

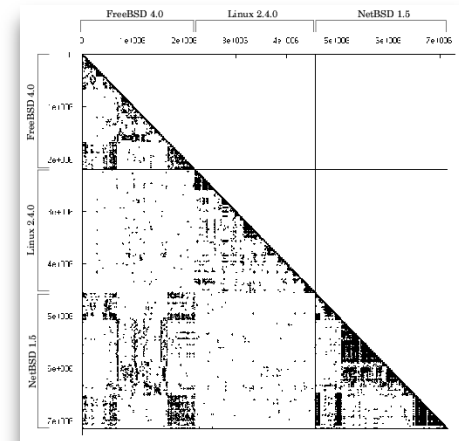
研究発表（CCTの解析結果）

久木田 雄亮

背景

- ソフトウェア開発・保守の過程では，生産性の向上を目的として既存コードが頻繁に再利用されることがある

- 同一のまたは類似するコード片がソースコード全体に遍在することが少なくない



UNIX系OS, Linux間のコードクローン[1]

- 欠陥除去などの理由によりコードクローンに変更を加える場合
 - 変更箇所が多岐に渡るため通常よりも保守コストが増加する
 - 変更漏れにより新たな不具合を混入してしまう

研究の動機

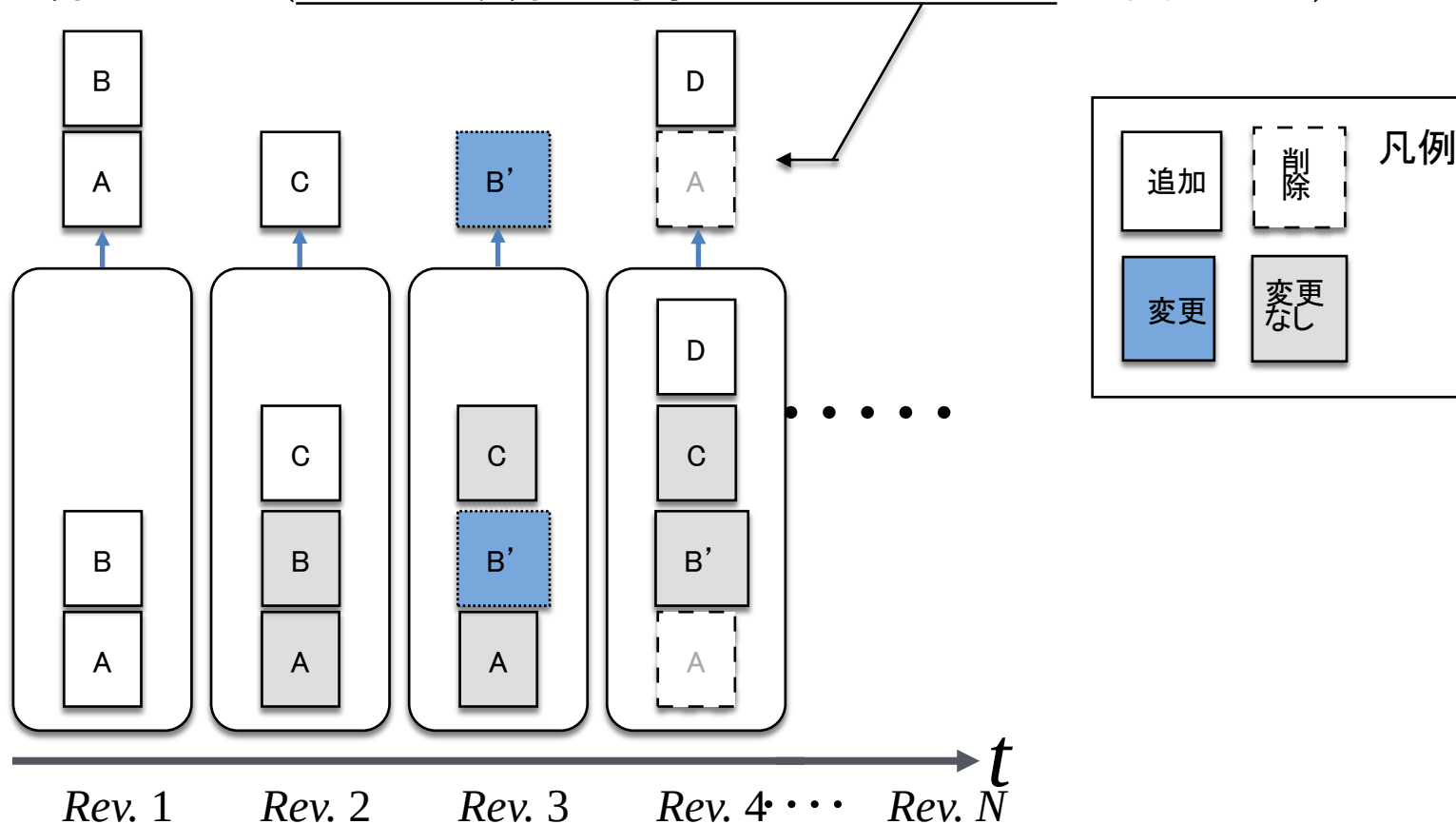
- コードクローンの作成過程とその影響については十分には明らかになっていない
- 特に、コードクローンの作成・利用に関与した開発者とSW品質の関係に着目した分析はほとんどおこなわれていない
 - 特定の開発者が作成したコードクローンと後に発見される不具合との関係
 - 開発者間でのコードクローンの利用（共有）とソフトウェア品質との関係， など
- **コードクローンの長期的かつ安全な管理方法について有益な知見を提供したい**
 - 近年のソフトウェアシステムは長期に渡り保守運用されることが多く担当者が入れ替わることも少なくない

CCT

- コードクローンの作成・利用過程における人的影響の調査を支援するためのツール
- CCTの処理手順
 1. コード片を特定する処理
 2. クローンを特定する処理
 3. クローンを追跡する処理
 4. クローンの作成・利用に関与した開発者の特定処理
 5. 不具合が混入したクローンを特定する処理

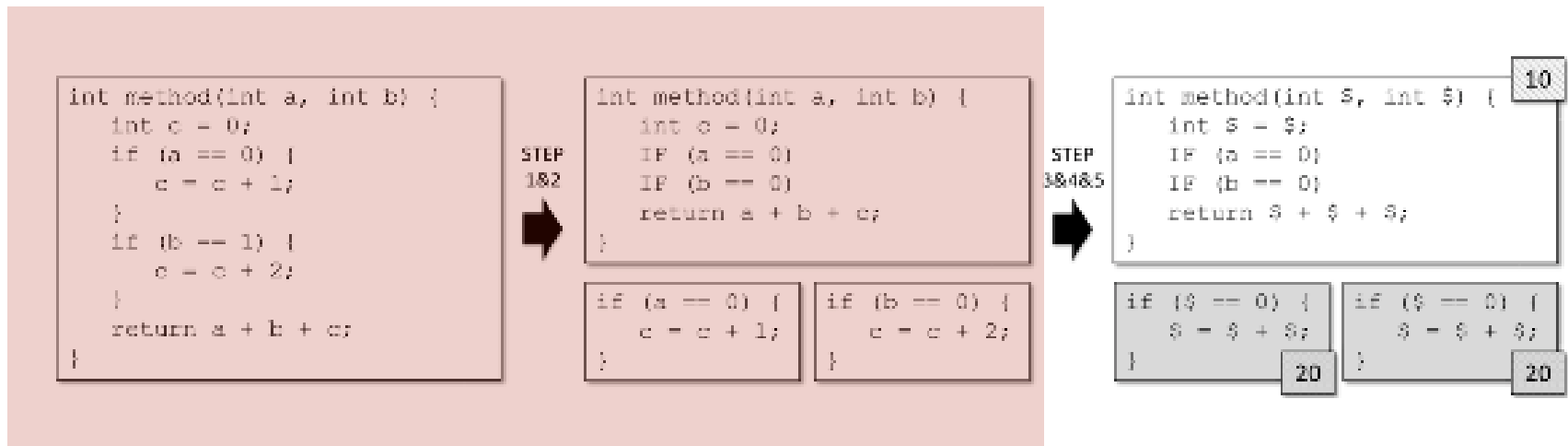
CCTの処理手順（１）：コード片の特定

- 分析対象ファイル中に含まれるすべてのコード片を特定
 - 追加・変更・削除されたファイルをすべてのリビジョンに対して特定する（コード片特定対象となるファイルを抽出する）



CCTの処理手順（１）：コード片の特定

- 分析対象ファイル中に含まれるすべてのコード片を特定
 - 抽出したファイル差分を対象にブロック単位でコード片を特定
 - ブロック：クラスやメソッド，for文やif文などのブロック文



ブロックの検出・正規化・ハッシュ化の流れ

CCTの処理手順（２）：クローンの特定

1. ブロックに含まれる変数名とリテラルを特殊文字で正規化
2. 正規化されたブロックの文字列をもとにハッシュ値を算出

```
int method(int a, int b) {  
    int c = 0;  
    if (a == 0) {  
        c = c + 1;  
    }  
    if (b == 0) {  
        c = c + 2;  
    }  
    return a + b + c;  
}
```

