リファクタリング支援を目的としたコードクローンの優先順位付けと可視化

石塚凌[1]　津田直彦[1]　鷲崎弘宜[1]　深澤良彰[1]
杉村俊輔[2]　保田裕一朗[2]

概要：コードクローンはコピー&ペースト等によって生じる重複したコード片を指す．コードクローンはソフトウェアの修正漏れや規模の増大を引き起こすため，リファクタリングによってクローンを減らす事が重要である．しかし，既存のクローン検出ツールは大量にクローンを出力してしまうため，修正するべきクローンを開発者が特定するのが困難である．本研究では，修正するべきクローンの特定を支援するため，クローンの優先順位付け・可視化する手法を提案する．具体的には，まず「親クラスの抽出」や「メソッドの引き上げ」等の各リファクタリングパターンに見合うようにクローンをファイル単位・クラス単位等に集約する．そして，リファクタリングの効果やし易さを加味した複数のソフトウェアメトリクスを用いて，クローンやクローンを含むファイルの並び替えを行う．また，我々は提案手法を可視化するツールを作成し，提案手法によってどのようなコードクローンが得られるか調査した．

キーワード：コードクローン，可視化，リファクタリング，ソフトウェア保守

Ranking and Visualizing Code Clones for Refactoring

RYO ISHIZUKA[†1]　NAOHIKO TSUDA[†1]
HIRONORI WASHIZAKI[†1]　YOSHIAKI FUKAZAWA[†1]
SHUNSUKE SUGIMURA[†2]　YUICHIRO YASUDA[†2]

Abstract: Code clones are duplicated code fragments in software systems. Refactoring code clones is important because code clones cause software omission and a large method. However, existing code clone detection tools report a lot of code clones. So, it is hard for developers to identify code clones that should be modified. In this paper, we introduce a method to rank and visualize code clones to identify them that should be modified. Specifically, we integrate code clones per clone, per file, and per class to match refactoring patterns such as "Extract Super Class" and "Pull Up Method". Next, we rank code clones and source files using software metrics taking into account the effect of refactoring. Then, we implement a tool that realizes our proposed method. Finally, we apply our method to one industrial software to investigate what type of code clones are extracted.

Keywords: Code Clone, Visualization, Refactoring, Software Maintenance

1. はじめに

ソフトウェア中に存在する重複したコード片の事を示すコードクローンは，コードの再利用を目的としたコピー&ペーストや，ソフトウェアの設計段階における共通機能の括りだしの失敗，統合開発環境によるソースコードの自動生成等，様々な要因によって作られる．

コードクローンはソフトウェアの修正漏れや，規模の増大を引き起こすため，コードクローン検出ツール[7, 13]によってコードクローンを検出し，リファクタリング[10]によってコードクローンを除去する事は重要である．しかし，全てのコードクローンをリファクタリングする事は時間の制約上困難である．そのため，開発者はクローン検出ツールが出力するコードクローンの一覧から，修正するべきコードクローンを特定する必要がある．

そこで，我々は開発者が修正するべきコードクローンの特定を支援するため，既存のコードクローン検出ツールの検出結果からコードクローンを抽出する手法を提案する．まず，コードクローンに関するリファクタリングパターン[12]に見合うように，検出されたクローンセットをソースファイル単位，クラス単位等に集約する．例えば，クラス単位の集約は「メソッドの引き上げ」と呼ばれる子クラス間で重複しているメソッド単位のコードクローンを除去する事を目的としている．次に，コードクローンを規模や広がり，結合度等の観点から評価できるように複数のソフトウェアメトリクスを算出する．更に，算出された偏差値としてスコア化する事で，修正するべきクローンセットやソースファイルの順位付けを行う．また，開発者は算出されたメトリクスの偏差値を参照する事で，そのコードクローンが上位に順位付けされた原因を容易に理解する事ができる．

1 早稲田大学
Waseda University.
2 小松製作所
Komatsu Ltd.

ⓒ2020 Information Processing Society of Japan

また，我々は，提案手法によって並び替えられたコードクローンを可視化するため，並び替えられたクローンセットの一覧や，クローンになっているコード片を実際に確認できる可視化ツールを作成した．可視化ツールを用いることで，着目したクローンセットに関連しているファイルの確認や，あるファイルと重複したクローンを持つファイルの特定を容易に行う事ができる．

以降，2 章では既存のコードクローンの抽出・可視化手法や，コードクローンに関するリファクタリングパターンについて説明する．3 章では我々の提案手法について説明し，4 章においてある商用プロジェクトに対して提案手法を適用した結果を示す．最後に，5 章において本研究のまとめと今後の課題について述べる．

2. 関連研究

2.1 有害なコードクローンの抽出

既存のコードクローン検出ツールから，修正するべきコードクローンのみを抽出する事は，コードクローンの保守のコストを削減するために有用である．Choi ら[8]は CCFInderX[7]で測定できる 3 種類のメトリクス（LEN: コードクローンのトークン行数, POP: クローンの出現回数, RNR: コードクローンの非繰り返し率）を用いてコードクローンを順位付けする手法を提案した．Arammongkolvichai ら[5]はコードクローンを決定木によって有害なコードクローンを抽出する手法を提案した．彼らは提案手法によってクローンの検出の適合率が向上した事と，学習モデルが過学習を引き起こしてしまう事を報告した．

