

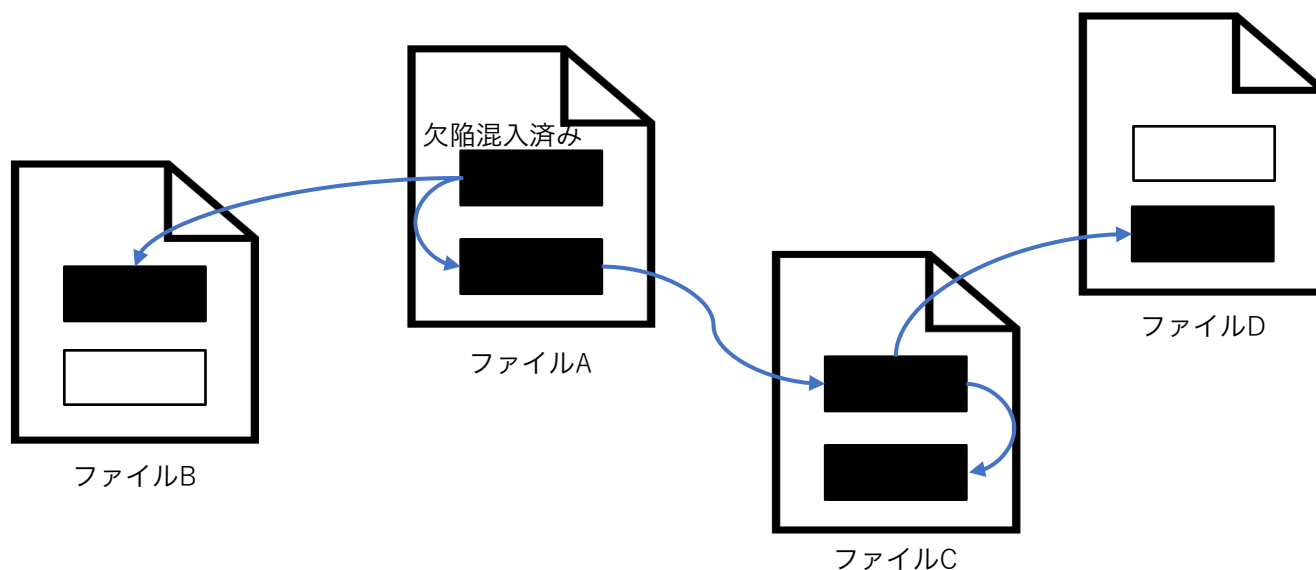
Gitに対応した欠陥混入 クローン特定手法の構築

オープンソースソフトウェア工学研究室

野口 耕二郎

背景

- 品質に悪影響を与えるコードクローンを優先して除去，管理する研究が盛んになってきている
- 品質に悪影響を与えるコードクローンの1つとして欠陥混入クローンが挙げられる



欠陥混入クローンに関する研究

- コードクローン作成時に10%の割合で欠陥が複製され[1]
- より最近変更されたコードクローンに欠陥が混入しやすい[2]
- 欠陥混入クローンの33%から欠陥が拡散する [3]

