社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# 高い拡張性を備えた SaaS型コードクローン分析システムの提案

松島　一樹†　　井上　克郎†

† 大阪大学大学院情報科学研究科　〒560—0871 大阪府吹田市山田丘 1—5
E-mail: †{k-matusm,inoue}@ist.osaka-u.ac.jp

**あらまし**　効率的なコードクローンの分析の支援を目的としたツールに関する研究が数多く行われている．しかし，既存のツールは特定のコードクローン検出ツールにのみ対応していたり，外部サービスとの連携が困難であったりするなど，いくつかの問題を抱えている．これらの問題を解決するため，本研究では SaaS 型コードクローン分析システムを提案する．これはユーザに WEB クライアントを提供し，サーバ上でコードクローンの検出を行うものである．バックエンドには既存の検出ツールを使用するが，検出ツール側にシステムとデータの伝達を行う数個の処理を実装するだけで任意のツールを追加することができる．また，コードクローン分析や検出結果に関する REST API を公開しており，外部サービスと容易に連携可能である．OSS に対し提案システムを用いてコードクローンを検出・分析し，提案システムがより正確な分析を可能にすることを確認した．
**キーワード**　コードクローン，SaaS, ソフトウェア保守

# A Proposal for a SaaS-based Code Clone Analysis Environment with an Extensible Architecture

Kazuki MATSUSHIMA† and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University　1–5, Yamadaoka, Suita, Osaka, 560–0871, Japan
E-mail: †{k-matusm,inoue}@ist.osaka-u.ac.jp

**Abstract**　Many studies on tools for efficient code clone analysis have been conducted in the past. However, these tools have some issues such that they support only a few kinds of code clone detectors or they are less compatible with external services. To address them, we propose a SaaS-based code clone analysis environment in this paper. This offers a WEB client and detects code clones in a project on the server side. Existing code clone detectors are used as the back-end of this system and any code clone detectors can be used if just a few functions for communicating to the system are implemented. Furthermore, it also exposes REST APIs for code clone analysis and detection results so that external services can cooperate with it easily. By identifying and analyzing code clones of an OSS using our proposed system, we confirmed that it enables more accurate code clone analysis.
**Key words**　Code Clone, SaaS, Software Maintenance

## 1. ま え が き

ソースコード中に存在する，全く同一もしくは互いに類似したコード片のことをコードクローンと呼ぶ．コードクローンは，主に既存のソースコードのコピー・アンド・ペーストによる再利用やコード生成ツールによる自動生成によって発生する [8]．既存研究において，コードクローンはバグの発生や伝播と関連していることが明らかになっており，ソフトウェア開発の保守工程における大きな問題の一つとして指摘されている [1] [2]．

そこで，効率的なコードクローンの分析の支援を目的としたツールが数多く提案されている．その例として Gemini [9] や VisCad [12] が挙げられるが，いずれもスタンドアローンのアプリケーションであり，利用するためにはローカル環境へのインストールが必要，特定の検出ツールにのみ対応，外部サービスとの連携が困難といった問題がある．

Bandi らによって提案された Clone Swarm [6] はクラウド

型コードクローン分析ツールであり，ローカル環境へのインストールが不要で，REST API が公開されているため外部サービスとの連携が容易といった特徴を持つ．一方で，コードクローンの検出には NiCad [11] のみが使用可能であり，その他の検出ツールには対応していない．

そこで本研究では，多様なコードクローン検出ツールに対応可能な SaaS 型コードクローン分析システムを提案する．

既存の分析ツールでは特定の検出ツールに依存してユーザインターフェイスが実装されていたが，提案システムでは外部から与えられた検出パラメータ定義から検出パラメータを設定するフォームを動的に生成する．

また，提案システムでは既存の複数のコードクローン検出ツールをバックエンドとして使用するが，検出ツールの起動方法および出力結果のフォーマットについて共通の仕様を定めた．これらの仕様を満たした検出ツールはシステムに一切変更を加えることなくバックエンドに新たに追加することができる．

