共通のクラスを継承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，ソフトウェア内部を理解しやすくするために，複雑な情報を整理して，効果的に情報を与え，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 では継承やインターフェースなどの仕組みが提供されているが，その中で共通の部分は記述のコピーペーストにより実現されることも考えられ，この部分に着目して支援を行うことが有効であると考えられる．本研究では，実際のソフトウェアに対して，共通したクラスを継承する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する．分析では，共通のクラスを継承したクラス群や共通のインターフェースを実装したクラス群を抽出し，その中に実際にどれくらいのコードクローンが存在するかを調査する．このような分析を通じて，特定のクラスを継承して新たなクラスが作成された場合に，どのような警告を出すことが有効であるかを評価する．

2　背景技術

2.1　コードクローン

コードクローン [1] とは，ソースコード中にある類似または一致したコード断片の事を指し，同一または良く似た処理をコピーペーストで実現した場合に生じることが多い．似たような機能を実現するために共通した必要な前処理や後処理を行う際にも生じやすく，開発者の何らかの意図によって作りこまれることが多い．コードクローンを修正する時には，類似する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．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 [3].

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，完全一致するコードクローン．(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)

2.2　コードクローンの種類とリファクタリングパターン

リファクタリング [2] 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していくことである．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．肥後らの研究 [3] では，前節で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．タイプ 3 のコードクローンへの対処方法として，テンプレートメソッドの形成などが上げられているが，今回の分析では，適用が難しい事例が大半であった．

表 1　リファクタリングパターンが対象とするコードクローン

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	－	－	○

2.3　共通のクラスを継承しているクラス群におけるコードクローンに関する分析

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しているクラスに着目し，そのクラスがクラス群中の各クラスにおいてどのように利用されているかを調査した [4]．結果では，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共通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継承したのちに共通の処理が記述され，同じような振る舞いを行うクラス群においてコードクローンを検出することができ，共通のクラスを継承したクラス群にコードクローンが一定以上存在することを示している．

3　共通したクラスを継承するクラス間に存在するコードクローンの分析

3.1　研究の動機

前述の分析では，共通のクラスを継承したクラス群にコードクローンが一定以上存在すること，似たような機能をもつクラス群が形成されていることを示している．実際のソフトウェアにおいてより詳細に分析を行い，プロジェクトごとの特徴，もしくはコードクローンを多く含むクラス群中における特徴を見つけ出すことで，コードクローンを適切に管理するための助言ができると考える．

3.2 目的

共通したクラスを継承したクラス群においてコードクローンがどのように出現するかをプロジェクト毎に分析を行い，コードクローンが出現しやすいプロジェクトの特徴を分析することで，どのような管理が可能かを推測する．

3.3 分析の手順

次の手順で分析を行う．

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，同一のクラスを継承しているクラス群や，同一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，それぞれのクラス群の中で，共通して記述されている部分に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，似たような機能を実現するためのテストで似たようなコードが大量に記述されたとしても保守性に大きな問題は出ないと考えて，除外した．

3.4 項目

上記の手順で入手した，クラス群とクラス群内のコードクローンの情報をもとに，以下の評価基準で評価を行う．

1. **[a] 共通のクラスから継承を行っているクラス群の数**
   各プロジェクト毎に，継承元ごとにクラス群を作成する．継承元をもたないクラスは考慮しない．この項目は，それぞれのプロジェクト毎に，共通の親クラスを持つクラス群がどれだけ生成されているかを示す．
2. **[b] 共通のインターフェースを実装したクラス群の数**
   各クラスが実装しているインターフェース毎に分類し，クラス群をそれぞれ作成する．複数のインターフェースを実装しているクラスは複数のクラス群に属することがある．この項目は，それぞれのプロジェクト毎に共通のインターフェースを持つクラス群がどれだけ生成されているかを示す．
3. **[c] 共通の親クラスから継承を行っているクラス群または共通のインターフェースを実装しているクラス群内で，コードクローンを持つクラス群の数と割合**
   a，b で示したクラス群の中で，共通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る．どのような場合に継承や実装と関連してコードクローンが生じやすいのかなどを，プロジェクト毎に評価するために利用する．
4. **[d] クラス群の大きさとコードクローンの種類別割合**
   a，b のクラス群全体と，c の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りコードクローンの生じやすさにの違いがあるかを調査する．また，どの種類のコードクローンが多いかを調査し，リファクタリング可能なものがどれだけ多いかを調査する．
5. **[e] 共通の親クラスからの継承を行っているクラス群または共通のインターフェースを実装しているクラス群内で，コードクローンがそのクラス群内にどれほど現れているかの割合**
   c で示したクラス群の中で，そのクラス群内で生じて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，その割合を侵食率として求める．

