特別研究報告

題目

ソフトウェアの変遷に伴う SBFL 適合性の変遷調査

指導教員
肥後 芳樹 教授

報告者
濱本 凪

令和 7 年 2 月 6 日

大阪大学 基礎工学部 情報科学科

令和 6 年度 特別研究報告

ソフトウェアの変遷に伴う SBFL 適合性の変遷調査

濱本 凪

内容梗概

デバッグ支援技術の 1 つに欠陥限局技術がある．欠陥限局とは，プログラムに含まれる欠陥箇所を推測する技術である．その中でも Spectrum-Based Fault Localization(SBFL) は，各テストケースについて，結果の成否と，どの文が実行されたかという情報を用いて，プログラム中の欠陥箇所を推測する技術である．同じ機能を有するプログラムであっても，構造の違いによってプログラムのどの文が実行されるかが変化するため，SBFL の精度に差が生じる．

佐々木らの先行研究では，プログラムが SBFL にどの程度適しているかを表す SBFL 適合性と，SBFL 適合性を評価するための数値指標である SBFL スコアが提案されている．ミューテーションテストの技術を活用して SBFL を実行し，SBFL スコアを算出する．評価実験として，単一メソッドで構成される約 10〜20 行程度の Java クラスファイルを用いて，リファクタリングを題材としたプログラム構造の違いによる SBFL スコアの比較を行った．結果として，同一の条件分岐先で実行される文の数が少ないほど，SBFL スコアが向上した．SBFL スコアを用いた応用的な研究として，実用的なプログラムを対象とし，プロダクトコードやテストコードの機能の追加や変更が，SBFL スコアにどう影響を与えるかについての分析が考えられる．

そこで本研究では，あるプログラムに対する SBFL スコアが，プログラムの進化に伴ってどう変化するのかを分析する．対象プログラムは GitHub から選定し，規模については，単一メソッドから複数メソッドに拡張する．対象のプロダクトコードとそのテストコードについて，過去のコミットから変更情報を取得し，各変更ごとに SBFL スコアを計測する．得られた結果から，SBFL の変化の傾向や，プログラムの変更が SBFL スコアに与える影響を分析する．

SBFL スコアの長期的な変遷を分析した結果，最終的に SBFL スコアが上昇するプログラムが多く見られた．プロダクトコードとテストコードを比較した結果，テストコードの方が SBFL スコアに与える影響が大きいと判明した．さらに，SBFL スコアが変化した際のプログラムの変更内容を分析した結果，プロダクトコードにおける public メソッドの追加は

SBFL スコアを低下させる一方で，プロダクトコードにおける private メソッドの追加やテストコードにおけるテストケースの追加は，SBFL スコアを上昇させる傾向が見られた．

主な用語

欠陥限局
Spectrum-Based Fault Localization(SBFL)
ミューテーションテスト

目 次

1　まえがき

ソフトウェア開発において，デバッグは多くの労力やコストを必要とする重要な作業である．一般的な商用ソフトウェアの開発組織では，適切なデバッグやテストにおけるコストが開発費全体の 50〜70%に達する [1]．よって，デバッグ作業をいかに低コストに抑えられるかがソフトウェア開発において重要な課題となっている．

デバッグ作業の支援のために，多くの研究が行われている．デバッグ支援の研究分野の 1 つに欠陥限局技術についての研究がある．欠陥限局技術とは，欠陥が含まれている箇所を推測する技術である．近年では，プログラム中で実行される文の情報 (実行経路情報) を用いた欠陥限局 (Spectrum-Based Fault Localization，以降，SBFL) に関する研究がさかんに行われている [2][3]．SBFL は，テストケースが失敗した時に実行された文には欠陥が含まれる可能性が高いという考えにもとづき，テストケースの成否と実行経路情報を用いて欠陥を含む文を推測する [4]．対象プログラムの各文に対して欠陥を含む可能性が数値化され，実際に欠陥を含む文の数値が他に比べて高いほど，SBFL の結果がより正確であるといえる．

佐々木らの先行研究では，プログラムが SBFL にどの程度適しているかを表す SBFL 適合性と，SBFL 適合性を評価するための数値指標である SBFL スコアが提案されている [5]．また，Java ソースコードを対象として，ミューテーションテスト [6] を活用して SBFL スコアの計測を行っている．具体的には，まずすべてのテストケースを通過するプログラムに対して意図的に欠陥を発生させる．その後欠陥を含むプログラムに対して SBFL を実行し，どの程度正確に欠陥を特定できたかをもとに，元のプログラムの SBFL スコアを計測する．単一メソッドのリファクタリングを想定して SBFL スコアの変化を分析した結果，同一の条件分岐先で実行される文の数が少ないほど，SBFL スコアが向上する事例が確認された．この先行研究の応用として考えられるのが，実用的なプログラムへの機能の追加や変更，さらにテストコードの追加や変更などが SBFL スコアにどのような影響を与えるかについての調査である．

そこで本研究では，あるプログラムに対する SBFL スコアが，プログラムの進化に伴ってどう変化していくのかを分析する．より実用的なプログラムに対して実験するために，対象プログラムは GitHub から選定し，規模については，単一メソッドから複数メソッドに拡張する．対象のプロダクトコードとそのテストコードについて，過去のコミットから変更情報を取得し，各変更に対して SBFL スコアを計測する．取得した SBFL の変遷データをもとに，SBFL スコアがどのように変化していくのか，また大きな変化が発生するのはプログラムにどのような変更が行われた場合かについて分析する．

以降，第 2 章では本研究の背景として，SBFL スコアの定義や計測方法について述べる．第 3 章では本研究で提案する手法について述べる，第 4 章では実験の手順や調査内容，デー

