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.

プルリクエスト型開発への統合を目的とした コードクローン修正支援システムの提案

中川　　将† 　肥後　芳樹† 　楠本　真二†

† 大阪大学大学院情報科学研究科
大阪府吹田市山田丘 1-5
E-mail: †{t-nakagw,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　コードクローン（クローン）はソフトウェアの保守作業を困難にする要因として知られている．そのため，ソフトウェア保守においてクローンに対する修正作業が重要である．既存研究では，クローンに対する効率的な修正作業を支援するために，クローンの変更情報を開発者に通知するシステムを提案している．しかし，既存システムは定期的な実行を前提としており，時間以外の外的要因に起因して実行できるようには設計されていない．そのため，既存システムはソースコード修正やブランチのマージといった開発作業に合わせた実行が容易ではない．そこで本研究では，この問題を解決するために，プルリクエスト（PR）型開発への統合を目的としたコードクローン修正支援システムを提案する．提案手法では，PR の作成時にクローンを追跡し，それらに行われた変更を分類することで，修正作業が必要なコード片を検出する．また，既存システムと比較してより正確にクローンの変更を検出できるように，既存のクローン追跡方法に三つの改善を加えた．提案システムを評価するために，三つの OSS に対してクローンが同時修正されなかった PR の割合の調査及び提案システムと既存システムのクローン追跡結果の比較を行った．結果として，11.9% ~ 30.4% の PR でクローンが同時に修正されていないことから，PR 型開発におけるクローンの修正支援に提案システムが有用であることが分かった．また，提案システムは既存システムと比較してより正確にクローンを追跡できることを確認した．

キーワード　コードクローン，ソフトウェア保守，プルリクエスト型開発

1. はじめに

コードクローン（以降，クローン）とはソースコード中に存在する他のコード片と一致もしくは類似しているコード片を指す．クローンはソフトウェア開発の保守作業における問題の一つとして指摘されている [1]．例えば，あるコード片にバグが存在した場合，そのコード片のクローンにも同様のバグが存在する可能性があるため，修正を検討しなければならない．そのため，クローンの同時修正支援やリファクタリング支援に関する研究が活発に行われている [2], [3].

これらの作業を効率化するために，既存研究ではクローン変更管理システム Clone Notifier を提案している [4]．Clone Notifier は対象プロジェクトの二つのリビジョンを入力とし，そのリビジョン間で行われたクローンの変更情報を開発者に通知する．Clone Notifier は企業での利用を対象に設計されており [5]，一日一回の定期実行を前提としている．一方，Clone Notifier は時間以外の外的要因に起因して実行できるように設計されていないため，ソースコードの修正やブランチのマージといった開発作業に合わせた実行が容易ではない．そのため，OSS 開発のような頻繁にソースコードの修正やブランチのマージが行われる開発において Clone Notifier の利用を考えた場合，以下のような課題があると著者らは考えた．

- 一つ目の課題点は，ソースコードの変更などで修正対象のコード片が発生しても，すぐには通知されない点である．Clone Notifier は定期実行されるため，一度実行されると次の実行までに一日の間隔が開いてしまう．一方，この課題を解決するために実行間隔を短く設定した場合，ソースコードの変更が行われていなくても頻繁にクローンの情報が開発者に通知されるため，開発者の負担になる可能性がある．開発作業に合わせたクローンの変更情報の通知が可能になると，ソースコードの変更などに合わせて修正すべきコード片が開発者に通知されるため，開発者はそれらに対して迅速に対応できると考えられる．
- 二つ目の課題点は，開発者に対して一度に大量の通知が行われる可能性がある点である．Clone Notifier のある実行と次の実行の間にソースコードが大量に変更され，多くのクローンが不適切に変更された結果，修正対象のコード片が大量に発生した場合を考える．この場合，修正対象のコード片は，Clone Notifier の次の実行時に一度に通知される．したがって，開発者は一度に大量の出力結果を確認しなければならず，見落としが発生する可能性がある．開発作業に合わせたクローンの変更情報の通知が可能になると，大量のソースコード変更に合わせてそれぞれの変更が行われる度に修正対象のコード片が通知されるため，開発者は修正対象のコード片の見落としを防げると考えられる．

