Circle Packingを用いたコードクローン可視化手法

村上 寛明[1,a)]　肥後 芳樹[1,b)]　楠本 真二[1,c)]

概要：コードクローンとはソースコード中に存在する同一または類似するコード片を指す．一部のコードクローンはソフトウェア保守を困難にする要因であると考えられている．その理由は，あるコード片に対してバグ修正等の変更を加える際，そのコード片の全てのコードクローンについても同様の変更が必要であるか否かを検討しなければならないためである．コードクローンに対する変更の必要性を効率的に検討するため，著者らは Circle Packing を用いてコードクローンを可視化する手法を提案した．さらに提案手法をツール ClonePacker として実装した．ClonePacker の有用性を調べるため，被験者実験を通じて既存ツール Libra と比較した．被験者実験では，クローン分析に要する時間およびユーザビリティを評価した．その結果，双方について ClonePacker は Libra より優れていることを示した．ClonePacker は `http://sdl.ist.osaka-u.ac.jp/~h-murakm/clonepacker/` で公開されている．

1. はじめに

コードクローン（以降，**クローン**と表記する）とは，ソースコード中に存在する同一または類似するコード片を指す [1], [2]．一部のクローンはソフトウェア保守を困難にする要因である [3]．例えば，あるコード片に含まれるバグを修正する際，そのコード片の全てのクローンについて同様の修正が必要であるか否かを検討しなければならない．そのためソースコードに含まれるクローンの位置を把握することはソフトウェア保守において重要である．

図 1 は JFreeChart[*1] に存在する 2 つのコード片である．図 1(a) の 608 行目と図 1(b) の 469 行目には共に“`plot.setFixedRangeAxisSpace(space);`”という命令が記述されていた．図 1(a) の 608 行目は 2007 年 12 月 4 日に変更が加えられ，図 1(b) の 469 行目は 2008 年 5 月 28 日に変更が加えられた．コミットログには図 1(a) における変更はバグ修正のための変更であり，図 1(b) における変更は修正漏れのための変更であると書かれていた．つまり，図 1(a) と図 1(b) は同時に変更されなければならなかったが，開発者は図 1(b) における変更を見逃していた．ここで図 1 に示す 2 つのコード片はクローンである．2007 年 12 月 4 日の時点で開発者がこれらのクローンの位置を把握できていれば，同時に変更を加えることができたはずである．以上よりソフトウェア保守において，クローンの位置の把握が重要であることは明白である．

```
604:  protected void setFixedRangeAxisSpaceForSubplots(AxisSpace space) {
605:    Iterator iterator = this.subplots.iterator();
606:    while (iterator.hasNext()) {
607:      XYPlot plot = (XYPlot) iterator.next();
608: -    plot.setFixedRangeAxisSpace(space);
     +    plot.setFixedRangeAxisSpace(space, false);
609:    }
700:  }
```

2007年12月4日に変更が加えられた

(a) CombinedDomainXYPlot.java

```
465:  protected void setFixedRangeAxisSpaceForSubplots(AxisSpace space) {
466:    Iterator iterator = this.subplots.iterator();
467:    while (iterator.hasNext()) {
468:      CategoryPlot plot = (CategoryPlot) iterator.next();
469: -    plot.setFixedRangeAxisSpace(space);
     +    plot.setFixedRangeAxisSpace(space, false);
470:    }
471:  }
```

2008年5月28日に変更が加えられた

(b) CombinedDomainCategoryPlot.java

図 1　JFreeChart における変更
Fig. 1　Modifications in JFreeChart

ソースコードの中からクローンを自動的に見つけるために，さまざまなクローン検出ツールが開発されている [4], [5], [6]．多くのクローン検出ツールは，検出されたクローンの位置情報（ファイル名，開始行，終了行等）をテキストファイルとして出力する．そしてクローンの位置情報を基にして，クローンを可視化するツール [7], [8], [9] も開発されている．

Libra [7] はクローンを可視化するツールの 1 つである．Libra の使用手順は以下の通りである．

(1) 使用者は検出対象となるソースファイル群と 1 つのコード片を入力する．

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
[a)] h-murakm@ist.osaka-u.ac.jp
[b)] higo@ist.osaka-u.ac.jp
[c)] kusumoto@ist.osaka-u.ac.jp

[*1] `http://www.jfree.org/jfreechart/`

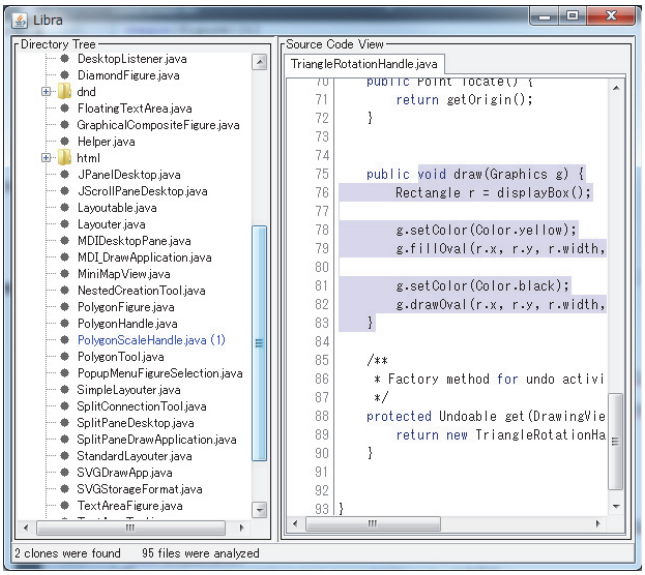


図 2 Libra のスクリーンショット
Fig. 2 Screenshot of Libra

(2) Libra は CCFinder [4] を用いて，入力されたコード片のクローンをソースファイル群から検出する．
(3) 使用者は検出されたクローンを目視で確認する．

図 2 に Libra のスクリーンショットを示す．左側にソースファイルのツリービュー，右側にツリービューによって選択されたソースファイルの内容が表示されている．クローンを含むソースファイルには，ソースファイル名の後にそのソースファイルが含むクローンの数が表示されている．右側のビューにおいてソースコード中のハイライトされた箇所がクローンである．しかし，Libra には以下に示す 2 つの課題点がある．

課題点 1: Libra はソースファイルの内容を全て表示し，ハイライトでクローンを示している．そのためクローンがソースファイルの下部にあると，スクロールバーを大きく動かさなければクローンを閲覧することができない．

課題点 2: Libra はクローンのタイプ（2.2 節で説明する）やサイズを可視化していない．

これらの課題点により Libra ではクローンの分析を効率的に行えない可能性がある．

Libra の課題点を改善するため，著者らは Circle Packing [10] を用いたクローン可視化手法およびメソッド単位のクローン検出手法を提案する．Circle Packing とは円の中に複数の円を描画することで，階層的なデータ構造を表現する手法である．Circle Packing については 2.4 節で述べる．本研究では Circle Packing を用いてファイルの階層構造（メソッド，ファイル，ディレクトリ）を表現し，それを用いてメソッド単位のクローンの情報（位置，タイプ，サイズ）を可視化する．著者らはこれまでにコード片単位のクローン検出手法を提案している [11]．本研究ではメソッド単位のクローンを可視化するために，メソッド単位のクローン検出手法を新たに提案する．またクローンのソースコードを表示させる際，スクロールバーの操作量を削減するために，ソースファイル全体ではなくクローンのみを表示させる．