一方で，特定のコードクローンの抽出を目的として，コードクローンに関するメトリクスの提案や評価が報告されている．肥後ら[3]は「クラスの抽出」と「メソッドの引き上げ」を目的としたメトリクスを複数提案している．Batova ら[6]は実施されたリファクタリングと複数のソフトウェアメトリクスの相関を調査した．Tsantalis ら[19]は検出されたコードクローンがリファクタリング可能か調査し，タイプ 1 のクローンや出現するファイルの位置が近いクローンほどリファクタリングしやすい事が報告されている．また，Higo ら[11]はソースコードの語彙がコードクローンの抽出に有用だったことを報告している．

上記の研究を踏まえ，我々はコードクローンを規模・広がり・複雑度・結合度・クローンタイプ・語彙の 6 観点で評価するように複数のソフトウェアメトリクスを用いてコードクローンの順位付けを行う．また，ソフトウェアメトリクスを偏差値として正規化する事で，有害なコードクローンの特定と，そのコードクローンが上位に順位付けされた原因の特定を支援する．

2.2 コードクローンの可視化

コードクローンの可視化手法について，Zibran ら[20]が樹形図や散布図等のコードクローンの可視化手法を纏めている．また，既存のコードクローン検出ツールでは，CCFinderX[7]は GUI フロントエンド GemX を提供している．GemX は CCFinderX で検出されたコードクローンがどのファイルに散らばっているか散布図によって可視化する．Yoshimura と Mibe[19]はコードクローンを樹形図によって可視化するツールを実装し，17000 ファイルの COBOL プロジェクトに対して適用を行った．また，Forbes ら[9]はコード片を入力として，そのコードに類似するコードを出力する Doppel-Code を提案した．Doppel-Code によって出力されるコード片はコード片の類似度とそのコード片がソースコード中に出現する回数によって並び替えられる．

2.3 コードクローンのリファクタリングパターン

コードクローンのリファクタリングパターンはコードクローンの除去や，悪いコードクローンから良いコードクローンへの置き換えを目的として複数提案されている[4, 10, 12]. Fowler[10]はコードクローン等の不吉な匂いを除去するためのリファクタリングパターンを提唱している．また， 肥後と吉田[4]はコードクローンに関するリファクタリングパターンと，それを支援する手法やツールについて紹介している．彼らによって紹介されているリファクタリングパターンを表 1 に示す.

表 1　肥後と吉田[4]によって紹介されているコードクローンに関するリファクタリングパターン

リファクタリングパターン	リファクタリングパターンの目的
メソッドの移動	複数ファイルに重複するメソッドを 1 つに集約する
メソッドの抽出	複数メソッドの間で共通しているロジックを特定し, 1 つのメソッドとして抽出する
（親）クラスの抽出	2 つ以上のクラスでの多くの機能が共通している場合に，親クラスを定義して共通部分を抽出する
メソッドの引き上げ	子クラス間で重複しているメソッドを親クラスに集約する
テンプレートメソッドの形成	子クラス間で異なる機能を残しつつ、共通部分を親クラスに集約する
メソッドのパラメータ化	異なる値を利用している複数のメソッドを，異なる値を引数として受け取るメソッドに集約する

また，我々はコードクローンとして検出される switch 文や if-else 文の羅列に対するリファクタリングパターンである「ポリモーフィズムによる条件記述の置き換え」に着目した.「ポリモーフィズムによる条件記述の置き換え」は同じ条件分岐の組み合わせが複数ファイルに現れている場合に，その条件分岐の各アクション部をサブクラスでオーバーライドするメソッドに移動する事である．条件記述をサブクラスに置き換える事で，新しいタイプコードの追加が必要になった時の修正作業を容易に行う事ができる．

ⓒ2020 Information Processing Society of Japan

「ポリモーフィズムによる条件記述の置き換え」の 1 例として図 1 を挙げる．図 1 ではクラス Shop が持つメソッド getMenu()と calcSalary()において，タイプコード_type に応じて返す値が異なっている．ここでは，2 つのタイプコード SHOP_A, SHOP_B をそれぞれサブクラスに置き換え，getMenu()と calcSalary()を各サブクラスでオーバーライドする．これによって，新しいタイプコードの実装を 1 つのサブクラスを定義するだけで実現できるため，switch 文の修正漏れを防ぐ事ができる．

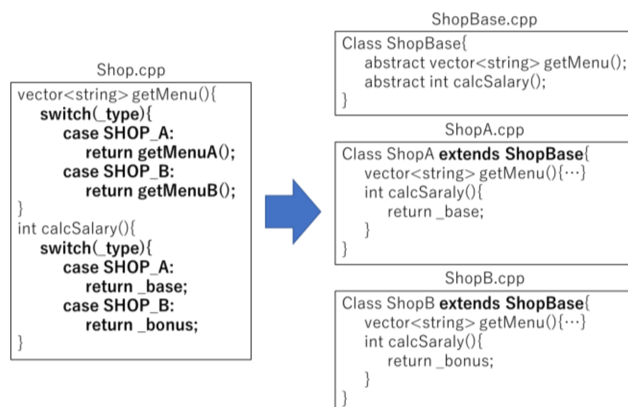

図 1　ポリモーフィズムによる条件分岐の置き換えの例

3. 提案手法

本研究の提案手法は，コードクローン検出ツールで検出されたコードクローンを表 2 に示す単位に集約し，それぞれの単位で修正するべきだと考えられるコードクローンを抽出・可視化する事を目的とする．提案手法の全体像を図 2 に示す．提案手法はソースコード，コードクローン検出ツールによって出力されたクローンセット，静的解析ツールによって出力されるクラスの継承構造のリストを入力とする．提案手法は次の 4 ステップに分かれる．

