特別研究報告

題目

メソッド純粋性の変化に伴う不具合に関する調査

指導教員
楠本 真二 教授

報告者
小倉 直徒

平成 27 年 2 月 13 日

大阪大学 基礎工学部 情報科学科

平成 27 年度 特別研究報告

メソッド純粋性の変化に伴う不具合に関する調査

小倉 直徒

内容梗概

関数型言語において関数の純粋性は重要な特性の 1 つである．関数型言語で用いられる関数の純粋性をオブジェクト指向型言語に適用し推定された，純粋性を持つメソッドを “純粋なメソッド” とよぶ．開発者にとってメソッドの純粋性は有用である．メソッドの外部の状態を変化させない純粋なメソッドは，並列実行による競合といった問題が起こらないため並列で実行されるメソッドの実装において重要である．また，`equals` メソッドや `hashCode` メソッド，`getter` メソッドなど呼び出しによってインスタンスのメンバ変数が変化しないと開発者が推測しているメソッドもある．

仕様として定めたメソッドの純粋性をソースコード中に記述し，実際に実装された純粋性を検証することで純粋性を保証することが，契約プログラミングの 1 つとして行われている．しかし契約プログラミングによる設計を行っているソフトウェアはほとんどなく，また仕様として定めた純粋性がその品質に影響を与えるかは不明である．

本研究では純粋であったメソッドが純粋でなくなることに起因する不具合を検証するため，2 つのオープンソースソフトウェアを対象に調査した．結果として，ソフトウェア開発において多くは純粋性の変化しないメソッドであるが，何度も純粋性の変化するメソッドがいくつかあることが分かった．また純粋性が変化する多くのメソッドのソースコードには修正が加えられていないことが分かった．これは開発者が意図せずメソッドの純粋性を変化させてしまう可能性があることを示している．メソッドが純粋でなくなった修正を対象に，その修正内容の分類を行い，機能の追加や不具合の修正が多く行われていることが分かった．一方，不具合の混入と判断された修正は 1 件のみであり，またその修正は不具合修正のために追加したデバッグ用の出力であった．

主な用語

静的解析
リポジトリマイニング
純粋性
オブジェクト指向

目 次

図 目 次

表 目 次

1　はじめに

関数の純粋性は，関数型言語における重要な性質の 1 つである．純粋な関数を評価して得られた値は，引数として与えられた値のみによって決定されるため，関数内での処理は副作用を持たない．関数における副作用とは，関数の内外の状態に変化を与えたり，入出力処理を行なったりすることである．例えば，グローバル変数や静的なローカル変数の変更，ファイルへの書き込みなどである．

関数が純粋であることによりプログラムに利点がある．関数を評価して得られた値は引数にのみ依存するので，関数の呼び出し順序を任意に並び替えることができる．また，関数を並列に実行することで変数を同時に変更するなどの競合の問題も起きないため，並列処理と相性が良い [1].

オブジェクト指向型言語において，純粋であることが望ましいメソッドがあるため，関数型言語の性質である関数の純粋性をオブジェクト指向型言語のメソッドに適用させる研究が行われている [2,3]．純粋なメソッドは同時に呼び出しても競合といった問題が起こらないため，並列で実行されるメソッドの実装において純粋性は重要である．また，開発者が純粋なメソッドであるとメソッド名から推測することがある．例えば `equals` メソッドや `hashCode` メソッド，`getter` メソッドなどである．メソッドが副作用を持つかの認識を開発者が誤ることで，ソフトウェアに不具合を混入させるおそれがある．

メソッドの意図しない純粋性による不具合を防ぐため，メソッドの純粋性を仕様としてソースコード中に記述し，メソッドのもつ副作用を検証することで純粋性を保証することが，契約プログラミングの 1 つとして存在する．[4]．また，契約プログラミングを言語仕様として取り入れたプログラミング言語がある [5,6]．しかし，契約プログラミングによる開発を行っているソフトウェアはほとんどなく，仕様として定めた純粋性がソフトウェアの品質に影響を与えるかは不明である．

本研究では純粋であったメソッドが純粋でなくなることに起因する不具合を調査するため，オープンソースソフトウェアのリポジトリを分析した．リポジトリとはソフトウェアの開発履歴を格納したデータベースである．

リポジトリに格納された各リビジョンの全メソッドの純粋性の推定を行い，コミットによる純粋性の変化を検出した．また，メソッドが実装されてから削除されるまで，メソッド名の変更やファイル移動が行われても正しく追跡した．全メソッドのうち純粋でないメソッドに変化したものを対象に、開発履歴における全ての修正内容を目視で判断することで，メソッドが純粋でなくなることによる不具合への影響を評価した．

2 章では，論文中で登場する用語や定義の説明を行う．3 章では，研究の動機を説明する．4 章では，調査する方法について述べる．5 章では，調査するための準備について述べる．6

章では，調査した結果を回答する．7章では，調査結果を受けての考察をする．8章では，調査の妥当性について述べる．9章では，この研究のまとめを行い，今後の課題を示す．

