
派生開発における コードクローンの 追跡と管理の支援

大平研究室 M1

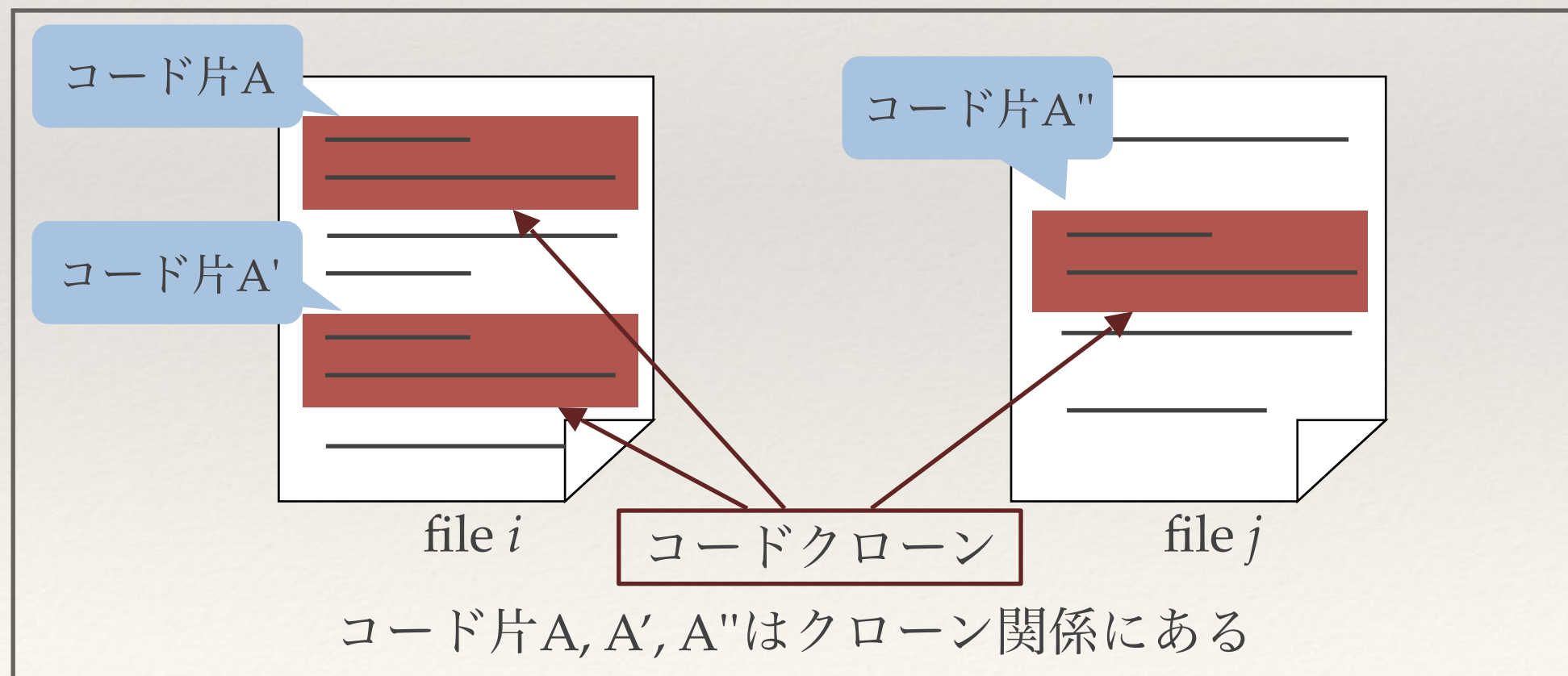
60160037 久木田 雄亮

研究の背景

- ❖ システムの大規模化・複雑化に伴い、近年のソフトウェア開発案件のほとんどは派生開発である
- ❖ 派生開発
 - ❖ 納期・品質を厳守するために、品質がある程度保証されている既存システムの一部（あるいは大部分）を流用するSW開発を指す
 - ❖ システムを長期的に運用するために、派生開発したシステムをさらに継続的に改良・拡張する
- ❖ 流用による開発とソフトウェア品質との関係は不明確
 - ❖ コードクローンに着目した研究が盛んに行われている