Step1. クローンセットを表 1 に示す単位に集約する．

Step2. 集約された単位毎に，目的とするリファクタリングパターンに見合った複数のソフトウェアメトリクスの算出を行う．

Step3. 各ソフトウェアメトリクスを偏差値を算出し，合計値でクローンセットやソースコードを並び替える．

Step4. コードクローンや算出されたソフトウェアメトリクス値の一覧をツール上に出力する．

表 2　集約する単位と目的とするリファクタリングパターンの対応

集約する単位	目的とするリファクタリングパターン
クローンセット	メソッドの移動，メソッドの抽出
ソースファイル	親クラスの抽出
クラス	メソッドの引き上げ
switch 文を含むクローン	ポリモーフィズムによる条件分岐の置き換え

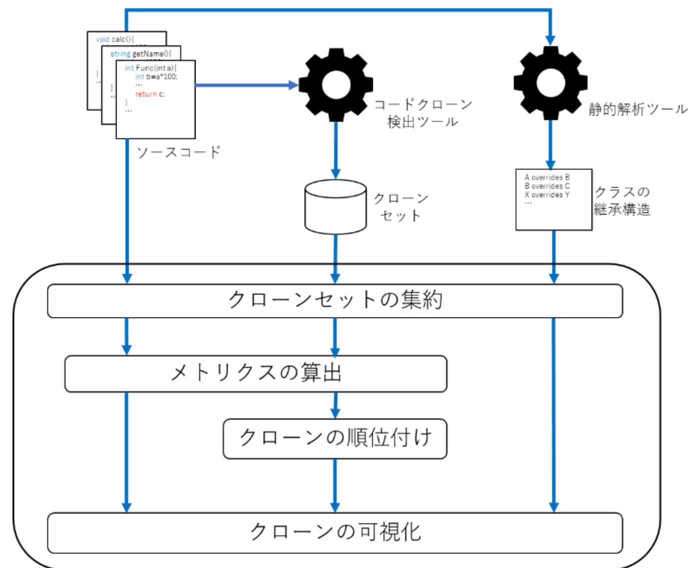

図 2　提案手法の全体像

以降の節では，各集約単位で利用するソフトウェアメトリクスや，我々が作成したツールが出力する内容について述べる．

3.1 クローンセットの抽出と可視化

修正するべきだと考えられるコードクローンの絞り込みを支援する事を目的として，クローン単位で出力される既存のクローン検出ツールが出力するクローンセットの並び替えと可視化を行う．

コードクローンを様々な観点から評価するため，我々は 5 つのクローンメトリクスを用いた（表 3）．5 つのメトリクスを用いる事で，コードクローンの規模・広がり・複雑さ・リファクタリングのしやすさ・結合度をそれぞれ評価する事ができる．

表 3　クローンセットの評価に用いるメトリクス一覧

メトリクス名	説明
LEN [7]	クローンセットに含まれるコードクローンのトークン長
NIF [7]	クローンセットに含まれるコードクローンが出現するファイル数
RNR [7]	クローンセットに含まれるコードクローンの非繰り返し率
CTYPE	クローンセットのコードクローンタイプ
NCM	クローンセットに含まれるコードクローンの関数呼び出し回数

次に，ソフトウェアメトリクスの正規化とコードクローンの並び替えを行うため，クローンメトリクス測定値の偏差値としてスコア化する．まず，クローンメトリクス測定値をスコア化できるように変換する．LEN, NIF, RNR, NCM については測定値をそのまま偏差値の算出に用いる．クロ

ⓒ2020 Information Processing Society of Japan

ーンタイプを表す CTYPE については，関連研究[18]で得られた結果を基に，以下の値を偏差値の算出に用いる．

$$CTYPE' = \begin{cases} 100\ (CTYPE = 1) \\ 0\ (CTYPE \neq 1) \end{cases}$$

次に，クローンセット i のクローンメトリクス値x_iを用いて偏差値$Deviation\ Value_{i,x}$を算出する．算出式を以下に示す．また，μ_xはクローンメトリクス x の平均値，σ_xはクローンメトリクス x の標準偏差をそれぞれ示す．

$$Deviation\ Value_{i,x} = \begin{cases} 50 + \dfrac{(x_i - \mu_x)}{\sigma_x}\ (\sigma_x > 0) \\ 50\ (\sigma_x = 0) \end{cases}$$

最後に，偏差値の平均値を各クローンセットのスコアとして算出し，偏差値の高い順にクローンセットを並び替える．クローンセットiのスコア$Score_i$は次のように算出する．

$$Score_i = \frac{1}{5} \sum_{x=\left\{\substack{LEN,NIF,RNR \\ CTYPE,NCM}\right\}} Deviation\ Value_{i,x}$$

可視化においては，並び替えられたクローンセットを可視化するため，我々のツールはクローンセットの一覧とコードクローンビューワーを表示する．一例として，GitHub で公開されている Sakura Editor[14]に対して CCFinderX[7]を用いてクローン検出を行った結果を図 3，図 4 に示す．図 3 に示すクローンセットの一覧を参照する事で，開発者は修正するべきコードクローンの候補を確認する事ができる．クローセットの一覧からクローンセットを選択すると，図 4 に示すようなコードビューワーが表示される．このコードビューワーは，選択したクローンセットを含むファイル一覧を表示するため，ユーザーは着目したクローンセットに関連しているファイルを即座に確認する事ができる．また，ファイル一覧から選択したファイルのコードを 2 ファイルまで確認できるため，選択した 2 ファイル間の共通点や違いをビューワー上で調べる事ができる．

