

生成抑止, 分析効率化, 不具合検出を中心とした コードクローン管理支援技術に関する研究動向

堀田 圭佑 肥後 芳樹 楠本 真二

ソフトウェアの大規模化, 複雑化に伴い, ソフトウェア保守は困難な作業となっている。ソフトウェア保守を困難にする要因の1つとしてコードクローンの存在が指摘されており, コードクローンに関する研究が盛んに行われている。しかし, コードクローンの主な生成要因であるコピーアンドペーストによる既存コードの再利用には, 既存の機能と類似する機能を高速に実現できるという利点がある。また近年の調査の結果, すべてのコードクローンが必ずしもソフトウェア保守に悪影響を与えているとは限らないという報告がなされている。このような背景から, コードクローンを管理する技術が求められているといえる。本稿ではこれまでに報告されている研究成果の中から, コードクローンの管理を行う上で有用な研究成果を紹介する。

Software maintenance tasks have become more and more difficult because of the massive scale and complexity of software systems. Code clones are regarded as one of the factors that makes software maintenance more difficult, which makes code clones as a hot research topic on the research area of software engineering. However, the basis that clones are harmful for software maintenance is countered by some opinions. One of them is that code reuse by copy-and-paste operations, which is regarded as a major cause of creation of clones, can promote rapid engineering of software systems. Another opinion is that some research results revealed that not all clones are harmful for software maintenance. These opinions show a high motivation for clone management. This paper presents a survey on research of code clones that is useful for effective clone maintenance.

1 まえがき

ソフトウェア保守は高いコストを要する困難な作業であり, ソフトウェア開発に要する資源の多くが保守作業に費やされている[74]。したがって, ソフトウェア保守をより効率的に行い, 要するコストを削減するために, ソフトウェア保守作業を支援する技術が必要とされている。

ソフトウェア保守を困難にするおそれのある要因の1つとして, ソースコード中に存在する重複コード

(Duplicated Code) の存在が指摘されている[11]。重複コードはコードクローン (Code Clones) とも呼ばれており, 主にコピーアンドペーストによる既存コードの再利用によってソースコード中に作り込まれるといわれている[36]。

コードクローンの存在がソフトウェア保守を困難にしている根拠として, 以下のような点が挙げられる。

- あるコード片に不具合が潜在している場合, そのコード片とコードクローンにある他のコード片にも同様の不具合が潜在している可能性が高い。
- あるコード片に修正を加える必要が生じたとき, そのコード片とコードクローン関係にある他のコード片についても同様の修正を検討する必要があるため, 変更の局所性が低下する。
- あるコード片に対する修正と同様の修正を他のコード片に行う必要があるにも関わらず, その修正を施し忘れた場合, 不具合が混入もしくは残存

A Survey on Code Clone Management Focusing on Prevention, Methodology for Efficient Analysis, and Bug Detection.

Keisuke Hotta, Yoshiki Higo, Shinji Kusumoto, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University. コンピュータソフトウェア, Vol.31, No.1 (2014), pp.14–29. [解説論文] 2013 年 4 月 9 日受付。

する。

コードクローンをソースコード中から検出し、ソースコード中のコードクローンを把握すること、あるいは必要に応じてそれらを除去することで、コードクローンによってもたらされるこれらの弊害を低減することが可能になり、ソフトウェア保守の効率化が期待できる。このような背景から、コードクローンに関する研究が盛んに行われている [19][22][27][36][39][45][65][86]。Zibran らが行った系統的レビューによると、2011 年にはコードクローンに関する文献が 92 名の著者によって約 50 本発表されており、コードクローンに対する注目度の高さを窺い知ることができる [86]。

コードクローンの存在はソフトウェア保守を困難にするとされているが、一方でコードクローンの主たる生成要因とされるコピーアンドペーストによる既存コードの再利用には、すでに実装されている機能と類似する機能を低コストで実現できるという利点がある。このため、コピーアンドペーストによる再利用はソフトウェア開発において頻繁に行われている [84]。限られた資源で開発を進めなければならないソフトウェア開発の現状を踏まえると、コピーアンドペーストによる既存コードの再利用を禁止することは現実的に困難である。

また近年、‘コードクローンの存在がソフトウェア保守に悪影響を与える’という前提が本当に成り立っているのかを定量的に調査する研究が多く研究者によって行われている。それらの調査報告によって得られた知見にはばらつきがあり、統一された結論は得られていない。しかしながら、それらを総合すれば、ソフトウェア保守に悪影響を与えるコードクローンは存在するが、すべてのコードクローンがそうであるとはいえず、ソフトウェア保守に悪影響を与えないコードクローンも少なからず存在する、といえる [37]。

これらのことから、すべてのコードクローンについて生成を防止する、あるいは除去することは、効率的なソフトウェアの開発、保守を目指すうえで適切な方法であるとは言い難い [75]。費やす労力をなるべく小さくしつつ、コードクローンによるリスクを最小限にとどめることが、より効率的なソフトウェアの開発、保守を実現する上で重要である。

コードクローンによってもたらされるリスクを効率的に抑えるために、コードクローンを管理する技術の必要性が高まっている。本稿では、コードクローン管理支援技術に関する近年の研究成果を俯瞰することを目指す。本稿ではコードクローン管理支援技術をその目的に応じて以下のように分類する。

- 検出
- 除去
- 生成、拡散の抑止
- 分析の効率化
- 不具合の混入防止、および検出

既存のコードクローンに関するサーベイ論文はコードクローンの検出、及び除去に関するトピックを中心に扱っている [19][22][27][36][39][45][65][86]。本稿は上述の分類のうち、既存のサーベイ論文で重点的に取り扱われていない、“生成、拡散の抑止”、“分析の効率化”、“不具合の混入防止、及び検出”に関する研究について紹介する。

以降の本稿の構成は以下の通りである。2 章ではコードクローンとは何か、及びコードクローンに関してどのような議論がこれまでに行われているのかについて述べる。3 章、4 章、5 章の各章でそれぞれの分類ごとに研究成果を紹介する。6 章で今後のコードクローン管理の展望について述べ、最後に 7 章で本稿をまとめる。

2 コードクローン

2.1 定義

コードクローンとは、ソースコード中に存在する同一の、あるいは類似したコード片のことを指す [7][27]。ただし、2 つのコード片が何をもって類似しているときとみなされるのか、という点に関しては厳密な定義は与えられていない。すなわち、コードクローンに関する厳密かつ普遍的な定義は存在しない。したがって、コードクローン検出手法の提案者はそれぞれに独自の定義を用いているのが実情である [19][27][45][65]。

なお、コード片の類似関係はその定義によっては推移律を満たさない可能性がある。すなわち、2 つのコード片 A 、 B が類似関係にあり、またコード片 B 、 C も類似関係にある場合であっても、2 つのコード片

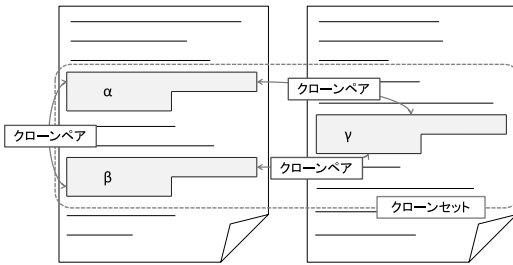


図1 クローンペアとクローンセット

A, C が類似関係にあるとは限らない。

2つのコード片 α , β が互いに類似しているとき, α と β は互いにクローン関係にあるといい, ペア (α, β) をクローンペアと呼ぶ。また, コードクローンを形成しているコード片のことをクローン片と呼ぶ。さらに, 互いにクローン関係にあるコード片の集合をクローンセットと呼ぶ[27]。なお, クローンセットには, クローンクラス, クロングループなどの同じ意味の呼称が他にも存在するが, 本稿ではクローンセットという呼称を用いる。図1の例では, 3つのクローンペア (α, β), (α, γ), (β, γ) が存在し, 3つのクローン片 α , β , γ が1つのクローンセットを形成している。

