情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

シーケンシャルパターンマイニングに基づくオブジェクト指向プログラムのための欠陥検出手法

山田 吾郎[†1]　吉田 則裕[†1]　井上 克郎[†1]

シーケンシャルパターンマイニングとよばれるデータマイニングの手法をソースコードに適用し，その結果を用いた欠陥検出手法についての研究が行われている．しかし，それらをオブジェクト指向言語に適用した研究は確認されていない．本研究では，その欠陥検出手法をオブジェクト指向言語に適用した．実際に欠陥を検出するツールを作成し，オープンソースプログラムに適用したところ，欠陥を検出することができた．

A Defect Detection Method for Object-Oriented Programs using Sequential Pattern Mining

Goro YAMADA,[†1] Norihiro YOSHIDA[†1]
and Katsuro INOUE [†1]

Recently, several strudies apply sequential pattern mining, which is a kind of data mining to source code, and detect defects using that results. However, there are no case studies that apply this defects detection method to source code written in Object-Oriented languages. In this research, we apply this method to an Object-Oriented program. We have developed a tool that detects pattern violations, and applied to open source programs written in Java language. The resultant violations involved a defect.

1. まえがき

データマイニングの手法の 1 つとして**シーケンシャルパターンマイニング**[2] (Sequential Pattern Mining) が挙げられる．これは，順序を持つ要素のシーケンス (系列) の集合から，頻出する部分シーケンスを抽出する手法である．

シーケンシャルパターンマイニングを用いることで，ソースコード中のメソッド定義の集合から，**メソッド呼び出しパターン** (頻出するメソッド呼び出しの系列) を抽出する手法がいくつか提案されている[11)12)]．図 1 は，このような手法を用いることで抽出されるメソッド呼び出しの系列を表している．図 1 では，5 つのメソッド定義において，メソッド呼び出しの系列 $[open, read, close]$ が頻出している．このため，これら 5 つのメソッド定義からは，メソッド呼び出しパターン $[open, read, close]$ が抽出される．

メソッド呼び出しパターンが，実装を行う上でのルールを表しているものを含む可能性が

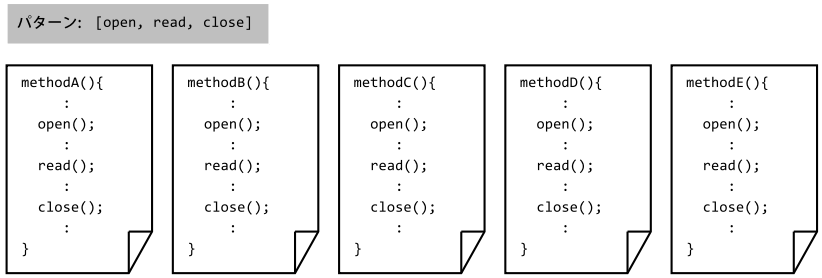

図 1 メソッド定義の集合から抽出されたシーケンシャルパターンの例
Fig. 1 Example of a sequential pattern that is exracted from a set of method definitions.

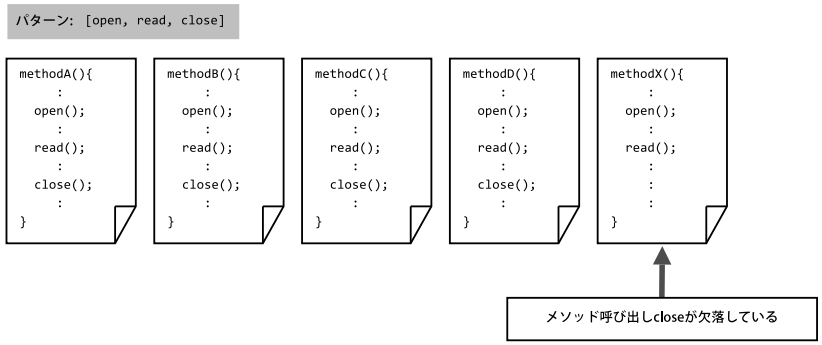

図 2 メソッド定義の集合から検出されたパターン違反の例
Fig. 2 Example of a pattern violation detected from a set of method definitions.

