コードクローンの特徴に基づくリファクタリング手法

大熊　祥平†　　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，ソフトウェア保守にかかるコストが増大していることが問題となっている．そのため，ソフトウェアに対してリファクタリングをすることが重要視されている．

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保ったまま，内部の構造を改善していくことである[1]．リファクタリングの必要を示す兆候は**「不吉な匂い」**と呼ばれ，その中で一番の原因として挙げられているものが**コードクローン(重複したコード)**である．

コードクローンをリファクタリングする手順は，構文の違い，類似部分の有無，クローンの位置関係によって異なるため適切な方法を考える手間がある．また，リファクタリングによる保守性の改善効果は，クローン部分の経路の数，リファクタリング時のクラス・メソッドの抽出数によって異なるため，どのコードクローンに対してリファクタリングを行えばよいかの判断が難しい．

本稿では，文献[1]で用いられている Java 言語を対象とした分析を行い，コードクローンの特徴ごとのリファクタリング手順と，リファクタリング効果を測定した結果から，最適なリファクタリング手法を提案する．

2. コードクローン

コードクローンとはソースコード中に存在する同一，または類似したコード断片のことである[2]．本稿ではコードクローンの検出に **CCFinderX**[3]を用いる．

あるソースコード中に存在する2つのコード断片 α，β が同一または類似しているとき，C(α, β) と書き，α は β とクローン関係をもつという．C は，反射律，推移律，対称律が成り立つ同値関係である．コードクローンの同値類を**クローンセット**という[2]．

3. クローンセットメトリクス

クローンセットメトリクスとは **GemX**[3]を CCFinderX のフロントエンドをして用いることで計測することができる，コードクローンの指標である．それぞれのクローンセットに対して表1の指標でメトリクスが計測される．

表1　クローンセットメトリクス

LEN	コードクローンのコード断片の長さ
POP	コードクローンになっているコード断片の数
NIF	コードクローンのコード断片を一つ以上含むソースファイル(クラス)の個数
RAD	コード断片を持っているソースファイル(クラス)の，ディレクトリ(パッケージ)上での広がり具合
RNR	コード断片のうち，繰り返し部分に含まれないトークンの割合
TKS	コード断片のトークンの種類
LOOP	コード断片に含まれるループの数
COND	条件分岐の数
McCabe	LOOPとCONDの合計

A Refactoring Method based on Code Clone Analysis.
† Shohei Ookuma　　† Saeko Matsuura
† Shibaura Institute of Technology Department of Electronic Information Systems

4. コードクローンの特徴抽出方法

学生が開発した**会議室予約システム**のコードクローンを分析し，コードクローンの特徴を「ソフトウェアを構成する要素」という観点から表2のものに分類した．ここで，宣言・代入群，ブロック処理の{，}で囲まれた処理の内，3行未満のものは「ソフトウェアの構成要素という単位としては短い」と判断し分類から除外した．

包含クローン(他のクローンセットの一部となっているもの)を除いた会議室予約システムのコードクローン 113 個に対して，特徴ごとのクローンセットメトリクスを集計し，範囲を設定した．その結果を表3に示す．ここで，複数のメソッド，switch 文，ブロックの連続の特徴で，POP ≠NIF のものは同一クラス内で共通部分を持つクローンが多く，これらのクローンはリファクタリングができないと判断し，範囲の設定から除いた．さらに，範囲を特徴ごとに明確にするため，複数の特徴を持つコードクローンは範囲の設定から除いた．また，設定した範囲を用いてすべてのコードクローン中の特徴の適合率を計測した結果を表4に示す．適合率とは範囲で絞り込んだ一覧中の特徴の割合であり，この値が高いほど目的の特徴を見つけ易い．

設定した範囲の中で，複数のメソッドは LEN の大きいもの，宣言・代入群とブロック処理は LEN の小さいもの，ブロックの連続は RNR の小さいものに分布が多く見られた．よって，設定したそれぞれの値の範囲で絞り込み，分布の多い値に着目することで，その特徴を持つコードクローンの抽出が可能となる．