2.2 コードクローンはなぜ生成されるのか

コードクローンがソースコード中に作り込まれる要因として, 以下のものが挙げられる[7]。

コピーアンドペーストによるコードの複製: すでに実装されている機能をコピーアンドペーストによって再利用した場合, 似たような記述がソースコード中の複数箇所に存在することにつながる。

コーディングスタイル・定型処理: エラー出力やユーザインターフェース処理などのある一定のスタイルに従って記述する必要のある処理は, しばしば似たような記述になることがある。また, キューへの挿入処理やデータ構造へのアクセス, 所得税の計算などの, 比較的簡単かつ頻繁に用いられる処理も同様にソースコード中に繰り返し出現する可能性がある。

適切な言語機能の欠如: 抽象データ型がサポートされていない言語を用いて開発を行っている場

合, 同じようなアルゴリズムを持つ処理を繰り返し記述する必要が生じる可能性がある。

パフォーマンス改善: 関数呼び出しなどによって生じるオーバーヘッドを低減することを目的に, 繰り返し出現する処理を抽象化せずにあえて繰り返し記述することがある。

偶然: コード片が偶然一致もしくは類似した場合, 意図的ではないが結果的にコードクローンとして検出される。

この他に, コード自動生成ツールの使用, コードの剽窃, 意図的なコードの複製によるコード行数の水増しなども, コードクローンの生成要因として指摘されている[39]。

これらのうち, “コピーアンドペーストによるコードの複製” がコードクローン生成の主たる要因であると考えられており, コピーアンドペーストによるコードの複製がなぜ行われるのか, という点に主眼を置いた調査が行われている。

Kapser と Godfrey は, 開発者がコードの複製 (cloning) を行う際のパターンを発見した[41]。彼らの発見したパターンは大きく以下の4種類に分類することができる。

Forking: すでに実装されている機能と類似した機能を実装する際に, 初めにすでに実装されているコードを複製し, そこに改良を加えていくことで, 実現したい機能を実装していく。すでに実装されている機能と, 新たに実装する機能の開発を独立して行う場合に生じ, 既存のコードに変更を加えることを避けるねらいがある。新たなハードウェアドライバを開発する際や, 既存のシステムを別のプラットフォームに対応させる際などに生じる複製パターンである。

Templating: 実装したい機能がすでに存在しているが, 継承や総称型などの適切な抽象化手法が利用できない場合に, 実現したい機能を実装しているコード片を複製し, 適宜変更を加えることで開発を行う。使用する言語に適切な機能が欠如している場合などに生じる。Forkingとは異なり, 複製元と複製先の開発を独立して行うことを前提としていないため, Templatingの結果生成され

たコードクローンは同時に修正をしなければならない可能性が高い。

Customization： 解決したい問題と似た問題を解決する既存コードが存在するが、拡張や変更が必要な場合に、既存コードを複製したうえで修正を加えていく。既存コードを再利用しやすい形に抽象化する作業に高いコストを要すると判断された場合に発生する。

Exact Matches： 特定の機能を実現するコードが繰り返し出現しているが、それらを抽象化して再利用しやすくする利点がありません場合に、既存コードを複製して利用する。複製するコード片が非常に小さい場合などに起こる。

開発者がなぜコードの複製を行うのか、という点に関する調査もいくつか行われている。Kim らは IBM に所属する研究者を対象として、プログラマがコピーアンドペーストを行う理由についての調査を行った[42]。その結果、プログラミング言語の制約上避けられないコードの重複が存在することを指摘している。

また、Zhang らは商用ソフトウェアの開発におけるコードの複製についての調査を行った[84]。開発者に対してアンケート調査を行った結果、コードの複製をどれくらいの頻度で行うのかという質問に対し、21 人の開発者のうち 10 人がコードの複製を頻繁に行うと回答し、残りの 11 人もしばしば行うと回答した。さらに、開発者がコードの複製を行う理由として、「既存コードの機能を壊さないため」、「抽象化による再利用が困難であるため」、などという技術的な要因が挙げられた他に、「定められた期日までに開発を終えるため」という時間的要因や、「再利用したい既存コードを変更する権限が開発者に与えられていないため」、という組織的要因も存在することが報告されている。そのうち時間的要因については、約 47% の開発者が開発時間短縮を目的にコードの複製を行っていたと報告されている。

2.3 コードクローンは有害か

近年、コードクローンがソフトウェア保守に与える影響に関する定量的調査が多く研究者によって行われている。これらの調査報告の中には、コードクロー

ンがソフトウェア保守に悪影響を与えるとする従来の前提を支持するような報告もあれば、反対にその前提が成り立っていないとする報告もある。

コードクローンがソフトウェア保守に悪影響を与えるとする調査研究として、コードクローンに起因する不具合が多数存在すると報告している研究がある。Juergens らはすべてのコードクローンのうち約半数に一貫性の無い修正が加えられ、さらに一貫性の無い修正が加えられたコードクローンのうち約 18% が不具合に関わっていると報告している[33]。また Li らはコードクローン検出ツール CP-Miner を Linux カーネルに適用して調査を行った結果、コードクローンに起因する不具合が 28 件存在していたと報告している[52]。

このほかに、コードクローンがソフトウェア保守に悪影響を与えているといえるような調査結果には以下のものがある。

- コードクローンを含むメソッドは含まないメソッドと比較して頻繁に修正される[54][55]
- コードクローンを含むメソッドは含まないメソッドと比較して保守に要するコストが高い[53][59]
- コードクローンを含むモジュールは含まないモジュールと比較して保守性が低い[60]

他方、コードクローンがソフトウェア保守に悪影響を与えているとは必ずしもいえないとする報告も存在する。Göde と Koschke が行った調査によれば、約 90% のコードクローンがその生存期間中に高々 1 回しか修正されず、またコードクローンに加えられた一貫性の無い修正のうち不具合につながる危険性の高いものは約 3% 程度であった[13]。また Bettenburg らはコードクローンに加えられた一貫性の無い修正をリリース単位で調査し、不具合をもたらしていたものは全体の約 1%~4% 程度であったと報告している[8]。

これら以外にも、コードクローンがソフトウェア保守に悪影響を与えているとは必ずしもいえないとする報告が以下のように多数存在する。

- コードクローンは生存期間が短いものが多い[44]
- クローン関係にあるコード片に対する修正漏れはあまり発生していない[2][66][70]
- 多くのコードクローンは不具合との関係がない

[63]

- コードクローンはそれ以外の箇所と比べて修正が加えられる頻度が小さい[12][14][24][47][48]
- コードクローンとそれ以外の箇所のどちらが修正されやすいかは分析対象のシステムによって異なる[59]
- 同様の不具合修正を行わなければならないなどの、ソフトウェア保守に悪影響を与えるようなコードクローンは少ない[41]

このように、調査の結果は調査手法や対象ソフトウェアによってまちまちであり、コードクローンとソフトウェア保守の関係についての統一された結論は得られていない。しかしながら、これらの調査報告によって得られた知見を総合すると、ソフトウェア保守に悪影響を与えるコードクローンもそうでないコードクローンも双方共に存在する、ということがいえる[37]。

2.4 コードクローン管理技術の必要性

2.2 で述べたように、開発時間の短縮や既存コードへの影響を回避することを目的として、あるいはプログラミング言語の制約上避けられないという理由から、コピーアンドペーストによる既存コードの再利用はソフトウェア開発現場において少なからず行われている。このことから、コピーアンドペーストを禁止してコードクローンの生成を完全に防ぐことは、現実的に極めて困難であるといえる。また、コードクローンの多くは簡単に除去することができないものであるという報告もある[43][44]。ことから、すでに作り込まれているコードクローンを完全に取り除くことは難しい。

