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

複数言語対応かつ算出式変更可能なバグローカリゼーションフレームワークの提案

下條 清史† 坂本 一憲† 鷲崎 弘宜† 深澤 良彰†

†早稲田大学大学院基幹理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1

E-mail: †kiyofumi-1906@akane.waseda.jp, washizaki@waseda.jp

あらまし バグローカリゼーションとは，ソフトウェアテスティングで得られる情報からコード内のバグの位置を推定する手法で，近年その有用性が注目を浴びている．しかし，既存のバグローカリゼーションツールでは，単一のプログラミング言語および単一の Suspiciousness 算出式にしか対応していない．そこで，我々は複数のプログラミング言語に対応し，Suspiciousness 算出式が容易に変更可能なバグローカリゼーションフレームワークを提案する．複数のプログラミング言語に対してそれぞれの Suspiciousness 算出式を適用し，比較を行うことで本手法の有用性を確認する．

キーワード バグローカリゼーション，フォールトローカリゼーション，デバッグ，複数プログラミング言語

Multiple Programming Languages Support Customizable Bug Localization Framework

Kiyofumi SHIMOJO† Kazunori SAKAMOTO† Hironori WASHIZAKI†
and Yoshiaki FUKAZAWA†

†Fundamental Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-0855 Japan

E-mail: †kiyofumi-1906@akane.waseda.jp, washizaki@waseda.jp

Abstract Bug localization technique is effectual approach for locating software bugs using the number of failing and passing test cases that execute the statement. However, existing bug localization tools only support one type of programming languages and one type of suspiciousness metrics. In this paper, We presents customizable bug localization framework supporting multiple programming languages and multiple suspiciousness metrics. By apply each suspiciousness metrics to multiple programming languages and comparing, we show the availability of our proposed.

Keyword Bug Localization, Fault Localization, Debugging, Multiple Programming Languages

1. はじめに

バグローカリゼーションとは，テスト失敗の原因となるコードの箇所を推定する手法で，テストケースの実行パスとテスト結果から，コード内の各ステートメントに対して，それらがバグを含んでいるかどうかの疑わしさを示すメトリクス Suspiciousness（Susp）の値を算出する．

バグローカリゼーションは，バグ位置推定においてその有効性が注目を浴びているが，既存のバグローカリゼーションツールは単一のプログラミング言語にしか対応しておらず，加えて単一の Susp 算出式しか適用できない．そこで我々は，複数のプログラミング言語に対応し，Susp 算出式が変更可能なバグローカリゼーションフレームワークを提案する．複数のプログラミング言語に対してそれぞれの Susp 算出式の適用を行うことで本手法の有用性を確認する．

本論文では以下の 5 点を研究課題をとする．

RQ1：実開発においてバグ位置推定に役立つか
RQ2：Susp 算出式を容易に変更可能か
RQ3：複数の異なるプログラミング言語で適用可能か
RQ4：新しい Susp 算出式を容易に追加できるか
RQ5：新しいプログラミング言語に容易に対応可能か

本論文の貢献を以下に示す．

・実開発におけるバグローカリゼーションの有効性の検証結果
・複数プログラミング言語に対応したバグローカリゼーションフレームワークの実装
・Susp 算出式が容易に変更・追加可能なバグローカリゼーションフレームワークの実装

2. 背景

2.1. バグローカリゼーション

バグローカリゼーションでは，テストケースの実行パスとテスト結果を用いて各ステートメントの Susp 値を算出する．デバッグ時は，各ステートメントを Susp 値が高い順にランキング化し，上位のステートメントから検証する．

2.1.1. Suspiciousness の算出

表 2.1 に Susp 値の算出例を示す．TC1～TC5 はテストケース，S1～S5 はステートメント，テストケースとステートメントが交わるセルの "✔" は，テストケースがそのステートメントを実行したことを示す．

ステートメントごとに成功・失敗テストケースの総数とステートメントを実行した成功・失敗テストケース数を集計し，それらを計算式にあてはめて Susp 値を算出する． 表 2.1 では，Susp 値同様に全ての失敗テストケースが実行し，なおかつ全ての成功テストケースが実行していない S3 が一番疑わしいと推測できる．

