同一のクラスから派生したクラス群内に存在するコードクローンの分析

2012SE096 加藤拓馬 2012SE228 下村碧 2011SE205 大橋篤貴
指導教員：横森励士

1 はじめに

ソフトウェアの大規模化に伴い，複雑な情報を整理して効果的な情報を与えることで，ソフトウェアの保守や理解を支援する方法が求められている．ソフトウェアの類似した機能を実現するために，Java では派生やインターフェースなどの仕組みが提供されている．本研究では実際のソフトウェアに対して，同一のクラスから派生したクラス群や，同一のインターフェースを実装したクラス群を抽出し，その中にどのようにコードクローンが出現するかを調査する．ソフトウェアの保守活動において特定のクラスから派生して新たなクラスが作成された場合に，適切なサポートを行うことで，保守性の低下を防ぐことができると考え，サポート方法に基づいてリサーチクエスチョンを設定し，分析を行う．

2 背景技術

2.1 コードクローンとリファクタリングによる解消

コードクローン [1] とは，ソースコード中にある同一または類似したコード断片の事を指し，類似した処理をコピーペーストで実現した場合に生じやすい．検出手法によって，得られるコードクローンには違いがあり，それは次のように分類される [2].

1 空白やタブ，括弧の位置などのコーディングスタイルを除いて，完全に同一なコードクローン．(タイプ 1)

2 変数名や関数名などのユーザ定義名，変数の型などの一部の予約語のみ異なるコードクローン．(タイプ 2)

3 タイプ 2 における違いに加えて，文の挿入や削除，変更が行われたコードクローン．(タイプ 3)

コードクローンを修正する時には，類似したコード断片それぞれにおいて修正が必要な場合も多く，保守においてはコードクローンを減らす対策をとる必要がある場面も多い．肥後らの研究 [2] では，分類したコードクローンの種類毎に適用可能なリファクタリング手法を表 1 のように紹介した．リファクタリング [3] とは，ソフトウェアの振舞いが提供する機能を維持しながら，ソフトウェアの内部構造を改良することをさす．ソフトウェアの拡張性や読みやすさ，理解しやすさなどを改善し，後の作業活動において活動しやすい状況を確保するために行われる．

表 1 コードクローンに適用可能なリファクタリング

リファクタリングパターン	対象コードクローン		
	タイプ1	タイプ2	タイプ3
メソッドの抽出	○	○	△
(親)クラスの抽出	○	○	−
メソッドの引き上げ	○	○	−
メソッドの移動	○	○	−
メソッドのパラメータ化	−	○	−
テンプレートメソッドの形成	−	−	○

2.2 コードクローンを持つクラス群から共通して利用されるクラスの利用方法の調査

似たような機能を持つクラス群には似たような記述が付加されることが多く，そのためコードクローンが多く発生すると考えられる．安藤らはコードクローン関係をもつクラス群が共通して利用するクラスに着目し，そのクラスがクラス群中でどのように利用されているかを調査した [4]. 43 種類のオープンソースソフトウェアからコードクローンを持つクラス群を抽出し，それらのクラス群が共通して利用しているクラスとして 620 のクラスを抽出した．その内の 339 のクラスがクラス群において同一の方法で利用されており，その中でも 144 のクラスはそのクラスを派生したクラス群において同一の処理が記述されていた．

[4] で示された利用方法それぞれに対してコードクローンの発生の仕方を調査することで，後述するサポートの方法によってソフトウェア内に存在するコードクローンが除去された状態，もしくは扱いやすい状態を維持することができるのではないかと考えた．本研究では [4] の分類結果の中で多く見ることができた，同一のクラスから派生したクラス群中に存在するコードクローンに着目して調査する．

2.3 想定しているサポートの方法

一般的にコードクローンは除去されるべきものであると認識されているが，現実には解消が難しいコードクローンも多く存在しており，コードクローンが残っている状態で開発が進行している場合も多く，ソフトウェアの内部に残っているコードクローンをどう扱うべきかは，ソースコードの保守性を維持するうえで重要な事項の一つである．そのようなソフトウェアにおいて，既にコードクローンが生成されているクラス群に実装された機能と似た機能を有するクラスを新たに追加する場合を考える．そのときには，次のような対処方法が考えられる．

1. **CCFinder** などの分析ツールを用いてコードクローンとなるコード片を抽出し，リファクタリングを適用してコードクローンを解消する．
2. 既に存在するクラス群中のコードクローンを利用例として提示し，既存のクラスの記述を参考に派生クラスを同じように定義するように誘導する．

3. 何も手を加えない.

