Vol.2013-SE-182 No.30
2013/10/25

版管理システムを用いた開発プロセスに適したコードクローン修正支援ツールの検討

藤原 雄介[1,a)]　藤原 賢二[1,b)]　吉田 則裕[1,c)]　飯田 元[1,d)]

概要：コードクローン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コード片を指す．コードクローンを変更する際に，対応する全てのコードクローンに対して一貫した修正を行うべきか検討する必要がある．コードクローンに対する修正の一貫性を確認するためには，開発者はソースコードを変更する度にコードクローン検出ツールを起動する必要があり，実装に専念する事が難しくなる．開発環境にコードクローン検出ツールを組み込み，開発者がソースコードを変更する度にコードクローン検出を実施する方法が解決策として考えられる．しかし，ソースコードの修正完了前にコードクローンに対する変更が頻繁に通知されるため，実装を妨げると考えられる．コードクローンに対する変更を通知する頻度として，版管理システムへのコミット単位が考えられる．コミット単位であれば，頻度は小さく，かつ編集作業が完了しているため，実装の妨げにならないと考えられる．本研究では，版管理システムとコードクローン検出ツールとの連携に関する研究について述べ，その後版管理システムを用いた開発プロセスに適したコードクローン修正支援ツールの要件を検討する．最後に，検討した要件を踏まえて開発中のツールを紹介する．

キーワード：コードクローン，版管理システム，開発プロセス，ソフトウェア保守

Discussion on Code Clone Modification Tool for Development Process with Version Control System

Yusuke Fujihara[1,a)]　Kenji Fujiwara[1,b)]　Norihiro Yoshida[1,c)]　Hajimu Iida[1,d)]

Abstract: A code clone is a code fragment that has identical or similar code fragments to it in the source code. If developers modify a code clone, they have to determine whether or not to modify the corresponding code clones in source code. To keep the consistency of code clones, developers run a clone detection tool when they modify one of the code clones. This prevents the developers from concentrating on coding. A considerable solution is integrating a code clone detection tool into IDE, and detecting code clones when a developer changes source code. However, if a number of clone changes are notified when a developer complete a modification, this also prevents the developers from concentrating on coding. A commit of a version control system is a considerable timing of notifying clone changes. Because it is not frequent but also means the completion of a modification, commit-level notification does not interrupt implementation. In this study, we describe existing works related with the integration of version control system and code clone detection tool, and then discuss requirements of a clone modification tool for development process with version control. Finally, we introduce a tool satisfying the requirements under development.

Keywords: Code Clone, Version Control System, Development Process, Software Maintenance

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[a)] yusuke-fuj@is.naist.jp
[b)] kenji-f@is.naist.jp
[c)] yoshida@is.naist.jp
[d)] iida@itc.naist.jp

1. はじめに

コードクローンとは，ソースコード中に存在する，互いに一致または類似したコード片を指す [1]．コードクロー

ンはコピーアンドペースト，同一の処理の実装等によって発生する．一般的にコードクローンはソフトウェアの保守を困難にすると言われている．例えば，開発者はコードクローンを変更する際に，対応する全てのコードクローンに対して一貫した修正を行うべきかを検討する必要がある．また，対応するコードクローンを見逃してしまう事により，バグが発生する原因となる可能性が高い [2]．特に，大規模なソフトウェアにおいては，対応するコードクローンを探し出す事は非常に困難な作業である．

この問題に対し，これまで数多くのコードクローン検出ツールが開発されてきた [1], [3]．コードクローンに対する修正の一貫性を確認するためには，開発者はソースコードを変更する度にツールを起動する必要があり，実装に専念する事が難しくなる．開発者が作業を行う度に自動的にツールを起動し，検出されたコードクローンの情報を提示する方法が解決方法として考えられる．例として，Kawaguchi らは統合開発環境にコードクローン検出ツールを組み込み，開発者がソースコードを変更する度にコードクローン検出を実施する SHINOBI を開発した [4]．しかし，ソースコードの修正完了前にコードクローンに対する変更が頻繁に通知されるため，実装を妨げると考えられる．

コードクローンに対する変更を開発者へ通知する頻度として，版管理システムへのコミット単位が考えられる．バグ修正，機能の追加等の変更をコミットする度にコードクローンに対する変更を通知するのであれば，頻度は小さく，かつ編集作業が完了しているため，実装の妨げにならないと考えられる．

