# 実務で使われるコードクローン検出・追跡システムをめざして

三木 聡
（株）フィックスターズ

大歳 始　浅原 明広　大澤 俊晴　千葉 滋
（株）Sider

日本ソフトウェア科学会第４０回大会（2023年度）

# 日本の開発現場におけるコピペ

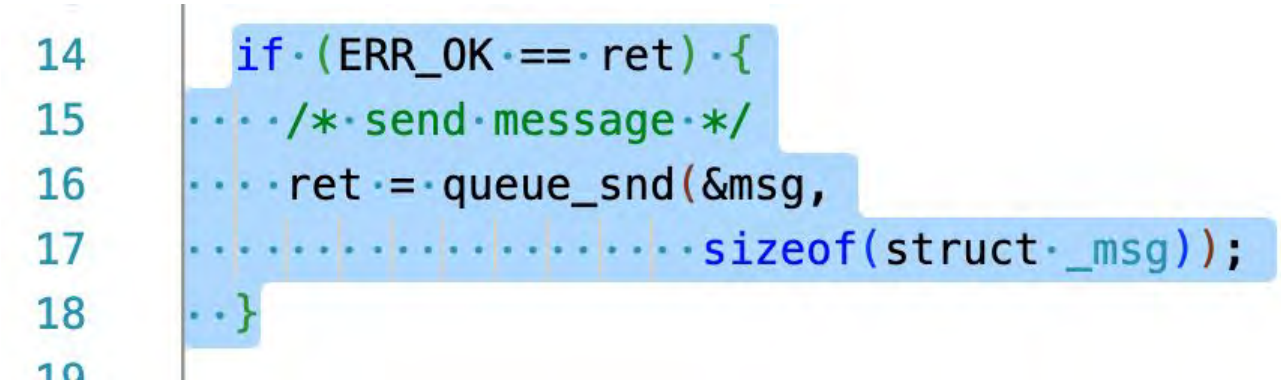
- コピペ（コピー＆ペースト）
  - 良くない、だが、まったくの悪ではない
- ある実例

（匿名化のため簡略化しています）

```
if (ERR_OK == ret) {
  /* send message */
  ret = queue_snd(&msg,
                  sizeof(struct _msg));
}
```

# コピー＆ペースト (1)

- コピー



# コピー＆ペースト (2)

- 別の場所に貼付け

```
14  if (ERR_OK == ret) {
15    /* send message */
16    ret = queue_snd(&msg,
17                    sizeof(struct _msg));
18  }
```

:

```
35    if (ERR_OK == ret) {
36      /* send message */
37      ret = queue_snd(&msg,
38                      sizeof(struct _msg));
39    }
```

# コピー＆ペースト (3)

- 目的に合わせて少し修正

```
14  if (ERR_OK == ret) {
15    /* send message */
16    ret = queue_snd(&msg,
17                    sizeof(struct _msg));
18  }
```

:

```
35  if (ERR_OK == ret) {
36    /* receive message */
37    ret = queue_rcv(&msg,
38                    sizeof(struct _msg));
39  }
```

FIXSTARS Speed up your Business

# Refactoring しますか？

- 関数にくくり出す

```
14    ret = access_queue_if_ok(ret, &msg, queue_snd);
                    :
35    ret = access_queue_if_ok(ret, &msg, queue_rcv);
                    :
70  int access_queue_if_ok(int ret, struct _msg* msg,
71                         int (*fptr)(struct _msg*, size_t)) {
72    if (ERR_OK == ret) {
73      /* send/receive message */
74      return fptr(msg, sizeof(struct _msg));
75    }
76    else
77      return ret;
78  }
```

# Refactoring しないと後で泣く？

- エラーコードの仕様変更があったら
  - 片方だけ直して対応したつもりに

```
14  if (ERR_OK == ret) {
15    /* send message */
16    ret = queue_snd(&msg,
17                    sizeof(struct _msg));
18  }
19        :
35  if (ERR_OK == ret) {
36    /* receive message */
37    ret = queue_rcv(&msg,
38                    sizeof(struct _msg));
39    ret = new_return_code(ret);
40  }
```

修正もれ（バグ）

# Type 3 コードクローン

- ほぼ同一だが少しだけ異なるプログラム断片の組
  - Duplicated code とも呼ばれる
  - queue_snd/rcv の例も Type 3 クローン