†1 大阪大学 大学院情報科学研究科
Graduate School of Information and Science Technology, Osaka University

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

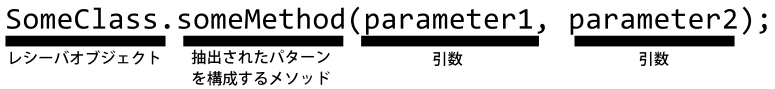

図 3 パターン違反検出の際に考慮する型
Fig. 3 Types considered when detecting pattern violations

指摘されている[9)12)]．このことに着目して，メソッド呼び出しパターンに違反するメソッド呼び出しの系列を含むコード片を検出することにより，欠陥の検出を行う研究が行われている[8)9)]．これらの研究では，あるメソッド呼び出しパターンの出現回数と比べて，その部分シーケンスの出現回数が著しく少ない場合に，その部分シーケンスをパターン違反として検出する．図 2 と同じく 5 つのメソッド定義を表しているが，*methodX* のメソッド定義で *close* のメソッド呼び出しが欠落している．このようなソースコードは，メソッド呼び出しの欠落による欠陥が生じている可能性が考えられる．

しかし，従来の研究で行われたパターン違反を用いた欠陥検出の事例は，手続き型言語である C 言語により記述されたプログラムを対象としており，近年普及してきているオブジェクト指向プログラムへの適用事例は我々が調査した限り存在しない．

本研究では，メソッド呼び出しパターンのパターン違反を用いた欠陥検出を，オブジェクト指向型言語である Java 言語で記述されたプログラムに適用した．従来のパターン違反検出は C 言語で記述されたプログラムを対象としていたため，関数の識別に関数名のみを用いていたが，本研究は Java 言語で記述されたプログラムを対象とするため，メソッドの識別にメソッド名に加え，**型情報**（レシーバクラスの型，引数の型の系列）を用いた（図 3）．適用実験では，Java 言語で記述されたプログラムから，メソッド呼び出しパターンに違反するコード片を検出するツールを作成し，オープンソースソフトウェアの 1 つである JDT Core[7)] のソースコードから，型情報を考慮する場合としない場合の 2 通りで検出を行った．その結果，型情報を考慮して検出したパターン違反の中のみに欠陥が含まれていた．

以降，2 節ではパターン違反を用いた欠陥検出の説明に必要な概念および関連研究について述べる．3 節では，本研究で提案する手法について説明する．4 節では，適用実験について述べた後，その考察を行い，最後に 5 節で本研究のまとめと今後の課題について述べる．

2. 背　　景

本節では，頻出パターンマイニングに基づいたパターン違反による欠陥検出に必要な概念を述べた後，関連研究について述べる．

2.1 メソッド呼び出しパターン

メソッド呼び出しパターンとは，ソースコード中のメソッド定義からメソッド呼び出し・制御構造を取り出した系列に対し，データマイニングの一種である**シーケンシャルパターンマイニング**を適用することで抽出される系列である．シーケンシャルパターンマイニング[2)]とは，順序付き列（シーケンス）の集合から，一定回数以上出現するシーケンスを抽出（マイニング）することを指す．シーケンシャルパターンマイニングにより得られたシーケンスは，シーケンシャルパターンとよばれる．本稿ではメソッド呼び出しパターンを単にパターンとよぶ．シーケンス中でパターンに一致する部分をパターンのインスタンスとよぶ．また，シーケンスの集合 S 中でパターン P が出現する数を，S におけるパターン P のサポート値と言う．

2.2 パターン違反

パターン違反とは，シーケンシャルパターンマイニングにより得られたパターンをルールとみなしたとき，そのルールに違反するインスタンスを指す．パターン違反を含むソースコードは，欠陥を含む可能性があることが指摘されている[8)9)]．例えば，図 2 の例では，*methodX* の定義中には，パターンとして抽出されたシーケンス $open, read, close$ が存在せず，*close* が欠落した部分シーケンスである $open, read$ がパターン違反として検出される．このパターン違反は，*close* メソッドの欠落による欠陥を示している可能性があると考えられる．

2.2.1 パターン違反の検出手法

パターン違反は相関ルール[1)] の確信度を用いて検出する．相関ルール $P_1 \Rightarrow P_2$ とは，シーケンス S において，パターン P_1 が存在したとき，パターン P_2 も存在すると言うことを表す．相関ルールの確からしさは確信度 (confidence) とよばれ $C(P_1 \Rightarrow P_2)$ と表記し，以下の式 1 のように条件付確率で表すことができる．式 1 は，相関ルール $P_1 \Rightarrow P_2$ の確信度が，シーケンス中にパターン P_1 が出現したときに，パターン P_2 が出現する条件付き確率であることを示している．

$$C(P_1 \Rightarrow P_2) = \frac{P_1 \text{と } P_2 \text{が同時に出現するシーケンス数}}{P_1 \text{のサポート値}} \ (0 \leq C \leq 1) \qquad (1)$$

