解　説

コードクローン検出技術の展開

神谷 年洋　肥後 芳樹　吉田 則裕

コードクローンとは，典型的には，開発者がソースコードをコピー＆ペーストにより再利用して作られたコード断片である．コードクローンはほぼあらゆるソフトウェアのソースコードに存在し，やっかいな問題を引き起こすと同時に，ソフトウェアに関するさまざまな知見をもたらす．本稿では，コードクローンに関する技術，特にコードクローン検出手法について，さらに，検出手法の応用について説明する．

A code clone is typically created by a developer reusing a code fragment in a copy-and-paste manner. Such copied code fragments usually exist in source code of almost every software products. The code clones are known as potential troubles in maintenance, while we can learn kinds of knowledge from the existence of such code fragments. This paper explains about topics about code clone, especially detection methods of code clones and various applications of the detection methods.

1　はじめに

コードクローンとは何か．その定義は研究者によって異なるが,「コード」と「クローン」が意味するところ，すなわち，ソースコードのコピー＆ペーストによって作られた (あるいは，作られたように見える) コードの断片 (図 1) はコードクローンと呼ぶことができる，というのは大方の研究者の一致するところであろう．

図 2 に，C 言語のソースファイルから 4.4 で後述する手法によって検出されたコードクローンの例を示す．この例では，2 つのコード断片は一字一句同じというわけではなく，改行の位置が異なっている，文字列リテラルが異なる，左側のコード断片の中程に右側のコード断片には存在しない処理がある，などの相違点がある．

コードクローンはなぜ問題なのか．それは，コードクローンがソースコード中に存在することがソフトウェアの保守を難しくするといわれているからである．ただし，それがどの程度の問題であるかは，現在に至るまで研究者の間で意見が分かれている (6.2 で後述)．

開発者がコードクローン問題に巻き込まれる典型

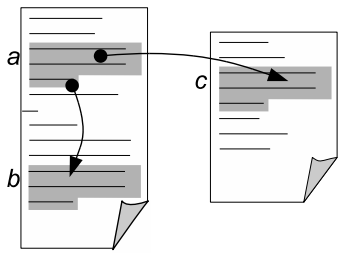

図 1　コピー＆ペーストによって作られるコードクローン．この図では，コード断片 (a) が，同じソースファイル内 (b) や別のソースファイル (c) へとペーストされている．これらのコード断片 a, b, c は互いにコードクローンとなる．

Evolving and Hot Topics on Code Clone Detection Techniques.

Toshihiro Kamiya, 公立はこだて未来大学システム情報科学部, School of Systems Information Science, Future University Hakodate.

Yoshiki Higo, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University.

Norihiro Yoshida, 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.

コンピュータソフトウェア, Vol.28, No.3 (2011), pp.29–42.
[解説論文] 2010 年 12 月 30 日受付.

```
2860 if ((pSMBr->resp.hdr.WordCount == 3)
2861     || ((pSMBr->resp.hdr.WordCount == 4)
2862     && (blob_len <
2863         pSMBr->resp.ByteCount))) {
2864
2865     if (pSMBr->resp.hdr.WordCount == 4) {
2866         bcc_ptr += blob_len;
2867         cFYI(1, ("Security Blob Length %d"
2868               blob_len));
2869     }
2870
2871     cFYI(1, ("NTLMSSP Challenge rcvd"));
2872
2873     memcpy(ses->server->cryptKey,
2874            SecurityBlob2->Challenge,
2875            CIFS_CRYPTO_KEY_SIZE);
. . .
2890     if (smb_buffer->Flags2 & SMBFLG2_UNICODE) {
2891         if ((long) (bcc_ptr) % 2) {
2892             remaining_words =
2893                 (BCC(smb_buffer_response)
2894                  - 1) / 2;
2895          /* Must word align unicode strings */
2896             bcc_ptr++;
2897         } else {
2898             remaining_words =
2899                 BCC
2900                 (smb_buffer_response) / 2;
2901         }
2902         len =
2903             UniStrnlen((wchar_t *) bcc_ptr,
2904                    remaining_words - 1);
```

```
3281 if ((pSMBr->resp.hdr.WordCount == 3)
3282     || ((pSMBr->resp.hdr.WordCount == 4)
3283     && (blob_len <
3284         pSMBr->resp.ByteCount))) {
3285     if (pSMBr->resp.hdr.WordCount == 4) {
3286         bcc_ptr +=
3287             blob_len;
3288         cFYI(1,
3289              ("Security Blob Length %d ",
3290               blob_len));
3291     }
3292
3293     cFYI(1,
3294          ("NTLMSSP response to Authenticate "));
3295
3296     if (smb_buffer->Flags2 & SMBFLG2_UNICODE) {
3297         if ((long) (bcc_ptr) % 2) {
3298             remaining_words =
3299                 (BCC(smb_buffer_response)
3300                  - 1) / 2;
3301             bcc_ptr++;    /* Unicode strings must be word align
3302         } else {
3303             remaining_words = BCC(smb_buffer_response) / 2;
3304         }
3305         len = UniStrnlen((wchar_t *) bcc_ptr,
3306                 remaining_words - 1);
```

図 2　Linux カーネル 2.6.28 のソースファイル/fs/cifs/connect.c 中から検出されたあるコードクローンの一部．左カラムのコード断片と右カラムのコード断片がコードクローンになっている．行番号の背景が黒くなっている行が一致部分．白い行が不一致部分または空行．詳細については 4.4 で後述．

的なシナリオは以下のようなものである:

1. 開発者は，プロダクトのある機能を修正するために，プロダクトのソースコードの中で，その機能を実現しているであろう部分を特定する．
2. 該当部分を修正し，テストを行う．すると，特定のテストケースでは直っているが，別のテストケースでは直っていないことが判明する．
3. ソースコードを確認してみると，修正した部分と同じようなコード断片が，何カ所もあることが分かる．開発者は，そのそれぞれについて，修正すべきか否か，判断しなければならない．

特に，大規模なソフトウェアであれば，この「同じようなコード断片」を見つける作業の手間だけでも相当なものになってしまう．結果として，コードクローンの存在はソースコードの変更をより困難にすると考えられるため， ソースコードは絶えず変化するものであるという (アジャイル開発手法の) 立場をとる開発者には当然ながら評判が悪く,「重複の理由が何であれ，いったん見つけたら，徹底的につぶすことだ」[81] と断じられている．しかし，その一方で，ソースコードをエディタで編集する作業がある限り，コードクローンを (少なくとも一時的に) ソースコード中に作り込むことなく開発を進めることは非現実的である．これは，コピー＆ペーストを禁じてソースコード，あるいはより一般に，何らかの文書を作ることを考えてみれば納得がいくであろう．ソースコードを人間が編集するようなソフトウェア開発プロセスにおいては，コードクローンは必ず発生するものである，というところから対策を始めるしかない．

本稿は，90 年代に始まったコードクローンの研究について， 現在に至るまで，コードクローン検出手法をはじめとする種々のコードクローンに関連する手法や応用の展開を俯瞰するものである．本稿で紹介されている研究の詳細については，引用されている文献を参考にされたい．また，本稿で触れていない (主に技術的な) トピックについては，7 で触れられている文献 [33] を参考にしていただきたい．

以下，2 ではコードクローンが作られてしまう原因について説明する．3 ではコードクローンを検出することで何ができるのか，その応用を紹介し，4 では，

それらの応用を可能にする，コードクローン検出手法について紹介する．さらに，5 では，検出されたコードクローンを，あるいは，コードクローンを用いて分析を行う手法を紹介する．6 では，コードクローンに関する研究コミュニティを紹介し，7 でまとめを述べる．

