クラス階層構造を利用したコードクローン集約支援手法の改良

佐野由希子† 肥後 芳樹†（正員）
楠本 真二†（正員）

Improving Code Clone Merging Method Based on Class Hierarchy

Yukiko SANO†, *Nonmember*, Yoshiki HIGO†, and Shinji KUSUMOTO†, *Members*

† 大阪大学大学院情報科学研究科，吹田市
Graduate School of Information Science and Technology, Osaka University, 1–5 Yamadaoka, Suita-shi, 565–0871 Japan

あらまし ソースコード中に存在する同一，または類似したコード片をコードクローンという．我々は，クラス階層構造を利用したコードクローン集約支援手法を提案してきたが，集約可能と正しく判定できない場合があった．本論文ではこの問題を改善した．

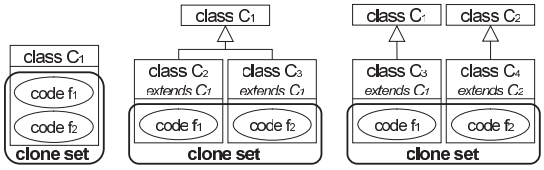
キーワード コードクローン，集約

1. まえがき

ソフトウェア保守を困難にしている一つの要因としてコードクローンが指摘されている [1]．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片のことである．例えば，あるコード片にバグが含まれていた場合，そのコード片のコードクローンすべてについて同様の修正の是非を検討しなければならない．

我々はこれまでに，コードクローンを一つのモジュールに集約する支援を行う手法を提案している [2]．その手法の中で，メトリックス DCH を定義している．DCH は，クラス階層内においてコードクローンがどの程度広く分布しているのかを表現するメトリックスである．一般的に，コードクローンはクラス階層において広く分布しているよりも，近くに存在している方が集約しやすい．しかし，DCH はクローンセット(注1)単位で計測されるため，クローンセット内の一部のコードクローンのみが離れて存在する場合と，すべてのコードクローンがクラス階層の様々な位置に分布している場合の区別がつかない．後者は集約を行うことが難しいが，前者については離れた位置に存在しているコードクローン以外は容易に集約することができる．

そこで本論文では，このメトリックス DCH の計測単位の細分化と，その細分化を実現するツールの試作を行った．以降，細分化の手法，及びそれを評価するために行った実験について述べる．



(a) $DCH = 0$ (b) $DCH = 1$ (c) $DCH = \infty$

図 1 コードクローンの分布状況と対応する DCH の値
Fig. 1 Code clone distributions and the corresponding DCH values.

2. 従来手法

2.1 メトリックス DCH

メトリックス DCH は文献 [2] にて下記のように定義されている．

> クローンセット S はコード片 $f_1, f_2, \cdots, f_n$ を含んでいるとする．C_i はコード片 f_i を含んでいるクラスとする．もしクラス $C_1, C_2, \cdots, C_n$ が共通の親クラスをもつ場合は，その共通の親クラスの中で，クラス階層的に最も下に位置するクラスを C_p で表すとする．また $D(C_k, C_h)$ はクラス C_k と C_h のクラス階層における距離（ノードをクラス，エッジを継承関係としたグラフ表現におけるホップ数）を表すとする．このとき，$DCH(S) = max\{D(C_1, C_p), \cdots, D(C_n, C_p)\}$ と表される．直観的には，メトリックス $DCH(S)$ はクローンセット S に含まれる各コード片間のクラス階層内における最大の距離を示す．例えば，S 中のすべてのコード片が一つのクラス内に存在する場合は $DCH(S)$ の値は 0（図 1 (a)），あるクラスとその直接の子クラス内に存在する場合は $DCH(S)$ の値は 1 となる（図 1 (b)）．例外的に，コードクローンが存在するクラスが共通の親クラスをもたない場合は $DCH(S)$ の値は ∞ とする(注2)（図 1 (c)）．

