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

結合テストを用いたテストの半自動生成手法の提案
-自然で網羅率の高い単体テストの生成を目的として-

小田　拓輝†　　栁本　真佑†　　楠本　真二†

† 大阪大学大学院情報科学研究科
E-mail: †{hiro-oda,shinsuke,kusumoto}@ist.osaka-u.ac.jp

あらまし　テストの効率的な作成支援を目的として，探索に基づく自動テスト生成手法が提案されている．この手法では遺伝的アルゴリズムに基づいて，網羅率の高い単体テストの集合を作り出す．しかしながら，自動で生成されたテストには不自然なパラメタが数多く含まれており，テストの可読性は高いとはいえない．本研究では自然で網羅率の高いテストの生成を目的として，結合テストを用いたテストの半自動生成手法を提案する．提案手法では，まず開発者が結合テストを記述する．次に結合テストを実行し，テスト対象プロダクトがどのようなパラメタによって呼び出されているかを解析する．ここで得られた自然なパラメタを，自動テスト生成手法の作り出したテストにマッピングする．これにより自然で網羅率の高いテストを得ることが出来る．

キーワード　EvoSuite，結合テスト，単体テスト，ソフトウェアテスト，自動テスト生成

1. はじめに

ソフトウェアの信頼性確保のためにソフトウェアテストは欠かせない．下流工程で実施される自動テストでは，テストケースそのものを一種のプログラムとして記述し，対象ソフトウェアの振る舞いを自動的に検証する．様々な利用状況を模したテストを設けることでテストの保守性が高くなる．

テスト作成に要する作業コストの低減を目的として，自動的なテスト生成技術が多数提案されている [1] [2]．自動テスト生成では，テスト対象のソースコードを入力として，ボトムアップ的にテストケースを自動生成する．作成されたテストは，要求に基づいて作成されるトップダウンなテストとは異なり，過去のバグの再発現の抑制やリファクタリング支援のために用いられる．自動生成の基本アイデアは探索であり，網羅率の最大化を目的としてプログラムの呼び出し系列や引数の値を探し出す．Java では遺伝的アルゴリズムを用いた EvoSuite [3] が広く知られており，高い網羅率を持つテスト生成が可能であると報告されている．

EvoSuite をはじめとする探索ベースの自動テスト生成の課題として，テストの可読性の低さが挙げられる [4]．探索手法は網羅率の最大化のみを目的関数としており，生成されたテストの読みやすさや自然さは軽視される傾向にある．特にテスト対象メソッドを呼び出す際のパラメタは，その型の情報のみに基づいてランダムに生成される．例えばユーザ名のような引数に対しては，一見してユーザ名とは判別できないランダムな文字列が生成される．また int 型の値を取る年齢に対しては，年齢としては取り得ないような巨大な数字となることもある．パラメタの不自然さはテスト内容の誤解を誘発し，そのテストの検証内容や意図の誤解に繋がると考えられる．

本研究では自然で網羅率の高いテストの生成を目的として，結合テストを用いたテストの半自動生成手法を提案する．本手法のアイデアは，「開発者が記述した結合テストは自然なパラメタで構成されている」ことである．提案手法では，まず開発者が結合テストを記述する．次に結合テストを実行し，テスト対象プロダクトがどのようなパラメタによって呼び出されているかを解析する．ここで得られた自然なパラメタを，EvoSuite の作り出したテストにマッピングする．これにより，EvoSuite が自動生成した網羅率の高いテストに含まれる不自然なパラメタを自然なパラメタに置換できるため自然で網羅率の高いテストを得ることが出来る．

本稿では，提案手法の有効性について調査を行った．具体的に，結合テストから抽出したパラメタの自然性とマッピングに失敗する原因について調査した．評価実験では，結合テストから抽出した全パラメタを目視で確認し自然なパラメタであるかを判別した．加えて，マッピングに失敗した原因の共通点を考察し，得られた共通点を元に提案手法の改善策について議論した．調査の結果，抽出パラメタは自然な値であることを確認した．また，マッピングに失敗する原因は提案手法のマッピング条件であることが判明し，具体的な解決策を提案した．