2 コードクローンが作られる理由および対策

コードクローンが作られる原因の代表的なものは，開発者がソースコードをコピー&ペーストにより再利用することであるが，これら以外にもさまざまな原因がある．以下，大きく，開発技術を原因とするもの，開発者の行為を原因とするもの，開発組織の方針 (あるいは管理) を原因とするものに分類して説明する．

2.1 開発技術を原因とするもの

特定の開発技術の性質あるいは制約により，コードクローンを作り込むことが避けられない場合がある．

コード生成ツール Lex/yacc をはじめとするコード生成ツールが生成するコードは，同じようなパターンが大量に含まれる (冗長な) ものとなる．人間が編集してコードクローンを作ったものではないため，コードクローンとは呼ばない向きもあるが，本稿ではコードクローンに含める．生成されたコードが (手作業で編集されるなどにより) 生成に用いた元のデータと対応がつかなくなると，冗長で，修正が面倒なものとなる．

プログラミング言語の機能不足 プログラミング言語の進化により，ある共通の機能が複数の場所で利用されるときに，その機能を 1 つのルーチンで書き下せる機会が増えている．たとえば，オブジェクト指向言語では，ポリモーフィズムにより，異なった種類のオブジェクトに対する操作を，同じメソッド呼び出しの列で書くことができる．こういった機能を持たない言語では，類似した処理を何度も書き下すことになる．

定型処理・横断的関心事 GUI(graphical user interface) やデータベースのライブラリなどで，特定の機能を呼び出すのにほぼ定型のコードを書くことを要求するものがある．たとえば，データベースをオープンし，クエリを作成して問い合わせ，結果を受け取り，クローズする，といった処理である．ログ生成ルーチンなどの横断的関心事は，モジュールのあちこちに，あるいは，多数のモジュールに出現する，比較的短い (数行程度の) コードクローンとなる．

これら，定型処理や横断的関心事は一般のコピー&ペーストの場合と異なり，誰が書いても同じようなコードになることが期待されるため，コードクローンではあるが許容される傾向にある．

あるプロダクトのソースコードで頻繁に用いられている定型処理の一覧をひな形 (あるいはレシピ) として用意しておけば，開発者の助けになるだろう．実際，サードパーティのライブラリはしばしば，そのライブラリの利用例や定型処理をライブラリのソースコードとともに配布される．

4 で後述する検出ツールによって検出されるコードクローンの中には，定型処理や横断的関心事に相当するものがあることが，複数の研究者によって指摘されている [3] [11] [12] [40] [42]．

2.2 開発者の行為を原因とするもの

開発者が何らかの作業をしてしまうこと，あるいは，何らかの作業に失敗することにより，コードクローンをソースコードに作り込んでしまうことがある．

コピー&ペーストによる再利用 既存の処理と同じ処理，あるいは，ほぼ同じで少しだけ異なる処理が必要になった際に，開発者が既存のルーチンをコピー&ペーストして新しいルーチンを作り，必要に応じて書き換える．極端な例としては，ソースファイルを丸ごとコピーして，定数を数ヶ所書き換えたようなものも存在する．

リファクタリングの欠如 上述のコピー&ペーストによる再利用が行われていても，再利用部分が管理できていて，適宜リファクタリング (共通部分を括り出したルーチンを作り，コードクローンになっているコード断片をそのルーチンへの呼び出しに修正) すれば，コードクローンの拡散を防ぐことができると考えられる．

コピー&ペーストによる再利用を放置し，リファ

クタリングの機会を逸すれば，ソースコード中にコードクローンになっているコード断片が徐々に増えていくことになる．

コードクローンを検出するツールのなかには，ソースコードを修正してコードクローンを除去するもの，あるいは，検出されたコードクローンを除去するためのリファクタリング手法を提示するものがある [35] [31] [82] [119]．このようなツールは，リファクタリングをサポートする目的で利用することができるであろう．

設計段階での共通機能の括り出しの失敗 プログラミング言語 (実装のための技術) に上述のような共通部分の括りだしのための機能が備わっていても，設計段階で共通の機能を括り出していなければ，同じようなルーチンが複数実装されることは防げない．コーディングを行う開発者がモジュールの単位や依存関係を変更することが許されていなければ，類似したコード断片を 2 つのモジュールに書くことになる．

コード剽窃 企業の開発者が GPL [27] ライセンスの下で配布されているプロダクトを不用意に (ライセンスを理解することなく，あるいは意図的に) 再利用し，クローズドソースのライセンスの下で出荷してしまうことで，GPL ライセンス違反が発生する [55] [92]．

このような状況では，オリジナルのオープンソースのプロダクトのソースコードと，企業のプロダクトのソースコードの間でコードクローンが存在することになる．

ソースコードを検査し，GPL 違反を検出するサービスを提供している企業 [8] [93] が登場した．また，ソースコードの剽窃を検出するのに特化したツールも出現している [90] [94]．

2.3 開発組織の方針を原因とするもの

開発組織の方針によっては，開発者がコードクローンを作り込むことが不可避になったり，あるいは，開発者に積極的にコードクローンを作り込ませることになる．

「動いているコードに触るな」 信頼性の確保を理由として，開発組織が開発者に対し，運用されているシステムのソースコードをなるべく修正するなという方針を課すことがある． 機能を修正する唯一の方法は，既存のソースコードのコピーを作り必要な修正を加えることになる．もちろん，上述のリファクタリングは全く行われない．結果として，プロダクトにはコードクローンが作り込まれることになる．

管理の欠如 例えば，開発組織が，開発者に対して極端に短い納期を強制し，ソースコードの品質を度外視する．開発者は納期に間に合わせるため，とにかく動くことを優先することになる．これにより，設計段階での共通機能の括り出しも，リファクタリングも行われなくなる．

また，開発組織が一貫した構成管理を導入しないと，ソースコードの管理が各々の開発者任せになり，それぞれの開発者が少しずつ異なった内容のソースファイルを手元に持つことになる．これらのソースファイルに含まれるコード断片は互いにコードクローンとなり，それらの差違については，なぜ異なるのか誰も正確には知らない，という状況になる．

行数による評価 開発者の生産性の評価が単なる行数で計測され，冗長で長ったらしいコードの方が評価される．結果として，開発者はソースコードをコピー&ペーストし，ソースコードを少しでも長くしようとする．

3 コードクローン検出の目的

コードクローン検出手法を応用し， ソフトウェア開発やソフトウェア工学研究に利用する方法が提案されている．

3.1 品質評価

ソフトウェアは長期間利用され，何度も修正が加えられるにつれて，当初の設計にそぐわない箇所が出てくる．どこかの時点で設計を見直さないかぎり，理解容易性や保守性といった品質が低下してしまう (より詳細な議論については 6.2 を参照)．プロダクトのソースコード中で，コードクローンになっているコー

ド断片が占める割合 (コードクローンによるカバレッジ [86] [87]) を，ソースコードの品質のメトリックとして利用する．実際に，成功しているオープンソースのプロダクトに対してコードクローンを調べた研究では，クローン含有率は 5〜10%という低い割合であった [106] [111].

