社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ソフトウェア保守におけるコードクローンの影響に関する調査方法の比較

佐々木　唯[†]　堀田　圭佑[†]　肥後　芳樹[†]　楠本　真二[†]

† 大阪大学大学院情報科学研究科

あらまし　一般的にコードクローンはソフトウェアの保守に影響を及ぼすと考えられ，これまでにコードクローンと修正の関係について調査する研究が行われている．しかし，調査方法や対象ソフトウェアが異なれば結果に違いが生じる可能性が高く，既存の研究１つ１つからは必ずしも一般的なことがいえない．本稿では，ソースコードへの修正の種類，コードクローン検出ツール，調査の粒度など，様々な条件な下で計測を行い，同一のソフトウェアに対して結果を比較した．その結果，特に調査手法が異なるとき，計測結果にも影響が出やすいことが分かった．
キーワード　コードクローン，ソフトウェア保守，バージョン管理システム

Comparison of Investigation Methods about Influence of Code Clone on Software Maintenance

Yui SASAKI[†], Keisuke HOTTA[†], Yoshiki HIGO[†], and Shinji KUSUMOTO[†]

† Graduate School of Information Science and Technology, Osaka University

Abstract　It is generally said that code clone affects software maintenance and there are many studies about relationship between code clone and modification. However, various investigation methods or target software may cause different results. In this study, we measured under various conditions such as types of modification, code clone detection tools, and granularities of investigation, and then compared results for the same software. The study found that it often happens that we obtain different results under different investigation methods.
Key words　Code Clone, Software Maintenance, Version Control System

1. まえがき

近年，ソフトウェアの保守作業量を増大させる要因として，コードクローンの存在が注目されている．ソースコード中で内容が一致または類似したコードが存在すると，その一部に修正が必要な場合，同様の修正がクローン関係にあるコードにも必要となる．そのため，コードクローン検出手法の提案や検出ツールの開発，リファクタリング支援，ソフトウェアに及ぼす影響など多岐にわたった研究が盛んに行われている [1] [2] [3] [4] [5] [6].

しかし，コードクローンがソースコードの修正にどの程度影響を与えているのかを定量的に調査した研究は少ない．また，既存の研究で提案されている手法は調査の粒度や修正量の定義が異なり，対象ソフトウェアも様々である．門田らは，１つのレガシーソフトウェアに対して調査を行った結果，コードクローンを含むモジュールは含まないモジュールより改版数が高く，より多くの保守コストが費やされていると述べている [7]. Krinke は，複数のオープンソフトウェアに対して調査を行い，コードクローンを含む行は含まない行より修正される割合が低く，コードクローンは安定していると結論づけた [8]．Krinke の調査は行単位で行われたが，これに対して Göde らは字句単位で修正割合の比較を行ったところ，コードクローンは削除される割合は高いものの，追加とそれ以外の修正を含めた割合は低く安定していると述べている [9]．Lozano らは，全てあるいは一部の期間でコードクローンを含んでいたメソッドは，全ての期間でコードクローンを含んでいないメソッドと比較して，より高い頻度で変更が加えられていると報告している [10]．このように研究によってそれぞれ異なった結論が導かれており，その原因はこれらの調査に用いる要素が異なること，利用するコードクローン検出ツールの違いにあると著者らは考えた．

本稿では，同一のソフトウェアに対してコードクローン検出ツールと調査手法の組み合わせを変えることで，結果に違いが出るかどうか調査を行った．また，ソースコードに対する修正の中でも，バグの修正による作業量が多ければソフトウェアの保守コストも大きくなると考えられ，これに基づいてコードクローンとバグ修正の関係を調査した研究も行われている [11]. そこで，実験対象のソフトウェアのうち１つは，ソースコード

に対する全ての修正を用いた場合と，バグの修正のみを用いた場合の 2 通りの結果の比較を行った．

2. 本稿で用いるコードクローン検出ツール

コードクローン（以下「クローン」と呼ぶ）とはソースコード中に存在する同一，あるいは類似するコード片のことであるが，どのような基準で類似していると判断するかは検出手法や検出ツールによって異なる．本稿では表 1 に示す 4 種類のコードコードクローン検出ツールを用いて，検出されるクローンの違いが結果にどのような影響を及ぼすのか調査を行っている．本節ではその 4 種類の検出ツールについて述べる．