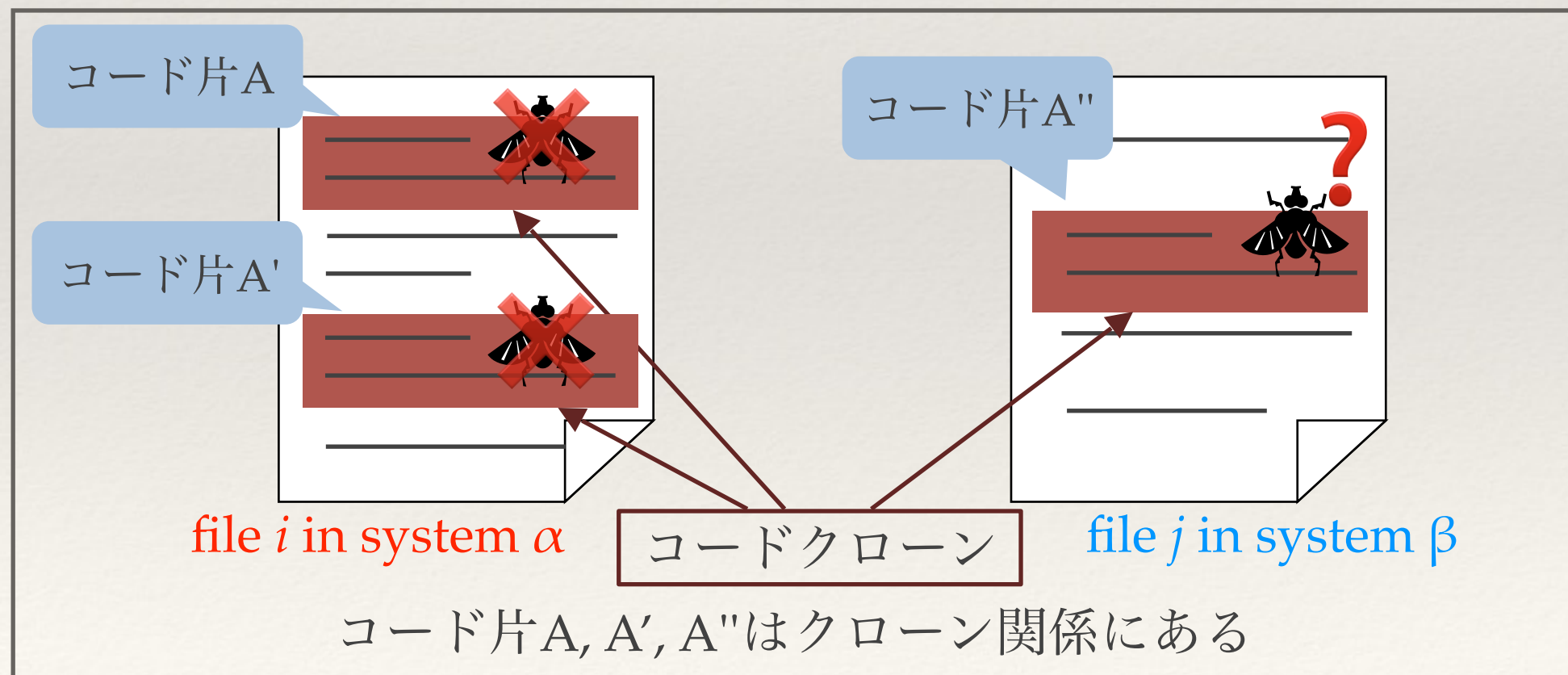
コードクローン

- ❖ ソースコード中に存在する「同一の」または「類似する」コード片のこと
 - ❖ 主に、既存コードを再利用（コピーアンドペースト）することにより作成される



派生開発におけるコードクローン

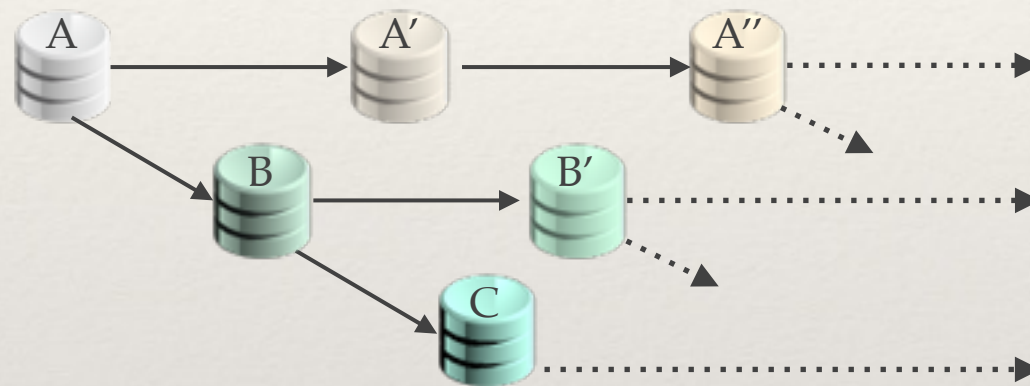
- ❖ 派生SWでは、改良・機能拡張のために大量のコードクローン (CC)を含むコードが変更され続ける
- ❖ コピー元のCCに対する**変更を追跡し、派生先のSWにも反映すべきかどうか**を慎重に判断する必要がある



