# 不適切に分割されたコミットに関する研究

有馬 諒　肥後 芳樹　楠本 真二

バージョン管理システムにおいて各コミットは 1 つのタスクからなるべきだと言われているが，1 つのコミットに複数のタスクが含まれているものや，1 つのタスクが複数のコミットに分割されているものも存在する．本研究では後者のコミットを IP コミットと呼ぶ．本研究ではまずどのような IP コミットがリポジトリに含まれているかを調査し，見つかった IP コミットを 3 種類に分類した．次に IP コミットの自動検出手法として，メソッドを頂点としたグラフを用いる手法を提案した．2 つのオープンソースソフトウェアのリポジトリに提案手法を適用した実験では適合率 0.8，F 値 0.7 の性能を示し，提案手法が IP コミットの検出に有効であることを示した．

Each commit in repositories of version control systems should include code changes for only a single task. However, in real repositories, there are many commits for multiple tasks and tasks split into multiple commits. We call the latter IP commits (inappropriately partitioned commits). In this research, we firstly investigate how many and what kinds of IP commits are included in repositories. Then, we classify the found IP commits into three categories. Based on the classification, we propose a new technique to detect IP commits automatically. This is the first research that proposes a technique to detect IP commits. To evaluate the proposed technique, we applied it to repositories of two open source software. The results showed that the proposed technique detected IP commits with high accuracy (precision is 0.8 and F-measure is 0.7).

## 1 はじめに

バージョン管理システムを利用したソフトウェア開発において，1 つのコミットにどれだけの変更内容を含めるべきかという問題がある．この指針としてタスクレベルコミットという考え方がある [1]．これは，タスクと呼ばれる機能追加やバグ修正のような意味のある変更内容が，1 つのコミットに 1 つだけ含まれるようにする方法である．Git のドキュメント [3] においても同様に，各コミットに含まれる変更が論理的に独立しているべきであると述べられている．

タスクレベルコミットが推奨されているのにもかかわらず，実際のリポジトリには複数のタスクを含む大規模なコミットが含まれていることが知られている．例えば，Murphy-Hill らは，リファクタリングは他のタスクと同じコミットに含まれることが多いことを明らかにしている [6]．このような大規模なコミットがリポジトリ分析の性能に悪影響を及ぼすことが明らかになっており [4]，大規模なコミットを複数のコミットに分割する手法について研究がなされている [4] [5]．

しかしこのようなコミットとは反対に，1 つのタスクが複数のコミットに分割されている場合については，これまでに調査されていない．分割されたコミットの例を図 1 に示す．これは，オープンソースソフトウェアである Apache Commons Collections[†1] のリポジトリに含まれるコミットである．図において + で始まる行はそのコミットで追加された行，- で始まる行はそのコミットで削除された行を表す．(a) のコミットではメソッドの戻り値を Read-only にする変

Investigating Inappropriately Partitioned Commits
Ryo ARIMA, Yoshiki HIGO, Shinji KUSUMOTO, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University.
[研究論文 (レター)] 2018 年 4 月 11 日受付.

†1 https://github.com/apache/commons-collections

```
  /**
   * Returns the values for the BeanMap.
   *
   * @return values for the BeanMap.
  Modifications to this collection
   *         do not alter the underlying BeanMap.
   */
  public Collection values() {
    ArrayList answer = new ArrayList( readMethods.
  size() );
    for ( Iterator iter = valueIterator();
  iter.hasNext(); ) {
      answer.add( iter.next() );
    }
-   return answer;
+   return Collections.unmodifiableList(answer);
  }
```

(a) 2002/03/25 06:53 のコミット

```
  /**
   * Returns the values for the BeanMap.
   *
-  * @return values for the BeanMap.
  Modifications to this collection
-  *         do not alter the underlying BeanMap.
+  * @return values for the BeanMap.  The returned
  collection is not
+  *         modifiable.
   */
  public Collection values() {
     ArrayList answer = new ArrayList( readMethods.
  size() );
     for ( Iterator iter = valueIterator(); iter.
  hasNext(); ) {
         answer.add( iter.next() );
     }
     return Collections.unmodifiableList(answer);
  }
```

