

Webアプリケーション特有の コードクローンの特徴分析

奈良先端科学技術大学院大学
特任助教 玉田 春昭

エンピリカルソフトウェア工学研究会
於 キャンパス・イノベーションセンター1階 国際会議室
2007.7.9

※ 玉田 春昭, 森崎 修司, 吉田 則裕, 楠本 真二, 井上 克郎, “APIの使用に伴うコードクローンの特徴分析”, 情報処理学会研究報告, Vol.2007-SE-156-9/2007-EMB-5-9, pp.57--62, May 2007.

背景

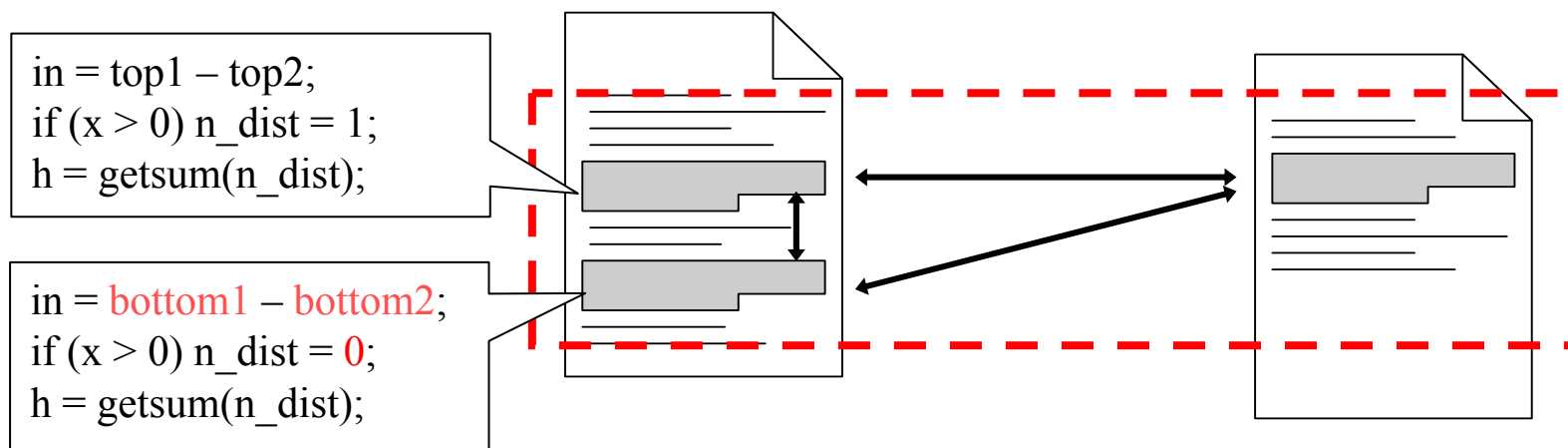
- 近年のソフトウェア開発ではAPIやフレームワークを使用することが多くなってきている
- APIやフレームワークは、一定の手順に従った使い方を想定している

⇒ 似たコードを書かざるを得ず、コードクローンが混入する

- そのようなソフトウェアの保守が重ねられると
 - 以下の区別が付きにくく、保守性を下げる原因となる
 - APIの使用により混入したコードクローン
 - コピー&ペーストで作成されたコードクローン

