論　文

コードクローン検索による類似不具合検出の実証的評価

森崎　修司[†]　吉田　則裕[††]　肥後　芳樹[††]　楠本　真二[††]
井上　克郎[††]　佐々木健介[†††]　村上　浩二[†††]　松井　恭[†††]

Empirical Evaluation of Similar Defect Detection by Code Clone Search

Shuji MORISAKI[†], Norihiro YOSHIDA[††], Yoshiki HIGO[††], Shinji KUSUMOTO[††], Katsuro INOUE[††], Kensuke SASAKI[†††], Koji MURAKAMI[†††], and Kyo MATSUI[†††]

あらまし　不具合修正時の修正前ソースコード片を検索キーとしたコードクローン検索による類似不具合の検出を実証的に評価した．これまでにオープンソースソフトウェアを対象とした研究により，コードクローン検索による類似不具合発見の有用性が確認されている．そこで本論文では，商用開発のソフトウェアを対象としコードクローン検索による類似不具合発見を商用開発の現場への適用する際の指針となることを目指す．対象はパナソニック MSE 株式会社において三つの異なるプロジェクトで開発された 3 件のソースコードであり，試験工程での不具合修正に伴う修正履歴が記録されたリリース済のものである．修正履歴に記録された不具合修正前のソースコード片を検索キーとしコードクローン検索を実施し類似不具合を検出した．その結果，対象とした商用開発においてもその有効性を確認できた．

キーワード　コードクローン分析，類似不具合，ソフトウェア保守，ソースコード解析

1. まえがき

ソフトウェアの高機能化，大規模化に伴い，既存のソフトウェアの流用による開発や長期にわたって保守されるソフトウェアが増えている．このようなソフトウェア開発で作成されたソースコード中には，コードクローンが多く含まれることが報告されている [2], [14]．コードクローンとは，ソースコード中に含まれる同一または類似したソースコード片のことである [3]．例えば，X Window system を構成するソースコード 71 万行のうち，19%がコードクローンであることが報告されている [2]．

不具合除去のために修正したソースコードとコードクローンとなっている部分には類似の不具合が存在する可能性がある．Li らは文献 [15] において，CP-Miner と呼ぶコードクローン検出ツールを利用してコードクローンに起因する未発見の類似不具合を発見している．対象ソフトウェアは Linux と FreeBSD であり，Linux において 421 件，FreeBSD において 443 件の不具合の候補を抽出し，それぞれ 49 件，31 件の不具合を発見している．また，泉田らは文献 [12] において，オープンソースの仮名漢字変換ソフトウェア “かんな” [1] のバージョン 3.6 と 3.6p1 の間で修正されたバッファオーバフロー対策のコードを対象とし，コードクローン検索により，類似のバッファオーバフロー箇所を発見し，UNIX コマンド grep によるキーワード検索との比較をしながら，その有効性を確認している．

Li らは，文献 [15] においてコードクローン発生の主原因として二つの理由を挙げている．一つは，抽象化（共通関数化）するよりもソースコードの一部をコピーする方がソースコード記述にかかる時間が小さくて済むこと，もう一つは抽象化による実行時のオーバヘッド（関数呼出しに伴う処理）が問題になることである．他の原因としてコーディングスタイルや偶発的なものを挙げている．商用のソフトウェア開発では，これらの原因に加え，抽象化によるソースコード変更が引き起こすデグレードによるリスクにより，保守性

[†] 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan
[††] 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan
[†††] パナソニック MSE 株式会社，横浜市
Panasonic MSE, Inc., Tsuzuki-ku, Yokohama-shi, 224–8539 Japan

よりも信頼性を優先し，コードクローンを除去していない場合がある．したがって，コードクローン分析による類似不具合の発見がより有用であると期待されるが，具体的な評価によって明らかにはされていない．

本論文では，コードクローン検索を用いた類似不具合検出を商用開発へ適用する際の指針となることを目的として，オープンソースソフトウェアの類似不具合の発見に役立つことが確認されているコードクローン検索が商用のソフトウェアに対しても役立つかを確認する．具体的には商用ソフトウェアを対象として，不具合修正の履歴を用いて適合率と再現率を求める．対象とした商用ソフトウェアはパナソニック MSE 株式会社で開発され，既にリリースが完了した 3 種類のソフトウェアである．対象ソースコードには，結合テスト，システムテスト以降の不具合除去時のソースコード修正履歴が記録されている．修正履歴をもとに，修正前のソースコード片を検索キーとしコードクローン検索を実施し，検索結果に検索キー以外の修正前ソースコード片が含まれるかを調べ，適合率，再現率，F1 値を算出した．また，検索キーとなるソースコード片の考察と作業者から得られた意見を述べる．

以降，**2.** でコードクローンとコードクローン検索ツール Libra について述べる．**3.** で評価の概要と結果について述べ，**4.** で考察する．**5.** で関連研究について述べ，**6.** でまとめと今後の課題について述べる．

2. 準　　備

2.1 コードクローン

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は類似しているものをコードクローンと呼ぶ．部分列はコードクローン検出手法により異なるがトークン列や文字列などである．また，二つ以上の一致若しくは類似している部分列の集合をクローンセットと呼ぶ．クローンセットの定義はコードクローン検出手法により異なる．本論文では，文献 [14] での定義を用いる．文献 [14] での検出手順は，字句解析，表記の正規化，変数名等の正規化，検出に大別され，以下のとおりである．

（1） 対象ソースコードの字句解析

対象ソースコードからトークンを抽出する．コメント，改行/タブ/空白等の字面上の違いがあってもコードクローンとして検出できるようになる．

（2） 変換

変換ルールにより表記方法を正規化する．ルールには `if(A)B;` を `if(A){B;}`に変換する等が含まれる．

（3） 名前の正規化

変数名，定数名を正規化する．変数，定数名が異なっていてもコードクローンとして検出できるようになる．これにより，`a = b;` と `c = d;` がクローンセットとして検出できるようになる．

（4） クローンセットの検出

指定された長さ以上一致するトークンをすべてクローンセットとして検出する．

図 1 はコードクローンの例を示したものである．図 1 の網掛け部が互いにクローン関係にあることを表しており，網掛け部の集合（点線で囲まれた三つの網掛け部）がクローンセットである．図 1 中右上のソースコードと左下のソースコードは変数名は異なるが，同じ処理を記述したものであり，変数名 `top1` が変数名 `bottom1` に，変数名 `top2` が変数名 `bottom2` になっている点，及び，if 文から始まる行の最後の定数の 1 が 0 になっている点，が異なり，変数名の正規化によりこの差異がなくなり，コードクローンとして検出できる．