タセットについて述べる．第 6 章では実験結果を踏まえた考察を述べる．第 7 章では本研究のまとめと今後の展望について述べる．

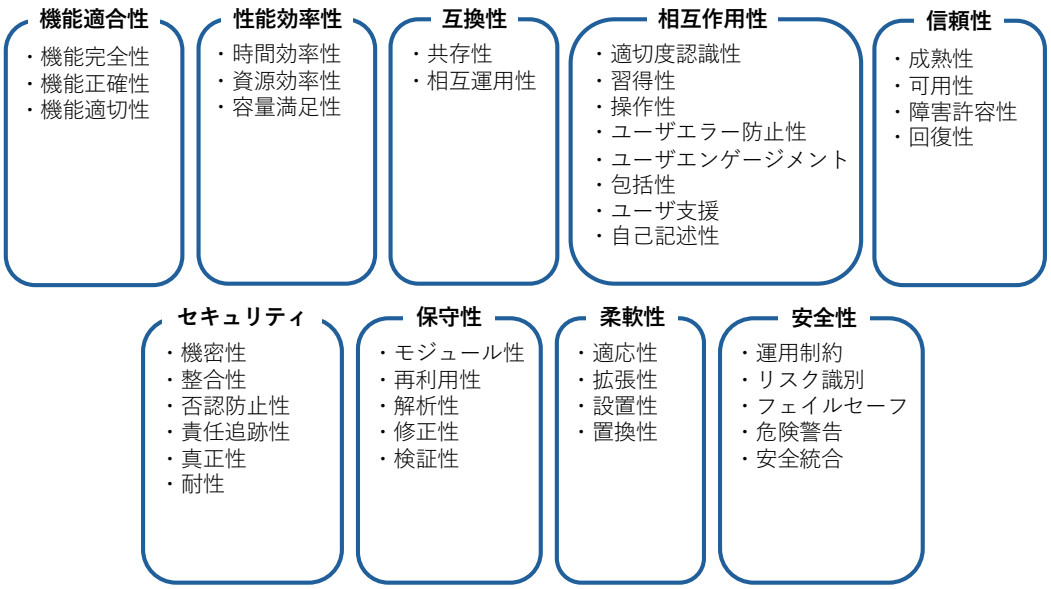

図 1: ソフトウェアの品質特性

2 背景

この章では，本研究の背景として，先行研究で用いられた SBFL スコアの定義や計測方法について述べる．

2.1 SBFL

SBFL(Spectrum-Based Fault Localization) は，テストケースの成否と実行経路情報を用いて，各行の疑惑値 (欠陥が含まれる可能性を示す値) を算出し，欠陥箇所を推測する．疑惑値の取りうる値は 0 以上 1 以下であり，疑惑値が高いほどその行に欠陥が含まれる可能性が高い．これまでに SBFL に関して様々な手法が提案されている [7][8]．佐々木らの先行研究では，プログラムが SBFL にどの程度適しているかを表す **SBFL 適合性**と，SBFL 適合性の 1 つの評価指標である **SBFL スコア**が提案されている [5]．SBFL 適合性と SBFL スコアの具体的な違いを以下で説明する．

- **SBFL 適合性**

  SBFL 適合性とは，ソフトウェアの品質特性として提案されている，SBFL がどれくらいうまく働くかという性質である．ソフトウェアには様々な品質特性があり，ISO/IEC 25010 では 9 つの品質特性，および各特性から派生する副特性が定義されている [9]．

具体的なソフトウェア品質特性とその副特性を図 1 に示す．“保守性” の副特性である “解析性” は，ソフトウェアの変更が与える影響の評価や，欠陥原因の診断，修正箇所の特定に対する有効性や効率の度合いを示す．

プログラムの機能やテスト内容が同じであっても，プログラムの構造の違いによってテストケースごとの実行経路情報が変わるため，SBFL の精度に違いが生じる場合がある．したがって，プログラム自体が SBFL の実行にどの程度適しているのかという特性を持っていると考えられる．このような考え方から，SBFL 適合性は品質特性の内の “保守性”，およびその副特性である “解析性” に含まれると見なすことができる．

- **SBFL スコア**

  SBFL スコアとは，SBFL 適合性を評価するための数値指標の 1 つである．SBFL スコアは疑惑値と同じく 0 以上 1 以下の値をとり，値が高いほど SBFL 適合性が高いといえる．

2.2 ミューテーションテスト

SBFL の計測には，ミューテーションテストの技術を活用する．ミューテーションテストとは，プログラムに意図的に欠陥を発生させ，テストコードの品質を評価するテスト手法である．欠陥が含まれるように変更が加えられたプログラムをミュータントと呼ぶ．近年，ミューテーションテストを用いて SBFL の精度向上に取り組む研究が多く行われている [10][11]．本来ミューテーションテストとは，テストスイートがミュータントをどの程度正確に欠陥として検出できるかを評価するが，本研究では，ミュータントに含まれる欠陥箇所を SBFL がどの程度正確に識別できるかを計測する．

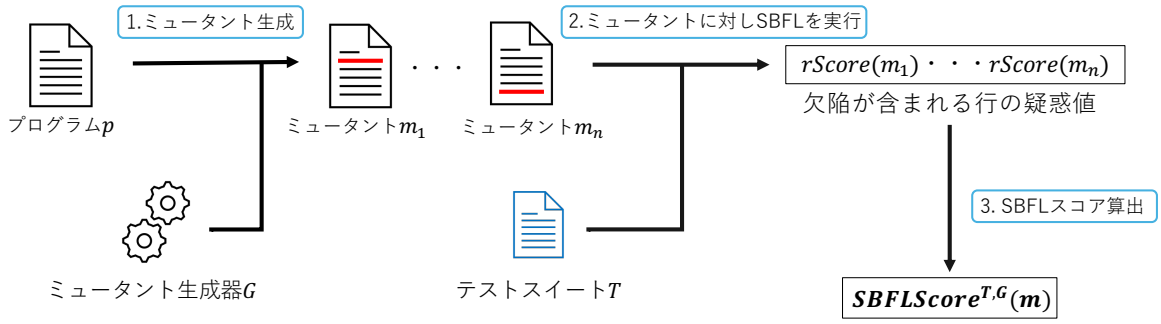

図 2: SBFL スコア計測の流れ

2.3 SBFL スコアの計測方法

SBFL スコアの計測方法を順を追って説明する．SBFL スコアの計測の流れを図 2 に示す．

2.3.1 ミュータントを生成

対象プログラム p に対してミュータント生成器 G を用いて複数のミュータントを生成する．この際，1 つのミュータントにつき発生させる欠陥は 1 つのみであり，出来る限り多くのミュータントを生成する．ミュータントの生成例を図 3 に示す．また，ミュータント生成器 G が発生させうる欠陥 (ミューテーション演算子) を表 1 に示す．なお，これらのミューテーション演算子は，オープンソースのミューテーションテストツールである PIT [12] を参考に実装されている．ここでプログラム p に対してミュータント生成器 G を用いて生成されたミュータントの集合を $M^G(p)$ と定義する．

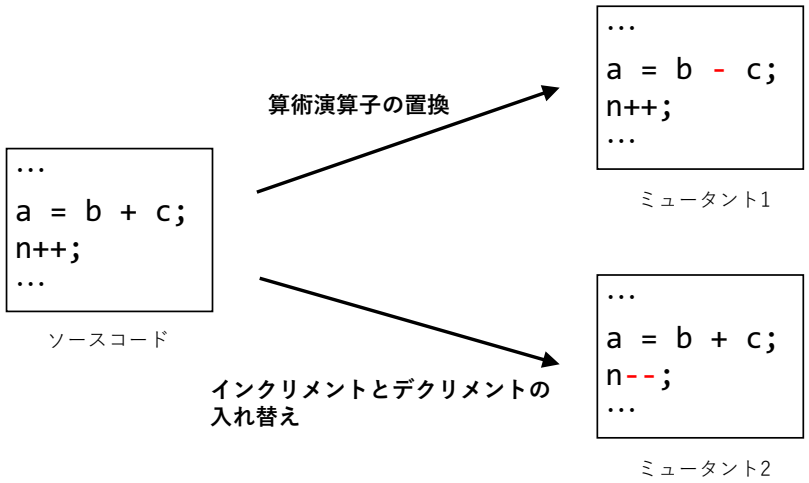

図 3: ミュータントの生成例

表 1: ミューテーション演算子

演算子	変更内容	変換例	
		変換前	変換後
Conditional Boundary	関係演算子の境界を変更	`a<b`	`a<=b`
Increments	インクリメントとデクリメントの入れ替え	`n++`	`n--`
Invert Negatives	負の数を正の数に置換	`-n`	`n`
Math	算術演算子を置換	`a+b`	`a-b`
Negate Conditionals	関係演算子を置換	`a==b`	`a!=b`
Void Method Calls	何の値も返さないメソッド呼び出しを削除	`method();`	`;`
Primitive Returns	プリミティブ型の戻り値を 0 に置換	`return 2;`	`return 0;`
Empty Returns	戻り値の型に応じて空値に置換	`return "str";`	`return "";`
False Returns	戻り値を false に置換	`return true;`	`return false;`
True Returns	戻り値を true に置換	`return false;`	`return true;`
Null Returns	戻り値を null に置換	`return object;`	`return null;`

