特別研究報告

題目

スペクトラムに基づく欠陥限局に対する Soft アサートの影響の調査

指導教員

楠本 真二 教授

報告者

三原 公平

令和 5 年 2 月 7 日

大阪大学 基礎工学部 情報科学科

令和 4 年度 特別研究報告

スペクトラムに基づく欠陥限局に対する Soft アサートの影響の調査

三原 公平

内容梗概

ソースコード中の欠陥を全自動で特定する方法として，Spectrum-Based Fault Localization（SBFL）がある．SBFL では，各テストの成否と実行経路情報を用いて欠陥箇所を特定する．この SBFL の欠陥限局精度を低下させる原因として Soft アサートが考えられる．Soft アサートはアサートの成否にかかわらずテストを最後まで実行するアサート文である．一般的なアサート文が失敗すると即座にテストが終了するが，Soft アサートではアサートが失敗した後もテスト実行を続けるため，欠陥箇所とは無関係なステートメントが失敗テストの実行経路に含まれてしまう．本研究では，この Soft アサート利用による実行経路の増加を，経路の汚染と考える．本研究では SBFL の精度向上を目的として，Soft アサートの利用実態，及び Soft アサートが SBFL の欠陥限局精度に与える影響について調査する．調査の結果，Soft アサートが実際に SBFL の精度を低下させることを確認した．

主な用語

欠陥限局，spectrum-based fault localization，SBFL，テスト，アサーション，Soft アサート，Hard アサート

目次

図目次

表目次

1　はじめに

デバッグを支援する技術の 1 つとして，スペクトラムに基づく欠陥限局手法（Spectrum-Based Fault Localization, SBFL）の研究が数多く行われている [1][2][3]．SBFL ではテストの実行経路を用いて，ソースコード中の欠陥の原因と疑わしき箇所を自動的に特定する．SBFL の基本的なアイデアは，失敗テストがよく通過する箇所ほど怪しく，逆に成功テストがよく通過する箇所ほど怪しくない，という考えである．SBFL は自動計測可能なテストのカバレッジ情報のみを利用するため，自動化が容易であり可用性に優れるという利点を持つ．開発者がデバッグを実施する際の補足情報としての活用 [4] だけでなく，自動デバッギングの実現 [5] や自動プログラム修正手法 [6][7] に対する前処理としての適用など，様々な活用が期待されている．

SBFL における欠陥限局の精度は，欠陥自体の性質やテストの内容，プロダクトコードの性質など様々な要因に左右される [8] が，本研究ではその要因の 1 つとして Soft アサートに着目する．多くのテストフレームワークでは，アサート文が期待値との比較に失敗すると，即座にテストを中断しそのテストが失敗したと判断する．それに対し Soft アサートでは，アサート文が失敗しても以降のテストの処理を継続する．そのため一度のテスト実行でテストの成否のみならず，全アサートの成否を一括で確認できる．具体的な活用シナリオとしては，単一テストに含まれる複数のアサートが互いに独立な関係にある場合が挙げられる．この場合，Soft アサートを利用することで，あるアサートのみが失敗したのか，それ以外のアサートも同時に失敗していたのかを把握できる．

Soft アサートの利用は失敗テストの経路，すなわちカバレッジの変化に繋がることから，SBFL の精度に影響することは明らかである．より具体的には，失敗テストの実行経路が広がってしまう点から，SBFL の精度低下に繋がると考えられる．あるアサートが失敗した時点で，そのテストの経路には欠陥の原因箇所がすでに含まれている．一般的なアサートではこの時点でテストを中断するが，Soft アサートにより中断しない場合，失敗テストの経路の拡大を引き起こす．本研究では，この Soft アサート利用による失敗テストの経路の広がりを「経路の汚染」と呼ぶ．経路の汚染により，欠陥とは無関係な箇所が SBFL によって欠陥原因箇所であると推測される可能性がある．

本研究では，Soft アサートが SBFL の欠陥限局精度に与える影響を調査する．まず予備調査として，次の 2 つの問いについて調査を実施する．

RQ1: Soft アサートをサポートするライブラリは何があるか?

RQ2: Soft アサートはどれだけ利用されているか?

もし Soft アサートが実際のテストコードにおいて頻繁に利用されている場合，Soft アサートによる SBFL の精度低下が多数のプロジェクトで発生する．このため，Soft アサートが実際のプロジェクトにおいてどれだけ利用されているかは，Soft アサートによる SBFL への影響を考察する上で重要な情報

となる．予備調査の結果，Soft アサートは多数のライブラリやフレームワークでサポートされていることを確認した．また，1,000 件のプロジェクトのうち，132 件のプロジェクトにおいて Soft アサートの利用を確認した．予備調査の結果に基づき，本研究の主となる以下の問いについて調べる．

RQ3: Soft アサートは SBFL の精度を低下させるか?