図 3 クローンペアとクローンセット
Fig. 3 Clone Pair and Clone Set

提案手法を Eclipse プラグインとして実装し，それを ClonePacker と名付けた．ClonePacker の有用性を調べるため，被験者実験を通じて Libra と比較した．被験者実験では，クローン分析に要する時間およびユーザビリティを比較した．その結果，双方について ClonePacker は Libra より優れていることを示した．

本研究の貢献は以下の通りである．

- Circle Packing を用いたクローン可視化手法を提案した．
- 提案手法をツール ClonePacker として実装した．ClonePacker は著者らの Web サイトで公開されている．
- 被験者実験を通じて ClonePacker と Libra を比較した．その結果，ClonePacker は Libra の課題点を解決していることを示した．

以下，本論文は次のように構成されている．2 章では本論文において用いる用語と技術について述べる．3 章および 4 章では提案手法および実装したツールについて説明する．次に 5 章および 6 章では被験者実験について述べる．7 章では実験の妥当性への脅威について述べる．8 章で関連研究を紹介し，最後に 9 章で本論文をまとめる．

2. 準備

本論文において用いる用語と技術について説明する．

2.1 クローンペアとクローンセット

クローンペアとはクローンの組を指す．クローンセットとはクローンの集合を指す．つまり，1 つのクローンセットに含まれる任意の 2 つのクローンはクローンペアである．図 3 にクローンペアとクローンセットの例を示す．図 3 における 3 つのコード片 α，β，γ をクローンとする．このとき，3 つの組 (α，β)，(β，γ)，(α，γ) はそれぞれクローンペアであり，1 つの集合 $\{\alpha, \beta, \gamma\}$ はクローンセットである．

```
1240: int max_stack = codeStream.stackMax;
1241: localContents[codeAttributeOffset + 6] = (byte) (max_stack >> 8);
1242: localContents[codeAttributeOffset + 7] = (byte) max_stack;
1243: int max_locals = codeStream.maxLocals;
1244: localContents[codeAttributeOffset + 8] = (byte) (max_locals >> 8);
1245: localContents[codeAttributeOffset + 9] = (byte) max_locals;
```

```
1815: int max_stack = codeStream.stackMax;
1816: contents[codeAttributeOffset + 6] = (byte) (max_stack >> 8);
1817: contents[codeAttributeOffset + 7] = (byte) max_stack;
1818: int max_locals = codeStream.maxLocals;
1819: contents[codeAttributeOffset + 8] = (byte) (max_locals >> 8);
1820: contents[codeAttributeOffset + 9] = (byte) max_locals;
```

(a) コード片単位のクローン

```
401: final public void bastore() {
402:   countLabels = 0;
403:   stackDepth -= 3;
404:   try {
405:     position++;
406:     bCodeStream[classFileOffset++] = OPC_bastore;
407:   } catch (IndexOutOfBoundsException e) {
408:     resizeByteArray(OPC_bastore); } }
```

```
3590: final public void lrem() {
3591:   countLabels = 0;
3592:   stackDepth -= 2;
3593:   try {
3594:     position++;
3595:     bCodeStream[classFileOffset++] = OPC_lrem;
3596:   } catch (IndexOutOfBoundsException e) {
3597:     resizeByteArray(OPC_lrem); } }
```

(b) メソッド単位のクローン

図 4 コード片単位およびメソッド単位のクローン
Fig. 4 Statement-based and Method-based Clones

2.2 クローンのタイプ

Bellon らはクローンを以下に示す 3 つのタイプに分類している [12].

タイプ 1 クローン: 空白，タブ，改行の有無を除いて完全に一致するクローン

タイプ 2 クローン: 変数名，関数名等のユーザ定義名，また変数の型等の一部の予約語のみが異なるクローン

タイプ 3 クローン: タイプ 2 クローンにおける違いに加えて文の挿入，削除，変更が行われ，いくつかのギャップを含むクローン

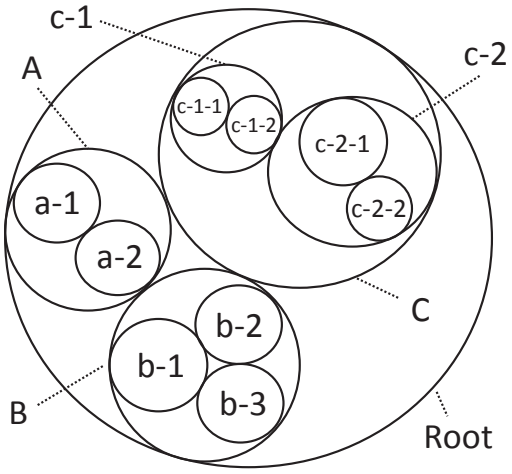
本論文も Bellon らのクローンの定義を用いる．

2.3 コード片単位およびメソッド単位のクローン

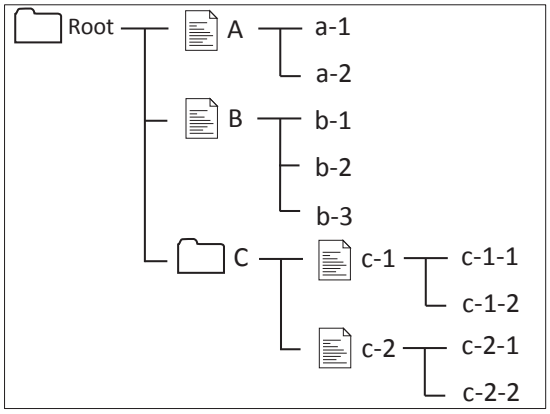
2 つのクローンが共に文の列で構成されているとき，それらをコード片単位のクローンと呼ぶ．また 2 つのクローンが共に 1 つのメソッドで構成されているとき，それらをメソッド単位のクローンと呼ぶ．図 4(a) および (b) は Bellon らのクローンのデータセット [12] に含まれるコード片単位およびメソッド単位のクローンの例である．図 4(a) の 2 つのクローンは共に変数宣言文および代入文で構成されている．図 4(b) の 2 つのクローンは共に 1 つのメソッドで構成されている．本研究ではメソッド単位のクローンの検出および可視化を行う．

2.4 Circle Packing

Circle Packing [10] とは円の中に複数の円を描画することで，階層的なデータ構造を表現する手法である．図 5(a) に Circle Packing の例を示す．図 5(a) において一番外側の円は A，B，C の 3 つの大きな円を含んでいる．さらにそれらの円はいくつかの円を含んでいる．例えば，円 A は円 a-1 および円 a-2 を含んでいる．また円 C は円 c-1 および円 c-2 を含んでおり，さらに円 c-1 は円 c-1-1 および円 c-1-2 を含んでいる．Circle Packing を用いることで，階層的なデータ構造が表現できる．例えば図 5(a) において円 a-1 と円 a-2 は共に円 A というカテゴリーに存在するが，円 a-1 と円 b-1 は異なるカテゴリーに存在する．さらに円 c-1-1 と円 c-2-1 は共に円 C というカテゴリーに存在するが，それらは異なるサブカテゴリーに存在する．



