修士学位論文

題目

開発履歴のメタ情報を用いた
マージコンフリクト解消支援手法

指導教員
井上 克郎 教授

報告者
白木 秀弥

令和２年２月５日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和元年度 修士学位論文

題目

白木 秀弥

内容梗概

現在，ソフトウェア開発では，バージョン管理システム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，しばしばマージコンフリクトが発生する．マージコンフリクトを解消することは，開発者にとって大きな負担となる．

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判定することができた．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が明らかになった．

主な用語

開発履歴
マージコンフリクト
機械学習

目 次

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 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) した箇所に対して，本流のブランチにおいて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，VCS 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，その割合はマージの平均約 19%，最大約 42%であることが明らかになっている [1].

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 [2].

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 [3]，マージコンフリクトの発生箇所の行数が多いほど，マージコンフリクトの発生率が高くなることが明らかになっている [4]．また，コミットの作成者に関する研究では，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 [5]，他の研究では，バグがマージコンフリクトの発生に繋がることも明らかになっている [3]．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，VCS の中でも最も多く使用されているものの一つである Git [6] を対象として，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，マージコンフリクトの解消方法の判定を行うモデルを，機械学習を用いて作成することで，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作成したモデルの判定の正答率を調べるとともに，モデルのパラメータの重要度を計測することで，どの

ようなメタ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する．最後に第 6 章でまとめを行う．

2　背景

2.1　マージコンフリクト

Git では，2 つのコミットをマージする場合，それらのコミットと共通の祖先の差分を用いてマージを行う，3-way マージが一般的である．3-way マージの利点として，行の追加だけでなく，共通の祖先に存在する行に対して，変更や削除などの編集を行った場合も，その編集を検出し，適切なマージを行うことが挙げられる [7]．しかし，両方のコミットの差分において同一箇所の編集が検出された場合，マージコンフリクトが発生する．

図 1 に，マージコンフリクトの例を示す．コミット A においてブランチが作成され，ファイル F の 10 行目に対して編集を行ったコミット P1 が作成される．次に，コミット A から別のブランチが作成され，ファイル F の 10 行目に対して別の編集を行ったコミット P2 が作成される．その後，コミット P1 と P2 は，それぞれの編集内容をマージしようと試みる．このとき，編集箇所に重複があるため，マージコンフリクトが発生する。

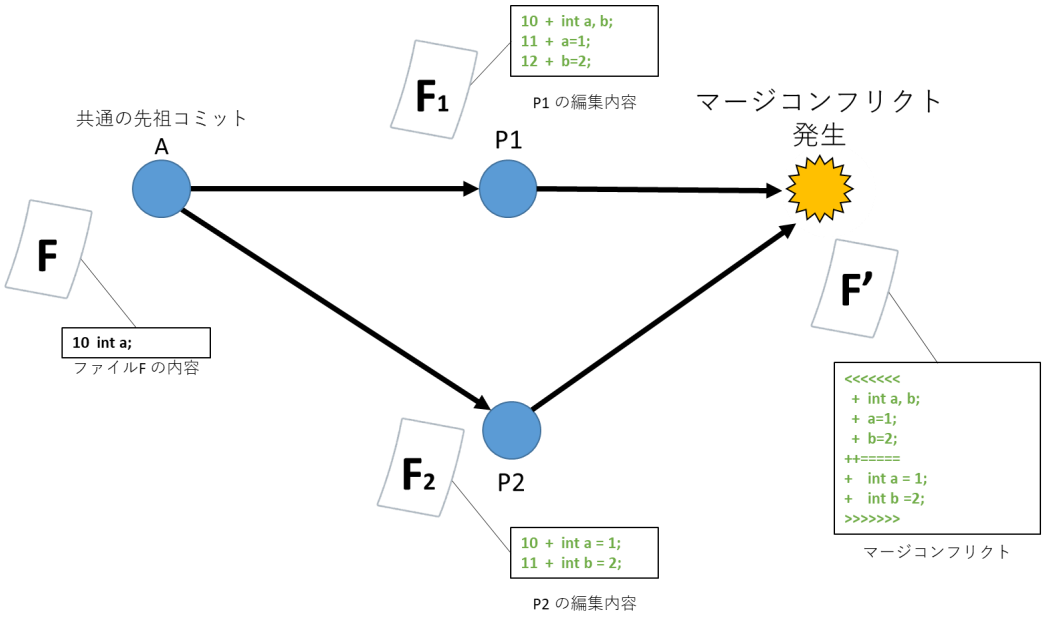

図 1: マージコンフリクトの発生状況

Mahmood らの研究によると，編集を行った作業の内容が，リファクタリングまたは機能の追加である場合にマージコンフリクトが発生しやすいという結果がある [8]．リファクタリングや機能の追加は，ソフトウェア開発において，日常的に行われている作業であるため，

マージコンフリクトという問題に，多くの開発者が直面している。実際に，開発者たちに調査を行ったところ，回答者の 54%が，マージを行う上で最も重要な問題はマージコンフリクトであると回答している [9].