DCH の値を用いることにより，そのコードクローンがどのクラスに集約可能であるかを判定できる．例えば，$DCH = 0$ ならコードクローンの存在するクラス内に，$DCH \geq 1$ なら共通の親クラス内に集約を行うことができる．ただし，$DCH = \infty$ の場合は，集

(注1)：クローンセットとは，互いにコードクローンとなっているコード片の集合である．
(注2)：文献 [2] 中ではこの場合の $DCH(S)$ の値は -1 となっているが，距離を示すという概念上，本論文では直観的に分かりやすいであろう ∞ を採用した．

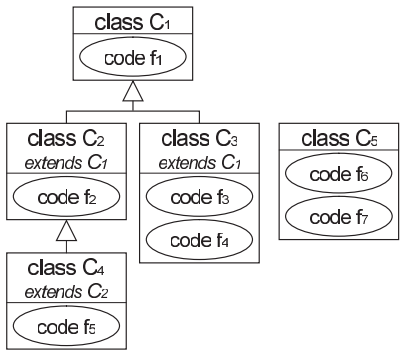


図 2　クローンセット S の例
Fig. 2　Example of clone set S.

約を行うことが難しい．

また，文献 [2] と同様に，メトリックス DCH は，JDK のクラスライブラリ等の修正不可能なクラスは除外して計算される．これにより，実際には修正不可能なクラスに対して集約を行おうとする場合には $DCH = \infty$ となり，そのような集約はできないことが分かる．

2.2 問　題　点

従来の DCH はクローンセット単位で計測されていた．この方法では，クローンセット内の一部のコード片のために DCH の値が大きくなってしまう場合がある．クローンセット S に含まれているコードクローンを f_1，f_2，$\cdots$，f_7 とし，これらが図 2 のように分布している場合を考える．この場合，クラス C_2 はクラス C_4 の親クラス，クラス C_1 はクラス C_2 とクラス C_3 の親クラスである．f_6 と f_7 の存在するクラス C_5 は，他のどのクラスとも共通する親クラスをもっていない．もしクローンセット S 内に f_6 と f_7 がなければ $DCH(S) = 2$ となるが，f_6 と f_7 のために値が ∞ になってしまう．

しかし，それは集約を行う上において好ましくない．同一クラス内に存在するクローンのみを集約したい，より多くのコード片を集約したい等の，細かな要求にこたえにくいためである．

3. 提 案 手 法

3.1 概　　要

従来手法の問題点を解決するため，本論文では，クローンセットをサブセットに分割し，そのサブセットごとに DCH を計測する手法を提案する．サブセットはすべての組合せで作成するのではなく，DCH の値が 0 となるサブセット，DCH の値が 1 となるサブセット，のように DCH の値ごとに作成する．

```
//入力：DCH計測対象クローンセット S_in
//出力：DCHの値により分割されたサブセットの集合 G
//(Gに含まれている各サブセットには，DCHの値が付与されている)
 1. group←φ, G←φ
 2. for(コードクローン∀f∈S_in){
 3.   C←fを所有するクラス
 4.   group[C]にfを追加
 5. }
 6. for(クラス∀C∈(S_inのコードクローンを所有するクラス群)){
 7.   if(2つ以上のコードクローンがgroup[C]に含まれる){
 8.     Gにgroup[C]を追加 //ここで追加されるサブセットのDCHは0
 9.   }
10. }
11. for(クラス∀C∈(S_inのコードクローンを所有するクラス群)){
12.   P←Cの親クラス(ただし，Cが親クラスを持たない場合は，P←NULL)
13.   h←1, S←group[C]に含まれるコードクローン
14.   while(P≠NULL){
15.     for(クラス∀M∈(S_inのコードクローンを所有するクラス群), ただしC≠M){
16.       if(M=P，もしくはMがPの1～h階層下の子クラスである){
17.         Sにgroup[M]に含まれるコードクローンを追加
18.       }
19.     }
20.     if(Sと同じコードクローンの組み合わせを持つサブセットがGにない){
21.       GにSを追加 //ここで追加されるサブセットのDCHはh
22.     }
23.     P←Pの親クラス(ただし，Pが親クラスを持たない場合は，P←NULL)
24.     h←h+1
25.   }
26. }
```

図 3　提案手法のアルゴリズム
Fig. 3　Algorithm of the proposed technique.