パターン違反の検出手法では，相関ルール $P_1 \Rightarrow P_2$ の確信度 $C(P_1 \Rightarrow P_2)$ が **1 ではない十分大きな値**であるとき，$P_1 \Rightarrow P_2$ に違反するシーケンス (P_1 は出現するが P_2 出現しないシーケンス) をパターン違反として検出する．

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

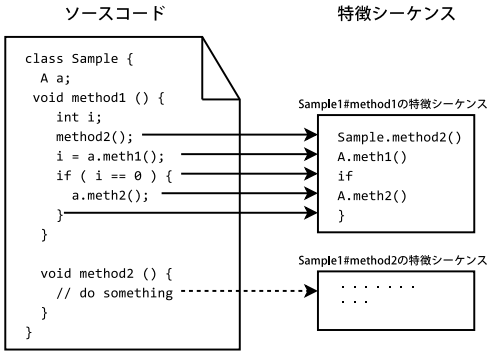

図 4 特徴シーケンスの例
Fig. 4 Example of characteristic sequences

2.3 関 連 研 究

本節では，本研究でメソッド呼び出しパターンの抽出に用いる Fung，および本研究で提案する手法を考案するにあたって参考にしたパターン違反検出法について述べる．

2.3.1 メソッド呼び出しパターン抽出ツール Fung

Fung[6] は Java 言語を対象としたシーケンシャルパターンマイニングの抽出ツールである．Fung はソースコード中のメソッド定義から，メソッド呼び出しのメソッド名と制御構造 (if 文などの条件文や while 文などの繰り返し文の開始・終了位置) からなる特徴シーケンスを入力のシーケンスとし，特徴シーケンスの集合に対しシーケンシャルパターンマイニングを行う．

Fung では，シーケンシャルパターンマイニングの実装に PrefixSpan アルゴリズム[10] を用いている．ソースコード中の全てのメソッド定義から特徴シーケンスを作成したのち (図 4)，それらの特徴シーケンスに対して PrefixSpan アルゴリズムによるシーケンシャルパターンマイニングを行う．マイニングの際には任意の最小サポート値 (パターンの最小出現回数) を設定する．以下で PrefixSpan アルゴリズムの概略を示す．

手順 1. それぞれの特徴シーケンスを構成している全ての要素のサポート値を計算する．

手順 2. 任意に設定した最小サポート値を越える要素を，シーケンシャルパターンとして出力する．

手順 3. 閾値を越える各要素について射影を行う．射影とは，全てのシーケンスから特定の要素に続く接尾辞を取り出す操作である．

手順 4. 手順 3 の射影により得られた特徴シーケンスに対し，再び手順 1 のからの操作を繰り返す．

2.3.2 パターン違反に基づく欠陥検出を行った研究

Li ら[9] は C 言語で記述されたプログラムの各関数から，関数呼び出しや変数の参照からなるパターンを検出し，それらパターンに違反する関数を欠陥を含む関数の候補として提示した．この研究ではパターンマイニングに，アイテムセットマイニング (item-set mining) を用いている．アイテムセットマイニングは，シーケンシャルパターンマイニングと異なり，順序を考慮しない．適用実験では，Linux カーネルのソースコードから欠陥を検出することができた

Kagdi ら[8] は，Linux カーネルなど，C 言語で記述された複数の大規模プログラムに対しアイテムセットマイニングとシーケンシャルパターンマイニングをそれぞれ適用し，欠陥検出と言う観点で有意なパターン違反の検出精度を比較した．その結果，シーケンシャルパターンマイニングの方がパターン違反の検出精度において優れていることが明らかになった．

2.4 オブジェクト指向プログラムへの適用にあたっての問題点

オブジェクト指向プログラムからパターン違反を検出するにあたって，C 言語で記述されたプログラムが対象の場合にはなかった問題点が生じる．C 言語では，同一の関数名であれば，同一の関数を指すため，Li らの手法や Kagdi らの手法は，関数呼び出しを，関数名のみで識別していた．一方，オブジェクト指向型言語では，同名のメソッドを定義することができる．そのため，メソッド名のみでメソッドを識別すると，異なるメソッドを同一のものとみなす可能性がある．これは検出漏れを引き起こす可能性がある．

考えられる検出漏れの 1 つに次のようなものがある．パターンのインスタンスを誤って多く抽出してしまい，相関ルールの左辺のサポート値が増加する．それに伴い確信度が低下し，パターン違反とみなす閾値に満たせず検出漏れが起こる場合である．