更に，提案システムではコードクローン分析や検出結果に関する REST API を公開する．REST API は HTTP を利用したステートレスな WEB API の設計原則であり，多くの WEB サービスで活用されている一般的なものである．

提案システムは **sel.ist.osaka-u.ac.jp/webapps/ccx/** で公開している．

以降，2. では，本研究の背景としてコードクローン，SaaS，REST API，そして関連研究について述べる．3. では提案システムとその詳細について述べる．4. では提案システムの適用事例について述べる．最後に 5. では本研究のまとめについて述べる．

## 2. 背　　景

### 2.1 コードクローン

コードクローンとは，ソースコード中に存在する「全く同一」もしくは「互いに類似した」コード片を指す．コードクローンは，主に既存のソースコードのコピー・アンド・ペーストによって発生する．一般的に互いにコードクローンとなる 2 つのコード片の組をクローンペアと呼び，コードクローンの同値類をクローンセットと呼ぶ．

コードクローンは，既存研究においてソフトウェア保守を困難にする要因の一つとして指摘されている．Barbour らは，複数のコードクローン検出ツールを使用して，システムの開発履歴におけるコードクローン変化のパターンを分析した [1]．そして，他の変化パターンと比較して，クローンペアでなくなったコード片同士が一方のコード片に変更が加えられたことで再びクローンペアとなるパターンと，両方のコード片に変更が加えられていたクローンペアがさらなる変更によりクローンペアではなくなるパターンが，バグが混入する可能性が高いことを明らかにした．

また，Mondal らは，7 個のプロジェクトを対象にコードクローンの変更履歴を調査し，過去にバグ修正が行われたコードクローンのうち 18.42% がバグ伝播に関与していることを明らかにした [2]．

表 1: 代表的な HTTP メソッドとその機能

| HTTP メソッド | 機能 |
|---|---|
| GET | リソースの取得 |
| POST | リソースの新規作成 |
| PATCH | リソースの部分更新 |
| DELETE | リソースの削除 |

開発者がコードクローンの管理や修正を行うことで，これらコードクローンによる影響を抑制しソフトウェアの保守性を維持することができる．そのため，開発者がコードクローンに関する情報を認識することは重要である．

### 2.2 Software as a Service (SaaS)

Software as a Service (以下，SaaS) とは，クラウド上にホストされ，インターネットを経由して利用可能なソフトウェアおよびその形態を意味し，近年のインターネットの拡大とインフラの整備に伴って急速に普及している [3]．

従来，ソフトウェアを利用するためには，ローカル環境へインストールし，セットアップを行う必要があった．一方，SaaS はインストール作業が不要であり，ブラウザからサービスサイトへアクセスするだけで利用可能であるため，従来のソフトウェアと比較して利便性において優れている．

### 2.3 REST API

Fielding は論文 [4] において，HTTP を用いた API の 設計原則として REST (REpresentational State Transfer) 原則を提案した．REST 原則では，クライアント - サーバ型アーキテクチャであること，ステートレスであること，リソースが URI にマッピングされていることなどが定められており，これらを満たした API を REST API と呼ぶ．

REST API では HTTP メソッドに基づいてリソースに対する操作を行う．表 1 に代表的な HTTP メソッドを示す．

例えば，ユーザ一覧を表すリソース /users に対して GET メソッドでリクエストを送信すると，ユーザ一覧が返却される．同様に，POST メソッドでリクエストを送信すると，新規ユーザが作成される．POST メソッドでは，リソースの作成に必要な情報をリクエストに含めることが一般的である．

REST API は Twitter や YouTube，Amazon をはじめとする多くの WEB サービスで利用されている．

### 2.4 関 連 研 究

コードクローンを管理・修正することは，ソフトウェアの保守性を維持するために重要である．しかし，全てのコードクローンを手作業で検出することは非現実的であるため，効率的なコードクローンの検出・分析を目的としたツールに関する研究が数多く行われている．

