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
MSS2018-78,SS2018-49(2019-01)

Javaメソッドの高精度な追跡のための細粒度版管理システムの提案

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: †{higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　関数やメソッド単位での変更履歴情報を容易に取得するための手法として，Historage が提案されている．Historage では関数やメソッドが単一のファイルとして扱われているため，それらの単位での変更履歴を容易に取得できる．しかし，Historage ではサイズの小さいファイルを多く扱うことから，ファイルの追跡精度に課題を残している．本研究では，追跡精度の向上を目指して，関数やメソッドを表すファイルの各行に単一の字句のみを配置するファイルフォーマットを提案する．さらに，実験に基づいた 3 つの経験則を導入し，さらなる高精度でのファイルの追跡を目指す．

キーワード　版管理システム，ソースコード解析，ソフトウェアリポジトリマイニング

1. はじめに

現代のソフトウェア開発では，Git(注1)等の版管理システムを用いてソフトウェアプロダクトを管理することは必須である．版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に戻る，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．また，開発者が，版管理システムに蓄えられた履歴情報をデバッグ，コードの理解，変更の意図の確認等に利用することもある [1]．さらに，GitHub(注2)や Bitbucket(注3)等の版管理システムのホスティングサービスを利用することで，課題やレビューの管理をプロダクトと紐付けて行える．これにより，ソフトウェア開発における開発者の活動が効果的に支援されると共にそれらが記録として残る [2]．

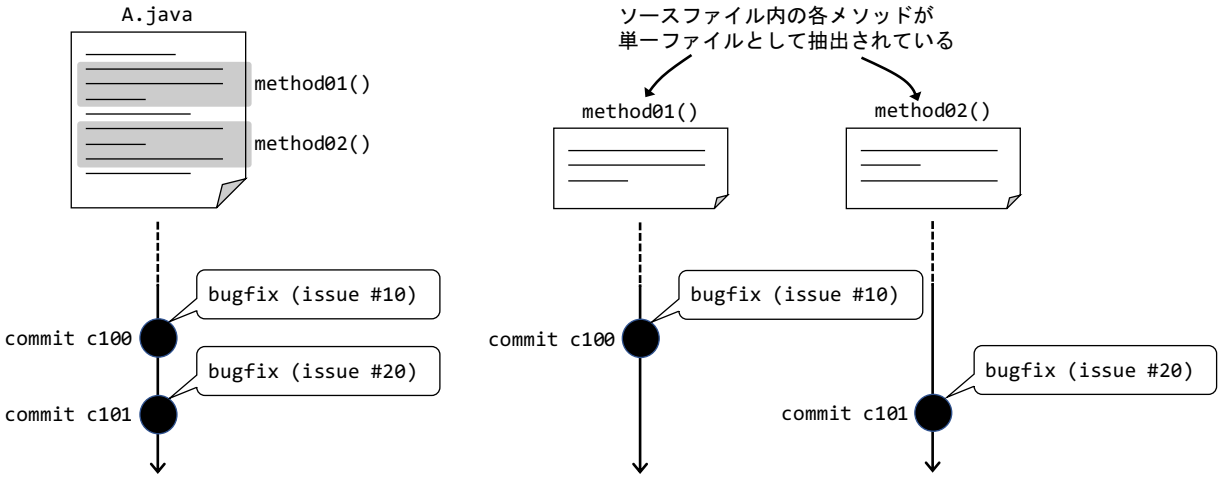
版管理システムやホスティングサービスに記録された情報から有益な知見を見つけ，それを将来のソフトウェア開発に活かすための研究が活発に行われている．その研究の主要な対象の 1 つとしてソフトウェアのソースコードが挙げられる．例えば，バグを含むモジュールを予測する研究や繰り返し行われる変更の内容を抽出する研究が数多く行われている．