図 5 に例を挙げた．パターン違反とみなす閾値を 0.7 としたとき，2 つのパターン $P1, P2$ の相関ルール $P1 \Rightarrow P2$ についてのパターン違反を考える．

パターン $P1$ は，正しくメソッドを識別できた場合，$methodA, methodB, methodC, methodX$ の 4 箇所で出現することになる．一方，パターン $P2$ は，$methodA, methodB, methodC$ の 3 箇所で出現しており，相関ルールの確信度は式 1 より 0.75 となる．これは閾値より大きいため，パターン $P1$ のメソッド定義 $methodX$ 中におけるインスタンスはパターン違反とみなされる．

しかし，メソッド名のみ考慮する手法では，先ほどの 4 つのメソッド定義に加え，実際は

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

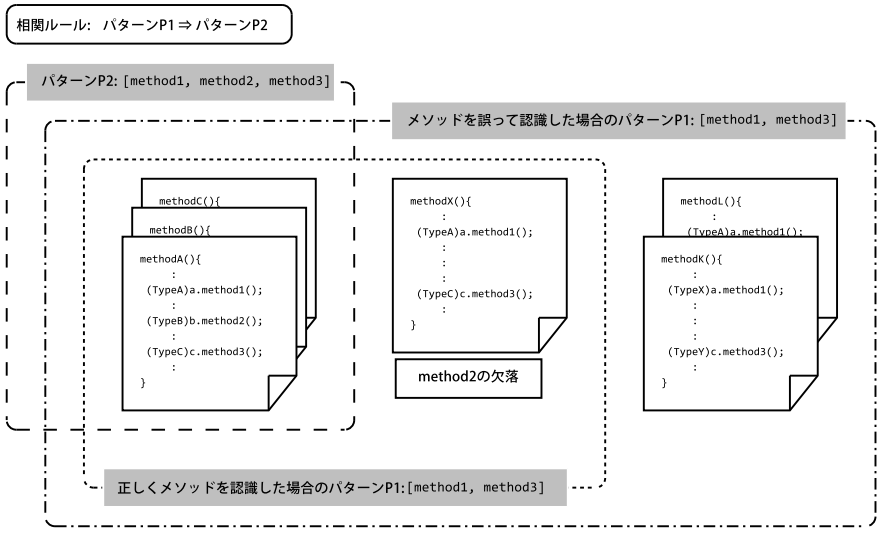

図 5 サポート値の増加に伴う確信度の減少による検出漏れの例
Fig. 5 Example of a false negative caused by incresing support values and corresponding confidence values

異なるメソッドを呼び出している 2 つのメソッド定義，$methodK, methodL$ も含まれてしまう．この増加に伴い，パターン P1 のサポート値が増加し，相関ルールの確信度は 0.5 となる．結果，閾値を下回ってしまいパターン違反として検出されない．

3. 提 案 手 法

本研究では，パターン違反検出の際，メソッド呼び出しのメソッド名に加えメソッドに関連する型情報 (図 3) も利用することで，前節 2.4 で述べたメソッド名のみでメソッドを識別すると，異なるメソッドを同一のものとみなす可能性がある問題の解決をはかる．

図 6 に本手法を用いたパターン違反検出の概要を示す．以下でそれぞれの処理の簡単な説明を行う．

処理 1．Fung を用いたメソッド呼び出しパターンの抽出　対象プログラムのソースコードを入力に Fung を実行する．抽出したメソッド呼び出しパターンは，型情報とともに XML ファイルに出力される．

処理 2．メソッド呼び出しパターンの型による分類　Fung から得たメソッド呼び出しパターンを，型情報を用いてインスタンス毎に分類する．これにより型情報を考慮したメソッド呼び出しパターンを得る．

処理 3．分類されたメソッド呼び出しパターンのグループ化　相関ルールを作成するために，型情報を考慮したメソッド呼び出しパターンをグループ化する．グループとは，メソッド呼び出しパターンとそのサブパターンからなる集合である．

処理 4．相関ルールの確信度を用いたパターン違反検出　それぞれグループについて，グループ内の型情報を考慮したメソッド呼び出しパターン間における相関ルールの確信度を計測し，パターン違反を検出する．

以降，3.1, 3.2, 3.3, 3.4 節で手法の詳細を説明する．

3.1 処理 1．Fung を用いたメソッド呼び出しパターンの抽出

2.3.1 節で述べたツール Fung を用い，メソッド呼び出しパターンの抽出を行う．ただし，Fung には若干の機能追加を行っており，従来の出力に加えメソッドに関連する型情報も出力する．メソッド呼び出しパターンを表す XML ファイルは大まかに次のような木構造をなす．なお，末尾に $*$ と記した要素はその要素が 0 回以上出現することを示し，$+$ は 1 回以上の出現を意味する．

