

SuiteRec: Automatic Test Suite Recommendation System Using Code Clone Detection Tool

*Note: Sub-titles are not captured in Xplore and should not be used

1st Ryosuke Kurachi

Information Science
Nara Institute of Science and Technology
Nara, Japan
kurachi.ryosuke.kp0@is.naist.jp

2nd Eunjong Choi

dept. name of organization (of Aff.)
Kyoto Institute of Technology
Kyoto, Japan
echoi@kit.ac.jp

3rd Given Name Surname

dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname

dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

5th Given Name Surname

dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

6th Given Name Surname

dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Abstract—It is important to support software testing, which is the key to ensuring software quality. So far, various automatic generation techniques have been proposed to reduce test creation costs. However, there is a problem that makes it difficult to carry out maintenance activities due to the fact that automatically generated test code is not generated based on the background and intention of creating the test target code. Ask questions about value. In this research, we introduce SuiteRec, a tool that recommends existing high-quality test codes on OSS to solve this problem. SuiteRec considers test reuse between clone pairs using a similar code search tool. A similar code is detected from the input code, and a test suite corresponding to the similar code is recommended to the developer. In addition, the recommendation ranking is ranked so that the test sumel, which is a metric representing the poor implementation of the test code, is presented to the developer and a higher quality test suite can be recommended. In the evaluation of the proposed tool, the test code was created depending on whether or not SuiteRec was used by the subject, and the degree of support for test creation was evaluated quantitatively and qualitatively. As a result, (1) it is effective in improving code coverage when creating test code for programs with many conditional branches, and (2) the test code created using SuiteRec is the number of detected test smells. (3) Developers were more confident in their own test code than when they were not using SuiteRec when it was created using SuiteRec.

Index Terms—clone detection, recommendation system, software testing, unit test

I. INTRODUCTION

In recent years, the requirements for software have become more sophisticated and diversified, while the demand for software quality assurance and cost reduction from users has also increased [1]. Among them, the proportion of the total development cost is large, and interest in technology that supports software testing, which is the key to ensuring quality,

is increasing. However, at present, most of the unit test creation work is done manually, and if you try to create many tests, the cost will increase accordingly. Against this background, various automation technologies have been proposed to achieve cost reduction while ensuring software quality.

EvoSuite [2] proposed in existing research is the most advanced tool for automatic unit test generation. EvoSuite statically analyzes the target code and expresses the program as a symbolic value. Then, conditions that pass the control path of the target code are collected, and concrete values that satisfy the conditions are generated. By automatically generating unit tests, developers can save manual creation time by automatic generation and improve code coverage. However, the test code that is automatically generated by existing tools is not readable because of the fact that it is not generated based on the creation process and intention of the target code, and it is not trusted by the developer and makes subsequent maintenance work difficult. There is a problem [3]. This raises questions about the value of practical use of auto-generation tools. Each time a test fails, the developer must identify the cause of the defect in the program under test or determine whether the test itself needs to be updated. It has been reported that auto-generated tests are more difficult to read than the time savings gained by auto-generation and interfere with rather than help maintenance work.

In this research, we introduce SuiteRec, a tool that recommends existing high quality test codes that exist in OSS to solve this problem. SuiteRec considers test reuse between clone pairs using a similar code detection tool. A similar code is detected from the input code, and a test suite corresponding to the similar code is recommended to the developer. In addition, the recommendation ranking is ranked so that the test sumel, which is a metric representing the poor implementation

Identify applicable funding agency here. If none, delete this.

of the test code, is presented to the developer and a higher quality test suite can be recommended.

In the evaluation of the proposed tool, the test code was created depending on whether the subject used SuiteRec or not, and the degree of support for test creation was evaluated quantitatively and qualitatively. As a result, it was found that the use of the proposed tool can improve the code coverage when creating a test suite for a complex program with many branches, and the quality of the test code created using the tool is high. In addition, a questionnaire was conducted after the experiment as a qualitative evaluation, and it was found that when using the recommendation tool, many subjects were confident in the test code they created.

II. BACKGROUND AND RELETED WORK

Unit testing. In the unit test execution task, the software is run, and it is confirmed whether the software behaves as expected in each test case. In order to reduce the cost of the test process, in test execution tasks, the use of automatic test execution tools such as JUnit is advancing in the industry for unit tests. However, test design tasks are still often performed manually, and the practical application and popularization of automation technology is expected.