2.1 *CCFinder*

CCFinder [2] は，字句単位でクローンを検出する．大規模ソフトウェアに対しても実用的な時間で検出を行うことが可能である．*CCFinder* は複数のプログラミング言語に対応しており，プレーンテキストや未対応の言語に対しても完全一致判定によるクローンは検出可能である．また，ソースコード中に含まれる変数などのユーザ定義名，定数のパラメータ化や，複雑な名前の正規化を行うことで，この違いを吸収したクローンを検出することができる他，クローンとみなすための閾値の指定も可能である．

2.2 *CCFinderX*

CCFinderX [3] は *CCFinder* のメジャーバージョンアップであり，*CCFinder* と同様に字句単位の検出に分類される．抽象構文木ベースの前処理を行っており，*CCFinder* からの改良点として，マルチスレッド化や，メトリクス分析への対応などが挙げられる．

2.3 *Simian*

Simian [4] は，行単位の検出に分類されるコードクローン検出ツールで，少ないメモリでも高速な検出が可能である．また，実用的でないクローンの検出数を軽減するために特定の文を無視したり，大文字小文字の違い，変数名の違い，数値の違い等を無視したりといった設定を行うことができる．

2.4 *Scorpio*

Scorpio [5] はプログラム依存グラフを用いてクローンを検出するツールであり，Java にのみ対応している．プログラム依存グラフを構築するという前処理を必要とするため，他の検出手法と比べて検出に要するコストが高いが，*Scorpio* では，プログラム依存グラフを探索する際に基点として用いる頂点に制限を設けることで，検出の際の計算コストを最大 36%にまで削減した．また，行単位や字句単位の手法では検出できない順序入れ替わりクローンなどの非連続クローンを検出することが可能である．

表 1　コードクローン検出ツールの比較
Table 1　Comparable of Code Clone Detection Tools

ツール名	略記	検出方法
CCFinder	ccf	字句単位
CCFinderX	ccfx	字句単位
Simian	sim	行単位
Scorpio	sco	プログラム依存グラフ

3. 本稿で用いる手法

本節では今回用いた 3 つの手法について述べる．いずれの手法もバージョン管理されているオープンソフトウェアを調査対象としているが，計測対象とするリビジョンや，評価指標が異なっている．その違いを表 2 にまとめる．各手法で定義する評価指標について以下で述べるが，いずれの定義でも R を計測対象リビジョンの集合，r を着目中のリビジョンとする．

3.1 佐野らの手法

佐野らの手法 [12] は，修正の加えられた連続する行を 1 つの修正箇所とみなし，クローン，クローン以外のどちらへの修正であったか，その割合からそれぞれの修正頻度を計測し，比較を行う．リビジョン r のソースコードの行数を $l(r)$，クローンを含む行数，含まない行数をそれぞれ $lc(r), ln(r)$ とし，また，クローンに加えられた修正箇所数を $mc(r)$，クローン以外に加えられた修正箇所数を $mn(r)$ とする．このとき，次式で算出される MF_c を**クローンへの修正頻度**，MF_n を**クローン以外への修正頻度**と定義し，評価指標とする．

$$MF_c = \frac{\sum_{r\in R} mc(r)}{|R|} \cdot \frac{\sum_{r\in R} l(r)}{\sum_{r\in R} lc(r)} \tag{1}$$

$$MF_n = \frac{\sum_{r\in R} mn(r)}{|R|} \cdot \frac{\sum_{r\in R} l(r)}{\sum_{r\in R} ln(r)} \tag{2}$$

3.2 Krinke の手法

Krinke の手法 [8] では，開発期間のうち 200 週間分を抽出し，1 週間ごとの最新リビジョンについて計測している．クローンを含む行の集合を $C(r)$，含まないの行の集合を $N(r)$ とし，各集合に対して，追加，削除，それ以外の修正が加えられた行を求める．このとき，クローン，クローン以外に対するそれぞれの追加割合を，

$$AC = \frac{\sum_{r\in R} |AC(r)|}{\sum_{r\in R} |C(r)|} \quad and \quad AN = \frac{\sum_{r\in R} |AN(r)|}{\sum_{r\in R} |N(r)|} \tag{3}$$