2. 準　　備

2.1 ソフトウェアテスト

ソフトウェアテストとはソフトウェアに対する信頼性を確保するための工程である．代表的なテストとして単一の機能を検証する単体テストと複数の機能を組み合わせた動作を検証する結合テストがある．一般にこれらのテストでは，テストの自動実行や実行結果の出力といった機能を持つテストフレームワークが使用される．Java に対するテストフレームワー

This article is a technical report without peer review, and its polished and/or extended version may be published eleswhere.
Copyright ©2025 by IEICE

```java
public class User {
  private String name;
  private int age;
  public User(String name, int age) {
    this.name = name;
    this.age = age;
  }
  public String getName() { return name; }
  public int getAge()     { return age;  }
}
```

図1　テスト対象の題材クラス

```java
@Test(timeout = 4000)
public void test1() throws Throwable {
  //不自然なパラメタを含むメソッド
  User user0 = new User("8oi_Q", 369);
  int int0 = user0.getAge();
  assertEquals(369, int0);
}
```

図2　不自然なパラメタを含む自動生成テスト

```java
@Test(timeout = 4000)
public void test1() throws Throwable {
  //自然なパラメタに置換したメソッド
  User user0 = new User("Taro", 20);
  int int0 = user0.getAge();
  assertEquals(20, int0);
}
```

図3　提案手法により自然なパラメタをマッピングしたテスト

クとしては JUnit(注1) が広く用いられている．テストケースはプロダクトコードのメソッドを実行する実行部と実行結果が正しいかを検証する assert 部で構成される．

ソフトウェアテストにおいてはテスト対象となるプロダクトコードをどの程度網羅的に検証したか，という指標が重要である．この指標は網羅率と呼ばれる．またテスト自体の可読性も重要な指標である．テストの可読性を高く保つことで，テスト自体の保守性を高めることが出来る．可読性を高める要素としてパラメタの自然さがある．

2.2　自動テスト生成

自動テスト生成とはソースコードを入力として，複数のテストケースを帰納的に生成する手法である．自動テスト生成はソースコードを基準として作成するため，信頼性確保のために作成されるトップダウンなテストとは異なる目的で利用される．具体的にはリファクタリング前の回帰テストとしての活用が挙げられる．自動テスト生成はこれまで数多くの研究が行われており [1] [2]，EvoSuite [3] や Randoop [5]，SUSHI [6] 等の様々な自動テスト生成手法が公開されている．以降，テストケースを単にテストとも呼ぶ．

Java を対象とした最も代表的な自動テスト生成手法, EvoSuite について説明する．EvoSuite は遺伝的アルゴリズムに基づく探索ベースの自動テスト生成手法である．EvoSuite は他にも，ハイブリッド探索 [7] や動的シンボリック実行 [8]，テスタビリティ変換 [9] といった技術を活用する．EvoSuite は他の自動テスト生成手法と比較して，より網羅率の高い単体テストの生成が可能である [10] と報告されている．

2.3　EvoSuite の課題

EvoSuite は可読性の低いテストを生成してしまう [4]．具体的には，テスト対象メソッドのパラメタが不自然である．以降，テスト内に含まれるテスト対象となるプロダクトコードのメソッドを単にメソッドと呼ぶ．例として，図 1 にテスト対象クラス，図 2 に EvoSuite が自動生成したテストを示す（以降，自動生成テストと呼ぶ）．User クラスのコンストラクタのパラメタの具体値はそれぞれ人間の名前，人間の年齢が自然であるが EvoSuite の生成したテストのパラメタはいずれも不自然な値となっている．