コードクローンの検出に関して CCFinder [5]，SourcererCC [10]，NiCad [11] などのツールが提案されている．これらのツールはコマンドラインで動作するが，検出結果をファイル名や行番号などの組み合わせとして出力するため，検出結果を直感的に理解することは困難である．

そこで，これらのツールの検出結果を可視化して，効率的

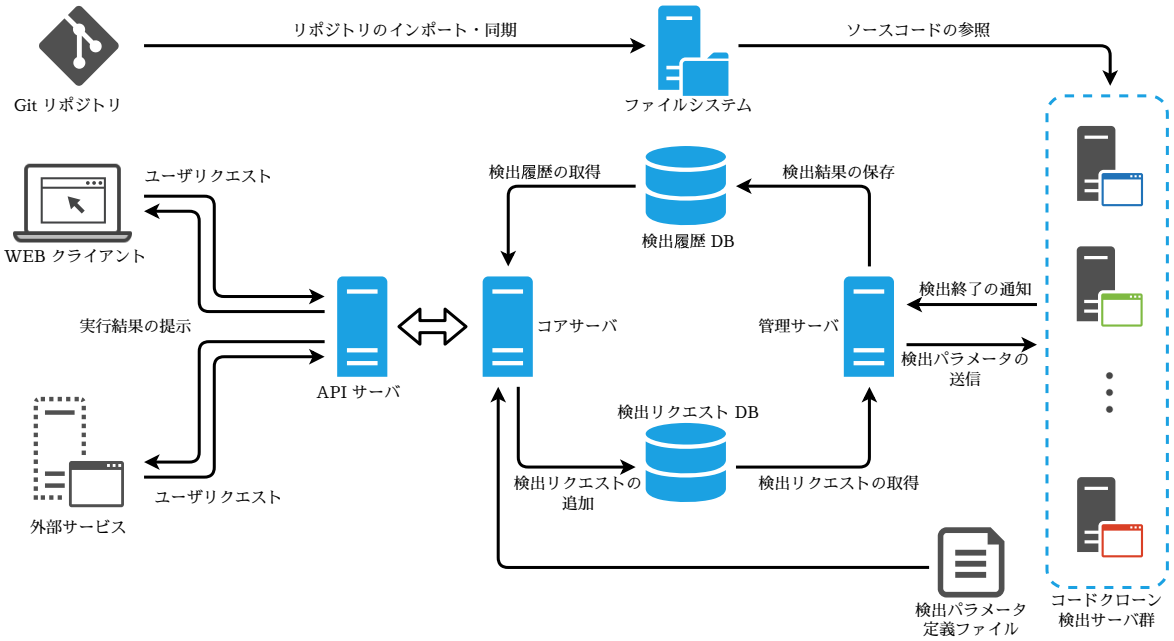



図 1: 提案システムのアーキテクチャ

なコードクローンの分析を支援するツールが提案されている．植田らは CCFinder の出力結果を散布図やメトリクスグラフとして可視化するツール Gemini を提案している [9]．また，Asaduzzaman らは NiCad の出力結果を散布図やラジアルマップ，ツリーマップとして可視化するツール VisCad を提案している [12]．

これらの分析ツールを利用するためには，ツール自体に加え，検出ツールをローカル環境にインストールし，セットアップを行う必要がある．また，Gemini は CCFinder に，VisCad は NiCad にのみ対応しており，その他の検出ツールの結果を分析することは困難である．

Bandi らはクラウド型コードクローン分析ツール Clone Swarm を提案した [6]．Clone Swarm は WEB クライアントから操作することができ，サーバ上でコードクローン検出を行う．そのため，ローカル環境にツールをインストールする必要がなく既存のツールと比較して容易に利用することが可能である．しかし，Clone Swarm は NiCad にのみ対応しており，その他のコードクローン検出ツールを使用することはできない．