2.2 コードクローン検索ツール Libra

不具合修正部分の修正前ソースコードとクローンセットとなっている部分があれば，その部分にも類似の不具合が存在する可能性が高い．修正前ソースコードを検索キーとしクローンセットを検索すれば，類似の不具合の可能性のある部分を効率的にさがすことができる．Libra [16] は検索対象ソースコード，及び，ソースコードの一部（検索キー）を入力とし，検索キーのクローンセットを出力する．

Libra はコードクローン統合分析環境（ICCA [11]）の 1 コンポーネントである．

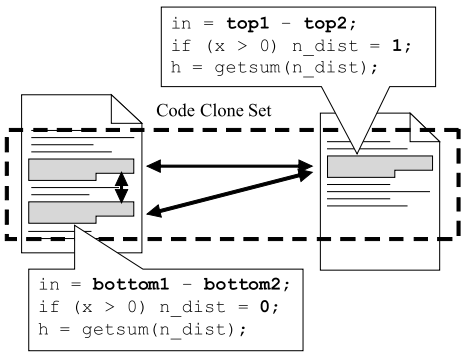

図 1　コードクローンとクローンセット
Fig. 1　Code clone and code clone set.

CCFinder [14] の GUI ラッパーであり，利用者から検索キーとして与えられたソースコード片と検索条件をコードクローン検出エンジンである CCFinder に与え，結果を利用者に提示する．図 2，図 3 は Libra のユーザインタフェースである．図 2 は検索条件を与えるユーザインタフェースであり，中央部の “Fragment” と書かれたテキストエリアに検索キーを指定する．ほかにも，Libra が使用する最大メモリ容量，検索対象のプログラミング言語，検索結果の最小一致トークン数，CCFinder の検索オプションを指定することができる．検索を実行すると，図 3 のように検索結果をユーザに提示する．図 3 の左側は，検索対象のソースコードのディレクトリツリーであり，各ノードはソースコードファイルを表す．ソースコードファイル名の後ろの括弧付きの数値はそのソースコードにいくつのクローンセットが含まれているかを表している．ノードを選択すると右側に検索ソースコード片とクローンセットとなっているソースコードが強調表示される．

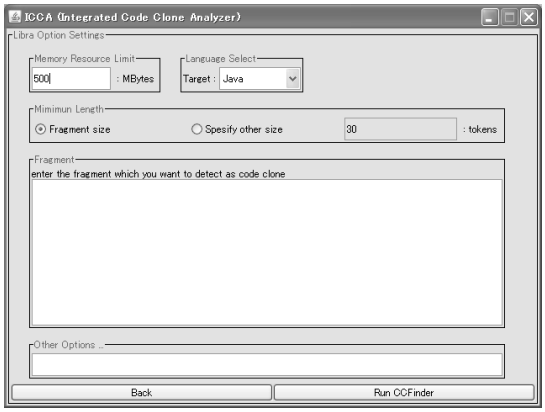

図 2　Libra のユーザインタフェース（検索条件指定画面）
Fig. 2　A screen shot of Libra user interface. (search key specification screen)

図 4 はクローンセットが類似の不具合の原因となるソースコードの例であり，図 4 中 A と B がクローンセットである．図 4 中の A に示されるソースコードと B に示されるソースコードとはともに配列をコピーし，ある条件を満たす場合には，コピー先の配列の末尾に `NULL` を付加する処理を記述したものである．いま，`NULL` を付加する条件の記述が不十分であることがテストにより発見され，図 4 中 A′ の網掛け部分に示すように if 文の条件に `|| (length % 16 == 0)` を追加する必要があることが分かったとする．このとき，図 4 中 B にも同様の修正が必要であるが，B に対してはテストがまだ未実行であるとする．図 4 中 A を検索キーとして，Libra に与えれば図 4 中 B を検索結

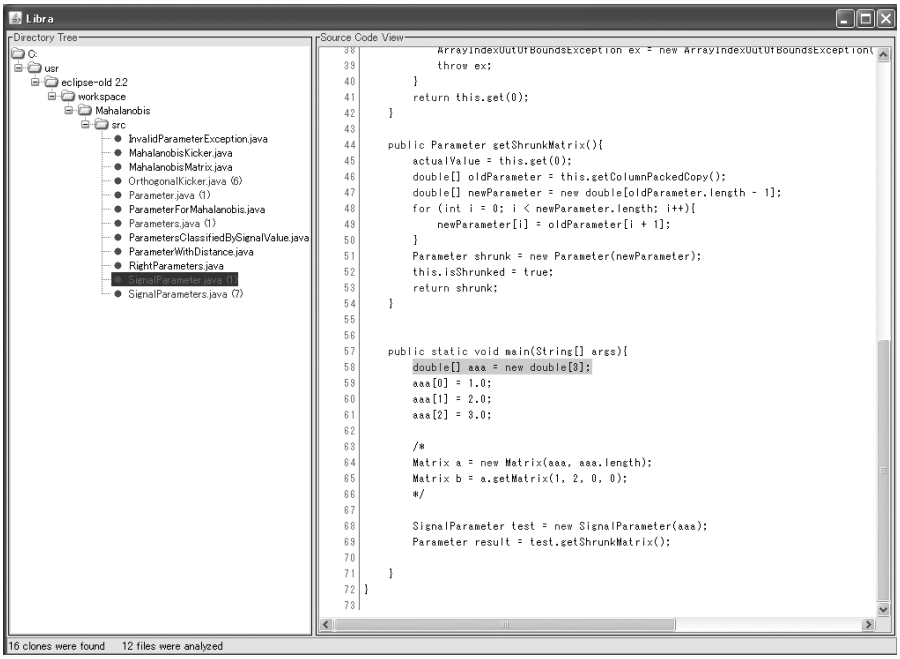

図 3　Libra のユーザインタフェース（検索結果表示画面）
Fig. 3　A screen shot of Libra user interface. (search result screen)

A
```
for (int i = 0; i < length; i++) {
    buffer[i] = io_buffer[i];
}

if (length < MAX_LENGTH) {
    buffer[length] = NULL;
}
```
A'
```
for (int i = 0; i < length; i++) {
     buffer[i] = io_buffer[i];
}

if ( (length < MAX_LENGTH) || (length % 16 == 0) ) {
     buffer[length] = NULL;
}
```
B
```
for (int i = 0; i < mes_length; i++) {
     buf[i] = nw_buffer[i];
}

if (mes_length < MAX_LENGTH) {
     buf[mes_length] = NULL;
}
```
B'
```
for (int i = 0; i < mes_length; i++){
     buf[i] = nw_buffer[i];
}

if ( (mes_length < MAX_LENGTH) || (mes_length % 16 == 0) ) {
   buffer[mes_length] = NULL;
}
```

図 4　同一クローンセットに属する類似不具合の例
Fig. 4　An example of similar bugs included in the same code clone.