2　準備

2.1　関数の純粋性

純粋な関数とは，関数内外の状態に変化を与えない関数のことを指す．グローバル変数の値を変更したり入出力処理を行った場合，その関数は関数外部の状態に変化を与えるため純粋な関数でない．また関数内で定義された静的変数を変更する場合も，関数内部の状態を変化を与えるので純粋な関数でない．一方，引数として与えられた値を変更する場合や，グローバル変数から値を取得するのみの場合，状態に変化を与えないので純粋な関数である．

状態を変化させる関数で並列処理を行う場合，同一データを同時に参照または変更する事で不具合が生じる可能性があるため，並列実行する関数は純粋であることが望ましい．

関数が純粋であることを保証するプログラミング言語として，Haskell などの純粋関数型言語がある．

2.2　純粋なメソッド

関数型言語で用いられる関数の純粋性を，オブジェクト指向型言語に適用したものを**メソッドの純粋性**と呼ぶ．メソッドの純粋性について以下の 6 つの属性を説明する．[7]

Stateless
外部の値を変更せず，かつ外部の値を取得しない．

Stateful
メンバ変数または静的変数から値を取得する．

StaticModfier
静的変数を変更する．

FieldModifier
メンバ変数を変更する．

ArgumentModifier
引数を変更する．

Native
入出力処理など OS と通信するメソッドを呼び出す．

各メソッドはその処理内容に応じて 1 つ以上の属性が推定される．本研究では純粋性が Stateless または Stateful のみのメソッドを，**純粋なメソッド**と定義する．StaticModifier，FieldModifier，ArgumentModifier，Native のいずれか 1 つでも推定されたメソッドは，純粋でな

いメソッドである．それぞれの属性が推定されるメソッドの例を図 1 に示す．`add` メソッドは，引数として与えられた値のみを用いて返り値を計算するため，メソッド外部の状態に影響を受けず，Stateless である．`getId` メソッドは，メンバ変数の `id` から値を取得するため，Stateful である．`setName` メソッドは，メンバ変数の `name` に値を設定するため，FieldModifier である．`setClassId` メソッドは，静的変数の `classId` に値を設定するので、StaticModifier である．`addIdToList` メソッドは，参照渡しによって与えられた引数の状態を変化させるため，ArgumentModifier である．`helloworld` メソッドは，出力処理を行なうため，Native である．

メソッドの純粋性は，そのメソッドの呼び出し元に伝播する．図 2 の `Person.getName` メソッドは FieldModifier であるため，`Person.getName` メソッドを内部で呼び出す `Person.toString` メソッドも FieldModifier となる．

オブジェクト指向型言語はポリモーフィズムを持つため，呼び出されるメソッドは実行時の状態に応じて変わる場合がある．図 3 において，`Vehicle` クラスの `getSpeed` メソッドはソースコードにおいて Stateless である．しかし，`Vehicle` クラスを継承し，`getSpeed` メソッドをオーバーライドした `Car` クラスが実装されている．`getSpeed()` によって呼び出されるメソッドが `Vehicle.getSpeed` か `Car.getSpeed` かは実行時にしか決定できないため，推定される純粋性は双方のメソッドがもつ属性となる．

2.3 版管理システム

版管理システムとは，様々なファイルの作成・修正・削除を記録し，修正履歴を管理するシステムで，バージョン管理システムとも呼ばれる．版管理システムを用いてプロジェクトを開発することで，プロジェクトの修正履歴を確認したり，バグが混入する前の状態に復元したりすることが容易になる．また，版管理システムには複数人での修正作業を想定した機能がある．プロジェクトのソースコードを複数人で修正する際，複数の開発者によって 1 つのファイルに異なる修正が行われることがある．このような場合には，修正の競合といった問題がある．版管理システムでは，ファイルが同時に修正されるのを制限することで競合を防いだり，1 つのファイルに対して行われた 2 つ以上の修正を統合する作業を支援することによって，修正の競合を解決したりする．広く用いられている版管理システムとしては，Subversion [8]，CVS [9]，Git [10] などがある．

以下で，版管理システムに関するいくつかの用語について説明する．

リポジトリ
ファイルの修正履歴を記録するデータベース．

リビジョン

```java
// Example of Statelsss
public int add(int a, int b) {
    return a + b;
}

// Example of Stateful
int id;
public int getId() {
    return this.id;
}

// Example of FieldModifier
String myname;
public void setName(String name) {
    this.name = name;
}

// Example of StaticModifier
static int classId;
public void setClassId(int id) {
    classId = id;
}

// Example of ArgumentModifier
public void addIdToList(List<int> ids) {
    ids.add(1);
}

// Example of Native
public void helloworld() {
    System.out.println("hello world");
}
```

図 1: メソッドの純粋性の例

```
class Person {
    String name;
    public getName() { // FieldModifier
        if (this.name == null) {
            this.name = "";
        }
        return this.name;
    }
    @Override
    public String toString() {
        return getName();
    }
}
```

