Vol.2022-SE-211 No.27
2022/7/29

Javaプログラムを対象としたソースコードの変更量と実行トレースの変化量間の相関調査

藤原 勇真[1]　神田 哲也[1]　嶋利 一真[2]　井上 克郎[3]

概要：ソフトウェア開発においてソースコードへ大小さまざまな規模の編集が行われている．ソースコードの大規模な編集はバグの埋込みと関連しているという報告がなされている．一方，ソースコードの変更量は大きいが振舞いが変更しない事例や，些細な変更が挙動を大きく変化させる事例も考えられ，ソースコードの変更量の大きさによってプログラムの挙動への影響の程度を予想できるかは不明である．本研究では，Java プロジェクトにおけるソースコードの変更量とプログラムの挙動の変化の関係について，プログラムの挙動の変化を実行トレースから得られる 4 種類のメトリクスの変化量によって観測し，分析を行った．その結果，ソースコードの変更量と実行トレースの変化量の間に相関関係があるケースはみられたものの，相関の強いメトリクスはプロジェクトにより様々であることがわかった．

キーワード：実行トレース，動的解析，プログラムの挙動

Investigating the Correlation between the Amount of Change in Source Code and the Amount of Change in Execution Trace for Java Programs

Yuma Fujiwara[1]　Tetsuya Kanda[1]　Kazumasa Shimari[2]　Katsuro Inoue[3]

Abstract: In software development, source code is edited on various scales, large and small. It is reported that large-scale editing tends to be related to bug embedding. However, we can consider the cases where the amount of code changes is large but the program behavior does not change and minor editing significantly change the program behavior. Therefore, the effect of the amount of change in the source code on the program behavior is uncertain. In this study, we investigated the relationship between the amount of change in the source code and program behavior in Java programs. The change in program behavior is observed from the changes in four metrics extracted from execution traces of the target programs. We found that the amount of change in the source code and the amount of change in the execution trace has a correlation in some cases, but metrics that have a strong correlation depend on the project.

Keywords: Execution trace, Dynamic analysis, Program behavior

1. はじめに

ソフトウェア開発において，バグ修正や機能の追加，性能の向上などの理由でソースコードへ大小さまざまな規模の編集が行われている．編集履歴を管理するため，バージョン管理システム (Version Control System: 以下，VCS) が広く利用されている．VCS を用いたソフトウェア開発では，ソースコードの変更箇所が確認できる．また変更が加えられたプログラムが仕様通りに動くか確認するため，ソフトウェアテストが広く実施されている．しかし，実行された命令列や変数の値など，プログラムの挙動の変化を確認することはできない．開発者の予期せぬ挙動の変化があった場合，障害が発生する可能性があり，ソフトウェア

[1] 大阪大学
Osaka University
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[3] 南山大学
Nanzan University

テストの結果のみで欠陥を発見することは困難である．また，大規模な編集は欠陥の埋込みと関連していると言われている [5] 一方で，ソースコードの変更量は大きいが振舞いが変更しない事例や，些細な変更が挙動を大きく変化させる事例もあり，ソースコードの変更量の大きさがプログラムの挙動へ与える影響は定かではない．これらの理由で，開発者はソースコードの変更量の大小に関わらず，プログラムの挙動の変化に注意する必要がある．しかし，開発者がプログラムの挙動の変化を全て確認することは困難なことである．仮にソースコードの変更量とプログラムの挙動の変化量間に関連性があれば，ソースコードの変更量からプログラムの挙動の変化量を予測することができ，開発者の負担を軽減できる．

そこで本研究では，ソースコードの変更量とプログラムの挙動の変化量を 5 つの Java プロジェクトに対して調査する．プログラムの挙動の変化量は，プログラムの実行された命令列である実行トレースのをもとに，変数の参照・代入回数，メソッドの実行回数，実行された観測命令の種類数を取得し，それらの差分を計算することで表現した．

