社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# 機械学習による開発履歴のメタ情報を用いたマージコンフリクトの解消パターン判定モデル

白木秀弥† 神田哲也† 井上克郎†

† 大阪大学
E-mail: †{s-siraki,t-kanda,inoue}@ist.osaka-u.ac.jp

**あらまし** 現在，ソフトウェア開発では，バージョン管理システム (以下，VCS) を用いた複数の開発者による並行開発が主流である．VCS を用いた開発では，複数の開発者が同一箇所の編集を行った場合，マージコンフリクトがしばしば発生する．マージコンフリクトを解消することは，開発者にとって負担となることが知られている．本研究では，ソフトウェアの開発履歴から，マージコンフリクトに関連するメタ情報，つまり，ソースコードの内容以外の情報から，マージコンフリクトの解消方法を判定するモデルを提案する．20 個のオープンソースソフトウェアを対象として，プロジェクトごとにテストを行ったところ，本研究で提案するモデルは，平均約 66%，最大約 94% の割合で適切なマージコンフリクトの解消方法を解答することができた．また，解消方法の判定には，マージコンフリクトの発生箇所の行数と，マージコンフリクトが発生したコミットからマージコンフリクトの発生箇所を編集した直近のコミット数などが大きく寄与していることも明らかになった．
**キーワード** 開発履歴，マージコンフリクト，メタ情報，機械学習

# Judgment Model of Merge Conflict Resolution Pattern Using Machine Learning Meta-Information

Shuya SHIRAKI†, Tetsuya KANDA †, and Katsurou INOUE †

† Osaka University
E-mail: †{s-siraki,t-kanda,inoue}@ist.osaka-u.ac.jp

**Abstract** Merge conflicts often occur in parallel development using version control system (VCS) . Resolving a conflict is cumbersome for developer, because they need to devise a resolution and make manual changes. In this paper, we proposed a method to build the machine learning model that suggests the solution using metadata in the development history. We conducted a case study with 20 OSS projects and the classification accuracy were 66% on average and 94% in the best case. Furthermore, it became clear that the number of conflicting lines and the number of commits between a conflicting commit and the commit with the causal edit have high contribution rate.
**Key words** Development History, Merge Conflict, Meta-Information, Machine Learning

## 1. ま え が き

現在，多くのソフトウェア開発では，複数の開発者によるチームが並行開発を行う開発方法が主流であり，その際にバージョン管理システム (以下，VCS) が頻繁に用いられる．VCS を用いた開発では，並行して開発を行う際，開発の本流となっているブランチから新しく派生ブランチを作成し，成果物が完成した後に本流のブランチに統合 (マージ) する．これによって，複数の開発者が同時に開発を進めることができ，効率的なソフトウェア開発が実現できる．

VCS の中で最も多く使われているものの一つが，Git である [1]．Git を用いて開発する際にしばしば生じる問題として，マージコンフリクトが挙げられる．新しく作成したブランチから派生元のブランチにマージを行うとき，派生ブランチで編集 (追記，削除) した箇所と，本流のブランチにおいて同一箇所に編集が行われていた場合，マージできないという状況が生じる．これがマージコンフリクトであり，Git を用いたソフトウェア開発においては，比較的頻繁に発生することが明らかになっている．Brun らの研究では，Git や Perl5 などを含む 9 個のオープンソースソフトウェア (以下，OSS) の開発履歴を対象に調査を行った結果，すべてのプロジェクトにおいてマージコンフリクトが発生しており，その割合は平均で約 19%，最大で約 42%

であることが明らかになった [2].

マージコンフリクトの問題として，解消に時間と手間がかかるという点が挙げられる．マージコンフリクトが発生した場合，開発者はその原因を調査し，その解消方法を考案し，手動で編集を行う必要がある．これらの作業は何日もの時間がかかることも少なくなく，開発者およびプロジェクト全体において大きなコストとなる [3].

