論　文

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 CLIONE*

中川　将[†a)]　肥後　芳樹[†b)]　楠本　真二[†c)]

CLIONE: Code Clone Modification Support System Aimed to Integrate into Pull Request Based Development*

Tasuku NAKAGAWA[†a)], Yoshiki HIGO[†b)], and Shinji KUSUMOTO[†c)]

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 CLIONE を提案する．CLIONE は，PR の作成時にクローンを自動で対応付け，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．そして，四つの実験から CLIONE の有用性を評価した．

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，GitHub

1. まえがき

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若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集約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バージョンを入力とし，そのバージョン間で変更されたクローンの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は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 [6] 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 を利用する際，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，ソースコードを変更した結果コード片に修正が必要になっても，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．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，開発者が煩わしく感じる可能性がある．ソースコードの変更に合わせて修正すべきコード片が開発者

† 大阪大学，吹田市
Osaka University, Suita-shi, 565–0871 Japan
a) E-mail: t-nakagw@ist.osaka-u.ac.jp
b) E-mail: higo@ist.osaka-u.ac.jp
c) E-mail: kusumoto@ist.osaka-u.ac.jp
＊本論文は，システム開発論文である．
DOI:10.14923/transinfj.2020JDP7071

に通知されることにより，そのような煩わしさがなくなるとともに迅速に修正対象のコード片に対応できる．

- 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，大量のコード片に修正が必要になった場合を考える．この場合，修正対象のコード片は全て一度に通知される．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ない．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，ソースコードが変更されるたびに修正対象のコード片が通知される．
- 三つ目の課題点は，版管理システム（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点である．VCS を用いた開発では，主流ブランチではバグが発生しておらず，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の有無を確認すべきである．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に修正対象のコード片を検出することは難しい．開発作業に合わせたクローンの変更情報の通知により，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すことができる．

そこで本研究では，このような課題を解決するために，開発作業の一環であるプルリクエスト（以降，PR）に着目し，PR 型開発への統合を目的としたクローン修正支援手法を提案する(注1)．提案手法は PR の作成ごとに自動でクローンの対応付けを行い，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．なお，本研究では，提案手法が検出した修正対象のコード片が実際に修正を必要とするかはその開発者が判断することを想定している．つまり，提案手法は修正が必要と考えられるコード片をなるべく漏れなく開発者に通知するという形でクローン修正支援を行う．本研究では，提案手法を Git リポジトリホスティングサービスの一つである GitHub 向けシステム CLIONE として実装した．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ために，四つの実験を行った．実験結果から，Java ファイルが変更された PR の 16.3% においてクローンの同時修正が行われていないことから，PR 型開発において CLIONE が有用であることが分かった．次に，CLIONE の実行時間は CI ツールと比較して短かったことから，PR 型開発へ CLIONE を導入する際の障壁は低いことを示した．加えて，CLIONE を用いた開発者に対する修正の提示及び CLIONE の通知に対する開発者のアンケート結果から，CLIONE の開発者に対する有用性を確認した．

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であり，PR#5096(注2) 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（図 1 (a)）のみが修正された．その結果，all_pBlockless メソッド（図 1 (b)）を JRuby 内で呼び出す Ruby の API を実行する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主流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(注3) 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同時に修正されなかったクローン（以降，非同時修正クローン）を検出できたため，このバグの主流ブランチへの混入を防げたと考えられる．

```
- 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();
+ }
```

(a) any_pBlockless メソッド

```
  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();
  }
```

(b) all_pBlockless メソッド

図 1　JRuby の二つのメソッド
Fig. 1　Two methods in JRuby.

(注1)：本論文は，著者らの論文 [7] に対し追加の実験を行い，フルペーパーとしてまとめた論文である．

(注2)：https://github.com/jruby/jruby/pull/5096
(注3)：https://github.com/jruby/jruby/pull/5298

VCS を用いた開発では，主流ブランチは常にバグがない状態が理想である．そのため，主流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．

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．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 [8] が挙げられる．NiCad は TXL [9] を用いて整形したコード片を，最長共通部分列検出アルゴリズムを用いて比較し Type-3 クローンを検出する．

字句ベースの検出手法

字句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，SourcererCC [10] が挙げられる．SourcererCC は字句の出現頻度の類似度から 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バージョンを入力として受け取り，そのバージョン間で行われたソースコード変更に基づいて二つのバージョンそれぞれで検出された全てのクローンセットを以下の四種類に分類する．

Stable　全てのクローンがバージョン間で変更されなかったクローンセット
Changed　一部または全てのクローンがバージョン間で変更されたクローンセット
New　全てのクローンが新バージョンで追加されたクローンセット
Deleted　全てのクローンが旧バージョンから削除されたクローンセット

Clone Notifier は，初めに既存のクローン検出器 [10]〜[12] を用いて対象プロジェクトの各バージョンからクローンを検出する．次に，既存研究 [13] で提案された位置重複値を計算して検出されたクローンを対応付ける．