削除割合を，

表 2　手法の比較
Table 2　Comparable of Measurement Methods

手法	計測単位	計測対象リビジョン	評価指標
佐野らの手法 [12]	箇所 (連続する行)	連続するリビジョン	修正箇所数の割合
Krinke の手法 [8]	行	1 週間ごとに抽出	修正行数の割合
Lozano らの手法 [13]	メソッド	連続するリビジョン	メソッドの変更割合 × 同時に変更されたメソッドの割合

$$DC = \frac{\sum_{r \in R} |DC(r)|}{\sum_{r \in R} |C(r)|} \quad and \quad DN = \frac{\sum_{r \in R} |DN(r)|}{\sum_{r \in R} |N(r)|} \quad (4)$$

修正割合を，

$$CC = \frac{\sum_{r \in R} |CC(r)|}{\sum_{r \in R} |C(r)|} \quad and \quad CN = \frac{\sum_{r \in R} |CN(r)|}{\sum_{r \in R} |N(r)|} \quad (5)$$

と定義する．ただし，$AC(r)$，$AN(r)$ は次のリビジョンまでに追加された，次のリビジョンの行の集合で，$DC(r)$，$DN(r)$，$CC(r)$，$CN(r)$ は次のリビジョンまでに削除あるいは修正された現在のリビジョンの行の集合である．以上の 6 つの式を評価指標とする．

3.3 Lozano らの手法

Lozano らの手法 [13] では，各メソッドについてクローンを含んでいた期間 P_C，含んでいなかった期間 P_N を調べ，両方の期間を持つ *SC-Method*，常にクローンを含んでいた *AC-Method*，常にクローンを含んでいなかった *NC-Method* の 3 種類に分類する．また，この手法ではある期間におけるメソッドの保守コストを定義する．このときクローンとクローン以外の保守への影響を比較する手段として，*AC-Method* に分類されるメソッドと *NC-Method* に分類されるメソッドの保守コストの分布の比較と，*SC-Method* の期間 P_C における保守コストと期間 P_N における保守コストの分布の比較の 2 通りを行う．

保守コストの計算方法は以下のようになっている．まず，m をメソッドチェイン，P をリビジョンの集合が表すある期間とし，

- *ChangedRevisions(m, P)*：期間 P 中に m に対して変更が行われたリビジョンの集合
- *Methods(r)*：r におけるメソッドの集合
- *ChangedMethods(r)*：r で変更されたメソッドの集合
- *CoChangedMethods(m, r)*：r で m と共に変更されたメソッドの集合

と定義する．更に，次式で *likelihood*，*impact* を定義し，その積 *work* の値を保守コストとする．

$$likelihood(m, P) = \frac{|ChangedRevisions(m, P)|}{\sum_{r \in P} |ChangedMethods(r)|} \quad (6)$$

$$impact(m, P) = \frac{\sum_{r \in P} \frac{|CoChangedMethods(m, r)|}{|Methods(r)|}}{|ChangedRevisions(m, P)|} \quad (7)$$

$$work(m, P) = likelihood(m, P) \cdot impact(m, P) \quad (8)$$

4. 実　　験

本節では，上述のコードクローン検出ツール，調査手法の組み合わせによって，結果に違いが出るかどうか調査するために行った実験，及びその結果について述べる．

4.1 実験手順

本実験の手順は以下の通りである．

（1） 計測対象リビジョンのソースファイルをリポジトリから取得．

（2） ソースファイルのうちテストケース，及び重複するクラスを含むファイルを削除．

（3） *CommentRemover*(注1) を使用してソースコードを正規化．

（4） 複数のコードクローン検出ツールを用いてそれぞれクローンの位置を特定．

（5） 計測対象リビジョンとその次のリビジョンのソースコード間の差分情報を，行単位で取得．

（6） （4）と（5）で得た内容とメソッドの位置情報を照合し，手法ごとに必要な値を計測．

（3）で行った正規化は以下の 5 種類である．

- 空白行の削除．
- ブロックコメント (/*…*/) の削除．
- ラインコメント (//…) の削除．
- インデントの削除．
- 中括弧 ({ }) のみの行を削除し，1 つ上の行に追加．

4.2 計測対象

本実験では *SourceForge*(注2) で公開されているオープンソースソフトウェアの中から，5 つのソフトウェアを対象に計測を行った． 対象としたソフトウェアを表 3 に示す．いずれもバージョン管理システム *Subversion* を用いて開発されており，開発言語は Java である．

Ant は計測対象リビジョンとして，全てのリビジョンを用いる場合と，バグの修正を含むリビジョンのみを用いる場合の 2 通りの実験を行った．なお，バグ情報の抽出は 1 リビジョンごとに人手によって行ったため，信頼性が高い．

