利用部品の共通性に基づくソフトウェア部品分類手法の評価
ー既存の手法に基づく類似部品抽出手法との比較ー

2016SE016 日比野佑紀　2016SE033 加藤達也　2016SE040　北川雄大
指導教員：横森励士

1　はじめに

近年のソフトウェアは大規模化しており構成する部品数も増大している．このような環境下では，部品間の類似性などを利用してソフトウェアの構成要素を効率よく把握することが求められる．我々の研究グループでは，ソフトウェアがどの部品を利用しているかに基づいて，部品対ごとに類似性を計算し，クラスタリングを行ってソフトウェアを分類する手法を提案した．ソフトウェア内で担う機能が似ている部品ごとに分類できたかを様々な観点で確認したが，他の既存の分類手法との比較が十分に行われておらず，その観点の評価が必要である．

本研究では他の既存の手法に基づいて類似部品を抽出した結果と比較を行うことで，我々の研究グループが提案した手法を用いたソフトウェア内の部品の分類が適切であることを確認する．具体的にはコードクローン，実装や継承の関係，所属パッケージの情報をもとにそれぞれ抽出した部品群について，それらが樹形図上でどのように表現できるかについて調査を行う．それぞれの手法から得られる情報の特性を調査し，我々の研究グループが提案した手法がソフトウェア内の多くの部品を類似部品の集合に分けることができていることを示すことで，既存のコードの効率的な理解を支援できる手法であることを示せると考えた．

2　関連研究

2.1　ソフトウェア部品と部品グラフ

ソフトウェア部品とは，その内容をカプセル化したうえで，ソフトウェアを実現する環境において交換可能な形で配置できるようにしたシステムモジュールの一部を指す．各クラスのソースコードを記述しているファイルを部品とみなし，各部品を構成要素とする部品グラフを構築する．部品グラフ上の頂点は各部品を表し，辺は部品間の関係である利用関係を表現する．ある部品 A が他の部品 B を利用している場合，A から B への利用関係が存在しているとみなし，頂点 A から B への有向辺で表現する．

2.2　コードクローンとコードクローンのタイプ

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片のことを指し，類似した処理をコピーペーストで実現した場合に生成されやすい．元のコードに不具合が存在した場合そのすべてのコードクローンに対して修正が必要か検討することが必要となるように，保守性を低下させる要因の一つである．検出方法によって，得られるコードクローンに違いがあり，それは次のタイプのように分類される [2].

タイプ 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片

タイプ 2　タイプ 1 の違いに加えて，変数名や関数名などのユーザ定義名，変数の型などが異なるコード片

タイプ 3　タイプ 2 の違いに加えて，文の挿入や削除，変更が行われたコード片

タイプ 1 は，完全に同一なコード片であるため，検出は容易である．タイプ 2 は，型や名前の変化，タイプ 3 ではさらに文の追加・削除などを考慮する必要があるので，タイプが 3 に近づくほど，得られるコードクローンの種類が増えるが，抽出に必要な情報や解析が増加する．

2.3　ソースコード解析に基づくパターン抽出手法

ソースコードを解析してパターンを抽出し，理解支援に活用する研究が行われている．Zhong らは API の利用順を抽出し，API の利用方法の学習に活用するシステムを提案した [3]．Li らは C の中から関数呼び出しの実行順を取り出し，それをプログラミングのルールとしてルールから逸脱している記述があるかを調べる仕組みを提供した [4].

我々の研究グループでは，ソフトウェアの利用先がどれだけ一致しているかから各部品対の類似度を計算し，その類似度をもとにソフトウェアを分類することで，機能や役割が似ていると思われる部品を抽出する手法を提案した [5]．[5] の手法では，ソフトウェア部品の利用先がどれだけ一致しているかから，各部品対の類似度や距離を計算し，距離行列を作成する．距離行列からクラスタリングによって得られた樹形図をもとにソフトウェア部品を分類する．評価実験として，分類結果が「類似した部品の集合」をどれくらい含んでいたかを適合率，再現率の観点から評価を行ったところ，適合率，再現率ともに高い割合を示し，分類結果として含まれるべき部品の多くを含んでいた．橋本らは，部品群中の部品の役割の共通点と部品群中の部品が共通して利用している部品の役割が多くの部品で一致していることを確認し，全体としてソフトウェア部品を適切に分類できていることを示した [6].

