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

M2012MM038 千賀英佑
指導教員：横森励士

1 はじめに

ソフトウェアの保守工程では，修正保守，適応保守，改善保守，予防保守などの活動において，既存のコードを理解しながら訂正や改良をおこなう [1]．これらの作業を通じて，コードクローンと呼ばれるコードの類似部分が作り込まれることがある．コードクローンはソフトウェアの保守性を低下させるので，取り除いたり作りこまないことが一般的に望ましいとされる [3, 4]．一方で，部品間の関係をモデル化し，部品の評価をおこなう手法として，コンポーネントランク [2] がある．コンポーネントランクは，ソフトウェア部品の利用関係を表すグラフから各部品の評価値を算出する．ただし，利用関係以外の関係は考慮しておらず，他の関係は考慮されていない．

本研究の目的は，コードクローンの関係を部品グラフに取り込むことで，コンポーネントランクを用いてコードクローンと関連する部品を検出する仕組みを作ることである．コンポーネントランクを拡張し，コードクローンの関係を評価値の計算に組み入れる手法を提案する．このとき評価値がどのような変化を起こすのか考察した上で，小さな仮想ソフトウェアに対して適用し，考察の妥当性を確認する．その後，実際のソフトウェアにも想定した評価値の変化が現れる事例があるか調査する．これらの実験・調査の結果の一般性を確認するためにはどのような分析をおこなう必要があるか，この手法をどのようにコードクローンの除去や発生の防止に利用できるか考察をおこなう．これらの考察を通じて，提案手法をソフトウェアの保守作業に役立てる方法について議論する．

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振る舞いを利用し合うことで一つの機能を提供する．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．

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，部品グラフから各部品のそのソフトウェアにおける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が各部品の評価値となる．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 $0 \leqq w(v) \leqq 1$ の重みを持ち，G の頂点の重みの総和は 1 とする．

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 $\Sigma_i d_{ij} = 1$ を満たす値とする．頂点 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 d_{ij} 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．この時，頂点 v_i の重みは 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する．実際には [2] で示すように，疑似的な辺を導入して計算の収束を保証している．

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．あるコードクローンを修正しようとする場合，開発者はそれと対応するすべてのコードクローンについて，同様の修正をおこなうか検討する必要があるからである．

3 本研究の概要

3.1 本研究の目的

本研究の目的は，コードクローンの関係を部品グラフ上に取り込むことで，コンポーネントランクを用いてコードクローンと関連する部品を検出することである．

3.2 コードクローン関係の考慮

コードクローンの関係に着目してコンポーネントランクの拡張をおこなおうとするとき，その拡張方法にはいくつかの方法が考えられるが，本研究ではコンポーネントランクの拡張方法としてコードクローンの関係を持つ頂点の統合を採用する．頂点の統合とは，コードクローンの関係を持つ部品を類似した部品と解釈し，1 つの頂点に統合してグラフ上に表現する方法である．この方法によって頂点を統合したグラフの例を図 1 に示す．

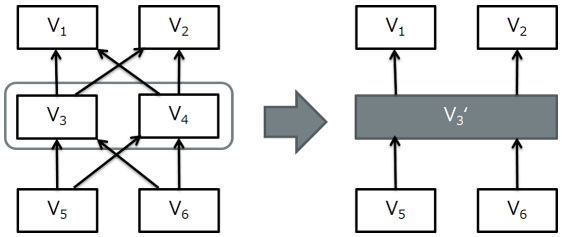

図 1 頂点を統合したグラフの例

図 1 の頂点 v_3 と v_4 の部品にコードクローンの存在が確認され，両頂点を統合したとする．頂点を統合するとは，部品グラフ上の頂点の集合 V から，v_3 と v_4 を消去し，新たに $v_{3'}$ を追加することを指す．$v_{3'}$ を終点とする有向辺の集合 $IN(v_{3'})$ は，$IN(v_3)$ と $IN(v_4)$ の和集合とし，$v_{3'}$ を始点とする有向辺の集合 $OUT(v_{3'})$ は，$OUT(v_3)$ と $OUT(v_4)$ の和集合とする．このとき頂点の統合前と統合後で各辺と頂点の重みが変化する．v_1 を終点とする辺が頂点の統合前には辺 e_{31} と e_{41} の 2 本であったのに対し，頂点の統合後には $e_{3'1}$ の 1 本と減少している．これにより v_1 の頂点の重み $w(v_1)$ は，頂点の統合前よりも減少すると考えられる．これは v_2 の頂点の重み $w(v_2)$ についても同様の影響が現れる．よって，コードクローンの関係を持つ部品同士を統合し 1 つの頂点とみなしたとき，コードクローンの関係を持つ部品が利用している部品の評価値は減少し，それ以外の部品の評価値が相対的に増加すると推測できる．

