

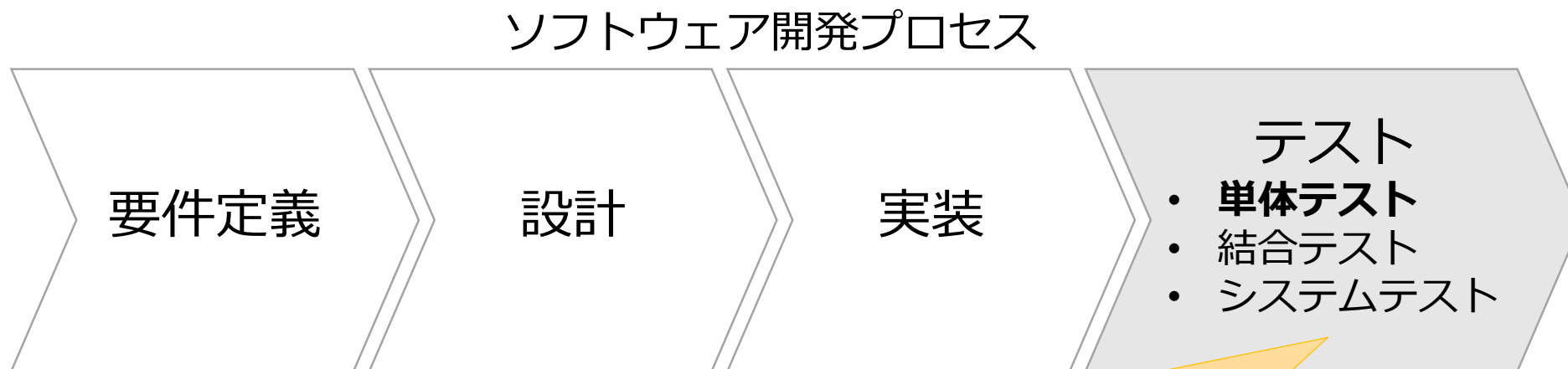
2020 修論発表会

ソースコードの類似性に基づいた テストコード自動推薦ツールSuiteRec

1811098 倉地亮介

ソフトウェアテスト

- ソフトウェア開発におけるソフトウェアの品質を確かめる工程



開発全体の30~50%の費用を占めると言われている[1]

[1] M. Ellims and J. Bridges and D. C. Ince. The Economics of Unit Testing. Empirical Software Engineering, 11(1):5-31, 2006.

テストコード自動生成ツール

- テスト工程を支援するために、これまでに様々な自動生成ツールが提案されてきた

EVASUITE

TestFul

Seeker



Randoop


PARASOFT®
Jtest®

Pex Grafter

自動生成ツールを利用することで、開発者の実装コストを削減し短期間でテストコードを作成できる

自動生成ツールにおける課題

- 自動生成されたテストコードは、保守作業を困難にする[2]
 - 対象コードの作成経緯や意図に基づいて生成されていないので開発者は理解しにくい
 - 開発者は自動生成されたコードを信用していない



テスト失敗の原因がテストコードの問題なのか、テスト対象のコードによるものなのか判断が難しい

[2] S. Shamshiri, J. M. Rojas, J. P. Galeotti, N. Walkinshaw, and G. Fraser. How do automatically generated unit tests influence software maintenance? In Proceedings of the International Conference on Software Testing, Verification and Validation (ICST), pages 250-261, 2018.

テストスメル

- テストコードの良くない実装を表す指標
 - Beckら[3]は、テストコードも適切に設計することの重要性を提唱した
 - Deursenら[4]は、11種類のテストスメルを提唱した(現在19種類)



Default Test

メソッド名が初期状態



Exception Handling

意図が分からない例外処理

```
public void test() throws Throwable {  
    Document document0 = new Document("", "");  
    assertNotNull(document0);  
    document0.procText.add((Character)'s');  
    String string0 = document0.stringify();  
    assertEquals("s", document0.stringify());  
    assertNotNull(string0);  
    assertEquals("s", string0);  
}
```



