

コードクローンに関わる特徴を取得するための可視化ツールの提案(仮)

大石 聖也

コードクローン

- ソースコード中に存在する, 互いに「一致」または「類似」するコード片
- 生産性向上のため, 既存コードの再利用が繰り返され, ソースコード全体に広がっている

コードクローンの問題

- ソフトウェアの品質を低下させる要因と考えられている
（例）一部を変更する際,コードクローン全体を変更しないといけない
 - コストがかかる
 - 変更忘れから, 新たな不具合が発生
- ソフトウェアの品質を低下させる要因として, 様々な研究がされてきた[1]

[1] 肥後 芳樹, 楠本 真二, 井上 克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌D, Vol.J91-D, No.6, pp.1465-1481, 2008年6月

既存研究

- コードクローン分析に関する研究[2]
 - 生存期間が長く, 複数回修正されるコードクローンがどの程度存在するかを調査
- コードクローン追跡に関する研究[3]
 - コードクローンの作成・利用過程における人的影響を調査することを目的として, コードクローン追跡ツールCCTを設計

[2] 堀田 圭佑, 肥後 芳樹, 楠本 真二. CRDを用いたコードクローンの生存期間と修正回数に関する調査. 情報処理学会論文誌 Vol.55 No.2 947-958 (Feb. 2014)

[3] 久木田 雄亮, 大平 雅雄. コードクローン変更過程における開発者のインタラクションとソフトウェア品質の関係. ソフトウェア・シンポジウム2017 in 宮崎. pp.120-129, 2017年6月

CCT (Code Clone Tracer)

- 対象
 - Gitなどの分散型バージョン管理システム
 - 機能
 - クローンの特定・追跡を行う機能
 - 不具合管理システムと連携する機能
- 開発者・不具合・コードクローンの紐づけが行える

CCTの利点

- 開発者・不具合・コードクローンの紐づけ
 - コードクローンの作成・利用過程における特定の開発者が及ぼす影響を調査できる
- 実際に, コードクローンを生むのは開発者自身
 - コードクローンが持つ特性を知るだけでなく, コードクローンに関わる開発者について, 調査が必要

CCTの問題点

- 検出結果がcsvファイル形式
 - ソフトウェアに対する, コードクローンの利用過程を俯瞰的に認識できない
- コードクローンの利用過程を十分に認識できず, コードクローンと開発者に関する特徴を分析できない

目的

- 目的
 - コードクローンの利用過程で, コードクローンから不具合を発生させる開発者の特徴を分析するための可視化ツールの構築



ツール概要

- 時系列表示
 - コードクローンから不具合を発生させている開発者を特定
- ファイル表示
 - 不具合を含むファイルの特定
- コード表示
 - 不具合を発生させた開発者が, コードクローンに加えた変更を確認

時系列表示

- x軸: 時間
- y軸: クローン密度
 - 各リビジョンの全コード行数にクローンがどれだけ含まれているか
 - $\text{クローン密度} = \frac{\text{全クローン行数}}{\text{全コード行数}}$
- 各ノード: リビジョン
 - 赤: コードクローンと不具合を結びつけたリビジョン