(a) Circle Packing の例



(b) (a) に対するファイルの階層構造表現

図 5 Circle Packing とファイルの階層構造表現
Fig. 5 Circle Packing and Its File Hierarchy

本研究では Circle Packing を用いてファイルの階層構造を表現する (図 5(b))．本研究ではファイルの階層構造を表現するものとしてメソッド，ファイル，ディレクトリの 3 つを用いる．つまり一番内側にある円をメソッド，メソッドを含む円をファイル，ファイルを含む円をデイレクトリとみなす．例えば，図 5(a) において一番外側にある円はディレクトリである．このディレクトリは 2 つのファイル（A，B）と 1 つのディレクトリ（C）を含んでいる．ファイル A は 2 つのメソッド（a-1 および a-2）を含んでいる．

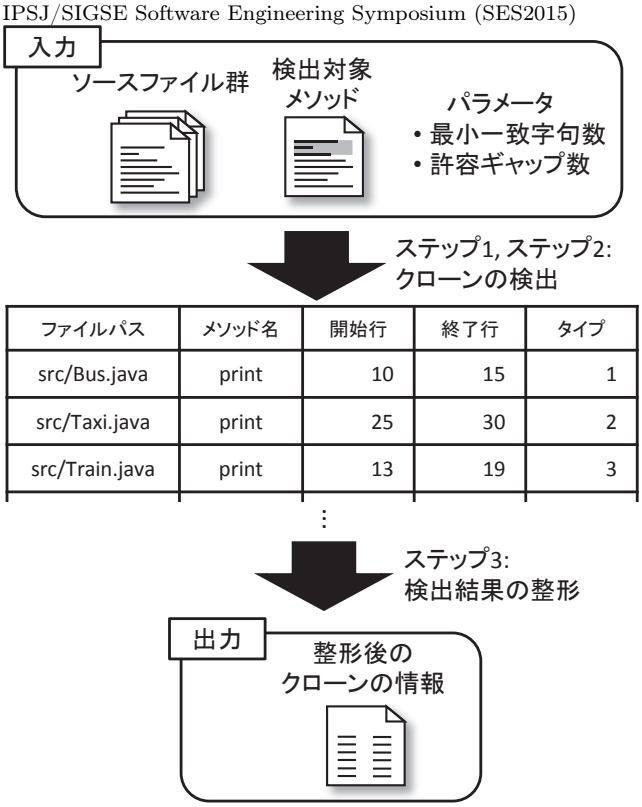


図 6 提案手法の概要
Fig. 6 Overview of Proposed Technique

ディレクトリ C は 2 つのファイル（c-1，c-2）を含み，ファイル c-1 は 2 つのメソッド（c-1-1，c-1-2）を含んでいる．

3. 提案手法

図 6 に提案手法の概要を示す．提案手法は以下に示す 3 つのステップから成る．

ステップ 1: タイプ 1 およびタイプ 2 クローンの検出
ステップ 2: タイプ 3 クローンの検出
ステップ 3: 検出結果の整形

提案手法の入力は以下の通りである．

- クローンセットの検出対象となるソースファイル群（以降，**ソースファイル群**と表記する）
- 変更を加える対象となるメソッド（以降，**検出対象メソッド**と表記する）
- クローンセットの検出におけるパラメータである最小一致字句数および許容ギャップ数（以降，**パラメータ**と表記する）

提案手法は与えられた入力を基に，検出対象メソッドとクローンの関係にある全てのメソッドを 1 つのクローンセットとして検出する．出力は，検出されたクローンセットに含まれる全てのクローンの情報（どのメソッドがクローンであるか，クローンであるならばどのタイプのクローンであるか）である．

以降，図 7 を用いて提案手法の各ステップを説明する．

3.1 ステップ 1: タイプ 1 およびタイプ 2 クローンの検出

タイプ 1 およびタイプ 2 クローンの検出は以下のステップで行われる．

ステップ 1-1: 字句解析および正規化
ステップ 1-2: 文の識別
ステップ 1-3: クローンの識別

各ステップについて説明する．

ステップ 1-1: 字句解析および正規化

まず入力されたソースファイル群に含まれるメソッドを抽出する．次に抽出したメソッドを字句に切り分ける (図 7(a), (b))．この際，タイプ 2 クローンを検出するためユーザ定義名は特殊文字に正規化される (図 7(c))．元のユーザ定義名はリスト（以降，**ユーザ定義名リスト**と表記する）に保持される．

ステップ 1-2: 文の識別

正規化後の字句列に対して文の識別を行う (図 7(d))．本研究では既存研究 [11] と同じく，セミコロン（“;”）もしくは中括弧（“{”，“}”）で区切られた字句列を文と定義する．また，文に含まれる字句の数も保持する．

ステップ 1-3: クローンの識別

ステップ 1-2 で得られた全ての文に対してハッシュを生成する．さらにユーザ定義名リストから 1 つのハッシュ（以降，**ユーザ定義名ハッシュ**と表記する）を生成する (図 7(e))．ここまでのステップで 1 つのメソッドから文のハッシュ列，各文に含まれる字句数，ユーザ定義名ハッシュが得られる．ここで，ある 2 つのメソッドについて文のハッシュ列およびユーザ定義名ハッシュが共に等しければ，その 2 つのメソッドをタイプ 1 クローンとみなす．文のハッシュ列は等しいがユーザ定義名ハッシュが異なるときは，その 2 つのメソッドをタイプ 2 クローンとみなす．文のハッシュ列が異なる場合はステップ 2 に進み，タイプ 3 クローンであるか否かを判定する (図 7(f))．

3.2 ステップ 2: タイプ 3 クローンの検出

2 つの文のハッシュ列に対して Smith-Watrman アルゴリズム [13] を応用して類似しているか否かを判定する (図 7(g))．Smith-Waterman アルゴリズムとは，2 つの配列の中から類似する部分配列のペアを 1 つ検出するアルゴリズムである．このアルゴリズムは類似する部分配列の中にいくつかのギャップが含まれていても，それを検出できるという特徴をもつ．Smith-Waterman アルゴリズムのステップを以下に示す．