Assertion Roulette

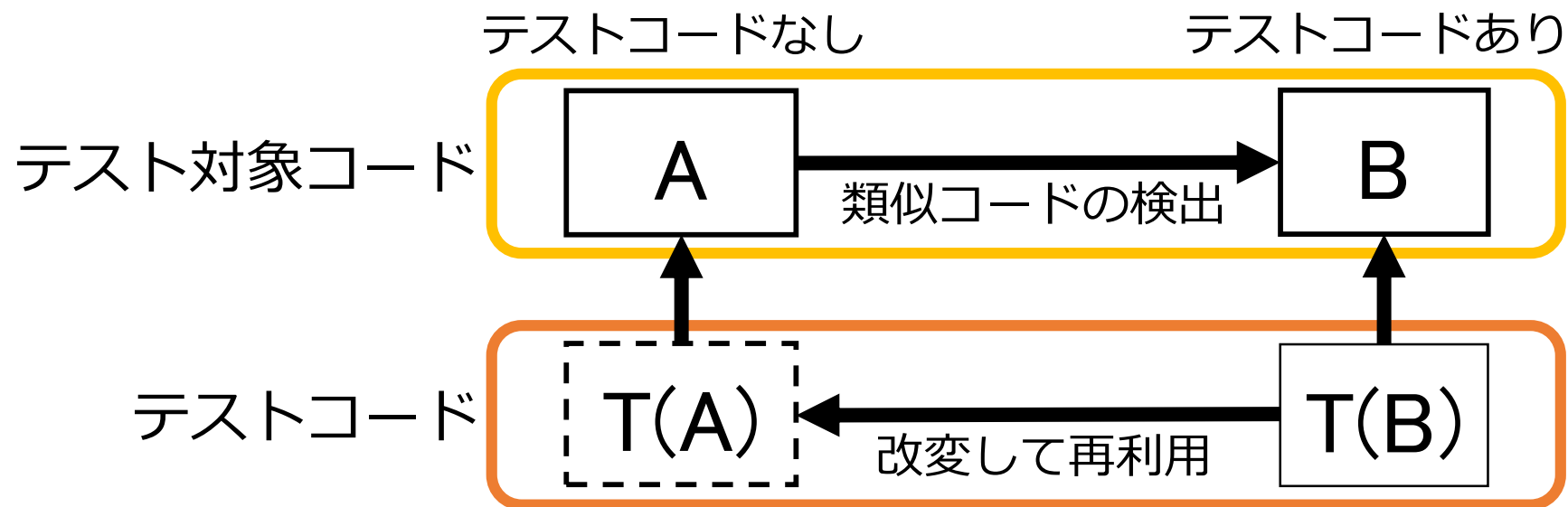
複数のassert文が存在する

[3] K. L. Beck. Test Driven Development: By Example. Addison-Wesley, 2002.

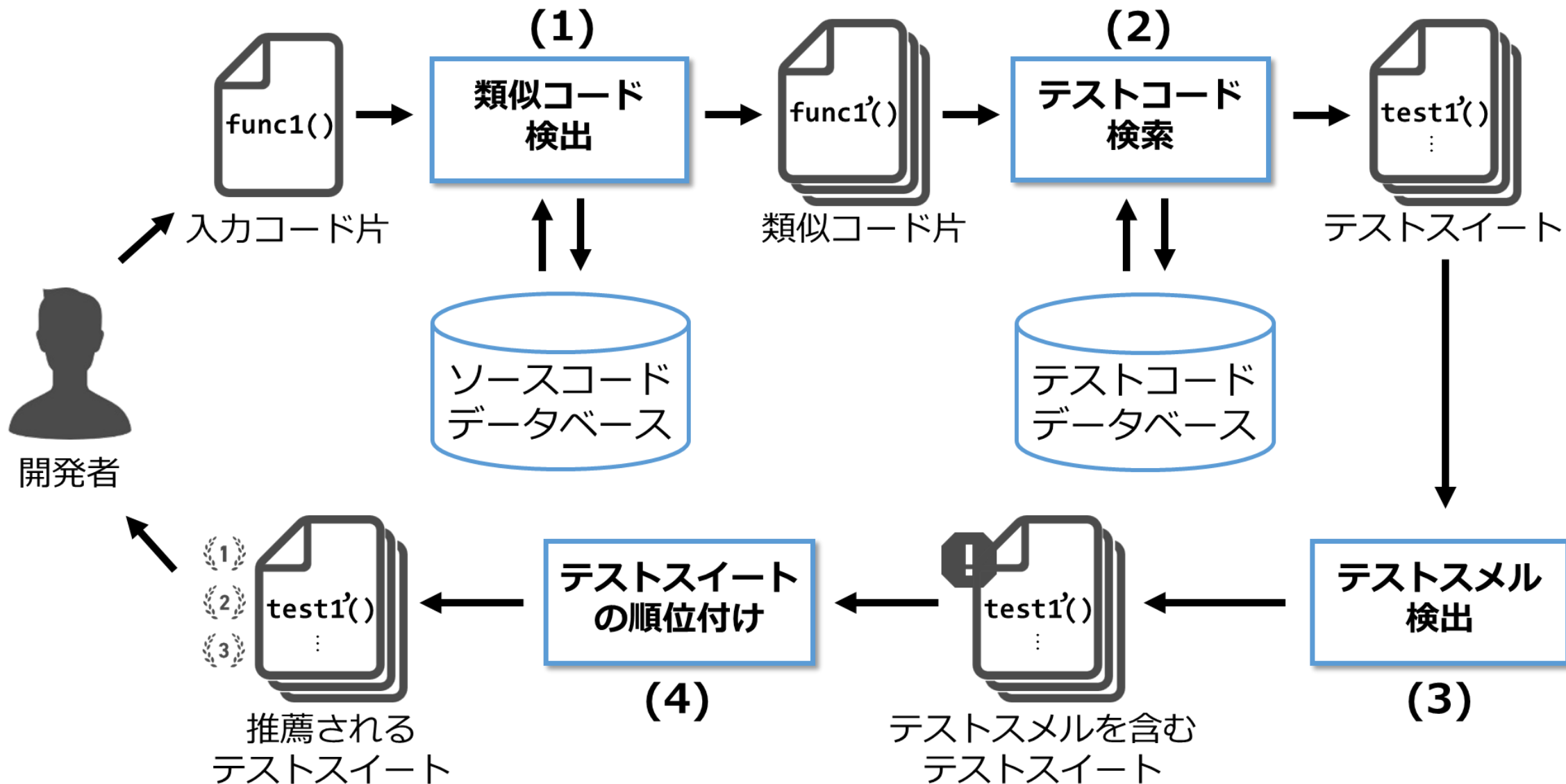
[4] A. Deursen, L. M. F. Moonen, A. Bergh, and G. Kok. Refactoring test code. Technical report, 2001.