図 2: メソッドの呼び出し関係とメソッドの純粋性

```
class Vehicle {
    public int getSpeed() { // Stateless
    return 0;
    }
}
    class Car extended Vehicle {
    @Override
    public int getSpeed() { // Native
        System.out.println("getSpped() called.");
        return 100;
    }
}
```

図 3: 子クラスの実装とメソッドの純粋性

版管理システムで管理されたソースコードの，ある特定の時点での状態を表す．開発者がプロジェクトに修正を加え，その修正をリポジトリに反映させることで，新しいリビジョンが作成される．また，開発者は任意のリビジョンをリポジトリから取得可能である．

コミット
ソースコードの修正をリポジトリに反映させること．コミットによってリビジョンが作成される．

コミットログ
コミットする際に，修正に関する情報を開発者が記したコメント．

2.4 ソフトウェアリポジトリマイニング

ソフトウェアリポジトリマイニングとは，版管理システムに格納された履歴情報や電子メールによる連絡内容などの開発時に蓄積されたデータを用いることで，プロジェクトにおけるプロセスや組織がどのようであったかをボトムアップ的に明らかにする研究である [11]．

例えばバグモジュール予測 (fault-prone bug prediction) では，バグ追跡システムの管理する情報も用いることで，バグを含む可能性が高い箇所を特定し [12]，レビューやテストの重点化に利用する．

3　調査の目的

3.1　メソッドの純粋性に関する既存研究

メソッドの純粋性を推定することは，メソッドの動作を知る上で重要である．しかし，メソッドの純粋性についてドキュメンに書かれていることは少ない．そのため，開発者はメソッドの純粋性について誤った認識をする可能性がある．David Pearce は，Java のソースコードからメソッドとフィールドの依存関係を解析し，メソッドの純粋性を推定した．またメソッドの純粋性の情報をアノテーションとしてソースコード中に挿入することを提案した [13].

Jiachen Yang は Java のバイトコードを解析することで，Java の標準ライブラリや依存ライブラリを含めて純粋性を推定することを可能にした [7]．Yang らの作成したツールによる実験では，標準ライブラリの中に適切でない純粋性を示すメソッドが含まれていることを示した．

3.2　動機

メソッドの中には純粋であることが望ましいものがある．例えば，並列に実行される可能性があるメソッドである．また，`hashCode` メソッドや `equals` メソッド，`compareTo` メソッドなど，状態に変化を与えないと開発者に期待されているメソッドなどがある．しかし，これらのメソッドの実装を意図せず変更してしまい，メソッドの純粋性が変化する場合があり，不具合となってソフトウェアの品質に影響を及ぼす可能性がある．

メソッドのあるべき純粋性を知る方法の 1 つとして，純粋性が仕様として設計時点で与えられる “契約プログラミング” がある．仕様として与えられたメソッドの純粋性と，すでに実装されたメソッドの純粋性を検証することで外部の状態を変化させることによる不具合を防ぐことができる．しかし，契約プログラミングを実施しているソフトウェアは少ない．

本研究では，メソッドが純粋でなくなることに起因する不具合があるのか，リポジトリに格納されたソフトウェアの開発履歴を用い調査することで，メソッドの純粋性の変化と不具合の混入の関係を明らかにする．

3.3　調査項目

本研究では，実ソフトウェアの開発履歴からメソッドの純粋性を推定し，純粋性の変化と不具合の混入との関係を明らかにすることを目指す．本調査の目的は以下の 2 点である．