調査の結果，ソースコードの変更量と実行トレースの変化量間には，必ずしも相関関係があるとは限らず，相関の強さも様々であった．また，プロダクトコードの変更量と実行トレースの変化量間，テストコードの変更量と実行トレースの変化量間にも，明確な関係性が見られなかった．以上のことから，ソースコードの変更量が必ずしも実行トレースの変化量に影響を及ぼす訳ではなく，影響の程度もプロジェクト毎で様々であるということを明らかにした．

以降 2 章では，研究背景や関連研究について述べる．3 章では本研究の調査内容や調査手法について述べ，4 章では得られた結果について述べる．5 章では，調査結果に対する考察，6 章では，妥当性の脅威について述べる．最後に 7 章では本研究のまとめと今後の課題を述べる．

2. 背景

2.1 バージョン管理システム

ソフトウェア開発において，機能の追加や欠陥修正，パフォーマンスの向上などを目的とし，ソースコードへ大小様々な規模の編集が行われている．これらの編集履歴を管理するため，今日のソフトウェア開発では VCS が広く利用されている．VCS とは，ファイルの変更履歴の保存，管理を行うソフトウェアである．ファイルの内容を，いつ誰がどのように編集したのか，時系列で記録に残すため，過去のあるバージョンを参照したり，過去のバージョンに戻したり，過去のバージョンとの差分を取ることが可能である．

代表的な VCS の 1 つである Git の場合，ファイルが変更された際に，ソースコードの編集箇所を開発者が容易に確認することができる．Git のリポジトリをホスティングできるサービスの 1 つである GitHub の場合，**図 1** のように，全体での変更されたファイルの数，追加された行数，削除された行数及び，各ファイルの変更箇所が確認できる．図 1 では，1 つのファイルが編集されており，3 行が追加され，2 行が削除されていることが分かる．変更箇所はハイライトされており，赤でハイライトされたソースコードが削除され，緑でハイライトされたソースコードが追加されたことが分かる．

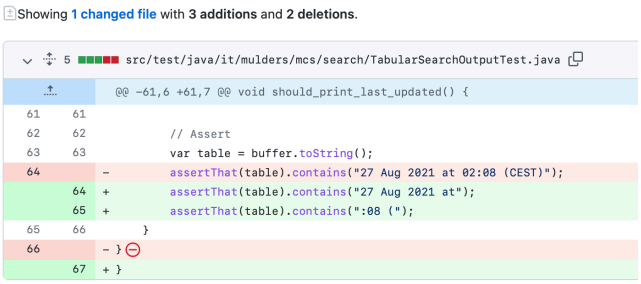

図 1: コミット前後のソースコードの変更箇所[*1]

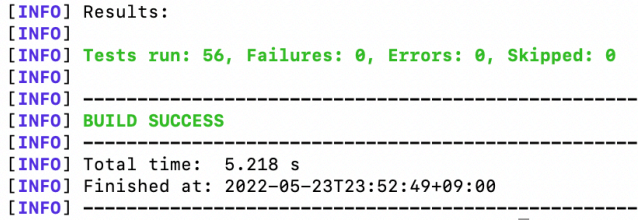

図 2: テストの結果

2.2 ソフトウェアテスト

ソフトウェアテストとは，開発したソフトウェアが設計通りに動作するかを評価及び検証するプロセスである．プログラムの仕様にない振る舞いまたは欠陥を検知することが目的である．Java で単体テストを実施するためのテスティングフレームワークの 1 つに JUnit がある．JUnit を用いてテストを実行した場合，**図 2** のようにテスト結果を確認することができる．今回の場合，56 個のテストケース全てにクリアしていることが分かる．このように開発者は単体テストを実行し，その結果を確認することでソフトウェアの動作を検証している．

しかし，ソフトウェアテストを用いて全ての欠陥を検知することは非常に難しい．

南らの研究によると，246 件の OSS プロジェクトのうち，自動テストにより 28 件のプロジェクトで 332 個の欠陥が発見されたが，そのうちの 7 件のプロジェクトで 17 個の欠陥が見逃されていた [13].