(注1)：https://junit.org/junit5/

3.　提 案 手 法

3.1　提案手法の概要

本研究の目的は，自然で高い網羅率を持つ単体テストの作成支援である．そのために，自動生成された単体テストと開発者の作成した結合テストを組み合わせたテストの半自動生成手法を提案する．提案手法では，最初に既存の自動テスト生成技術を用いて網羅率の高い単体テスト群を得る．次に開発者が記述した結合テストを実行し，メソッド呼び出し時の自然なパラメタを抽出する．そして抽出したパラメタを自動生成されたテストに適切にマッピングする．この一連の流れにより自然かつ網羅率の高い単体テストの生成を実現する．図 3 に提案手法を用い，結合テストから抽出した自然なパラメタを図 2 の自動生成テストにマッピングした例を示す．元の不自然なパラメタを含むコンストラクタ呼び出し `new User("8oi_Q", 369)` を結合テストから抽出した自然な呼び出し `new User("Taro", 20)` に置換することで可読性が向上したテストを生成できる．

3.2　提案手法の流れ

提案手法の流れを図 4 に示す．提案手法は，ソースファイルおよび結合テストを入力として受け取り，単体テスト群を出力する．提案手法は以下の 5 ステップから構成される．

S0　単体テストの自動生成
S1　自動生成テストからの実行情報の収集
S2　結合テストからの実行情報の収集（パラメタ抽出）
S3　抽出したパラメタを自動生成テストへマッピング
S4　assert 部の置換

以降，S0 から S4 の各ステップの詳細を説明する．

S0. 単体テストの自動生成

本ステップではテスト対象のソースコードを入力として単体テスト群を自動生成する．自動生成されたテストは，2.2 節でも説明した通り網羅率が高いテストである．

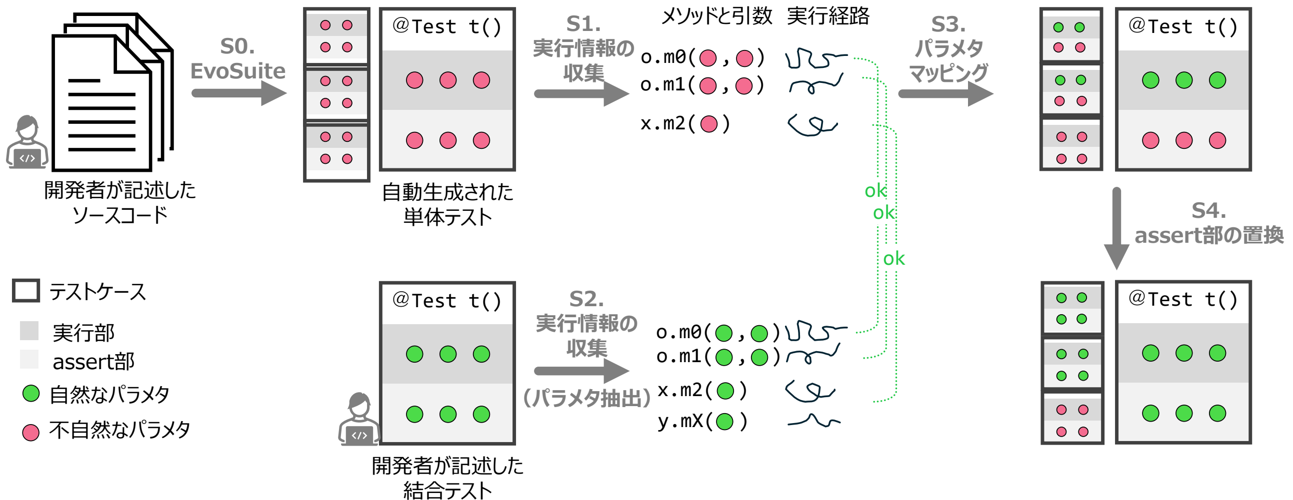

図 4　提案手法の流れ

S1. 自動生成テストからの実行情報の収集

本ステップでは S0 で得られた自動生成テストを実行し，テスト実行時の情報を得る．テスト実行時の情報は，自動生成テストを動的解析することにより収集する．収集すべき情報は，テスト対象となったメソッド，すなわちプロダクトコードのメソッド名とその実行時パラメタの具体値である．図 4 では，テストケース `t()` に対しプロダクトコードのメソッド名 `m0`, `m1`, `m2` と各メソッドの実行時パラメタを取得している．動的解析により実行された各メソッドのメソッド名と実行時のパラメタの具体値を記録する．この時，S1 から得られたパラメタの具体値は 2.3 節でも説明したような不自然な値である．さらに各メソッドの実行経路も回収する．実行経路の取得は，各メソッドについて実行時に通過した行を動的解析の結果から収集することで実現する．この時，各メソッドの呼び出し関係にあるメソッドについても実行経路を解析する．