ステップ 2-1: 表の作成および初期化
ステップ 2-2: セルの値の計算
ステップ 2-3: トレースバック
ステップ 2-4: 類似配列の識別

各ステップについて説明する．

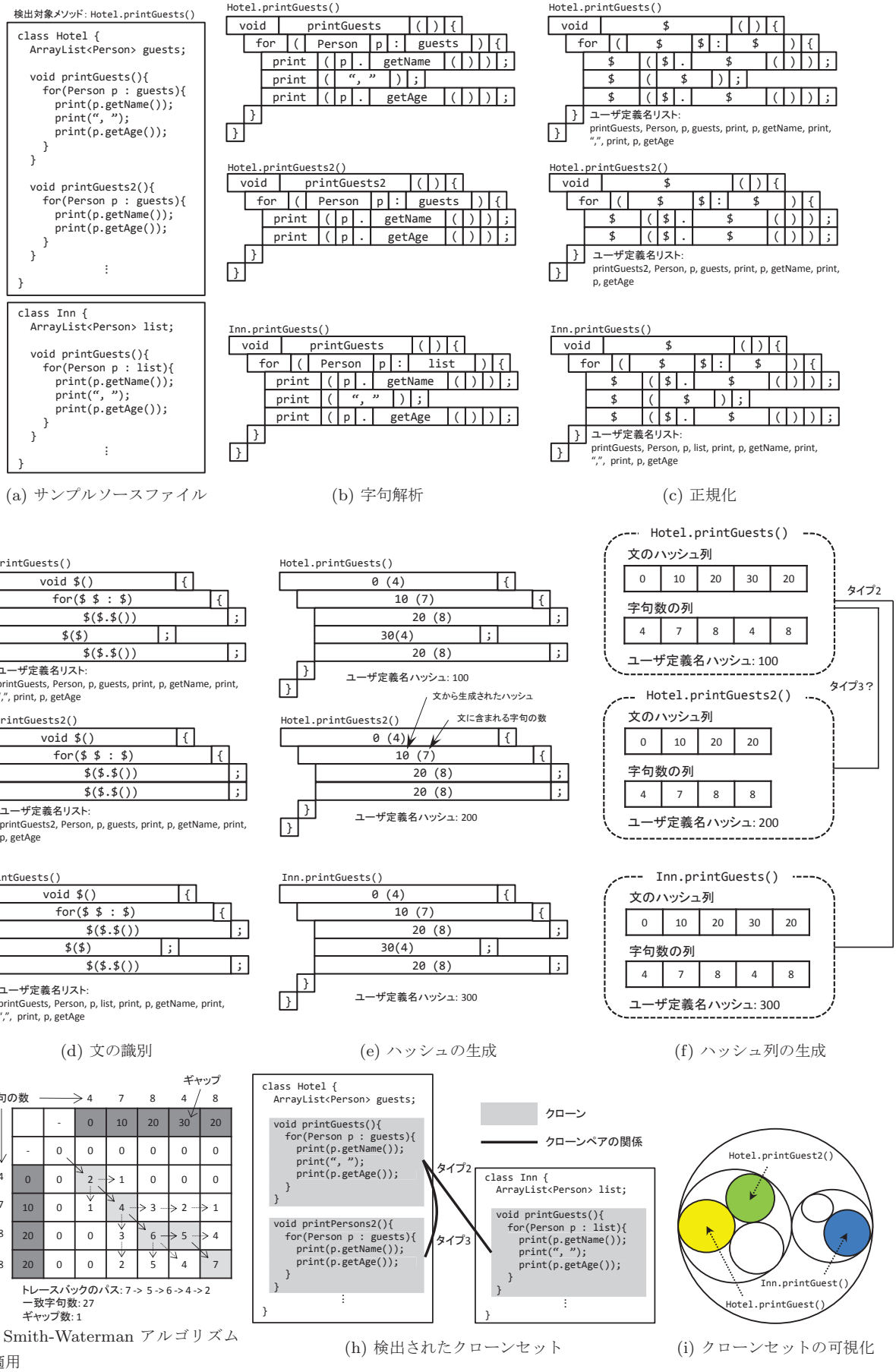


図 7 提案手法のステップ

Fig. 7 Steps of Proposed Technique

ステップ 2-1: 表の作成および初期化

入力された 2 つの配列が $\langle a_0, a_1, \cdots, a_{N-1}\rangle$ と $\langle b_0, b_1, \cdots, b_{M-1}\rangle$ のとき，$(N + 2) \times (M + 2)$ の表を作る．次に表の一番上の行と一番左の列を入力された配列で埋める．さらに 2 番目の行と列を 0 で埋める．

ステップ 2-2: セルの値の計算

以下の数式にしたがって残りのセルの値を計算する．

$$v_{i,j}(2 \leq i, 2 \leq j) = max \begin{cases} v_{i-1,j-1} + s(v_{0,i}, v_{j,0}), \\ v_{i-1,j} + gap, \\ v_{i,j-1} + gap, \\ 0. \end{cases} \quad (1)$$

$$s(v_{0,i}, v_{j,0}) = \begin{cases} match & (v_{0,i} = v_{j,0}), \\ mismatch & (v_{0,i} \neq v_{j,0}). \end{cases} \quad (2)$$

ここで $v_{i,j}$ は i 行 j 列に位置するセルの値を表す．また $match$，$mismatch$，gap は任意の整数値を表す．一般的に ($match$, $mismatch$, gap) の値としてそれぞれ (2, -2, -1) が用いられており，本研究においても同じ値を用いる．

$c_{i,j}$ を i 行 j 列に位置するセルとする．各セルの値を計算している間，$v_{i,j}$ を求めるために使用されたセルから $c_{i,j}$ へのポインタが作られる．例えば図 7(g) において，$v_{4,4}(= 6)$ は $v_{3,3}(= 4)$ と $s(v_{0,4}, v_{4,0})(= 2)$ の和である．この場合，$c_{3,3}$ から $c_{4,4}$ へのポインタが作られる．

ステップ 2-3: トレースバック

トレースバックとはステップ 2-2 で作ったポインタを逆にたどる操作を意味する．トレースバックは表中の最大の値をもつセルから始まり，0 の値をもつセルの前のセルで終わる．図 7(g) において，薄くハイライトされたセルはトレースバックのパスを表している．

ステップ 2-4: 類似配列の識別

トレースバックのパスが指している要素が類似配列として識別される．図 7(g) において，トレースバックのパスは $\langle 0, 10, 20, 30, 20\rangle$ と $\langle 0, 10, 20, 20\rangle$ を指している．したがって，これらの配列が類似配列として識別され，類似配列の基になった文の列が類似しているとみなされる．各文に含まれる字句数は保持されているため，文の列に含まれる字句数の和を求めることができる．字句数の和は，$\langle 0, 10, 20, 30, 20\rangle$ については $31(= 4 + 7 + 8 + 4 + 8)$，$\langle 0, 10, 20, 20\rangle$ については $27(= 4 + 7 + 8 + 8)$ である．このうち小さい方の値を一致字句数とする．さらに図 7(g) において，$v_{5,4}(= 5)$ は $v_{4,4}(= 6)$ から求められている．これは，式 (1) における上から 2 番目の式 $v_{i,j} = v_{i-1,j} + gap$ の結果である．式 (1) において上から 2 番目の式で $v_{i,j}$ が求められた場合，$v_{i,0}$ はギャップとみなされる．また式 (1) において上から 3 番目の式で $v_{i,j}$ が求められた場合，$v_{0,j}$ はギャップとみなされる．つまり $\langle 0, 10, 20, 30, 20\rangle$ における “30” はギャップとみなされる．以上より，2 つの類似配列 $\langle 0, 10, 20, 30, 20\rangle$ および $\langle 0, 10, 20, 20\rangle$ について一致字句数は 27，ギャップ数は 1 であることが分かる．一致字句数が最小一致字句数以上でありかつギャップ数が許容ギャップ数以下であるならば，ステップ 2 の入力であった 2 つのハッシュ列の基になった 2 つのメソッドをタイプ 3 クローンとみなす．