2.3.2 各ミュータントに対して SBFL を実行

次に生成されたミュータントに対して，テストスイート T を用いて SBFL を実行する．具体的には各ミュータントに対して行ごとに疑惑値を算出し，その後各行に対して順位付けを行う．ここで $M^G(p)$ に含まれるミュータント m の各行 l について，l の疑惑値を $susp^T(l)$，l の疑惑値の順位を $rank^T(l)$，l の疑惑値順位を正規化した値を $rScore^T(l)$ と定義する．疑惑値の算出方法については，Ochiai の類似係数の計算式 [13] を用いる．Abreu らは疑惑値の計算に有効な 3 つの計算式を比較し，Ochiai の計算式が最も優れているという結果を得ている [14].

行 l に成功テストケースの数を $pass^T(l)$，行 l を実行した失敗テストケースの数を $fail^T(l)$，失敗したテストケースの総数を $totalFail^T$ とすると，Ochiai の計算式を用いた行 l の疑惑値は以下のように計算される．

$$susp^T(l) = \frac{fail^T(l)}{\sqrt{totalFail^T \times (fail^T(l) + pass^T(l))}}$$

次に行 l の疑惑値の順位 $rank^T(l)$ を計算する．順位は疑惑値の高い順に行を並べた時に最大で確認しなければならない行の数とする． 例えば 4 つの行 (l_1, l_2, l_3, l_4) から構成されるソースコードについて，

$$susp^T(l_1) = 0.7,\ susp^T(l_2) = 0.9,\ susp^T(l_3) = 0.7,\ susp^T(l_4) = 0.5$$

のとき，

$$rank^T(l_1) = 3,\ rank^T(l_2) = 1,\ rank^T(l_3) = 3,\ rank^T(l_4) = 4$$

となる．

さらに，求められた疑惑値順位 $susp^T(l)$ を，0 以上 1 以下の範囲で線形に正規化する．正規化を行う理由としては，2 つの異なるソースコードで同じ順位の行が存在しても，ソースコード内の総行数の違いによって，順位の価値が異なるからである．例えば同じ 5 位であっても，10 行の中での 5 位と 100 行の中での 5 位では，後者の方が価値が高いといえる．実行される文の総数を $totalStatements^T$ とすると，行 l の疑惑値の正規化順位 $rScore^T(l)$ は以下の式で表される．

$$rScore^T(l) = 1 - \frac{rank^T(l) - 1}{totalStatements^T - 1}$$

また，ミュータント m に含まれる欠陥の疑惑値の正規化順位を $rScore^T(m)$ と定義し，m の欠陥を含む行を ${l^m}_{fault}$ とすると，$rScore^T(m)$ は行 ${l^m}_{fault}$ の疑惑値の正規化順位である．

$$rScore^T(m) = rScore^T({l^m}_{fault})$$

2.3.3 SBFL スコアの算出

各ミュータントに対して $rScore^T(m)$ が算出できたら，最後に SBFL スコアの算出を行う．プログラム p に対して，テストスイート T とミュータント生成器 G を用いたときの SBFL スコアを $SBFLScore^{T,G}(p)$ とすると，$SBFLScore^{T,G}(p)$ は各ミュータントの $rScore^T$ の平均値となる．

$$SBFLScore^{T,G}(p) = \frac{1}{|M^G(p)|} \sum_{m \in M^G(p)} rScore^T(m)$$

2.4 先行研究からの発展的アイデア

ここでは，本研究のアイデアを述べる．

2.4.1 対象プログラムの進化に着目

先行研究では，対象プログラムのリファクタリングを題材として，リファクタリング前後の SBFL スコアの違いを評価した．本研究ではより実用的な実験として，対象プログラムの複数回の変更による SBFL スコアの変遷の評価を試みる．プログラムの変化に伴う SBFL スコアの変遷を把握すれば，SBFL 適合性が高くなるようなコーディングの知見を得られると考える．

2.4.2 対象プログラムの規模を拡大

先行研究では，単一メソッドで構成される約 10〜20 行の簡単な Java クラスファイルとテストファイルを作成し，実験の対象としている．本研究では実験対象を拡大し，Github 上に存在するプログラムに対して SBFL スコアの計測を試みる．既存プログラムに対してスコア計測を実行できれば，より実用的な場面での SBFL 適合性の評価に貢献できると考える．