表 2.1 入力情報と Susp 値（Tarantula）の算出例

	テストケース（TC）					
ステートメント（S）	TC1	TC2	TC3	TC4	TC5	Susp
S1	✔	✔	✔	✔	✔	0.500
S2	✔			✔	✔	0.429
S3	✔		✔			1.000
S4			✔	✔		0.600
S5		✔			✔	0.000
テスト結果	失敗	成功	失敗	成功	成功	

2.1.2. ランキングの作成と活用

表 2.2 に表 2.1 の例を Susp 値が大きい順にソートしたランキングを示す．Susp 値を算出した後，その値が大きい順，つまりバグの存在が疑わしいステートメント順にソートしてランキング化する．

デバッグ時は，このランキングの上位ステートメント，すなわちバグが潜んでいる可能性が高いと推測されたステートメント順に検査する．

表 2.2 Susp 値によるランキング

ランク	ステートメント（S）	Susp（Tarantula）
1	S3	1.000
2	S4	0.600
3	S1	0.500
4	S2	0.429
5	S5	0.000

2.2. 既存のバグローカリゼーション

2.2.1. Tarantula

代表的な既存のバグローカリゼーションツールとして Tarantula[1][2]があげられる．(1)~(3)式に Tarantula の Susp 算出式を示す．Pass(All)・Fail(All)は成功・失敗テストケースの総数，Pass(Sn)・Fail(Sn)は対象のステートメントを実行した成功・失敗テストケース数を示す．

$$\%\mathrm{Pass}(\mathrm{Sn}) = \frac{\mathrm{Pass}(\mathrm{Sn})}{\mathrm{Pass}(\mathrm{All})} \quad \ldots(1)$$

$$\%\mathrm{Fail}(\mathrm{Sn}) = \frac{\mathrm{Fail}(\mathrm{Sn})}{\mathrm{Fail}(\mathrm{All})} \quad \ldots(2)$$

$$\mathrm{Susp}(\mathrm{Sn}) = \frac{\%\mathrm{Fail}(\mathrm{Sn})}{\%\mathrm{Fail}(\mathrm{Sn}) + \%\mathrm{Pass}(\mathrm{Sn})} \quad \ldots(3)$$

図 2.1 に Tarantula の出力に用いる色の範囲を示す．Tarantula の出力は色を用いて知覚的に提示される．Susp の値に応じて緑から赤の色（彩度）を決定し，コードにハイライトする形式で出力する．色が赤に近いほどバグの存在が疑わしい部分であり，緑に近ければ安全，その中間は黄色で表現される．

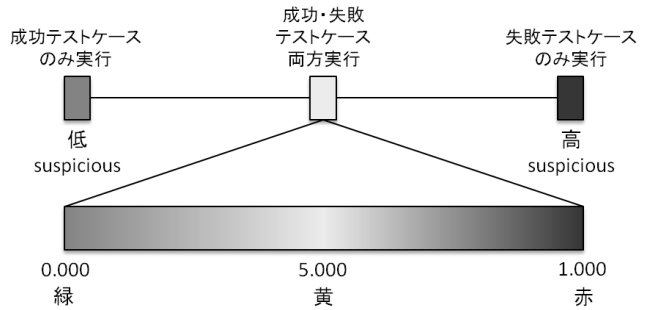

図 2.1 Tarantula の出力で使用する色の範囲

2.2.2. その他の Suspiciousness 算出式

(4)~(6)式に Tarantula 以外の Susp 算出式を示す．比較的有名なメトリクスである Ochiai[3]，Jaccard[4]，Russell[5]の Susp 算出式を示す．提案手法においても，まずは Tarantula に Ochiai，Jaccard，Russell を加えた４種類を Susp 算出式を切り替える対象としている．

$$\mathrm{Ochiai:Susp}(\mathrm{Sn}) = \frac{\mathrm{Fail}(\mathrm{Sn})}{\sqrt{\mathrm{Fail}(\mathrm{All}) \times \{\mathrm{Fail}(\mathrm{Sn}) + \mathrm{Pass}(\mathrm{Sn})\}}} \ldots(4)$$

$$\mathrm{Jaccard:Susp}(\mathrm{Sn}) = \frac{\mathrm{Fail}(\mathrm{Sn})}{\mathrm{Fail}(\mathrm{All}) + \mathrm{Pass}(\mathrm{Sn})} \quad \ldots(5)$$

$$\mathrm{Russell:\ Susp}(\mathrm{Sn}) = \frac{\mathrm{Fail}(\mathrm{Sn})}{\mathrm{Fail}(\mathrm{All}) + \mathrm{Pass}(\mathrm{All})} \ldots(6)$$