また 2.3 で述べた通り、すべてのコードクローンが必ずしもソフトウェア保守に悪影響を与えているとはいえない、ということが明らかになりつつある。このことから、あらゆるコードクローンを悪しきものとみなし、コードクローンを全く生成させないことや完全に取り除くことに労力を費やすことは効率的であるとはいえない[75]。

これらを勘案すると、ソフトウェア保守に悪影響を与える危険性の高いコードクローンについてその生

成抑止や除去を行う、あるいは、コードクローン検出ツールを用いてソースコード中のコードクローンを把握することなどを通じ、コードクローンを適切に管理していくことが必要であるといえることができる。

2.5 本稿で扱うコードクローン管理技術

コードクローンの管理、という用語は抽象的であるため様々な定義が考えられるが、本稿では文献[46]で Koschke が述べた定義に従い、“コードクローンに対して注意を払う、あるいは何らかの意思決定を行うすべての行為”と定義する。この定義に従えば、コードクローンの検出、除去のみならず、生成、拡散の抑止、あるいは可視化手法などの幅広い技術がコードクローン管理を支援する技術とみなされる。

著者らがコードクローンの管理に関わる文献を調査し、それらの分類を行った。分類を行うにあたり、著者らはそれぞれの文献で報告あるいは提案されている内容が何を目的としているのか、という点に注目して文献を調査した。はじめに、コードクローンの研究分野において最も重要な基盤技術である検出技法を 1 つのカテゴリとした。次に、検出されたコードクローンを対象とした管理技法を提案している文献について、最も代表的な対処法である除去技法を 1 つのカテゴリとし、またコードクローンによってもたらされる弊害の最たるものとして指摘されている修正漏れによる不具合に着目した技法を別のカテゴリとした。さらに、現存するコードクローンを対象とした技法のうち、前述の 2 つのカテゴリに当てはまらない文献を調査した結果、それらは大量に検出されるコードクローンに対しての適切な管理の実現を模索する研究であると判断し、それらを 1 つのカテゴリとした。最後に、現在あるコードクローンに対してではなく、これから生まれるコードクローンに対する未然の対処法を提案する文献を 1 つのカテゴリとした。

以上より、本稿ではコードクローン管理に関わる文献を大まかに以下の 5 つのカテゴリに分類する。

- 検出
- 除去
- 生成、拡散の抑止
- 分析の効率化

- 不具合の混入防止, および検出

これまでに述べたように, コードクローンを検出して除去する, という単純な方法ではコードクローンを適切に管理することは難しい. したがって, “検出” 及び “除去” 以外のカテゴリに分類される研究の重要度は今後ますます高くなると考えられる. しかし, これまでのサーベイ論文は “検出” 及び “除去” に重点が置かれている. そこで本稿では, これまでのサーベイ論文が重点的に扱っていないが, 今後重要度が高くなると考えられる “生成, 拡散の抑止”, “分析の効率化”, “不具合の混入防止, および検出” に関する研究成果を紹介する.

3 生成, 拡散の抑止

2.2 で議論したように, コピーアンドペーストによるコードの複製にはいくつかの利点があり, また時として避けられないことである. しかしながら, 野放図にコードを複製しコードクローンを大量に生成することは, コードクローンによる悪影響を被る危険性を高める. したがって, コードの複製によってもたらされる利益と, 生成されるコードクローンによってもたらされる危険性のある弊害とを考慮し, コードの複製を許可するかどうかを決定することが重要である.

コードの複製を許可するか否かを決定する戦略は以下のようにいくつか考えられる.

- A. コードの複製が必要である場合のみ許可する.
- B. 生成されるコードクローンが危険なものになると予想される場合は許可しない.
- C. 生成されるコードクローンが安全なものになると予想される場合のみ許可する.

Laguë らは妥当な理由がある場合のみコードの複製を許可する “Preventive Control” を提唱している [49]. 彼らの提唱する方法では関数単位の複製を対象としている. 開発者がコードの複製を行ったときに, 開発者になぜ複製を行ったのかを説明する義務が課せられる. そして, 説明された理由が妥当なものであると判断された場合のみコードの複製を許可することで, コードクローンの生成を制御している.

Wang らは開発者がコードの複製を行おうとしたときに, その結果生じるコードクローンが危険なもの

となり得るかどうかを予測する手法を提案している [76]. 彼らの手法では, “コピー元とコピー先に同様の修正をする必要が生じる” と予測される場合に, 生成されるコードクローンが危険なものであると判断する. 反対に, “生成されるコードクローンが今後修正されない”, もしくは, “コピー元とコピー先のコード片に同様の修正をしなければならないという状況が起きない”, と予測される場合は, コードの複製によって生じるコードクローンが危険ではないと判断している. コピーするコード片やコピー先の位置などから 21 の特徴量を算出し, それらを用いた機械学習によって生成されるコードクローンの危険性を予測している. なお彼らの提案している手法は, 上述の戦略のうち B., および C. の双方に用いることが可能であり Microsoft 社の製品を対象とした評価実験において, 戦略ごとに手法の有効性を示している.

Venkatasubramanyam らは統合開発環境上でのコードクローンの生成抑止に関するフレームワークを提案している [73]. 彼らの手法では統合開発環境上の開発者の振る舞いを監視し, コードの複製を行おうとした際にその複製があらかじめ定められた制約を満たすかどうかを判定する. そして, コードの複製が制約を満たしていない場合はコードの複製をさせない, というアプローチを取っている. コードの複製に関する制約を利用者が自由に決定することができる, という点がこのフレームワークの大きな特徴である. 文献中には制約の例として, 『サイクロマチック数が 5 以上のコード片を複製することを禁じる』などが挙げられている.

4 分析の効率化

コードクローン検出ツールを用いた自動的なコードクローン検出における課題点として, 膨大な量のコードクローンが検出されることが指摘されている [32]. 大量に検出されるコードクローンを効率的に分析し, 求める情報を得るために, 手法やツールによる支援が重要となっている. 本章では, 膨大なコードクローン情報を効率的に分析する上で有益な手法, ツールを紹介する.

表 1 コードクローンの分類, クラスタリング手法

手法	分類・クラスタリングの基準	手法の概要
Jiang らの手法 [32]	利用者が指定	利用者が指定した条件を満たさないクローンセットを除外する.
楊らの手法 [80]	TF-IDF 法を用いた機械学習	与えられた学習データに対して TF-IDF 法を適用し, 未分類のクローンセットが“有用”である確率を算出する.
Zhang らの手法 [85]	SQL 命令として指定	条件を SQL 命令を与えることで条件を満たすクローンセットを取得する.
Tairas と Gray の手法 [69]	潜在的意味解析手法	同じ名前の識別子を多く共有するクローンセット同士をクラスタリングする.
Kapser と Godfrey の手法 [40]	クローン片の位置関係	クローンペアを構成するクローン片の位置関係により, <i>Same File, Same Dir, Different File</i> などに分類する.
Merlo と Lavoie の手法 [57]	クローン片を所有するブロック	クローンペアを構成するクローン片が属するブロックにより, メソッド同士のクローンペア, ブロック同士のクローンペア, メソッドとブロックのクローンペアに分類する.
Balazinska らの手法 [4]	クローン片を共有するメソッド間の差分	メソッド間の差分により, <i>Identical</i> や <i>Called methods</i> (クローン片が呼び出しているメソッドの名前が違う場合) など, 合計 18 種類に分類する.
Xing らの手法 [78]	プログラム依存グラフ上の差分	クローンペアのそれぞれのクローン片から構築したプログラム依存グラフを比較し, その差分によって <i>Unmatched Block Pair</i> (同じ種類のブロック文を含んでいるが, そのブロック文内で行われている処理が違う) など, 合計 7 種類に分類する.
Kamiya の手法 [37]	<i>Makefile</i> などの構成情報	それぞれの構成に必要なソースファイルの集合と, クローンセットを構成するクローン片が属するソースファイルの情報を用い, 単一の構成で出現するクローンセット, ある構成では出現しないが別の構成では出現するクローンセット, などに分類する.
Kamiya の手法 [38]	関数の呼び出し関係	関数を単位とする. 同じ関数を呼び出している関数同士を <i>content clone</i> とし, 同じ関数から呼び出されているメソッド同士を <i>context clone</i> とする.