- メソッドの純粋性が変化するような修正が行われるのか確認する．
- 純粋性の変化を引き起こすようなソースコードの修正内容を確認する．

これらの調査目的を達成するために，以下に挙げる 2 つの研究課題を設定した．

RQ1: メソッドの純粋性が変化することがあるのか

RQ2: メソッドの純粋性が変化した際に，同時にどのようなソースコード上の編集が行われたのか

RQ1 ではメソッドの純粋性が変化することを調べ，変化することが確認できた場合，加えてメソッドの純粋性の変化の回数などついて調査する．

RQ2 ではメソッドの純粋性が変化と同時に行われた修正内容を分類する．また修正と同時に不具合が混入したのかを確認し，純粋性の変化が不具合に及ぼす影響を調査する．

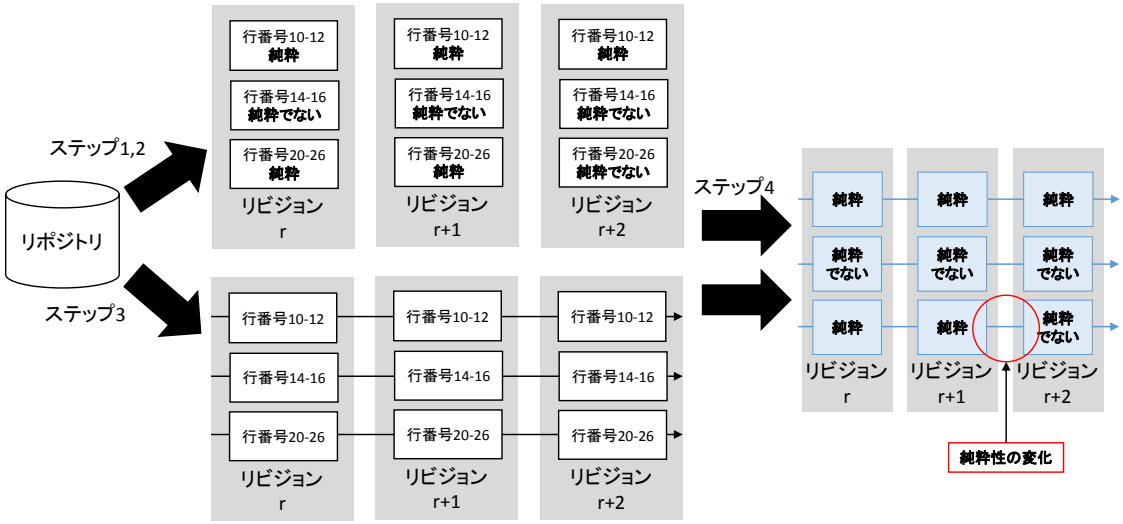

図 4: 純粋性の変化を検出する手法の概要

4　調査方法

この章では調査方法について説明する．

4.1　調査方法の概要

コミットによって純粋性が変化したメソッドを特定し，そのメソッドにコミットの前後で行われた修正内容を調査する．純粋でなくなったメソッドの修正内容を分類し，不具合に関するものかを目視で判断する．本研究では，調査を 4 つのステップに分ける．以下に各ステップを示す．

ステップ 1 リビジョンごとにソースコードをコンパイルし Java バイトコードを生成

ステップ 2 Java バイトコードからメソッドごとの純粋性を推定

ステップ 3 リビジョン間でのメソッドの対応付け

ステップ 4 純粋性が変化したメソッドの抽出

図 4 に本研究で用いる純粋性変化の検出方法の概要を示す．以降，それぞれのステップについて述べる．

4.2 ステップ 1：リビジョンごとにソースコードをコンパイルし Java バイトコードを生成

全てのリビジョンについて，リポジトリよりソフトウェアのソースコードを取得しコンパイルすることで Java バイトコードを得る．コンパイルには Ant や Maven などの自動ビルドツールを用いる．

4.3 ステップ 2：Java バイトコードからメソッドごとの純粋性を推定

メソッドの純粋性の推定には Yang らの作成した Purano [7] を用いる．Purano はコンパイル済みの Java バイトコードから，メソッドの純粋性を推定する．入力として対象とするソフトウェアのクラスファイルおよび依存ライブラリのクラスファイルを与えることで，Purano は Java バイトコードを静的解析し，フィールドやメソッドの依存関係を分析することでメソッドの純粋性を推定する．Purano の出力はメソッドの純粋性である．メソッドが純粋でない場合は，純粋でない理由を加えて出力する．ソースコードを同時に入力として与えることで，推定したメソッドのソースコード上の位置も出力する．

例として図 5 をコンパイルした Java クラスファイルとソースコードを Purano に入力として与えた結果の出力を，図 6 に示す．メソッドごとに以下の情報が出力される．

- シグネチャ (クラス名, メソッド名, 引数)
- ソースコードのファイルパスと，ソースコード上の位置 (開始行と終了行)
- メソッドの純粋性
- メソッドの呼び出し関係や，変数の依存関係

図 6 の `TestClass#<init>` は TestClass クラスのコンストラクタを表す．Purano はソースコード上で実装していない，コンパイラが自動生成したメソッドも解析する．ソースコードに対応するメソッドがない場合は，対応するファイル上の位置は `(0-0)` と出力する．

4.4 ステップ 3：メソッドの系譜を検出

ソフトウェアの開発において，メソッドのシグネチャや名前，ファイルの位置が変更される場合がある．そのため，コミットの前後でメソッドのシグネチャが変化した場合でも，同一のメソッドであることを特定する必要がある．実装されたリビジョンから削除される（あるいは最新の）リビジョンまでメソッドを追跡したものをメソッドの系譜と呼ぶ．

メソッドの系譜の検出には堀田らが作成した ECTEC [14] を用いる．ECTEC の動作を図 7 に示す．ECTEC は版管理システムのリポジトリを入力として，すべてのリビジョンに亘るコードブロックの追跡をする．コードブロックとは波括弧で囲まれた “{” から “}” までの

```
1  package test;
2  import java.util.Date;
3  public class TestClass {
4      int c;
5      public int add(int a, int b) {
6          return a + b;
7      }
8      public void setC(int c) {
9          this.c = c;
10     }
11 }
```

図 5: Purano のテスト用クラス (TestClass.java)

```
TestClass
void TestClass#<init> ()
(0-0)
Stateless
int TestClass#add (int, int)
C:\test\TestClass.java(5-7)
Stateless
@Depend()
void TestClass#setC (int)
C:\test\TestClass.java(8-10)
FieldModifier
@Field(type=int.class, owner=TestClass.class, name="c")
```