このようなサブセットの情報を文献 [2] で得られる各メトリックス値の情報と併用することによって，より多くの集約可能なコードクローンを検出し，より柔軟に集約の候補を提示することができる．よって，従来手法では $DCH = \infty$ で集約不可能と示されていたクローンセットに対しても，集約可能なサブセットを提示することができる．

3.2 アルゴリズム

提案するアルゴリズムの入力はクローンセットであり，出力は DCH の値に基づいて分割されたサブセットの集合である．このアルゴリズムは，$DCH = 0$ のサブセットを生成する STEP1 と，$DCH \geq 1$ のサブセットを生成する STEP2 からなる．2〜10 行では，STEP1 を行っている．まず，入力されたクローンセットは，コードクローンを含むクラスごとにサブセットとして分割される．そして，二つ以上のコードクローンをもつサブセットは，$DCH = 0$ のサブセットとして G に追加される．

11〜26 行は STEP2 を行っている．STEP2 では，STEP1 で生成されたサブセットを一つ指定し，そのサブセットのコードクローンを含むクラスを基点にして，下記の処理を行いながら親クラスを順にたどる．

（1）　現在注目しているクラスと，基点となるクラスとの距離（図 3 のアルゴリズムでは，変数 h である）を計測する（13 行目，24 行目）．

（2）　現在注目しているクラスから h 階層内に位置

表 1 実験結果
Table 1 Experimental result.

ソフトウェア	ソフトウェアの規模		集約可能と判定されたコードクローンの数					
			Pull Up Method			Extract Method		
	ファイル数	行数	従来手法	提案手法	差分	従来手法	提案手法	差分
(a)Apache Ant(v1.6.0)	650	189,915	65	68	3	68	72	4
(b)Areca backup(v5.5.6)	323	55,179	37	48	11	30	44	14
(c)Eclipse Checkstyle Plug-in(v4.4.0)	206	43,349	9	9	0	20	24	4
(d)JasperReports-Java Reporting(v2.0.4)	948	211,619	292	309	17	143	205	62
(e)jEdit(v4.2)	394	140,665	3	8	5	153	155	2
(f)JFreeChart(v1.0.9)	538	194,470	262	307	45	176	275	99
(g)Jimm-Mobile Messaging(v0.5.1)	56	27,166	0	0	0	28	28	0
(h)Robocode(v1.5.2)	209	48,717	17	17	0	60	60	0
(i)XUI RIA Framework(v1.0.4)	439	76,770	34	34	0	43	47	4
(j)ZK-Simply Ajax and Mobile(v3.0.2)	961	132,597	47	47	0	66	72	6
合計	4,724	1,120,447	766	847	81	787	982	195

するクラスに含まれるコードクローンを，指定したサブセットに併合する（15〜19 行目）．

（3） 同様のコードクローン集合をもつサブセットが G に登録されていない場合は，生成されたサブセットを $DCH = h$ のサブセットとして G に登録する（20〜22 行目）．

この処理を STEP1 で生成されたすべてのサブセットに対して行う．

3.3 従来手法との比較

この提案手法を用いることにより，従来の手法では $DCH(S) = \infty$ となってしまっていた図 2 のクローンセット S の DCH が，以下のように計測される．

