一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2021-2,KBSE2021-14(2021-07)

複数ファイルの履歴を考慮したGitファイル追跡精度改善の提案

前島　葵† 肥後　芳樹† 楠本　真二†

† 大阪大学大学院情報科学研究科
〒 565-0871 大阪府吹田市山田丘 1-5
E-mail: †{a-maejim,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　Git に代表される版管理システムから得られるファイル変更履歴は，バグモジュールの検出や同時変更コードの分析などマイニングソフトウェアリポジトリ（MSR）に関する様々な研究に役立てられている．そのため，変更履歴に誤りが存在すれば MSR の研究に影響を与えてしまう．本研究では，版管理システムとして広く用いられている Git を対象にファイル変更履歴の追跡精度の調査を行う．Git にはファイルの名前変更や移動を検出し，ファイルの変更を追跡する機能が備わっている．しかし，Git の追跡機能はファイルの追跡を個別に行うため，名前変更時に複数ファイルの追跡結果が被ってしまう問題がある．そこで，Git の追跡ではどの程度変更履歴が被ってしまうのか調査を行い，より精度の高い追跡を目指した複数ファイルの変更を考慮する手法を提案する．オープンソースプロジェクトに対する調査の結果，145 件の Java プロジェクトのうち 45 件のリポジトリにおいて変更履歴の被りが存在した．また，複数ファイルの変更を考慮することで追跡精度が 17%向上し提案手法が有効であることが示された．
キーワード　ソフトウェアリポジトリマイニング，Git，バージョン管理システム

1. はじめに

Git に代表されるバージョン管理システムは現在多くのソフトウェア開発プロジェクトに利用されている．開発者は，バージョン管理システムが提供する機能を用いることで，過去の任意のバージョンの状態に復元する，バージョン間の差分を確認する，複数の開発者が書いたコードを統合する等が容易に行える．

バージョン管理システムに記録される変更履歴には，開発者と研究者の両者にとって有用な情報が豊富に含まれている．開発者は，ソースコードがどのように進化したか理解するため，コードレビューやチーム開発での情報共有の情報源として変更履歴を利用する [1]〜[3]．研究者は，ソフトウェアリポジトリを解析し，蓄積された情報から有用な知見を見つけ将来のソフトウェア開発に活かすソフトウェアリポジトリマイニングを行なっている．例えば，バージョン管理システムが提供する履歴を利用することで，バグを含む変更の予測や同時に変更される可能性の高いコードの予測を行う [4], [5]．

ソフトウェア開発において変更の理解は重要であり，バージョン管理システムから得られる変更履歴は，ソースコード理解のための重要な情報源として利用されている．そのため，変更履歴に誤りが存在すればソフトウェアリポジトリマイニングの研究に影響を与えてしまう．

ソフトウェア開発の過程では，多くの変更が加えられる．ソースコードの変更には、単一のコードを書き換える簡単な変更から，ファイル全体に影響を及ぼすクラス名の変更やパッケージの移動など複雑な変更まで存在する．バージョン管理システムではソースコードはファイル単位で管理される．Git のコマンドを用いることで，どのコミットで誰がどのファイルにどのような変更を加えたかといった情報が容易に取得できる．

Git にはファイルの名前変更や移動を検出し，変更を追跡する機能が備わっている．ファイル名が変更された時，以前のコミットに存在したファイルの中で最も類似度の高いファイルを変更前のファイルとして名前変更が行われたと見なす．しかし，Git の追跡は同時に複数のファイルの変更を考慮せず，1 つのファイルを対象とした変更しか追跡しない．そのため，個々には最適な追跡が行われていても，個々に得た変更履歴をまとめ，全てのファイルの変更履歴を確認すると途中からファイルの変更履歴が被ってしまう問題が存在する．145 件の Java オープンソースプロジェクトリポジトリに対して調査を行なったところ，47 件のリポジトリにおいて名前変更による変更履歴の被りが存在した．

本研究では，より精度の高いファイル追跡を目指し，複数ファイルの変更を考慮する手法を提案する．提案手法はオープンソースの Git クライアントである JGit を拡張し実装を行なった．名前変更による変更履歴の被りが存在した変更に対して既存の Git と比較実験を行なった結果，既存の Git の追跡の適合率 63%に対して，拡張後 JGit の適合率は 17%向上し，80%の適合率となった．

