

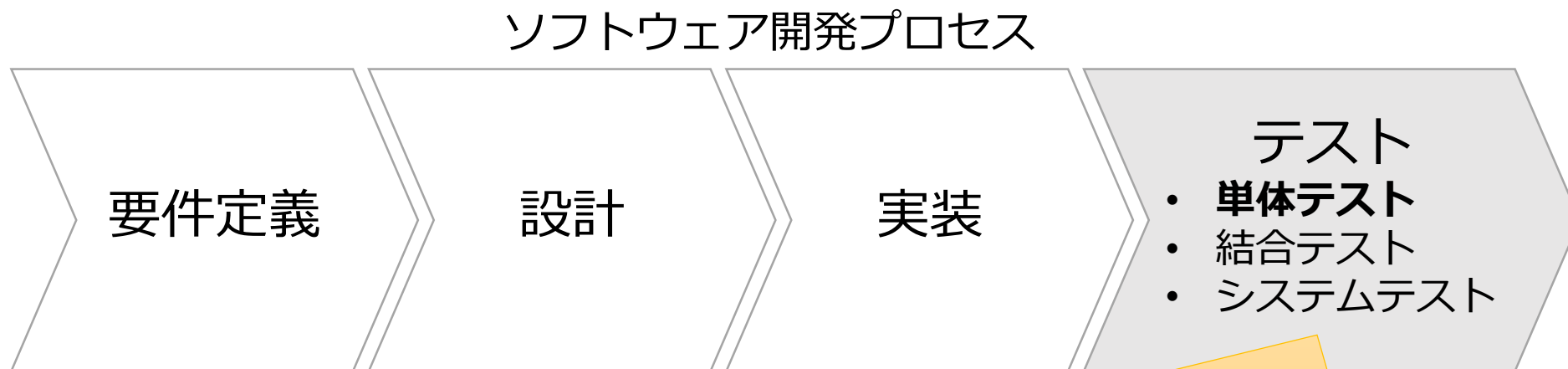
2020 修論発表会

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



開発全体のコストの内、30%~50%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

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
- 開発者は自動生成されたコードを信用していない



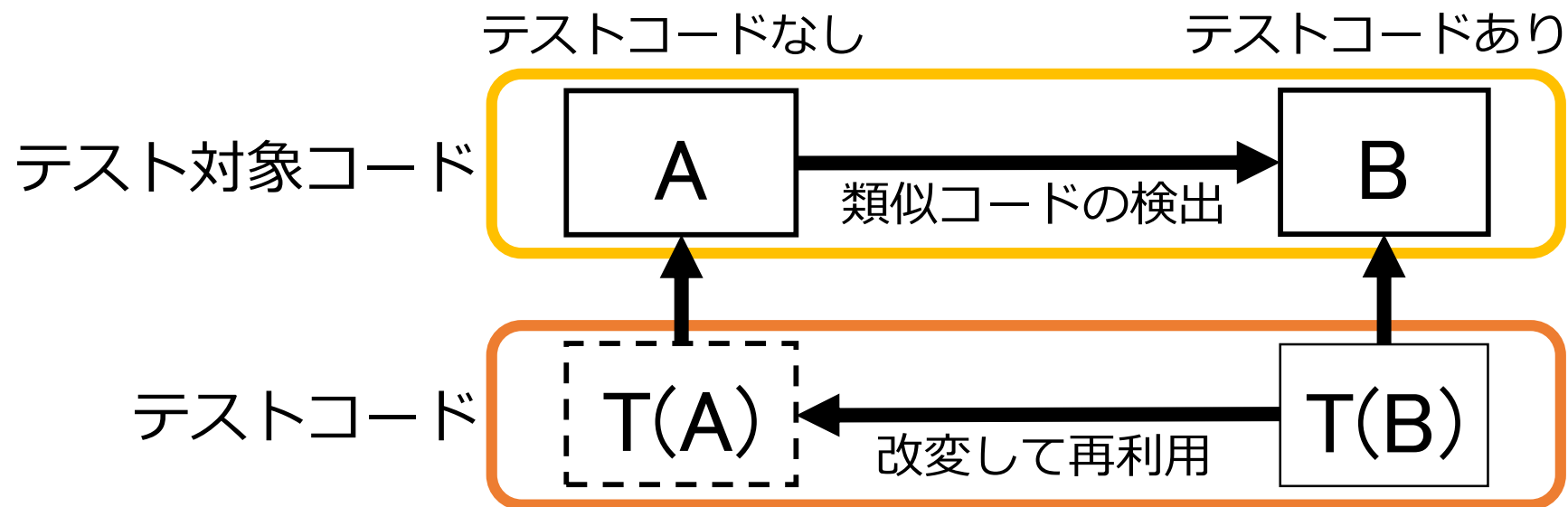
テスト失敗の原因を特定するのが難しい