## 3. 提案システム

本研究では，SaaS 型コードクローン分析システムを提案する．以降，提案システムのアーキテクチャおよび各機能の詳細について述べる．

### 3.1 アーキテクチャ

図 1 に提案システムのアーキテクチャを示す．提案システムは数種類のサーバ，データベースおよび WEB クライアントから構成される．

API サーバは WEB クライアントや外部サービスと直接通信を行うサーバであり，ユーザ認証機能を兼ね備える．また，WEB クライアントや外部サーバからのリクエストを解析し，必要な処理をコアサーバに依頼する．

コアサーバは，API サーバからのリクエストに応じて，プロジェクトの管理や検出結果の解析などを行う．また，外部の Git リポジトリからソースコードを取得し，ファイルシステムに保存する機能を持つ．コードクローン検出を実行するリクエストに対しては，パラメータ定義ファイルからパラメータ定義を読み取り，パラメータ値のバリデーションを行い，正当なリクエストであれば，検出リクエストを検出リクエスト DB に追加する．

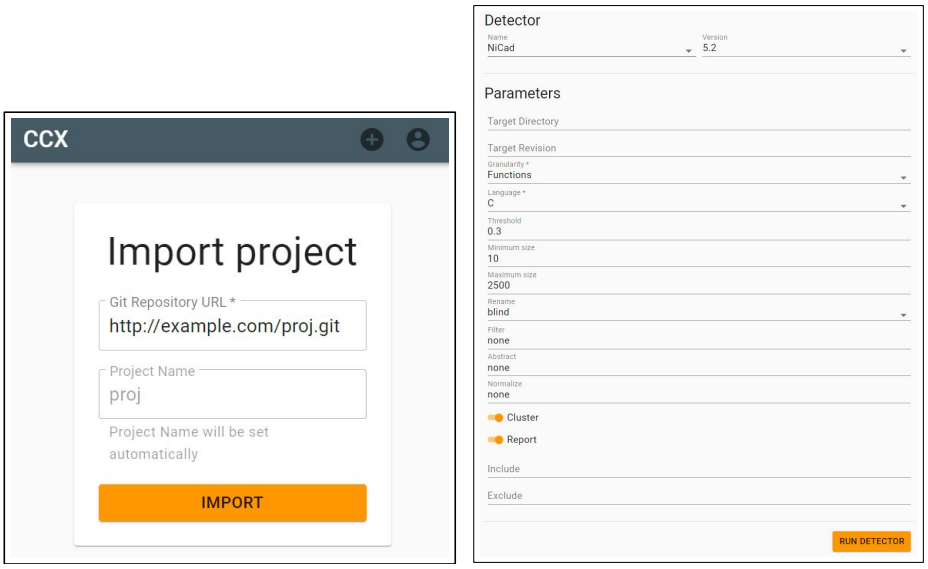
管理サーバはコードクローン検出に関する処理を管理するサーバである．検出リクエスト DB からリクエストを取得し，負荷状況を考慮しつつ，コードクローン検出サーバにコードクローン検出の実行を指示する．また，検出完了の通知を受信すると検出履歴 DB に検出結果を保存する．

コードクローン検出サーバは，管理サーバからコードクローン検出パラメータを受信すると，内部で検出ツールを起動してコードクローンの検出を行うサーバである．コードクローン検出が完了すると，検出結果を管理サーバに送信する．管理サーバ - コードクローン検出サーバ間のデータ伝達には共通の仕様が定められており，この仕様を満たす任意のコードクローン検出サーバをシステムのバックエンドに追加することが可能である．

WEB クライアントは，ブラウザから操作可能な提案システムのフロントエンドである．プロジェクトのインポートやコードクローン検出の実行など，提案システムが提供する様々な機能を利用可能である．また，コアサーバの読み取ったパラメータ定義から動的に検出パラメータを設定するフォームを生成する機能を持つ．