表2　コードクローンの特徴

複数のメソッド	メソッドを複数含む
switch文	switch文を含む
宣言・代入群	変数の宣言，代入処理の連続(3行以上)を含む
ブロック処理	{，}で囲まれた処理(3行以上)を含む
ブロックの連続	{，}で囲まれた処理の連続を含む

表3 クローンセットメトリクスの範囲

	LEN	POP	NIF	RAD
複数のメソッド	100≤LEN	POP≤3	2≤NIF≤3	RAD=1
switch文	LEN<250	POP≤10	2≤NIF≤10	1≤RAD
宣言・代入群	LEN<400	POP≤6	NIF≤4	
ブロック処理	LEN<250	POP≤6	NIF≤3	RAD≤1
ブロックの連続	LEN<250	POP≤4	2≤NIF≤4	1≤RAD
	RNR	TKS	LOOP	COND
複数のメソッド	0.2≤RNR	20≤TKS		1≤COND
switch文	0.2≤RNR<0.4	10≤TKS<20	LOOP=0	1≤COND≤2
宣言・代入群	0.2≤RNR	10≤TKS≤30	LOOP≤1	COND≤2
ブロック処理	0.7≤RNR	10≤TKS≤30	LOOP≤2	COND≤4
ブロックの連続	0.1≤RNR	10≤TKS≤30	LOOP=0	1≤COND≤8

表4　特徴の適合率

	特徴の数	全体の該当数	適合率
複数のメソッド	12	27	0.444
switch文	3	3	1.000
宣言・代入群	26	68	0.382
ブロック処理	11	39	0.282
ブロックの連続	4	15	0.267

5. リファクタリング手順

実際に会議室予約システムのコードクローンに対してリファクタリングを行い，表5の特徴ごとのリファクタリ

ング手順を定義した．ここで複数のメソッド以外の特徴で，メソッドごとコードクローン内に含まれている場合，メソッド単位のリファクタリングを行う．

リファクタリングを行わないケースとして，switch 文の特徴で switch 文全体を含んでいないもの，宣言・代入群で配列・オブジェクトへの代入処理が途切れているもの，ブロックを抽出すると break 文がループ文の外に出るもの，抽出するブロック内に return 文含むもの，が考えられる．ただし break・return 文までの処理が 3 行以上のものは，その部分を抽出単位としてリファクタリングを行う．また，別パッケージ(関連のないクラス間)のものは抽出場所が定められないためリファクタリングを行わない．

各特徴のメソッド抽出時，親クラスへメソッドを引き上げる際に，抽出部分で使われる変数があれば引数とする．類似部分に用いられる変数の型が違う場合，前後の処理を含めて「Template Method の形成」を行う．また，代入処理を含む場合に，代入先の変数が抽出部分内のみ用いられる変数ではなくインスタンス変数でもない場合，インスタンス変数に置き換える．

注意点としてメソッド単位のリファクタリングを行う際に，共通の親クラスを持つべきかを考慮する必要がある．子クラス同士に共通性が見られない場合，クローンとなっているメソッドの存在すべき場所が他にある可能性が考えられる．その場合，他のクラスへ処理を委譲するか，新たにクラスを作成し処理を委譲する方法が考えられる．