cloneID	Average Score	LEN	LEN(Score)	NIF	NIF(Score)	RNR	RNR(Score)	CTYPE	CTYPE(Score)	NCM	NCM(Score)
1256	84.13864303	61	44.30006413	25	235.8981058	0.754097999	52.89132430	2	45.21389583	2.0	42.38982503
35	70.97289542	280	80.07310347	1	46.93357689	0.25	34.96788431	1	70.89382023	42.0	121.9960922
1463	69.71520474	235	72.72247895	1	46.93357689	0.727660000	51.95130886	1	70.89382023	34.0	106.0748387
3266	69.67410232	51	42.66659202	15	157.1628854	0.980392	60.93731323	2	45.21389583	2.0	42.38982503
1133	66.51211816	270	78.43963136	1	46.93357689	0.396296	40.16950697	1	70.89382023	29.0	96.12405537
3906	66.05551670	207	68.14875703	1	46.93357689	0.845411	56.13800069	1	70.89382023	25.0	88.16342865
1236	64.89503061	190	65.37185443	1	46.93357689	0.984211000	61.07309956	1	70.89382023	21.0	80.20280194
1858	64.60549368	273	78.92967299	1	46.93357689	0.556777000	45.87548393	2	45.21389583	34.0	106.0748387
1337	64.41155709	348	91.18071386	2	54.80709893	0.747126000	52.64343158	2	45.21389583	20.0	78.21264526
2041	63.94925614	324	87.26038078	2	54.80709893	0.904320999	58.23257328	2	45.21389583	18.0	74.23233190
433	63.20339574	331	88.40381126	1	46.93357689	0.882175	57.44516189	1	70.89382023	7.0	52.34060843
2558	62.81516473	291	81.86992280	1	46.93357689	0.670103	49.90484282	2	45.21389583	26.0	90.15358533
60	62.41408085	252	75.49938155	2	54.80709893	0.571429	46.39644263	2	45.21389583	26.0	90.15358533
2183	62.20111095	300	83.34004770	2	54.80709893	0.936667	59.38265042	2	45.21389583	15.0	68.26186186
2212	62.18786190	266	77.78624251	1	46.93357689	0.864662	56.82247899	2	45.21389583	23.0	84.18311529
793	62.01963540	127	55.08098010	8	102.0482311	0.937008	59.39477484	2	45.21389583	5.0	48.36029507
1468	61.98525247	158	60.14474366	2	54.80709893	0.892404999	57.80889433	1	70.89382023	14.0	66.27170518
211	61.66887579	231	72.06909010	4	70.55414301	0.735931	52.24538813	2	45.21389583	15.0	68.26186186
2610	61.30340900	137	56.71445221	1	46.93357689	0.722627999	51.77239374	1	70.89382023	21.0	80.20280194
3103	60.93012469	418	102.6150186	1	46.93357689	0.0454545	27.69517346	2	45.21389583	22.0	82.19295861
851	60.89953750	226	71.25235405	1	46.93357689	0.867257	56.91474543	2	45.21389583	23.0	84.18311529
2481	60.36163256	153	59.32800760	1	46.93357689	0.908496999	58.38105291	1	70.89382023	14.0	66.27170518
3970	60.2922323	116	53.28416077	1	46.93357689	0.844828	56.11727186	1	70.89382023	18.0	74.23233190
3102	60.26538979	399	99.51142165	1	46.93357689	0.0952381	29.46525263	2	45.21389583	21.0	80.20280194

図 3　並び替えられたクローンセットの一覧

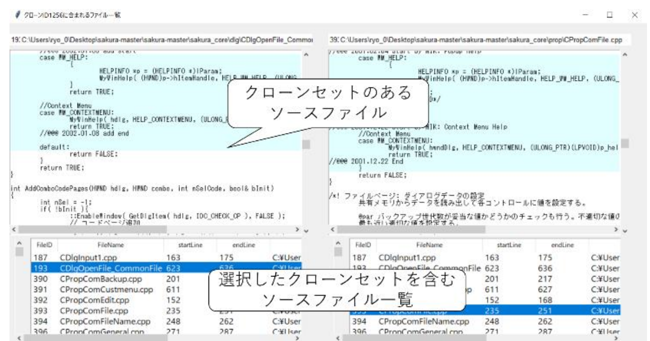

図 4　クローン単位のコードビューワー

3.2 ソースファイルの抽出と可視化

○○1.cpp や○○2.cpp といった本来共通化するべき機能を持つファイルを特定する事を目的として，ソースファイルの抽出と可視化を行う．まず，各ファイルに含まれているコードクローンの一覧を取得する．次に，ソースファイルの評価に用いた 4 つのクローンメトリクスを表 4 に示す．

表 4　ソースファイルの評価に用いるメトリクス一覧

メトリクス名	説明
CLEN	ソースファイルのコード片でコードクローンに含まれているトークンの長さ
NSF	ソースファイルに対して RST[2]の値が閾値以上になる他ファイルの数
RSI [7]	ソースファイルのトークンの内，そのファイル内で出来たコードクローンに含まれているトークンの割合
NCM	ソースファイルのコードクローンに含まれているコード片の関数呼び出しの回数