版管理システムでは，ソースコードはファイル単位で管理される．ファイル単位で管理されているため，どこのコミットでどのファイルに変更が行われたのか，各ファイルに何回変更が加えられたのかといったファイル単位の情報が容易に取得できる．分析がファイル単位で行われる場合は，版管理システムで管理されているファイルをそのまま用いれば良い．しかし，ファイルよりも細かい粒度である関数やメソッド単位での分析を必要とする研究も多い．そのような研究では，版管理システムからファイルを取得したあとに構文解析を行い，関数やメソッドを取得する必要がある．さらに，各関数や各メソッドに変更が加えられたかどうかを調査するためには，版管理システムから提供されるファイル単位での差分情報と，関数やメソッドの開始位置および終了位置を突き合わせる必要があり，効率的に分析することが難しい．

Hata らは，容易に関数やメソッド単位の変更履歴情報を取得するための細粒度版管理システム Historage を提案した [3]．Historage は既存の Git リポジトリを入力として受け取り，その中に含まれる Java ソースファイル内の各メソッドを単一のファイルとして切り出し，それらを管理するリポジトリを構築する．Historage リポジトリでは，Java の各メソッドがファイルになっているため，メソッド単位での変更情報を容易に取得できる．構文解析や位置情報の突き合わせを行う必要はない．

図 1 は，Git リポジトリとそこから構築した Historage リポジトリの例である．Git リポジトリ（図 1(a)）では `A.java` が管理されている．コミット `c100` と `c101` では，`A.java` が変更されている．ファイル単位での変更履歴は，“`git-log`” 等のコマンドを利用することで容易に取得できる．どのメソッドが変更されたのかを調査する場合には，`A.java` の構文解析を行うことによりメソッドの位置情報を特定し，メソッドの位置情報とファイル単位での変更の位置情報を突き合わせる必要がある．しかし，そのような分析を行うためにはツールの整備等の手間が必要である．それに対して，Historage リポジトリ（図 1(b)）では，各メソッドがファイルとして管理されているため，“`git-log`” 等のコマンドを利用することで容易にメソッド単位での変更情報を得ることができる．図 1 の例では，Git のリポジトリからは `A.java` に対して二度のバグ修正が行われたことがわかる．それに対して，Historage リポジトリを利用することで，メソッド `method01()` とメソッド `method02()` に，一度

(注1)：https://git-scm.com/
(注2)：https://github.com/
(注3)：https://bitbucket.org/



(a) Git リポジトリ　　(b) Historage リポジトリ

図 1　Git と Historage の違い

ずつバグ修正が行われたことが容易にわかる．

しかし，Historage には課題もある．図 2 は，`Person.java` というファイルに対して名前変更リファクタリングが行われ，その内部のフィールド名も同時に変更された様子を表している．変更前ファイルにおいて赤で強調された行と変更後ファイルにおいて緑で強調された行は，変更された行を表している．図 2(a) に示すようにファイル名が変更された場合であっても変更されていない行の割合が大きければ，Git のリネーム検出機能でファイルを追跡できる．しかし，Historage リポジトリの場合は，図 2(b) のように，ファイル全体に対する変更行の割合が大きくなってしまうため，追跡できなくなってしまう．Historage リポジトリでは，Git リポジトリに比べてファイルが小さくなってしまうために，ファイル名が変更された場合の追跡能力が落ちてしまうことが課題である．

Hata らは対応策として，変更されていない行の割合が低くても追跡を行えるように，しきい値を変更することが有効であると主張している [3]．しかし，しきい値を下げることにより，まったく関係のないファイルを誤って同じファイルと見なしてしまう危険性が強くなってしまう．例えば，図 2(b) の場合は，追跡に必要なしきい値をデフォルト値の 0.6 から Hata らが有効だと主張している 0.3 に下げることにより，“`Person/getLength`” から “`Engineer/getHeight`” へと追跡可能になるが，“`Person/getLength`” から誤って “`Engineer/setHeight`” を追跡してしまう可能性もある．

本研究では，追跡精度の向上を目的として，メソッドを切り出したファイルの各行に単一の字句のみを配置することを提案する．また，実験に基づいた 3 つの経験則を導入することにより，さらなる精度向上を試みる．