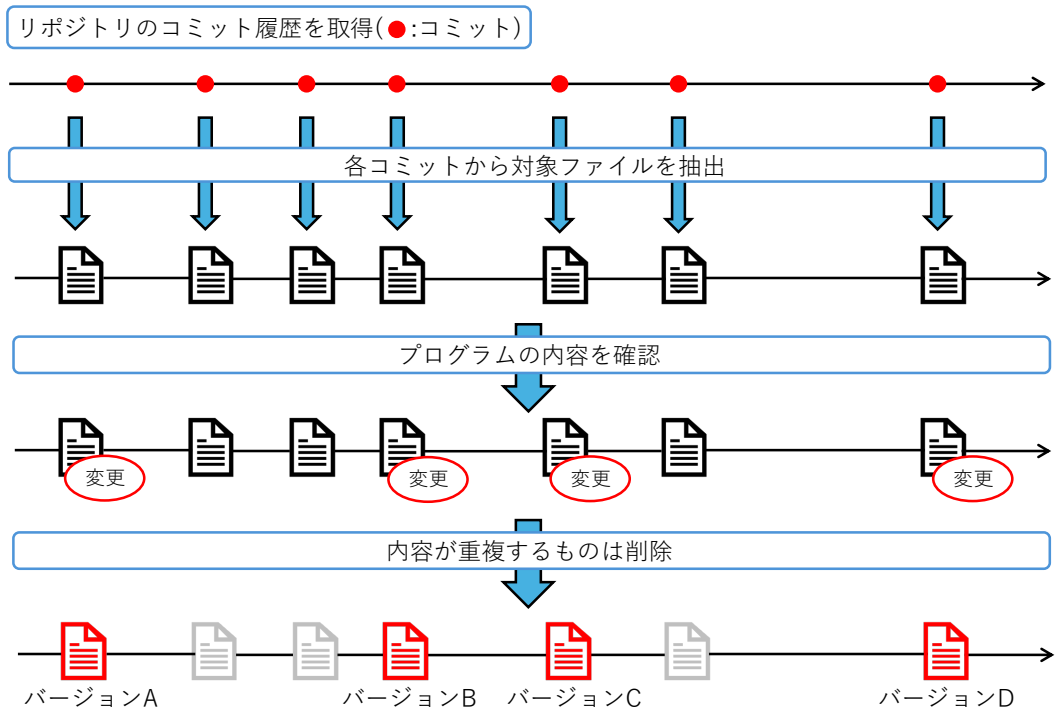

図 4: 対象プログラムの過去バージョン取得の流れ

3 提案手法

この章では，SBFL スコアの変遷を計測するための提案手法について述べる．これ以降，対象プログラムが更新されていくそれぞれの段階を**バージョン**と呼ぶ．

3.1 プログラムのバージョンの自動取得

本研究を行うにあたって，対象のプロダクトコードとテストコードの過去のバージョンを取得する必要がある．しかし，プログラムが何回も変更されている場合，変更されたプログラムを GitHub から手作業で 1 つずつ取得するのは非常に煩雑である．そこで，プログラムの過去のバージョンを GitHub から自動で取得するツールを JGit[18] で作成した．JGit とは，Git の基本的な機能 (リポジトリの作成，ブランチ管理，コミット操作，プッシュ，プルなど) を Java プログラム内で実行するために作られたライブラリである．対象プログラムの過去のバージョンを JGit を用いて取得する流れを説明する．また，過去のバージョンを取得する流れを図 4 に示す．

1. **コミット履歴の取得**

   対象リポジトリの過去のコミット ID を全て取得する．コミット ID とは，各コミットを識別するために割り振られる 40 桁のランダムな英数字列である．本研究で対象としたリポジトリに関しては，コミットは先頭 8 桁で区別可能であるため，先頭 8 桁を取得するようにツールを作成した．

2. **各コミットからプログラムを抽出**

   取得した各コミット ID から対象のプログラムのみを抽出して 1 つのディレクトリ内に格納する．JGit では コミット ID とファイル名の指定によって，特定のコミットでのプログラムを取得できる．

3. **同一内容プログラムの削除**

   コミット間で対象プログラムを比較していき，同一内容であれば削除する．ファイルの内容をハッシュに変換し，ハッシュの内容が一致するファイルが存在する場合は，1 つを残して残りは削除する．

以上の流れによって，対象プログラムの過去のバージョンを取得できる．

3.2 SBFL スコア変遷の計測

先行研究 [5] で用いられたツールは，1 つのプロダクトコードとテストコードの組に対して SBFL スコアを 1 回のみ計測するように作成されている．本研究では，プロダクトコードまたはテストコードに変更が加わるたびに SBFL スコアを計測していかなければならない．そこで，プロダクトコードとテストコードの過去のバージョンを入力として与えると SBFL スコアの変遷が出力されるようにツールを改変した．

なお，プロダクトコードとテストコードが同時に変更された場合は，どちらの変更がスコア変化の原因であるかを認識できない．プロダクトコードとテストコードのどちらかが先に変更されたとみなした場合でも，テストコードがプロダクトコードに対して，存在しないメソッドを呼び出すために，適切な計測結果が得られない可能性がある．よって，SBFL スコアの計測はそのまま実行するが，プログラムの変更が SBFL スコアに与える影響を分析する際には，同時変更によるスコアの変化は考慮しないという対処をとった．SBFL スコアの変遷を計測していく流れを図 5 に示す．図 5 の例では，C は同時変更であるため，A〜B，C〜D の SBFL スコアの変化は考慮するが，B〜C については考慮しない．

