる：

$$\boldsymbol{c}(m_i) = (\ c(m_i, d_1),\ \cdots,\ c(m_i, d_D)\ )\ . \qquad (3)$$

そして，二つのモジュール m_i, m_j の間での “開発者の貢献度” のコサイン類似度は次式で算出できる．本稿ではこれを，開発の貢献度に着目したロジカルカップリングの度合い $S_{own}(m_i, m_j)$ と定義する：

$$S_{own}(m_i, m_j) = \frac{\boldsymbol{c}(m_i) \cdot \boldsymbol{c}(m_j)}{|\boldsymbol{c}(m_i)||\boldsymbol{c}(m_j)|}\ . \qquad (4)$$

ここで再び， 表 1 に示した例について考える．まず，(2) 式について

- $c(m_1, d_1) = 25 (= 15 + 10)$
- $c(m_1, d_2) = 55 (= 30 + 10 + 5 + 10)$
- $c(m_1, d_3) = 25 (= 10 + 5 + 10)$
- $c(m_2, d_1) = 25 (= 20 + 5)$
- $c(m_2, d_2) = 45 (= 15 + 10 + 20)$
- $c(m_2, d_3) = 0$

であるため，$\boldsymbol{c}(m_1) = (25, 55, 25)$，$\boldsymbol{c}(m_2) = (25, 45, 0)$ が得られる．よって，(4) 式より $S_{own}(m_1, m_2) = 0.921023$ を得る．

3.2 提案手法：共変更の新しさと開発者の貢献度の両方を考慮したロジカルカップリングの検出

前小節では，二つのモジュール m_i, m_j における共変更の新しさに着目したロジカルカップリングの度合いを $S_{co}(m_i, m_j)$ と定義した．さらには別の視点として，二つのモジュール開発・保守に関与している開発者の貢献度に着目したロジカルカップリングの度合いを $S_{own}(m_i, m_j)$ と定義した．これらは異なる視点からモジュール間のロジカルカップリングの度合いを定量化しようとしたものである．本稿では，これら二つのメトリクスを合成した新たなメトリクスを提案し，提案メトリクスによって共変更発生の予測を行う．

合成に先立って，それぞれのメトリクス値の標準化を行う．これは，メトリクス値の分布の偏りによって，合成値がどちらか一つのメトリクスに支配されることのないようにするための措置である．具体的には，二つのメトリクスそれぞれの値のプロジェクト内での平均値と標準偏差を用い，次のように標準化したメトリクス値 $NS_{co}(m_i, m_j)$ 及び $NS_{own}(m_i, m_j)$ を考える：

$$NS_{co}(m_i, m_j) = \frac{S_{co}(m_i, m_j) - \mu_{co}}{\sigma_{co}}\ , \qquad (5)$$

$$NS_{own}(m_i, m_j) = \frac{S_{own}(m_i, m_j) - \mu_{own}}{\sigma_{own}}\ , \qquad (6)$$

ただし，μ_{co} 及び σ_{co} はそれぞれ $S_{co}(m_i, m_j)$ のプロジェクト内での平均と標準偏差であり，同様に μ_{own} 及び σ_{own} はそれぞれ $S_{own}(m_i, m_j)$ のプロジェクト内での平均と標準偏差である．

そして，二つの標準化されたメトリクス値を α:$(1-\alpha)$ の比率で合成した $S_\alpha(m_i, m_j)$ を新たなメトリクスとする：

$$\begin{aligned} S_\alpha(m_i, m_j) = \ & \alpha \cdot NS_{co}(m_i, m_j) \\ & +(1-\alpha) \cdot NS_{own}(m_i, m_j)\ , \end{aligned} \qquad (7)$$

ただし，α は 0 以上 1 以下の実数であり，次章ではこれをパラメータとした共変更予測の実験を行う．

4. 実験

本章では，オープンソースソフトウェアの開発を対象として，そこでの共変更の発生を提案メトリクスによって予測できるかどうかを検証する．

4.1 対象

本実験では三つのオープンソースソフトウェア Angry IP Scanner[*1], MediathekView[*2] 及び Nakloid[*3] を対象とした（**表 2**）．

今回は，2 名の開発者のみで開発されていて比較的新しいプロジェクト（MediathekView 及び Nakloid ; 執筆時点で開発期間が約 3 ∼ 4 年程度）とより多くの開発者によって開発されていて比較的古くから続いているプロジェクト（Angry IP Scanner ; 開発期間が 10 年程度）を選んだ．なお，データ収集の都合上，いずれもソースファイルのバー