コードクローン

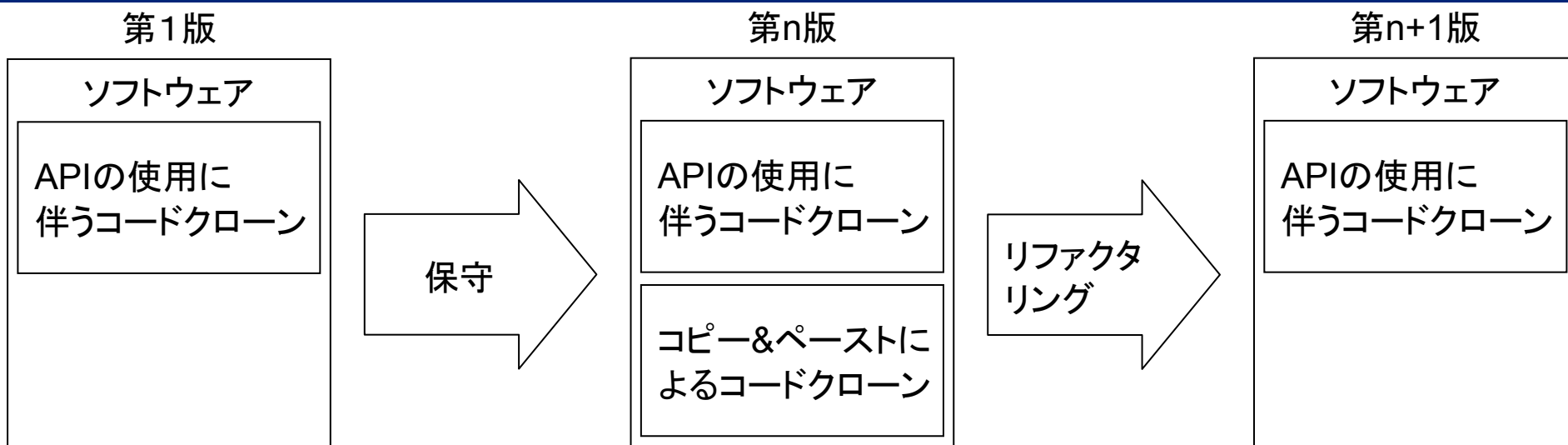
- ソースコード中に含まれる任意の文字列のうち、他の部分列と一致または類似しているもの。
 - 保守を困難にされている
- 一致もしくは類似している部分列の対はクローン関係を持つという
- クローン関係の定義は、本研究ではCCFinder^[1]の検出方法とする



[1] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, IEEE Trans. on Software Engineering, Vol.28, No.7, pp.654-670 (2002).

※ コードクローン分析は大阪大学主体で行っています。

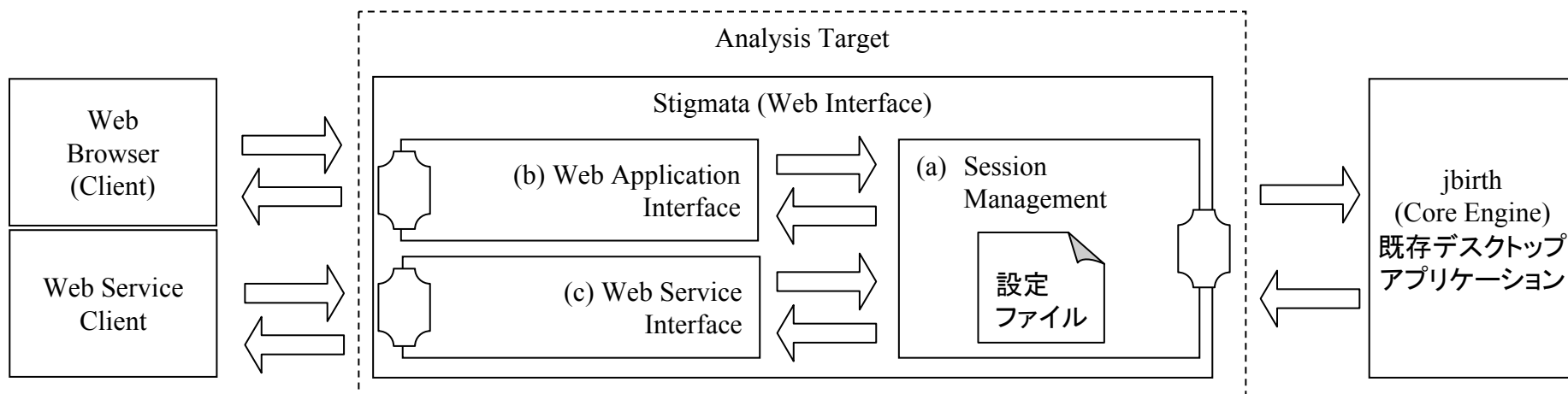
目的



- APIの使用に伴うコードクローンなのか、コピー&ペーストによるコードクローンなのかを区別することが重要
- ツールではこれらを区別することが難しい
- そこで、コードクローンとして抽出されたコード断片の意味解析を行い、コードクローンの特徴を元に、除去可能なクローンであるのか、除去できないクローンであるのかを分類する
 - 最終的な目的は、これらを自動判別することである