以降 2 章では版管理システムとコードクローン検出ツールとの連携に関する関連研究について述べ，3 章では版管理システムを用いた開発プロセスに適したコードクローン修正支援ツールに求められる要件を検討する．4 章では検討した要件を踏まえて開発中のコードクローン修正支援ツールを紹介し，5 章ではまとめと今後の課題について述べる．

2. 関連技術

本章では，関連技術としてコードクローン検出ツールと版管理システムについて説明した上で，それらを用いたコードクローン修正支援ツールについて述べる．版管理システムについては，本研究で開発中のツールで用いる分散型版管理システムを主に説明する．

2.1 コードクローン検出ツール

字句解析ベースの検出ツールである CCFinder がある [1]．CCFinder は字句解析を行った後に変数名や関数名と同一のトークンとして変換する処理を行う．そのため空白やタブ，改行などのコーディングスタイルだけを除いて完全一致するコードクローンや，ユーザ定義名や一部の予約語が変更されたコードクローンを検出する事が出来る．CCFinder はソースコードであるファイル群を入力に，ソースコードの一致または類似したコード片の一部である**クローン片**の位置情報及び，互いにコードクローンであるクローン片の集合である**クローンセット**の情報を出力する．

植田らは大規模ソフトウェアにおけるコードクローンの分析を行うための Gemini を開発した [5]．Gemini は CCFinder の適用結果からコードクローンに関するメトリクス及びそのグラフ，コードクローンの分布状態を俯瞰するための散布図等を提示する．Gemini はソフトウェア保守を行う際に有益なコードクローン情報を得る事が出来るが，分析のためには常に最新のソースコードに対してツールを起動する必要がある．

2.2 版管理システム

版管理システムとはソースコードの変更履歴を管理するシステムである．版管理システムでは**リポジトリ**と呼ばれるデータベースにソースコードの履歴を保持させる．このソースコードの変更履歴の単位を**リビジョン**と呼ぶ．版管理システムは大きく二種類に分ける事ができる．CVS や Subversion に代表される 1 つのプロジェクトに対してリポジトリが単一である集中型版管理システムと，Git や Mercurial に代表される 1 つのプロジェクトに対してリポジトリが複数存在する分散型版管理システムである．

分散型版管理システムでは**リモートリポジトリ**をサーバ上に置き，開発者はそれぞれ**ローカルリポジトリ**を持つ．普段はローカルリポジトリで開発を進め，ある程度開発が進めばリモートリポジトリに反映させるという開発プロセスが版管理システムを用いた開発プロセスの例として挙げられる．

分散型版管理システムを用いた開発は以下のプロセスで行われる．

(1) リモートリポジトリの変更をローカルリポジトリに取り込む（**プル**）
(2) ソースコードに対してバグの修正，機能の追加等の変更を行う
(3) 変更をローカルリポジトリへ書き込む（**コミット**）
(4) ローカルリポジトリの変更をリモートリポジトリへ反映させる（**プッシュ**）

分散型版管理システムの利点としてローカルリポジトリの変更は共同開発者のソースコードに影響を与えないので，コミットのやり直しやリビジョンの改変といった操作も容易に行う事が出来る，リモートリポジトリにアクセス出来ない環境に於いてもローカルリポジトリを用いて開発を行う事が出来る等の点が挙げられる．

版管理システムにはリビジョンを枝分かれさせる**ブランチ**という機能がある．ブランチによって枝分かれしたリビジョンは互いに変更に影響を受ける事が無いので，バグの

修正や新しい機能を追加する等の作業を並行して行う事が出来る．枝分かれしたリビジョンは必要に応じて**マージ**という操作を行う事で双方の変更を 1 つにまとめる事が出来る．

2.3 版管理システムを用いたコードクローンの修正支援に関する研究

コードクローン修正支援ツールとは，コードクローンの追加，修正，削除等の変更に際して，一貫した変更をコードクローンに適用することを支援するツールである．本節では，版管理システムを用いたコードクローンの修正支援に関する既存研究及びツールを紹介し，各既存研究についてコードクローンの修正支援として不十分な点を述べる．