*1 http://angryip.org/
*2 http://zdfmediathk.sourceforge.net/
*3 https://www.openhub.net/p/nakloid

ジョン管理には Git を使っているものを対象とした．

4.2 方法

本実験では，提案メトリクスによってロジカルカップリングが強いと判定されたファイル対で実際に共変更が発生するかどうかを検証する．

まず，対象ソフトウェアの開発において，最初のコミットから現在までの開発期間における前半の 2/3 の時刻までの期間を “学習期間” とし，この期間の履歴を使って各モジュール間でのロジカルカップリングの度合い S_{co} 及び S_{own} を算出する．ここで，学習期間の最終時刻を “測定時刻” と呼ぶことにする．

次に，測定時刻から現在までの期間（全開発期間の後半 1/3）を “検証期間” とし，各モジュール対で共変更が起こっているかどうかを調べる（**図 1**）．

そして，S_{co} と S_{own} をそれぞれ NS_{co} と NS_{own} に標準化し，これらを合成した S_α を各モジュール対について算出し，S_α の降順に共変更が起こっていたかどうか（起こっていた場合は 1，起こっていなかった場合は 0）を累積していく（**図 2**）．累積値のグラフの立ち上がりが早く，曲線下の面積が大きいほど，使用したロジカルカップリングの度合い（S_α）が共変更の発生予測に有用であることを意味する．この検証を，パラメータ α を 0 から 1 まで 0.1（10%）刻みで変化させながら 11 通り行う．