2. 研究動機

オープンソースプロジェクトである OpenRefine における例を示す．図 1 は，“`git log --follow`” コマンドを，ファイル “ReconMarkNewTopicsOperation.java” と “ReconDiscardJudgmentsOperation.java” に対して実行した結果である．Git

```
$ git log --follow --name-status --oneline
./main/src/com/google/refine/operations/recon/ReconMarkNewTopicsOperation.java
e4e73d0b9 M main/src/com/google/refine/operations/recon/ReconMarkNewTopicsOperation.java
...
c98a8ad55 R095 src/main/java/com/metaweb/gridworks/model/operations/ReconMarkNewTopicsOperation.java
              src/main/java/com/metaweb/gridworks/operations/ReconMarkNewTopicsOperation.java
1227c9dff R068 src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
              src/main/java/com/metaweb/gridworks/model/operations/ReconMarkNewTopicsOperation.java
4b2e48614 M src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
604dd53eb M src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
8c41af9c1 M src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
...
bacb71ab6 A src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
```

(a) ReconMarkNewTopicsOperation.java の変更履歴

```
$ git log --follow --name-status --oneline
./main/src/com/google/refine/operations/recon/ReconDiscardJudgmentsOperation.java
e4e73d0b9 M main/src/com/google/refine/operations/recon/ReconDiscardJudgmentsOperation.java
...
c98a8ad55 R095 src/main/java/com/metaweb/gridworks/model/operations/ReconDiscardJudgmentsOperation.java
              src/main/java/com/metaweb/gridworks/operations/ReconDiscardJudgmentsOperation.java
1227c9dff R056 src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
              src/main/java/com/metaweb/gridworks/model/operations/ReconDiscardJudgmentsOperation.java
4b2e48614 M src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
604dd53eb M src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
8c41af9c1 M src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
...
bacb71ab6 A src/main/java/com/metaweb/gridworks/model/operations/ApproveNewReconOperation.java
```

(b) ReconDiscardJudgmentsOperation.java の変更履歴

図 1　名前変更によって変更履歴が被る例

```
$ jgit log --follow --name-status --oneline
./main/src/com/google/refine/operations/recon/ReconDiscardJudgmentsOperation.java
e4e73d0b9 M main/src/com/google/refine/operations/recon/ReconDiscardJudgmentsOperation.java
...
c98a8ad55 R098 src/main/java/com/metaweb/gridworks/model/operations/ReconDiscardJudgmentsOperation.java
              src/main/java/com/metaweb/gridworks/operations/ReconDiscardJudgmentsOperation.java
1227c9dff R057 src/main/java/com/metaweb/gridworks/model/operations/DiscardReconOperation.java
              src/main/java/com/metaweb/gridworks/model/operations/ReconDiscardJudgmentsOperation.java
4b2e48614 M src/main/java/com/metaweb/gridworks/model/operations/DiscardReconOperation.java
604dd53eb M src/main/java/com/metaweb/gridworks/model/operations/DiscardReconOperation.java
8c41af9c1 M src/main/java/com/metaweb/gridworks/model/operations/DiscardReconOperation.java
...
bacb71ab6 A src/main/java/com/metaweb/gridworks/model/operations/DiscardReconOperation.java
```

図 2　提案手法による改善結果

によって追跡を行った結果，“ReconMarkNewTopicsOperation.java” はコミット 1227c9dff で “ApproveNewReconOperation.java” から名前変更が行われいる．また，“ReconDiscardJudgmentsOperation.java” も同様に，コミット 1227c9dff で “ApproveNewReconOperation.java” から名前変更が行われいる．個別に得られた 2 つのファイルの変更履歴を合わせると，共にコミット 1227c9dff 以前の変更履歴が被ってしまっていることがわかる．しかし実際には，“ReconDiscardJudgmentsOperation.java” は “ApproveNewReconOperation.java” からの名前変更ではなく，“DiscardReconOperation.java” からの名前変更である．