Kawaguchi らは Microsoft Visual Studio のプラグインとして，コードクローン検出機能を組み込んだ SHINOBI を開発した [4]. SHINOBI はエディタ画面のカーソル移動を検知して，カーソル近傍にあるコード片のコードクローンをリアルタイムに自動検出し，修正中のソースコードと関係の強い順に並べて提示する．

SHINOBI は開発者にコードクローンを常に意識させる事を目的として作られたツールだが，ソースコードの編集が完了する前に頻繁にクローンの検出，提示が行われる．また，コードクローンを提示するペインが常に表示されるので，作業画面を圧迫する事になる．開発者のソースコードの編集中や編集完了等の作業段階に応じて，開発に専念させる事とコードクローンを意識させる事を切り分けられない点がこのツールの不十分な点である．

山中らはコードクローンに対する一貫した修正，集約等の保守作業を支援する事を目的としてコードクローン変更管理システムの開発を行った [6]. このシステムは一定時間ごとに最新リビジョンのソースコードに対してクローン検出を行う．そして，前回のクローン検出結果と比較する事によってコードクローンを修正，追加，削除等の変更内容によって分類する．コードクローンが変更された事を開発者に認識させるため，電子メールで通知する．その上で，保守作業の必要性を判断させるためのウェブユーザインターフェースを提供する．

このシステムは版管理システムに対する操作に関係なく一定時間ごとに通知が行われ，リビジョンはその時点の最新のソースコードとして用いられる．コードクローンに対する変更が行われた時点で通知されるのが望ましいが，一定時間が経過するまで通知が行われない点がこのシステムの不十分な点である．

Nguyen らは Eclipse で SVN のリポジトリにアクセスするプラグインである Subclipse のアドオンとして，構成管理システム Clever を開発した [7]．Clever はクローンの検出，リポジトリ間のコードクローンの変化を提示する．また，コミット時の一貫していないクローンの修正の通知，一貫した修正の方法及び集約の方法を提示する．

Clever は集中型版管理システムである Subversion に対するツールなので，分散型版管理システムにおけるローカルリポジトリへコミットを行った時の一貫していないクローンの修正の通知は行う事が出来ない．

3. ツールに求められる要件の考察

本章では，2.3 節を踏まえたコードクローン修正支援ツールに求められる要件を考察する．

3.1 開発者に通知する時期及び頻度

コードクローンの検出および提示の頻度は，開発者への負担の度合いを大きく左右する．コードクローンを修正した場合にその旨の通知は出来る限り早く行うべきである．一方で，通知の頻度が高ければ，開発者に対して負担がかかる事になる．

以上を考慮すると，機能の追加やバグの修正等のソースコードの編集作業が終了した時点が望ましいと考える．版管理システムへのコミットはこれらの作業ごとに行うので，この時点でコードクローンに対する変更が通知されれば開発者の負担を最小限に抑え，かつコードクローンの一貫性を保つ事が出来ると考えられる．特に分散型版管理システムを用いて開発を行うのならば，ローカルリポジトリへのコミットは共同開発者のソースコードに影響を与えない．そのため，クローンの一貫した修正が行われるまではローカルリポジトリで開発を進め，一貫した修正が完了した時点でリモートリポジトリに反映させるといったクローンの修正プロセスを支援出来る．**分散型版管理システムにおけるローカルリポジトリへのコミット時点で通知する事が求められる (要件 I-A)**.

版管理システムを用いて開発を行うならば，コミットに限らず新しいリビジョンが作成され，クローンに対して一貫していない修正が発生する可能性がある．**マージ，リベース等の新しいリビジョンが作成される操作が行われた時点でも通知する事が求められる (要件 I-B)**. 例としてブランチ A とブランチ B に分岐する前にコードクローンがあるとする．ブランチ A でのコミットでそのコードクローンが 1 つ増えた場合，ブランチ B コードクローンに対して一貫した修正を保っていたとしても，マージを行えば増えたクローンに対しては一貫した修正がなされていない状態になる．

3.2 提示するコードクローンの情報

コードクローンに対する追加，修正，削除等の変更を検出して開発者に提示するべきである．ソフトウェアの規模が大きくなるにつれて，検出されるコードクローンの数も大きくなる．開発者が関心のあるコードクローンを検出された全てのコードクローンの中から探し出す事は開発者の