2.2 マージコンフリクトの種類

2.1 節で述べたマージコンフリクトは，狭義にテキストのコンフリクトと呼ばれる．これは，2 つのコミット間におけるファイルのテキスト上の衝突によって生じるもので，Git による 3-way マージによって検出できる．

テキスト上のコンフリクトの他に，ビルドのコンフリクトやテストのコンフリクトと呼ばれるマージコンフリクトが存在する [1,10]．ビルドのコンフリクトやテストのコンフリクトとは，テキスト上では正常にマージされたように見えるが，実際にプログラムのビルドやテストを実行した際に，マージを原因とするエラーが発生することである．例えば，ビルドのコンフリクトの場合，片方のコミットであるメソッドの呼び出しが追加され，もう片方のコミットで呼び出されるはずのメソッドが削除されていた場合，マージ後に正常にビルドが行われないことは明らかである．このように，一見マージが正常に行われたかのように見えて，実際にプログラムを実行しようとした際に発覚するマージコンフリクトを，高次のマージコンフリクトと呼ぶ [1,10,11]．高次のコンフリクトは，Git による 3-way マージでは検出されない．高次のコンフリクトに関する研究として，並行開発中に，それぞれのブランチにおける各メソッドを監視することで，その振る舞いを比較し，高次のコンフリクトを検出する手法がある [12].

高次のコンフリクトは，テキストのコンフリクトに比べて，発生率が半分以下であることが明らかになっている [10]．そのため，本研究では，高次のコンフリクトは扱わず，以後，マージコンフリクトとは，テキストのコンフリクトを指すものとする．

2.3 マージコンフリクトの探索

マージコンフリクトが発生した場合には，新たにコミットは作成されず，マージコンフリクトを解消した後に改めてコミットすることでマージコミットが作成される．そのため，開発過程で発生したマージコンフリクトの情報は Git に残されていない．開発履歴からマージコンフリクトの調査を行う場合，2 つの親コミット P1, P2 を持つマージコミットにおいて，マージコンフリクトが発生したかを調べる．以下のコマンドを実行し，コミット P1,P2 間にマージコンフリクトが発生していた場合，図 2 の出力結果の 1 行目のように CONFLICT と出力される。

```
git checkout P1
git merge P2
```

```
git checkout P1
git merge P2
  1 CONFLICT (content): Merge conflict in Test/Main.java
  2 Automatic merge failed; fix conflicts and then commit the result.
```

図 2: マージの結果コンフリクトが発生した場合

2.4 マージコンフリクトの解消方法

いくつかの既存研究において，マージコンフリクトの解消方法に関する調査が行われている．開発においてマージコンフリクトが発生した場合，開発者は経験則に基づいてマージコンフリクトの解消方法を決定し，その解消を行っていることが明らかになっている。[6] また，具体的なマージコンフリクトの解消方法に関して，3 のように，2 つの親コミットのどちらか片方の編集を採用し，もう片方の親の編集を削除することで，マージコンフリクトの解消が行われるケースが多く存在していることが知られている。これを片側採用といい，湯月らの研究によって，Java プロジェクト中のメソッド内部で発生したマージコンフリクトにおいては，全体の約 98% が片側採用で解消されていることが明らかになっている [13].

2.5 マージコンフリクトの軽減

マージコンフリクトを軽減するための手法として，様々な研究が行われており，マージ時に編集内容に対して操作を加えることで，マージコンフリクトの回避を試みる傾向にある [14].

例えば，構文木を用いた構造的マージによってマージを行う [15–17] という手法がある．構造的なマージとは，ソースコードを構文木に変換し，構文木の形によってマージを行うというものである．つまり，テキスト上ではマージコンフリクトが発生していたとしても，プログラムの挙動に衝突が無ければ，適切な形の構文木となるようにマージを行う．構文木を用いた他の手法として，4-way 差分アルゴリズムという手法を用いたマージも提案されている [18]．4-way 差分とは，3-way マージの際に用いる差分に加えて，マージを行うコミットペアにおいて，共通の祖先から変更が行われなかった箇所を差分として用いる．