位置重複値とは，旧バージョンのあるコード片と新バージョンのあるコード片に対して，その二つのコード片以外の箇所が修正されなかったと仮定した上で，その二つのコード片の間で重複している行数の割合を表す．位置重複値は 0 から 1 の間の値を取る．二つのコード片の位置重複値が 1 の場合，そのコード片は二つのバージョン間で完全に同一であり，0 の場合，それらは異なるコード片だと考えられる．Clone Notifier は対象の二つのバージョンで同名ファイル中のクローンに対して位置重複値を計算する．旧バージョンから検出されたクローンを c_o，新バージョンから検出されたクローンを c_n としたとき，位置重複値は以下の関数 $LO(c_o, c_n)$ で計算される．

$$LO(c_o, c_n) = \frac{min(n_e, o_e) - max(n_s, o_s)}{n_e - n_s} \qquad (1)$$

ここで，o_s は c_o の開始行から，その行までに c_o を含むファイル中から削除された行数を引いた値を表し，o_e は c_o の終了行から，その行までに削除された行数を引いた値を表す．n_s と n_e も同様に，それぞれ c_n の開始行と終了行までに c_n を含むファイルに追加された行数を引いた値を表す．c_o と c_n に対する位置重複値が 0.3 以上となった場合，Clone Notifier はその二つのクローンを対応付ける．

クローン対応付けの例を図 2 に示す．図では，main

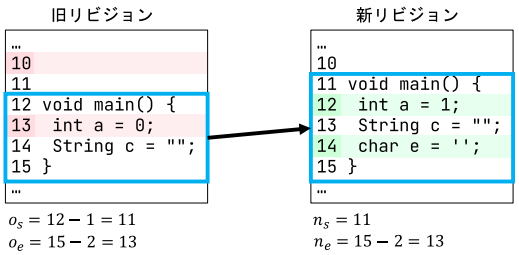

図 2　クローン対応付けの例
Fig. 2　An example of clone mapping.

メソッドを対応付けている．旧バージョンにおける main メソッドの開始行までの削除行は 1 行，終了行までの削除行は 2 行であり，新バージョンにおける main メソッドの開始行までの追加行は 0 行，終了行までの追加行は 2 行となっている．そのため，$o_s = 12-1 = 11$，$o_e = 15-2 = 13$，$n_s = 11$，$n_e = 15-2 = 13$ となる．したがって，位置重複値は 1 となるため，このメソッドは対応付けられる．

最後に，クローンの対応付け結果に基づき各クローンセットを *Stable*，*Changed*，*New*，*Deleted* の四種類に分類する．また，Clone Notifier は一日一回の定期実行を想定して開発されている [5]．

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機能追加やリファクタリング，バグ修正を行う．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 [14]．

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手法を提案する．提案手法では，開発作業の一環である PR に着目し，以下の仕組みを用いてクローン修正支援を PR 型開発へ統合する．

- PR の作成を起因として，自動で修正対象のコード片を検出する．
- 検出結果を，PR へのコメントとして開発者に通知する．

提案手法では，修正対象のコード片を検出するために，PR の二つのコミット間でクローンの対応付けを行い，クローンセットに行われた変更を分類する．本研究では，提案手法を GitHub 向けシステム CLIONE として実装した．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．

CLIONE の概要を図 3 に示す．CLIONE を利用したい開発者は，初めに CLIONE に自身の 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 を作成するたびに GitHub から CLIONE に

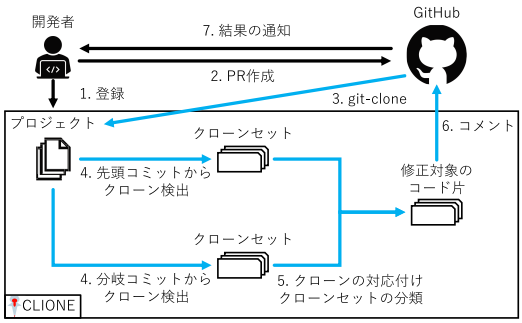

図 3　CLIONE の概要．青線は CLIONE の処理を表す
Fig. 3　An overview of CLIONE. Blue lines mean processes of CLIONE.

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）間のクローンを対応付ける．

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ダウンロード（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セットを検出する．クローン検出には，既存のクローン検出器である NiCad [8] と SourcererCC [10] が利用できる．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．

クローンセット検出が完了すると，CLIONE は修正対象のコード片を検出するために，先頭コミットと分岐コミット間でクローンの対応付けを行う．CLIONE では，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新規追加クローンセットを検出する．

非同時修正クローンセットとは，分岐コミットで検出されたクローンセットの内，先頭コミットで一部のクローンのみが変更されたクローンセットを指す．非同時修正クローンセット中の変更されなかったクローンと対応付けられる先頭コミットのコード片は修正漏れの可能性がある．そのため，CLIONE は非同時修正クローンセット中のクローンと対応付けられる先頭コミットのコード片を修正対象のコード片として検出する．以下の手順で CLIONE は分岐コミット中のそれぞれのクローンセットが非同時修正クローンセットであるかを判定する．

（1） 分岐コミットのクローンセット中のクローンを，先頭コミットのコード片（クローンがメソッド単位で検出されている場合はメソッド，ブロック単位の

場合はブロック）と対応付ける．対応付けには，Clone Notifier と同様に位置重複値 [13] を用いる．対象のクローンと先頭コミット中の全てのコード片との位置重複値を計算し，その値が 0.3 以上かつ最も高くなったコード片をそのクローンと対応付ける．