3.3 アプローチ

3.2 節の考察も踏まえ，「統合する頂点がある頂点に対する共通した出力辺を持つとき，統合する頂点のコードクローンの中で，ある頂点を同様の利用の仕方をしている」との仮説が成り立つとすると，頂点統合後のグラフはコードクローンの関係がまとめられたグラフと見ることができる．このとき頂点統合後のグラフを頂点統合前のグラフを比較すると，コードクローンを持つ頂点が利用している頂点が影響を受け，評価値は減少すると推測できる．提案手法では，「コードクローンを持つ部品を統合したときに評価値が減少する部品は，コードクローンから同じように利用されている」と推測し，評価値が減少する部品を検出する．実際にこれらの推測が成り立つか確認するために，次の実験と調査をおこなう．

1. 小さな仮想的ソフトウェアに対してコードクローンを考慮した評価値計算を適用し，評価値の変化を観測する．これにより部品グラフ上で部品を統合したとき，統合される部品から共通して利用される部品の評価値が大きく減少することを確認し，部品の統合という手法が統合により入力辺がまとめられる部品の検出に利用可能であることを確認する．
2. 実際のソフトウェアにおいて本手法を適用し，評価値が減少する部品がコードクローンを持つ部品から利用される部品であるという事例を調査する．これにより，コードクローン内に評価値が減少した部品を利用する記述があることを確認し，提案手法を用いてコードクローンから同じように利用する部品の検出に利用できる可能性があることを確認する．

3.4 評価値の変動率

評価値の増減の判定には，統合前後の各頂点の重みの変動率を用いる．部品数により 1 つの頂点あたりの重みが異なるので，変動率にはグラフにおける頂点の総数を考慮する．ある頂点 v_i の頂点統合前のグラフでの評価値を $w(v_i)$，頂点統合後のグラフでの評価値を $w(v_i')$，また，頂点統合前のグラフにおける頂点の総数を $|V|$，頂点統合後のグラフにおける頂点の総数を $|V'|$ とする．このとき頂点 v_i の評価値の変動率を式 (3) で定義する．

$$\frac{w(v_i') \times |V'|}{w(v_i) \times |V|} \times 100 \tag{3}$$

4 ツールの実装

コードクローンの関係を計算に組み入れたコンポーネントランク計算ツールの実装をおこなう．本ツールは，Java プログラムを対象とし，各クラスをソフトウェア部品とみなし，コンポーネントランクの計算をおこなう．図 2 に本ツールの構成を示す．

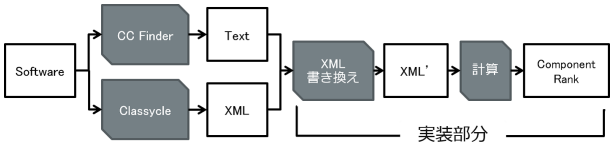

図 2 ツールの構成

本ツールは，コードクローンの検出に CCFinder[4] を用い，クラスの利用関係の検出には Classycle[5] を用いる．本研究では，これらの検出結果からコードクローンを考慮した評価値計算をおこなう部分を PHP で実装した．ツールが評価値を計算する具体的な手順を次に示す．

1. Java プログラムを CCFinder と Classycle に入力し，テキスト形式のデータと XML 形式のデータを得る

2. CCFinder の結果に基づき，Classycle が出力した XML データを操作し，コードクローンが検出された部品を統合した XML データに書き換える
3. 書き換えられた XML データを基に，評価値の計算をおこなう

5 実験

5.1 目的

3.2 節では，部品グラフ上で部品の統合がおこなわれたとき，統合される部品から共通して利用される部品は，片方から利用される部品と比べ，評価値が大きく減少すると推測した．この推測が正しいことを確認するために，実際のソフトウェアを想定した小さな部品群に対して，頂点を統合しない従来のコンポーネントランクと頂点を統合した後のコンポーネントランクを計算し，評価値の比較をおこなう．