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

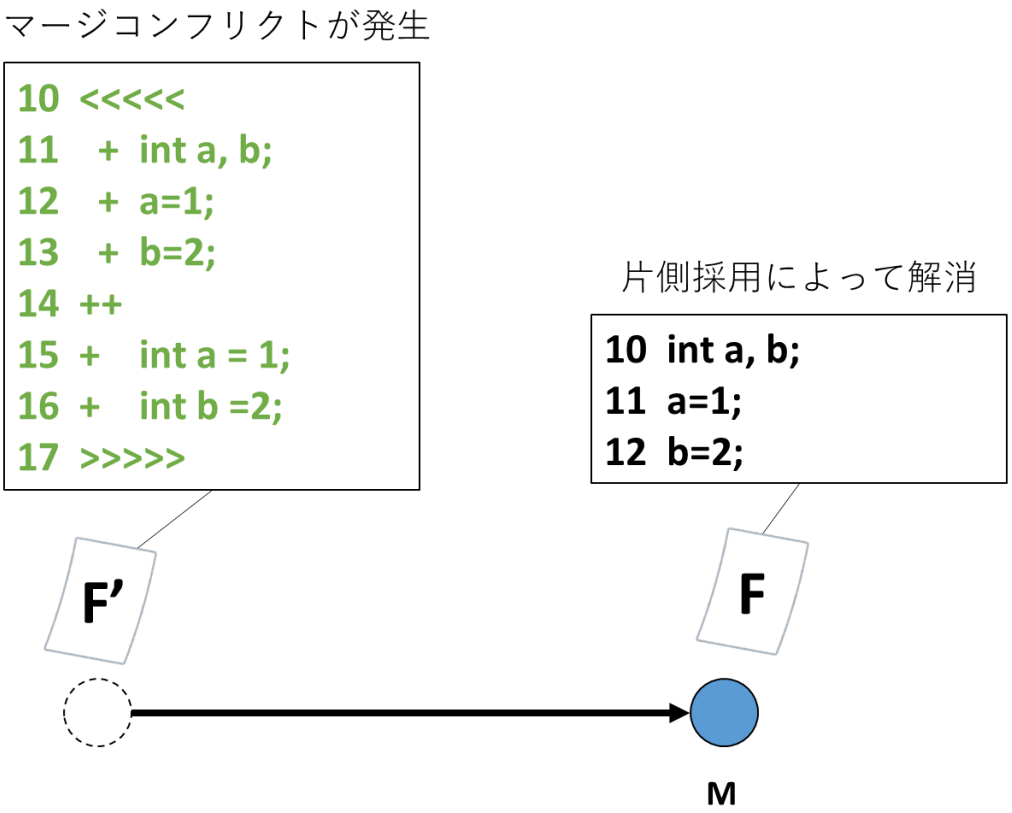

図 3: 片側採用によるマージコンフリクトの解消

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手法も存在し，適切な開発者上位 3 名を提示できた割合が 98%であったという結果がある [19].

2.6 開発履歴のメタ情報

開発履歴のメタ情報とは，開発者のコミット履歴や，プロジェクトのコミット数，コミットの作成日時などの情報を指す。さらに，本研究では，マージコンフリクトの発生したファイル名やマージコンフリクトの発生行数なども，メタ情報として扱う。コミットの作成日時や，コミットの作成者のプロジェクト全体のコミット率は，バグの発生率と関連があるという結果 [5] や，マージコンフリクトとその解消の複雑さの関連 [4] が明らかになっている．したがって，これらのメタ情報が，マージコンフリクトの解消に役立つ可能性は十分に考えられる。

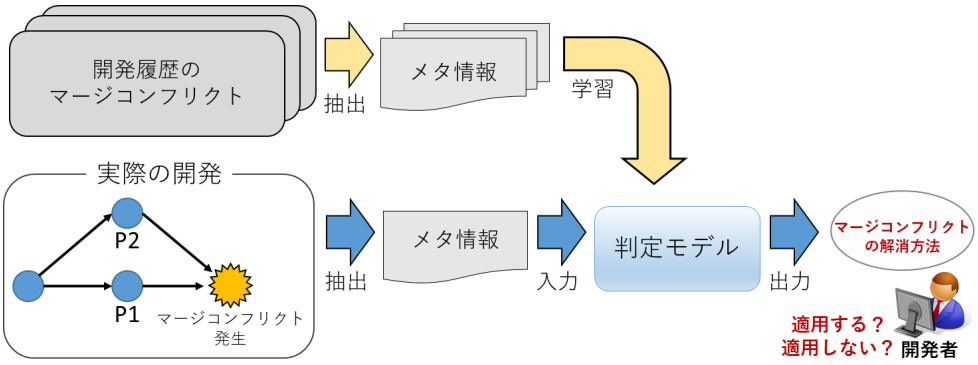

図 4: マージコンフリクトの解消方法の判定モデル

3 提案手法

ソフトウェア開発においてマージコンフリクトが発生した際．開発者に対してマージコンフリクトの解消をどのように行うか提案することを目指す．マージコンフリクトの解消方法の提案のために，機械学習を用いて，過去に発生したマージコンフリクト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作成する．モデルの概要を，図 4 に示す．マージコンフリクトの解消方法を判定するモデルは，以下の手順で作成する．

STEP1　マージコンフリクトのメタ情報の収集と分析
STEP2　コミットのメタ情報の収集
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

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 5 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 P1，新たに派生したブランチ上にあるコミットを P2 とする．

ここで，マージコンフリクトの発生箇所に対して，表 1 の情報を，本研究ではマージコンフリクトのメタ情報と呼ぶ．これらのメタ情報を活用して得られる情報を用いて，マージコンフリクトの解消方法の判定モデルの作成を行う．また，マージコンフリクトが発生したコミット P1・P2 それぞれのメタ情報についても，マージコンフリクトの解消方法の判定モデルの作成に使用する．