図 6: Purano の出力例 (一部)

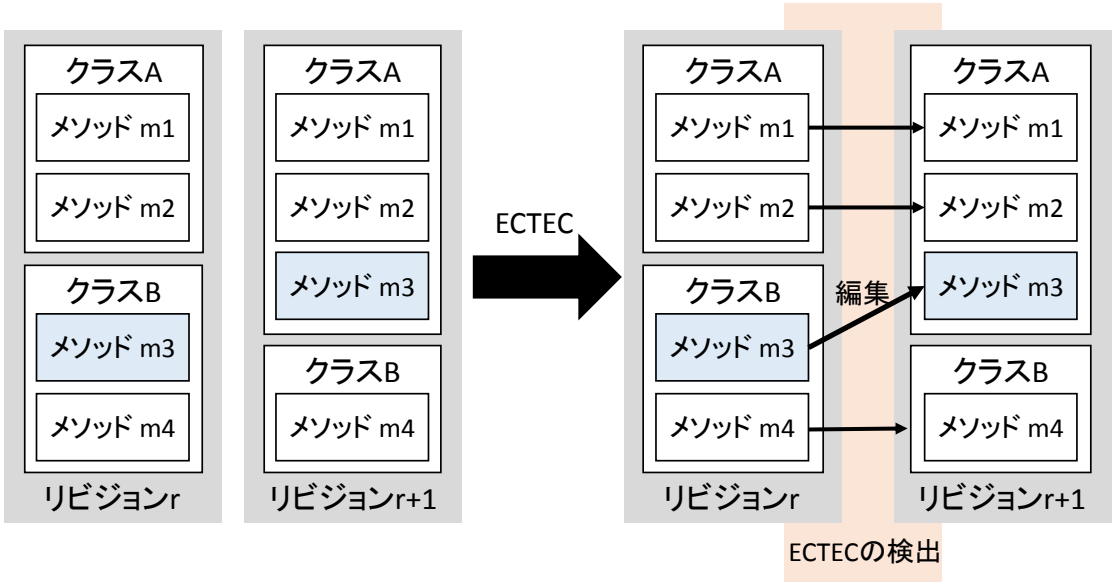

図 7: ECTEC の動作

コード上の範囲のことである．今回の調査ではメソッドのコードブロックを出力の単位として設定することで，メソッドの系譜を出力する．また，メソッドが編集されたコミットの特定が可能である．

4.5 ステップ 4：純粋性の変化を検出

ステップ 2 とステップ 3 の出力を用いて，系譜のリビジョンごとに純粋性を割り当てる．

ステップ 2 とステップ 3 の 2 つの出力には，リビジョンごとにメソッドの位置情報としてファイルパスと行番号が記述されている．ファイルパスと行番号が一致するメソッドを同一のメソッドとしてみなすことで，ステップ 2 とステップ 3 における同一のメソッドを特定し，メソッドの系譜に純粋性の情報を付加する．このようにして，すべてのメソッドの系譜を調べ，メソッドの純粋性が変化したコミットおよび系譜を検出する．

4.6 調査のための解析

RQ2 に答えるため，追加して行う解析を説明する．

全てのメソッドの系譜のうち，あるコミットの前後においてメソッドの編集が行われ，かつその編集によって純粋でなくなったメソッドの系譜を抽出する．このメソッドの系譜全体の編集を調べることで，純粋性を失ったコミットについて判断する．