4.1 絞り込み, 分類

大量に検出されたコードクローンを効率的に分析するための方法として, 検出結果に対する絞り込み作業を行い, 分析対象とするコードクローンの量を削減するというアプローチがある. あるいは, コードクローンをいくつかのカテゴリに分類, もしくは似た特徴を持つコードクローン同士をクラスタリングすることで, コードクローン分析の効率化を目指す

手法が提案されている. 表 1 にそれらの概要を示す. これらの手法は, コードクローン検出ツールの利用者が検出されたコードクローンの一部のみを対象とした分析を行う際に, その効率化に寄与する.

Jiang らはデータマイニングの技術を用いたコードクローンの絞り込み手法を提案している [32]. この手法では有益でないコードクローンを検出結果から除外することで絞り込みを実現している. ここで, あ

るコードクローンが“有益”かどうかは調査の目的や調査実施者の主観に依存するため、Jiang らの手法ではコードクローンの有益性の判断基準は手法の利用者が決定可能となっている。例えば、ある特定のサブシステム間に存在するコードクローンを調査したい、という場合、それ以外のサブシステムに存在するコードクローン情報はすべて“有益でない”として除外することができる。

楊らは機械学習を用いたコードクローンの絞り込み手法を提案している[80]。この手法では、利用者がいくつかのコードクローンを見て“有用”かそうでないかを入力することで、残りのコードクローンが“有用”か否かを予測し、提示している。“有用”か否かを判断する基準がコードクローン分析や分析者の主観によって異なると考えられるが、このような判断基準の違いに柔軟に対応できることが楊らが提案する手法の最大の特長である。

Zhang らはコードクローン検出ツール Clone Miner [5] の出力結果をリレーショナルデータベースに格納することで、SQL を用いたコードクローンの検索、フィルタリングを実現している[85]。また Zhang らはこのフィルタリング手法を GUI 上で使用することができるように、統合開発環境 Eclipse のプラグイン Clone Visualizer として実装している。

Tairas と Gray は潜在的意味解析手法を用いたクローンセットのクラスタリング手法を提案している[69]。この手法ではクローンセット内の各コード片に出現する識別子を語とみなし潜在的意味解析手法を適用している。すなわち、同じ名前の識別子を多く共有しているクローンセット同士が類似するとみなされ、クラスタリングされる。

コードクローンを分類する手法として、ファイルやディレクトリなどの位置情報を用いた分類手法[40]や、ブロックやメソッドなどの構文情報を用いた分類手法[57]、クローン関係にあるコード片の間に存在する差分の種類に基づく分類手法[4][78]、*Makefile*などに記述されている、実行可能ファイルを生成するためのコマンドに着目した分類手法[37]、及び関数の呼び出し関係に着目した分類手法[38]がある。これらの分類手法はソフトウェア中に存在するコードクローン

を把握、あるいは理解する上で有用であると期待される。

4.2 可視化

ソフトウェアの可視化 (Software Visualization) はソフトウェアの理解や保守に必要な労力を削減する上で非常に有効である[6]。このような理由から、コードクローンの分野に対する可視化手法がいくつか提案されている。これらのコードクローン可視化手法は、システム全体に存在するコードクローンの様子を俯瞰する際に有用である。

Rieger らは Polymetric View をコードクローンの可視化に応用することを提案している[64]。Polymetric View とは、複数の特徴量の組み合わせを単一のビューとして表現する表現形式である[50]。Rieger らは Polymetric View を用いた 6 種類のコードクローン可視化手法を提案している。その 1 つである Duplication Web の例を図 2 に示す。この可視化方法は、すべてのソースファイルを頂点として同心円状に配置し、ファイル間のコードクローン共有関係を辺として表現している。辺の太さが太いほど、そのファイル間で共有されているコードクローンの合計行数が多いことを示している。また、横幅の広い頂点はクローン関係がそのファイル内に閉じているコードクローンの量が多いことを示している。Duplication Web を用いることで、どのソースファイルにコードクローンが大量に存在するのか、またどのファイルとどのファイルの間にコードクローンが存在するのか、といった情報が一目で把握できる。

Wattenberg は文字列中に出現するパターンを可視化する手法 Arc Diagrams を提案している[77]。図 3 に Arc Diagrams の例を示す。この可視化手法はソー

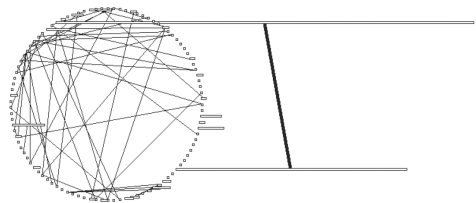


図 2 Rieger らの手法 (Duplication Web)(文献[64]より引用)

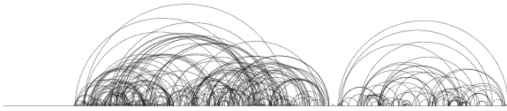


図3 Wattenberg の手法 (文献[77]より引用)

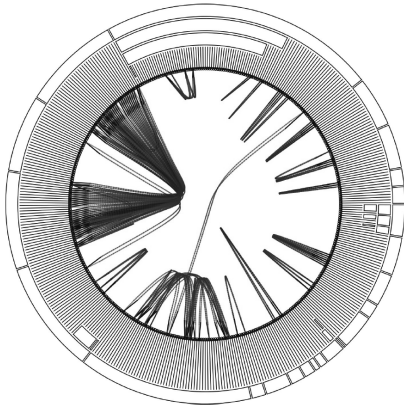


図4 Hauptmann らの手法 (文献[16]より引用)

スコードのみならず、バイトコード、あるいはDNA系列や楽譜などの様々な対象に応用することが可能である。

Hauptmann らは Bundle Edge View を用いた可視化手法を提案している[16]。Bundle Edge View はグラフの可視化方法の一種であり、出発点と到達点がどちらも近い位置にあるような辺について、それらを束ねることでグラフをより見やすくしたものである[23]。Hauptmann らが提案する可視化の例を図4に示す。それぞれの辺はファイル間のコードクローン共有関係を表している。また、円の外周上に存在する四角はディレクトリ構造を表しており、外周から中心に向かう線は個々のファイルを表している。この可視化方法により、大規模なシステムを対象とした場合でもファイル間やディレクトリ間のコードクローン共有関係を把握することができる。

Yoshimura と Mibe はファイル間の類似関係を可視化する手法を提案している[83]。その例を図5に示す。Yoshimura らの可視化手法はソフトウェア開発の熟練者ではないステークホルダーにコードクロンの分散状況を説明することを主な目的としており、

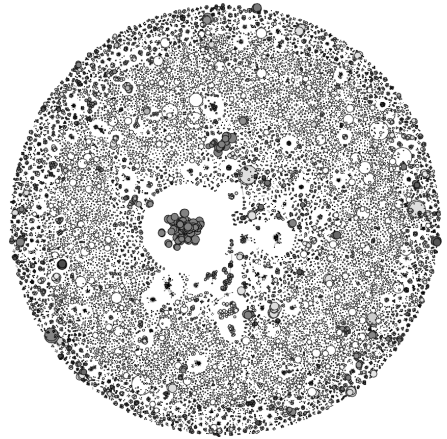


図5 Yoshimura と Mibe の手法 (文献[83]より引用)

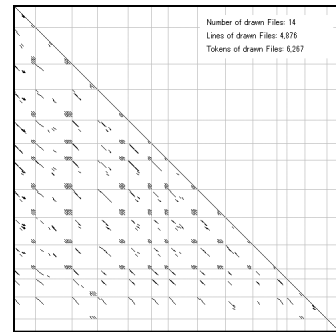


図6 Gemini[18]が表示する散布図の例

この可視化手法を用いることでソフトウェア中のコードクロンの分散状況を知ることができる。

植田らはコードクローン検出ツール CCFinder[35]の GUI フロントエンドである Gemini を開発した[72]。Gemini はソースコード中のコードクローンを鳥瞰するために散布図を表示する(図6)。またコードクローンに関する特徴量をいくつか定義し、それらを提示することでそれぞれのコードクローンがどのような特徴を持っているのかを把握することを支援する Metric Graph を提案している。さらに、Gemini はメトリクス値を用いたコードクロンの絞り込み機能も有しており、Metric Graph 上で視覚的に絞り込みを行うことが可能である。Gemini に関連する研究として、宮崎らはコードクローンに関する散布図に対してどこから分析を行えば良いか判断する作業を支

援するために、散布図上にコードクロンの頻出パターンを提示する手法を提案している [58].