• 三つ目の課題点は，バージョンコントロールシステム（以降，VCS）を用いた開発において，修正対象のコード片が主流ブランチに入り込んでしまう可能性がある点である．このような開発では，主流ブランチはバグがなく，いつでもリリースできる状態が理想とされる．そのため，派生ブランチのマージ前に，バグを含んでいる可能性がある修正対象のコード片が存在しないか確認すべきである．しかし，Clone Notifier は定期実行を前提としているため，ブランチのマージ前での修正対象のコード片の検出は難しい．開発作業に合わせたクローン変更情報の通知が可能になると，派生ブランチのマージ前に修正対象のコード片を検出できるため，バグが主流ブランチに入り込む可能性を減らせると考えられる．

そこで本研究では，このような課題を解決するために，開発作業への統合を目的としたクローン修正支援システム CLIONE を提案する．CLIONE はプルリクエスト（以降，PR）型開発への統合により，PR の作成毎にクローンの追跡を行い，クローンに行われた変更を分類することで修正対象のコード片を検出し，開発者に通知する．また，Clone Notifier と比較してより正確なクローン追跡のために，Clone Notifier のクローン追跡方法に三つの改善を加えた．現在，CLIONE は `https://github.com/T45K/CLIONE` にて公開している．

CLIONE の有用性を評価する実験として，三つの OSS に対して，CLIONE が開発者に通知する対象である，クローンが同時に修正されなかった PR の割合の調査と，CLIONE と Clone Notifier のクローン追跡結果の比較を行なった．実験の結果，11.9% ~ 30.4% の PR においてクローンが同時に修正されていないことから，開発者がそのような修正をブランチのマージ前に確認することに対して CLIONE が有用であることが分かった．また，Clone Notifier と比較して，CLIONE はより正確にクローンを追跡できることを確認した．

2. 研究動機

JRuby の二つのメソッドを図 1 に示す．この二つのメソッドは互いにクローンとなっており，PR#5096[1]で両方のメソッドが同時に修正されるべきところを，any_pBlockless メソッドのみが修正された．その結果，Enumerable#all メソッドを引数付きで呼び出すとエラーが発生するバグが作り込まれた．開発者はこのバグに気付かないままこの修正を master ブランチにマージしてしまい，バージョン 9.2.0.0 としてリリースしてしまった．このバグは約三ヶ月後，PR#5298[2]にて修正された．

もし開発者が CLIONE を利用していれば，ブランチのマージ前にこの非同時修正を検出できたため，このバグの master ブランチへの混入を防げたと考えられる．VCS を用いた開発では，master ブランチは常にバグがない状態が理想なので，master ブランチをバグがない状態に保つために開発作業へ統合可能なクローン修正支援が必要である．

（注1）: https://github.com/jruby/jruby/pull/5096
（注2）: https://github.com/jruby/jruby/pull/5298

core/src/main/java/org/jruby/RubyArray.java

```
   private IRubyObject all_pBlockless(ThreadContext context) {
     for (int i = 0; i < realLength; i++) {
       if (!eltOk(i).isTrue()) return context.runtime.getFalse();
     }

     return context.runtime.getTrue();
   }

- private IRubyObject any_pBlockless(ThreadContext context) {
-   for (int i = 0; i < realLength; i++) {
-     if (eltOk(i).isTrue()) return context.runtime.getTrue();
-   }
-
-   return context.runtime.getFalse();
- }

+ private IRubyObject any_pBlockless(ThreadContext context, IRubyObject[] args) {
+   IRubyObject pattern = args.length > 0 ? args[0] : null;
+   if (pattern == null) {
+     for (int i = 0; i < realLength; i++) {
+       if (eltOk(i).isTrue()) return context.runtime.getTrue();
+     }
+   } else {
+     for (int i = 0; i < realLength; i++) {
+      if (pattern.callMethod(context, "===", eltOk(i)).isTrue())
+         return context.runtime.getTrue();
+     }
+   }
+
+   return context.runtime.getFalse();
+ }
```

図 1: JRuby の二つのメソッド

3. 準備

3.1 コードクローン

クローンとは，ソースコード中の他のコード片と一致または類似しているコード片を指す．全てのコード片の組が一致または類似しているコード片の集合はクローンセットと呼ばれる．

クローンは，一般的にその類似度に基づいて以下の三種類に分類される．

Type-1　空白やコメントを除いて完全に一致するクローン
Type-2　変数名や関数名などが異なるクローン
Type-3　文の追加，削除や変更が行われたクローン

