コードクローンの複雑度メトリクスを用いた開発者の特徴分析

東　　誠[†]　肥後芳樹[†]　早瀬康裕[†]
松下　誠[†]　井上克郎[†]

ソフトウェアシステム内に存在する重複したコード片であるコードクローンは，単に取り除くのではなく適切に管理する必要がある．本研究では，コードクローンの管理を目的として，コードクローンを編集した開発者に着目したコードクローンの編集傾向分析を行う．具体的には，リポジトリに保存されたソフトウェア編集履歴から，コードクローンに対する変更を取り出し，そのコードクローンに対するメトリクス値の変化を計算する．そして，開発者によってメトリクス値の変化量に差があるかを分析する．提案する調査方法を用いて，実際のソフトウェア開発履歴を分析した結果，メトリクスの変化は開発者ごとに差があることが分かった．

Extracting Developers' Characteristics from Complexity Metric Values on Code Clones

MAKOTO HIGASHI,[†] YOSHIKI HIGO,[†] YASUHIRO HAYASE,[†]
MAKOTO MATSUSHITA[†] and KATSURO INOUE[†]

A duplicated code fragment in a software system is called code clone, or simply clone. Clones degrade software maintainability, but it is more practical to manage clones than to remove all of them. This paper proposes a method to analyze developers' tendencies to edit clones. At first, the snapshots of the source code of the target software system are checked out from its software repository. Then, clone metric values for each snapshot are computed. Next, differences of metric values between adjacent snapshots are calculated. Finally, differences of metric values are grouped by developers and the developers are assessed statistically. Applying the method to an open software project shows that developers edit clones in different manner.

1. はじめに

ソフトウェアシステムの保守性を悪化させる問題のつとして，ソフトウェアシステム内に存在する重複したコード片（コードクローン）がある[9]．例えば，あるコード片に欠陥が存在する場合，そのコード片とコードクローンになっているコード片にも同様の修正を検討する必要があり，修正の労力を増大させる[3]．

しかし，すべてのコードクローンを除去することが効果的なわけではない[4]．このため，すべてのコードクローンを取り除くのではなく，コードクローンを適切に管理することが必要であるとされている[4),6)]．

そこで本研究では，コードクローン管理のための基礎として，各開発者の編集時に他の開発者に比べてコードクローンの複雑度に表れる特徴（以下，**編集傾向**とする）を調査する．各開発者は様々な役割や目的に従って編集するため，それぞれの編集傾向は異なると考えられる．また，編集傾向によって各コードクローンの複雑度の変化を予測できると考えられる．よって，編集傾向を用いて，すべてのコードクローンの中から将来複雑になって除去し難くなるものとそうでないものを区別できると考えられる．

具体的には，版管理システムの1つであるCVS[1)]に管理されているリポジトリに保存されたソフトウェアの編集履歴からコードクローンへの変更を取り出し，そのコードクローンに対するメトリクス値の変化を計算する．そして，開発者ごとにメトリクス値の変化に差があるかを調査する．

以降，2章では編集傾向の詳細な定義を述べる．3章では手法の説明として，版管理システムからソフトウェアの編集履歴を取り出し，それを利用して開発者の編集傾向の有無を検定する方法について述べる．4章では本手法を実際のソフトウェアに対して適用した

† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

結果と，それに対する考察を述べる．最後に，5 章で本研究のまとめと今後の課題について述べる．

2. 編集傾向の定義

本研究では，編集傾向を「ある開発者のソースコードに対する編集作業の前後において，ソースコードに含まれるコードクローンの複雑度の変化に関する，他の開発者と比べた傾向」として定義する．これは「複雑度を大きくする傾向にある」「複雑度を小さくする傾向にある」「傾向があるとはいえない」のいずれかであり，それぞれ +，，= と表す．

本節では，編集傾向を定義するために必要となる諸定義について述べる．

2.1 クローン対応関係

編集によるコードクローンの複雑度の変化を取得するために，編集前後のソースファイル中で同じ位置にあるコードクローンの対応関係を定義する．編集前後のソースファイル F_t, F_{t+1} にそれぞれ存在するコードクローン C_t, C_{t+1} について，以下の関係 CR(C_t, C_{t+1}) が真である時，C_t は C_{t+1} 間で**クローン対応関係**があるとする．

CS(F):ソースファイル F に存在するすべてのコードクローンの集合

Sim(c_a, c_b):コード片 c_a,c_b の類似度．詳細は後述する

$O(c_c, c_d, F) = (\text{Sim}(c_c, c_d) > 0) \wedge (\forall c_a \in \text{CS}(F)$ Sim$(c_c, c_a) \leq \text{Sim}(c_c, c_d))$