CLEN, RSI, NCM をそれぞれ用いる事で，各ソースファイルに含まれているクローンの規模・複雑度・結合度を評価する事ができる．NSF はそのファイルの多くがコードクローンの関係になっている他ファイル数を表す．NSF の算出で用いられる RST[2]は 2 ファイル間のクローンペアによってファイルが覆われる割合を示す．すなわち，NSF の値はそのファイルと機能を共通化すべきファイルの候補の数を表している．ソースファイルiの NSF と RST は次のように算出される．LEN_iはソースファイルiのトークン長，$CLEN_{i,j}$は 2 ファイル間で検出されたクローンペアのトークン長を表す．

$$NSF_i = \sum_{j=1, j \neq i} \begin{cases} 1\ (RST_{i,j} \geq threshold) \\ 0\ (RST_{i,j} < threshold) \end{cases}$$

$$RST_{i,j} = \frac{CLEN_{i,j}}{LEN_i}$$

次に，3.1 節と同様に，各クローンメトリクスの偏差値の算出と，ソースファイルの並び替えを行う．クローンメトリクスの偏差値の算出において，CLEN, NSF, NCM の測定値は偏差値の算出にそのまま用いる．RSI は測定値が大き

ⓒ2020 Information Processing Society of Japan

いほどコードクローンが単純な形の繰り返しになっている可能性が大きいため[7]，以下の値を偏差値の算出に用いる．

$$RSI' = 1 - RSI$$

可視化においては，クローンセットの可視化と同様に，我々のツールはソースファイルの一覧とコードビューワーを表示する．コードビューワーの Sakura Editor への適用例を図 5 に示す．このコードビューワーはファイル一覧から選択されたファイルが含むコードクローンの一覧や，RST を用いて他ファイルとの類似度を表示する．ユーザーは RST の値から，選択したファイルと重複した機能を持つファイルを容易に特定できる．

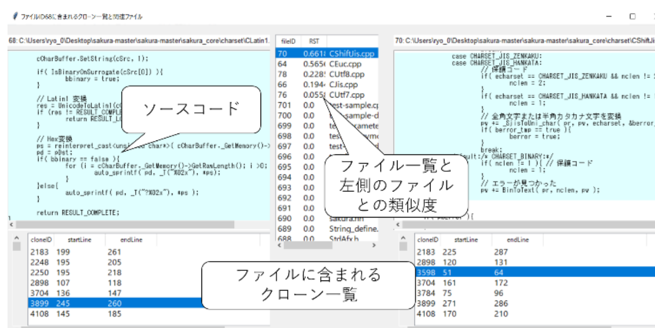

図 5　ファイル単位のコードビューワー

3.3 子クラス間で重複しているメソッドの抽出と可視化

子クラス間で重複しているメソッドを検出する事を目的として，クラス単位でのコードクローンの抽出と可視化を行う．そのため，コードクローンの検出結果とは別に，静的解析ツールによって得られる各クラスの継承関係のリストを入力として与える必要がある．継承関係のリストを基に，各クラスの子クラスが含むメソッドの一覧を作成する．ここで，関数名と引数の型が同じメソッドは 1 つに纏める．次に，子クラス間に含まれるメソッドの評価に用いた 3 つのクローンメトリクスを表 5 に示す．

表 5　子クラス間に含まれる各メソッドの評価に用いるメトリクス一覧

メトリクス名	説明
NCC	子クラス間に含まれる同名のメソッドのうち，コードクローンを含むクラスの数
CLEN	子クラス間に含まれる同名メソッドにおいて，クローンに含まれるトークン長の平均
NCM	子クラスに含まれる同名メソッドにおいて，コードクローンに含まれるコード片の関数呼び出し回数

クローンメトリクスの算出方法について，図 6 を例に挙げて述べる．図 6 において，クラス ShopBase を親に持つ 2 つの子クラス ShopA と ShopB が存在し，子クラスそれぞれが同名のメソッド printReceipt()と calcSalary()を持っている．そして，printReceipt()はコードの形が 2 クラス間で同一だが，calcSalary()はコードの形が 2 クラス間で異なっている．そのため，ShopBase の子クラス間で持つメソッド一覧には printReceipt()と calcSalary()が含まれる．そして，2 クラス間でクローンになっていた printReceipt()の NCC の値は 2 と算出され，2 クラス間でクローンになっていない calcSalary()の NCC の値は 0 と算出される．

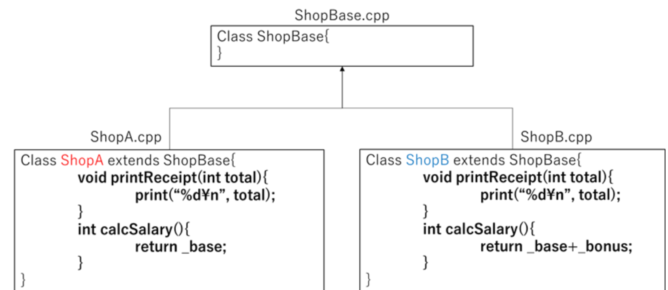

図 6　同名メソッドを含む 2 つの子クラス

次に，各メソッドのクローンメトリクスの偏差値の算出と，子クラス間に含まれるメソッドの並び替えを行う．クローンメトリクスの偏差値の算出において，NCC, CLEN, NCM の全てのメトリクス測定値をそのまま偏差値の算出に用いる．また，今回は各クラスにおいて，子クラスから引き上げるべきメソッドの特定を行えるよう，クラス毎に偏差値の算出を行った．

可視化においては，各クラス毎に並び替えられたメソッドの一覧と，メソッド一覧で選択したメソッドを含むコードクローンを確認できるビューワーをツール上に表示する．Sakura Editor への適用例を図 7，図 8 にそれぞれ示す．また，クラスの継承関係の抽出には Understand[18]を用いた．ユーザーは各クラスでコードクローンの多いメソッドの特定や，メソッドにどのようなコードクローンが含まれているか容易に確認する事ができる．