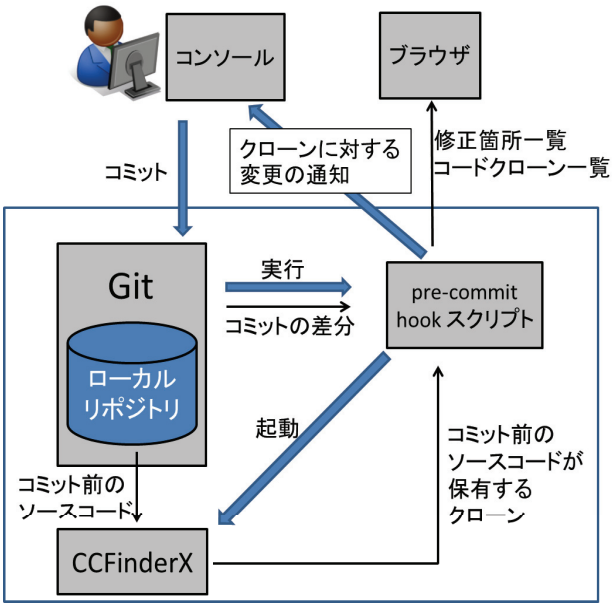

図 1 開発中のツールの概要
Fig. 1 Overview of the tool under development

```
Following diff modify clone piece!

 try{
-  cockpitServers["PPE"]=Config.GetParameter(c_iqmFile,
-       "PPE","cockpitServer",this.c_ppeCockpitServer);
-  cockpitPorts["PPE"]=Config.GetIntParameter(
-       c_iqmFile,"PPE","cockpitPort",this.c_ppeCockpitPort);
-  collectionDirs["PPE"]=Config.GetParameter(c_iqmFile,
-       "PPE","collectionDir",this.c_ppeCollectionDir);
+  cockpitServers["BLU"]=Config.GetParameter(c_iqmFile,
+       "BLU","cockpitServer",this.c_ppeCockpitServer);
+  cockpitPorts["BLU"]=Config.GetIntParameter(
+       c_iqmFile,"BLU","cockpitPort",this.c_ppeCockpitPort);
+  collectionDirs["BLU"]=Config.GetParameter(c_iqmFile,
+       "BLU","collectionDir",this.c_ppeCollectionDir);
    if ( Config.GetIntParameter(
-         c_iqmFile,"PPE","rankDataAvailable",
+         c_iqmFile,"BLU","rankDataAvailable",
          c_ppeRankDataAvailable ) == 1 ){
      m_numCollections++;

(中略)

詳細の HTML を閲覧しますか？ [yes/no]
```

図 2 コンソール上に表示される情報
Fig. 2 Information shown on console

負担となる．開発者の関心事は，現在行っている修正が，コードクローンへの修正かどうかであると考える．**変更が行われたコードクローンとそのコードクローンが属するクローンセット，コミットによる修正箇所を提示する事が求められる (要件 II-A)**．

開発者はコードクローンに対する変更を発見すると，対応するクローン片も同様の修正を行うか検討する必要がある．このような一貫した修正を行うべきかを検討するための情報は開発者にとって有益である．

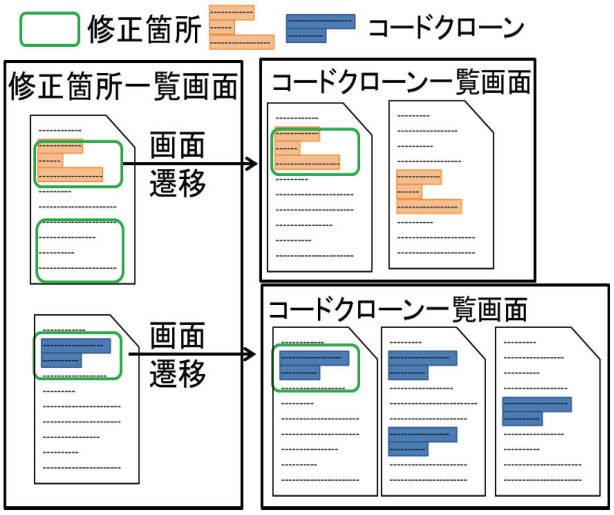

図 3 ブラウザ上において提示される画面の概要
Fig. 3 Overview of the views on screen

コミットによってクローンセット内で修正されたクローン片の数と割合を提示する事が求められる (要件 II-B)．例として，クローンセット内のクローン片が 10 個として今回のコミットでそのうち 9 個が修正されている場合，残り 1 つのクローン片に対しても同様の修正を行うべきである可能性が高い．