4.3 Ant についての結果

本稿では紙面の都合上，Ant の全てのリビジョンに対する結果のみを詳細に説明する．佐野らの手法による結果を図 1 に，Krinke の手法による結果を図 2 に，Lozano らの手法による結果を図 3 に示す．なお，Lozano らの手法による結果は *AC-Method* と *NC-Method* を比較した図のみ掲載する．

佐野らの手法では，いずれの検出ツールを用いた場合もクローンの修正頻度が低いという結果になった．Krinke の手法でも検出ツール間に大きな差はなかった．また，追加割合はクローン以外に対する値が大きく，削除割合と修正割合はクロー

表 3　計測対象ソフトウェア
Table 3　Target Software Systems

プロジェクト名	最終リビジョン		開発期間
	番号	総行数	
OpenYMSG	194	14,111	08.11.19-10.12.06
EclEmma	1,220	31,409	06.09.21-10.12.29
Masu	1,620	79,360	06.11.08-10.12.25
TVBrowser	6,829	264,796	03.04.22-10.11.15
Ant	711,860	198,864	01.09.21-

（注1）：http://sdl.ist.osaka-u.ac.jp/~higo/cgi-bin/moin.cgi/commentremover
（注2）：http://sourceforge.net/

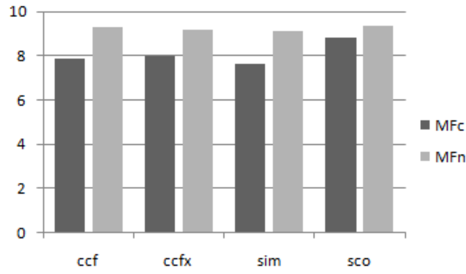


図 1　佐野らの手法による結果
Fig. 1　Result by Sano's Method

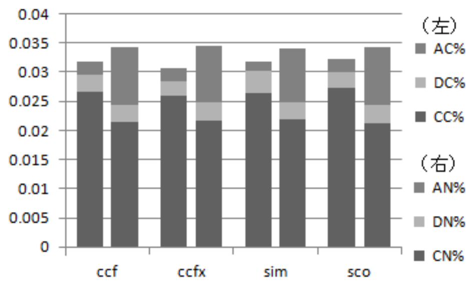


図 2　Krinke の手法による結果
Fig. 2　Result by Krinke's Method

ンに対する値が大きいという結果が得られた．

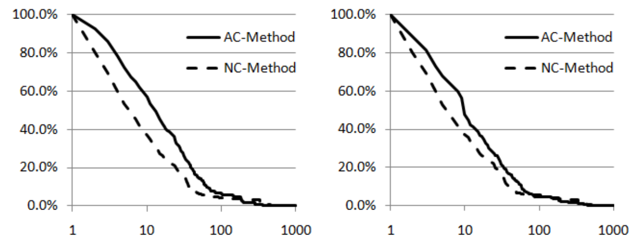
Lozano らの手法ではメソッドによって保守コストに大きなばらつきがあるため，累積度数でグラフに表現している．x 軸は比較対象の保守コストの最大値を 1,000 等分したときの各値を表し，それ以上の値を持つメソッド数の割合をプロットしている．なお，x 軸は対数軸である．図 3 ではいずれの検出ツールにおいても，ある値以上の保守コストの値を持つメソッド数の割合が常に *AC-Method* の方が高いことから，クローンを含むメソッドの保守コストが高いという結果が得られた．また，ここには載せていない *SC-Method* の比較については，*CCFinderX*, *Scorpio* を用いた検出ではクローンを含んでいた期間 P_C の保守コストが高く，他の 2 つの検出ツールを用いた検出ではあまり差は出なかった．

4.4　全ての実験結果

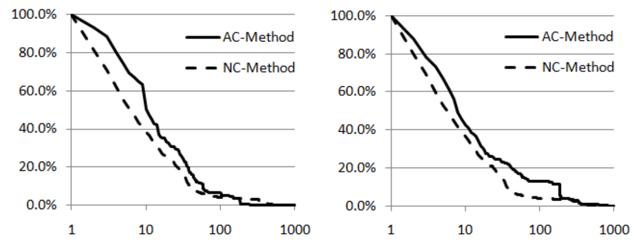
同様に，各ソフトウェアに対して 4 種類のコードクローン検出ツールと，3 種類の調査手法による全 12 通りの結果を得た．その結果を表 4 に示す．ここではクローンに対する修正が大きいという結果を C，クローン以外に対する修正が大きいという結果を N，どちらも差がないという結果を-で表している．なお，手法によって異なる評価指標を比較できないため，各手法における結果は次の基準で判定する．