図 7　子クラス間に含まれるメソッド一覧

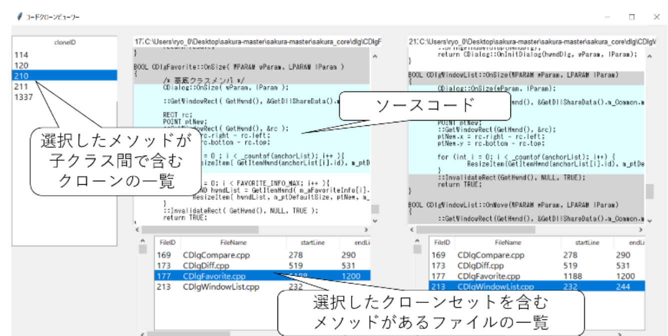

図 8　クラス単位のコードビューワー

ⓒ2020 Information Processing Society of Japan

3.4 switch 文を含むコードクローンの抽出と可視化

条件分岐を置き換える事でソースコードの保守性や可読性を向上させるため，我々は switch 文を含むコードクローンの抽出と可視化を行う．まず，字句解析した時に”switch”, “case”というキーワードの出現回数（NKS）が閾値以上で，RNR[7]の値が閾値以下のクローンセットを抽出する．また，switch 文のクローンセットは重複して出力されるため，検出された他の switch 文のクローンセットに完全に包含されるクローンセットを除去する[1].

次に，switch 文を含むクローンセットを評価するのに用いた 5 つのクローンメトリクスを表 6 に示す．

表 6 switch 文をクローンセットの評価に用いるメトリクス一覧

メトリクス名	説明
NKS	クローンセットを字句解析した時の，”switch”と”case”の出現回数
RDI	クローンセットに含まれる識別子（関数名や引数名）の重複率
NFSC	クローンセットと語彙の類似度が一定以上のクローンセットが出現するファイル数
CTYPE	クローンセットのコードクローンタイプ
NCM	クローンセットに含まれるコードクローンの関数呼び出し回数の平均

NKS, CTYPE, NCM は，各クローンセットの規模・リファクタリングのしやすさ・結合度を評価する．RDI はクローンセットに含まれる識別子の種類数を，クローンセットに含まれる識別子の出現回数で割った値に等しい．また，RDI の値が大きい時ほど同じ識別子が繰り返し使われているため，除去しやすい switch 文だと考えられる．

NFSC はそのクローンセットの switch 文に関連している switch 文がどれだけのファイルで出現しているかを表す．クローンセットの語彙の類似度の測定には jaccard 係数を用いた Higo らによる手法を用いた[11]．ただし，各クローンセットの語彙は次のように抽出した．

Step1. クローンセット内に含まれる識別子を字句解析によって抽出する．

Step2. 得られた各識別子を区切り文字で分割する．次に，各識別子を大文字や数字で区切る．ただし，大文字が連続している場合は区切らない．

Step3. 分割後の識別子を小文字に統一する．

次に, 3.1 節と同様に各クローンメトリクスの偏差値の算出と，クローンセットの並び替えを行う．クローンメトリクスの偏差値の算出には，全てのメトリクス測定値をそのまま用いる．

可視化においては, 3.1 節の可視化と同様に，クローンセットの一覧とコードクローンビューワーを表示する．コードビューワーの Sakura Editor への適用例を図 9 に示す．ユーザーは選択したクローンセットに含まれる switch 文や，その switch 文に関連する switch 文がどのファイルに存在するか容易に確認する事ができる．

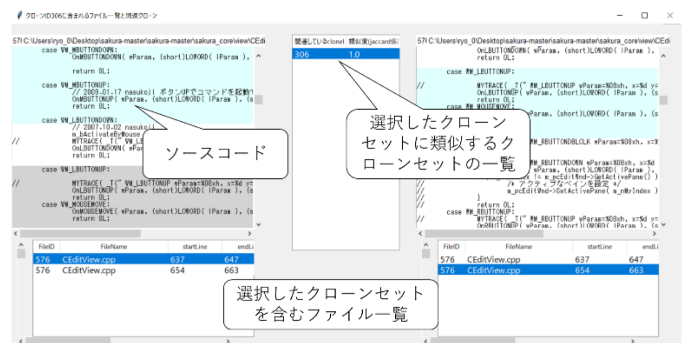

図 9 switch 文を含むクローンセットのコードビューワー

4. ケーススタディ

提案手法によって各集約単位においてどのようなコードクローンが抽出されるか評価するため、我々はある商用ソフトウェアに対して提案手法の適用を行った．提案手法を評価するためのリサーチクエスチョンを以下に示す．

RQ1: 提案手法によるクローンセットの並び替えによって修正するべきコードクローンを絞り込めるか？

RQ2: 提案手法によるソースファイルの並び替えによって共通化するべきソースファイルの特定できるか？

RQ3: 提案手法による子クラス間に含まれるメソッドの並び替えによって，修正するべきメソッドの特定できるか？

RQ4: 提案手法による switch 文を含むクローンセットの並び替えによって，修正するべき switch 文を特定できるか？

4.1 データセット

我々はある組み込みソフトウェアの 1 モジュールをデータセットとして用いた．表 7 に本研究で用いたソフトウェアの概要を示す．

表 7 解析対象のソフトウェアの概要

ファイル数	コード行数	開発言語
55	4077	C++

次に，コードクローンの検出には CCFinderX を用いた．まず，RQ1, 2, 3 の評価においては，CCFinderX のデフォルトの設定($LEN \geq 50,\ TKS \geq 12$)を用いた．その結果，26 個のクローンセットが検出された．RQ4 の評価においては，多くの switch 文を検出するために，デフォルトの設定より TKS の閾値を低く設定した($LEN \geq 50,\ TKS \geq 8$)．また，switch 文のクローンと見做す NKS と RNR の閾値をそれぞれ 3, 0.5 にそれぞれ設定した．その結果，46 個のクローンセットが検出され，その内 8 個のクローンセットが switch 文を含むクローンセットとして抽出された．

