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―

2009SE007 安藤正憲　2011SE052 堀江大河　2011SE065 井田裕之

指導教員：横森励士

1　はじめに

近年ソフトウェアの規模が, 大規模化してきており，それにともないソフトウェア部品の数も増大化し，内部の関係も複雑化している．このような環境下では，ソフトウェアをモデル化して，メトリクスに基づいてある特性を持つ部品を抽出するという方法が有効であると考える．千賀が行った先行研究では，コンポーネントランクの計算にコードクローンを反映させることで，コードクローンから共通して利用されやすい部品を選出する手法 [1] を提案した．実際のオープンソースプロジェクトにおいて，想定した状況が検出できる事例があることを示したが，常に有効であるかどうかの検証は行っていない．

本研究では，先に挙げた千賀が提案した手法の有効性を検証するために，実際のオープンソースプロジェクトのソースコードに適用することで評価実験を行う．プロジェクトを部品グラフとしてモデル化し，コードクローン関係を持つ部品同士を結合する．その結合された部品群から共通して利用されている部品を抽出する．抽出した部品が提案手法において想定されるような形で利用されているかを主な評価基準とする．結合された部品群から共通して利用されている部品の利用方法が，結合元の部品のコードクローン発生の原因となっている割合を調査する．次に，結合された部品群から共通して利用されている部品の具体的利用法について調査する．さらに，部品評価値の減少率の大きい部品を選出し，その部品がコードクローン発生の原因となっている割合を調査する．これらの評価実験を通じて，千賀が提案した手法の有効性を確認することで，コードクローンから共通して利用されやすい部品をさらに利用する際に，助言を出すなどの利用方法が実現可能かどうかについて検討する．

2　背景技術

2.1　部品グラフ

一般に，ソフトウェア部品とはモジュールや関数，クラスなどのソフトウェアの構成要素を指す．以降，ソフトウェア部品を単に部品と呼ぶ．ソフトウェアは構成要素の部品間で相互に属性やふるまいを利用し合うことで一つの機能を提供する．本研究では，ある部品がある部品を利用するとき，この部品間に利用関係が存在すると考え，部品グラフとしてモデル化する．部品を頂点，利用関係を有向辺で表したグラフを部品グラフと呼ぶ．以降，V を部品 (頂点) の集合，E を有向辺の集合として，部品グラフを $G = (V, E)$ と表現する．

2.2　コンポーネントランク

コンポーネントランク [2] は，利用頻度に基づいて部品グラフからそのソフトウェアにおける各部品の重要度の評価値を算出する手法である．コンポーネントランクでは，部品グラフ $G = (V, E)$ 上の個々の辺および頂点に対して重みを計算し，対応する頂点の重みを各部品の評価値として，順位に基づいて評価を行う．重みとは，次のように定義される．

頂点の重み

部品グラフ G 上の各頂点 v は $0 \leqq \mathrm{w(v)} \leqq 1$ の重みを持ち，G の頂点の重みの総和は 1 となる．

辺の重み

頂点 v_i から v_j への辺 e_{ij} に関する辺の重み $w'(e_{ij})$ を式 (1) のように定義する．

$$w'(e_{ij}) = d_{ij} \times w(v_i) \tag{1}$$

d_{ij} は配分率と呼び，$0 \leqq d_{ij} \leqq 1$ かつ $\sum_i d_{ij} = 1$ を満たす値とする．原点 v_i から v_j へ利用関係が存在しない場合，$d_{ij} = 0$ とする．この配分率 dij は，次に示す頂点の重みの定義において，有向辺の終点となる頂点の重みの決定に利用される．

頂点の重みの再計算

$IN(v_i)$ を v_i を終点とする有向辺の集合とする．このとき，頂点 v_i が終点となる有向辺 e_{ki} の重みの総和とする．式 (2) で頂点の重みを再計算する．

$$w(v_i) = \sum_{e_{ki} \in IN(v_i)} d_{ki} \times w(v_k) \tag{2}$$

繰り返し計算の手順

1. 評価値の初期値として，各頂点に正の値を与える
2. 値の変化が一定以下になるまで，次の計算を繰り返す
   (a) 辺の重みを現在の頂点の重みから決定する
   (b) 頂点の重みを再計算する

このように各頂点の重みを決定し，対応する部品の評価値として，順位に基づいて評価する．

2.3　コードクローン

コードクローン [3, 4] とは，ソースコード中での類似または一致した部分であり，同一の部品内だけにとどまら

ず，異なる部品間に存在する場合もある．コードクローンは無意味に出現するものではなく，同一処理や似た処理が必要になるなど，開発者の何らかの意図によって作りこまれる場合が多い．コードクローンはソフトウェアの保守工程においてプログラム管理の手間を増大させる可能性を持つので，コードクローンとなる部品を有する部品は，常に着目する必要がある．本研究においては，2 つ以上の部品のソースコードを比較したとき，同じようなトークンが 30 以上連続して並ぶ部分をコードクローンとみなす．