果として得ることができる．得られたコードクローンを開発者が検討し，不具合であると判断すれば，B に対するテストを実施する前に類似不具合を修正することができる．

2.3　Libra の有効性評価 [12]

文献 [12] において，オープンソースの仮名漢字変換ソフトウェア “かんな” [1] の不具合修正を対象として Libra による評価を実施している．対象ソースコードの規模は約 21k ステップである．評価の対象とした不具合は 1 件でありバージョン 3.6 と 3.6p1 の間で実施されたバッファオーバフローに関する修正である．修正箇所は全部で 21 箇所あり，21 箇所のうちの 1 箇所を検索コード片として Libra に与え，残り 20 箇所の修正箇所が検索結果に含まれるかを調べている．また，修正箇所に含まれる変数の変数名を UNIX コマンドの grep に与えて得られた検索結果と比較している．

文献 [12] での評価では，適合率と再現率のバランスを示す F1 値の比較において grep よりも Libra の方がよい結果が得られている．具体的には，grep を用いた検索の適合率が 34%，Libra を用いた検索の適合率が 100%，再現率が grep では 95%，Libra では 81%となっている．

文献 [12] での評価対象は一つのオープンソースソフトウェアであり，調査の対象となった不具合はバッファオーバフローに関するもの 1 件である．オープンソースソフトウェア以外のソフトウェア，バッファオーバフロー以外の不具合を含む複数の不具合修正に関する調査は残された課題であった．

3.　評　　価

3.1　概　　要

2.3 で述べた課題に取り組むため，複数の商用ソフトウェアを対象とし，バッファオーバフロー以外の不具合を含む複数の不具合修正を対象とした．具体的には，パナソニック MSE 株式会社の三つの異なるプロジェクトで開発されたソースコードを対象として 2 種類の評価 A，B を実施した．三つのプロジェクトはウォータフォールモデル型の派生開発（既存システムの機能追加，一部改変）プロジェクトである．また，設計，コーディングの両フェーズにおいて設計ドキュメントのレビュー，コードレビューが実施され，次工程に不具合が持ち越されないよう工程移行の判定会議で品質が検討される．各プロジェクトの概要を表 1 に示す．プロジェクトに関する知識がない作業者であってもコードクローン検索により類似不具合を検出できるかを明らかにするため，評価 A は当該プロジェクトのプロジェクトリーダが実施し，評価 B は当該プログラミング言語でのコーディング，テストの業務経験はもつが，当該プロジェクトの知識をもたない作業者が実施した．いずれのプロジェクトで開発されたソフトウェアも稼動実績があり，評価実施時点で発見済みかつ未修正の不具合は存在しない．

評価で対象としたソースコードには，結合テスト，システムテストで発見された不具合の修正に伴う修正履歴がコメントとして記録されている．記録された修正前コードを検索キーとしてクローンセットを検索し，検索結果に他の修正箇所が含まれているかを調べる．コーディング，単体テスト中に発見された不具合修正は今回の評価の対象外である．

3.2　対象ソースコードの修正履歴

対象プロジェクトでは，テストフェーズで発見された不具合の修正手順が定義されている．定義には修正に伴うソースコード修正履歴の記録方法が述べられている．

対象プロジェクトでは，テストフェーズで発見された不具合の修正手順が定義されており，修正に伴う

表 1　対象プロジェクト
Table 1　Target projects.

プロジェクト	概要	プログラミング言語	規模（ソースコード行数）	修正箇所
A-1	Windows アプリケーション	C++	13.5k step	30
A-2	Windows クライアントサーバアプリケーション	Java	7.2k step	28
B	UNIX サーバアプリケーション	C++	20.7k step	36

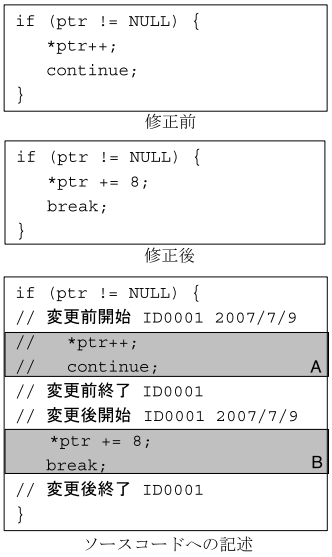

図 5　ソースコード中の修正履歴記述例
Fig. 5　An example of notation of fixed source code.

ソースコード修正履歴の記録，及び，記録方法も定義に含まれている．そのため，当該プロジェクトに関する知識をもたない場合でも，ソースコードの修正を記録から再現できる．

図 5 は修正履歴の記録例である．図 5 中 “修正前” から同図 “修正後” のようにソースコードを修正した場合，同図の “ソースコードへの記述” のように修正履歴をコメントとして記録する．具体的には，図 5 中 “ソースコードへの記述” に示すような修正前ソースコードとその開始，終了位置，修正後ソースコードの開始，終了位置と修正 ID を記す．図 5 は，修正 ID 0001 の修正の際に，if 文の中の “`*ptr++; continue;`（図中 A)” を “`*ptr += 8; break;`（図中 B)” と修正した記録である．ソースコードには，もともと記述されていた “`*ptr++; continue;` ” の前後にコメントとして修正開始と修正終了を記述し，対応する修正 ID（ID 0001）の記録を加える．また，修正前のコードである A の部分をコメントアウトし，その前後に変更前であることを示すコメントを挿入する．変更後のコードである B の部分の前後に変更後であることを示すコメントを挿入する．それぞれのコメントには修正に対応する修正 ID を記述する．

同一の不具合修正に伴ってソースコードの複数箇所が修正された場合，それぞれの修正箇所に同一の ID が記録される．評価では一つの修正 ID から一つの修正前ソースコードを選び検索キーとし，クローンセットを検索した．得られた検索結果が修正履歴に含まれるかを調べ，含まれない場合には潜在的な類似不具合でないか調べた．図 5 の例であれば，`*ptr++; continue` とクローンセットとなる部分が，同一の修正 ID の修正履歴に含まれているかを確認する．