メソッドの編集されたコミットを横断してコードの修正内容を確認できるツールを実装し

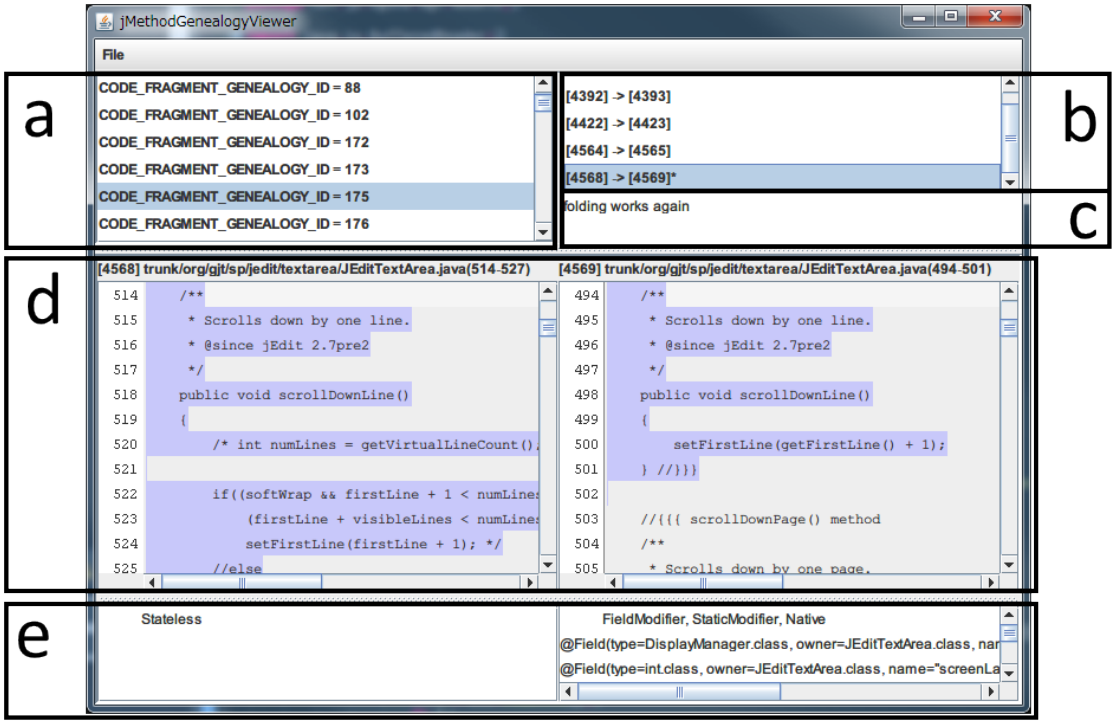

図 8: 純粋性の変化とソースコードを表示するツール

た．ツールの出力例を図 8 に示す．各領域に表示される項目を以下に示す．

a 調査すべきメソッドの系譜の一覧

b 選択したメソッドの系譜に含まれる，メソッドが変更されたコミット

c 選択したコミットのコミットログ

d 選択したコミット前後のソースコード．ハイライト部分は調査すべきメソッド

e コミット前後の純粋性

コミットに編集が行われた理由は著者らが目視で判断する．必要に応じてリビジョン全体のソースコードの取得や同時に編集されたファイルを参照し調査した．

5 調査の準備

5.1 調査手法の実装

調査手法の手順を自動的に行うためのツールを実装した．ソフトウェアのディレクトリ構成の変化に応じて動作を修正する必要があるため，動作の修正が容易な Python を用いた．現在のところ ECTEC が Subversion のみを対象としてるため，Subversion を用いているソフトウェアのみを調査対象とした．また，Purano および ECTEC は Java のみを対象としているため，本ツールが対象とするソフトウェアの開発言語は Java のみとした．ソフトウェアのリポジトリを入力として，純粋性が変化したメソッドとコミットの組の集合を出力する．

5.2 調査対象

本研究では，jEdit [15] と JFreeChart [16] の 2 つのオープンソースソフトウェアを対象に調査を行った．それぞれのソフトウェアの概要を表 1 に示す．“リビジョン数” は調査対象下のファイルが追加または削除，変更されたリビジョンの数を表している．“行数” は終了リビジョンにおける調査対象以下のソースコードの行数を表している．

これらはいずれも Java で実装されたソフトウェアであり，Subversion を用いて管理されている．Ant または Maven の自動ビルドツールを用いてソフトウェアをビルドすることができる．

表 1: 対象ソフトウェア

ソフトウェア	調査対象	開始リビジョン (日付)	終了リビジョン (日付)	リビジョン数	行数
jEdit	/jEdit/trunk/	3,789 (2001/9/2)	22,016 (2012/8/17)	5,302	183,093
JFreeChart	/branches /jfreechart-1.0.x -branch/	1 (2007/6/19)	2,527 (2013/1/28)	1,606	323,497

6　調査結果

6.1　RQ1: メソッドの純粋性は変化するのか

メソッドの純粋性が変化した系譜の数と割合　2つのソフトウェアについて，式1を求める．

$$\frac{\text{変化した系譜の数}}{\text{調査した系譜の数}} \tag{1}$$

結果を表2に示す．

純粋になる変化か純粋でなくなる変化か　表2で示した変化した系譜のうち，メソッドが純粋になった系譜の数と純粋でなくなった系譜の数，および2回以上変化した系譜の数を表3に示す．“2回以上変化”とは，1つの系譜の純粋性が2回以上変化したことを表しており，純粋への変化と純粋でなくなる変化のどちらも含む．

純粋性の変化はソースコードの修正を伴うか　メソッドの純粋性の変化が，そのメソッドのソースコードの修正を伴うのかを調べた．純粋性の変化のうち，純粋性の変化したメソッドとソースコードの修正が同時に行われた回数を表4に示す．

系譜の純粋性は何回変化するのか　jEditにおいてメソッド系譜の純粋性が変化した回数を図9に示す．JFreeChartにおいては2回以上変化した系譜は存在しなかった．

6.2　RQ2: メソッドの修正内容と純粋性の変化

メソッドの純粋性の変化と同時に行われたソースコードの修正内容を参照し，どのような修正が加えられたのかを目視で判断する．メソッドの純粋性が変化した理由を調査した結果を表5に示す．不具合の混入となる修正はほとんど見つからなかった．

jEditにおいて不具合が混入したメソッドの例を図10に示す．リビジョン19698のHyperSearchResult.javaの75行目 (図10の下線部) に標準エラー出力が追加された．出力処理の追加のためにメソッドが純粋でなくなったが，その後のコミットにおいてこの行は削除されており，著者らはこの修正内容を不具合の混入と判断した．

