4L-2

リファクタリング適用箇所抽出による Java プログラム作成支援

田中 幹人†　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，ソフトウェア開発における大規模なプログラムの増加に伴い，オブジェクト指向言語に基づく保守性および再利用性に優れるプログラムの開発が求められている．プログラムの再利用性および保守性を高めるための有効な改善手法の一つとして挙げられるのが，M.ファウラーの提唱する 72 のリファクタリングである[1]．リファクタリングとは，外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を変化させることである．プログラムの改善を行う際には，まずプログラム内の振る舞いを改善者が理解しなければならない．しかしその振る舞いが理解しにくい形であれば，振る舞いを変えない範囲で改善者自身の解釈を示すようなプログラムへの改善が必要となる．

本研究では 72 のリファクタリングから，改善者自身の解釈を示す形に変更する上で有効なリファクタリングを定める．そしてそれらについて適用すべき箇所の発見を自動で行い，さらにそのリファクタリングに必要な情報の提示を行う．これらによってより良いプログラム作成の支援を行うことを目的とする．

2　リファクタリングとその問題点

2．1　リファクタリングの手順

M.ファウラーの提唱する 72 のリファクタリングはそれぞれ名前，要約，動機，手順，例の５つの部分から構成されている．ユーザーは改善を行いたいソースコードに対し，まず動機を元にそのリファクタリングを行うべき箇所を発見する．そして手順に従ってリファクタリングを行うことにより，ソースコードの改善を行うことができる．リファクタリングを行う際には必ずテストを行う．これによってソフトウェアの外部的振る舞いが変わらないことを保証できる．

2．2　手順を踏む上での問題点

リファクタリング「条件記述の統合」を例に問題点を考察する．

リファクタリング名：条件記述の統合

動機：同じ結果をもつ一連の条件判定があるならば，それらを１つの条件記述にまとめる．
手順：その条件記述が副作用を持っていないことを確かめる．（…以下の手順省略）

図２.１ リファクタリング「条件記述の統合」

まずこのリファクタリングを適用しようと考えた場合，プログラム内のソースコード全てから動機に示されるような箇所を発見しなければならない．これは多量のコードでは大変困難になる．また手順においては「副作用を持っていないことを確かめる」とある．例えば図 2-2 のような文を統合した場合は NullPointerException が発生しプログラムが正常動作しなる．
以上から，問題点として以下に示すものが挙げられる．

```
String name;
double disabilityAmount(){
   if(name==null) return 0;
   if(name.equals("defaultName")) return 0;
```

図２.２ 「条件記述の統合」適用箇所例

・動機をもとに適用箇所を探すことが簡単とは限らない．
・手順に従ってリファクタリングを行うためには上記の例のように様々な情報が必要とされその抽出はプログラムの規模によっては大きな労力となる．

また個々のリファクタリングは，各手順と成果の粒度が非常に小さいため各リファクタリングを組み合わせて使うことが常である．ゆえに

・大きな目的に対し，どのようなリファクタリングをどのような順序で行えば目的を達成できるのか明確でない．

といった課題も挙げられる．

3　問題解決のための支援方法

3．1　問題に対する解決法の考察

２章で挙げた三つの問題点について，それに対する支援方法を考察する．まず適用箇所の発見については，M.ファウラーの各リファクタリングにおいて定められている動機から明確な抽出ルールを定めることで，自動抽出が可能になり，容易に適用箇所の発見を行うことができる．

手順に従ってリファクタリングを行ううえで必要となる情報の抽出に対しては，手順において必要とされる情報を明確に定めることで，自動抽出が行える．

目的に対しどのようなリファクタリングを使用すべきであるかは，各リファクタリングに対し目的に有効なものを調査しユーザーに行うべきリファクタリングを示すことで支援できる．本研究では改善者自身の解釈を示す形に変更するためのリファクタリングを定める．

3．2　本研究での支援内容

以上を踏まえ本研究では M.ファウラーの各リファクタリングに対し，以下のような３つのデータを自動提示することで問題点を解決する．

- **ユーザーの目的に対するリファクタリングの指針**
- **リファクタリング適用箇所**
  （リファクタリングを行うべき箇所の情報）
- **手順用データ**
  （手順に従う上で必要となるプログラム内のデータ）

4　リファクタリングを用いたプログラム改善例

ここでは実際に稼動しているシステムに対して，リファクタリングの適用を行った例から，改善者自身の解釈を示し自身が読みやすい形に変更するための支援方法を考察する．

4．1　対象と問題点

本学では，グループワークによるシステム開発を行う「情報実験Ⅱ」という授業を行っている．本研究室では受講生が効率的に作業を行えるよう，「グループワーク支援システム」の開発，運営および保守を行っている．このシステムの機能一つに掲示板機能がある．この掲示板では一週間以内に投稿された新着書き込みを支援システムのトップページにおいて提示していたが，その表示速度が非常に遅いという欠点があった．そこで掲示板のプログラムを

A Method for Refactoring Java Programming

†Mikito Tanaka　‡Saeko Matsuura
†‡Shibaura institute of technology
Department of electronic information system

解析し改善を行おうとしたが，プログラム内の１つのメソッド（jsp）の記述が非常に長い，if 文が非常に複雑である等の理由からプログラムが読み辛い状態にあった．そこでリファクタリングによって改善者自身の解釈を示す形に変換し読みやすくしたうえで，問題箇所を改善していくことにした．今回はこの読みやすい形への変換に着目した．