S2. 結合テストからの実行情報の収集

結合テストを実行してテスト実行時の情報を収集する．収集すべき情報は，S1 と同様プロダクトコードのメソッド名とその実行時パラメタの具体値である．テスト実行時の情報は，結合テストに対し動的解析を行うことで収集する．結合テストは開発者が作成したテストであるため，結合テストの実行過程で発生するパラメタの具体値は自然な値になっていると考えられる．加えて各メソッドの実行経路も回収する．なお実行対象が結合テストであるため，単体テストの場合と比較するとメソッドの呼び出し関係は深くなる傾向にある．提案手法では全てのメソッド呼び出しについて，実行時情報を回収し S3 のマッピング対象とする．

S3. 抽出したパラメタを自動生成テストへマッピング

S1 と S2 で得られた実行時情報に基づいて，パラメタのマッピングを試みる．パラメタマッピングのアイデアは「メソッド呼び出し時の実行経路が同一となるパラメタは交換可能（テストの振る舞いが保たれる）」という点である．例えば図 2 の自動生成テストにおいて，`new User(String name, int age)` に対して，`User("8oi_Q", 369)` と `User("Taro", 20)` が同一の経路で実行されたならば，この 2 つのパラメタはどちらに入れ替えても振る舞いは同一である．

上記のアイデアを実現するために，マッピングの条件を以下の通り定めた．パラメタのマッピング条件は，「自動生成テストのテストケースに含まれる全てのメソッドに対し，実行経路が同一であるメソッドが結合テストの実行経路に含まれるとき」である．図 4 では，EvoSuite が生成したテストケース `t()` において，プロダクトコードのメソッド `m0()`, `m1()`, `m2()` と実行経路が同一であるメソッドが結合テストの実行経路に含まれるため，パラメタのマッピング条件を満たしている．この条件により，EvoSuite が生成したテストケース全体の実行経路がパラメタをマッピングする前後で変わらないので，高い網羅率を保った状態でパラメタの置換ができる．

なお，本稿では 2 つの実行経路の同一性を次のように考える．あるインスタンス `o` のメソッド `m()` を呼び出した場合，`m()` 実行に至るまでに `o` の複数のメソッドが呼び出される可能性がある．`m()` の経路を含め `o` の呼び出されたメソッドの全ての経路が同一のとき，2 つの実行経路が同じであると見なす．さらに，結合テスト側では `o` に対して過剰にメソッド呼び出しが行われる可能性がある．対して，自動生成された単体テストはメソッド呼び出しは最小化されることが多い．このギャップを埋めるために，完全一致だけではなく結合テスト側の経路が単体テストの経路を包含しているときも実行経路が同一と判断する．

S4. assert 部の置換

S3 で実行部のマッピングをした単体テスト群に対し，assert 部の置換を行う．S3 で生成された単体テスト群の各 assert メソッドの期待値はマッピング前のテストにおける期待値であるため，マッピング後のテストに対する期待値に置換する必要がある．例えば，図 2 の自動生成テストに対しマッピングを行った図 3 では，`assertEquals(369, int0)` を `assertEquals(20, int0)` に置換している．マッピングしたテストの期待値は，実行部のパラメタをマッピングをした単体テスト群を実行し，各 assert メソッドの実行値を収集することで取得する．

3.3 提案手法の実装

自動テスト生成手法として EvoSuite を使用した．EvoSuite を使用する理由は，先行研究において他の自動テスト生成手法と比較して，より網羅率の高い単体テストを生成することが指摘されている [10] ためである．テストの動的解析では，SELogger(注2) を用いた．