いくつかの既存研究において，マージコンフリクトやその解消方法の特徴が明らかになっている．マージコンフリクトの発生箇所の行数には偏りがあり [4]，マージコンフリクトの発生箇所の行数は，マージコンフリクトを複雑化させる要因の一つであることが明らかになっている [1]．また，マージコミットの両親のコミットから共通の祖先までを調査し，コミット作成者と編集したファイルから，マージコンフリクトの解消に必要な知識を持つ開発者を推奨する研究もおこなわれている [5]．コミットの作成者に関する別の研究では，コミット作成者のプロジェクトに対するコミット率が小さいほど，バグの発生率やソフトウェア欠陥が発生する可能性が高くなることが明らかになっており [6]，マージコンフリクトとバグの間には関連があることも知られている [4]．マージコンフリクトの解消方法については，マージコンフリクトが発生したコミットペアに対して，どちらかの編集内容を採用し，もう片方の編集を削除することで，解消されることが多いことが明らかになっている [7]．本研究では，これらのマージコンフリクトの特徴となるメタ情報を活用することで，適切なマージコンフリクトの解消方法を判定するモデルを作成することを目指す．

本研究では，機械学習を用いて，マージコンフリクトの行数やコミットの作成日時，作成した開発者などのメタ情報から，解消方法の判定を行うモデルを作成することで，マージコンフリクトの解消支援を目指す．評価実験では，実際の開発履歴から判定の正答率を算出するとともに，モデル作成時にパラメータとして与えたメタ情報の重要度を計測することで，どのような情報がマージコンフリクトの解消方法に寄与するのかを調査する．

本論文では，第 2 章ではデータの収集とモデルの作成について手法の提案を行い，第 3 章では，作成したモデルの評価を行う．第 4 章では本研究の妥当性への脅威について議論し，第 5 章では今後の課題について述べる．最後に第 6 章でまとめを行う．

## 2. 提 案 手 法

機械学習を用いて，マージコンフリクトが発生したコミットペアに関連するメタ情報から，マージコンフリクトの解消方法を判定するモデルを，以下の手順で作成する．

STEP1　マージコンフリクトの履歴の収集
STEP2　メタ情報の収集
STEP3　解消方法の取得
STEP4　マージコンフリクトの解消方法判定モデルの作成

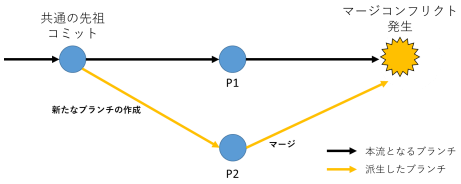


図 1　マージコンフリクトが発生したコミットペアの区別

モデルによる解消方法の判定は，コミットペアのそれぞれのコミットに対し行う．図 1 のように，マージコンフリクトが発生したコミットペアのそれぞれのコミットを区別する．本流となるブランチ上にあるコミットをコミット P1，新たに派生したブランチ上にあるコミットをコミット P2 とする．

### 2.1　マージコンフリクトの発生箇所の特定

マージコンフリクトが発生したコミットペアに対して，マージコンフリクトが発生したファイルパスと，マージコンフリクトが発生した行を特定する．マージコンフリクトが発生したコミット P1・P2 に対して，リポジトリから P1 にチェックアウトし，P2 にマージすることで，図 2 のような結果が得られる．

図 2 の場合，出力結果の 3 行目がマージコンフリクトが発生したファイルのコミット P1 におけるファイルパス，4 行目がマージコンフリクトが発生したファイルのコミット P2 におけるファイルパスとなっている．そして，6 行目に表示されている 3 個の 2 数の組が，マージコンフリクトが発生した行を示している．「a,b」という数字の組があった場合，a はマージコンフリクトが発生した先頭の行番号，b は発生した行数を表す．(ただし，b=1 のとき，b は省略される．) つまり，「10,3」とは，コミット P1 における 10 行目から 10+3(=13) 行目までにおいてマージコンフリクトが発生したことを示し，「11,2」とは，コミット P2 における 11 行目から 11+2(=13) 行目までにおいてマージコンフリクトが発生したことを示している．また，「15,6」とは，マージコンフリクトが発生した後のファイルが，15 行目から 15+6(=21) 行目までにマージコンフリクトが発生した状態で保存されていることを意味する．