ⓒ2020 Information Processing Society of Japan

メトリクスの算出においては，3.2 節の NSF の算出に用いる RST の閾値を 0.8 に，3.4 節の NFSC の算出に用いる jaccard 係数の閾値を 0.7 にそれぞれ設定した．

4.2 データセットへの適用結果

4.2.1 クローンセットの抽出と可視化の結果

提案手法によって並び替えられた上位 5 つのクローンセットと下位 5 つのクローンセットのメトリクス測定値を表 8 と表 9 にそれぞれ示す．それぞれクローンについて調査を行った結果，上位 3 件のクローンセットはどれも同一クラスの子クラス間に拡散していたクローンである事が分かった．残りの 2 つのクローンセットのクローン片は出現しているファイル数こそ少なかったが，どのファイルもそのコードクローンによってほぼ全体が覆われていた．よって，クローンセットを含むファイルでコードの再利用が行われた可能性がある事が読み取れた．

一方で，下位 5 件のクローンセットはどれもファイル内クローンになっており，単純な関数宣言やメソッド呼び出し，case 句の羅列がクローンになったものであった．また，下位 3 件のクローンは同じ switch 文が重複して出力されたものだった．

表 8 提案手法によって抽出された上位 5 件のクローンセット

順位	1	2	3	4	5
平均偏差値	61.397	60.459	58.927	57.221	57.129
LEN	109	60	50	362	210
NIF	12	16	15	2	4
RNR	0.771	0.867	0.840	0.845	0.805
CTYPE	1	1	1	2	2
NCM	9	5	4	8	14

表 9 提案手法によって抽出された下位 5 件のクローンセット

順位	26	25	24	23	22
平均偏差値	41.508	42.287	42.251	42.934	44.841
LEN	66	53	72	108	150
NIF	1	1	1	1	1
RNR	0.288	0.509	0.472	0.176	0.127
CTYPE	2	2	2	2	2
NCM	3	2	2	5	7

4.2.2 ソースファイルの抽出と可視化の結果

表 10 に提案手法によって並び替えられた上位 10 件のソースファイルを示す．結果として，aaa.cpp を除く 9 件のファイルで，他のファイルと類似しているファイルが存在していた．また，ファイル名が○○(数字).cpp となっているファイルのクローンメトリクス測定値が同一である事から，これらのファイルがコードの再利用によって作られた可能性がある事を読み取る事ができる．

また，ファイル名が異なる zzz.cpp が yyy1.cpp, yyy2.cpp, yyy3.cpp とファイル全体を覆うクローンを持つ事が分かった．これらのファイルについて調査した所，4 つのファイルがあるファイルと共通して関連している事が分かった．すなわち，人手では見落としがちなファイル間のコードの重複についても提案手法によって抽出できると考えられる．

表 10 提案手法によって抽出された上位 10 件のソースファイル

ファイル名	平均偏差値	CLEN	NSF	RSI	NCM
xxx2.cpp	66.392	574	1	0.0	20
xxx3.cpp	66.392	574	1	0.0	20
yyy1.cpp	61.633	211	3	0.0	14
yyy2.cpp	61.633	211	3	0.0	14
yyy3.cpp	61.633	211	3	0.0	14
zzz.cpp	61.633	211	3	0.0	14
aaa.cpp	58.518	394	0	0.406	30
bbba.cpp	58.401	254	2	0.0	10
bbbb.cpp	58.401	254	2	0.0	10
bbbc.cpp	58.401	254	2	0.0	10

4.2.3 子クラス間で重複しているメソッドの抽出と可視化

ある 1 クラスの約 30 個の子クラスに含まれるメソッドを提案手法によって並び替えた結果を表 11 に示す．表 11 から，子クラス間に含まれているメソッド fa()に多くのコードクローンが含まれている事が読み取れる．

表 11 提案手法によって抽出された上位 10 件のソースファイル

メソッド名	平均偏差値	NCC	CLEN	NCM
fa()	75.499	28	100.0	7.129
fb(v1,v2)	57.901	3	82.0	4.0
fb(v3,v4,v1)	57.348	1	70.0	5.0
fb(v5,v6,v1)	55.884	1	70.0	4.0
fb(v1,v7)	55.699	1	68.0	4.0
fb(v1)	55.673	13	35.765	2.294
fc()	54.038	10	51.261	1.130
fb(v8,v1)	43.106	0	0.0	0.0
fd()	43.106	0	0.0	0.0
fb(v9,v10,v1)	43.106	0	0.0	0.0

一方で，1 クラスでしか出現していないメソッド fb(v3, v4, v1)等が，10 クラス以上でクローンが出現している fb(v1)や fc()より上位に並べられていた．fb(v1)や fc()を含む各クラスについて確認したところ，クローンを含まない同名のメソッドが多数存在し，CLEN や NCM の値を押し下げている事が分かった．そのため，今後はクローンメトリクスの測定値で平均値以外の値を取る，子クラスが独自で持つメソッドと子クラス間で共通して持っているファイル