これらの二つのファイルは共に共通した親クラスを持つためソースコードの内容が似通っていた．Git のアルゴリズムでは対象とするファイル 1 つに対して，類似度が最も高いファイルを選び追跡を行う．そのため，個別には最適な追跡ができているように見えても，2 つのファイルに対しての変更履歴を合わせると途中から同じファイルを追跡してしまっていた．

提案手法では，単に最も類似度の高い追加，削除ファイル間を名前変更と見なすのではなく，変更が行われた全てのファイル間の類似度を考慮することで誤った追跡を防ぐ．上述した状況において，提案手法を “ReconDiscardJudgmentsOperation.java” に対して実行した結果が図 2 であり，提案手法が有効に働いている．

3. 予 備 調 査

まず初めに，実際のソフトウェアリポジトリ中にはどの程度変更履歴の被りが存在するか調査を行なった．

3.1 調 査 対 象

Borges らのデータセット [6] を使用し，GitHub に存在するスター数上位のオープンソースプロジェクトを調査対象とした．Borges データセットには，GitHub におけるスター数上位 2,279 のプロジェクトが含まれる．その中から，Java プロジェクトと分類される 202 件のプロジェクトのうち 2021 年 5 月に GitHub 上にリポジトリが存在した 197 件のプロジェクトを対象とした．さらに，テストコードを除き Java ファイルを 2 つ以上持つプロジェクト 145 件を調査対象とした．図 3 には調査対象とした 145 件のプロジェクトについて，コミット数と Java ファイル数の分布を示している．分布からわかる通り，様々な規模のプロジェクトを本研究では調査対象とした．Java ファイル数の規模が大きなプロジェクトは順に，elasticsearch が 9,201 ファイル，gradle が 8,624 ファイルである．また，コミット数の規模が大きなプロジェクトは順に，platform_frameworks_base が 560,597 件，intellij-community が 302,536 件であった．

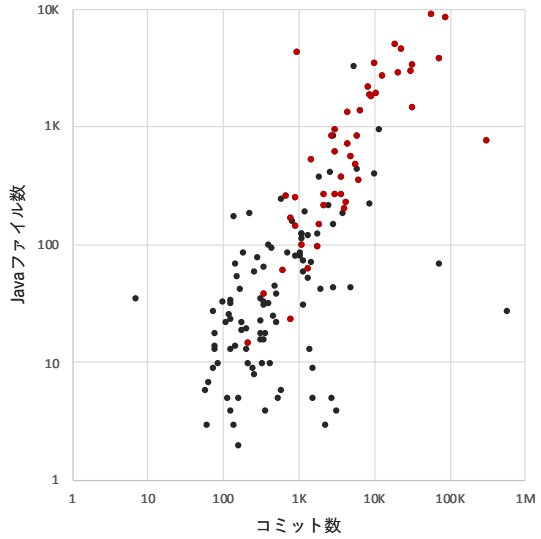


図 3　調査対象プロジェクト規模

3.2 調 査 方 法

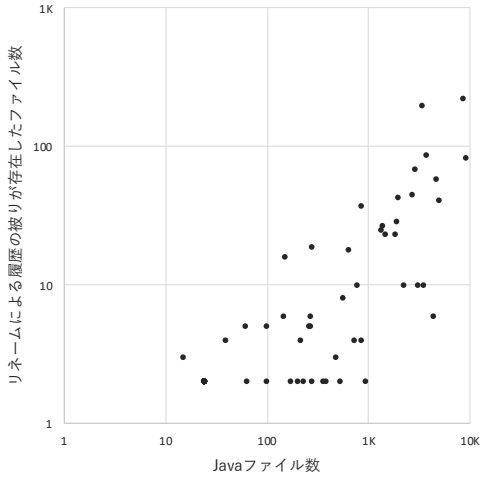
調査対象プロジェクトに存在する Java ファイルに対して Master の最新コミットを起点に “`git log --follow [file]`” コマンドを実行し，変更履歴を得た．変更履歴を分析し，同一のコミットにおいて名前変更前ファイルが被っているファイルを抽出した．