$DCH = 0$ のサブセット：$\{f_3,f_4\}$,$\{f_6,f_7\}$

$DCH = 1$ のサブセット：$\{f_1,f_2,f_3,f_4\}$, $\{f_2,f_5\}$

$DCH = 2$ のサブセット：$\{f_1,f_2,f_3,f_4,f_5\}$

この結果を用いることにより，例えば，$DCH = 2$ のサブセット $\{f_1,f_2,f_3,f_4,f_5\}$ をクラス C_1 に集約し，更に，$DCH = 0$ のサブセット $\{f_6,f_7\}$ をクラス C_5 に集約することができる．このように集約を行った場合，七つあったコードクローンを二つに減らすことができる．

4. 実　　験

コードクローン集約支援環境 Aries [2] の出力するコードクローン情報を入力とし，改良手法を用いて計測された DCH を出力とするツールを実装し，提案手法の有効性を調べるための実験を行った(注3)．

実験では，従来手法を用いた場合と提案手法を用いた場合において，集約可能と判定されるコードクローンの数の差を計測した．表 1 に表しているオープンソースソフトウェア 10 種を実験の対象とし，それらに対して，コードクローン集約パターン “Pull Up Method” と “Extract Method” の二つを適用した．これらを用いて集約を行うための詳細な条件については，文献 [2] を参照されたい．なお，この実験では，30 字句（行数に換算して約 5，6 行）以上のコードクローンを対象とした．

対象ソフトウェアのクローンセット数を l，クラス階層の階層数を m，クローンセット内のコードクローンを含むクラス数を n とすると，アルゴリズムの計算時間は $O(lmn^2)$ である．しかし，10 万行規模のソフトウェアでもクローンセット数は数百個程度，階層数は数層程度，クラス数は数個から 10 数個程度であるため，ツールの実行時間はさほど長くない．コードクローン検出から DCH の計測まで，数分以内で処理が終了した．

4.1 実 験 結 果

従来手法と提案手法を実験対象ソフトウェアに適用し，“Pull Up Method” と “Extract Method”，の条件に一致するコードクローン数を計測した結果を，表 1 の「集約可能と判定されたコードクローンの数」に示す．差分には提案手法で計測したコード片数から従来手法で計測したコード片数を引いた値を示している．

表 1 より，提案手法を用いた場合，従来手法を用いた場合よりも “Pull Up Method” では 81 個，“Extract Method” では 195 個多くのコード片が検出されている．

(注3)：実験を行った環境は CPU が Pentium4 2.66 GHz，主記憶容量が 2.00 GByte，OS が Windows XP/Professional SP2 である．

表 2　提案手法で集約可能となったコード片数に対応するクローンセット数
Table 2　Number of clone sets corresponding number of code fragments which can be aggregated by the proposed technique.

ソフトウェア	集約可能となったコード片数										合計
	2	3	4	5	6	7	8	9	10	11	
(a)	2	1	0	0	0	0	0	0	0	0	3
(b)	2	1	3	0	1	0	0	0	0	0	7
(c)	2	0	0	0	0	0	0	0	0	0	2
(d)	6	1	3	0	1	0	2	0	3	0	16
(e)	1	0	0	1	0	0	0	0	0	0	2
(f)	14	3	14	0	0	2	2	0	1	1	37
(g)	0	0	0	0	0	0	0	0	0	0	0
(h)	0	0	0	0	0	0	0	0	0	0	0
(i)	0	0	1	0	0	0	0	0	0	0	1
(j)	1	0	1	0	0	0	0	0	0	0	2
合計	28	6	22	1	2	2	4	0	4	1	70

4.2 考　　察

多くの場合において (13/20)，提案手法を用いた方が，多くのコードクローンを集約可能と判定した．ソフトウェアの規模と差分とを比較すると，行数の多いものは差分も多い傾向がある．10 万行以上の規模のソフトウェアについては，従来手法と提案手法とでコード片 6 個分以上の差が出ている．規模が 10 万行未満のソフトウェアは，かなりの差が出るものもあれば，全く差が出ないものもあった．また，ファイル数と差分には関連性は見られなかった．