Adar と Kim はグラフを用いてコードクロンの可視化をするためのライブラリ SoftGUESS を提案している [1]. Adar と Kim は SoftGUESS を用いた可視化例として、コードクロンがどのように進化してきたのかを可視化したグラフ (Genealogy Browser), クローン片を持つメソッドやクラス、パッケージの階層構造を木構造で表現したグラフ (Encapsulation Browser), クローン片間のパッケージレベルの依存関係を表示するグラフ (Dependency Browser) を紹介している.

また Saha らは、大規模なソフトウェアにコードクロンの進化の様子をより効率的に分析するために、コードクロンの進化の様子を散布図を用いて可視化する手法を提案している [67].

5 不具合の混入防止, 検出

コードクロンによってもたらされる弊害の 1 つに不具合の混入がある. あるコード片に対して修正を加えた際に, そのコード片とクローン関係にある他のコード片に対する修正を怠った場合, いわゆる修正漏れが発生した場合に不具合が混入する危険性がある. また, あるコード片に対する不具合修正を他のコード片に施し忘れた場合, 不具合がソースコード中に残存することになる. コードクロンに起因する不具合は, 似たような不具合が繰り返し出現することにつながる危険性がある. 同じような不具合が何度も顕在化することは, ソフトウェア開発者に対する利用者の信頼を大きく損ないかねない.

本章では, コードクロンに関わる不具合の混入を防止する, あるいはそれらを検知する上で有用な手法, ツールを紹介する.

5.1 不具合の混入防止

コードクロンに対する修正漏れを未然に防ぐためのアプローチとして, クローン関係にあるコード片に対する同時修正を支援する手法や, 開発者がコード片を修正しているときにそのコード片とクローン関係にあるコード片の情報を提示する手法, 修正漏れと

思しき箇所が見つかった場合にそれを開発者に通知する手法などがある.

Toomim らは同時修正を支援する技術である Linked Editing を提唱している [71]. Linked Editing では, 使用者が 2 つのコード片を明示的にリンクさせることで, 一方のコード片に対する修正をもう一方のコード片に自動的に適用することができる. また, 一方のコード片のみを修正したいという場合は, そのように指定をすることも可能である. 一方, Higo らは開発者が指定したコード片とクローン関係にあるコード片のみを検出し, 提示することで同時修正を支援する手法を提案している [21].

同時修正を支援する他の技術として, 開発者のコピーアンドペーストの操作を検知し, その結果生じたコードクロンに対する同時修正を支援する技術が提案されている [9] [25] [28] [29]. また, コードクロンの情報をコメントとしてソースコードに付加することで修正漏れを防ぐという提案もなされている [68].

Nguyen らはコードクロン管理ツール JSync を提案している [61]. JSync はコードクロンに対する一貫性のない修正を自動的に検知し, それを開発者に通知するだけでなく, どのように修正すればよいのかを併せて提示する機能を持っている.

Duala-Ekoko と Robillard はコードクロンに対する変更を開発者に通知するシステム CloneTracker を開発した [10]. CloneTracker はあるコード片に対して修正を加えたときに, そのコード片とクローン関係にあるコード片の位置情報を開発者に提示する. このシステムではコードクロン追跡手法 CRD (Clone Region Descriptors) を使用している. これにより, コードクロンとして検出された後に修正が加えられたような場合でも, 開発者に対してクローン関係にあるコード片の情報を提示することが可能となっている.

山中らはソースコードのバージョン間でコードクロンがどのように変化しているのかを通知するシステムを提案している [79]. このシステムでは, 追加, 変更, あるいは削除が行われたコードクロンを発見した際に, その情報をウェブベースとメールベースで開発者に通知する. 日本電気株式会社のウェブシステ

ムに対して評価実験を行い、開発者が認知していなかった、保守作業の対象となるようなコードクローンを提示することができたと報告している。

5.2 不具合の検出

コードクローンに起因する不具合を検出するアプローチの1つとして、クローン片の間に存在する不一致箇所の情報を用いる方法がある。ただし、用いる不一致箇所の情報は個々の研究手法によって異なる。これまでに、識別子名の対応関係の不一致を用いる手法[26][52]、クローン片を所有する制御構造の不一致を用いる手法[31]が提案されている。また、複数のコードクローン検出結果間の差分を利用して不具合の検出を行う手法も提案されている[20]。

Li と Ernst はある不具合を発見した時、同様の不具合が存在する可能性が高い箇所を自動的に特定するツール CBCD (Cloned Buggy Code Detector) を提案している[51]。CBCD は入力として与えられた不具合箇所のソースコードからプログラム依存グラフを構築し、そのプログラム依存グラフと同形部分グラフとなっている箇所を同様の不具合が存在する可能性が高い箇所として提示する。この他に、不具合箇所の情報を入力として与えることで類似した不具合を検出することを目的とした手法として、コード片に出現する識別子の情報を用いる手法[81][82]や並列処理プログラムを対象としたもの[30]がある。

Lucia らはコードクローンに関する不具合検出ツールは誤検出が多いと主張し、不具合検出ツールの結果を洗練することで誤検出を振るい落とす手法を提案している[56]。この手法を他の不具合検出ツールと組み合わせることで、より効率的に不具合を検出することができる。同様のアプローチとして、Hayase らは CP-Miner の誤検出を取り除くためのフィルタリング方法を提案している[17]。

この他に、Fault-prone モジュール予測にコードクローン情報を用いるという研究も行われている[3][34]。

また近年、コードクローンに特化した版管理システムが提案されている[15][62]。過去に発生した修正漏れの結果、現在のバージョンではコードクローンとして検知されなくなっているような場合、現在のバー

ジョンのソースコードのみから修正漏れによる不具合を検知することは難しい。このような場合に、これらの版管理システムから入手できる過去のバージョンにおけるコードクローン情報が有益であると考えられる。さらに、これらの版管理システムを用いる利点はこのような過去の修正漏れの発見のみにとどまらない。畑らが文献[15]で述べているように、コードクローンに特化した版管理システムを用いることで、コードクローンに関するプロセスや人的属性の履歴についてのデータを得ることが可能となる。このような情報を用いることで、近年活発に研究されているリポジトリマイニングの技術をコードクローン履歴のマイニングに応用することができる。

6 今後の展望

コードクローン検出技術の導入に際しての障壁の1つとして、検出に要する時間的、空間的コストの大きさがある。ソースコードの比較に必要な計算量は比較アルゴリズムによって異なるが、例えば接尾辞木探索を用いる場合 $O(n)$ (ただし、 n は字句数) となる[27]。検出時間については、例えば広く利用されているコードクローン検出ツールの1つである CCFinder[35]の場合、およそ 10,000,000 行のソースコードに対するコードクローン検出に要した時間が約 68 分であることが報告されている[27]。