3.3 調 査 結 果

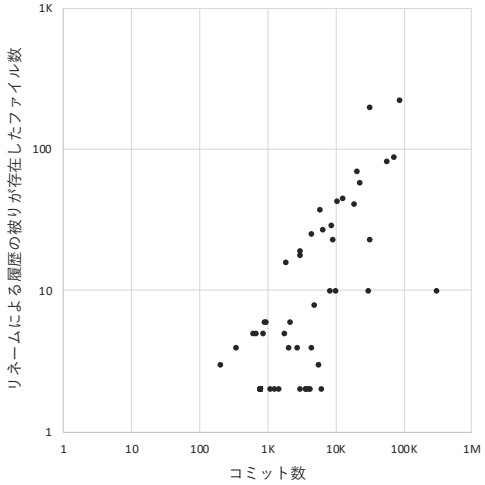
図 3 に示した調査対象プロジェクトの中で，赤い点は名前変更による変更履歴の被りが存在したプロジェクトの分布を示している．調査対象プロジェクト 145 件に対して，変更履歴の被りを持つプロジェクトは 47 件 (32%) 存在した．また，Java ファイル数が 10 以上のプロジェクトに対象を絞ると，126 件中の 47 件 (37%) である．Java ファイル数が 100 以上のプロジェクトに対象を絞ると，63 件中の 41 件 (65%) である．Java ファイル数が 1000 以上のプロジェクトに対象を絞ると，19 件中の 17 件 (89%) である．コミット数や Java ファイル数が少ないプロジェクトでは履歴の被りが発生せず，プロジェクトの規模が大きいほど履歴の被りが存在し易い結果になった．

図 4 に Java ファイルに対する名前変更による変更履歴の被り数，コミット数に対しての名前変更による変更履歴の被りが存在したファイル数を示す．変更履歴の被りが存在したファイル数が多いプロジェクトは順に，8,624 ファイルに対して 2.6%の 224 ファイルに被りが存在した gradle，3,447 ファイルに対して 5.8%の 199 ファイルに被りが存在した spring-boot，3,807 ファイルに対して，2.3%の 87 ファイルの被りが存在した neo4j である．変更履歴の被りが存在した割合が多いプロジェクトは順に，15 ファイルに対して 20.0%の 3 ファイルに被りが存在した otto，153 ファイルに対して 10.5%の被りが存在した retrofit である．

図 4 に示される通りプロジェクトの規模 Java ファイル数と変更履歴の被りには相関があり，規模が大きなプロジェクトほど大規模な名前変更やパッケージの移動が起こり易いため変更履歴が被りやすい結果になった．



(a) Java ファイル数に対しての被りファイル数



(b) コミット数に対しての被りファイル数

図 4　名前変更による変更履歴の被り数

4. 提 案 手 法

本研究では，ファイルのより高精度な追跡を目指し複数ファイルの変更を考慮する手法を提案する．提案手法は以下の Step からなる．図 5 に提案手法の概要を示す．提案手法の入力は追加ファイル群と削除ファイル群である．提案手法は，入力として与えられた各追加ファイルについて，十分に類似した削除ファイルが見つかった場合にはそれらを対応づけることによりファイル名が変更されたと見なす．類似した削除ファイルが見つからなかった場合にはそのファイルは真に追加されたと見なす．以下の Step は，Java ファイルが追加および削除された各コミットに対して実行される．

Step-1　追加ファイルと削除ファイルの全てのペアに対して，それらファイル間の類似度を計算．

Step-2　追加削除ファイルのペアを類似度の降順に整列．

Step-3　整列した各ペアについて，類似度の降順に以下の処理を行う．そのペアに含まれる削除ファイルが他の追加ファイルの名前変更と見なされていない場合には，その削除ファイルを現在のペアである追加ファイルの名前変更と見なす．すでにその削除ファイルが他の追加ファイルの名前変更と見なされている場合には，そのペアについてはなにもしない．

図 5　提案手法の概要

Step-4　しきい値よりも類似度の高い削除ファイルを持つが，どの削除ファイルとも名前変更と見なされなかった追加ファイルについては，もっとも類似度が高い削除ファイルのコピーであると見なす．