過去にクローンを変更した開発者の名前，時間及びコミットメッセージを提示する事が求められる (要件 II-C)．ならば，クローンの変更した理由が分かり，変更した開発者に直に相談する事も可能になると考える．

過去にクローンセットに対して一貫した修正がなされたかどうかを提示する事が求められる (要件 II-D)．過去に一貫した修正がなされたならば，そのクローンセットは引き続き一貫した修正を行うべきである可能性が高い．

3.3 開発者に提示する方法とインターフェース

コミットを行った時点で通知がすぐに見られるように，コミットを行ったツール上に通知するのが望ましいと考える．版管理システムでは，コンソールによる操作は標準で利用できる．**コンソール上にコードクローンの変更が通知される事が求められる (要件 III-A)**．ただし，コンソール上にあまりに多くの情報を表示させる事は開発者に煩わしい思いをさせる事になる．変更内容であるコミットによる修正箇所とそれがコードクローンを変更した旨のみを通知，詳細情報は別途提示するべきである．

OS などのプラットフォームに左右されない情報の提示方法として HTML や PDF が挙げられる．HTML は JavaScript によってアニメーションやインタラクティブな UI の拡張を行う事が出来る．**HTML によって詳細情報を提示する事が求められる (要件 III-B)**．

修正箇所一覧

修正したファイル	修正箇所（Unidiff形式）	修正されたクローンセット

CodeSnippet1.cs	@@ -1,22 +1,22 @@
	try{
	- cockpitServers["PPE"]=Config.GetParameter(c_iqmFile,
	- "PPE","cockpitServer",this.c_ppeCockpitServer);
	- cockpitPorts["PPE"]=Config.GetIntParameter(
	- c_iqmFile,"PPE","cockpitPort",this.c_ppeCockpitPort);
	- collectionDirs["PPE"]=Config.GetParameter(c_iqmFile,
	- "PPE","collectionDir",this.c_ppeCollectionDir);
	+ cockpitServers["BLU"]=Config.GetParameter(c_iqmFile,
	+ "BLU","cockpitServer",this.c_ppeCockpitServer);
	+ cockpitPorts["BLU"]=Config.GetIntParameter(
	+ c_iqmFile,"BLU","cockpitPort",this.c_ppeCockpitPort);
	+ collectionDirs["BLU"]=Config.GetParameter(c_iqmFile,
	+ "BLU","collectionDir",this.c_ppeCollectionDir);
	if (Config.GetIntParameter(
	- c_iqmFile,"PPE","rankDataAvailable",
	+ c_iqmFile,"BLU","rankDataAvailable",
	c_ppeRankDataAvailable) == 1){
	m_numCollections++;
	- rankDataAvailable["PPE"] = true;
		ID:1

図 4 コミットによる修正箇所一覧画面
Fig. 4 View for diffs by a commit

4. 開発中のツール

本章では，3 章で検討した要件を基に現在我々が開発中のコードクローン修正支援ツールについて説明する．

開発中のツールは分散型版管理システムである Git と連携したコードクローン修正支援ツールである．このツールは pre-commit hook スクリプトとして実装されている．hook スクリプトは特定のアクションが発生した時に実行されるスクリプトであり，pre-commit hook スクリプトはコミットが行われる直前に実行されるスクリプトである．コードクローンの検出には CCFinderX[*1]を用いる．現在，**要件 I-A，II-A，III-A，III-B** に対応していて，**要件 I-B，II-B，II-C，II-D** に関しては未実装である．

開発中のツールの概要を図 1 に示す．コンソールから Git へコミットのコマンドが実行された時点で，hook スクリプトが実行される．hook スクリプトは CCFinderX を起動しコミット前のソースコードに対してコードクローン検出を行う．検出されたコードクローンと Git から取得したコミットの差分を比較し，コードクローンが変更された箇所を発見する．コードクローンが変更された箇所が 1 つ以上あった場合，コンソールにて通知を行い，ブラウザで閲覧可能な修正箇所一覧と変更されたクローンセットに属するコードクローン一覧を HTML ファイルで生成する．