### 3.2 WEB クライアント

提案システムはブラウザからのアクセスに対し WEB クライ




(a) プロジェクトのインポート画面　　(b) 検出パラメータ設定画面

図 2: WEB クライアントの画面例

アントを提供する．WEB クライアントの実際の画面例を図 2 に示す．

図 2a では外部の Git リポジトリをインポートする．ユーザがフォームに値を入力するとバリデーションが行われ，入力が正しい Git リポジトリ URL であれば IMPORT ボタンが有効化される．正しい Git リポジトリ URL でなければ，フォームにエラーメッセージが表示され，正しい URL の入力を促す．

図 2b ではインポートしたリポジトリに対し，検出パラメータを設定し，コードクローン検出を実行することができる．Detector セクションから使用する検出ツールとそのバージョンを，Parameters セクションで検出パラメータを設定する．ユーザが選択する検出ツールに応じて Parameters セクションに表示されるフォームは変化する．

従来のツールでは，このような検出パラメータ設定フォームのユーザインターフェイス はプログラムにハードコーディングされていたため，新たな検出ツールに対応させるためにはソースコードの修正が必要であった．提案システムでは，コアサーバに与えられた検出パラメータ定義をもとに動的に検出パラメータ設定フォームを生成する．

### 3.2.1 検出パラメータ定義ファイル

検出パラメータ定義ファイルは YAML 形式のテキストファイルであり，検出ツールごとにパラメータ定義を記述する．図 2b の検出パラメータ設定フォームを生成する検出パラメータ定義の一部を図 3 に示す．nicad は検出ツールの ID，name は WEB クライアントに表示される検出ツールの名前である．また，parameters には検出パラメータ定義を配列で記述する．検出パラメータ定義に記述するフィールドを表 2 に示す．検出パラメータ定義の type には directory, revision, input, int, float, select, switch の 7 種類が存在する．

WEB クライアントでは，検出パラメータ定義ファイルに記述された内容から検出パラメータ設定フォームを動的に生成し，

```
nicad:
  name: NiCad
  parameters:
    - key: directory
      label: Target Directory
      type: directory
      revisionKey: revision
    - key: revision
      label: Target Revision
      type: revision
    - key: granularity
      label: Granularity
      type: select
      rule:
        default: functions
        values: [Functions, Blocks]
    - key: language
      label: Language
      type: select
      rule:
        default: C
        values: [C,Java,C#,Python,PHP,Ruby,WSDL,ATL]
    - key: threshold
      label: Threshold
      type: float
      rule:
        default: 0.3
        min: 0
        max: 1
    - key: minsize
      label: Minimum size
      type: int
      rule:
```

図 3: NiCad の検出パラメータ定義の例

表 2: 検出パラメータ定義のフィールド

| フィールド名 | 属性 | 機能 |
|---|---|---|
| key | 必須 | 検出パラメータのフィールド名 |
| label | 任意 | WEB クライアントで表示されるパラメータ名 |
| required | 任意 | 検出パラメータが必須であるかどうか |
| type | 必須 | 検出パラメータの種類 |
| rule | 必須 | 検出パラメータの制約 |

各パラメータの値が rule に記述された制約を満たしているかリアルタイムでバリデーションを行う．制約を満たしていなければ，対応するエラーメッセージをフォームに表示する．