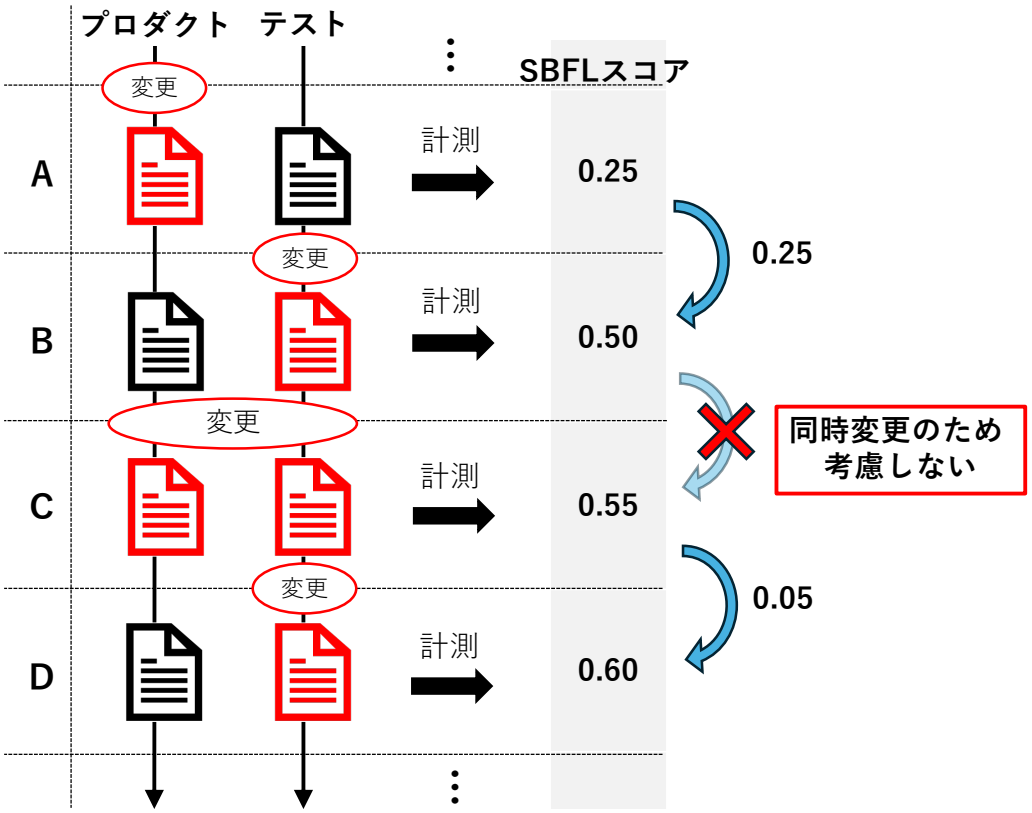

図 5: SBFL スコア変遷の計測

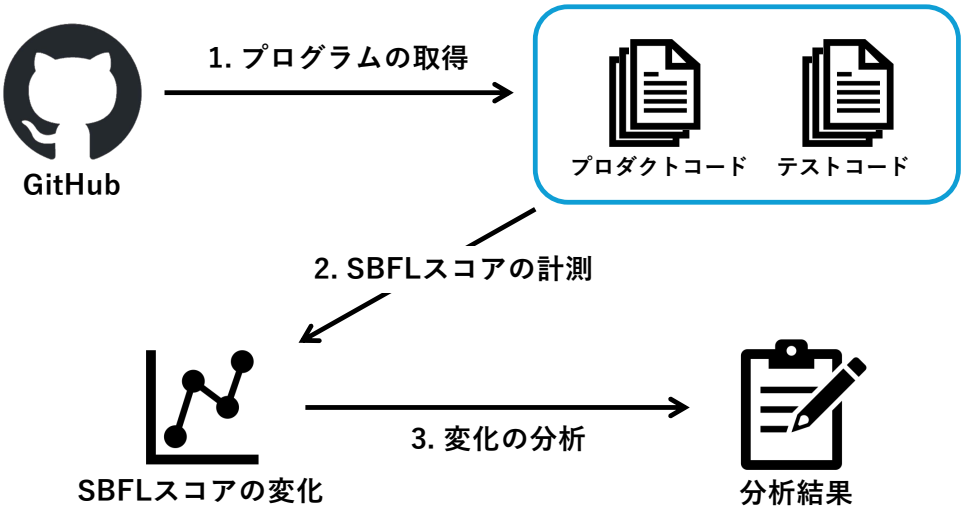

図 6: 実験の手順

4　実験手法

この章では，実験の手順や調査内容，対象とするデータセットについて述べる．

4.1　実験手順

具体的な実験の手順を説明する．まず，GitHub から実験対象のプロダクトコードとそのテストコードの過去バージョンを取得する．過去バージョンの取得方法については 3.2 節で詳しく説明している．次に取得したプロダクトコードとテストコードについて，SBFL スコア計測ツールを用いてスコアの変遷を計測する．最後に，得られた計測結果について分析，考察を行う．実験の手順を図 6 に示す．

4.2　調査内容

本研究で調査する内容は，大きく分けて以下の 2 つである．

調査 1：バージョン遷移による SBFL スコアの変化

初期のバージョンと最終のバージョンの SBFL スコアを比較して，スコアがどう変化しているか，またどの程度スコアが変化しているかを調査する．

調査 2：プログラムの各変更と SBFL スコアの関係

プロダクトコードまたはテストコードの各変更が SBFL スコアにどのような影響を与えるかを，以下の 3 つの観点で調査する．

- SBFL スコアの変化の傾向
- SBFL スコアの変化量
- プログラムの変更内容

4.3 データセット

実験対象として用いるプログラムについて説明する．本研究では先行研究 [5] と同様に Java クラスファイル (プロダクトコード) とそのテストファイル (テストコード) を対象とする．2.4.2 節でも説明しているが，先行研究では単一メソッドで構成される約 10〜20 行程度のクラスファイルとテストファイルを自作して計測実験を行った．これに対して本研究では規模を拡大させて，GitHub 上で実際に稼働しているプロジェクトからソースコードを選択する．データセット作成のために，Defects4J に含まれるプロジェクトを参考にした．Defects4J とは，ソフトウェアのバグ修正に関する研究を目的としたリポジトリである [15]．実際のオープンソースプロジェクトで発生した欠陥が収集されており，実用的な環境でテスト手法を評価できる．また，欠陥を修正したソースコードも含まれているため，本研究に適していると考えた．

Defects4J に含まれるプロジェクトの中から対象のプログラムを選定する際に，SBFL スコア計測の上での制約が発生した．

制約 1: 他クラスファイルを利用するクラスファイルについて計測が実行できない

SBFL スコア計測ツールの仕様上，単体で実行可能なファイルのみスコア計測が可能である．よってプロジェクト内の他のクラスファイルを利用しているクラスファイルは，SBFL スコアの計測ができない．そのため，他クラスを利用しない独立したクラスファイルのみを計測の対象とした．

制約 2: 新しいテスト環境で書かれたテストファイルで計測が実行できない

Defects4J に含まれるテストファイルは JUnit[17] というフレームワークを用いて記述されている． 現在は JUnit5 が最新であるが，本研究で用いる SBFL スコア計測ツールは，JUnit5 のリリースより前に作成された．よって JUnit4 以前のテストファイルにしか対応していないため，JUnit5 で記述されたテストファイルとそれに対応するクラスファイルは，対象外とした．なお，JUnit5 で記述されているテストファイルであっ

ても，JUnit5 独自の処理を用いていなければ，JUnit4 への書き換えによって実験対象とした．

以上の制約を踏まえて，Apache Commons というプロジェクトを対象にした．Apache Commons とは，アプリケーション開発の効率化を目的とした，Java の再利用可能なライブラリコレクションである [16]．本研究では 8 つのライブラリの 20 のプロダクトコードとテストコードの組について，SBFL スコアの計測を行った．実験対象のプロダクトコードを表 2 に示す．なお，対象ファイルの規模は最小で約 60 行，最大で約 1,500 行であり，テストコードも含めた変更回数は最小で 4 回，最大で 42 回である．

表 2: 実験対象のプロダクトコード

ライブラリ	ID	Java ファイル
Cli	1	DeprecatedAttributes.java
	2	Util.java
Collections	3	ArrayStack.java
Compress	4	BitInputStream.java
DBCP	5	PoolKey.java
IO	6	ByteOrderParser.java
	7	Charsets.java
Lang	8	ArrayFill.java
	9	BitField.java
	10	ClassPathUtils.java
	11	Conversion.java
Numbers	12	Digamma.java
	13	MultidimensionalCounter.java
	14	SmallPrimes.java
	15	SortlnPlace.java
	16	Trigamma.java
Text	17	CosineSimilarity.java
	18	FuzzyScore.java
	19	IntersectionResult.java
	20	LevenshteinResults.java

5　実験結果

この章では，実験の結果について，調査内容に分けて説明する．

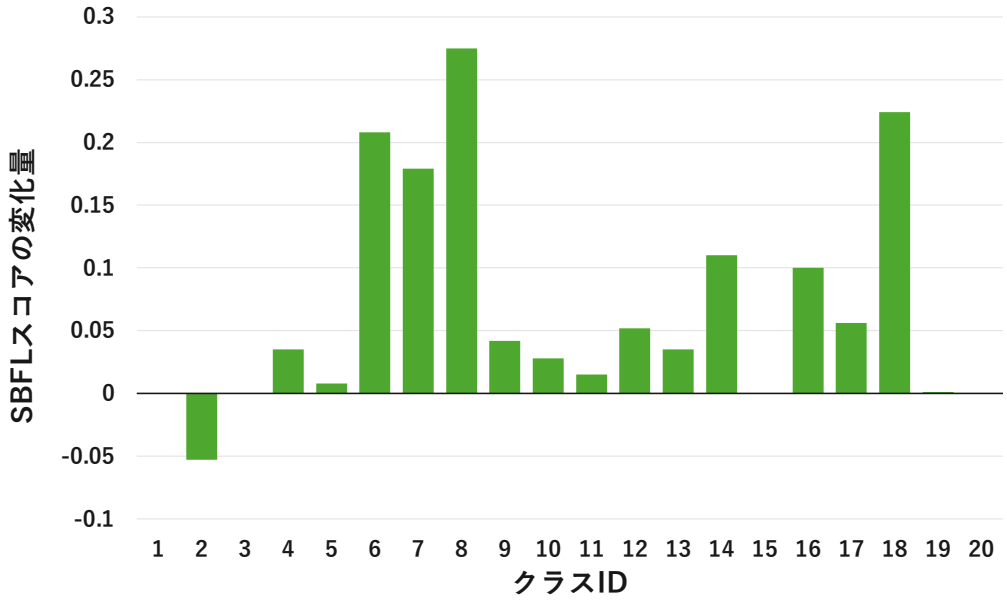

図 7: 初期スコアと比較した最終スコアの変化量

5.1　調査 1：バージョン遷移による SBFL スコアの変化

実験対象とした 20 個のプロダクトコードについて，SBFL スコアの変化量を表したグラフを図 7 に示す．また，各プロダクトコードの初期スコアと最終スコア，そしてスコアの変化量を表 3 に，初期スコアと比較した最終スコアの変化の分類を表 4 に示す．なお，SBFL スコアは小数点第 3 位を四捨五入して考える．