バグデータセットのテストコードにおいて，一般的なアサート文を用いた場合とアサート文を Soft アサートに書き換えた場合とで精度を比較した．比較の結果，35% の欠陥において精度の低下が見られた．また，実際のプロジェクトにおける欠陥についても同様の比較実験を行い，精度低下を確認した．

2　準備

2.1　スペクトラムに基づく欠陥限局（SBFL）

テストを用いた自動的な欠陥箇所特定技術の 1 つとして，スペクトラムに基づく欠陥限局（Spectrum-Based Fault Localization, SBFL）がある [1][2][3]．SBFL ではテスト実行時の通過ステートメントを一種のスペクトラムと見なし，そのスペクトラムの傾向を利用して欠陥の原因箇所を推測する．SBFL のキーアイデアは，失敗テストがよく通過するステートメントほど欠陥を含む可能性が高く，逆に成功したテストが多数通過するステートメントは欠陥を含みにくい，という考えである．なお，欠陥特定の対象となる箇所の単位はメソッドやステートメント，式など様々であるが，基本的にはカバレッジの計測単位がその単位となる．SBFL は単体テスト等の自動化可能な技術のみを利用するため，継続的インテグレーションをはじめとする自動化技術と親和性が高く，可用性に優れる．よって，開発者のデバッグを支援する技術の 1 つとして期待されている．

SBFL による具体的な欠陥箇所の特定方法について説明する．まず全てのテストを実行し，各テストの成否，及びその実行経路（スペクトラム）を記録する．次に，スペクトラムを利用して，疑惑値と呼ばれる「欠陥箇所であると疑われる可能性の高さ」を，全てのステートメントに対して算出する．この疑惑値の具体的な算出方法には複数の選択肢がある．ここでは Ochiai[9] と呼ばれる式を題材に説明する．ここで，$totalFails$ を失敗テストの総数，s を任意のステートメント，$fail(s)$ を s を通過した失敗テストの数，$pass(s)$ を s を通過した成功テストの数とすると，s の疑惑値 $susp(s)$ は以下の式で算出される．

$$susp(s) = \frac{fail(s)}{\sqrt{totalFails \times (fail(s) + pass(s))}}$$

また，Jaccard[9] では以下の式で算出される．

$$susp(s) = \frac{fail(s)}{totalFails + pass(s)}$$

この $susp(s)$ を全ての s に対して算出し，その値の高い s ほど欠陥の原因箇所であると推測する．よって，$susp(s)$ 自体の絶対的な値の高さではなく，他の s と比較したときの相対的な $susp(s)$ の高さが重要な情報となる．よって SBFL の精度を論じる際は，$susp(s)$ のランキングリストに対して実際に欠陥箇所であったステートメントが何位であったか，という相対的な指標が用いられることが一般的である．

なお，SBFL の疑惑値の算出においては失敗テストのスペクトラムが最も重要な要素となる．失敗テストがどのような経路を通過し，逆にどのような経路を通過しなかったかが，テストの失敗，すなわち

バグの原因箇所に対する強いヒントとなるためである．上記 Ochiai 式や Jaccard 式のいずれも，分子が $fail(s)$ であることからも，失敗テストのスペクトラムの重要さがうかがえる．

2.2 Soft アサート

単体テスト等のソースコードベースのテストは，テスト対象（プロダクト）の実行箇所，及び実行結果と期待値との比較箇所の 2 つから構成される．後者の期待値との比較は，一般的に `assert()` 等のアサート文を用いて実現される．Java の JUnit や Python の unittest など，ほぼ全てのテストフレームワークでは，アサート文がその期待値との比較に失敗すると，即座にテストプログラムを終了し，テストが失敗したと結論づける．それに対して Soft アサートでは，あるアサート文が期待値との比較に失敗しても，即座にテストを終了せず処理を続ける．本論文では通常のアサート文を Soft アサートと対比して Hard アサートと呼ぶこととする．

Soft アサートの利点の 1 つは，一度のテスト実行で全てのアサート結果が確認できる点にある．単純な Bean クラスの全フィールドの確認のような，アサート文の間に依存を持たない場合などが適したシナリオである．Hard アサートでは，あるアサートそれのみが失敗したのか，それ以外のアサートも実は期待値とは異なっていたのかは区別ができないのに対し，Soft アサートでは全アサートの結果を一括で確認できる．一方で，アサート文の間に強い依存を持つ場合は Hard アサートが適する．例えば，データベースとの疎通確認のような，本質的なテスト処理の前提となる部分で失敗した場合は，続くアサートが全て失敗することが自明であるため，Hard アサートが適する．

2.1 節で述べた SBFL と同様，Soft アサートは失敗テストに対して強い意味を持つ．テストが成功したときの挙動は Hard・Soft アサート共に全く同じであり，テストが失敗するときのみその挙動，すなわち実行経路に変化が現れる．よって，Soft アサートの利用により SBFL の結果に差が生まれると考えられる．

```
1  public class User {
     ...
2    public void setName(String name) {
3      this.name = name;
4    }
5    public void setPwd(String pwd) {
6      this.name = pwd;
7    }
8    public void login() {
9      if(isLoggedin()) {
         ...
```