CR$(C_t, C_{t+1}, F_t, F_{t+1}) = O(C_t, C_{t+1}, F_{t+1}) \vee O(C_{t+1}, C_t, F_t)$

また，F_t, F_{t+1} にそれぞれ存在するコード片 c_a, c_b の類似度 Sim(c_a, c_b) は以下のようにして求める．

まず，F_t, F_{t+1} をトークン列 P_t, P_{t+1} に変換し，各トークンに通し番号を振る．

次に，P_t, P_{t+1} にそれぞれ存在するトークン T_c, T_d がトークン列中で同じ位置にあることを表す対応関係 TR(T_c, T_d) を求める．関係 TR(T_c, T_d) は以下の条件をすべて満たす時に真とする．

T_c, T_d がそれぞれ，差分情報中の P_t, P_{t+1} の変更箇所に含まれていない

P_t, P_{t+1} 中の各トークンの通し番号を対応付ける[5]と，T_c, T_d の通し番号が一致する

そして，TR(T_c, T_d) の情報を用いて Sim(c_a, c_b) を計算する．Sim(c_a, c_b) は以下の式によって求められる．

TS(C):コード片 C 中に存在するすべてのトークンの集合

TokPair(c_a, c_b):$\{(c_c, c_d)$ |TR$(c_c, c_d), c_c \in$TS$(c_a), c_d \in$TS$(c_b)\}$

TokSum(c_c, c_d) =TokPair(c_c, c_d) の要素数

Sim(c_a, c_b) =TokSum$(c_a, c_b) \times 2 \div (c_a$ のトークン数 $+c_b$ のトークン数)

2.2 検定用データ

複雑度の変化と開発の相関を検定するために，開発者を行，コードクローンの複雑さを表すメトリクス (以下，**複雑度メトリクス**とする) を列とする表を検定用データとする．本研究では，複雑度メトリクスをコードクローンの長さ（LEN），コードクローンのトークンの種類数（TKS），コードクローン中のループの数（LOOP），コードクローン中の条件分岐の数（COND），LOOP+COND（McCabe）[2] とする．

行が開発者 A であり，列がメトリクス X である升目の要素は，開発者 A が特定のソフトウェアのソースコードに対して行ったすべての編集についてクローン対応関係にあるコードクローン C_t,C_{t+1} のメトリクス X の値の差のうち，以下のいずれかの条件を満たすもののリストとする．

C_t と C_{t+1} のメトリクス X の値の差が 0 でない

開発者 A が編集したファイルの変更差分と重複する箇所に C_t が存在する

2.3 開発者 A によるメトリクス X の値の変化頻度

各メトリクス値と開発者の相関を表すために，開発者 A の編集時に他の開発者と比べて表れることが多いメトリクス X の値の変化を，**開発者 A によるメトリクス X の値の変化頻度**として定義する．これは「大きくなることが多い」「小さくなることが多い」「頻度に有意差が見られない」のいずれかであり，それぞれ +，,N で表す．

検定用データ中でメトリクス X の列に対し，開発者 A が他の開発者と有意差がなければ開発者 A によるメトリクス X の値の変化頻度を N とする．有意差がある場合，X の列の要素を開発者 A の群とそれ以外の群に分け，各群の要素の順位平均 A_a,A_r を求める．$A_a > A_r$ なら開発者 A によるメトリクス X の値の変化頻度は +，そうでなければ　とする．

2.4 開発者の編集傾向

開発者 A の編集傾向は，開発者 A によるすべての複雑度メトリクスの値の変化頻度の中で，+ が　より多い場合に +，　が + より多い場合に　，両者の個数が等しい場合に = とする．