3.3 ステップ 3: 検出結果の整形

ステップ 1 およびステップ 2 で得られたクローンセットを可視化するために検出結果を整形する (図 7(h))．クローンセットにおける各クローンを含むファイルのパスが分かるため，各ファイルの階層構造を求めることができる．階層構造の情報およびクローンの情報（どのメソッドがクローンであるか，クローンであるならばどのタイプのクローンであるか）を基に，Circle Packing を描画するためのファイルを出力する (図 7(i))．

4. クローンセット可視化ツール

本章では提案手法の実装，および実装したツールの使い方について述べる．

4.1 実装

提案手法を Eclipse プラグインとして実装し，それを ClonePacker と名付けた．ClonePacker は著者らの Web サイトで公開されている [*2]．Circle Packing の描画には，JavaScript ライブラリである D3[*3] を用いた．なお D3 を用いて可視化を行うためには，可視化を行う対象となるデータが特定の形式で記述されたファイルが必要である．本実装では，検出されたクローンの情報を基に可視化を行うためのファイルを作成し，それを D3 に与える．その後 D3 は与えられたファイルを用いて，検出結果を表す Circle Packing を描画する．

4.2 使い方

図 8 は ClonePacker のスクリーンショットである．ClonePacker の使用手順は以下の通りである．

(1) 使用者はキャレットをあるメソッドの上に合わせることで，そのメソッドを検出対象メソッドとする．図 8 ではキャレットの位置は 114 行目である．この場合，109 行目〜127 目のメソッド *draw* が検出対象メソッドとして選択される．

(2) 使用者がボタン A を押すことで，ClonePacker は検出対象メソッドを含むプロジェクトのソースファイル群から検出対象メソッドを含むクローンセットを検出する．クローンセットの検出後，ClonePacker は検出結果を表すビュー（ビュー B およびビュー C）を使用者に提示する．

[*2] http://sdl.ist.osaka-u.ac.jp/~h-murakm/clonepacker/
[*3] http://d3js.org/

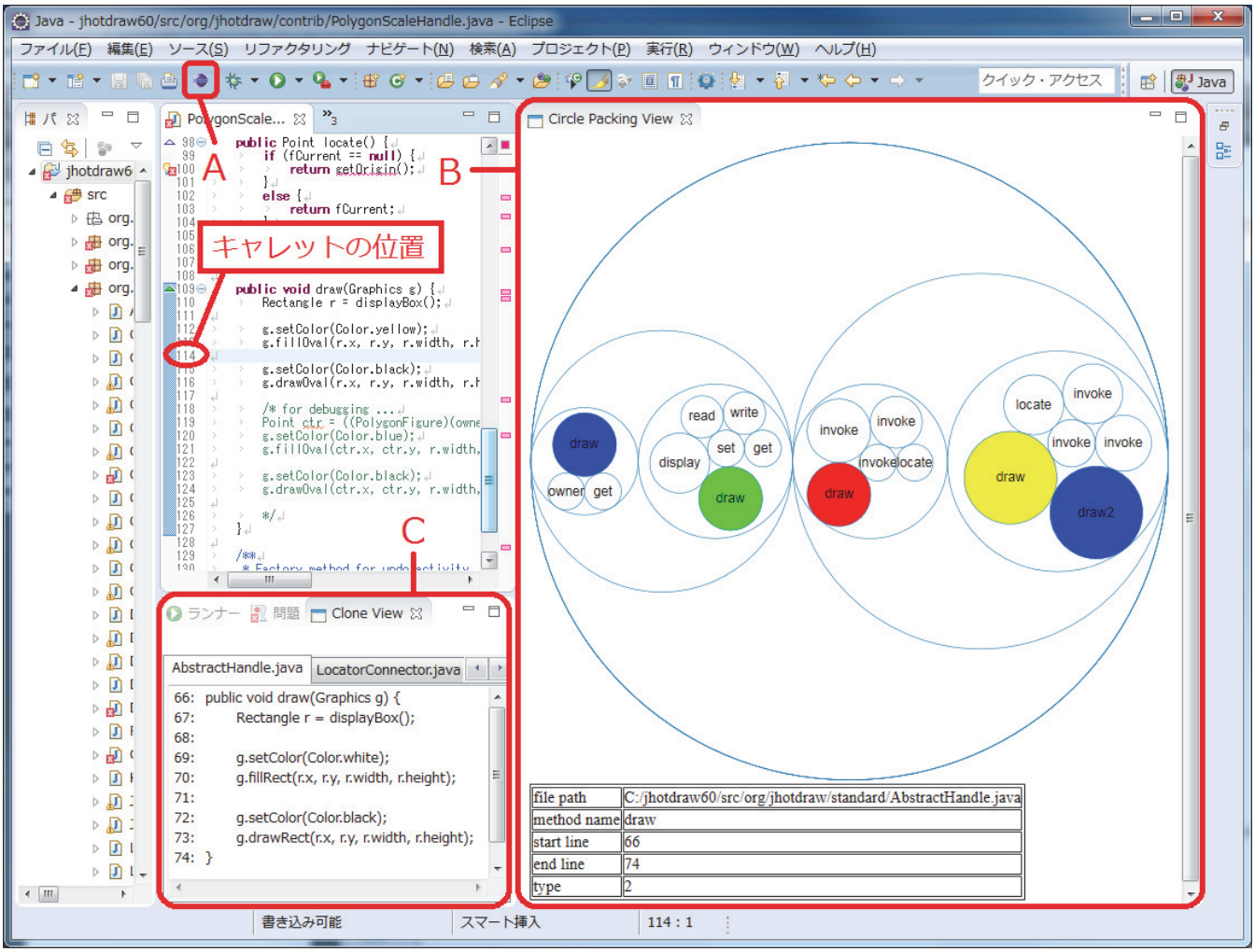


図 8　ClonePacker のスクリーンショット
Fig. 8　Screenshot of ClonePacker

(3) ビュー B およびビュー C を用いて，使用者はクローンの分析を行う．ビュー B には検出結果を示す Circle Packing が描画されており，ビュー C にはクローンのソースコードが表示されている．

ビュー B において黄色い円は検出対象メソッドを表している．赤い円，青い円，緑の円はそれぞれタイプ 1 クローン，タイプ 2 クローン，タイプ 3 クローンを表している．この例ではタイプ 1 クローンが 1 つ，タイプ 2 クローンが 2 つ，タイプ 3 クローンが 1 つ検出されている．さらに，検出されたクローンのうちの 1 つが検出対象メソッドと同じファイルに存在し，残りのうちの 1 つが検出対象メソッドを含むファイルとは異なるファイルであるが同じディレクトリに存在し，残りの 2 つが異なるディレクトリに存在していることが分かる．また円の大きさはそのメソッドの行数を表している．つまり，大きな円は行数の多いメソッドであり，小さな円は行数の少ないメソッドである．円をクリックすることで，そのクローンの情報（ファイルのパス，メソッド名，開始行，終了行，タイプ）がビュー B の下部に表示される．