分析対象のアプリケーション — 概要

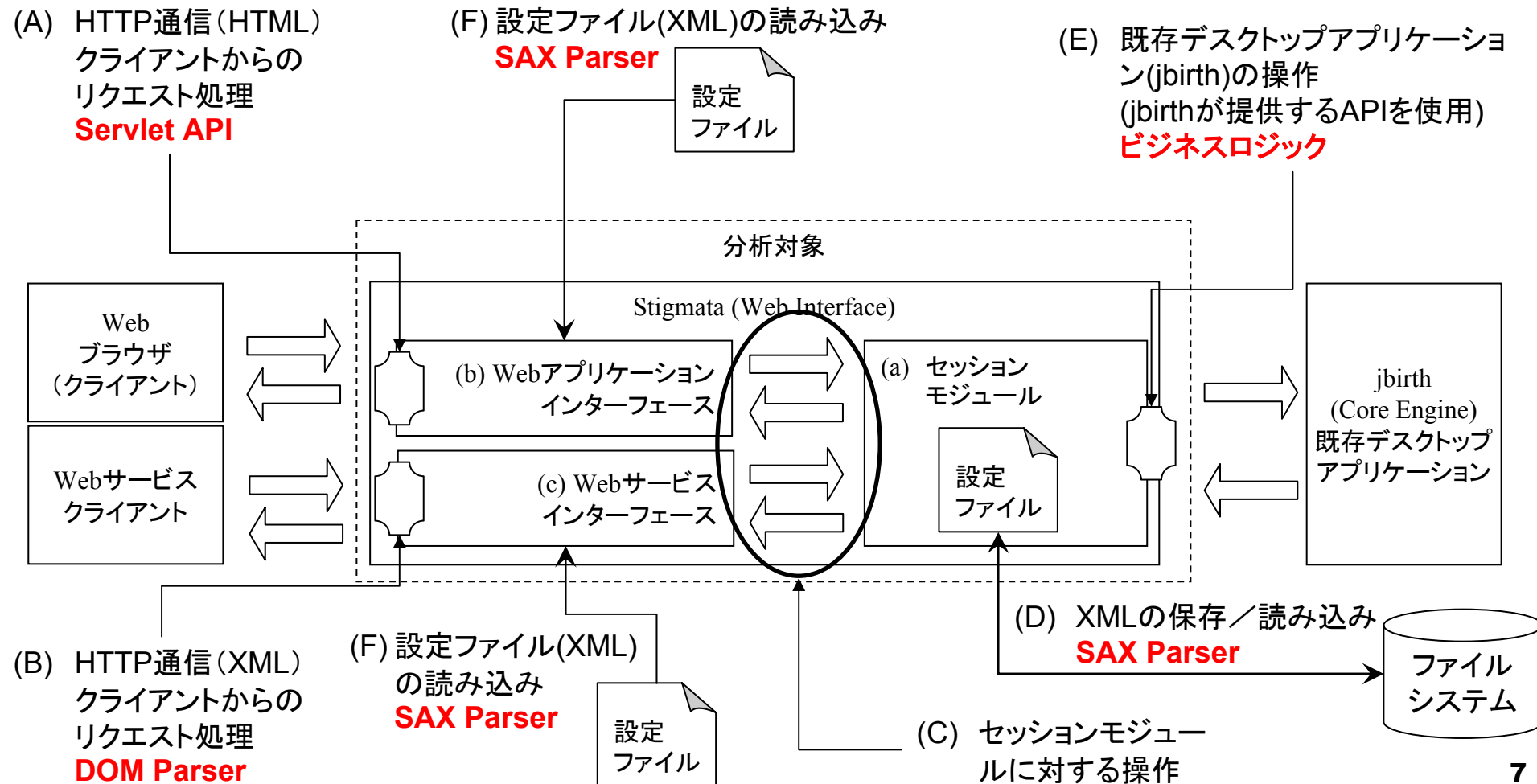
- 分析対象のWebアプリケーション
 - 既存デスクトップアプリケーションをWebアプリケーション化した
 - Webブラウザインターフェース, Webサービスインターフェースを持つ
 - Webインターフェース部分はスクラッチから作成された
 - Java Servlet, Apache Axis2を使用, Apache Tomcat上で動作する