3.2 コード修正前のチェック

ソースコードのある部分を修正する前に，同じようなコード断片がプロダクト中に存在しないかをチェックする [30] [43] [89]．存在するようであれば，それらすべてのコード断片について，これから行おうとしている修正を適用すべきかどうか判断する．コードクローン検出技術により (1) 修正対象のコード断片中の名前が書き換えられていても，類似コードを検索できる，(2) 修正対象のコード断片の長さが数十行以上であっても，一度に検索できる (検出のパラメータを調節することで，修正対象のコード全体と類似しているコード断片を検出することも，一部と類似しているコード断片を検出することもできる).

3.3 コード剽窃の検出・防止

前節の 2.2 で述べたように，他者のソースコードを配布時のライセンスや法律によって許可されている方法以外のやりかたで再利用したときには，たとえその再利用が開発者個人の行為であったとしても，開発組織に対して責任が問われることになる．組織の方針として，GPL の下で配布されているソースコードからのコピー＆ペーストを発見するツールを運用すれば，このようなリスクをかなりの程度回避できるであろう.

3.4 コードパターン／バグ検出

コードパターンは，前述の定型処理同様，多くのコード断片が同じようなパターンに従った処理になっていることを意味する[†1].

†1 ただし，コードパターンと呼ぶ場合には，定型処理と呼ぶ場合と比較して パターンに当てはまるもの／逸脱するものといった区別の意識が強く，また，デザインパターンに見られるように，あるパターンを構成するコード断片がソースコード中の複数の箇所に分かれて現れるものを指すことが多いようである.

従って，互いにコードクローンである多数のコード断片があったとき，1 つのコード断片だけが，他のコード断片と異なっているなら，あるいは，パターンから逸脱しているなら，そのコード断片はコピー＆ペーストした後，修正を間違えたものかもしれない [53]．そのような少数派のコード断片のいくつかは実際に不具合の原因となっているという研究報告がある [74].

3.5 変化の観察

「変化の観察」はまだ具体的な応用には乏しいが，これからのコードクローン検出技術の応用において重要な位置を占めると期待される分野である [4] [68] [75] [77]．たとえば，以下のような応用が可能になると考えられる.

- プロダクトのソースコードの履歴の各バージョン (リビジョン) に対して，クローン含有率を算出していく．含有率が急激に増える兆しはないか，あるいは，リファクタリング後に含有率が低下しているか確認する [100].
- プロダクト間にコードクローンがあったときに，そのコード断片がコピーされた向きを調べる [26] [67].
- コードクローンになっているコード断片がどのように変更されていくかを調べる [58] [80] [101] [116].

3.6 その他の目的

上述の代表的なもの以外にも， さまざまな目的，および，それらの目的のための手法やツールが提案されている.

- ソースコードの品質を評価することを目的とした，コードクローンの含有量を考慮した規模メトリクスの提案 [88]
- 各々の開発者の作業を調査することを目的とした，開発者のソースファイル毎に コードクローンを検出する研究 [5] [45] [70]
- レガシーコードの書き換えに必要なコストの見積もりを目的とした，コードクローンメトリクス

を利用する研究 [69]

- ソースコードを編集する開発者をサポートすることを目的とした，コードクローン情報をソースコードにコメントとして埋め込むことで可視化する研究 [102]
- 開発者間の共同作業の状況を調査することを目的とした，ソースファイル間にコードクローンがあるものについて同時に変更される頻度を調べた研究 [25]
- ソフトウェアプロダクトの進化を調べることを目的とした，共有するソースコードの含有率を用いてプロダクトの系統樹を作る研究 [117]
- コーディング規約が守られているか調査することを目的とした，類似したコードに含まれる名前のバリエーションを調べる研究 [52] [60]

4 コードクローン検出手法

これまでに，さまざまなコードクローンの検出手法が提案されてきている．これらの検出手法は基本的に以下のステップを順に行う．

1. ソースコードを何らかのモデルで表現し，
2. 必要に応じて正規化を行い，
3. そのモデルにおける，何らかの同型 (あるいは類似) を示す述語 (predicate) によって一致すると判定されるようなコード断片を，
4. 何らかのアルゴリズムによってソースコードから抽出する

ここで，正規化とは，そのモデルにおいて，複数の異なった表現になりえるが，コードクローン検出では同型と見なしたいコード断片を， モデル上で定義された変形操作により同じ表現になるように修正することである．たとえば，文字列リテラル`"abc"`と式`"a" + "b" + "c"`とを同型と見なすため，後者を`"abc"`に変形する (いわゆる「定数たたみ込み」) などである．

以下では，これらの手法を，ソースコードをどのようなモデル (およびそのためのデータ構造) で表現しているかによって分類し，紹介する．ただし，検出手法によっては，複数のデータ構造，およびそれらの構造に適したアルゴリズムを併用することもある．

4.1 文字／行の並び

ソースコードを文字 (または行) の並びで表現し，一定以上の長さで同じ文字 (行) の並びが出現する部分を検出する．検出のためのアルゴリズムとしては接尾辞木 (suffix tree) [6] [48] [104] や接尾辞配列 (suffix array) [1] [118]，頻出アイテムセット抽出 (frequent itemset mining) [113], n-gram [17] などが用いられる．これらのアルゴリズムは自然言語処理で用いられてきたものであり，自然言語処理と同じようなヒューリスティックを利用することができる．たとえば，n-gram であれば，綴りの差異を含むような名前を類似していると判定することができる．また，ソースコード中のコメントもコードクローンの検出対象とするためには，検出手法は自然言語処理を含むことになるが，文字の並びを用いれば，単一のデータ構造でコメント以外の部分とコメントの両方を無理なく表現することが可能になる．

4.2 トークンの並び

いくつかの手法は，ソースコードをトークンの並びで表現する [1] [28] [38] [50] [105] [120]．ここで，トークンとはプログラミング言語で規定されている「単語」である (たとえば，C 言語なら，`int` などの組み込み型，`123` や`"abc"`などのリテラル，`{}[]` などの括弧，演算子などがトークンになる)．文字の並びと比較したときの利点は，(1) フリーフォーマットのプログラミング言語における，意味を変えない空白文字 (スペースや，改行，インデント) に関する正規化が容易なこと，(2) 識別子の名前の長短が検出結果に影響を与えないこと，などである．

4.3 AST

ソースコードを AST(抽象構文木) に変換し，類似する部分木を検出する [12] [16] [22] [46] [71] [73]．ソースコードの制御構造 (ブロック) を意識したコードクローン検出ができる．たとえば，関数定義のブロック全体がコードクローンになっているものだけを検出したいといった要求に応えるのがより容易になる．

AST はコンパイラやコード生成ツールでよく利用されることからも分かるように，正規化やコードク

ローンの除去[9] (コードクローンになっているコード断片を独立したルーチンとして取り出し，元のコード断片を新しいルーチンの呼び出しに書き換える) にも都合のよいデータ構造である．

(コードクローン検出とは少し異なるが，文献[97]では，あるバージョンのあるコード断片が，異なるバージョンではどこに移動したかを調べるために，ASTを使った類似度判定アルゴリズムを利用している．)

4.4 依存関係

ソースコードから PDG(program dependence graph) を取り出し，同型部分グラフを検出するアルゴリズムにより類似した部分グラフを取り出す．ただし，手法によって，制御依存もデータ依存も考慮するもの[23][64][65]，データ依存のみを考慮するもの[24][44][95] など，さまざまな変種がある．この手法の強みは，コード断片がコピー＆ペーストされた後で，文の順序が入れ替えられたり，(依存関係の意味で) 無関係な文が挿入されたような場合でも，PDG としては完全に一致する部分グラフとしてコードクローンを検出できることである．