Test case generation. 既存の研究 [4] は、既存のテストケースを再利用、自動生成、または再適用できることによって、ソフトウェア開発のテスト工程における時間とコストを大幅に節約できることを示している。テスト生成技術は、主にランダムテスト (RT)、記号実行 (SE)、サーチベーステスト (SBST)、モデルベース (MBT)、組み合わせテストの5つに分類できる。SE はさらに静的記号実行 (SSE) と動的記号実行 (DSE) に分けられる。

RT とは、ソフトウェアにランダムな入力を与えるテスト手法である。無造作・均一にテストを実行するランダムテストは自動化に適しているが、コードカバレッジ率向上、バグ検出の観点において、テストケース 1 件当たりの効率は著しく悪い。

SE は対象コードを静的解析してプログラムを記号値で表現し、コード上のそれぞれのパスに対応する条件を抽出し、パスごとにパスを通るような入力値が満たすべき条件を集める。そして、パスごとにその条件を SMT ソルバ [5] などの制約ソルバを用いて解き、得られた具体値をテスト入力値とする。

SBST は、達成したい要件に対する達成度合いを定量的に評価できるように設計した評価関数に基づいて、ヒューリスティック探索アルゴリズムを用いて達成したい要件を満足するテストスイートを生成する技術の総称である。

MBT はモデルに基づいてテストスイートを生成する技術の総称である。モデルは何らかの形でテスト対象を記述したものであり、要求分析や設計のためのモデルを活用することもあれば、テストのためにモデルを作成することもある。

CT は、パラメータ間の相互作用に起因する不具合を効果的に発見するためにテストケースとしてパラメータに割り当てる値の組み合わせを生成する手法である。

Test Smell. プロダクションコードだけでなく、テストコードの適切なプログラミングの慣習に従って設計する必要があります [44]。テストコードの適切に設計するこ

との重要性は元々 Beck [7] によって提唱されました。さらに、Van Deursen ら [50] は 11 種類のテストスメルのカタログ、すなわちテストコードの良くない設計を表す実装とそれらを除去するためのリファクタリング技術を定義しました。このカタログはそれ以降、18 個の新しいテスト臭を定義した Meszaros [42] によってより拡張されました。最近の研究では、テストスメルの存在は開発者のテストスイートの理解に悪い影響を与えるだけでなく、テストコードがプロダクションコード内の不具合を見つけるのにあまり効果的でなくなると言われています。

III. SUITEREC

SuiteRec は、開発者からの関数単位のコード片を入力とし、その入力コードの類似コードを検索します。そして類似コードに対応するテストスイートを優先順位の高い順に並び替え開発者に提示します。図 1 は、SuiteRec によってテストスイートが推薦されるまでの流れを示しています。推薦手法は、主に以下の 4 つのステップから構成されます。

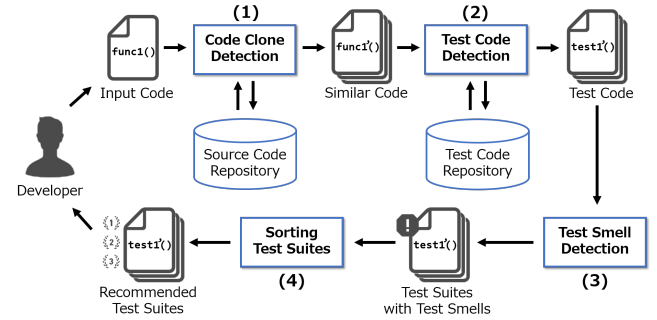


Fig. 1. Overview of SuiteRec.

- (1) SuiteRec は、入力されたコード片を受け取ると、そのコード片をコードクローン検索ツールにかけ入力コードの類似コードを検出します。
- (2) 複数の類似コード片が検出されると、次にその類似コード片に対応するテストスイートをテストコードリポジトリ内から検索します。
- (3) 各類似コード片のテストスイートが検出されると、次にそれらをテストスメル検出ツールにかけ各テストスイートに含まれるテストスメルを検出します。
- (4) 最後に、1 で得られた類似コードと入力コードの類似度と 3 で検出されたテストスメルの数を基に出力されるテストスイートの順番がランキングされる。