2. 提 案 手 法

本研究では，Git のメカニズムを用いて Java メソッドの高精度な追跡を行う手法を提案する．提案手法は大きく以下の 2 つの項目からなる．

- Java メソッドを切り出したファイル名の命名規則
- Java メソッドを切り出したファイルの行フォーマット

ファイル名の命名規則

本研究では，Java メソッドを表すファイルはその名前に以下の情報を含むとする．

- そのメソッドを含むクラスの名前
- そのメソッドのアクセス修飾子
- そのメソッドの返値の型
- そのメソッドの名前
- そのメソッドの引数の型

これらの情報をファイル名に含めることにより，対象とするメソッドの周辺におけるさまざまな変更は以下のようにメソッドのファイルに反映される．

ケース 1　対象メソッドを含むクラス名が変更された場合は，対象メソッドのファイルの名前は変わるが内容は変更されない．

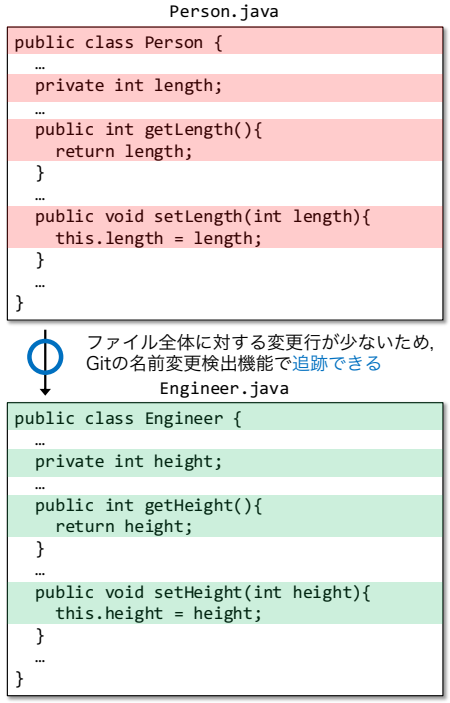
ケース 2　対象メソッドを含むクラスの別のメソッドが変更された場合は，対象メソッドのファイル名も内容も変更されない．

ケース 3　対象メソッドのシグネチャが変更された場合は，対象メソッドのファイル名は変わるが内容は変更されない．

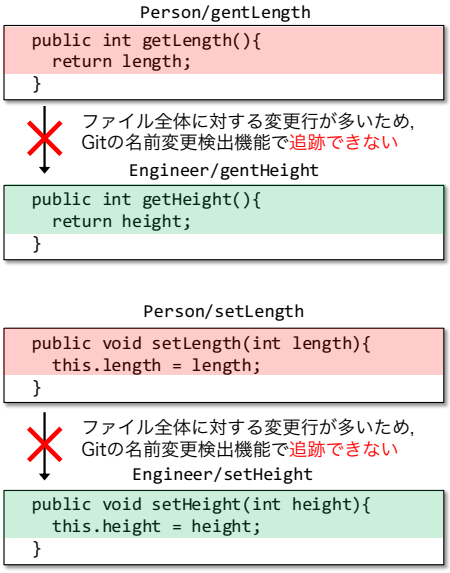
ケース 4　対象メソッドの内容が変更された場合は，対象メソッドのファイル名は変わらないが，内容が変更される．

上記いずれのケースであっても Git のメカニズムを利用することで追跡可能である．提案手法が追跡できないのは，メソッドのシグネチャやそのメソッドを含むクラスの名前が変更され，かつメソッドの内容が大幅に変更された場合である．

Historage でも Java メソッドを表す各ファイルには，そのメソッドのシグネチャ情報が含まれている．Historage と提案手法の違いは，Historage が Java の各クラスを表すディレクトリを作成しそれ以下に Java メソッドを表すファイルを配置するのに対して，提案手法では Java メソッドを表す各ファイルは元の Java ファイルと同一ディレクトリに配置される．提案手法で各クラスからディレクトリを作成するのを避けた理由は，Historage は大規模なリポジトリに対しては処理時間が長くなってしまう傾向にあること，およびその原因の 1 つが各ク



(a) Git



(b) Historage

図 2　Git と Historage の違い

ラスからディレクトリを作成することによるリポジトリのさらなる大規模化であるからである．提案手法を用いた場合でも管理するファイル数は元の Git リポジトリよりも多くなってしまうが，Historage のようにディレクトリ数まで増えてしまうことはない．