2.3 プログラムの挙動の変化

ソースコードへ変更が加えられた場合，プログラムの挙動が変化する．プログラムの挙動には，メソッド実行系列

[*1] https://github.com/mthmulders/mcs

や，変数に対する値の読み書きなどが挙げられる．開発者は，変更後のプログラムの挙動を確認するために，ソフトウェアを実行する [4]．開発者の予期せぬようにプログラムの挙動が変化した場合，障害が発生する場合がある．障害が発生した場合，開発者は欠陥を特定し修正する，デバッグを行う必要がある．

効率的なデバッグにおいて重要となるのが，プログラムの挙動の観測である [9]．プログラムの挙動の観測するにはいくつか方法がある．多くの開発者は，変数の値を出力する printf デバッグを有用な手段だと考え，利用している [2]．また，プログラムの挙動を観測するために，処理の状況を示すメッセージや変数の値等のデータをプログラムの外部に出力するロギング処理が広く用いられている [3]．プログラムの実行を任意の箇所で一時停止するブレークポイントデバッグも頻繁に利用されており [8]，変数の値やメソッドの呼び出し状態など，プログラムの挙動が観察されている．

また，ソースコードに対する変更がプログラムにどのような影響を与えるかを解析する，影響波及解析という手法がある [1]．この手法では，変更によって影響の受ける可能性のあるソースコードを特定する [10]．

しかし，ソースコードの変更量とプログラムの挙動の変化の関係性に関しては調査が行われていない．大きな修正はバグの埋め込みと関連があると言われている [5] 一方で，些細な変更がプログラムの挙動を大きく変化させる場合もあり，開発者はプログラムの挙動の変化に注意する必要がある．しかし，開発者がプログラムの挙動の変化を全て確認することは困難なことである．

2.4 Omniscient Debugging

Omniscient Debugging とは，プログラムの実行中のメモリ状態を時系列で完全に記録することで，網羅的に情報を収集してデバッグする手法の 1 つである [7]．この手法では，プログラムの実行開始から実行終了までに実行された全ての命令と，それによる値の変化を実行トレースとして記録する．従って，任意の時点でのプログラムの内部状態を計算機上で再現し，命令の実行順序や変数の値を観測することが可能である．

3. 調査

本研究では，Java プログラムを対象とし，ソースコードの変更量とテストケースに対する実行トレースの変化量との関連性について調査する．ソースコードの変更量と実行トレースの変化量間の関連性があるかどうかを調査することで，今後の実行トレースの変化に関するツール作成や議論の基盤になることが期待される．

表 1: 調査対象の Java プロジェクト

プロジェクト名	コミット数	テストケース数
commons-lang	6,787	8,072
commons-collections	3,712	16,922
commons-codec	2,234	1,336
commons-csv	1,789	437
commons-cli	1,161	438

3.1 調査対象

今回の調査では，JUnit のテストケースが用意された Apache の Java プロジェクト 5 つを調査対象とする．調査対象のプロジェクト名，コミット数，テストケースの数（2022 年 6 月 14 日 13 時時点）を表 1 に示す．

3.2 調査手法

調査手順を以下に示す．

STEP1. テストケースが用意されており，Java ファイルの編集があるコミットを抽出する．

STEP2. プロジェクトの状態を STEP1 で抽出した各コミットの状態に移す

STEP3. プログラム変更前後でのソースコードの変更量を求める．

STEP4. SELogger を有効にしてテストを実行し，プログラム変更前後の実行トレースの取得を行う．

STEP5. プログラム変更前後の実行トレースから，各メトリクスを取得する．

STEP6. STEP1 から STEP5 を繰り返し行う．

STEP7. ソースコードの変更量と実行トレースの変化量間の分析を行う．

STEP1 では，Git のコミット履歴を確認し，該当するコミットのコミット ID を記録する．設定ファイルの変更による影響は本研究での調査対象外のため，設定ファイルの変更を含むコミットは対象外とする．

STEP2 では，git checkout コマンドを使用し，手元にあるプロジェクトの状態を対象のコミットの状態に移す．

STEP3 では，git diff コマンドを用いて，プログラム変更前のコミット及びプログラム変更後のコミットから，ソースコードの変更量を取得する．ここでソースコードの変更量を以下の 2 種類定義する．