A. Code Clone Detection

本研究では、類似コード検出ツールとして NICAD [3] を採用した。NICAD は検索対象のコードフラグメントのレイアウトを統一的に変換させ、行単位で関数単位のコードフラグメントを比較することで、クローンペア検出するツールであり、このような手法を取ることで、高精度・高再現率でのクローンペアの検出を実現している。

NICAD は入力コード対応する類似コードを大規模なオープンソースプロジェクトをホストしている Github のリポジトリから検索します。図 1 のソースコードリポジトリは、テストコードが存在する Github プロジェクトが格納されてい

ます。具体的には、プロジェクト内にテストフォルダが存在し、JUnitのテストフレームワークを採用しているプロジェクトを選択しました。NICADは、一度に検索できるプロジェクトの規模限度があります。我々は、検索時間を短縮するために大規模なプロジェクトは分割し、小規模なプロジェクトは統合させた状態で検索処理を複数並列して走らせることで現実的な時間での類似コードの検索を実現しました。また、検出設定についてはNICADの標準設定で提案ツールに実装されている。

B. Test Code Detection

類似コード片から対応するテストコードを検索するためにテスト対象コードとテストコードの対応付けを行う。本研究では、厳密にテストコードと対象コードを対応付けるために以下の2つのステップを踏みます。

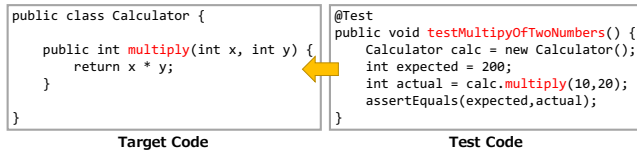


Fig. 2. Example of mapping test code to target code.

- (1) テストコードを静的解析し、メソッド呼び出しを確認
- (2) テストメソッドを区切り文字や大文字で分割し、対象メソッドと部分一致した時対応付ける

単体テストでは、図の例のようにテストコード内でオブジェクトの生成が行い、テスト対象コードのメソッド呼び出して実行されます。すなわち、テストコードリポジトリ内のテストコードを静的解析し、メソッド呼び出しを取得することで、テスト対象コードとテストコードを対応付けます。しかし、テストメソッド内では複数のメソッドが呼び出されていることも考えられるのでさらに、メソッド名の比較も行います。テストメソッド名の記述方法としてテスト対象メソッドの処理の内容を忠実に表すことが推奨されており、対象メソッドの名前が記述されていることが多い。したがって、テストメソッドの名前を区切り文字や大文字で分割し、対象メソッドと部分一致した場合、対応付けるように実装した。

図1のテストコードリポジトリには、ソースコードリポジトリ内のプロダクションコードに対応するテストコードが格納されています。我々は、テストコードの検索時間を短縮化するために前処理として事前に大規模なプロジェクトに対して静的解析を行い、プロダクションコードとテストコードの対応付けた情報をDBに保持し、DBを介して高速にテストコードを検索できるようにした。

C. Test Smells Detection

本研究では、テストスメル検出ツールとして tsDetect[6]を採用した。tsDetectはASTベースの検出手法で実装されたツールであり、19個のテストスメルを検出できるツールである。また、85%~100%の精度と90%~100%の再現率でテストスメルを正しく検出できることが報告されている。本研究では、tsDetectで検出できる19個のテストスメルの内テストコードの推薦を考える上で重要な以下の6種類のテストスメルを提示するように実装しました。

TABLE I
SUBJECT TEST SMELLS

Name	Description
Assetion Roulette	1つのテストメソッド内に複数のassert文が存在するテストコード。各assert文は異なる条件をテストするが、テストが失敗した場合開発者へ各assert文のエラーメッセージは提供されないで、失敗を特定することが困難になる。
Conditional Test Logic	テストメソッド内に複数の制御文が含まれているテストスイート。テスト成功・失敗は制御フロー内にあるassert文に基づく予測するのが難しい。
Default Test	JUnitなどのテストフレームワークを使用したテストコードの内、テストクラスやテストメソッドの名前がデフォルトの状態であるテストコード。テストコードの可読性の向上のために適切な名前に変更する必要がある。
Eager Test	テスト対象クラス内の複数のメソッドを呼び出しているテストコード。1つのテストメソッド内で複数のメソッド呼び出しを行うと、正確に何をテストしているかについて混乱が生じる。
Exception Handling	テストメソッド内で例外処理が含まれているまたは例外を投げるテストコード。例外処理はプロダクションコードに記述し、テストコード内で例外処理が正しく行われるかどうかを確認するようにリファクタリングする必要がある。
Mystery Guest	テストメソッド内で、外部リソースを利用するテストコード。テストメソッド内だけで完結せず外部のファイルなど、外部リソースを利用すると外部との依存関係が生じ、外部リソースが壊れた場合テストも失敗してしまう。