4.3 結果

結果の一部として，Nakloid に対し S_α の降順を横軸，

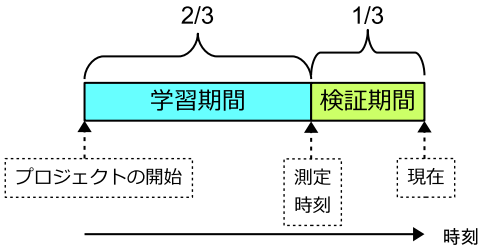

図 1 学習期間と検証期間

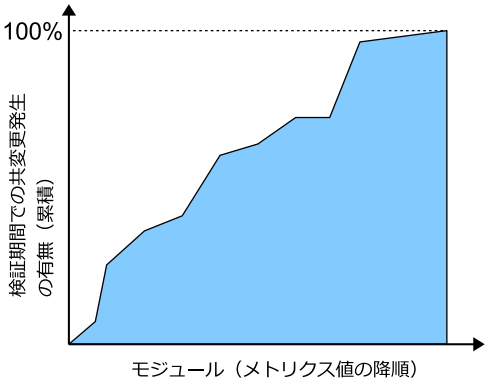

図 2 共変更発生の有無に関する累積値グラフの例

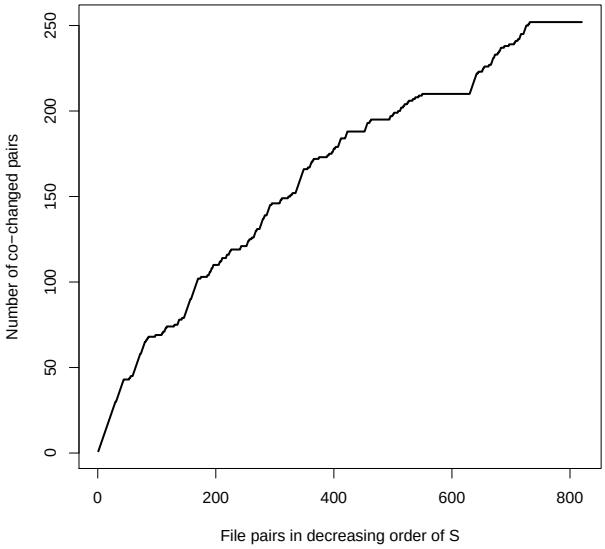

図 3 Nakloid での共変更発生組数の累積（$\alpha = 0$）

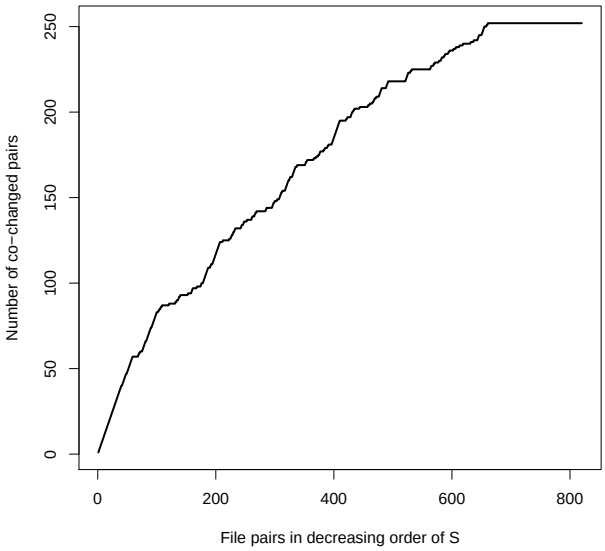

図 4 Nakloid での共変更発生組数の累積（$\alpha = 0.5$）

共変更が発生した組数の累積値を縦軸にとった累積グラフを**図 3**, **図 4** 及び**図 5** に示す．これらはそれぞれ $\alpha = 0$ の結果，$\alpha = 0.5$ の結果及び $\alpha = 1$ の結果に対応している．つまり，開発者の貢献度のみに着目してロジカルカップリングの度合いを定量化した場合（$\alpha = 0$），開発者の貢献度と共変更の新しさの平均値をロジカルカップリングの度合いとした場合（$\alpha = 0.5$）及び共変更の新しさのみに着目してロジカルカップリングの度合いを定量化した場合（$\alpha = 1$）である．

図 3 ～ 図 5 を重ね合わせたグラフを **図 6** に示す．$\alpha = 0.5$ の結果と $\alpha = 1$ の結果は同一であったために一つ

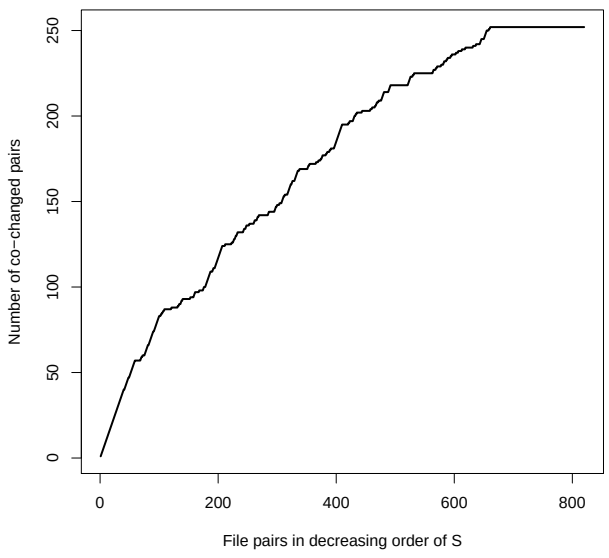

図 5 Nakloid での共変更発生組数の累積（$\alpha = 1$）

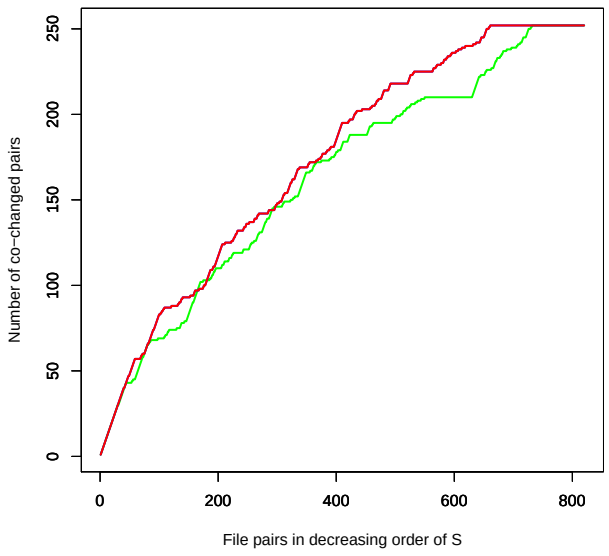

図 6 Nakloid での共変更発生組数の累積（緑 : $\alpha = 0$，赤 : $\alpha = 0.5$ 及び $\alpha = 1$）

にまとめてある．図 6 から，グラフの形状がパラメータによって劇的に変化することはなく，細かな差にとどまっていることが分かる．実際，他のパラメータや他のソフトウェアでも同様の傾向にあった．

次に，より定量的に比較を行うため，累積グラフにおける曲線と横軸に挟まれた領域の面積を算出した結果を **表 3** に示す．下線を引いた値はそのプロジェクト内での最大値である．ただし，ここでは比較のため，グラフの原点を左下の頂点，グラフの最終点（収束点）を右上の頂点とした長方形を考え，その面積を 1 とした値に正規化したもの