計測を行った 20 個のプロダクトコードのうち，最終の SBFL スコアが初期バージョンと比べて上昇したのは 15 個である．一方で，スコアが低下したのは Util のみである．SBFL スコア上昇の最大値は FuzzyScore の 0.224 となっており，0.3 以上の大幅なスコアの上昇は見られなかった．

表 3: 各プロダクトコードの SBFL スコアの変化

プロダクトコード	初期スコア	最終スコア	スコア変化量
DeprecatedAttributes.java	0.789	0.789	0
Util.java	0.669	0.616	-0.053
ArrayStack.java	0.619	0.619	0
BitInputStream.java	0.622	0.657	0.035
PoolKey.java	0.689	0.697	0.008
ByteOrderParser.java	0.792	1	0.208
Charsets.java	0.750	0.929	0.179
ArrayFill.java	0.705	0.98	0.275
BitField.java	0.913	0.955	0.042
ClassPathUtils.java	0.455	0.483	0.028
Conversion.java	0.965	0.980	0.015
Digamma.java	0.511	0.563	0.052
MultidimensionalCounter.java	0.605	0.640	0.035
SmallPrimes.java	0.592	0.702	0.110
SortlnPlace.java	0.328	0.328	0
Trigamma.java	0.419	0.519	0.100
CosineSimilarity.java	0.198	0.254	0.056
FuzzyScore.java	0.191	0.415	0.224
IntersectionResult.java	0.854	0.855	0.001
LevenshteinResults.java	0.548	0.548	0

表 4: 初期スコアと比較した最終スコアの変化

上昇	低下	変化なし
15	1	4

5.2　調査 2：プログラムの変更と SBFL スコアの関係

調査 2 の結果について，SBFL スコアの変化の傾向，SBFL スコアの変化量，プログラムの変更内容の 3 つの側面からそれぞれ分析する．

5.2.1　SBFL スコアの変化の傾向

計測を行った 20 個のプロダクトコードとそのテストコードについて，合計で 274 個の変更が見られた．そのうち 174 個がプロダクトコードの変更であり，100 個がテストコードの変更である．各変更が SBFL スコアをどう変化させるかについての分類を表 5 に示す．また，結果を割合としてまとめたグラフを図 8 に示す．プロダクトコードとテストコードのどちらに関しても，変更の約 75%は「変化なし」であり，SBFL スコアに影響を与えなかった．プロダクトコードについて，スコアが上昇した変更と低下した変更の個数に大きな差は見られない．対してテストコードについては，スコアが上昇した変更が低下した変更に比べて多い．

表 5: プログラム変更前と比較した変更後の SBFL スコア変化の分類

	上昇	低下	変化なし	計
プロダクトコードの変更	21	20	133	174
テストコードの変更	19	6	75	100

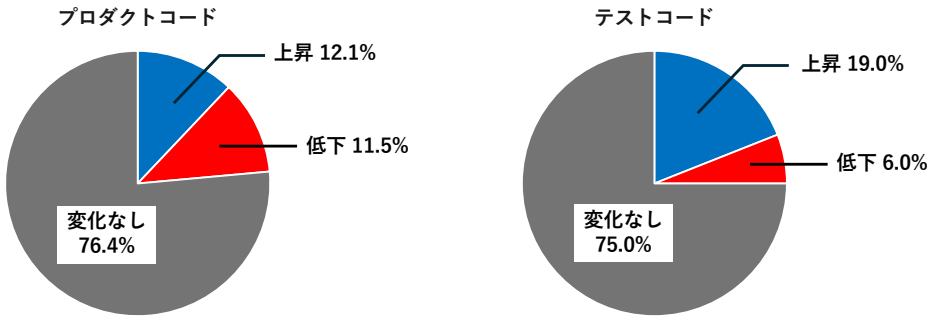

図 8: プログラム変更前と比較したプログラム変更後の SBFL スコア変化の割合

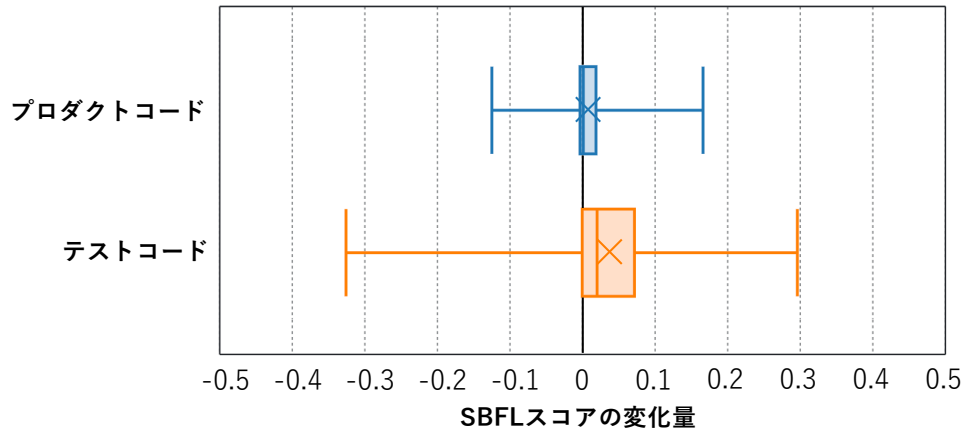

図 9: プログラム変更前と比較したプログラム変更後の SBFL スコアの変化量

5.2.2 SBFL スコアの変化量

SBFL スコアが変化した際のプログラムの変更について分析する．プログラムの変更によって SBFL スコアが変化した際のスコアの変化量について，箱ひげ図で表したグラフを図 9 に示す．グラフの横軸は SBFL スコアの変化量であり，各ボックスの × 印は平均値を表している．また箱ひげ図で表示されている各データを表 6 にまとめる．Q_1, Q_2, Q_3 はそれぞれ第一四分位数，第二四分位数 (中央値)，第三四分位数である．

プロダクトコードとテストコードを比較すると，テストコードの方が全体の範囲と四分位数範囲ともに大きく，よりばらつきがみられる．よって，プロダクトコードと比べてテストコードの方が，変更した際の SBFL スコアへの影響が大きいと考えられる．

表 6: SBFL スコアが変化した際のスコア変化量についての各データ

	最小値	Q_1	Q_2	Q_3	最大値	平均値
プロダクトコード	-0.125	-0.004	0.001	0.019	0.166	0.001
テストコード	-0.326	-0.001	0.020	0.072	0.296	0.04

5.2.3 プログラムの変更内容

SBFL スコアが変化した際に，対象プログラムにどのような変更がされているかを分類した．分類結果を表 7 に示す．プロダクトコードの変更については，プログラムの進化という要素の 1 つであるメソッドの増減に着目して分類した.「その他」には，軽微なリファクタリングや異常値の検出処理の追加が含まれる．テストコードの変更については，テストケースやアサーションの増減や変更に着目して分類した.「その他」には，軽微なリファクタリングが含まれる．また，図 9 の箱ひげ図に対する，変更内容についての分類を図 10，11 に示す.