STEP
1&2
➡

```
int method(int a, int b) {  
    int c = 0;  
    IF (a == 0)  
    IF (b == 0)  
        return a + b + c;  
}
```

```
if (a == 0) {  
    c = c + 1;  
}
```

```
if (b == 0) {  
    c = c + 2;  
}
```

STEP
3&4&5
➡

```
int method(int $, int $) {  
    int $ = $;  
    IF {a == 0}  
    IF {b == 0}  
        return $ + $ + $;  
}
```

```
if {$ == 0) {  
    $ = $ + $;  
}
```

```
if {$ == 0) {  
    $ = $ + $;  
}
```

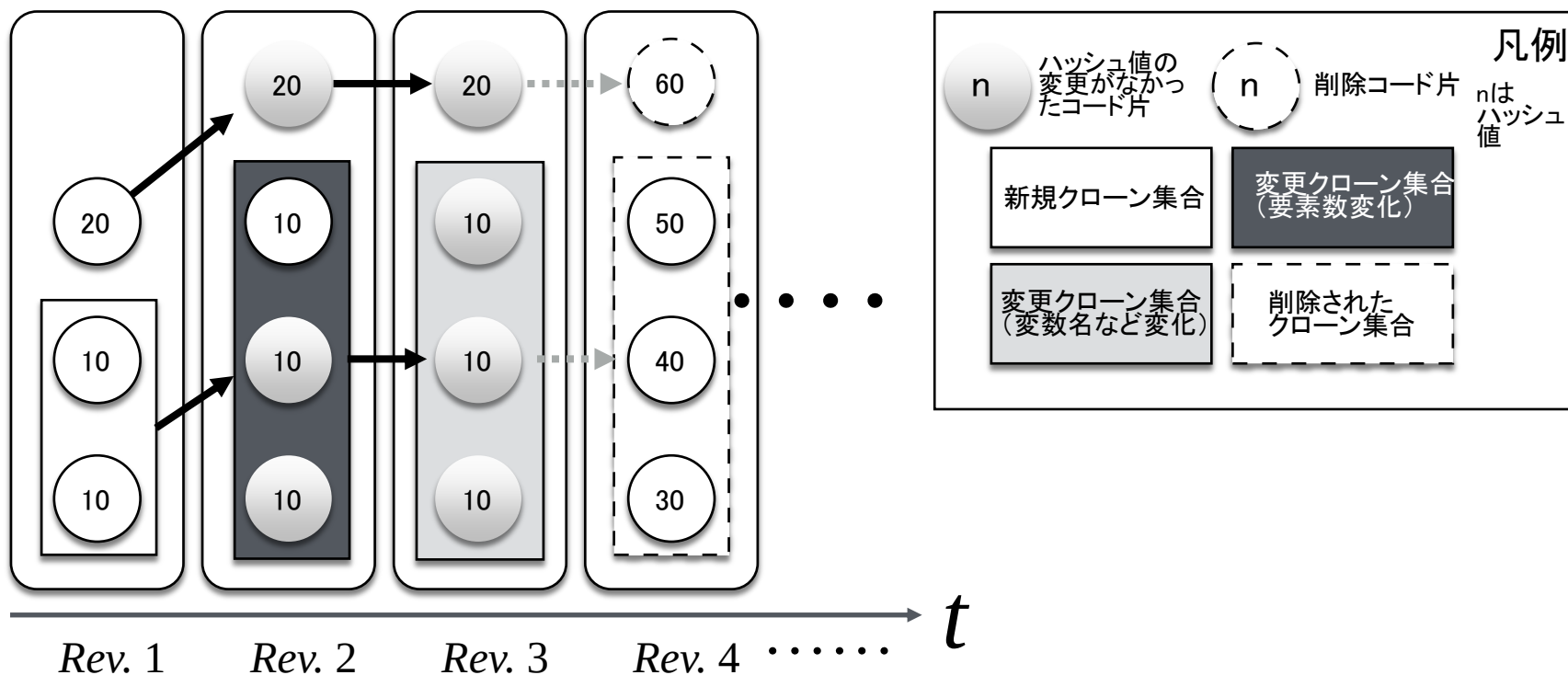
10
20
20

ブロックの検出・正規化・ハッシュ化の流れ

3. ハッシュ値の一致するブロックをクローンと見なしクローンを特定

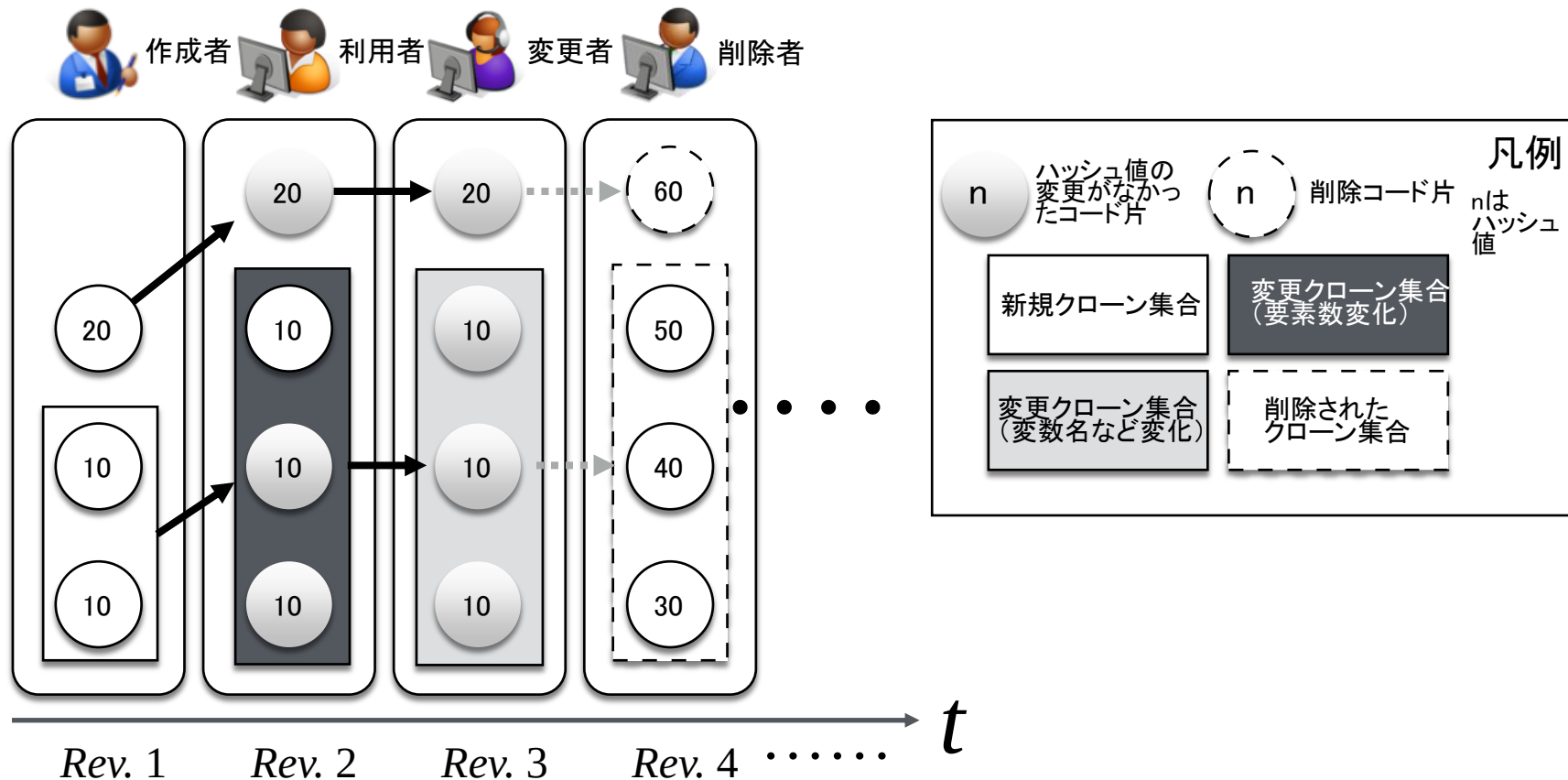
CCTの処理手順（3）：クローンの追跡

- リビジョン間でのクローン集合の前後関係を追跡できるようにする
 - 追加・変更・削除されたコード片のハッシュ値を調べ、直前のリビジョンのクローンのハッシュ値と比較する