コンソールに表示される情報の例を図 2 に示す．このソースコードの変更例は，実際のプロジェクトでコードクローンの修正が行われた例である [8]．コンソール上にコードクローンが変更された旨，コミットによる修正箇所を表示した後，HTML ファイルを開くかどうかを開発者に問い合わせる．開発者が希望すれば修正箇所一覧と変更されたクローンセットに属するコードクローン一覧の HTML ファイルが自動的に開く．なお，この HTML ファイルは後からでも閲覧可能である．

ブラウザ上において提示される画面の概要を図 3 に示す．HTML ファイルを開くとまず，コミットによる修正箇所一覧画面が表示される．コミットによる修正箇所一覧画面の例を図 4 に示す．一覧は表形式で表示され，左から修正したファイルのパス，修正箇所のひとまとまり，コミットによって変更されたコードクローンのクローンセットを表示する．この修正箇所のひとまとまりは Git が提供する差分を hunk という単位で分割する機能によるものである．例では 1 つ分の hunk だけを表示しているが，1 つの hunk が 1 つのセルに表示される．修正箇所は Unidiff 形式という Git で差分を閲覧する場合に標準である形式を用いる．`@@-1,22 +1,22 @@`は変更前のファイルの 1 行目から 22 行目まで，変更後のファイルの 1 行目から 22 行目までがこの hunk で変更された範囲である事を示している．削除された行は先頭に ‘-’ が付き，赤色のフォントで表示される．追加された行は先頭に ‘+’ が付き，緑色のフォントで表示される．修正箇所によって変更されたコードクローンのクローンセットは ID で表示され，リンクになっている．このリンクをクリックするとそのクローンセットに属するコードクローンの一覧画面へジャンプする．変更されたクローンセットに属するコードクローン一覧画面を図 5 に示す．コードクローンになっている行は黒枠で囲まれて表示される．修正箇所の表示方法はコミットの修正箇所一覧画面と同様である．

