社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－SE－154 (7)
2006／11／27

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

川口 真司† 松下 誠‡ 井上 克郎‡ 飯田元†

†奈良先端科学技術大学院大学情報科学研究科
‡大阪大学大学院情報科学研究科

ソフトウェアの保守工程における大きな問題の一つとしてクローンが指摘されており，これまでに様々なクローン検出手法が提案されている．しかし，ソフトウェアが大規模になるほど検出されるクローンも膨大なものとなるため，どのクローンが対処が必要なのかについての判断の支援が求められている．本研究では，そのような支援の一環としてクローンの作成，編集に関わった開発者に着目する．そのために版管理システムが保持している過去の履歴情報を解析する．また解析手法を PostgreSQL に対して適用し，得られた情報を対処が必要かどうかの判定に用いることができるか分析を行った．

Evaluation of Code Clone Using Clone History Browser

Shinji Kawaguchi† Makoto Matsushita‡ Katsuro Inoue‡ Hajimu Iida†

†Graduate School of Information Science and Technology, Nara Institute of Science and Technology
‡Graduate School of Information Science and Technology, Osaka University

Code clones are serious problem in software maintenance process. To solve this problem, many code clone detection methods are proposed however, not only showing clones, but also supporting to decide which clones are deleted. Therefore, we present the analysis to detect developers who edit each clone. We apply the method to PostgreSQL and analyze the relationship between clone triage and developers who write or touch the clone.

1 はじめに

保守工程におけるさまざまな問題のなかでも非常に大きな問題の一つとして，ソースコード中に含まれる重複コード(以下，クローン)が挙げられる [4].

これまでにクローンを検出するための様々な手法が提案されており，そのいくつかは実際に利用可能なシステムとして実用化されている．さらに，発見されたクローンを効率的に閲覧，除去するためのシステムも提案されている．これらのシステムを用いることによってソフトウェアに含まれているコードクローンを調査，除去するための労力が大幅に軽減される．

しかし，これらのクローン摘出ツールによる様々な適用実験の結果，大規模なソフトウェアにはそれだけ大量のクローンが含まれていること，それら大量のクローン全てが必ずしも削除するべきクローンでないことがわかってきた．ある種のイディオム的な一連のコードの流れや，デザインパターンを構成するコード片などはその一例と言える．

そのため，得られたクローン情報をソフトウェアの品質改善に結びつけるには，どのクローンが優先的に対処すべきかを提示する必要がある．

本研究では，クローンが優先して対処されるべきかどうかを評価するために，そのクローンの履歴と過去にクローンの作成や編集に関わった開発者に着目する．例えば，熟練した開発者であれば不用意にコピー&ペーストをすることは少なく，重複した部分も何らかの意図があって残していると考えられる．また，そのような開発者が作成したクローンならば残すべき価値が高いクローンの可能性が高い．逆に未熟な開発者が作成したクローンにはそのような必然性がなく，何らかの対応が必要であるという予測が成り立つ．

このような情報を分析するためには，ソースコードをいつ誰がどのように編集したかという詳細な履歴情報が必要となる．従来であればこのような情報を網羅的に蓄積することは非常に困難であったが，近年の版管理システムの普及により自動的にこれらの情報を収集することが可能となった．

そこで，本研究では版管理システムが保持する情報から各クローンを編集したのが誰かを特定する，クローン関係者抽出手法を提案する．本手法を用いることで，開発者の能力を加味したクローンの評価や，例えばクローンを最も多く作成しているのは誰か，といった開発者の評価の指針になりうる情報の提供が可能になる．

そして，クローン履歴や提案する手法によって抽出されたクローンを作成，編集した開発者やその履歴がクローンの品質にどのような影響があるかについての分析を行う．そのために PostgreSQL に対して提案手法を適用し，その結果から開発者や変更履歴とクローンの性質との関係について議論する．

2 前提とするシステム

2.1 クローン分析システム

クローン分析手法には大きくわけてソースコードの字句解析に基づく手法 [1,4,6] と，特徴メトリクスに基づく手法 [3,7] に分けられる．ソースコードの字句解析に基づく手法 では，ソースコード中で同一の文字列を検索することでクローンの検出を行う．特徴メトリクスに基づく手法では，例えばクラスや関数，ファイルのようなプログラム中のある種の単位ごとに特徴メトリクスを定義・算出し，それらのメトリクス値が類似したものをクローンとして抽出する手法である．一般には字句解析に基づく手法のほうがコストが増えるが，より細粒度なクローンを抽出できる．

本研究では，字句解析ベースの検出ツール CCFinder [4] を利用してクローン履歴の分析を行う．CCFinder は高いスケーラビリティを有しており，大規模なソフトウェアに対しても実用的な時間でクローンの抽出を行える．また，実際にさまざまな大規模ソフトウェアへ適用され，その有用性が確認されている [8].