3. 開発者ごとの編集傾向の調査手法

本節では，開発者ごとの編集傾向の調査手法について述べる．調査手法の概要を**図 1** に示す．本手法は図

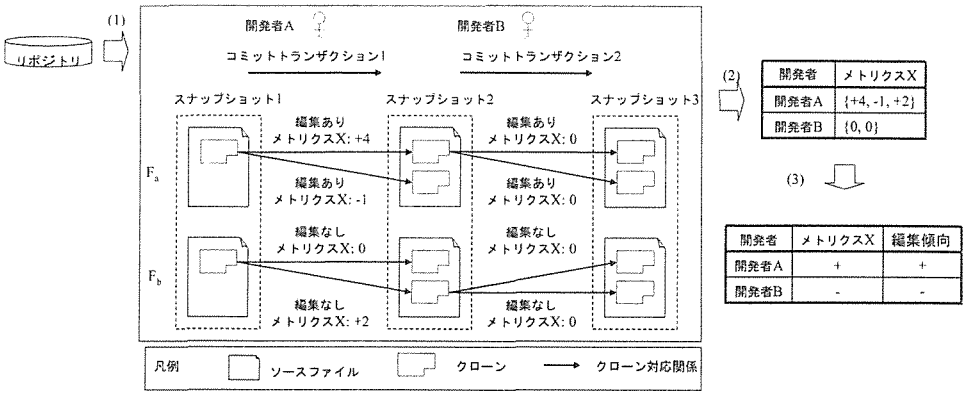

図 1 検定手法の概要
Fig. 1 abstract of method to test

中に示す (1) から (3) の 3 つのステップからなる.

まず, (1) で一度のコミット作業による一群のファイルに対する変更 (以下, **コミットトランザクション**とする) の情報を抽出し, 各コミットトランザクション前後の時点のソースコードからコードクローンを検出する. 次に, (2) で対応関係にあるコードクローンから, 検定用データを取得する. 最後に, (3) で検定用データを対象として検定する.

以降, 各ステップの詳細を説明する.

3.1 コミットトランザクション情報の抽出

まず, 版管理システムのリポジトリに保存されているソフトウェアの編集履歴から, コミットトランザクションの情報を抽出する (図 1 の (1)). 各コミットトランザクションには, 編集されたファイルのパス名や変更差分, 編集後のリビジョン番号やファイルの状態, およびコミットの日時やログ, 開発者の情報が含まれる. また, コミットトランザクションはコミットの日時が早い順に並べ, 1 から順に通し番号を振る.

そして, コミットトランザクション 1 の直前の時点および各コミットトランザクションの直後の時点のソースコード (以下, **スナップショット**とする) を取得し, 日時の早い順に通し番号を振る. その後, スナップショット中のコードクローンを検出し, スナップショットに存在するソースファイル間のクローン対応関係を抽出する.

3.2 検定用データの取得

次に, 3.1 節で取得したすべてのコミットトランザクション中のクローン対応関係から, 検定用データを取得する (図 1 の (2)).

図 1 では, 開発者 A のコミットトランザクションではソースファイル F_a のコードクローンが編集されているため, メトリクス X の値の差はすべて検定用データの要素に含める. また, ソースファイル F_b のコードクローンは編集されてないので, メトリクス X の値の差のうち, +2 だけを検定用データの要素に含める.

一方, 開発者 B のコミットトランザクションではすべてのメトリクス X の値の差が 0 だが, F_a のコードクローンは編集されているので, そのメトリクス X の値の差は検定用データの要素に含める.

3.3 検定用データを対象とした検定

3.2 節で取得した検定用データを対象として検定する (図 1 の (3)).

検定用データの開発者 A による各メトリクスの値の変化頻度を求めるため, まず, 各メトリクスを対象として Kruskal-Wallis 検定[7] を用いる. これにより, 1 人以上の開発者が他の開発者と有意差があるかを調査する.

次に, 有意差があると判断されたメトリクスの列の検定用データに対し, 各開発者と他の開発者間の有意差の有無を検定する. 開発者 A とメトリクス X に対するこの検定を行うには, メトリクス X の列にある要素を開発者 A の群とそれ以外の群に分け, Mann-Whitney の U 検定[8] (以下, 単に U 検定とする) を行う.

4. ケーススタディ

3 章で説明した手順に従って, PostgreSQL の調査を行った. PostgreSQL の CVS リポジトリから, 1997/01/01 ～ 1999/06/30 の 2 年半を半年間ずつに分け, 各期間中の編集履歴を調査した. なお, 各調査期間は 1～5 の通し番号をつける. 2.1 項のソースファイルのトークン列変換および 3.1 節のコードクローン検出には CCFinderX[2] を用いた.

各期間中の編集履歴から得られた検定用データに対

し有意水準 5%で Kruskal-Wallis 検定を行った結果を**表 1** に示す.

また, Kruskal-Wallis 検定によって有意差があると判断されたメトリクスの検定用データに対し, 3.3 節で述べた U 検定による方法を有意水準 5%で用いて, 各開発者によるメトリクス値の変化を調べた結果を**表 2** に示す.

表 2 より, 以下のことがわかる.

開発者 G の編集傾向はすべての期間で + だが, 開発者 A の編集傾向は各期間で　と = のいずれかである. これより, 開発者 G, 開発者 A の編集するコードクローンはそれぞれ複雑, 簡単になる傾向があるといえる.