(b) 2002/03/25 07:00 のコミット

**図 1　分割されたコミットの例**

更が行われているが，それに合わせたコメントの修正は (b) のコミットで行われている．本研究ではこのように不適切に分割されたコミットを inappropriately partitioned commit (*IP* コミット) と呼ぶ．

IP コミットの問題点として以下のものが考えられる．

**リポジトリマイニングの性能低下：** ロジカルカップリング [2] とは，ソフトウェアのあるモジュールを変更する場合にはそのモジュールと同時に変更されるべき別のモジュールが存在する，というモジュール間の論理的な依存関係であり，リファクタリングの推薦 [2] や修正漏れの検知 [7] に利用されている．森らは同一作業コミットを統合することによって，ロジカルカップリングを用いた変更漏れの推薦の性能が向上することを明らかにしており [8]，IP コミットが存在すると，本来検出されるべきモジュール間の論理的な依存関係が検出されなくなってしまうため，ロジカルカップリングの検出精度が低くなってしまう恐れがある．このように，リポジトリマイニング手法の中には，IP コミットの存在が悪影響を受けるものがある．そのような手法を IP コミットを含むリポジトリに適用した場合，期待する性能を得ることができない．

**コミット履歴の可読性の低下：** ソースコードの変更内容や変更者，コメントなどがコミットには含まれており，コミット履歴をたどると誰がどのような意図をもって変更を加えたのかを知るのに役に立つ．しかし 1 つのタスクが複数のコミットに分割されている場合，コミット履歴の可読性が低下しコミット履歴を用いた変更内容の理解が難しくなる．

本研究ではまず，実際のリポジトリにどのような IP コミットがどの程度含まれているか目視による調査を行った †2．調査の結果，多数の IP コミットを発見し，それらの特徴を分析することで 3 つに分類した．次に，調査で得られた特徴をもとにした自動検出手法を提案した．提案手法の評価のために行った実験において，以下の結果を得た．

- 目視による調査を行ったコミットに対して提案手法を適用し，目視調査の結果と比較した実験では F 値が 0.7 であった．
- より多くのコミットに対して提案手法を適用し結果を目視によって確認した実験では適合率が 0.8 であった．

## 2　調査

本研究ではまず，実際にソフトウェア開発が行われているリポジトリにはどのような IP コミットが含まれているかを調べるために，目視による調査を行った．この調査では調査対象のコミットの変更内容を目視によって確認することで，各コミットに含まれるタスクを特定した．本研究ではタスクを，「ソフトウェアのある 1 つの機能に関する変更」と定義し調査を行った．特定したタスクを元に，同じバージョンのリリースに関わるコミットで，同じタスクを含む 2 つ

†2　本論文の内容は MSR2018 に採録されている

のコミットの組を IP コミットとした．本研究では以下の 2 つのリポジトリを調査対象とした．

**Apache Commons Collections:** 様々なデータ構造を提供するライブラリである．以降 Collections と呼ぶ．Version 1.0 リリースまでの 55 コミット 1,485 組を対象とした．

**Retrofit:** HTTP 通信を行うためのライブラリである †3．Version 0.5 リリースまでの 45 コミット 229 組を対象とした．

調査の結果，Collections リポジトリからは 49 組，Retrofit リポジトリからは 32 組の IP コミットを検出した．また，検出された IP コミットを調査によって得られた特徴を基に，Type-1，Type-2，Type-3 の 3 つのタイプに分類した．見つかった IP コミットのタイプ別の個数とその調査した組数に対する割合を表 1 に示す．以降，各タイプの IP コミットの特徴について述べる．ここで，$c_1$ と $c_2$ は IP コミットを構成するコミットを表し，$c_1$ は $c_2$ よりも古いコミットであるとする．

**Type-1 IP コミット:** Type-1 の IP コミットは，$c_2$ での変更内容が $c_1$ での変更漏れの追加や変更の取り消し，およびコメントの修正やソースコードの整形のようなソフトウェアの振る舞いに影響しない変更等の，$c_1$ での変更内容の修正であるものを指す．Type-1 の IP コミットの特徴は，それぞれのコミットでの変更箇所が同じメソッド内など，同一，もしくは非常に近いコードを変更していることである．図 1 で示した例は，Type-1 IP コミットである．このタイプの IP コミットをまとめて 1 つのコミットとすることでコミット履歴の可読性が向上すると著者らは考えた．