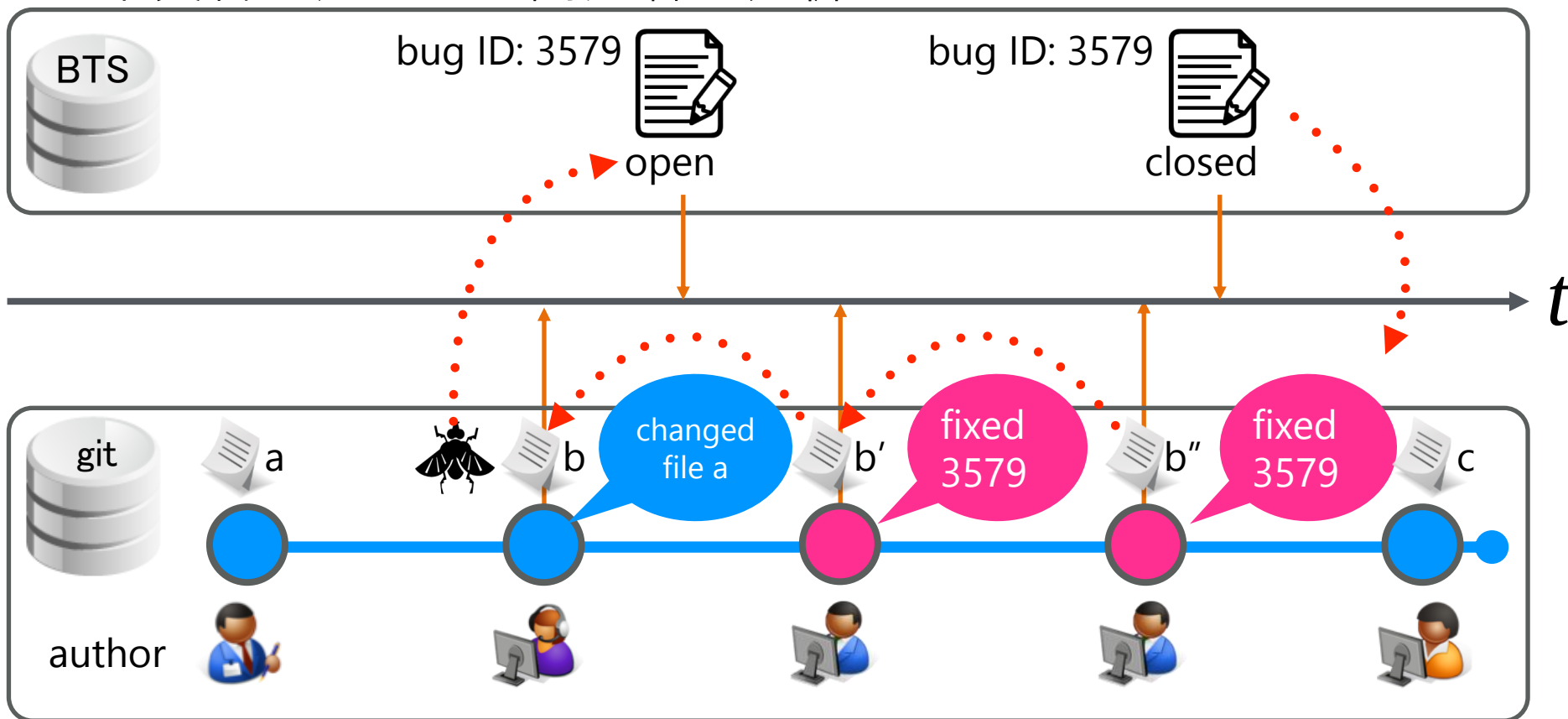
CCTの処理手順（４）：開発者の特定

- クロンの作成・利用過程に関与した開発者を4種類に分けて特定する



CCTの処理手順（５）：開発者の特定

- SZZアルゴリズムを用いて不具合混入コミットを特定し、不具合を混入した開発者を分析する



分析項目

- コードクローン追跡・分析を大規模に行う（GitHubのプロジェクトを対象とする）
- コードクロンの作成・利用に関与した開発者とコードクローン品質との関係をより詳細に分析する
 - i. 作成・利用に関与した開発者の数とソフトウェア品質
 - ii. 開発者の社会的関係とソフトウェア品質
 - iii. コードクローンの変更間隔および変更担当者とソフトウェア品質

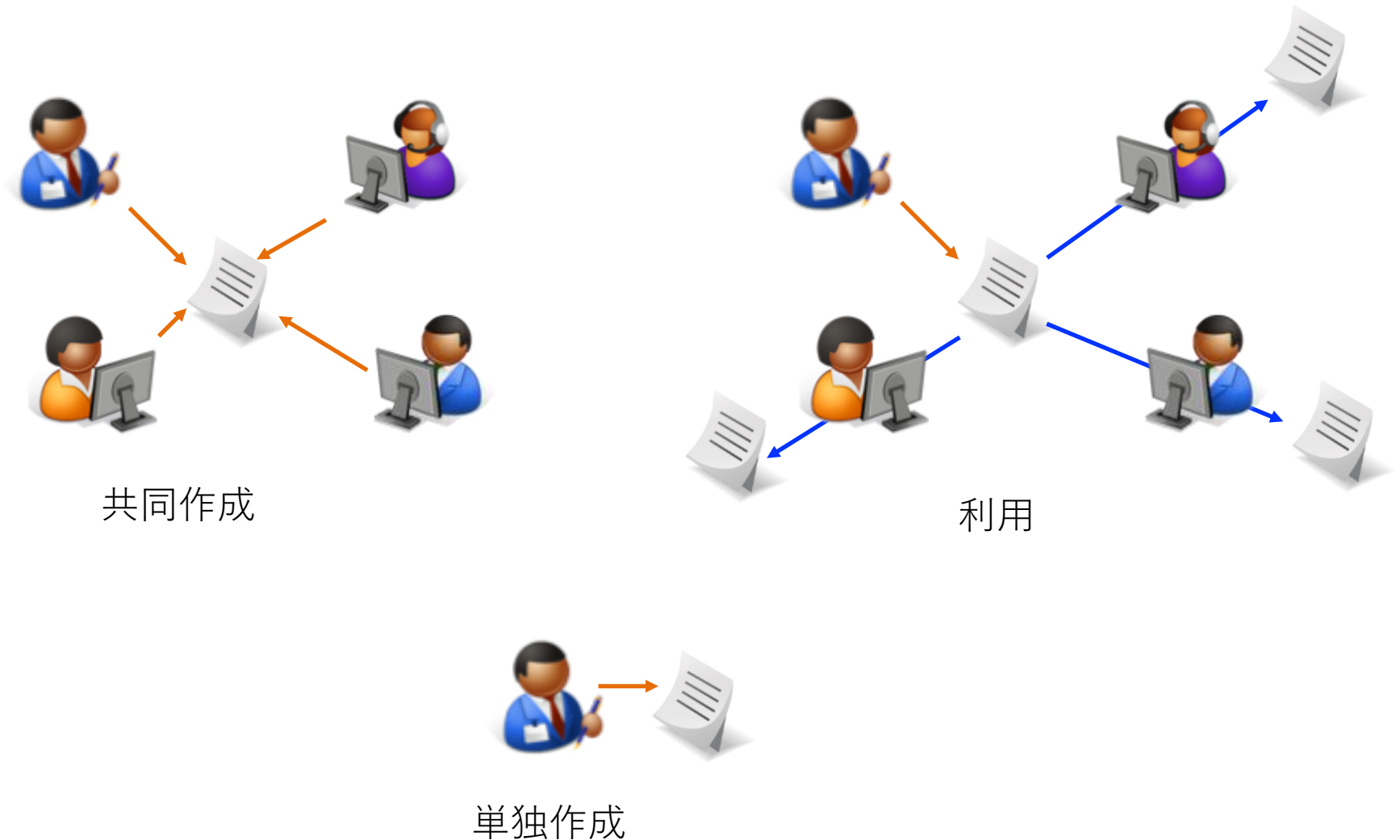
対象プロジェクト

プロジェクト	データ取得期間	リビジョン数(*1)	最新リビジョンのファイル数	クローンセット数 (*2)	クローン利用者/ 全開発者数	不具合混入クローンセットの修正数/ 全不具合報告数
Apache Ant	2001/01~2012/07	944	839	219	12/55	82/5,658
Apache Struts	2006/02~2016/04	4,375	740	81	4/61	112/4,622
CheckStyle	2001/06~2016/07	5,968	336	61	7/61	37/3,329
Jabref	2003/10~2016/05	7,942	723	93	11/117	122/1,529
OrientDB	2010/03~2016/06	11,949	1,119	365	27/132	1,863/6,369