研究課題

❖ 派生開発では，CCの追跡と管理の支援がより重要

❖ 複数システム間でのCCの変更を追跡できることが必要



❖ コピー元CCの変更がコピー先CCに与える影響の把握が必要

❖ CCの構造的特徴の分析

❖ 抽象構文木 (AST) などを用いたソースコード解析に基づいたコード品質の検証など

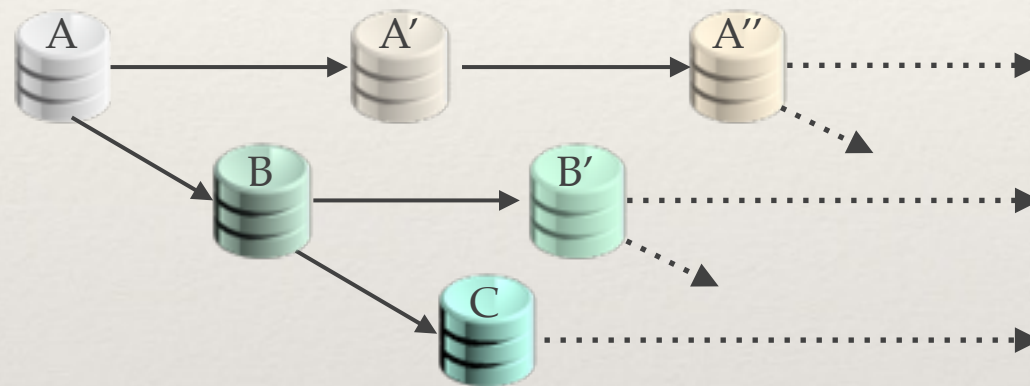
❖ CCの属人的特徴の分析

❖ CC作成者とCCの品質的特徴のマッピングに基づく再利用性 (=安全性) の検証など

研究課題

- ❖ 派生開発では，CCの追跡と管理がさらに重要になる

- ❖ 複数システム間でのCCの変更を追跡できることが必要



M1

- ❖ コピー元CCの変更がコピー先CCに与える影響の把握が必要

- ❖ CCの構造的特徴の分析

- ❖ 抽象構文本 (AST) などを用いたソースコード解析に基づいたコード品質の検証

M2
前半

- ❖ CCの属人的特徴の分析

- ❖ CC作成者とCCの品質的特徴のマッピングに基づく再利用性 (=安全性) の検証

M2
後半

先行研究：コードクローン検出手法

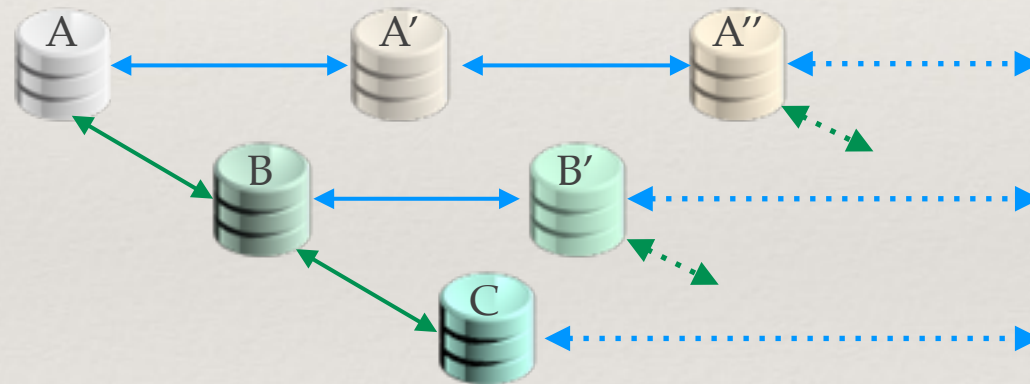
- ❖ CCを「検出」するための様々な手法が提案されている
 - ❖ 行単位での検出
 - ❖ 字句単位での検出
 - ❖ 抽象構文木での検出
 - ❖ プログラム依存グラフでの検出
 - ❖ ブロック単位での検出
- ❖ CCの「変更を追跡」できる手法は存在しない
 - ❖ 既存手法は、SW中に存在する「**ある時点**」のCCを検出するのみ