(1) 追加行数と削除行数の合計値

(2) git diff コマンドで表示される，各ブロックの追加行数または削除行数の最大値の和

1 つ目の定義について説明する．git diff コマンドを用い

て得られた差分から、変更前のコミットで削除された行数と，変更後のコミットで追加された行数を測定する．1つ目の定義ではこれらの和をソースコードの変更量と定義する．1つ目の定義では，ソースコードの編集・追加・削除といった特徴が把握できない．そこで，ソースコードの編集・追加・削除を考慮するため，2つ目のソースコードの変更量を定義した．ただし，双方ともソースフォルダ内にある Java ファイルのみを対象とする．また，空行やスペースのみの編集も除く．

2つ目の定義について説明する．git diff コマンドを用いて得られた差分から，各ブロックの追加行数または削除行数の最大値を測定する．2つ目の定義では，これらの和をソースコードの変更量と定義する．git diff コマンドの出力例を以下に示し，具体的に説明を行う．git diff コマンドを使用することで，編集されたソースコード箇所がブロック単位で表示される．出力例では，“@@” を用いて各ブロックが区切られている．追加された行は “+”，削除された行は “-” を用いて各ブロック毎に表示される．出力例のうち，1つ目のブロックでは，変数 a の型が int 型から double 型に変更されている．この変更を，“ソースコードが 2 行変更された” と解釈するのではなく，“ソースコードが 1 行編集された” と解釈する．このブロックでは，追加行数が 1 行，削除行数が 1 行であるため，追加行数または削除行数の最大値は 1 行となる．従ってこのブロックでは，ソースコードの変更量が 1 行であると解釈する．2 つ目のブロックでは，sum+=a というソースコードが 1 行追加されており，削除されたソースコードはない．追加行数又は削除行数の最大値は 1 となり，このブロックでのソースコードの変更量は 1 行であると解釈する．このように，各ブロックでソースコードの変更量を求め，各ブロックのソースコードの変更量の和を求める．今回の例の場合，各ブロックのソースコードの変更量は 1 行であるため，全体でのソースコードの変更量は 2 行となる．このように定義した 2 つ目のソースコードの変更量の求め方では，ソースコードの追加・削除・編集をある程度考慮し，ソースコードの変更量を求めることができる．

git diff の出力例

```
@@
 . . .
+  double a = 0
-  int a = 0
 . . .
@@
 . . .
+  sum += a
 . . .
```

STEP4 では，テスト実行時に Omniscient Debugging の実装の 1 つである SELogger[11] を利用し実行トレースの取得を行う．SELogger は，REMViewer の既存のトレースレコーダーを拡張したものである [12]．SELogger は，Java 仮想マシン内で Java エージェントとして動作する．プログラムのロードされたクラスにロギング命令を挿入し，実行トレースを取得する．このようにして，プログラムの実行開始から実行終了までに実行された全ての命令と，それによる値の変化を実行トレースとして記録する．プログラム変更前及び変更後でそれぞれテストを実行し，それぞれの実行トレースを取得する．ただし，ビルドが出来ないコミットは除く．

STEP5 では，STEP4 で得られたプログラム変更前後の 2 つの実行トレースからプログラムの挙動の変化に関連するメトリクスを測定する．メトリクスの詳細やその取得方法については，3.3 節で説明を行う．

STEP6 では，最新のコミットから過去のコミットに遡りながら，STEP1 から STEP5 を繰り返し行う．本研究では，メトリクスを計 50 回取得するまで繰り返し行った．

STEP7 では，ソースコードの変更量と実行トレースの変化量間の関連性について分析を行う．本研究では，関連性の評価に相関係数を用いる．まず，Java ファイル全体のソースコードの変更量と実行トレースの変化量間の関連性について調査する．収集した全コミットに対し，ソースコードの変更量と実行トレースの変化量間の相関係数を各プロジェクト毎に求める．また編集がプロダクトコードに対するものかテストコードに対するものかによって実行トレースの変化量に与える影響が異なる可能性を調査するため，収集したコミットを編集された Java ファイルの含まれるフォルダによって，プロダクトコードのみ編集されたコミット，テストコードのみ編集されたコミット，双方編集されたコミットに分類する．分類したそれぞれのコミットに対し，ソースコードの変更量と実行トレースの変化量間の相関係数を各プロジェクト毎に求める．