### 2.2　メタ情報の収集

モデル作成に用いるパラメータとして，表 1 に示すメタ情報を用いる．収集するメタ情報は，マージコンフリクトの発生箇所の行数，コミット作成日時，コミット作成者のコミット率，

```
git checkout P1
git merge P2
git diff –U0
  1 diff --cc Test.java
  2 index ******
  3 --- a/ Main.java
  4 +++ b/Main.java
  5 @@@ -10,3 -11,2 +15,6 @@@ public class Main
  6 ++<<<<<<< HEAD
  7  +       int a, b;
  8  +       a = 1;
  9  +       b = 2;
 10 ++=======
 11 +        int a = 1;
 12 +        int b = 2;
 13 ++>>>>>>> c3d4c3d4c3d4
```

図 2　マージコンフリクトの発生箇所の例

表 1　モデルの作成に使用したパラメータ

| | パラメータ名 | |
|---|---|---|
| P1 | lines(P1) | P1 のマージコンフリクト発生箇所の行数 |
| | time(P1) | P1 のコミット作成日時 |
| | author(P1) | P1 のコミット作成者のコミット率 |
| | evolutionary(P1) | P1 と Evolutionary Commit 間の距離 |
| P2 | lines(P2) | P2 のマージコンフリクト発生箇所の行数 |
| | time(P2) | P2 のコミット作成日時 |
| | author(P2) | P2 のコミット作成者のコミット率 |
| | evolutionary(P2) | P2 と Evolutionary Commit 間の距離 |
| 差分 | lines(d) | lines(P2)-lines(P1) |
| | time(d) | time(P2)-time(P1) |
| | author(d) | author(P2)-author(P1) |
| | evolutionary(d) | evolutionary(P2)-evolutionary(P1) |

Evolutionary Commit までの距離の 4 種類であり，これを P1・P2 それぞれについて開発履歴から取得する．また，それらの差分についてもパラメータとして利用する．以降，それぞれのパラメータを開発履歴から取得する方法について述べる．

**2.2.1**　コミットのメタ情報

マージコンフリクトが発生したコミットペア P1・P2 について，それぞれのコミット作成日時とコミット作成者のプロジェクト全体に対するコミット率を取得する．

まず，コミット率の取得方法について説明する．プロジェクトのすべてのコミットについてコミットの作成者を調べる．コミット作成者それぞれについて，プロジェクトの総コミット数に対するコミット作成者のコミット数の割合を算出する．これがあるコミット作成者のコミット率となる．

次に，マージコンフリクトが発生したコミットペア P1・P2 について，それぞれのコミット作成日時とコミット作成者を抽出する．先ほど取得したコミット率の情報と照らし合わせて，各コミット作成者のコミット率を取得する．

**2.2.2**　マージコンフリクトの発生行数

マージコンフリクトの発生行数は，図 2 より，マージコンフリクトの発生箇所を特定した際の出力から取得できる．

**2.2.3**　Evolutionary Commit までの距離

マージコンフリクトが発生したコミット P1，P2 それぞれにおいて，マージコンフリクトの発生箇所を直前に編集したコミットを Evolutionary Commit と呼ぶ [8]．

モデル作成のパラメータとして，Evolutionary Commit とマージコンフリクトが発生したコミット間の距離を取得する．ここで，コミット間の距離とは，図 3 のように，一方のコミットから他方のコミットに至るまでに存在するコミット数を意味する．

開発履歴には，ある行をいつ編集したのかという情報がすべて記録されているため，P1・P2 から履歴を古いほうにたどることにより Evolutionary Commit を見つけ出し，そこまでのコミット数を計算することが可能である．

### 2.3　解消方法の収集

マージコンフリクトの解消方法として，マージコンフリクトが発生したコミットペアのコミットそれぞれに対して，表 2 の

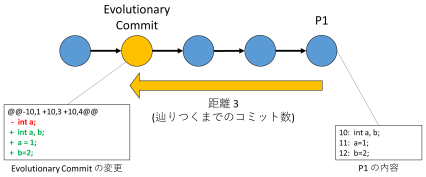


図 3　Evolutionary Commit の例

表 2　マージコンフリクト解消方法の定義

| | |
|---|---|
| ADOPT | 編集内容を全て採用する |
| DELETE | 編集内容を全て削除する |
| EDIT | 編集内容の一部を採用する，追加の編集を行うまたはその両方 |
| ZERO | 編集箇所が 0 行である |