3　既存の抽出手法との比較

3.1　研究の動機

過去の研究では，[5] の手法がソフトウェア部品を適切に分類できていることを複数の観点から示している [5][6]．しかし，他の既存の手法との比較が行われておらず，このような分類がこの手法でしかできないのかということが

明らかになっていない．本研究では，様々な観点から部品群を形成し，[5] の手法で得られた部品群との比較を行う．[5] の手法では，過去の実験からソフトウェア部品全体を関連性のある部品の集合に分類できているが，他の既存の手法ではそのような分類が難しいことを示す．これにより，コード理解を支援するときに類似部品を示す方法として [5] の手法が適切であることを示す．本研究では，様々な関連から部品群を形成するにあたり，以下を用いた．

- コードクローンの分析結果

  類似コード片を互いに共有する部品同士は，ソースコードの構造が似ており，関連をもつと考えた．関連をもつ部品同士をまとめることで，部品群を抽出する．

- 実装・継承に関する情報

  実装や継承を用いて部品を定義することで，それらの部品の間には共通の概念や共通の処理が存在することになり，関連性が生じる．派生 (実装) 先と元の部品同士を関連性があるとしてまとめ，部品群を抽出する．

- 所属パッケージに関する情報

  部品数が多くなった場合，開発者はパッケージを用いて機能的な面などから分類する．親子関係を考慮しながら，それぞれのパッケージに所属する部品を一つの部品群として抽出する．

3.2 比較の手順について

それぞれの手法で求めた部品群を，[5] の手法を行った結果の樹形図上で評価する．また，[5] の手法を行った結果得られた部品群を，他の手法で分類した結果上に示すことで評価を行う．具体的な手順を以下に示す．

1 各分類手法ごとに 3.1 節で説明した方法で，部品間の関係を表す図を作成する

2 それぞれの図から対応する部品群を抽出する

3 それぞれの手法で得られた部品群を [5] の手法で得た樹形図上で表現し，部品群の適切さを確認する

4 [5] の手法で得た樹形図から得られた部品群をそれぞれの図の上で表現し，それぞれの分類手法に特徴があるかを確認する

3.3 比較対象に用いたコードクローン分析ツール

本研究では，CCFinder と SourcererCC の 2 つのコードクローン分析ツールにより得られた結果を利用して，大きく分けて 2 通りの部品群を作成する．CCFinder は，ソースコードからトークン列を抽出し，トークン列中のトークンの種類が一致する部分をコードクローンとして検出する．トークンの種類をもとに判定するので，CCFinder は変数名や関数名などの異なるコード片も，コードクローンとして検出できる．このような処理方法により，数百万行規模のシステムに対し実用的な時間でタイプ 2 のコードクローンを検出することができる [1]．SourcererCC は，各ソースファイルからトークンの集合を抽出し，各ソースファイル間のトークン集合の類似度を計算することで，類似したファイルのペアを抽出する．この手法を用いることで，多くのプログラミング言語に適用可能となり，コード片が追加されたり一部が変更されたようなタイプ 3 のコードクローンも高速に検出可能となる [7]．CCFinder では一致するトークン列の長さを，SourcererCC では類似度をコードクローン検出の閾値として設定可能である．いくつかの閾値を設定し，それぞれの検出結果をもとに評価する．

表 1 CCFinder と SourcererCC の各閾値で生成した部品群の数と部品群中の部品数

CCFinder	種類	部品数	SourcereCC	種類	部品数
50	0	0	0.8	1	4
40	0	0	0.7	2	6
30	1	2	0.6	5	12
25	3	8	0.5	7	17
20	5	12	0.4	7	31
			0.3	1	43