表 3 共変更の発生に関する累積グラフの面積

α	Angry IP Scanner	MediathekView	Nakloid
0.0	0.6370951	0.5923530	0.6466173
0.1	<u>0.6389693</u>	0.6012958	<u>0.6874226</u>
0.2	0.6389629	0.6022343	<u>0.6874226</u>
0.3	0.6389619	0.6026272	<u>0.6874226</u>
0.4	0.6389599	0.6029062	<u>0.6874226</u>
0.5	0.6389600	0.6030402	<u>0.6874226</u>
0.6	0.6389598	0.6031380	<u>0.6874226</u>
0.7	0.6389595	<u>0.6031957</u>	<u>0.6874226</u>
0.8	0.6389595	0.6028737	<u>0.6874226</u>
0.9	0.6389502	0.6028542	<u>0.6874226</u>
1.0	0.6389502	0.6027842	<u>0.6874226</u>

表 4 NS_{co} と NS_{own} の間の相関係数と無相関検定の結果

ソフトウェア	r	ρ
Angry IP Scanner	0.353	0.468
	($p < 2.2 \times 10^{-16}$)	($p < 2.2 \times 10^{-16}$)
MediathekView	0.260	0.211
	($p < 2.2 \times 10^{-16}$)	($p < 2.2 \times 10^{-16}$)
Nakloid	0.525	0.297
	($p = 9.61 \times 10^{-10}$)	($p < 2.2 \times 10^{-16}$)

を用いている．この正規化により，ソフトウェアごとでのファイル数に依存しない比較が可能である．

Angry IP Scanner では $\alpha = 0.1$，MediathekView では $\alpha = 0.7$，そして Nakloid では $\alpha = 0.1 \sim 1$（$\alpha = 0$ を除くすべて）において，面積が最大となった．

4.4 考察

まず，全プロジェクトに共通して見られた傾向として，パラメータ α の値の変化に対して，累積グラフの面積が大きく変化することはなかった．このことはグラフの形状からも予想されるものであったが，定量的な比較でもその差の小ささは明らかになり，プロジェクトごとでの面積の“最大値と最小値の差”を最小値で割った変動率は Angry IP Scanner で 0.3%，MediathekView で 1.8%，Nakloid で 6.3% であった．

そこで二つの（標準化後の）メトリクス $NS_{co}(m_i, m_j)$ と $NS_{own}(m_i, m_j)$ の間の相関係数をピアソンの相関係数 r とスピアマンの相関係数 ρ の両方について計算した．結果を **表 4** に示す．

表 4 から分かるように，無相関検定の結果から“相関なし”とは言えないが，強い相関関係は二つのメトリクスの間には見られなかった．したがって，二つのメトリクスが同じ結果を示していたわけではないことが確認された．

つまり，今回合成に用いた二つのメトリクスが異なる視点からロジカルカップリングの強さを評価していたことがうかがえる．そして，効果の差は小さいものの，両者を合成したメトリクスが最大の面積を示していたことを考えると，二つのメトリクスのいずれかで強いロジカルカップリ

表 5　$S_{own}(m_i, m_j) = 0$ となる組の割合

ソフトウエア	$S_{own}(m_i, m_j) = 0$ となる組の割合
Angry IP Scanner	71.5%
MediathekView	88.3%
Nakloid	4.5%

ングとして評価されたファイル対では，実際に共変更が起こりやすかったものと思われる．

今回の実験結果では，表 3 に見られるように Nakloid に関しては，開発者の貢献度のみを用いた $\alpha = 0$ の場合を除き，どのパラメータでも面積が最大となっていた．そこで，三つのソフトウェアにおける開発者の貢献度に基づいたメトリクス $S_{own}(m_i, m_j)$ について追跡調査したところ，**表 5** に示すように Nakloid だけはメトリクス値 $= 0$ となる組が他に比べてかなり少ないことが分かった．つまり，Nakloid に関しては，ほとんどのファイル対において開発者の貢献度に類似度（コサイン類似度）が見られ，結果としてほとんどのファイル対で強度の差はあれど，ロジカルカップリング有りと判定されてしまったことが原因の一つと考えられる．

4.5　妥当性への脅威

本稿では，実験対象として三つのオープンソースソフトウェアを使用した．実験結果の有用性を損なわないよう，比較的著名なものを対象として選定したが，結果として小規模で少人数によって開発されているプロジェクトだけになってしまっている．したがって，より規模の大きなプロジェクトやより多くの開発者が携わっているプロジェクトでは傾向が変わってしまう可能性は否定できない．つまり，結果の一般性については，今回の実験だけでは弱いと言わざるを得ない．しかしながら本稿の主旨は，共変更の新しさと開発者の貢献度という二つの異なる視点を合成したロジカルカップリング評価法の提案とその適用可能性の検討であり，今回の実験でその可能性は確認できたと考える．