2.4 コードクローンとコンポーネントランクへの関係の反映

千賀は，コードクローン [3，4] を持つ部品同士を部品グラフ上で結合する前後で，コンポーネントランクの評価値が下がりやすいことに着目し，コードクローンから共通して利用される部品を選出する方法を提案した．コードクローンの検出に CCFinder[4] を，クラスの利用関係の検出には Classycle[5] を用いて実際に以下の手順で調査を行っている．

1. プロジェクトのソースコードに対し，Classycle を用いて利用関係を取得し，部品グラフを作成する
2. 1 の部品グラフを基にコンポーネントランクを計算する
3. CCFinder を用いてコードクローン関係を取得し，コードクローンを持つ部品同士を結合する
4. 3 の部品グラフを基にコンポーネントランクを計算する
5. 2，4 のコンポーネントランクを比較し，各部品の評価値の変動率を求める

千賀は，実際のオープンソースプロジェクトにおいて，想定した状況が検出できる事例があることを示したが，提案手法が常に有効であるかどうかの検証は行っていない．より多くの実際のオープンソースプロジェクトに関して適用し，提案手法の有効性を評価することが必要である．

3 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についての研究

3.1 研究の目的

本研究ではコンポーネントランクを用いて，実際のオープンソースプロジェクトを対象にコードクローンと関連する部品を検出する仕組みが，有効かどうか検証することを目的とする．評価においては，選出される部品が先行研究の提案手法の利用方法で想定されるような利用のされ方をしているかを主な評価基準とする．

3.2 評価実験の手順

実際のオープンソースプロジェクトに対し，以下の手順で調査を行う．

1. 実際のオープンソースプロジェクトを収集し，各プロジェクトからソースコードを 1 バージョン入手する
2. CCFinder を用いてコードクローン関係の抽出を行い，Classycle を用いてクラスの利用関係の抽出を行う
3. 2 で得られた結果から部品グラフを構築し，千賀が提案した手法を適用してコードクローン関係を持つ部品同士を結合する
4. 結合されなかった部品を以下の 3 つのグループに分ける

   GroupA　コードクローン関係により結合された部品群内の部品の 2 つ以上から共通して利用される部品．

   GroupB　コードクローン関係により結合された部品群内の部品の高々 1 つから利用される部品

   GroupC　コードクローン関係により結合された部品群から全く利用されない部品
5. 後に述べる評価基準に基づき，評価を行う

   本研究では主に，GroupA の部品に着目する

3.3 コードクローン発生の原因となる部品

3.2 節で挙げた手順により得られたコードクローン関係とクラスの利用関係を基に，GroupA に属する部品がコードクローン発生の原因となっているか調査する．GroupA に属する部品を利用している部品のソースコードを実際に目視で確認した上で，GroupA に属する部品がソースコード内でどのようなふるまいをしているかを判断基準とする．以下の場合の部品を，コードクローン発生の原因である部品とする．

- 検出されたコードクローン内部で，共通して利用され，ふるまいが同じ部品．
- 検出されたコードクローン部分以外において，共通して利用され，ふるまいが同じ部品．

3.4 評価基準

3.2 節で挙げた手順により得られた調査結果を次の評価基準により評価する．

項目 1　GroupA の部品の中で，実際にコードクローン発生の原因となっている部品の割合

3.2 節で定義した GroupA の部品について，その利用元となるクラス群中に存在するコードクローンの中身を調査する．GroupA の各部品が，コードクローン発生の原因となっているかを調査し，発生の原因となる部品の割合を計算する．選出された GroupA の部品数全体に対して，コードクローン発生の原因となっている部品の割合を適合率として利用する．『互いにコードクローンを持つクラス群が共通して利用している部品が，コードクローン発生の原因である．』という仮説の妥当性を評価する．

項目 2　コードクローン関係により結合された部品から利用されている部品が，どのように利用されているか

コードクローン関係により結合されたクラス群の各部品について，その部品が GroupA に属する部品をどう利用しているか調査する．これによって，実際にターゲットとなる部品としてどのようなものがあるか，その傾向を想定することができる．

項目 3　評価値の減少率と適合率の関係を分析

各プロジェクトの GroupA に属する部品を，評価値の減少率が大きい部品から順番に並び替えたときに，上位 20 個の部品について，コードクローン発生の原因となる部品の割合をプロジェクト毎にまとめグラフで表す．これによって，評価値の減少率の大小を部品の絞込みに活用できるかを確認する．