図 8 に示すように ClonePacker はクローンのタイプを色で，クローンのサイズを円の大きさで表現している．さらにビュー C においてソースファイル全体ではなく，クローンのソースコードのみを表示している．これより ClonePacker は 1 章で示した Libra の課題点を解決しているといえる．また Libra はスタンドアローンのツールであるが，ClonePacker は開発環境に組み込まれている．そのため開発や保守の最中にウィンドウを切り替えることなくクローンを閲覧できるという利点を ClonePacker は有している．

5. クローン分析に要する時間の評価

クローン分析に要する時間について ClonePacker と Libra を比較した．この実験の目的は，ClonePacker が Libra に比べてクローンの分析を効率的に行えるのか否かを調べることである．なお本実験は，メソッドに変更を加える際，他に同様の変更を加えるべきメソッドの位置を把握するためにツールを使う状況を想定している．本節では，その方法と結果について述べる．

表 1 タスクの詳細
Table 1 Details of Tasks

タスク	検出対象メソッド	検出対象メソッドを含むファイルのパス (検出対象メソッドの開始行 - 終了行)	タイプ 1 クローンの数	タイプ 2 クローンの数	タイプ 3 クローンの数
タスク 1	suite	src/org/jhotdraw/test/samples/minimap/MinimapSuite.java (37 - 57)	0	2	0
タスク 2	handles	src/org/jhotdraw/figures/GroupFigure.java (67 - 74)	0	2	1
タスク 3	draw	src/org/jhotdraw/contrib/PolygonScaleHandle.java (111 - 129)	4	3	1
タスク 4	store	src/org/jhotdraw/util/SerializationStorageFormat.java (62 - 68)	0	1	0
タスク 5	fillRoundRect	src/org/jhotdraw/contrib/zoom/ScalingGraphics.java (212 - 217)	0	2	2
タスク 6	handles	src/org/jhotdraw/contrib/TextAreaFigure.java (299 - 303)	5	0	1

5.1 方法

被験者は大阪大学に所属する 8 名の大学院生と 2 名の学部生の計 10 名である．まず，被験者を 2 つのグループ（G_A，G_B）に分割した．次に被験者は ClonePacker もしくは Libra を使用して 6 つのタスクに取り組み，そのタスクを終えるのに要した時間を計測した．タスクは「著者らが指定したメソッドを検出対象メソッドとしたとき，検出された全てのクローンの位置を報告せよ」というシンプルなものである．計測する時間は，被験者が検出対象メソッドをツールに入力した時点から全てのクローンの位置を報告し終わった時点までである．本実験では，オープンソースソフトウェアである JHotDraw[*4] 6.0 beta 1 に含まれるソースファイル群を対象とした．タスクの詳細を表 1 に示す．例えばタスク 2 において，ClonePacker は 3 つのクローン（タイプ 2 クローンを 2 つ，タイプ 3 クローンを 1 つ）を検出する．しかし，Libra は同じタスクに対して 2 つのタイプ 2 クローンのみを検出する．その理由は Libra はクローンの検出に CCFinder を用いており，CCFinder はタイプ 3 クローンを検出することができないためである．また，ClonePacker はクローンのタイプを出力するにも関わらず，実験ではクローンの位置のみを報告するように被験者に指示している．その理由は Libra はクローンのタイプを出力しておらず，クローンのタイプまで報告するように指示するとツール間で不公平が生じるためである．

さらに，公平な実験を行うためタスクの半分を終えた時点で使用するツールを変更した．つまり G_A に属する被験者は ClonePacker を使用してタスク 1，タスク 2，タスク 3 に取り組み，Libra を使用してタスク 4，タスク 5，タスク 6 に取り組んだ．一方 G_B に属する被験者は Libra を使用してタスク 1，タスク 2，タスク 3 に取り組み，ClonePacker を使用してタスク 4，タスク 5，タスク 6 に取り組んだ．

なお，本実験ではクローン検出のパラメータである最小一致字句数を 30，許容ギャップ数を 2 に設定した．

5.2 結果

図 9 にタスクを終えるのに要した時間を示す．なお，この時間はクローンの検出に要した時間と被験者の作業時間の和である．しかし各ツールがクローンの検出に要した時間は非常に短く，ほぼ同じであったため，作業時間の比較には問題はないと著者らは考えている．図 9 において横軸は各タスク，縦軸はそのタスクを終えるのに要した時間をクローンの数で割った値である．また青い箱ひげ図は Libra を使用した被験者，赤い箱ひげ図は ClonePacker を使用した被験者を示している．例えばタスク 1 において，ClonePacker を使用した被験者の中で最も早くタスクを終えた人は，クローン 1 つ当たり 20 秒要したことが分かる．なお全ての被験者は，各タスクにおける全てのクローンの位置を正確に報告したことを著者らは確認している．

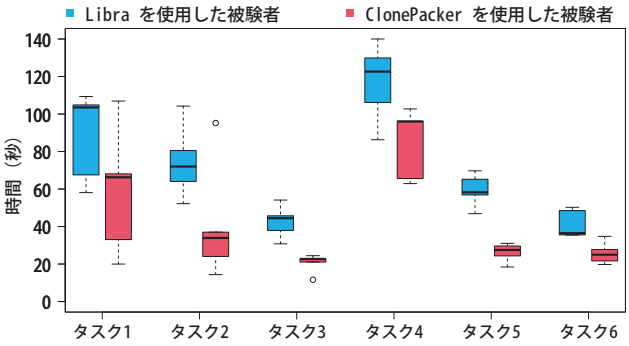


図 9 タスクを終えるのに要した時間
Fig. 9 Results of Task Completion Time

5.3 考察

図 9 から，ClonePacker を使用した被験者は Libra を使用した被験者より概ね早くタスクを終えた，と著者らは判断した．タスクを終えるのに要した時間についてツール間で有意差があるか否かを調べるため，次に示す帰無仮説と対立仮説を立てた．

帰無仮説: ClonePacker を使用した被験者がタスクを終えるのに要した時間は，Libra を使用した被験者がタスクを終えるのに要した時間に比べて短くない．

対立仮説: ClonePacker を使用した被験者がタスクを終えるのに要した時間は，Libra を使用した被験者がタスクを終えるのに要した時間に比べて短い．

まず著者らは ClonePacker と Libra について，タスクを終えるのに要した時間が有意水準 0.05 の基で等分散でありかつ正規分布に従わないことを確認した．したがって

[*4] http://www.jhotdraw.org/

Wilcoxon 検定を行うことにした．検定の結果，得られた p 値は $6.724E-05$ であった．p 値が有意水準 0.05 を下回ったため，帰無仮説を棄却し対立仮説を採択した．検定の結果より，ClonePacker を使用した被験者は Libra を使用した被験者より早くタスクを終えた，と著者らは結論付けた．

6. ユーザビリティの評価

次に ClonePacker と Libra についてユーザビリティの評価を行った．この評価の目的は，ClonePacker が実際に使いやすいツールであるか否かを調べることである．本節では，その方法と結果について述べる．

6.1 方法

評価は System Usability Scale [14] を用いて行った．各被験者は以下に示す 10 個の質問に対して，1（まったくそう思わない），2（そう思う），3（どちらとも思わない），4（そう思う），5（強くそう思う）の 5 段階で回答した．

質問 1: このツールをしばしば使いたいと思うか.
質問 2: このツールは不必要なほど複雑だと感じるか.
質問 3: このツールは容易に使えると思うか.
質問 4: このツールを使うのに専門家のサポートが必要と感じるか.
質問 5: このツールのさまざまな機能が良くまとまっていると感じるか.
質問 6: このツールは一貫性のないところが多くあったと思うか.
質問 7: 大抵のユーザはこのツールの使用方法をすばやく学べると思うか.
質問 8: このツールはとても扱いにくいと思うか.
質問 9: このツールを使うのに自信があると感じるか.
質問 10: このツールを使用するのに多くの予備知識が必要だと感じるか.