近年の計算機能力の向上やアルゴリズムの改良によって、検出に要するコストは以前と比較して低減されている。コードクローン検出技術はコードクローンに関する研究の中でも最も熱心に研究されている分野の1つであり、今後さらなるコスト低減や検出精度向上が期待できる。また近年、ソースコードのみならず、UML 図やモデルなどに対するクローン検出技術も研究されている。今後ソフトウェアの様々な構成要素に対してコードクローン検出技術の応用が進むものと期待される。

一方、コードクローン検出における課題点として、コードクローンとは何か、という点が厳密に定まっていない点がある。この点はこれまでも繰り返し議論されているが、今後さらなる議論が必要である。

コードクローン管理における今後の重要な課題と

して、どのようなコードクローンが有害であるのかを特定することが必要であると考え、2.3で述べたようにコードクローンの有害性に関する議論は共通の結論を得ておらず、また得ることは困難であるといえるが、コードクローンの中には有害なものとそうでないものが存在するということはいえるであろう。有害なコードクローンが持つ特徴を分析していくことで、大量に検出されたコードクローンの中から有害なものを効率的に見つけ出すことが実現可能となる。

有害なコードクローンの特定は、本稿で紹介した様々な研究の発展のための基盤技術になり得ると期待される。例えばコードクローンの除去であれば、有害なコードクローンのみに着目してそれらに対する除去を検討することによって、より少ない労力でコードクローンによる弊害を効率的に防ぐことができる。生成、拡散の抑止については、有害なものについては生成、拡散を防止するという対処が可能となる。分析については、有害なものに対して優先的に分析を進めることで分析効率を高めることができると期待される。

しかしながら、どのようなコードクローンが有害である危険性が高いのか、という点については明らかにされていない。また、どのようなコードクローンが有害なのか、という点については議論の余地が広く残されている。今後この点に関する議論を進めたいうえで、有害なコードクローンが持つ特徴を明らかにしていくことが重要になると考える。コードクローンの有害性については、修正漏れによる不具合が主に指摘され、研究されているが、その他の側面についても議論をする必要があるという指摘がなされている[75]。

7 あとがき

本稿では、コードクローン管理支援技術に関する様々な研究成果を紹介した。本稿ではコードクローン管理支援技術を大きく5つのカテゴリに分類した上で、既存のサーベイ論文で取り扱われていないカテゴリに重点を置いた。コードクローンの検出及び除去技術については他のサーベイ論文及び解説記事に詳しいので、参照されたい[19][22][27][36][39][45][65][86]。

謝辞 本論文は、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)、挑戦的萌芽研究(課題番号:24650011)、特別研究員奨励費(課題番号:25・1382)、並びに文部科学省科学研究費補助金若手研究(A)(課題番号:24680002)の助成を得て作成された。

参考文献

- [1] Adar, E. and Kim, M.: SoftGUESS: Visualization and Exploration of Code Clones in Context, in *Proc. of the 29th Int. Conf. on Softw. Eng. (ICSE'07)*, 2007.
- [2] Aversano, L., Cerulo, L. and Penta, M. D.: How Clones are Maintained: An Empirical Study, in *Proc. of the 11th European Conf. on Softw. Maintenance and Reengineering (CSMR'07)*, 2007, pp. 81–90.
- [3] 馬場慎太郎, 吉田則裕, 楠本真二, 井上克郎: Fault-Prone モジュール予測へのコードクローン情報の適用, 電子情報通信学会論文誌, Vol. J91-D, No. 10(2008), pp. 2559–2561.
- [4] Balazinska, M., Merlo, E., Dagenais, M., Lagüe, B. and Kontogiannis, K.: Measuring clone based reengineering opportunities, in *Proc. of the 6th Int. Softw. Symposium on Softw. Metrics (METRICS'99)*, 1999, pp. 292–303.
- [5] Basit, H. A. and Jarzabek, S.: Detecting Higher-level Similarity Patterns in Programs, in *Proc. of the 13th Int. Symposium on Foundations of Softw. Eng. (ESEC/FSE'05)*, 2005, pp. 156–165.
- [6] Bassil, S. and Keller, R. K.: Software Visualization Tools: Survey and Analysis, in *Proc. of the 9th Int. Workshop on Program Comprehension (IWPC'01)*, 2001, pp. 7–17.
- [7] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M. and Bier, L.: Clone Detection Using Abstract Syntax Trees, in *Proc. of the 14th Int. Conf. on Softw. Maintenance (ICSM'98)*, 1998, pp. 368–377.
- [8] Bettenburg, N., Shang, W., Ibrahim, W. M., Adams, B., Zou, Y. and Hassan, A. E.: An empirical study on inconsistent changes to code clones at the release level, *Science of Computer Programming*, Vol. 77, No. 6(2012), pp. 760–776.
- [9] de Wit, M., Zaidman, A. and van Deursen, A.: Managing Code Clones Using Dynamic Change Tracking and Resolution, in *Proc. of the 24th Int. Conf. on Softw. Maintenance (ICSM'09)*, 2009, pp. 169–178.
- [10] Duala-Ekoko, E. and Robillard, M. P.: Clone Region Descriptors: Representing and Tracking Duplication in Source Code, *ACM Trans. on Softw. Eng. and Methodology*, Vol. 20, No. 1(2010), pp. 3:1–3:31.
- [11] Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D.: *Refactoring: Improving the Design of*