欠陥混入クローン特定手法が重要

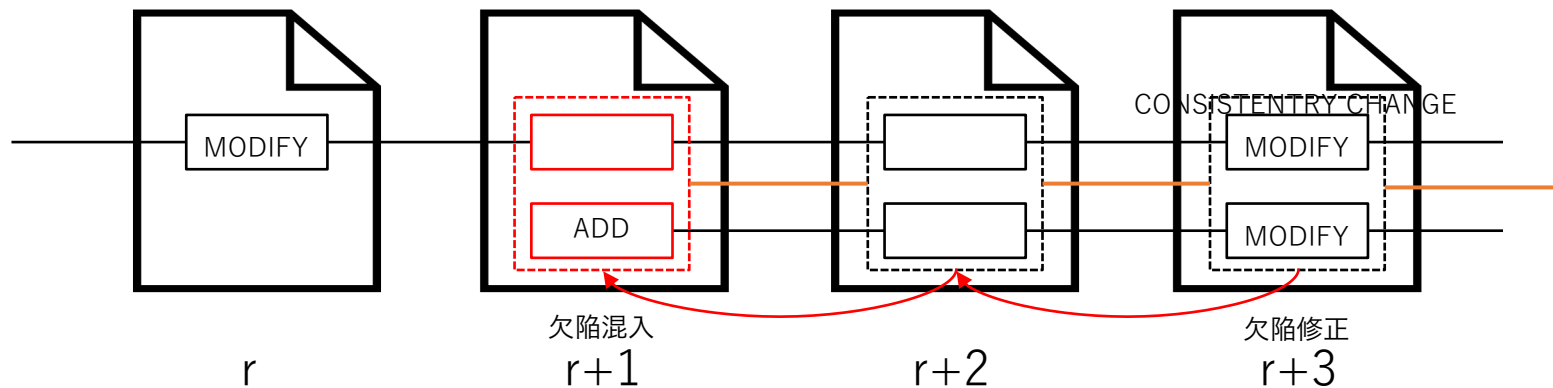
[1] J.F. Islam, M. Mondal, and C.K. Roy, "Bug replication in code clones: An empirical study," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol.1, pp.68–78, 2016.

[2] M. Mondal, C.K. Roy, and K.A. Schneider, "Identifying code clones having high possibilities of containing bugs," Proceedings of the 25th International Conference on Program Comprehension, p.99–109, ICPC '17

[3] M. Mondal, C.K. Roy, and K.A. Schneider, "Bug propagation through code cloning: An empirical study," 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.227–237, Sep. 2017.

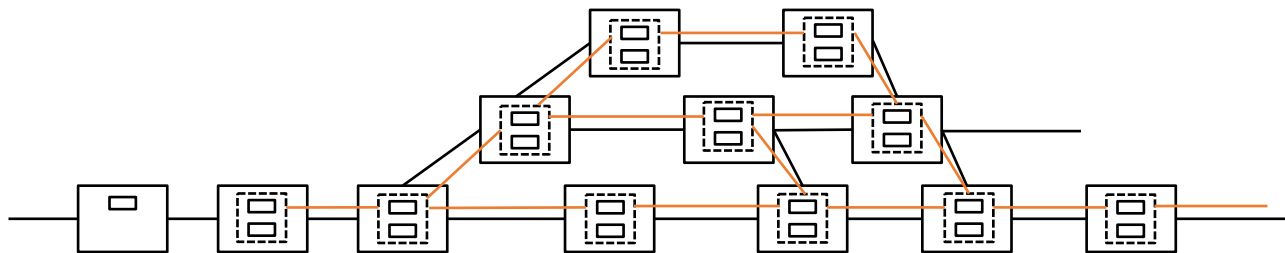
既存の欠陥混入クローン特定技術[4]

1. 欠陥修正を含むクローンを特定する($r+3$)
2. 履歴を追跡し, 欠陥混入リビジョンでクローンが作成されたことを確認($r+1$)
3. 欠陥混入から修正までの間に変更が加えられていないことを確認($r+2$)



既存手法の問題点

- SVNなどの集中型バージョン管理システムからの系譜抽出を想定したもの
- 現在主流のGitなどの分散バージョン管理システムには対応できない
 - Gitのブランチ・マージ発生に伴う追跡ができない
 - 変更履歴が複雑になり **計算コストが膨大になる**
(SVN: $O(n^3)$ → Git: $O(n^4)$)