1 が実行できれば，同一の処理を一か所にまとめることで修正がしやすく保守性が高い状態を維持できるが，必ずしもできるわけではない．実際には 3 の方法で対処されることが多い．同じような処理を複数の実現方法で記入した事例が発生しやすくなり，問題が生じた時にコード断片ごとに解決方法が異なるコードクローンが大量発生し，保守性が大幅に低下する．2 の対処方法を用いた場合，コードの一貫性を維持するための手間は増大するが，同じような実現方法で記述されるので，解決方法が似たものとなると想定され，保守性の低下は限定的になると考えられる．ただし，2 の対処方法を実現するためには，クラスを新しく追加するときの記述について，ソフトウェアに既に存在するコードクローンと共通性があることと，そのような共通性を持つコードクローンがソフトウェア内に多く発生していることが把握できていることが条件となる．

1 の対処方法の適用を優先したうえで，解決が難しいコードクローンに対しては 2 の対処方法を推進する方向で進めるという形で支援を行うことが現実的であると考えられる．1，2 の対処方法は方向性が全く異なるので，どのような場合にどちらの対処方法をとるかの判断をする必要がある．[4] からコードクローン発生を引き起こしやすいクラスの利用の種類ごとに知見を得て傾向を確認し，どのクラス間にコードクローン関係や利用関係があるかを把握できている状態を前提として，それぞれのコードクローンの細かい分析をしなくても，1 か 2 かの対処方法の判断が可能であるようなツールを作成する方針を提案することが最終的な研究の目標である．

3 同一のクラスから派生したクラス群内に存在するコードクローンの分析

3.1 研究の動機

本研究では，[4] の分類結果の中で多く見ることができた、同一のクラスから派生したクラス群内に存在するコードクローンに着目し分析を行う．派生したクラス群中では，それらのクラスにおいて求めれられる処理が類似することから，派生先の各クラスにおいて実装すべきメソッドなどにおいて同一の記述が生じやすく，コードクローンが発生しやすいと考えられる．同時に，新しいクラスを追加するときに，そのクラスから派生させようとしたという事実が共通性を示すことになると考えられるので，早期に共通点を把握し，指摘可能であるという点も利点として上げられる．

実際のソフトウェアにおいて各クラスの派生先と派生元の情報を入手し，派生先のクラス群に存在するコードクローンがどのような形で発生しているか調査する．以下のようなリサーチクエスチョンを設定し，分析を行う．

RQ1 検出されたコードクローンのタイプの違いで，想定している対処方法が変わってくるのか？

RQ2 分化の仕組みとして，派生とインターフェースの実装があるが，それらの傾向の違いはあるか？

RQ3 派生によって生じたタイプ 3 のコードクローンに，解消や利用例を示すときに利用できる特性はあるか？

次に，ツールの適用を考え，どのクラス間にコードクローン関係や利用関係があるかの情報のみが存在するという前提で，以下のような設問を設定する．

RQ4 プロジェクト毎の同一のクラスから派生しているクラス群中にコードクローンが存在する割合から，対処方法を決定できるか？

RQ5 コードクローンが発生しているクラス群の大きさで，対処方法を決定できるか？そのクラス群内にどれほどコードクローンが及んでいるかで，対処方法を決定できるか？

3.2 分析の手順

1. 20 のオープンソースプロジェクトそれぞれについて，ソースコードを 1 バージョン入手する．
2. プロジェクト内のソースコードを分析し，同一のクラスから派生しているクラス群や，同一のインターフェースを実装しているクラス群をすべて求める．
3. 対象となるソフトウェアのソースコードに対して，CCFinder[1] を用いてコードクローンを抽出する．
4. 3 の結果を用いて，同一のクラスから派生またはインターフェースを実装しているクラス群の中で，同一の記述がされている部分に派生に関係したコードクローンが存在しているかを調査する．ただし，テスト用であると思われるクラスについては分析の対象から除外した．