Step-5　どの削除ファイルとも名前変更もしくはコピーと見なされていない追加ファイルについては，真に追加されたファイルであると見なす．

以降，各 Step について説明する．Step-1 では，追跡対象であるファイルと削除された各ファイル間の類似度だけでなく，追加された各ファイルと削除された各ファイルの全てのペアについて類似度を計算する．JGit ではファイルの名前変更は，ファイルの内容の類似度とファイルパスの類似度の二つを利用して判断される．従来の JGit では，ファイルパスの類似度はパス全体の文字列の一致度であるが，提案手法では，ファイルパスの類似度をレーベンシュタイン距離を用いて行うように変更した．また，名前変更と見なす類似度のしきい値は Git のデフォルト値である 50%とした．

Step-2 では，類似度の高いファイルペアから名前変更と見なすために，追加削除ファイルのペアを類似度の降順に整列する．Step-3 では，整列した各ペアについて，類似度の降順に以下の処理を行い名前変更ファイルペアを決定する．そのペアに含まれる削除ファイルが他の追加ファイルの名前変更と見なされていない場合には，その削除ファイルを現在のペアである追加ファイルの名前変更と見なす．すでにその削除ファイルが他の追加ファイルの名前変更と見なされている場合には，そのペアについてはなにもしない．Step-4 では，コピーの判定を行う．名前変更のペアが決定しなかった追加ファイルはしきい値よりも高い類似度を持つ削除ファイルが存在した場合に，その中で最も類似度が高い削除ファイルのコピーと見なす．この時，低い類似度のファイルが誤って追跡されることを防ぐため，名前変更ファイル間の類似度に対して 15%以上類似度差があるファイルはコピーと見なさない．例えば，追加ファイル “RenameA.java” と削除ファイル “A.java” が 90%で名前変更と見なされたとする．別の追加ファイルである “CopyA.java” と削除ファイル “A.java” の類似度が 80%であるならば，名前変更ペアに対しての類似度差は 10%である．この場合，類似度差が 15%以内のためコピーと見なす．図 6 に目視結果によってファイルのコピーであると見なした 63 の Java ファイルについて，名前変更ファイルペアに対する類似度差を示している．図 6 から本研究では正しいコピーを 75%程度含める 15%を閾値に採用した．

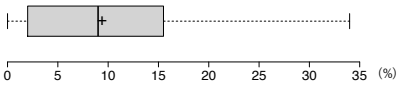


図 6　名前変更ペアに対するコピーの類似度差

```
commit 2752370baf5f1b928805f8fb4312afe3703fb740
R076
subprojects/dependency-management/src/main/java/org/gradle/api/internal/artifacts/repositories/layout/PatternRepositoryLayout.java
subprojects/dependencymanagement/src/main/java/org/gradle/api/internal/artifacts/repositories/layout/DefaultIvyPatternRepositoryLayout.java

-public class PatternRepositoryLayout extends RepositoryLayout {
+public class DefaultIvyPatternRepositoryLayout extends AbstractRepositoryLayout implements IvyPatternRepositoryLayout {
     private final Set<String> artifactPatterns = new LinkedHashSet<String>();
     private final Set<String> ivyPatterns = new LinkedHashSet<String>();
     /**
-     * Adds an Ivy artifact pattern to define where artifacts are located in this repository.
-     * @param pattern The ivy pattern
+     * {@inheritDoc}
      */
     public void artifact(String pattern) {
         artifactPatterns.add(pattern);
     }
....
```

(a) DefaultIvyPatternRepositoryLayout.java

```
commit 2752370baf5f1b928805f8fb4312afe3703fb740
R054
subprojects/dependency-management/src/main/java/org/gradle/api/internal/artifacts/repositories/layout/PatternRepositoryLayout.java
subprojects/core/src/main/groovy/org/gradle/api/artifacts/repositories/IvyPatternRepositoryLayout.java

-public class PatternRepositoryLayout extends RepositoryLayout {
-    private final Set<String> artifactPatterns = new LinkedHashSet<String>();
-    private final Set<String> ivyPatterns = new LinkedHashSet<String>();
-    private boolean m2compatible;
+public interface IvyPatternRepositoryLayout extends RepositoryLayout {

     /**
      * Adds an Ivy artifact pattern to define where artifacts are located in this repository.
      * @param pattern The ivy pattern
      */
-    public void artifact(String pattern) {
-        artifactPatterns.add(pattern);
-    }
+    void artifact(String pattern);
....
```

