情報処理学会論文誌 Vol.54 No.2 883–893 (Feb. 2013)

コードクローン変更管理システムの開発と実プロジェクトへの適用

山中 裕樹[1,a)] 崔 恩瀞[1] 吉田 則裕[2] 井上 克郎[1] 佐野 建樹[3]

受付日 2012年5月12日, 採録日 2012年11月2日

概要：ソフトウェア保守における大きな問題の1つとしてコードクローンが指摘されている．コードクローンとは，ソースコード中に，互いに一致または類似した部分を持つコード片のことである．コードクローンに対する主な保守作業として，クローンセット（互いにコードクローンとなっているコード片の集合）に含まれるすべてのコード片を一貫して編集する同時修正と，クローンセットを1つのサブルーチンにまとめる集約があげられる．本研究では，コードクローンに対する保守作業を支援することを目的としたコードクローン変更管理システムの開発を行った．そして，企業で行われているソフトウェア開発に適用することによって，本システムの有用性を確かめることができた．

キーワード：コードクローン，ソフトウェア保守，変更管理

A Development of Clone Change Management System and Its Application to Actual Project

Yuki Yamanaka[1,a)] Eunjong Choi[1] Norihiro Yoshida[2] Katsuro Inoue[1] Tateki Sano[3]

Received: May 12, 2012, Accepted: November 2, 2012

Abstract: Code clone is one of the major problems for software maintenance. A code clone is a code fragment that has identical or similar portion in source code. In order to manage code clones, software developers should consider consistent modification of clone sets (i.e., a set of code clones identical or similar to each other) and merging clone set into a single function. In this study, we developed a code clone change management system for clone maintenance. Moreover, we confirmed the usefulness of the developed system by applying to industrial software development.

Keywords: code clone, software maintenance, change management

1. まえがき

ソフトウェアの保守工程における大きな問題の1つとしてコードクローンが指摘されている．コードクローンとは，ソースコード中に互いに一致または類似した部分を持つコード片のことであり，主にソースコードのコピーアンドペーストなどによって発生する [1]．コードクローンに対する主な保守作業として，同時修正と集約があげられる．同時修正とは，クローンセット（互いにコードクローンとなっているコード片の集合）に含まれるすべてのコード片を一貫して編集することである．たとえば，クローンセット中の1つのコード片に欠陥が含まれている場合，そのコード片とコードクローンになっている他のコード片も一貫した修正を行う必要が考えられる．また，集約とはクローンセットを1つのサブルーチンにまとめることである．集約を行うことによって，ソースコード中のコードクローンを削減することが可能である．

上述したコードクローンに対する保守作業を効率良く行

[1] 大阪大学
Osaka University, Suita, Osaka 565–0871, Japan
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[3] 日本電気株式会社
NEC Corporation, Minato, Tokyo 108–8001, Japan
[a)] y-yuuki@ist.osaka-u.ac.jp

うために，コードクローンの変更管理が必要である．たとえば，欠陥を含むクローンセット中の一部のコード片が編集された場合，同時修正が行われておらず，修正漏れの可能性がある．また，ソフトウェア開発企業では，システムテストに大きなコストを要した場合，コードクローンを集約し再度システムテストを行うということは，コスト面の問題から避けられることが多い．したがって，システムテスト前に発生したコードクローンを発見し，集約することで，大きなコストをかけずにコードクローンを削減することが可能となる．しかし，検出されたコードクローンの量が膨大となる場合，同時修正が行われていないクローンセットや，新たに発生したコードクローンなどの変更履歴を人手で確認することは困難である．

そこで本研究では，コード片の追加，編集，削除といった変更履歴に基づいたコードクローンの分類手法を提案する．そして，分類に基づいて，保守作業の対象となるコードクローンの情報を定期的に開発者に通知するコードクローン変更管理システムの開発を行った．本システムの利用者は，コードクローンに対して保守作業が必要であるか否かの判断を行うソフトウェア開発者である．また，評価実験として，開発したシステムを企業で行われているソフトウェア開発に約 40 日間適用した．実験の結果，本システムを用いることによって，開発者は保守作業の対象となる 11 個のクローンセットを発見することができた．

本稿の構成は次のとおりである．2 章では，本研究の背景を述べる．3 章では，コードクローンの分類手法について説明する．4 章では，本研究で開発したコードクローン変更管理システムの概要について説明する．5 章では，コードクローン変更管理システムの実プロジェクトに対する適用実験について述べる．6 章では，本研究の関連研究について述べる．7 章では，本研究のまとめと今後の課題について述べる．

2. 背景

本章では，本研究の背景として，コードクローンとその検出ツール CCFinder，および，コードクローンの変更管理の必要性について述べる．

2.1 コードクローン

コードクローン（Code clone）とは，ソースコード中に存在する，互いに一致または類似した部分を持つコード片のことである [1]．一般的に，コードクローンの存在はソフトウェアの保守を困難にするといわれている．たとえば，コードクローンとなっているコード片中に欠陥が存在する場合，そのコード片と一致または類似した他のコード片についても同様の欠陥が含まれている可能性があり，確認が必要である．

一般的に，互いに一致または類似したコードクローンの対をクローンペア（Clone pair）と呼ぶ．また，クローンペアにおいて推移関係が成り立つコードクローンの集合をクローンセット（Clone set）と呼ぶ．

2.2 コードクローン検出ツール CCFinder

ソフトウェアの規模が大きい場合，ツールを用いた自動的なコードクローン検出が行われる．本研究では，字句解析ベースの検出ツールである CCFinder [2] を用いてコードクローンの検出を行っている．CCFinder は，字句解析でソースコードをトークン列に変換し，変換処理で変数名や関数名などを同一のトークンに変換する．そして，閾値以上の長さの共通トークン列を探索し，すべてのコードクローンの対のリストを出力する．

CCFinder を用いることによって，空白やタブの有無などのコーディングスタイルを除いて完全に一致するコードクローンだけではなく，ユーザ定義名や一部の予約語のみが異なるコードクローンを検出することが可能である [3]．