各被験者は ClonePacker と Libra について上記の質問に回答した．さらに意見やコメントを集めるため，自由記述欄を設けた．なお本評価は無記名で行った．

また System Usability Scale では，各被験者について以下のステップで評価値を求める [14].

ステップ A: 奇数番号の質問は肯定的な質問，偶数番号の質問は否定的な質問である．そのため，奇数番号の質問は回答から 1 を引いた値へ，偶数番号の質問は 5 から回答を引いた値へ変換することで各回答のスコアが 0〜4 となるように正規化を行う．

ステップ B: 正規化後のスコアの和を 2.5 倍した値を評価値とする．評価値は最小で 0，最大で 100 である．評価値が高いほどユーザビリティの高いツールである．

6.2 結果

実験結果を表 2 に示す．ClonePacker と Libra について，各被験者の質問 1〜10 に対する回答および評価値が示されている．表 2 より，全ての被験者について Libra より ClonePacker の方が高い評価値となった．以上より ClonePacker は Libra より使いやすいツールであるといえる．

6.3 考察

表 2 には ClonePacker と Libra のスコアの差（正規化後の ClonePacker のスコアから正規化後の Libra のスコアを引いた値）が記載されている．表 2 より大半のスコアは，同等もしくは ClonePacker の方が優れていることが分かる．しかし，2 箇所のみスコアの差が負の値であった．スコアの差が負の値であった質問は質問 2 と質問 6 である．これより，ClonePacker は複雑である，また一貫性のないところが多いと感じた被験者がいたということが分かる．具体的には，質問 2 については「複数のビューを見比べるのが面倒である」との意見を頂いた．質問 6 については自由記述欄には何も書かれていなかった．今後，頂いた意見を参考にして ClonePacker の改良に取り組む予定である．

7. 妥当性への脅威

本章では実験の妥当性への脅威について述べる．

7.1 クローンの検出

本実験では最小一致字句数を 30，許容ギャップ数を 2 としてクローンの検出を行った．クローンの検出結果はパラメータに影響される．例えば，最小一致字句数を小さくすると検出されるクローンの数は多くなるが，誤検出が増える．Wang らはクローン検出に対するパラメータを自動的に定める手法を提案している [15]．もし Wang らの手法を用いてパラメータを定めた後に実験を行うと，本実験で得られた結果とは異なる結果を得る可能性がある．

7.2 対象ソフトウェア

本実験では，1 つのソフトウェア（JHotDraw）を実験対象とした．そのため JHotDraw とは異なるソフトウェアを対象として実験を行った場合，本実験で得られた結果とは異なる結果を得る可能性がある．それを確かめるためには，ClonePacker をより多くのソフトウェアに適用させる必要がある．

またメソッドを多く含むソフトウェアに対して ClonePacker を適用すると，クローンの検出結果として多くの円が描画される．その場合 1 つ 1 つの円が小さくなり，クローンの分析が行いづらくなる．メソッドを多く含むソフトウェアについては，クローンの粒度をファイル単位もしくはディレクトリ単位に変えることでその欠点を補うことができると著者らは考えている．

表 2　System Usability Scale の結果
Table 2　Results of System Usability Scale

被験者	ツール	質問 1	質問 2	質問 3	質問 4	質問 5	質問 6	質問 7	質問 8	質問 9	質問 10	評価値
A	ClonePacker	3	1	4	2	4	2	4	2	4	3	**72.5**
	Libra	3	4	2	2	3	2	2	4	3	4	42.5
	スコアの差	0	3	2	0	1	0	2	2	1	1	30.0
B	ClonePacker	4	2	4	2	4	2	4	2	4	2	**75.0**
	Libra	3	2	2	2	3	2	4	4	3	2	57.5
	スコアの差	1	0	2	0	1	0	0	2	1	0	17.5
C	ClonePacker	4	2	4	4	4	2	5	2	4	4	**67.5**
	Libra	4	3	2	4	2	2	2	4	3	4	40.0
	スコアの差	0	1	2	0	2	0	3	2	1	0	27.5
D	ClonePacker	4	1	4	3	3	2	4	1	3	3	**70.0**
	Libra	2	4	1	4	3	3	2	3	2	3	32.5
	スコアの差	2	3	3	1	0	1	2	2	1	0	37.5
E	ClonePacker	3	2	4	3	3	2	4	2	3	2	**65.0**
	Libra	3	2	3	4	3	2	3	2	3	3	55.0
	スコアの差	0	0	1	1	0	0	1	0	0	1	10.0
F	ClonePacker	3	2	4	2	4	3	4	2	4	4	**65.0**
	Libra	2	3	2	3	3	2	3	3	2	4	42.5
	スコアの差	1	1	2	1	1	-1	1	1	2	0	22.5
G	ClonePacker	4	4	4	2	2	2	4	4	2	2	**55.0**
	Libra	1	2	4	4	1	2	4	5	1	2	40.0
	スコアの差	3	-2	0	2	1	0	0	1	1	0	15.0
H	ClonePacker	4	1	4	2	4	1	5	2	4	2	**82.5**
	Libra	2	3	1	4	2	2	1	5	2	4	25.0
	スコアの差	2	2	3	2	2	1	4	3	2	2	57.5
I	ClonePacker	4	1	5	2	4	1	4	2	3	3	**77.5**
	Libra	2	3	3	4	2	2	3	4	3	4	40.0
	スコアの差	2	2	2	2	2	1	1	2	0	1	37.5
J	ClonePacker	4	1	4	1	4	2	4	1	4	4	**77.5**
	Libra	2	1	4	2	3	2	4	3	4	4	62.5
	スコアの差	2	0	0	1	1	0	0	2	0	0	15.0

7.3　被験者

本実験では被験者は 2 つのグループに分かれ, ClonePacker および Libra を用いて 6 つのタスクに取り組んだ．もしグループ間でクローンの知識の差が大きいと，タスクを終えるのに要した時間に影響を与える可能性がある．しかし本実験では，被験者を 2 つのグループに分ける際，クローンの知識が可能な限り等しくなるように考慮した．