4 種類を定義する．モデルによる出力は，解消方法の組，(P1 の解消方法，P2 の解消方法) となる．

マージコンフリクトが発生した状況と比較して，マージコミットに対してどのような編集が行われたかを調べる．マージコミット M において，その親コミットペア P1・ P2 の間でコンフリクトが発生している場合，P1・P2 のマージを行った結果と，M との差分を分析することで解消方法を特定することができる．ここで，図 4 で緑色で示されている行は，マージコンフリクトを解消する際に，削除された行である．実行結果の 5 行目から Test.java の 15 行目が削除されたことが，7 行目から Main.java の 19 行目から 23 行目までが削除されたことが分かる．この結果を図 2 の結果と比較すると，P1 の編集である 16 行目から 18 行目の内容が採用されていると判定できる．したがって，P1 の解消方法は ADOPT であり，P2 の解消方法は DELETE であると言える．

マージコンフリクトの解消方法が EDIT となる例として，図 5 のような場合がある．8 行目の「`- int c = a + b;`」は，マージコンフリクトの解消を行った際に，新たに追加したことを意味する．つまり，この場合には，P1 の解消方法は EDIT であり，P2 の解消方法は DELETE となる．今回の解消方法の判定モデルでは，どのような編集を行うかは特定しない．

マージコンフリクトの解消方法が ZERO となる場合として，マージコンフリクトの発生箇所が一度編集された後に，その編集が削除され，マージを試みた場合が挙げられる．図 6 のように，共通の祖先 A からブランチを作成し，コミット B において一度編集が行われたの後に，コミット P2 において編集が行

```
git checkout P1
git merge P2
git diff –U0 M
  1 diff --git Test.java
  2 index ******
  3 --- a/ Main.java
  4 +++ b/Main.java
  5 @@@ -14,0 -15 @@@ public class Main
  6 +<<<<<<< HEAD
  7 @@@ -18,0 -19,4 @@@ public class Main
  8 +=======
  9 +         int a = 1;
 10 +         int b = 2;
 11 +>>>>>>> a1b2a1b2a1b2
```

図 4　マージコンフリクトの解消方法の判定

```
git checkout P1
git merge P2
git diff –U0 M
  1 diff --git Test.java
  2 index ******
  3 --- a/Main.java
  4 +++ b/Main.java
  5 @@@ -14,0 -15 @@@ public class Main
  6 +<<<<<<< HEAD
  7 @@@ -18,0 -19,4 @@@ public class Main
  8 -         int c = a + b;
  9 +=======
 10 +         int a = 1;
 11 +         int b = 2;
 12 +>>>>>>> a1b2a1b2a1b2
```

図 5　マージコンフリクトの解消方法が EDIT になる場合

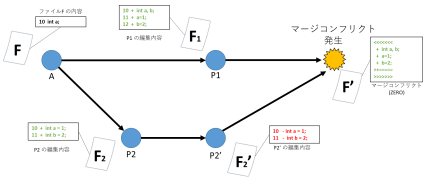


図 6　マージコンフリクトの解消方法が ZERO になる場合

われた箇所が削除されたとする．このとき，Git 上には同一箇所を編集したという事実が存在するが，ファイル上では変更が見られないという状況が発生し，編集箇所が 0 行のマージコンフリクトが発生する．マージコンフリクトの発生箇所が 0 行の場合，解消後のマージコミット M から，ADOPT，DELETE，EDIT のいずれかを判断することができないため，マージコンフリクトの解消方法の一つに ZERO を定義した．

### 2.4　マージコンフリクトの解消方法判定モデルの作成

マージコンフリクトの解消方法の判定にあたって，開発履歴から抽出したマージコンフリクトのメタ情報を入力とし，マージコンフリクトが発生したコミットのペアに対して，どのような解消方法が適切か，解消方法の組み合わせを出力するモデルを作成するアルゴリズムはランダムフォレストを用いる．

## 3.　提案手法の評価

提案手法の評価を行うために，実際に公開されている OSS の開発履歴を用いて，以下の点から評価を行う．

RQ1　：モデルの判定は，どれくらいの精度か？
RQ2　：判定に寄与しているメタ情報は何か？

評価は以下の手順で行う．

STEP1　：プロジェクトの開発履歴からマージコンフリクトを収集する．
STEP2　：収集したマージコンフリクトを，教師データとテストデータに分割する．
STEP3　：教師データから，メタ情報と解消方法を抽出し，モデルを作成する．
STEP4　：テストデータから，メタ情報を抽出し，モデルに入力する．
STEP5　：モデルの判定結果と，実際の解消方法が一致するかを調べる．