表 2: メソッドの純粋性が変化した系譜の数

ソフトウェア	調査した系譜	変化した系譜	メソッドの純粋性が変化した系譜の割合
jEdit	6,271	331	0.0527
JFreeChart	7,790	55	0.0070

表 3: メソッドの純粋性の変化の方向

ソフトウェア	1 回変化		2 回以上変化
	純粋になった	純粋でなくなった	
jEdit	59	115	157
JFreeChart	37	18	0

表 4: メソッド純粋性の変化とソースコードの修正

ソフトウェア	修正を伴う	修正を伴わない
jEdit	258	443
JFreeChart	18	37

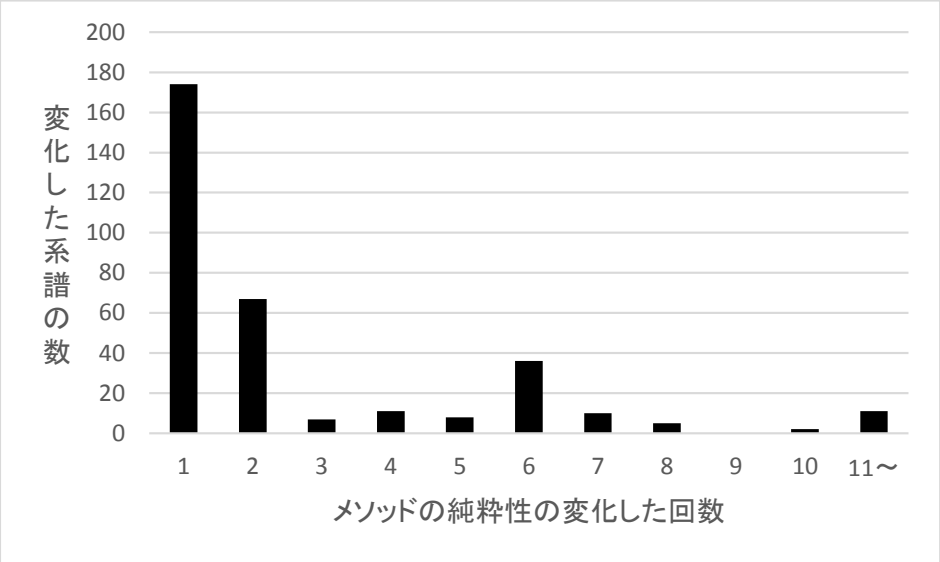

図 9: jEdit におけるメソッドの純粋性が変化した回数

表 5: メソッド純粋性の変化と修正内容

ソフトウェア	機能の追加	不具合修正	リファクタリング	不具合混入
jEdit	68	34	9	1
JFreeChart	5	5	0	0

```
56 //{{{ getSelection() method
57 /**
58 * Returns an array of selection objects pointing to the occurrences
59 * of the search term on the current line. The buffer must be opened
60 * first.
61 * @since jEdit 4.2pre5
62 */
63 public Selection[] getSelection()
64 {
65     if(buffer == null)
66         return null;
67
68     Selection[] returnValue = new Selection[occurCount];
69     Occur o = occur;
70     int i = 0;
71     while(o != null)
72     {
73         int start = o.startPos.getOffset();
74         int end = o.endPos.getOffset();
75         System.err.println("#" + i + ": startPos=" + o.startPos + "(=" + start + "),endPos=" +
               o.endPos + "(=" + end + ")");
76         Selection.Range s = new Selection.Range(
77             start,
78             end
79         );
80         returnValue[i++] = s;
81         o = o.next;
82     }
83     return returnValue;
84 } //}}}
```

図 10: jEdit において不具合が混入したメソッドの例

7　考察

RQ1 の表 2 から，実社会において開発されているソフトウェアにおいて，純粋性の変化が存在することが分かった．表 3 は変化した回数が 1 回の系譜の純粋性の変化が，純粋になったのか純粋でなくなったのかを表している．純粋になる変化も純粋でなくなる変化もどちらも起こりうることが分かった．表 4 はメソッドの純粋性の変化がその変化したメソッド自身のソースコードの編集を伴うのかを調べた．修正を伴わない純粋性の変化が多く行われており，開発者は気づかずに純粋性を変化させてしまうことを示している．図 9 からメソッドの純粋性の変化が複数回行われることが分かった．変化が 1 回のみの系譜は 52%であり，純粋が変化した系譜のうち，約半分は複数回変化する．

RQ2 の表 5 から，修正を伴い純粋性でなくなる変化が 1 件見つかった．この修正はデバッグ用に `System.err.println(String)` を追加することで純粋でなくなった．開発者はこの出力を消し忘れていたために後のリビジョンでこの標準エラー出力を削除した．