(a) User クラス

```
1  @Test public void testUser() {
2    User u = new User("mihara", "qwerty123");
3    Soft.assert(u.getName()).isEqualTo("mihara");
4    Soft.assert(u.getPwd()).isEqualTo("qwerty123");

5    u.login();
6    Soft.assert(u.isLoggedin()).isTrue();
     ...
7  }
```

(b) Soft アサートを用いた単体テスト

図 1　Soft アサートによって SBFL の精度が低下する例

3　Motivating Example

Soft アサートの利用は SBFL の欠陥限局精度の低下に繋がる可能性がある．その理由は，Soft アサートを利用すると，欠陥箇所とは無関係なステートメントが失敗テストの実行経路に含まれてしまう可能性があるためである．本稿では，欠陥箇所と無関係なステートメントが実行経路に含まれることを実行経路の汚染と呼ぶ．

Soft アサートによって SBFL の精度が低下する例を図 1 に示す．この例は単一ユーザを表す User クラス（図 1a）と，User クラスに対する単体テスト（図 1b）で構成される．User クラスはユーザ名，パスワードなどのユーザ情報の登録機能及びログイン機能を持つ．単体テストでは，4 行目までででユーザ情報の登録機能の確認を行い，5–6 行目でログイン機能の確認を行っている．User クラスにはパスワードのセッター（図 1a の 6 行目）に欠陥があり，本来 `this.pwd` プロパティに `pwd` を代入すべきところを，`this.name` プロパティに代入している．これによって，図 1b のテストは 4 行目のアサート文が失敗する．

欠陥箇所は User クラスのパスワードのセッターにあり，テスト 5–6 行目で確認しているログイン機能は欠陥とは無関係である．しかし，Soft アサートではアサート文失敗後もテストを継続するため，テ

スト 5–6 行目も実行される．したがって，図 1a の `login` メソッドまでがテストの実行経路として含まれてしまう．このように，Soft アサートをアサート文として用いると，失敗テストで実行されるプロダクトコードが多くなり，失敗テストの実行経路が汚染されてしまう．

実行経路の汚染が発生すると，欠陥箇所とは無関係なステートメントの疑惑値が大きくなり，欠陥箇所の疑惑値の順位が相対的に低くなってしまう可能性がある．例えば図 1a の 9 行目は，Hard アサートを用いた場合実行されない．したがって 9 行目の疑惑値の順位は，欠陥箇所である 6 行目よりも低くなる．しかし，Soft アサートを用いた場合には 9 行目が実行され，疑惑値が大きくなる．これによって，9 行目の疑惑値の順位が欠陥箇所よりも高くなってしまう可能性がある．

このように，Soft アサートをアサート文に用いた場合，実行経路の汚染によって欠陥箇所の疑惑値の順位が低下し，SBFL の欠陥限局精度が低下してしまう可能性がある．

なお，アサート文が成功するとき，Soft アサートは Hard アサートと等価な振る舞いをする．したがって成功テストの実行経路は Soft アサートと Hard アサートで変化せず，SBFL の欠陥限局精度には影響を与えない．実行経路の汚染は失敗テストにのみ発生する現象である．

4 Research Questions

本研究では，Soft アサートが SBFL の欠陥限局精度に与える影響を調査する．まず予備調査として，Soft アサートの利用実態を調査する．Soft アサートが実際のテストコードで頻繁に利用されている場合，多数のプロジェクトやソフトウェアで SBFL の精度低下が発生してしまう．このため，Soft アサートが実際のテストにおいてどれだけ利用されているかは，Soft アサートによる SBFL への影響を考察する上で重要な情報となる．

予備調査として RQ1 と RQ2 を設定し，Soft アサートの利用実態を調査する．その後，本研究の主題である，Soft アサートが SBFL に与える影響について RQ3 を設定して調査する．

RQ1: Soft アサートをサポートするライブラリは何があるか?

Soft アサートの利用実態の 1 つとして，Soft アサートがどのようなライブラリやフレームワークでサポートされているか調査する．Java を主とした複数の言語において，多数のライブラリやフレームワークを確認し，Soft アサートをサポートしているか調査する．

RQ2: Soft アサートはどれだけ利用されているか?

RQ1 に引き続き Soft アサートの利用実態として，実際のプロジェクトにおいて Soft アサートがどれだけ利用されているか調査する．この調査では，RQ1 で発見した Soft アサートをサポートするライブラリやフレームワークの利用率を計測し，Soft アサートがどれだけ利用されているか調査する．

RQ3: Soft アサートは SBFL の精度を低下させるか?