4　変化の影響を受けた部品についての分析

調査対象として，43 種類のオープンソースプロジェクトをダウンロードした．これらのオープンソースプロジェクトに対して適用した解析結果を項目別で示す．

4.1　項目 1 の評価実験結果

項目 1 の調査対象として，総数 620 個の GroupA に属する部品を得た．これらの部品を，3.3 節を基にコードクローン発生の原因となっている部品と，コードクローン発生の原因となっていない部品に分類する．そしてコードクローン発生の原因となっている部品を○と表し，コードクローン発生の原因となっていない部品を × と表す．部品全体に対して○の数の割合を適合率として，それぞれの数の内訳を次の表 1 で示す．この表からは，GroupA に属する部品の約半数が，コードクローン発生の原因となっていることが分かる．

表 1　コードクローン発生原因の割合

部品総数	○の数	× の数	適合率
620 個	339 個	281 個	55%

さらに，プロジェクト毎の適合率を表したグラフを次の図 1 で示す．ただし各プロジェクトの GroupA に属する部品が 2 個以下の場合，適合率が極端な値になりやすい為，そのオープンソースプロジェクトは対象から除外している．この図からは，適合率が 50%～59%，60%～69% となるプロジェクトが多く見られる．また，適合率が 100% となるプロジェクトも多く見られる．

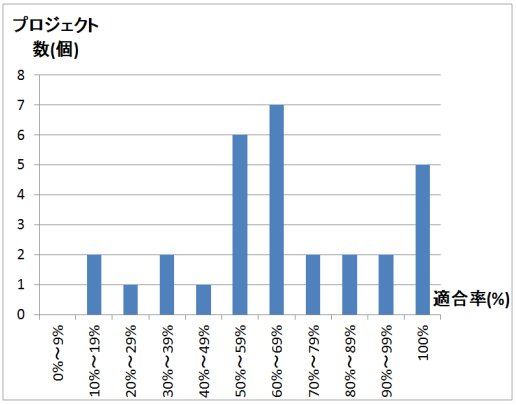

図 1　プロジェクト毎の適合率

また，プロジェクトの規模の大きさと適合率の関係を散布図として表したグラフを次の図 2 で示す．この図 2 においても図 1 と同様に，各プロジェクトの GroupA に属する部品が 2 個以下の場合，そのオープンソースプロジェクトは対象から除外している．この図からは，プロジェクトの規模が大きくなるにつれて，適合率は減少する傾向が読み取れる．

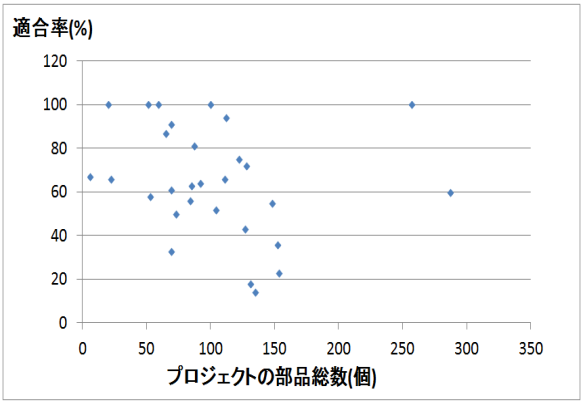

図 2　プロジェクトの規模と適合率の関係

4.2　項目 2 の評価実験結果

項目 2 の調査対象として，項目 1 と同様に総数 620 個の GroupA に属する部品を得た．これらの部品を○と × に分類した後，実際のソースコードを目視で確認をして，各部品の具体的な利用方法を調査した．○となった 339 個の部品について，どのような形でコードクローンから利用されているか調査した．そのうちの 195 個については，コードクローンとなる部分において，メソッド呼び出しや，インスタンスの生成などの処理で利用されていることを確認した．残りの 144 個については，その部品を継承する形でクラス群中のクラスが作成されており，その中でコードクローンとなる共通の記述があることで，コードクローンと関係があると判断した．その他に例外処理として，継承と同様にクラス群中のクラスが作成されており，共通の記述があることで，コードクローンと関係があると判断した．

× の部品については，ソースコードを見比べたとき，それぞれの部品において利用方法が異なっているので，コードクローンと関係があるとみなせない部品であった．

4.3 項目 3 の評価実験結果

項目 3 の調査項目として，GroupA に属する部品が 20 個以上あるプロジェクトを選出する．3 種類のプロジェクトを対象に，各プロジェクトの GroupA に属する部品の中から，評価値の減少率の大きい部品を 20 個ずつ選出する．その部品を評価値の減少率が大きい部品から順番に並び替えたときに，上位 20 個の部品について，コードクローン発生の原因となっている部品の割合をそれぞれ調査した．評価値の減少率が 1 番大きい部品の適合率から上位 20 個の部品における部品の適合率まで調査の範囲を広げたとき，各プロジェクトの適合率の推移を図 3 に示す．この図から，減少率の大きい部品が必ずしも，コードクローン発生の原因となっているとは限らないが，全体的な傾向としてグラフが右肩下がりであることから，減少率の大きい部品はある程度コードクローン発生の原因となっていると言える．