- 数多くの検出ツールが存在
  - 大学や研究機関から
    - CCFinder, NiCad, SourcererCC, CCAligner, LVMapper, NIL, …
  - 商用製品（コード解析ツールの機能の一つとして）
    - SonaSource SonarQube, Synopsys Coverity

# 普及しないクローン検出

- 国内 ソフトウェア開発現場の関心は低い
- なぜか？
  - 検出されたクローンの扱いに困る
    - リファクタリングはしばしば困難
      - クローンによっては簡単に除去できない
      - 金融や官公庁向けは万一のリグレッションが...
    - 放置するなら誰が判断する？
  - 既存ツールは一般技術者には難しい
    - 別の可視化ツールと組み合わせる必要がある
    - 検出パラメータの調整が難しい
      - どれだけ違ったらクローンでなくなる？

# 普及しないクローン検出（続）

- なぜか？
  - プロジェクトマネージャ（PM）の無関心
    - PM は
      - プログラムの品質に最終責任を負う
      - クローン検出器などツールの導入の決定権をもつ
    - 現実の PM は
      - プログラムの品質にあまり関心がない
      - ツールの導入は実際の開発をする協力会社が考えれば良いと思っている

# CloneTracker

CloneTracker
Duplicated Code Management Software
https://clonetracker.com

- 実務で使われるクローン検出器をめざして開発中
  - エンジニアに Aha/Wow moment を届ける製品を
  - ひいては日本の 開発現場のコード品質への意識を高めたい

https://clonetracker.com

**「コピペが絡み合ったスパゲッティコード、
仕様変更は次から次へと入ってくる。
リファクタリングしたいがそんな時間もない。」
そんなあなたのために 「CloneTracker」 を作りました。**

# CloneTracker

CloneTracker
Duplicated Code Management Software
https://clonetracker.com

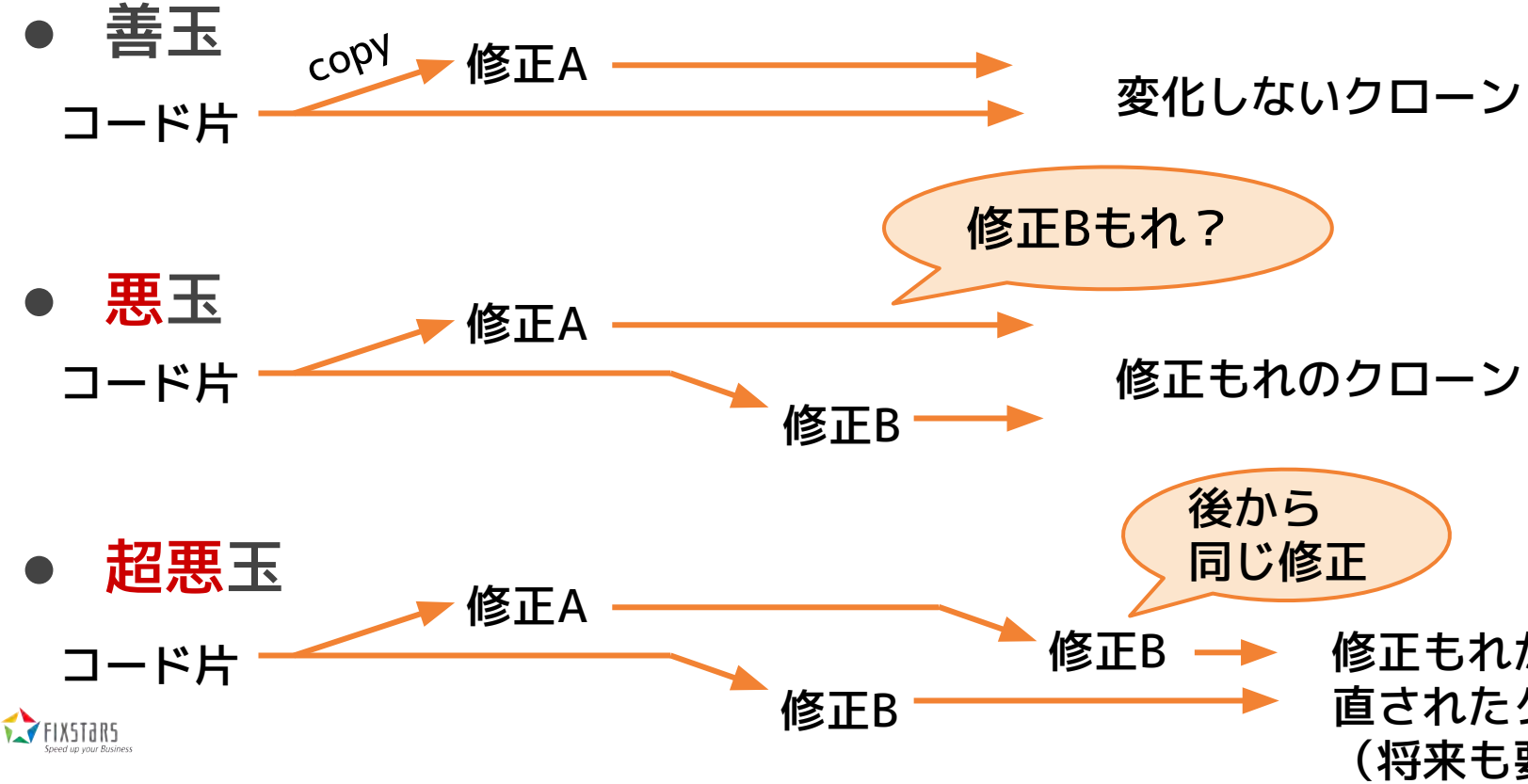
- エンジニアはコードクローンを放置してよい
  - 代わりにツールがクローンを継続的に追跡・管理
    - 存在を管理できればクローンは危なくない
    - Refactoring を無理に求めない