理解しやすく良質なテストコードを作成する必要がある

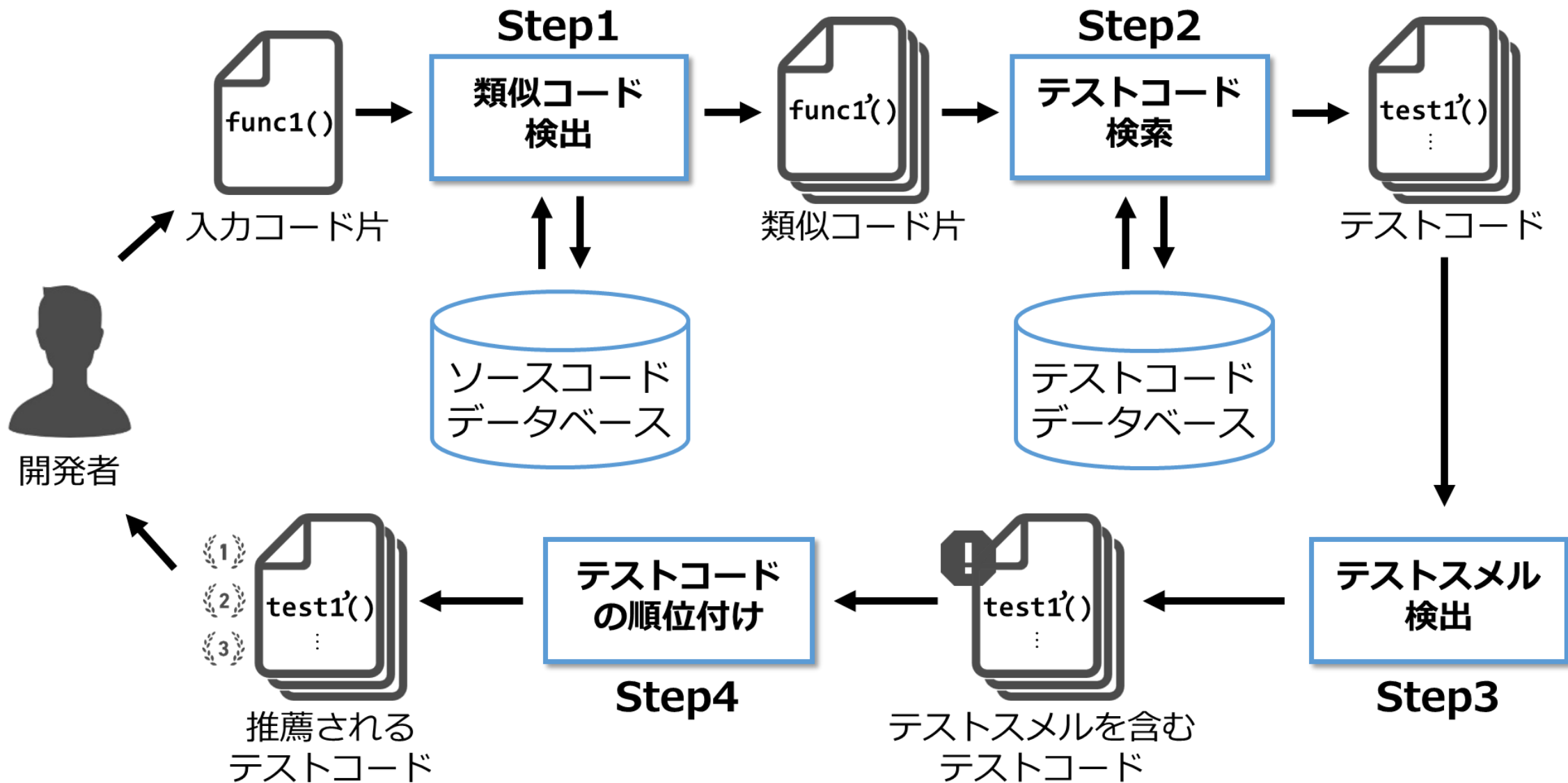
[2] S. Shamshiri, J. M. Rojas, J. P. Galeotti, N. Walkinshaw, and G. Fraser. How do automatically generated unit tests influence software maintenance? In Proceedings of the International Conference on Software Testing, Verification and Validation (ICST), pages 250–261, 2018.