ファイルの行フォーマット

Git はハッシュ値を用いてファイルの比較を行う．ハッシュ値は各行から算出されるが，一行が 64 バイトを超える場合には，64 バイト毎にハッシュ値が生成される．このため，行内における変更が小さい場合でも，その行全体やその 64 バイト全体が変わったとみなされる．よって，本研究では，各行に単一の字句のみを配置することを提案する．各行が単一の字句のみを持つため，各ハッシュ値は単一の字句から生成される．

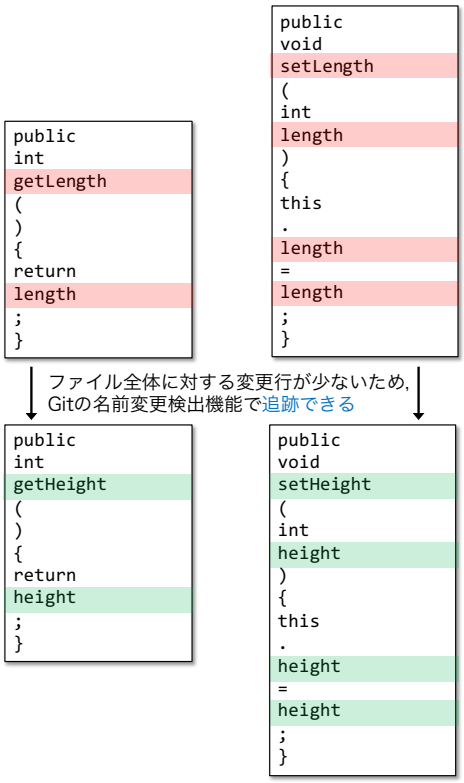


図 3　Git と Historage の違い

図 2(b) では，`getLength` および `setLength` メソッドにおいてメソッド名と変数名が変更されているのみである．しかし，それらを含む行全体が変更されたとみなされてしまうため，ファイル全体に対するされなかった行の割合が 1/3 になってしまい，デフォルトのしきい値である 0.6 より小さくなるため，追跡できない．

図 2(b) の変更を，提案手法を用いて表すと図 3 のようになる．提案手法を利用すると，`getLength` メソッドの変更されなかった行の割合は 8/10(=0.80)，`setLength` の変更されなかった行の割合は 11/15(=0.73) となり，どちらも 0.6 を上回るため追跡できる．

提案手法にはもう 1 つ利点がある．各行が単一字句であるのですでに字句解析が完了した状態であり，そのあとの分析作業に用いやすい．Historage の場合は，例えばメソッドの字句数を計測する場合には，字句解析器を用いる必要があるが，提案手法ではファイルの行数を数えるだけでよい．メソッド内で利用されているユーザ定義名も容易に取得できる．

2.1　予 備 実 験

提案手法の追跡能力を確認するために予備実験を行った．ここでは，追跡の精度を手作業で確認するために，小規模なオープンソースソフトウェアである `paho.mqtt.android`(注4) を対象とした．対象ソフトウェアのリポジトリは 2018 年 9 月 13 日に取得した．取得時点でのコミット数は 195，最新コミットにおけるメソッド数も 195 である．この 195 のメソッドに対して，

(注4)：https://github.com/eclipse/paho.mqtt.android

Historage と提案手法それぞれを利用した場合の追跡の違いを調査した．なお，どちらの手法もしきい値として 0.3 と 0.6 の 2 つを用いた．

予備実験では，しきい値 0.6 を用いた場合の Historage を基準として，その他の 3 つの手法がより長い期間追跡したメソッドおよびより短い期間追跡したメソッドを特定した．そして，それらのメソッドについて，その「より長い期間の追跡」もしくは「より短い期間の追跡」が正しいかどうかを手作業により調査した．