- **プロダクトコード**

  public メソッドの追加は新しい処理機能の追加と考えられる．private メソッドの追加は，既存処理の新しいメソッドへの移動なので，広義的なリファクタリングと考えられる．表 7(a) から，2 種類のメソッド追加に関して，SBFL スコアが上昇する変更と低下する変更の両方が見られた．しかし図 10 を見ると，public メソッドの追加は負の範囲に，private メソッドは正の範囲に変化量が多く分布している．よって，private メソッドを追加する方が SBFL スコアを上昇させやすいと考えられる

- **テストコード**

  テストコードの変更で特に目立っているのはテストケースの追加である．表 7(b) から，SBFL スコアが上昇する変更の約半分がテストケースの追加であり，さらに，テストケースの追加によって SBFL スコアが低下する事例は見られなかった．図 11 を見ると，テストケースの追加が 0〜0.3 の範囲で広く分布している．よって，テストケースの追加が SBFL スコアを上昇させやすいと考えられる．

表 7: SBFL スコアが変化した際のプログラム変更内容の分類

(a) プロダクトコード

変更内容	上昇	低下
public メソッドの追加	2	3
private メソッドの追加	3	1
private メソッドの削減	0	1
その他	16	15

(b) テストコード

変更内容	上昇	低下
テストケースの追加	10	0
テストケースの削減	0	1
テストケースの内容変更	2	1
アサーションの追加	1	1
アサーションの削減	1	0
アサーションの内容変更	0	1
その他	5	2

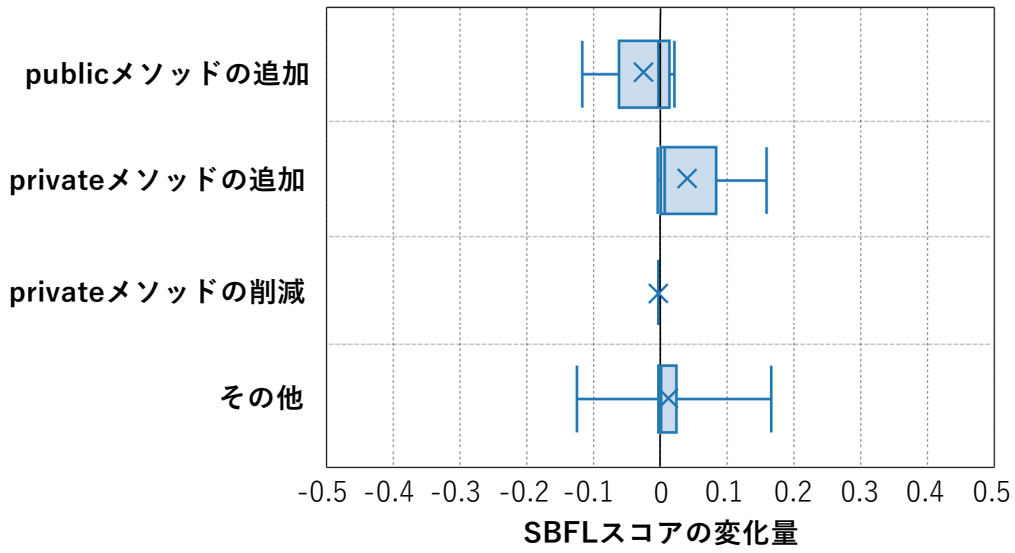

図 10: プロダクトコードの変更内容ごとの SBFL スコアの変化量

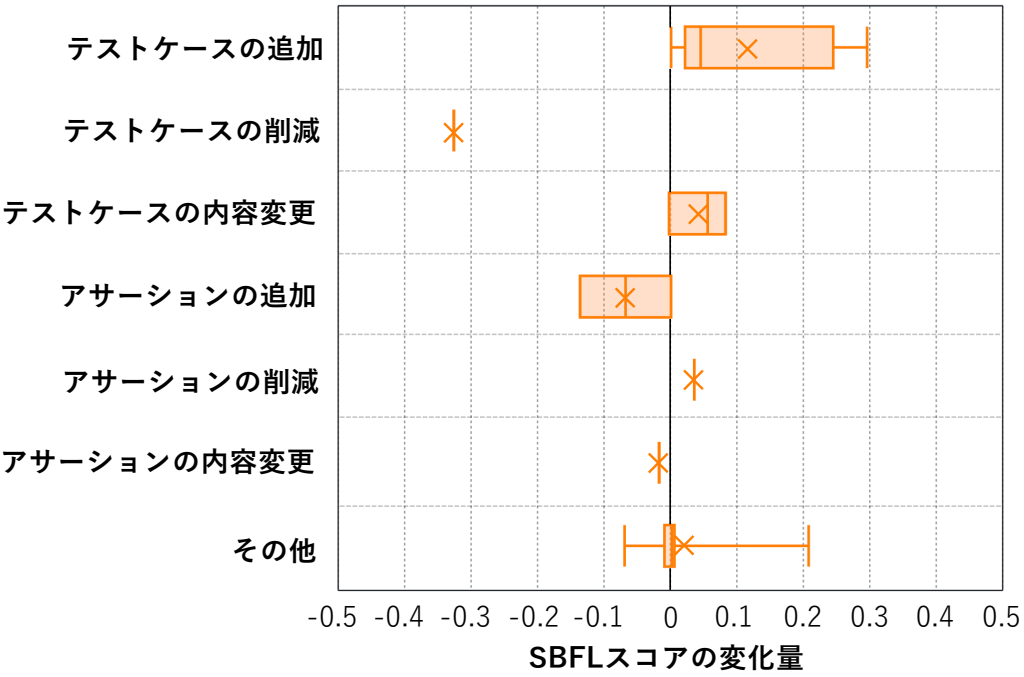

図 11: テストコードの変更内容ごとの SBFL スコアの変化量

6　考察

この章では，実験結果について順に考察する．

6.1　バージョン遷移と SBFL スコアの変化の関係

対象プログラムに対して，初期バージョンと比較した最終バージョンの SBFL スコアの変化を分析した結果，多くのプログラムについてスコアの上昇が確認された．この結果から，プログラムの長期的な進化によって，SBFL 適合性は向上されると考えられる．