4 評価実験

4.1 JavaPlot に対する適用結果

JavaPlot は GNUPlot を利用して，Java プログラム上でグラフを生成するためのライブラリで，51 のソースファイル (部品) で構成されている．[5] の手法で分類したときの樹形図上での分類を図 1 に示す．樹形図に沿って部品群内の部品はすべて関連性をもつように部品群の範囲を決定したところ，10 種類 32 個の部品が抽出された．

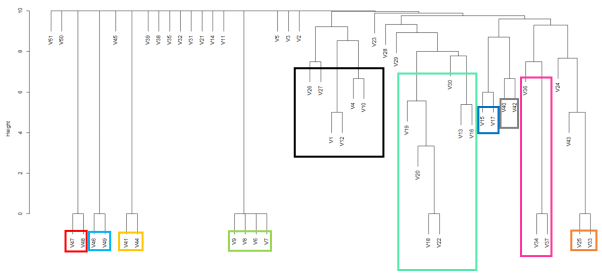

図 1 [5] の手法で分類した樹形図

表 1 は 2 つのコードクローン分析ツールで検出されたコードクローンをもとに作成した類似部品群数とその中の部品数を示す．上から厳密な条件でのコードクローン検出を行う閾値となっており，下にいくほど検出量は増えるが，関連性の強さは低下している．CCFinder におけるクローン検出結果をもとに生成した類似部品群を図 1 の樹形図で表現した結果を図 2 と図 3 に示す．図 2 は一致するトークン列の長さを 30 にしたときの結果である．類似部品の情報は 1 組だけであった．図 3 は一致するトークン列の長さを 20 にしたときの結果である．部品群数が 5 種類で一番多い結果となっているが，それでも分類対象となった部品数は 12 個であった．CCFinder を用いて得られた部品群としては，抽出した結果の精度と抽出した部品群の均衡が

取れた結果となった．しかし，これらのコードクローンによって抽出された部品群では，大まかな共通点はあるが細かな共通点は少なく，[5] の手法の樹形図では得られていた細かく分類された情報が失われている．

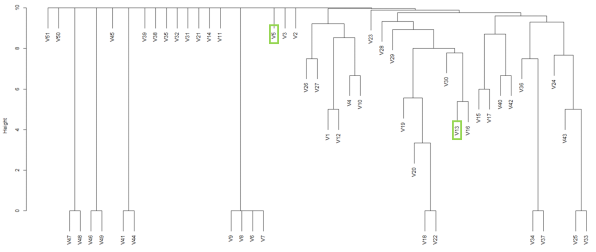

図 2　CCFinder(閾値 30) の部品群を反映した樹形図

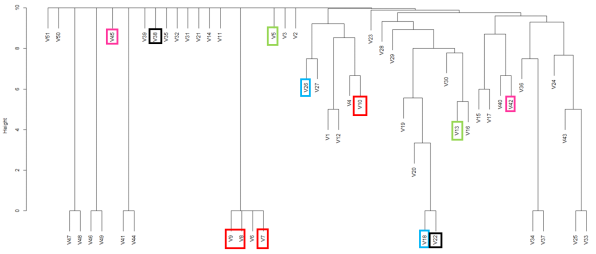

図 3　CCFinder(閾値 20) の部品群を反映した樹形図

SourcererCC におけるクローン検出結果をもとに生成した類似部品群を図 1 の樹形図で表現した結果を図 4 と図 5 に示す．図 4 は一致度の閾値を 0.8 にしたときの結果である．関連の強い部品を部品群として抽出できているが，部品数が 4 個のみでとても少なかった．得られた部品群中の部品同士は，同一の機能を実装するための部品でありプログラムの構成が似ている部品同士であった．図 5 は一致度の閾値を 0.4 にしたときの結果である．部品群数が 7 種類で一番多く，CCFinder の場合とは異なる部品群が抽出されているが，それでも分類対象になった部品数は全体の約 6 割にとどまった．閾値を 0.3 にすると 1 つのグループに集約されてしまい，分類結果としての精度は著しく低下した．CCFinder における実験結果と同様に，[5] の手法の樹形図では得られていた細かく分類された情報が失われている．