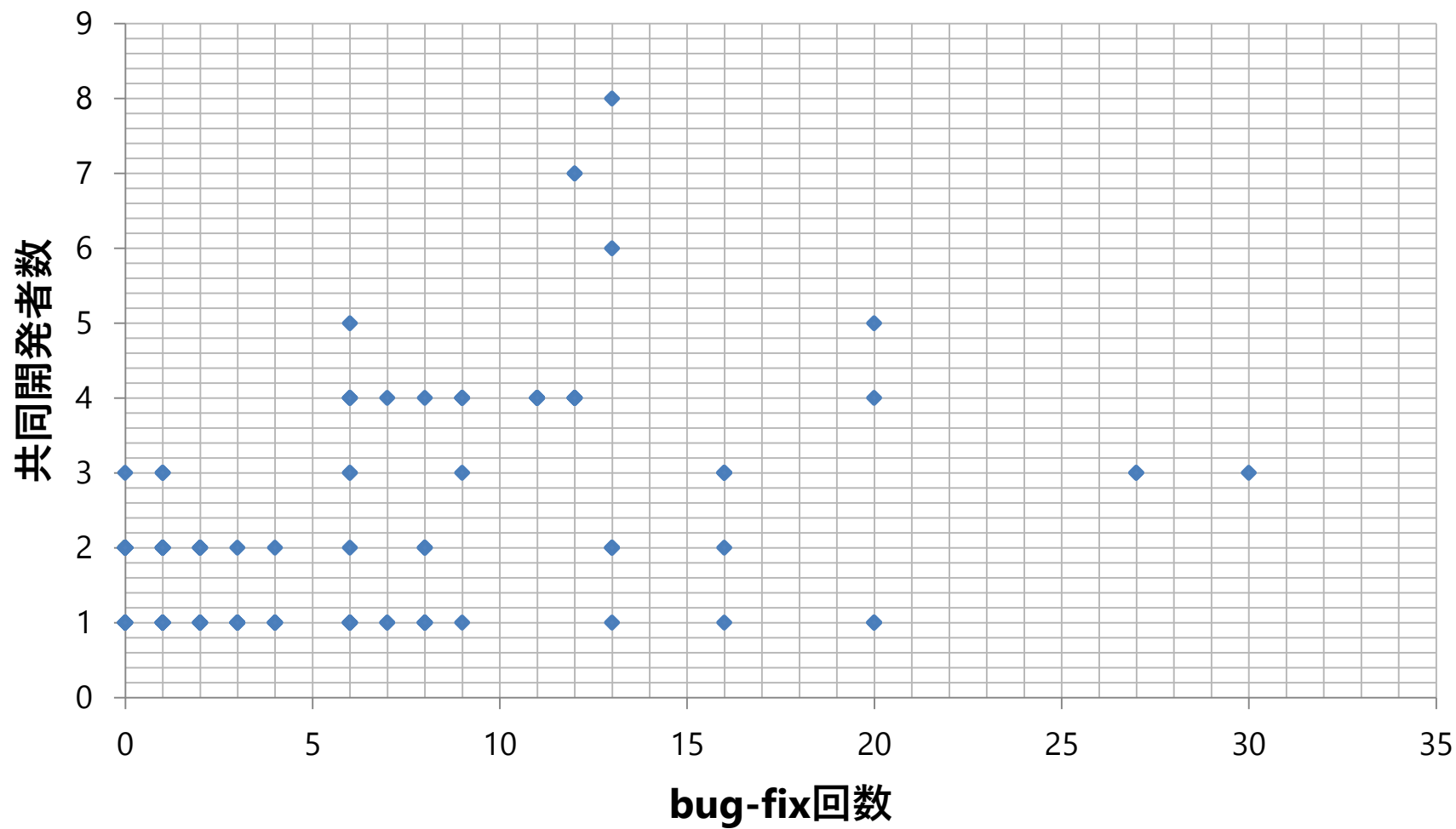
(*1) 対象ディレクトリ内のファイル変更時のみカウント

(*2) 最新リビジョンに含まれるクローンセット数

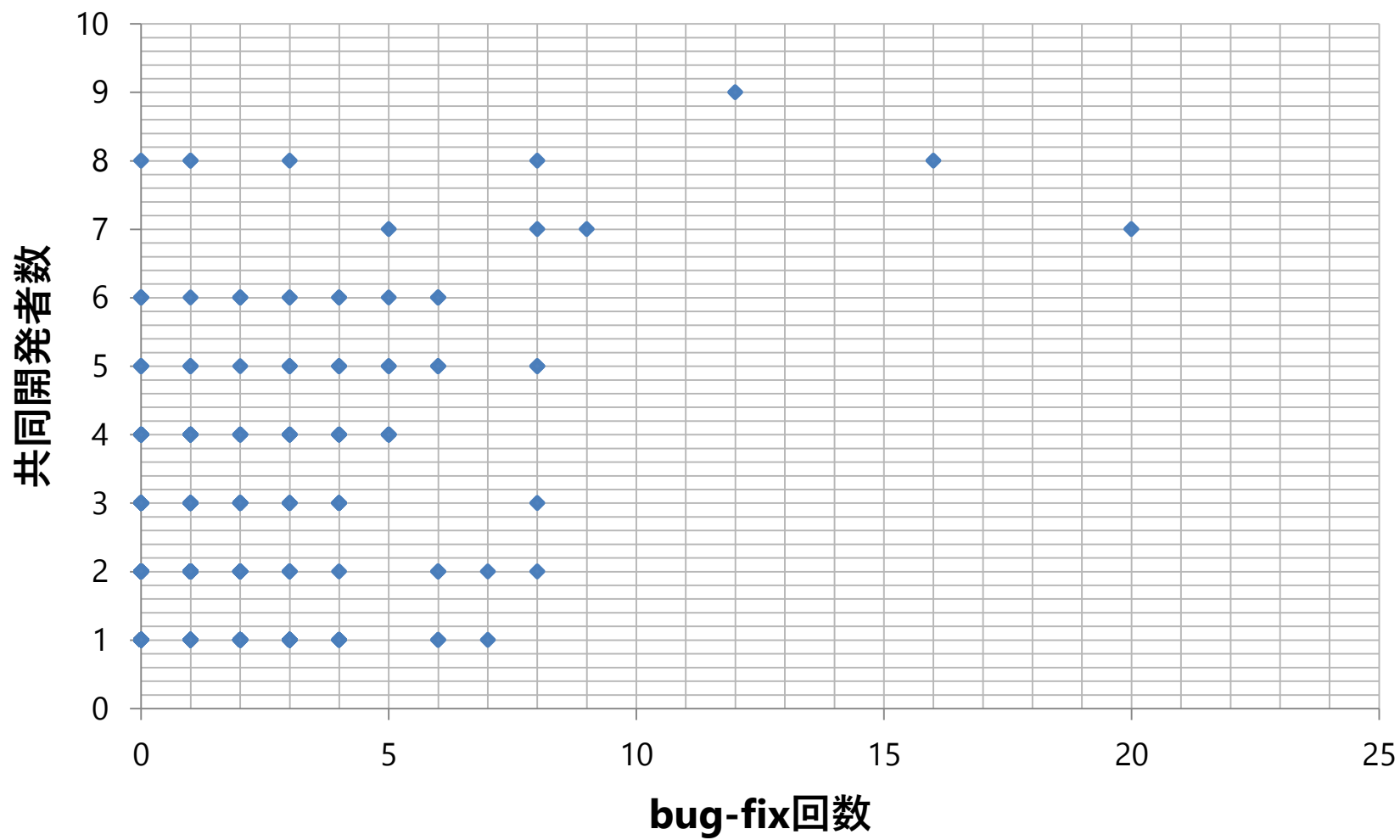
i. コードクローンの作成・利用に関与した開発者の数とソフトウェア品質



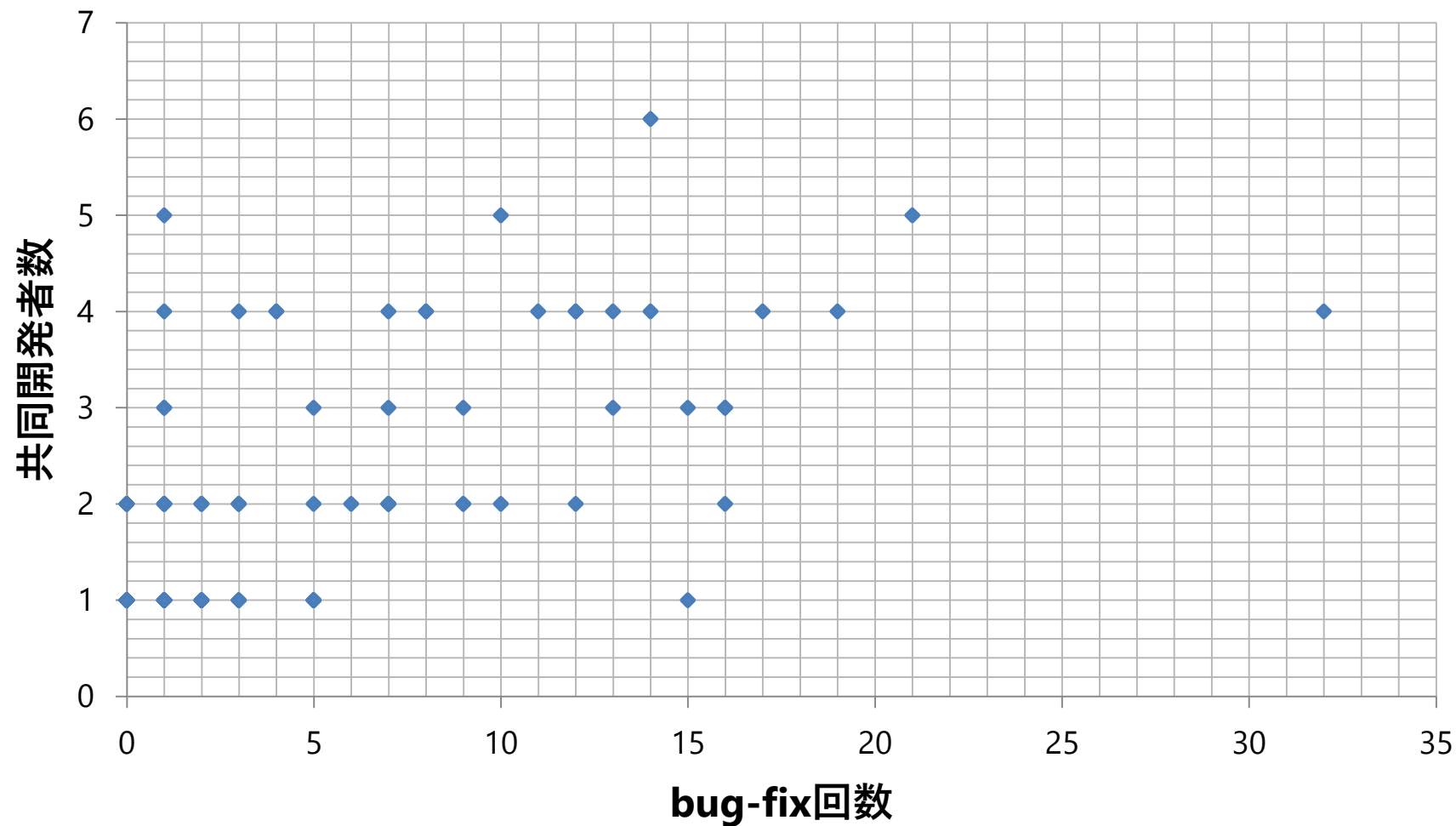
Struts



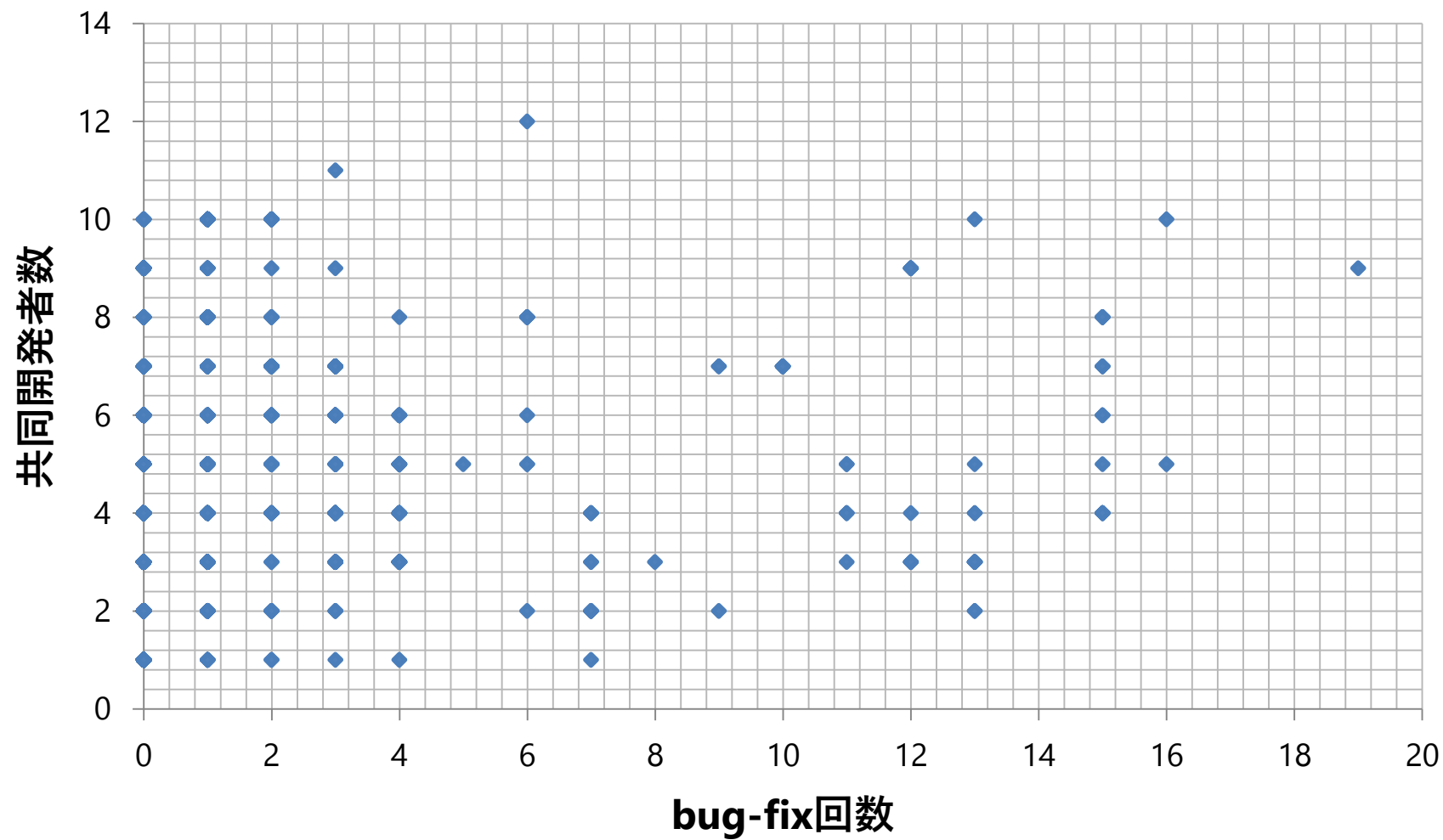
