失敗テストの実行経路に着目することで SBFL の疑惑値算出精度を向上させる手法の提案

2019SE008 疋田涼也
指導教員：名倉正剛

1 はじめに

ソフトウェアを開発する上でデバッグは甚大な労力と費用を必要とされる作業である．これはこのデバッグ作業にかかる費用はソフトウェア開発の全体のコストに対し半数を超えているという報告もある程である [1]．本研究はこの SBFL を使用し，疑惑値の算出方法を変更することにより欠陥箇所推定の精度を向上させることを目的とする．

2 Spectrum-Based Fault Localization (SBFL)

SBFL は，失敗するテストケースにより実行されたステートメントには欠陥が含まれている可能性が高いという考えに基づき，テストケースごとの成否と，プログラム中で実行されるステートメントの実行経路の情報を用いて，欠陥を含む箇所を推測する技術である [2]．SBFL 技術はプログラム中のステートメントごとに対し性を疑惑値として数値化する．この疑惑値を使用して原因箇所を推定する．

SBFL により原因箇所特定を実施する際には，Java の単体テストツールである JUnit[*1]により，各テストケースに対する成否をアサーションで判別する方法で疑惑値を算出する．

本研究では疑惑値算出に Gzoltar[*2]を使用する．

3 本研究で対象にする課題

3.1 SBFL の課題

SBFL での疑惑値算出の課題として，複数のアサーションを含むテストメソッドの一部のアサーションが条件を満たさない場合，条件を満たすアサーションの実行経路も含めて記録されることがある．その結果成功するテスト実行の実行経路が失敗したテスト実行の実行経路として扱われることになり，疑惑値の算出が適切に行われないことになる．

3.2 具体例

具体例として Ochiai の疑惑値算出方法をサンプルプログラムに対して使用した場合の疑惑値算出と原因箇所特定の例を表 1 (a) 部分に示す．表 1 のプログラムコードの欄に記載した関数はデータを配列に保存しており，配列の長さよりも保存するデータが多くなると保存に失敗し，−1 を返す仕様になっている．ここでは簡単化のため，この配列の長さを 2 と想定した場合のテストの結果を示している．

このプログラムに対しては test1 ～ test3 の 3 つのテストケースを用意しており，それらのテストケースを ソースコード 1 に示す．

ソースコード 1 複数アサーションを含むテストケース

```
void test1() { // テストケース#1 (test1)
    assertEquals(add_data(1), 1);
}
void test2() { // テストケース#2 (test2)
   assertEquals(add_data(1), 1);
   assertEquals(add_data(2), 2);
   assertEquals(add_data(3), 3);
}

void test3() { // テストケース#3 (test3)
   assertEquals(add_data(1), 1);
   assertEquals(add_data(2), 2);
   assertEquals(add_data(3), 3);
   assertEquals(add_data(4), 4);
   assertEquals(add_data(5), 5);
}
```

このプログラムの失敗の原因は 2 行目の配列の長さより保存するデータが多いことであり，失敗するアサーションの実行の際には 1〜3 行目が実行される．しかしそれ以前に実行されるアサーションが成功しているので，失敗したテストメソッドの実行経路に成功するアサーションの実行経路が含まれる．このように，失敗するアサーション実行に関連しない 4〜5 行目の成功する実行の経路も疑惑値が高く算出され，このことが原因箇所の特定を困難にする．

4 提案

4.1 手順概要

以前に記述した課題に対し，失敗するテストケースに着目し，テストメソッドの実行においてはじめて対象のメソッド呼出が失敗する実行の実行経路のみを使用する手法を提案し，提案手法の手順を以下に示す．

[1] テスト実行の都度実行経路を保存

[2] 失敗するテスト実行の経路を保存した場合その時点までに収集した成功テストの実行経路を削除

これらの手順により，テストメソッドに成功する実行と失敗する実行が混在する場合でも，失敗するテストメソッドの実行経路に成功する実行による実行経路を含めずに疑惑値を算出することができる．

4.2 手順 1

上記で示した既存の gzoltar を使用して疑惑値を算出する場合，gzoltar の実行時に生成されるファイルにテストメソッドごとで実行経路情報を記録していく．