(b) IvyPatternRepositoryLayout.java

図 7　gradle における履歴の被り例

5. 評価実験

5.1 実験概要

本実験の目的は，既存の Git と拡張後の JGit の変更履歴を比較し，どの程度適合率が向上するか調査することである．Git において変更履歴の被りが存在したファイルに対して，拡張後 JGit を利用し変更履歴の追跡を行う．

得られた Git,JGit の結果に対して，名前変更による変更履歴の被りが発生した変更前後のソースコードの目視確認を行い，名前変更が正しいかどうか確認を行なった．この時，目視によって正しく追跡できていると判定できた割合を適合率とする．

5.2 実験対象

3. で述べた予備実験において名前変更による変更履歴の被りが存在した 47 件のオープンソース Java プロジェクトにおける 677 の Java ファイルを実験対象とした．

5.3 実験結果

表 1 に Git，拡張後の JGit が正しく追跡できたファイル数とその割合を示す．Git の追跡の適合率が 63%に対して，拡張後は 17%適合率が向上し，適合率は 80%となった．また，表 2 に 3 ファイル以上被りが存在した変更に対象を限定し，適合率を示す．多くのファイルの変更が被ってしまった複雑なコミットに対しても，21%適合率が向上し提案手法が有効であるとわかった．

表 3 に 677 の Java ファイルについて，Git，拡張後 JGit における追跡の正解数とその割合を示している．名前変更全体の 57.3%は Git，拡張後 JGit で共に正しく追跡し，全体の 14.6%は Git，拡張後 JGit で共に誤った追跡を行なった．

表 1　適　合　率

名前変更全体	677
Git の適合率	426(63%)
拡張後 jGit の適合率	542(80%)

6. 考　　察

本章では 5. で述べた評価実験についての考察を行う．追加ファイルと削除ファイルの全てのペアを考慮することで追跡適合率の向上が見られた例，向上が見られなかった例について考察する．

図 7 に追跡適合率の向上が見られた例を示す．図 7 にオープンソースプロジェクトである gradle において，従来の Git で変更履歴が被ってしまった 2 つの Java ファイルを示す．‘DefaultIvyPatternRepositoryLayout.java” と ‘IvyPatternRepositoryLayout.java” はそれぞれ 76%，54%の類似度によって共に ‘PatternRepositoryLayout.java” からの名前変更と見なされている．拡張後の JGit では前者の追跡のみを行い，後者の追跡は行わないため低い類似度のファイルを名前変更と見なしていた誤検出の削減に成功した．このプロジェクトに含まれる全ての Java ファイルには Apache ライセンスに関する条文が含まれており，その存在が関係ない削除ファイルと追加ファイル間の類似度を高くしていたため，従来の Git では

表 2　適合率（3 ファイル以上被りが存在した変更）

名前変更全体	191
Git の適合率	99 (51%)
拡張後 jGit の適合率	139(72%)

表 3　追跡の正誤数とその割合

	Git で正しく追跡	Git で誤った追跡
jGit で正しく追跡	388(57.3%)	151(22.3%)
jGit で誤った追跡	39 (5.8%)	99 (14.6%)

```
$ git log -p -U100 -follow client/src/main/java/org/asynchttpclient/netty/channel/TransportFactory.java
commit 1e1a84409e551d8455faa9595c09b56d46e63f74
R067 client/src/main/java/org/asynchttpclient/netty/channel/NioSocketChannelFactory.java
     client/src/main/java/org/asynchttpclient/netty/channel/TransportFactory.java
 /*
- * Copyright (c) 2016 AsyncHttpClient Project. All rights reserved.
+ * Copyright (c) 2019 AsyncHttpClient Project. All rights reserved.
  *
  * This program is licensed to you under the Apache License Version 2.0,
  * and you may not use this file except in compliance with the Apache License Version 2.0.
  * You may obtain a copy of the Apache License Version 2.0 at
  *     http://www.apache.org/licenses/LICENSE-2.0.
  *
  * Unless required by applicable law or agreed to in writing,
  * software distributed under the Apache License Version 2.0 is distributed on an
  * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the Apache License Version 2.0 for the specific language governing permissions and limitations there under.
  */
 package org.asynchttpclient.netty.channel;

+import io.netty.channel.Channel;
 import io.netty.channel.ChannelFactory;
-import io.netty.channel.socket.nio.NioSocketChannel;
+import io.netty.channel.EventLoopGroup;

-enum NioSocketChannelFactory implements ChannelFactory<NioSocketChannel> {
+import java.util.concurrent.ThreadFactory;

-  INSTANCE;
+public interface TransportFactory<C extends Channel, L extends EventLoopGroup> extends ChannelFactory<C> {
+
+  L newEventLoopGroup(int ioThreadsCount, ThreadFactory threadFactory);

-  @Override
-  public NioSocketChannel newChannel() {
-    return new NioSocketChannel();
-  }
 }
```