- 簡単に導入・利用できる
  - 可視化ツールつき
  - 細かい設定不要（パラメータ設定済みで提供）
    - 現状は git 上の C/C++ と C# プログラムに対応
      - 各言語ごとのチューニングをしているため

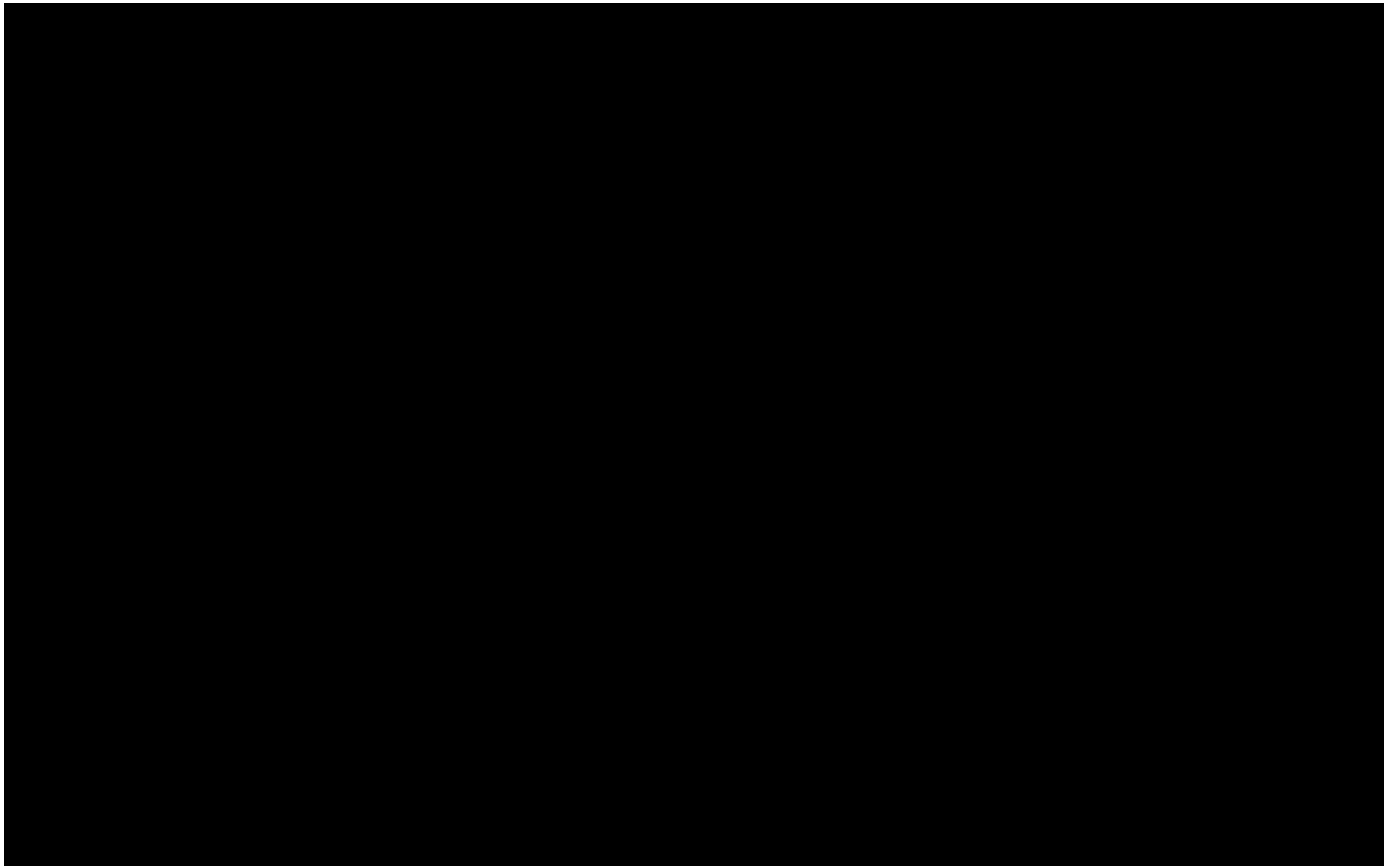
# クローンの修正履歴を追跡

- 最初の解析時には3ヶ月前から現在までを追跡
  - その後はコミットごとに差分を解析
- 修正履歴を分析して
  - 「修正もれ」の危険性が高いクローンを上位に表示
  - 「悪玉」の指摘
    - 過去に同じ修正がなされたクローンの組
    - 将来も同様の可能性あり、要監視