2.3 コードクローンの変更管理の必要性

コードクローンに対する主な保守作業として，以下の項目があげられる．

同時修正：クローンセット中のコード片を一貫して編集すること．

集約：クローンセット中のコード片に共通するロジックを実装するサブルーチンを作り，各コード片をそのサブルーチンの呼び出し文に置き換えること．

位置情報の記録：ソースコード中にコメントとして，コードクローンとなっている他のコード片の位置を書くこと．

上述したコードクローンに対する保守作業を効率良く行うために，コードクローンの変更管理が必要である．たとえば，欠陥を含むクローンセット中の一部のコード片が修正された場合，そのクローンセットに含まれる他のコード片についても同様の修正を検討する必要がある．また，ソフトウェアのテスト終了後におけるクローンセットの集約は，不具合を生む可能性がありコストが大きい．新たに発生したコードクローンを集約の対象とすることで，不具合を生むコストを削減することが可能である．また，集約することが困難である場合，新たに発生したコードクローンは位置情報の記録の対象になる可能性がある [4]．

本研究で開発を行ったコードクローン変更管理システムでは，同時修正が行われていないクローンセットや，新たに発生したコードクローンなどの変更履歴に基づいて，コードクローンの分類を行っている．開発者はこれらの分類結果を見ることによって，保守作業が必要であるコードクローンを把握することが可能となる．たとえば，同時修正が行われていないクローンセットに含まれるコード片は，修正漏れの可能性があり，一貫した修正を行うか否か

判断する必要がある．また，新たに発生したコードクローンに対して，集約，および，位置情報の記録を行うか否か判断する必要がある．

もし検出されたコードクローンの量が膨大となる場合，その中から，同時修正が行われていないクローンセットや新たに発生したコードクローンなどの変更履歴を人手で確認することは困難である．また，ほとんどのコードクローンは短期間の開発では変更されない [5]．そのため，バージョン間のコードクローン検出結果を人手で比較することで，変更されたコードクローンのみを発見することは非効率的な作業であるといえる．本システムを用いてコードクローンの変更管理を行うことによって，保守作業の対象となるコードクローンの確認コストを削減することが可能になると考えられる．

3. コードクローンの分類手法

本研究では，保守作業の対象となるコードクローンの情報を開発者に提供するため，コード片の追加，編集，削除といった変更履歴に基づいたコードクローンの分類手法を提案する．以下に，本手法の手順を示す．入力は，分析の対象となる 2 バージョンのソースコード V_{t-1}，V_t である．V_t は最新バージョンのソースコード，また，V_{t-1} は 1 つ前のバージョンのソースコードを表す．また，V_i のソースコードに含まれるすべてのコードクローンの集合を C_i と表す．

手順 1：V_{t-1}, V_t 全体に CCFinder を適用し，コードクローン集合 C_{t-1}，C_t の検出を行う．

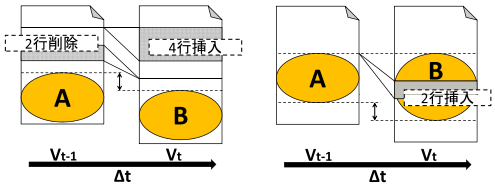
手順 2：コードクローンの変更履歴を分析するために，C_{t-1} に含まれるコードクローンと C_t に含まれるコードクローンの対応関係を求める必要がある．そこで，3.1 節で説明する定義に基づいて，コードクローンの親子関係を求める．

手順 3：3.2 節で説明する定義に基づいて，2 バージョンのソースコードに含まれるすべてのコードクローンを分類する．

手順 4：3.3 節で説明する定義に基づいて，2 バージョンのソースコードに含まれるすべてのクローンセットを分類する．

3.1 コードクローンの親子関係

コードクローンの変更履歴を分析するためには，C_{t-1} に含まれるコードクローンと C_t に含まれるコードクローンの対応関係を求める必要がある．本手法では，文献 [6] のコードクローンの履歴分析と同様の手法を用いてコードクローンの親子関係を求める．ここでは，あるコードクローン $A \in C_{t-1}$ に対応するコードクローンが $B \in C_t$ である場合，コードクローン B をコードクローン A の**子クローン**，コードクローン A をコードクローン B の**親クローン**と定義する．



(a) ケース 1　(b) ケース 2

図 1 コードクローンの親子関係

Fig. 1 Parent-child relationship of code clones.

図 1 はコードクローンの親子関係の例を示している．図 1 (a) では，コードクローン $A \in C_{t-1}$ の前で 4 行の挿入と 2 行の削除が行われているため，A に対応するコード片 B の開始行番号と終了行番号は，それぞれ A の開始行番号と終了行番号に 2 行追加した値となる．B が C_t に含まれる場合，コードクローン A の子クローンはコードクローン B となる．また，図 1 (b) では，コードクローン A の前で編集操作は行われていないため，対応するコード片 B の開始行番号は A の開始行番号と同じになる．一方，A に 2 行の挿入が行われているために，B の終了行番号は A の終了行番号に 2 行追加した値となる．B が C_t に含まれる場合，コードクローン A の子クローンはコードクローン B となる．このように，あるコードクローン $A \in C_{t-1}$ に対応するコード片をその開始行番号と終了行番号の対応に基づいて求め，そのコード片 B がコードクローンとなっている場合，コードクローン A とコードクローン B の間に親子関係を定義する[*1]．

3.2 コードクローンの分類

まず最初に，任意のコードクローン X について以下の 4 つの命題を定める．

- $P(X)$：X の親クローンが存在する．
- $C(X)$：X の子クローンが存在する．
- $M(X)$：前バージョンから X が編集されている．
- $CP(X)$：X の親クローンと X がクローンペアである．

本手法では，これらの命題を用いて 2 バージョン間のソースコード V_{t-1}，V_t に含まれるすべてのコードクローンを以下の 5 項目に分類している．

Stable Clone：$P(X) \wedge \neg M(X)$ を充足するコードクローン $X \in C_t$ とその親クローンを指す．すなわち，Stable Clone は，2 バージョン間で変更がないコードクローンを意味する．

Modified Clone：$P(X) \wedge M(X) \wedge CP(X)$ を充足するコードクローン $X \in C_t$ とその親クローンを指す．すなわち，Modified Clone は，変数名の変更などの編集がなされたが，編集後も同じクローンセットに属する