DVCSから欠陥混入クローンを収集するためには、
変更履歴の複雑さに対処する必要がある

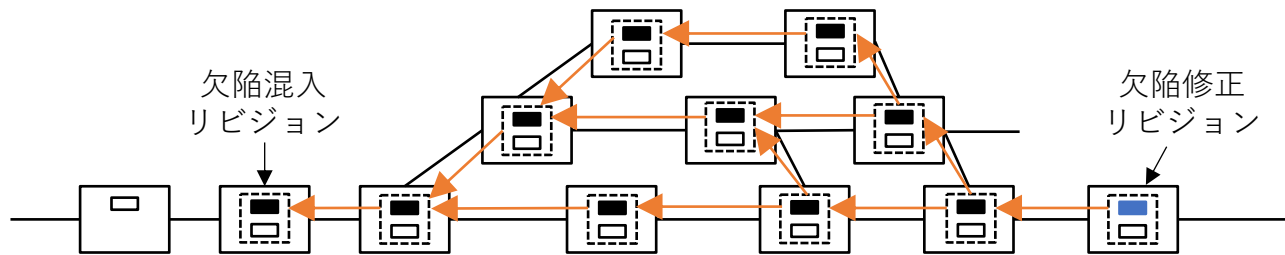
目的とアプローチ

- 目的：Git対応欠陥混入クローン特定手法の構築
 - マージ・ブランチを伴う追跡に対応
 - 計算量の増加を抑える
- 計算量を抑えるためのアプローチ
 - リビジョン抽出
 - 欠陥混入リビジョン・修正リビジョンに絞り込む
 - 欠陥混入クローン特定
 - 欠陥混入・修正ファイル間のクローンの位置を補正する