時系列表示

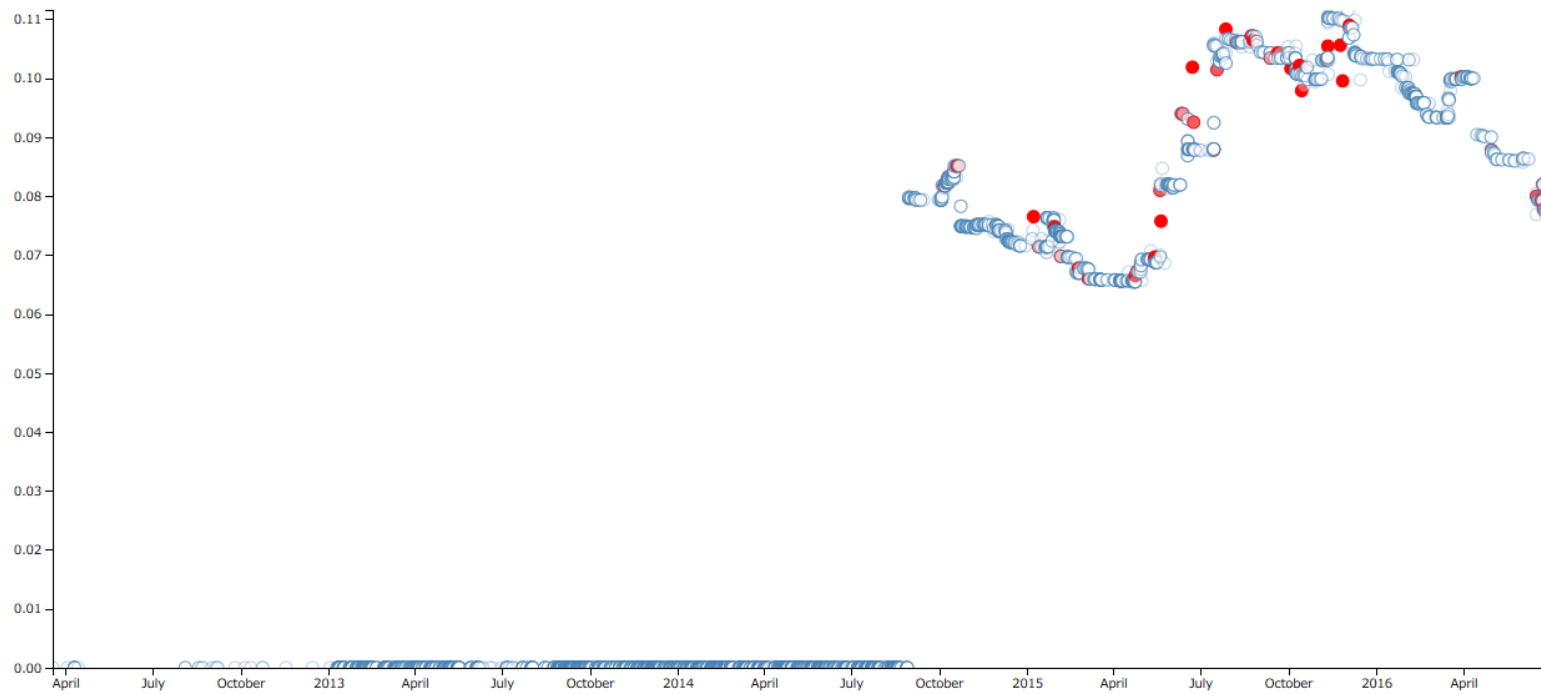
可視化

コードクローンの変更内容

☐ created ☐ changed ☐ co-changed ☐ delete

コミット数

入力 リセット



時系列表示

- x軸: 時間
- y軸: クローン密度
- 各ノード: リビジョン
 - 赤: コードクローンと不具合を紐づけしたリビジョン

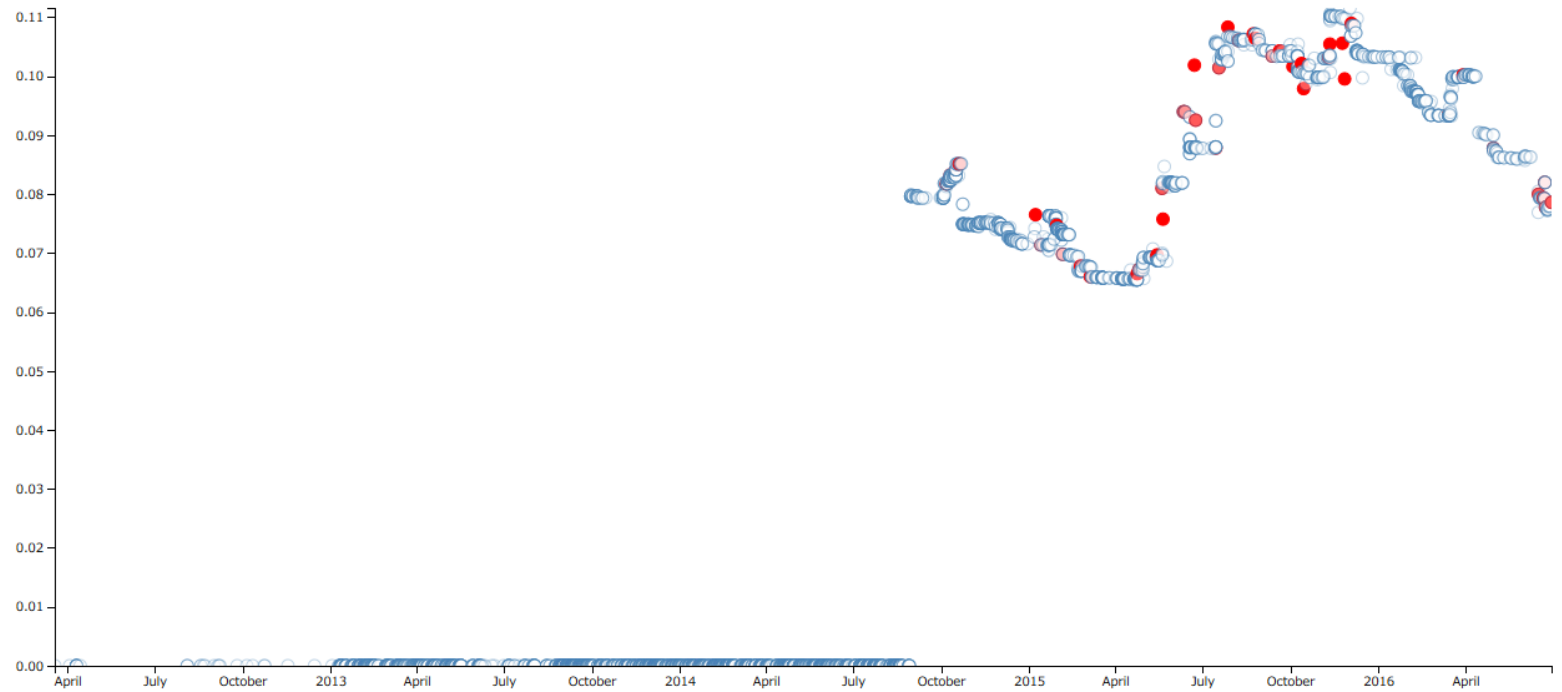
可視化

コードクローンの変更内容