本稿ではマッピング対象のメソッドは，返り値がプリミティブ型あるいは `String` 型であるメソッド，またはコンストラクタに限定している．加えて，`static` メソッドまたはクラス内クラスを使用するメソッドを含むテストはマッピングの対象外としている．

4. 評価実験

4.1 RQ と実験題材

提案手法により，自動生成テストに対し結合テストから抽出した自然なパラメタをマッピングする能力を評価する．評価実験にあたり，以下の 3 つの Research Question を設定した．

RQ1 結合テストから抽出したパラメタは自然か

RQ2 自動生成テストに対し，結合テストから抽出したパラメタのマッピングに成功したテストはどの程度か

RQ3 EvoSuite が生成した不自然なテストのうち，自然なパラメタをマッピングできたテストはどの程度か

RQ1 は提案手法の前提が成り立つかを確認している．ここでの前提とは，結合テスト実行時に得られた各パラメタはそのメソッドの引数として自然である，という性質である．RQ2 は RQ3 の準備としてマッピングに成功したテスト数を確認し，パラメタのマッピング条件の妥当性について調査する．RQ3 は自動生成テストのもつ不自然なパラメタを自然なパラメタに置換した数を調査することで提案手法の有効性を調査することを目的としている．

実験題材は，著者らが用意した例題プロジェクト 4 つ（電卓，BMI 計算，図書館システム，学業成績評価システム）と，実用プロジェクト（ jsoup(注3) ）である．電卓はオブジェクト指向プログラミングにおける一般的な例として使用する．BMI 計算は，人間の名前・年齢・身長・体重など複数のフィールド変数が存在するクラスを含むプロジェクトの例として用いる．図書館システムと生徒の成績評価システムは，4 個程度のクラスから構成される小規模プロジェクトの例として用いる．

jsoup は Java による HTML の解析ライブラリである．提案手法が実際のプロジェクトでも有効であるかを確認するために採用した．提案手法の入力となる結合テストとしては，網羅率が最も高い org.jsoup.integration.ConnectTest.java を用いた．本テストクラスは 53 個のテストケースを含み，jsoup プロダクト全体の 48.8% を網羅する．自動テスト生成の対象クラスは一部に限定した．具体的には ConnectTest.java の実行形路上に全く現れないクラス，つまりパラメタマッピング出来る可能性がないクラスを排除した．

(注2)：`https://github.com/takashi-ishio/selogger`
(注3)：`https://jsoup.org/`

4.2 RQ1. 結合テストから抽出できた自然なパラメタの割合

結合テストから抽出したパラメタの総数とその中に含まれる自然なパラメタの総数を調査する．抽出したパラメタと対応するメソッドを全て目視で確認し，そのメソッドが取るべきパラメタとして自然かを確認した．

実験の結果を表 1 に示す．全てのプロジェクトにおいて結合テストから抽出したパラメタはほぼ全て自然であることがわかる．これは結合テストが自然なパラメタで構成される場合，実行部のメソッドの呼び出し関係にあるメソッドのパラメタも自然なパラメタで構成されるからだと考えらえる．

表 1 結合テストから抽出できた自然なパラメタの割合（RQ1）

プロジェクト名	抽出できたパラメタの総数	自然なパラメタの数（割合）
電卓（例題系）	18	18 (100%)
BMI 計算（例題系）	4	4 (100%)
図書館（例題系）	23	23 (100%)
成績評価（例題系）	14	14 (100%)
jsoup（実用系）	6,270	6,259 (99.8%)

4.3 RQ2. 抽出パラメタのマッピング成功割合

EvoSuite が生成したテストケースの総数と，パラメタのマッピングに成功したテストケースの総数を検証する．それに加え，結合テストから抽出したパラメタの総数と，マッピングに成功したパラメタの総数を調査する．例題プロジェクトと実用プロジェクトでプロジェクトの性質が異なるため例題プロジェクトと実用プロジェクトそれぞれについて議論を行う．