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 共通クラスから継承を行っているクラス群の数と共通インターフェイスを実装したクラス群の数

項目 a，項目 b について，20 のプロジェクトに対して分析した結果を示す．表 2 は共通のクラスから継承を行っているクラス群の数と，インターフェースを実装しているクラス群の数を示したもので，プロジェクト毎に継承が何種類存在するか，複数回現れてクラス群の中に存在する継承が何種類存在するかを，継承元がプロジェクト内で宣言されたものであるか，標準クラスライブラリを継承しているかで分けて示している．以下，継承または実装したクラスがプロジェクト内で宣言されているものを内部継承および内部実装，標準ライブラリを継承または実装しているものを外部継承および外部実装と呼ぶ．この結果からは，681 種類の継承の内，50% 程が複数のクラスによって継承していることがわかる．一方，428 種類のインターフェースの内，40% 程が複数のクラスで実装されていた．

4.2 共通の親クラスからの継承または共通のインターフェースの実装を行っているクラス群内で，コードクローンをその中に持つクラス群の数と割合

項目 c について，表 2 で抽出したクラス群を調べたところ，共通のクラスを継承した 321 のクラスの内，75 のクラス群の中にコードクローンが存在し，その内 35 のクラス群がタイプ 1，2 のコードクローンを含むクラス群であった．図 4.2 にプロジェクト毎に，コードクローンを持つクラス群の割合を示している．タイプ 1，2 のコードクローンは全てのコードクローンに対して約半分検出された．共通のインターフェースを実装した 166 のクラス群の内，52 のクラス群の中にコードクローンが存在し，その内 36 のクラス群がタイプ 1，2 のコードクローンを持つクラス群であった．図 4.2 にプロジェクト毎に，コードクローンを持つクラス群の割合を示している．タイプ 1，2 のコードクローンは全てのコードクローンに対して約 7 割検出された．これらの分析から，コードクローンを持つクラス群の割合が高い (30% 以上) と低いプロジェクトに二分するこ

表 2　共通のクラスから継承を行っているクラス群の数とインターフェースを実装しているクラス群の数

	継承の数			クラス群の数		
プロジェクト名	総数	内部継承	外部継承	総数	内部継承	外部継承
barbecue-1.5-beta1	20	12	8	8	7	1
GeoAPI-3_0_0	55	47	8	35	31	4
HoDoKu	15	2	13	7	1	6
neuroph_2.1.0_beta	37	17	20	14	9	5
JGPSTrackEdit	16	7	9	12	6	6
mondrian-0.5	152	106	46	67	43	24
xbrlapi-5.0-sorce	2	2	0	2	2	0
peers-0.2	21	17	4	17	13	4
texlipse	59	52	7	39	32	7
jphonelite-src-1.0	19	3	16	6	1	5
joChess-1.0.1-sources	9	3	6	3	1	2
SweetHome3D-1.7-src	77	24	53	17	7	10
atan_0_03	8	3	5	3	0	3
lwjgl-source-1.0-rc1	45	33	12	21	17	4
KludgopolB_src	9	4	5	5	5	0
Jsoko_1.57-src	37	15	22	21	7	14
horizon-12-10-12	7	3	4	3	1	2
JEPlus	52	25	27	26	14	12
javaGeom-0.7.0-SRC	30	1	29	9	1	8
AppSever4RPG2013-12-17	11	4	7	6	5	1
合計	681	380	301	321	203	118

	実装の数			クラス群の数		
	総数	内部実装	外部実装	総数	内部実装	外部実装
barbecue-1.5-beta1	0	0	0	0	0	0
GeoAPI-3_0_0	4	1	3	0	0	0
HoDoKu	14	0	14	7	0	7
neuroph_2.1.0_beta	11	1	10	5	1	4
JGPSTrackEdit	24	7	10	6	2	4
mondrian-0.5	115	102	13	18	13	5
xbrlapi-5.0-sorce	2	0	2	0	0	0
peers-0.2	10	6	4	9	6	3
texlipse	54	49	5	18	15	3
jphonelite-src-1.0	24	11	13	9	6	3
joChess-1.0.1-sources	9	0	9	5	0	5
SweetHome3D-1.7-src	35	1	34	14	1	13
atan_0_03	5	0	5	1	0	1
lwjgl-source-1.0-rc1	37	23	14	27	5	22
KludgopolB_src	2	0	2	2	0	2
Jsoko_1.57-src	20	1	19	15	1	14
horizon-12-10-12	7	0	7	3	0	3
JEPlus	29	23	6	13	7	6
javaGeom-0.7.0-SRC	25	24	1	13	12	1
AppSever4RPG2013-12-17	1	1	0	1	1	0
合計	428	250	171	166	70	96