3.3 分析項目

リサーチクエスチョンから，次の分析項目を設定する．

1. **[a] 検出されたコードクローンのタイプ**
   検出されたコードクローンの中身を調査し，それらを [2] で紹介されている基準に従って分類した．
2. **[b] 同一のクラスから派生した（同一のインターフェースを実装した）クラス群内で，コードクローンを持つクラス群の数と割合**
   分析したクラス群の中で，同一の処理が記述されることでコードクローンが生じているクラス群を求め，それらの数と割合を求め，プロジェクト毎に評価する．以降，同一のクラスから派生したクラス群を派生クラス群，同一のインターフェースを実装したクラス群を実装クラス群と定義する．
3. **[c] クラス群の大きさとコードクローンの種類別割合**
   分析したクラス群全体と，コードクローンを持つクラス群それぞれについてクラス群の大きさで分布し，クラス群の大きさによる違いがあるのかを調査する．
4. **[d] 同一のクラスを利用しているクラス群でのコード**

クローンが現れているクラスの割合
分析したクラス群の中で，そのクラス群内で最も広範囲に及んでいるコードクローンがどれほどの数のクラスに及んでいるかを調査し，その割合を求める．以降この割合を浸食率と定義する．

4　分析の結果

20 のソフトウェアプロジェクトのソースコードに対して分析を行った．その分析結果を項目ごとに紹介する．

4.1　項目 a：検出されたコードクローンのタイプ

抽出した派生クラス群は 321，その内 35 のクラス群がタイプ 1，2 を含むクラス群，52 のクラス群がタイプ 3 を含むクラス群であった．また，抽出した実装クラス群は 166，その内 36 のクラス群がタイプ 1，2 を含むクラス群，33 のクラス群がタイプ 3 を含むクラス群であった．これらのクラス群におけるコードクローンについてリファクタリングパターンの適用しやすさを調査したところ，タイプ 1，2 では比較的簡単なリファクタリングパターンを適用することができるものが多かった．しかし，タイプ 3 に対して，表 1 のリファクタリング手法が可能かを調査したが，適用が難しいものが大半であった．このことからタイプ 1，2 では，1 の対処方法を適用すべきであると考える．また，タイプ 3 では，2 の対処方法を適用すべきであると考える．

4.2　項目 b：コードクローンを持つクラス群の数と割合

図 1 で派生クラス群の中でコードクローンが存在するクラス群の割合を，図 2 で実装クラス群の中でコードクローンが存在するクラス群の割合をそれぞれ示す．派生クラス群では，タイプ 1，2 は全てのクラス群に対して 1 割程，タイプ 3 は 1.5 割程検出された．実装クラス群では，タイプ 1，2 は全てのコードクローンに対して 2 割程，タイプ 3 も同様の割合で検出された．プロジェクト単位でみると，派生クラス群の中には，タイプ 1，2 と 3 のコードクローンがプロジェクト毎に混在しているが，実装クラス群の中では，タイプ 1，2 が含まれている割合が高いことがわかる．全体として見ると，派生元によって分類されたクラス群にてコードクローンを含む割合が高いプロジェクトは，タイプ 1，2 を含む可能性が高いことがわかる．

4.3　項目 c：クラス群の大きさとコードクローンの種類

クラス群の大きさがコードクローンの種類に与える影響について調査した．派生クラス群と，実装クラス群について，その中に存在するコードクローンをタイプ別に分類した．分類結果を表 2 に示す．派生クラス群と実装クラス群の両方において，小さめのクラス群に修正しやすいタイプ 1，2 が多く存在している．よって，クラス群の規模が小さいうちの 1 の対処方法が有効であると考える．