4.3.1 例題プロジェクトに対する実験結果

例題プロジェクトに対する実験の結果を表 2 に示す．EvoSuite が生成したテストケースの 93 件うち，パラメタのマッピングに成功したテストは 40 件であり約 60% のテストはマッピングに失敗した．また結合テストから抽出したパラメタ 59 件のうち，マッピングに成功したパラメタは 54 件であった．

自動生成テストに対しパラメタのマッピングが失敗した原因は，EvoSuite が生成したテストケースが結合テストで実行しないメソッドを含んでいたためである．結合テストで実行しないメソッドとして getter メソッドが多く該当した．マッピングに成功するテスト数を増加させるために，結合テストの網羅率を高める必要がある．

加えて，結合テストから抽出したパラメタを自動生成テストにマッピングできなかった原因は，マッピングに失敗したパラメタで構成されるメソッドの実行経路を EvoSuite が生成したテストケースが含んでいなかったからである．自動生成テストの網羅率が 100% でない場合，マッピングに失敗するパラメタが存在する可能性がある．

4.3.2 実用プロジェクトに対する実験結果

実用プロジェクトに対する実験の結果を表 2 に示す．EvoSuite が自動生成したテストケース 2,060 件のうち，マッピングに成功したテストケースはわずか 2 件であった．また抽出したパラメタ 6,270 件のうち，マッピングに成功したパラメタは 26 件であった．

表 2　結合テストから抽出したパラメタをマッピングできた自動生成テストの総数 (RQ2, RQ3)

プロジェクト名	S: 自動生成テストの総数	A: マッピングできたテストの総数 (A/S)	B: 可読性が改善したテストの総数 (B/A)	抽出できたパラメタの総数	マッピングできたパラメタの総数
電卓（例題系）	19	14 (73.7%)	9 (64.3%)	18	13 (72.2%)
BMI 計算（例題系）	23	6 (26.1%)	6 (100%)	4	4 (100%)
図書館（例題系）	30	10 (33.3%)	6 (60.0%)	23	23 (100%)
成績評価（例題系）	21	10 (47.7%)	8 (80.0%)	14	14 (100%)
合計（例題系）	93	40 (43.0%)	29 (72.5%)	59	54 (91.5%)
jsoup（実用系）	2,060	2 (0.0%)	0 (0.0%)	6,270	26 (0.4%)

パラメタのマッピングに失敗した代表的な原因は，結合テストで実行しないメソッドを自動生成テストケースが含んでいたためである．複数のコンストラクタを含むクラスに対し，実行内容は同じだが呼び出すコンストラクタが違うことにより，マッピングできないテストがあった．実行経路は等しくないがテストの振る舞いが等しいメソッドはテストケースの実行結果に影響を与えないため，実行内容が等しいメソッドに対するパラメタのマッピング条件を見直すことでマッピング成功数を増加させることができる．また 4.3.1 節と同様，マッピングに失敗したパラメタで構成されるメソッドは自動生成テストの実行経路に含まれていなかった．

4.4　RQ3. マッピングにより可読性が向上したテストの割合

マッピングに成功したテストケースの総数と，マッピングに成功したテストケースのうち不自然なパラメタを自然なパラメタに置換したテストケースの総数を調査する．例題プロジェクトと実用プロジェクトでプロジェクトの性質が異なるため例題系と実用系それぞれについて議論を行う．

4.4.1　例題プロジェクトに関する実験結果

例題プロジェクトについて，実験の結果を表 2 に示す．マッピングに成功したテストケース 40 件のうち 29 件がパラメタマッピングにより不自然なパラメタを自然なパラメタに置換できた．この結果から，クラス数少ない小規模プロジェクトについては提案手法は十分有効であるとわかる．

4.4.2　実用プロジェクトに関する実験結果

実用プロジェクトについて，実験の結果を表 2 に示す．マッピングに成功したテストケースのうち提案手法で不自然なパラメタを自然なパラメタに置換できたテストケースはなかった．しかしながら，RQ2 においてマッピングに成功したテストが少ないため，マッピング成功数を増加させ再度実験を行う必要がある．