提案ツール: SuiteRec

- **SuiteRec**: OSSに存在する高品質のテストコードを推薦する
 - 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
- **アイディア**: 類似するコード間でテストコードを再利用



SuiteRecの概要

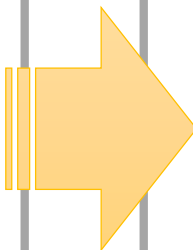


Step1: 類似コード片の検出

- 類似コード検出ツール: NiCad[3]
 - 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

[3] Chanchal, K. R. and James, R. C.: NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization, Proc. of ICPC 2008, pp. 172–181 (2008).

Step2: テストコードの検索

- テストコードと対象コードの対応付け
 1. 命名規則によるテストクラスと対象クラスの対応付け
 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テストメソッドと対象メソッドの対応付け

<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>

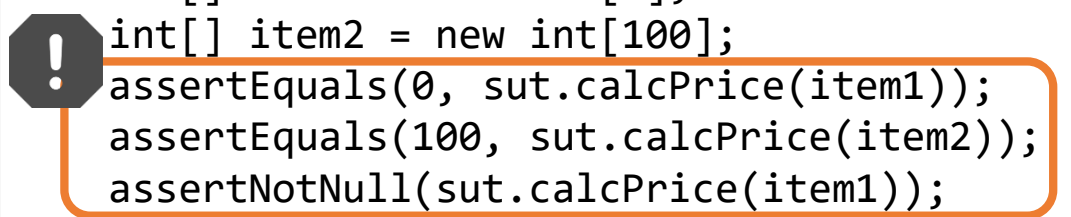
```
@Test  
public void testCalcPrice() throws Throwable{  
    CalcPrice sut = new CalcPrice();  
    int[] item1 = new int[0];  
    int[] item2 = new int[100];  
    assertEquals(0, sut.calcPrice(item1));  
    assertEquals(100, sut.calcPrice(item2));  
    assertNotNull(sut.calcPrice(item1));  
}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

テストコード

Step3: テストスメルの検出

- テストコードの良くない実装を表す指標
 - Deursenら[4]は、11種類のテストスメルを提唱した(現在19種類)