☐ created ☐ changed ☐ co-changed ☐ delete

コミット数

入力 リセット

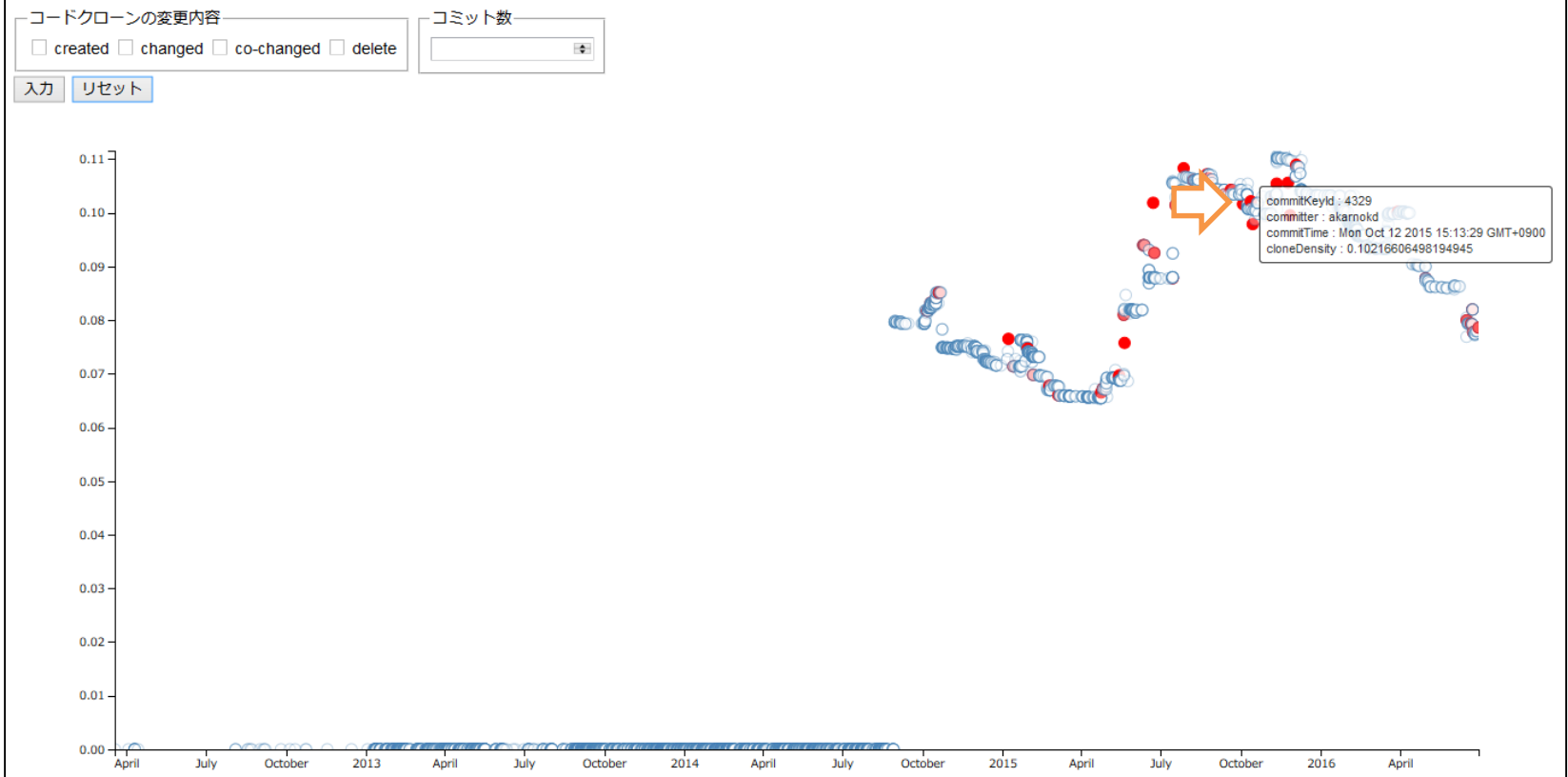


時系列表示

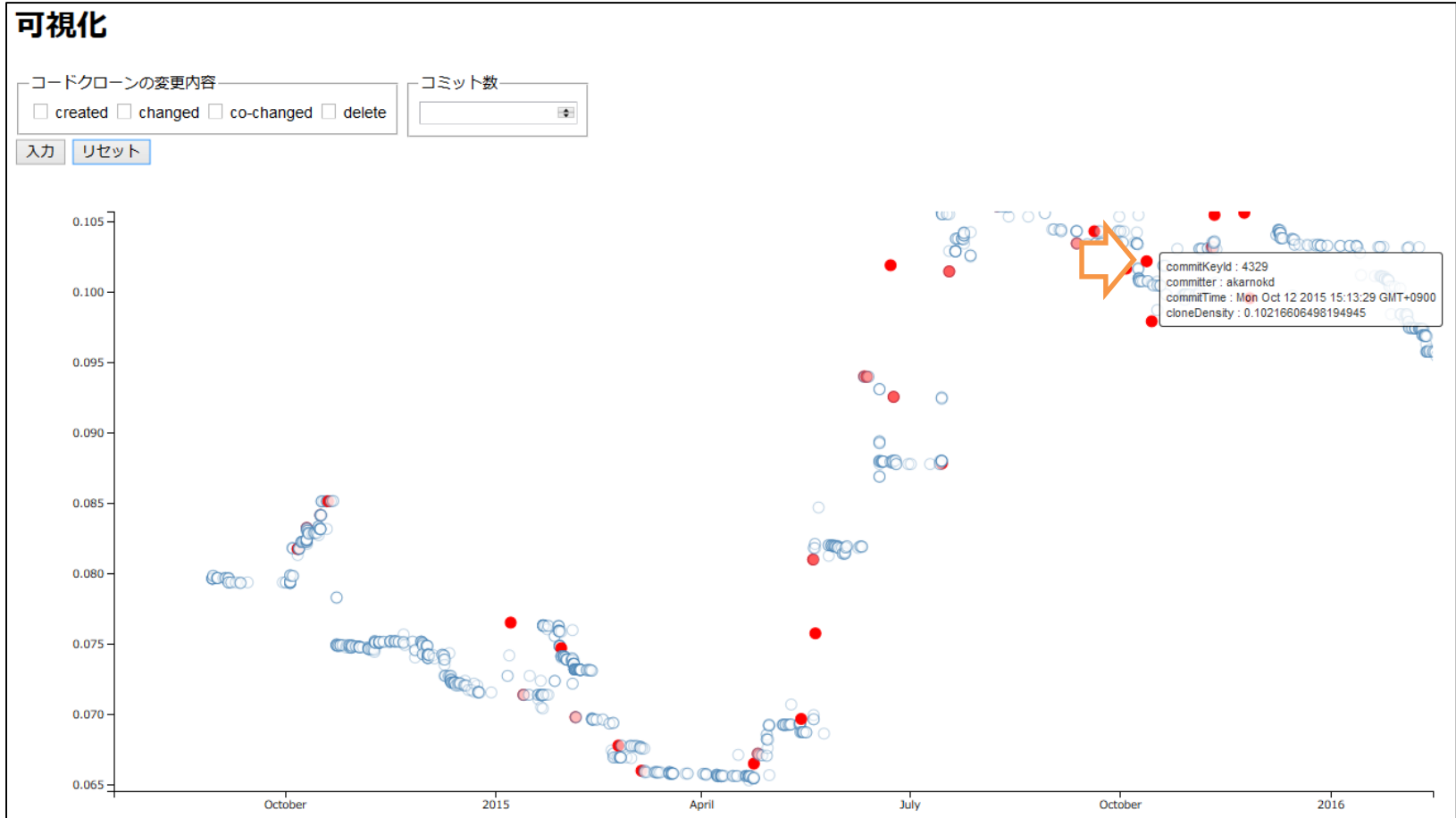
- 機能
 1. 拡大・縮小
 2. コードクローン変更リビジョンの確認
 3. 各ノードの情報確認
 4. コードクローンの変更内容によるフィルタリング
 5. 開発者のコミット数によるフィルタリング

