APIの使用に伴うコードクローンの特徴分析

玉田春昭[†]　森崎修司[†]　吉田則裕[††]
楠本真二[††]　井上克郎[††]

本稿では，ある特定の API を使うときに現れやすいコードクローンを見つけ出し，その特徴を分析することを目的とする．分析の対象としてある開発会社 A 社が開発した Web アプリケーションを選択した．この実プロジェクトで開発された Java ソースコードを分析し，Servlet や Web Service API, XML SAX Parser，XML DOM Parser などの API やフレームワークを使うときにコードクローンが現れることを確認した．そして，そこに使用している API 固有の特徴が現れていることを確認した．

An analysis of code clone characteristics in use of APIs

Haruaki Tamada,[†] Shuji Morisaki,[†] Norihiro Yoshida,[††]
Shinji Kusumoto[††] and Katsuro Inoue [††]

In this paper, we aim to find out code clones to appear in use of certain APIs, and analyze characteristics of them. As the target of analysis, we chose a web application which certain A company developed. We analysed Java source codes of the web application and we can find out code clones in use of APIs and frameworks such as servlet, web service API, xml sax parser, and xml dom parser. Moreover, we recognized those code clones have unique characteristics of APIs.

1. はじめに

ソフトウェアを開発するとき，往々にしてコードクローンが混入する．コードクローンとは同一の，もしくは非常に似通ったコード片（ソースコードの断片）のことを指す．コードクローンが多く含まれているソフトウェアは保守が困難になると古くから指摘されている[2),3)]．例えばあるコード片にバグが含まれていた場合，そのコード片を含む全てのコードクローンに対して修正済みか否か，また，修正を要するのか，そうでないかを精査する必要があり，保守に要するコストが増大する．

しかしながら，API やフレームワークを使えば，特定の使い方を強要され，似通ったコードを書かざるを得ない場合が多い．そのようなクローンは従来は避けようのないクローンとして扱われてきた．しかし，避けようのないクローンを大量に含んだソフトウェアがバージョンを重ねていくと，避けようのないクローンとそうでないクローンの区別が付きにくく，また，避けようのないクローン自身が保守性を下げる恐れもある．API のバージョンを上げるとき，使い方に多少の変更が出てくる場合がこれにあたる．

そこで本稿ではまず，実業務において開発されたソフトウェアに含まれるクローンにはどのような特徴があるのかを分析する．そのため A 社が開発した Java Servlet を用いた Web アプリケーション Stigmata に CCFinder[9)] を適用し，得られたコードクローンを分析した．その結果，Java Servlet，そして，XML の入出力を扱う上で特徴的なクローンが得られた．

2. 関連研究

Kapser らは理解支援を目的としてコードクローンの自動分類手法の提案を行っている[5)]．この手法では，コードクローンが所属するディレクトリ階層やソースコードの構造により分類を行っている．また，彼らが開発したツール CLICS は，この自動分類機能を実装している[6)]．

Balazinska は，コード上の差異に基づいたコードクローンの分類手法を提案している[13)]．また，彼らはこの手法を用いたリファクタリング支援手法を提案している[14)]．

† 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†† 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

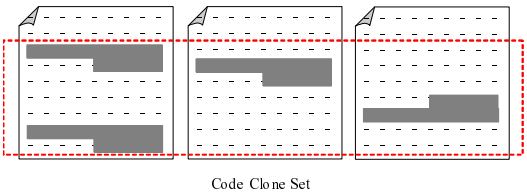

図 1 コードクローンとコードクローンセット

Kapser らは，コードクローンの作成が一概に問題を引き起こすとは限らないと主張している[7]．この論文の中で，コードクローンが作成される理由をいくつか挙げ，理由ごとにコードクローンを作成する利点や欠点，作成されたコードクローンの管理方法について述べている．理由の一つに，特定の API やライブラリを用いた開発を挙げており，利点，欠点をそれぞれ述べている．