- メソッド呼び出しパターン $+$
  - インスタンス $+$

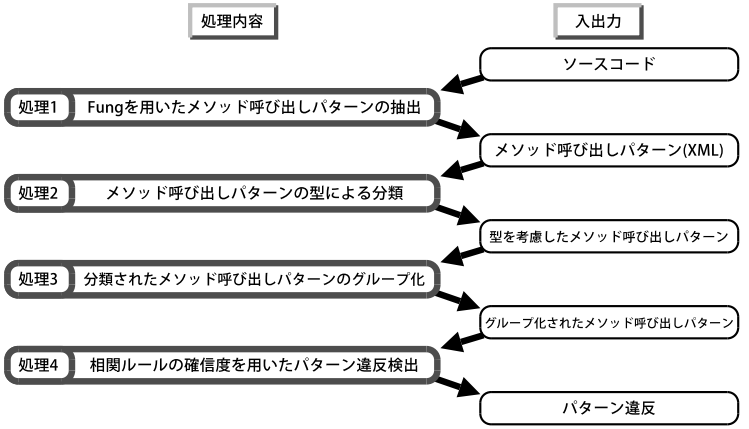

図 6 提案手法の概要
Fig. 6 Overview of our propose method

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

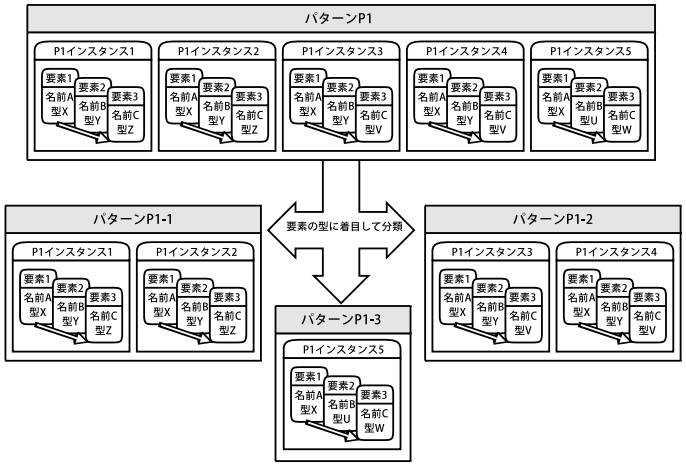

図 7 型によるメソッド呼び出しパターンの分類
Fig. 7 Classification of method call patterns based on type analysis

- インスタンスの出現ファイル
- インスタンスの出現クラス (オーナークラス)
- インスタンスの出現メソッド (オーナーメソッド)
- インスタンスの要素 +
  - メソッド名および制御構造の種類 (条件文，繰り返し文)
  - ソースコード中でインスタンスの要素が出現する行と列

制御構造の種類は次の 2 種類である．

- ループ
- 分岐

さらに，インスタンスの要素がメソッド呼び出しであった場合，上記の 2 つの子要素に加えメソッド呼び出しに関連する型情報も持つ．

- 型情報 (メソッドのみ)
  - レシーバクラスの型
  - 引数の型 *

3.2 処理 2. メソッド呼び出しパターンの型による分類

3.1 節で出力したメソッド呼び出しパターンを，XML に含まれるメソッド呼び出しに関連する型情報を用いて分類を行う．メソッド名のみ考慮して抽出したメソッド呼び出しパターンを，型情報を考慮して分類を行うことで，メソッドの識別をより厳密に行う．

3.3 処理 3. 分類されたメソッド呼び出しパターンのグループ化

細分化された，型情報を考慮するメソッド呼び出しパターンをグループ化する．グループとは，あるパターンと，そのサブパターンからなるメソッド呼び出しパターンの集合である．

グループ化は確信度の低い相関ルールを作成しないために行う．なぜなら, 2.2.1 節で述べたように，パターン違反を検出するためには相関ルールの確信度を用いるため，どの 2 つのメソッド呼び出しパターンを相関ルールとするか決定する必要があるからである．確信度が低い場合は閾値によりパターン違反とみなされない．また，確信度は 2 つのメソッド呼び出しが同じメソッド定義で出現する割合と言える．したがって，同じメソッド定義で現れると保証されているパターン同士の確信度を作成すべきである．サブパターンには，その元となるパターンの出現するメソッド定義に必ず出現するため，グループ内のメソッド呼び出しパターン間で作成した相関ルールの確信度は，それ以外のルールと比較して高くなると考えられる．そのため，グループ内で相関ルールを作成することで，低い確信度になると考えられる相関ルールの作成を避けることができる．