[*1] コードクローンの親子関係は 2 バージョン間での位置の対応関係を表しており，それらがクローンペアの関係にあるとは限らない．

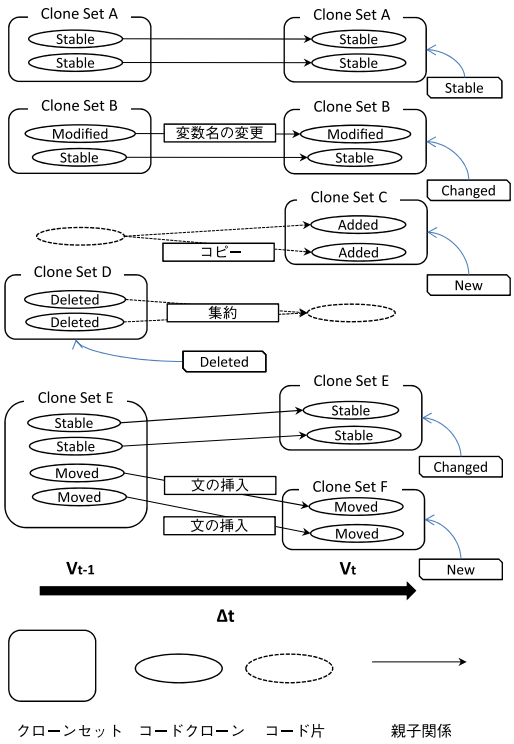


図 2 コードクローン・クローンセットの分類例
Fig. 2 Example of categorization of code clones and clone sets.

コードクローンを意味する．

Moved Clone：$P(X) \wedge M(X) \wedge \neg CP(X)$ を充足するコードクローン $X \in C_t$ とその親クローンを指す．すなわち，Moved Clone は，文の挿入など大幅な編集がなされたため，異なるクローンセットに属するようになったコードクローンを意味する．

Added Clone：$\neg P(X)$ を充足するコードクローン $X \in C_t$ を指す．すなわち，Added Clone は，コード片のコピーアンドペーストなどによって V_t で新たに発生したコードクローンを意味する．

Deleted Clone：$\neg C(X)$ を充足するコードクローン $X \in C_{t-1}$ を指す．すなわち，Deleted Clone は，クローンセットの集約やコード片の削除などによって除去されたコードクローンを意味する．

3.3 クローンセットの分類

本手法では，変更履歴に基づいて，2 バージョン間のソースコード V_{t-1}，V_t に含まれるすべてのクローンセットを以下の 4 項目に分類している（図 2[*2]）．

Stable Clone Set：図 2 のクローンセット A のように，V_{t-1}，V_t の 2 バージョンにわたって存在するクローンセットで，属するコードクローンがすべて Stable Clone に分類されるクローンセットを指す．

Changed Clone Set：V_{t-1}，V_t の 2 バージョンにわたって存在するクローンセットで，属するコードクローンに 1 つでも Stable Clone 以外のコードクローンが含まれるクローンセットを指す．たとえば，図 2 のクローンセット B は一部のコードクローンに対して変数名が変更されたため，Modified Clone に分類されている．この場合，一方の Stable Clone に対しても同様の編集が必要となる可能性がある．また，クローンセット E では，一部のコードクローンに対して文の挿入がなされたため，V_t では異なるクローンセット F を形成している．この場合，V_t 中のクローンセット E に対しても，クローンセット F と同様に文の挿入が必要となる可能性がある．このように，同時修正がなされておらず，修正漏れの可能性があるクローンセットは Changed Clone Set に分類される．

New Clone Set：V_t のみに存在するクローンセットを指す．たとえば，図 2 のクローンセット C のようにコード片のコピーアンドペーストなどによって新たに発生したクローンセットを意味する．2.3 節で述べたように，新たに発生したクローンセットは集約，あるいは，位置情報の記録の対象となる可能性がある．したがって，開発者は New Clone Set に分類されたクローンセットを確認することによって，保守作業の対象となるコードクローンを発見することが可能となる．

Deleted Clone Set：V_{t-1} のみに存在するクローンセットを指す．たとえば，図 2 のクローンセット D のように集約などによって除去されたクローンセットを意味する．したがって，開発者は Deleted Clone Set に分類されたクローンセットを確認することによって，集約が行われたクローンセットを確認することが可能となる．

4. コードクローン変更管理システム

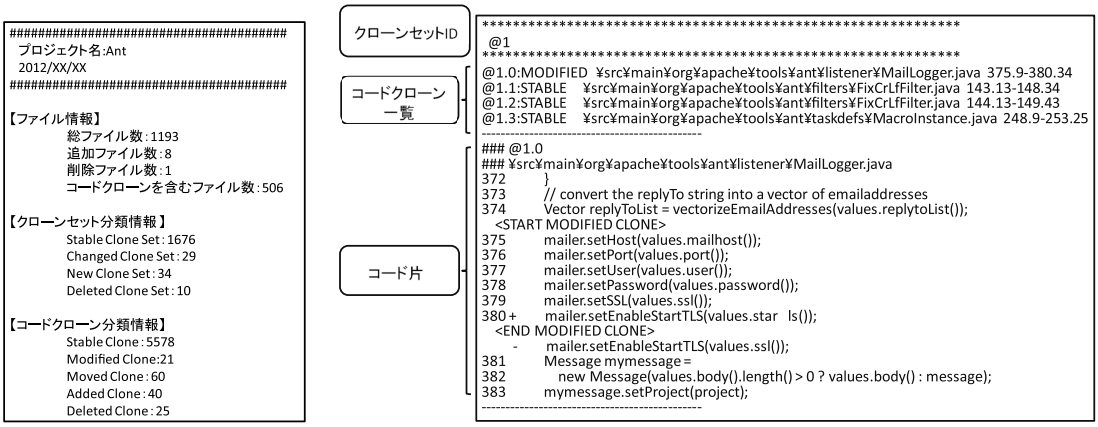
本研究では，3 章で説明したコードクローンの分類に基づいて，保守作業の対象となるコードクローンの情報を開発者に提供するコードクローン変更管理システムの開発を行った．分析結果の提供方法として，テキストベースの電子メールによる通知とウェブベースのユーザインタフェースの提供を実現している．