# 「善玉」コピペと「悪玉」コピペ



# デモ

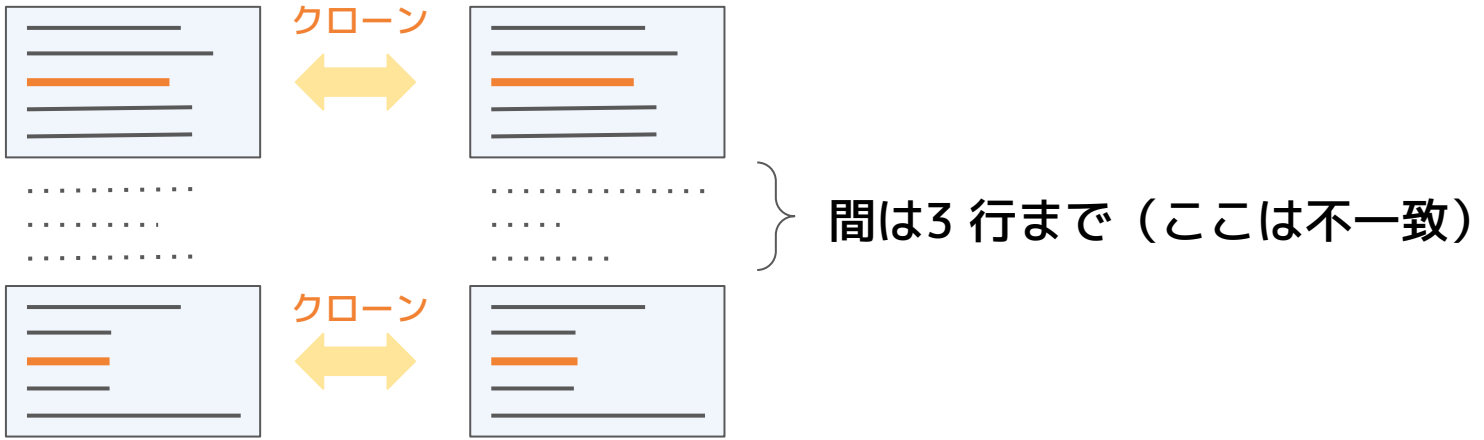


# コードクローン検出

1. トークン列を元に検出
    - CCFinder, CCAligner と同様
    - マクロ展開前のコードをトークン列に変換
2. 5 行の sliding window ごとにハッシュ値を計算
    - {, }, ; やカンマだけの行は 5 行に含めない
3. 中身が同じ 5 行の組を見つける
    - ハッシュ値かつ中央行の生テキストが一致するもの
    - 一致度の高い組（＝重要なクローン）だけを見つける