研究目的とアイデア

- **目的:** 既存の高品質のテストを推薦することで開発者を支援
 - 命名規則に従った可読性の高いテストコードを利用できる
 - 人によって作成された信頼性の高いテストコードを利用できる
- **アイデア:** 類似するコード間でテストコードを再利用



提案ツールの概要

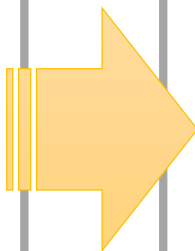


Step1: 類似コード片の検出

- 類似コード検出ツール: NiCad[5]
 - ソースコードのレイアウトを変換させ、行単位でソースコードを比較することで類似コード片を検出
 - 高精度・高再現率で類似コード片を検出可能

```
public int countPrice(int item[]){  
    int totalprice = 0;  
  
    for(int i=0; i < item.length; i++){  
        totalprice += item[i];  
    }  
    return totalprice;  
}
```

入力コード片



```
public int calcPrice(int ...cost){  
    int totalcost= 0;  
    int num = cost.length;  
    for(int i=0; i < num; i++){  
        totalcost += cost[i];  
    }  
    return totalcost;  
}
```

類似コード片

Step2: テストコードの検索

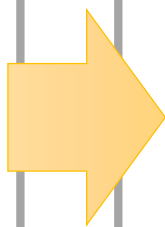
- 類似コード片に対応するテストコードを検索

- ① 命名規則によるクラス単位での対応付け
- ② テストコード内のメソッド呼び出しを確認
- ③ メソッド名の比較による対応付け

<CalcPriceクラス>

```
public int calcPrice(int ...cost){  
    int totalcost= 0;  
  
    for(int i=0; i < cost.length; i++){  
        totalcost += cost[i];  
    }  
    return totalcost;  
}
```

類似コード片(テスト対象)



<CalcPriceTestクラス>

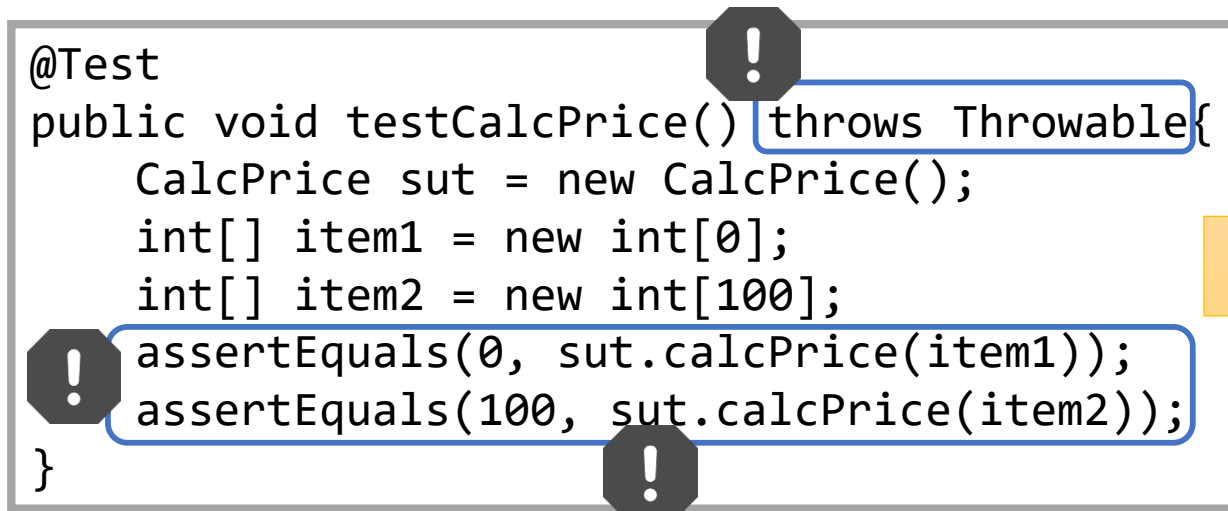
```
@Test  
public void testCalcPrice() throws Throwable{  
    CalcPrice sut = new CalcPrice();  
    int[] item1 = new int[0];  
    int[] item2 = new int[100];  
    assertEquals(0, sut.calcPrice(item1));  
    assertEquals(100, sut.calcPrice(item2));  
}
```