目的とアプローチ

- ❖ 目的：コードクローンを追跡するための手法の構築

- ❖ 変更前後のCCの把握：リビジョン単位でCCを追跡できること
 - ❖ リビジョン：版管理システムで管理されるSWのスナップショット
- ❖ CCの波及関係を把握：派生システム間でCCを追跡できること



- ❖ アプローチ

- ❖ リビジョン単位でCCを検出し，リビジョン間のCCの関係をマッピングする
 - ❖ CRDベースのCC検出ツールECTECを拡張する
- ❖ ブランチ（派生版）間のCCの関係をマッピングする
 - ❖ さらにECTECを拡張し，分散型版管理システムGitに対応させる

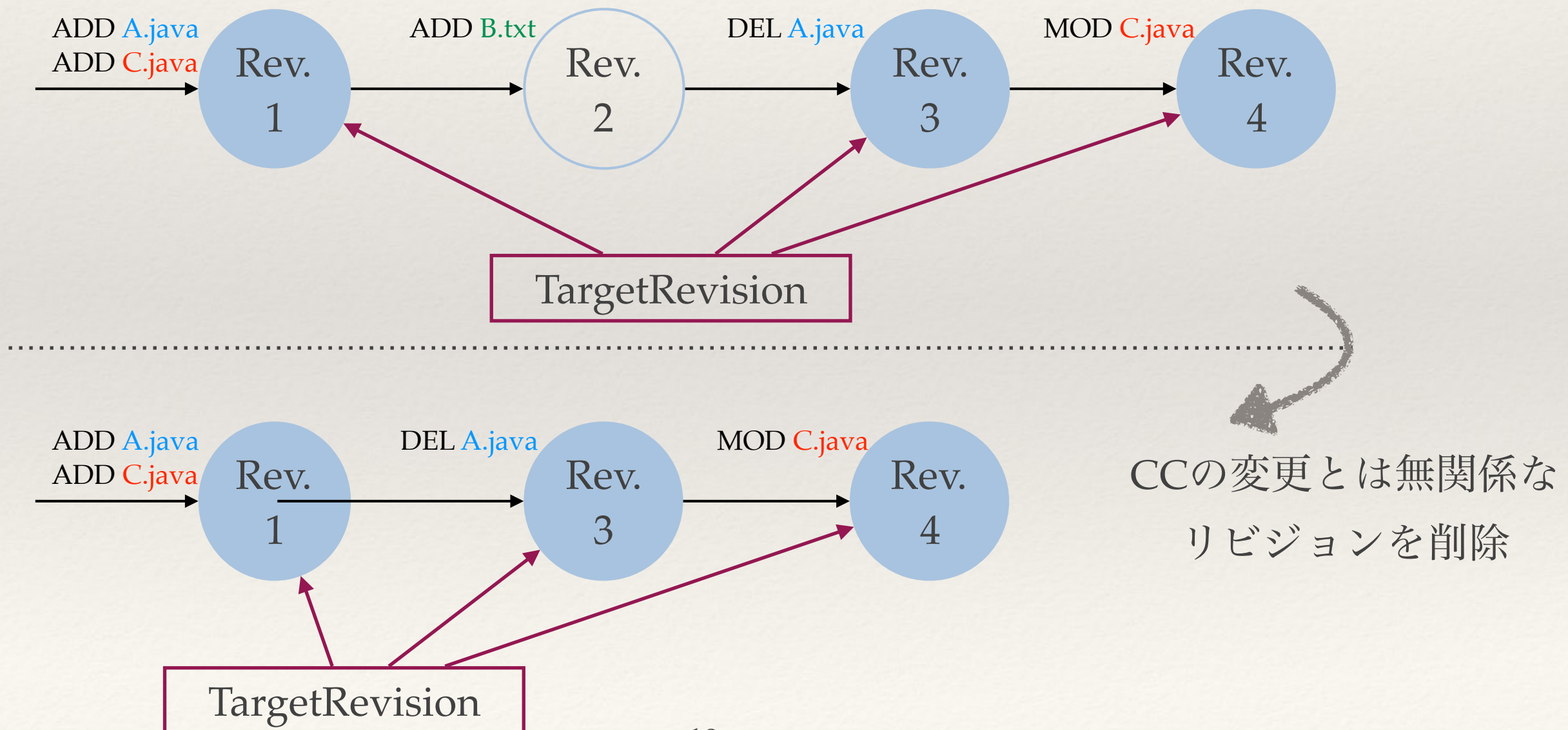
CRDベースのCC検出ツール：ECTEC

- ❖ 版管理システムからコードクローンを検出するツール
 - ❖ ○ リビジョン単位で解析できる
 - ❖ ○ CRDベース（ブロック単位）でコードクローンを検出できる
 - ❖ ✕ 集中型版管理システム（Subversionなど）にのみ対応している
 - ❖ 派生開発で主流の分散型版管理システム（Gitなど）には別途対応が必要
- ❖ ECTECを用いたCCの解析手順
 1. 分析対象のリビジョンと対象ファイルを特定する
 2. ブロックの特定し追跡する
 3. コードクローンを特定する
 - ❖ CCを追跡しリビジョン間のCCの関係をマッピングさせる必要がある