とがわかった．特に，タイプ 1，2 のコードクローンに限定した場合，0% のプロジェクトとそれ以外にわけることができる．割合が 0% のプロジェクトはコードクローンの対策が施されたプロジェクトではないかと考える．また，共通のインターフェースを実装したクラス群中のコードクローンは，共通のクラスを継承したクラス群よりもタイプ 1，2 のコードクローンを含むことが多い．

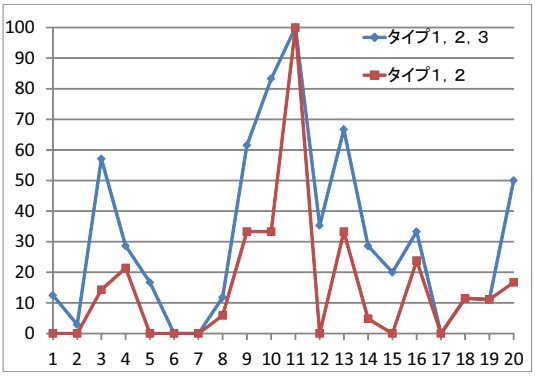

図 1　共通の親クラスを継承したクラス群におけるコードクローンを持つクラス群の割合

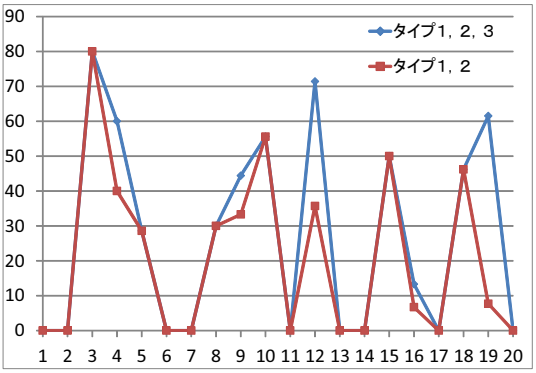

図 2　共通のインターフェースを実装したクラス群におけるコードクローンを持つクラス群の割合

4.3　クラス群の大きさとコードクローンの種類別の割合

20 のプロジェクトに対して，項目 d について，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．共通の親クラスを持つクラス群と，共通のインターフェースを実装したクラス群について，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 3 に示す．この表から，共通したクラスを継承するクラス群と共通のインターフェースを実装するクラス群両方とも，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，クラス群の規模が大きくなるほど修正しにくいタイプ 3 が多く存在することがわかり，クラス群の規模が小さいうちのリファクタリングが有効であると考える．

表 3　クラス群の大きさとコードクローンの種類別の割合

共通のクラスを継承したクラスのクラス群				
子クラスの数	タイプ1	タイプ2	タイプ3	クラス群の数
2	8	4	8	149
3	2		10	54
4	1	3	7	33
5	3	3	9	24
6	1	1	4	17
7	1	1	3	11
8	1	1	2	9
9	1	1	3	5
10				1
11~15		1	3	9
16~20	1	3	2	4
21~			1	5
計	19	18	52	321

共通のインターフェースを実装したクラス群				
子クラスの数	タイプ1	タイプ2	タイプ3	クラス群の数
2	4	4	6	72
3	5	5	9	39
4	2	3	5	23
5	2	1	2	8
6		1	1	4
7	1	2	1	3
8			1	1
9		1		1
10		1		1
11~15		1	3	3
16~20	1	1	2	6
21~	1	4	3	5
計	16	24	33	166

4.4　クラス群におけるコードクローンの浸食率について

項目 e について，20 のプロジェクトに存在する共通の親クラスを持つクラス群の中で，互いにコードクローンを持つ 75 のクラス群と，タイプ 1，2 のコードクローンを有する 35 のクラス群において，クラス群の大きさと侵食率で分類を行った．その結果を表 4 に示す．この表から，タイプ 1，2 の場合は，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．一方でタイプ 3 の場合は，クラス群の大きさが 20 までのクラス群に対しては侵食率はあまり変わらなかった．ここから，継承を持つクラス群の場合は，コードクローンのタイプによってクラス群内のコードクローンの発生のしやすさに差があると考えられる．同様に，共通のインターフェースを実装