テストコード

Step3: テストスメルの検出

- テストスメル検出ツール: tsDetect[6]
 - 21種類のテストスメルを検出可能
 - 各テストスメルの検出精度: 85%~100%、再現率: 90%~100%

```
@Test
public void testCalcPrice() throws Throwable{
    CalcPrice sut = new CalcPrice();
    int[] item1 = new int[0];
    int[] item2 = new int[100];
    assertEquals(0, sut.calcPrice(item1));
    assertEquals(100, sut.calcPrice(item2));
}
```



テストコード

テストスメル	
1	Assertion Roulette
2	Exception Handling
3	Eager Test

[6] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni and F. Palomba: "On the distribution of test smells in open source android applications: An exploratory study", Pro. of CASCON, pp. 193-202 (2019).

Step4: 推薦されるテストスイートの順位付け

- テストスイートは、以下の2つの要素を基に順位付けられる

類似度

入力コード片と類似コード片
間の類似度(Step1)

テストスメル

テストスイート内に含まれる
テストスメルの数(Step3)

類似度を優先として並び替え、類似度が同じ場合
テストスメルの数で順位づける

SuiteRecのインターフェース

Clone Pairs 1 : 71.4% **3**

1 Input Code		2 Similarity Code	
<pre>public String fizzBuzz(int number) { if (number <= 0) { return "Not Natural Number"; } if (number % 15 == 0) { return "fizzbuzz"; } else if (number % 3 == 0) { return "fizz"; } else if (number % 5 == 0) { return "buzz"; } else { return Integer.toString(number); } }</pre>		<pre>public String fizzBuzz(int number) { if (number <= 0) { throw new RuntimeException(); } if (number % 15 == 0) { return "FizzBuzz"; } else if (number % 3 == 0) { return "Fizz"; } else if (number % 5 == 0) { return "Buzz"; } else { return Integer.toString(number); } }</pre>	

Test Suite					
Assertion Roulette	Conditional Test Logic	Default Test	Eager Test	4 Exception Handling	Mystery Guest
Lines 8 - 13 of tcs-expt-similar/src/test/java/jp/tcs/expt02/FizzBuzz1Test.java					
<pre>@Test(expected=RuntimeException.class) public void test1() { FizzBuzz1 fb = new FizzBuzz1(); fb.fizzBuzz(-1); }</pre>					
Lines 14 - 21 of tcs-expt-similar/src/test/java/jp/tcs/expt02/FizzBuzz1Test.java					
<pre>@Test public void returnBuzz_input5() throws Throwable { FizzBuzz1 fizzBuzz = new FizzBuzz1(); String actual = fizzBuzz.fizzBuzz(5); String expected = "Buzz"; assertEquals(expected, actual); }</pre>					

1 入力コード

2 類似コード

3 類似度(UPI)

4 テストスメル

5 テストスイート

- 実験概要

- 情報科学を専攻する修士課程の学生10人に3つのタスクのテストコードを作成してもらう

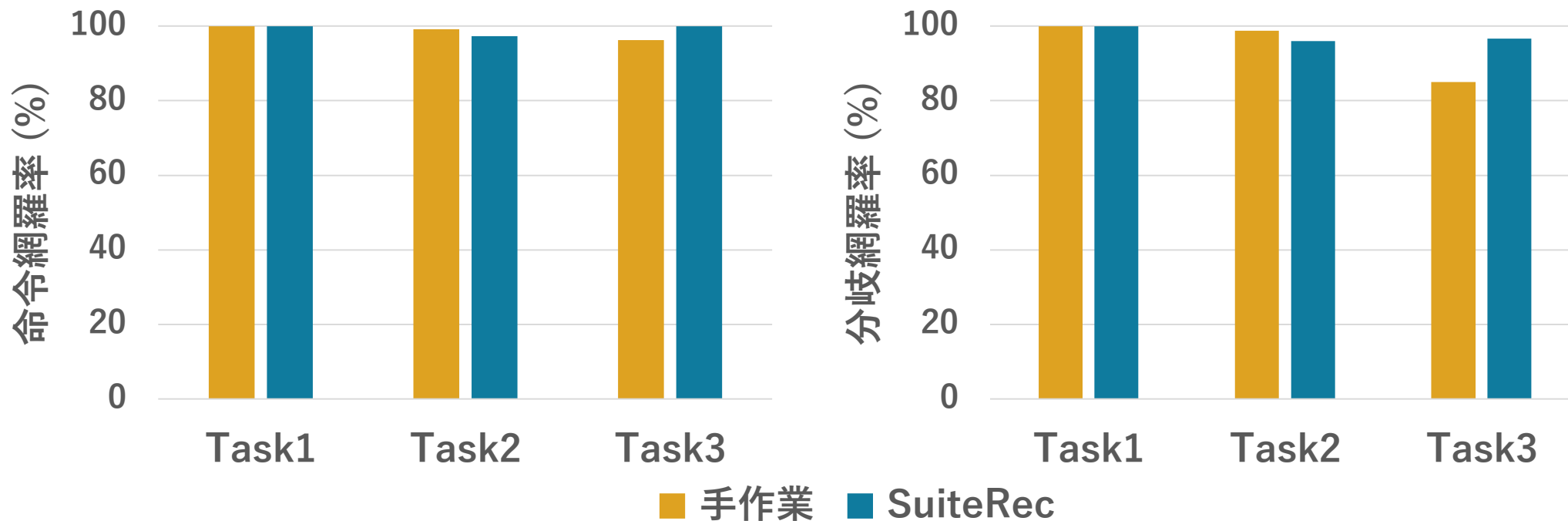
	Task1	Task2	Task3
概要	典型的なFizzBuzzの関数	第1引数に応じて計算方法を変更し, 計算結果を返す	2つの入力値に基づいて試験の合否を判定する
分岐数	8	16	24