ECTECを用いた解析手順：

1. 分析対象リビジョンとファイルの特定

- ❖ 対象とするファイル (*.java) が修正されているリビジョンを対象とする



ECTECを用いた解析手順：

2. ブロックの特定と追跡

❖ CRD (Clone Region Detection)

❖ コード片の位置情報を示す表示形式

```
public class DeleteManager {  
    ...  
    public void delete(int n) {  
        ...  
        for (int i = 0; i < delete.size(); i++) {  
            ...  
            if (delete.get(i) instanceof ElementNode) {  
                // some code  
            }  
            ...  
        }  
    }  
}
```

CRD

package name.DeleteManager.java,DeleteManager,5
delete(int),5
for,delete.size(),4
if,delete.get(i) instanceof ElementNode,2

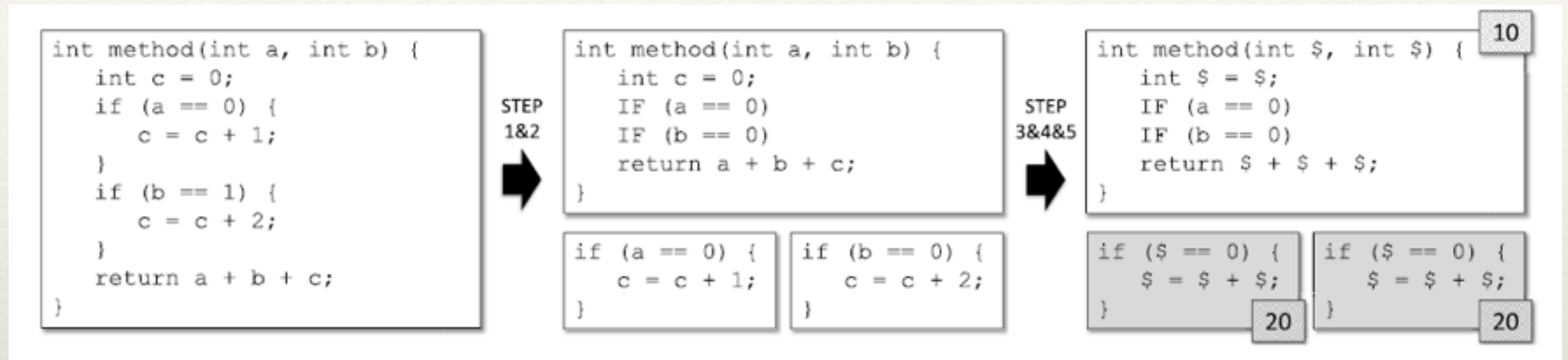
```
<CRD> ::= <file> <class> <CM> [<method>]  
<method> ::= <signature> <block> *  
<block> ::= <btype> <anchor> <CM>  
<btype> ::= 'for' | 'while' | 'do' | 'if' | 'else' | 'switch'  
            | 'try' | 'catch' | 'finally' | 'synchronized'
```

○ CM：各ブロックのメトリクス値
(サイクロマチック数 + fan-out)

CRDの文字列が一致・類似するものを同じブロックとして追跡する

ECTECを用いた解析手順：

3. コードクローンの特定制



- ❖ ブロックの特定制：クラス，メソッド，ブロック文（for, ifなど）を特定制
- ❖ 正規化：変数名，リテラルの置き換え
- ❖ ハッシュ値の算出：各ブロック文の文字列をハッシュ値に変換
- ❖ **コードクローンの特定制：ハッシュ値が一致するものをクローンとして特定制**

コードクローンの特徴



- ❖ 各ブロック文の文字列をハッシュ関数を用いて変換
- ❖ ハッシュ値が一致するものをクローンとして特定
- ❖ コード片を追跡するCRDと併用することによってコードクローンを追跡する

今後の課題

- ❖ ツールの拡張

- ❖ 派生開発で主流の分散型版管理システム（Gitなど）に対応する

- ❖ 分析

- ❖ CCの構造的特徴の分析
 - ❖ 抽象構文木 (AST) などを用いたソースコード解析に基づいてコード品質を検証する
 - ❖ CCの属人的特徴の分析
 - ❖ CC作成者とCCの品質的特徴のマッピングに基づく再利用性（＝安全性）を検証