Ant



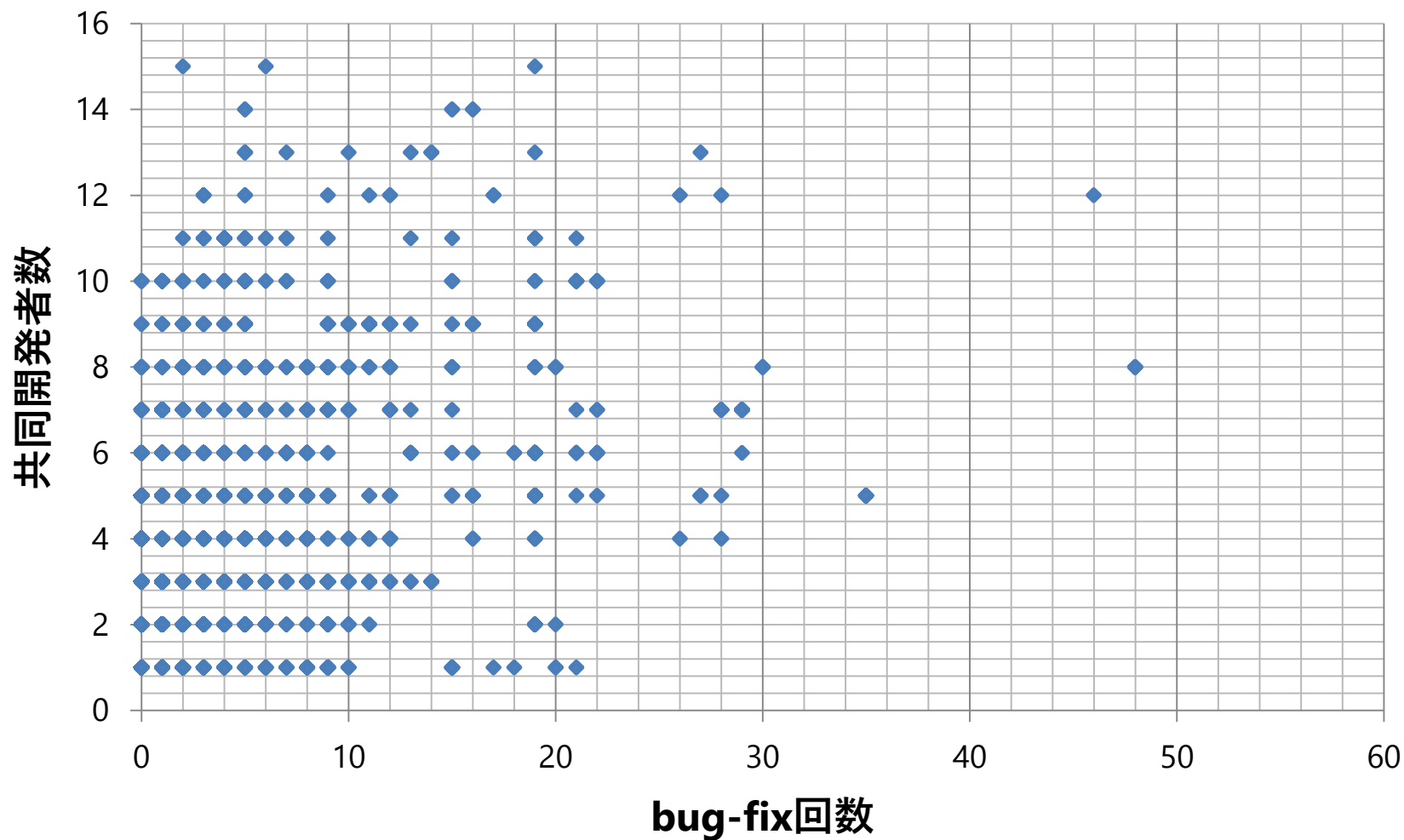
Checkstyle



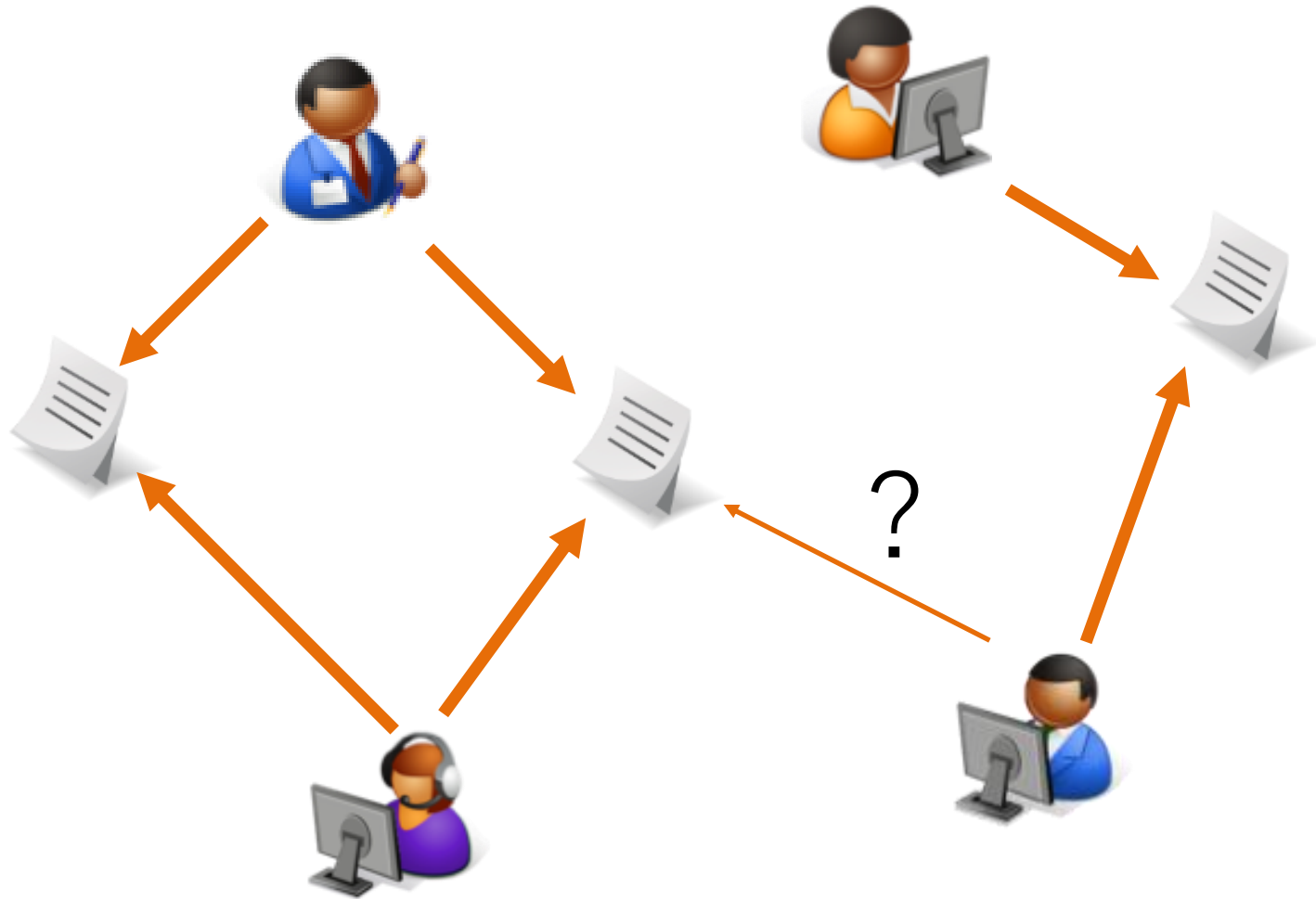
Jabref



OrientDB

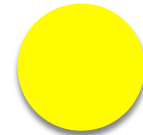


ii. 開発者の社会的関係とソフトウェア品質

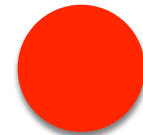


pajek(struts)