開発者 B, 開発者 E の編集傾向は期間によって + と　の両方があり, 正反対になっている.

これらの観察と, 開発者が近傍のソースコードを連続して編集することが多いことから, 以下のような方法でコードクローンを管理することが考えられる.

開発者 G, 開発者 A が最近編集した所の近傍にあるコードクローンは, それぞれ今後複雑, 簡単になる可能性が高い. そのため, 前者の除去は優先し, 後者の除去は後回しにすべきだと考えられる. 開発者 B や開発者 E は時期によって編集傾向が異なっているため, 今後も継続して編集傾向を調査すべきだと考えられる.

表 1 Kruskal-Wallis 検定の結果
Table 1 The result of Kruskal-Wallis test

期間	1	2	3	4	5
LEN	有	有	有	有	有
TKS	無	有	有	有	有
LOOP	有	無	無	無	有
COND	有	有	有	有	有
McCabe	有	無	有	有	有

表 2 U 検定の結果
Table 2 The result of Mann-Whitney U Test

人	期間	LEN	TKS	LOOP	COND	McCabe	傾向
A	1	−	N	N	N	N	−
	2	+	N	N	−	N	=
	3	N	N	N	N	N	=
	4	N	N	N	−	−	−
	5	+	N	N	−	−	−
B	1	−	N	+	−	−	−
	2	N	+	N	N	N	+
	3	N	−	N	N	N	−
	4	+	−	N	+	+	+
	5	−	N	N	N	N	−
C	1	N	N	N	+	+	+
	2	N	N	N	N	N	=
	3	+	N	N	N	N	+
	4	N	N	N	−	N	−
	5	−	N	+	N	N	=
D	1	+	N	−	N	N	=
	2	−	N	N	+	N	=
	3	−	N	N	−	−	−
	4	−	N	N	−	−	−
	5	−	N	+	+	+	+
E	3	+	N	N	+	+	+
	4	N	+	N	+	+	+
	5	−	N	N	−	−	−
F	4	N	N	N	N	N	N
	5	−	−	−	−	−	−
G	4	+	N	N	N	N	+
	5	−	+	+	+	+	+
H	1	N	N	N	N	N	=
I	3	N	N	N	N	N	=
J	5	N	+	N	N	N	+
K	5	N	N	N	N	N	=
L	5	N	−	N	−	−	−

5. まとめと今後の課題

本研究では, 複雑度メトリクス値の変化を用いて, コードクローンの編集傾向を調査した. その結果, メトリクスの変化は開発者ごとに差があり, 各開発者の編集傾向によるコードクローン管理方法が提案できることが分かった.

今後の課題としては, 他のメトリクス値を用いた編集傾向の調査や, モジュールや開発時期といった開発者以外の要因による影響の除外が挙げられる.

参 考 文 献

1) Berliner, B.: CVS II: Parallelizing Software Development, *Proc. USENIX Winter 1990 Technical Conference*, pp. 341-352 (1990).
2) Kamiya, T.: CCFinder Official Site, available from 〈http://www.ccfinder.net/〉 (accessed 2008-5-20).
3) Geiger, R., Fluri, B., Gall, H. C. and Pinzger, M.: Relation of code clones and change couplings, *In Proc. FASE 2006*, pp. 411-425 (2006).
4) Kapser, C. and Godfrey, M. W.: "Cloning Considered Harmful" Considered Harmful, *Proc. 13th Working Conference on Reverse Engineering*, pp.19-28 (2006).
5) 川口真司, 松下誠, 井上克郎, 飯田元：コードクローン履歴閲覧環境を用いたクローン評価の試み, 情報処理学会研究報告, Vol. 2006, No. 125, pp. 49-56 (2006).
6) Kim, M. and Notkin, D.: Using a clone genealogy extractor for understanding and supporting evolution of code clones, *Proc. 2nd Int. Workshop on Mining Software Repositories*, pp. 17-21 (2005).
7) Kruskal, W. H. and Wallis, W. A.: Use of ranks in one-criterion variance analysis, *J.Am.Stat.Assoc.*, Vol. 47, No. 260, pp. 583-621 (1952).
8) Mann, H. B. and Whitney, D. R.: On a test of whether one of two random variables is stochastically larger than the other, *Annals of Mathematical Statistics*, Vol. 18, pp. 50-60 (1947).
9) Roy, C. K. and Cordy, J. R.: A Survey on Software Clone Detection Research, *Technical Report 2007-541* (2007).