3.3 メトリクスの取得方法

本節では，3.2 節の STEP5 で述べた実行トレースからのメトリクスの取得について，詳細な説明を行う．本調査での実行トレースは，観測命令とその命令とその実行回数の組み合わせから成る．観測命令は，ソースコードにおける位置と実行されたバイトコード命令の種類の組み合わせから成る．

本研究では，この実行トレースから以下の 4 つのメトリクスを収集する．

1. 実行された観測命令の種類数の変化量

2. 変数の参照回数の変化量

3. 変数の代入回数の変化量

4. メソッド実行回数の変化量

以降，各メトリクスの説明とその取得方法について述べる．

3.3.1 実行された観測命令の種類数の変化量

観測命令は，ソースコードにおける位置と実行されたバイトコード命令の組み合わせから成る．本メトリクスは観測命令のうち，実行されたものの種類数のみを数える．つまり，ソースコードにおいて記述されていても，一度も実行されなかった命令については数えない．本メトリクスを用いることで，単純なソースコードの変更だけではなくその変更による実行への影響の有無に着目することが可能となるため，実行トレース全体の変化を定量的に把握することが可能となる．

3.3.2 変数の参照回数の変化量および変数の代入回数の変化量

本節では，変数の参照回数の変化量と，変数の代入回数の変化量の二つのメトリクスについて説明する．本メトリクスにおいて参照・代入関数の取得の対象とした変数は，局所変数，クラス変数，インスタンス変数のすべての種類の変数である．

取得の手順としては，プログラム内で宣言された変数に対して，全ての観測命令からその変数に対して参照が行われた観測命令を抽出し，その参照回数の取得を行う．参照された回数の取得は，観測命令において変数の参照が発生していた場合の観測命令の実行回数から算出する．この処理をプログラムにおいて実行されたすべての変数に対して実行し，各変数ごとに参照回数の差分を求める．そして，全変数から得られた参照回数の差分の絶対値を足し合わせることで，変数の参照回数の変化量を取得する．

変数の代入回数の変化量についても同様の変数を対象として，参照の代わりに代入が行われた命令とその回数を数えることで計測可能である．

3.3.3 メソッドの実行回数の変化量

本調査でのメソッド実行回数とは，調査対象プロジェクトのソースコード内のメソッドが呼び出された回数を記録する．したがって，外部ライブラリのメソッド呼出しの情報は記録していない．理由としては，プロジェクト内の実行の変化に着目するためで，頻繁に実行される外部ライブラリなどによる影響を避けるためである．

本メトリクスは，プログラム変更前後でのそれぞれのメソッド実行回数を記録しておき，メソッドごとに実行回数の変化量の絶対値を算出しその合計を数えることで計測する．

4. 結果

定義した 2 種類のソースコードの変更量と実行トレースの変化量間の相関係数を表 2 及び表 3 に示す．相関係数が 0.30 以上の場合に正の相関関係があると判断し，表中太字で示している．ソースコードの変更量と実行された観測命令の種類数の変化量については，3 つのプロジェクトで正の相関関係が見られたが，他の 2 つのプロジェクトでは相関関係が見られなかった．同様に他のメトリクスについても，ソースコードの変更量と強い正の相関関係があるプロジェクトや，全く相関関係がないプロジェクトがあるなど，全てのプロジェクトに一貫として相関関係のあるメトリクスは存在しなかった．このことからソースコードの変更量と実行トレースの変化量の間の相関の強さは様々であり，一概な関係性がないことが分かる．プロジェクト毎に実行トレースの変化量を見ると，commons-lang を除いて，変数の参照回数の変化量，代入回数の変化量，メソッドの実行回数の変化量の各相関係数は類似している．この 3 つのメトリクスは，ソースコードの変更量に対し類似した変化を示すことが分かる．