3.4 処理 4. 相関ルールの確信度を用いたパターン違反検出

グループ内に含まれる全てのメソッド呼び出しパターンについて相関ルールを生成し，式 1 によりその確信度を求める．確信度が閾値以上であり 1.0 でなければパターン違反とみなす．

4. 適 用 実 験

3 節で述べた手法を実現するツールを実装し，適用実験を行った．本節ではその内容および結果と，結果に対する考察を述べる．

4.1 実 験 目 的

実験目的は以下の 2 つである．

目的 1. オブジェクト指向プログラムにおいても，手続き型言語と同様，メソッド呼び出しパターンのパターン違反による欠陥検出が有効か確かめる．

目的 2. 型情報を考慮することで，2.4 節で述べたように，型情報を考慮しない場合に検出できなかったパターン違反が検出できるようになるか確かめる．

4.2 評 価 方 法

Fung を用いて，JDT バージョン 2 の Core コンポーネント[7](以下 JDT Core と表記す

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

る) からメソッド呼び出しパターンを抽出し，型情報を考慮する有効性を評価するため，型情報を考慮しない場合とする場合との 2 通りでパターン違反の検出を行い，以下の項目について調査を行った．

- **目的 1.** を検証するため，実際に欠陥を含んだパターン違反が検出されるかどうか調べる．
- **目的 2.** を検証するため，型情報を考慮した場合としなかった場合とで検出した欠陥の差分を調査し，型情報を考慮することの有効性を調べる．

実験には，Intel Xeon X5472×2，メモリ 16GB，OS が FreeBSD 7.1 の計算機を用いた．

4.3 実験対象

適用対象の JDT(Java Development Tools) とは，オープンソースの統合開発環境である Eclipse[5] に標準で付属する，Java 言語の開発ツールを提供するプラグインである．用いたのは 2002 年 11 月 26 日にコミットされたバージョンである．実験に際し，プログラムの機能に直接関係のないテストコードは除外した．対象の規模は表 1 に記載した．行数は，テストコードを除いた数値である．

4.4 実験結果

型情報を考慮する場合としない場合とでの，マイニングとパターン違反検出に要した時間，およびメソッド呼び出しパターン，グループ，パターン違反の数を表 2 に記す．メソッド呼び出しパターンの最小サポート値は 30，メソッド呼び出しパターンの構成要素の最小値を 4 とし，パターン違反とみなす相関ルールの確信度の最小値を 0.9 とした．また，検出されたすべてのパターン違反を調査し欠陥の数を調べた．

4.5 考察

前節 4.4 の結果に基づき，4.2 節で述べた検証をそれぞれ 4.5.1, 4.5.2 節で行う．

4.5.1 オブジェクト指向プログラムに対するメソッド呼び出しパターンのパターン違反を用いた欠陥検出の有効性の評価

実験により，型情報を考慮した場合において 1 つの欠陥が発見された．以下でその欠陥について説明を行う．

行数	ファイル数	メソッド数
334,595	1,654	9,668

表 1 JDT Core の行数とファイル数，メソッド数
Table 1 Number of lines, files, and methods in JDT Core

パッケージ: org.eclipse.jdt.internal.compiler.codegen
ファイル : ConstantPool.java

欠陥を含むメソッド定義 : literalIndexForJavaLangObjectGetClass

```
  writeU1(NameAndTypeTag);
  writeU2(nameIndex);
  writeU2(typeIndex);
}
index = wellKnownMethods[GETCLASS_OBJECT_METHOD] = currentIndex++;
                          if文のブロックが欠落している
writeU1(MethodRefTag);
writeU2(classIndex);
writeU2(nameAndTypeIndex);
```

正しいメソッド定義 : literalIndexForJavaLangClassForName 他29メソッド

```
  writeU1(NameAndTypeTag);
  writeU2(nameIndex);
  writeU2(typeIndex);
}
index = wellKnownMethods[FORNAME_CLASS_METHOD] = currentIndex++;
if (index > 0xFFFF){
  this.classFile.referenceBinding.scope.problemReporter().
    noMoreAvailableSpaceInConstantPool(
      this.classFile.referenceBinding.scope.referenceType()
    );
}
writeU1(MethodRefTag);
writeU2(classIndex);
writeU2(nameAndTypeIndex);
```