するクラス群の中で，互いにコードクローンを持つ 52 のクラス群と，タイプ 1，2 のコードクローンを有する 36 のクラス群それぞれにおいて，クラス群の大きさと侵食率で分類を行った．その結果を表 5 に示す．この表から，コードクローンのタイプにかかわらず，クラス群中のクラス数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．ここから，インターフェースを実装するクラス群の場合は，コードクローンのタイプによってそのクラス群内のコードクローンの発生のしやすさにあまり差がないと考えられる．

表 4 共通のクラスを継承するクラス群の侵食率

		クラス群の大きさ											
		タイプ1,タイプ2,タイプ3						タイプ1,タイプ2					
		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						1			
	21~30%		4						4				
	31~40%	8	2					3	2				
	41~50%	3	1		1			2	1				
	51~60%		1										
	61~70%	7	4	2	1			3			1		
	71~80%	5		2						1			
	81~90%		2						1				
	91~100%	27	2		1			16					

表 5 インターフェースを実装するクラス群の侵食率

		クラス群の大きさ											
		タイプ1,タイプ2,タイプ3						タイプ1,タイプ2					
		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					2	1
	11~20%		1	1	1		1		1	1			1
	21~30%		2						2				
	31~40%	2						2					
	41~50%	5	1	1	1	1		3	1		1	1	
	51~60%												
	61~70%	7						5					
	71~80%	6	1					2					
	81~90%		1										
	91~100%	16	1					13					

5 考察

5.1 共通の親クラスからの継承またはインターフェースの実装を行っているクラス群内で，コードクローンをその中に持つクラス群の数と割合

項目 c では，項目 a，b で求めたクラス群の中で，共通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の数と割合を求めた．コードクローンを持つ割合が高いプロジェクトと低いプロジェクトという形で差がはっきりと分かる内容であった．タイプ 3 のコードクローンに対して，テンプレートメソッドの形成によるリファクタリングが可能かを調査したが，適用するのが難しいものが大半であった．この結果からタイプ 1，2 のコードクローンの割合が低いプロジェクトはよく管理されたプロジェクトで，本研究で想定したような事象を既に考慮して整理が行われていると考えられる．一方，割合が高いプロジェクトではその部分まで管理が行き届いていないと考えられる．

5.2 クラス群の大きさとコードクローンの種類別の割合

項目 d では，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．この分析から，共通したクラスを継承するクラス群は，比較的小さいクラス群に修正しやすいタイプ 1，2 が多く存在し，クラス群の規模が小さいうちにリファクタリングを行うほうがよいと考える．

5.3 クラス群におけるコードクローンの浸食率

項目 e では，クラス群の大きさと侵食率で分類した．共通のクラスを継承するクラス群の場合は，タイプ 1，2 においては，侵食率の高いクラス群は，比較的小さいクラス群に集中していた．ここから，修正のしやすいコードクローンはクラス群の規模が小さいうちにリファクタリングを行う方がよいと考えられる．また，タイプ 3 に分類されるコードクローンは，クラス群の大きさに関係なくコードクローンの侵食率が高く，解消が難しいことがわかる．

5.4 考察まとめ

共通したクラス群中にコードクローンが少ないプログラムについては，その存在しているコードクローンもタイプ 3 などの解消が難しいタイプのものが多く，助言を出す場合の方向性として，コードクローンの解消を促すような役割での助言は有効ではなく，既に似た事例となるクラスが存在することを伝えるような役割となると考えられる．一方で，共通したクラス群中にコードクローンが多く含まれるプロジェクトでは，コードクローンの解消を目的とした助言が有効であると考えられ，特に規模の小さいクラス群に対して，コードクローンが既に存在しているということを示すことで解消を促すことが有益で，ソフトウェアの品質向上に貢献できると考えられる．

6 まとめ

本研究では，共通したクラスを継承したクラス群内にどのようにコードクローンが出現したかを分析し，出現率が高いプロジェクトと低いプロジェクトに大きく二分されることを確認した．出現率が高いプロジェクトに対しては，コードクローン解消の余地が大きく，コードクローン解消を促すような助言が有効であると考えられる．今後，助言の方針を決める際にこれらの知見を活用できると考える．

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] 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.

[3] 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.

[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.