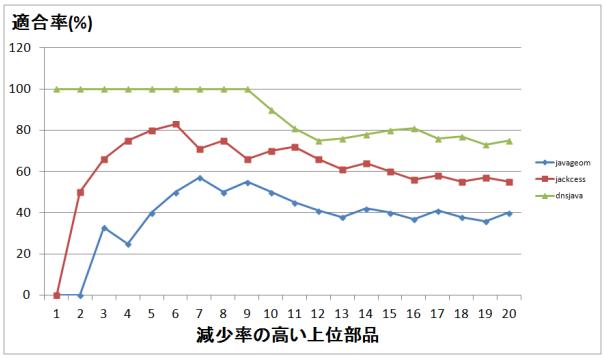

図 3 評価値の減少率の大きさと適合率の関係

5 考察

4 章で得た実験結果に対して，項目毎に考察をする．

項目 1 では，『互いにコードクローンを持つクラス群が共通して利用している部品が，コードクローン発生の原因である.』という仮説の有効性を検証するために，GroupA に属する部品全体に対して，コードクローンの発生の原因となっている部品の割合を適合率として調査した．実際に仮説としては成立するかどうか疑わしい仮説であったが，55% 程の部品がコードクローン発生の原因となることが確認され，想定した値と比較すると大きな値となった．この方法を用いて，支援のためのツールを作成できる可能性はあるが，精度の向上をはかる必要がある．

項目 2 では，コードクローン関係により結合されたクラス群について，クローン内の処理の一部で該当クラスを利用していることがわかった．精度向上のための手法としてクローン内部の記述を参考にすると考えたとき，クローン中でどのように利用されているかの典型的な例を参考にして絞込みを行う方法が考えられる．今回，検出できた例を基に利用パターンを想定することができると考えられる．特に特定のクラスを継承した部品間の共通処理によってコードクローンが発生している場合は，そのクラスを継承するとコードクローンが発生しやすいという助言を出す方法が有効であると考えられる．

項目 3 では，GroupA に属する部品が 20 個以上あるプロジェクトを対象に，その GroupA の部品のうち評価値の減少率が大きい部品上位 20 個ずつ選出した．評価値の減少率が大きい部品から順番に並び替えたときに，上位 20 個の部品について，コードクローン発生の原因となっている部品の割合をそれぞれ調査した．評価値の減少率が 1 番大きい部品の適合率から上位 20 個の部品における部品の適合率まで，調査の範囲を広げたときの各プロジェクトの適合率の推移を表すグラフから，減少率の高い部品が必ずしもコードクローン発生の原因となっているとは限らないが，全体的な傾向としてグラフが右肩下がりであることから，減少率の高い部品はある程度コードクローン発生の原因となっていることが言える．

6 まとめ

本研究では，千賀が提案した手法の有効性を検証するために，実際のオープンソースプロジェクトに適用し，結合される部品群から共通して利用される部品が，実際のコードクローンの発生に関わっているかどうかを確認した．結果としては，想定より多くの部品がコードクローン発生の原因となっており，ソフトウェアツールの支援としてある程度利用できると考えられる．今後の課題として，パッケージ間の距離で結合を制限する方法が精度の向上に役立つと考えられるが，制限を行ったときに元の GroupA に含まれる○，× の部品がそれぞれどのような影響を受けるかについての調査を行いたいと考える．

参考文献

[1] 千賀英佑，” コードクローンを利用したソフトウェア部品の評価手法についての考察，” 南山大学大学院数理情報研究科 2013 年度修士論文，2014.

[2] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto, ”Ranking Significance of Software Components Based on Use Relations,” Transaction on Software Engineering, vol.31, no.3, pp.213-225, 2005.

[3] Ira D.Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant’ Anna, and lorraine Bier, ”Clone Detection Using Abstract Syntax Trees,” International Conference on Software Maintenance ’98, pp.368-377, 1998.

[4] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, ”CCFinder: A Multi-Linguistic Tokenbased Code Clone Detection System for Large Scale Source Code,” Transaction on Software Engineering, vol.28, no.7, pp.654-670, 2002.

[5] ”Classycle: Analysing Tools for Java Class and Package Dependecies,” http://classycle.sourceforge.net/.