コードクロンの影響調査

- ❖ 最近の研究ではコードクロンがプロジェクトに与える影響について調査したものがある[]
- ❖ 調査によって、全てのコードクロンが保守に悪影響を与えるわけではないことが確認できた
- ❖ しかし、現状ではどのようなコードクロンが悪影響を及ぼすか、そうでないかはわからない

目的

- ❖ 修正を必要としないコードクローンの作成の支援を目的とする
- ❖ 修正を必要としないコードクローンを作成するには、開発経験とプロジェクトに関する理解が不可欠であると考えた

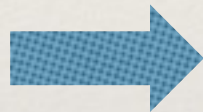
アプローチ

- ❖ コードクローン追跡ツールを用いて，作成されたコードクローンが何回修正されたか分析することにより，修正が必要であるコードクローンと修正があまり必要でないコードクローンがわかる

ECTECの問題点

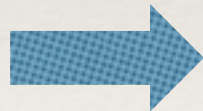
❖ バージョン管理システム

- ❖ GitHubの登場により，これまで使われていたSubversion（集中型）からGit（分散型）に移行するプロジェクトが増加した
Gitリポジトリに対応することが必要



- ❖ 集中型から分散型に移行したことで，履歴情報が複雑になる

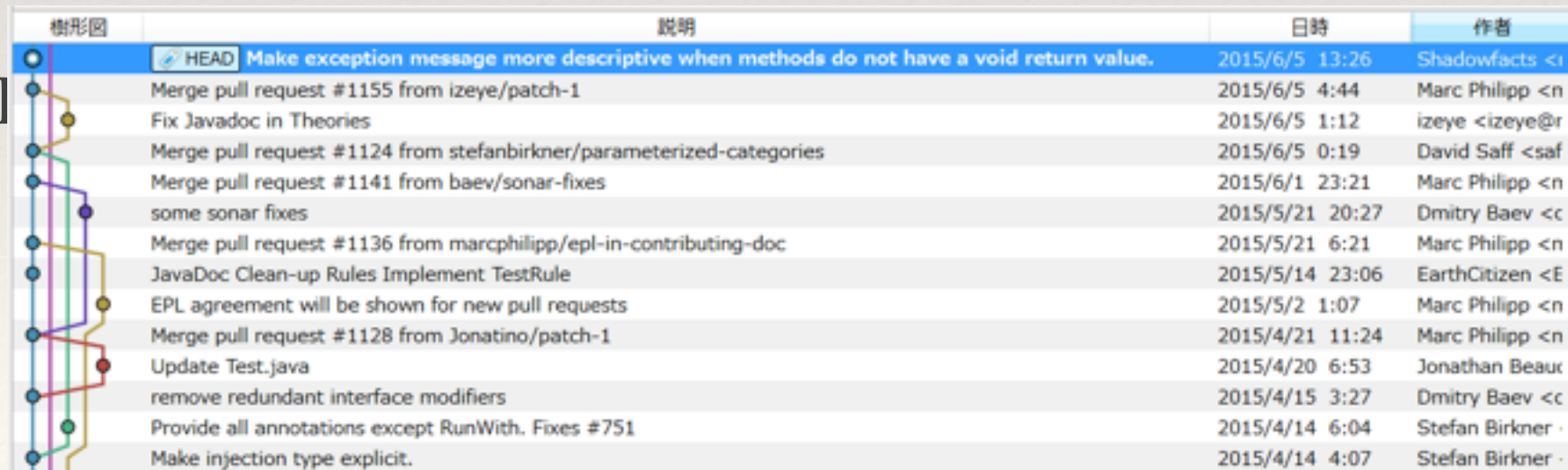
コードクローンを追跡するためには
履歴に一貫性を持たせなければならない



ECTEC : Gitへの対応

- ❖ ECTECが対応リポジトリの追加
 - ❖ ECTECはSubversionリポジトリのみ対応
 - ❖ JGitを用いてGitリポジトリにも対応するように実装
- ❖ GitリポジトリはSubversionリポジトリに比べて変更履歴が複雑になることが多い