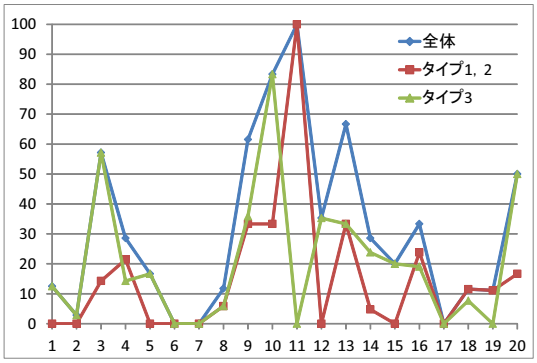

図 1　コードクローンを持つ派生クラス群の割合

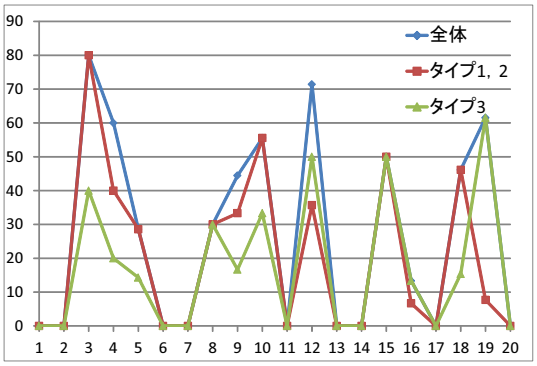

図 2　コードクローンを持つ実装クラス群の割合

表 2　クラス群の大きさとコードクローンの種類別の割合

派生クラス群				実装クラス群			
子クラスの数	タイプ1,2	タイプ3	クラス群の数	実装クラスの数	タイプ1,2	タイプ3	クラス群の数
2	12	8	149	2	8	6	72
3	2	10	54	3	10	9	39
4	4	7	33	4	5	5	23
5	6	9	24	5	3	2	8
6~10	8	12	43	6~10	6	3	10
11~15	1	3	9	11~15	1	3	3
16~20	4	2	4	16~20	2	2	6
21~		1	5	21~	5	3	5
計	37	52	321	計	40	33	166

4.4　項目 d：コードクローンの浸食率

派生クラス群を調査し，クラス群の大きさと侵食率で分類を行った結果を表 3 に示す．この表から，タイプ 1，2 の場合，タイプ 3 の場合共に，クラス群の大きさが増加するにつれて侵食率は減少する傾向があることが分かった．同様に，実装クラス群の大きさと侵食率で分類を行った．表 4 はその結果で，表 3 の結果と同様の傾向があることが分かった．また表 2 において 11 以上のクラスから成るクラス群にタイプ 1，2 のコードクローンが発生していたが表 4 より侵食率は非常に低い結果であった．

5　考察

研究の動機において記述したリサーチクエスチョンに対して分析の結果を用いて次のように回答する．

RQ1 検出されたコードクローンのタイプの違いで，想定している対処方法が変わってくるのか？
対処方法は変わる．タイプ 1，2 は解消のためにリファクタリングを推奨するべきであるため，1 の対処方法の適用を優先するが，タイプ 3 は解消が難しい例が大

表 3 派生クラス群の侵食率

		クラス群の大きさ											
		タイプ1,タイプ2						タイプ3					
		2~5	6~10	11~15	16~20	21~25	26~	2~5	6~10	11~15	16~20	21~25	26~
侵食率	0~10%												1
	11~20%			1									
	21~30%		4										
	31~40%	3	2					5					
	41~50%	2	1					1			1		
	51~60%								1				
	61~70%	3			1			4	4	2			
	71~80%			1				5		1			
	81~90%		1						1				
	91~100%	16						11	2		1		

表 4 実装クラス群の侵食率

		クラス群の大きさ											
		タイプ1,タイプ2						タイプ3					
		2~5	6~10	11~15	16~20	21~25	26~	2~5	6~10	11~15	16~20	21~25	26~
侵食率	0~10%					2	1						
	11~20%		1	1			1				1		
	21~30%		2										
	31~40%	2											
	41~50%	3	1		1	1		2		1			
	51~60%												
	61~70%	5						2					
	71~80%	2						4	1				
	81~90%								1				
	91~100%	13						3	1				