3. コードクローン分析

3.1 コードクローン

一般的に，コードクローンとはソースコード中に含まれる任意の部分列のうち，他の部分列と一致また類似しているものを指す．また，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コードクローン検出手法により異なるが，コードクローン検出ツール CCFinder のように検出するクローン関係が同値関係であるとき，同値類を特にクローンセットと呼ぶ．

図 1 に示す網掛け部それぞれがクローンとなった部分列を表しており，網掛け部の集合 (点線で囲まれた 4 つの網掛け部) がクローンセットである．

また，一つのファイル中に含まれるクローンの量を表すため，ファイルに含まれるクローンのトークン数のファイルの全トークン数での割合をクローン含有率として表す[4]．

3.2 キーアイデア

開発されたソフトウェアにコードクローンが含まれないことはほとんどなく，何らかのコードクローンが存在する．コードクローンが混入する原因として，主に以下の 6 種類が考えられている[3),9)]．

(a) コピー&ペーストによる再利用により現れるクローン
(b) 定型処理により現れるクローン
(c) 言語が未サポート機能の実現のためのクローン
(d) パフォーマンス改善を目的にした意図的なクローン
(e) 自動生成により現れるクローン
(f) 偶然により現れるクローン

一方，今日の開発では多くの場合，フレームワークやライブラリを用いている．それらの API を用いるときにはある一定の使い方を強要され，非常に似通ったコードとなることが多い．このような API を使う上で混入するコードクローンは，コードクローンが混入する原因のうち (b) や (c) に相当し，一概に悪いコードクローンではないと言われていた[7]．このようなコードクローンを修正すべきコードクローン，すなわち悪いコードクローンに対する意味で，良いコードクローンとここでは呼ぶ．

しかし，開発するアプリケーションの規模が大きくなるにつれ，一般に使用するフレームワークの規模も大きくなり，また利用するライブラリの数も増加する．それに伴い，API を用いることによって混入するコードクローンの量が増加すると考えられる．そのため，良いクローンであるのか，悪いクローンであるのかを区別することが非常に難しくなってくる．加えて，良いクローンであってもクローンセットの要素数が多くなっていくと，API に対する操作の変更に伴うコード修正のコストが増加するため，保守性も下がっていく．

そこで，まず，API 使用に伴って現れる API 固有のクローンの特徴を分析するため，スクラッチから開発されたアプリケーションからコードクローンを抽出する．そして，得られたクローンをコードの処理内容ごとにグループに分類し，グループごとのクローンの特徴を調査する．

クローンの特徴を明示することによって，そのクローン部分を修正するとき，その特徴に従った修正であれば，保守性を下げにくく，その特徴を破るような修正であれば保守性を著しく下げることに繋がると考えられる．

4. 分 析 対 象

コードクローン分析の対象としたソフトウェアは A 社がスクラッチから開発した Java Servlet を用いた Web アプリケーションである．以降，この Web アプリケーションを Stigmata と呼ぶ．Stigmata は既存デスクトップアプリケーションに Web アプリケーションのインターフェースを与えたものであり，Web 上から与えられたパラメータを処理エンジンである既存デスクトップアプリケーション (jbirth) に渡している．Stigmata のプロダクト規模は Java ソースコードが 50 ファイル，約 8,000 ステップ，JSP が 27 ファイル，約 2,500 ステップと小規模なものである．また，開発規模は 3 人月である．なお，既存デスクトップアプリケーション jbirth は業務として開発されたものではないため，今回の分析には含めていない．

Stigmata の仕様はあるフォーマットに従ったファイルをアップロードし，サーバ側でアップロードされたファイルに対して処理を行って，処理エンジンに渡す．そして処理エンジンからの結果を XML もしくは CSV で返すものである．クライアントごとにユーザ認証は行わないが，各クライアントを識別するため，