1. 拡大・縮小

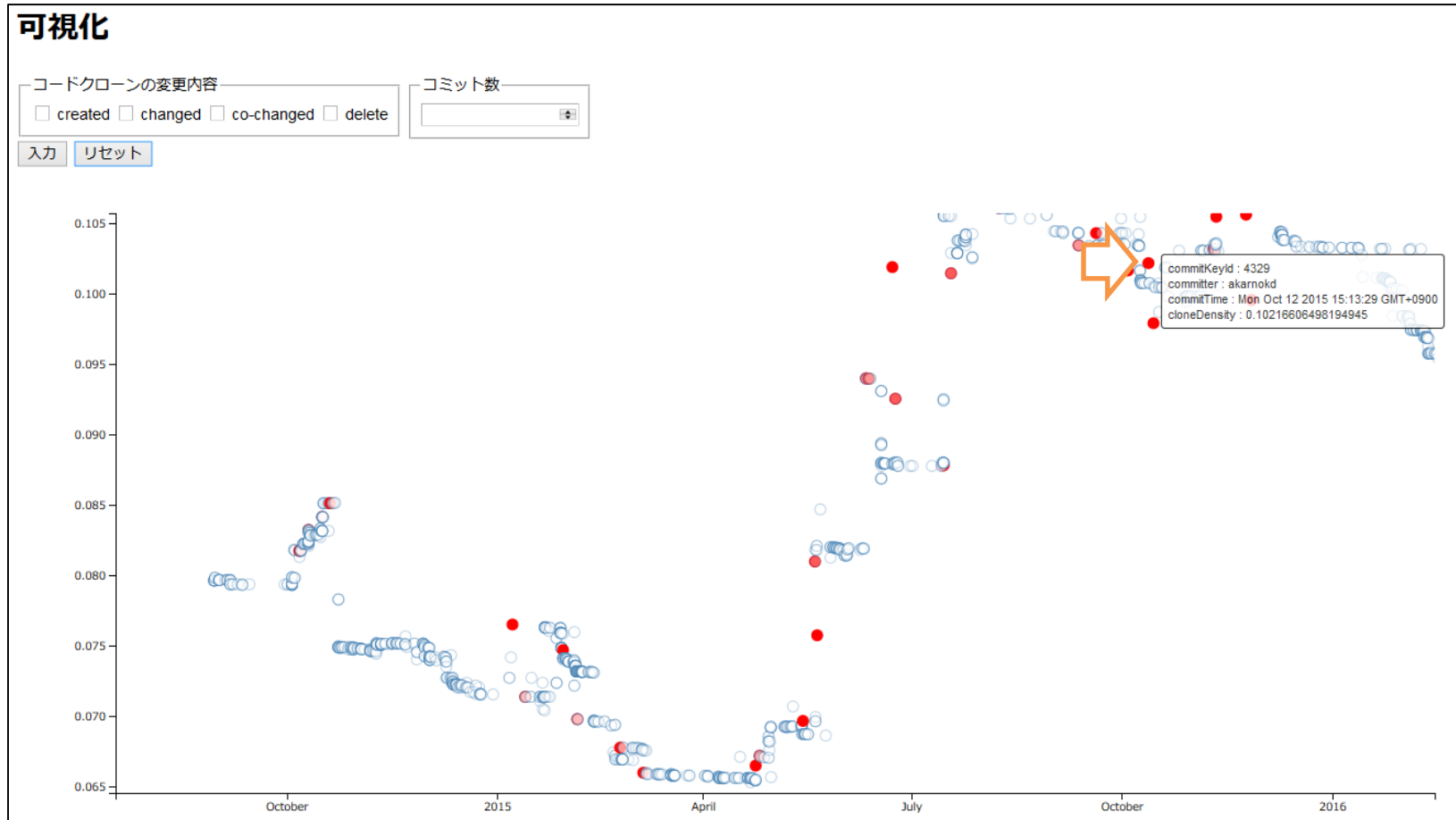
可視化



1. 拡大・縮小



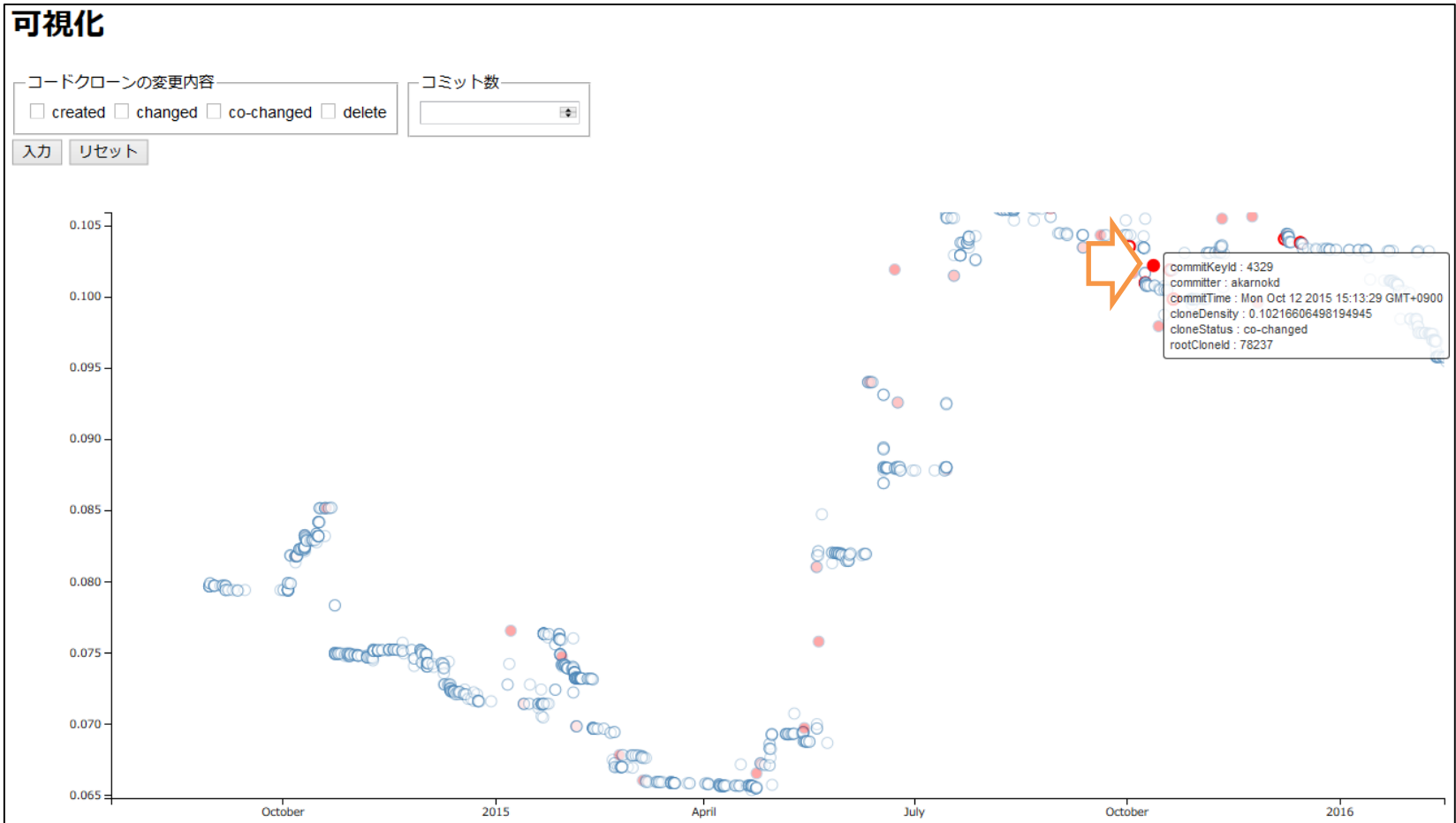
2. コードクローン変更リビジョンの確認



2. コードクローン変更リビジョンの確認



3. 各ノードの情報確認



4. コードクローンの変更内容による フィルタリング

可視化

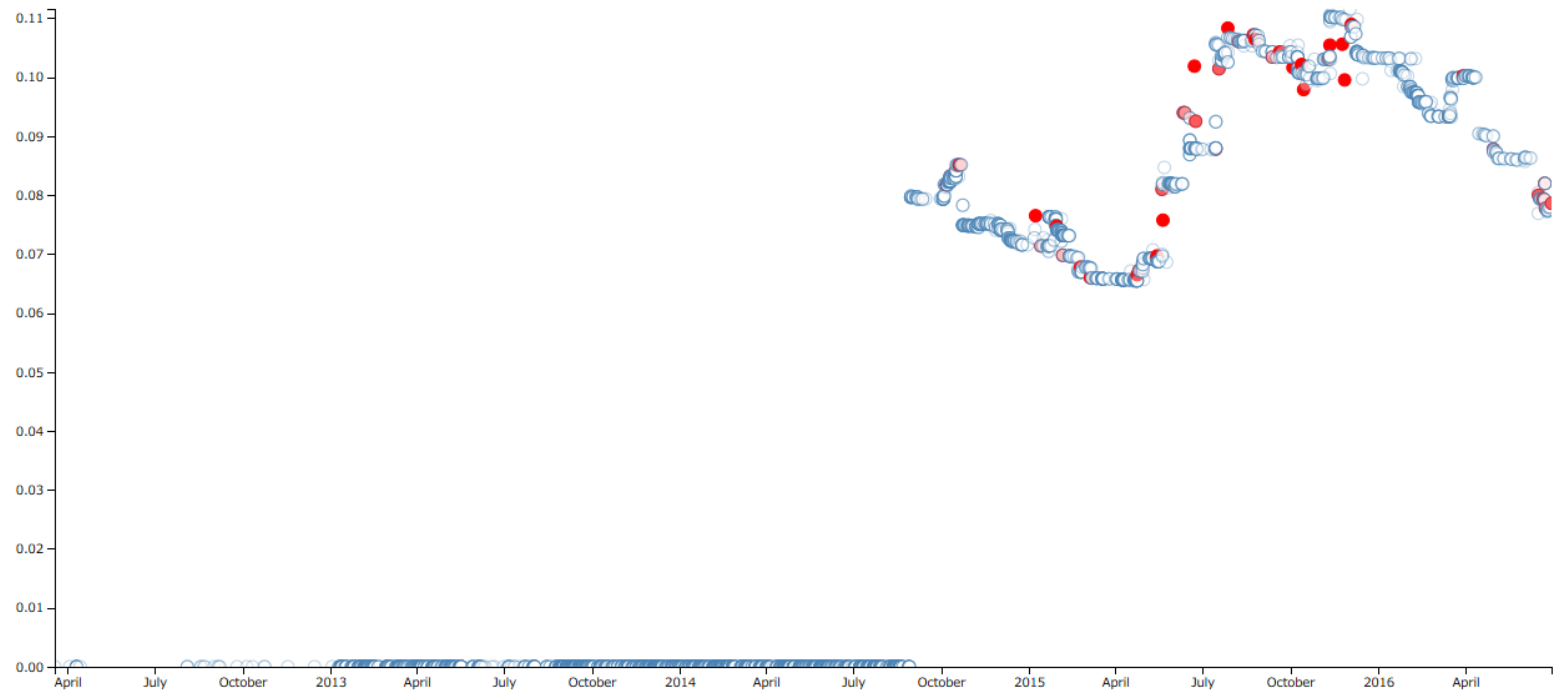
コードクローンの変更内容

☐ created ☐ changed ☐ co-changed ☐ delete

入力

リセット

コミット数