コードクローン変更管理システムの処理手順を以下に示す（**図 3**）．なお，本システムでは，ソフトウェア開発に CVS[*3]，Subversion[*4] などの版管理システムを用いることを想定している．

手順 1：最新バージョンのソースコード V_t を版管理システムから取得する．V_{t-1} は過去のバージョンのソースコードであるため，前回の分析時に最新バージョンと

[*2] 図 2 の V_{t-1} に含まれるコードクローンの分類に関しては，その子クローンが V_t に存在する場合，子クローンと同じ分類となる（3.2 節参照）．

[*3] http://cvs.nongnu.org/
[*4] http://subversion.apache.org/



(a) プロジェクト情報　　(a) クローンセット一覧

図 4　テキストファイルの出力例

Fig. 4　Example of text-based visualization.

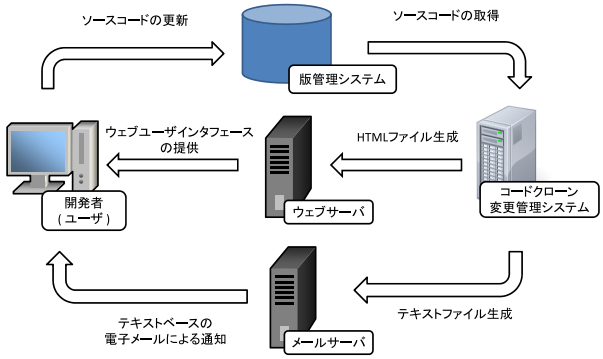


図 3　コードクローン変更管理システムの処理の流れ

Fig. 3　Process of code clone change management system.

して分析したソースコードを V_{t-1} として用いる．

手順 2：3 章で説明した手法で，変更履歴に基づいて V_{t-1} と V_t のソースコードに含まれるコードクローン，および，クローンセットの分類を行う．

手順 3：手順 2 の分類結果に基づいて，HTML ファイルと電子メールによる通知のためのテキストファイルの生成を行う．

本システムでは，手順 3 で生成したテキストファイルを添付した電子メールを送信することによって，変更されたコードクローンの存在を開発者に認識させる．そして，開発者はウェブユーザインタフェースを用いて，実際に変更されたコードクローンに対する保守作業の必要性の判断を行う．

4.1　電子メールを用いた通知

電子メールに添付されるテキストファイルの例として，Apache Ant プロジェクト[*5]のある 2 バージョン間に適用した結果を**図 4** に示す．テキストファイルには以下の情報が出力される．

- **プロジェクト情報**（図 4 (a)）
  - ファイル情報：総ファイル数，追加ファイル数，削除ファイル数，コードクローンを含むファイル数を示す．
  - クローンセット分類情報：クローンセットの各々の分類数を示す．
  - コードクローン分類情報：コードクローンの各々の分類数を示す．
- **クローンセット一覧**（図 4 (b)）
  Changed Clone Set，New Clone Set，Deleted Clone Set に分類されたクローンセットの一覧が出力される[*6]．図 4 (b) は Changed Clone Set に分類されたクローンセットの一例を示している．各々のクローンセットに関して，以下の情報を出力している．
  - クローンセット ID
  - クローンセットに属するコードクローンの一覧
  - コードクローンとなっているコード片

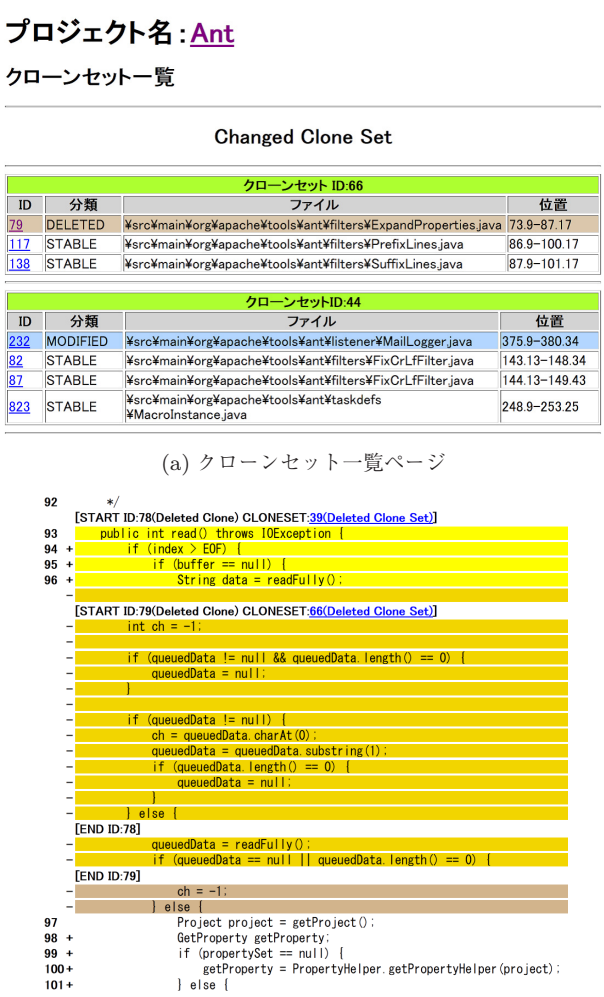
4.2　ウェブユーザインタフェースの提供

ウェブユーザインタフェースの例として，Apache Ant プロジェクトのある 2 バージョン間に適用した結果を**図 5** に示す．本稿では，クローンセット一覧ページとソースファイルページについて説明する．

- **クローンセット一覧ページ**（図 5 (a)）
  2 バージョン間に含まれるクローンセットの一覧が，Changed Clone Set，New Clone Set，Deleted Clone Set，Stable Clone Set の順で表示される．また，各々のクローンセットに関して，属するコードクローンの一覧が表示される．各々のコードクローンは，ID，分類，コードクローンが含まれるソースファイル名，