クローン分析を行う際，CCFinder は空白や改行，コメント等を除去するとともに，入力テキスト中の変数名や関数名等を同一記号に縮退させる．その後に，しきい値以上の長さの共通字句列を探索し，全ての対のリストを出力する．コピー& ペーストによってクローンが作られた場合，変数名などが変更されないことは稀であり，何らかの形で変数名などがコピー先のコンテキストに合致するよう書きかえられることが多い．CCFinder では変

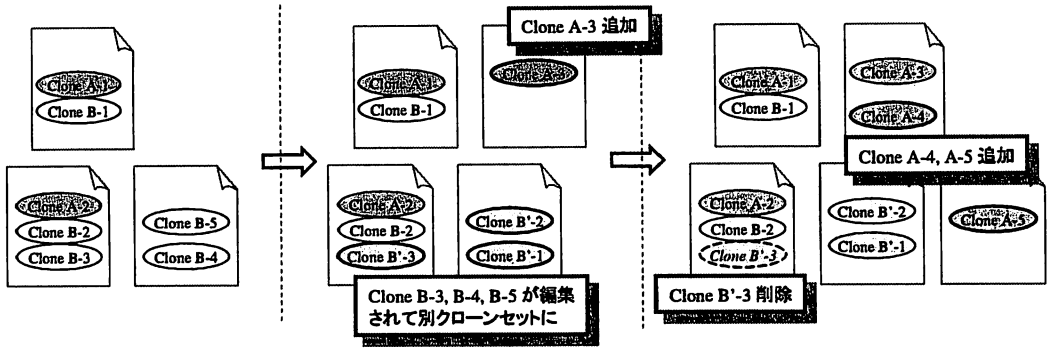

図 1: クローンの変更履歴

数名，関数名を同一記号とみなすことによって適切にコピーを検出できる．また，長大な共通字句列が存在したときに，その部分字句列が併せて出力されるのを防ぐために，それぞれ互いに包含関係でないもののみをクローンとして出力する．

2.2 版管理システム

版管理システムとは CVS [2] や Subversion [9] のようにプロダクトの保管・管理に用いられるシステムである．過去のクローンの状態を知るためには，何らかの方法で過去の時点でのクローンを知る必要があるが，版管理システムが普及する以前は過去のソースコードを取得する標準的な方法は存在しなかった．版管理システムでは，管理下のプロダクトを任意の時点の状態に復元して取得する機能が提供されている．

版管理システムには様々な実装が存在するが，ほとんどの版管理システムはプロダクトそのものだけでなく，「いつ」「だれが」「どの部分」を編集したかを逐一保存している．これらの情報を抽出し，CCFinder を用いて得られるクローン情報と突きあわせることで，それぞれのクローンを作成した開発者を特定することができる．なお，本稿では版管理システムとして CVS を用いるものとする．

3 クローン関係者分析手法

本節では，クローンの作成や削除，途中の編集などに関わった開発者を分析するクローン関係者分析手法について述べる．まず，3.1 節にてクローンに関する単語の定義を行い，次に 3.2 節にて，過去に筆者らが提案したクローン履歴分析手法について述べる．それらを踏まえてクローン関係者手法を 3.3 節にて述べる．

3.1 諸定義

本解析手法では時間と共に変更されるソースコードを Δt ごとに区切って考える (図 2)．分析の対象を版管理システムの管理下にあるソースコードファイルの集合とし，ある時刻 t における集合を **プロダクト (Product)** F_t と呼ぶ．

いま，各ファイルを文字列と考え，その連続する部分列を**コード片 (Code snippet)** と呼ぶ．そして，F_t に含まれるコード片のうち，共通字句列となるコード片の対 (a, b) を F_t に関する**クローンペア (Clone pair)**，a や b をそれぞれ F_t に関する**クローン (Clone)** と呼ぶ．このとき a と b は**クローン関係 (Clone relationship)** にあるという．また，クローン関係を同値関係と考え，その同値類を**クローンセット (Clone set)** と呼ぶ．