4. コードクローンの変更内容による フィルタリング

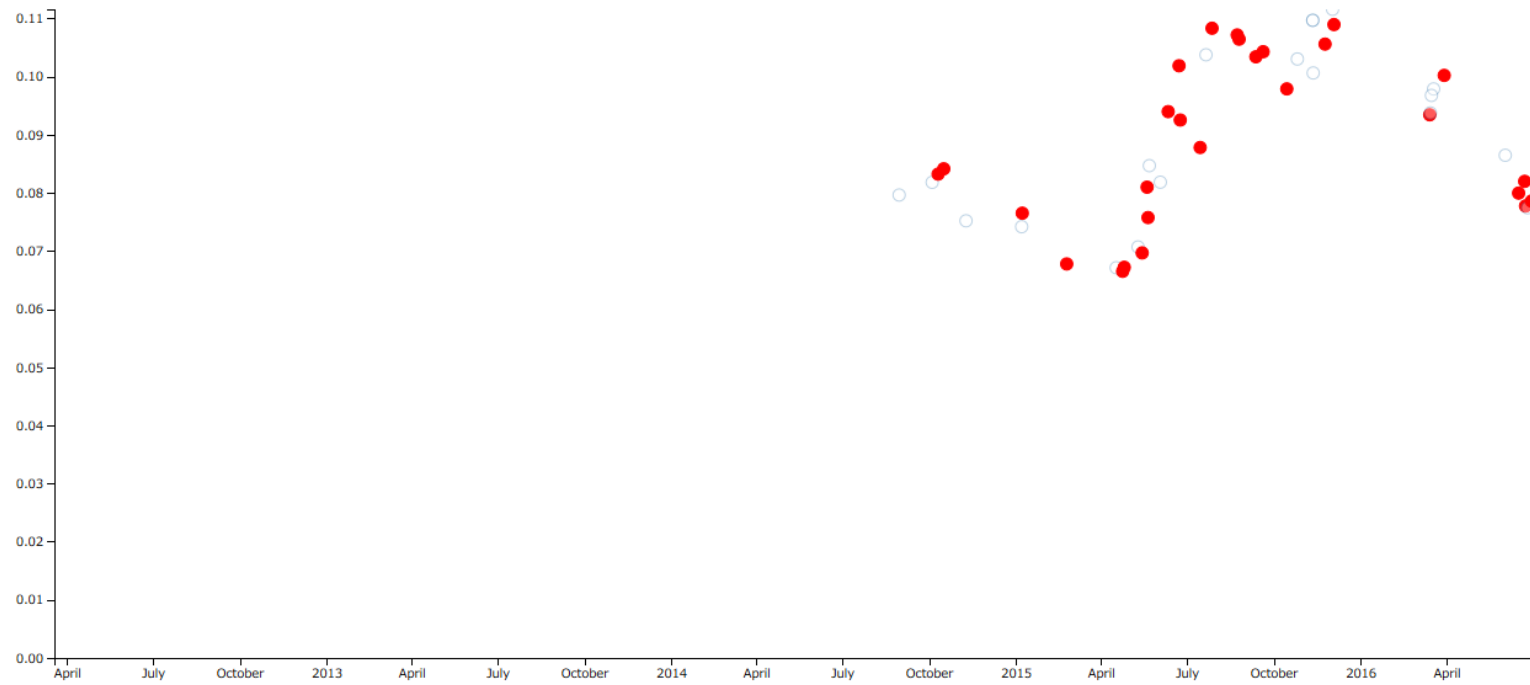
可視化

コードクローンの変更内容

☒ created ☐ changed ☐ co-changed ☐ delete

コミット数

入力 リセット



4. コードクローンの変更内容による フィルタリング

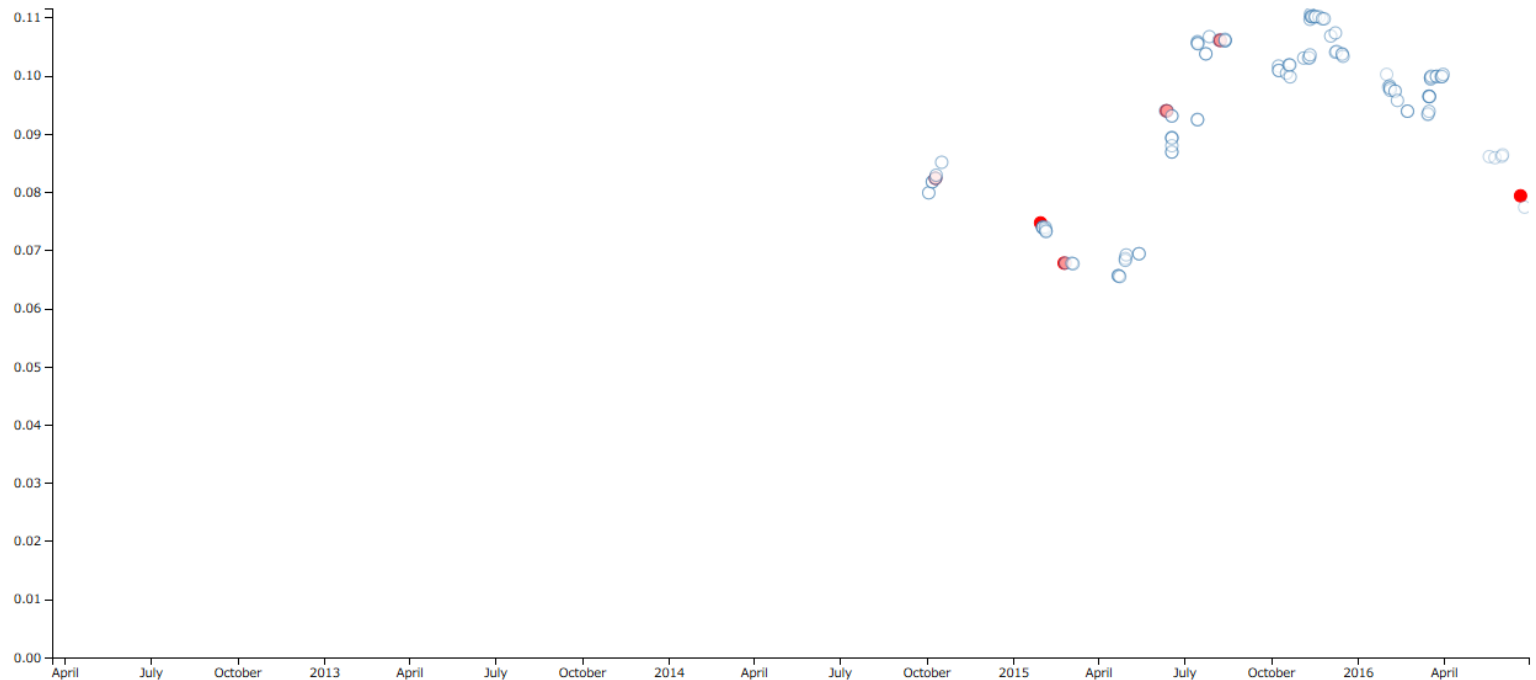
可視化

コードクローンの変更内容

☐ created ☒ changed ☐ co-changed ☐ delete

コミット数

入力 リセット



4. コードクローンの変更内容による フィルタリング

可視化

