クラス名における命名バグの検出手法

有村 徳崇
大分大学
v1653005@oita-u.ac.jp

紙名 哲生
大分大学
kamina@oita-u.ac.jp

要旨

オブジェクト指向におけるクラスは多くの場面で再利用される．しかし，クラス名がその機能を正しくユーザに伝えられなければ，ユーザはそのクラスの機能を理解するのに失敗し，再利用性を損なう．本研究では，クラス名に命名バグがあるかどうかをそのクラスの設計者に伝えるための検出手法の開発を試みた．まず，あるクラスにおいてその意味内容とクラス名からユーザが読み取る意味内容が異なるならばクラス名に「命名バグ」があるとして，クラス名における命名バグを「クラスメンバにあるものをクラス名が表現できていない場合」と「クラス名が示すものがクラスメンバに存在しない場合」に大別する．次に，クラス名とクラスメンバの特徴をソースコードから抽出し，それを予め準備された「クラスカテゴリ」（クラス名とクラスメンバの特徴に関するルール）の集まりと順次比較する．抽出されたクラス名とクラスメンバの特徴と、クラスカテゴリで示したルールのうちのどれかの間に矛盾があった場合，命名バグとしてユーザに報告する．オープンソースソフトウェア（*OSS*）をもとにデザインパターンに関するいくつかのクラスカテゴリを作成し，その妥当性を評価した．

1. はじめに

オブジェクト指向プログラミングは現在も多くのソフトウェア開発で用いられている．オブジェクト指向的開発環境は様々なソフトウェアシステムに共通のニーズを満たす再利用可能なクラスを提供し，オブジェクト指向で開発されたクラスは多くの場面で再利用される．しかし，クラス名がその機能を正しくユーザに伝えられなければ，ユーザにそのクラスを理解するための負担がかかり，再利用性を損なう．既存の命名規則は命名に統一性を与えるが，意味のある適切な命名を保証しない．

本研究では，クラス名に命名バグがあるかどうかをそのクラスの設計者に伝えるための検出手法の開発を試みた．まず，あるクラスにおいてその意味内容とクラス名からユーザが読み取る意味内容が異なるならばクラス名に「命名バグ」があるとして，クラス名における命名バグを「クラスメンバにあるものをクラス名が表現できていない場合」と「クラス名が示すものがクラスメンバに存在しない場合」に大別する．次に，クラス名とクラスメンバの特徴をソースコードから抽出し，それを予め準備された「クラスカテゴリ」（クラス名とクラスメンバの特徴に関するルール）の集まりと順次比較する．そして，抽出されたクラス名とクラスメンバの特徴とクラスカテゴリで示したルールのうちのどれかの間に矛盾があった場合，命名バグとしてユーザに報告する．

本研究では，本手法の有効性を検証するため，以下の二つの適用実験を行った．

1. 実際のオープンソースソフトウェア（OSS）のソースコードを用いて，作成したクラスカテゴリの妥当性を検証する実験
2. 作成・検証されたクラスカテゴリを用いて命名バグを検出する実験

1. については，デザインパターンに関するいくつかのクラスカテゴリと実際のクラス定義において，クラス名に含まれる単語とクラス要素に含まれる単語に高い頻度で重複のあることが確認できた．2. については，1. で作成した Visitor パターンのクラスカテゴリを用いて，簡単なサンプルプログラムにおいて命名バグの自動検出に成功した．

本論文の構成は以下のとおりである．2 節で関連研究

について紹介する．3 節で本研究で提案する命名バグの検出手法について詳説する．4 節で実装した命名バグ検出ツールを紹介し，それを用いて行った適用実験の結果について報告する．最後に，5 節で結論を述べる．

2. 関連研究

2.1. 命名バグに関する研究

Høst らは，メソッド名の命名バグを検出する手法を実現した [6]．この手法ではメソッド名を抽象化し，その抽象化したメソッド名が満たすべき機能を求め，それをルールとして運用することで命名バグを検出する．Binkley らは，品詞タグ付けの技術を用いて不適切な命名がなされているクラスメンバの検出を行った（例えば動詞で始まるフィールド名など）[4]．これらの手法はいずれも，メソッド名には通常動作を表す品詞が用いられるなどの経験則を用いて，その動作を抽象化して機能やルールの分類を行う．一方でクラス名の場合，それが表すのは通常名詞であるため，これらの手法をそのまま本研究に適用することはできない．そこで本研究では，クラス名とクラスメンバに基づいたカテゴリを考え，クラスがカテゴリに属する場合にクラス名とクラスの内容が同時に満たすべきことを記述し，それをルールとして運用することで命名バグを検出する．