既存手法との違い：リビジョン抽出

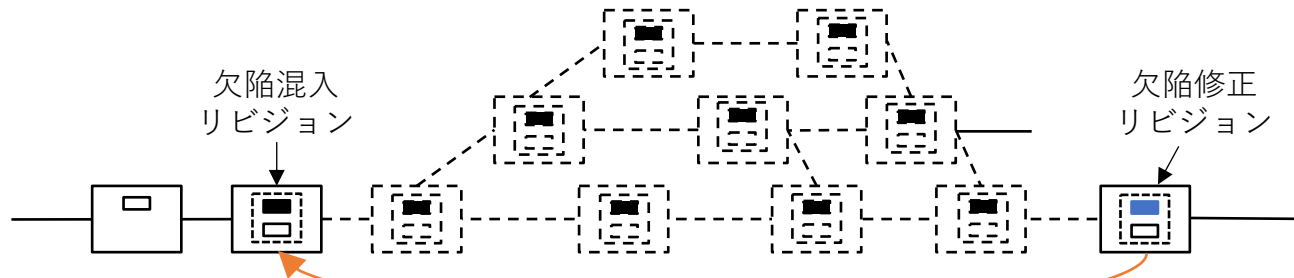
• 既存手法

- 追跡のために全リビジョンのコード片，コードクローン検出をおこなう



• 提案手法

- 欠陥混入リビジョン・修正リビジョンに絞り込む

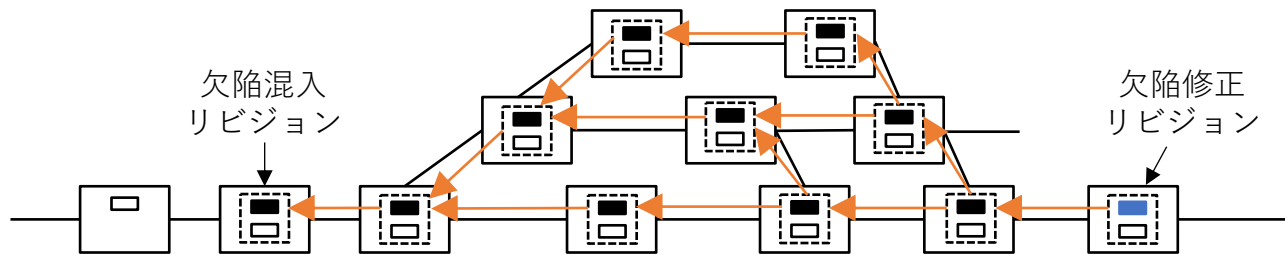


コミットログから欠陥修正リビジョンを特定し，
欠陥が混入したリビジョンまで遡る

既存手法との違い：欠陥混入クローン特定

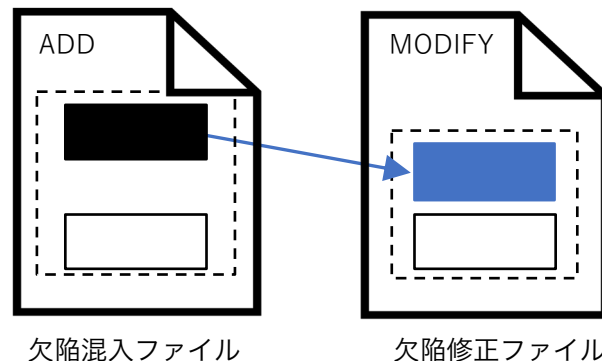
• 既存手法

- クローンの位置をリビジョン毎に確認し補正する



• 提案手法

- 欠陥混入・修正ファイル間のクローンの位置を補正する



計算量の比較（机上評価）

- 既存手法
 - Gitに対応させた場合： $O(n^4)$
- 提案手法
 - コード片登録： $O(n^3)$
 - 欠陥混入，修正コード片ペア特定： $O(n^3)$
- 提案手法（ $O(n^3) + O(n^3)$ ）＜既存手法（ $O(n^4)$ ）
であり，計算量を削減することができる

評価実験：実行時間

- 実験1

- 目的

- 実行時間が実際に削減できるかを実データで確かめる

- 方法

- 既存手法と提案手法の処理性能を実行時間で比較する
 - 全リビジョンを対象にした場合のクローン特定までの実行時間
 - 欠陥混入，修正リビジョンのみを対象にした場合のクローン特定までの実行時間