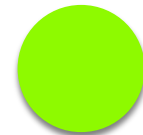
凡例



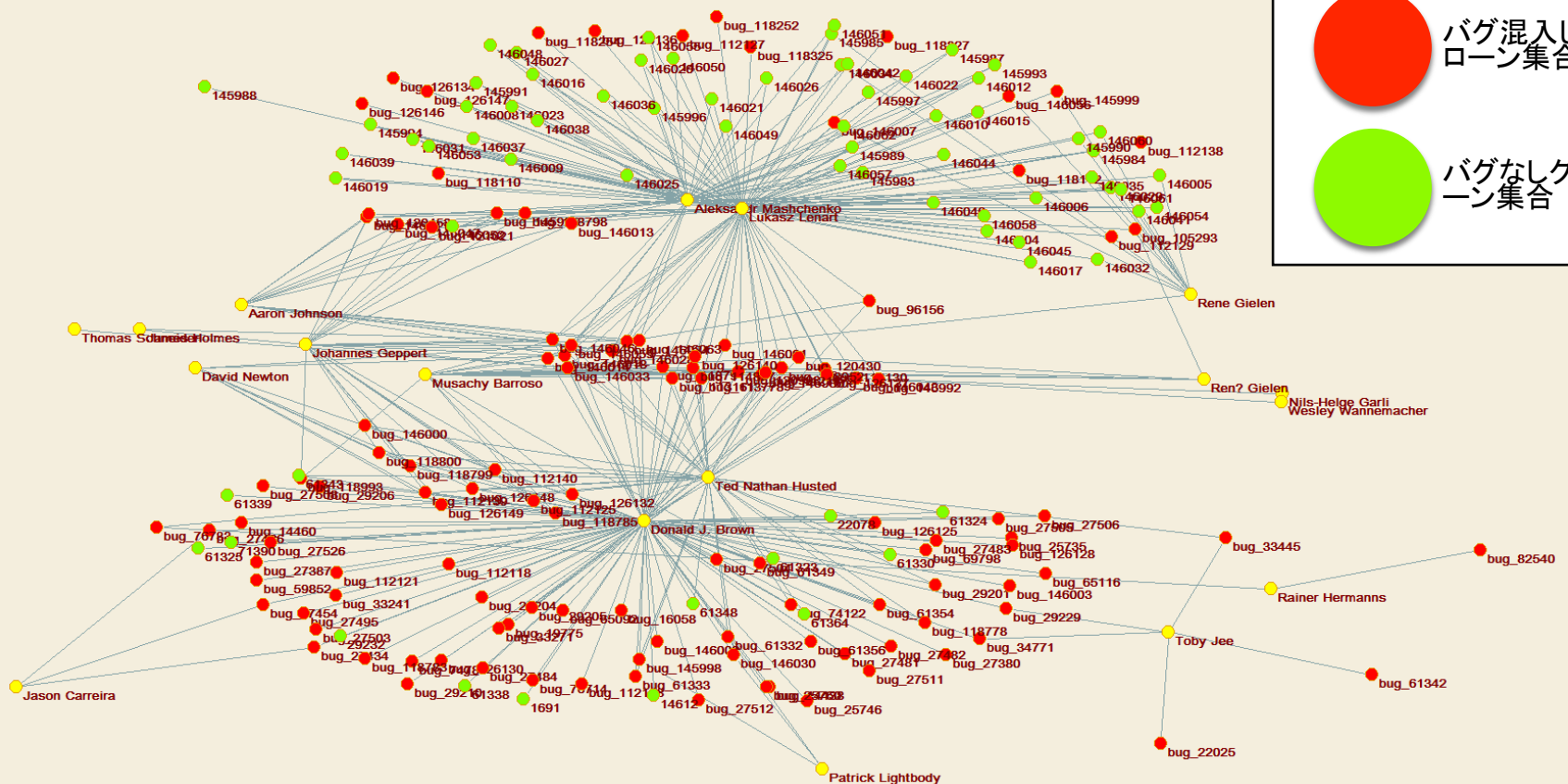
開発者



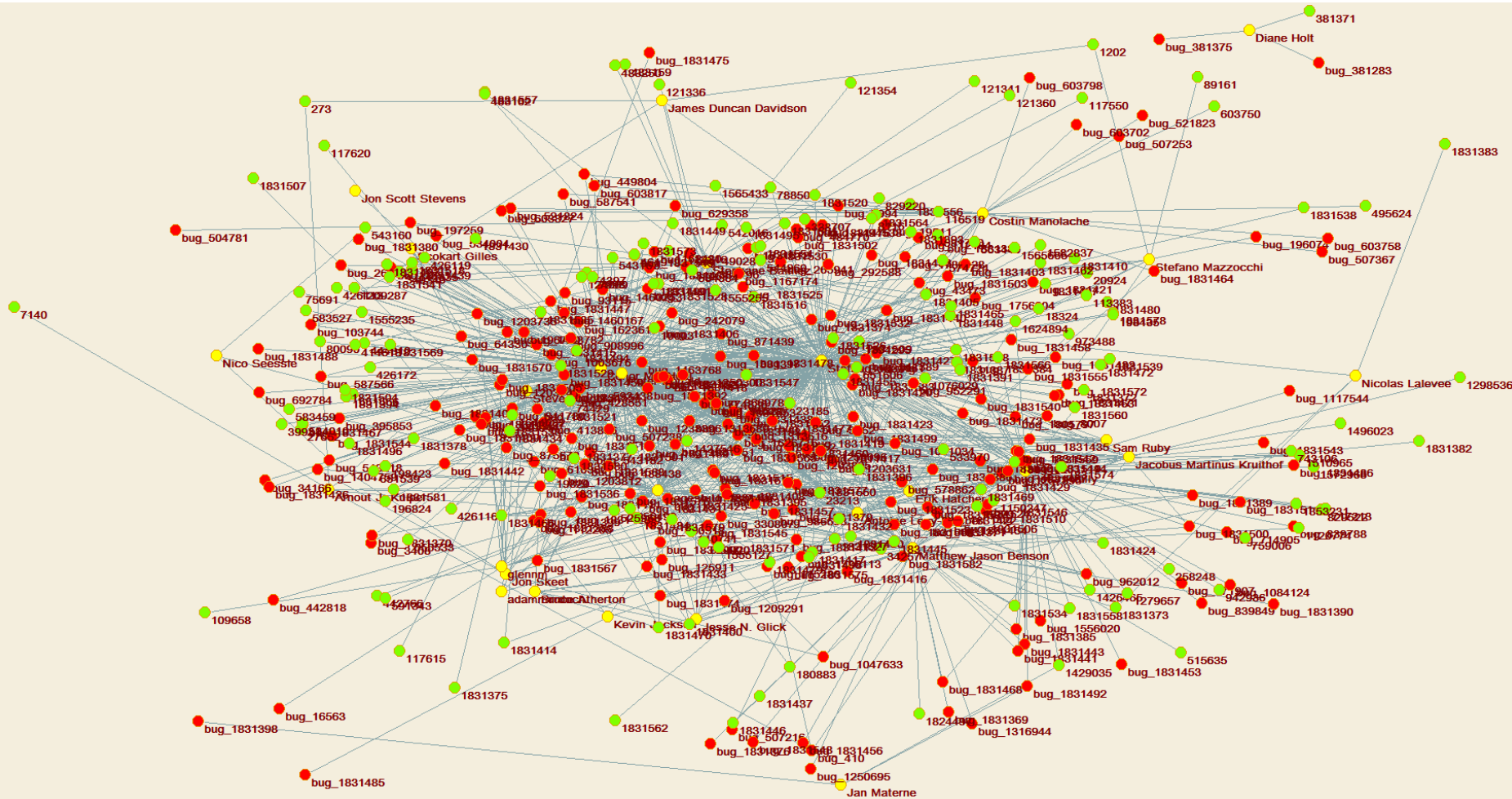
