4L-1

オブジェクト指向プログラム改善箇所判断のための特徴量抽出

齋藤　拓弥†　　松浦 佐江子‡
芝浦工業大学システム工学部電子情報システム学科†‡

1. はじめに

近年，ソフトウェアの大規模化・複雑化に伴い，オブジェクト指向が注目され，本学においてもオブジェクト指向教育が行われている．しかしオブジェクト指向言語を用いて作られたプログラムであっても必ずしもオブジェクト指向の特徴が生かされているわけではない．そのためにオブジェクト指向の利点である保守性や拡張性が失われてしまうという問題がある．

本研究ではまずオブジェクト指向の特徴であるカプセル化と継承に着目し，その改善箇所の発見を目的とする．その際，リファクタリング[1]を行う際に注目する箇所に基づき，カプセル化と継承の観点からプログラムの不適切箇所を特定するための特徴量の決定と抽出を行う．

2. リファクタリング

リファクタリングとは，ソフトウェアの外部的振る舞いを変えずに内部構造を改善する作業である．

リファクタリングによりオブジェクト指向の利点を生かしたプログラムに改善できる場合がある．表 1 に本稿で扱う二つのオブジェクト指向の特徴に関係するリファクタリングと，その際に着目する箇所をまとめた．これをもとにカプセル化されていないことと，継承の不適切な使用を 3，4 節で定義し，5 節でそれを判断する特徴量と判断方法を示す．

表 1. カプセル化・継承に関係するリファクタリング

	リファクタリング名	着目箇所
カプセル化 (グループ化)	メソッドの移動	所属するクラスから使用される回数と他のクラスから使用される回数
	フィールドの移動	
	クラスの抽出	クラス内のフィールドへアクセスするメソッド
	メソッドの抽出	メソッドの責務の大きさ
カプセル化 (隠蔽)	フィールドのカプセル化	アクセス修飾子とアクセス元となるクラス，メソッド
	メソッドの隠蔽	
継承	階層の平坦化	基底クラスとサブクラスのメソッド

3. カプセル化されていない定義

カプセル化には大きく分けてデータと手続きのグループ化と，情報の隠蔽という二つの目的がある．

グループ化に関するリファクタリングに，使用頻度の多いクラスへ特性を移動するメソッドの移動・フィールドの移動がある．また，責務の大きなメソッドやクラスは，ひとまとめにできる部分を抽出し，新たなクラスやメソッドとして定義するクラスの抽出・メソッドの抽出がある．

A Method to Extract Features
for Improved Object Oriented Programs
† Takuya Saito　‡ Saeko Matsuura
†‡ Shibaura Institute of Technology
Department of Electronic Information Systems

よって適切なグループ化がなされていないとは，これらのリファクタリングを行う必要がある箇所であり，次の 3 つと定義する．

3-1 所属するクラスからよりも他のクラスから使用されることが多い特性を持つ
3-2 メソッド内で多くの処理を行っている
3-3 メソッドとフィールドの関連が分割できる

ここで言う特性とは，クラス内のフィールドとメソッドの総称である．また，メソッド内の処理とは，他のオブジェクトにメッセージを送る回数である．メソッドとフィールドの関連とは，同じクラス内においてあるフィールドを共有するメソッド同士は関連しているとすることで，関連していないメソッドとフィールドは分割できるということである．

隠蔽に関しては，公開されているフィールドや他のクラスで使用されていないメソッドのアクセス制限を強めるフィールドのカプセル化・メソッドの隠蔽が関連している．これをもとに隠蔽がなされていないことを次のように定義する．

3-4 クラスの特性が必要以上に公開されている

4. 継承の不適切な使用の定義

継承とは既に定義されているクラスに変更や拡張を加えた新たなクラスを定義することである．継承に関するリファクタリングにサブクラスを基底クラスにまとめる階層の平坦化がある．これはサブクラスと基底クラスの振る舞いがほぼ同じ場合に，クラスを統合することである．本稿ではサブクラスと基底クラスの振る舞いを比較する場合に，基底クラスのメソッドの内容をサブクラスで書き換えるメソッドのオーバーライドに着目し，不適切な使用を次のように定義する．

4-1 基底クラスのメソッドをサブクラスでほとんどオーバーライドしている

5. 特徴量とその判断

3, 4 節での定義で用いたリファクタリングで着目する箇所をもとに，それぞれの定義を判断する特徴量を以下のように決定した．例えば 3-1 では，メソッドの移動を用いる時に着目する箇所はクラスの各特性へのアクセス回数であり，所属するクラスと外部クラスからのアクセス回数を比較することでリファクタリングを行うので，特徴量は表 2 の①と決定した．

表 2. 抽出する特徴量

	特徴量
カプセル化	①クラスの各特性にアクセスするメソッドとアクセス回数 ②各メソッドから他メソッドを呼び出す回数 ③各特性のアクセス修飾子
継承	④継承階層 ⑤基底クラスのメソッドのオーバーライド回数

これら①～⑤の特徴量を用いて，カプセル化・継承を判断する方法を次に示す．

3-1 の判断には①を用いる．まずクラスの各フィールド，

メソッドにアクセスするメソッドとその回数を抽出し，所属するクラス内からのアクセス回数と外部クラスからのアクセス回数を比較し，外部からのアクセスが多い場合にはフィールドの移動・メソッドの移動を行う．