3.3　評価方法

評価では，修正履歴をもとにすべての修正箇所を修正前の状態（不具合が再現する状態）に戻した．次に，一つの修正 ID から一つの修正前ソースコード（ソースコード片）を取り出し，クローンセットとなっている箇所を対象ソースコードから検索する．一つの修正 ID が複数のソースコード片にわたる場合には，修正履歴に記された日付が最も古いソースコード片を選ぶ．検索結果が同一 ID の他のソースコード（類似不具合）を正しく検索できているかどうかを作業者が確認し，適合率，再現率，F1 値を求める．適合率は，検索結果に含まれていたクローンセットのうち，実際に修正されていたものの割合である．再現率は，修正部分にクローンセットが含まれていたものの割合である．F1 値は適合率と再現率のバランスを示す指標であり，次式で求める．

$$\frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}}$$

検索結果の判定手順は以下のとおりである．

(a)　検索結果が修正履歴に含まれる場合

得られた検索結果が修正履歴に含まれる（修正前ソースコードとして記録されている）場合に不具合を正しく検索できたものとする．

(b)　検索結果が修正履歴に含まれない場合

得られた検索結果が修正履歴に含まれていない場合には，まず，作業者が修正が必要であるが実際には修

正されていない潜在的な不具合であるかどうかを確認し，潜在的な不具合であると判断された場合には正しく検索できたものとする．類似不具合であると判断されなかった場合には，正しく検索できなかったものとする．

なお，評価 A の作業者は当該プロジェクトのプロジェクトリーダであり，(b) の類似不具合かどうかの判断は，プロジェクトに関する知識をもって判断した．評価 B では対象プログラミング言語にてコーディング，テストの実務経験があるが，当該プロジェクトに関する知識をもたない作業者が判断した．

修正前コードとクローンセットとなるソースコード片が対象ソースコードに大量に含まれる場合（例えば `a = b;` のようなソースコード中に頻出する代入文の場合）には検索結果も膨大になり，調査の手間が大きくなる．評価 A では，実際にプロジェクトを担当しているプロジェクトリーダが本来の業務を妨げない範囲で調査を実施する必要があったため，検索結果が 10 件以上になった場合には検索結果が修正箇所を含むかどうかの確認を省略した．評価 B ではすべての検索結果を調査した．

3.4 評価 A の結果

プロジェクト A-1，A-2 ともに，修正履歴に残されていないが修正しなければならない不具合は発見されなかった．評価に利用した計算機は Pentium M 1 Ghz メモリ 1 GByte のものであり，OS は WindowsXP である．すべての検索においてコードクローン検索時間は数秒程度であった．

3.4.1 検索精度

結果を表 2 に示す．表 2 中の “修正箇所” は修正 ID が付与されたソースコード修正履歴の記録の総数である．“修正 ID 数” は不具合数であり，修正箇所から修正 ID の重複を取り除いたものである．“検索結果が 10 件以上となる修正 ID 数” はクローンセットの数が 10 件以上となった修正 ID の数である．“検索キーの平均トークン数（全体）” はすべての検索キーのトークン数の平均である．“検索キーの平均トークン数（10 件未満）” はクローンセットの検索結果が 10 件未満となるような検索キーの平均トークン数である．“適合率”，“再現率”，“F1 値” は，修正 ID ごとの適合率，再現率，F1 値の平均をとったものである．なお，クローンセットの数が 10 件以上となった修正 ID は適合率，再現率，F1 値算出の対象外である．クローンセットの数が 10 件以上となったコードを表 3 に示す．

表 2 に示すとおり，A-1 では 19 件，A-2 では 13 件の修正 ID が記録されており，それらの修正箇所はそれぞれ 30 箇所，28 箇所である．検索結果が 10 件以上となった件数は A-1，A-2 ともに 3 件ある．検索キーの平均トークン数は A-1 で平均 35.8 トークン，A-2 で平均 33.8 トークンとなった．検索結果が 10 件未満となるような検索キーの平均トークン数は A-1 で平均 38.1 トークン，A-2 で平均 42.0 トークンとなった．A-1，A-2 ともに検索結果が 10 件未満となる検索キーの方がトークン数が大きい．適合率の平均は A-1 が 91.8%，A-2 が 80.4%と A-1 の方が A-2 よりも大きかった．再現率の平均は，A-1 が 86.9%，A-2 が 85.4%と同程度であった．

3.4.2 修正箇所が多いコード片

A-1，A-2 それぞれのプロジェクトにおいて，同一修正 ID で修正箇所が多かったものを表 4 に示す．A-1 で

表 2 評価 A の結果（検索結果が 10 件以上となる修正は対象外）
Table 2 Precision, recall and F1 value in trial A.

	修正箇所	修正 ID 数	検索結果が 10 件以上となる修正 ID 数	検索キーの平均トークン数（全体）	検索キーの平均トークン数（10 件未満）	適合率	再現率	F1 値
A-1	30	19	3	35.8	38.1	91.8%	86.9%	0.872
A-2	28	13	3	33.8	42.0	80.4%	85.4%	0.826

表 3 検索結果 10 件以上の検索キー評価 A
Table 3 Search keys that produced ten or more search results in trial A.

プロジェクト	検索キー	検索結果件数
A-1	`pStrRes = (InfB_Anrmsig *)pby_wk;`	410
A-1	`CSysCom::TimerCntl(this, enTimer_Cntcnd, FALSE, bCntcndTimeFlg);`	48
A-1	`for( int i = 0; i < 5; ++i)`	29
A-2	`if( output != IsCom.RTN_OK )`	424
A-2	`output = m_reset.getTermStatus(request);`	246
A-2	`Line2 += MachAdd;`	60

表 4　修正すべき箇所が最も多いソースコード片（評価 A）
Table 4　Search keys that require largest numbers of fixes in trial A.

プロジェクト	修正すべきコード（同一 ID）	検索結果に含まれていたか
A-1	`sys->usSysStat |= SYSST_UNYO_U;`	検索キー
A-1	`sys->usSysStat |= SYSST_SYSTEM_ON;`	○
A-1	`sys->usSysStat |= SYSST_UNYO_S;`	○
A-1	`sys->usSysStat&= ~SYSST_UNYO_S;`	×

A-2	`String filename = JspData.g_MODrive + 文字列 A + m_main.reqyear +`
	`((m_main.reqmonth < 10) ? "0" : "") + m_main.reqmonth +`
	`((m_main.reqday < 10) ? "0" : "") + m_main.reqday + "_" +`
	`((m_main.reqkukan < 10) ? "0" : "") + m_main.reqkukan + ".dat";`
		検索キー

A-2	`String filename = JspData.g_MODrive + 文字列 B + m_main.reqyear +`
	`((m_main.reqmonth < 10) ? "0" : "") + m_main.reqmonth +`
	`((m_main.reqday < 10) ? "0" : "") + m_main.reqday + "_" +`
	`((m_main.reqkukan < 10) ? "0" : "") + m_main.reqkukan + ".dat";`
		○