[*5] http://ant.apache.org/

[*6] Stable Clone Set は変更がないクローンセットであるため，ユーザに提供する価値は低くテキストベースの通知では省略している．



プロジェクト名 : Ant

クローンセット一覧

Changed Clone Set

クローンセット ID:66			
ID	分類	ファイル	位置
79	DELETED	¥src¥main¥org¥apache¥tools¥ant¥filters¥ExpandProperties.java	73.9–87.17
117	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥PrefixLines.java	86.9–100.17
138	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥SuffixLines.java	87.9–101.17

クローンセットID:44			
ID	分類	ファイル	位置
232	MODIFIED	¥src¥main¥org¥apache¥tools¥ant¥listener¥MailLogger.java	375.9–380.34
82	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥FixCrLfFilter.java	143.13–148.34
87	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥FixCrLfFilter.java	144.13–149.43
823	STABLE	¥src¥main¥org¥apache¥tools¥ant¥taskdefs ¥MacroInstance.java	248.9–253.25

(a) クローンセット一覧ページ

```
92        */
          [START ID:78(Deleted Clone) CLONESET:39(Deleted Clone Set)]
93        public int read() throws IOException {
94  +         if (index > EOF) {
95  +             if (buffer == null) {
96  +                 String data = readFully();
  -
          [START ID:79(Deleted Clone) CLONESET:66(Deleted Clone Set)]
  -             int ch = -1;
  -
  -             if (queuedData != null && queuedData.length() == 0) {
  -                 queuedData = null;
  -             }
  -
  -             if (queuedData != null) {
  -                 ch = queuedData.charAt(0);
  -                 queuedData = queuedData.substring(1);
  -                 if (queuedData.length() == 0) {
  -                     queuedData = null;
  -                 }
  -             } else {
          [END ID:78]
  -                 queuedData = readFully();
  -                 if (queuedData == null || queuedData.length() == 0) {
          [END ID:79]
  -                     ch = -1;
  -                 } else {
97                      Project project = getProject();
98  +                   GetProperty getProperty;
99  +                   if (propertySet == null) {
100 +                       getProperty = PropertyHelper.getPropertyHelper(project);
101 +                   } else {
```

(b) ソースファイルページ

図 5 ウェブユーザインタフェースの例

Fig. 5 Example of web-based user interface.

ソースファイル中のコードクローンの位置が表示される．Modified Clone，Moved Clone，Added Clone，Deleted Clone の 2 バージョン間で変更されたコードクローンはハイライトされ，見やすくなっている．コードクローン ID をクリックすることによって，そのコードクローンが含まれるソースファイルページへ移動する．

- **ソースファイルページ**（図 5 (b)）
  コードクローンが含まれるソースファイルが表示される．コードクローンとなっているコード片は黄色の背景色が付いている．“+” は追加行を，“-” は削除行を示し，削除行は灰色の背景色がついている．

4.3 本システムの利用例

コードクローンの変更管理を行う際の本システムの利用例を以下に示す．

(1) 電子メールによる通知により，開発プロジェクトメンバに対して，一定時間ごとに，図 4 に示すようなコードクローンの分類結果が出力されたテキストファイルが送付される．

(2) プロジェクトメンバは，電子メールに添付されたテキストファイルによって，同時修正が行われていないクローンセットや，新たに発生したコードクローンなどの変更を確認する．確認の例を以下に示す．

- Changed Clone Set に分類されたクローンセットで Stable Clone が含まれるものは，同時修正が行われておらず，不具合を含む可能性がある．たとえば，図 4 (b) では，クローンセット中の 1 個のコード片のみが編集されている．このようなクローンセットは一貫した修正が必要となる可能性がある．
- New Clone Set に分類されたクローンセットは，2.3 節で述べたように，集約，および，位置情報の記録が必要となる可能性がある．
- Deleted Clone Set に分類されたクローンセットを確認することで，計画どおりにクローンセットの集約が行われているか把握することが可能となる．

(3) 電子メールによる通知で，同時修正が行われていないクローンセットや新たに発生したコードクローンが見つかった場合，図 5 に示すようなウェブユーザインタフェースを用いて，周囲のソースコードを見ながら実際に保守作業が必要であるか否か判断する．

(4) 保守作業が必要であると判断したコードクローンに対して，実際に保守作業を行う．

5. 適用実験

実験として，企業で行われているソフトウェア開発に対してコードクローン変更管理システムの適用を行った．適用実験の目的は以下のとおりである．

- 開発者がコードクローン変更管理システムを用いて保守作業が必要となるコードクローンを発見することができたか，本システムの有用性を評価する．
- 保守作業の対象となる可能性が高いコードクローンを開発者に提示できるようにシステムを改善するため，評価実験において保守作業の対象となったコードクローンの特徴を調査する．

5.1 実プロジェクトに対する評価実験

5.1.1 実験内容

本研究では，日本電気株式会社のウェブアプリケーションソフトウェア開発を対象に評価実験を行った．対象としたソフトウェアの実装言語は Java であり，ファイル数は約 350，行数は約 12 万である．コードクローン変更管理システムを 2011/12/18 から 2012/01/31 の約 40 日間適用し，1 日ごとに電子メールによる通知とウェブユーザインタフェースの更新を行った．なお，本実験では CCFinder のトークンの閾値をデフォルトである 30 に設定した．そして，コードクローン変更管理システムの有用性を評価す

表 1 保守作業の対象となったクローンセット
Table 1 Clone sets that required additional maintenance.