表 1 IP コミットの個数とその割合

| | Type-1 | Type-2 | Type-3 |
|---|---|---|---|
| Collections | 13 | 21 | 49 |
| | (0.9%) | (1.4%) | (3.3%) |
| Retrofit | 4 | 9 | 32 |
| | (1.7%) | (3.9%) | (14.0%) |

```
     * Insert an element into queue.
     *
     * @param element the element to be inserted
+    *
+    * @exception ClassCastException if the specified
  <code>element</code>'s
+    * type prevents it from being compared to other
  items in the queue to
+    * determine its relative priority.
     */
-    void insert( Comparable element );
+    void insert( Object element );
```

(a) 2002/03/19 13:34 のコミット

```
    /**
     * Insert an element into queue.
     *
     * @param element the element to be inserted
     */
-   public synchronized void insert( final Comparable
      element )
+   public synchronized void insert( final Object
      element )
    {
       m_priorityQueue.insert( element );
    }
```

(b) 2002/03/19 22:19 のコミット

図 2 Type-2 IP コミットの例

**Type-2 IP コミット:** Type-2 の IP コミットは，$c_2$ での変更が $c_1$ での変更に依存しているものを指す．それぞれのコミットで変更されたメソッド間に，呼び出し関係やオーバーライドの関係があることが特徴である．Collections での Type-2 IP コミットの例を図 2 に示す．(a) のコミットではインターフェース中のメソッドの引数が `Comparable` から `Object` に変更され，より広い型の値を受け取るように変更されている．このコミットの変更を受けて (b) のコミットではそのインターフェースを実装するクラスのメソッドの引数も `Comparable` から `Object` に変更されている．このタイプの IP コミットをまとめることで，ロジカルカップリングなどのリポジトリ分析性能の向上が期待できる．

**Type-3 IP コミット:** Type-1 および Type-2 IP コミットは $c_1$ での変更を受けて $c_2$ の変更が行われており，コミットの順序が重要となる．しかし，2 つのコミットの間に順序が重要となる依存関係はないものの，変更内容が互いに協調して 1 つの機能を実現しているコミットの組が多く存在した．このようなコミットの組を Type-3 IP

†3 https://github.com/square/retrofit

```
+    public void testListAdd() {
+      List list = makeList();
+      if(tryToAdd(list,"1")) {
+        assert(list.contains("1"));
+        if(tryToAdd(list,"2")) {
+          assert(list.contains("1"));
+          assert(list.contains("2"));
+          if(tryToAdd(list,"3")) {
```

(a) 2001/04/26 09:06 のコミット

```
+  public void testListSetByIndexBoundsChecking2() {
+    List list = makeList();
+    tryToAdd(list,"element");
+    tryToAdd(list,"element2");
+
+    try {
+      list.set(Integer.MIN_VALUE,"a");
+      fail("List.set should throw
   IndexOutOfBoundsException [Integer.MIN_VALUE]");
```

(b) 2001/05/05 01:34 のコミット

**図 3　Type-3 IP コミットの例**

コミットと呼ぶ．Collections での Type-3 IP コミットの例を図 3 に示す．(a) のコミットでは，`List` クラスに関するテストケースが，`TestList` クラスに追加されている．(b) のコミットでは，それとは別のテストケースが `TestList` クラスに追加されている．Type-3 の IP コミットを用いることで，1 つの機能に関連あるコミット群を得ることができるため，履歴の理解に役立つ．

## 3　IP コミット検出手法の提案

本章では，IP コミットの自動検出手法を提案する．提案手法の対象は Java プロジェクトのリポジトリであり，入力は 2 つのコミットの組，出力はそのコミットの組が IP コミットであるかどうかである．IP コミットではそれぞれのコミットでの変更箇所が近いことを利用するために提案手法では，(1) ソースコードからグラフを構築し，(2) グラフ上での変更箇所間の距離からスコアを計算することで入力のコミットの組が IP コミットであるかを判定する．以降，(1)，(2) の詳細について述べる．