図 2 のコードクローンは，ソースコードからデータ依存関係のグラフを作成し，同型部分グラフを取り出すことによって検出されたものである．構文解析や上述の定数たたみ込み，正規化により，空白やインデントの差違は吸収され，コメント部分は無視され，リテラル (定数) は値が異なっていても同じ頂点と見なされる．例えば，左カラムの 2871 行および右カラムの 3293 行から 3294 行にある関数 `cFYI()` の呼び出しは，改行やリテラルが異なっているが同型であると見なされている．また，左カラム 2873 行から始まる `memcpy()` の呼び出しは，コード断片のそれ以外の部分とデータ依存関係を持っていなかったため，データ依存関係の部分グラフに含まれていない．従って，左カラムのコード断片と右カラムのコード断片から同型のグラフが抽出され，コードクローンとして検出された．

4.5 モジュールの構造

関数の依存関係 (ファンイン，ファンアウト等) や複雑度メトリクス (サイクロマチック数等) により，関数に相当するコード断片の特徴を数値ベクトル (特徴ベクトルとする) で表現する．それら特徴ベクトルが一致するもの，あるいは，類似したものを何らかのアルゴリズムにより特定し，コードクローンとみなす[62][83]．

ただし，既存の研究には，特徴ベクトルのみでは検出精度がそれほど高くないため，特徴ベクトルで抽出されたものをコードクローンの候補として，さらに，候補間の類似性を DP(dynamic programming) で判定する方法をとるものもある[7]．

4.6 その他

上記以外にも，ソースコードをコンパイルしたバイトコードを解析するもの[18][85]，プログラム実行時のメモリの内容を解析するもの[61] がある．また，プログラムの実行履歴を分析する手法において，実行履歴を圧縮するために類似した履歴を検出し除去する技術が提案されている[109]．このような技術もコードクローンの検出に応用できるであろう．

5 分析／編集の手法

プロダクトのソースコードの規模が大きくなれば，コードクローン検出ツールによって数千，数万のコードクローンが検出されることも珍しくない．そのような場合には，検出結果を分析するためのさらなる手法により，開発者 (あるいは分析者) をサポートすることが必要である．

ここ数年，コードクローンの研究において，コードクローンの新たな応用が登場するにつれて，分析をサポートする手法もめざましく発達してきている．

5.1 散布図

主に，コードクローンがソースコード内のどこに分布しているかを俯瞰するために用いる，マクロなビューである[6][21][108][111]．

散布図 (図 3) の原理は，2 つのソースコード (同じものであってもよい) のトークンを，縦軸と横軸に沿って並べ，縦軸と横軸のトークンが同じであるセルに点を打つ，というものである．図では，2 つのトー

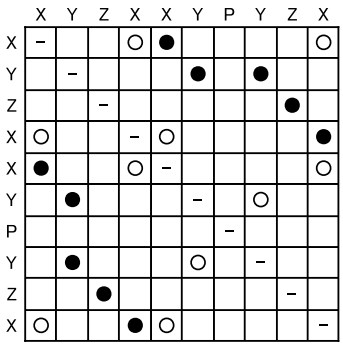

図 3　散布図の概念図．P, X, Y, Z をトークンとして，トークンの並び “XYZXXYPYZX” の中に含まれるコードクローンの位置を示したもの．“-” は縦横のトークンが同じ位置になるセル．“●” および “○” はトークンの並びの中で同じトークンが現れている位置を表す．“●” は 2 つ以上の隣接するトークンが一致している部分，すなわち，トークンで測ったときの長さが 2 以上のコードクローンとなる．

図 4　アラインメントの概念図．入力となるトークンの並びは図 3 と同じもの．2 つ以上の隣接するトークンが一致している部分について対応しているトークンを線で結んでいる．(従って図 3 の “●” のそれぞれについて線が引かれている)

クンの並びがここで，トークンの代わりに文字や行，ファイル，モジュール，プロダクトなど，目的に応じて必要な粒度を指定することで，いろいろな「解像度」の散布図を作ることができる．

たとえば，プロダクト (やモジュール) を粒度とする散布図であるヒートマップ [76] では，点の代わりに色で含まれるコードクローンの密度を示す．縦軸と横軸に並べられたプロダクトの間でコードクローンが多量に含まれるほど，対応するセルが暖色で塗られる．

5.2　テキスト／アラインメント

検出されたコードクローンのコード断片をひとつずつ見ていくためのよりミクロなビューである [14] [21]．GUI ベースの diff ツール等で見られるような表示であり，コードクローンとなった 2 つあるいはそれ以上のコード断片を並べ，一致する部分，不一致部分 (アラインメント) を線やハイライトなどで示す．図 4 は，トークン間の対応としてアラインメントを表示している．図 5 に示すツールでは，2 つのソースファイル間のコードクローンのアラインメントをより大きな単位で，連続して一致しているトークンの並び (図では数行の大きさのコード断片) の対応としてアラインメントを表示している．図の中程にある 2 つの縦線がソースファイル全体を，縦線の間に走っている斜め線がコード断片間の対応としてのアラインメントの表示．それらのコード断片の内容は，図の右側のウィンドウのハイライトされたテキストとして表示されている．図 2 で示したような，不一致部分を含むようなコード断片を分析する際に便利である．

5.3　フィルタリング／ランキング

目的に依存して，検出されて欲しいコードクローンは異なる．たとえば，リファクタリングを行って重複コードを削減することが目的なら，複雑なロジックを含むコード断片ほどより問題が大きいと考えるだろう．あるいは，定型処理に相当するコードクローンを特定することが目的なら，コード断片の長さは短くても頻出するものを見つけたいと思うかもしれない．コードクローンを何らかの方法で分類する，あるいは，順序を付けることにより，優先度を付けてコードクローンに対処することが可能となる [14] [31] [32] [121]．

コードクローンをランキングするためのメトリクスには，コード断片のサイクロマチック数や，コード断片が単調なコードの繰り返しであるかを判定するためのメトリクス (LNR [32])，コードクローンを含むソースファイルに関するメトリクス (RSA, RST [110]) など，さまざまなものが提案されている．

5.4　分類／グループ化

コードクローンを分析する作業をサポートするために，コード断片の出現位置 [2] [34] [51] [56] [57] [84] [98] やコード断片の類似度 [39] [47]，ソースコードのビルドのための設定 (configuration) への依存 [54] 等よってグループ化する手法が提案されている．

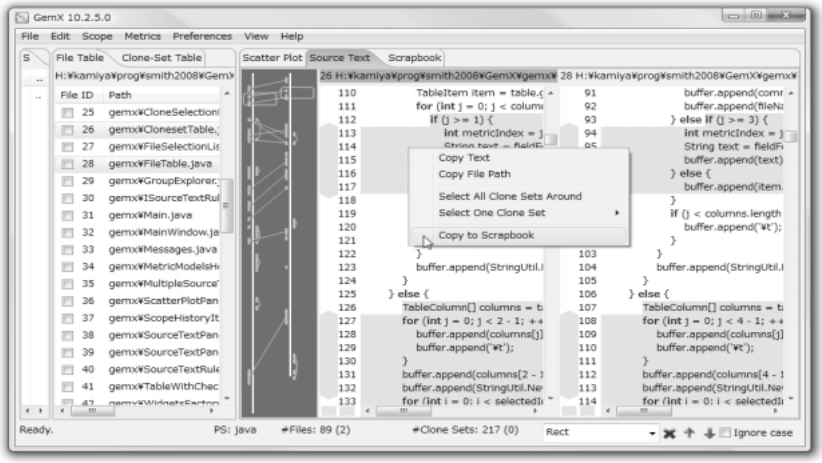

図 5 ツール CCFinderX [14] の GUI フロントエンド．2 つのソースファイル間でコードクローンになっている部分について，その対応が表示されている．