次に，各コミットにおけるソースコードの変更量をプロダクトコードの変更量，テストコードの変更量，双方の変更量として分類して行なった調査結果を示す．分類後の各コミット数を表 4 に示す．変更されたコードの種類ごとのソースコードの変更量と実行トレースの変化量間の相関係数を表 5 及び 6 に示す．変更されたコードの種類ごとのソースコードの変更量と実行トレースの変化量の相関関係についても様々であり，明確な関係性や傾向は見られない．ソースコードの種類を分類後についても，プロジェクト毎に実行トレースの変化量を見ると，commons-lang を除いて，変数の参照回数の変化量，代入回数の変化量，メソッドの実行回数の変化量の各相関係数は類似している．

以上のことから，ソースコードの変更量と実行トレースの変化量間には明確な関係性は見られず，ソースコードの変更量が必ずしも実行トレースの変化量に影響しないことが分かる．また，参照回数の変化，代入回数の変化，メソッド実行回数の変化は，ソースコードの変更量に対し類似した変化を示すことが分かる．

5. 考察

ソースコードの変更量と実行された観測命令の種類数の変化量間について，相関関係があると考えていたが，5 つのプロジェクトのうち 2 つのプロジェクトではほとんど相関なしという結果が得られた．このことから，変数の型変更やソースコードの移動など命令系列に影響のない変更も多々あることが考えられる．

ソースコードの変更量と実行トレースの変化量間について，一般的な関連性は見つからなかったが，特徴が見られるプロジェクトもあった．例えば，表 6 から commons-cli においてはプロダクトコードの変更量と実行トレースの変化量間に正の相関関係があることが分かる．この場合，プ

表 2: ソースコードの変更量 (1) と実行トレースの変化量の相関係数

プロジェクト名	実行された観測命令の種類数の変化量	変数の参照回数の変化量	変数の代入回数の変化量	メソッド実行回数の変化量
commons-lang	**0.577**	-0.011	**0.536**	0.042
commons-collections	0.025	**0.306**	**0.318**	**0.303**
commons-codec	**0.834**	**0.658**	**0.605**	**0.806**
commons-csv	**0.637**	0.214	0.094	0.087
commons-cli	0.110	-0.015	-0.014	0.01

表 3: ソースコードの変更量 (2) と実行トレースの変化量の相関係数

プロジェクト名	実行された観測命令の種類数の変化量	変数の参照回数の変化量	変数の代入回数の変化量	メソッド実行回数の変化量
commons-lang	**0.610**	0.173	**0.586**	-0.195
commons-collections	0.051	0.267	0.278	0.263
commons-codec	**0.812**	**0.765**	**0.722**	**0.763**
commons-csv	**0.672**	0.243	0.120	0.111
commons-cli	0.112	0.000	0.000	0.015

表 4: 編集ファイル分類後のコミット数

プロジェクト名	プロダクトコードのみの編集	テストコードのみの編集	双方の編集
commons-lang	23	15	12
commons-collections	13	21	16
commons-codec	24	16	10
commons-csv	12	25	13
commons-cli	22	10	18

ロダクトコードの変更量から実行トレースの変化量を予測できる可能性がある．このように，ソースコードの変更量と実行トレース間の関連性はプロジェクト依存であるが，関連性が見つかったプロジェクトについては以降の開発に活用できる可能性がある．

また表 5 及び表 6 から，プロダクトコードのみの変更量と実行トレースの変化量の相関関係と，テストコードのみの変更量と実行トレースの変化量の相関関係が大きく異なっていることが分かる．このことから，プロダクトコードのみを編集した場合とテストコードのみを編集した場合では，プログラムの挙動の変化が大きく異なることが分かる．

さらに，表 2 と表 3 を比較すると，表 3 における実行トレースの変化量との相関係数の値が，表 2 に比べて全体的に 1 に近づいていることが分かる．このことから，ソースコードの変更において，追加・削除・編集という情報が重要であることが考えられる．

6. 妥当性への脅威

調査したコミットの作業内容，具体的には完全な新規開発，機能追加，不具合修正などによって，実行トレースの変化量の傾向が変化する恐れがある．今回調査対象としたプロジェクトはいずれも長期にわたって開発が続けられている著名なプロジェクトであり，それらの各最新 50 コミットについて調査を行なったため，完全な新規開発は含まれていないなど，ソースコードの変更内容や実行トレースの変化に偏りがある可能性がある．