加えて，最終バージョンの SBFL スコアに 0.3 以上の変化が見られないという結果も得られた．これは，各プログラムに SBFL スコアの上界が決まっているためであると考える．SBFL 適合性という品質特性がプログラム自体に備わっているため，プログラムの構造によって SBFL スコアの上界が決まっていると推測できる．よって，それ以上のスコアには上昇させられないと考えられる．

6.2　プログラムにおける各変更と SBFL スコアの変化の関係

対象プログラムにおけるプロダクトコードとテストコードの各変更に対する，SBFL スコアの変化との関係について分析した．プロダクトコードとテストコードの比較については，テストコードの方が SBFL スコアに大きな影響を与えやすいという結果が得られた．プロダクトコードの変更については，public メソッドの追加が SBFL スコアを低下させやすく，private メソッドの追加が SBFL スコアを上昇させやすいという結果が得られた．一方でテストコードの変更については，テストケースの追加が SBFL スコア上昇を上昇させるという結果が得られた．

これらの結果は，SBFL スコアがコードカバレッジに影響されるためであると考える．コードカバレッジとは，テストによって実際に実行されたコードの割合を表す指標である．SBFL に用いる実行経路情報はコードカバレッジと深く関係しており，欠陥を含む行を実行するテストケースが多いほど SBFL における疑惑値が高くなるため，コードカバレッジが高いほど疑惑値，および SBFL スコアも高くなる．public メソッドが追加される際，テストコードにおいて，追加された public メソッドを検証するテストケースが存在しないため，コードカバレッジは低下し，それによって SBFL スコアも低下すると考えられる．また private メソッドが追加される際，機能の変化は行われないため，コードカバレッジは大きく低下しない．よって SBFL は大きく低下しないと考えられる．テストケースが追加される際，プロダクトコードにおいて検証される部分が多くなり，コードカバレッジは上昇しうるので，SBFL スコアも上昇すると考えられる．

7　まとめと今後の課題

本研究では，SBFL スコア計測ツールを用いて，対象プログラムの進化に着目した SBFL スコアの変遷を分析した．分析を行った結果，プログラムの長期的な進化による SBFL スコアの上昇が確認できた．プログラムの変更内容の分類においては，プロダクトコードに比べてテストコードの方が SBFL スコアに与える影響が大きく，中でもテストケースの追加によるスコアの上昇が多く確認できた．プロダクトコードに関しては，public メソッドの追加による SBFL スコアの低下，private メソッドの追加による SBFL スコアの上昇が確認できた．

今後の課題として，以下の 3 点が挙げられる．

実験対象の追加

本研究では 20 のプロダクトコードとそのテストコードを対象としたが，より多くのプログラムに対する分析によって，正確な傾向を把握できると考える．

SBFL スコア計測ツールの改変

4.3 節で述べたように，SBFL スコアを計測する上で制約が発生したため，ツールの機能の拡張によって，計測可能なプログラムが増加すると考える．

メソッド数やテストケース数と SBFL スコアの関係性調査

本研究の結果から，メソッドの追加とテストケースの追加が特に SBFL スコアに影響を与えると判明した．メソッド数とテストケース数の着目によって，より詳細な SBFL スコアの性質を分析できると考える．

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肥後芳樹教授には，研究テーマの決定から研究活動に関するご指導，そして本論文の執筆に至るまで，様々な点から多くのご指導を賜りました．心より深く感謝申し上げます．

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 Kula Raula Gaikovina 教授には，中間報告会において，貴重なご意見を頂きました．心より深く感謝申し上げます．

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，研究活動に対して多くのご助言をいただき，中間報告会などでも貴重なご意見を頂きました．心より深く感謝申し上げます．

ノートルダム清心女子大学情報デザイン学部情報デザイン学科神田哲也准教授には，中間報告会などで貴重なご意見を頂きました．心より深く感謝申し上げます．

最後に，本研究に関して様々なご指導やご助言を賜りました，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肥後研究室の皆様に，心より深く感謝申し上げます．

参考文献

[1] Hailpern,B. and Santhanam,P.: Software Debugging,Testing,and Verification, *IBM Systems Journal*, Vol.41, No.1, pp.4–12, 2002.

[2] Yihao,Li. and Pan,Liu.: A Preliminary Investigation on the Performance of SBFL Techniques and Distance Metrics in Parallel Fault Localization, *IEEE TRANSACTIONS ON RELIABILITY*, Vol.71, No.2, pp.803-817, 2022.

[3] Calaghan,D. and Fischer,B. : Improving Spectrum-Based Localization of Multiple Faults by Iterative Test Suite Reduction, *ISSTA 2023: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp.1445-1457, 2023.

[4] Wong,W.E., Gao,R., Li,Y., Abreu,R. and Wotawa,F.: A Survey on Software Fault Localization, *IEEE TSE*, Vol.42, No.8, pp.707-740, 2016.

[5] 佐々木唯, 肥後芳樹, 松本真佑, 楠本真二 : プログラムに対する欠陥限局の適合性計測, 情報処理学会論文誌, Vol.62, No.4, pp.1029-1038, 2021.

[6] Jia,Y. and Harman,M. : An Analysis and Survey of the Development of Mutation Testing,*IEEE TSE*, Vol.37, No.5, pp.649–678, 2011.

[7] Jones,J.A., Harrold,M.J. and Stasko,J. : Visualization of Test Information to Assist Fault Localization, *Proc.ICSE*, pp.467–477, 2002.

[8] Dallmeier,V., Lindig,C. and Zeller,A. : Lightweight Defect Localization for Java, *Proc.ECOOP*, pp.528–550, 2005.

[9] ISO/IEC 25010. `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`. Accessed: Feb 2025

[10] Dutta,A., Srivastava,S.S., Godboley,S. and Mohapatra,D.P. : Combi-FL: Neural network and SBFL based fault localization using mutation analysis, *Journal of Computer Languages*, Vol.66, pp.101064, 2021.

[11] Z,Cui., M,Jia., X,Chen., L,Zheng. and X,Liu. : Improving Software Fault Localization by Combining Spectrum and Mutation, *IEEE Access*, Vol.8, pp.172296-172307, 2020.

[12] PIT. `https://pitest.org/`.Accessed: Feb 2025.

[13] da Silva Meyer,A., Garcia,A.A.F., de Souza,A.P. and de Souza Jr.,C.L. : Comparison of Similarity Coefficients Used for Cluster Analysis with Dominant Markers in Maize (Zea Mays L), *Genetics and Molecular Biology*, Vol.27, No.1, pp.83–91, 2004.

[14] Abreu,R., Zoeteweij,P. and van Gemund,A.J.C. : On the Accuracy of Spectrum-Based Fault Localization, *Proc.TAIC PART*, pp.89–98, 2007.

[15] Defects4J. `https://github.com/rjust/defects4j`. Accessed: Feb 2025

[16] Apache Commons. `https://commons.apache.org/`. Accessed: Feb 2025

[17] JUnit. `https://junit.org/junit5/`. Accessed: Feb 2025

[18] JGit. `https://www.eclipse.org/jgit/`. Accessed: Feb 2025