これまでに数多くのクローン検出手法が提案されている [6]．本研究と関連する二つのクローン検出手法について説明する．

テキストベースの検出手法

テキストベースのクローン検出手法では，初めに変数の正規化やソースコードの整形といった一定のルールに従ってソースコードを変形し，変形されたソースコード中のメソッドやブロックといったコード片の比較によりクローンを検出する．代表的なツールとして NiCad [7] が挙げられる．NiCad は TXL [8] を用いてソースコードの整形を行い，最長共通部分列検出アルゴリズムを用いて Type-3 クローンを検出する．

トークンベースの検出手法

トークンベースのクローン検出手法では，初めにソースコードに対して字句解析を行い，得られた字句情報を用いてクローンを検出する．代表的なツールとして，CCFinder [9] と SourcererCC [10] が挙げられる．CCFinder は字句列の比較により Type-2 クローンを検出する．SourcererCC はトークンの種類やその出現回数の類似度から Type-3 クローンを検出する．

3.2 Clone Notifier

Tokui らは開発者に対するクローン修正支援のために Clone Notifier を開発した [4]．Clone Notifier は対象プロジェクトの二つのリビジョンを入力として受け取り，そのリビジョン間で行われたソースコード変更に基づいてクローンセットを *Stable*，*New*，*Deleted*，*Changed* の四種類に分類する．

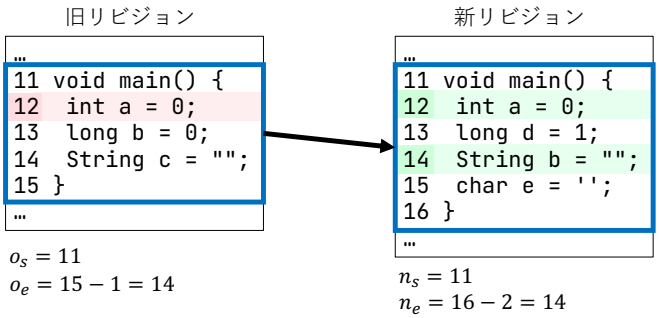


図 2: クローン追跡の例

Clone Notifier は，初めに既存のクローン検出器 [9]〜[11] を用いて対象プロジェクトの各リビジョンからクローンを検出する．次に，既存研究 [12] で提案された位置重複関数を用いて検出されたクローンを追跡し対応づける．

位置重複関数とは，二つのコード片がどの程度位置的に重複しているかを計算する関数である．Clone Notifier は対象の二つのリビジョンで同名ファイル中のクローンに対して位置重複関数を計算する．旧リビジョンから検出されたクローンを $c1$，新リビジョンから検出されたクローンを $c2$ とした時，位置重複関数 $LO(c1, c2)$ の計算式は以下の通りである（$0 \leq LO(c1, c2) \leq 1$）．

$$LO(c1, c2) = \frac{min(n_e, o_e) - max(n_s, o_s)}{n_e - n_s} \quad (1)$$

ここで，o_s と o_e はそれぞれ $c1$ での削除行を除いた開始行，終了行を表し，n_s と n_e はそれぞれ $c2$ での追加行を除いた開始行，終了行を表す．$c1$ と $c2$ に対する位置重複関数の計算結果が 0.3 以上となった場合，Clone Notifier はその二つのクローンを変更前後での対応するクローンとして追跡する．

クローン追跡の例を図 2 に示す．図では，旧リビジョンと新リビジョンの main メソッドを追跡している．main メソッドは，旧リビジョンでは 1 行の削除，新リビジョンでは 2 行の追加が行われている．削除，追加行を除くと，各リビジョンの main メソッドの開始行，終了行はそれぞれ $o_s = 11$，$o_e = 15 - 1 = 14$，$n_s = 11$，$n_e = 16 - 2 = 14$ となる．したがって，位置重複関数の計算結果は 1 となるため，このメソッドは追跡される．

最後に，クローンの追跡結果に基づき各クローンセットを分類する．また，Clone Notifier は対象プロジェクトに対して一日一回の定期実行を想定して開発されている．

3.3 プルリクエスト型開発

PR 型開発とは，GitHub の機能の一つである PR を用いて行われる開発手法を指す．PR とは，あるブランチを他のブランチにマージする前に，ソースコードの変更などの開発情報を他の開発者に通知する機能である．PR 型開発では，開発者は主流ブランチではなく派生ブランチ上で作業を行う．作業が完了すると，開発者は PR を作成し変更内容を他の開発者に通知する．変更内容に問題がなければ，その派生ブランチは主流ブランチにマージされる．