- 佐野らの手法ではクローンとクローン以外の修正頻度を比較し，その差が値の大きい方の 5%以上である場合，値の大きい方を採用する．
- Krinke の手法では，クローンとクローン以外への削除割合と修正割合の和を比較する．これは，ソフトウェアの開発過程において追加されるコードは保守作業と関係のないものも多く含まれると考えられるためである．佐野らの手法と同様に，比較する値の差が大きい方の 5%以上である場合，値の大きい方を採用する．
- Lozano らの手法では，次の手順で比較を行う．

（1） *AC-Method* に含まれるメソッドの保守コストと，*NC-Method* に含まれるメソッドの保守コストを，Mann-Whitney の U 検定を用いて比較する．有意水準を 5%とし，有意差の



(a) ccf　(b) ccfx



(c) sim　(d) sco

図 3　Lozano らの手法による結果
Fig. 3　Result by Lozano's Method

表 4　各組み合わせにおける結果
Table 4　Overall Result

プロジェクト名	手法	結果			
		ccf	*ccfx*	*sim*	*sco*
OpenYMSG	佐野	N	C	C	N
	Krinke	N	C	C	N
	Lozano	-	-	N	-
EclEmma	佐野	N	N	-	N
	Krinke	N	N	N	-
	Lozano	N	N	-	-
MASU	佐野	C	-	C	C
	Krinke	C	C	C	C
	Lozano	C	C	C	C
TVBrowser	佐野	N	N	N	N
	Krinke	C	C	N	C
	Lozano	C	C	C	C
Ant (all)	佐野	N	N	N	N
	Krinke	C	C	C	C
	Lozano	C	C	C	C
Ant (bug)	佐野	N	N	N	N
	Krinke	C	C	C	C
	Lozano	C	C	-	-

あった場合，大きいと判定された方を採用する．

（2） 上記の比較で有意差のなかった場合，*SC-Method* に含まれるメソッドの，期間 P_C における保守コストと期間 P_N における保守コストを，Wilcoxon の符号順位和検定を用いて比較する．これも有意水準を 5%とし，有意差のあった場合，大きいと判定された方を採用する．

5. 考　　察

表 4 から次のような傾向が得られた．

- OpenYMSG は組み合わせによってばらつきがみられる．
- EclEmma と MASU は，組み合わせによる違いがみられない．
- TVBrowser と Ant は，佐野らの手法ではクローン以外に対する修正が大きく，他の手法ではクローンに対する修正が大きい．
- Ant の全ての修正を用いた場合とバグの修正のみを用いた場合の結果に違いは見られない．

本節では，特に Ant に対する計測結果について考察を行う．

5.1 修正情報の違い

Ant について，全ての修正とはソースコードに修正の行われた 5,412 リビジョンを，バグの修正とはバグを含んでいた 460 リビジョンを指す．この 2 種類の対象について各手法における評価指標を比べると，佐野らの手法，Krinke の手法では，クローンに対する指標とクローン以外に対する指標の大小関係は変わらないものの，全ての修正を用いた方が評価指標が大きいという結果が得られた．Lozano らの手法でも，どちらの場合もクローンを含むメソッドの保守コストが高いという結果であったが，分類されたメソッドごとに保守コストの値が最大のメソッドを比較すると，全ての修正を用いた方が保守コストが大きいという結果であった．

このことから，バグの修正のみを用いても全ての修正を用いても，クローンとクローン以外に対する修正の作業量の関係に違いはなく，バグの修正のみに絞った場合の方が修正の作業量は少ないといえる．

5.2 手法の違い

Ant は，佐野らの手法とそれ以外の手法で異なる結果が得られたため，その原因について詳細な調査を行った．

5.2.1 評価指標の違い

佐野らの手法と Krinke の手法はどちらも，クローンとクローン以外への評価指標を直接比較しているため，この 2 つの手法の違いを調べる．

Krinke の手法は修正の種類を分けているため，佐野らの手法でも同様に修正の種類ごとに計測し，Krinke の手法とグラフの表示方法を統一した．その結果を図 4 に示す．グラフから，いずれの検出ツールにおいても追加，修正される頻度はクローン以外に対して高く，削除される頻度はクローンに対して高いという結果となっている．本稿では Krinke の手法を適用した場合，ソースコードの追加に関する指標を無視して比較しているため，この図においても追加に関する値を無視し，削除，修正による頻度のみを考慮する．その結果，*CCFinderX*,*Scorpio*