5.　議　　論

5.1　マッピング成功率を増加させる方法

抽出パラメタのマッピング率の低さが本研究の課題である．マッピング率を増加させるための手法を議論する．本稿では 2 つの方法を提案する．

1 つ目の方法として，パラメタのマッピング条件を緩和することが挙げられる．本稿では「自動生成テストのテストケースに含まれる全てのメソッドに対し，実行経路が同一であるメソッドが結合テストの実行経路に含まれるとき」をパラメタのマッピング条件とした．提案手法の適用条件を緩和し，「自動生成テストのテストケースに含まれる任意のメソッドに対し，そのメソッドが結合テストの実行経路に含まれるとき」をマッピング条件とした場合について議論する．このように条件を修正した場合，マッピングに成功するテスト数は増加する．しかし元のテストケースと振る舞いが異なるテストケースが生成される危険性がある．図 5 に条件変更により生成される振る舞いが異なるテストケース，図 6 にマッピング前の元のテストケースを示す．Library クラスの findBook メソッドは，引数の String 型文字列をタイトルに持つ Book インスタンスが登録されている場合その Book インスタンスを返し，登録されていない場合 null を返す．振る舞いが異なるテストケースは，結合テスト内で Book オブジェクトを返す findBook メソッドが実行されていない場合に生成される．この時，図 5 の 7 行目において book1 に null が代入されるため，9 行目の book1.isAvailable() に対してエラーが発生する．このことから，テストケース全体の実行結果が等しくなるマッピング条件を考察する必要があるとわかる．

```
@Test
public void test0() throws Throwable {
  Library library0 = new Library();
  //パラメタの置換を行ったメソッド
  Book book0 = new Book("TestBook", "Author");
  library0.addBook(book0);
  Book book1 = library0.findBook("ex08.Book");
  //エラーが発生するメソッド
  boolean boolean1 = book1.isAvailable();
}
```

図 5　マッピング条件変更時に生成される振る舞いが異なるテスト

```
@Test
public void test0() throws Throwable {
  Library library0 = new Library();
  //パラメタの置換ができたメソッド
  Book book0 = new Book("ex08.Book", "");
  library0.addBook(book0);
  //パラメタの置換ができなかったメソッド
  Book book1 = library0.findBook("ex08.Book");
  boolean boolean1 = book1.isAvailable();
}
```

図 6　振る舞いが異なるテストのマッピング前の自動生成テスト

```java
@Test(timeout = 4000)
public void test1()  throws Throwable  {
  User user0 = new User("8oi_Q", 369);
  int int0 = user0.getAge();
  user0.getName();  //作用のないメソッド
  assertEquals(369, int0);
}
```

図7　テスト結果に影響を与えないメソッドを含むテストの例

2つ目の方法は，作用のないメソッドを含む場合のマッピング条件を改善する方法である．アイデアは，「作用のないメソッドを実行したとしてもテスト結果に影響を与えない」である．例えば，図1をテスト対象とした図7のテストにおいて user0.getName() はテスト結果に影響を及ぼさない．したがって，「結合テストの実行経路で包含されない実行経路を含む自動生成テストに対し，包含されていない実行経路がテストの実行結果に影響を与えない場合はマッピング可能」とする条件を考察することで，マッピング結果改善が実現すると予測できる．

5.2　マッピングするパラメタの選択方法

提案手法の抱える問題点として，マッピング可能なパラメタが複数存在する場合，どのパラメタをマッピングするべきかわからないことが挙げられる．結合テストの実行経路の中に実行経路が同一であるメソッドが複数存在する場合に，マッピング可能なパラメタが複数存在する．本稿では抽出順が一番早いパラメタをマッピング対象としている．そのため同じパラメタを何度もマッピングする場合がある．EvoSuite は実行部で呼び出すメソッドが同じテストケースを複数生成する可能性があるため，パラメタの種類が豊富であることが望ましい．マッピングしていないパラメタを優先的にマッピングするアルゴリズムを導入して改善を目指す．