PR 型開発は OSS 開発で広く採用されている [13]．また，既存のソースコード解析技術を PR 型開発に統合することでソフトウェア開発を効率化する研究が行われている [14], [15]．

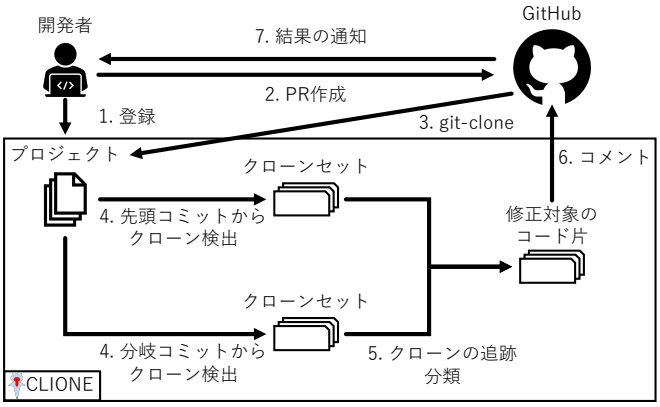


図 3: CLIONE の概要

4. 提案システム: CLIONE

本研究では，開発作業への統合を目的としたクローン修正支援システム CLIONE を提案する．CLIONE は，PR の作成時にクローンを追跡し，クローンに行われた変更を分類することで修正対象のコード片を検出する．CLIONE はサーバサイドアプリケーションとして実装されており，PR 作成時に GitHub から送信される HTTP リクエストを受け取る．

4.1 CLIONE の概要

CLIONE の概要を図 3 に示す．初めに，CLIONE を利用したい開発者は，CLIONE に対して自身の GitHub アカウントとリポジトリを登録する．この登録以降，開発者が PR を作成する度に GitHub から CLIONE に HTTP リクエストが送信され CLIONE が実行される．CLIONE は作成された PR の先頭コミットと，この PR のブランチの分岐元となったコミット（以降，分岐コミット）間のクローンを追跡する．

開発者が PR を作成すると，CLIONE は対象プロジェクトをローカル環境に git-clone する．次に，先頭コミットと分岐コミットそれぞれからクローンを検出する．クローン検出には，既存のクローン検出器である NiCad [7] と SourcererCC [10] を利用できる．これらのクローン検出器を選択した理由は，どちらも高精度で Type-3 クローンを検出できるからである．

それぞれのコミットでのクローン検出が完了すると，CLIONE は次にその PR の先頭コミットと分岐コミット間でのクローンの追跡を行う．CLIONE では，Clone Notifier と同様に，既存研究 [12] で提案された位置重複関数を用いて二コミット間のクローンを追跡する．

クローンの追跡を行った後に，クローンに行われた変更から，修正対象のコード片として非同時修正クローンセットとリファクタリング対象クローンセットを検出する．具体的には，変更されたクローンと変更されなかったクローンが含まれているクローンセットを非同時修正，全てまたは一部のクローンが新しく追加されたクローンセットをリファクタリング対象として扱う．また，Clone Notifier と比較してより正確にクローンの変更を検出できるように，CLIONE では Clone Notifier のクローン追跡方法に三つの改善を加えている．詳細については 4.2 にて述べる．