- 実験2

- 目的

- 提案手法の実用性を実行時間から検証する

- 方法

- コード片（≡リビジョン数）を増やしながら実行し，欠陥混入クローン特定までの実行時間を計測

評価実験：クローン特定精度

- 実験3

- 目的

- クローン特定処理の変更（GumTreeによるクローン位置の補正）によるクローン特定精度への影響を検証する

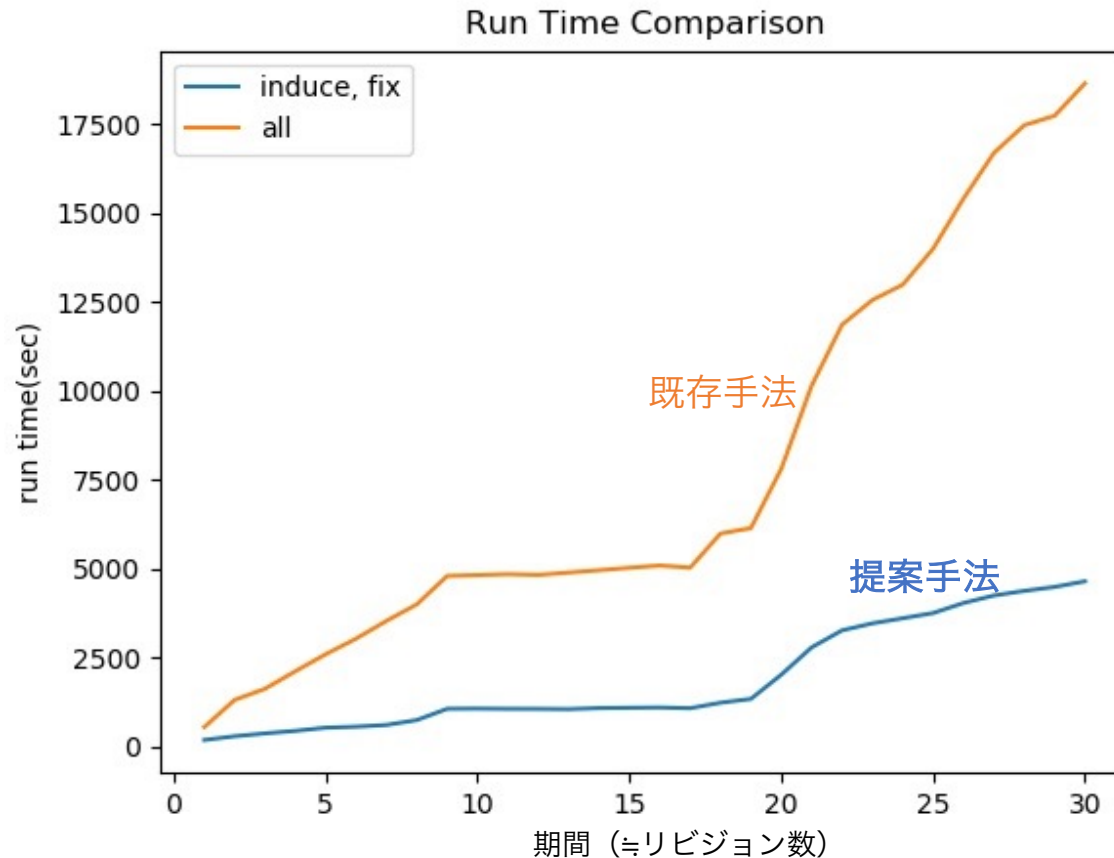
- 方法

- 誤検出（偽陽性）
 - 特定した欠陥混入クローン104件を手作業で確認
 - 見逃し（偽陰性）