バグ混入しクローン集合



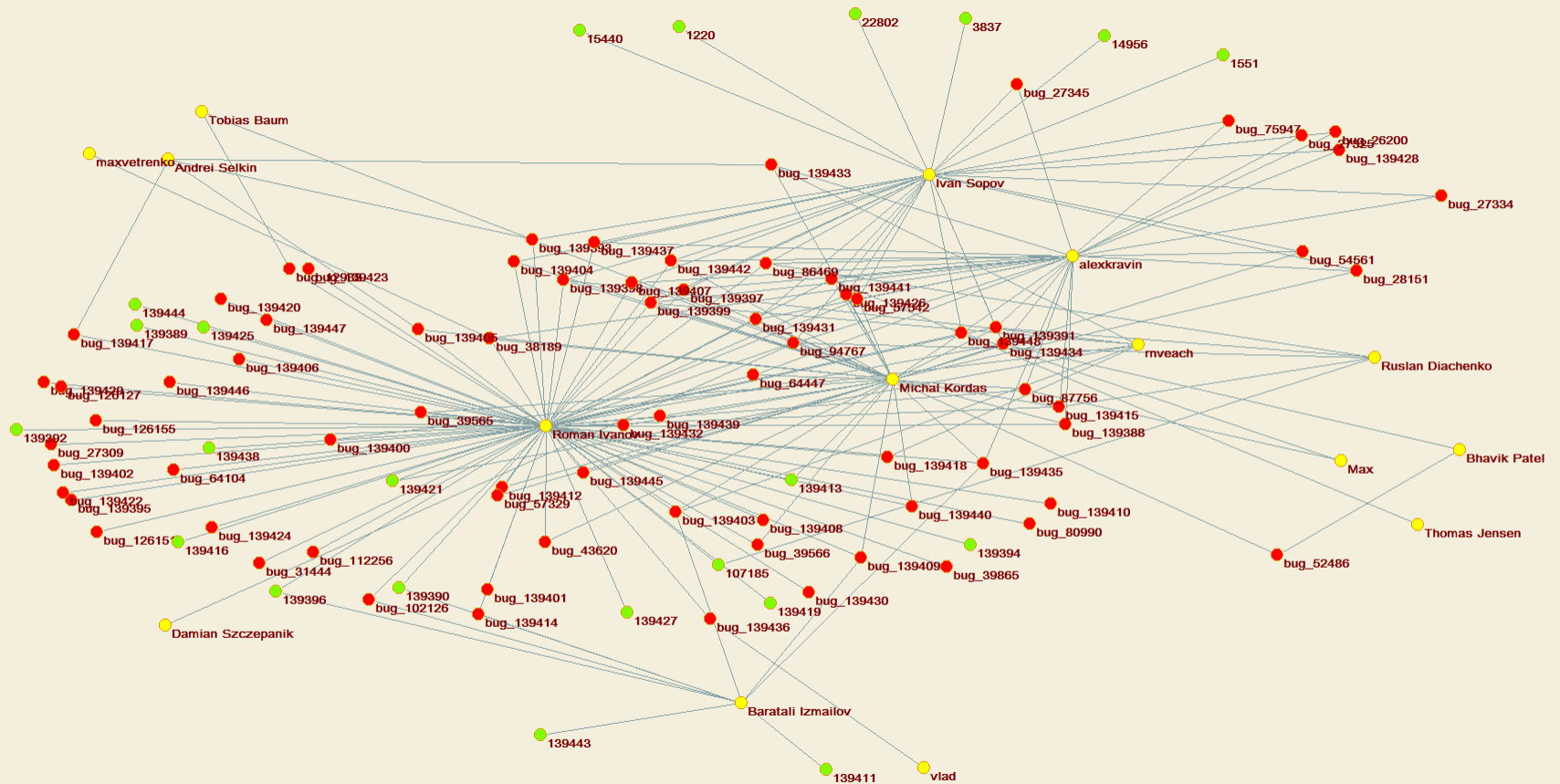
バグなしクローン集合



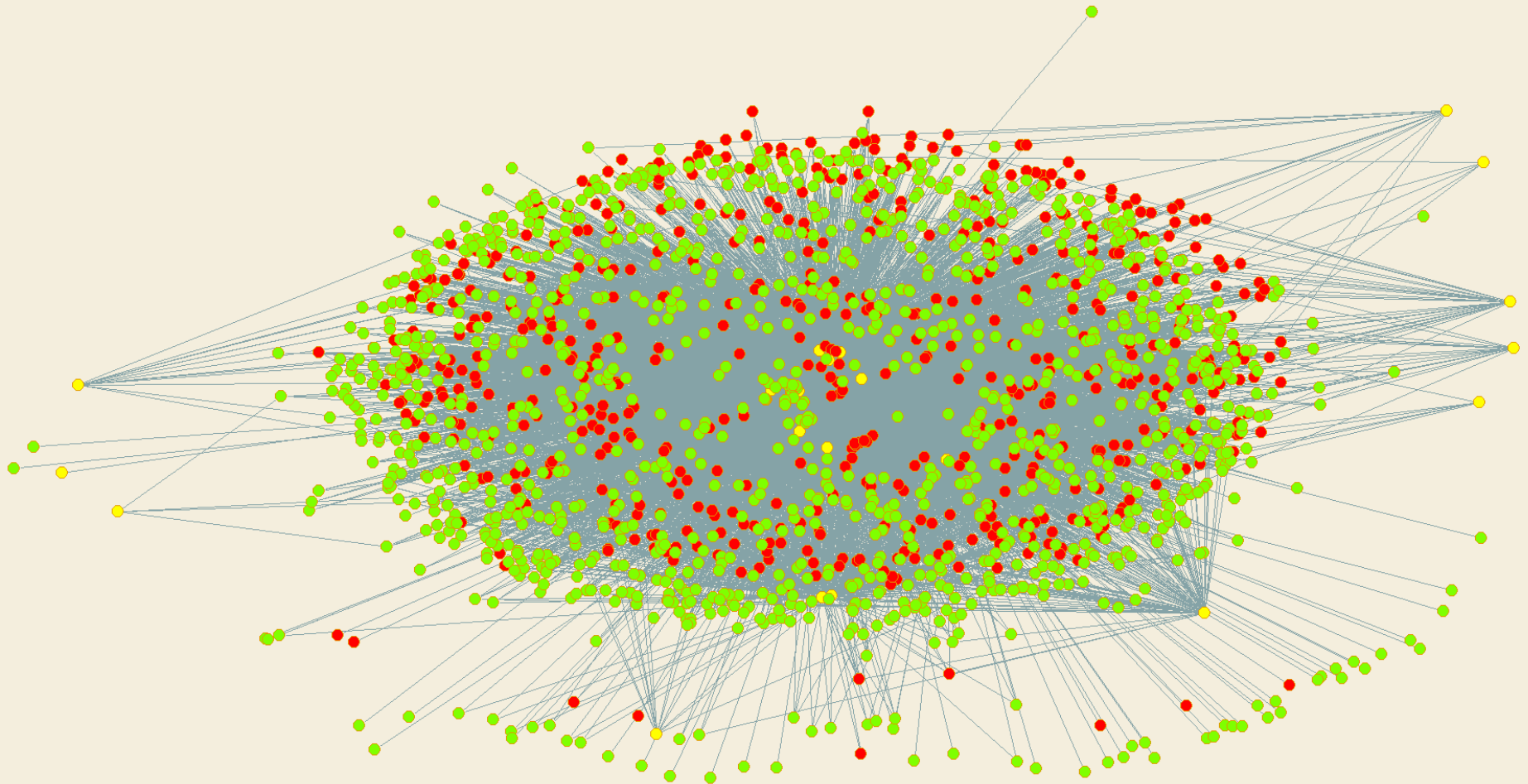
pajek(Ant)