また，ソースコードの変更量を git diff コマンドを用いて取得しているため，コメントの編集量もソースコードの変更量として含まれている．コメントはプログラムの挙動には影響を与えないため，実行トレースの変化量との関係を考える上では考慮すべきではないものといえる．今回収集した全 250 コミットのうち，78 コミットにコメントの編集が含まれていた．コメントの編集量を除くことで，ソースコードの変更量と実行トレースの変化量間の関連性について変化が見られる可能性がある．

さらに実行トレースの変化量の取得において，プログラム全体で比較し変化量を求めているため，厳密な比較を行なっていない．例えば，実行された観測命令の種類数の変化量において，プログラム変更後にある観測命令の種類が 2 つ増え，別の観測命令の種類数が 2 つ減った場合，変化量は 0 となる．ファイル単位やメソッド単位など細かい比較を行うことで，実行トレースの変化がより観測され，ソースコードの変更量と実行トレースの変化量の相関関係に変化が見られる可能性がある．

本研究では，テストスイートに対するプログラムの挙動の変化を観測している．テストカバレッジが不十分だった場合，プログラムの挙動の変化を正しく取得できていない可能性がある．Singh らはテストコードのカバレッジを調査しており，テストが実施されている OSS プロジェクト 327 件の平均テストカバレッジは 41.96%であったと報告している [6]．今回の調査対象に関してテストコードのカバレッジ率は算出していないが，編集ファイルを分類した通りテストコードに対しても一定の編集が加えられていたことから，テストスイート自体は適宜保守されている場合に

表 5: 変更されたコードの種類ごとのソースコードの変更量 (1) と実行トレースの変化量の相関係数

変更されたコードの種類	プロジェクト名	実行された観測命令の種類数の変化量	変数の参照回数の変化量	変数の代入回数の変化量	メソッド実行回数の変化量
プロダクトコードのみ	commons-lang	-0.124	-0.084	-0.083	0.114
	commons-collections	**0.798**	-0.109	-0.321	-0.102
	commons-codec	0.289	0.122	-0.112	0.290
	commons-csv	**0.573**	-0.034	-0.016	-0.166
	commons-cli	0.287	0.225	0.192	0.270
テストコードのみ	commons-lang	**0.367**	-0.24	**0.671**	-0.337
	commons-collections	-0.011	**0.318**	**0.331**	**0.318**
	commons-codec	**0.812**	**0.973**	**0.953**	**0.976**
	commons-csv	0.069	0.043	0.033	0.043
	commons-cli	**0.984**	-0.102	-0.115	0.000
プロダクトコードとテストコード	commons-lang	**0.770**	-0.098	**0.510**	0.168
	commons-collections	-0.262	0.180	**0.663**	0.005
	commons-codec	**0.977**	**0.981**	**0.980**	0.290
	commons-csv	**0.583**	0.121	0.069	0.072
	commons-cli	0.019	-0.082	-0.082	-0.065

表 6: 変更されたコードの種類ごとのソースコードの変更量 (2) と実行トレースの変化量の相関係数

変更されたコードの種類	プロジェクト名	実行された観測命令の種類数の変化量	変数の参照回数の変化量	変数の代入回数の変化量	メソッド実行回数の変化量
プロダクトコードのみ	commons-lang	-0.037	0.078	-0.076	0.108
	commons-collections	**0.724**	-0.220	-0.348	-0.246
	commons-codec	**0.435**	0.237	-0.077	**0.435**
	commons-csv	**0.727**	-0.069	-0.200	-0.200
	commons-cli	**0.505**	**0.444**	**0.400**	**0.488**
テストコードのみ	commons-lang	**0.383**	-0.204	0.706	-0.302
	commons-collections	0.041	**0.339**	**0.350**	**0.388**
	commons-codec	**0.783**	**0.950**	**0.936**	**0.949**
	commons-csv	0.094	0.077	0.063	0.078
	commons-cli	**0.918**	-0.042	-0.045	0.060
プロダクトコードとテストコード	commons-lang	**0.787**	0.143	**0.551**	0.207
	commons-collections	-0.242	0.054	**0.594**	-0.124
	commons-codec	**0.992**	**0.994**	**0.991**	0.277
	commons-csv	**0.624**	0.151	0.113	0.115
	commons-cli	0.020	-0.065	-0.065	-0.050