テストにおいて，Hard アサートを用いた場合と Soft アサートを用いた場合で SBFL の精度が実際に変化するかどうか調査する．また，どのようなテストにおいて精度が低下するか考察する．

5 調査

5.1 RQ1: Soft アサートをサポートするライブラリは何があるか?

調査方法

Soft アサートの実現方法の把握を目的として，複数のアサーションライブラリやテストフレームワークについて調査を行った．調査の方法としては，ライブラリやフレームワークそれぞれのドキュメントを目視で確認し，Soft アサートを実現しているかどうか判定した．主たる調査対象の言語は Java としたが，それ以外にも Kotlin，JavaScript，Python 及び C#について調査した．

結果

調査結果を表 1 に示す．表中の FW はテストフレームワークを，Lib はアサーションライブラリを表す．Java プロジェクトで最も広く用いられているテストフレームワークである JUnit において，2 つのクラスで Soft アサートがサポートされていた．その他の Java テストフレームワークでは TestNG や Spock が，アサーションライブラリでは AssertJ や Truth が Soft アサートをサポートしていた．また，Java 以外のそれぞれの言語においても，Soft アサートをサポートするライブラリやフレームワークの存在が確認された．ここから，Java をはじめとした様々な言語において，Soft アサートには一定の需要があると考えられる．

表 1　Soft アサートをサポートするライブラリやフレームワーク

言語	ライブラリ名	種別	クラス名/メソッド名	使用方法
Java	JUnit [*a]	FW	ErrorCollector	`ErrorCollector.checkThat(…)`
			Assertions	`assertAll(()->assertEquals(…),()->assertEquals(…),…)`
	TestNG [*b]	FW	SoftAssert	`SoftAssert.assertEquals(…)`
	Spock [*c]	FW	Specification	`verifyAll{expr1,expr2,…}`
	AssertJ [*d]	Lib	SoftAssertions	`SoftAssertions.assertThat(…).isEqualTo(…)`
			JUnitSoftAssertions	`JUnitSoftAssertions.assertThat(…).isEqualTo(…)`
			AutoCloseableSoftAssertions	`AutoCloseableSoftAssertions.assertThat(…).isEqualTo(…)`
	Truth [*e]	Lib	Expect	`Expect.that(…).isEqualTo(…)`
Kotlin	Kotest [*f]	FW	assertSoftly	`assertSoftly{expr1,expr2,…}`
	Strikt [*g]	Lib	expect	`expect{that(…).isEqualTo(…),that(…).isEqualTo(…),…}`
	assertk [*h]	Lib	assertAll	`assertAll{assertThat(…).isEqualTo(…),…}`
JavaScript	Jasmine [*i]	FW	expect	`expect(…).toBe(…)`
	soft-assert [*j]	Lib	jsonAssertion	`jsonAssertion.softAssert(…)`
Python	softest [*k]	FW	soft_assert	`self.soft_assert(…)`
	assertpy [*l]	Lib	soft_assertions	`with soft_assertions():assertThat(…).is_equal_to(…)`
C#	Fluent Assertions [*m]	FW	AssertionScope	`using(new AssertionScope()){(…).Should().Be(…)}`
	NUnit [*n]	FW	Assert.Multiple	`Assert.Multiple(()=>{Assert.AreEqual(…);…})`

[*a] https://junit.org
[*b] https://testng.org
[*c] https://spockframework.org
[*d] https://joel-costigliola.github.io/assertj
[*e] https://truth.dev
[*f] https://kotest.io
[*g] https://strikt.io
[*h] https://github.com/willowtreeapps/assertk
[*i] https://jasmine.github.io
[*j] https://github.com/sarunya/soft-assert
[*k] https://pypi.org/project/softest
[*l] https://assertpy.github.io
[*m] https://fluentassertions.com
[*n] https://nunit.org

5.2 RQ2: Soft アサートはどれだけ利用されているか?

調査方法

Soft アサートが実際のプロジェクトにおいてどれだけ利用されているか調査するため，GitHub プロジェクトを対象に，Soft アサートの利用率を計測した．本調査では，Java を主言語とするプロジェクトのうち，スター数の上位 1,000 件を調査対象とした．調査の具体的な手法としては，対象プロジェクトそれぞれについて，Soft アサートに関するキーワードがソースコードに含まれているか検索し，検索ヒット数を確認した．検索キーワードには，RQ1 で発見した，Soft アサートを実現するクラス名やメソッド名を用いた．具体的には，“ErrorCollector”，“assertAll”，“SoftAssert” 及び “verifyAll” である．RQ1 では Soft アサートをサポートするライブラリとして Truth を発見したが，Truth に特有のキーワードが存在しなかったため，RQ2 では調査対象外とした．それぞれのキーワードをプロジェクト内で検索後，キーワードがヒットしたプロジェクト数を計測し，次の式によって Soft アサートの利用率を計算した．

$$\text{利用率} = \frac{\text{キーワードがヒットしたプロジェクト数}}{\text{全プロジェクト数（1,000 件）}} \times 100\%$$

結果

調査の結果として，Soft アサートの利用率を表 2 に示す．表中の “プロジェクト数” とは，Soft アサートに関するキーワードがヒットしたプロジェクト数を指す．Soft アサートをサポートしているライブラリ全体について利用率を計測すると，Soft アサートを利用しているプロジェクトは全体の 13.2% [*1]であり，10 件に 1 件以上のプロジェクトが Soft アサートを利用していることが分かる．

表 2　Soft アサートの利用率

キーワード	ライブラリ名	プロジェクト数	利用率 (%)
ErrorCollector	JUnit	27	2.7
AssertAll	JUnit	96	9.6
SoftAssert	AssertJ, TestNG	21	2.1
verifyAll	Spock	50	5.0