表 1 は，予備実験の結果を表している．「差異メソッド数」は追跡期間が異なったメソッドの数を表しており，「差異の正誤」は手作業による調査結果を表している．Historage(0.3) および提案手法 (0.6) では，基準と比べて 4 つのメソッド（全メソッドの約 4%）のみの追跡結果が異なっていた．一方，提案手法 (0.3) では，30 のファイル（全メソッドの 15%）について基準と異なる結果であった．Historage(0.3) および提案手法 (0.6) は，4 つの異なる追跡結果のうち 3 つが正しく追跡できたメソッドであり，残りの 1 つが誤った追跡となってしまったメソッドであった．一方，提案手法 (0.3) では，正しく追跡できていたメソッドは 6 つのみであり，残りの 24 のメソッドは誤った追跡であった．

予備実験の結果をまとめると次のようになる．

- 提案手法でしきい値 0.6 を用いた場合には，Historage のしきい値 0.3 を用いた場合は追跡の能力はほどんど変わらない．
- 提案手法でしきい値 0.3 を用いた場合には，より長い期間追跡できるメソッドは増えるが，誤検出も増えてしまう．

次節では，より長い期間追跡できたメソッドの数をできるだけ減らさずに，誤検出のみを減らすための経験則について述べる．

2.2 経験則の導入

メソッドが誤って追跡されてしまうのは，あるコミットにおいて削除されたメソッドと追加されたメソッドの類似度が偶然しきい値以上になってしまうためである．よって，経験則は，そのような異なるメソッドについて，しきい値を超えないような工夫となる．本研究では，以下の 3 つの経験則を導入する．

経験則 1　メソッドのシグネチャが一致している場合やメソッド名が一致している場合のみ，低いしきい値を用いる．

経験則 2　括弧やセミコロン等の字句について，その種類を詳細に分ける．

経験則 3　メソッドに必ず含まれる字句を類似度の計測対象から外す．

以降，各経験則について詳細に述べる．

経験則 1

類似度を利用してメソッドを追跡するかどうかを判断するのは，そのメソッドのファイルと同名のファイルが存在しない場合である．メソッドのファイル名が変更されるのは，そのメソッドのシグネチャが変更された場合に加えて，そのメソッドを含むクラスのクラス名が変更された場合や，そのメソッドが他のクラスに移動した場合である．調査の結果，メソッド内が大きく変更されており，かつメソッドのファイル名が変わってしまっていても，そのメソッドのシグネチャやメソッド名は保持されていることが多かった．また，類似度が 0.3~0.6 の間で誤って追跡してしまったメソッドはその多くでシグネチャやメソッド名が異なるという結果であった．このことから，追跡のしきい値を以下の戦略で用いる経験則 1 を導入する．

- メソッドのシグネチャが同じ場合には低いしきい値（例えば 0.3）で追跡する．
- メソッドのシグネチャは異なるがメソッド名は同じ場合にはしきい値をやや高くして（例えば 0.45）追跡する．
- メソッドのシグネチャもメソッド名も異なる場合には Git デフォルトのしきい値（0.6）で追跡する．

表 1　予備実験の結果

手法 （しきい値）	Historage (0.3)	提案手法 (0.6)	提案手法 (0.3)
差異メソッド数	長 2　短 2	長 3　短 1	長 28　短 2
差異の正誤	正 3　誤 1	正 3　誤 1	正 6　誤 24

経験則 2

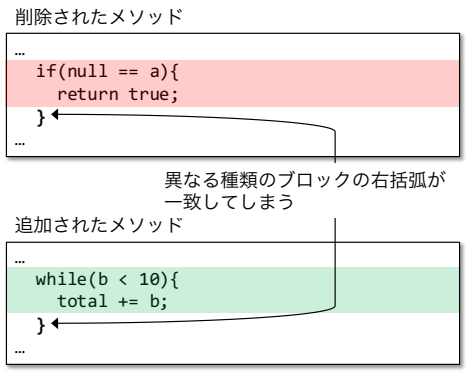
Java のソースコードには括弧やセミコロンが多数出現する．それらはさまざまなプログラム要素の一部として用いられている．例えば，中括弧（“`{`” と “`}`”）は，`if` ブロックや `for` ブロック等の範囲を表現する以外に，配列の初期化にも用いられる．そのため，各行に単一字句を配置する提案手法では，異なる役割を持つ括弧やセミコロンが偶然一致してしまい，類似度が不適切に高くなってしまう場合がある．そこで，経験則 2 として，字句の役割を表す文字列を各行に追記することを提案する．これにより，異なった役割を持つ括弧やセミコロンが偶然一致してしまうことを避けることができる．