7.4　System Usability Scale

本実験では System Usability Scale を用いて, ClonePacker および Libra のユーザビリティを評価した．System Usability Scale におけるアンケート項目は本来英語で記述されているが，本実験では和訳したアンケート項目を用いた．そのため英語の細かいニュアンスを正しく伝えられていない可能性がある．しかし英語の細かいニュアンスの違いにより結果が大きく変わるようなアンケート項目は無いと著者らは考えている．

8.　関連研究

植田らは Gemini というクローン分析ツールを開発している [16]．Gemini は CCFinder [4] が出力したクローンの情報を散布図およびメトリクスグラフを用いて可視化する．メトリクスグラフとは検出されたクローンの特徴を表すグラフであり，分析するクローンのフィルタリングが可能である．

Asaduzzaman らは VisCad というクローン分析ツールを開発している [17]．VisCad の入力は対象プロジェクトのソースファイル群および特定のクローン検出ツール（NiCad [5], CCFinder [4] 等）をそのソースファイル群に適用させて得られたクローンの情報である．VisCad に入力を与えた後，使用者は出力された散布図，ツリーマップ，階層依存グラフを用いてクローンの分析を行うことができる．

Rieger らは Polymetric View [18] を用いたクローン可視化手法を提案している [19]．Polymetric View とは各

ノードが四角形で表されたグラフである．彼らの手法は，Polymetric View におけるノードの位置，高さ，幅，色等でクローンの情報を表している．彼らの手法を用いることで，クローンの数やどのくらいのディレクトリにクローンが広がっているか等の情報を瞬時に把握できる．

Hauptmann らは Edge Bundle を用いたクローン可視化手法を提案している [20]．彼らの手法の特徴はクローンの検出結果をファイルの階層構造に紐づけていることである．著者らの手法と彼らの手法は共にファイルの階層構造を可視化するという共通点がある．

これらの既存手法 [16], [17], [19], [20] と提案手法の最大の違いは，その手法が使われる状況である．これらの既存手法は，対象プロジェクトに含まれる全てのクローンを分析することを目的としているのに対し，提案手法は与えられたメソッドのクローンのみを分析することを目的としている．

Forbes らは Doppel-Code というクローン分析ツールを開発している [9]．Doppel-Code はソースファイル群とコード片をユーザから受け取り，そのソースファイル群から入力されたコード片のクローンを検出し，検出されたクローンに優先順位 [21] をつけてユーザに提示する．Doppel-Code はクローンのソースコードのみを提示しているのに対し，ClonePacker はクローンのソースコードに加えてクローンの階層構造やタイプ等を提示している点が異なる．また Doppel-Code は公開されていないため，本実験で比較対象とすることができなかった．

9. おわりに

本論文では，Circle Packing を利用したクローンセット可視化手法を提案した．さらに提案手法をツール ClonePacker として実装した．ClonePacker の有用性を調べるため，既存ツールである Libra との比較実験を行った．実験ではコードクローン分析に要する時間，およびユーザビリティの 2 つを比較した．その結果，双方について ClonePacker は Libra より優れていることを確認した．

今後は，ユーザビリティの評価においていただいた意見を参考にして，ClonePacker の改良に取り組む予定である．また規模の大きなソフトウェアに対しても，ClonePacker を用いることで効率的なコードクローン分析を行うことができるのかを調査する予定である．

謝辞 本研究は，日本学術振興会科学研究費補助金基盤研究 (S)(課題番号：25220003) および文部科学省科学研究費補助金若手研究 (A)(課題番号：24680002) の助成を得た．

参考文献

[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. 91-D, No. 6, pp. 1465–1481 (2008).
[2] 神谷年洋，肥後芳樹，吉田則裕：コードクローン検出技術の展開，コンピュータソフトウェア，Vol. 28, No. 3, pp. 28–42 (2011).
[3] Wang, X., Dang, Y., Zhang, L., Zhang, D. and E. Lan, H. M.: Can I Clone This Piece of Code Here?, *ASE*, pp. 170–179 (2012).
[4] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multi-linguistic Token-Based Code Clone Detection System for Large Scale Source Code, *IEEE TSE*, Vol. 28, No. 7, pp. 654–670 (2002).
[5] Roy, C. K. and Cordy, J. R.: NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization, *ICPC*, pp. 172–181 (2008).
[6] Jiang, L., Misherghi, G., Su, Z. and Glondu, S.: Deckard : Scalable and Accurate Tree-Based Detection of Code Clones, *ICSE*, pp. 96–105 (2007).
[7] Higo, Y., Ueda, Y., Kusumoto, S. and Inoue, K.: Simultaneous Modification Support based on Code Clone Analysis, *APSEC*, pp. 262–269 (2007).
[8] Murakami, H., Higo, Y. and Kusumoto, S.: ClonePacker: A Tool for Clone Set Visualization, *SANER*, pp. 474–478 (2015).
[9] Forbes, C., Keivanloo, I. and Rilling, J.: Doppel-Code: A Clone Visualization Tool for Prioritizing Global and Local Clone Impacts, *COMPSAC*, pp. 366–367 (2012).
[10] Wang, W., Wang, H., Dai, G. and Wang, H.: Visualization of Large Hierarchical Data by Circle Packing, *CHI*, pp. 517–520 (2006).
[11] 村上寛明，堀田圭佑，肥後芳樹，井垣宏，楠本真二：Smith-Waterman アルゴリズムを利用したギャップを含むコードクローン検出，情報処理学会論文誌，Vol. 55, No. 2, pp. 981–993 (2014).
[12] Bellon, S., Koschke, R., Antniol, G., Krinke, J. and Merlo, E.: Comparison and Evaluation of Clone Detection Tools, *IEEE TSE*, Vol. 33, No. 9, pp. 577–591 (2007).
[13] Smith, T. F. and Waterman, M. S.: Identification of Common Molecular Subsequences, *Journal of Molecular Biology*, Vol. 147, No. 1, pp. 195–197 (1981).
[14] Lewis, J. R. and Sauro, J.: The Factor Structure of the System Usability Scale, *HCD*, pp. 94–103 (2009).
[15] Wang, T., Harman, M., Jia, Y. and Krinke, J.: Searching for Better Configurations: A Rigorous Approach to Clone Evaluation, *FSE*, pp. 455–465 (2013).
[16] Ueda, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: Gemini: Maintenance Support Environment Based on Code Clone Analysis, *METRICS*, pp. 67–76 (2002).
[17] Asaduzzaman, M., Roy, C. K. and Schneider, K. A.: VisCad: Flexible Code Clone Analysis Support for NiCad, *IWSC*, pp. 77–78 (2011).
[18] Lanza, M. and Ducasse, S.: Polymetric Views - A Lightweight Visual Approach to Reverse Engineering, *IEEE TSE*, Vol. 29, No. 9, pp. 782–795 (2003).
[19] Rieger, M., Ducasse, S. and Lanza, M.: Insights into System-Wide Code Duplication, *WCRE*, pp. 100–109 (2004).
[20] Hauptmann, B., Bauer, V. and Junker, M.: Using Edge Bundle Views for Clone Visualization, *IWSC*, pp. 86–87 (2012).
[21] Guo, P. J., Zimmermann, T., Nagappan, N. and Murphy, B.: Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows, *ICSE*, pp. 495–504 (2010).