5.5 Clone-Aware IDE

ソースコードの編集に Eclipse などの IDE(統合開発環境) が使われることも多い．IDE は，開発者が編集しているソースコードを解析することで，開発者にヒントを与える (コード補完や構文エラーの指摘など) ことができる．

Clone-Aware IDE は，編集中のソースコードと似たコード断片が開発中のプロダクトに既に存在するなら，それを開発者に報告することで，コードの再利用や設計の修正を促す機能を持つ IDE である [59] [115]．

5.6 同時編集

コードクローンになっているコード断片を同時に編集できるようにすることで，あるいは，コード断片の 1 つに対して行われた修正作業を，そのコード断片のコードクローンとなっているコード断片の全てに対して適用することで，開発者の編集作業をサポートしようとするアプローチである [20] [37]．

同時編集を実現するツールは，コードクローン検出技術によって重複したコード断片を (後から) 集める手法以外にも，開発者を見張り，コピー&ペースト作業から直接コードクローン情報を生成する手法を用いる．

6 研究コミュニティ

コードクローンについての研究は，現状では，国内より国外で盛んである．従って，最新の研究が国際会議で発表されることも多い．

6.1 会議・ワークショップ

IWSC(International Workshop on Software Clones) は，コードクローンを主要なトピックとする国際ワークショップであり，2010 年度，2011 年度は IEEE ICSE(International Conference on Software Engineering) に併設して開催された (される)．(ワークショップの名称に「ソフトウェア・クローン」とあるのは，対象をソースコードだけではなく，より広範なドキュメントにするため．)

国内の研究会で，あるいは，国際会議 IEEE ICSE (International Conference on Software Engineering), IEEE ICSM (International Conference on Software Maintenance), IEEE CSMR (European Conference on Software Maintenance and Reengineering), MSR (Working Conference on Mining Software Repositories, ただし 2006 年までは International Workshop on Mining Software Repositories), WCRE (Working Conference on Reverse Engineering) などでコードクローンに関する手法や実験が発

表されている (参考文献を参照されたい).

他にも，2006 年のソフトウェアの重複・冗長・類似性に関する Dagstuhl セミナー[63] や，2009 年のソースコードの類似性ワークショップ[41] など，不定期ながら，コードクローンについて集中的に議論する場が設けられてきた.

6.2 近年注目を集めているトピック

新しい技術として，関数型プログラミング言語向け[72]，あるいは，データフロー言語向け[19][29][91]，UML のクラス図を対象とした[107] コードクローン検出技術の研究も始まっている.

近年，コードクローンの存在が保守性に悪い影響を与えるかについての議論が盛んになっている．これを実験的に検証しようとする研究では，コードクローンになっているコード断片とそれ以外のコード断片との修正の頻度を比較する手法が主流になっている．実験の結果は研究により以下のようにまちまちである.

- コードクローンの方が頻繁に修正される[78][103]
- 修正の頻度にそれほど差はない[36]
- コードクローンではない方が頻繁に修正される[66][96]
- コードクローンのコード断片が長いものは頻繁に修正される[86][87]
- コードクローンのコード断片の一方が修正されると，修正のコストが増大する[49][79]

コードクローンの検出を分散処理で行う方法も研究されている．超大規模なソースコードを対象としてコードクローンを検出するものや[76]，開発者の手元にあるソースコードとレポジトリ内のソースコードと間のコードクローンを素早く (コンマ数秒で) 検出するもの[38][73] がある.

コードクローン検出手法・ツールを比較する実験も，継続的に行われている[10][13][15][99][112][114].

7 まとめ

本稿では，コードクローンに関するさまざまな技術やその応用について説明した.

コードクローンに興味を持たれた方は，本稿に併せて文献[33] も読まれたい．本稿では紙面の都合で説明していない以下のトピックについてわかりやすく説明されている.

- クローンペア，クローンクラス (セット)
- クローンタイプ，ギャップトクローン
- 検出手法・ツールの実験的な比較

謝辞 本研究は，日本学術振興会 科学研究費補助金 (挑戦的萌芽研究) (研究課題番号: 21650008) の助成を受けたものである．本研究は一部，日本学術振興会科学研究費補助金 研究活動スタート支援 (課題番号:22800040) の助成を得た.

参 考 文 献

[1] Basit, H. A., Puglisi, S. J., Smyth, W. F., Turpin, A. and Jarzabek, S.: Efficient Token Based Clone Detection with Flexible Tokenization, in *Proc. 6th Joint Meeting on European Softw. Eng. Conf. and the ACM SIGSOFT Symposium on the Foundations of Softw. Eng.* (*ESEC/FSE 2007*), 2007, pp. 513–516.

[2] Adar, E. and Kim, M.: SoftGUESS: Visualization and Exploration of Code Clones in Context, in *ICSE 2007*, 2007, pp. 762–766.

[3] Al-Ekram, R., Kapser, C. and Godfrey, M.: Cloning by Accident: An Empirical Study of Source Code Cloning Across Software Systems, in *Proc. 4th Int'l Symposium on Empirical Softw. Eng.* (*ISESE 2005*), 2005, pp. 376–385.

[4] Antoniol, G., Villano, U., Merlo, E. and DiPenta, M.: Analyzing Cloning Evolution in the Linux Kernel, *Information and Softw. Technology*, Vol. 44, No. 13 (2002), pp. 755–765.

[5] 東, 肥後, 早瀬, 松下, 井上: コードクローンの複雑度メトリクスを用いた開発者の特徴分析, ソフトウェアエンジニアリング最前線 *2008*, 近代科学社, 2008, pp. 103–106.

[6] Baker, B. S.: On finding Duplication and Near-Duplication in Large Software System, in *WCRE '95*, 1995, pp. 86–95.

[7] Balazinska, M., Merlo, E., Dagenais, M., Lagüe, B. and Kontogiannis:, K. A.: Measuring Clone Based Reengineering Opportunities, in *Proc. 6th IEEE Int'l Symposium on Softw. Metrics* (*METRICS '99*), 1999, pp. 292–303.

[8] Black Duck Software, http://www.blackducksoftware.com/.

[9] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M. and Bier, L.: Clone Detection Using Abstract Syntax Trees, in *IEEE ICSM '98*, 1998, pp. 368–377.