- SuiteRecを使用した場合とそうでない場合で被験者が作成したテストコード比較する
- 実験後にテスト作成タスクに関するアンケートに回答してもらった

リサーチクエスチョン(RQ)

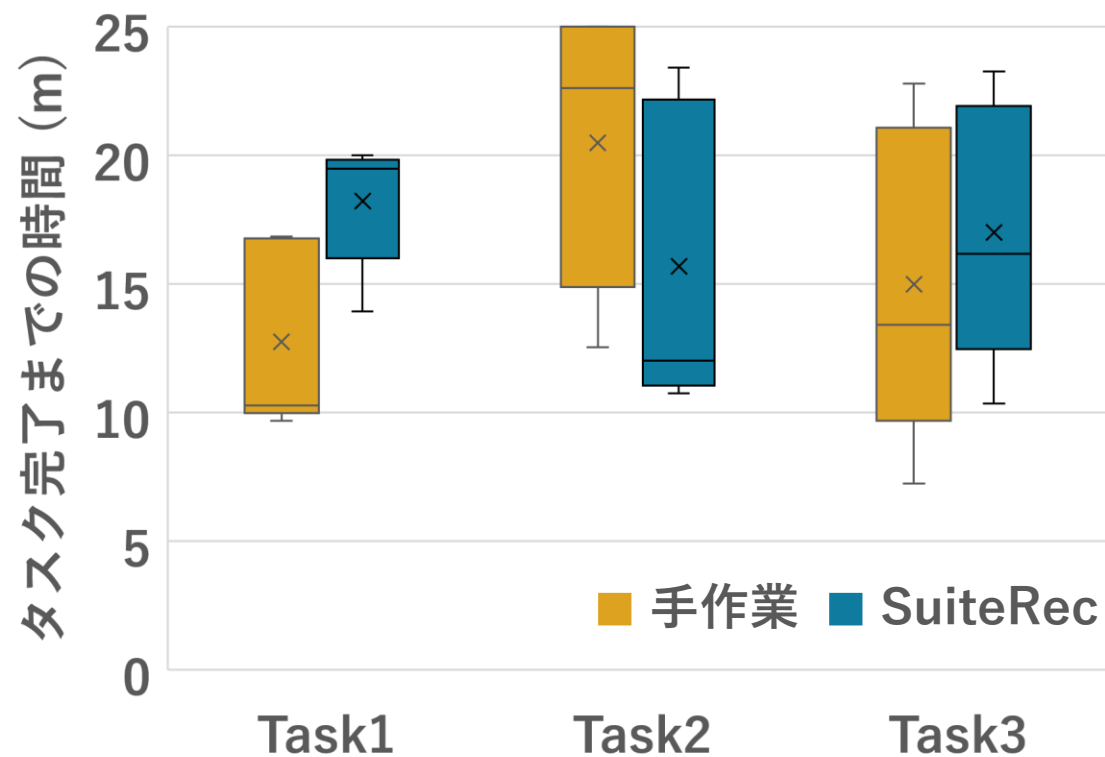
- RQ1.** SuiteRecは、高いカバレッジを持つテストコードの作成を支援できるか？
- RQ2.** SuiteRecは、テストコード作成時間を削減できるか？
- RQ3.** SuiteRecは、テストスメルの数が少ないテストコードの作成を支援できるか？
- RQ4.** SuiteRecの利用は、開発者のテストコード作成タスクの認識にどう影響するか？