また、前処理として推薦テストコードとしてふさわしくない以下の4つのテストスメルを含むテストコードを事前にテストコードリポジトリから削除し、推薦されるテストコードでとして出力されないようにした。

- **Empty Test.** テストメソッド内にテストの記述はなくコメントのみが含まれているテストコード
- **Ignored Test.** @Ignore アノテーションがあり、実行されないテストコード
- **Redundant Assertion.** 必ずテストが成功する意味のないテストコード
- **Unknow Test.** assertメソッドが存在しないテストコード

D. Sort Recommended Test Suites

入力コードと検出された類似コードの類似度とテストスイート内に含まれるテストスメルの数を基に推薦されるテストスイートの並び替えを行った。我々の以前の調査で、OSS上の有名プロジェクト内の両方のコードフラグメントにテストコードが存在するクローンペアを対象にプロダクションコードとなるクローンペアの類似度とそれに対応するテストコードの類似度を調査した。その結果、テストコードの間の類似度と対象のクローンペアの類似度には相関関係があり、プロダクションコードの類似度が高いほど、テストコード間の類似度も高いことが分かっている。したがって、入力コードと類似コード間の類似度が高いクローンペアほどテストコードの再利用がしやすいと考える。SuiteRecで

はこの結果を基に類似度が高いクローンの順に並び替えさらに類似度が同じだった場合、テストスメルの数で順番を決めるような推薦ランキングを実装した。

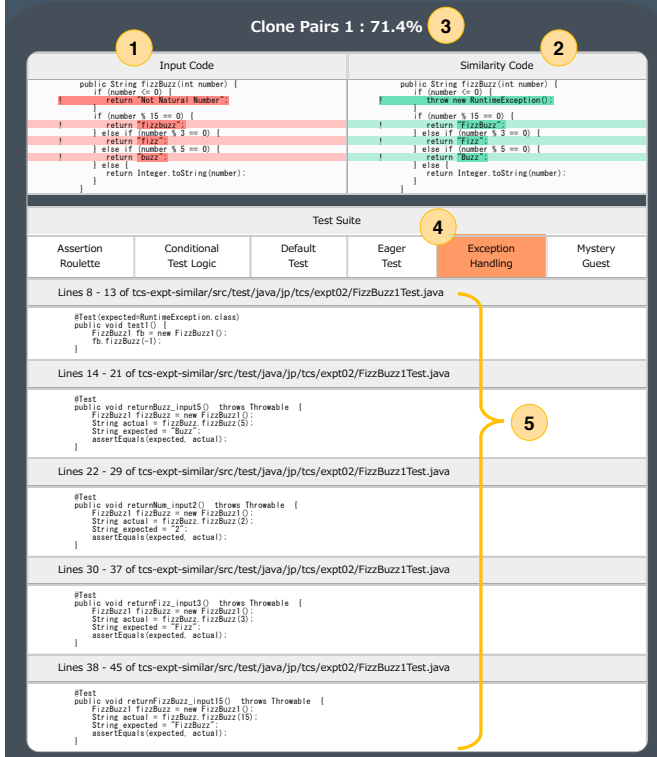


Fig. 3. Test suite recommended by SuiteRec.

- ① **Input Code.** 開発者が入力した関数単位のプロダクションコードが表示されます。
- ② **Similarity Code.** 入力コードに対する類似コードが表示されます。入力コードと類似度コードの違いが分かるように差分がハイライトされます。
- ③ **Degree of similarity.** 入力コードと類似コードの類似度が表示されます。類似度は NICAD で用いられている計算方法 Unique Percentage of Items(UPI) を採用しています。
- ④ **Test Smells.** 推薦されるテストスイート内にテストスメルが含まれている場合、そのテストスメルがオレンジ色にハイライトされ開発者にテストスメルの存在を提示させます。
- ⑤ **Recommend Test Suites.** 推薦されるテストスイートが表示されます。また、どのプロジェクトからテストコードが参照されたのかを示すためにファイルパスも表示されます。