ⓒ2020 Information Processing Society of Japan

を別々に評価する，等の改善が必要だと考えられる．

4.2.4 Switch 文を含むコードクローンの抽出と可視化

表 12 に提案手法によって並び替えられた上位 5 件の switch 文を含むクローンセットを示す．上位 3 件のクローンセットは文字リテラルのみが異なる同一のメソッドを呼び出す case 句の連続になっていた．そのような単純な switch 文はハッシュテーブル等を用いて取り除く事ができると考えられる．

一方で，クローンセットの語彙の類似度を用いたクローンセット間の結びつきは殆ど得られなかった．原因として，クローン検出ツールによって検出できなかった switch 文が存在した事や，タイプコード等の識別子が if 文の中等の swtich 文以外の形で使われている事が分かった．そのため，switch 文に出現するタイプコードがどのファイルに出現しているか等を評価するメトリクスを設計する必要があると考えられる．

表 12　提案手法によって抽出された上位 5 件の switch 文を含むクローン

順位	1	2	3	4	5
平均偏差値	57.963	55.404	53.287	49.497	49.328
NKS	9	7	7	6	8
RDI	0.636	0.643	0.600	0.667	0.556
NFSC	2	2	2	1	1
CTYPE	2	2	2	2	2
NCM	8	7	6	0	7

5. おわりに

我々は既存のコードクローン検出ツールの出力結果からコードクローンを複数の単位に集約し，目的としたリファクタリングパターンに見合うようにコードクローンを抽出・可視化する手法を提案した．また，提案手法を 1 つの商用ソフトウェアに適用し，提案手法によって修正するべきクローンセットの絞り込みや，人手では見落としがちなファイル間のコードの重複を抽出できる事を確認した．

今後の課題として，まず，コードクローンの並び替えの精度を向上させるために多くのクローンメトリクスを測定できるようにする必要がある．次に，提案手法を他のソフトウェアに対して適用し，どのようなコードクローンが得られるか分析する事が挙げられる．例えば，既存のクローンベンチマークの BigCloneBench[12, 13]等に対してクローンの抽出結果を適用し，提案手法によるクローンの抽出精度を定量的に評価する必要がある．最後に，提案手法のツールを開発者に利用して貰い，実際のクローン保守作業に向けてどのような改善が必要かフィードバックを受ける必要があると考えている．

参考文献

[1] 石津卓也，吉田則祐，崔恩瀞，井上克郎，”コードクローンに対するリファクタリング可能性に基づいた削減可能ソースコード量の調査，” 情報処理学会研究報告, Vol.2017-SE-197, No.7, pp.1-8, 2017
[2] 植田泰士，神谷年洋，楠本真二，井上克郎，“開発保守支援環境を目指したコードクローン環境，” 電子情報通信学会 D-I, J86-D-I(12), pp.863-871, 2003.
[3] 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，” 電子情報通信学会論文 D-I, J88-D-I(2), pp.186-195, 2005.
[4] 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，” コンピューターソフトウェア, Vol.28, No.4, pp.43-56, 2011.
[5] Vara Arammongkolvichai, *et al.*, “Improving Clone Detection Precision using Machine Learning Techniques,” Proceeding of the 2019 10th International Workshop on Empirical Software Engineering in Practice, pp.31-36, 2019.
[6] Gabriele Batova, *et al.*, “An experimental investigation on the innate relationship between quality and refactoring,” The Journal of Systems and Software, Vol. 107, pp.1-14, 2015.
[7] CCFinderX, http://www.ccfinder.net/index.html, Last Accessed Feb 2020.
[8] Eunjong Choi, *et al.*, “Extracting Code Clones for Refactoring Using Combinations of Clone Metrics,” Proceeding of the 5th ICSE International Workshop on Software Clones, pp.7-13, 2011.
[9] Christopher Forbes, *et al.*, “Doppel-Code: A Clone Visualization Tool for Prioritizing Global and Local Clone Impacts,” 2012 IEEE 36th Annual Computer Software and Applications Conference, pp.366-367, 2012.
[10] Martin Fowler, “Refactoring: Improving the Design of Existing Code,” Addison Wesley, 1999.
[11] Yoshiki Higo and Shinji Kusumoto, “How Should We Measure Functional Sameness from Program Source Code? An Exploratory Study on Java Methods,” Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp.294-305, 2014.
[12] Michele Lanza and Radu Marinescu, “Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems,” Springer-Verlag, 2006.
[13] Nicad Clone Detector, https://www.txl.ca/txl-nicaddownload.html, Last Accessed Feb 2020.
[14] Sakura-Editor, https://github.com/sakura-editor/sakura, Last Accessed Feb 2020.
[15] Jeffery Svajlenko and Chanchal K. Roy, “Evaluating clone detection tools with BigCloneBench,” 2015 IEEE International Conference on Software Maintenance and Evolution, pp.131-140, 2015.
[16] Jeffery Svajlenko, *et al.*, “Towards a Big Data Curated Benchmark of Inter-project Code Clones,” 2014 IEEE International Conference on Software Maintenance and Evolution, pp.476-480, 2014.
[17] Nikolaos Tsantalis, *et al.*, “Assessing the Refactorability of Software Clones,” IEEE Transactions on Software Engineering, Vol. 41, No. 11, pp.1055-1090, 2015.
[18] Understand, https://www.techmatrix.co.jp/product/understand/, Last Accessed Feb 2020.
[19] Kentaro Yoshimura and Ryota Mibe, “Visualizing Code Clone Outbreak: An industrial Case Study,” 2012 6th International Workshop on Software Clones, pp.96-97, 2012.
[20] Minhaz F. Zibran, “Analysis and Visualization for Clone Refactoring,” 2015 IEEE 9th International Workshop on Software Clones, pp.47-48, 2015.

ⓒ2020 Information Processing Society of Japan