	検出日	保守実施日	分類	Q1	Q2	RAD(S)	LEN(S)	RNR(S)	NIF(S)	POP(S)	DFL(S)
1	'11/12/28	'12/01/05	New	NO	集約	0	141	78	1	2	141
2	'11/12/28	'12/01/05	New	NO	集約	1	36	83	2	2	36
3	'12/01/13	-	New	NO	集約	1	48	95	7	7	288
4	'12/01/13	-	New	NO	位置情報の記録	3	54	90	2	2	54
5	'12/01/13	-	New	NO	集約	0	52	61	1	2	52
6	'12/01/13	'12/01/17	New	NO	集約	0	57	94	1	2	57
7	'12/01/13	-	New	NO	集約	0	32	93	1	2	32
8	'12/01/16	-	New	NO	集約	0	37	83	1	2	37
9	'12/01/16	-	New	NO	集約	2	32	84	2	3	64
10	'12/01/18	-	New	NO	集約	0	53	90	1	2	53
11	'12/01/24	-	New	NO	集約	0	72	86	1	3	144

るために，アンケートを実施した．アンケートの対象者は，Java 言語の経験が 10 年以上のプロジェクトマネージャであり，本プロジェクトにおいてコードクローンに対する保守作業の必要性の判断を行っている．主なアンケート内容を以下に示す．

Q1 保守作業の対象となったコードクローンの存在をすでに知っていたか

Q2 どのような保守作業を行ったか

- 集約
- 同時修正
- 位置情報の記録
- その他

本実験では，アンケートの対象者であるマネージャがコードクローン変更管理システムを用いて，各々のコードクローンの分類に対して保守作業が必要となるか否かの判断を行う．そして，実際に開発者に対して保守作業の指示を出す．

5.1.2 結果と考察

実験の結果，アンケート対象者であるマネージャは，コードクローン変更管理システムを用いて保守作業の対象となる 11 個のクローンセットを発見することができたことが分かった．**表 1** 左部は，保守作業の対象となった各々のクローンセットに対するアンケートの回答を示している[*7]．Q1 に対する回答はすべて “NO” であり，これらのクローンセットは本システムを用いて新たに発見することができたものである．

図 6 に，本アンケートで集約の対象となったクローンセットの例を示す[*8]．クローンセット 1 は，コンテンツの移動機能（コードクローン 1）とコピー機能（コードクローン 2）の 2 つのコードクローンから構成されている．これらのコードクローンは，呼び出すメソッド名のみが異なっている．また，クローンセット 2 は，最近参照されたコンテンツをリストに追加する機能（コードクローン 1）とユーザから指定されたコンテンツをリストに追加する機能（コードクローン 2）の 2 つのコードクローンから構成されている．これらのコードクローンは，変数名のみが異なっている．このような呼び出すメソッド名や変数名のみが異なっているクローンセットは，今後同時修正が必要となる可能性があり，集約の対象となると考えられる．また，集約するコストが大きい場合は位置情報の記録が必要であると考えられる [4]．

図 6 は実際に集約が行われた 2 つのクローンセットの例であるが，他のクローンセットについても，フィードバックから，呼び出すメソッド名や変数名のみが異なっているため，今後同時修正が必要となる可能性があると判断されている．したがって，マネージャによって今後保守作業において悪影響を及ぼすと判断されたため，集約，および，位置情報の記録の対象となった．

上述したように，開発者は本システムを用いて 11 個の保守作業の対象となったクローンセットを発見することができた．単一バージョンのソースコードに対してコードクローン検出を行う場合，集約が困難であったり，悪影響を及ぼさないなどの理由から集約が行われなかったクローンセットも検出される [7]．したがって，以前にも検出され，すでに保守作業が必要か否か判断されたコードクローンが多く含まれる可能性がある．**図 7** は，本実験における 1 日ごとのクローンセット数の遷移と，それぞれのクローンセットの分類数を示している．この図からも，実際に変更があったクローンセットは最大でも全体の 8.9%であり，大半が Stable Clone Set に分類されていることが分かる．Stable Clone Set に分類されたクローンセットに対して再び保守作業が必要が否かを人手で判断することは冗長になり，非効率的であると考えられる．アンケート結果から

*7 今回の適用実験では，納期が近く集約するためのコストが大きいと判断されたため，保守作業が実施されなかったクローンセットが存在する．保守実施日が記述されていないクローンセットは，'12/01/31 時点で保守作業が実施されていないものを意味している．

*8 本例では，機密情報保護のため実物のソースコードに対して変数名の変更などの整形を施している．

保守作業対象クローンセット1

コードクローン1：コンテンツの移動機能

```
for (inti  = 0; i < contents.size(); i++) {
    try {
        Content content1 = contents.get(i);
        Content content2 = (Content) content1.clone();
        content2.setTitle(StringU  l.concatPath(toPath, content1.getName()), true);
        if (content1 instanceof Page) {
              movePage((Page) content1,(Page) content2);
        } else if (content1 instanceof File) {
              moveFile((File) content1,(File) content2);
        } else if (content1 instanceof Category) {
              moveCategory((Category) content1,(Category) content2);
        }
     } catch (Excep  on e) {
         if (!(e instanceof AccessDeniedExcep  on))
               throw e;
     }
}
```

コードクローン2：コンテンツのコピー機能

```
for (inti  = 0; i < contents.size(); i++) {
    try {
        Content content1 = contents.get(i);
        Content content2 = (Content) content1.clone();
        content2.setTitle(StringU  l.concatPath(toPath, content1.getName()), true);
        if (content1 instanceof Page) {
              copyPage((Page) content1,(Page) content2);
        } else if (content1 instanceof File) {
              copyFile((File) content1,(File) content2);
        } else if (content1 instanceof Category) {
              copyCategory((Category) content1,(Category) content2);
        }
     } catch (Excep  on e) {
         if (!(e instanceof AccessDeniedExcep  on))
               throw e;
     }
}
```

保守作業対象クローンセット2

コードクローン1：最近参照されたコンテンツのリスト追加

```
if ((recentContent != null) &&(recentContent.size() >0)) {
    Element element = new Element("div");
    JDOMUtil.setAttribute(element, "class", ID.PLUGIN_RECENT);
    recentElement.addContent(recentContent);
    contents.add(recentElement);
}
```

コードクローン2：ユーザに指定されたコンテンツのリスト追加

```
if ((htmlContent != null) &&(htmlContent.size() >0)) {
    Element element = new Element("div");
    JDOMUtil.setAttribute(element, "class", ID.PLUGIN_RECENT);
    recentElement.addContent(htmlContent);
    contents.add(recentElement);
}
```

図 6　集約の対象となったクローンセットの例

Fig. 6　Example of a clone set that needs merging.