[10] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and Evaluation of

Clone Detection Tools, *IEEE TSE*, Vol. 33, No. 9 (2007), pp. 577–591.
[11] Bruntink, M., van Engelen, R. and Tourwé, T.: On the Use of Clone Detection for Identifying Crosscutting Concern Code, *IEEE TSE*, Vol. 31, No. 10 (2005), pp. 804–818.
[12] Bulychev, P. and Minea, M.: Duplicate Code Detection using Anti-Unification, in *Spring/Summer Young Researchers' Colloquium on Software Eng.* (*CYRCoSE*) *2008*, taken from http://clonedigger.sourceforge.net/duplicate_code_ detection_bulychev_minea.pdf (2010/12/28).
[13] Burd, E. and Bailey, J.: Evaluating Clone Detection Tools for Use during Preventative Maintenance, in *Proc. 2nd IEEE Int'l Workshop on Source Code Analysis and Manipulation* (*SCAM 2002*), 2002, pp. 36–43.
[14] CCFinderX, http://www.ccfinder.net/.
[15] Cheng, H. and Mockus, A.: Evaluation of Source Code Copy Detection Methods on FreeBSD, in *MSR 2008*, 2008, pp. 61–65.
[16] Chilowicz, M., Duris, E. and Roussel, G.: Syntax Tree Fingerprinting for Source Code Similarity Detection, in *Proc. 17th Int'l Conf. Program Comprehension*, 2009, pp. 243–247.
[17] CodeDepot, http://www.sra.co.jp/codedepot/.
[18] Davis, I. J. and Godfrey, M. W.: Detecting Source Code Clones by Analyzing Assembler, in *WCRE 2010*, 2010, pp. 242–246.
[19] Deissenboeck, F., Hummel, B. and Schaetz, B.: Model Clone Detection in Practice, in *IWSC 2010*, 2010, pp. 57–64.
[20] Duala-Ekoko, E. and Robillard, M. P.: Clone Region Descriptors: Representing and Tracking Duplication in Source Code, *ACM Transactions on Software Engineering and Methodology*, Vol. 20 (2010), pp. 1–31.
[21] Ducasse, S., Rieger, M. and Demeyer, S.: A Language Independent Approach for Detecting Duplicated Code, in *ICSM '99*, 1999, pp. 109–118.
[22] Evans, W. S., Fraser, C. W. and Ma, F.: Clone Detection via Structural Abstraction, in *WCRE 2007*, 2007, pp. 150–159.
[23] Gabel, M., Jiang, L. and Su, Z.: Scalable Detection of Semantic Clones, in *ICSE 2008*, 2008, pp. 321–330.
[24] Gallagher, K. and Layman, L.: Are Decomposition Slices Clones?, in *Proc. 11th IEEE Int'l Workshop on Program Comprehension* (*IWPC 2003*), 2003, pp. 251–256.
[25] Geiger, R., Fluri, B., Gall, H. and Pinzger, M.: Relation of Code Clones and Change Couplings, in *FASE 2006 Proc.*, NCLS 3922, Springer, 2006, pp. 411–425.
[26] German, D. M., Di Penta, M., Gueheneuc, Y. and Antoniol, G.: Code Siblings: Technical and Legal Implications, in *MSR 2009*, 2009, pp. 81–90.
[27] GNU General Public License, http:// www.gnu.org/ licenses/ gpl.html
[28] Göde, N. and Koschke, R.: Incremental Clone Detection, in *CSMR 2009*, 2009, pp. 219–228.
[29] Gold, N., Krinke, J., Harman, M. and Binkley, D.: Issues in Clone Classification for Dataflow Languages, in *IWSC 2010*, 2010, pp. 83-84.
[30] 服部, 吉田, 早瀬, 肥後, 松下, 楠本, 井上: 識別子の共起関係に基づく類似コード検索法の提案と欠陥検出への適用, 信学技報, SS2007-47, Vol. 107, No. 392 (2007), pp. 55–60.
[31] Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: ARIES: Refactoring Support Environment Based on Code Clone Analysis, in *Proc. 8th IASTED Int'l Conf. Softw. Eng. and Applications* (*SEA 2004*), 2004, pp. 222–229.
[32] 肥後, 神谷, 楠本, 井上: リファクタリングのための類似コード片抽出ツール Gemini, 情報処理学会組み込みソフトウェアシンポジウム 2004 論文集, IPSJ Symposium Series Vol. 2004, No. 10, 2004, pp. 142–143.
[33] 肥後, 楠本, 井上: 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. J91-D, No. 6 (2008), pp. 1465–1481.
[34] 肥後, 宮崎, 楠本, 井上: グラフマイニングアルゴリズムを用いたギャップを含むコードクローン情報の生成, 電子情報通信学会論文誌 D, Vol. J93-D, No. 9 (2010), pp. 1727–1735.
[35] 肥後, 植田, 神谷, 楠本, 井上: コードクローン解析に基づくリファクタリングの試み, 情報処理学会論文誌, No. 45, Vol. 5 (2004), pp. 1357–1366.
[36] Hotta, K., Sano, Y., Higo, Y. and Kusumoto, S.: Is Duplicate Code More Frequently Modified Than Non-Duplicate Code in Software Evolution?: an Empirical Study on Open Source Software, in *Proc. Joint ERCIM Workshop on Softw. Evolution* (*EVOL*) *and Int'l Workshop on Principles of Softw. Evolution* (*IWPSE 2010*), 2010, pp. 73–82.
[37] Hou, D., Jablonski, P. and Jacob, F.: CnP: Towards an Environment for the Proactive Management of Copy-And-Paste Programming, in *Proc. IEEE 17th Int'l Conf. Program Comprehension* (*ICPC 2009*), 2009, pp. 238–242.
[38] Hummel, B., Juergens, E., Heinemann, L. and Conradt, M.: Index-Based Code Clone Detection: Incremental, Distributed, Scalable, in *ICSM 2010*, 2010, pp. 1–9.
[39] Imai, T., Kataoka, Y. and Fukaya, T.: Evaluating Software Maintenance Cost Using Functional Redundancy Metrics, in *Proc. 26th Int'l Computer Softw. and Applications Conf.* (*COMPSAC 2002*), 2002, pp. 299–306.
[40] Ishio, T., Date, H., Miyake, T. and Inoue, K.: Mining Coding Patterns to Detect Crosscutting Concerns in Java Programs, in *WCRE 2008*, 2008, pp. 123–132.
[41] 石尾, 山本, 佐々木: ソースコードの類似性ワークショップ開催報告, 情報処理学会研究報告, Vol. 2009-SE-166, No. 20 (2009), pp. 1–8.
[42] 石尾, 伊達, 三宅, 井上: シーケンシャルパターンマ