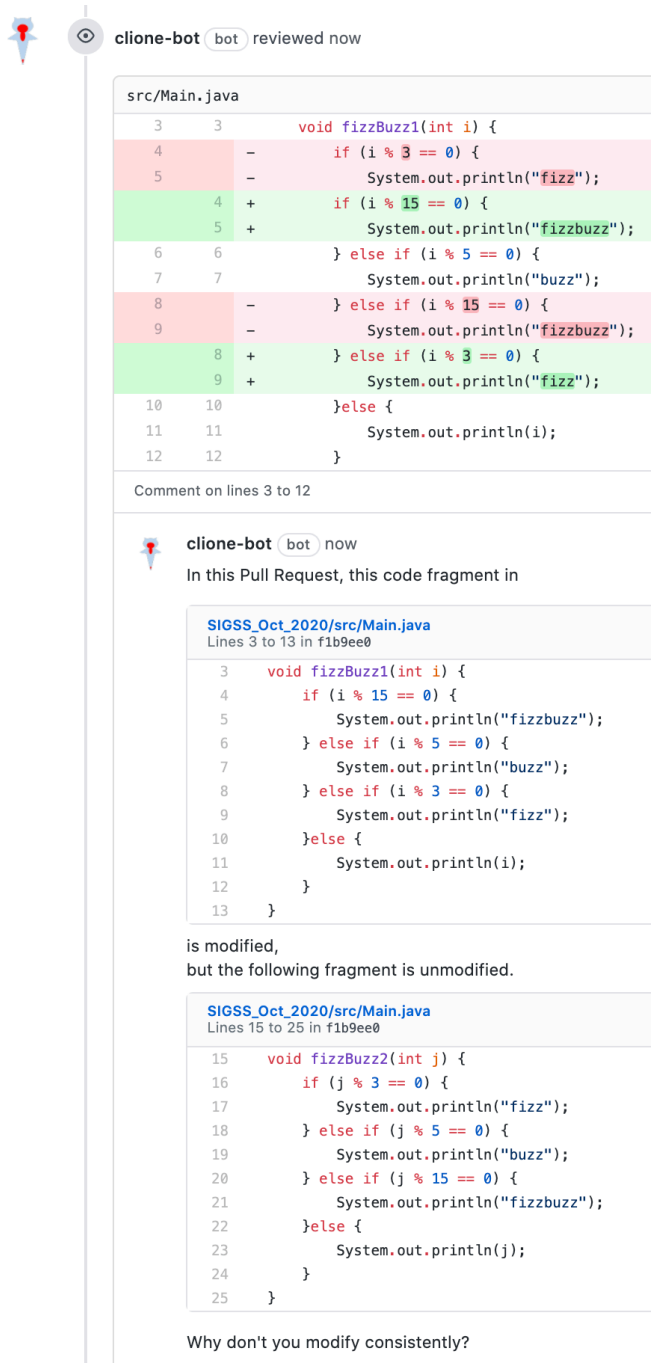
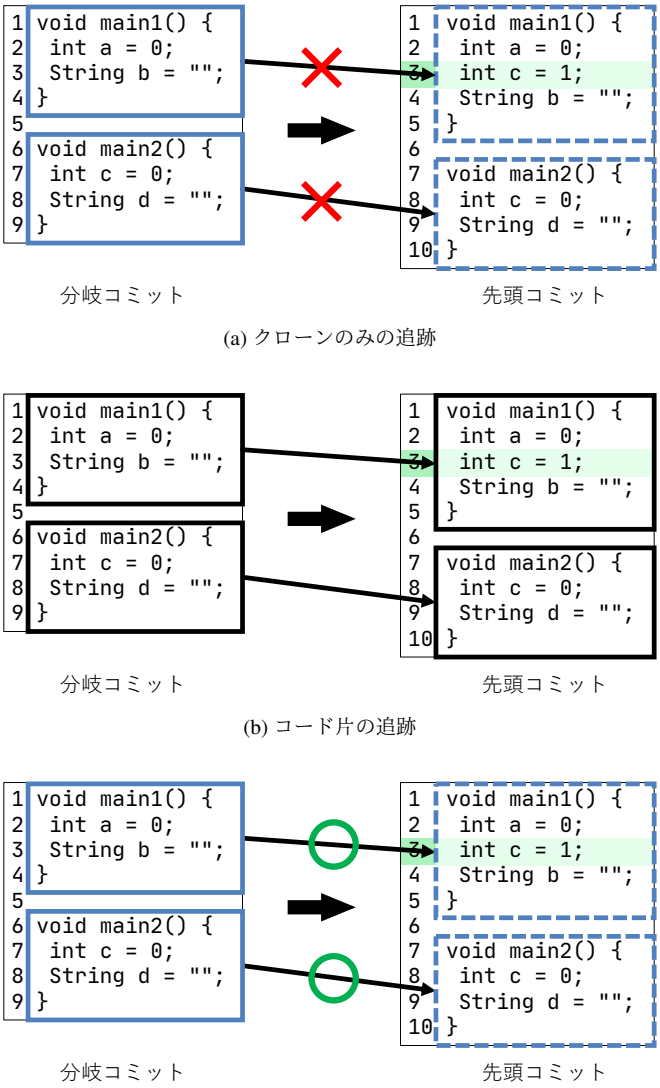