### 3.2.2 検出結果の分析

WEB クライアントでは，ユーザが選択した 2 つの検出履歴から検出結果を比較することができる．検出結果の比較には論文 [7] で提案された手法を用いる．

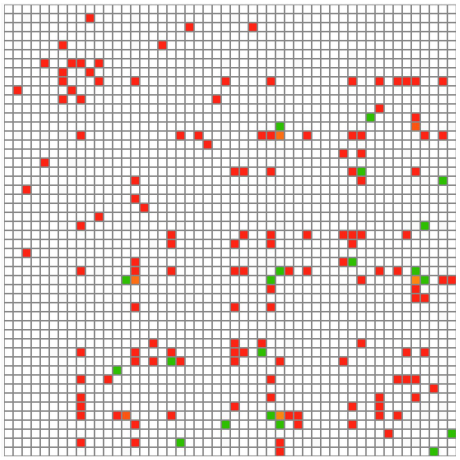


図 4: 表示される散布図の例

```
249
250     @Override
251     public JwtBuilder setIssuedAt(Date iat) {
252         if (iat != null) {
253             ensureClaims().setIssuedAt(iat);
254         } else {
255             if (this.claims != null) {
256                 //noinspection ConstantConditions
257                 this.claims.setIssuedAt(iat);
258             }
259         }
260         return this;
261     }
262
263     @Override
264     public JwtBuilder setId(String jti) {
265         if (Strings.hasText(jti)) {
266             ensureClaims().setId(jti);
267         } else {
268             if (this.claims != null) {
269                 claims.setId(jti);
270             }
271         }
272         return this;
273     }
274
275     @Override
276     public JwtBuilder claim(String name, Object value) {
277         Assert.hasText(name, "Claim property name cannot be null or empty.");
278         if (this.claims == null) {
```

図 5: コードクローンのソースコード表示画面の一部

図 4 は WEB クライアントに出力される散布図の例である．散布図には左上を原点として，縦横両軸に同じ順番でファイルが並べられている．他の検出結果に含まれていないコードクローンが多いほど，対応する点が赤色に近い色で表示される．逆に，他の検出結果に含まれているコードクローンが多いほど，その点は黄色に近い色で表示される．完全に他の検出結果と一致する場合のみ，その点は緑色で表示される．

また，図 5 のように，散布図の各点に対応するコードクローンをソースコードレベルで確認することもできる．この画面では， 2 つの検出結果の両方に含まれているコードクローンは黄緑色で，片方の検出結果にのみ含まれているコードクローンはどちらに含まれているかによってオレンジ色もしくは紫色でそれぞれハイライトされる．

### 3.3 REST API

提案システムは外部サービスからアクセス可能な REST API を提供している．REST API は多くの WEB サービスで利用されている一般的な HTTP ベースの API である．提案システムが提供する REST API のうちいくつかを以下に示す．

**POST** /projects

外部の Git リポジトリをファイルシステムにインポートする．リクエストには対象となる Git リポジトリの URL が含まれる．インポートしたリポジトリに対する一連の検出作業をプロジェクトと呼ぶ．インポートに成功するとプロジェクト名やインポート日時を JSON 形式で返却する．失敗した場合，エラー情報を JSON 形式で返却する．

**POST** /projects/{name}/history

{name} で指定されたプロジェクトに対してコードクローン検出を行う．リクエストには検出に使用する検出ツールや検出パラメータが含まれる．与えられた検出パラメータに間違いがなく，リクエストが検出リクエスト DB に追加されると，リクエストに対応する検出履歴 ID を JSON 形式で返却する．失敗した場合，エラー情報を JSON 形式で返却する．

**GET** /projects/{name}/history/{id}/artifacts/clones.json

{name} で指定されたプロジェクトの，{id} と等しい ID を持つ検出履歴の検出結果を JSON 形式で返却する．

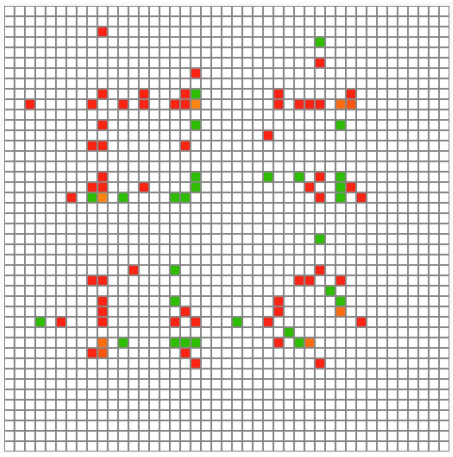


図 6: 比較結果の散布図

## 4. 適用事例

本章では，提案システムを用いたコードクローン分析の事例について述べる．この事例では，異なる検出ツールを用いてコードクローンを検出し，検出結果を比較することで多角的にコードクローンを分析する．そして，一方では検出できなかった集約可能なコードクローンを発見することを目的とする．