イニングを用いた コーディングパターン抽出, 情報処理学会論文誌, Vol. 50, No. 2 (2009), pp. 860–871.
[43] 泉田, 植田, 神谷, 楠本, 井上: ソフトウェア保守のための類似コード検索ツール, 電子情報通信学会論文誌, Vol. J86-D-I, No. 12 (2003), pp. 906–908.
[44] Jia, Y., Binkley, D., Harman, M., Krinke, J. and Matsushita, M.: KClone: A Proposed Approach to Fast Precise Code Clone Detection, in *IWSC 2009. Workshop Proc. CSMR 2009*, 2009, pp. 12–16.
[45] Jiang, M. and Zhang, J.: Maintaining Software Product Lines, an Industrial Practice, in *ICSM 2008*, 2008, pp. 444–447.
[46] Jiang, L., Misherghi, G., Su, Z. and Glondu, S.: DECKARD: Scalable and Accurate Tree-based Detection of Code Clones, in *ICSE 2007*, 2007, pp. 96–105.
[47] Johnson, J. H.: Visualizing Textual Redundancy in Legacy Source, in *Proc. 1994 Conf. of the Centre for Advanced Studies on Collaborative research* (*CASCON '94*), IBM Press, 1994, pp. 9–18.
[48] Juergens, E., Deissenboeck, F. and Hummel, B.: CloneDetective - A workbench for clone detection research, in *ICSE 2009*, 2009, pp. 603–606.
[49] Juergens, E., Deissenboeck, F., Hummel, B. and Wagner, S.: Do Code Clones Matter?, in *ICSE 2009*, 2009, pp. 485–495.
[50] 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 (2002), pp. 654–670.
[51] 神谷 : コードの入れ子関係を用いたコードクローンのグループ化手法の提案, 信学技報, SS2007-46, Vol. 107, No. 392 (2007), pp. 49–54.
[52] Kamiya, T.: Variation Analysis of Context-Sharing Identifiers with Code Clone, in *ICSM 2008*, 2008, pp. 464–465.
[53] 神谷: コードクローンをテンプレートして用いることにより識別子のバリエーションを分析する手法の提案, 信学技報, SS2008-1〜12 (2008), pp. 59–64.
[54] Kamiya, T.: Classifying Code Clones with Configuration, in *IWSC 2010*, 2010, pp. 75–76.
[55] 鹿島, 早瀬, 吉田, 真鍋, 井上 : ソフトウェアライセンスがコピーアンドペーストによる再利用に与える影響の調査, 信学技報, Vol. 110, No. 227, SS2010-32 (2010), pp. 25–30.
[56] Kapser, C. J. and Godfrey, M. W.: Supporting the Analysis of Clones in Software Systems: A Case Study, *Softw., J.,Maintenance and Evolution: Research and Practice*, Vol. 18, issue 2 (2005), pp. 61–82.
[57] Kapser, C. and Godfrey, M. W.: Improved Tool Support for the Investigation of Duplication in Software, in *ICSM 2005*, 2005, pp. 305–314.
[58] 川口, 松下, 井上, 飯田: コードクローン履歴閲覧環境を用いたクローン評価の試み, 情報処理学会第 154 回ソフトウェア工学研究会, 2006, pp. 49–56.
[59] Kawaguchi, S., Yamashina, T., Uwano, H., Fushida, K., Kamei, Y., Nagura, M. and Iida, H.: SHINOBI: A Tool for Automatic Code Clone Detection in the IDE, in *WCRE 2009*, 2009, pp. 313–314.
[60] Kim, S., Pan, K. and Whitehead, Jr., E. J.: When Functions Change Their Names: Automatic Detection of Origin Relationships, in *WCRE 2005*, 2005, pp. 143–152.
[61] Kim, H., Jung, Y., Kim, S. and Yi, K.: MeCC: Memory Comparision-based Clone Detector, in *ICSE 2011*, 2011, pp. 301–310.
[62] Kontogiannis, K., DeMori, R., Merlo, E., Galler, M. and Bernstein, M.: Pattern Matching Techniques for Clone Detection, *Automated, J.,Softw. Eng.*, Kluwer Academic Publishers, Vol. 3, 1996, pp. 77–108.
[63] Koschke, R., Merlo, E. and Walenstein, A.(Eds.): *Duplication, Redundancy, and Similarity in Software*, Dagstuhl Seminar Proc. 06301, 2007.
[64] Komondoor, R. and Horwitz, S.: Using Slicing to Identify Duplication in Source Code, in *Proc. 8th Int'l Symposium on Static Analysis* (*SAS 2001*), LNCS 2126, 2001, pp. 40–56.
[65] Krinke, J.: Identifying Similar Code with Program Dependence Graphs, in *WCRE 2001*, 2001, pp. 301–309.
[66] Krinke, J.: Is Cloned Code More Stable than Non-cloned Code?, in *Proc. 8th IEEE Int'l Working Conf. Source Code Analysis and Manipulation* (*SCAM 2008*), 2008, pp. 57–66.
[67] Krinke, J., Gold, N., Jia, Y. and Binkley, D.: Cloning and Copying between GNOME Projects, in *MSR 2010*, 2010, pp. 98–101.
[68] Lagüe, B., Merlo, E. M., Mayrand, J. and Hudepohl, J.: Assessing the Benefits of Incorporating Function Clone Detection in a Development Process, in *ICSM '97*, 1997, pp. 314–321.
[69] Lajios, G., Schmedding, D. and Volmering, F.: Supporting Language Conversion by Metric Based Reports, in *CSMR 2008*, 2008, pp. 314–316.
[70] LaToza, T. D., Venolia, G. and DeLine, R.: Maintaining Mental Models: A Study of Developer Work Habits, in *ICSE 2006*, 2006, pp. 492–501.
[71] Lee, H. and Doh, K.: Tree-Pattern-based Duplicate Code Detection, in *Proc. Int'l Workshop on Data-intensive Software Management and Mining* (*DSMM*), 2009, pp. 7–12.
[72] Li, H. and Thompson, S.: Similar Code Detection and Elimination for Erlang Programs, in *Practical Aspects of Declarative languages 2010* (*PADL 2010*), LNCS 5937, Springer-Verlag, 2010, pp. 104–118.
[73] Lee, M., Roh, J., Hwang, S. and Kim, S.: Instant Code Clone Search, in *Proc. 18th ACM/SIGSOFT Int'l Symposium on the Foundations of Softw. Eng.* (*FSE 2010*), 2010, pp. 167–176.
[74] Li, Z., Lu, S., Myagmar, S. and Zhou, Y.: CP-Miner: Finding Copy-Paste and Related Bugs