2.3. バグローカリゼーションにおける問題点

2.3.1. 実開発での有効性の検証

バグローカリゼーションは有効性が認識されつつも実開発に導入されていない．表 2.3 に既存のバグローカリゼーションに関する研究で用いられる主な評価用プログラムのデータを示す[6]．既存研究において評価に用いられるプログラムのほとんどが表 2.3 に示すプログラムである．これらのプログラムが多用される理由は，既にバグの位置情報や豊富なテストケースが提供されているからである．しかしながら，実開発と比較して規模が小さいプログラムが多く，バグローカリゼーションの効果が対象プログラムとテストケースに依存する[7]特性を考慮すれば，実開発に対するバグローカリゼーションの有効性の検証が十分になされてきたとは言えない．

表 2.3 既存研究で用いられる評価プログラム

プログラム	バージョン	LOC	テストケース数
print_tokens	7	472	4056
print_tokens2	10	399	4071
replace	32	512	5542
schedule	9	292	2650
schedule2	10	301	2680
space	58	6218	13585
tcas	41	148	1578
tot_info	23	440	1054

2.3.2. 単一のプログラミング言語対応

既存のバグローカリゼーションツールのほとんどは，単一のプログラミング言語にしか対応していない．そのため，一つのバグローカリゼーションツールで複数のプログラミング言語に対してバグローカリゼーションを適用することが出来ず，適用範囲が限定される．たとえば，WEB アプリケーションでよく行われる複数のプログラミング言語を用いた開発では，バグローカリゼーションが適用出来ないほか，言語置換（プログラミング言語の変更）が行われると継続して適用できなくなる．

2.3.3. 単一の Suspiciousness 算出式対応

バグローカリゼーションで最も一般的な研究は，Susp 算出式の提案・改良であり，上述した 4 種類のほかにも多種多様に存在している[8]．さらに Susp 算出式同士の有効性の優劣は，比較に使用するプログラムやテストケースなどのデータセットによって左右されるため，一概にどの Susp 算出式が最も優れているかを断定することは出来ない．

このような特性を持ちながらも，既存のバグローカリゼーションツールでは単一の Susp 算出式しか適用できないため，1 つの対象プログラムに対して複数の Susp 算出式を適用して合議制を用いてバグの位置を推定しようとした場合，単一のツールでは対応できないために適用が困難であり，複数のツールの使用はコストが余計にかかる．

2.3.4. 拡張性の欠如

上述のように，既存のバグローカリゼーションツールでは単一のプログラミング言語に対して，単一の Susp 算出式しか適用できない．そのため，プログラミング言語のバージョンアップや新規のプログラミング言語に対する拡張性が欠如しているだけでなく，開発者自身がカスタマイズした Susp 算出式の適用にも対応できないため，ツールが適用できる範囲が非常に限定的であり，実開発への適用に則していない．

3. 提案手法

3.1. 問題解決のアプローチ

実開発におけるバグローカリゼーションの有効性を検証するために，提案手法を実開発に適用した．本論文では，この実開発での適用結果に基づいてバグローカリゼーションの実開発での有効性を検証する．

また，提案手法ではオープンソースのカバレッジ測定フレームワークの Open Code Coverage Framework（以下，OCCF）[9][10][11][12]をベースに拡張することで，複数のプログラミング言語に対してテストケースの実行パスの取得を実現している．

OCCF は C/C++，Java，Python2，Python3 に対してテストカバレッジが取得できるため，複数のプログラミング言語に対してテストケースの実行パスが取得可能である．この機能により提案手法においても複数のプログラミング言語対応を実現している．

複数プログラミング言語対応の実現により，提案手法では複数のバグローカリゼーションツールを別々に用意することなくプログラミング言語横断的なプロジェクトに対してバグローカリゼーションを適用することが可能となり，言語置換などのプログラミング言語の変更にも対応が容易となる．