４．２　理解しやすい形への改善の流れ

改善者自身の解釈を示す形への変換方法として以下を定義した．

- **変数の責務の明確化**
- **メソッドの責務の明確化**
- **条件分岐の条件対象の明確化**

各手段の具体的なリファクタリング内容と適用箇所，および手順用データを表 4.2 に示す．

表 4.2.1　変数の責務の明確化を行うリファクタリング一覧

・一時変数の分離：多数の責務を持つ一時変数を分離し責務を明確化する．　適用箇所：条件分岐の実行部で無いルートにおいて，複数回の代入が行われている変数．　手順用データ：抽出した変数への参照箇所．
・パラメータへの代入の除去：本来代入されるべきでないパラメータとしての責務を明確化する．　適用箇所：パラメータへの代入を行っている変数．　手順用データ：抽出した変数の使用箇所．
・引数の削除：使われていないパラメータの削除により責務の無い変数を減らす．適用箇所：メソッド内部で使用されていないパラメータ．　手順用データ：抽出したメソッドの使用箇所．
・クラスによるタイプコードの置き換え ・サブクラスによるタイプコードの引き上げ ・State/Strategyによるタイプコードの置き換え それ自体が直接の意味を持つものでないタイプコードを置き換える．　適用箇所：タイプコードである変数．…① 手順用データ：抽出変数への参照箇所，代替値への参照法．
・制御フラグの削除：自身が責務を持たない制御フラグを削除する．　適用箇所：制御フラグである変数…② 手順用データ：制御フラグのタイプごとの具体的削除法．
・問い合わせによる一時変数の置き換え （一時変数のインライン化，説明用変数の導入も考慮） 一時的に値を委譲されるだけの一時変数を問い合わせで置き換え．　適用箇所：一度だけ返り値を代入する変数． 手順用データ：抽出変数への参照，代替の問い合わせ文．

表 4.2.2　変数の責務の明確化を行うリファクタリング一覧

・重複した条件分岐の断片の統合：どの文がどのような条件を持たなければならないのかを明確化する． 適用箇所：全分岐で同コードが抽出可能な条件実行部． 手順用データ：具体的変更法．
・ガード節による入れ子条件記述の置き換え：条件分岐においての正常ルートを明確化させる．　適用箇所：条件分岐にて一方のルートがもう一方に対し極端に短く，ガード節(break,return文を使う節)が使われない条件実行部．ただし極端に短いと判断する具体値はユーザーによる． 手順用データ：具体的変更法．
・setメソッドの削除：値が生成時以外に変更されない属性のsetterを削除し，修飾子finalを付けることで，その属性が定数として使われていることを明確化． 適用箇所：使用されないsetメソッド．　手順用データ：set される属性．属性への他の set メソッド．

表 4.2.3　メソッドの責務の明確化を行うリファクタリング一覧

・問い合わせと更新の分離：問い合わせと更新の役割ごとにメソッド分けすることでメソッドの責務が明確になる． 適用箇所：問い合わせと更新を共に行うメソッド． 手順用データ：このメソッドへの参照箇所．
・メソッドの隠蔽：アクセス修飾子を限定することでメソッドの使用範囲を明確化する．　適用箇所：アクセス修飾子をより隠蔽しても他の使用箇所に影響が無いメソッド． 手順用データ：変更可能なアクセス修飾子．
・明示的なメソッド群による引数の置き換え：引数に特定の値を送ることによって異なるコードが実行されるメソッドに対し，振る舞いごとに別々のメソッドに分割することによってメソッドの責務を明確化．適用箇所：メソッドの引数値に対して別々の振る舞いをするメソッド． 手順用データ：このメソッドへの参照箇所，具体的変更法．

４．３　改善結果

以上のリファクタリングを行った結果得られたシステムの改善結果を表 4-3 に示す．掲示板の新着記事数に対し表示するまでの時間を測定した．責務の無い箇所を明らかにしたことにより，不要な jsp へのアクセスを削除し，結果として件数に関わらず約 140 ミリ秒の改善が見られた．

表 4.3　改善結果

掲示板記事数［件］	全記事容量［byte］	改善前［ミリ秒］	改善後［ミリ秒］
200	108,008	965	813
100	53,837	446	271
50	26,781	264	104
30	17,189	209	50
1	646	143	0

４．４　改善箇所，判断材料の自動抽出プログラム作成

改善箇所，判断材料自動抽出について，表 4.2 の適用箇所に関する記述から自動抽出するプログラムを作成した．なお構文解析を行う際には java ファイルを xml 化する JJmlt というツールを利用した[2]．表 4.2 のうち詳細な解釈が必要なものについては以下のように定義を行った．

①におけるタイプコードの定義：条件文以外で参照されない．一部の値だけが代入され，かつその全ての値が条件文において個別に参照される．他の変数を参照する代入が行われない．以上を全て満たすもの．

②における制御フラグの定義：条件文以外で参照されない．あるループが上記の条件文の実行部によってのみ break 文の実行や return 文の実行,もしくはループの継続判定に使われる変数の書き換えが行われる．ループ外で他の変数を参照する代入が行われないかつその代入箇所は変数の分離が行えない．以上を全て満たすもの．

５　おわりに

以上のように，改善者自身の解釈を示す形にプログラムを変更するという目的に対して，行うべきリファクタリングを定義し，それらのリファクタリングの適用箇所自動抽出および適用の判断材料となる情報の抽出を行うことができた．今後は今回行った目的以外のものに効果を発揮するリファクタリングの定義することで，リファクタリングをより様々な改善に役立てることを目指す．

６　参考文献

[1]Martin Fowler：リファクタリング，ピアソンエデュケーション，2000
[2]JJmlt:http://www.hpc.cs.ehime-u.ac.jp/~aman/project/JJmlt/