in Large-Scale Software Code, *IEEE TSE*, Vol. 32, No. 3 (2006), pp. 176–192.
[75] Livieri, S., Higo, Y., Matsushita, M. and Inoue, K.: Analysis of the Linux Kernel Evolution Using Code Clone Coverage, in *MSR 2007*, 2007, pp. 22-1–22-4.
[76] Livieri, S., Higo, Y., Matsushita, M. and Inoue, K.: Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder, in *ICSE 2007*, 2007, pp. 106–115.
[77] リビエリ, 肥後, 松下, 井上: コードクローン検出技術を用いた Linux カーネル進化の調査, 電子情報通信学会論文誌 D-*I*, Vol. J91-D, No. 2 (2008), pp. 509–511.
[78] Lozano, A., Wermelinger, M. and Nuseibeh, B.: Evaluating the Harmfulness of Cloning: a Change Based Experiment, in *MSR 2007*, 2007, pp. 19–20.
[79] Lozano, A. and Wermelinger, M.: Assessing the Effect of Clones on Changeability, in *ICSM 2008*, 2008, pp. 227–236.
[80] Lozano, A. and Wermelinger, M.: Tracking clones' imprint, in *IWSC 2010*, 2010, pp. 65–72.
[81] マーチン, R., ニューカーク, J. W., コス, R. S. (瀬谷訳): アジャイルソフトウェア開発の奥義 — 原則・デザインパターン・プラクティス完全統合, 初版, 2004, ソフトバンクパブリッシング, p. 20. Martin, R. C.: *Agile Software Development, Principles, Patterns, and Practices*, Prentice Hall, 2002.
[82] 政井, 吉田, 松下, 井上: 類似メソッドの集約のための差分抽出支援, 信学技報, SS2010-8, Vol. 110, No. 60 (2010), pp. 45–50.
[83] Mayland, K. J., Leblanc, C. and Merlo, E. M.: Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics, in *ICSM '96*, 1996, pp. 244–253.
[84] 宮崎, 肥後, 井上: アイテムセットマイニングを利用したコードクローン分析作業の効率向上, 信学技報, SS2008-13〜26, Vol. 108, No. 173 (2008), pp. 31–36.
[85] Molina, A. J. M. and Shinohara, T.: On Approximate Matching of Programs for Protecting Libre Software, in *Proc. IBM Conf. Center for Advanced Studies on Collaborative Research* (*CASCON 2006*), 2006, pp. 275–289.
[86] Monden, A., Nakae, D., Kamiya, T., Sato, S. and Matsumoto, K.: Software Quality Analysis by Code Clones in Industrial Legacy Software, in *Proc. 8th IEEE Symposium on Softw. Metrics* (*METRICS 2002*), 2002, pp. 87–94.
[87] 門田, 佐藤, 神谷, 松本: コードクローンに基づくレガシーソフトウェアの品質の分析, 情報処理学会論文誌, Vol. 44, No. 8 (2003), pp. 2178–2188.
[88] Monden, A., Uchida, S. and Matsumoto, K.: On Building a Better Program Size Measure, in *Proc. Int'l Conf. Softw. Process and Product Measurement* (*MENSURA 2010*), 2010, pp. 305–314.
[89] 森崎, 吉田, 肥後, 楠本, 井上, 佐々木, 村上, 松井: 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, 電子情報通信学会論文誌 D, Vol. J91-D, No. 10 (2008), pp. 2466–2477.
[90] Moss, http://theory.stanford.edu/~aiken/moss/.
[91] Nguyen, H. A., Nguyen, T. T., Pham, N. H., Al-Kofahi, J. M. and Nguyen, T. N.: Accurate and Efficient Structural Characteristic Feature Extraction for Clone Detection, in *Proc. 12th Int'l Conf. Fundamental Approaches to Softw. Eng.: Held as Part of the Joint European Conferences Theory and Practice of Softw., ETAPS 2009*, LNCS 5503, Springer-Verlag, 2009, pp. 440–455.
[92] 岡原, 真鍋, 山内, 門田, 松本: ソースコード流用のコードクローンメトリクスに基づく検出手法, 信学技報, Vol. 109, No. 307, KBSE2009-43 (2009), pp. 73–78.
[93] Palamida, http://www.palamida.com/.
[94] Prechelt, L., Malpohl, G. and Phlippsen, M.: Finding Plagiarisms among a Set of Programs with JPlag, *Universal, J., Computer Science*, Vol. 8, No. 11 (2002), pp. 1016–1038.
[95] Quartet, http://www.remics.org/.
[96] Rahman, F., Bird, C. and Devanbu, P.: Clones: What is that smell?, in *MSR 2010*, 2010, pp. 72–81.
[97] Reiss, S. P.: Tracking Source Locations, in *ICSE 2008*, 2008, pp. 11–20.
[98] Rieger, M., Ducasse, S. and Lanza, M.: Insights into System-Wide Code Duplication, in *WCRE 2004*, 2004, pp. 100–109.
[99] Roy, C. K., Cordy, J. R. and Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, *Science of Computer Programming*, Vol. 74, issue 7 (2008), pp. 470–495.
[100] Rysselberghe, F. V. and Demeyer, S.: Mining Version Control Systems for FACs (frequently applied changes), in *MSR 2004*, 2004, pp. 48–52.
[101] 齋藤, 吉田, 松下, 井上: コードの生存期間を考慮したコードクローンと欠陥修正の関係調査, 信学技報, Vol. 110, No. 227, SS2010-28 - SS2010-38 (2010), pp. 19–24.
[102] 佐々木, 肥後, 神谷, 楠本, 井上: プログラム変更支援を目的としたコードクローン情報付加ツールの実装と評価, 電子情報通信学会論文誌 D-I, Vol. J87-D-I, No. 9 (2004), pp. 868–870.
[103] 佐藤, 亀井, 上野, 門田, 川口, 名倉, 松本, 飯田: コードクローンの長さとソフトウェア信頼性の関係の分析, 信学技報, Vol. 108, No. 242 (2008), pp. 43–48.
[104] Selim, G. M. K., Foo, K. C. and Zou, Y.: Enhancing Source-Based Clone Detection Using Intermediate Representation, in *WCRE 2010*, 2010, pp. 227–236.
[105] Smith, R. and Horwitz, S.: Detecting and Measuring Similarity in Code Clones, in *IWSC 2009, Workshop Proc. CSMR 2009*, 2009, pp. 28–34.
[106] Spinellis, D.: A Tale of Four Kernels, in *ICSE 2008*, 2008, pp. 381–390.
[107] Störrle, H.: Towards clone detection in UML domain models, in *Proc. 4th European Conf. Software Architecture* (*ECSA 2010*), 2010, pp. 285–293.

[108] Tairas, R., Gray, J. and Baxter, I.: Visualization of Clone Detection Results, in *Proc. 2006 OOPSLA Workshop on Eclipse Technology Exchange*, 2006, pp. 50–54.

[109] 谷口, 石尾, 神谷, 楠本, 井上: プログラム実行履歴からの簡潔なシーケンス図の生成手法, コンピュータソフトウェア, Vol. 24, No. 3 (2007), pp. 153–169.

[110] 植田, 神谷, 楠本, 井上: 開発保守支援を目指したコードクローン分析環境, 電子情報通信学会論文誌 D-I, Vol. J86-D-I, No. 12 (2003), pp. 863–871.

[111] Uchida, S., Kamiya, T., Monden, A., Matsumoto, K., Ohsugi, N. and Kudo, H.: Software Analysis by Code Clones in Open Source Software, *Computer, J.,Information Systems*, Vol. XLV, No. 3 (2005), pp. 1–11.

[112] Van Rysselberghe, F. and Demeyer, S.: Evaluating Clone Detection Techniques from a Refactoring Perspective, in *Proc. 19th IEEE Int'l Conf. Automated Softw. Eng.* (*ASE 2004*), 2004, pp. 336–339.

[113] Wahler, V., Seipel, D., Wolff, J., Gudenberg, V. and Fischer, G.: Clone Detection in Source Code by Frequent Itemset Techniques, in *Proc. 4th IEEE Int'l Workshop on Source Code Analysis and Manipulation* (*SCAM 2004*), 2004, pp. 128–135.

[114] Walenstein, A., Jyoti, N., Li, J., Yang, Y. and Lakhotia, A.: Problems Creating Task-relevant Clone Detection Reference Data, in *WCRE 2003*, 2003, pp. 285–294.

[115] 渡邉, 増原: 類似プログラムの提示ツール Selene, 日本ソフトウェア科学会第 24 回大会 (JSSST2007) 講演論文集, 3B-2, 2007.

[116] Weisgerber, P. and Diehl, S.: Identifying Refactorings from Source-Code Changes, in *Proc. 21st IEEE Int'l Conf. Automated Softw. Eng.* (*ASE 2006*), 2006, pp. 231–240.

[117] 山本, 松下, 神谷, 井上: クローン検出ツールを用いたソフトウェアシステムの類似度調査, 信学技報, SS2001-15, Vol. 101, No. 240 (2001), pp. 25–32.

[118] 山科, 上野, 伏田, 亀井, 名倉, 川口, 飯田: コードクローンに着目したソフトウェア保守支援ツールの設計と実装, 信学技報, SS2008-1〜12, Vol. 108, No. 64 (2008), pp. 65–70.

[119] Yoshida, N., Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: On Refactoring Support Based on Code Clone Dependency Relation, in *Proc. 11th IEEE Int'l Softw. Metrics Symposium*, 2005, pp. 10–16.

[120] 吉田, 服部, 早瀬, 井上: 類義語の特定に基づく類似コード検索法, 情報処理学会論文誌, Vol. 50, No. 5 (2009), pp. 1506–1519.

[121] Zhang, Y., Abdul Basit, H., Jarzabek, S., Anh, D. and Low, M.: Query-based Filtering and Graphical View Generation for Clone Analysis, in *ICSM 2008*, 2008, pp. 376–385.