さらに，提案手法ではコマンド実行におけるオプション指定によって算出式の種類を指定することで Susp 算出式の切り替えを可能とし，複数種類の Susp 算出式を容易に使用可能としている．Susp 算出式の変更が可能となることで，プログラム毎に算出式を切り替えてバグローカリゼーションの適用が出来るほか，複数の Susp 算出式の合議制によるバグ位置推定も可能となる．

3.2. 複数言語対応かつ算出式変更可能な バグローカリゼーションフレームワーク

3.2.1. 提案手法の全体像

図 3.1 に提案手法の全体像を示す．まず，カバレッジ測定用コードをソースコードおよびテストコードに挿入する．カバレッジ測定用コード挿入後のソースコードおよびテストコードでテストを実行することで，カバレッジデータを取得する．

次に，カバレッジデータから得られる各テストケースの実行パスとテスト結果を記述したファイルから読み込んだテスト結果を用いて，ソースコードの各ステートメントの Susp 値を算出する．このとき，Susp 算出式は Python 言語のスクリプト形式で記述したファイルから読み込む．

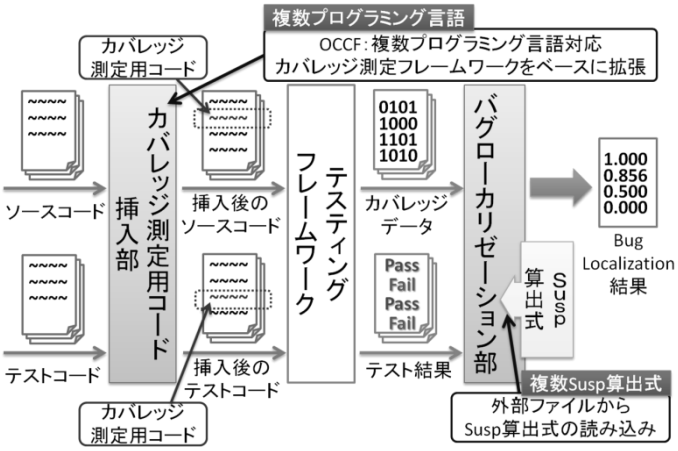

図 3.1 提案手法の全体像

3.2.2. カバレッジ測定用コードの挿入

カバレッジコード挿入部では，主にソースコードとテストコードにカバレッジ測定用コードを挿入する．

カバレッジコードの挿入には OCCF の機能を利用しているが，C 言語のプログラムにカバレッジ測定用コードを挿入するには，プリプロセス後のコードである必要があり，これを起因としてバグローカリゼーション結果出力時に表示されるステートメントの行番号が正しく表示されない問題が生じる．そこで，C 言語のプログラムに対して正しい行番号を表示するための機能を追加した．

3.2.3. C 言語における行番号対応付け機能

C 言語のプログラムにカバレッジ測定用コードを挿入するにはプリプロセス後のコードである必要があるのだが，プリプロセス後のコードにカバレッジ測定用コードを挿入した場合，バグローカリゼーション結果の出力時に表示されるステートメントの行番号がプリプロセス後のコードの番号となる．特にライブラリなどを多く使用した場合，プリプロセス前後のコードで大きく行番号がずれるため，バグローカリゼーションによって推定した位置が大きくずれる．

図 3.2 に行番号対応付け機能の概要を示す．まず，元のソースコードの行番号を記録するためにプリプロセス前に行番号測定用コードを挿入する．

リスト 3.1 と 3.2 に行番号測定用コードの挿入前後のコード例を示す．対象コードに対してステートメントの行の下に「#line 行番号」のようなコードを挿入する．

リスト 3.3 にリスト 3.2 のコードのプリプロセス後の行番号測定用コードを示す．挿入したコードがプリプロセス後に「#行番号 "相対パス"」のように変化していることが分かる．この記述をもとに行番号参照用ファイルを生成する．生成が完了したら，カバレッジ測定用コードの挿入を行う．