継承 (実装) 元と先の関係を表現した有向グラフを図 6 に示す．実線が継承元から継承先への有向辺を表しており，点線がインタフェースから実装先への有向辺を表している．図 6 のように 7 種類 41 個の部品が抽出された．図 6 の部品群を図 1 の樹形図上に表現した結果を図 7 に示す．一つの部品群内において関連がある部品同士も存在するが，機能的関連性が低い部品も多く存在している．例えば図上の紫色の部品群に着目すると，紫色で囲まれた部品群は他の色に比べて部品数が多い．部品の多くは JavaPlot の端末に関する部品であったが，関係のない部品も含まれ

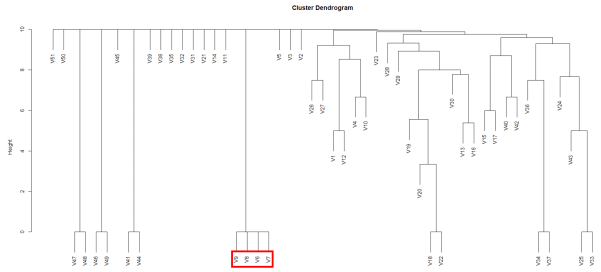

図 4　SourcererCC(閾値 0.8) の部品群を反映した樹形図

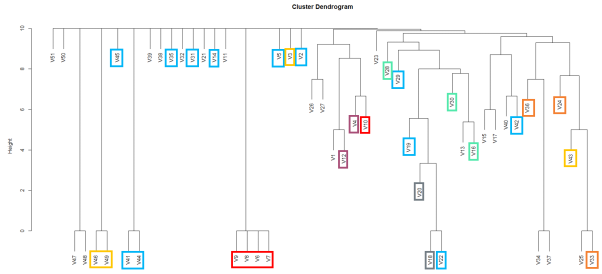

図 5　SourcererCC(閾値 0.4) の部品群を反映した樹形図

ていた．他の部品群中の部品と機能的に類似している部品も顕在していることから分類手法としての精度は低い．また，[5] の手法での分類結果を有向グラフに表現したものを図 8 に示す．黄緑色の部品群のように同一の部品群同士で有向辺が結ばれている組み合わせや，v33 からピンクの部品群への有向辺が複数存在しているように，機能的関連がある部品同士が有向辺で繋がっている部分も一部存在しているが，多くの部品群が様々な場所に分布されており，機能的に分類することが困難である．

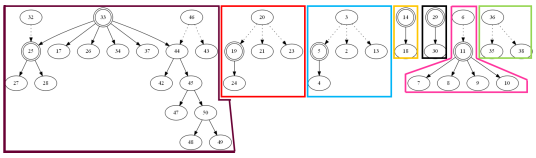

図 6　継承 (実装) 元と先の関係を表現した有向グラフ

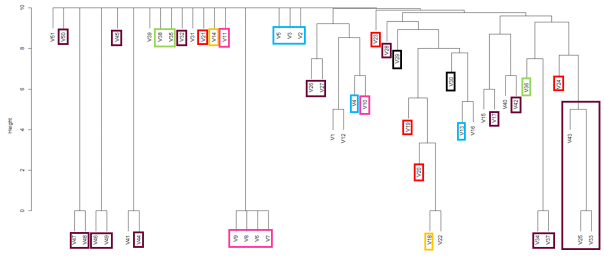

図 7　図 6 の部品群を図 1 の樹形図上に表現した結果

パッケージを一つの部品群とみなした上で，[5] の手法によって得られた部品群がどのように配置されたか図 9 に表

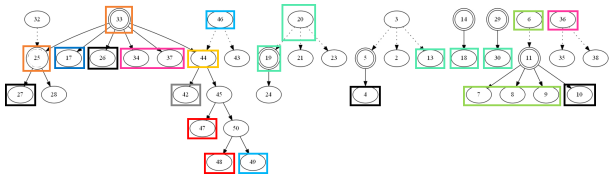

図 8　図 1 の部品群を図 6 のグラフ上で表現した結果

現する．[5] の手法によって得られた部品群の多くはパッケージによる分類に沿ったもので，開発者によるパッケージ分類結果を反映できている．ただし，パッケージ内のすべての部品が一つの部品群となっているわけではなく，パッケージを横断した部品群も多くみられる．