 - 変更が加えられたコードクローン1,458件を手作業で確認
 - 見逃した欠陥混入クローンがないかを確認
 - 適合率，再現率により評価

データセット

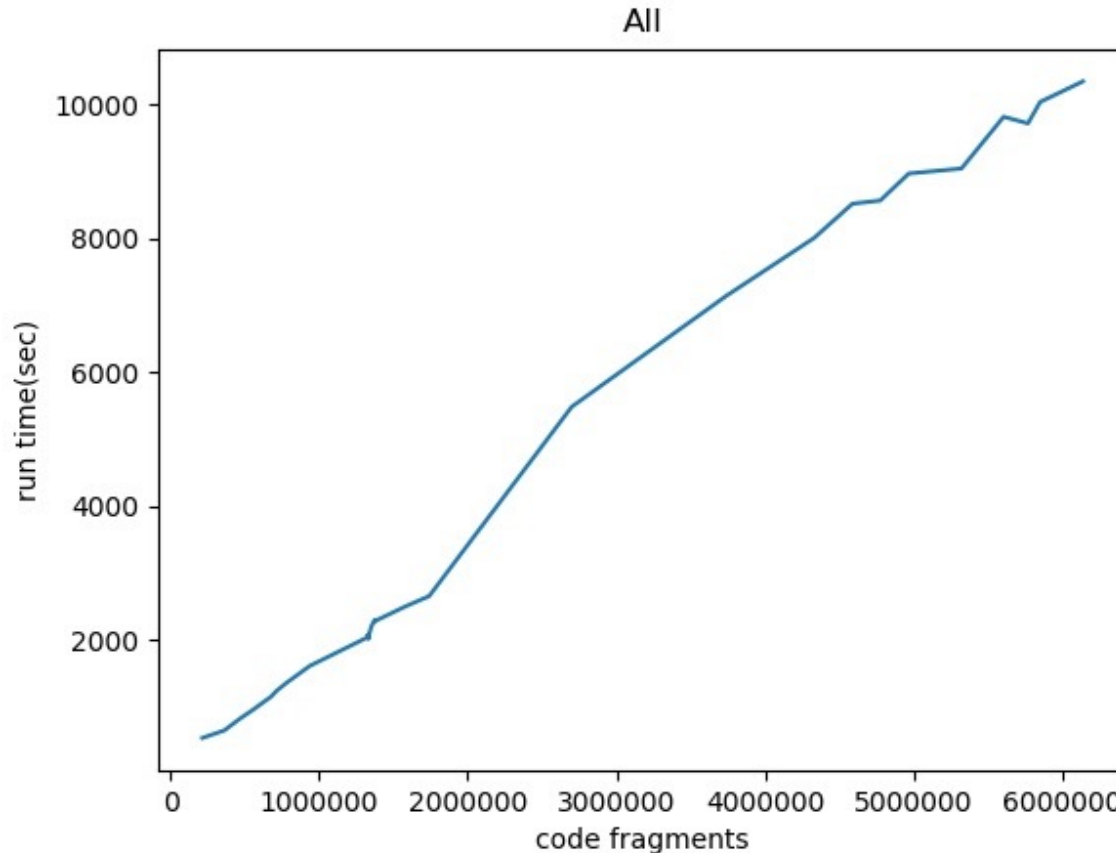
- プロジェクト : RxJava
- 期間 : 2015/1/1 – 2017/7/1 (30ヶ月)
- リビジョン数
 - 全体 : 1,409件
 - 欠陥混入, 修正 : 371件
- コード片数 : 6,138,787件
- クローン数
 - 変更が加えられたコードクローン : 1,458件
 - 欠陥混入クローン特定結果 : 104件