（2） 対応付けられたコード片とそのクローンの間で変更があったかを判定する．CLIONE では，クローンが変更されたかを判定するために，対応付けられたコード片とそのクローンをそれぞれ字句列に変換し，その字句列を比較する．

（3） クローンセット中に変更されたクローンと変更されなかったクローンが存在した場合，そのクローンセットを非同時修正クローンセットと判定する．

新規追加クローンセットとは，先頭コミットで検出されたクローンセットの内，一部または全てのクローンが先頭コミットで新しく追加されたクローンセットを指す．クローンはソフトウェアの保守性低下に繋がるため，なるべく早い段階で集約されることが望ましい．クローンセットが新規に追加されたタイミングは集約の良い機会の一つであると著者らは考える．そのため，CLIONE は新規追加クローンセットを修正対象のコード片として検出する．以下の手順で CLIONE は先頭コミット中のそれぞれのクローンセットが新規追加クローンセットであるかを判定する．

（1） 非同時修正クローンセット検出と同様にして，先頭コミットのクローンセット中のクローンを分岐コミット中のコード片と対応付ける．

（2） あるクローンに対して，分岐コミット中に対応付けられるコード片が存在しなかった場合，そのクローンは先頭コミットで新たに追加されたクローンと判定する．

（3） 先頭コミットで追加されたクローンを含むクローンセットを新規追加クローンセットと判定する．

また，実行時間を抑えるために，変更が加わっていないファイル中に存在するクローンは対応付けを行わない．なぜなら，そのようなクローンは対応付けを行わなくても変更されていないと判定できるからである．

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に対するコメントの例を図 4 に示す．図では，一番上（図 4 中 a）に変更されたコード片の変更箇所が diff 形式で表示され，中央（図 4 中 b）に変更されたコード片全体が，その下（図 4 中 c）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．

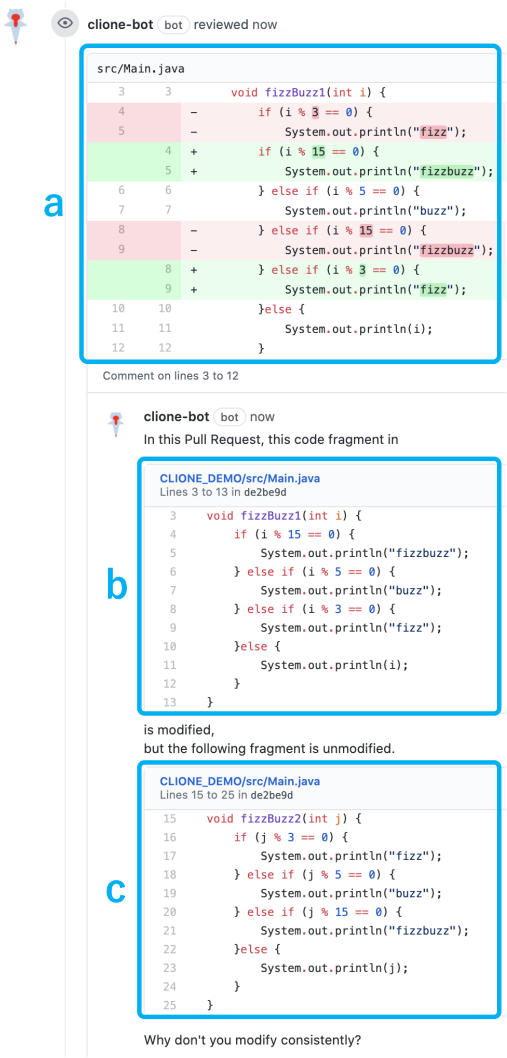

図 4 非同時修正クローンセットへのコメントの例
Fig. 4 An example of CLIONE's comment about a non-simultaneously modified clone set.

5. 評価実験

本研究では，四つの実験を通して，PR 型開発におけるクローン修正支援に対する CLIONE の有用性を評価した．

- 実験 1 では，PR 型開発への CLIONE の導入が有用かを評価する．OSS の過去の PR に対して，クローンが同時に修正されなかった PR の割合を調査

した．

- 実験 2 では，PR 型開発への CLIONE の導入が容易かを評価する．PR を作成してから CLIONE が結果を通知するまでの応答時間を測定し，CI ツールによるビルド時間と比較した．
- 実験 3 では，CLIONE の通知内容が開発者に有用であるかを評価する．CLIONE を複数の OSS に適用し，著者らがその通知内容を元にソースコードを修正しその OSS の開発者に提示した．
- 実験 4 では，同じく CLIONE の通知内容が開発者に有用であるかを評価する．CLIONE を著者らが開発している OSS に適用し，その通知内容に関して開発者へのアンケートを行った．

5.1 実　験　1

実験 1 では，PR 型開発に CLIONE を導入する有用性を評価するために，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．CLIONE は修正が必要と考えられるコード片をなるべく漏れなく開発者に通知することを目的としている．CLIONE は修正漏れの可能性がある非同時修正クローンセットが発生した PR に対して通知を行う．非同時修正 PR の割合が多ければ，CLIONE を導入することでクローンの非同時修正による修正漏れを防ぎやすくなる．そのため，CLIONE の PR 型開発への導入は有用であると考えられる．また，比較のために，クローンの同時修正が行われた PR（以降，同時修正 PR）の個数も併せて調査した(注4)．この実験では，対象プロジェクトのマージ済みの 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 1 では，三つの OSS を実験対象とした．表 1 に OSS の名前，2021 年 3 月 19 日時点での総 PR 数，Java ファイルが変更された PR（以降，対象 PR）数を示す．

表 1　対 象 OSS
Table 1　Target OSS.