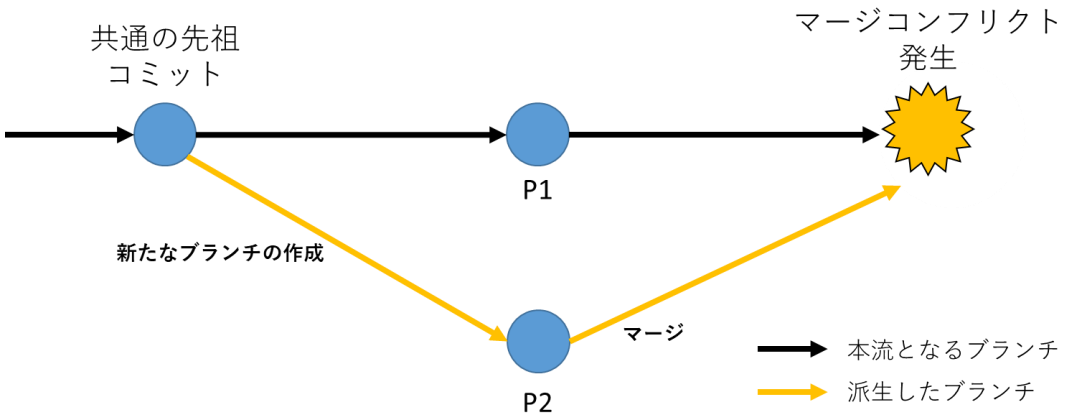

図 5: マージコンフリクトが発生したコミットペアの区別

表 1: マージコンフリクトのメタ情報

P1 のハッシュ値
P1 におけるマージコンフリクトが発生したファイル名
P1 におけるマージコンフリクトが発生した開始行
P1 におけるマージコンフリクトが発生した行数
P2 のハッシュ値
P2 におけるマージコンフリクトが発生したファイル名
P2 におけるマージコンフリクトが発生した開始行
P2 におけるマージコンフリクトが発生した行数

3.1 マージコンフリクトのメタ情報の収集と分析

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し，マージコンフリクトのメタ情報を得る．マージコンフリクトが発生したコミットペア P1・P2 に対して，以下のコマンドを実行することで，図 6 のような結果が得られる．

```
git checkout P1
git merge P2
git diff -U0
```

図 6 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

```
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 6: マージコンフリクトの発生箇所の例

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．

3.1.1 Evolutionary Commit と距離

マージコンフリクトが発生したコミット P1, P2 のそれぞれにおいて，共通の祖先から P1, P2 に至るブランチに，マージコンフリクトが発生した箇所を編集したコミットが存在する．マージコンフリクトの発生箇所を編集した中で，最も P1, P2 に近いコミット達を Evolutionary Commit と呼ぶ．[20]

マージコンフリクトのメタ情報を用いて，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し，マージコンフリクトの解消方法判定モデルの作成に使用する．ここで，コミット間の距離とは，図 7 のように，一方のコミットから遡って，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．

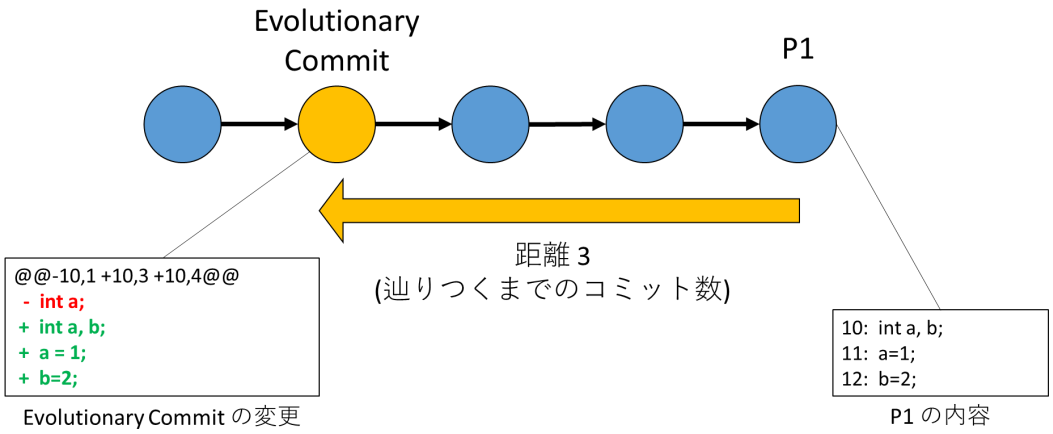

図 7: Evolutionary Commit の例

P1 の Evolutionary Commit を特定するためには，P1 のマージコンフリクトの発生箇所が $file_{P1}$ の $start_{P1}$ 行目から end_{P1} 行目までであるとすると，以下のコマンドを実行する．

```
git log -L start_P1,end_P1:file_P1 P2..P1
```

同様に，P2 の Evolutionary Commit は，以下のコマンドで特定できる

```
git log -L start_P2,end_P2:file_P2 P1..P2
```

P1 の Evolutionary Commit を E とすると，P1 から遡り，E にたどり着くまでのコミット一覧は，以下のコマンドで出力できる．