Arnaoudova らは，命名に関するアンチパターン [5] を定義し（例えば “set” メソッドが値を返すなど），ソースコード中からそのアンチパターンに従うコードを検出するツールを提案した [2]．アンチパターンとは，否定的な結果に導くような一般的にみられる開発方法を記述した形式のことである．ここでは，メソッドやフィールドの命名についてのアンチパターンは議論されているものの，クラス名についての議論はない．また，この検出には予め定義されたアンチパターンが必要になる．本研究でも予め定義したクラスカテゴリが必要になるが，このカテゴリはクラスメンバに出現する用語の頻度などの比較的単純な統計情報に基づいており，こうしたアンチパターンと比べて自動生成が容易になると考えられる．

2.2. 命名を支援する手法に関する研究

Allamanis らはメソッドやクラスの命名を推薦する手法を提案している [1]．命名の推薦は一般的に局所的な情報のみを用いて行うことは難しく，ここではソースコード中における命名を対象とした確率ニューラル言語モデル [3] に基づいて，ソースコード中の文脈における命名法を学習する．このような推薦と命名バグの検出は相互に補間するものであり，本研究では，命名の推薦は行わない代わりに適切でない命名が行われているクラスを検出する．

```
public class TestObj{
    public boolean visit(Element e) {
        // ...
    }
}
```

図 1. クラスメンバにあるものをクラス名が表現できていない命名バグ

3. 本研究のアプローチ

上述したとおり，メソッド名やフィールド名における命名バグの検出についてはこれまでに一定の成果があがっているものの，クラス名については筆者らの知る限りほとんど取り組まれてきていない．一つには，クラスはメソッドやフィールドよりも抽象度が高く，品詞やアンチパターンなどを用いた分類が難しいからであると考えられる．しかしながら，クラス名はその機能を正しくユーザに伝える重要な役割を持っており，不適切な命名はクラスの再利用性を損うことに繋がる．

本研究では，クラス名に命名バグがあるかどうかを，そのクラスの設計者に伝えるための検出手法について提案する．

3.1. クラスにおける命名バグ

クラス名とそのクラスの定義は対応するべきである．このことを「クラス名がある特徴的な要素を持つならばそれに応じた要素を持つクラスメンバを持つ」,「クラスメンバがある特徴的な要素を持つならばそれに応じた要素を持つクラス名を持つ」という仮定に落とし込み，これらの仮定に基づき命名バグを検出する．

ここでクラス名における命名バグを「クラスメンバにあるものをクラス名が表現できていない場合」と「クラス名が示すものがクラスメンバに存在しない場合」に大

```
public class TestVisitor{
    public void print() {
        // ...
   }
}
```

図 2. クラス名が示すものがクラスメンバに存在しない命名バグ

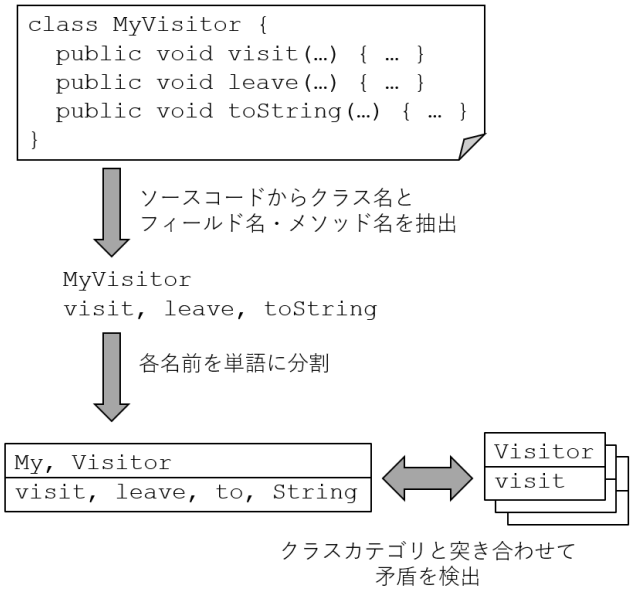

図 3. 提案手法の概要

別する．図 1 は，クラスメンバにあるものをクラス名が表現できていない場合を示した例である．この例では visit することがこのクラス名からはわからない．デザインパターンの一つである Visitor パターンを用いているならば Visitor という単語を含むべきである．visit という操作をされるとしてもこのクラス名では visit という特徴的な操作を想定できず不適当である．図 2 は，クラス名が示すものがクラスメンバに存在しない場合を示した例である．この例では，クラス名に Visitor とあるがこのクラスは visit しない．