分析対象のソフトウェア — 規模, 仕様

- 仕様
 - ファイルをアップロードし, 結果を表示する
 - 結果をXML形式, CSV形式でダウンロードできる
 - クライアントを互いに識別する
 - セッション管理を行う
 - 一定期間経過後, セッションはクリアされる
 - 認証は行わない
 - セッション管理に伴う雑多な情報はサーバ側のディスクにXML形式で保存される
- プロダクトの規模はJavaソースコード50ファイル, 8,000ステップ
- 開発規模は3人月

分析対象のソフトウェア — 設計



Session management

Web Application Web Services

Session management

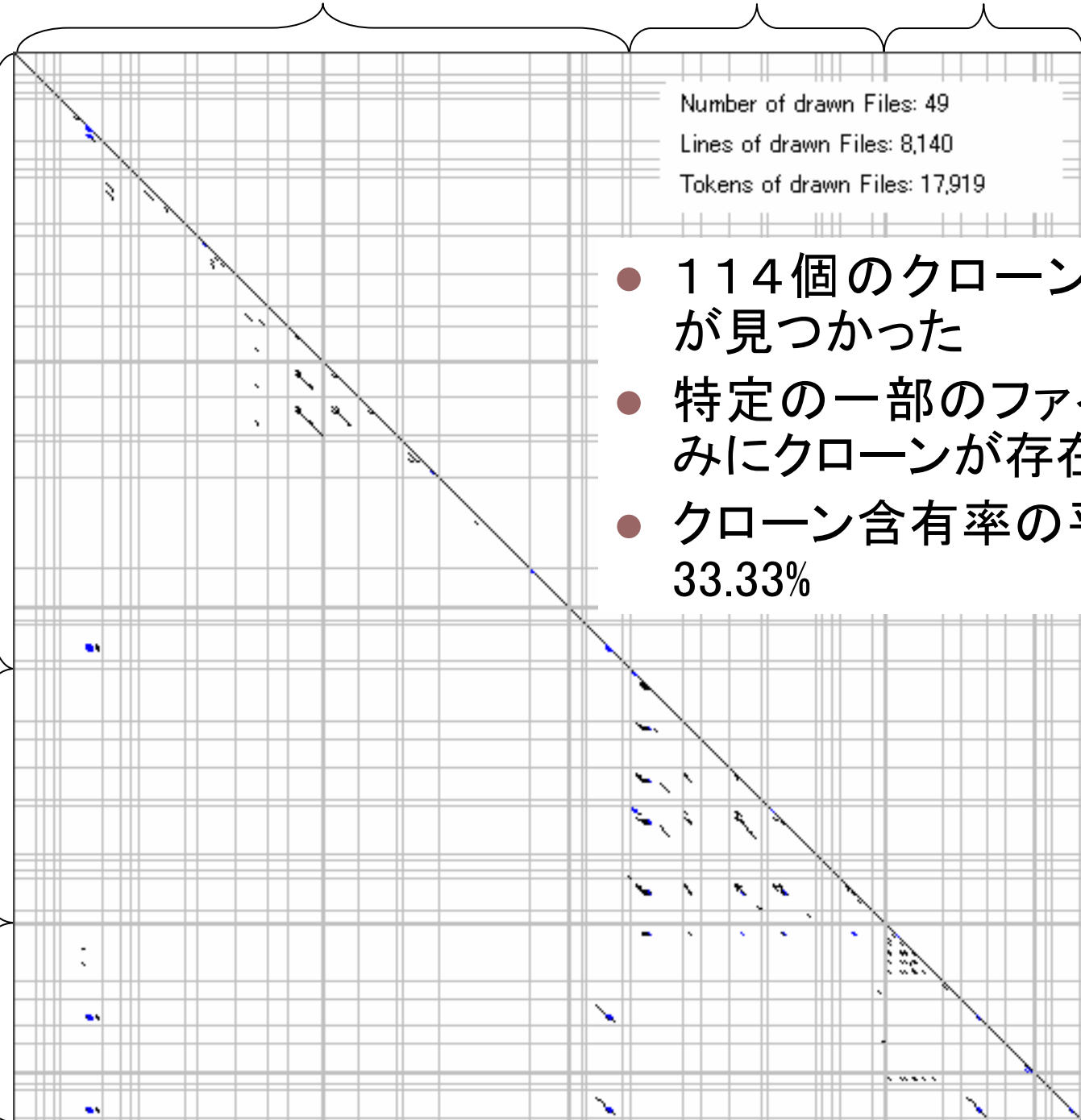
Web Application
Web Services

Number of drawn Files: 49

Lines of drawn Files: 8,140

Tokens of drawn Files: 17,919

- 114個のクローンセットが見つかった
- 特定の一部のファイルのみにクローンが存在する
- クローン含有率の平均が33.33%



検出されたクローンの分類

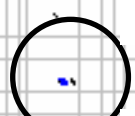
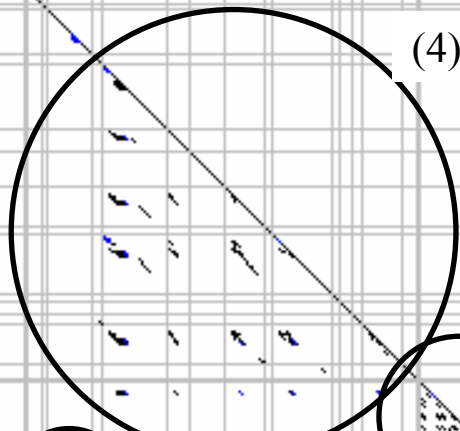
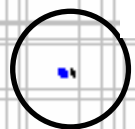