```
@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));
    assertNotNull(sut.calcPrice(item1));
}
```



テストスメルの例: **Assertion Roulette**

テストスメル検出ツール: tsDetect[5]

- 19種類のテストスメルを検出可能
- 各テストスメルを高精度で検出可能

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

[5] 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: 推薦されるテストコードの順位付け

- 推薦されるテストコードを以下の2つの要素を基に順位付ける

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

テストコード内に、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付ける

SuiteRecの有用性を定量的・定性的に評価

評価実験1: テストコードの作成支援に関する実験

- 被験者が作成したテストコードのカバレッジ、作成時間、テストスメルを比較して評価した

評価実験2: 推薦されるテストコードの順位付けに関する実験

- アンケート調査を実施し、開発者が参考にしたいテストコードを上位に推薦できるかを評価した

※本発表では、時間の都合上紹介されません

評価実験1

- 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

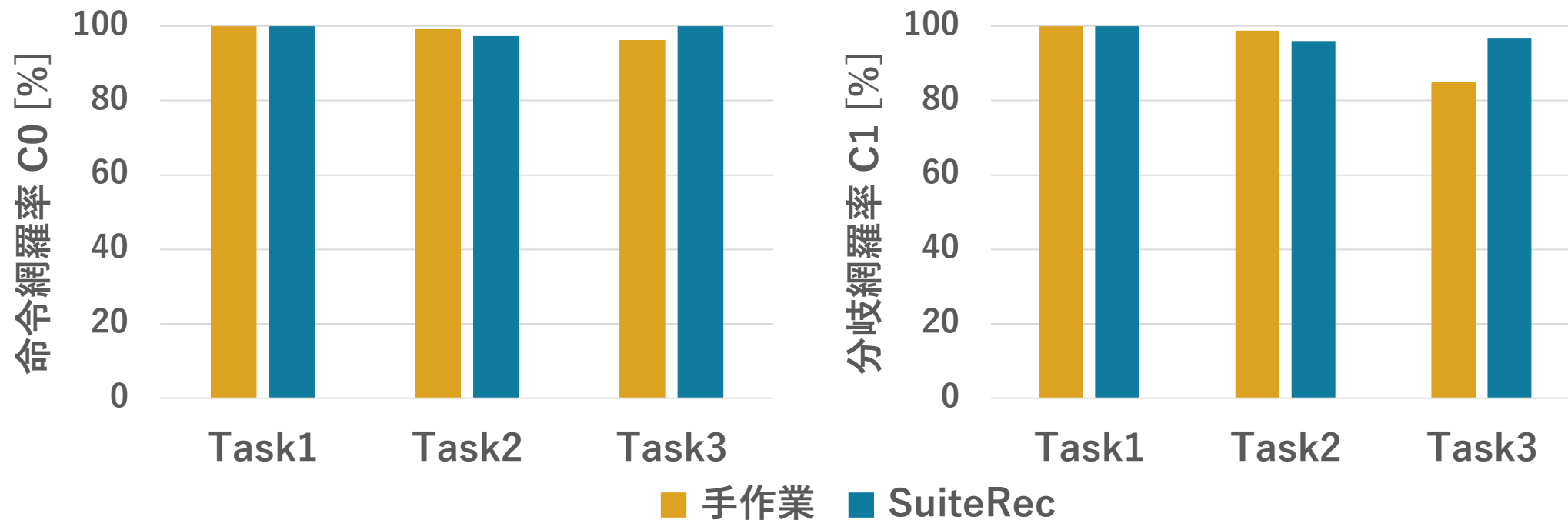
	Task1	Task2	Task3
プログラムの概要	典型的なFizzBuzzの関数	第1引数に応じて計算方法を変更し, 計算結果を返す	2つの入力値に基づいて試験の合否を判定する
条件分岐数	8	16	24

- SuiteRecを使用した場合と手作業の場合で、被験者が作成したテストコードを比較し評価する
- 実験後にテストコード作成タスクに関するアンケートを実施した

リサーチクエスチョン(RQ)

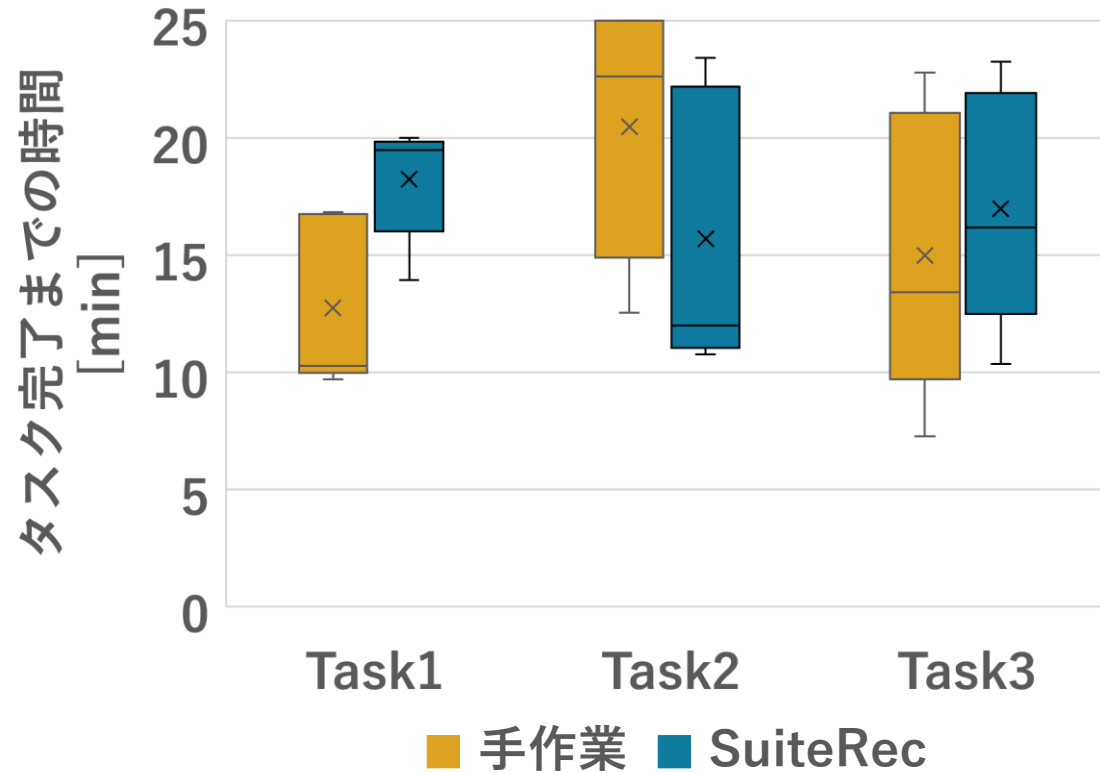
- RQ1.** SuiteRecは、高いカバレッジを持つテストコードの作成を支援できるか？
- RQ2.** SuiteRecは、テストコード作成時間を削減できるか？
- RQ3.** SuiteRecは、テストスメルの数が少ないテストコードの作成を支援できるか？
- RQ4.** SuiteRecの利用は、開発者のテストコード作成タスクの認識にどう影響するか？

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？