3.2 クローン履歴分析

我々はこれまでに過去のクローンから新しい時点のクローンへの対応関係をクローン履歴関係と定義し，その抽出を行う手法を提案している [11]．

このような履歴関係を用いることで，例えば図 1 のように当初二つのクローンセット A, B が 3 つのクローンセット A, B, B' に変化しているとき，このような関係をたどることで，A-3 は過去の時点では A-1，A-2 からコピーされたことや，B'-1, B'-2, B'-3 は元々 B-1, B-2 のコピーだったものが編集された結果，別々のクローンセットとなっていることなどがわかる．

この手法では，指定された解析開始時間 t_{begin}，解析終了時間 t_{last} で定義される期間を与えられた

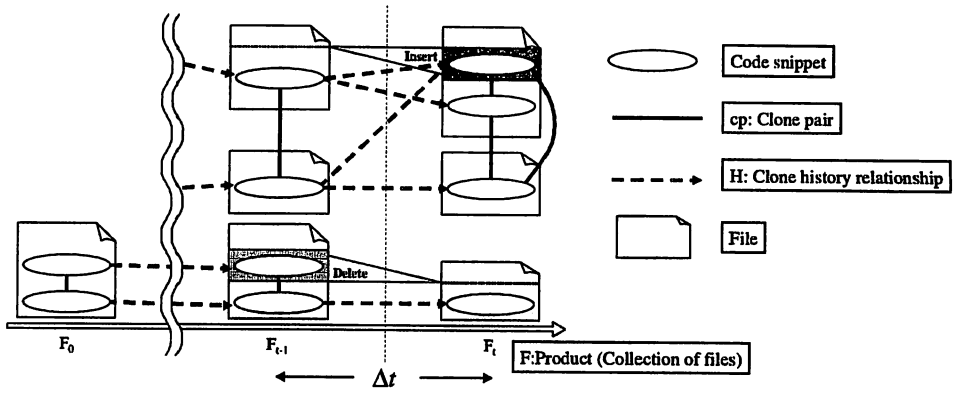

図 2: クローンとクローン履歴関係

解析間隔 Δt で区切る．そして，それぞれの時刻 t において CCFinder を用いてクローンを抽出し，$t-1$ に存在するクローンが t のクローンのどれに対応するかを解析を行う (図 2).

3.2.1 行番号対応関係

差分情報の解析にあたり，時刻 t より Δt だけ過去の時点におけるプロダクト F_{t-1} に含まれるコード片に対して，F_t に含まれるコード片への写像を定義している．本写像はクローン履歴分析だけでなく，後述のクローン関係者抽出手法でも利用しているため，その概要についてここで述べる．

もし a よりも前の部分に変更箇所があれば，その内容に応じて写像先 b の開始行，終了行を調整する．図 3 の Case 1 はコード片 a の前で編集操作が行われた場合の例である．このとき，a の前で行われた編集操作の全てを勘案して a の写像先 b の開始行，終了行は a のそれのそれぞれ 4 行後ろとする．また a に含まれる部分に変更が加えられていた場合には，その内容に応じて b の終了行を調整する．図 3 の Case 2 では a 内に 2 行新しい行が追加されているため，b の終了行は a のそれに対して 2 行追加した値とする．

最後に，クローン片の端の部分で書きかえ操作が発生した場合について述べる．図 3 の Case 3 ではコード片 a の開始行を跨がる形で書きかえされている．このような場合，「コード片 a の上に 2 行挿入があった」という解釈と，「コード片 a に 2 行の挿入がされた」という解釈，およびその中間 (1 行がコード片の上に，1 行はコード片 a 中に挿入された) が成りたつ．そこで，これら全てのケースをコード片 a の子とする．このように複数の子が生成されるケースは以下の 3 つである．

- 開始行，終了行が編集されている箇所を含む (図 3 Case 3)
- 開始行の一行前に挿入
- 終了行に挿入

本研究においては $F_{t-1} \rightarrow F_t$ への写像のみを扱うため，削除操作ではこのような複数候補を考える必要はない．

以上述べた 3 つの編集操作による影響をすべて足しあわせて写像先を決定する．

3.3 クローン関係者抽出手法

3.3.1 概要

本手法では，CVS によって得られるログ情報と，各リビジョンにて何行目が編集されたかという情報を元にクローン関係者の同定を行う．

まず解析対象の各ファイルについてログ情報を取得する．ログ情報からは「リビジョンの一覧」と，各リビジョンごとの「リビジョン番号」「編集日時」「編集を行った開発者」「コミットログ」が得られる．