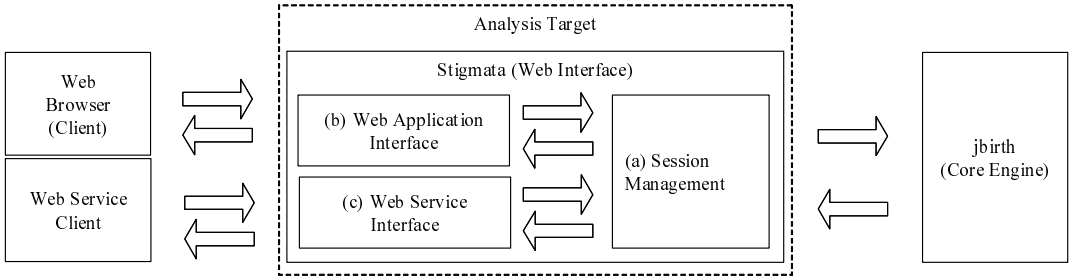

図 2 分析対象アプリケーションの概要

セッションを用いている．そして，アップロードされたファイルなどはセッションに基づいてサーバ側のディスクに保存され，一定時間が経過すると削除される．サーバ側でのファイル保持に伴う雑多な情報は XML 形式で保存されている．

Stigmata では以下の API を使用している．ただし，今回の開発では工期が短く，作成するアプリケーション自体も大規模なものではないため，Struts[1] や Spring[11] などの大規模フレームワークは使用していない．

- Java Servlet
- Web Service
- XML SAX Parser
- XML DOM Parser

Stigmata は大きく 3 つのモジュールから成り立っている．それぞれ，(a) クライアントのセッションを管理する部分，(b)Web ブラウザからのリクエストを処理する Web アプリケーション部分 (Servlet)，(c)Web サービス部分である．

モジュール (a) はセッションを管理する．このモジュールはクライアントからのリクエストに応じてセッションを発行し，クライアントの情報を XML 形式でディスクに保存する．また，セッションが切れた場合に自動的に保持された内容の削除処理も行う．

モジュール (b) では，Web ブラウザからのリクエストを処理する部分である．このモジュールではクライアントからのリクエストに応じてモジュール (a) に対して属性値の追加・削除を行い，処理エンジン部にリクエストを渡し，結果をクライアントに返す処理を行う．

モジュール (c) はモジュール (b) とほぼ処理内容であるがインターフェースが JSP を介した Web アプリケーションではなく，Web サービスとなっている．

分析対象アプリケーションの概要を図 2 に示す．図中の矢印はデータの流れを表す．図中の Analysis Target 部分が A 社が開発した部分で，本研究の分析対象である．

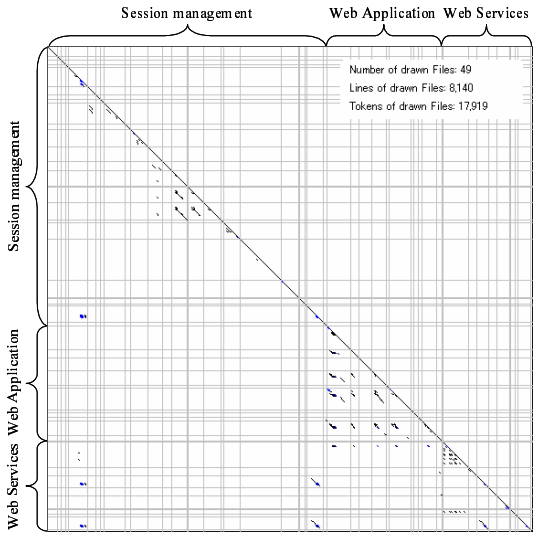

図 3 見つかったクローンの概観

5. 分 析 結 果

5.1 分析結果の概要

対象アプリケーションに CCFinder を使ってコードクローンを計測したところ，114 個のクローンセットが見つかった．

見つかったクローンの概観を図 3 のスキャッタープロットに示す．縦軸と横軸にソースコードのトークンを出現順に並べ，水平方向のトークンと垂直方向のトークンが一致した場合に点をプロットしている．また，図中の縦軸・横軸に記している名前は各モジュールの名前である．図を見ると，特定の一部のファイルにのみクローンが見つかっていることがわかる．