- Existing Code*, Addison-Wesley Professional, 1999.
- [12] Göde, N. and Harder, J.: Clone Stability, in *Proc. of the 15th European Conf. on Softw. Maintenance and Reengineering (CSMR'11)*, 2011, pp. 65–74.
 - [13] Göde, N. and Koschke, R.: Frequency and Risks of Changes to Clones, in *Proc. of the 33rd Int. Conf. on Softw. Eng. (ICSE'11)*, 2011, pp. 311–320.
 - [14] Harder, J. and Göde, N.: Cloned code: stable code, *Jour. of Softw. : Evolution and Process*, (2012). doi: 10.1002/smr.1551.
 - [15] 畑秀明, 肥後芳樹, 楠本真二: リポジトリマイニング可能なコードクローン版管理システムの提案, 情報処理学会論文誌, Vol. 54, No. 2(2013), pp. 894–902.
 - [16] Hauptmann, B., Bauer, V. and Junker, M.: Using Edge Bundle Views for Clone Visualization, in *Proc. of the 6th Int. Workshop on Softw. Clones (IWSC'12)*, 2012, pp. 86–87.
 - [17] Hayase, Y., Lee, Y. Y. and Inoue, K.: A Criterion for Filtering Code Clone Related Bugs, in *Proc. of the Int. Workshop on Defects in Large Softw. Systems (DEFECTS'08)*, 2008, pp. 37–38.
 - [18] Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: Method and implementation for investigating code clones in a software system, *Information and Softw. Technology*, Vol. 49, No. 9–10(2007), pp. 985–998.
 - [19] 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, Vol. 91-D, No. 6(2008), pp. 1465–1481.
 - [20] Higo, Y., Sawa, K. and Kusumoto, S.: Problematic Code Clones Identification using Multiple Detection Results, in *Proc. of the 16th Asia-Pacific Softw. Eng. Conf. (APSEC'09)*, 2009, pp. 365–372.
 - [21] Higo, Y., Ueda, Y., Kusumoto, S. and Inoue, K.: Simultaneous Modification Support based on Code Clone Analysis, in *Proc. of the 14th Asia-Pacific Softw. Eng. Conf. (APSEC'07)*, 2007, pp. 262–269.
 - [22] 肥後芳樹, 吉田則裕: コードクローンを対象としたリファクタリング, コンピュータソフトウェア, Vol. 28, No. 4(2011), pp. 42–56.
 - [23] Holten, D.: Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data, *IEEE Trans. on Visualization and Computer Graphics*, Vol. 12, No. 5(2006), pp. 741–748.
 - [24] Hotta, K., Sasaki, Y., Sano, Y., Higo, Y. and Kusumoto, S.: An Empirical Study on the Impact of Duplicate Code, *Advances in Softw. Eng.*, Vol. 2012(2012).
 - [25] Hou, D., Jablonski, P. and Jacob, F.: CnP: Towards an Environment for the Proactive Management of Copy-and-Paste Programming, in *Proc. of the 17th Int. Conf. on Program Comprehension (ICPC'09)*, 2009, pp. 238–242.
 - [26] Inoue, K., Higo, Y., Yoshida, N., Choi, E., Kusumoto, S., Kim, K., Park, W. and Lee, E.: Experience of Finding Inconsistently-Changed Bugs in Code Clones of Mobile Software, in *Proc. of the 6th Int. Workshop on Softw. Clones (IWSC'12)*, 2012, pp. 94–95.
 - [27] 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(2001), pp. 47–54.
 - [28] Jablonski, P. and Hou, D.: CReN: A Tool for Tracking Copy-and-Paste Code Clones and Renaming Identifiers Consistently in the IDE, in *Proc. of the OOPSLA Workshop on Eclipse Technology eXchange (ETX'07)*, 2007, pp. 16–20.
 - [29] Jacob, F., Hou, D. and Jablonski, P.: Actively Comparing Clones Inside The Clone Editor, in *Proc. of the 4th Int. Workshop on Softw. Clones (IWSC'10)*, 2010, pp. 9–16.
 - [30] Jalbert, K. and Brandbury, J. S.: Using Clone Detection to Identify Bugs in Concurrent Software, in *Proc. of the 26th Int. Conf. on Softw. Maintenance (ICSM'10)*, 2010, pp. 1–5.
 - [31] Jiang, L., Su, Z. and Chiu, E.: Context-Based Detection of Clone-Related Bugs, in *Proc. of the 15th Int. Symposium on the Foundations of Softw. Eng. (ESEC/FSE'07)*, 2007, pp. 147–156.
 - [32] Jiang, Z. M. and Hassan, A. E.: A Framework for Studying Clones In Large Software Systems, in *Proc. of the 7th Working Conf. on Source Code Analysis and Manipulation (SCAM'07)*, 2007, pp. 203–212.
 - [33] Juergens, E., Deissenboeck, F., Hummel, B. and Wagner, S.: Do Code Clones Matter?, in *Proc. of the 31st Int. Conf. on Softw. Eng. (ICSE'09)*, 2009, pp. 485–495.
 - [34] Kamei, Y., Sato, H., Monden, A., Kawaguchi, S., Uwano, H., Nagura, M. and Matsumoto, K.: An Empirical Study of Fault Prediction with Code Clone Metrics, in *Proc. of the Joint Conf. of Int. Workshop on Softw. Measurement and Int. Conf. on Softw. Process and Product Measurement (IWSM/Mensura'11)*, 2011, pp. 55–61.
 - [35] 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 Softw. Eng.*, Vol. 28, No. 7(2002), pp. 654–670.
 - [36] 神谷年洋: コードクローンとは, コードクローンが引き起こす問題, その対策の現状, 電子情報通信学会論文誌, Vol. 87, No. 9(2004), pp. 791–797.
 - [37] Kamiya, T.: Classifying Code Clones with Configuration, in *Proc. of the 4th Int. Workshop on Softw. Clones (IWSC'10)*, 2010, pp. 75–76.
 - [38] Kamiya, T.: Conte*t Clones or Re-thinking Clone on a Call Graph, in *Proc. of the 6th Int. Workshop on Softw. Clones (IWSC'12)*, 2012, pp. 74–75.
 - [39] 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(2011), pp. 28–42.
 - [40] Kapsner, C. and Godfrey, M. W.: Aiding Comprehension of Cloning Through Categorization, in