半であったため，2 の対処方法の適用を優先する．

RQ2 **分化の仕組みとして，派生とインターフェースの実装があるが，それらの傾向の違いはあるか？**

本研究の分析で派生とインターフェースの実装の間に多少傾向の差がみられた．コードクローンを持つクラス群の割合において，派生クラス群の場合では，全体とタイプ 1，2 とタイプ 3 の割合がそれぞれ類似した変化をしていないが，実装クラス群の場合では，3 つ全ての割合が類似して変化している傾向が少しみられた．派生クラス群の場合と実装クラス群の場合で傾向に差はなく，同じ方向性の助言をすべきだと考える．

RQ3 **派生によって生じたタイプ 3 のコードクローンに，解消や利用例を示すときに利用できる特性はあるか？**

派生においては，コードクローンの解消に利用できる特性は確認できなかった．しかし，同一の名前を持つメソッドが多く存在し，その中にタイプ 3 のコードクローンとして検出されにくい部分にも類似したコード断片が見つけられた．タイプ 3 のコードクローンであってもソフトウェア中に共通性を持つコードクローンが多く存在していることを示すことができており，これを用いて利用例を示すことができる．

RQ4 **プロジェクト毎の同一のクラスから派生しているクラス群中にコードクローンが存在する割合から，対処方法を決定できるか？**

示すことは可能であると考える．コードクローンを持つクラス群の割合が高いプロジェクトには，タイプ 1，2 が存在する可能性が高いため，1 の対処方法を適用すべきと考える．割合が低い場合では，既にコードクローンへの対策が講じられている場合が多く，タイプ 3 のみ残っている可能性が高いため 2 の対処方法を適用すべきだと考える．図 1 では，全体とタイプ 1，2 とタイプ 3 の割合のグラフがそれぞれあまり類似した形になってないことが分かるので，プロジェクト数を増やして分析の量も増やすことも必要だと考えるが，図 1，図 2 どちらの場合も全体の割合が 30% を目安に 1 か 2 の対処方法を決めるべきだと考える．

RQ5 **コードクローンが発生しているクラス群の大きさで，対処方法を決定できるか？そのクラス群内にどれほどコードクローンが及んでいるかで，対処方法を決定できるか？**

コードクローンを含むクラス群の数が小さいほど，タイプ 1，2 の存在を多く確認できたため，1 の対処方法の適用を予想する．一方，10 を超えるような大型のクラス群ではタイプ 1，2 をほとんど確認できなかったため，2 の対処方法を適用すべきだと考える．

クラス群の大きさが増加するにつれて侵食率は減少する傾向がある．クラス群の規模が大きいときタイプ 1，2 がほとんど確認できなかったことから，侵食率が低いクラス群には 2 の対処方法を適用をすべきだと考える．またクラス群の規模が小さいときタイプ 1，2 の存在を多く確認したため，侵食率が高いクラス群には 1 の対処方法を適用すべきであると考える．

6 まとめ

本研究では，実際のソフトウェアに対して，同一のクラスから派生したクラス間のコードクローンがどのように出現するかを分析し，コードクローンが出現するクラス群における特徴を調査した．ソフトウェアの保守活動において，同一のクラスから派生して新たなクラスが作成された場合に，有効なサポートの方法を考察した．本研究で考察したサポート方法の判断基準に基づいて，ツールを作成し，助言を行うことでプロジェクト内でコードクローンの存在を認識させながら，ソフトウェアに対する機能追加を行うことで，保守性の低下を最低限にすることができる．

参考文献

[1] T.Kamiya, S.Kusumoto, and K.Inoue : “CCFinder:A multi-linguistic token-based code clone detection system for large scale source code，”IEEE Transactions on Software Engineering，vol.28，no.7，pp.654-670，2002.

[2] 肥後芳樹，吉田則裕 : “コードクローンを対象としたリファクタリング，”コンピュータソフトウェア，vol.28，no.4，pp.43-56，2011.

[3] マーチン・ファウラー : “リファクタリング，” ピアソン・エデュケーション，2006.

[4] 安藤正憲，堀江大河，井田裕之 : “コードクローンを用いたコンポーネントランク拡張手法の有効性評価—変化の影響を受けた部品についての分析—，” 南山大学情報理工学部 2013 年度卒業論文，2014.