[*1] https://junit.org/junit5/
[*2] https://gzoltar.com/

表 1 通常の手法と提案手法を使用した場合の疑惑値算出と原因箇所特定の例

#	プログラムコード	(a) test1 (x=1) [1 回実行]	(a) test2 (x=1,2,3) [3 回実行]	(a) test3 (x=1,2,3,4,5) [5 回実行]	(a) 疑惑値	(b) test1 (x=1) [1 回実行]	(b) test2 (x=1,2,3) [3 回実行]	(b) test3 (x=1,2,3,4,5) [5 回実行]	(b) 疑惑値
1	`public int add_data(int x){`	✓	✓	✓	0.816	✓	✓	✓	0.894
2	`  if(idx >= data.length){`	✓	✓	✓	0.816	✓	✓	✓	0.894
3	`  return -1;`		✓	✓	1.0		✓	✓	1.0
	`  }else{`				×				×
4	`  data[idx++] = x;`	✓	✓	✓	0.816	✓			0
5	`  return x;`	✓	✓	✓	0.816	✓			0
	`  }`				×				×
	`}`				×				×
	期待する返り値	(1)	(1,2,3)	(1,2,3,4,5)		(1)	(1,2,3)	(1,2,3,4,5)	
	得られた返り値	(1)	(1,2,-1)	(1,2,-1,-1,-1)		(1)	(1,2,-1)	(1,2,-1,-1,-1)	
	テスト結果	T	F	F		T	F	F	

成功する実行と失敗する実行を区別するためにアサーションの実行ごとの結果と実行経路情報が必要であるため，アサーションの実行の都度それぞれの実行経路情報と成否を保存するようにする．

4.3 手順 2

疑惑値を正しく算出するためには失敗するテストメソッドに含まれる成功するアサーションの実行経路を保存しないようにすることが求められる．手順 1 でアサーションの実行の都度それぞれの実行経路情報と成否を保存したことにより，アサーションの実行ごとにテストメソッド全体の成否を判別することができる．そこで実行結果を保存する際に，テストメソッド全体の成否が該当のアサーションの実行で成功から失敗に変化する場合，その時点までに収集した実行経路を破棄することが可能になる．この破棄により，失敗するテスト実行の経路以外の成功するテスト実行の経路情報のみを実行記録から削除することができる．そして gzoltar 疑惑値算出に使用される実行経路情報が記録されるファイルから実行経路情報が無くなった状態で失敗するテスト実行の経路情報を記録する．このとき記録される情報は失敗するテスト実行の経路のみとなる．

5 実施例

提案手順により，テストメソッドに成功するアサーションと失敗するアサーションが混在する場合でも，失敗するテストメソッドの実行経路に成功するアサーションの実行による実行経路を含めずに疑惑値を算出した例を以下に 4 つ示す．

- 規定の実行回数を超えると失敗するケース
- switch 文の条件が間違っている箇所を通ることにより実行が失敗するケース
- for ループで繰り返し実行すると成否が変わってしまうケース
- 共有変数を使用することにより欠陥箇所の通過以降失敗するケース

第 3 章で疑惑値算出の例で使用したプログラムは規定の実行回数を超えると失敗するケースの例である．要旨では 4 つの中でもこの例のみを紹介する．表 5.1 に提案手法を適用し，疑惑値を算出した例を表 1 の (b) に示す．

このプログラムは `test2()` と `test3()` の 3 回目のアサーションが失敗する実行となり，テストメソッド全体の成否が成功から失敗へ変化する．したがって，提案手法に基づき 3 回目のアサーションの実行経路を保存する前にこの時点までに収集した実行経路を破棄している．この結果，(a) に対して (b) は，`test2()` と `test3()` の 3 回目の失敗する実行を保存した際，それまでに記録した 1 回目と 2 回目の成功する実行経路を破棄し疑惑値を計算している．この結果 5 行目と 6 行目の疑惑値が 0 となり欠陥箇所の候補を絞り込むことができた．

6 おわりに

本研究では疑惑値算出の精度向上を目的として，失敗するテストケースに着目し，テストメソッドの実行においてはじめて対象のメソッド呼出が失敗する実行の実行経路のみを使用する手法を提案した．失敗テストケースの実行経路は成功する実行の経路と失敗する実行の経路とで区別し，疑惑値算出に使用する実行経路情報の記録工程を変更することにより，具体例として挙げたプログラムの 4 分の 3 において，既存の疑惑値算出方法に対し SBFL の原因箇所特定の精度を向上させることができた．

今後の課題としては，より複雑で規模の大きいプログラムを利用した提案手法の評価，成功するテストケースの実行経路を完全に使用しないことによる疑惑値算出への影響の有無の調査があげられる．

7 参考文献

[1] B. Vancsics，A.S.Arp et al Relationshipbetween the Effectiveness of Spectrum-Based Fault Localization and Bug-Fix Types inJavaScript Programs 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering , pp. 308-319，2020.

[2] W. E. Wong, R. Gao, et al A Survey on Software Fault Localization, 2016 IEEE Transactions on Software Engineering, vol. 42, no. 8, pp. 707-740, 2016,