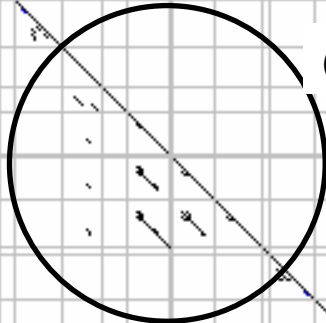
- クローンを構成するトークンの意味解析による分類
 - クローンの混入原因を調査する
- クローンとして得られたコードの処理内容ごとにグループ分けを行った
 - 実務開発経験者2名により分類された

Session management

Web Application Web Services

Session management

Web Application
Web Services



Number of drawn Files: 49
Lines of drawn Files: 8,140
Tokens of drawn Files: 17,919

検出されたクローンの分類

- クローンを構成するトークンの意味解析による分類
 - クローンの混入原因を調査する
- クローンとして得られたコードの処理内容ごとにグループ分けを行った
 - 開発経験者2名により分類された

(1), (3), (6)~(10)	SAX Parser を用いたXML文書の読み込み処理により混入したコードクローン。 各種設定ファイルの読み込み
(2)	セッションモジュールに対する操作に伴い混入したコードクローン
(4)	Java Servlet によるWebブラウザからのリクエスト処理の振り分けに伴い混入したコードクローン
(5)	DOM Parser を用いたXML文書の読み込み処理により混入したコードクローン。 Webサービスクライアントからのリクエスト処理

(1), (3), (6), (7), (8), (9), (10)