表 1 には集約可能なコード片の総数を示しているが，各クローンセットについて詳細に調べたところ，表 2 のようになった．なお，表 2 中のアルファベットは表 1 のソフトウェアのアルファベットと対応している．つまり，(a) の Apache Ant には二つのコード片が集約可能となったクローンセットが二つ，三つのコード片が集約可能となったクローンセットが一つあったというように見る．表 2 より，提案手法を用いて多くのコード片が新たに集約可能となったクローンセットはあまりなかった．新たに 5 個以上のコード片が集約可能となったクローンセットが 14 個だったのに対し，新たに 2 個しか集約可能にならなかったものが 28 個であった．

集約できるコード片の数が少ない場合，一つひとつコード片の内容を見て集約すべきか判断するのでは，集約により減らせる保守コストよりも集約に必要となるコストの方が高いかもしれない．しかし，その後も頻繁に修正の繰り返される可能性が高いコード片であれば，それらを集約することによって，将来的な保守コストを大きく減らすことができる．頻繁に修正される場所を予測するための手法としては，過去の修正履歴を調査するといった方法が考えられる．過去に頻繁に修正が行われている場所については，その後も修正の繰り返される可能性が高いと推測される．本手法は，そのような予測手法とともに使うと効果が高くなるだろう．

本手法を用いて新たに見つかった集約可能なコードクローンを調査した結果，同じディレクトリ内に存在しているものが多いことが判明した．同じディレクトリ内のファイルは似た機能を実装していることが多いので，これらのコードクローンは意味的に似たものである可能性が高い．例えば，四つのコードクローンが

ant\taskdefs\optional\ejb\BorlandGenerateClient.java
ant\taskdefs\optional\XMLValidateTask.java
ant\taskdefs\Property.java
ant\util\ClasspathUtils.java

の四つのファイルに分布するクローンセットは，従来手法では $DCH = \infty$ となったが，提案手法では taskdefs ディレクトリ内のファイルに分布する三つのコードクローンは，$DCH = 1$ で集約可能と判定された．四つのファイルのうち，ClasspathUtils.java ファイルは ant のクラスパスに関する機能を提供しているが，それ以外の taskdefs ディレクトリ内の三つのファイルは，どれも ant のタスクの動作を定義している．このようなコードクローンは集約してよいと思われる．

対象ソフトウェアによって，その差分の量は大きく変わるものの，従来手法を用いるよりも提案手法を用いた方が多くの集約可能なコード片を検出できた．更に，新たに検出された集約可能なコードクローンは，集約を行ってよいものであることも分かった．また，対象ソフトウェアの規模が大きいほど，差分が大きくなる傾向があるので，規模の大きいソフトウェアに本手法を用いれば，集約可能なコード片を多く検出するのに役立つと考えられる．ただし，本手法は集約可能なコード片を検出することを目的としており，実際に集約すべきかどうかは開発者や保守管理者が見て判断する必要がある．

5. む　す　び

本論文では，メトリックス DCH の計測単位の細分化を行った．また，従来の計測方法に比べ，提案した計測方法が集約可能なコードクローンを多く検出できることを実験で確認した．

謝辞 本研究は，文部科学省科学研究費補助金基盤研究（A）（課題番号：17200001），基盤研究（C）（課題番号：20500033）及び若手研究（スタートアップ）（課題番号：19800022）の助成を得た．

文　献

[1] M. Fowler, Refactoring: Improving the design of existing code, Addison Wesley, 1999（児玉公信，友野晶夫，平澤　章，梅澤真史（訳），リファクタリング　プログラミングの体質改善テクニック，ピアソン・エデュケーション，2000.）.

[2] 肥後芳樹，神谷年洋，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支援環境,” 信学論（D-I），vol.J88-D-I, no.2, pp.186–195, Feb. 2005.

（平成 21 年 3 月 5 日受付，7 月 27 日再受付）