条件分岐が多く複雑なプログラムのテストコードを作成する際、カバレッジ(C1)の向上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？

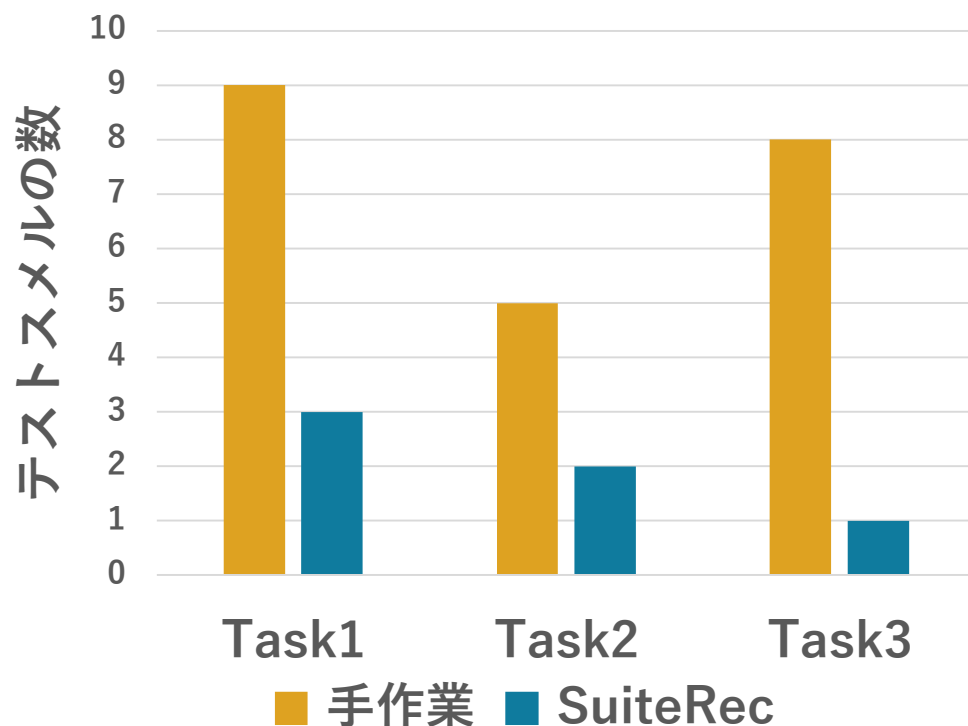


<SuiteRecを利用した場合>

- タスク1と3は、タスク完了までの時間が長くなる
 - 原因: 推薦された複数のテストコードを理解し、変更する必要がある
- タスク2は、タスク完了までの時間が短くなる
 - 手作業の場合、テスト項目の重複が多い

開発者はテストコード作成に多くの時間を費やす

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？



<SuiteRecを利用した場合>

- すべてのタスクにおいて、検出されたテストスメルの数が少ない

<手作業の場合>

- 多く検出されたテストスメル
 - Assertion Roulette
 - Default Test
 - Eager Test
- 既存研究でも、これらのテストスメルがプロジェクト内で多く検出されたと報告

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

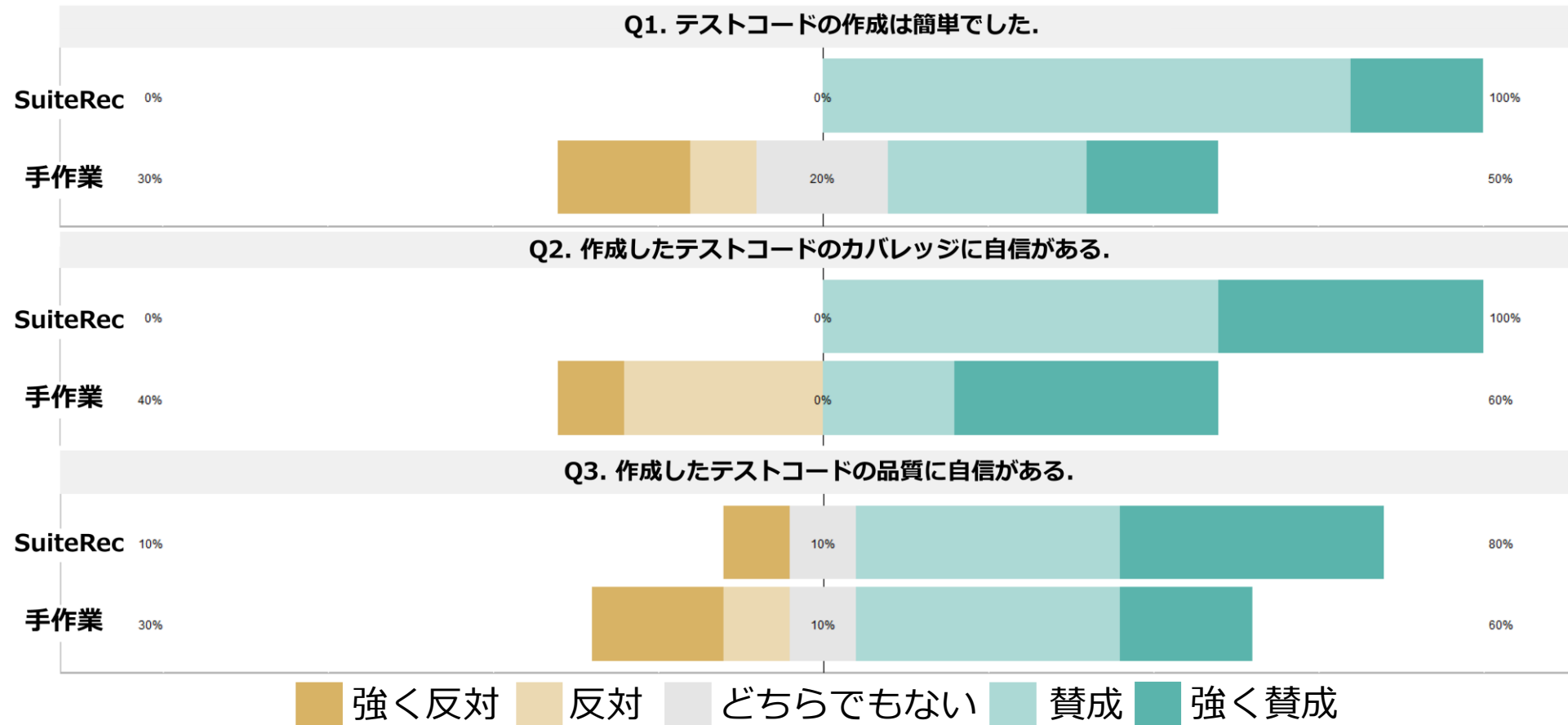
RQ4. SuiteRecの利用は、開発者のテストコード作成タスクの認識にどう影響するか？