図 8 欠陥を含むコード片と正しいコード片
Fig. 8 Defective and correct fragments

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

```
[writeU1 → writeU2 → writeU2 → writeU1 → writeU2 → writeU2]
⇒
[writeU1 → writeU2 → writeU2 → problemReporter → referenceType
 → noMoreAvailableSpaceInConstantPool → writeU1 → writeU2 → writeU2]
```

図 9 パターン違反を生じていた相関ルール
Fig. 9 Violated association rule

実験により得られた欠陥を図 8 に示す．出現したファイル名は ConstantPool.java である．この欠陥は，図 9 のような相関ルールも含め計 192 のルールに違反している．

以下，図 9 の相関ルールの左辺のメソッド呼び出しパターンを $P1$，右辺のメソッド呼び出しパターンを $P2$ とよぶ． $P1$ はそのインスタンスが合計 31 回出現しているのに対し，$P2$ は 30 回しか出現していない．さらに，$P1$ が出現する全てのメソッド定義で $P2$ も出現している．パターン違反は，右辺のメソッド呼び出しパターンが出現するメソッド定義の集合から，左辺のメソッド呼び出しパターンが出現するメソッド定義の集合を引いた差集合に存在する左辺のパターンのインスタンスである．したがって，この場合は図 8 上段に示した，メソッド定義 `literalIndexForJavaLangObjectGetClass` 中に存在する， $P1$ のインスタンスがパターン違反となる．

図 8 に示したように，メソッド定義 `literalIndexForJavaLangObjectGetClass` 以外の 30 箇所では，`index` に値が代入された後，`index` の値が 16 進数で `FFFF` を上回っていないかチェックし，上回っていればエラー処理を行っている．しかしメソッド定義 `literalIndexForJavaLangObjectGetClass` ではこれが行われていない． `index` に渡される値を追跡したが，値が `FFFF` を上回らないための処理などは見当たらないことから，欠陥であると判断した．

	型情報を考慮しない場合	型情報を考慮する場合
マイニングに要した時間	161.61 秒 (共通)	
違反検出に要した時間	4.90 秒	3.24 秒
メソッド呼び出しパターン	260	121
グループ	56	13
パターン違反	456	295
欠陥	0	1

表 2 JDT Core から抽出したメソッド呼び出しパターンとグループ，パターン違反，欠陥の数
Table 2 Number of method call patterns, groups, pattern violations and defects in JDT Core

ただし，JDT の後のバージョンで，この欠陥を含むファイルに大規模な改変が加えられ，欠陥の出現したメソッド定義も含め多数のメソッド定義が削除されていた．そのバージョンでのコミットログは改変のみに言及しているため，開発履歴情報からはこのパターン違反が欠陥であると言う確証は得られなかった．

しかし，実験で見つかった欠陥のようなコードの欠落による欠陥は起こりうるものである．したがって，オブジェクト指向プログラムにおいても，手続き型プログラムと同様に，シーケンシャルパターンマイニングにより抽出されたメソッド呼び出しパターンに対して，パターン違反を用いての欠陥検出が可能であることを示すのには十分な例であると考える．

実験では，型情報を考慮しない場合で検出できる欠陥が，型情報を考慮する場合に検出できなかった例はなかったが，実際には十分起こり得る．検出漏れが起こってしまう原因はいくつか考えられる．例えば，型情報を考慮する際にメソッド呼び出しパターンのインスタンスの数が減少してしまい，最小サポート値に満たない場合などが挙げられる．

4.5.2 型情報を考慮することの有効性の評価

適用実験では，表 2 に記載したように，型情報を考慮しなかった場合に検出できなかった欠陥が，型情報を考慮することで検出できるようになった．これは，2.4 節で述べた，相関ルールの左辺のメソッド呼び出しパターンに型の異なるメソッド呼び出しが混入することで出現するメソッド定義が増加し，相関ルールの確信度が低下してしまうこと原因である．

また，メソッド呼び出しパターン $[bind \rightarrow bind \rightarrow append \rightarrow bind \rightarrow append]$ は，型情報を考慮しない場合のインスタンス総数が 32 であるが，そのうち 1 つのインスタンスのレシーバクラスの型が異なる．そのため，このメソッド呼び出しパターンを左辺に持ち，その他のインスタンスのみを持つメソッド呼び出しパターンを右辺に持つ相関ルールで，確信度が高くなってしまい，誤検出の増加につながっている．