表 3　コミットとマージコンフリクトの件数

| プロジェクト名 | マージコンフリクト |
|---|---|
| beam | 981 |
| camel | 37 |
| cassandra | 13,132 |
| cordova-android | 700 |
| curator | 255 |
| dubbo | 426 |
| flink | 2,152 |
| geode | 427 |
| groovy | 294 |
| hbase | 108 |
| hive | 4,809 |
| ignite | 1,703 |
| incubator-heron | 56 |
| jmeter | 394 |
| lucene-solr | 1,606 |
| mahout | 105 |
| maven | 248 |
| nifi | 591 |
| nutch | 442 |
| rocketmq | 46 |

テストの対象として，Apache が提供する OSS の中から，20 個の Java プロジェクトを収集した．テストに使用するプロジェクトから抽出したマージコンフリクトの件数は，表 3 の通りであった．

### 3.1　モデルの正答率

モデルによる解消方法の判定の正答率を算出するにあたって，収集したマージコンフリクトを 5 分割し，一つをテストデータ，残りを教師データとする 5 分割交差検証を行い，その正答率の平均を求めた．正答率とは，テストデータの内，主流となるブランチ上のコミット P1 と，新たに作成された派生ブランチ上のコミット P2，両方においてモデルが判定した解消方法が一致していた割合を意味する．テストを行った結果の正答率を，表 4 に示す．

テストの結果，最大 94.43% という高い正答率が得られており，正答率が 80% 以上のプロジェクトが 20 個中 6 個存在した．この結果から，本研究の手法によって作成されたモデルは，一部のプロジェクトにおいては，有用なマージコンフリクトの判定モデルとなっていることが分かる．また，20 個のプロジェクトにおける正答率の平均は 66.41% であったが，出力となるマージコンフリクトの解消方法の組み合わせが 16 種類であることを考慮すると，十分に解消方法を判定していると考えられる．

その一方で，最も低い正答率を持つプロジェクトは，incubator-heron であり，その正答率は 23.83% であった．正答率の低下の原因として，表 3 より，incubator-heron のマージコンフリクトの発生数は 56 件であることから，モデルの学習不足が考えられる．

判定精度が非常に良かったプロジェクトについて，さらに調査を行う．判定精度が 90% を超えていたプロジェクトの一つに beam がある．beam のテストデータにおいて，マージコンフ

表 4　モデルの判定によるペアごとの正答率

| | 正答率 |
|---|---|
| beam | 90.92% |
| camel | 43.06% |
| cassandra | 48.49% |
| cordova-android | 50.70% |
| curator | 66.44% |
| dubbo | 74.01% |
| flink | 65.66% |
| geode | 57.48% |
| groovy | 65.57% |
| hbase | 44.23% |
| hive | 63.58% |
| ignite | 67.88% |
| incubator-heron | 23.83% |
| jmeter | 94.43% |
| lucene-solr | 63.42% |
| mahout | 82.15% |
| maven | 80.03% |
| nifi | 92.94% |
| nutch | 70.20% |
| rocketmq | 83.29% |

表 5　beam における解消方法 (DELETE, ADOPT) に対する判定

| | | 判定結果 | |
|---|---|---|---|
| | | TRUE | FALSE |
| 実際の解消方法 | TRUE | 112 | 5 |
| | FALSE | 1 | X |

リクトのおよそ 60% は，(DELETE, ADOPT) の片側採用によって解消されていた．解消方法に偏りがある場合，機械学習によって作成したモデルが，偏った解消方法ばかりを解答する恐れがある．実際の解消方法が (DELETE, ADOPT) であったマージコンフリクトと，判定モデルが (DELETE, ADOPT) であると判断したマージコンフリクトについて調査すると，表 5 のようになった．この表から，図 5 のテストにおいて，解消方法 (DELETE, ADOPT) に対する Recall=0.99，Precision=0.96，F1 値=0.97 となり，このモデルはプロジェクト内の解消方法の偏りに対して的確な判定を行っている．