バグローカリゼーション結果出力時には，行番号参照用ファイルを読み込んで Dictionary 化し，プリプロセス後の行番号から元の行番号を検索する．

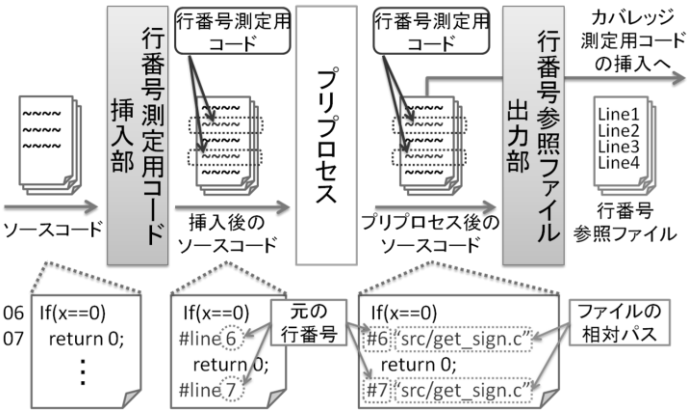

図 3.2 行番号対応付け機能

リスト 3.1 行番号測定用コード挿入前

```
06:      if (x == 0)
07:          return 0;
```

リスト 3.2 行番号測定用コード挿入後

```
07:      if (x == 0)
08:  #line 6
09:          return 0;
10:  #line 7
```

リスト 3.3 プリプロセス後の行番号測定用コード

```
11:      if (x == 0)
12:  # 6 "get_sign/get_sign.c"
13:          return 0;
14:  # 7 "get_sign/get_sign.c"
```

3.2.4. Suspiciousness 値の算出

Susp 値の算出には，まず生成されたカバレッジデータファイルから各テストケースの実行パスを取得する．テスト結果は，テスト結果を記述した外部のファイルを読み込むことで取得している．

また，提案手法では容易に Susp 算出式を変更可能にするため，Susp 算出式を Python 言語のスクリプト形式で記述したファイル（以降，スクリプトファイル）から読み込む形式で実装しており，Tarantula，Ochiai，Jaccard，Russell の 4 種類の Susp 算出式は標準で搭載しているほか，Susp 算出式の定義は非常に記述量が少なく，Tarantula のスクリプトファイルの記述量は 16 行である．

図 3.3，図 3.4 に C 言語に対する提案手法の出力結果の例を示す．図 3.3 は Tarantula，図 3.4 は Ochiai で算出した結果である．「ファイルパス> 行番号：Susp 値，%P(Sn)，%F(Sn)」の形式で出力されれおり，Tarantula と Ochiai で Susp 値が違うことが分かる．

```
mid -t mid/klee-out-0 -m Ochiai.py
metrics : Ochiai.py
risk, executedAndPassedCount / passedCount, executedAndFailedCount
src/get_mid.c>: 8 - 13                    : 0.522, 1.000, 1.000
src/get_mid.c>: 9 - 12                    : 0.577, 0.250, 0.667
src/get_mid.c>: 10                        : 0.000, 0.250, 0.000
src/get_mid.c>: 12                        : 0.816, 0.000, 0.667
src/get_mid.c>: 15 - 20                   : 0.522, 1.000, 1.000
src/get_mid.c>: 16 - 19                   : 0.000, 0.750, 0.000
```

図 3.3 Ochiai の出力結果

```
mid -t mid/klee-out-0 -m Jaccard.py
metrics : Jaccard.py
risk, executedAndPassedCount / passedCount, executedAndFailedCount
src/get_mid.c>: 8 - 13                    : 0.273, 1.000, 1.000
src/get_mid.c>: 9 - 12                    : 0.400, 0.250, 0.667
src/get_mid.c>: 10                        : 0.000, 0.250, 0.000
src/get_mid.c>: 12                        : 0.667, 0.000, 0.667
src/get_mid.c>: 15 - 20                   : 0.273, 1.000, 1.000
src/get_mid.c>: 16 - 19                   : 0.000, 0.750, 0.000
```

図 3.4 Jaccard の出力結果

4. 評価

4.1. 実開発においてバグ位置推定は役立つか

提案手法は C 言語の実開発に Tarantula の Susp 算出式を適用した．バグローカリゼーションを用いたデバッグでは，改善すべき点が見られたものの適用したプロジェクトの方から一定の評価を得た．