まず，提案システムに Joda-Time(注1) の Git リポジトリをインポートし，NiCad と CCVolti [13] を使用してコードクローン検出を行った．NiCad は Linux 上で，CCVolti は Windows 上で動作するコードクローン検出ツールである．これらのツールを同一マシンで動作させるためには，仮想マシンや Cygwin(注2) などのインストールが必要であるが，提案システムではブラウザからアクセスするだけで利用することができる．

次に，提案システムを用いて CCVolti と NiCad の検出結果を比較した．図 6 に WEB クライアントに表示された比較結果の散布図を示す．緑色の点は NiCad で検出されたコードクローンが全て CCVolti でも検出されていることを意味する．また，赤色やオレンジ色の点は NiCad でのみ検出されたコードクローンが存在していることを意味する．NiCad でのみ検出

(注1)：https://github.com/JodaOrg/joda-time.git
(注2)：https://www.cygwin.com/

EthiopicChronology.java

```
public static EthiopicChronology getInstance(DateTimeZone zone, int minDaysInFirstWeek) {
    if (zone == null) {
        zone = DateTimeZone.getDefault();
    }
                                ⋮
                if (zone == DateTimeZone.UTC) {
                    // First create without a lower limit.
                    chrono = new EthiopicChronology(null, null, minDaysInFirstWeek);
                    // Impose lower limit and make another EthiopicChronology.
                    DateTime lowerLimit = new DateTime(1, 1, 1, 0, 0, 0, 0, chrono);
                    chrono = new EthiopicChronology
                        (LimitChronology.getInstance(chrono, lowerLimit, null),
                         null, minDaysInFirstWeek);
                } else {
                    chrono = getInstance(DateTimeZone.UTC, minDaysInFirstWeek);
                    chrono = new EthiopicChronology
                        (ZonedChronology.getInstance(chrono, zone), null, minDaysInFirstWeek);
                }
                chronos[minDaysInFirstWeek - 1] = chrono;
            }
        }
    }
    return chrono;
}
```

CopticChronology.java

```
public static CopticChronology getInstance(DateTimeZone zone, int minDaysInFirstWeek) {
    if (zone == null) {
        zone = DateTimeZone.getDefault();
    }
                                ⋮
                if (zone == DateTimeZone.UTC) {
                    // First create without a lower limit.
                    chrono = new CopticChronology(null, null, minDaysInFirstWeek);
                    // Impose lower limit and make another CopticChronology.
                    DateTime lowerLimit = new DateTime(1, 1, 1, 0, 0, 0, 0, chrono);
                    chrono = new CopticChronology
                        (LimitChronology.getInstance(chrono, lowerLimit, null),
                         null, minDaysInFirstWeek);
                } else {
                    chrono = getInstance(DateTimeZone.UTC, minDaysInFirstWeek);
                    chrono = new CopticChronology
                        (ZonedChronology.getInstance(chrono, zone), null, minDaysInFirstWeek);
                }
                chronos[minDaysInFirstWeek - 1] = chrono;
            }
        }
    }
    return chrono;
}
```

図 7: NiCad でのみ検出されたクローンペア

されたコードクローンの 1 つを図 7 に示す．

この 2 つのコードクローンは図 7 中下線部の違いを除いて一致している．CCVolti は TF-IDF を用いてコードブロックをベクトル化し，そのベクトルのコサイン類似度からコードクローンを判定する．この適用事例ではこのコサイン類似度の閾値として 0.9 を用いたが，図 7 のコード片はコサイン類似度が閾値を下回ったため CCVolti では検出されなかったと考えられる．CopticChronology クラスと EthiopicChronology クラスのコンストラクタの動作は，共に親クラスである BasicFixedMonthChronology クラスのコンストラクタを呼び出すのみである．そのためこのコードクローンを解消するためには BasicFixedMonthChronology クラスに getInstance メソッドを集約すれば良い．実際に手作業でこのコードクローンを集約したところ約 40 行のコードを削減することができた．また，集約後のソースコードは Joda-Time リポジトリに用意してあるテストを全て通過した．