5.2 実験対象と実験内容

実験の対象として，図 3 のような 11 個の頂点と 29 個の辺で構成される部品グラフを用いる．この部品グラフは，v_1 から v_3 で表されるデータ構造を v_4 から v_7 のクラスが操作する．それらをもとに v_8 から v_{10} であるまとまった機能を実現し，v_{11} がアプリケーション本体であるような小規模ソフトウェアを想定している．

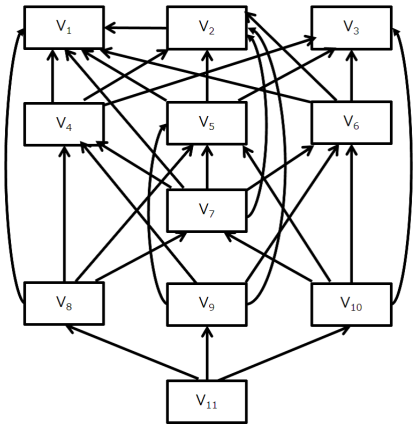

図 3 実験に使う部品グラフ

実験においては，v_1 から v_{11} のうち 2 つを選ぶ，${}_{11}C_2$ 通りの組み合わせについて，選んだ 2 つの部品を統合した際，統合した頂点以外の評価値の変動率を調査する．それらの頂点を次の 3 つのグループに分け，平均値に差があることを示す．

Group 1 統合する頂点の両方から利用されている頂点
Group 2 統合する頂点の片方から利用されている頂点
Group 3 どちらからも利用されていない頂点

5.3 結果

実験に基づいてグループ分けをおこない，該当するグループの評価値の変動率，平均，標準偏差を示した結果を表 1 に示す．Group 1 の頂点の変動率は 66.2%の例から 93.5%の例とばらつきが小さく，変動率の平均は Group 2 と Group 3 に比べて低くなった．Group 2 の頂点の変動率は 65.9%の例から 170.5%の例があり，Group 3 の頂点の変動率も 47.2%の例から 138.6%の例があり，ともにばらつきが大きかったが，変動率の平均は Group 3 の方が低くなった．

表 1 変動率の平均と標準偏差

	標本数	平均	標準偏差
Group 1	39	85.0%	6.8
Group 2	180	104.9%	15.9
Group 3	275	91.5%	15.7

これらの平均値に有意な差があるかどうか，Group 1 と Group 2 についてと Group 1 と Group 3 について，それぞれ Welch の t 検定をおこなった．これら 2 つの検定において，効果量 0.5，有意水準 5%，検定力 0.8 となるために必要な標本数は，標本数の比が 1:4 程度である Group 1 と Group 2 の場合はそれぞれ 31 と 125，標本数の比が 1:7 程度である Group 1 と Group 3 の場合はそれぞれ 29 と 199 であり，今回の実験は条件を満たしている．

実際に検定をおこなったところ，次のような結果を得た．

Group 1 と Group 2 の平均値の差の検定結果

$$t = 12.24, p < 0.05, d = 1.62, 1 - \beta = 1.00$$

Group 1 と Group 3 の平均値の差の検定結果

$$t = 4.81, p < 0.05, d = 0.53, 1 - \beta = 0.93$$

これらの結果はいずれも有意水準 5%の条件下で平均値に差があることを示しており，「統合される部品から共通して利用される部品は，片方のみから利用される部品，あるいはその他の部品と比べ，評価値が大きく減少する」という推測がこの部品グラフ上では成り立つことを示している．

6 事例の調査

実際のソフトウェアにおいて，評価値が減少する部品がコードクローンを持つ部品から利用される部品である事例を調査した．その結果，評価値が減少した部品を利用する部品にはコードクローンが存在し，そのコードクローン内に評価値が減少した部品を利用する記述が存在したという事例を次のように 3 件は確認した．これにより提案手法がコードクローンから同じように利用する部品の検出に有効な場合があることを確認した．

事例 1

Mantissa 7.2 において，NonNullRange クラスの評価値が 34.7%減少した．部品統合により，NonNullRange クラスへの入力辺は 6 から 3 へと減少した．このクラスを利用するクラスである DiagonalMatrix クラスと LowerTriangularMatrix クラスに共通のコードクローンが存在し，コードクローン内に NonNullRange オブジェクトを生成する同様の記述が確認された．