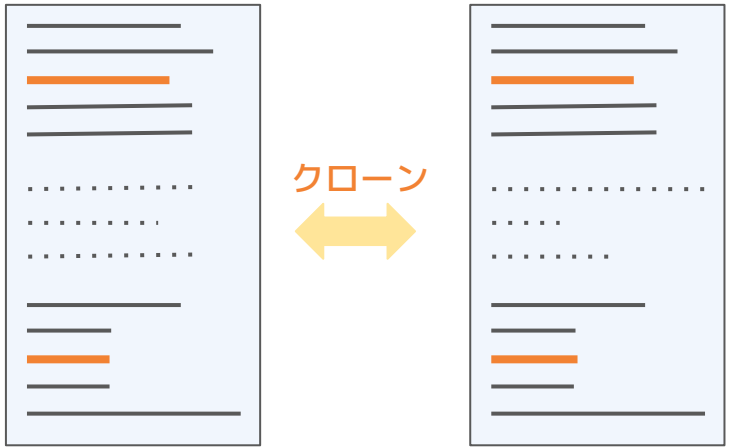
# 4. 近接するクローンの連結

- 連続 or 近接する５行の組を一つのクローンに連結



# 4. 近接するクローンの連結

- 連続 or 近接する５行の組を一つのクローンに連結




一組のクローン

# 見つけて欲しいコードクローンを見つける

- 重要度が高い
  - 行数が多い
  - 生テキストで比較して一致度が高い
  - if や while など制御構造に関わるトークンが多い
- 重要度が低い
  - 同じトークン列の行が繰り返す
  - 特定のトークン列や慣用表現を多く含む

# 特定のトークン列や慣用表現

- を多く含むクローンは重要度が低い
  - 変数宣言

    <名前> <名前> = <数値> ;

    `size_t a = 1;` と `size_t b = 3;` が同一となる

    変数宣言の並びを
    クローンと誤認

  - Java の慣用表現

```
public static void main(String args[]) {
  Options opts = Options.parseArgs(args);
```

# 実装

- スタンドアロン Desktop App.
  - 外部にソースコードを転送する必要なし
- Tauri　（Electron から乗換えた）
  - Rust + TypeScript
  - Clang の字句解析器を Emscripten+WebAssembly で実行
  - コードクローン検出器は TypeScript で実装

# 初回の解析時間

- 現在、三ヶ月前、その中間の３つの版を一度に解析

| | Apache | 1MLOC (Java) | OpenSSL |
|---|---|---|---|
| Core i9+32GB | 2m 55s | 3m 26s | 9m 55s |
| Core i7+16GB | 4m 33s | 6m 29s | 17m 08s |
| Apple M2+8GB | 5m 54s | 9m 29s | 28m 53s |
| 行数 | 351,414 | 1,445,144 | 736,033 |
| ファイル数 | 583 | 12,381 | 1,984 |

1MLOC は現在の版のみ解析

# 関連研究（クローンの修正履歴を追跡）

- CCEvovis [Honda et al., ICPC’19]
  - ２つのコミットの違いを分析し「修正もれ」を指摘
- Clonetracker [Duala-Ekoko et al., ICSE’08]
  - Eclipse plug-in
  - クローンの一方をエディタで修正しようとすると、他方も合わせて修正するよう促す

CloneTracker は git と連携して多数のコミットを継続的に追跡して分析、他のヒューリスティックスを加えて、重要度を判定

# まとめ

- CloneTracker の紹介
  - コードクローンを検出
  - 修正履歴を継続的に追跡、クローンを管理
  - 修正もれ、悪玉など、重要度つきでクローン一覧を表示
- ベータ版無償公開中
  - Windows, Mac, Linux
  - C/C++ または C# のみ　（他言語は設定を変更すると動く）
  - https://clonetracker.com/