おける調査結果になっていると考えている．

7. まとめと今後の課題

本研究では，ソースコードの変更量と実行トレースの変化量の関連性について調べるため，複数の Java プログラムに対して両者の関連について分析を行なった．相関係数を用いた評価では，ソースコードの変更量と実行トレースの変化量に明確な関係性がなく，ソースコードの変更量が必ずしも実行トレースの変化量に影響しないことを明らかにした．また，ソースコードの変更をプロダクトコードのみの変更，テストコードのみの変更，双方の変更に分類し，実行トレースの変化量との関連性について調査を行なった．結果としては，一概な関係性は見られなかったが，プロダクトコードのみ編集とテストコードのみの編集とでは，ソースコードの変更量と実行トレースの変化量間の関係性が全く異なることが分かった．

本研究では，ソースコードの変更量に着目をしたが，今後はソースコードの変更内容に着目し，どのような変更がプログラムの挙動に大きな影響を及ぼすのかなどの調査を進めていきたい．また，挙動の変化が大きい際に開発者へ挙動を確認するよう警告を行うツールや，プログラムの挙動の変化の可視化を行うツールの作成などが今後の課題として挙げられる．

謝辞

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

参考文献

[1] Arnold, R. S.: *Software change impact analysis*, IEEE Computer Society Press (1996).
[2] Beller, M., Spruit, N., Spinellis, D. and Zaidman, A.: On the dichotomy of debugging behavior among programmers, *Proceedings of the 40th International Conference on Software Engineering*, pp. 572–583 (2018).
[3] Hassani, M., Shang, W., Shihab, E. and Tsantalis, N.: Studying and detecting log-related issues, *Empirical Software Engineering*, Vol. 23, No. 6, pp. 3248–3280 (2018).
[4] Jiang, S., McMillan, C. and Santelices, R. A.: Do Programmers do Change Impact Analysis in Debugging?, *Empirical Software Engineering*, Vol. 22, pp. 631–669

(2017).
[5] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A. and Ubayashi, N.: A large-scale empirical study of just-in-time quality assurance, *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, pp. 757–773 (2013).
[6] Kochhar, P. S., Thung, F., Lo, D. and Lawall, J.: An Empirical Study on the Adequacy of Testing in Open Source Projects, *Proceedings of the 21st Asia-Pacific Software Engineering Conference*, APSEC 2014, pp. 215–222 (2014).
[7] Lewis, B.: Debugging Backwards in Time, *CoRR*, Vol. cs.SE/0310016 (2003).
[8] Murphy, G. C., Kersten, M. and Findlater, L.: How are Java software developers using the Eclipse IDE?, *IEEE Software*, Vol. 23, No. 4, pp. 76–83 (2006).
[9] Pothier, G., Tanter, E. and Piquer, J.: Scalable Omniscient Debugging (2007).
[10] Ren, X., Shah, F., Tip, F., Ryder, B. G. and Chesley, O.: Chianti: A Tool for Change Impact Analysis of Java Programs (2004).
[11] Shimari, K., Ishio, T., Kanda, T. and Inoue, K.: Near-omniscient debugging for java using size-limited execution trace, *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution*, ICSME 2019, pp. 398–401 (2019).
[12] 松村俊徳, 石尾隆, 鹿島悠, 井上克郎：REMViewer: 複数回実行された Java メソッドの実行経路可視化ツール, コンピュータ ソフトウェア, Vol. 32, No. 3, pp. 3_137–3_148 (2015).
[13] 南智孝：OSS 開発における欠陥数とカバレッジの関係に基づく継続的インテグレーションの有効性検証, 修士論文, 奈良先端科学技術大学院大学 (2017).