3.2. アプローチの概要

図 3 に提案手法の概要を示す．まず，検査対象となるソースコードを構文解析することで各クラスのクラス名と，全てのフィールド名とメソッド名を得る．次に，これらクラス名，フィールド名，メソッド名それぞれを単語に分割する．単語への分割の際には，Java で用いられることの多いキャメルケースで書かれた名前を対象に，大文字ごとに区切る方法を採用した．大文字ごとに区切るだけでは対応できない名前（例えば “XML” のように大文字のみから構成される単語を含む場合）については，今回は除外し考慮しないことにした．なお，ここではクラス名を構成する単語を**命名要素**，フィールド名やメソッド名を構成する単語をそのクラスの**特徴的要素**と呼ぶことにする．

こうして得られた命名要素及び特徴的要素を，別途用意した複数のクラスカテゴリと突き合わせることで命名バグの有無を検査する．別の言い方をすると，クラスカテゴリとは，クラスの命名に関して「ある命名要素を含むクラスのフィールド名やメソッド名には，関連する特徴的要素を持たなければならない」というルールを記述したものであり，そのルールに則ってクラス名が付けられているかどうかを検査する．

命名要素	Visitor
特徴的要素	visit, Visit

図 4. Visitor パターンが期待する命名要素と特徴的要素を表すクラスカテゴリの例

3.3. クラスカテゴリ

本研究では，クラス名とクラスメンバの命名に関する対応関係をまとめるためにクラスをカテゴライズする．各カテゴリには，ある命名要素に対して期待される特徴的要素が情報としてまとめられており，それをファイルに保存する．これは一種の単純な命名パターンであると見なすこともでき，先述のアンチパターンを用いた方法とは逆の方法を用いている．つまり，クラス名においては明確なアンチパターンを探すことが難しいため，逆に「あるべきパターン」に矛盾する命名を検出するという方法をとっている．

例として，クラスカテゴリ「Visitor」を考える（図 4）．デザインパターンの Visitor パターンを採用したクラスにはクラス名の単語として Visitor，クラスメンバにはメソッドとして visit という特徴的な要素がある．クラスカテゴリ「Visitor」はこれらを内容とする．

3.4. ルールの適用

ソースコードを構文解析することによって得られた各クラスの命名要素と特徴的要素の組を，複数のクラスカテゴリと突き合わせ，両者の間の矛盾の有無を調べることによって命名バグの有無を検査する．つまり，クラス名の要素とクラスメンバの要素の両方で，クラスカテゴリに設定された特徴的要素に当たる要素が存在する（あるいは要素の両方でそのクラスカテゴリに設定された特徴的要素に当たる要素が一切存在しない）場合にはそのクラスメンバを持つクラスのクラス名はそのカテゴリにおいて命名バグに当たらないと判断し，それ以外の場合において命名バグを報告する．

なお本研究では，メソッド名は適切に付けられていることを想定している．メソッド名は適切に付けられているにもかかわらずクラス名が不適切な場合としては，例えばプログラムの改修を進めていくうちにクラスの役割りが変わっていき、名前が適切でなくなるケースなどが有り得る．

4. 命名バグ検出ツールの実装

本手法の有効性を検証するため，実際に命名バグを検出するツールを実装した．実装したツールは主に以下の二つの機能に分けられる．

- コードベースにあるソースコードをもとに，クラスカテゴリの生成を助ける機能
- クラスカテゴリを用いて，検査対象のソースコードから命名バグを検出する機能

なお，今回対象としたプログラミング言語は Java である．構文解析には統合開発環境の Eclipse に付随する JDT の ASTParser クラスを用いた．