- 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？



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
- ・ 提案ツールによって、開発者の高品質なテストコードの作成を支援

・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

ソフトウェア開発にかかる費用

- ソフトウェア開発各工程での費用

	コストの割合	「運用と保守」を除いた割合
要求分析	3%	9%
仕様書	3%	9%
設計	5%	15%
コーディング	7%	21%
テスト	15%	46%
運用と保守	67%	—

「基本から学ぶソフトウェアテスト」 Cem Kaner, Jack Falk, Hung Quoc Nguyen 著,
テスト技術者交流会訳, 日経BP社, ISBN4-8222-8113-2より

推薦プロセスの高速化

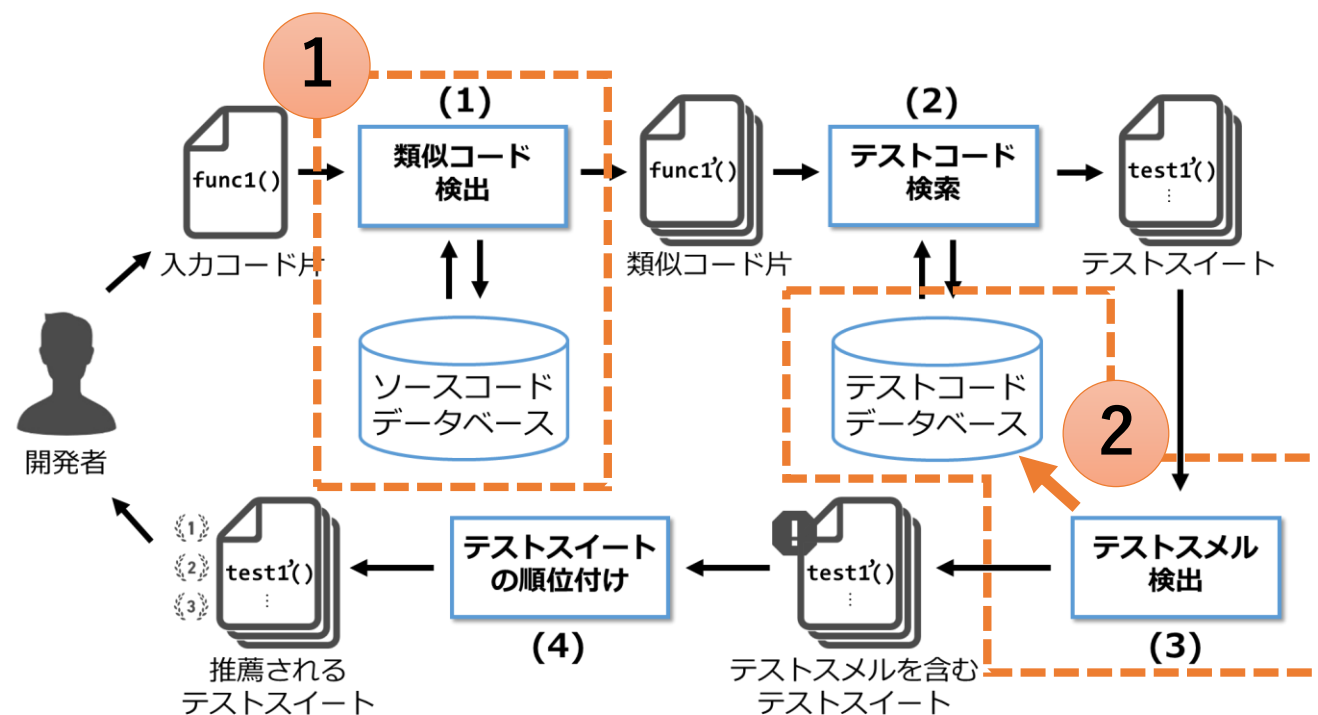
- 類似コード検索を複数並列に実行(Step1)
- データベースに必要なデータを事前に格納(Step3)

Step1

- 前処理としてプロジェクトのサイズを調整して格納
- 類似コード検索処理を複数並列に実行

Step3

- 事前にテストスメルを検出
- テストスメルの情報をテストコードに紐づけて格納