5. おわりに

本研究では，版管理システムとコードクローン検出ツールとの連携に関する関連研究について述べ，各ツールの不十分な点を述べた上で版管理システムを用いた開発プロセ

[*1] http://www.ccfinder.net/ccfinderxos-j.html

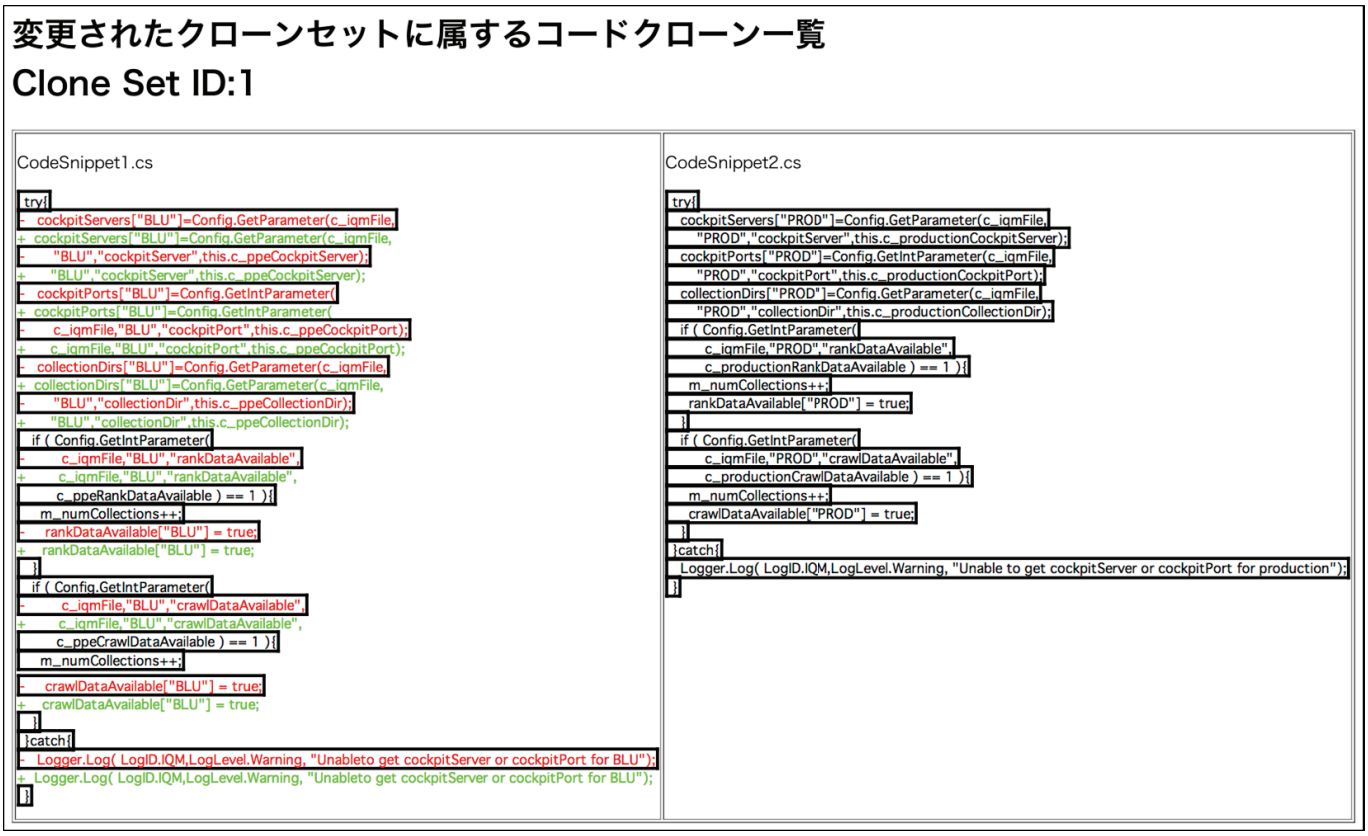

図 5 変更されたクローンセットに属するコードクローン一覧画面
Fig. 5 View for code clones in a modified clone set

スに適したコードクローン修正支援ツールに求められる要件を検討した．最後に，検討した要件を踏まえて現在我々が開発中であるコードクローン修正支援ツールを紹介した．開発中のツールは，分散型版管理システムである Git の hook スクリプトとして実装され，コミット時点でクローンに対する変更の通知及び修正箇所一覧と修正されたクローンセットに属するコードクローン一覧を提示する．

今後の課題は開発中のツールが検討した要件の内，未実装であるものを満たすように拡張する事である．未実装である要件は以下の通りである．

- マージ，リベース等の新しいリビジョンが作成される操作が行われた時点でも通知される
- コミットによってクローンセット内で修正されたクローン片の数と割合を提示する
- 過去にクローンを変更した開発者の名前，時間及びコミットメッセージを提示する
- 過去にクローンセットに対して一貫した修正がなされたかどうかを提示する

参考文献

[1] 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. 7, pp. 654–670 (2002).

[2] 西田皓司，伏田享平，川口真司，　飯田元：コードクローンに対する変更の一貫性と欠陥発生との関連性に関する分析，電子情報通信学会技術研究報告，Vol. 109, No. 456, pp. 67–72 (2010).

[3] Jiang, L., Misherghi, G., Su, Z. and Glondu, S.: DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones, *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pp. 96–105 (2007).

[4] Kawaguchi, S., Yamashina, T., Uwano, H., Fushida, K., Kamei, Y., Nagura, M. and Iida, H.: SHINOBI: A Tool for Automatic Code Clone Detection in the IDE, *Proceedings of the 16th Working Conference on Reverse Engineering*, WCRE 2009, pp. 313–314 (2009).

[5] 植田泰士，神谷年洋，楠本真二，井上克郎：開発保守支援を目指したコードクローン分析環境，電子情報通信学会論文誌 D-I，　Vol. 86, No. 12, pp. 863–871 (2003).

[6] 山中裕樹，　崔恩瀞，吉田則裕，井上克郎，佐野建樹：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，情報処理学会論文誌，　Vol. 54, pp. 883–893 (2013).

[7] Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. M. and Nguyen, T. N.: Clone-Aware Configuration Management, *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pp. 123–134 (2009).

[8] Wang, X., Dang, Y., Zhang, L., Zhang, D., Lan, E. and Mei, H.: Can I clone this piece of code here?, *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pp. 170–179 (2012).