表 5　特徴ごとのリファクタリング手順

複数のメソッド(メソッド単位のリファクタリング)
①.共通の親クラスを作成する ②.コードクローンを持つクラスは作成した親クラスを継承する ③.コードクローンのメソッド内で使用されているインスタンス変数を引き上げる(元のクラスで他に使われている場合，親クラスの代入部分を抽象setterメソッドで置き換え，子クラスでインスタンス変数への代入処理を記述する) ④.メソッドの引き上げを行う(可視性がprivateとなっている場合，protectedに変更する)
switch文
①.switch文の処理の固まりをメソッドとして抽出する ②.抽出部分をメソッドの呼び出しで置き換える ③.複数のメソッドの手順①へ
宣言・代入群
①.コードクローン内の変数宣言部分をローカル変数からインスタンス変数にする(抽出部分内のみ用いられるものは除く) ②.宣言・代入群をメソッドとして抽出する ③.抽出部分をメソッドの呼び出しで置き換える ④.複数のメソッドの手順①へ(同一クラス間のクローンは終了)
ブロック処理
①.ブロックで囲まれた範囲をメソッドとして抽出する(条件式も含まれている場合，抽出部分に含める) ②.抽出部分をメソッドの呼び出しで置き換える ③.複数のメソッドの手順①へ(同一クラス間のクローンは終了)
ブロックの連続
①.クローンとなっているブロックの連続をメソッドとして抽出する(ブロック間の制御文，条件式も含む) ②.抽出部分をメソッドの呼び出しで置き換える ③.複数のメソッドの手順①へ

```
public String ResSt(int St){
  switch (St){
    case 0:  return ("仮予約");
    case 1:  return ("本予約");
    case 2:  return ("既定予約");
    ...
  }
}
```

図 1　共通性が見られないクラス間のコードクローン

例として図 1 を示す．このコードは予約の種類を表す番号を整数型で受け取り，対応する予約名を文字列型で返すというメソッドで，予約のキャンセル，予約の状況確認といった共通性の見られないクラス間で見られた．他に予約の内容を管理するクラスが存在し，このクローンに関してはそのクラスへ移動することが適当であると判断した．

6.　リファクタリング効果

特徴ごとにリファクタリングした効果を測定した結果を表 6 に示す．効果の測定には **WMC(クラス単位の循環的複雑度の総和)**を用いる．循環的複雑度とは，制御フローにおいて全経路を網羅するのに最低限必要な経路の数である．WMC が大きいものは，コードを読む上で多くの経路を追わなければならない，テスト時の実行数が多いといったことを示す．よって WMC の減少数が多いものは，可読性，ソフトウェア保守の観点でリファクタリングの効果が高いといえる．また，リファクタリング前は WMC の合計が 1556，実行ステップ数が 8415 である．

表 6 より switch 文を含むコードクローンをリファクタリングすると効果が高いことがわかる．これは，switch 文は多数への分岐処理をしていること，POP が大きいものが多いといったことが要因として考えられる．

複数のメソッドを含むコードクローンは LEN が大きいため，分岐処理を多く含む場合があり効果が高いといえる．

ブロックの連続はコードの長さは短いものの，複数のブロックをまとめて一つのメソッドとして抽出するため，効果があるといえる．

ブロック処理は条件式ごと抽出するもの，ブロック内にブロックを含むものがあり，WMC の減少が見込める．

宣言・代入群を含むコードクローンは，分岐を含まないため WMC は減少しない．反対にメソッドの抽出によって WMC の合計は増えてしまうため効果が低いといえる．

表 6　リファクタリング効果

	WMCの合計の増減(1セット辺りの平均)	実行ステップの増減(1セット辺りの平均)	優先順位
複数のメソッド	-31.3	-123.8	2
switch文	-55.0	-137.0	1
宣言・代入群	+6.5	+20.6	5
ブロック処理	-2.0	-5.0	4
ブロックの連続	-12.7	-40.7	3

7.　まとめ

本稿では，コードクローンを特徴で分類し，クローンセットメトリクスを用いた特徴の抽出方法と，特徴ごとのリファクタリング手順・効果を分析した．

今後の課題として，特徴の抽出方法で適合率の低い特徴に対しては再現率との比較を行い，より適切なクローンセットメトリクスの範囲を定義する必要がある．さらに，他のソフトウェアに対して本手法を適用し，同様の効果が得られるかを確認する．また，手法が適用できない事例がある場合，原因について分析する必要がある．

8.　参考文献

[1]Martin Fowler. Refactoring: improving the design of existing code, Addison Wesley, 1999.

[2]肥後 芳樹，楠本 真二，井上 克郎．コードクローンを対象としたリファクタリングの有効性に関する調査，電子情報通信学会技術研究報告，SS2006-30, Vol.106, No.201, pp.37-42, 2006.

[3]CCFinderX, http://www.ccfinder.net/