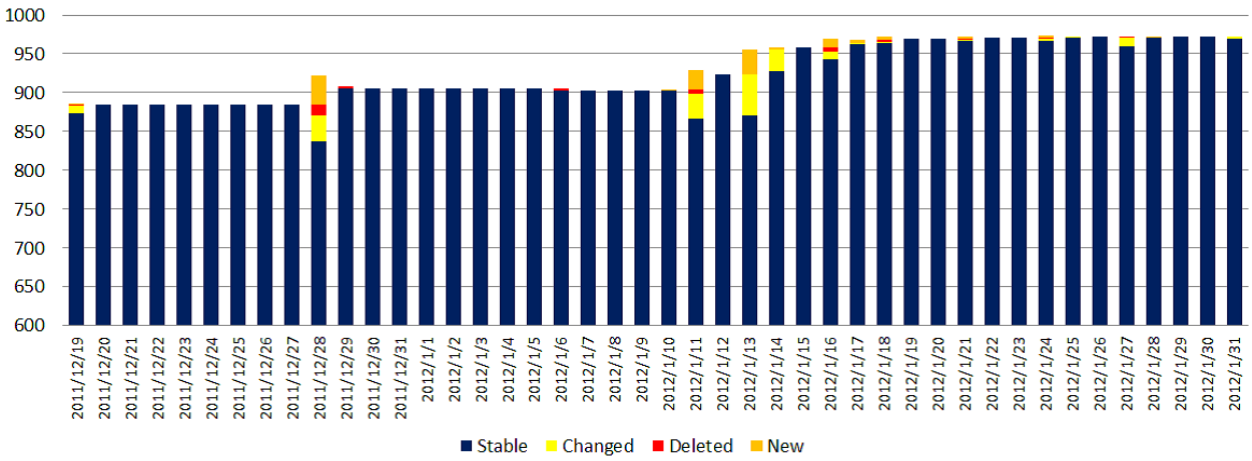


図 7　クローンセットの分類

Fig. 7　Categorization of clone sets.

も，実際に New Clone Set に分類されたクローンセットが集約の対象として判断されていることが分かる．本システムを用いて，Stable Clone Set 以外に分類された各クローンセットに対してのみ，保守作業が必要か否か判断することによって，全体のコードクローンの確認コストを削減できたと考えられる．

5.2　保守作業の対象となったコードクローンの調査

本実験では，適用期間中に 119 個のクローンセットが New Clone Set に分類されたが，実際に保守作業の対象となったのはその中の 11 個である．そこで，より保守作業の対象となる可能性が高いクローンセットを開発者に提示できるようにシステムを改善するため，保守作業の対象となったコードクローンと，保守作業の対象とはならなかったコードクローンの特徴の違いについて調査を行った．

5.2.1　調査方法

本調査では，Gemini [8], [9] を用いてクローンセットメトリクスの抽出を行った．あるクローンセット S が与えられたとき，Gemini で分析することができるクローンセットメトリクスを以下に示す．

表 2 マンホイットニー U 検定の結果
Table 2 Result of Mann-Whitney U test.

	RAD(S)	LEN(S)	RNR(S)	NIF(S)	POP(S)	DFL(S)
p 値	0.5106	0.2419	0.0368	0.5955	0.2793	0.2283
帰無仮説	採択	採択	棄却	採択	採択	採択

RAD(S)：クローンセット S 中のコード片が含まれるファイル集合が，ファイルシステムの中でディレクトリ構造的にどれだけ分散しているかを表す．

LEN(S)：クローンセット S 中のコード片のトークン数の平均値を表す．

RNR(S)：クローンセット S 中のコード片がどの程度非繰返しであるかを表す．f をクローンセット S 中のコード片，$TOC(f)$ をコード片 f を構成している字句の数，$TOCrepeated(f)$ をコード片 f を構成している字句のうち，繰返し要素の字句の数とする．このとき RNR(S) は以下の式で表される．

$$RNR(S) = \left(1 - \frac{\sum_{f \in S} TOCrepeated(f)}{\sum_{f \in S} TOC(f)}\right) \times 100$$

RNR(S) が低い場合，“ソフトウェア開発・保守を行う視点でコードクローン情報を扱う場合に特に対象とする必要がないもの” である可能性が高いことが分かっている [10]．

NIF(S)：クローンセット S 中のコードクローンを所有するファイルの数を表す．

POP(S)：クローンセット S 中のコード片単位の要素数を表す．

DFL(S)：クローンセット S 中の全コード片を集約した場合，減少が予測されるトークン数を表す．

そして，保守作業の対象のクローンセットと保守作業の対象以外のクローンセットの各々のメトリクス値の分布に対して違いがあるか否か，マンホイットニー U 検定を用いて調査した．なお，同時修正が必要であったクローンセットは確認できなかったため，集約の対象となったクローンセットについてのみ分析を行う．ここでは，“集約対象のクローンセットと集約対象以外のクローンセットの間でメトリクス値による違いがない” という帰無仮説が棄却されるか否か，有意水準 0.05 で片側検定を行った．

5.2.2 調査結果

表 1 の右部に保守作業の対象となったクローンセットの各々のメトリクス値を示す．また，**表 2** に各々のメトリクス値におけるマンホイットニー U 検定の結果を示す．

この結果，RNR(S) のみが帰無仮説が棄却され，集約対象のクローンセットと集約対象以外のクローンセットの間で違いが大きいことが分かった．**図 8** では，集約対象となったクローンセットの RNR(S) メトリック値とそれ以外の RNR(S) メトリック値の分布を比較している．この図から，RNR(S) のメトリック値が比較的大きいクローンセットが集約の対象として選ばれていることが分かる．開発者からも，変数宣言の羅列などのコードクローンが多く検出されているとフィードバックがあった．これらの結果から，RNR(S) メトリック値によるフィルタリングを行うことによって，集約の対象となる可能性が高いクローンセットを開発者に提示することが可能となると考えられる．

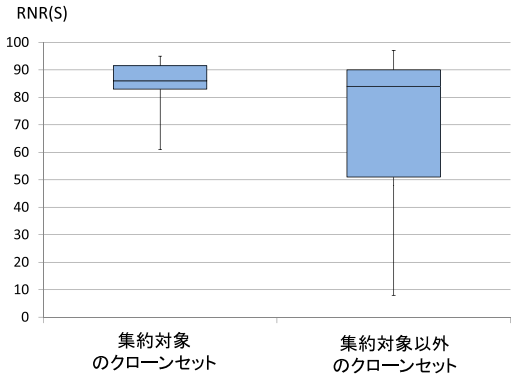


図 8 RNR (S) メトリックの比較
Fig. 8 Comparison of RNR (S) metric.