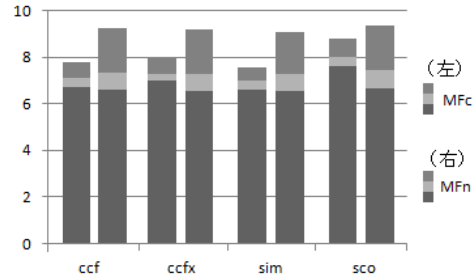


図 4 佐野らの手法による結果（追加を無視した場合）
Fig. 4 Results by Sano's Method without Additional

を用いた検出では，表 4 に示す結果とは異なり，クローンの修正頻度が高いという結果が得られた．

また，全ての実験対象について同様の比較を行ったところ，表 5 に示す 11 箇所の結果は，表 4 に示す佐野らの手法適用結果と異なるものとなった．このうち 10 箇所については，クローンに対する修正が大きいという結果により近づいていることが分かる．したがって，佐野らの手法ではコードの追加による影響が大きい場合があっても，用いられる評価指標では修正の内容について考慮できないため，正しい結果が得られていない可能性がある．

5.2.2 粒度の違い

次に，各手法における粒度の違いが結果に影響を与えているのかどうか調査を行った．

佐野らの手法，Krinke の手法について，計測対象リビジョンごとに修正頻度や修正行数の割合を見ると，佐野らの手法ではクローン以外への修正頻度が高いが，Krinke の手法ではクローンへの修正行数の割合が高いリビジョンがいくつか存在した．そのうちのあるリビジョンでは，お互いがクローン関係にある 2 つのメソッドに対して，共通部分の抽出によるリファクタリングが行われていた．このメソッドのみに着目すると，修正された行数は 85 行，修正箇所数は 5 箇所で，1 箇所における最大行数は 43 行であった．このように連続する長いコードに対してまとめて修正や削除が行われた場合，評価指標が行数に依存する Krinke の手法では，修正のコストを多く計算してしまう．

また，このリファクタリングが行われた後は 2 つのメソッドはクローンを含まないため，Lozano らの手法では *SC-Method* に分類される．この 2 つのメソッドは *SC-Method* に含まれるメソッドの中で，クローンを含む期間 P_C における保守コスト

表 5 佐野らの手法における結果の違い
Table 5 Difference of Results by Sano's Method

プロジェクト名	結果			
	ccf	*ccfx*	*sim*	*sco*
OpenYMSG	N → -			
EclEmma			- → N	
MASU		- → C		
TVBrowser	N → -	N → -		N → C
Ant (all)		N → -		N → C
Ant (bug)		N → -	N → C	N → C

が特に高い値を示していた．そこで，この２つのメソッドのそれぞれの修正経緯を全計測対象リビジョンについて調査したところ，リファクタリング前，すなわちクローンを含んでいた時期に行われた修正回数が多く，ほとんどが両メソッドに対して同時に行われたものであった．このように共通のコードを含む部分への度重なる修正は，保守の作業量を増大させ，Lozanoらの手法ではその影響が評価指標に明確に表れていることが分かった．

以上のことから，調査の粒度や調査方法が異なると結果に影響を与える事例が見られ，単一の調査方法に絞った結果だけでは，不十分であるといえる．また，コードクローン検出ツールによる違いはあまり多く見られなかったが，全く同じ結果が得られたわけではないため，更なる考察が必要である．

6. 結果の妥当性

対象ソフトウェアの規模

規模の小さいソフトウェアは定義されたメソッド数が少ないため，Lozanoらの手法を適用した際に保守コストの値の分布が極端なものとなる傾向があり，適切な比較が行えていない恐れがある．

計測対象

本稿で対象としたソフトウェアは５つであり，Javaで開発されたオープンソースのソフトウェアに限定して調査を行った．このため，より多くのソフトウェアを対象として実験を行った場合や，異なる言語で記述されたソフトウェアや商用ソフトウェアを対象とした場合は，今回得られた結果と異なる結果が得られる可能性がある．