実験において，共変更の理由については追跡できていないため，本研究で目指している “同時修正の漏れ” の検出に提案手法が真に有効であったのかどうかまでは今回は検討できていない．しかしながら，実際に共変更が発生している組の予測には概ね成功しており，対象プロジェクトにおいて同時修正漏れが多発していない限り，本手法の妥当性を損なうものではないと考える．

実験対象プロジェクトでの開発言語は Java と C++ のみとなっており，他の開発言語の場合については十分に考慮できていない．しかしながら，Java と C++ は実際に多くのプロジェクトにおいて使われている主要なプログラミング言語であり，これらの基本的な文法や構造は，他の手続き型言語やオブジェクト指向言語と大きな差はないため，そのような開発言語の違いが結果の妥当性を脅かす可能性は低いと思われる．

5.　おわりに

本稿では，ソフトウェアの開発・保守の過程において異なるソースファイル間での共変更の発生を予測するため，次の二つの視点に立ったロジカルカップリングを考えた．

(1) ファイル間で過去に発生した共変更の新しさ
(2) それまでの開発における各開発者の貢献度の類似性

（1）は従来から多くの研究で考えられている基準である．比較的近い過去に二つのファイルが同時修正された場合，両者の間には一種の関連性があると考えられ，それゆえ近い将来にも再び両者は同時に修正されることが予想されるというものである．

一方，（2）はソースファイルのオーナーシップに基づいた考え方であり，二つのファイルの開発・保守に携わっている開発者（個人または集団）が同じであり，開発者の貢献度合いも類似しているようならば，ファイル間には（1）とはまた違った視点での関連性があると考えられる．

本稿では，（1）と（2）をそれぞれ定量化して線形結合した融合メトリクスを提案した．そして，三つのオープンソースソフトウェアの開発を対象として，評価実験を実施した．その結果，実際の開発において（1）と（2）は異なる視点での関連性（ロジカルカップリング）を評価しており，両者を合成することでより効率的に共変更の発生を予測できることが分かった．今後，より多くのプロジェクトに対して評価実験を行い，提案手法の有効性・一般性を確認していく予定である．また，今回は（1），（2）とも単純なメトリクスを用い，合成も単純な線形結合にとどまっており，メトリクスや合成方法についてもさまざまな種類を試してみる必要があると考えている．

参考文献

[1] Bohner, S. A. and Arnold, R. S. eds.: *Software Change Impact Analysis*, IEEE Computer Society Press, Los Alamitos, CA (1996).
[2] Pressman, R. S.: *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Columbus, OH (2005).
[3] 井上克郎, 神谷年洋, 楠本真二：コードクローン検出法, コンピュータソフトウェア, Vol. 18, No. 5, pp. 529–536 (2001).
[4] Barbour, L., Khomh, F. and Zou, Y.: An empirical study of faults in late propagation clone genealogies, *Journal of Software: Evolution and Process*, Vol. 25, No. 11, pp. 1139–1165 (2013).
[5] Gall, H., Hajek, K. and Jazayeri, M.: Detection of logical coupling based on product release history, in *Proc. International Conference on Software Maintenance*, pp. 190–198 (1998).
[6] Hassan, A. E.: The Road Ahead for Mining Software Repositories, in *Proc. Frontiers of Software Maintenance (FoSM)*, pp. 48–57 (2008).
[7] Zimmermann, T., Zeller, A., Weissgerber, P. and Diehl, S.: Mining version histories to guide software

changes, *IEEE Transactions on Software Engineering*, Vol. 31, No. 6, pp. 429–445 (2005).

[8] 早瀬康裕, 中村高士, 天笠俊之, 北川博之：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, 情報処理学会論文誌, Vol. 56, No. 2, pp. 657–668 (2015).

[9] Posnett, D., D'Souza, R., Devanbu, P. and Filkov, V.: Dual ecological measures of focus in software development, in *Proc. 35th International Conference on Software Engineering*, pp. 452–461 (2013).

[10] Mondal, M., Roy, C. K., Schneider, K., et al.: Improving the detection accuracy of evolutionary coupling by measuring change correspondence, in *Proc. 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering*, pp. 358–362 (2014).

[11] Bug Prediction at Google, http://google-engtools.blogspot.jp/2011/12/bug-prediction-at-google.html.

[12] 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 2005 論文集, pp. 104–112 (2005).

[13] 徳永健伸：情報検索と言語処理, 東京大学出版会, 東京 (1999).

[14] 石田基広：R によるテキストマイニング入門, 森北出版, 東京 (2008).

ⓒ 2015 Information Processing Society of Japan