### 3.1　グラフの構築

まず，入力として与えられた 2 つのコミットの組 $(x, y)$ について，それぞれのコミットが作成された時点でのソースコードを解析することによって以下の情報を取得する．

- ソースコードに含まれるメソッド
- メソッドが定義されているクラス
- メソッドの呼び出し関係

この結果を用いて，重み付き無向グラフを構築する．グラフの頂点は，各コミットのソースコードに含まれるメソッドとする．本研究ではコミット $c$ でのメソッド $m$ を表す頂点を $m^c$ とする．辺は以下に示す $E_{same}$，$E_{method}$，$E_{def}$ の 3 種類を用いる．

- $E_{same}$ は 2 つのコミットで同じメソッドを表す頂点間に張る辺である．辺の重みは 1 とする．
- $E_{method}$ は 2 つのメソッドに呼び出し関係があるときに張る辺である．辺の重みは $w_{method}$ とする．
- $E_{def}$ は 2 つのメソッドが同じクラスで定義されているときに張る辺である．辺の重みは $w_{def}$ とする．

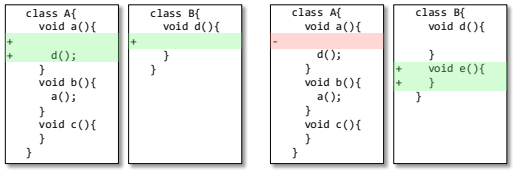
### 3.2　スコアの計算

次に構成したグラフ上でのメソッド間の距離をもとにスコアを計算する．

まず，コミット $x$ で変更された各メソッドについて，コミット $y$ で変更されたメソッドの中で一番近いものまでの距離を求める．提案手法で構築したグラフでは，コミット $x$ のメソッドから，コミット $y$ のメソッドへの経路には少なくとも 1 つの $E_{same}$ の辺が含まれるため，その距離は 1 以上となる．次に，各メソッドで求めた距離の逆数をとり，その平均を $s_x$ とする．$s_x$ は 0 以上 1 以下となる．

同様にコミット $y$ で変更された各メソッドについて，コミット $x$ で変更されたメソッドで一番近いものまでの距離を求め，求めた距離の逆数の平均を $s_y$ とする．

最後に $s_x$ と $s_y$ の平均と，$s_y$ のうち高いほうをスコアとする．これは調査において，古いほうのコミット $x$ で変更されたメソッドの一部を新しいコミット $y$ においても変更している IP コミットが多く見られたためである．このスコアが閾値（*threshold*）よりも高ければ IP コミットであると判定し出力する．




(a) コミット $x$ 時点のソースコード　(b) コミット $y$ 時点のソースコード

**図 4　スコア計算の例で用いるソースコード**

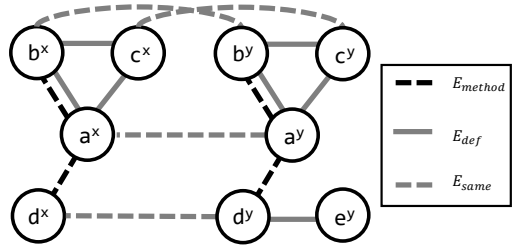



**図 5　ソースコードから生成されたグラフ**

### 3.3　スコア計算の例

ここでは提案手法を用いたスコア計算の例を，図 4 のソースコードを用いて示す．この例ではパラメータとして以下の値を用いる．

$$\begin{aligned} threshold &= 0.7 \\ (w_{method}, w_{def}) &= \left(\frac{3}{14}, \frac{3}{7}\right) \end{aligned}$$

これは $threshold = 0.7$ としたとき，以下の式を満たすように定めた．

$$\begin{aligned} threshold &= \left(2w_{method} + 1\right)^{-1} \\ &= \left(w_{def} + 1\right)^{-1} \end{aligned}$$

はじめに，入力として与えられたコミットの組 $(x, y)$ から，それぞれのコミットの時点でのソースコードを取得する．コミット $x$ の時点でのソースコードを図 4(a)，コミット $y$ の時点でのソースコードを図 4(b) に示す．ここで，+ で始まる行はそのコミットで追加された行，− で始まる行はそのコミットで削除された行を表す．得られたソースコードを解析し，解析結果を用いてグラフを作成する．作成したグラフを図 5 に示す．