実験1：実行時間（性能比較）



既存手法の実行時間の約75.1%削減できた

実験2：実行時間（実用性の検証）



コード片数が6,000,000を越えても2, 3時間で全体の処理が完了する

結果：精度

- 適合率： $TP/(TP+FP)=39/(39+65)=37.5\%$
- 再現率： $TP/(TP+FN)=39/(39+105)=27.1\%$
- 検出結果の内，正しく取得できたのは約4割で実用性が乏しい

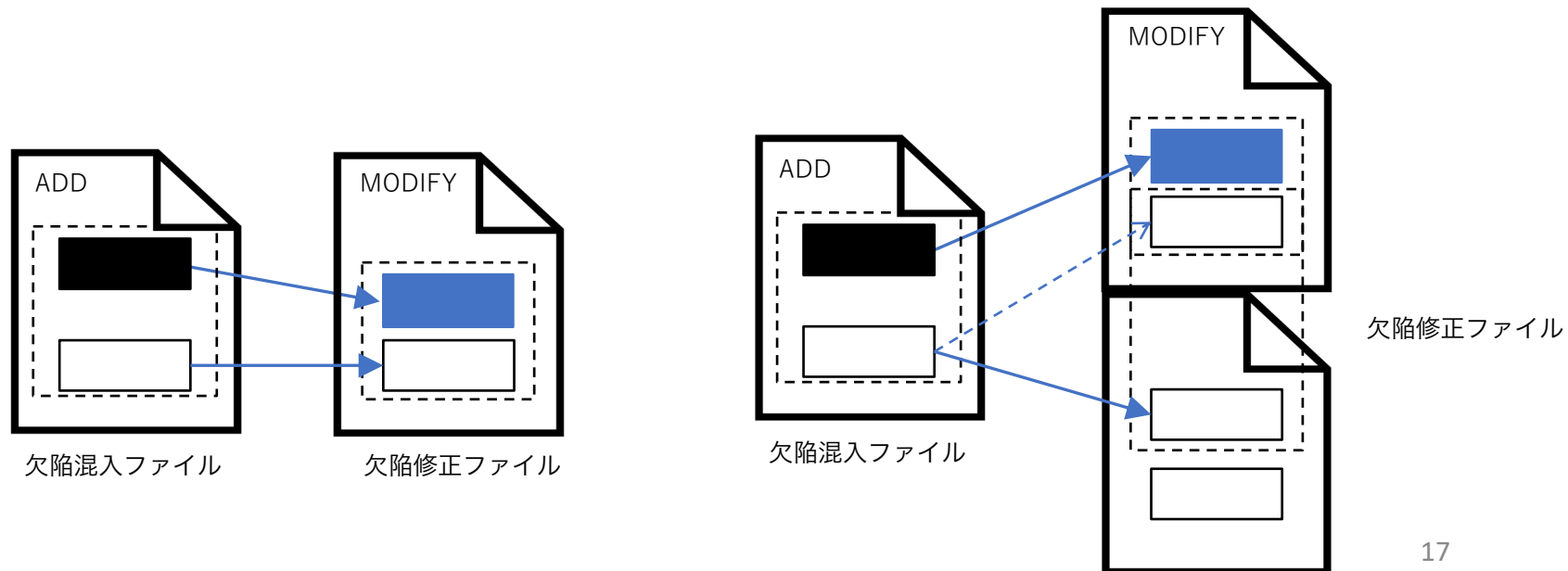
考察：精度

- 精度低下の原因

- 欠陥混入クローンが属するファイルが修正時に異なるにも関わらず修正されたクローンとして特定していた

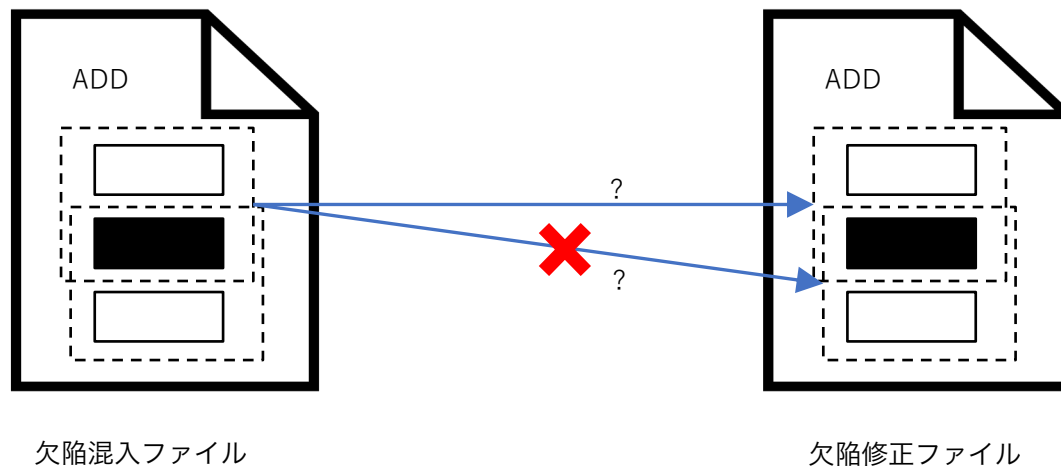
- 対策

- 欠陥混入ファイルと欠陥修正ファイルが同一であることを欠陥混入クローン特定時の条件に加える



手法修正後の精度

- 結果
 - 適合率：28.5%
 - 再現率：74.5%
- 同じクローンが複数ペアある場合に欠陥混入，修正クローンペアを絞り込む必要がある



今後の予定

- 10～11月

- 位置補正機能修正
- 評価対象プロジェクトの追加（+2件）

- 12～1月

- 欠陥混入クローンの特徴分析
- 修論執筆

- 研究実績

- 査読付口頭発表論文
 - Yusuke Kukita, Kojiro Noguchi, and Masao Ohira, "Code Clone Tracer (CCT): a Tracking Tool for Analyzing Human and Social Factors in Creating and Reusing Code Clones," In SNPD '19, pages 519--524, July 2019.
 - 野口 耕二郎, 大平 雅雄, "コードクローンへの欠陥混入防止に向けた欠陥混入クローンの特徴分析," ソフトウェア・シンポジウム 2019 in 熊本, pages 56--65, 2019年6月.
- ポスター発表
 - 野口 耕二郎, 大平 雅雄, "コードクローンに混入した不具合の発生原因の調査: RxJavaプロジェクトを対象として," 第25回 ソフトウェア工学の基礎ワークショップ FOSE2018, 2018年11月.
- 受賞
 - 荒木富松特別奨励賞: 野口 耕二郎@ソフトウェア・シンポジウム 2019 in 熊本

補足資料

提案手法フローチャート