図 4: 非同時修正クローンセットに対するコメントの例

最後に，CLIONE は修正対象のコード片を PR へのコメントの形式で開発者に通知する．非同時修正クローンセットの場合，変更されたコード片と変更されなかったコード片が表示される．リファクタリング対象クローンセットの場合，クローンセットに含まれる全てのコード片が表示される．非同時修正クローンセットに対するコメントの例を図 4 に示す．図では，一番上に変更されたコード片の変更箇所が diff 形式で表示され，真ん中に変更されたコード片全体が，その下に変更されていないコード片が表示される．PR へのコメントとして修正対象のコード片を通知することで，開発者に対して迅速なフィードバックが可能になる．



(a) クローンのみの追跡

(b) コード片の追跡

(c) 正確なクローンの追跡

図 5: クローン追跡の改善

4.2 クローン追跡方法の改善

CLIONE では，Clone Notifier のクローン追跡方法に一つの手法的改善，二つの実装的改善，計三つの改善を加えた．

（1） コード片の追跡
（2） ファイル名の変更検出
（3） 字句列比較によるクローンの変更判定

コード片の追跡

手法的な改善として，クローンではなく，コード片の追跡を行うようにした．Clone Notifier はクローンとして検出されたコード片を追跡の対象として扱う．そのため，あるコード片が片方のコミットではクローンとして検出され，もう片方のコミットでクローンとして検出されなければ，Clone Notifier はそのコード片を追加あるいは削除されたと判定する．一方，クローンとなっているコード片の片方に変更が加えられた結果，変更前はクローンとして検出されても，変更後は検出されなくなる場合がある．例として，図 5(a) のように，分岐コミットではクローンだったメソッドの片方に変更が加えられた結果，先頭コミットではクローンとして検出されなくなった場合を考える．

この場合，先頭コミットでクローンが検出されなくなったため，Clone Notifier はこの二つのメソッドから構成されるクローンセットが削除されたと分類する．しかし，実際には main1 メソッドにのみ変更が加えられているため，この変更は非同時修正と分類されるべきである．

一方，CLIONE はクローンだけでなく，メソッドやブロックといったコード片も追跡の対象とする．そのため，片方のコミットでクローンが検出されなくても，クローンだったコード片さえ残っていればそのコード片を追跡できる．コード片の追跡の例を図 5(b) に示す．この例では，CLIONE はメソッド自体の追跡を行っている．したがって，メソッド自体の追跡結果から分岐コミットでクローンとなっているコード片を追跡できる（図 5(c)）．結果として，CLIONE はコード片の追跡を行うことで，このクローンを非同時修正と正しく分類できる．

ファイル名の変更検出

実装的な改善として，ファイル名の変更を検出できるようにした．Clone Notifier では，入力として受け取る二つのリビジョンにおいて，同名のファイル中に存在するクローンを追跡の対象とする．一方，ソフトウェア開発や保守において，ファイル名の変更はたびたび行われる．リビジョン間でファイル名が変更された場合，Clone Notifier は旧リビジョンでファイルが削除され，新リビジョンでファイルが追加されたと判定する．そのため，Clone Notifier はファイル名が変更された場合，ファイルの中身に変更がなくても，旧リビジョンでそのファイル中のクローンが削除され，新リビジョンでクローンが追加されたと誤って分類してしまう．

一方，CLIONE は Git のリネーム検出機能を用いることで，ファイル名の変更があった場合でも，そのファイルの内容の類似度からファイルの追跡を行える．結果として，そのファイル中のコード片を追跡できるため，クローンの誤分類を減らせる．

字句列比較によるコード片の変更判定

実装的な改善として，字句列の比較によりコード片が変更されたかどうかを判定するようにした．Clone Notifier では，クローンであるコード片内に対象のリビジョン間で追加あるいは削除された行がある場合，そのクローンは変更されたと判定される．一方，この判定方法では，フォーマットやコメントの変更といったプログラムに振る舞いを与えない変更であってもクローンが変更されたと判定される．特にフォーマットの変更は，変更がソースコード全体の広範囲に及ぶことがあるため，大量のクローンが変更されたと判定される場合がある．そのため，追加あるいは削除行の有無でクローンが変更されたかを判定すると，プログラムの振る舞いには影響を与えない変更であっても，大量に変更情報が通知される場合があるため，開発者の確認に対して負担になる可能性がある．

一方，CLIONE は追跡したコード片を字句列に変換し，その字句列の比較によりコード片が変更されたかを判定する．したがって，フォーマットやコメントの変更といった振る舞いに影響を与えない変更を無視した上で，クローンが変更されたかを判定できる．これにより，開発者に対して確認する必要性が低いクローンの変更を通知しないようにできる．