IV. EVALUATION

このセクションでは、SuiteRec の定量的及び定性的に評価するために、被験者による実験を行います。被験者は、3つのプロダクションコードのテストコードを作成してもらいます。そして SuiteRec を使用して作成した場合とそうでない場合のテストコードを比較することで評価を行います。実験を通してコードカバレッジ、実験タスクを終了するま

での時間およびテストコードの品質に関するデータを収集することで、以下のリサーチクエストに答えることを目指します。

- **RQ1 : SuiteRec の利用は、開発者の作成したテストコードのカバレッジにどう影響するか？**ソフトウェアの品質を確認する1つの指標としてカバレッジは重要な要素である。テストコード内で一度も実行されない行が存在するとその部分の品質を確保することはできません。SuiteRec の利用は高いカバレッジを達成するために役に立つのでしょうか？
- **RQ2 : SuiteRec の利用は、開発者のテストコード作成時間に影響するか？**開発者は SuiteRec で推薦されるテストコードを参考にするすることで、テストコード作成時間を短縮化できるのか？
- **RQ3 : SuiteRec の利用は、作成したテストコードの品質にどう影響するか？**開発者は SuiteRec で推薦されるテストコードを参考にするすることで、品質の高いテストコードを作成することができるのか？
- **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しますか？**SuiteRec を利用した場合、テストコードの作成が容易になり、自分で作成したテストコードに自信が持てるのか？

A. Participant Selection

我々は、基本的なプログラミングスキルを保有し、ソフトウェアテストに理解がある情報系の修士の学生10人に対して行った。事前アンケートによると9割以上の学生が2年以上のプログラミング経験があり、8割以上の被験者が1年以上のJava言語の経験があった。また、すべての学生が授業などの講義でソフトウェアテストに関する基本的な知識を持っており、8割以上が単体テストの作成経験があった。

B. Object Selection

実験を行うために、3つのプロダクションコードを用意した。被験者はテストコードを作成するのでプロダクションコードの仕様を十分理解していることが前提になる。そこで、我々はプロダクションコードとして競技プログラミングをよく用いられる典型的な計算問題を選択した。また、そのプロダクションコードの仕様を確認できるように自然言語で書かれた仕様書を用意した。3つの各問題で違いを出すために問題1, 2, 3の順に条件分岐の数を8, 16, 24と多くなるように設定した。図4は、出題したプロダクションコードの一例である。実験後のアンケートで、実験タスクについての理解を確認したがすべての被験者が実験タスクの理解についてポジティブな意見を述べたことが分かっている。また、十分な実験時間があったかどうかに関する質問に対してもネガティブな回答はなかった。したがって、被験者は与えられた実験タスクに対して十分に理解し、作業時間も十分にあったことが分かる。

C. Experiment Procedure

まず初めにソフトウェアテストに関する基本的な知識からJUnitを使用に関する30分の講義と練習問題を実施し、テストコードの記述に対する理解を確認した。そして本番の実験課題の3つのプロダクションコードのテストコードを作成してもらった。実験タスクの終了は被験者に判断し


```

public class Experiment03 {
    public String returnResult(int score1, int score2){
        if((score1 < 0 || score2 < 0) || (score1 > 100 || score2 > 100)){
            return "Invalid Input";
        }else if( score1 == 0 || score2 == 0 ){
            return "failure";
        }else if( score1 >= 60 && score2 >= 60 ){
            return "pass";
        }else if((score1 + score2) >= 130){
            return "pass";
        }else if((score1 + score2) >= 100 && (score1 >= 90 || score2 >= 90)){
            return "pass";
        }else {
            return "failure";
        }
    }
}

```

Fig. 4. Example of a experimental task.