図 4 は経験則 2 の有無による類似度の違いを表す例である．図 4(a) は，Historage の出力した Java メソッドのファイルを表している．対象コミットで削除されたメソッドには `null` チェックをするための `if` ブロックが含まれており，同じコミットで追加されたメソッドには変数 `b` の値を変数 `total` に加える `while` ブロックが含まれている．この `if` ブロックと `while` ブロックを含むメソッドは異なるため，追跡すべきではない．Historage の場合は，これらのブロックの最終行が偶然一致してしまい，ブロック部分の類似度は 1/3(=0.33) となる．提案手法を用いた場合（図 4(b)）は，`if` 文と `while` 文の丸括弧と中括弧，および `return` 文のセミコロンと式文のセミコロンが一致してしまうため，類似度は 5/12(=0.42) になる．一方，経験則を導入することで（図 4(c)）これらの丸括弧，中括弧，セミコロンは一致しなくなるため，類似度は 0/12(=0.00) となる．

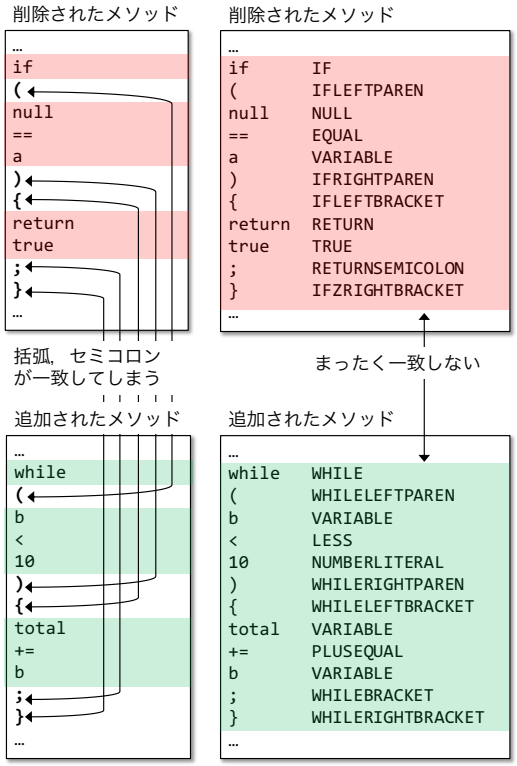
経験則 3

Java のメソッドには構文規則上，引数を囲むための丸括弧とメソッド本体を囲むための中括弧が存在する．つまり，メソッドの類似度を計測した場合，これら 4 つの字句は必ず一致する

(注4)：シグネチャが一致する場合のしきい値は 30，メソッド名が一致する場合のしきい値は 45，それ以外の場合のしきい値は 60.