次に，コミットログによって得られたリビジョンの集合に対して，各リビジョン間においてどのような編集操作が行われたかを逐次的に取得する．この編集操作の中に，編集日時に存在するクローンを編集しているものがあれば，「編集を行った開発者」をそのクローンの編集者とする．差分から編集されたクローンを特定する方法については次

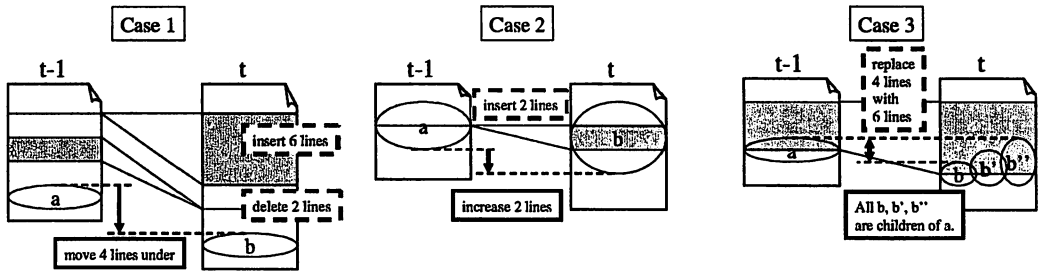

図 3: コード片の写像

節で詳述する.

上記の手続きをくりかえして，クローンが編集されたそれぞれの時点において，どの開発者がそれを行ったかを特定する.

3.3.2 差分からのクローン特定

リビジョン間の編集情報は「編集が行われた領域」と「編集内容」の繰り返しとなっている．このうち，クローン特定に用いる「編集が行われた領域」の情報は次の 3 つの情報から構成される.

(1) 編集元の領域．編集元が何行目から何行目か.
(2) 編集操作．追加，削除，編集の 3 種類.
(3) 編集先の領域．編集先が何行目から何行目か.

これらの情報から，そのコミットにおいて開発者がどの部分を編集したかがわかる．それらの領域の中にクローンが存在した場合，そのクローンはこの編集によって何らかの変更をされていると考えられる．そこで，F_{t-1} に含まれるクローンのうち，編集元の領域に存在するクローン，および F_t に含まれるクローンのうち編集先の領域に存在するクローンについて，その時点でのクローンの編集者をコミットを行った開発者とする.

ただし，行番号は $t-1, t$ 間で行なわれた他の編集内容による影響を受ける．例えば，時間 $t-1$ と t の間に 2 つのコミットがあり，それぞれ t', t'' にコミットされている場合を考える $(t-1 < t' < t'' < t)$. このとき，F_{t-1} と $F_{t'}$ ではファイルの状態が異なる．そのため，t'' における解析では F_{t-1} での行番号をそのまま用いることはできない．そこで，3.2 節で述べた行番号対応関係分析手法を t' で行なわれた編集内容に対して適用し，行番号の対応付けを行う.

ソフトウェア名	PostgreSQL
解析対象モジュール	pgsql/src/backend/commands
解析期間	2003/01/01 ～ 2004/01/01
解析対象の行数	32092 行 (2004/01/01 時点)

表 1: 実験対象の詳細

4 実験

4.1 実験の概要

ここでは著名なオープンソースソフトウェアの一つである PostgreSQL の一部ソースコードに対して本手法を適用し，そこから得られる情報とクローンの対処優先度についての関係について考察する．解析対象の詳細は表 1 に示す.

本実験では，クローン履歴分析およびクローン関係者分析を PostgreSQL に対して適用し，各開発者がクローンの編集を行った回数などを計測することで，それらの計測値とクローンの性質の間にどのような関係があるかを調査した.

実験は (1) 各開発者ごとにクローンの増減回数に差が見られるか，(2) 大規模な変更に含まれるクローンと小規模な変更に含まれるクローンの性質に差があるか，という 2 つの視点から分析を行った.