```
git log --graph --oneline E..P1
```

3.2 コミットのメタ情報の収集

3.2.1 コミットの作成日時

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，すべての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の取得方法について説明する．以下のコマンドを実行し，プロジェクトの最新コミット以前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．

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	編集内容を全て消去する
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

```
git checkout master
git log --date=iso --pretty=format:%an
```

この結果より，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．

```
git checkout P1
git log --date=iso --pretty=format:%ad -1 git log --pretty=format:%an -1
```

得られたコミット作成者を，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．

3.3 マージコンフリクトの解消方法

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 4 種類を定義する．マージコンフリクトの解消方法判定モデルによって開発者に提案する解消方法は，マージコンフリクトが発生したコミット P1・P2 に対する解消方法の組 (P1 の解消方法，P2 の解消方法) となる．

マージコンフリクトの解消方法を特定するためには，以下のコマンドを実行し，その結果を分析する必要がある．

```
git checkout P1
git merge P2
git diff -U0 M
```

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．以下のコマンドを実行すると，図 8 のような結果が得られる．こ

```
git checkout P1
git merge P2
git diff –U0
  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 8: マージコンフリクトの解消方法の判定

こで，図 8 で緑色で示されている行は，マージコンフリクトの解消を行った際に，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**??**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 9 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．今回のマージコンフリクトの解消方法の判定モデルでは，EDIT においてどのような編集を行うかについては関与しない．