図 4.1 に適用対象の Susp 値の範囲の割合とバグ累積結果を示す．図 4.1 の折れ線は Susp 値のによるステートメントの累積とバグの累積を示し，棒は範囲内の Susp 値を示したステートメント数を示す．適用により，コードの 10%以内に約 90%のバグ，5%以内に約 66%のバグを局所化した．この結果は，既存研究に照らし合わせても有効的であり，実開発での適用でもバグローカリゼーションの有効性を確認できた．

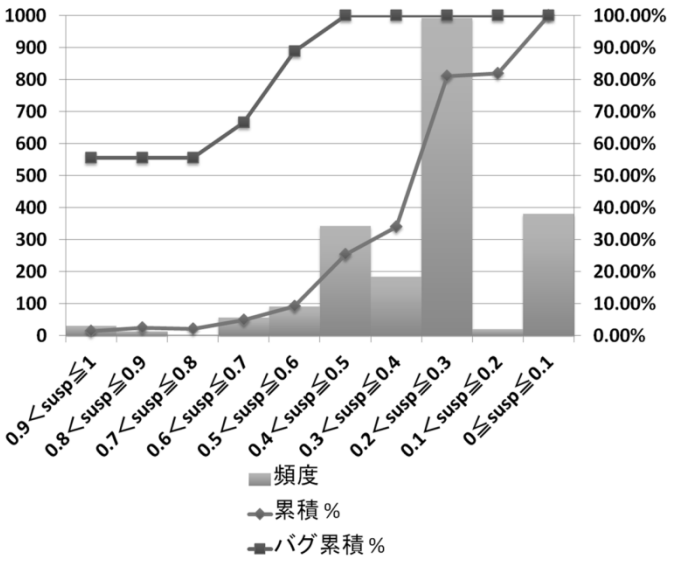

図 4.1 バグローカリゼーションの適用結果

4.2. Suspiciousness 算出式を容易に変更可能か

提案手法では，バグローカリゼーションの実行時に，標準で搭載している Susp 算出式が記述されたスクリプトファイル名をオプション指定して実行することで容易に Susp 算出式が変更可能である．

図 3.3 と図 3.4 では，上部で-m オプションの後に Susp 算出式が記述されたスクリプトファイル名を指定することで，Susp 算出式が切り換えられている．

ファイル名の指定だけで，少なくとも標準搭載している Tarantula，Ochiai，Jaccard，Russell の Susp メトリクスに変更可能であり，十分に容易であるといえる．

4.3. 複数の異なるプログラミング言語で適用可能か

提案手法では，C 言語に加えて Java 言語に対しても適用を確認できた．図 4.3 に Java 言語に対して適用した出力例を示す．OCCF が対応している Python 言語に対してはまだ適用できておらず，現状出力として確認できるのは 2 種類のプログラミング言語である．現状では C 言語と Java 言語の 2 種類だけではあるが，複数の異なるプログラミング言語に適用した．

```
metrics : Tarantula.py
risk, executedAndPassedCount / passedCount, executedAndFailedCo

src>GetMid.java>class GetMid>method getMid>: 8 - 13    : 0.500,
src>GetMid.java>class GetMid>method getMid>: 9 - 12    : 0.667,
src>GetMid.java>class GetMid>method getMid>: 10        : 0.500,
src>GetMid.java>class GetMid>method getMid>: 12        : 1.000,
src>GetMid.java>class GetMid>method getMid>: 14 - 19   : 0.500,
src>GetMid.java>class GetMid>method getMid>: 15 - 18   : 0.000,
```

図 4.1 Java 言語に対して適用した出力例

4.4. 新しい Suspiciousness 算出式を容易に追加できるか

情報理工学専攻の大学院生 2 名に対して Tarantula のスクリプトファイルを参考に Ochiai，Jaccard，Russell のスクリプトファイルを作成させる被験者実験を実施した．表 4.1 に被験者実験の結果を示す．

3 種類の Susp 算出式の追加に要した時間は，1 種類平均で約 6 分半であった．被験者は Python 言語の使用経験がない学生であったが，記述量の少なさもあり短時間で作成できた．今回の実験では Ochiai 追加時に math モジュールを import して次乗根を記述する部分に時間を要したため，標準搭載のスクリプトファイルであらかじめ import しておくなどの改善を図ることでさらに時間が短縮され，あまり複雑でない Susp 算出式であれば 2~3 分で容易に追加が可能である．