次に，変更されたメソッド間のグラフ上での距離を求める．コミット $x$ で変更されたメソッドは $\{a^x, d^x\}$，コミット $y$ で変更されたメソッドは $\{a^y, e^y\}$ であり，それぞれ $V^x$，$V^y$ とする．

- $a^x$ から $V^y$ の中で一番近いメソッドは $a^y$ であり，その距離は 1 である
- $d^x$ から $V^y$ の中で一番近いメソッドは $a^y$ であり，その距離は $17/14 = 1.21$ である．
- $a^y$ から $V^x$ の中で一番近いメソッドは $a^x$ であり，その距離は 1 である．
- $e^y$ から $V^x$ の中で一番近いメソッドは $d^x$ であり，その距離は $10/7 = 1.43$ である．

以上より，$s_x = 0.91$，$s_y = 0.85$ である．よって $score = max((s_x + s_y)/2, s_y) = 0.88$ となり，このコミットの組のスコアは 0.88 である．よって，この 2 つのコミットの組は IP コミットであると提案手法は判定する．

## 4　評価実験

提案手法の評価のため，以下の 2 種類の実験を行った．

**実験 1**　2 章で行った 2 つのリポジトリに対する調査によって得られたデータセットに対して提案手法を適用した．評価指標として，提案手法が IP コミットであると判定したもののうち実際に IP コミットであったものの割合である適合率，IP コミットのうち提案手法が IP コミットであると判定したものの割合である再現率，およびこれら 2 つの調和平均である F 値を用いた．

**実験 2**　実験 1 よりも長い期間（331 コミット，18,619 組）に対して提案手法を適用した．提案手法によって IP コミットであると判定されたコミットの組が本当に IP コミットであるかを目視により判断し，その割合を適合率として求めた．

### 4.1　実験結果

実験結果を表 2 に示す．実験 1 において F 値は，Apache では 0.714，Retrofit では 0.745 であった．提案手法が誤判定したものには，プログラムのエントリーポイントのようなタスクの内容に関わらず多くのコミットで変更されるメソッドによって IP コミットでないものが IP コミットであると判定されたものや，複数のタスクを含む大規模なコミットによる影響

で IP コミットであるものが IP コミットでないと判定されたものがあった．

実験 2 において適合率は，Apache では 0.822，Retrofit では 0.884 であった．よって，提案手法はより大きなデータセットに対しても有効である．検出された IP コミットの例を以下に示す．

- Retrofit リポジトリの 2 つのコミット`#57a57d2`，`#1eab56a` が IP コミットとして検出された．この IP コミットでは，`#57a57d2` で変更したメソッドのソースコードを`#1eab56a` コミットで整形していた．これらのコミットを 1 つにまとめることで，コミット履歴を簡潔にすることができる．
- Collections リポジトリにある 2002/03/02 から 2002/03/20 にかけての 6 つのコミットの中から，13 組が IP コミットと判定された．各コミットの変更内容を確認すると，これらの 6 つのコミットはすべて `ComparatorChain` クラスに関する変更であった．このように Type-3 IP コミットは，ある機能に関するコミットの検索に有効である．

## 5 妥当性への脅威

本研究の妥当性への脅威として以下の項目が挙げられる．

**タスクの定義** 本研究ではタスクを「ソフトウェアのある 1 つの機能に関する変更」と定義し調査や自動検出手法の提案を行ったが、2 つの問題が考えられる．1 つは，どれだけの変更内容を 1 つのタスクと考えるかはソフトウェアや開発者によって異なる可能性があるという点である．もう 1 つは，リポジトリマイニング性能の向上に IP コミットを用いる場合に，タスクをどのように定義するかによって大きく性能が変わる可能性があるという点である．

**表 2 実験結果**

| | 実験 1 | | | 実験 2 |
|---|---|---|---|---|
| | 適合率 | 再現率 | F 値 | 適合率 |
| Collections | 0.714 | 0.714 | 0.714 | 0.822 |
| Retrofit | 1.000 | 0.594 | 0.745 | 0.884 |

**対象ソフトウェアおよび対象コミット** 精度の評価では，2 つのソフトウェアのリポジトリのみを扱っており，またそのコミットの中でも開発序盤のコミットのみが対象であった．他のソフトウェアのリポジトリや，開発後半のコミットでは本研究と異なる結果となる可能性がある．