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 10 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．

```
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 9: マージコンフリクトの解消方法が EDIT になる場合

3.4 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，図 4 のようなモデルを作成する．開発履歴から収集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である．

マージコンフリクトの解消方法判定モデルの作成に使用するパラメータは，P1, P2 それぞれのメタ情報と，それらメタ情報の P1 と P2 の差分 (P2 のパラメータ - P1 のパラメータ) である．表 3 に，パラメータの一覧を示す．

モデル作成のためのアルゴリズムは，ランダムフォレストを用いる．

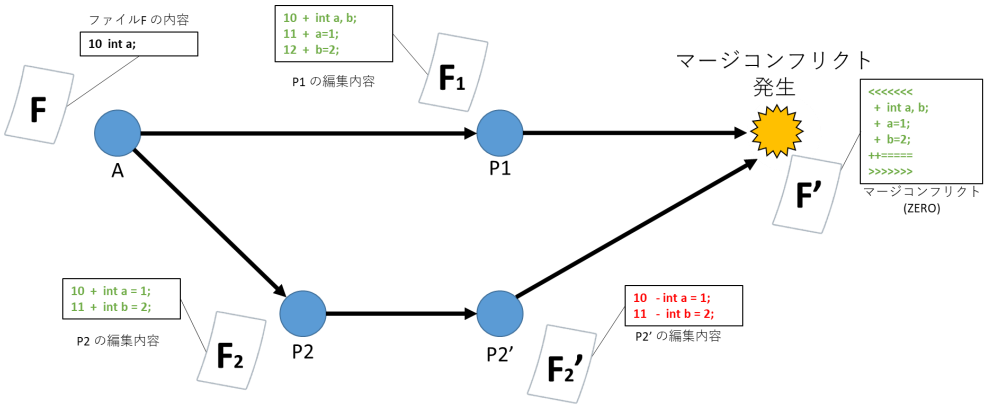

図 10: マージコンフリクトの解消方法が ZERO になる場合

表 3: モデルの作成に使用したパラメータ

	パラメータ名	
P1	line_num(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_ratio(P1)	P1 のコミット作成者のコミット率
	distance(P1)	P1 とその Evolutionary Commit 間の距離
P2	line_num(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_ratio(P2)	P2 のコミット作成者のコミット率
	distance(P2)	P2 とその Evolutionary Commit 間の距離
差分	line_num(d)	line_num(P2) - line_num(P1)
	time(d)	time(P2) - time(P1)
	author_ratio(d)	author_ratio(P2) - author_ratio(P1)
	distance(d)	distance(P2) - distance(P1)

4　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：　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？

RQ2：　どのようなメタ情報が，判定に寄与しているか？

RQ3：　複数のプロジェクトのメタ情報を用いたモデルは，どのように変化するか？

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．

4.1　データセット

テストの対象として，Apache[1]が提供するOSSの中から，表4に示す20個のJavaプロジェクトを収集した．また，各プロジェクトにおいて，コミット数や開発履歴から収集したマージコンフリクトなどは，表5の通りであった．

4.2　解消方法の判定

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

表 4: テストに用いたプロジェクトの一覧

beam	camel	cassandra
cordova-android	curator	dubbo
flink	geode	groovy
hbase	hive	ignite
incubator-heron	jmeter	lucene-solr
mahout	maven	nifi
nutch	rocketmq	

[1]https://www.apache.org/

表 5: データセットのコミット，マージコミット，マージコンフリクトの件数

プロジェクト名	コミット	マージコミット	マージコンフリクトを含むマージコミット	マージコンフリクト
beam	21898	7159	43	981
camel	37666	203	19	37
cassandra	24837	9885	3941	13132
cordova-android	3761	537	117	700
curator	2615	430	69	255
dubbo	3446	127	31	426
flink	16929	653	161	2152
geode	4268	217	54	427
groovy	16060	767	97	294
hbase	16517	19	6	108
hive	13480	325	159	4809
ignite	26085	6123	610	1703
incubator-heron	2930	349	26	56
jmeter	16212	1	1	394
lucene-solr	31876	621	134	1606
mahout	4115	59	9	105
maven	10519	34	6	248
nifi	5145	360	22	591
nutch	2910	253	16	442
rocketmq	1064	128	13	46

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 6 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解消方法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 5 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．

表 6: モデルの判定によるペアごとの正答率

プロジェクト名	正答率	プロジェクト名	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%
		最大値	94.43%
		最小値	23.83%
		平均値	66.41%

正答率が高かったプロジェクトについて，さらに調査を行う．正答率が 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，マージコンフリクトのおよそ 60%は，(DELETE, ADOPT) 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 7 のようになった．表 7 から，解消方法 (DELETE, ADOPT) に対して，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．

表 7: 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

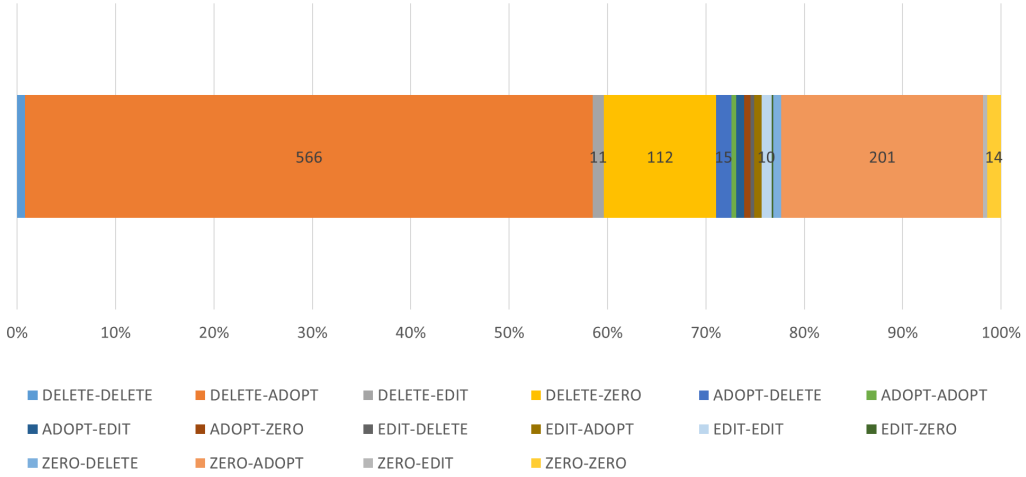

図 11: beam のテストデータと判定結果

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．