クローン含有率ごとのファイル数を図 4 に示す．横軸がクローン含有率で縦軸が対応するクローン含有率に含まれるファイル数である．クローンを含まないファイルが一番多く，クローン含有率が 40%を超えるファイルの数が全ソースファイルの 36%である 18 ファイルとなっている．また，含有率が 80%以上であるファイルは存在しなかった．そして，クローン含有

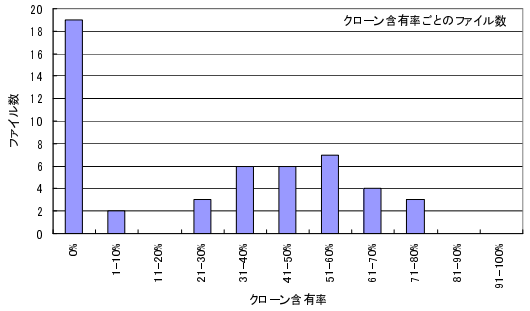

図 4 クローン含有率ごとのファイル数

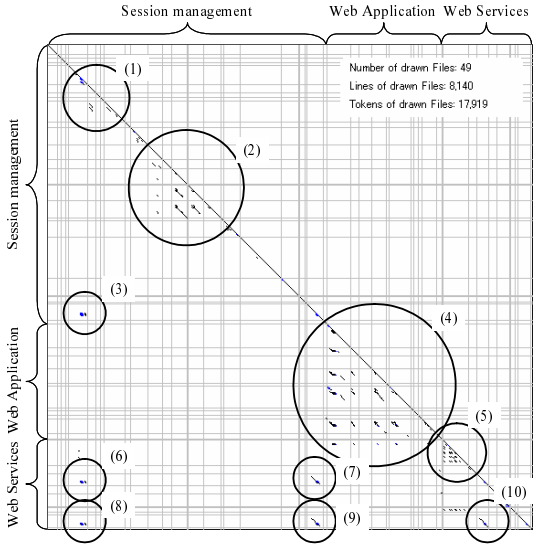

図 5 処理内容ごとのクローンの散布状況

率の平均が 33.33%と産業界でのソフトウェアとしては少なくなっている[4].

出力されたクローンを分析した結果，いくつかのグループに分かれることがわかった．図 5 に処理内容ごとにクローンをグループ分けした図を示す．図中の円は各グループを表しており，円の右上にグループの番号を示している．それぞれのグループに存在するクローンは必ずしも同一のクローンセットに含まれているわけではないが，クローンとして得られたコードの断片から開発経験者 2 名により分類された．それぞれのグループの処理内容を表 1 に示す．なお，モジュール間で機能の切り分けが成されていたため，モジュールをまたがるグループは存在しなかった．

5.2 分析結果の詳細

クローンのグループ (1), (3), (6)–(10) はいずれも SAX (Simple API for XML) による XML の読み込み処理であり，XML で書かれた設定ファイルを読み込む部分で使われている．SAX は XML における開始タグの読み込み時，タグの中身の読み込み時，終了タグ読み込み時の各イベントに応じて Handler と呼ばれるクラスの所定のメソッドが呼び出される．それらのメソッドの処理内容はタグ名によって振り分けられることが一般的であり，Sun が提供する XML 操作のためのライブラリ JAXP の SAX のサンプルコード[10] においても if 文を使った処理の振り分けが行われている．Stigmata の当該部分においても，やはりタグ名により if 文を用いて処理を振り分けるコードが抽出された．更に，SAX Parser をインスタンス化して，Handler と設定を読み込むための入力ストリームを Parser に登録する部分が各読み込み処理のためのクラスに存在した．この部分は各クラスから完全に同一のコード断片が抽出された．このコード断片は図 6 に示す通りである．そのため，この部分については共通のスーパークラスを作り，その中で処理を行うべきであろう．

```
public void parse(InputStream in) throws IOException {
  try {
    SAXParserFactory factory =
      SAXParserFactory.newInstance();
    SAXParser parser = factory.newSAXParser();
    parser.parse(in, this);
    paseEnd = true;
  } catch (ParserConfigurationException e) {
    throw new IOException(e.getMessage());
  } catch (SAXException e) {
    throw new IOException(e.getMessage());
  }
}
```