5. 評価実験

CLIONE を評価するために，以下の二つの実験を行なった．

実　験　1

実験 1 では，CLIONE が開発者に対して通知する対象である，クローンが同時に修正されなかった PR（以降，非同時修正 PR）の割合を調査した．この実験では，対象プロジェクトの PR に対して CLIONE を手動で実行し，各 PR 間でクローンが同時修正されているかを確認した．

実　験　2

実験 2 では，CLIONE と Clone Notifier のクローン追跡結果を比較した．この実験では，対象プロジェクトの PR に対して CLIONE と Clone Notifier を実行し，クローンの追跡結果を目視で確認し，結果が異なる PR の個数を数えた．

5.1 実験対象

本研究では，三つの OSS を実験対象として選択した．表 1 に実験対象の OSS の名前，2020 年 7 月 20 日時点での総 PR 数，少なくとも 1 つの Java ファイルが変更された PR（以降，対象 PR）数を示す．これらの OSS を実験対象として選んだ理由は，クローンの研究において実験対象になることが多く [16]〜[18]，かつプルリクエスト型開発が行われているからである．

5.2 実験 1 の結果

実験 1 の結果を表 2 に示す．表から，全ての対象プロジェクトにおいてクローンが同時に修正されなかった PR（以降，非同時修正 PR）が存在しており，その割合は 11.9% ~ 30.4% であることが分かる．これらの非同時修正には 2. で紹介したように，実際にバグが発生する場合もある．そのため，ブランチのマージ前に開発者は非同時修正クローンを確認し，修正の必要性を判断することが重要である．仮にこれらのプロジェクトの開発者が CLIONE を利用していた場合，PR 作成毎に非同時修正クローンが通知されるため，非同時修正クローンに容易に対応できる．以上から，CLIONE は PR 型開発におけるクローンの修正支援に有用であると考えられる．

5.3 実験 2 の結果

CLIONE と Clone Notifier のクローン追跡結果を比較した結果，合計 44 個，非同時修正 PR の 34% において，4.2 で述べたクローン追跡方法の改善によってクローン追跡結果が異なった．

表 1: 対象 OSS

名前	# 総 PR	# 対象 PR
jruby/jruby	2,248	292
junit-team/junit4	848	236
google/gson	317	69

表 2: 実験 1 の結果

名前	# 対象 PR	# 非同時修正 PR	割合
jruby/jruby	292	89	30.4%
junit-team/junit4	236	28	11.9%
google/gson	60	10	16.7%
合計	588	127	21.5%

著者らが判断した限り，異なる追跡結果全てにおいて CLIONE が正しかった．クローンの追跡結果が異なる要因となった改善方法の内訳を表 3 に示す．表 3 から，コード片の追跡によりクローンの追跡結果が異なった PR の数が最も多いことが分かる．一方，ファイル名の変更検出や字句列比較によるコード片の変更判定は，追跡結果が改善した PR の数自体は少ない．しかし，追跡結果が改善した PR において，ファイル名変更やフォーマット変更により Clone Notifier が大量のクローンの変更を誤分類したのに対して，CLIONE は正しく分類できた．つまり，これらの改善は，改善した PR の個数は少ないが，その PR において確認する必要がないと考えられる大量のクローンの追跡結果を開発者に通知しないため，開発者の負担を大きく軽減できると考えられる．

したがって，この三つの改善により，CLIONE は Clone Notifier に比べてより正確にクローンを追跡できるため，開発者により有益なクローン変更情報を提供できると考えられる．

6. おわりに

本研究では，PR 型開発への統合を目的としたクローン修正支援システム CLIONE を提案した．提案手法では，PR の作成時にその PR の先頭コミットと派生コミット間でのクローン変更により発生した修正対象のコード片を検出する．また，より正確にクローンを追跡するために，Clone Notifier が利用しているクローンの追跡方法に三つの改善を行った．評価実験として，非同時修正 PR の割合調査と，CLIONE と Clone Notifier のクローンの追跡及び変更分類結果の比較を行った．実験結果から，11.9% ~ 30.4% の PR においてクローンが同時に修正されていないことから，CLIONE が PR 型開発におけるクローンの修正支援に有用であることが分かった．また，CLIONE は Clone Notifier と比較してより正確にクローンを追跡できることを確認した．