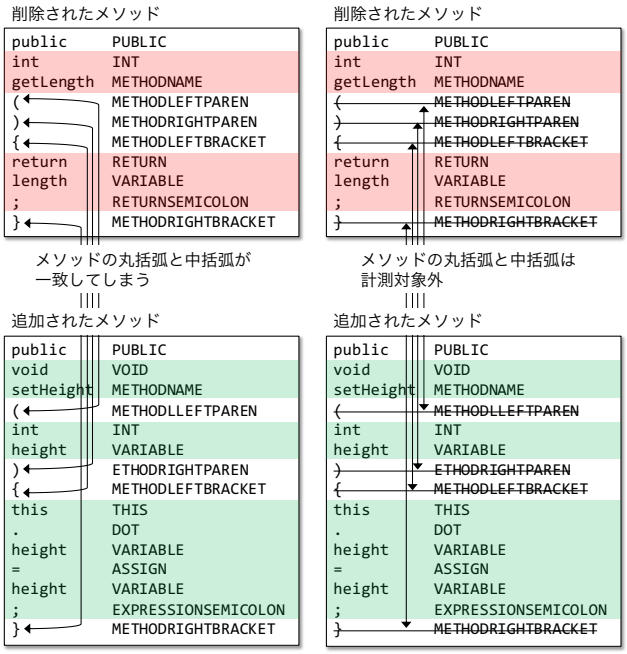


(a) Historage



(b) 提案手法（経験則 2 無し）　(c) 提案手法（経験則 2 有り）

図 4　経験則 2 の例



(a) 提案手法（経験則 3 無し）　(b) 提案手法（経験則 3 有り）

図 5　経験則 3 の例

ため，類似度を不適切に高めてしまう．メソッドが多数の字句を含む場合は，4 つの字句が一致してもその影響は少ないが，オブジェクト指向で多用されるセッターやゲッター等の小さなメソッドの場合は，4 つの字句の一致は影響が大きい．そこで本研究では，経験則 3 としてこれら 4 つの字句を類似度の計測対象から外す．これにより，メソッドの類似度が不適切に高くなってしまうことを抑制できる．

図 5 は，経験則 3 の有無の違いを表す例である．この例は，図 2 における変更前の `getLength` と変更後の `setHeight` 間の類似度を計測する例である．これらは別のメソッドであるため，類似度は低い方が望ましい．経験則 3 を用いない場合（図 5(a)）は，メソッドの丸括弧と中括弧が一致してしまうため，類似度は 5/10(=0.5) となってしまう．それに対して，経験則 3 を導入するとこれらが計測対象から外れるため，類似度は 1/6(=0.17) に下がる．

3. 実　　装

2. 節で述べた 3 つの経験則を含む提案手法をオープンソースソフトウェア **FinerGit**(注5) として実装した．FinerGit の入力は Java のソースコードが管理されている Git リポジトリであり，出力は各 Java メソッドがファイルに切り出された Git リポジトリである．

出力された Git リポジトリにおいて，各 Java メソッドは `.mjava` という拡張子を持つファイルである．これらのファイルに対して，“`git-log`” 等のコマンドを利用することにより，他の種類のファイルと同様に，そのファイルの変更履歴を取得できる．また，FinerGit では “`git-sv`” という新しいコマンドが用意されており，経験則 1 を利用したメソッドのシグネチャやメソッド名の同一性に基づいたファイルの追跡を行うことができる．

`.mjava` ファイルはファイル名にクラス名とメソッドシグネチャの情報を含むため，ファイル名が非常に長くなってしまう場合がある．これは，標準的に用いられている OS とは相性がよくない．例えば，Windows10 の場合では，ファイルの絶対パスの文字数が 260 を超えるとファイルマネージャーからアクセスできない等の問題が発生する．Linux や MacOS の場合はファイル名が 255 文字を超えることはできない．そのため，FinerGit では生成したファイル名が長くなる場合にはファイル

表 2　メソッド追跡の設定

設定名	行フォーマット	閾値	経験則 1	経験則 2	経験則 3
FG	単一字句	0.6	有(注6)	有	有
HIS30	元ファイルと同じ	0.3	無	無	無
HIS60	元ファイルと同じ	0.6	無	無	無

(注5)：`https://github.com/kusumotolab/FinerGit`
(注6)：シグネチャが同一の場合はしきい値 30，メソッド名のみが同一の場合はしきい値 45 とした．

名の後部を削除し，その代わりにファイル名全体から生成したハッシュ値を付与することにより，ファイル名が長くなりすぎるのを防ぐ．

4. 評　　価

FinerGit を用いて提案手法の評価を行った．メソッド単位でファイルを切り出すシステムとしては Historage [3] が存在するため，Historage との比較により，FinerGit を評価した．この実験では，FinerGit と Historage の細部の実装差が計測に与える影響を憂慮し，FinerGit に Historage と同様の機能を実装して，提案手法と比較した．表 2 は，この実験における FinerGit と Historage の設定を表している．

HIS30 と HIS60 は Historage を模した追跡であり，ファイルの各行は元ファイルと同じである．3 つの経験則は追跡に用いられない．

評価の対象として，GitHub に公開されている Eclipse プロジェクトのスター数が上位 10 件の Java プロジェクト(注7)を選んだ．表 3 は，対象プロジェクト名，コミット数，最新コミットにおけるメソッド数に加えて，3 つのうちの 2 つの追跡を比べた結果も表している．例えば，「FG 対 HIS60」は，この 2 つの設定におけるメソッドの追跡結果を比べた結果であり，「FG」の列は FG の方がより長く追跡できたメソッドの数を表し，「HIS60」は HIS60 の方が長く追跡できたメソッドの数を表している．この表の結果をまとめると以下のようになる．