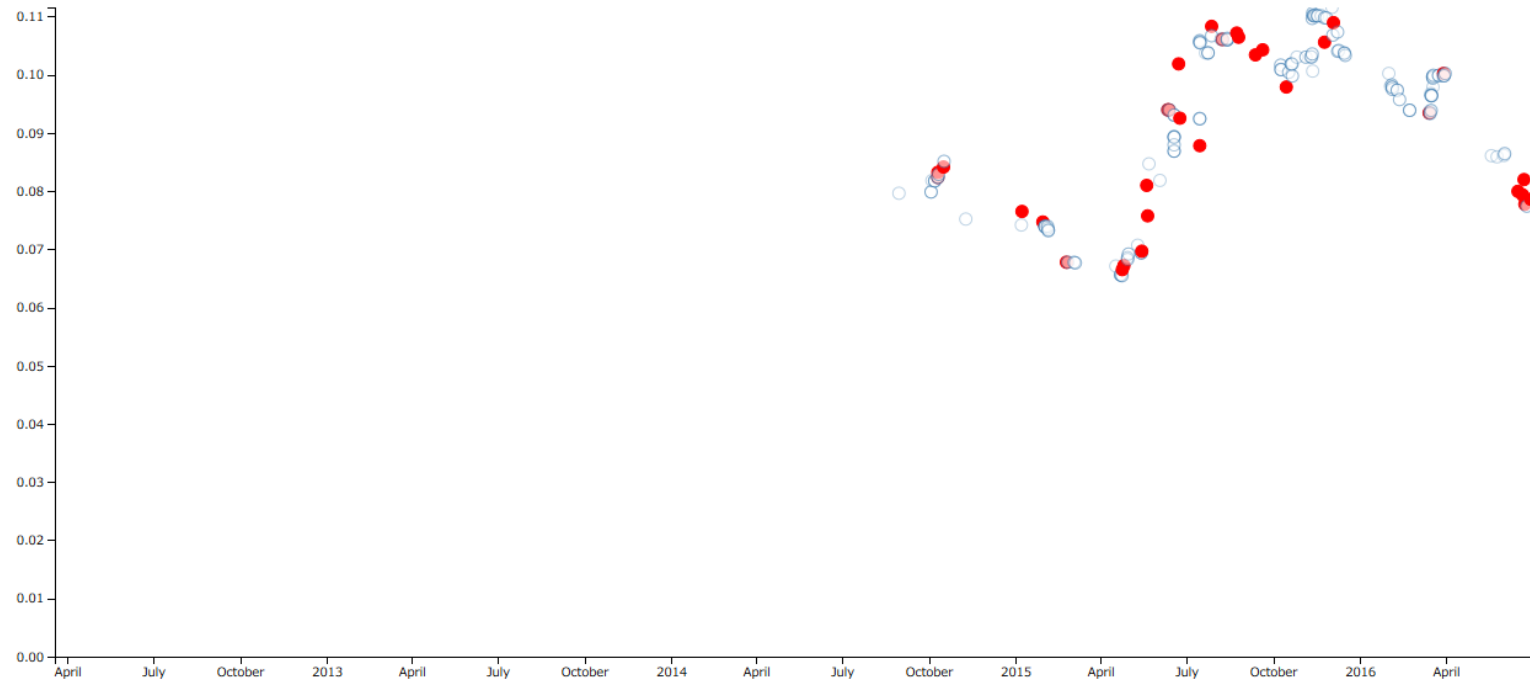
コードクローンの変更内容

☒ created ☒ changed ☐ co-changed ☐ delete

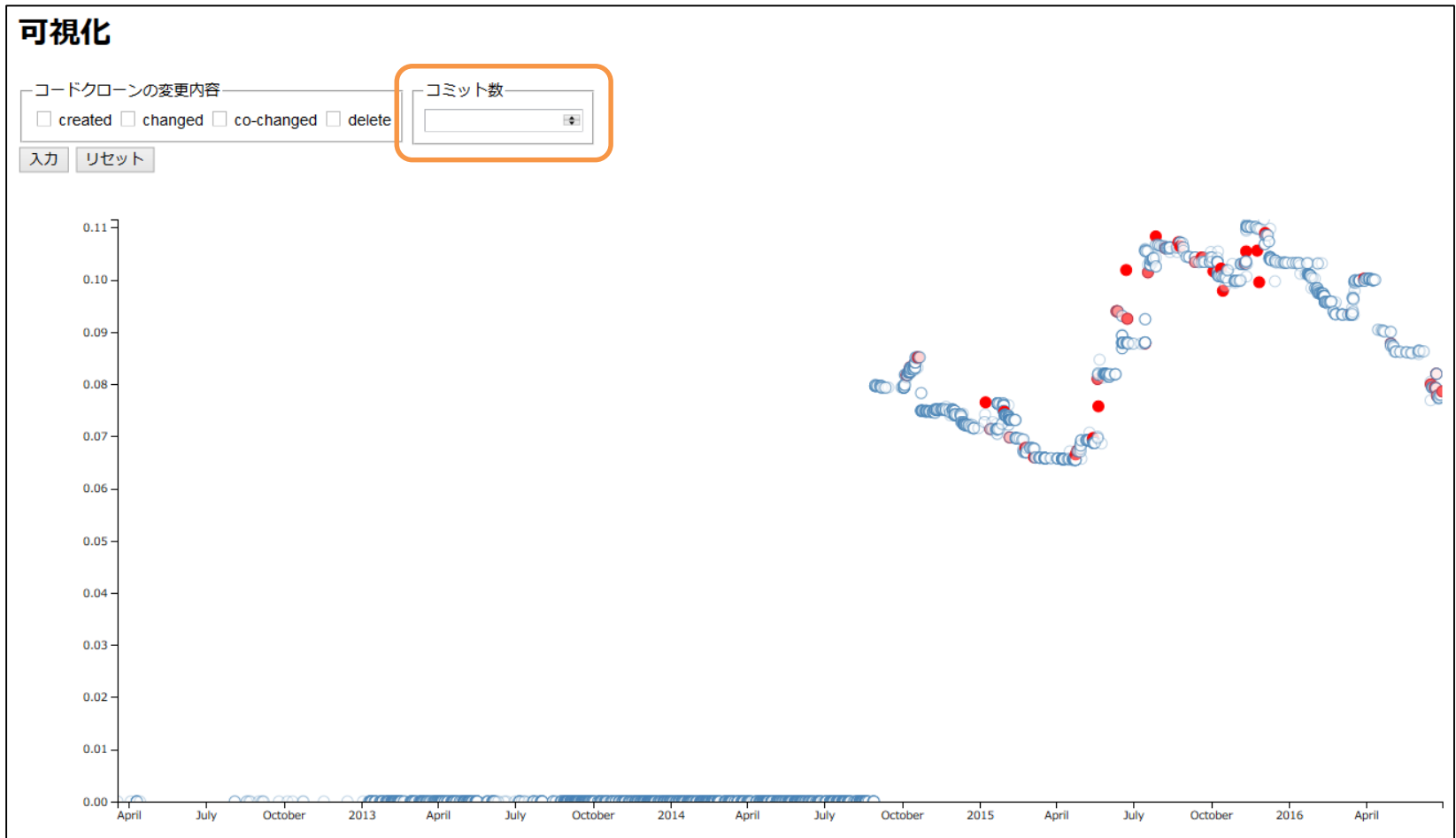
入力

リセット

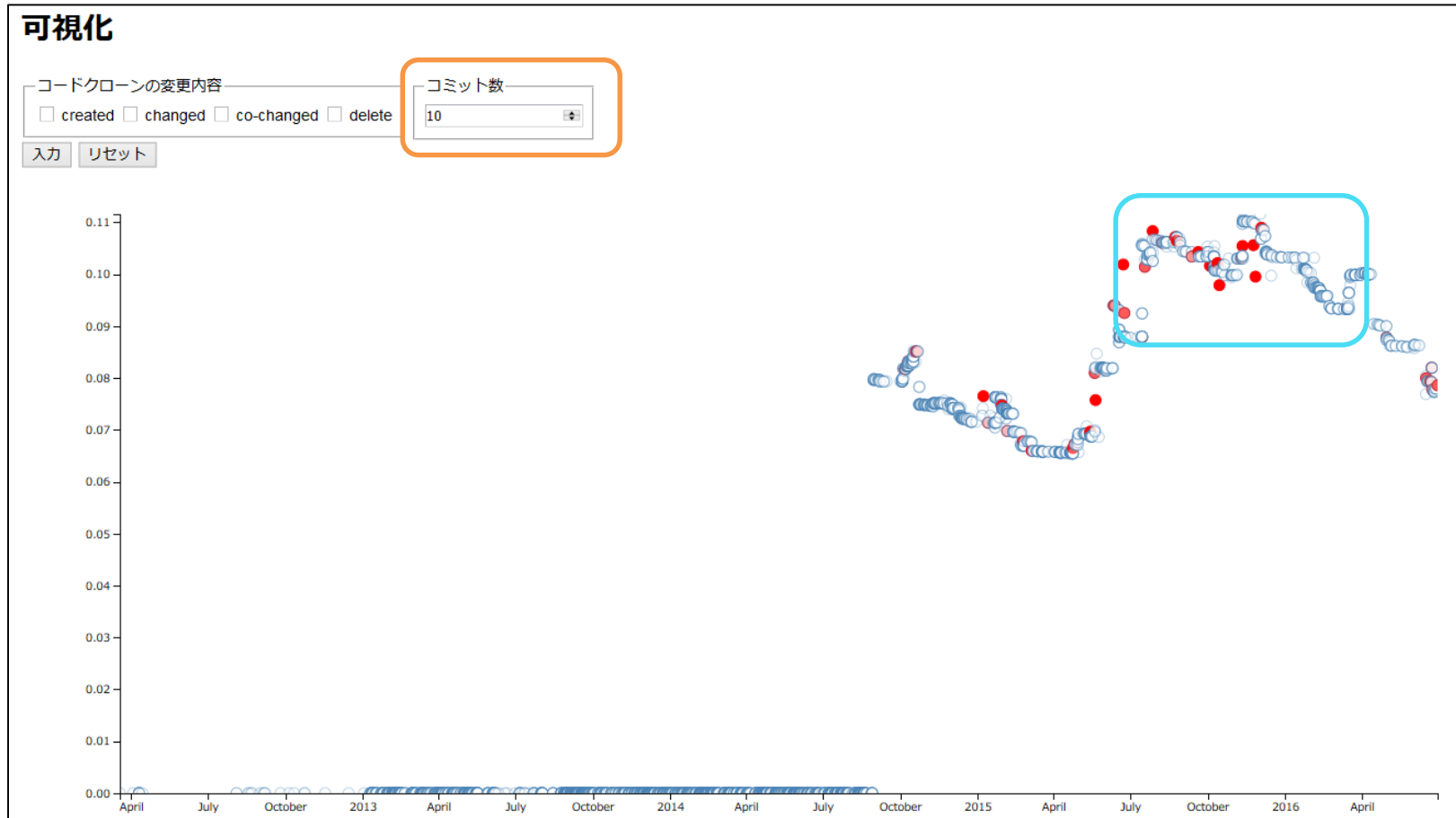
コミット数



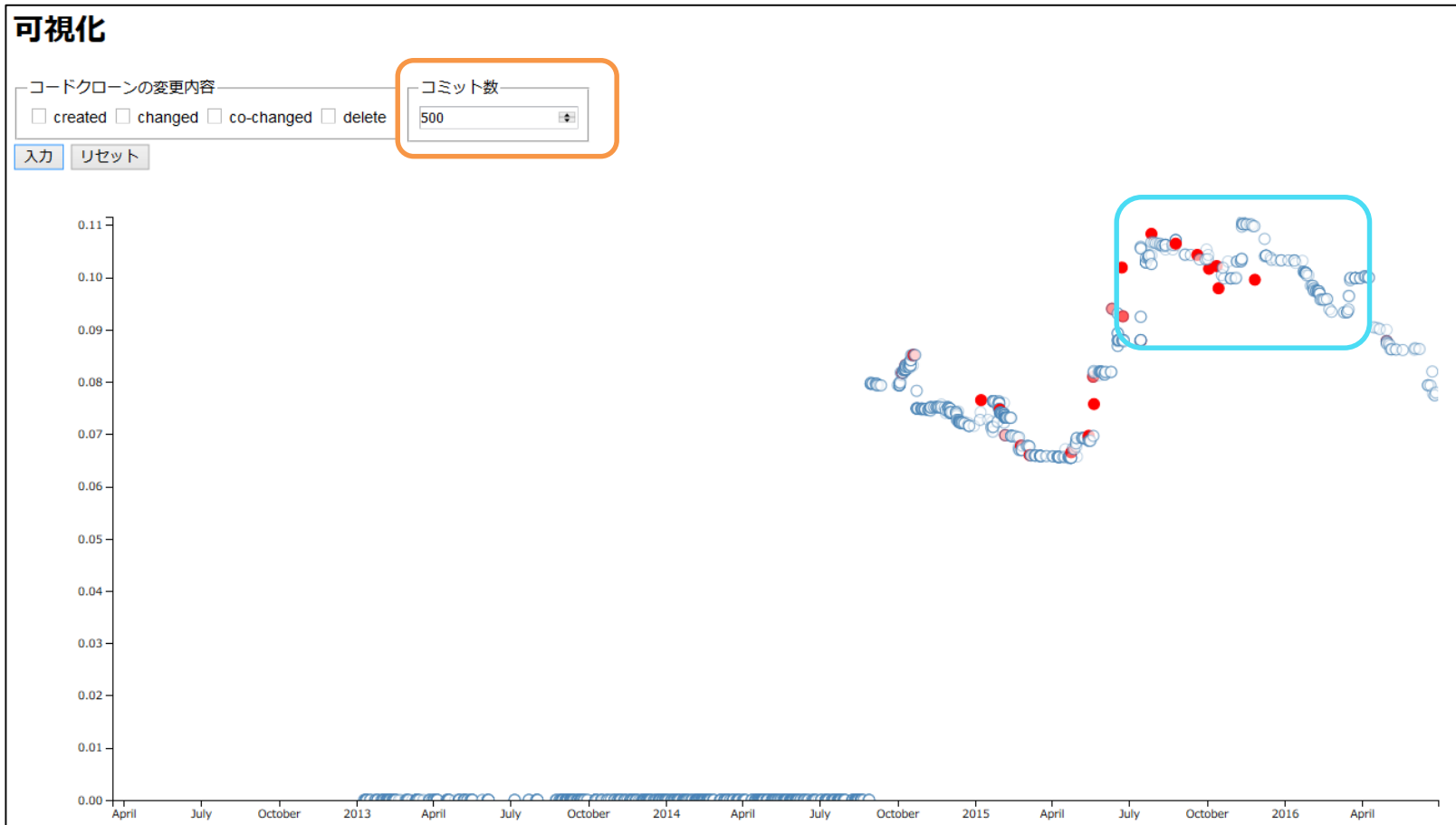
5. 開発者のコミット数によるフィルタリング



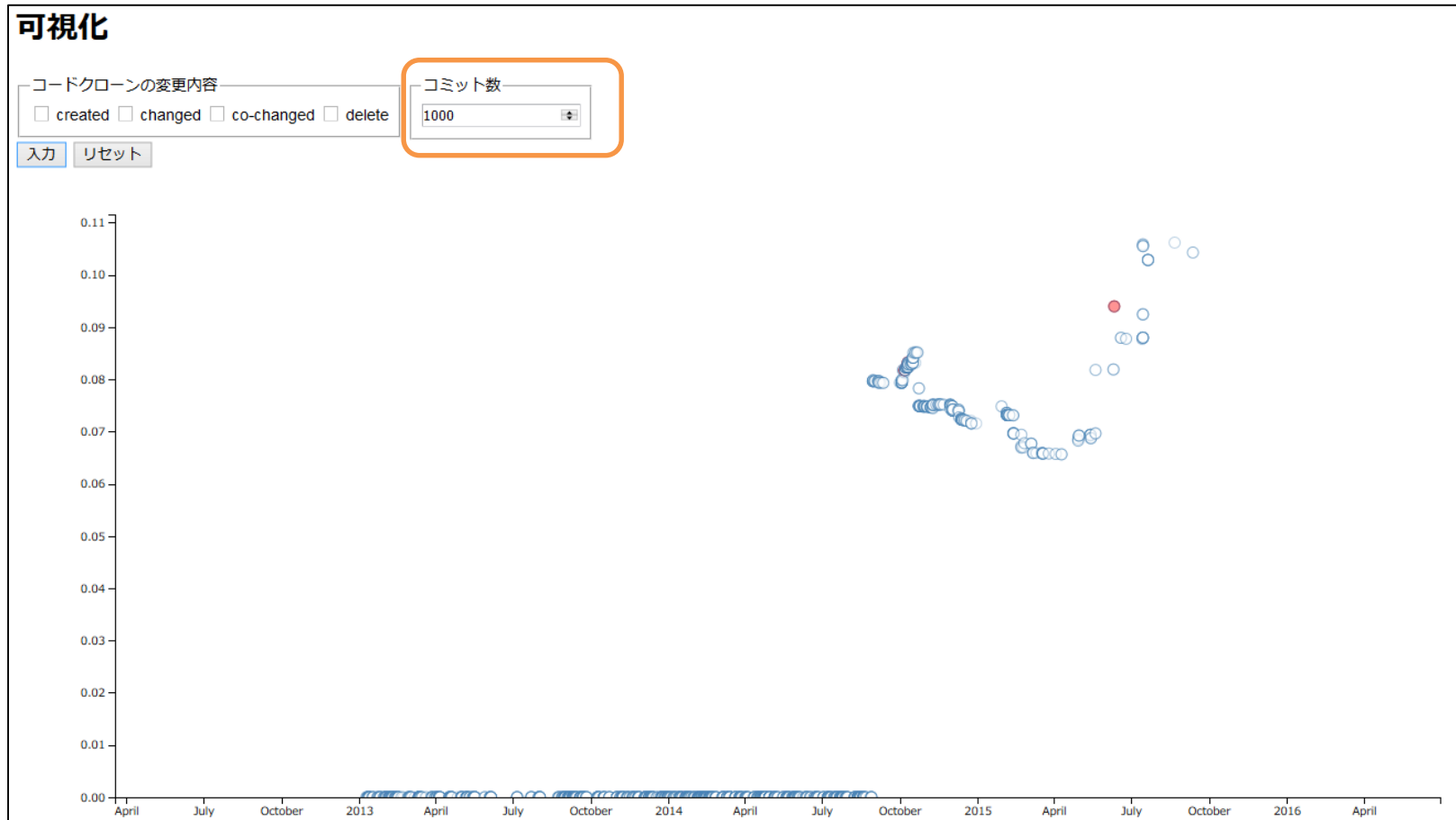
5. 開発者のコミット数によるフィルタリング