表 4.1 Susp 算出式追加に要した時間

被験者	Ochiai		Jaccard		Russell	
	実装順	時間（分）	実装順	時間（分）	実装順	時間（分）
A	3	15:24	1	8:38	2	2:16
B	2	5:56	3	1:47	1	5:28

4.5. 新しいプログラミング言語に容易に対応可能か

提案手法ではテストケースの実行パス取得に OCCF のカバレッジ測定機能を利用している．既に坂本ら[10][11][12]が新しい言語に対応するための保守性を評価しており，提案手法でも同様の保守性を有する．

5. 関連研究

Jones ら[1]は，算出した Susp 値を色に変換することで，視覚的にバグ位置推定を支援する Tarantula を提案している．Tarantula の出力結果は視覚的であるものの，単一の算出式を C 言語のプログラムにのみしか適用していない．これに対して提案手法では，Tarantula を含む複数の Susp 算出式が使用でき，複数のプログラミング言語に対してバグローカリゼーションを適用できる．

6. まとめ

本論文では，複数のプログラミング言語に対応し，Susp 算出式が変更可能なバグローカリゼーションフレームワークを提案した．提案手法により，一つのツールで複数のプログラミング言語に対してバグローカリゼーションが適用可能になったほか，柔軟な拡張性によって新しいプログラミング言語や複数言語で実装されたプログラムへの適用可能性を示した．さらに，Susp 算出式を容易に変更・追加可能なため，適用対象に応じた柔軟な Susp 算出式の切り替えやカスタマイズした Susp 算出式の適用を可能とし，複数算出式による合議制でのバグ推定の実現可能性も示すとともに，実開発においてバグローカリゼーションを適用し，実開発での有効性も検証した．

展望として，対応するプログラミング言語の拡大や，ランキング表示の拡充や HTML 出力によるコードの記述内容と対応づけた視覚的な出力結果表示の追加などが考えられる．

文　献

[1] Tarantula, http://pleuma.cc.gatech.edu/aristotle/Tools/tarantula/index.html

[2] Y. Yu, J. Jones and M. Harrold, "An Empirical Study of the Effects of Test-Suite Reduction on Fault Localization", in Proc. of Intl. Conference on Software Engineering ICSE 2008, ACM Press, 2008.

[3] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization." In Testing: Academic and Industrial Conference, Practice and Research Techniques, Windsor, UK, September 2007.

[4] P. Jaccard, “Etude comparative de la distribution florale dansune portion des Alpes et des Jura”, Bull. Soc. Vaudoise Sci.Nat, vol. 37, pp. 547–579, 1901.Simson Garfinkel

[5] P. Russel and T. Rao, “On habitat and association of species of anopheline larvae in south-eastern Madras”, J. Malar. Inst. India, vol. 3, pp. 153–178, 1940.

[6] Y. Yu, J. Jones and M. Harrold, "An Empirical Study of the Effects of Test-Suite Reduction on Fault Localization", in Proc. of Intl. Conference on Software Engineering ICSE 2008, ACM Press, 2008.

[7] Vidroha Debroy, W. Eric Wong, "On the Consensus-Based Application of Fault Localization Techniques". COMPSAC Workshops, 2011, 506-511

[8] Z. Zhang, WK Chan, TH Tse, YT Yu, P. Hu, "Non-Parametric Predicate-Based Fault Localization", Journal of Systems and Software 2011

[9] Open Code Coverage Framework https://github.com/exKAZUu/OpenCodeCoverageFramework

[10] K. Sakamoto, K. Shimojo, R. Takasawa, H. Washizaki and Y. Fukazawa, "OCCF: A Framework for Developing Test Coverage Measurement Tools Supporting Multiple Programming Languages," 6th IEEE International Conference on Software Testing, Verification and Validation, 2013.

[11] K. Sakamoto, H. Washizaki, Y. Fukazawa, "Open Code Coverage Framework: A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages," 10th International Conference on Quality Software, 2010.

[12] K. Sakamoto, F. Ishikawa, H. Washizaki, and Y. Fukazawa, "Open Code Coverage Framework: A Framework for Consistent, Flexible and Complete Measurement of Test Coverage Supporting Multiple Programming Languages," IEICE Transactions on Information and Systems, Vol.E94-D, No.12, pp.2418-2430, 2011