4.1. クラスカテゴリの作成

クラスカテゴリの生成を補助するため，クラス名の特定の単語とクラスメンバ名の特定の単語がどれほどの頻度で重複して用いられているかを調べるツールを実装した．このツールは，ユーザが設定したクラスカテゴリが妥当なものかどうかを調べるため，コードベースにあるソースコードとクラスカテゴリの比較を行う．例えば，コードベース上で，あるクラスカテゴリが表す命名要素と特徴的要素（メンバ名で用いられる単語群）の間に高い頻度での重複を確認することができれば，そのクラスカテゴリが示す命名規則は広く用いられているということを意味するので，そのクラスカテゴリは妥当であると考えることができる．

命名要素	Iterator, Iterable
特徴的要素	List, list, Next, next, Remove, remove, Has, has

図 5. Iterator パターンのカテゴリ

今回は，図 4 で示した Visitor パターンに関するカテゴリのほか，Iterator パターンに関するカテゴリを準備した（図 5）．コードベースには，GitHub 上にあり，Java で書かれていて，評価数が多くソースコード量の多い物として，elasticsearch[1]，google/guava[2]，TheAlgorithms/-Java[3]，RxJava[4]を用いた．コードベースに含まれる Java ソースコードのファイル数はおよそ 19800 個になる．またコードベース中には，単語 “Visitor” を含むクラスは 41 個，単語 “Iterator” を含むクラスの数は 214 個，単語 “Iterable” を含むクラスは 88 個存在した．

これらのカテゴリをコードベース上で試した結果を表 6 に示す．この表からは，命名要素 Visitor と Iterator の二つにおいて，カテゴリ名とクラスメンバの特徴的要素が高い頻度で重複していることが確認できる．Visitor については，一つのクラス中に単語 “visit” を含むメソッドが複数存在した．よって，Visitor と Iterator についてはこれらのクラスカテゴリは世の中の命名規則をよく反映していると言える．一方 Iterable については，ある程度の重複が確認できるものの，必ずしもこれらの特徴的要素を持たないクラスも多く存在する．

また，コードベース中でこれらのクラスカテゴリに従わなかったクラス名（つまり本手法で命名バグだと検出されるケース）は，“Visitor” を含むクラス名で 7 個，“Iterator” を含むクラス名で 53 個，“Iterable” を含むクラス名で 53 個存在した．これらの中には，例えば抽象的な Visitor を継承しているクラスなどが多く存在しており，必ずしも命名バグであるとは言えない．現時点では，本手法はクラスの構文解析から得られる情報しか用いておらず，スーパークラスの情報などを活用できてい

[1] https://github.com/elastic/elasticsearch
[2] https://github.com/google/guava
[3] https://github.com/TheAlgorithms/Java
[4] https://github.com/ReactiveX/RxJava

命名要素と特徴的要素の対応	マッチング回数
Visitor-visit	657
Iterator-next	187
Iterator-Next	144
Iterator-remove	73
Iterator-List	46
Iterator-has	105
Iterator-Remove	26
Iterator-Has	12
Iterable-Next	13
Iterable-next	4
Iterable-remove	7
Iterable-Remove	1
Iterable-Has	3
Iterable-has	3
Iterable-List	4

図 6. コードベース上におけるクラスカテゴリのマッチング結果

ない．今後はクラス階層解析などの結果を用いて本手法を拡張したうえで，改めて検出精度の評価などを行っていく予定である．

4.2. 命名バグの検出

人工的に命名バグを含むプログラムを作り、その命名バグを検出できるかを確認する実験を行った．デザインパターンの Visitor パターンに基づいたカテゴリを設定し，例 1，例 2 のように想定した二種の命名バグを含むクラスの Java ソースコードをそれぞれ作成した．これらを対象に命名バグ検出を行なった．クラスカテゴリファイル「Visitor.csv」にクラス名の特徴的要素「Visitor」とクラスメンバの特徴的要素「visit」を設定した．命名バグを含むクラスは「testVisitor というクラス名で visit という単語を持つメンバを持たないクラスの Java ソースコード」と「visit というメソッドを持つ Visitor という単語を含まないクラス名のクラスの Java ソースコード」の二つを用意した．

これらクラスカテゴリファイルと Java ソースコードを開発したツールに与え，命名バグ検出処理を実行した．結果は「クラス名 testVisitor が特徴的要素 Visitor を持つが，そのカテゴリ Visitor に応じる要素を持つクラスメンバがない」という命名バグと「クラスメンバ visit が特徴的要素 visit を持つが，そのカテゴリ Visitor に応じる要素をクラス名が持たない」という命名バグを検出することができた．