名前	# 総 PR	# 対象 PR
JRuby(注5)	2,404	1,016
JUnit4	856	296
Gson	326	106

(注4)：非同時修正 PR と同時修正 PR は必ずしも排他的ではない．ある RP に非同時修正クローンセットと同時修正クローンセットが含まれていた場合，その PR は両方にカウントされる．
(注5)：JRuby はマルチモジュールプロジェクトであり，本研究では core プロジェクトのみを対象とした．

これらの OSS を実験対象とした理由は，クローンの研究において実験対象になることが多く [15]〜[17]，かつ PR 型開発が行われているからである．

実験 1 の結果を表 2 に示す．表から，全ての対象 OSS 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 9.4%〜18.8% であることが分かる．また，非同時修正 PR の個数は同時修正 PR と比較して同程度またはより多いことが分かった．次に，非同時修正 PR または同時修正 PR で発生した非同時修正クローンセットと同時修正クローンセットの個数の箱ひげ図を図 5 に示す．図から分かるように，PR 当たりの非同時修正クローンセットの個数は同時修正クローンセットと比較して多くなる傾向があることが分かった．これらの非同時修正クローンセットでは，**2.** で紹介したように実際にバグが発生する場合がある．そのため，開発者はブランチのマージ前に，非同時修正クローンセットの修正の必要性を判断することが重要である．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 の作成ごとに非同時修正クローンセットが通知されるため，非同時修正クローンセット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．

また，著者らはこの実験で検出された非同時修正 PR の幾つかに対して，実際に修正を行い開発者に提示す

表 2　実験 1 の結果
Table 2　Results of experiment 1.

名前	# 対象 PR	# 非同時修正 PR	割合	同時修正 PR
JRuby	1,016	192	18.8%	135
JUnit4	296	28	9.4%	10
Gson	106	12	11.3%	13
合計	1,418	232	16.3%	158

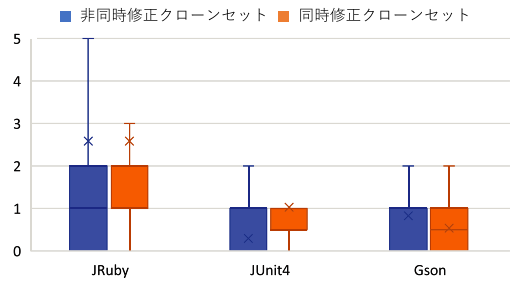

図 5　非同時修正クローンセットと同時修正クローンセットの個数の箱ひげ図
Fig. 5　A box plot of the number of non-simultaneously modified and simultaneously modified clone sets.

表 3　各 PR における処理時間
Table 3　Processing time of each PRs.

PR 番号	CLIONE の応答時間	CI ツールによるビルド時間
6424	1 分 24 秒	4 分 35 秒
6412	1 分 16 秒	14 分 35 秒
6410	1 分 18 秒	12 分 53 秒
6407	1 分 17 秒	4 分 20 秒
6406	1 分 17 秒	13 分 26 秒

ることで CLIONE の通知が有用であるかを評価する実験を行っている．実験内容の詳細及び結果は **5.3** を参照されたい．

5.2 実　験　2

実験 2 では，CLIONE が PR 型開発へ導入可能かを評価するために，PR 作成から CLIONE の通知結果が表示されるまでの応答時間を計測した(注6)．また，PR 型開発で広く採用されている CI ツール(注7)の実行時間と比較した [14]．CLIONE の応答時間が CI ツールの実行時間より短ければ，CLIONE を PR 型開発へ導入する際の障壁は低いといえる．

この実験では，実験 1 で対象にした OSS の一つである JRuby の対象 PR に対し，その PR を著者らが手作業で再現し，CLIONE の応答時間を計測した．JRuby を対象とした理由は，表 1 の三つの OSS の内，CI ツールを利用しているのが JRuby のみだからである．本来なら，JRuby の対象 PR 全てに対して実験を行うべきだが，292 件の PR 全てを手作業で再現するのは困難だと考えた．そのため，この実験では 2020 年 10 月 6 日時点でマージ済みの PR の中から，Java ファイルが一つ以上変更されている最新五つの PR を対象とした．