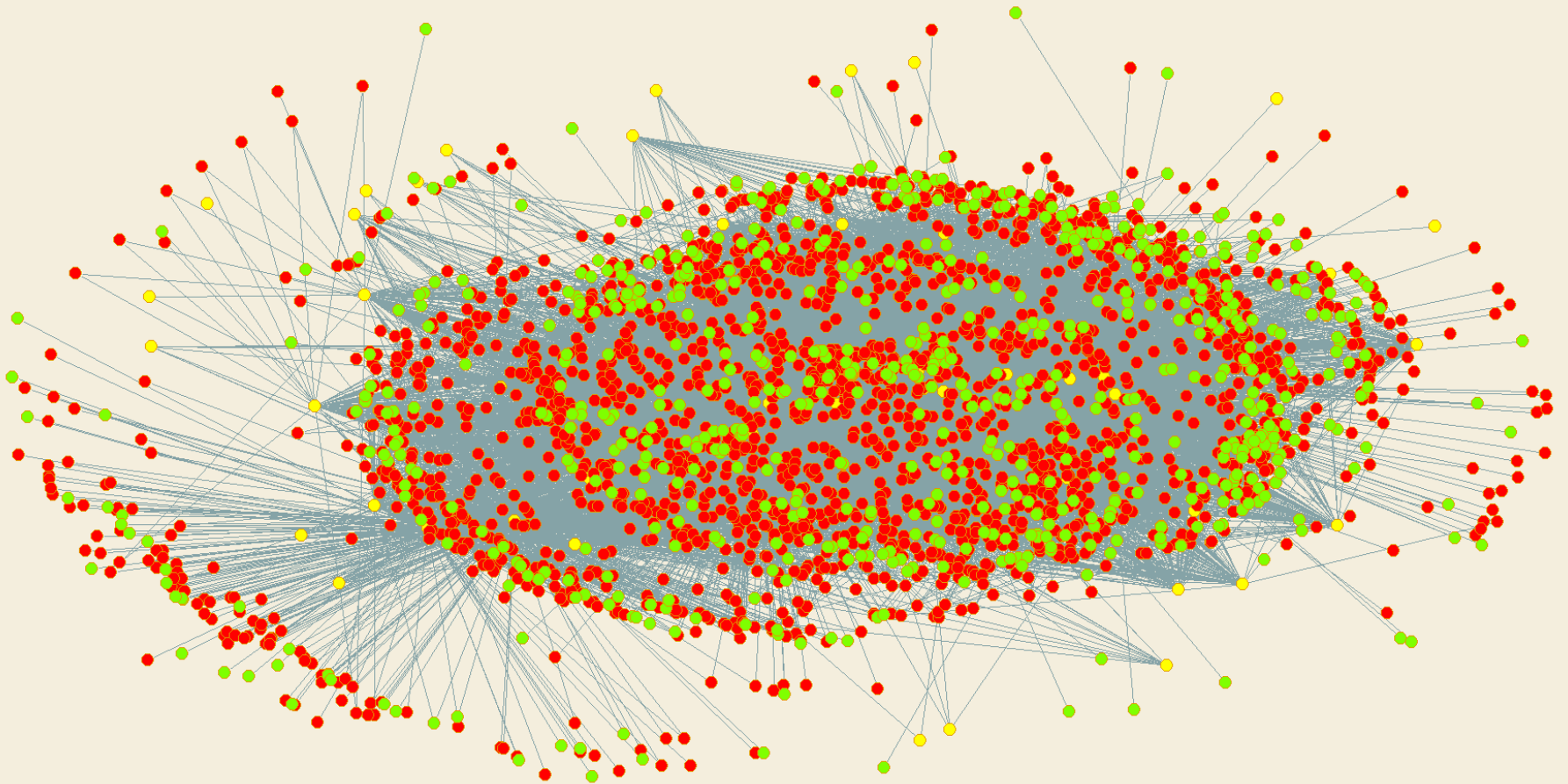
pajek(Checkstyle)



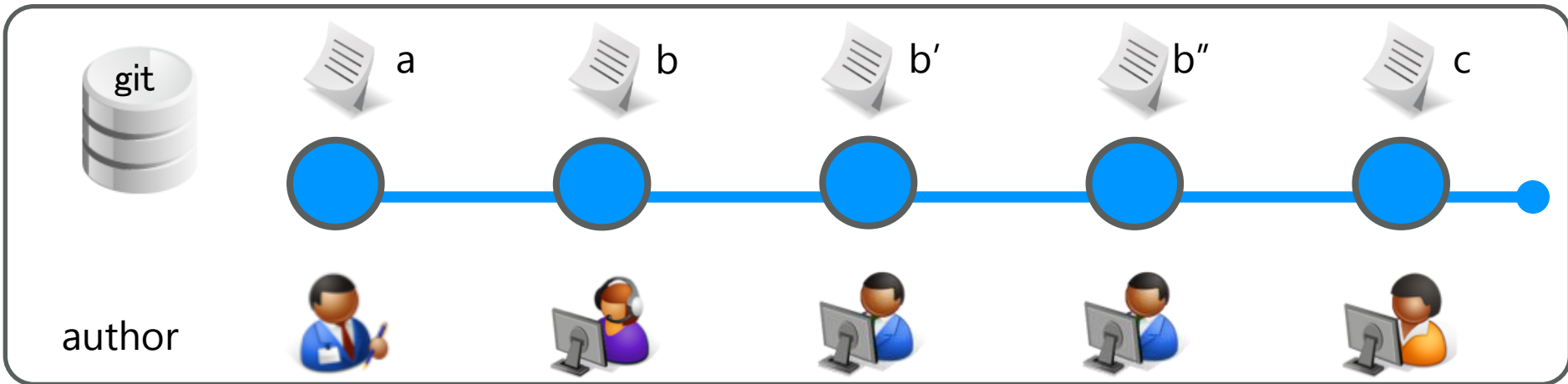
pajek(Jabref)



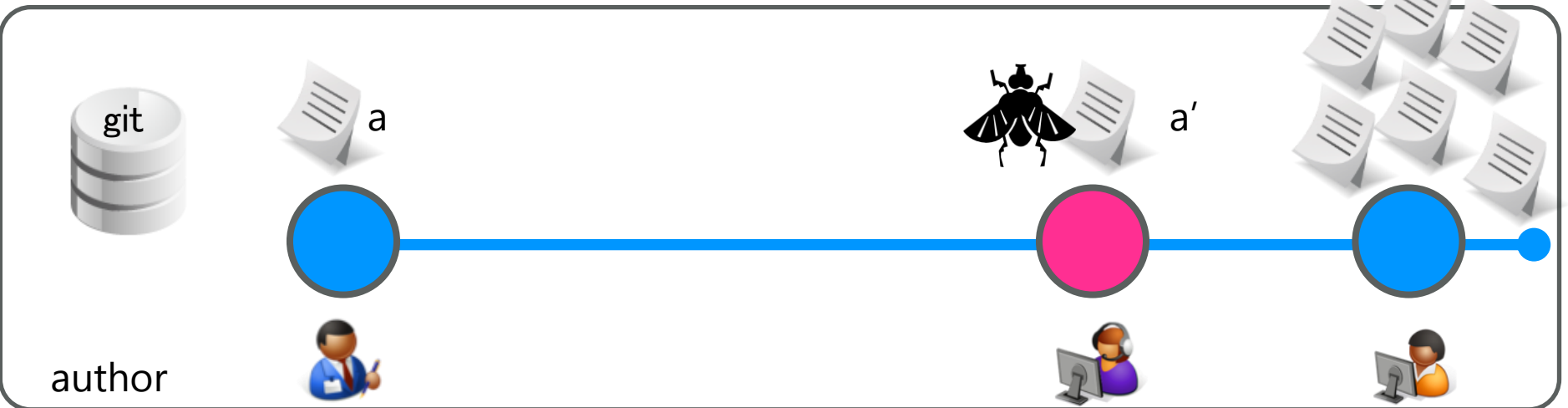
pajek(orientdb)



iii. コードクローンの変更間隔および 変更担当者とソフトウェア品質



V.S.



iii. コードクローンの変更間隔および
変更担当者とソフトウェア品質

未完了