5. 開発者のコミット数によるフィルタリング



5. 開発者のコミット数によるフィルタリング



評価実験

- コードクローンの利用過程で, コードクローンから不具合を発生させる開発者の特徴を分析できることを示す
 - CCTをそのまま用いた場合と, 構築したツールを用いた場合とで, 複数人にRQを比較してもらう

Research Question

- RQ1
 - ソフトウェアに対する, コードクローンの利用過程を俯瞰的に認識できるか
- RQ2
 - コードクローンの利用過程から特定の開発者を識別できるか
- RQ3
 - コードクローンから不具合を発生させる開発者にはどんな特徴がみられるか

RQ1の評価方法

- RQ1
 - ソフトウェアに対する, コードクローンの利用過程を俯瞰的に認識できるか
- コードクローンの利用過程を俯瞰的に認識できたかをYes/Noで答えてもらう
 - クローン密度の増減から判断

RQ2の評価方法

- RQ2
 - コードクローンの利用過程から特定の開発者を識別できるか
- コードクローンの利用過程から特定の開発者を識別できたかをYes/Noで答えてもらう
 - コードクローンを利用する開発者から特定の人物を見つけられるかで判断

今後の課題

- RQ3の評価方法
 - なにができたなら, 特徴を発見できるようになったと言えるか
 - 何が特徴と言えるか
- 1つのリビジョンに, 不具合と紐づけしたコードクローンが複数存在する場合を考慮する