- 「FG 対 HIS60」の列から，FinerGit を利用することにより，より長く追跡できたメソッドが多数存在していることがわかる．しかし，その一方で，Historage の方がより長く追跡できたメソッドも存在していた．
- 「HIS30 対 HIS60」の列から，Historage のしきい値を 0.6 から 0.3 に変更することで，より長く追跡できるメソッドが増えることがわかる．しかしながら，しきい値が 0.6 の場合の方が長く追跡できたメソッドも存在していた．
- 「FG 対 HIS30」の列から，FinerGit を利用することにより，Historage のしきい値として 0.3 を用いた場合よりも多くのメソッドをより長く追跡できた．しかし，eclipse-collections については，Historage の方が長く追跡できたメソッドが多かった．

また，追跡の精度についても評価を行った．精度については，比較的メソッド数の少ない `paho.mqtt.java` に対して，FG が HIS60 よりも長く追跡できた 40 のメソッドと，HIS30 が HIS60 よりも長く追跡できた 29 のメソッドについて，その追跡が正しいのかどうかを手作業により確認した．その結果，FG がより長く追跡した 40 のメソッドのうち 38 は正しい追跡であったこと，および HIS30 がより長く追跡できた 29 のメソッドのうち 27 は正しい追跡であったことがわかった．その結果，より長く追跡できたメソッドの精度は FG において 0.95（=38/40），HIS30 において 0.93（=27/29）であった．よって，Historage でしきい値として 0.3 を用いた場合と，FinerGit でしきい値 0.6 を用いた場合に，追跡の精度は同じであるが，提案手法がより長く追跡できたメソッドの方が多かった．

5. おわりに

本稿では，Java メソッドの開発履歴を高精度で追跡するための手法を提案した．提案手法は，FinerGit というツールとして実装され，現在すでに GitHub で公開されている．FinerGit と既存手法の Historage を比較したところ，FinerGit の方が追跡能力が優れていることが確認できた．今後は，FinerGit を利用して，ソフトウエアリポジトリマイニングの研究のレプリケーションを行い，精度の高いメソッドの追跡により，既存研究の実験結果が変わることがないかを調査する予定である．

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．

文　　献

[1] M. Codoban, S.S. Ragavan, D. Dig, and B. Bailey, “Software History Under the Lens: A Study on Why and How Developers Examine It,” Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution, pp.1–10, 2015.

[2] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, “Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository,” Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, pp.1277–1286, 2012.

[3] H. Hata, O. Mizuno, and T. Kikuno, “Historage: Fine-grained Version Control System for Java,” Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution, pp.96–100, 2011.

(注7)：2018 年 9 月 13 日時点のランキング

表 3　評価対象 Java プロジェクト

プロジェクト	コミット数	メソッド数	FG 対 HIS60		HIS30 対 HIS60		FG 対 HIS30	
(GitHub のプロジェクトフォルダ名)	(2018/9/13 時点)	(2018/9/13 時点)	FG	HIS60	HIS30	HIS60	FG	HIS30
`eclipse-vertx/vert.x`	3,652	5,520	950	53	570	35	502	76
`eclipse/che`	6,658	36,142	2,224	16	1,415	8	1,010	128
`eclipse/jetty.project`	16,063	12,236	1,783	33	1,093	31	895	107
`eclipse/openj9`	3,494	18,812	37	1	18	1	20	1
`eclipse/paho.mqtt.android`	195	195	8	8	8	1	2	9
`eclipse/eclipse-collections`	875	15,959	325	8	147	5	196	21
`eclipse/smarthome`	4,800	13,613	899	153	675	151	328	89
`eclipse/paho.mqtt.java`	509	1,523	40	0	29	0	17	4
`eclipse/jgit`	6,198	8,391	506	10	310	12	248	32
`eclipse/egit-github`	857	1,910	127	13	122	13	44	35