開発者ごとのクローン変更回数は，あるコミットであるクローンセットに対して何らかの変更を施していた場合を 1 回とする．つまり 1 回のコミットで 3 つのクローンセットを同時に変更していた場合には 3 回の変更をした，とカウントされる．また，クローンセットに対する変更を以下の 3 つに分類する.

クローンの追加 クローンセットに新しいクローンが追加されている.

クローンの削除 クローンセット内のクローンが減少している，もしくはクローンセット全体が削除されている.

	momjian	petere	tgl
クローン追加	6	10	32
クローン削除	2	3	17
クローン編集	35	27	135
(合計)	43	40	184

表 2: 開発者ごとのクローン変更回数

tgl	1428 (42.4%)
momjian	1317 (39.1%)
petere	143 (4.2%)
scrappy	143 (4.2%)
vadim	121 (3.6%)
その他	214 (6.3%)

表 3: 解析対象における開発者ごとのコミット回数

クローンの編集クローンの中身が編集されている．

クローンセット内のクローン数に変化はない．

なお，追加，削除時に同時に編集が行われていた場合でも，それぞれクローンの追加，クローンの削除が行われたものとする．

4.2 実験結果

4.2.1 開発者ごとのクローン追加，削除回数の比較

コミット時のクローン操作内容を開発者ごとに集計したものを表 2 に示す．

最もクローンの変更回数が多いのは tgl であるが，tgl は作業量が多いために，それだけクローンに関わる回数も多くなっている．また，実際にはコミッタではない開発協力者が書いたソースコードを tgl が責任者としてコミットしていることも，tgl の編集回数が多い一因となっている．解析対象における各開発者のコミット回数を表 3 に示す．

表 3 のコミット回数を考慮すると，petere のクローン追加の比率は他の開発者のそれに比べると高いことがわかる．実際に petere による機能追加にはコピー & ペーストが多く，petere が作成したクローンについては注意を払う必要がある可能性がある．

4.2.2 コミットごとのクローンセット数の分析

次に，一度のコミットにおいて編集されたクローンセット数に着目し，その分布について調査を行った．図 4 はクローンセット数によるヒストグラムである．X 軸はコミットによって変更したクローンセット数，Y 軸はその頻度を表している．つまり，1 つのクローンセットのみを編集しているコミットが 21 回，2 つのクローンセットを編集しているコミットが 13 回あったことを示している．

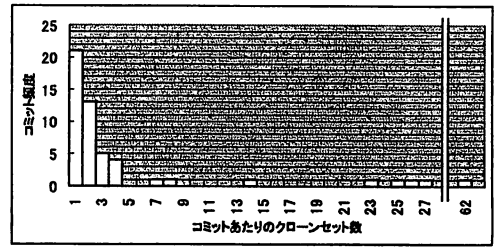

図 4: コミットによって変更されたクローンセット数の度数分布図

このグラフから，ほとんどのコミットは単一のクローンセットを編集しているが，いくつかのコミットでは非常に多数のクローンセットを編集していることがわかる．この中でも多数のクローンセットを編集している上位 5 つのコミットログを表 4 に示す．これらのログからもわかるとおりその編集内容はエラーメッセージ出力方法の統一など機械的に変更可能な箇所となっている．実際に編集内容を確認した限りでは，変更の際には文字列検索などを利用して機械的に変更箇所を把握したものと推測される．

逆に言えば，このようなクローンセットは必要に応じて機会的に網羅することが容易であり，そのまま残しておいても保守の障害とはならない，すなわちクローン削除の優先度が低いクローンセットであると言える．

5 考察

本実験では，PostgreSQL の一部モジュールを対象として，各開発者ごとにクローンの追加や削除に関して特徴的な挙動を抽出できるかどうかを試みた．その結果，ある開発者についてクローン追加の比率が高いことが判明した．実験対象のモジュールについてクローンの編集を行っている開発者が当初の予想より少ない 3 人のみだったため，この事象のみを持って断定的な主張を展開することは難しいが，開発者によってクローンに対する意識が異なる可能性は高い．

次に，コミットに含まれるクローンセット変更数に着目した結果，多数のクローンセットが単一のコミットで編集されている場合には，それらの

編集クローンセット数	コミットログ
62	Another round of error message editing, covering backend/commands/.
27	First bits of work on error message editing.
26	Adjust 'permission denied' messages to be more useful and consistent.
25	pgindent run.
23	Message editing: remove gratuitous variations in message wording, standardize terms, add some clarifications, fix some untranslatable attempts at dynamic message building.