実験 2 の結果を表 3 に示す．表から，全ての PR において CLIONE の応答時間は CI ツールによるビルド時間の半分以下と大幅に短いことが分かる．ここで，CI ツールによるビルド時間が PR によって大きく異なるのは，JRuby では PR の変更内容によって 2 種類の異なるビルドが行われるからである．各 PR でクローンとして検出されたコード片の数とクローンセット数，及び CLIONE の応答時間の内訳を表 4 に示す．本実験ではクローン検出器として SourcererCC を利用し，ブロック単位でクローンを検出した．全ての PR 間でクローンの追加，削除，及び変更は行われなかった．この表から，CLIONE の応答時間における，クローンの対応付けや分類などの CLIONE 特有の処理が占める割合は非常に小さいことが分かる．また，git-clone が CLIONE の応答時間の大半を占めていることから，CLIONE 実行時に毎回対象プロジェクトの git-clone を行うのではなく，差分のみをダウンロード（git-pull）するように CLIONE の実装を改善すれば，高速化を見込める．

(注6)：CLIONE の実行環境は次のとおりである．CPU: Intel Xeon E5-2620, メモリ: 32GB，OS: Ubuntu 18.04
(注7)：継続的インテグレーション（Continuous Integration）ツール．PR の作成やコミットのプッシュ時に自動でビルドやテストを行う．

表 4　CLIONE の応答時間の内訳
Table 4　Breakdown of CLIONE's processing time.

PR 番号	# クローン	# クローンセット	実行時間（秒）		
			git-clone	クローン検出	その他
6424	5,664	3,174	46	34	4
6412	5,679	3,176	47	28	1
6410	5,679	3,175	46	29	3
6407	5,665	3,175	47	27	3
6406	5,173	2,925	47	26	4

加えて，CLIONE の応答時間の上限を見積もる実験を行った．**4.** で述べたように，CLIONE は変更されなかったファイル中に存在するクローンは対応付けを行わない．そのため，PR 内で変更されたファイルが多いほど，対応付け対象のクローンが増え，CLIONE の応答時間は増加する．つまり，プロジェクト中の全ての Java ファイルが変更されたとき，CLIONE の応答時間は最も大きくなると考えられる．そこで，JRuby の全ての Java ファイルの末尾に空行を追加する変更を加えた PR を作成し，CLIONE の応答時間を測定した．この PR では全部で 1,459 個の Java ファイルが変更された．この PR に対する CLIONE の応答時間は 3 分 37 秒であり，表 3 中の応答時間と比べると増加したが，CI ツールによるビルド時間よりも短いという結果になった．

以上から，CLIONE は CI ツールによるビルドよりも高速に修正対象のコード片の検出結果を通知できることから，PR 型開発へ容易に導入できると考えられる．

5.3 実　験　3

実験 3 では，開発者に対する CLIONE の有用性を評価するために，CLIONE を用いて OSS の過去の PR から非同時修正クローンを検出し，著者らが修正を行った上でその OSS の開発者に PR として提示する実験を行った．PR がマージされた場合，その OSS が CLIONE を利用していれば，PR 作成時に非同時修正クローンに対応できたといえる．そのため，CLIONE は開発者に対して有用であると考えられる．

この実験では，多様な OSS の開発者に対する CLIONE の有用性を確認するために，以下の条件を満

表 5　作成した PR 一覧
Table 5　A list of PRs we made.

名前	URL	修正内容	状態
JRuby	https://github.com/jruby/jruby/pull/6360	利用している API の変更	Merged
JUnit4	https://github.com/junit-team/junit4/pull/1671	変数の明示的なボクシング	Merged
fastjson	https://github.com/alibaba/fastjson/pull/3444	コードの移動	Merged
RxJava	https://github.com/reactivex/Rxjava/pull/7080	メソッド名の変更	Merged
Jenkins	https://github.com/jenkinsci/jenkins/pull/4945	メソッド抽出	Merged
Guava	https://github.com/google/guava/pull/4036	不要なコードの除去	Merged
Glide	https://github.com/bumptech/glide/issues/4353	コードの移動	Merged
libGDX	https://github.com/libgdx/libgdx/pull/6201	エラーメッセージの修正	Merged
checkstyle	https://github.com/checkstyle/checkstyle/pull/8872	メソッド抽出	Merged
druid	https://github.com/alibaba/druid/pull/3956	利用している API の変更	Merged
JRuby	https://github.com/jruby/jruby/pull/6361	不要なコードの除去	Closed
Druid	https://github.com/apache/druid/pull/10414	事前条件の追加	Open
Zuul	https://github.com/Netflix/zuul/pull/900	不要なコードの除去	Open
Jacoco	https://github.com/jacoco/jacoco/pull/1099	コードの移動	Open
Jackson	https://github.com/fasterXML/jackson-core/pull/642	if 文の追加	Open

たす OSS から非同時修正クローンを検出した．

- GitHub 上で PR 型開発が行われている
- Java で記述されている
- GitHub のスター(注8)数が 5,000 以上である

Java で記述されている OSS を選択した理由は，著者らが Java に慣れ親しんでおり，検出された非同時修正クローンを修正するのが容易だったからである．最終的に 14 個の OSS から 15 件の PR を作成した．