❖ Pull Request



樹形図	説明	日時	作者
●	HEAD Make exception message more descriptive when methods do not have a void return value.	2015/6/5 13:26	Shadowfacts <i>
●	Merge pull request #1155 from izeye/patch-1	2015/6/5 4:44	Marc Philipp <n
●	Fix Javadoc in Theories	2015/6/5 1:12	izeye <izeye@r
●	Merge pull request #1124 from stefanbirkner/parameterized-categories	2015/6/5 0:19	David Saff <saf
●	Merge pull request #1141 from baev/sonar-fixes	2015/6/1 23:21	Marc Philipp <n
●	some sonar fixes	2015/5/21 20:27	Dmitry Baev <c
●	Merge pull request #1136 from marcphilipp/epl-in-contributing-doc	2015/5/21 6:21	Marc Philipp <n
●	JavaDoc Clean-up Rules Implement TestRule	2015/5/14 23:06	EarthCitizen <E
●	EPL agreement will be shown for new pull requests	2015/5/2 1:07	Marc Philipp <n
●	Merge pull request #1128 from Jonatino/patch-1	2015/4/21 11:24	Marc Philipp <n
●	Update Test.java	2015/4/20 6:53	Jonathan Beau
●	remove redundant interface modifiers	2015/4/15 3:27	Dmitry Baev <c
●	Provide all annotations except RunWith. Fixes #751	2015/4/14 6:04	Stefan Birkner ·
●	Make injection type explicit.	2015/4/14 4:07	Stefan Birkner ·

ECTEC：一貫性を持つ履歴

- ❖ Subversionでは履歴が一直線で、複雑でないことが多いため追跡が容易に行えた
- ❖ Gitでは履歴が複雑になることが多いため、履歴の一貫性が取得しづらく、追跡が困難