てもらう。具体的には、被験者自身が作成したテストコードのカバレッジ・品質に満足した時、実験タスクを終了してもらった。実験時間は1問につき最大25分の時間を設けた。推薦ツールの利用効果が問題によって偏らないように、被験者によってツールを利用の有無を問題によって変えるように割り当てた。また、推薦ツールを利用した場合の学習効果を防ぐために、3つの問題で連続してツールを利用しないようにタスクの割り当てを行った。また、過去の回答を参考にできないようにした。

V. RESULTS

このセクションでは、10人の被験者による SuiteRec の定量的および定性的評価結果を報告する。前のセクションで説明したように、4つの研究課題について分析結果を提示します。

A. RQ1: SuiteRec の利用は、開発者の作成したテストコードのカバレッジにどう影響するか？

本実験では、被験者によって提出されたテストスイートの命令網羅と分岐網羅の2種類のコードカバレッジの計算した。カバレッジの計算には統合開発環境 Eclipse のプラグインとして搭載されている EcEmma を利用した。図1と図2はそれぞれ被験者による命令網羅と分岐網羅の平均カバレッジを示す。結果として、命令網羅の割合は3つの問題すべてにおいてツールを利用した場合とそうでない場合で網羅率にほとんど違いはなく、どの問題も網羅率が90%を超えている。図2の分岐網羅についても分岐数が少ない TASK1 と TASK2 についてはツールを使用した場合とそうでない場合でほとんど差がないことが分かる。しかし、プロダクションコードの分岐数が最も多い TASK3 については、実験者の平均カバレッジに10%以上の差があることが分かった。この結果は、分岐が多いプロダクションコードのテストコードを作成する際に、SuiteRec で推薦されるテストコードは網羅率を向上するのに役に立つことが考えられる。実際に実験後のアンケートの記述欄には、推薦コードによって見落としていたテスト項目をフォローすることができたという報告が複数存在した。

B. RQ2: SuiteRec の利用は、開発者のテストコード作成時間に影響するか？

図5は、SuiteRec を使用した場合と何も使用しない場合で、テストコード作成タスクの終了までに費やされた時間を比較しています。3つの問題の内、2つの問題で SuiteRec を



Fig. 5. Statement coverage (C0).



Fig. 6. Branch coverage (C1).

使用した場合そうでない場合と比べてテスト作成時間が大きくなっていることが分かる。この結果は SuiteRec によって推薦される複数のテストスイートを読み理解するのに時間がかかる可能性があります。被験者は、推薦されるテストコードをそのままの形で再利用することができません。入力したプロダクションコードと検出された類似コードの差分を見てテストコードを書き換える必要があります。また、実験後のアンケートではテスト対象のオブジェクト生成の記述を再利用する際にその都度書き換える必要があり、時間がかかってしまったと述べている。問題2については、SuiteRec を利用した場合の方がテスト作成時間が短いことが分かる。我々は、提出されたテストコード調査したところカバレッジに差はないものの SuiteRec を使用しない場合はテストケース(項目)の数多くなっていることが分かった。この結果は、被験者は無駄なテストケースを多く記述するのに無駄な時間を費やしてしまった可能性がある。

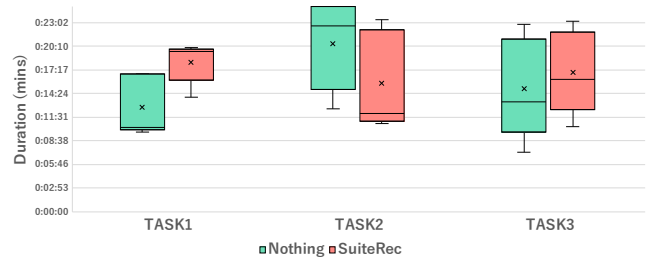


Fig. 7. Time taken to create test code.

C. RQ3 : SuiteRec の利用は、作成したテストコードの品質にどう影響するか？

図6は SuiteRec を使用した場合とそうでない場合で、提出されたテストコード内のテストスメル数を比較していま

す。すべての TASK に対して、SuiteRec を使用して作成されたテストコードはテストスメルをあまり含んでいないことが分かる。この結果は、推薦されるテストコード自体の品質が高く開発者はそれを再利用することで品質を維持したままテストコードを作成したと考えられる。また、ツールの出力画面で推薦されるテストスイート内に含まれているテストスメルを提示することで、それを基にテストコードを書き替えより品質の高いテストコードを提出した可能性が考えられる。実際のアンケートの記述でも提示されたテストスメルを理解し、それをなくすようにリファクタリングしテストコードを作成したという報告がされている。一方で、テストスメルが含まれていることは気づいていたがリファクタリングの方法が分からずそのまま提出したと述べている被験者も存在した。これは今後のツールの課題であり、テストスメルのリファクタリング方法も提示する改良の必要がある。SuiteRec を使用しなかった場合は、使用した場合と比べ全体として 5 倍以上の被験者はテストスメルを埋め込んでいた。その中でも多く埋め込まれていたテストスメルとして、Assertion Roulette, Default Test, Eager Test が挙げられる。多くの被験者は、初期状態のテストメソッドの名前を変更せず一つのテストメソッド内でコピーアンドペーストによって Assert 文を記述していたのが原因だと考えられる。実際に既存研究でもこれらのテストスメルが既存プロジェクトで多く検出されていることが報告されている [6]。



Fig. 8. Number of detected test smells.