作成した 15 件の PR 及び 2020 年 11 月 3 日時点での状態の一覧を表 5 に示す．作成した PR の内 10 件がマージ，1 件がクローズ，4 件がオープン状態となった．マージされた 10 件の PR に関しては，その OSS の開発者が CLIONE を利用していれば，該当する PR を作成した時点で非同時修正クローンに対応できたと考えられる．一方，1 件の PR がクローズされた．この PR では，著者らは CLIONE が検出した非同時修正クローンを修正が必要だと考え修正したが，開発者は修正は不要だと判断した．

以上から，開発者が修正の要否を判断する必要はあるが，CLIONE が検出する非同時修正クローンの情報は開発者に対して有用であると考えられる．

5.4 実　験　4

実験 4 では，開発者に対する CLIONE の有用性を評価するために，CLIONE の通知内容に関して開発者へアンケートを行った．具体的には，著者らが開発している OSS である kGenProg [18] の過去の PR に対して CLIONE を実行し，出力された通知内容から 10 件を選択し，その PR を作成した開発者 4 名に通知内容に関するアンケートを行った．なお，本論文の筆頭著者はその開発者に含まれていない．アンケート対象の PR と，その PR 内で行われたクローンの変更内容を表 6 に示す．また，アンケート内容は以下のとおりである．

表 6　アンケート対象の PR 一覧
Table 6　A list of PRs that are targets of questionnaires.

PR 番号	クローンの変更
40	新規追加
76	新規追加
154	新規追加
442	新規追加
449	新規追加
482	新規追加
491	新規追加
496	新規追加
633	非同時修正
663	新規追加

（1） この PR 内で行われたクローンの変更を認識していましたか．

（2） 認識していた場合，修正（新規追加クローンの場合は集約，非同時修正クローンの場合は同時修正）を行わなかった理由は何ですか．

（3） PR 作成時に CLIONE の通知があった場合，修正を行いますか．

以降，アンケートに対する回答の紹介と考察を行う．

質問（1）で開発者がクローンの変更を認識していなかったという回答が 2 件（PR#633，663）得られた．どちらも質問（3）に対して，修正対象のコード片が通知されていれば修正を行うと回答していた．現在の kGenProg の実装を確認したところ，どちらの修正対象

(注8)：GitHub 利用者が気に入ったリポジトリを保存するための機能．スター数が多いほど人気があるリポジトリだといえる．

表 7　アンケート結果
Table 7　Questionnaire results.

PR 番号	質問（1）の回答	質問（2）の回答	CLIONE の改善案
633，663	認識していなかった	—	—
154，442，482	認識していた	集約する必要がない	フィルタリング機能の追加
491，496		可読性を優先した	—
40		実装速度を優先した	Issue 作成機能の追加
76		集約が難しい	集約に適したクローン検出器の利用
449		集約方法を知らない	修正方法提示機能の追加

のコード片も修正がなされていた．以上から，CLIONE を利用していれば，開発者は PR 作成時に修正対象のコード片を修正できたといえる．

一方，質問（1）で開発者がクローンの変更を認識していたという回答が 8 件得られた．以降，この 8 件についてより詳細に分類し，考察を行う．初めに，3 件（PR#154，442，482）に関しては，質問（2）で “対象のクローンを集約する必要がない” という旨の回答を得られた．対象のクローンを確認したところ，全て行数が 6 行程度だった．以上から，開発者は行数の小さいクローンを集約対象として扱わないと考えられる．したがって，新規追加されたクローンに対して，行数でフィルタリングを行った後に通知を行うように CLIONE を改善することで，このような開発者に有用でない通知を送信しないようにできると考えられる．

次に，2 件（PR#491，496）に関しては，質問（2）で “クローンを集約することで可読性が悪化すると考えた” という旨の回答が得られた．これらの PR で新しく追加されたクローンは，似たような処理を行っているがどちらも親クラスが異なっているためメソッドの引き上げが難しく，また，無理にメソッドの抽出を行うと可読性が悪化する可能性があるクローンだった．一方，これは追加されたコードにおける継承関係などの設計の悪さがクローンの集約を困難にしているとも考えられる．実際に，PR#491，496 で追加されたコードに対して設計を改善するリファクタリングが行われていた(注9)．このことから CLIONE は，集約対象のクローンのみでなく，プロジェクトの設計の悪さを指摘しているとも考えられる．

次に，1 件（PR#40）に関しては，質問（2）で “実装速度を重視してあえて集約を行わなかった” という旨の回答が得られた．この回答から，クローンを集約するかどうかは開発者の判断によることがわかる．一方，このような実装速度を重視した場当たり的な実装は技術的負債と呼ばれ，できるだけ早い段階で修正すべきだと考えられている [19]．このようなコード片の対処方法として，Issue の作成が挙げられる．Issue とは GitHub の機能の一つであり，Issue を用いることでタスクを管理したり，他の開発者へ助けを求めたりできる．技術的負債を修正するタスクを Issue として管理することで，その後の技術的負債に対する修正作業を容易にできる．したがって，CLIONE に Issue 作成機能を実装することで，このような修正対象のコード片の管理を支援できると考えられる．

次に，1 件（PR#76）に関しては，質問（2）で “クローン間で変数の型が異なるので集約が困難である” という旨の回答が得られた．これについては，一般的なクローン検出器はクローンを検出する際に識別子の正規化を行うが，同時に型名も正規化するのが原因である．したがって，型名を正規化しない設定をもつようなクローン検出器を CLIONE で利用することによって，このような集約に適していないクローンの検出を避けられると考えられる．