表 4: 編集クローンセット数上位 5 件のコミットログ

クローンは自動的に網羅可能であり，ソフトウェア保守工程においてもコスト増大要因としての性格は薄いと考えられる．

ただし，そのような変更がクローンセットの極一部でのみ行われており，その他の部分は依然として，網羅することが難しい部分である可能性も否定できない．今後は，そのような部分についてもより踏みこんだ分析を行っていきたい．

6 関連研究

クローンの履歴を調査する研究としては，Kim らの研究 [5] が挙げられる．Kim らは我々の手法と同様に CCFinder を用いてクローンの履歴抽出を試みており，クローンセットの生存期間に着目して生存期間の分布がどのようになっているか，生存期間の違いによってクローンにどのような特徴があるかを調査している．Kim らの研究では，履歴抽出の対象となっているのはクローンセットであり，個々のクローンについての詳細な履歴は対象としていない．それに対して本研究では一つ一つのクローンを単位とした履歴を抽出することでより細粒度な解析をすることで，開発者などの情報を取りこむことを可能にしている．

また，吉田ら [10] は Java ソースコードを対象として，クローンが存在するクラスに着目したクローンメトリクスを定義し，それらの値からリファクタリングを支援する手法を提案している．このようなファイルの内容を解析した情報の活用もまた重要であると考えられる．

7 まとめ

本研究では，版管理システムが保持する履歴情報とクローン履歴情報に基づいて，クローン編集者情報を自動的に抽出する手法を提案した．また，クローン関係者やクローン履歴を用いてクローンを削除すべきか否かをを判定する指標として活用できるかを検討するための分析を行った．

今後の課題として，今回の分析で得られた知見を検証するためのより広汎な分析が挙げられる．特に開発者に関する分析では関わっている開発者は 3 人のみであった．これは調査対象を一部モジュールに限定しすぎたことが原因であり，より多くのソースコードに対して分析が必要である．

また，版管理システムの多くはブランチと呼ばれる機能を持っている．これはプロダクトの状態をいくつか並列に持てるようにする機能であるが，提案手法ではブランチを考慮していない．そのため，ブランチ上で行われた編集作業は全く考慮されていない．大規模なオープンソースソフトウェアではブランチを利用している開発グループも多く，提案手法でもブランチを考慮した分析を行えるよう拡張を考慮していきたい．

参考文献

[1] Baker, B. S.: A Program for Identifying Duplicated Code, *Computing Science and Statistics*, Vol. 24, pp. 49–57 (1992).

[2] CVS: . `http://www.cvshome.org/`.

[3] Johnson, J. H.: Identifying redundancy in source code using fingerprints, *Proc. Centre for Advanced Studies on Collaborative research (CASCON'93)*, Toronto, Ontario, Canada, pp. 171–183 (1993).

[4] Kamiya, T., Kusumoto, S. and Inoue, K.: 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] Kim, M., Sazawai, V., Notkin, D. and Murphy, G. C.: An Empirical Study of Code Clone Genealogies, *Proc. 10th European Software Engineering Conf. and 13th Foundations of Software Engineering (ESEC/FSE 2005)*, Lisbon, Portugal, pp. 17–21 (2005).

[6] Li, Z., Lu, S., Myagmar, S. and Zhou, Y.: CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code., *Proc. Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, pp. 289–302 (2004).

[7] Merlo, E., Antoniol, G., Penta, M. D. and Rollo, V. F.: Linear Complexity Object-Oriented Similarity for Clone Detection and Software Evolution Analyses, *Proc. 20th IEEE Int. Conf. on Software Maintenance (ICSM'04)*, Chicago, Illinois, USA, pp. 412–416 (2004).

[8] 門田暁人, 佐藤慎一, 神谷年洋, 松本健一: 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, Vol. 44, No. 8, pp. 2178–2188 (2003 年 8 月).

[9] Subversion: . `http://subversion.tigris.org/`.

[10] Yoshida, N., Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: On Refactoring Support Based on Code Clone Dependency Relation, *Proc. 11th IEEE Int. Software Metrics Symposium (METRICS2005)*, Como, Italy, p. 16 (2005).

[11] 川口真司, 松下誠, 井上克郎: 版管理システムを用いたコードクローン履歴分析手法の提案, 電子情報通信学会論文誌 D, Vol. J89-D, No. 10, pp. 2279–2287 (2006).