A-2	`String filename = JspData.g_MODrive + 文字列 C + m_main.reqyear +`
	`((m_main.reqmonth < 10) ? "0" : "") + m_main.reqmonth +`
	`((m_main.reqday < 10) ? "0" : "") + m_main.reqday + "_" +`
	`((m_main.reqtiten < 10) ? "0" : "") + m_main.reqtiten + "_" +`
	`m_main.reqcar + ".dat";`
		○

A-2	`String filename = JspData.g_MODrive + 文字列 D + m_main.reqyear +`
	`((m_main.reqmonth < 10) ? "0" : "") + m_main.reqmonth +`
	`((m_main.reqday < 10) ? "0" : "") + m_main.reqday + "_" +`
	`((m_main.reqtiten < 10) ? "0" : "") + m_main.reqtiten + "_" +`
	`m_main.reqcar + ".dat";`
		○

A-2	`String filename = JspData.g_MODrive + 文字列 E + m_main.reqyear +`
	`((m_main.reqmonth < 10) ? "0" : "") + m_main.reqmonth +`
	`((m_main.reqday < 10) ? "0" : "") + m_main.reqday +`
	`((m_main.reqShour < 10) ? "0" : "") + m_main.reqShour + "_" +`
	`((m_main.reqtiten < 10) ? "0" : "") + m_main.reqtiten + "_" +`
	`m_main.reqcar + ".dat";`
		×

は，同一修正 ID で修正箇所が 4 件のものが最も修正箇所が多かった．表 4 に示すとおり，`sys->usSysStat` のビット操作に関する修正が実施されている．修正すべき四つのコードのうち，3 箇所がクローンセットとして検索できた．検索できなかったコードはビット操作演算子のオペレータが異なっていた（3 箇所は or 演算子，1 箇所は and 演算子）．また，4 箇所のうち演算子の右側のオペランドの変数名が一致しているものは 2 箇所だけで，他の 2 箇所は変数名が異なるため，変数の正規化を実施しなければ検出できない．

A-2 では，同一修正 ID で修正箇所が 5 件のものが最も修正箇所が多かった．“文字列 A” から “文字列 E” はアルファベットから構成されるすべて異なる文字列定数であり，システム固有の名称をこれらの文字列に置き換えている．表 4 に示すとおり，修正すべき 5 箇所のうち，4 箇所がクローンセットとして検索できた．4 箇所は同一の構造で文字列定数部分がすべて異なる．クローンセットとして検索できなかった 1 箇所は，連結する文字列が他より 1 項目（`m_main.reqShour`）多いものであった．

3.4.3　検索結果が 10 件を超えたコード片

表 3 は，クローンセットとして検索した件数が 10 件を超えた検索キーを示している．A-1 では，キャストを伴う比較的単純な代入文（`pStrRes = (InfB_Anrmsig *)pby_wk;`）の検索結果が 410 件と最も多かった．次に静的メソッド呼出しが 48 件，for ループが 29 件と続いている．A-2 では，構造が比較的単純な if 文（`if( output != IsCom.RTN_OK )`）の検索結果が 424 件と最も多くなった．次にメソッド呼出しが 246 件，変数の加算が 60 件と続いている．

3.5　評価 B の結果

プロジェクト B においても修正履歴に残されていないが修正しなければならない不具合は発見されなかった．評価に利用した計算機は Pentium M 1.6 Ghz メモリ 768 MByte のものであり，OS は WindowsXP である．すべての検索においてコードクローン検索時間は数秒程度であった．

3.5.1　検索精度

結果を表 5 に示す．表 5 中の “修正箇所”，“修正 ID 数”，“検索キーの平均トークン数” 等は表 2 と同

表 5　評価 B の結果
Table 5　Precision, recall and F1 value in trial B.

	修正箇所	修正 ID 数	検索キーの平均トークン数	適合率	再現率	F1 値
すべて	36	22	21.3	37.2%	89.3%	0.394
検索結果が 10 件未満のもの	18	10	28.5	63.5%	90.0%	0.654

表 6　評価 B で検索結果が多かったソースコード片
Table 6　Search keys that produced larger numbers of search results in trial B.

検索キー	検索結果件数	修正すべき件数	検索結果に含まれていた修正すべき件数
`if( pst_Position->c_smeter_sts == DEF_DMCAT_METER_ON)`	568	1	1
`if (_mst_config->us_cleansing == 1)`	568	1	1
`char c_Mode;`	511	3	2
`sprintf((char *)_muc_Method, DEF_LOG_SS_CTDREGULARIZE, "mRegularize");`	201	1	1
`_mLogPut( APLOG, DEF_LOG_TRACE, uc_ApMsg, (UCHAR*)DEF_LOG_MSG03 );i_ret = RTN_NON;`	122	4	4

表 7　修正すべき箇所が最も多いソースコード片（評価 B）
Table 7　Search keys that require largest numbers of fixes in trial B.

修正すべきコード（同一 ID）	検索結果に含まれていた

`sprintf((char *)pc_Edit, "XXX0=[%02X%02X%02X%02X%04X%04X%02X%02X%02X%02X%08X%08X%02X%02X%02X%02X%02X%02X]",`
`pst_RawData->c_cmd, pst_RawData->c_subcmd, pst_RawData->c_center_id, pst_RawData->c_trance_id,`
`pst_RawData->us_data_size, pst_RawData->XXX1, pst_RawData->XXX2, pst_RawData->XXX3,`
`pst_RawData->XXX4, pst_RawData->XXX5, (INT)pst_RawData->XXX6, (INT)pst_RawData->XXX7,`
`pst_RawData->c_rcv_mon, pst_RawData->c_rcv_day, pst_RawData->c_rcv_hour, pst_RawData->c_rcv_min,`
`pst_RawData->c_rcv_sec, pst_RawData->c_yobi);`
	検索キー

`sprintf((char *)pc_Edit, "XXX0=[%02X%02X%02X%02X%04X%04X%02X%02X%02X%02X%08X%08X%02X%02X%02X%02X%02X%02X]",`
`pst_RawData->c_cmd, pst_RawData->c_subcmd, pst_RawData->c_center_id, pst_RawData->c_trance_id,`
`pst_RawData->us_data_size, pst_RawData->XXX1, pst_RawData->XXX2, pst_RawData->XXX3,`
`pst_RawData->XXX4, pst_RawData->XXX5, (INT)pst_RawData->XXX6, (INT)pst_RawData->XXX7,`
`pst_RawData->c_rcv_mon, pst_RawData->c_rcv_day, pst_RawData->c_rcv_hour, pst_RawData->c_rcv_min,`
`pst_RawData->c_rcv_sec, pst_RawData->c_yobi);`
	○