最後に，1 件（PR#449）に関しては，質問（2）で “集約を行いたかったが，方法を知らなかった” という旨の回答が得られた．対象のクローンは共通の親クラスをもち，処理の一部のみが異なるクローンだった．現在の kGenProg の実装を確認したところ，このクローンは Template Method パターン [20] によって集約されていた．これに関しては，先ほど述べたような Issue 作成機能を CLIONE に実装することで，PR 作成時でのクローンの集約が困難だと開発者が判断しても，Issue を作成することで他の開発者に助けを求められる．また，質問（3）の回答に “CLIONE が「こう集約するといいですよ」という提案をしてくれるとなお良い” という記述があった．そのため，修正対象のコード片を通知するだけでなく，その修正方法も提示できるように CLIONE を改善することも重要であると考えられる．

本実験のアンケート結果を表 7 に示す．アンケート結果から，開発者は 10 件中 2 件のクローンの変更を認

(注9)：https://github.com/kusumotolab/kGenProg/pull/501

識しておらず，変更に関する通知があればクローンの修正を行っていたという回答を得られた．また，残り 8 件の変更に関しては，質問（2）の回答から，幾つかの CLIONE の改善案を得られた．以上から，CLIONE は開発者に対して有用な通知を行え，また，現時点で有用でないと開発者に判断された通知であっても，CLIONE に幾つかの改善を加えることで，有用でない通知を送信しないようにしたり，通知以外の方法で開発者を支援したりする可能性を示した．したがって，CLIONE は開発者のクローン修正支援に対して有用であると考えられる．

6. 関 連 研 究

6.1 PR 型開発支援システム

Alizadeh らは，自動リファクタリングを PR 型開発に統合するシステム RefBot を提案した [21]．RefBot は PR の作成ごとに自動で対象プロジェクトのリファクタリングを行い，結果を開発者に提示する．

Carvalho らは，静的解析を PR 型開発に統合するシステム C-3PR を提案した [22]．C-3PR は PR の作成ごとに対象プロジェクトに対して既存の静的解析ツールを実行し，出力された警告を自動で修正する．

Alizadeh 及び Carvalho らは開発したシステムを実開発に適用する実験を行っており，それぞれ自動リファクタリングと静的解析の PR 型開発への統合は開発者に有用であると結論付けている．CLIONE はクローン修正支援を PR 型開発に統合するシステムであり，同様に開発者に有用であると考えている．

6.2 クローン修正支援

NGuyen らは，クローンの同時修正支援を行うツール JSync を提案した [23]．JSync は Eclipse(注10) のプラグインとして開発されており，コードのコミット時に非同時修正クローンを自動で検出する．また，修正されたクローンと修正されなかったクローン間の差分を計算し，クローンの修正方法を開発者に提示する．

Wit らはクローン管理ツール CLONEBOARD を提案した [24]．CLONEBOARD も Jsync と同様に Eclipse のプラグインとして開発されており，コード片のコピーアンドペースト操作を記録し，ペーストされたコード片が修正された際に，そのコピー元のコード片を修正対象として開発者に提示する．

これらの既存ツールと CLIONE の大きな違いは，既存ツールがクライアントサイドアプリケーションとして実装されているのに対し，CLIONE がサーバサイドアプリケーションとして実装されているという点である．CLIONE はその特徴から以下の長所がある．

（1） CLIONE は開発者個人の環境に依存しない利用が可能である．既存ツールは Eclipse プラグインとして実装されていることから，Eclipse を利用している開発者しか利用できない．

（2） CLIONE は開発者個人が複雑な設定を行うことなく，GitHub のプロジェクト単位で開発への導入や利用の解除を行える．

一方で以下の短所が考えられる．

（1） 外部サーバにプロジェクトをダウンロードできないプロジェクトに対して CLIONE を利用したい場合，オンプレミスでサーバを用意する必要がある．

（2） 開発者個人の好みに合わせた CLIONE の設定の変更は困難である．

また，CLIONE は PR 型開発を支援するという点においても既存ツールと異なる．PR 型開発は今日の OSS 開発で広く採用されているため，PR 型開発を支援する重要性は高いと著者らは考えている．

また，著者らはこれらの手法と CLIONE は必ずしも排他的ではないと考えている．例えば，JSync のクローン修正方法の提示を CLIONE に統合させることで，より有用な開発者支援が行えると考えている．

7. 妥当性の脅威

本研究では，特定の OSS を対象に実験を行った．そのため，異なるプロジェクトを対象とした場合，異なる結果が得られる可能性がある．また，実験 4 ではアンケートを行ったが，アンケート対象の開発者 4 名の中に本論文の著者 1 名が含まれている(注11)．そのため，その開発者はアンケートに好意的な回答をしている可能性がある．しかし，著者らが確認した限り，他の開発者が行ったアンケート結果と比較して特段好意的な回答は見られなかった．

CLIONE はクローン検出に既存のクローン検出器 NiCad と SourcererCC を用いている．異なるクローン検出器を用いた場合，今回の実験結果とは異なる結果が得られる可能性がある．