以上の結果より，RQ1 に対する答えは以下の通りとなる．

**RQ1. メタ情報によるマージコンフリクトの解消方法の判定は，どれくらいの精度か？**

20 個のプロジェクトに対してテストを行ったところ，平均で 66.41% であった．90% 以上の正答率という精度が得られる例もあり，そのようなプロジェクトはマージコンフリクトの解消方法に偏りがある傾向にあるが，モデルはその偏りに過剰に反応することなく判定可能である．

マージコンフリクトの発生件数が少ないプロジェクトに関しては，十分に学習できず，精度が低下する恐れがある．

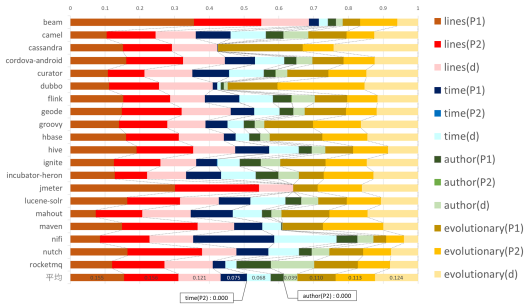


図 7　パラメータの重要度

### 3.2　パラメータの重要度

作成したモデルがどのパラメータによって解消方法を分類しているか調べるために，重要度という指標を用いる．重要度とは，作成されたランダムフォレストのモデルにおいて各パラメータが分類に対して寄与している割合であり，すべてのパラメータの重要度の総和は 1 となる．

プロジェクトごとに，収集したマージコンフリクトの 8 割を教師データとして与えてモデルを作成し，各パラメータの重要度を調べたところ，図 7 の通りとなった．ここで，図 7 の平均は，各パラメータに対して 20 個のプロジェクトの平均を取ったものである．

図 7 より，マージコンフリクトの発生箇所の行数 `lines(P1)` と `lines(P2)` が，共に 0.15 以上と大きくなっており，重要度が極端に小さくなるコミットは存在しない．また，`lines(d)` に関しても，重要度が 0.1 を超えるプロジェクトが多い．そのため，マージコンフリクトの発生箇所の行数は，いずれのコミットにおいても，マージコンフリクトの解消方法の判定に大きく寄与しているメタ情報であると言える．また，コンフリクトが発生したコミットから Evolutionary Commit までの距離，`evolutionary(P1)` と `evolutionary(P2)`，そしてその差分 `evolutionary(d)` の重要度の平均値も，0.11 以上と大きい値を取っている．

それに対して，コミット作成日時やコミット作成者のコミット率は，判定に大きく寄与していると言えない値となった．特に，表 7 から，`time(P2)` や `author_ratio(P2)` といった，主流となるブランチから新たに作成したブランチ上のコミットのメタ情報は，ほぼマージコンフリクトの解消方法の判定に寄与していないことが分かる．しかし，コミット作成日時については，主流となるブランチ上のコミット A の情報 `time(P1)` は，0.1 以上となっているプロジェクトもいくつか存在するため，マージコンフリクトの判定に寄与していると言える．

**RQ2：どのようなパラメータが，判定に影響を与えているか．**

モデルの重要度から，マージコンフリクトの発生行数 (`lines(P1)`, `lines(P2)`, `lines(d)`) と，マージコンフリクトが発生したコミットから Evolutionary Commit までの距離 (`evolutionary(P1)`, `evolutionary(P2)`, `evolutionary(d)`) が，マージコンフリクトの解消方法に大きく寄与している．ま

た，主流となるブランチ上に存在するコミットのコミット作成日時 (`time(P1)`) も大きくはないが判定に寄与している．

それに対して，主流から新しく作成されたブランチ上のコミットのコミット作成日時 (`time(P2)`) や，コミット作成者のコミット率 (`author(P2)`) は，あまり判定に寄与していないと言える．

## 4. 妥当性への脅威

### 4.1 パラメータの選定に関する問題

本研究では，モデル作成のパラメータに用いるメタ情報として，コミットの作成日時やコミット作成者のコミット率，マージコンフリクトの発生行数，Evolutionary Commit までの距離を用いたが，開発履歴には今回活用していないメタ情報も潜んでいる．そのため，本研究で作成したモデルが活用したメタ情報の組み合わせが最適であるとは限らない．しかし，本研究で得られた正答率より，本研究のモデルは，マージコンフリクトの解消に活用できる可能性がある考える．