SAXによるXML読み込み部のコードクローン

- 似た構造のコード断片
 - XMLのタグ名による振り分けによるコードクローン
 - 開始タグ読み込み時, タグの中身読み込み時, 終了タグ読み込み時のイベント処理
- 完全に同一のコード断片
 - SAXパーサをインスタンス化し, Handlerと入力ストリームを設定する部分のコードクローン

```
public void parse(InputStream in) throws IOException {  
    SAXParserFactory factory = SAXParserFactory.newInstance();  
    SAXParser parser = factory.newSAXParser();  
    parser.parse(in, this);  
    parseEnd = true;  
} // try-catch 節は省略
```

(2) セッションモジュールに対する操作に伴い混入したコードクローン

- 内部的に2つのListを保持しており、それぞれのListに対する処理が異なるメソッドで定義されていた
- このクローンは避けるべきクローン
 - このグループに属するクローンは特定のAPIに依存するものではない
 - 2, 3個の変数や定数のみが異なっているため、一つのメソッドにまとめることが可能

```
public void addX(InputStream in,
                String obj){
    String[] tags = { X_AXIS_OBJECT, obj };
    addObject(in, tags);
    if(!targetX.contains(objectName)){
        if(targetX.add(objectName)){
            saveSession();
        }
    }
    else{
        removeObject(tags);
        throw new IOException(...);
    }
}

public void addY(InputStream in,
                String obj){
    // addXとほぼ同一の内容
}
```

(4) Webブラウザからのリクエスト処理に伴い発生したコードクローン

- リクエストがあったURLと処理内容を表す文字列で、実際に行うべき処理を決定している
 - URL文字列と、クライアントから渡される処理内容を示す文字列の2つから処理を振り分けるif文が見られた

```
public void doGet(HttpServletRequest req, HttpServletResponse res){
    :
    if(action.getActionCommand().equals(POST_ACTION_COMPARE)){
        someProcess1();
    }
    else if(action.getActionCommand().equals(POST_ACTION_COMPARE)){
        someProcess2();
    }
    else if...
}
```

(5) DOMによるXML読み込み部のコードクローン

- 検出されたコードクローンでは、以下の2つの処理が行われている
 - Iteratorによるタグの列挙のための繰り返し
 - 繰り返しの中に含まれるタグ名による振り分けのための条件分岐

```
for(Iterator itr = element.getParent().getChildren(); itr.hasNext(); ){
    OMLElement oMLElement = (OMElement) itr.next();

    if(oMLElement.getLocalName().equalsIgnoreCase("birthmarks")){
        :
    }
    else if(oMLElement.getLocalName().equalsIgnoreCase("target")){
        :
    }
    else if(oMLElement.getLocalName().equalsIgnoreCase("format")){
        format = oMLElement.getText();
    }
}
```

除去可能性によるコードクローンの分類

- 除去が難しいコードクローン
 - 一概に悪いと言えないコードクローン
 - 誰が書いても大きな差が出ないコード
 - サンプルプログラムにも同じクローンが見られることが多い
- 除去可能なコードクローン
 - 避けるべきコードクローン
 - 特定のAPIに起因しないコードクローン
 - コピー&ペーストで作成されたと考えられる
 - コストとの兼ね合いにより避けるべきコードクローン
 - 本来は避けるべきクローンだが、あえて混入させているクローン
 - 十分に枯れた, 十分にテストされたコードのコピー&ペースト
 - APIの使用に起因するコードクローン
 - クローンを避けるための代替手段を行うためのコストが高いもの

まとめと今後の課題

- まとめ
 - スクラッチから開発されたWebアプリケーションのコードクローン分析を行った
 - Servlet, SAX Parser, DOM Parserを使うときに現れるコードクローンを発見し、混入した原因を分析した
 - 見つかったコードクローンを3種類に分類することができた
 - 一概に悪いと言えないコードクローン
 - 避けるべきコードクローン
 - コストとの兼ね合いにより避けるべきコードクローン
- 今後の課題
 - 3種類のコードクローンが版を重ねることでどのように変化するのかを分析する
 - コードクローンの成長と保守性の関連性についての調査を行う
 - APIの使用に伴うコードクローンとコピー&ペーストにより混入したコードクローンの自動判別