CLIONE によるクローン対応付け及び変更の分類は

（注10）：https://eclipse.org/

（注11）：その開発者が回答したアンケートの PR 番号は次のとおり: 76, 154，442，482，491，496，633

クローン検出の対象となるプログラミング言語に依存しない．そのため，CLIONE が利用しているクローン検出器がクローンを検出可能なプログラミング言語で記述された全てのプロジェクトに対して CLIONE を適用できる．しかし，本研究では Java で記述された OSS のみを実験対象として扱っている．そのため，他の言語に対して実験した場合，結果が異なる可能性がある．

8. むすび

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手法を提案し，GitHub 向けシステム CLIONE として実装した．CLIONE は PR の作成時にその PR の先頭コミットと分岐コミット間でのクローン変更により修正が必要になったコード片を検出する．加えて，四つの実験から CLIONE の有用性を確認した．

今後の課題として以下を考えている．

既存システムの二つ目の課題に対する評価

著者らは **1.** にて，既存システムは大量の修正対象のコード片を一度に開発者に通知してしまうことを課題として挙げている．しかし，現時点では提案手法がこの課題を解決できているかの評価を行えていない．この課題に対する評価実験は今後の重要な課題である．

実装の改善

今後は CLIONE の実装の改善を行う予定である．具体的には，**5.4** で述べたように，集約に適しないクローンのフィルタリング機能，Issue 作成機能，及び CLIONE が検出した修正対象のコード片に対する修正方法の推薦機能の実装を予定している．

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,” Int. Conf. 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,” Int. Conf. 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,” Int. Conf. Program Comprehension, pp.199–206, 2013.

[6] J. Feller and B. Fitzgerald, Understanding Open Source Software Development, Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[7] T. Nakagawa, Y. Higo, and S. Kusumoto, “CLIONE: Clone modification support for pull request based development,” Asia-Pacific Software Engineering Conference, pp.455–459, 2020.

[8] J.R. Cordy and C.K. Roy, “The NiCad clone detector,” Int. Conf. Program Comprehension, pp.219–220, 2011.

[9] J. Cordy, “The TXL source transformation language,” Science of Computer Programming, pp.190–210, Aug. 2006.

[10] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C.V. Lopes, “SourcererCC: Scaling code clone detection to big-code,” Int. Conf. Software Engineering, pp.1157–1168, 2016.

[11] “CCFinderX,” http://www.ccfinder.net/

[12] 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.

[13] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, “An empirical study of code clone genealogies,” Int. Symp. Foundations of Software Engineering, pp.187–196, 2005.

[14] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, “Wait for it: Determinants of pull request evaluation latency on GitHub,” Working Conf. Mining Software Repositories, pp.367–371, 2015.

[15] 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.

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

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

[18] Y. Higo, S. Matsumoto, R. Arima, A. Tanikado, K. Naitou, J. Matsumoto, Y. Tomida, and S. Kusumoto, “kGenProg: A high-performance, high-extensibility and high-portability APR system,” Asia-Pacific Software Engineering Conference, pp.697–698, 2018.

[19] F.A. Fontana, V. Ferme, and S. Spinelli, “Investigating the impact of code smells debt on quality code evaluation,” Int. Workshop on Managing Technical Debt, pp.15–22, 2012.

[20] E. Gamma, Design patterns: elements of reusable object-oriented software, Pearson Education India, 1995.

[21] V. Alizadeh, M.A. Ouali, M. Kessentini, and M. Chater, “RefBot: Intelligent software refactoring bot,” Int. Conf. Automated Software Engineering, pp.823–834, 2019.

[22] 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,” Int. Conf. Software Analysis, Evolution and Reengineering, pp.161–171, 2020.

[23] H.A. Nguyen, T.T. Nguyen, N.H. Pham, J. Al-Kofahi, and T.N.

Nguyen, “Clone management for evolving software,” IEEE Trans. Software Engineering, pp.1008–1026, 2012.

[24] M. de Wit, A. Zaidman, and A. van Deursen, “Managing code clones using dynamic change tracking and resolution,” Int. Conf. Software Maintenance, pp.169–178, 2009.

（2020 年 11 月 16 日受付，2021 年 3 月 24 日再受付，
5 月 18 日早期公開）

中川　　将

2019 大阪大学基礎工学部情報科学科卒．2021 同大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程了．在学中，コードクローン分析に関する研究に従事．

肥後　芳樹　（正員）

2002 大阪大学基礎工学部情報科学科中退．2006 同大学大学院博士後期課程了．2007 同大学大学院情報科学研究科コンピュータサイエンス専攻助教．2015 年同准教授．博士（情報科学）．ソースコード分析，特にコードクローン分析，リファクタリング支援，ソフトウェアリポジトリマイニング及び自動プログラム修正に関する研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE 各会員．

楠本　真二　（正員）

1988 大阪大学基礎工学部卒．1991 同大学大学院博士課程中退．同年同大学基礎工学部助手．1996 同講師．1999 同助教授．2002 同大学大学院情報科学研究科助教授．2005 同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価に関する研究に従事．情報処理学会，IEEE，IFPUG 各会員．