4.3 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判定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，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 12 の通りとなった．ここで，図

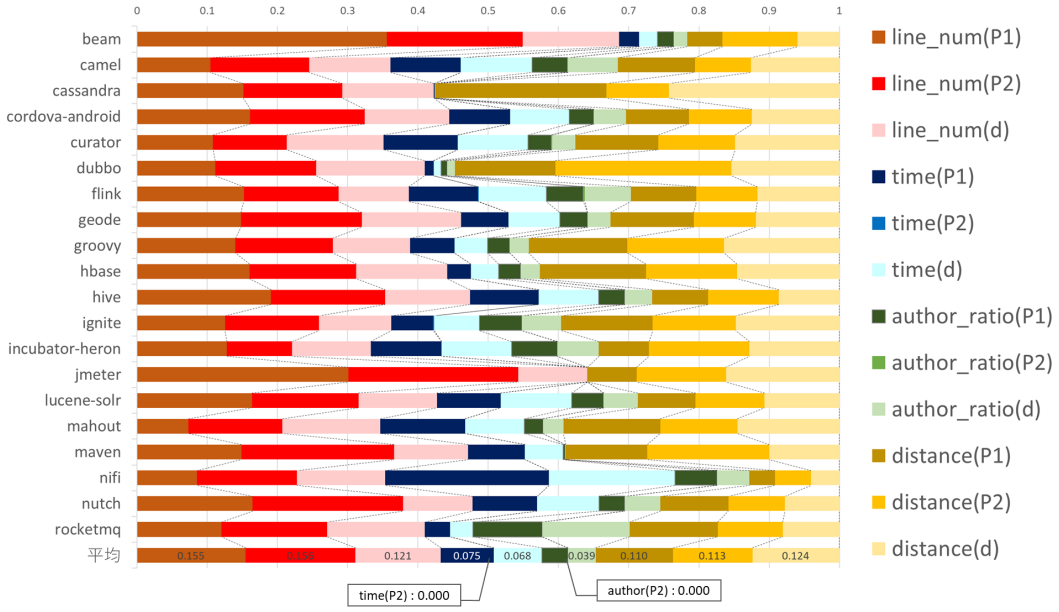

図 12: パラメータの重要度

12 の平均は，各パラメータの重要度に対して，20 個のプロジェクトの平均を取ったものである．

表 8 より，マージコンフリクトの発生箇所の行数は，`line_num(P1)` と `line_num(P2)` が，共に 0.15 以上と大きくなっている．また，図 12 より，`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，いずれのプロジェクトにおいても，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，`distance(P1)` と `distance(P2)`，そしてその差分 `distance(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，`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．

表 8: パラメータの重要度の平均値

パラメータ名	重要度
line_num(P1)	0.155
line_num(P2)	0.156
line_num(d)	0.121
time(P1)	0.075
time(P2)	0.000
time(d)	0.068
distance(P1)	0.110
distance(P2)	0.113
distance(d)	0.124
author_ratio(P1)	0.037
author_ratio(P2)	0.000
author_ratio(d)	0.039

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 (line_num(P1), line_num(P2), line_num(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 (distance(P1), distance(P2), distance(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_ratio(P2)) は，あまり判定に寄与していないと言える．

4.4 複合モデルの作成

今回のテストに用いた 20 個すべての OSS のマージコンフリクトを用いて，各プロジェクトに対して行ったテストと同様のものを行う．全ての OSS から収集したマージコンフリクトの発生件数は，28,512 件であった．

データを 5 分割交差検証によってテストした結果は，正答率は平均で 58.1%であり，分類は図 13 のようになった．58.1%という正答率は，プロジェクトごとに判定した場合に比べて正答率が低下している．

表 9: 複数のプロジェクトからなるモデルの重要度

パラメータ名	重要度
line_num(P1)	0.149
line_num(P2)	0.134
line_num(d)	0.103
time(AP1	0.069
time(P2)	0.000
time(d)	0.056
distance(P1)	0.154
distance(P2)	0.089
distance(d)	0.159
author_ratio(P1)	0.044
author_ratio(P2)	0.000
author_ratio(d)	0.043

また，複数のプロジェクトから作成したモデルの場合，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対しては判定精度が向上する能性がある．

次に，パラメータの重要度を見ると，表 9 のようになった．その傾向は，各プロジェクトから作成したモデルの重要度の平均値と，ほぼ同等であった．つまり，マージコンフリクトの発生件数が多いプロジェクトに，モデルの判定方法は過度な影響を受けていないと言える．

RQ3：複数のプロジェクトのメタ情報を用いたモデルは，どのように変化するか．

マージコンフリクトの解消方法に対する正答率は約 58%であり，プロジェクトごとのモデルに比べて低下する傾向にある．しかし，今後プロジェクト数やパラメータの種類などを増加させることで，判定精度が上昇する可能性が残されている．
また，モデルの重要度に関しては，プロジェクトごとに作成したモデルの平均値と比べて，変わった傾向は見られなかった．

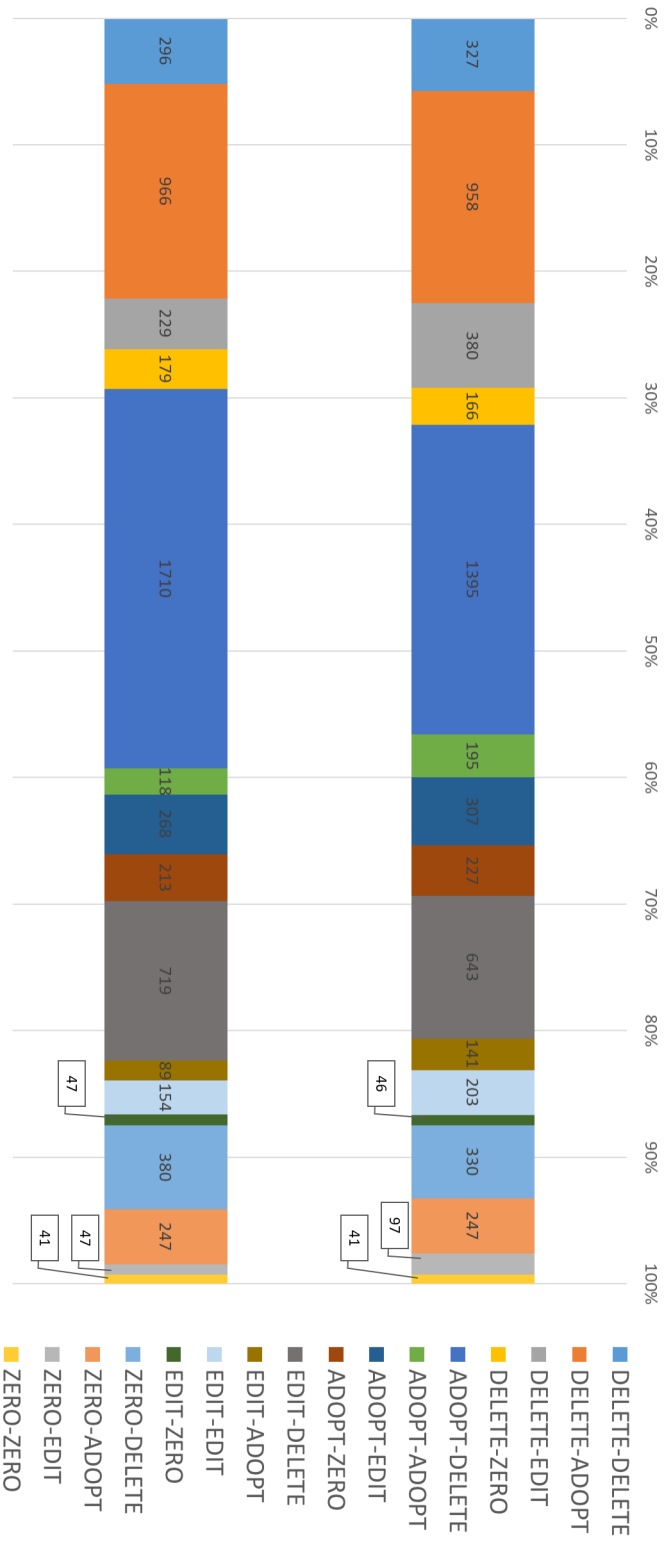

図 13: 複数のプロジェクトからなるモデルによる分類結果

5　妥当性への脅威

5.1　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と考える．

5.2　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 が提供する 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，開発規模や用途が偏っているものではないため，多様な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．

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．

また，複数のプロジェクトの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は，プロジェクトを増やすことで，今後さらなる精度が向上の余地がある．もしも実用可能な精度のモデルができれば，適用するプロジェクトのマージコンフリクトの発生件数に左右されることがないため，小規模なプロジェクトやプロジェクトのスタートアップ時にも適用できるようになり，ソフトウェア開発に大きく貢献すると思われる．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授には，研究において多くの御指導およびい御助言を賜りました．井上教授の御指導のおかげで，本研究を進めることができ，本論文の完成に至りました．井上 教授に心より感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には，研究の各段階において多くの御助言を賜りました．多くの御指導および御助言を頂いた松下 准教授に深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田 哲也 助教には，研究において大変多くの御指導および御助言を賜りました．また，研究に関する数多くの相談にも乗っていただき，マージコンフリクトや機械学習に関する数多くの知識をご教示頂きました．神田 助教に心より感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 春名 修介 特任教授には，研究において御指導および助言を賜りました．春名 特任教授に深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 嶋利 一真 氏には，研究に関する相談に乗っていただき，特に，論文執筆にあたって数多くの御指導および御助言を賜りました．嶋利 一真 氏に心より感謝いたします．