[*1] ライブラリそれぞれの利用率の総和は 19.4% であるが，複数のライブラリを利用するプロジェクトも存在するため，Soft アサートを利用しているユニークなプロジェクトは 13.2% となった．

5.3 RQ3: Soft アサートは SBFL の精度を低下させるか?

調査方法

Soft アサートによる SBFL の欠陥限局精度への影響を確認するため，Soft アサートと Hard アサートで精度の比較実験を 2 種類行った．2 種類の実験では，実験対象とするプロジェクトだけが異なる．実験の対象は以下の通りである．

実験 1 実バグのデータセットである Defects4J[10] に含まれる 66 件の欠陥.
　　全てのテストが Hard アサートで書かれている．
実験 2 実プロジェクトである Parse Server[*2]における 10 件の欠陥.
　　全てのテストが Soft アサートで書かれている．

調査した限りのバグデータセットには，Soft アサートで書かれたテストは少なく，実際に Soft アサートで書かれたテストを用いた欠陥を多数用意することは難しい．そのため，実験 1 ではバグデータセットを用いて多数の欠陥を対象に実験を行い，実験 2 で実際に Soft アサートを用いた欠陥に対して実験を行う．

具体的な実験の手順は以下の通りである．

操作 1 SBFL を適用し，精度を算出する
操作 2 失敗テストの全アサートを Soft/Hard アサートに書き換える
操作 3 再度 SBFL を適用し，精度を算出する

操作 1 では，まず SBFL を適用してプロダクトコードの各ステートメントについて疑惑値を算出する．疑惑値の算出結果から，SBFL の欠陥限局精度を算出する．SBFL の精度を評価する指標として EXAM[11] を用いる．EXAM は次の式で定義される．

$$\mathrm{EXAM} = \frac{\text{欠陥箇所の疑惑値の順位}}{\text{総ステートメント数}} \times 100\%$$

欠陥箇所の疑惑値の順位とは，SBFL により算出された疑惑値の値が高い順にステートメントを順位付けしたときの欠陥箇所の順位である．欠陥箇所の順位が高いほど EXAM の値は小さくなる．したがって，EXAM の値が小さいほど欠陥限局精度が高く，大きいほど精度が低い．EXAM の式において欠陥箇所の順位を総ステートメント数で割る理由は，欠陥を含むプロジェクトの総ステートメント数に対して順位を正規化し，プロジェクトの規模による精度の変化を除去するためである．

SBFL の精度を算出した後，2 つ目の操作として失敗テストのアサート文の書き換えを行う．実験 1

[*2] https://github.com/parse-community/parse-server

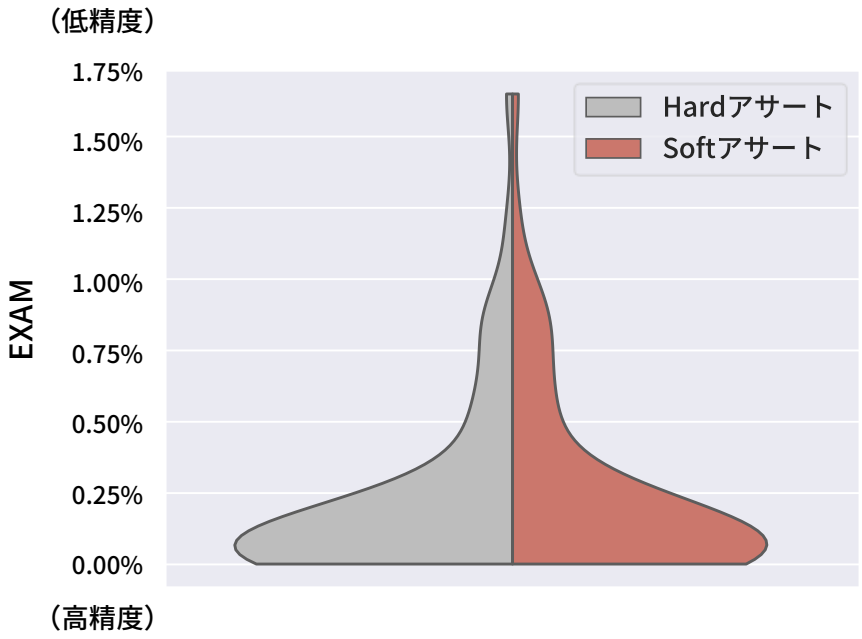

図 2　各アサートを用いた場合の SBFL の精度の分布

では Hard アサートを Soft アサートに，実験 2 では Soft アサートを Hard アサートに書き換える．なお，Soft アサートは失敗した場合のみ Hard アサートと異なる挙動をする．したがって書き換えるアサート文は失敗するテストのアサート文だけで十分である．

アサート文の書き換え後，3 つ目の操作として再度 SBFL を適用し，アサートを書き換えた場合の精度を算出した．精度の算出後，Soft アサートと Hard アサートの場合を比較し，精度が変化した理由について考察を行った．