`sprintf((char *)pc_Edit, "XXX0=[%02X%02X%02X%02X%04X%04X%02X%02X%02X%02X%08X%08X%02X%02X%02X%02X%02X%02X]",`
`pst_RawHead->c_cmd, pst_RawHead->c_subcmd, pst_RawHead->c_center_id, pst_RawHead->c_trance_id,`
`pst_RawHead->us_data_size, pst_RawData->XXX1, pst_RawData->XXX2, pst_RawData->XXX3,`
`pst_RawData->XXX4, pst_RawData->XXX5, (INT)pst_RawData->XXX6, (INT)pst_RawData->XXX7,`
`pst_RawData->c_rcv_mon, pst_RawData->c_rcv_day, pst_RawData->c_rcv_hour, pst_RawData->c_rcv_min,`
`pst_RawData->c_rcv_sec, pst_RawData->c_yobi);`
	○

`sprintf((char *)pc_Edit, "XXX0=[%02X%02X%02X%02X%04X%04X%02X%02X%02X%02X%08X%08X%02X%02X%02X%02X%02X%02X]",`
`pst_RawHead->c_cmd, pst_RawHead->c_subcmd, pst_RawHead->c_center_id, pst_RawHead->c_trance_id,`
`pst_RawHead->us_data_size, pst_RawData->XXX1, pst_RawData->XXX2, pst_RawData->XXX3,`
`pst_RawData->XXX4, pst_RawData->XXX5, (INT)pst_RawData->XXX6, (INT)pst_RawData->XXX7,`
`pst_RawData->c_rcv_mon, pst_RawData->c_rcv_day, pst_RawData->c_rcv_hour, pst_RawData->c_rcv_min,`
`pst_RawData->c_rcv_sec, pst_RawData->c_yobi);`
	○

`sprintf((char *)pc_Edit, "XXX0=[%02X%02X%02X%02X%04X%04X%02X%02X%02X%02X%08X%08X%02X%02X%02X%02X%02X%02X]",`
`pst_RawData->c_cmd, pst_RawData->c_subcmd, pst_RawData->c_center_id, pst_RawData->c_trance_id,`
`pst_RawData->us_data_size, pst_RawData->XXX1, pst_RawData->XXX2, pst_RawData->XXX3,`
`pst_RawData->XXX4, pst_RawData->XXX5, (INT)pst_RawData->XXX6, (INT)pst_RawData->XXX7,`
`pst_RawData->c_rcv_mon, pst_RawData->c_rcv_day, pst_RawData->c_rcv_hour, pst_RawData->c_rcv_min,`
`pst_RawData->c_rcv_sec, pst_RawData->c_yobi);`
	○

じである．なお，評価 A との比較のためにクローンセットの数が 10 件未満の結果とすべての検索結果を含めた場合の結果の両方を示している．クローンセットの検索結果数上位 5 件を表 6 に示す．

表 5 に示すとおり，22 件の不具合が修正されており，そのうちクローンセットの検索結果が 10 件未満となるものは 10 件あった．それらの修正に必要となった修正箇所は，それぞれ，36 箇所，18 箇所である．検索キーの平均トークン数は全体で 21.3 トークン，クローンセットの検索結果が 10 件未満の検索キーに限

定すると平均 28.5 トークンであった．評価 A と同様に検索結果が 10 件未満の方がトークン数が大きい．適合率の平均は全体で 37.2%，検索結果が 10 件未満のものに限定すると 63.5%である．検索結果が 10 件未満となるような検索キーに限定した方が適合率が大きい．評価 A の検索結果が 10 件未満の適合率と比較すると評価 B のそれは小さい．再現率の平均は全体が 89.3%，検索結果が 10 件未満となるような検索キーの場合には 90.0%となった．再現率は評価 A と比較すると評価 B の方が大きい．

3.5.2 修正箇所が多かったコード片

評価 B において，同一修正 ID で修正箇所が最も多かったものを表 7 に示す．評価 B では，同一修正 ID で修正箇所が 5 件のものが最も修正箇所が多かった．“XXX0” は日本語の文字列定数，“XXX1” 以降はアルファベットから構成される互いに異なる文字列定数または変数名であり，システム固有の名称を置き換えている．表 7 に示すとおり，修正すべき 5 箇所のうち，すべてがクローンセットとして検索できた．なお，5 箇所は全く同一のソースコード片である．

3.5.3 検索結果が 10 件を超えたコード片

表 6 は，検索件数の上位 5 位を示している．最もクローンセットが多かった 2 件は，左辺に構造体メンバ，右辺に定数をもち，それらの一致を確かめる if 文であった．ともに 568 件のクローンセットが得られ，修正すべき箇所も，ともに 1 件であった．3 番目に検索結果が多かったのは，文字列型変数の宣言であり，511 個のクローンセットが検索結果として得られた．修正が必要な箇所は 3 箇所あり，2 箇所が検索結果の中に含まれていた．4 番目に検索結果が多かったのは，sprintf によるログ出力部分であり，201 件のクローンセットが検索結果として得られた．修正すべき箇所は 1 件であり，検索結果の中に含まれていた．5 番目に検索結果が多かったのは，関数呼出しと代入文の 2 ステートメントからなるソースコード片であり，122 件のクローンセットが検索結果として得られた．修正すべき箇所は 4 件あり，それらすべてが 122 件の検索結果の中に含まれていた．

4. 考　　察

4.1 検索キーとなるコード片

すべての評価において大きな再現率を得ることができ，類似の不具合の発見においてコードクローン検索が有用であることを示す結果が得られた．Higo らの報告 [9]，泉田らの報告 [12]，Li らの報告 [15] で確認されたオープンソースソフトウェアを対象としたコードクローン検索による類似不具合の発見と同様に，商用のソフトウェアを対象とした場合にも，類似不具合の発見にコードクローン検索が役立つことを示す結果が得られた．

検索対象のソースコードに頻出しないソースコード片（特徴的なソースコード片）が検索キーとなる場合には，再現率，適合率ともに大きな値が得られた．本評価で対象としたソースコードでは表 4 や表 7 に示すような，ビット演算，文字列連結，複雑な引数をもつ関数呼出し，において特によい結果が得られた．例外/エラー処理，ログ出力等，特徴的な部分の修正においてコードクローン検索による類似不具合検索が特に効果を発揮することが期待される．

適合率，再現率ともに 100%となるような検索キーは，複数行にわたるものが多かった．例えばエラー処理の追加等で，修正箇所（検索キー）の前に条件分岐の条件部分（エラー条件）を追加，修正箇所の後に else 節（エラー処理）を追加するようなものがあった．

表 3 や表 6 に示すような対象ソースコード中に頻出する単純な代入文，条件が単純な if 文，メソッド呼出しについては，検索結果数が非常に大きくなり，適合率が小さくなる結果が得られた．また，すべての評価において，すべての検索キーの平均トークン数よりも検索結果が 10 件未満となる検索キーの平均トークン数の方が大きかった．