最後に，大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様には，多くの御指導及び御助言を頂きました．本論文を完成させることができたことは，井上研究室の皆様の御協力のおかげであると感じております．井上研究室の皆さまに，心より感謝いたします．

参考文献

[1] 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.

[2] 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.

[3] 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.

[4] Klissiomara Dias, Paulo Borb, and Marcos Barretoa. Understanding predictive factors for merge conflicts. *Information and Software Technology*, Vol. 121, No. 106256, 2020.

[5] 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.

[6] 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.

[7] Ernst Lippe. Operation-based Merging. In *the fifth ACM SIGSOFT symposium on Software development environments(SDE5)*, pp. 78–87, 1992.

[8] Wardah Mahmood, Moses Chagama, Thorsten Berger, and Regina Hebig. Causes of Merge Conflicts: A Case Study of ElasticSearch. In *International Working Conference on Variability Modelling of Software-Intensive Systems*, 2020.

[9] Shaun Phillips, Jonathan Sillito, and Rob Walker. Branching and Merging:An Investigation into Current Version Control Practices. In *the 4th International Workshop on Cooperative and Human Aspects of Software Engineering(CHASE)*, pp. 9–15, 2011.

[10] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Proactive Detection of Collaboration Conflicts. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, p. 168–178, 2011.

[11] Jae young Bang, Daniel Popescu, and Nenad Medvidovic. Enabling Workspace Awareness for Collaborative Software Modeling. In *ACM Conference on Computer Supported Cooperative Work(CSCW)*, 2012.

[12] Fabrizio Pastore, Leonardo Mariani, and Daniela Micucci. BDCI: Behavioral Driven Conflict Identification. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 570–581, 2017.

[13] 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.

[14] Tom Mens. A State-of-the-Art Survey on Software Merging. No. 5, pp. 449–462, 2002.

[15] Sven Apel, Jörg Liebig, Benjamin Brandl, and Christian Kästner Christian Lengauer. Semistructured Merge:Rethinking Merge in Revision Control Systems. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 190–200, 2011.

[16] Sven Apel, Olaf Leßenich, and Christian Lengauer. Structured merge with autotuning: balancing precision and performance. In *International Conference Automated Software Engineering(ASE)*, pp. 120–129, 2012.

[17] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. Understanding semistructured merge conflict characteristics in open-source Java projects. Vol. 23, p. 2051–2085, 2018.

[18] MARCELO SOUSA, ISIL DILLIG, and SHUVENDU K. LAHIRI. Verified Three-Way Program Merge. *Proceedings of ACM on Programming Languages*, Vol. 2, No. 165, pp. 1–29, 2018.

[19] 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.

[20] 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.