8　妥当性について

8.1　実験対象

jEdit，JFreeChart を対象として調査を行った．調査対象が異なれば調査結果が変わる可能性がある．

8.2　調査手法

不具合であるかは著者の目視によって判断している．著者は開発者でないため，判断に誤りが混入している可能性がある．判断の誤りを減らすため，不具合であるとの判断は複数人で行った．

内部クラスや無名クラス，インターフェースのメソッドについては，用いたツールの実装上調査できなかった．

メソッドの系譜の検出では，系譜が途中で途切れることを考慮していない．すなわち，あるコードクローンが一旦消滅した後，再度出現した場合，消滅前後の系譜はそれぞれ別の系譜として扱われる．

9 おわりに

本研究では，実社会における 2 つのオープンソースソフトウェアに対してリビジョン間におけるメソッドの純粋性の変化を調査し，同時に行われたソースコードの修正を分類した．純粋性の変化に関係した不具合の混入は見つからなかった．一方，粋性の変化と同時に不具合の修正が多く行われていることが分かった．

今後の課題としては，メソッドの純粋性を純粋・純粋でないの 2 値ではなく，Stateless な純粋・Stateful な純粋・純粋でないの 3 値に拡張することが考えられる．楊らは Java 標準ライブラリに対して純粋性の推定を行い，あるべき純粋性が Stateful な純粋である `FilePermission.hashCode()` が JDK1.6 において Stateless な純粋であることを見つけた [7]．JDK における過去の実装は Stateful であり，この例では 3 値による純粋性の変化の検出から不具合を見つけることが可能であることを示している．

今回の調査では用いたツールの実装上調査できなかったメソッドが存在するためこれらの調査や，他の複数のソフトウェアを対象に調査を行うことで，新たな知見が得られる可能性がある．

謝辞

本研究を行うにあたり，理解あるご指導を賜り，常に励まして頂きました，楠本真二教授に心から感謝申し上げます．

本研究に関して，有益かつ的確なご助言を頂きました，岡野浩三准教授に深く感謝申し上げます．

本研究において，多大なるご助言を頂きました，井垣宏特任准教授に深く感謝申し上げます．

本研究の全過程を通し，終始熱心かつ丁寧なご指導を頂きました，肥後芳樹助教に深く感謝申し上げます．

本研究に用いたツールの大部分を設計, 実装してくださり，また本研究に関して多大なるご助言，ご助力を頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士後期課程 2 年の楊嘉晨氏に深く感謝申し上げます．

本研究を行うにあたり，多大なるご助言，ご助力を頂きました，堀田圭佑氏に深く感謝申し上げます．

その他の楠本研究室の皆様からいただいたご助言やご協力に心より感謝致します．また，本研究に至るまでに，講義，演習，実験等でお世話になりました大阪大学基礎工学部情報科学科の諸先生方に，この場を借りて心から御礼申し上げます．

参考文献

[1] Grzegorz Czajkowski. Application isolation in the java virtual machine. In *ACM Sigplan Notices*, Vol. 35, pp. 354–366. ACM, 2000.

[2] B. W. Lampson, J. J. Horning, R. L. London, J. G. Mitchell, and G. J. Popek. Report on the programming language euclid. *SIGPLAN Not.*, Vol. 12, No. 2, pp. 1–79, Feb. 1977.

[3] Richard C. Holt and James R. Cordy. The turing programming language. *Commun. ACM*, Vol. 31, No. 12, pp. 1410–1423, Dec. 1988.

[4] Gary T Leavens and Yoonsik Cheon. Design by contract with jml, 2006.

[5] D 言語. http://dlang.org/.

[6] RUST. http://www.rust-lang.org/.

[7] Jiachen Yang, Keisuke Hotta, Yoshiki Higo, and Shinji Kusumoto. Revealing purity and side effects on functions for reusing java libraries. In *14th International Conference on Software Reuse*, Vol. LNCS 8919, pp. 314–329. Springer, Jan. 2015.

[8] Apache Subversion. https://subversion.apache.org/.

[9] CVS. http://www.nongnu.org/cvs/.

[10] git. http://git-scm.com/.

[11] 松下誠. ソフトウェア工学の新潮流 (1) リポジトリマイニング. ソフトウェアエンジアリング最前線 2009, pp. 21–24, Sep. 2009.

[12] Sunghun Kim, T. Zimmermann, Kai Pan, and E.J. Whitehead. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*, pp. 81–90, Sept 2006.

[13] David Pearce. JPure: a modular purity system for java. In Jens Knoop, editor, *Compiler Construction*, Vol. 6601 of *Lecture Notes in Computer Science*, pp. 104–123. Springer Berlin / Heidelberg, 2011.

[14] Yoshiki Higo, Keisuke Hotta, and Shinji Kusumoto. Enhancement of crd-based clone tracking. In *Proc. of the 13th International Workshop on Principles of Software Evolution (IWPSE2013)*, pp. 28–37, Aug. 2013.

[15] jEdit. http://sourceforge.net/projects/jedit/.

[16] JFreeChart. http://www.jfree.org/jfreechart/.