3-2 の判断には②を用いる．あるメソッドから他のメソッドを呼び出す回数を抽出し，多すぎる場合にはメソッドの抽出を行う．

3-3 の判断には①と②を用いる．同一クラス内のメソッド同士が共有するフィールドを持たない場合には，クラスの抽出を行う．

3-4 の判断には①と③を用いる．各特性の所属するクラスと特性にアクセスするクラスの関係から，フィールドのカプセル化やメソッドの隠蔽を行う．

4-1 の判断には④と⑤を用いて，基底クラスのメソッドをほとんどオーバーライドしている場合には階層の平坦化を行う．

6. 抽出実験

実際に特徴量を抽出し，その特徴量から改善箇所が判断できるかを検証する実験を行った．また，オブジェクト指向の特徴が生かされていると考えられる Java クラスライブラリからも特徴量を抽出し，実験対象のプログラムから抽出したものと比較することで，特徴量の妥当な指標を得ることを検討した．

6.1 実験対象と方法

本稿における抽出対象は，本学 3 年次で行う情報実験Ⅱで作成された自動販売機のシミュレータのプログラムとした．クラス数は約 100 個，ライン数は約 7800 行である．

特徴量の抽出はツールを作成し，自動化を行った．抽出する特徴量はメソッドやコンストラクタの内部の処理を解析する必要があるため，構文解析を行った．一度 Java ソースコードから javaML[2]と呼ばれる XML 形式に変換する既存のツール[3]を用いて変換した後，必要な特徴量を XML 文書から抽出する部分を自動化した．

6.2.抽出結果

実際に情報実験Ⅱで作成したプログラムから特徴量を抽出し，考察する．

まず①を抽出した結果，最もよく使用されたメソッドは標準出力を行うメソッドで計 208 回のアクセスがあった．これはクラスライブラリのクラスを使用したものであったので移動はできないが，よく使用されるクラスではメソッドとして抽出しまとめることが考えられる．

次に②を抽出した．各メソッドごとに他のメソッドを呼び出す回数を見ると，最も多いものでのべ 103 回あった．Java クラスライブラリでは他のメソッドへのアクセスは 1～20 回程度が最も多く，100 回を超えるものはほとんど無い．そのため定義 3-2 より，このクラスの責務が大きすぎると判断できる．実際にプログラムを見ると，長さが 200 行以上ある大きなメソッドであり，103 回のうち標準出力のメソッドを除いた最も多くメッセージを送っているメソッドに注目してみると，if 文や switch 文の中で多く使用される重複したコードであった．

次に定義 3-4 について①と③を抽出した．特に public フィールドについては，5 つのクラスで計 24 個使用されていたが，これらに直接アクセスする他のクラスは無く，static や final といった修飾子もついていなかった．また，外部クラスからは get メソッドを用いて参照が行われていた．このことから，public フィールドのアクセス制限を強めるべきであることがわかる．これ以外では，他のメソッドから全く使用されないメソッドがいくつか存在した．例えば他から使用されない set メソッドやインスタンスを返す get メソッドを持つクラスは，意図せず他のクラスからフィールドの値を変更されてしまう可能性がある．しかしライブラリなどのコンポーネントを目的としたプログラムでは意図的に作られている場合がるため，一概に判断はできないと考えられる．

継承に関しての④，⑤は，情報実験Ⅱのプログラムでは GUI に関する Swing コンポーネントと Thread のみであり，オーバーライドは行われていなかった．また，スーパークラスの特性を全く使用していない継承も見られなかった．しかし，インターフェースの実装部分が共通する部分多く，継承を使うべき箇所を判断することはできなかった．理由としては，抽象化を行ううえで類似部分の発見のための特徴量を定義できなかったことが挙げられる．

6.3 プログラム改善による特徴量の変化

6.2 の結果をもとにプログラムのリファクタリングを行った．メソッドの責務が大きかったものは，最も多く外部へのメッセージを送る箇所に着目し，メソッドの抽出や移動を行うことで，リファクタリング前に比べて，各クラスの特性に対して，同一クラス内のメソッドからのアクセスが多くなり，逆に外部クラスからのアクセスは減少した．これによりクラス内の関連が強まりモジュール強度が上がったと判断できる．その他の特徴量についても，情報の不必要な隠蔽が無くなり，プログラムの改善ができたと考える．

表 3．特徴量の変化

特徴量	増減・変化
①	減少
②	減少
③	制限が強まった
④，⑤	変化なし

7. まとめと今後の課題

情報実験Ⅱのプログラムや Java クラスライブラリを調べた結果，抽出した特徴量からプログラムの改善すべき箇所を見つけることができた．しかし特徴量によって改善すべき箇所を発見したとしても，実際にはプログラム内における意味を考慮しなければならない場合がある．さらにメソッドの分割や移動はかならずしもできるわけではなく，今回の特徴量だけではその判断は主観に依存することになる．

また，プログラムの性質によって特徴量に違いがみられる．完結したプログラムではアクセスされる回数が 0 回というメソッドは冗長なコードであるが，クラスライブラリのようなプログラムでは外部プログラムに使用されることを想定して書かれるメソッドが多い．

今後は他のオープンソースのプログラムからの特徴量の抽出と，この特徴量以外で，特に継承とその他のオブジェクト指向の特徴を判断するために必要な特徴量がないか検討したい．

参考文献

[1] Martin Fowler, Refactoring Improving the Design of Existing Code, Addison-Wesley, 2000

[2] Greg J Badros：JavaML:A Markup Language for java source Code（2000）

[3]JJmlt:http://www.hpc.cs.ehime-u.ac.jp/~aman/project/JJmlt/