実験 1 結果

Hard アサートと Soft アサートを用いた場合の EXAM の分布を図 2 に示す．図を分析すると，Soft アサートの分布が Hard アサートよりもわずかに低精度に偏っているが，アサート間で分布に大きな差はない．結論として，66 件の欠陥全体で見た場合，Soft アサートの利用による精度低下の影響は確認できなかった．

より詳細な分析として，欠陥ごとの精度の変化を確認する．精度が変化した欠陥の割合を図 3 に示す．図 3 を見ると，約 35%（23 件）の欠陥では精度が低下し，約 64%（42 件）の欠陥では精度が変化していない．精度が低下した 23 件の欠陥では，EXAM の値が平均で 0.08% 増加していた．これは，例えばプログラムが 1 万ステートメント存在するプロジェクトにおいて，欠陥箇所の順位が 8 個低下することを意味する．結論として，個々の欠陥の精度低下の程度は小さいが，精度低下が発生する欠陥数は多いことが分かる．

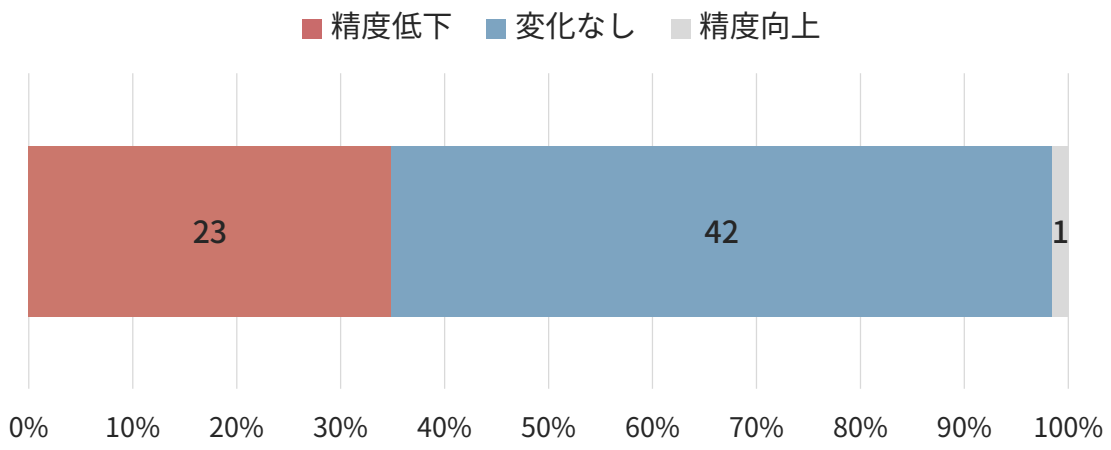

図 3　精度が変化した欠陥の割合

```
 1  public void testBasicFunctions() {
        ...
17      // emacs calc: [-4, 0, 3, 1, -6, 3] vn --> 6
18      double d_getLInfNorm = v5.getLInfNorm();
19      assertEquals("compare values  ", 6.0, d_getLInfNorm);
20
21      //octave =  sqrt(sumsq(v1-v2))
22      double dist = v1.getDistance(v2);
23      assertEquals("compare values  ",v1.subtract(v2).getNorm(), dist );
        ...
149     RealVector v_projection_3 = v1.projection(v2.getData());
150     double[] result_projection_3 = {1.662337662337662, 2.0779220779220777, 2.493506493506493};
151     assertClose("compare vect", v_projection_3.getData(), result_projection_3, normTolerance);
152 }
```

図 4　精度が低下した欠陥の失敗テスト

続いて，精度が変化した，あるいは変化しなかった原因について考察する．EXAM の値が 0.9% 低下した欠陥の失敗テスト[*3]を図 4 に示す．この欠陥において精度が低下した理由は，失敗テストの実行経路が強く汚染されたからである．図 4 のテストでは，失敗するアサート文の後に，欠陥とは無関係なステートメントの呼び出しが多数存在する．アサート文を全て Soft アサートに書き換えると，これらのステートメントが全て実行され，テストの実行経路が汚染される．その結果，欠陥とは無関係なステートメントの疑惑値の順位が上昇し，欠陥箇所の順位が相対的に低下した．

次に，精度が変化しなかった原因について考察する．図 5 に精度が変化しなかった欠陥の失敗テスト[*4]を示す．このテストでは，失敗するアサートがテストの最後にあり，その後新しいステートメントの呼び出しが無い．したがって，アサート文を Soft アサートに書き換えてもテストの実行経路は変化

[*3] https://github.com/Spirals-Team/defects4j-repair/blob/a4bc4d8a76926dcce78bd428acf719b3e7569aad/src/test/java/org/apache/commons/math/linear/ArrayRealVectorTest.java#L1080

[*4] https://github.com/Spirals-Team/defects4j-repair/blob/e3462056e30ba121dacbc9d4b94813e20a927a7e/src/test/java/org/apache/commons/math3/linear/RectangularCholeskyDecompositionTest.java#L76

```
 1  @Test public void testMath789() {
        ...
31      RealMatrix root3 = new RectangularCholeskyDecomposition(m3, 1.0e-10).getRootMatrix();
32      RealMatrix rebuiltM3 = root3.multiply(root3.transpose());
33      Assert.assertEquals(0.0, m3.subtract(rebuiltM3).getNorm(), 1.0e-16);
34  }
```