図 8　async-http-client の例

誤って名前変更と見なしてしまっていた．提案手法では，全ての削除追加ファイルのペアに対して類似度を算出するため，関係ないファイル間の類似度がたとえしきい値以上となってしまっても，本来名前変更と見なされるべきファイルのペアの類似度がそれ以上に高くなるために正しく名前変更と見なすことができた．

しかし，提案手法でも正しく追跡できなかった名前変更も存在した．図 8 に追跡適合率の向上が見られなかった例を示す．図 8 にオープンソースプロジェクトであるに async-http-client おいて，誤って名前変更と見なされた変更の例を示す．誤って追跡を行なった原因は，ファイル行数の少なさによる類似度の精度不足であった．ファイル全体に対し，ライセンスに関する条文などプロジェクト全体に共通で多く含まれる部分の割合が高い場合は，類似度が高く算出されてしまう．図 8 はファイルの全行数 26 行のうち，コメント部分が半数の 13 行を占めている．そのため，関連のない変更がなされても類似度が 67%と高く算出された．この問題に対しては，コメントなどプロジェクト全体における共通部分を除いた類似度の算出ができれば解決できると考える．

7. 妥当性への脅威

内部的妥当性に対する脅威は，目視確認に関する評価である．精度の調査での判定は，目視を行なった著者らの主観に依存している．この脅威は，データの一部に対して目視確認を二人以上で行い，合意を得ることで軽減した．

8. おわりに

本研究では，オープンソース Java プロジェクトにおける名前変更による変更履歴の被りについて調査を行なった．調査の結果，145 件のリポジトリ中，32%の 47 件のリポジトリにおいて名前変更による変更履歴の被りが存在した．また，ファイル数やコミット数の規模が大きなプロジェクトほど，変更履歴の被りが発生し易いことがわかった．

本研究では，より精度の高いファイル追跡を目指し複数ファイルの変更を考慮する手法を提案した．提案手法はオープンソースの Git クライアントである JGit を拡張し実装を行なった．名前変更による変更履歴の被りが存在した変更に対して既存の Git と比較実験を行なった結果，既存の Git の追跡の適合率 63%に対して，拡張後 JGit の適合率は 17%向上し，80%の適合率となった．

今後の課題として，ソフトウェアリポジトリマイニングの研究において精度の高い追跡を行うことで，既存の実験結果が変わることがないか調査する予定である．

文　献

[1] K. Maruyama, E.Kitsu, T.Omori, and S.Hayashi, “Slicing and replaying code change history,” Proc. IEEE/ACM International Conference on Automated Software Engineering, p.246–249, 2012.

[2] O. Kononenko, O. Baysal, and M.W. Godfrey, “Code review quality: How developers see it,” Proc. International Conference on Software Engineering, p.1028–1038, 2016.

[3] A.J. Ko, R. DeLine, and G. Venolia, “Information needs in collocated software development teams,” Proc. International Conference on Software Engineering, p.344–353, 2007.

[4] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, “Mining version histories to guide software changes,” Proc. International Conference on Software Engineering, p.563–572, 2004.

[5] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” Proce. International Conference on Software Engineering, pp.284–292, 2005.

[6] H. Borges, A. Hora, and M.T. Valente, “Understanding the factors that impact the popularity of github repositories,” Proc. IEEE International Conference on Software Maintenance and Evolution, pp.334–344, 2016.