差分検出の粒度

本稿では行単位でソースコードの差分を検出しているが，この方法ではソースコードの整形など重要でない修正も含まれている．また，字句単位でクローンを検出しても修正情報は行単位であるため，厳密にクローンに対する修正であったかどうか判断ができていない恐れがある．このため，字句単位で差分の検出を行うか，差分の検出単位をクローン検出ツールの検出単位に揃えて実験を行えばより正確な調査を行えていたかもしれない．

7. あとがき

本稿では，コードクローンがソースコードの修正に与える影響について，計測の粒度や評価指標，コードクローン検出ツールの違いによって結果にどのような違いが出るのか調査した．また，あるソフトウェアについてはバグの修正のみを対象とした場合と，全ての修正を対象とした場合の結果を比較した．その結果，バグの修正のみに絞っても結果に大きな違いは出ないことが分かった．また，同一のソフトウェアでも調査手法とコードクローン検出ツールの組み合わせの違いは結果に違いをもたらすことがあり，特に調査方法の違いが影響を与えやすいことが分かった．

従って，コードクローンの影響を調査するに辺り，手法を限定するのではなく複数の調査方法による結果から総合的に判断することが望ましい．今後の課題としては，より多くのソフトウェアに対しての調査，今回用いなかったコードクローン検出ツールや調査手法の適用など，多くの条件を揃えてより一般的な結果の傾向を把握することや，クローンを含むコードと含まないコードに対する，実際の保守作業量の違いを明確に表す指標との比較が挙げられる．

謝辞 本研究は一部，文部科学省「次世代IT基盤構築のための研究開発」(研究開発領域名：ソフトウェア構築状況の可視化技術の開発普及) の委託に基づいて行われた．また，日本学術振興会科学研究費補助金基盤研究 (A)(課題番号：21240002) および (C)(課題番号：23650014)，文部科学省科学研究費補助金若手研究 (B)(課題番号：22700031) の助成を得た．

文献

[1] 肥後, 楠本, 井上：“コードクローン検出とその関連技術”, 電子情報通信学会論文誌, **J91–D**, 6, pp. 1465–1481 (2008).

[2] T. Kamiya, S. Kusumoto and K. Inoue: “CCFinder: A multi-linguistic token-based code clone detection system for large scale source code”, IEEE Transactions on Software Engineering, **28**, 7, pp. 654–670 (2002).

[3] CCFinderX. `http://www.ccfinder.net/ccfinderx-j.html`.

[4] “Simian - similarity analyser”. `http://www.harukizaemon.com/simian/`.

[5] 肥後, 楠本：“コードクローン検出に必要な計算コストの削減を目的としたプログラム依存グラフ頂点集約手法の提案”, ソフトウェアエンジニアリング最前線 2010(ソフトウェアエンジニアリングシンポジウム 2010 予稿集), pp. 127–134 (2010).

[6] J. Johnson: “Substring matching for clone detection tools”, Proc. International Conference on Software Maintenance 94, pp. 120–126 (1994).

[7] 門田, 佐藤, 神谷, 松本：“コードクローンに基づくレガシーソフトウェアの品質の分析”, 情報処理学会論文誌, **44**, 8, pp. 2178–2188 (2003).

[8] J. Krinke: “Is cloned code more stable than non-cloned code?”, Eighth IEEE International Working Conference on Source Code Analysis and Manipulation, pp. 57–66 (2008).

[9] N. Göde and J. Harder: “Clone stability”, European Conference on Software Maintenance and Reengineering, pp. 65–74 (2011).

[10] A. Lozano, M. Wermelinger and B. Nuseibeh: “Evaluating the harmfulness of cloning: a change based experiment”, IEEE Fourth International Workshop on Mining Software Repositories (2007).

[11] 斎藤, 吉田, 松下, 井上：“コードの生存期間を考慮したコードクローンと欠陥修正の関係調査”, 電子情報通信学会技術研究報告, **110**, 227, pp. 19–24 (2010).

[12] 佐野, 肥後, 楠本：“重複コードと非重複コードにおける修正頻度の比較”, 電子情報通信学会技術研究報告, **109**, 456, pp. 43–48 (2010).

[13] A. Lozano and M. Wermelinger: “Asessing the effect of clones on changeability”, International Conference on Software Maintenance, pp. 227–236 (2008).