RQ1. SuiteRecは、高いカバレッジを持つ テストコードの作成を支援できるか？



条件分岐が多いプログラムのテストコードを作成する際
カバレッジ(C1)を向上するのに役立つ可能性がある

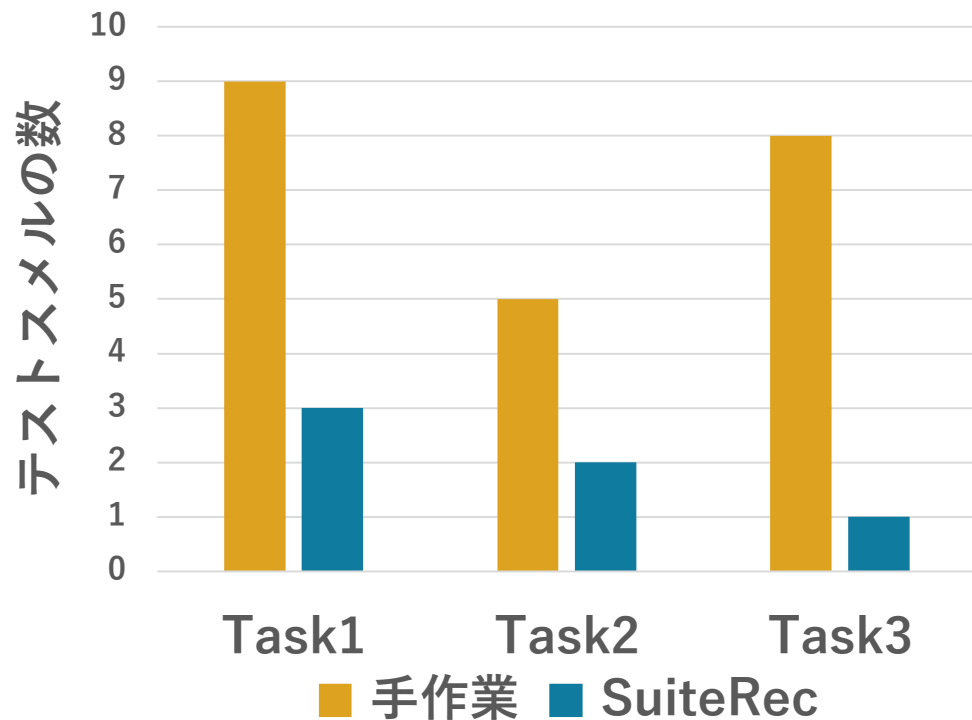
RQ2. SuiteRecは、テストコードの作成時間を削減できるか？



- Task1とTask3は、SuiteRecを使用した場合、タスク終了までの時間が長い
- Task2はSuiteRecを使用しない場合の方がタスク終了までの時間が長い
- 多くの被験者は無駄なテストを作成するのに時間を費やした可能性がある

推薦されたコードを理解し編集が必要なので、開発者はテストコード作成に多くの時間を費やす可能性がある

RQ3. SuiteRecは、テストスメルの数が少ない テストコードの作成を支援できるか？



- すべてのタスクにおいて、SuiteRecを使用した場合検出されたテストスメルの数が少ない
- 多く検出されたテストスメル
 - Assertion Roulette
 - Default Test
 - Eager Test

開発者は、推薦される高品質のテストスイートを参考にする
ことで品質の高いテストコードを作成できる

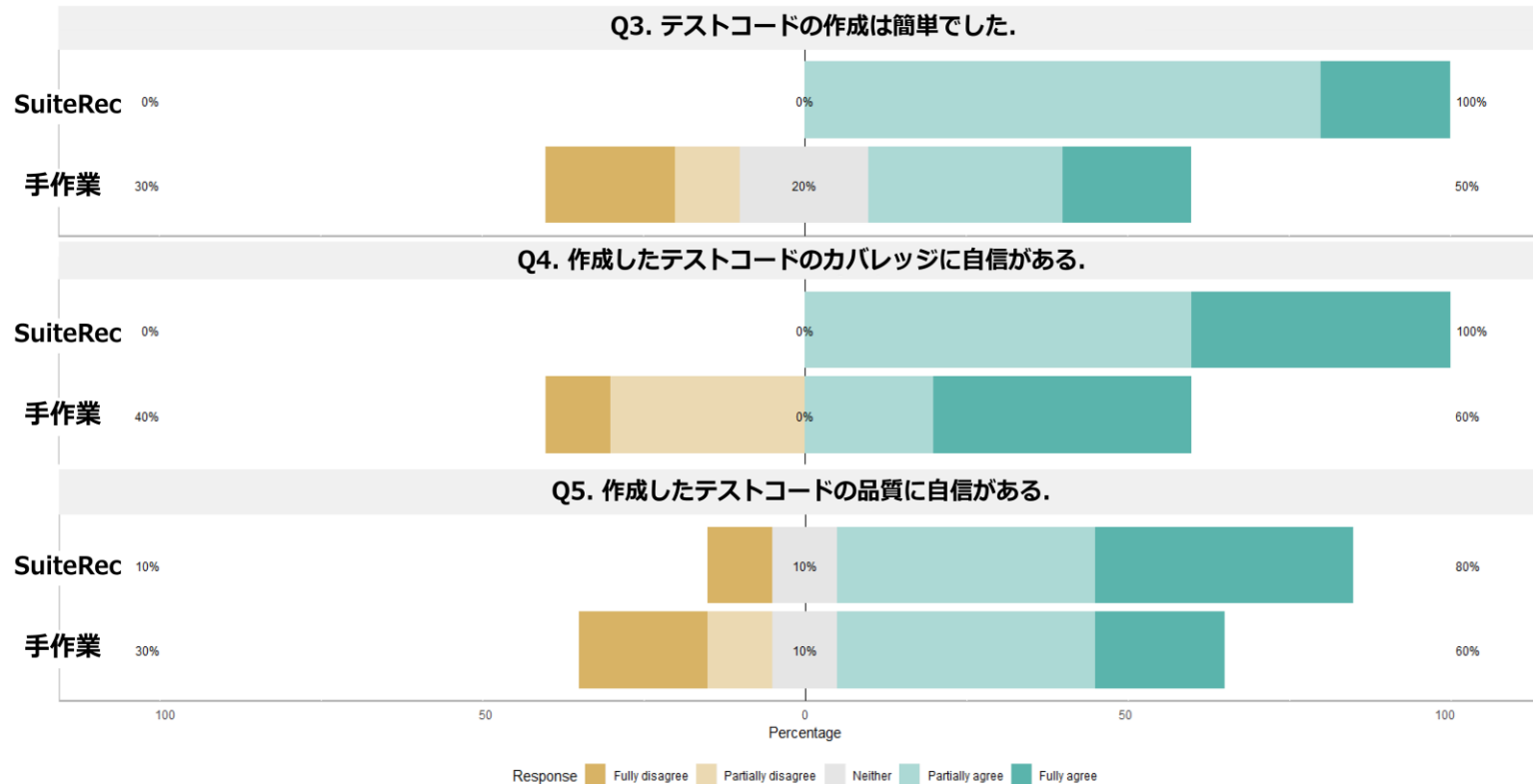
RQ4. SuiteRecの利用は、開発者のテストコード作成タスクの認識にどう影響するか？