4.3. 妥当性への脅威

本研究で実施した命名バグの検出は、簡単なプログラムを対象に、意図的に挿入された命名バグが実際に検出できるかどうかを確認したものである．実際にユーザが書いたプログラム上でこうした命名バグの検出が役に立つのかどうかについては、今後実証的な実験を重ねて確認していく必要がある．

今回設定したクラスカテゴリについては，その妥当性についてコードベース上で評価を行なっている．このコードベースは評価数やソースコード量の多い OSS を用いて作られており，実際に広く用いられているソフトウェアである．したがってソースコードの品質の高さにおいては一定以上のレベルにあると考えられるため，評価結果については信頼のおけるものであると筆者らは考えている．しかしながら，対象としたソフトウェアは４種類と未だ少なく，それぞれ異なった分野から選択されてはいるものの，例えば GUI アプリケーションを含んでいないなど，分野にはまだ偏りがあるため，コードベースをさらに充実させた上でさらなる検証が必要であると考えられる．

また，今回設定したクラスカテゴリはいずれもデザインパターンに関するもので，命名方法について一定の慣習があるものから選ばれている．一方で，こうした明らかな慣習は存在しないものの，不適切な命名によってプログラマにクラスの機能について誤解を与えるケースは様々であると考えられる．例えば，ドメイン分析などからわかる命名規則がある場合，それに従わないような命名を検出できるかなど，様々なケースに適用できる一般性を本研究で示したアプローチが持っているかどうかについて，今後検討を重ねていく必要がある．

なお，本研究はメソッドは適切に命名されているという前提に立っている．メソッド名が不適切につけられているようなケースでは，本研究のアプローチは必ずしも成功しない．ただし，メソッド名の命名バグについてはメソッド本体を調べることによって検出できる場合があることが知られており [6]，こうした手法と組み合わせることによって提案手法の有用性を高めることができる

と考えられる．

5. 結論

オブジェクト指向で開発されたクラスは多くの場面で再利用され，クラス名がその機能を正しくユーザーに伝えられなければ、ユーザーにそのクラスを理解するための負担がかかり，再利用性を損なう．そこでクラスの意味内容とそのクラス名から読み取れる意味内容が異なる場合にそのクラス名には「命名バグ」があるとし，クラス名とクラスメンバの関連性から命名ルールを導き出すことを目指した．

本研究では，クラス名とクラスメンバは対応関係にあると仮定し，この対応関係をクラスカテゴリとしてまとめ，このクラスカテゴリをルールとして運用してクラス名の命名バグ検出するツールを開発した．実験の結果，少なくとも一部のデザインパターンについてはこうした対応関係について確認することができ，それを用いて命名バグを実際に検出できることが示された．

5.1. 今後の課題

このツールを活用するには十分な数のクラスカテゴリファイルが必要である．しかし現段階ではごくわずかな数しか用意できていない．そのため，実験で示された結果も限定的なものとなっている．今後はより多くのコードベースを用いて様々なクラスカテゴリを設定し，実際のユーザのコードに適用させるなどの実験を重ねていく必要がある．そのための一つの方策として，クラスのカテゴライズの自動化方法を検討している．

クラスカテゴリを用いた命名バグの検出方法としては，現時点での構文解析から得られる情報だけでなく，クラス階層構造などを含めたより広範な静的プログラム解析から得られる情報をもとにしたより精度のよいものを目指したうえで，改めて評価を行うことを検討している．

参考文献

[1] Miltiadis Allamanis, Earl T. Baar, Christian Bard, and Charles Sutton, Suggesting accurate method and class names. *ESEC/FSE'15*, pp.38–49, 2015.

[2] Venera Arnaoudova, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc, A new family of software anti-patterns: linguistic anti-patterns. *2013 17th European Conference on Software Maintenance and Reengineering*, pp.187–196, 2013.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, A neural probabilistic language model. *Journal of Machine Learning Research*, vol.3, pp.137—1155, 2003.

[4] Dave Binkley, Matthew Hearn, and Dawn Lawrie, Improving identifier informativeness using part of speech information. *MSR'11*, pp.203—206, 2011.

[5] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley and Sons, Inc, Canada, 1998.

[6] Einar W. Høst and Bjarte M. Østvold, Debugging method names. *ECOOP 2009 – Object-Oriented Programming*, LNCS 5653, pp.294–317, 2009.