ECTECを用いた分析

- ❖ ECTECによって検出したクローンに対して修正が発生したとき、どの時点でコードが埋め込まれたかを調べる
 - ❖ SZZアルゴリズムを併用

今後の課題

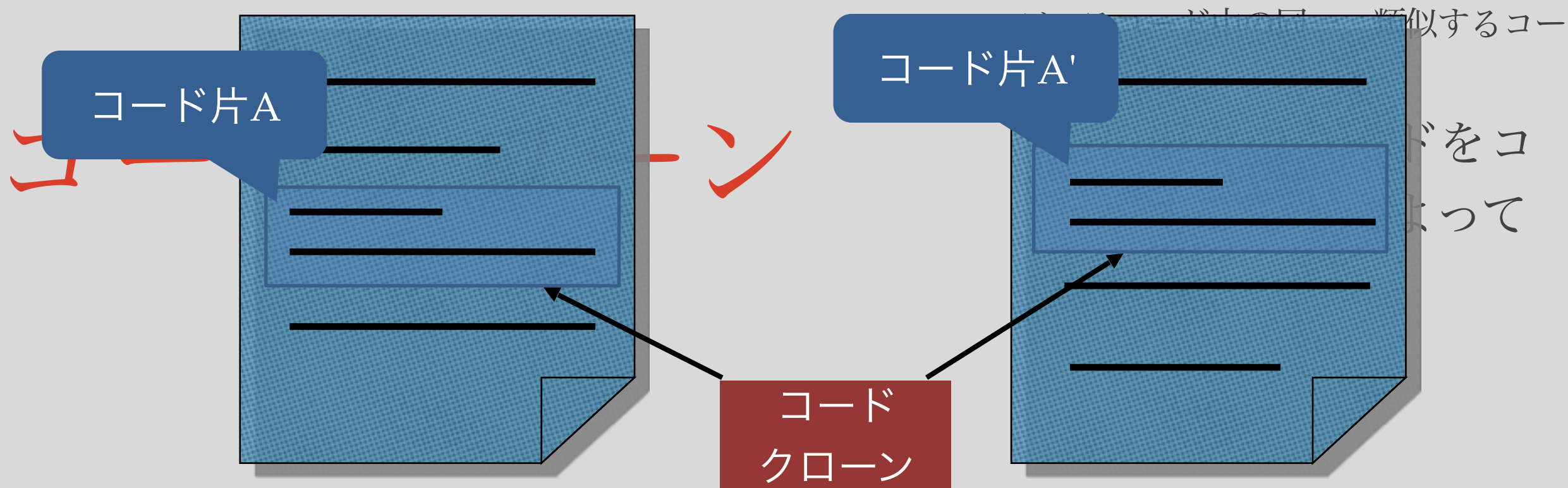
Lorem Ipsum Dolor

修正を必要としないコード ドクローン分析に向けた 調査

大平研究室 M1

60160037 久木田 雄亮

Learn Learn Delay

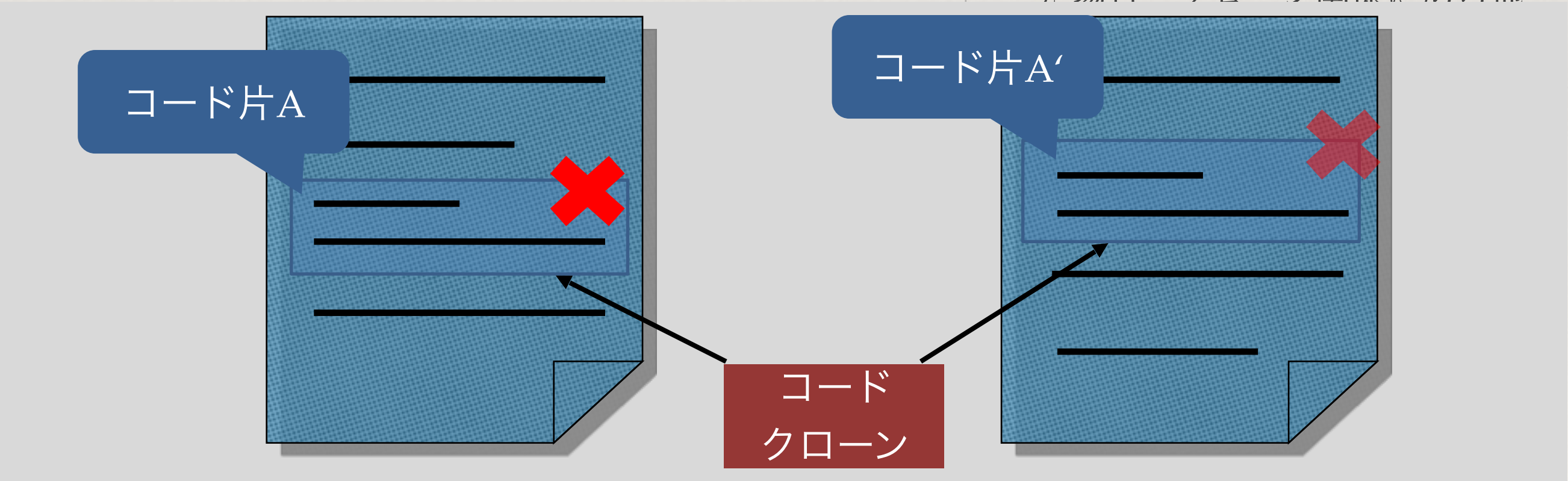


コード片Aとコード片A'はクローン関係にある

Lorem Ipsum Dolor

コードクロンの

コードクロンの存在は保守に悪影響を与える
あるコード片に不具合修正を行っ
た場合 クローン関係にある他



Lorem Ipsum Dolor

コードクローンの 発生原因

既存コードのコピーアンドペースト

既存コードを流用し、一部変更を加える場合
定型処理

ソフトウェア内で頻繁に用いられる処理はコード
クローンとなる

コード生成ツールの生成コード

ツールによって自動的にコードが生成されるため
類似したコードが生成される

偶然

Lorem Ipsum Dolor

コードクローン検 出技術

ソフトウェアの規模が大きいとコードクローンの数が膨大になるため見つけ出すのが困難
そのため、自動的にコードクローンを検出する手法が様々提案されてきた

(例) メトリクスを用いた検
出

Lorem Ipsum Dolor

コードクロンの 進化

最近の研究ではコードクロンの進化について調査したものがあ

る

87.8%のコードクローンは一度も修正されることがなかった。

コードクローン修正の内、14.8%が修正漏れが見つかった。

しかし、現状ではどのようなコードクローンが修正を必要とするのかまではわからない

Lorem Ipsum Dolor

目的とアプローチ

目的

修正を必要としないコードクローンの作成の支援を目的とする

アプローチ

修正が発生したコードがコードクローンであるかを調べることで、どのようなコードクローンを作成すると修正が発生しやすいのかを分析する
修正が発生しやすいコードクローンを理解するこ

Lorem Ipsum Dolor

分析方法

不具合修正が発生したコード
がクローンであるかを判定す
る.
クローンであった場合,

Lorem Ipsum Dolor

分析方法 - 1

1. SZZアルゴリズムを用いて、不具合修正したコードと不具合を埋め込んだコードを特定
-

Lorem Ipsum Dolor

分析方法 - 2

不具合を埋め込んだコードが
コードクローンであるかを判
定

Lorem Ipsum Dolor

分析方法 - 3

クローンであった場合、どの開発者が埋め込んだか、どのような構造のコードクローンであるかを分析する

Lorem Ipsum Dolor

使用するコードク ローン検出手法

Lorem Ipsum Dolor

今後の課題