事例 2

JGraphX 2.4.0.2 において，mxGraphMlUtils クラスの評価値が 43.8%減少した．部品統合により，mxGraphMlUtils クラスへの入力辺は 8 から 4 へと減少した．このクラスを利用するクラスである mxGraphMlEdge クラス，mxGraphMlNode クラス，mxGraphMlPort クラスに共通のコードクローンが存在し，コードクローン内に mxGraphMlUtils クラスのメソッドを呼び出す同様の記述が確認された．

事例 3

jlGui 3.1 において，TagInfo クラスの評価値が 65.8%減少した．部品統合により，TagInfo クラスへの入力辺は 7 から 4 へと減少した．このクラスを利用するクラスである MpegInfo クラスと OggVorbisInfo クラスの間に，複数のコードクローンが存在し，MpegInfo クラスと OggVorbisInfo クラスは共に，TagInfo クラスをインタフェースとして実装していた．

7 考察

7.1 一般性の検証

小さな仮想ソフトウェアを対象とした場合に，部品グラフ上で部品の統合がおこなわれたとき，統合される部品から共通して利用される部品は，片方から利用される部品と比べて評価値が大きく減少することを確かめた．また，事例の調査では，評価値が減少する部品がコードクローンを持つ部品から利用される部品となっているソフトウェアの事例があることを確かめた．今後，提案した手法が一般的に有効であることを示すために次のような分析をおこなう必要があると考える．

- 部品の統合という手法がコンポーネントランクにおける評価値の減少につながるのか，より多くのプロジェクトについて調査する
- 実際のソフトウェアにおいて，評価値が減少した部品が，コードクローンを持つ部品群から同じように利用されている割合がどれくらいか，より多くのプロジェクトについて調査する

これらの実験をおこなうための予備実験として，標本の大きさが大ならば標本平均の確率分布が正規分布に近づくという中心極限定理を利用し，標本平均の収束の度合いから必要なサンプル数を見積もる必要がある．その後，それぞれのプロジェクト毎に適用をおこない，プロジェクト毎の結果を取得し，評価をおこなう．

実際のプロジェクトに対して適用をおこなう場合，統合前の部品に対する入力辺の数と比べ，統合後の部品に対する入力辺の数がどれだけ減るかによって，評価値の減少の大小が異なる可能性があることから，辺の数の減少度が評価値の減少度にどのような影響を与えるかについても分析する．

7.2 結果の利用方法

提案手法により，評価値が減少した部品をコードクローンの検出や除去に役立てることを考えた場合，これには次のような利用方法が考えられる．

- 評価値が減少した部品を新しい部品が利用しようとするとき，コードクローンを作り込む可能性を警告したり，類似した利用の仕方をしている部品が既に存在する可能性を通知する
- 部品を整理するとき，評価値が減少した部品を利用する部品群について，類似した機能を持つ部品が含まれる可能性を示し，統合したときに効果がありそうな部品を調べる

提案手法をこのように利用することで，コードクローンの発生を抑制したり，リファクタリングなどにおいて整理の候補を提示することができ，コードの訂正や改良に役立てることができる．これはソフトウェアの保守作業の支援につながるものであると考える．

8 おわりに

本研究は，コンポーネントランクを用いてコードクローンと関連する部品を検出する仕組みを作ることを目的とし，コードクローンの関係を評価値の計算に組み入れる手法を提案した．このとき評価値がどのような変化を起こすのか考察した上で，小さな仮想ソフトウェアにおいて考察の妥当性を確かめ，実際のソフトウェアにおける事例を調査した．考察として，手法の有効性の一般性を示すためにはどのような分析をする必要があるか，この手法をどのようにコードクローンの除去や発生の防止に利用できるか考察をおこなった．今後，提案手法を用いたリファクタリングツールなどを実現し，コードの訂正や改良に利用することで，ソフトウェア保守作業の支援が期待できる．

参考文献

[1] Shari Lawrence Pfleeger, “Software Engineering: Theory and Practice, ” Pearson PLC, 2001.

[2] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto, “Ranking Significance of Software Components Based on Use Relations, ” Transactions 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 Token-based Code Clone Detection System for Large Scale Source Code, ” IEEE Trans. Software Engineering, vol. 28, no. 7, pp. 654-670, 2002.

[5] Franz-Josef Elmer, “Classycle: Analysing Tools for Java Class and Package Dependencies, ” http://classycle.sourceforge.net/