- Proc. of the 7th Int. Workshop on Principles of Softw. Evolution (IWPSE'04)*, 2004, pp. 85–94.
- [41] Kapser, C. J. and Godfrey, M. W.: “Cloning considered harmful” considered harmful: patterns of cloning in software, *Empirical Softw. Eng.*, Vol. 13, No. 6(2008), pp. 645–692.
- [42] Kim, M., Bergman, L., Lau, T. and Notokin, D.: An Ethnographic Study of Copy and Paste Programming Practices in OOPL, in *Proc. of the 3rd Int. Symposium on Empirical Softw. Eng. (ISESE'04)*, 2004, pp. 83–92.
- [43] Kim, M. and Notokin, D.: Using a Clone Genealogy Extractor for Understanding and Supporting Evolution of Code Clones, in *Proc. of the 2nd Int. Workshop on Mining Softw. Repositories (MSR'05)*, 2005, pp. 1–5.
- [44] Kim, M., Sazawal, V., Notokin, D. and Murphy, G. C.: An Empirical Study of Code Clone Genealogies, in *Proc. of the 13th Int. Symposium on Foundations of Softw. Eng. (ESEC/FSE'05)*, 2005, pp. 187–196.
- [45] Koschke, R.: Survey of Research on Software Clones, in *Duplication, Redundancy, and Similarity in Softw., Dagstuhl Seminar*, 2006.
- [46] Koschke, R.: Frontiers on Software Clone Management, in *Proc. of the Frontiers of Softw. Maintenance in the 24th Int. Conf. on Softw. Maintenance (ICSM'08)*, 2008, pp. 119–128.
- [47] Krinke, J.: Is Cloned Code more stable than Non-Cloned Code?, in *Proc. of the 8th Int. Working Conf. on Source Code Analysis and Manipulation (SCAM'08)*, 2008, pp. 57–66.
- [48] Krinke, J.: Is Cloned Code Older than Non-Cloned Code?, in *Proc. of the 5th Int. Workshop on Softw. Clones (IWSC'11)*, 2011, pp. 28–33.
- [49] Laguë, B., Proulx, D., Mayrand, J., Merlo, E. M. and Hudepohl, J. P.: Assessing the Benefits of Incorporating Function Clone Detection in a Development Process, in *Proc. of the 13th Int. Conf. on Softw. Maintenance (ICSM'97)*, 1997, pp. 314–321.
- [50] Lanza, M. and Ducasse, S.: Polymetric Views – A Lightweight Visual Approach to Reverse Engineering, *IEEE Trans. on Softw. Eng.*, Vol. 29, No. 9(2003), pp. 782–795.
- [51] Li, J. and Ernst, M. D.: CBCD: Cloned Buggy Code Detector, in *Proc. of the 34th Int. Conf. on Softw. Eng. (ICSE'12)*, 2012, pp. 310–320.
- [52] Li, Z., Lu, S., Myagmar, S. and Zhou, Y.: CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code, *IEEE Trans. on Softw. Eng.*, Vol. 32, No. 3(2006), pp. 176–192.
- [53] Lozano, A. and Wermelinger, M.: Assessing the effect of clones on changeability, in *Proc. of the 23rd Int. Conf. on Softw. Maintenance (ICSM'08)*, 2008, pp. 227–236.
- [54] Lozano, A. and Wermelinger, M.: Tracking clones’ imprint, in *Proc. of the 4th Int. Workshop on Softw. Clones (IWSC'10)*, 2010, pp. 65–72.
- [55] Lozano, A., Wermelinger, M. and Nuseibeh, B.: Evaluating the harmfulness of cloning: a change based experiment, in *Proc. of the 4th Int. Workshop on Mining Softw. Repositories (MSR'07)*, 2007.
- [56] Lucia, Lo, D., Jiang, L. and Budi, A.: Active Refinement of Clone Anomaly Reports, in *Proc. of the 34th Int. Conf. on Softw. Eng. (ICSE'12)*, June 2012, pp. 397–407.
- [57] Merlo, E. and Lavoie, T.: Computing Structural Types of Clone Syntactic Blocks, in *Proc. of the 16th Working Conf. on Reverse Eng. (WCRE'09)*, 2009, pp. 274–278.
- [58] 宮崎宏海, 肥後芳樹, 井上克郎: 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, 電子情報通信学会技術研究報告, Vol. 108, No. 173, 2008, pp. 31–36.
- [59] Mondal, M., Roy, C. K., Rahman, M. S., Saha, R. K., Krinke, J. and Schneider, K. A.: Comparative Stability of Cloned and Non-cloned Code: An Empirical Study, in *Proc. of the 27th Annual ACM Symposium on Applied Computing (SAC'12)*, 2012, pp. 1227–1234.
- [60] 門田暁人, 佐藤慎一, 神谷年洋, 松本健一: 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.
- [61] Nguyen, H. A., Nguyen, T. T., Pham, N. H., Al-Kofahi, J. and Nguyen, T. N.: Clone Management for Evolving Software, *IEEE Trans. on Softw. Eng.*, Vol. 38, No. 5(2012), pp. 1008–1026.
- [62] Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. M. and Nguyen, T. N.: Clone-aware Configuration Management, in *Proc. of the 24th Int. Conf. on Automated Softw. Eng. (ASE'09)*, 2009, pp. 123–134.
- [63] Rahman, F., Bird, C. and Devanbu, P.: Clones: What is that Smell?, in *Proc. of the 7th Working Conf. on Mining Softw. Repositories (MSR'10)*, 2010, pp. 72–81.
- [64] Rieger, M., Ducasse, S. and Lanza, M.: Insights into System-Wide Code Duplication, in *Proc. of the 11th Working Conf. on Reverse Eng. (WCRE'04)*, 2004, pp. 100–109.
- [65] Roy, C. K. and Cordy, J. R.: A Survey on Software Clone Detection Research, School of Computing Technical Report 2007-541, Queen’s University, 2007.
- [66] Saha, R. K., Asaduzzaman, M., Zibran, M. F., Roy, C. K. and Schneider, K. A.: Evaluating Code Clone Genealogies at Release Level: An Empirical Study, in *Proc. of the 10th Working Conf. on Source Code Analysis and Manipulation (SCAM'10)*, 2010, pp. 87–96.
- [67] Saha, R. K., Roy, C. K. and Schneider, K. A.: Visualizing the Evolution of Code Clones, in *Proc. of the 5th Int. Workshop on Softw. Clones (IWSC'11)*, 2011, pp. 71–72.
- [68] 佐々木亨, 肥後芳樹, 神谷年洋, 楠本真二, 井上克郎: 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, Vol. J87-D-I, No. 9(2004), pp. 868–870.
- [69] Tairas, R. and Gray, J.: An Information Retrieval Process to Aid in the Analysis of Code Clones, *Empirical Softw. Eng.*, Vol. 14, No. 1(2009), pp. 33–56.
- [70] Thummalapenta, S., Cerulo, L., Aversano, L. and Penta, M. D.: An empirical study on the maintenance of source code clones, *Empirical Softw. Eng.*, Vol. 15, No. 1(2010), pp. 1–34.
- [71] Toomim, M., Begel, A. and Graham, S. L.: Managing Duplicated Code with Linked Editing, in *Proc. of the 2004 Symposium on Visual Languages and Human Centric Computing (VL-HCC'04)*, 2004, pp. 173–180.
- [72] 植田泰士, 神谷年洋, 楠本真二, 井上克郎: 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, Vol. J86-D-I, No. 12(2003), pp. 863–871.
- [73] Venkatasubramanyam, R. D., Singh, H. K. and Ravikanth, K.: A Method for Proactive Moderation of Code Clones in IDEs, in *Proc. of the 6th Int. Workshop on Softw. Clones (IWSC'12)*, 2012, pp. 62–66.
- [74] Vliet, H. V.: *Software Engineering: Principles and Practices*, West Sussex, England: John Wiley & Sons, 2000.
- [75] Wang, W. and Godfrey, M. W.: We Have All of the Clones, Now What? Toward Integrating Clone Analysis into Software Quality Assessment, in *Proc. of the 6th Int. Workshop on Softw. Clones (IWSC'12)*, 2012, pp. 88–89.
- [76] Wang, X., Dang, Y., Zhang, L., Zhang, D., Lan, E. and Mei, H.: Can I Clone This Piece of Code Here?, in *Proc. of the 27th Int. Conf. on Automated Softw. Eng (ASE'12)*, 2012, pp. 170–179.
- [77] Wattenberg, M.: Arc Diagrams: Visualizing Structure in String, in *Proc. of the IEEE Symposium on Information Visualization (InfoVis'02)*, 2002, pp. 110–116.
- [78] Xing, Z., Xue, Y. and Jarzabek, S.: CloneDifferentiator: Analyzing Clones by Differentiation, in *Proc. of the 26th Int. Conf. on Automated Softw. Eng. (ASE'11)*, 2011, pp. 576–579.
- [79] 山中裕樹, 崔恩潯, 吉田則裕, 井上克郎, 佐野建樹: コードクローン変更管理システムの開発と実プロジェクトへの適用, 情報処理学会論文誌, Vol. 54, No. 2(2013), pp. 883–893.
- [80] 楊嘉晨, 堀田圭佑, 肥後芳樹, 井垣宏, 楠本真二: 機械学習を用いた類似度に基づく有用なコードクローンの自動特定手法, 情報処理学会論文誌, Vol. 54, No. 2(2013), pp. 807–819.
- [81] 吉田則裕, 服部剛之, 早瀬康裕, 井上克郎: 類義語の特定に基づく類似コード片検索法, 情報処理学会論文誌, Vol. 50, No. 5(2009) pp. 1506–1519.
- [82] Yoshida, N., Ishio, T., Matsushita, M. and Inoue, K.: Retrieving Similar Code Fragments based on Identifier Similarity for Defect Detection, in *Proc. of the Int. Workshop on Defects in Large Softw. Systems (DEFECTS'08)*, 2008, pp. 41–42.
- [83] Yoshimura, K. and Mibe, R.: Visualizing Code Clone Outbreak: An Industrial Case Study, in *Proc. of the 6th Int. Workshop on Softw. Clones (IWSC'12)*, 2012, pp. 96–97.
- [84] Zhang, G., Peng, X. and Zhao, Z. X. W.: Cloning Practices: Why Developers Clone and What can be Changed, in *Proc. of the 28th Int. Conf. on Softw. Maintenance (ICSM'12)*, 2012, pp. 285–294.
- [85] Zhang, Y., Basit, H. A., Jarzabek, S., Anh, D. and Low, M.: Query-based Filtering and Graphical View Generation for Clone Analysis, in *Proc. of the 23rd Int. Conf. on Softw. Maintenance (ICSM'08)*, 2008, pp. 376–385.
- [86] Zibran, M. F. and Roy, C. K.: The Road of Software Clone Management: A Survey, Technical Report, Department of Computer Science, University of Saskatchewan, Canada, 2012.



堀田 圭佑

2010 年大阪大学基礎工学部情報科学科卒業。2012 年同大学大学院情報科学研究科博士前期課程修了。現在、同研究科博士後期課程在学中。日本学

術振興会特別研究員 DC。コードクローン分析に関する研究に従事。IEEE、情報処理学会、電子情報通信学会各会員。



肥後 芳樹

2002 年大阪大学基礎工学部情報科学科中退。2006 年同大学大学院情報科学研究科博士後期課程修了。2007 年同研究科コンピュータサイエンス専

攻助教。博士(情報科学)。ソースコード分析、特にコードクローン分析やリファクタリングに関する研究に従事。IEEE、情報処理学会、電子情報通信学会、日本ソフトウェア科学会各会員。



楠本 真二

1988 年大阪大学基礎工学部情報工学科卒業。1991 年同大学大学院博士課程中退。同年同大学基礎工学部情報工学科助手。1996 年同学科講師。1999 年同学科助教授。2002 年大阪大学大学院情報科学研

究科コンピュータサイエンス専攻助教授。2005 年同
専攻教授。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。IEEE, JFPUG, 情報処理学会, 電子情報通信
学会各会員。