D. 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しますか？

図 7 は、実験後のアンケートの回答の結果をまとめたものです。初めの 2 つの質問から、被験者は、実験タスクを明確に理解し (質問 1), 実験タスクを終えるのに十分な時間があったことが分かる (質問 2)。残りの質問については、SuiteRec を使用した場合とそうでない場合で、実験タスクに対する意見に違いがあることが分かります。

被験者はテストコードを作成する際に、SuiteRec を用いるとテストコード作成を容易に感じることができます。しかし、この結果はこの結果は実際のタスクの終了時間と長さ (図 2) とは対照的であり、SuiteRec を使用した場合の方がタスクの終了時間が遅いことが分かります。被験者は、推薦された複数提案されるテストスイートを読み理解して再利用するかどうかを決定します。また、テストコードはそのままの状態でも適用することはできず、入力コードと検出された類似コードの差分を理解しテストコードに適切な修正を加える必要があります。我々は、SuiteRec を使用した

場合被験者はこの部分に多くの時間を費やすことがあると推測しています。アンケートによるツールの改善点への自由記述では、テストコードの編集作業を支援する機能 (クラス名やメソッド名を入力コード対応する名前に自動編集する機能など) を追加した方が良いという多くの意見を頂戴しました。SuiteRec の更なる改善は、実験タスクの完了時間を短縮できる可能性を示しています。

被験者は、SuiteRec を使用した場合、自身で作成したテストコードのカバレッジに自信があることが分かる (質問 5)。一方で、何も使用しなかった場合 40% の被験者がネガティブな回答を報告している。しかし、実際に提出されたテストコードのカバレッジにはほとんど差がないことが分かっています (図 3)。自身が作成したテストコードのカバレッジに自信を持つことは重要です。開発者は、自分の書いたコードに責任を持ち、不安なくソフトウェアをユーザに提供できることは、ソフトウェアテストを行う目的の一つです。

被験者は、何も使わずテストコードを作成した場合 40% の被験者が自身の書いたテストコードの品質に自信が持てません。実際の提出されたテストコード内のテストスメルの数も SuiteRec を使わなかった場合は、使った場合と比べて多く存在していることが分かります (図 4)。開発者は無意識の内にテストスメルを埋め込みそれが後のメンテナンス活動を困難にさせます。SuiteRec の利用は、開発者にテストコードの品質に対する意識を与えることでテストスメルの数を減らし、作成したコードに自信をもたらします。一方で、SuiteRec を利用した場合でも品質に関してネガティブな意見も存在します。アンケートの記述項目では、テストスメルの存在は意識できたが具体的にどう修正してなくすることができるのか分からなかったと報告されています。これは SuiteRec の更なる改善の必要性を示しており、各テストスメルに対するリファクタリング方法も提示する機能を追加すべきだと考えている。

VI. RELATED WORK

Code recommendation. コード推薦システムは、他のプログラムのコードフラグメントを提示し再利用できるようにしたりすることで開発者を支援します。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手法を提案している。Mostafa[2] らは、自身のプロジェクトだけでなく他のプロジェクトを横断してクローンペアを検出しテストコード再利用することの有効性を調査した。

VII. CONCLUSION AND FUTURE WORK

SuiteRec は、ユーザーが入力した関数単位のプロダクションコードに対して、類似コード検出ツールを用いて OSS 上に存在する既存のテストコードを推薦するツールです。さらに、テストコードの良くない実装を表すメトリクスであるテストスメルを開発者に提示し、より品質の高いテストスイートを推薦できるように推薦順位がランキングされています。分岐が多くテスト項目の作成が難しいプロダクションコードに対して、SuiteRec を使用してテストコード作成するとカバレッジを向上できる可能性があります。また、品質の高いテストコードを作成でき、開発者は自分で書いたコードに自信が持てることができます。今後の課題としては、より実践的な利用に備えてツールを改善する必要があります。