4.2 作業者との議論

評価 A を実施した作業者であるプロジェクトリーダ，評価 B を実施した作業者との議論から，以下のような意見が得られた．

- 修正コードに関する記憶や知識があれば，再現率が大きくなるように意図的に検索キーを短くしたり，検索キーを変化させながら複数回検索する等，より再現率が大きくなるような工夫ができる．
- 派生製品や類似機能の不具合調査など迅速かつ網羅的な確認に特に有用である．
- 今回は開発終了後（出荷後）のソースコードで結合，システムテスト以降の記録を対象としたが，コーディング，単体テスト実施等にプログラマが類似箇所を調査する際にも役立つ．
- コピーアンドペーストした後，変数名，関数/メソッド名を変更することは実際の開発でよく起こる．定数，変数名の正規化により，そのような場合でもク

ローンセットとして検索結果に現れるので，キーワード検索と比較して，再現率がより大きくなっている．

- 本評価で実施したような類似不具合の検索を，grep 等のキーワード検索等の方法で検索キーを試行錯誤しながら検索した場合，1 件当り数時間必要となることがある．コードクローン検索による方法で，同様に検索キーを試行錯誤しながら検索した場合でも，数十分程度に短縮できそうである．

評価 B において検索結果を 10 件未満に限定した場合と限定しなかった場合を比較すると再現率はほとんど変わらないが，10 件未満の方が適合率が大きくなる結果が得られた．評価 A では，作業者の開発業務との兼合いにより，検索結果が 10 件以上となる検索キーに関する調査を途中で断念した．評価 A の作業者に評価 B の結果を見せたところ，評価 A において検索結果が 10 件以上となる検索キーにおいても，評価 B と類似の結果となるだろうという意見が得られた．

4.3 コードクローン検索の有効性

対象としたソースコードの類似不具合検出方法（従来の検出方法）は，開発担当者，プロジェクトリーダの記憶やドキュメントに基づいた修正部分の推測，修正部分を検索キーとしたソースコードのキーワード検索，レビューやテストによる確認，である．対象ソースコードに記された修正日時から，従来の検出方法で同時に修正されていなかったと推測される（修正日時に隔たりがある）修正箇所をコードクローン検索で検出できているものがあった．一方，従来の検出方法で同時に修正されていたと推測される（修正日時が近い）類似不具合のうちコードクローン検索で検出できていないものもあった．そのような類似不具合には異なる文法表現を用いられたもの（例えば，表 4 に示したような`&=` 演算子と`|=` 演算子）や検索キーとなるソースコード片の一部にステートメントや変数の追加，削除があったものである．

対象とした 3 プロジェクトはいずれもウォータフォールモデル型の派生開発（既存ソフトウェアへの機能追加，一部改変）プロジェクトであり，保守性向上を目的としたクラス階層の変更を伴うような大幅なリファクタリングは含まれない．不具合の多くはレビュー，あるいは，テストにより同一工程内で除去される．今回対象とした修正には大規模なソースコード変更を伴う修正は存在しなかった．今回対象としたソフトウェアと異なる開発プロセスや形態についても，類似不具合の発見にコードクローン検索が有効か検討する必要がある．

評価では，結合テスト以降に発見され同一の修正 ID が付与されているものを類似不具合とし，文献 [14] の定義によるコードクローン検索を用いて，類似不具合をどの程度検出できるか調べた．文献 [14] の定義では，対象ソースコードの変数名や定数名を正規化し，一定以上の長さのトークンが一致すればコードクローンとしており，表 4，表 7 に示したような変数名や定数名が異なるビット演算，文字列連結，複雑な引数をもつ関数呼出しに関する類似不具合を検出することができた．一方，表 3，表 6 に示したような変数名や定数名の正規化により検索結果件数が大幅に増加したと考えられるものもあった．表 3，表 6 に示したような対象ソースコード中に頻出するコード片が検索キーとなる場合は，別の手法との併用や正規化を変数や定数ごとに個別に抑制する等の工夫が考えられる．

5. 関 連 研 究

ソースコードの静的解析により潜在的な不具合の検出を試みる手法が提案されている [7], [8]．また，Lint [13]，Findbugs [10] をはじめ静的解析ツールも多く開発されている．これらの手法やツールは，特定の状況で起こりやすいコーディングミス（初期値の代入忘れ）や論理的な不具合を手法の提案者やツールの開発者がパターン化し，そのパターンに当てはまるソースコードの一部を潜在的な不具合の候補として利用者に提示するものである．本研究ではそのようなパターン化の作業は必要ない．また，これらの静的解析手法やツールでは，本研究のように文脈に依存する（与えられた不具合と類似の）不具合の候補を検出することはできない．

保守性の向上を目的として，多くのコードクローン検出手法が提案されている．例えば，文献 [4] や文献 [17] では，関数やメソッドを最小単位としてコードクローンを検出する手法を提案している．また，文献 [2] では，行単位でのコードクローン検出手法が提案されている．しかしながら，本論文で対象とするような修正ソースコードの検索ではより細かい粒度でコードクローンを検索できる必要がある．

ソースコードの類似部分の理解を支援することを目的とした可視化手法が提案されている．例えば，文献 [5] や [6] では散布図により，二つのソースコードの一致する部分を行単位で可視化している．具体的には，縦軸と横軸にそれぞれ一つのソースコードを割り当て，

一致している行を点として二次元平面にプロットし，利用者に提示する．ただし，いずれの手法も変数名が変更されると異なる行として認識するため本論文のような変数名を正規化した上でのコードクローンの提示はできない．

6. むすび

コードクローン検索による類似不具合の発見を商用開発の現場へ適用する際の指針となることを目的とし，パナソニック MSE 株式会社で開発された三つの異なるプロジェクトで開発されたソースコードを対象に，類似不具合の検出を実証的に評価した．コードクローンの検索結果が 10 件未満となるようなソースコード片を対象とした 3 件の評価において，平均で 78.6%の適合率，87.4%の再現率が得られた．また，検索結果を限定しない評価 1 件において，37.2%の適合率，89.3%の再現率が得られた．

キーとなるソースコード片のトークン数が大きな場合に大きな適合率が得られることが分かった．評価は当該プロジェクトのプロジェクトリーダと当該プログラミング言語での実務経験があるがプロジェクトに関する知識のない者により実施した．評価により，従来確認されていたオープンソースソフトウェアのソースコードだけでなく，三つの異なる商用プロジェクトで開発されたソースコードにおける類似不具合発見にもコードクローン検索が有効であることを確認した．また，評価 B が実施できたことにより，ソースコード修正履歴があれば，プロジェクトに関する知識のない作業者でも検索が実施できること，及び，当該プロジェクトに関する知識のある作業者が実施した評価 A と同程度の再現率が得られることを確認した．