図 5　精度が変化しなかった欠陥の失敗テスト

せず，SBFL の精度に影響を与えない．このように，Soft アサートによって実行経路の汚染が発生しない欠陥も存在する．実験 1 では，精度が変化しなかった欠陥のうち，実行経路の汚染が発生していない欠陥は 38 件であった．

また，実行経路が汚染されているが SBFL の精度が低下しなかった欠陥も 4 件存在する．この 4 件の欠陥で SBFL の精度が低下しなかった理由は，失敗するアサートの後に呼び出されるステートメントが成功テストによって頻繁に実行されていたためであった．SBFL では，成功テストに実行されている回数が多いステートメントは，失敗テストによる実行回数が増加しても疑惑値の上昇値は小さくなる．実行経路が汚染されているにもかかわらず精度が変化しなかった欠陥では，このようなステートメントが多く，欠陥箇所の順位に影響を与えるには至らなかったと考えられる．

Soft アサートの利用は SBFL の精度の低下に繋がるか，あるいは変化させないかのいずれかであると考えていたが，実験 1 では 1 件の欠陥において精度向上を確認した．精度向上の理由としては，複数の失敗テストの存在が考えられる．複数のテストが失敗するとき，失敗テストのアサートを Soft アサートに変更すると，欠陥箇所を実行する失敗テスト数が増え，結果的に EXAM の値が小さくなる可能性がある．例えば Hard アサートを用いたテストにおいて，欠陥箇所が失敗テスト t1 によってのみ実行され，失敗テスト t2 では欠陥箇所を実行する前にテストが失敗していたとする．このとき，アサートを Soft アサートに変換すると，t2 でも欠陥箇所が実行され，欠陥箇所を実行する失敗テスト数が増える．欠陥箇所を実行する失敗テスト数が増えると，欠陥箇所の疑惑値は大きくなり，欠陥箇所の疑惑値の順位が上がる．このように，複数の失敗テストが存在するとき，Soft アサートは SBFL の欠陥限局精度を向上させる可能性がある．しかし本調査で精度が向上した欠陥を確認すると，失敗するテストは 1 件だけであった．この欠陥で精度が向上した原因は現時点で不明であり，この原因の調査は今後の課題の一つである．

実験 2 結果

実験 2 における EXAM の変化を表 3 に示す．実験 2 においても，EXAM の値が大きくなる，すなわち精度が低下する欠陥が 2 つ存在した．また，7 件の欠陥では精度が変化せず，1 件の欠陥では精度が向上した．この結果より，実際に Soft アサートで書かれたテストを用いる欠陥においても，Soft アサートによる精度低下が存在することが分かる．

実験 1 と同様に，精度が低下した原因及び精度が変化しなかった原因についてテストコードの分析を行った．分析の結果，精度低下が見られた欠陥のテストでは，失敗するアサート文の後に，欠陥とは無関係なステートメントの呼び出しが多数存在した．また，精度が低下しなかった欠陥のテストでは，テストケースの最後のアサートが失敗していた．これより，実際に Soft アサートを利用する欠陥においても，Soft アサートによって実行経路の汚染が発生した場合に精度が低下することが確認できた．

表 3 に示す通り，Parse-Server-bug5 では精度が向上している．この欠陥で精度が向上した原因について調査を行ったところ，精度向上の原因は，実験対象としたプロジェクト特有の条件にある可能性が高いことが分かった．実験対象のプロジェクトでは MongoDB が使用されており，テストがアサート失敗によって強制終了する場合，異常終了時におけるデータベースの処理が行われる可能性が考えられる．この処理は Soft アサート使用時には実行されないため，Soft アサートを用いると，Hard アサート時よりも実行されるステートメント数が減少する．実行されるステートメント数が減少すると，欠陥箇所の疑惑値の順位が相対的に上がり，精度が向上する．このため，この欠陥では精度が向上したと考えられる．

表 3　実験 2 における EXAM の変化

欠陥 ID	Hard アサート時の EXAM	Soft アサート時の EXAM	EXAM の変化
Parse-Server-bug1	7.5771	7.5771	0
Parse-Server-bug2	0.0073	0.0145	0.0073
Parse-Server-bug3	0.0721	0.0721	0
Parse-Server-bug4	0.0651	0.0651	0
Parse-Server-bug5	16.7129	16.2650	-0.4479
Parse-Server-bug6	3.4974	3.4974	0
Parse-Server-bug7	0.0969	0.0969	0
Parse-Server-bug8	4.0458	4.3246	0.2788
Parse-Server-bug9	17.0528	17.0528	0
Parse-Server-bug10	2.8014	2.8014	0