図 6 SAX パーサの初期化部分

次に，クローンのグループ (4) は Web クライアントからのリクエストに応じた処理を行うため，処理を振り分けるコードが抽出された．Stigmata で使われている Servlet の構造はリクエストのあった URL と処理内容を表す文字列によって実際に行う処理を決定している．そのため，URL を表す文字列とクライアントからのリクエストを表す文字列とで，if 文による振り分けを行うコードが見られた．

そして，クローンのグループ (5) は DOM ツリーのルートの子を順番に処理するため `Iterator` を用いて子タグを順に取り出し，タグの名前により処理を振り分けている．抽出されたコードは for 文の中にタグ名による処理の振り分けのための if 文であった．

一方，クローンのグループ (2) はセッションに値を追加・削除しているコードであり，具体的なコードは `List` に値を追加，削除しているものである．ただし，セッションは二つの `List` を持っており，それぞれの `List` に対して別のメソッドで処理が行われている．具体的なコードは図 7 に示す通りである．このグループに含まれるコードは特定の API に依存するものではなく，また，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である．

表 1　各グループの処理内容

(1), (3), (6)–(10)	SAX による XML で書かれた設定ファイルの読み込み処理.
(2)	セッションに値を追加/削除している処理.
(4)	Web ブラウザからのリクエストの振り分け処理.
(5)	DOM による XML 形式の Web Service クライアントからのリクエストの読み込み処理.

```
public void addX(InputStream in, String obj)
    throws IOException, StigmataSessionException{
  String[] tags = { X_AXIS_OBJECT, obj, };
  addObject(in, tags);
  if(!targetX.contains(objectName)){
    if(targetX.add(objectName)){
      saveSession();
    }
    else{
      removeObject(tags);
      throw new IOException(MessageFormat.format(
        getMessage("exception.session.target.add"),
        new Object[]{ "X Axis", tags[0], tags[1] }
      ));
    }
  }
}

public void addY(InputStream in, String obj)
    throws IOException, StigmataSessionException{
  String[] tags = { Y_AXIS_OBJECT, obj, };
  addObject(in, tags);
  if(!targetY.contains(objectName)){
    if(targetY.add(objectName)){
      saveSession();
    }
    else{
      removeObject(tags);
      throw new IOException(MessageFormat.format(
        getMessage("exception.session.target.add"),
        new Object[]{ "Y Axis", tags[0], tags[1] }
      ));
    }
  }
}
```

図 7　グループ (4) に含まれるコード例

6. 考　　察

得られたコードクローンの各グループを見ると SAX を使った部分である (1), (3), (6)–(10) のクローンのうち，XML のタグ名により処理を振り分けている部分は SAX を使っている限り除去することが不可能であると考えられる．この部分は全く異なる人が書いても差がそれほど出るものではないためである．

また，DOM ツリーを辿るコードであるクローングループ (5) は，Stigmata においては for 文で Iterator を使ってタグを辿り，タグを辿る繰り返しの中で if 文を用いての条件分岐という形になっている．ただし，DOM ツリーを辿り目的のノードを得る方法は得られたコードのみではなく，例えば XPath[8] などを用いることもできる．しかし，一つのソフトウェア内で DOM ツリーの辿り方に一貫性がなければ DOM ツリーの辿り方の意図を掴むことが困難になり，保守性が下がると考えられる．そのため，DOM を利用したソフトウェアでは，DOM ツリーの辿り方に応じたクローンが抽出されると考えられる．