- 被験者に実験タスク終了後にアンケートを実施した

	項目
Q1-a	テストコードの作成は簡単でした(ツールあり)
Q1-b	テストコードの作成は簡単でした(ツールなし)
Q2-a	作成したテストコードのカバレッジに自信がある(ツールあり)
Q2-b	作成したテストコードのカバレッジに自信がある(ツールなし)
Q3-a	作成したテストコードの品質に自信がある(ツールあり)
Q3-b	作成したテストコードの品質に自信がある(ツールなし)

※5段階評価：強く反対・反対・どちらでもない・賛成・強く賛成

RQ4. SuiteRecの利用は、開発者のテストコード作成タスクの認識にどう影響するか？



SuiteRecを利用した場合

- 被験者は、テストコードの作成を容易に感じるが、タスク完了までの時間は長くなる(RQ2)
- 多くの被験者は、作成したテストコードのカバレッジ・品質に自信が持てる

SuiteRecを利用した場合、開発者はテスト作成タスクを容易だと認識し、作成したテストコードに自信が持てる

まとめ・今後の課題

・まとめ

- ・ 類似コード検出技術を用いて、既存の高品質のテストコードを推薦するツールを提案
- ・ 提案ツールの有用性を定量的・定性的に評価

・今後の課題

- ・ より実用的な利用に備えてツールを改善
- ・ 被験者数を増やした更なる評価実験の実施
- ・ 複数の類似コード検出ツールに対応するようにツールを拡張する

補足資料

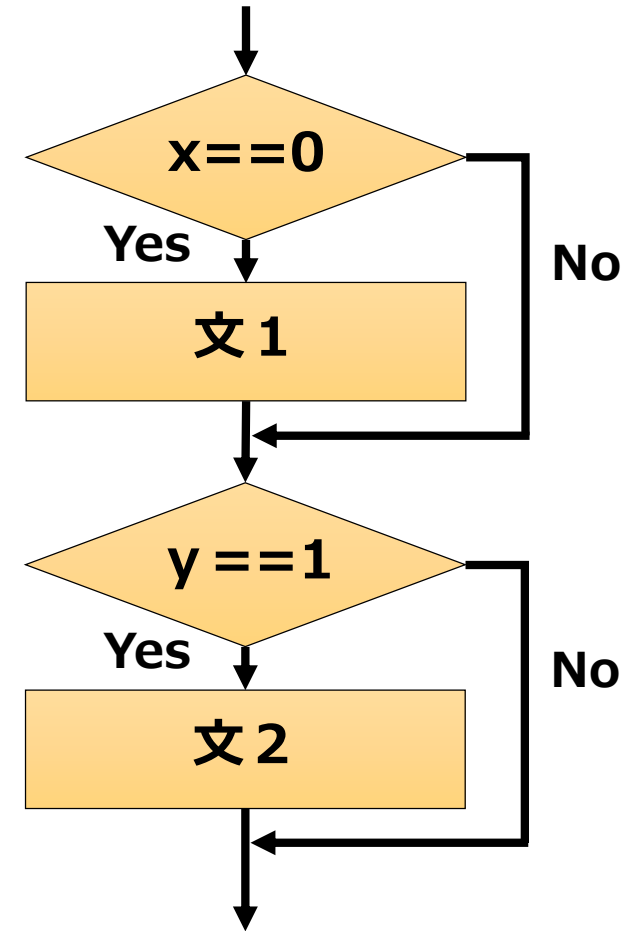
カバレッジの種類

● 命令網羅 C0

- 全ての命令を最低一回実行
- フローチャートの全節点通過
 - ✓ ($x=0$, $y=1$) を入力

● 分岐網羅 C1

- 全ての条件分岐について、then、else いずれも最低一回以上実行
- フローチャートの全辺通過
 - ✓ ($x=0$, $y=1$) と ($x=1$, $y=2$) を入力