6　妥当性への脅威

5 章で行った調査について，妥当性への脅威が存在する．まず内的妥当性への脅威として，RQ2 の調査において実際のプログラムのコードを確認していないことが挙げられる．RQ2 では，Soft アサートに関するキーワードが検索で見つかったプロジェクトを，Soft アサートを利用しているプロジェクトとしている．実際にプログラムのコードを確認して Soft アサートが利用されているか判断する必要がある．外的妥当性への脅威としては，SBFL の疑惑値の計算手法として Ochiai のみを用いたことが挙げられる．他の疑惑値計算手法を用いた場合に，Soft アサートが SBFL の精度に影響を与えるか確認する必要がある．

7　おわりに

本研究では，SBFL の欠陥限局精度向上を目的として Soft アサートを調査した．予備調査として Soft アサートの利用実態を調査した．調査の結果，複数の言語において Soft アサートをサポートするライブラリの存在を確認した．また，Java の 1,000 件のプロジェクトのうち，137 件で Soft アサートの利用を確認した．次に，実バグのデータセット Defects4J を対象に，テストにおいて Hard アサートを用いた場合と Soft アサートを用いた場合で SBFL の精度を比較した．その結果，35% の欠陥において Soft アサートによる精度低下を確認した．また，実際に Soft アサートを利用するプロジェクトである Parse Server を対象に SBFL の精度比較実験を行った．その結果，実際に Soft アサートを利用するプロジェクトでも精度低下が発生することを確認した．

本研究では，実験 2 において，実際に Soft アサートを用いているプロジェクトで，Hard アサートを用いた場合と比較して精度が低下しているか調査した．しかし，実験 2 で対象とした欠陥数は 10 件と少なく，より多くの欠陥に対しての実験が必要であると考えている．このため，実験 2 における対象の欠陥数の増加が今後の課題である．

謝辞

本研究はたくさんの方に助けられ，遂行することができました．

楠本 真二教授には様々な場面で的確な助言をいただきました．また，時にはお菓子の差し入れなどもいただき，研究を表側と裏側の両面から支援していただきました．

肥後 芳樹教授は，旧井上研究室の教授に就任された後も楠本研究室へ顔を出してくださり，発表への指摘などを通じて研究を指導していただきました．

柗本 真佑助教には本研究の全ての過程においてご尽力いただき，多くのことを教わりました．研究のテーマ決めの際は多数のテーマや参考になる論文を提示してくださったり，実験の仕様決めに迷ったときは的確なアドバイスをいただきました．進捗がよくないときには力強く励ましてくださったり，原稿を書いているときには何度も添削をしてくださいました．発表についての様々なアドバイスやアイデアもいただき，私の研究を効果的に人に伝えることができました．

事務補佐員の橋本 美砂子様には，研究室配属や出張の手続き，研究室環境の整備など研究活動を裏から支えていただき，何不自由なく研究活動を行うことができました．

楠本研究室の先輩方には，日々の研究活動において何度も助けていただきました．実験で些細なことで何度も躓いていると，先輩が横から見てくださり，自分では気づかなかった様々な気づきを得られました．また，発表時の工夫や，研究室での生活についてもたくさんのことを教えていただきました．

楠本研究室の同期の方々とは，研究室仲間として，友達として，切磋琢磨し研究に励むことができました．大学院入試の勉強を共にしたり，時には研究を離れて雑談をして，充実した研究室生活を送ることができました．

家族，それから友人には，生活を支えていただき，時には研究の悩みを聞いてもらうこともあり，力強い心の支えになりました．

本研究を支えてくださった全ての方に，心から感謝申し上げます．

参考文献

[1] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, Vol. 42, No. 8, pp. 707–740, 2016.

[2] Higor A. de Souza, Marcos L. Chaim, and Fabio Kon. Spectrum-based software fault localization: A survey of techniques, advances, and challenges. *arXiv e-prints*, p. arXiv:1607.04347, 2016.

[3] Fabian Keller, Lars Grunske, Simon Heiden, Antonio Filieri, Andre van Hoorn, and David Lo. A critical evaluation of spectrum-based fault localization techniques on a large-scale software system. In *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security*, pp. 114–125, 2017.

[4] J.A. Jones, M.J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the International Conference on Software Engineering*, pp. 467–477, 2002.

[5] Chris Parnin and Alessandro Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the International Symposium on Software Testing and Analysis*, pp. 199–209, 2011.

[6] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. Automatic software repair: A survey. In *Proceedings of the International Conference on Software Engineering*, p. 1219, 2018.

[7] Alexandru Marginean, Johannes Bader, Satish Chandra, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, and Andrew Scott. Sapfix: Automated end-to-end repair at scale. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*, pp. 269–278, 2019.

[8] Shaimaa Ali, James H. Andrews, Tamilselvi Dhandapani, and Wantao Wang. Evaluating the accuracy of fault localization techniques. In *Proceedings of the International Conference on Automated Software Engineering*, pp. 76–87, 2009.

[9] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, Vol. 82, No. 11, pp. 1780–1792, 2009.

[10] René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the International*

Symposium on Software Testing and Analysis, pp. 437–440, 2014.

[11] Qiong Shi, Zhenyu Zhang, Zhifang Liu, and Xiaopeng Gao. Enhance fault localization using a 3d surface representation. In *Proceedings of the International Conference on Computer Research and Development*, pp. 720–724, 2010.