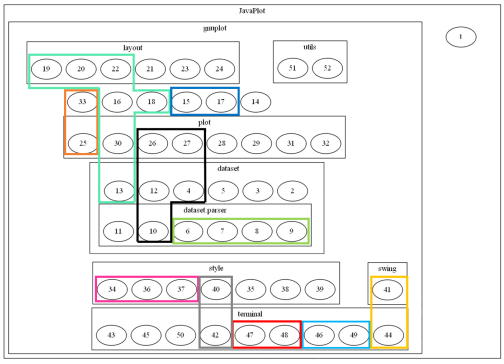

図 9　図 1 の部品群をパッケージ分類上で表した図

5　考察

コードクローン分析ツールから得られた類似情報から類似部品群を抽出した場合，コードクローン検出の精度を上げると正しいグループが得られるが，対象部品の数が少なく，ソフトウェア全体の部品から情報を入手できているわけではない．一方で，精度を下げてたくさんの部品を対象にしても，グループ分けの精度が著しく低下し適切な分類にならなかった．適切な閾値で行った場合もそれらの中間の結果になり，ソフトウェア全体を適切に分類するといった [5] の手法で重視している点を満たすことは難しかった．

実装 (継承) 元と先の関係から得られた類似部品群を抽出した場合，部品群内に機能的関連性が低い部品が含まれることがある．共通の概念や処理の観点から分類がある程度できており，対象部品数は多いがグループ分けの精度としては低く，適切な分類にはならなかった．

所属パッケージから部品群を分類すると，システム全体の部品をある程度は分類できており，[5] の手法によって得られた部品群も所属パッケージに従っている事例も多くみられた．ただし，パッケージ内の全ての部品が一つに分類されているわけではなく，[5] の手法による分類結果はさらなるサブパッケージ化において利用できるかもしれない．また，システムを横断した部品群もいくつか見られ，これらは横断的関心事としてアスペクト化などを考慮すべき対象かもしれない．今後，このような事例を細かく調査することが必要であると考えられる．

6　まとめ

本研究では [5] の手法を行って得られた部品群について，他の既存の手法を用いることで得た部品群と比較を行った．結果として，他の既存の手法で得た部品群は，ソフトウェア全体から類似部品を適切に抽出するという目的にはそぐわないものが多く，[5] の手法で重視していた点を実現することが難しいことが分かった．今後はさらなる分析を行うことで，他の支援手法に適用可能かの調査，他の観点から得た類似部品群との比較，他のソフトウェアでも同様の結果が得られるかについての調査が必要となる．

参考文献

[1] T. Kamiya, S. Kusumoto, and K. Inoue : “CCFinder: Amultilinguistic 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. 44, 2011.

[3] H.Zhong, T.Xie, L.Zhang, J.Pei, and H.Mei, ”Mapo: Mining and recommending api usage patterns,”in proceedings of the 23rd European Conference on Object-Oriented Programming (ECOOP 2009), 2009, pp.318-343.

[4] Z.Li and Y.Zhou, “Pr-miner: automatically extracting implicit programming rules and detecting violations in large software code,” in proceedings of the 10th European software engineering conference,2005,pp.306-315.

[5] Reishi Yokomori, Norihiro Yoshida, Masami Noro, Katsuro Inoue: ”Use-Relationship Based Classification for Software Components”, Proceedings of the 6th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2018), pp.59-66, 2018.

[6] 橋本敬太，川瀬史也 :“利用部品の共通性に基づくソフトウェア部品分類手法の評価-共通して利用している部品の観点からの評価-”，南山大学 理工学部 2018 年度卒業論文，2019.

[7] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, Cristina V. Lopes : “SourcererCC: Scaling Code Clone Detection to Big-Code” IEEE International Conference on Software Engineering, 38th, pp. 1157-1168, 2016.