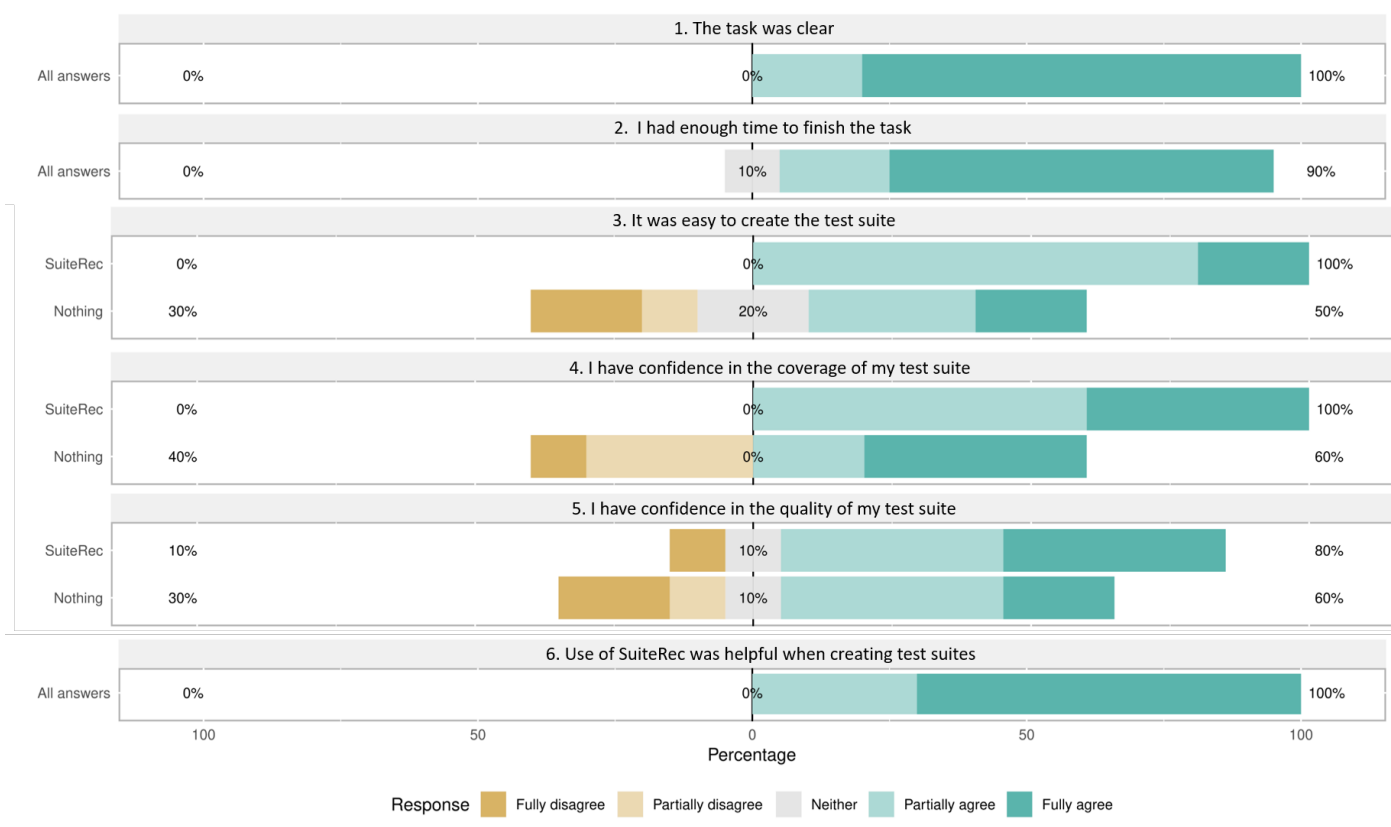


Fig. 9. キャプション

す。さらに SuiteRec が推薦するテストスイートの優先順位に対する妥当性評価も実施する予定である。

REFERENCES

- [1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
- [2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
- [3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
- [4] K. Elissa, "Title of paper if known," unpublished.
- [5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
- [6] Y. Yoroazu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetism Japan, p. 301, 1982].
- [7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.