クライアントからのリクエスト処理の振り分けを行っているグループ (4) のクローンを除去することは可能である．現在の実装において，クライアントからの GET リクエストや POST リクエストに応じて，Servlet の doGet や doPost 内でリクエストの振り分け処理を行っている．この振り分け処理の存在は 1 つの Servlet が 1 つ以上の仕事を行っていることを示している．これを 1 つの Servlet が行う処理を 1 つのみにすれば Servlet 特有のクローンを除去することが可能である．ただし，似た機能であるからこそ 1 つの Servlet に処理をまとめていると考えられるため，クローンの除去のみを目的にコードを変更することは，Servlet の数が多くなりすぎて逆に保守性が下がる可能性もあり，また，多大なコストがかかるため，現実問題として適用すべきではないと考えられる．

Servlet 固有のクローンを除去するための異なるアプローチとして，Struts や Spring Framework，EJB などの上位フレームワークを使用することも考えられる．ただ，このアプローチもアーキテクチャ変更のコストやそれらフレームワークを導入するコストなど，許容できるコストを見極める必要がある．

最後にクローングループ (2) について考える．このグループは API 使用に伴うクローンではなく，アプリケーション固有のクローンであると考えられる．そのため，修正すべきクローンであると判断できる．

7. ま と め

本稿では，ある特定の API を使うときに現れやすいコードクローンを見つけ出し，その特徴を分析することを目的に，A 社が開発した Java 言語の Web アプリケーションを分析した．そして，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，そのクローンが混入した原因を分析した．その結果，悪いコードクローンと良いコードクローン，コストとの兼ね合いにより良いコードクローンに変更可能なコードクローンの 3 種類に分類できた．

今後の課題として，悪いコードクローン，良いコードクローン，回避可能なコードクローンの 3 種類が版を重ねることでどのように変化していくのかを分析す

る．そして，コードクローンの成長と保守性の関連性を調査していく．

謝辞 本研究の一部は、文部科学省「eSociety 基盤ソフトウェアの総合開発」の委託に基づいて行われた．

参 考 文 献

1) Apache Software Foundation: Apache Struts (2000). http://struts.apache.org/.
2) Baker, B.S.: A Program for Identifying Duplicated Code, *Computing Science and Statistics*, Vol.24, pp.49–57 (1992).
3) Baxter, I. D., Yahin, A., Moura, L. M. D., Sant'Anna, M. and Bier, L.: Clone Detection Using Abstract Syntax Trees, *ICSM: the International Conference on Software Maintenance*, pp.368–377 (1998).
4) 肥後 芳樹, 吉田 則裕, 楠本 真二, 井上 克郎: "産学連携に基づいたコードクローン可視化手法の改良と実装", 情報処理学会論文誌, Vol.48, No.2, pp.811-822., 2007
5) Cory Kapser, Michael W. Godfrey: Aiding Comprehension of Cloning Through Categorization. IWPSE 2004: pp. 85-94, 2004.
6) Cory Kapser, Michael W. Godfrey. "Improved Tool Support for the Investigation of Duplication in Software," Proc. 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 305-314, (2005).
7) Cory Kapser, and Michael W. Godfrey, "'Cloning Considered Harmful' Considered Harmful", Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), pp. 19-28 (2006).
8) Clark, J. and DeRose, S.: XML Path Language (XPath) Version 1.0 (1999). `http://www.w3.org/TR/1999/REC-xpath-19991116`.
9) 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).
10) Sun Microsystems, Inc.: Code sample —JAXP Using SAX (2007). `http://developers.sun.com/sw/building/codesamples/sax/index.html`.
11) www.springframework.org: Spring Framework (2003). `http://www.springframework.org/`.
12) 井上克郎，神谷年洋，楠本真二 : コードクローン検出法，コンピュータソフトウェア，Vol.18, No.5, pp.47–54 (2001).
13) Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and K.A. Kontogiannis, "Measuring Clone Based Reengineering Opportunities", Proc. 6th IEEE Int'l Symposium on Software Metrics (METRICS '99), pp. 292-303, Boca Raton, Florida, Nov. 1999.
14) Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and Lostas Kontogiannis, "Advanced Clone-Analysis to Support Object-Oriented System Refactoring", WCRE 2000, pp. 98-107, 2000.