**目視での確認** 調査や実験での正解データは目視により作成したため，分類者の主観による影響や誤りが含まれる可能性がある．

## 6 おわりに

本研究ではまず，2 つのオープンソースソフトウェアのリポジトリに対して目視による調査を行い，1,714 組のコミットの組の中から 81 組の IP コミットを発見した．また，発見した IP コミットを 3 つのタイプに分類した．次に，目視による調査から得られた IP コミットの特徴を利用した IP コミットの自動検出手法を提案した．提案手法の評価のために行った実験において，目視による調査を行ったコミットに対して提案手法を適用し調査結果と比較した実験では F 値 0.7，より多くのコミットに対して提案手法を適用し結果を目視によって確認した実験では適合率 0.8 の精度で IP コミットを検出することができた．

本研究の貢献は以下の通りである．

- 1 つのタスクが複数のコミットに分割されているコミットである IP コミットがリポジトリに含まれていることを示した．
- IP コミットの自動検出手法を提案した．

今後の課題は以下の通りである．

- より多くのリポジトリに対して実験を実行
- 提案手法を実装した Eclipse プラグインの開発

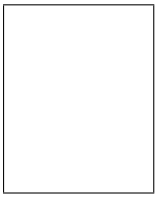
**謝辞** 本研究は，日本学術振興会科学研究費補助金基盤研究（B）（課題番号：17H01725）の助成を得て行われた．

## 参 考 文 献

[1] Berczuk, S. and Appleton, B.: *Software Configuration Management Patterns*, Addison-Wesley., 2002.

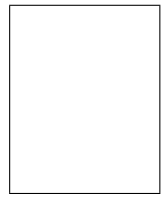
[2] Bieman, J. M., Andrews, A. A., and Yang, H. J.: Understanding change-proneness in OO software

through visualization, *Proceedings of 11th International Workshop on Program Comprehension*, Portland, Oregon, May 2003, pp. 44–53.
[3] Chacon, S. and Straub, B.: Git - Book.
[4] Herzig, K. and Zeller, A.: The Impact of Tangled Code Changes, *Proceedings of the 10th International Workshop on Mining Software Repositories*, May 2013, pp. 121–130.
[5] Kirinuki, H., Higo, Y., Hotta, K., and Kusumoto, S.: Hey! Are You Committing Tangled Changes?, *Proceedings of the 22nd International Conference on Program Comprehension*, June 2014, pp. 262–265.
[6] Murphy-Hill, E., Parnin, C., and Black, A. P.: How We Refactor, and How We Know It, *Proceedings of the 31st International Conference on Software Engineering*, 2009, pp. 287–297.
[7] Zimmermann, T., Weißgerber, P., Diehl, S., and Zeller, A.: Mining version histories to guide software changes, *IEEE Transactions on Software Engineering*, Vol. 31, No. 6(2005), pp. 429–445.
[8] 森達也, アンダースハグワード, 小林隆志: 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, 情報処理学会論文誌, Vol. 58, No. 4(2017), pp. 807–817.



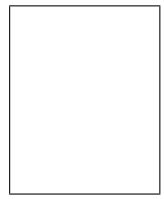
**有 馬 諒**

平成 29 年大阪大学基礎工学部情報科学科卒業．現在，大阪大学大学院情報科学研究科博士前期課程在学中．リポジトリマイニングに関する研究に従事．



**肥 後 芳 樹**

平成 14 年大阪大学基礎工学部情報科学科中退．平成 18 年同大学大学院博士後期課程修了．平成 19 年同大学院情報科学研究科コンピュータサイエンス専攻助教．平成 27 年同准教授．博士（情報科学）．ソースコード分析，特にコードクローン分析やリファクタリング支援に関する研究に従事．情報処理学会，電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．



**楠 本 真 二**

昭和 63 年大阪大学基礎工学部卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部助手．平成 8 年同講師．平成 11 年同助教授．平成 14 年同大学大学院情報科学研究科助教授．平成 17 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価に関する研究に従事．情報処理学会，電子情報通信学会，IEEE，IFPUG 各会員．