いずれの場合も，型情報を考慮することで解決された．ゆえに，型情報を考慮してパターン違反検出を行うことは，型情報を考慮しない場合に対して利点が存在すると言える．

5. ま と め

シーケンシャルパターンマイニングを用いて得られたメソッド呼び出しパターンに対し，メソッド呼び出しパターンのパターン違反から欠陥を検出する手法がある．この手法は手続き型言語にのみ適用例が確認されているが，本研究ではオブジェクト指向プログラムに適用し，欠陥を検出できることを示した．オブジェクト指向プログラムへの適用に際し，メソッドを識別できないと言う問題が考えられたが，メソッド呼び出しパターンを構成するメソッ

Vol.2009-SE-164 No.15
Vol.2009-EMB-13 No.15
Vol.2009-CSEC-45 No.15
2009/5/29

ド呼び出しのレシーバクラス，および引数の型情報を考慮する手法を提案し，有効性を確認した．

今後の課題としては，適用したプログラムが現在 JDT Core のみであるため，他のプログラムにも適用実験を行うことで，提案手法の評価を重ねることが挙げられる．また，評価実験において，欠陥かどうかの判断をソースコードを追い目視で行った．しかし，評価の信頼性を向上させるためには，Bugzilla[3] などの欠陥追跡システムの情報を用いて，欠陥候補が欠陥として報告されているか調査する必要がある．ただし，欠陥が報告されていたとしても，対象となっているバージョンが古いため入手できず，調査が行えない可能性がある．iBUGS[4] は，Bugzilla を元に構築された欠陥に関する履歴のデータベースであり，欠陥の報告や修正前後のソースコードを閲覧することができる．iBUGS を用いることで，パターン違反により検出した欠陥候補が，実際に欠陥として報告されているか確認することができる．

本手法では，メソッド呼び出しに関連する型を型階層を無視して比較している．そのため，共通の親クラスでのメソッドを子クラスがそれぞれ呼び出していて，かつ子クラスの両方が親クラスで宣言されていない場合，同じメソッドを呼び出しているにも関わらず異なるメソッドと判断され，パターン違反などの検出漏れが起こってしまうと考えられる．したがって，型情報を考慮する際に型階層を辿りながら比較を行う機能が必要である．

謝辞 Fung の開発を行った大阪大学 大学院情報科学研究科 伊達 浩典氏，三宅 達也氏，石尾 隆氏に深く感謝する． 本研究は，日本学術振興会 科学研究費補助金基盤研究 (A)(課題番号: 21240002)，特別研究員奨励費 (課題番号:20・1964) の助成を得た．

参 考 文 献

1) Agrawal, R. and Srikant, R.: Fast Algorithms for Mining Association Rules, *Proc. of VLDB*, Santiago de Chile, Chile, pp.487–499 (1994).
2) Agrawal, R. and Srikant, R.: Mining sequential patterns, *Proc. of ICDE 1995*, Taipei, Taiwan, pp.3–14 (1995).
3) Bugzilla: `http://www.bugzilla.org/`.
4) Dallmeier, V. and Zimmermann, T.: Extraction of bug Localization Benchmarks from History, *Proc. of ASE 2007*, Atlanta, GA, USA, pp.433–436 (2007).
5) Eclipse: `http://www.eclipse.org/`.
6) Fung: A pattern mining tool for java method calls: `http://sel.ist.osaka-u.ac.jp/~ishio/fung/`.
7) JDT: `http://www.eclipse.org/jdt/`.
8) Kagdi, H., Collard, M.L. and Maletic, J.I.: Comparing Approaches to Mining Source Code for Call-Usage Patterns, *Proc. of MSR 2007*, Los Alamitos, CA, USA, pp.123–130 (2007).
9) Li, Z. and Zhou, Y.: PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code, *Proc. of ESEC/FSE 2005*, Lisbon, Portugal, pp.306–315 (2005).
10) Pei, J., Han, J., Mortazavi-Asl, B., Pinto., H., Chen, Q., Dayal, U. and Hsu, M.-C.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, *Proc. of ICDE 2001*, Heidelberg, Germany, pp.215–224 (2001).
11) 中山 崇，松下 誠，井上克郎：ソースコードの差分を用いた関数呼び出しパターン抽出手法の提案，情報処理学会研究報告， Vol.2006, No.35, pp.49–56 (2006).
12) 石尾 隆，伊達浩典，三宅達也，井上克郎：シーケンシャルパターンマイニングを用いたコーディングパターン抽出，情報処理学会論文誌， Vol.50, No.2, pp.860–871 (2009).