評価では，**4.1** に示したようなエラー処理，表 7 で示したようなログ出力をはじめとして，検索対象のソースコードに頻出しないソースコード片（特徴的な部分）において特に大きな適合率，再現率が得られた．また，評価を実施したプロジェクトリーダとの議論から，本評価のようなシステムテスト時の類似不具合検出だけでなく，派生製品の類似不具合検出，コーディングや単体テスト時の類似不具合検出にも有用であるとの評価を得た．

一方，検索ソースコード片の長さ（トークン数）が小さい場合には，検索結果が非常に多数になり，効率的でない場合があることが確認された．このような場合，コードクローン検索時の変数名の正規化の制限，grep 等のキーワード検索など別の手法との併用が考えられる．

今後は，コードクローン検索による検出が効果的な類似不具合を分類した上で，本論文での検索手順を実際のソフトウェア開発プロセスに組み込んでいく際の課題を挙げ，実際に進行中の開発への適用可能性を検討する予定である．

謝辞 EASE プロジェクトのメンバに感謝する．CCFinder の作者の神谷年洋氏（現独立行政法人産業総合研究所）に感謝する．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，「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．

文　献

[1] 日本語入力システム「かんな」, http://canna.sourceforge.jp/

[2] B. Baker, “On finding duplication and near-duplication in large software systems,” Proc. Second Working Conference on Reverse Engineering, p.86, 1995.

[3] I. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” Proc. International Conference on Software Maintainance, pp.368–377, 1998.

[4] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis, “Measuring clone based reengineering opportunities,” Proc. Sixth IEEE International Symposium on Software Metrics, pp.292–303, 1999.

[5] K. Church and J. Helfman, “Dotplot: A program for exploring self-similarity in millions of lines of text and code,” J. Computational and Graphical Statistics, vol.2, no.2, pp.153–174, 1993.

[6] S. Ducasse, M. Rieger, and S. Demeyer, “A language independent approach for detecting duplicated code,” Proc. International Conference on Software Maintenance, pp.109–118, 1999.

[7] D. Engler and K. Ashcraft, “RacerX: Effective, static detection of race conditions and deadlocks,” Proc. ACM Symposium Operating Systems Principles, pp.237–252, 2003.

[8] S. Hallem, B. Chelf , Y. Xie, and D. Engler, “A system and language for building system-specific, static analyses,” Proc. ACM SIGPLAN Conference Programming Language Design and Implementation, pp.69–82, 2002.

[9] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue, “Simultaneous modification support based on code clone analysis,” Proc. 14th Asia-Pacific Software Engineering Conference, pp.262–269, 2007.

[10] D. Hovemeyer and W. Pugh, “Finding bugs is easy,” SIGPLAN Notice, vol.39, no.12, pp.92–106, 2004.

[11] ICCA (Integrated Code Clone Analyzer): http://sel.ics.es.osaka-u.ac.jp/icca/index.html
[12] 泉田聡介，植田泰士，神谷年洋，楠本真二，井上克郎，“ソフトウェア保守のための類似コード片検索ツール，” 信学論（D-I），vol.J86-D-I, no.12, pp.906–908, Dec. 2003.
[13] S. Johnson, “Lint, a C program checker,” Unix Programmer’s Manual, AT&T Bell Laboratories, 1978.
[14] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A multi-linguistic token-based code clone detection system for large scale source code,” IEEE Trans. Softw. Eng., vol.28, no.7, pp.654–670, 2002.
[15] Z. Li, S. Lu, and S. Myagmar, “CP-Miner: Finding copy-paste and related bugs in large-scale software code,” IEEE Trans. Softw. Eng., vol.32, no.3, pp.176–192, 2006.
[16] Libra, http://sel.ist.osaka-u.ac.jp/icca/libra.html
[17] J. Mayrand, C. Leblanc, and E. Merlo, “Experiment on the automatic detection of function clones in a software system using metrics,” Proc. IEEE International Conference on Software Maintenance, pp.244–253, 1996.

（平成 20 年 1 月 29 日受付，5 月 23 日再受付）

森崎　修司

平 13 奈良先端大・情報科学・博士課程了．同年（株）インターネットイニシアティブ入社．平 17 奈良先端大・産学官連携研究員．平 19 同大学・情報科学・助教．博士（工学）．エンピリカルソフトウェア工学，インターネットを介した知識共有の研究に従事．IPSJ，IEEE 各会員．

吉田　則裕

平 16 九工大・情報工・知能情報卒．平 18 阪大大学院博士前期課程了．現在同大学院博士後期課程在学中．リファクタリング支援及びデザインパターン適用支援の研究に従事．情報処理学会，人工知能学会，IEEE 各会員．

肥後　芳樹　（正員）

平 14 阪大・基礎工・情報中退．平 18 同大大学院博士後期課程了，平 19 阪大・情報・コンピュータサイエンス・助教．博士（情報科学）．コードクローン分析やリファクタリング支援に関する研究に従事．情報処理学会，IEEE 各会員．

楠本　真二　（正員）

昭 63 阪大・基礎工・情報卒．平 3 同大大学院博士課程中退．同年同大・基礎工・情報・助手．平 8 同大講師．平 11 同大助教授．平 14 阪大・情報・コンピュータサイエンス・助教授．平 17 同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．IPSJ，IEEE，JFPUG，PM 各会員．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校情報工学科・助教授．平元阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．平 14 阪大・情報・コンピュータサイエンス・教授．工博．ソフトウェア工学の研究に従事．IPSJ，JSSST，IEEE，ACM 各会員．

佐々木健介

平元松下システムエンジニアリング（株）（現パナソニック MSE（株））入社．公共インフラシステムのソフトウェア開発，SE 業務を経て，CMM 認定推進活動に従事．産官学連携の COSE プロジェクトに従事．ユビキタスネットワーク事業部 e ソリューション開発グループ主任技師．所属開発組織の SEPG，及び品質管理担当．

村上　浩二

昭 63 松下システムエンジニアリング（株）（現パナソニック MSE（株））入社．公共インフラシステムのソフトウェア開発，産官学連携の COSE プロジェクトに従事．ユビキタスネットワーク事業部 e ソリューション開発グループ主任技師．

松井　　恭

昭 62 松下システムエンジニアリング（株）（現パナソニック MSE（株））入社．公共インフラシステムのソフトウェア開発・産学官連携の業界標準化 SE 活動を経て CMM 認定推進等の開発組織運営に従事．ユビキタスネットワーク事業部 e ソリューション開発グループグループマネージャー．