ソースコードデータベース

Github上に存在する3,205個のOSSプロジェクト のプロダクションコード

既存のコード検索エンジン[7]で利用されたデータセットの中から、以下の条件を満たすプロジェクトを選択

- テストフォルダが存在する
- JUnitのテストフレームワークを採用している

[7] K. Kim, D. Kim, T. F. Bissyand ´ e, E. Choi, L. Li, J. Klein, and Y. Le Traon. Facoy - a code-to-code search engine. In Proceedings of the International Conference on Software Engineering (ICSE), pages 946–957, 2018.

テストコードデータベース

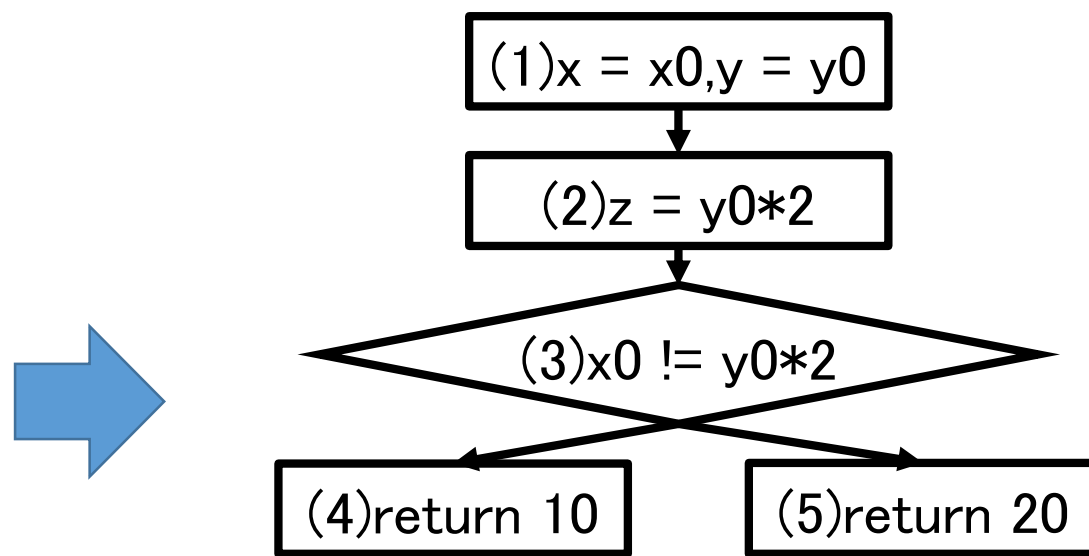
ソースコードデータベースに格納されている プロダクションコードに対応するテストコード

再利用対象のテストコードとして相応しくない、以下の
テストスメルを含むテストコードはTDBから除去される


- Empty Test
- Ignored Test
- Redundant Assertion
- Unknown Test

- 1.対象コードを静的解析し，プログラムを記号値で表現する
- 2.コード上のそれぞれのパスに対応する条件を抽出
- 3.パスごとにパスを通るような条件を集め，条件を満たす具体値を生成

```
public int
  testme(int x,int y)
{
  int z = y*2;
  if(x != z){
    return 10;
  }else{
    return 20;
  }
}
```



パス番号	パス条件	パス
1	$x0 \neq y0 * 2$	(1), (2), (3), (4)
2	$!(x0 \neq y0 * 2)$	(1), (2), (3), (5)

- RQ1から、単純な構造のプログラムのテストコードを作成する場合、SuiteRecの利用の有無でカバレッジに差がない
- 
- RQ2から、SuiteRecを利用せずにテストコード作成した方が、開発時間を節約できる
-
- 複雑な構造のプログラムのテストコードを作成する場合、SuiteRecを使用するとカバレッジ(C1)を向上することができる

- 類似コード間のテスト再利用
 - Zhang[1]らは、クローンペア間でコードを移植を行い、移植前と移植後のテスト結果を比較しその情報を基にテストを再利用するツールGrafterを提案した
 - Sohaら[2]は、開発者が詳細な再利用計画を決めることで、コード片を再利用する際に、テストスイートの関連部分を半自動で再利用および変換を行うツールSkipperを提案した

SuiteRecは、既存ツールと2つの視点で異なる

- OSS上からテストコードを検出することができる
- クローンペア間のテスト再利用計画は開発者に委ねていること

[8] T. Zhang and M. Kim. Automated transplantation and differential testing for clones. Proc. of ICSE, pages 665–676, 2017.

[9] S. Makady and R. Walker. Validating pragmatic reuse tasks by leveraging existing test suites. Software: Practice and Experience, 43:1039–1070, 2013.