RQ2. SuiteRecは、テストコードの作成時間を削減できるか？

	Task1		Task2		Task3	
	手作業	SuiteRec	手作業	SuiteRec	手作業	SuiteRec
カバレッジ (C0)	100	100	99.2	97.3	96.2	100
カバレッジ (C1)	100	100	98.8	96.0	85.0	96.7
テスト項目 の数(平均)	6	6	24	11	16	12

タスク2は、カバレッジに差はないものの手作業の場合、テスト項目の数が多く無駄なテストコードを作成している

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>					

5

- 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** テストコード

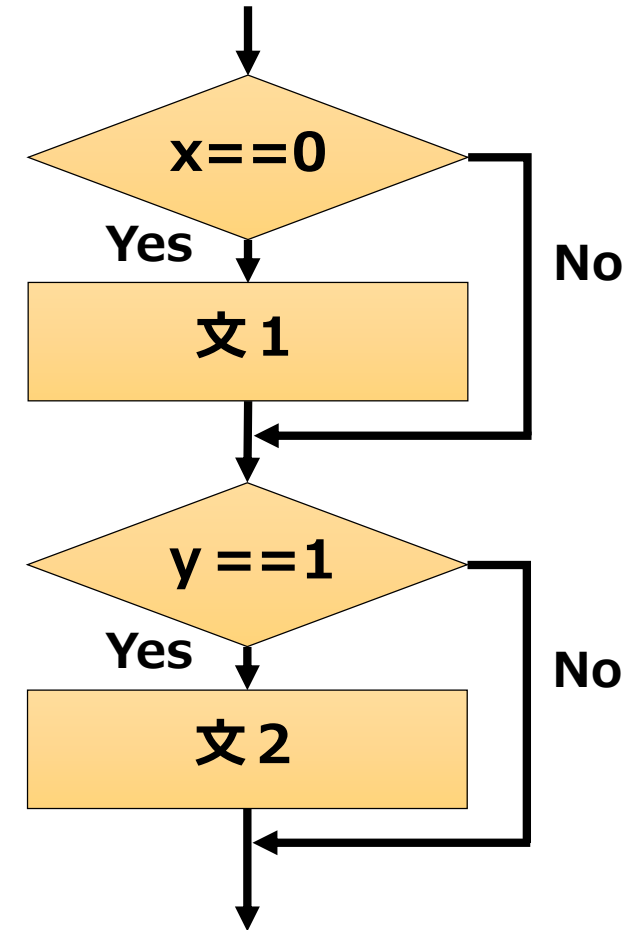
カバレッジの種類

● 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

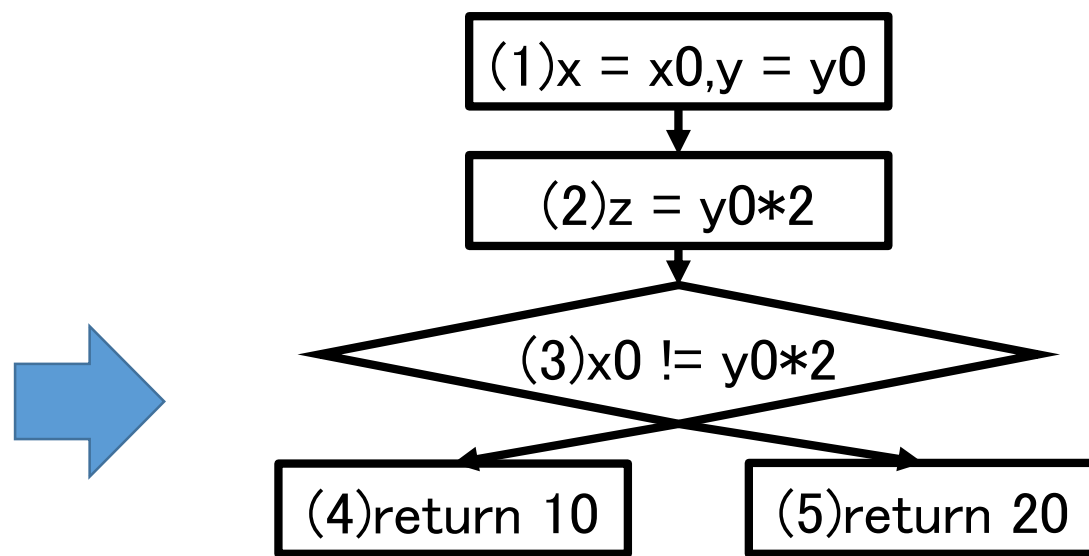
ソースコードデータベースに格納されている プロダクションコードに対応するテストコード

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.