5.3　提案手法の発展

本稿で提案した手法はさらなる発展が期待できる．ここでは，マッピングに失敗したパラメタの利用方法について考察する．結合テストから抽出したパラメタを自動生成テストにマッピングできなかった原因は，マッピングに失敗したパラメタを含むメソッドの実行経路が自動生成テストの実行経路に含まれていなかったためである．このことから，マッピングに失敗したパラメタを含むメソッドを用いたテストケースを生成すれば，EvoSuite が生成した単体テスト群以上に網羅率の高い単体テスト群を生成できると予測できる．

6.　おわりに

本研究では，自然で網羅率の高いテストの生成を目的として，自動テスト生成手法と結合テストを用いたテストの半自動生成手法を提案した．提案手法では，網羅率の高い自動生成テストに対し結合テストから抽出した自然なパラメタをマッピングすることで自然で網羅率の高いテストを生成する．例題プロジェクトと実用プロジェクトに対して提案手法の妥当性とパラメタのマッピング成功割合を評価した．評価実験により，結合テストから抽出したパラメタは自然であることが分かった．またパラメタのマッピングに失敗するテストケースの共通点を特定し，改善案を提示した．

今後の研究課題として，マッピングに成功する自動生成テストの割合が少ないことが挙げられる．マッピングに失敗する代表的な原因はマッピング条件であった．マッピング条件の改善によりマッピングに成功するテストケース数が増加する可能性があるため，マッピング条件の改善方法について調査が必要である．加えて，結合テストの実行経路で包含されない実行経路を含むテストに対するマッピングアルゴリズムを改善することよりマッピング率の増加を目指す．

加えてマッピング可能なパラメタが複数存在する場合にどのパラメタをマッピングするべきかわからないという課題がある．本研究ではパラメタの抽出順が最も早いパラメタをマッピング対象としたが，同じパラメタを複数マッピングしパラメタの豊富さが損なわれる．マッピング可能なパラメタが複数存在する場合，マッピングしていないパラメタを優先的にマッピングするアルゴリズムを提案手法に組み込むことで改善を目指す．

提案手法の発展として，マッピングに失敗したパラメタの利用方法について考察した．マッピングに失敗したパラメタで構成されるメソッドを含むテストケースを生成することで自動生成テストより網羅率の高い単体テストを生成することを目指す．

謝辞　本研究の一部は,JSPS 科研費（JP24H00692, JP21H04877，JP21K18302）による助成を受けた．

文　献

[1] P. McMinn, “Search-based software test data generation: a survey,” Journal on Software Testing, Verification and Reliability, vol.14, pp.105–156, 2004.

[2] F. Kifetew, X. Devroey, and U. Rueda, “Java unit testing tool competition - seventh round,” International Workshop on Search-Based Software Testing, pp.15–20, 2019.

[3] G.Fraser and A.Arcuri, “EvoSuite: automatic test suite generation for object-oriented software,” In Proceedings of European Conference on Foundations of Software Engineering, pp.416–419, 2011.

[4] G. Fraser and A. Arcuri, “Evosuite: On The Challenges of Test Case Generation in the Real World,” Journal on Software Testing, Verification and Validation, pp.362–369, 2013.

[5] C.Pacheco and M.D. Ernst, “Randoop: Feedback-directed random testing for java,” In Proceedings of Object-Oriented Programming, Systems, Languages and Applications, pp.815–816, 2007.

[6] P. Braione, G. Denaro, A. Mattavelli, and M. Pezzè, “SUSHI: A test generator for programs with complex structured inputs,” In Proceedings of International Conference on Software Engineering, pp.21–24, 2018.

[7] M. Harman and P. McMinn, “A theoretical and empirical study of search-based testing: Local, global, and hybrid search,” IEEE Transactions on Software Engineering, p.226 – 247, 2010.

[8] P. Godefroid, N. Klarlund, and K. Sen, “DART: Directed automated random testing,” In Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, p.213 – 223, 2005.

[9] M. Harman, L. Hu, R. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper, “Testability transformation,” IEEE Transactions on Software Engineering, pp.3–16, 2004.

[10] G. Fraser and A. Arcuri, “Whole test suite generation,” IEEE Transactions on Software Engineering, vol.39, pp.276–291, 2013.