### 4.2 実験対象に関する問題

本研究で用いたプロジェクトは，すべて Apache の提供する OSS であり，他の OSS や商用のプロジェクトにおいて，本研究の結果と同様の結果になる確証は無い．しかし，研究の対象としたプロジェクトは，開発規模やドメインが極端に偏っているものではないため，広いプロジェクトに対して調査を行うことができたと考える．

## 5. 今後の課題

本研究のモデル作成に用いたメタ情報以外に，Git 上に残された情報として，コミットメッセージやタグがある．コミットに付与されたコミットメッセージやタグは，そのコミットでの編集内容を表す大きな情報である．例えば，マージコンフリクトが発生したコミットやその Evolutionary Commit のコミットメッセージに，バグ修正や機能の追加に関連する単語が含まれているかによって，マージコンフリクトのが発生したときの解消方法に影響があるかもしれない．したがって，今後の課題として，コミットメッセージやタグを調査し，モデルによるマージコンフリクトの解消方法の判定に用いるパラメータを増やすという点が挙げられる．

また，プロジェクトの開発履歴においてマージコンフリクトの発生件数が少ない場合，適切な判定ができないという点は，実際の開発に本研究のモデルを活用するために解決すべき課題である．開発履歴においてマージコンフリクトの発生件数が多いということは，プロジェクトが発足してからある程度の期間が経過しているということに他ならない．つまり，このモデルによる判定は，複数の開発者による編集が最も盛んであり，マージコンフリクトの発生が懸念されるプロジェクトのスタートアップ時には適用できない．この問題の解決策として，複数のプロジェクトの開発履歴から収集したマージコンフリクトのメタ情報からモデルを作成することで，適用するプロジェクトの開発履歴のみに依存しない汎用性のあるモデルを目指す．

## 6. ま と め

開発履歴のメタ情報を用いて解消方法の判定を行うモデルを作成することで，ソフトウェア開発におけるマージコンフリクトの解消を支援する手法を提案した．実際に公開されている OSS を用いたテストの結果，本研究の手法で作成した判定モデルの正答率が，平均約 65%，最大 94% という，高い精度で解消方法を判定できることが明らかになった．しかし，機械学習を用いる欠点として，開発履歴から取得できるマージコンフリクトの件数が少ないプロジェクトにおいては，モデルの学習不足により，判定の精度が低下すると予想される．この結果より，マージコンフリクトが多く発生するプロジェクトにおいては，開発者がマージコンフリクトに直面した際に，その解消方法を判断する指針として役立つと考えられる．

## 謝　　辞

本研究は JSPS 科研費 JP18H04094，JP19K20239 の助成を受けた．

## 文　　献

[1] Shane McKee, Nicholas Nelson, Anita Sarma, and Danny Dig. Software practitioner perspectives on merge conflicts and resolutions. In *International Conference on Software Maintenance and Evolution (ICSME)*, pp. 467–478, 2017.

[2] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Early detection of collaboration conflicts and risks. *IEEE Transactions on Software Engineering*, Vol. 39, No. 10, pp. 1358–1374, 2013.

[3] Bakhtiar Khan Kasi and Anita Sarma. Cassandra: Proactive conflict minimization through optimized task scheduling. In *International Conference on Software Engineering(ICSE)*, pp. 732–741, 2017.

[4] Iftekhar Ahmed, Caius Brindescu, Umme Ayda Mannan, Carlos Jensen, and Anita Sarma. An empirical examination of the relationship between code smells and merge conflicts. In *International Symposium on Empirical Software Engineering and Measurement(ESEM)*, pp. 58–67, 2017.

[5] Catarina Costa, Jair Figueiredo, Leonardo Murta, and Anita Sarma. Tipmerge: Recommending experts for integrating changes across branches. In *Fast Software Encryption(FSE)*, pp. 523–534, 2016.

[6] Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commitbugginess? In *Mining Software Repositories (MSR)*, pp. 153–162, 2011.

[7] 湯月亮平. 開発履歴を用いたメソッドコンフリクトの分析. 修士論文, 奈良先端科学技術大学院大学, 2015.

[8] Mehran Mahmoudi, Sarah Nadi, and Nikolaos Tsantalis. Are refactorings to blame? an empirical study of refactorings in merge conflicts. In *International Conference on Software Analysis, Evolution and Reengineering(SANER)*, pp. 151–162, 2019.