今後は以下の課題に取り組む予定である．

被験者実験

本研究では，二つの実験から CLIONE の有用性を評価した．一方，CLIONE が開発者に対して実際に有用かどうかの評価を行なえていない．また，著者らは 1. にて，開発作業に合わせて修正対象のコード片を開発者に通知することで見落としを防げると主張しているが，実際に開発者の見落としに効果があるのかの評価も必要である．CLIONE を実開発に適用し，開発者に対する有用性を評価する実験は今後の重要な課題である．

修正方法の推薦

現在 CLIONE は，クローンの追跡によって修正対象のコード片を検出し，それらを開発者に通知するのみに止まっている．検出のみでなく，そのコード片の修正方法を推薦できれば，より有用なシステムになると考えられる．CLIONE に対するそのような機能の実装も今後の重要な課題である．

表 3: 改善方法の内訳

改善方法	# 追跡結果が異なった PR
コード片の追跡	35
ファイル名の変更検出	3
字句列比較によるコード片の変更判定	6
合計	44

謝辞　本研究は，日本学術振興科学研究費補助金基盤研究（B）（課題番号:20H04166）の助成を得て行われた．

文　献

[1] M. Mondal, C.K. Roy, and K.A. Schneider, “Bug propagation through code cloning: An empirical study,” International Conference on Software Maintenance and Evolution, pp.227–237, 2017.

[2] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue, “Simultaneous Modification Support based on Code Clone Analysis,” Asia-Pacific Software Engineering Conference, pp.262–269, 2007.

[3] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, “Refactoring Support Based on Code Clone Analysis,” Product Focused Software Process Improvement, pp.220–233, 2004.

[4] S. Tokui, N. Yoshida, E. Choi, and K. Inoue, “Clone Notifier: Developing and Improving the System to Notify Changes of Code Clones,” International Conference on Software Analysis, Evolution and Reengineering, pp.642–646, 2020.

[5] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, “Applying clone change notification system into an industrial development process,” International Conference on Program Comprehension, pp.199–206, 2013.

[6] A. Sheneamer and J.K. Kalita, “A Survey of Software Clone Detection Techniques,” International Journal of Computer Applications, pp.1–21, 2016.

[7] J.R. Cordy and C.K. Roy, “The NiCad Clone Detector,” International Conference on Program Comprehension, pp.219–220, 2011.

[8] J. Cordy, “The TXL source transformation language,” Science of Computer Programming, vol.61, pp.190–210, 08 2006.

[9] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: a multilinguistic token-based code clone detection system for large scale source code,” Transactions on Software Engineering, vol.28, no.7, pp.654–670, 2002.

[10] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C.V. Lopes, “SourcererCC: Scaling Code Clone Detection to Big-Code,” International Conference on Software Engineering, pp.1157–1168, 2016.

[11] K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, “Investigating Vector-Based Detection of Code Clones Using BigCloneBench,” Asia-Pacific Software Engineering Conference, pp.699–700, 2018.

[12] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, “An Empirical Study of Code Clone Genealogies,” International Symposium on Foundations of Software Engineering, p.187–196, 2005.

[13] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, “Wait for It: Determinants of Pull Request Evaluation Latency on GitHub,” Working Conference on Mining Software Repositories, pp.367–371, 2015.

[14] V. Alizadeh, M.A. Ouali, M. Kessentini, and M. Chater, “RefBot: Intelligent Software Refactoring Bot,” International Conference on Automated Software Engineering, pp.823–834, 2019.

[15] A. Carvalho, W. Luz, D. Marcílio, R. Bonifácio, G. Pinto, and E. Dias Canedo, “C-3PR: A Bot for Fixing Static Analysis Violations via Pull Requests,” International Conference on Software Analysis, Evolution and Reengineering, pp.161–171, 2020.

[16] T. Nakagawa, Y. Higo, J. Matsumoto, and S. Kusumoto, “How Compact Will My System Be? A Fully-Automated Way to Calculate LoC Reduced by Clone Refactoring,” Asia-Pacific Software Engineering Conference, pp.284–291, 2019.

[17] C. Ragkhitwetsagul and J. Krinke, “Using compilation/decompilation to enhance clone detection,” International Workshop on Software Clones, pp.1–7, 2017.

[18] K. Uemura, A. Mori, E. Choi, and H. Iida, “Tracking method-level clones and a case study,” International Workshop on Software Clones, pp.27–33, 2019.