このように，異なる検出ツールから得られた検出結果を比較することで，より正確なコードクローン分析が可能となる．従来，動作環境の違いや煩雑なセットアッププロセスなどから同一マシン上で複数の検出ツールを利用することは手間のかかる作業であった．提案システムではブラウザからアクセスするだけで複数の検出ツールを利用することができ，より正確なコードクローン分析を支援する．

## 5. ま と め

本研究では，SaaS 型コードクローン分析システムを提案した．提案システムでは，多様なコードクローン検出ツールに対応するため，コードクローン検出サーバとシステム間でデータ伝達方式の共通仕様を定め，検出パラメータの定義を外部から与える．これにより，システムに手を加えることなく新たな検出ツールをバックエンドに追加し，動的に検出パラメータ設定フォームを生成することが可能となった．

また，適用事例から，提案システムを用いることにより異なる検出ツールの検出結果の比較を容易に行えることを示した．

今後の課題として，提案システムに散布図やツリーマップなどのコードクローン可視化手法を取り入れ，既存の分析ツールと比較した使いやすさや有用性について評価を行うことが挙げられる．

提案システムは **sel.ist.osaka-u.ac.jp/webapps/ccx/** で公開されており，GitHub や GitLab などでホスティングされた Git リポジトリに対してコードクローンの検出を行うことができる．

**謝辞** 本研究は JSPS 科研費 18H04094 の助成を受けた．

## 文 献

[1] L. Barbour, F. Khomh, and Y. Zou, "An empirical study of faults in late propagation clone genealogies," *Journal of Software: Evolution and Process*, Vol. 25, pp. 1139–1165, 2013.

[2] M. Mondal, B. Roy, C. K. Roy, and K. Schneider, "An Empirical Study on Bug Propagation through Code Cloning," *Journal of Systems and Software*, Vol. 158, 2019.

[3] V. Cho and A. Chan, "An integrative framework of comparing SaaS adoption for core and non-core business operations: An empirical study on Hong Kong industries," *Information Systems Frontiers*, Vol. 17, pp. 629–644, 2015.

[4] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.

[5] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, Vol. 28, pp. 654–670, 2002.

[6] V. Bandi, C. K. Roy, and C. Gutwin, "Clone Swarm: A Cloud Based Code-Clone Analysis Tool," in *Proc. IWSC 2020*, pp. 52–56, 2020.

[7] K. Matsushima and K. Inoue, "Comparison and Visualization of Code Clone Detection Results," in *Proc. IWSC 2020*, pp. 45–51 , 2020.

[8] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?," in *Proc. ICSE 2009*, pp. 485–495, 2009.

[9] Y. Ueda, Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, "Gemini: Code clone analysis tool," in *Proc. ISESE 2002*, pp. 31–32, 2002.

[10] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: scaling code clone detection to big-code,", in *Proc. ICSE 2016*, pp. 1157–1168, 2016.

[11] J. R. Cordy and C. K. Roy, "The nicad clone detector," in *Proc. ICPC 2011*, pp. 219–220, 2011.

[12] M. Asaduzzaman, C. K. Roy, and K. Schneider, "VisCad: flexible code clone analysis support for NiCad," in *Proc. IWSC 2011*, pp. 77–78. 2011.

[13] 横井 一輝, 崔 恩瀞, 吉田 則裕, 井上 克郎, "情報検索技術に基づく細粒度ブロッククローン検出," コンピュータソフトウェア, Vol.35, No.4, pp.16–36, 2018.