6. 関連研究

本研究と同様にコードクローンの履歴を調査する研究として，文献 [11] がある．この研究では，クローンセットの履歴に対してモデルを定義し，長期間にわたって分析することによってコードクローンの存在する期間とその特徴の関係について調査を行っている．文献 [11] では，分析の対象となっているのはクローンセットのみであるが，本研究ではコードクローンも詳細に分類を行っている．また，本研究はソフトウェア開発者への保守作業の対象となるコードクローンの情報の提供を目的としている．そのため，コードクローン変更管理システムの開発を行い，企業で行われているソフトウェア開発に適用することによって，その有用性を評価している．

また，本研究と同様に，企業のソフトウェア開発に適用した研究として文献 [10], [12] があげられる．文献 [10] では，産業界への適用と意見交換に基づいて，本研究でも利用した Gemini の改良を行っている．本研究では，ユーザである日本電気株式会社の開発者からのフィードバックに基づいて，コードクローン変更管理システムの有用性の評価，および，保守作業の対象となったコードクローンの特徴の調査を行っている．

7. まとめと今後の課題

本研究では 2 バージョン間のコード片の変更履歴に基づいたコードクローンの分類手法を提案した．そして分類結果に基づき，保守作業が必要である可能性があるコードクローンの情報を開発者に提供することを目的とした，コードクローン変更管理システムの開発を行った．さらに，実際に企業で行われているソフトウェア開発に適用し，その有用性を確かめることができた．

今後の課題として，以下が考えられる．

- 長期にわたる適用と，様々なプロジェクトに対して本システムの有用性を評価する必要がある．また，コードクローン，クローンセットの分類の妥当性も評価する必要がある．
- 今回のアンケート対象者はプロジェクトマネージャ 1 人であったが，他の開発者からのフィードバックも調査する必要がある．
- 保守作業の対象となる可能性が高いコードクローンの情報を開発者に提示できるように，システムを改善する必要がある．具体的には，RNR(S) メトリック値に基づいたフィルタリングが考えられる．

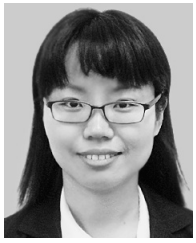
謝辞 本研究において様々なご協力をいただいた日本電気株式会社三橋二彩子氏，岩崎新一氏に深く感謝する．また，本研究は日本学術振興会科学研究費補助金基盤研究（A）（課題番号：21240002）の助成を得た．

参考文献

[1] 肥後芳樹，楠本真二，井上克郎：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，Vol.J91-D, No.6, pp.1465–1481 (2008).
[2] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol.28, No.1, pp.654–670 (2002).
[3] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and evaluation of clone detection tools, *IEEE Trans. Softw. Eng.*, Vol.31, No.10, pp.804–818 (2007).
[4] 井上克郎，楠本真二，神谷年洋：コードクローン検出法，コンピュータソフトウェア，Vol.18, No.5, pp.47–54 (2001).
[5] Krinke, J.: Is Cloned Code More Stable than Non-cloned Code?, *Proc. SCAM '08*, pp.57–66 (2008).
[6] 川口真司，松下 誠，井上克郎：版管理システムを用いたクローン履歴分析手法の提案，電子情報通信学会論文誌，Vol.J89-D, No.10, pp.2279–2287 (2006).
[7] Kapser, C. and Godfrey, M.W.: Cloning Considered Harmful Considered Harmful, *Proc. WCRE '06*, pp.19–28 (2006).
[8] Ueda, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: Gemini: Maintenance support environment based on code clone analysis, *Proc. METRICS '02*, pp.67–76 (2002).
[9] Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: Method and Implementation for Investigating Code, *Information and Software Technology*, Vol.49, No.9-10, pp.95–98 (2007).
[10] 肥後芳樹，吉田則裕，楠本真二，井上克郎：産学連携に基づいたコードクローン可視化手法の改良と実装，情報処理学会論文誌，Vol.48, No.2, pp.811–822 (2007).
[11] Kim, M., Sazawal, V., Notkin, D. and Murphy, G.C.: An empirical study of code clone genealogies, *Proc. SIGSOFT/FSE '05*, pp.187–196 (2005).
[12] 吉村健太郎，ガネサンダルマリンガム，ムーティックディルク：プロダクトライン導入に向けたレガシーソフトウェアの共通性・可変性分析法，情報処理学会論文誌，Vol.48, No.8, pp.2482–2491 (2007).

山中 裕樹

平成 24 年大阪大学基礎工学部情報科学科卒業．現在，大阪大学大学院情報科学研究科博士前期課程 1 年．コードクローン管理に関する研究に従事．



崔 恩瀞

平成 24 年大阪大学大学院情報科学研究科博士前期課程修了．現在，大阪大学大学院情報科学研究科博士後期課程 1 年．コードクローン管理やリファクタリング支援手法に関する研究に従事．

吉田 則裕 （正会員）

平成 16 年九州工業大学情報工学部知能情報工学科卒業．平成 21 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

井上 克郎 （フェロー）

昭和 59 年大阪大学大学院基礎工学研究科博士後期課程修了（工学博士）．同年大阪大学基礎工学部情報工学科助手．昭和 59～61 年ハワイ大学マノア校コンピュータサイエンス学科助教授．平成 3 年大阪大学基礎工学部助教授．平成 7 年同学部教授．平成 14 年大阪大学大学院情報科学研究科教授．平成 23 年 8 月より大阪大学大学院情報科学研究科研究科長．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．

佐野 建樹 （正会員）

平成元年日本電気株式会社入社．平成 21 年より同社ソフトウェア生産革新部シニアエキスパート．ソフトウェア開発方法論や開発支援ツールに関する業務に従事．