# コードクローン検出ツールによる ソフトウェア開発成果物の可視化効果

## Visualization Efficacy of Software Development Deliverables with Code Clone Detection Tool

暮　井　　豊

**要　約**　近年，ソフトウェア開発の成果物の一つであるプログラム・ソースコードを「“コードクローン”と呼ぶ重複コード片」の存在状態に基づいて可視化することができる「コードクローン検出ツール」が登場してきた．

筆者は，日本ユニシスグループにおけるソフトウェア開発・保守プロジェクトの協力を得て，コードクローン検出ツールを大規模業務ソフトウェアに適用し，品質向上の観点から，その有効性について検証してきた．検証は，大規模業務ソフトウェアへの適用に際して必要となった機能，各プロジェクトでの適用目的に応じて必要となった機能を補完しながら進めた．その結果，コードクローン検出ツールの活用がソフトウェア品質の測定あるいは改善に対して必然的な効果をもたらすものとして位置づけるまでには至らなかった．しかし，いくつかの適用事例を通して，“既存ソフトウェアの改変や再構築時の計画・見積・実行”，“外注納品物の検収”などにおいて有益な情報を得る手段となることが確認できた．本稿では，これらの結果を踏まえ，現時点で有効と考える適用場面を提案し，その期待効果について述べる．

**Abstract**　In recent years, several “Code Clone Detection Tools”, which can visualize the program source codes as one of software development deliverables based on the existence state of duplicate code segments (called “Code Clones”), have been available.

The author has investigated efficacy of a Code Clone Detection Tool from the viewpoint of quality improvement by applying it to large-scale business software in cooperation with several projects of software development and maintenance in Nihon Unisys Group. This investigation was carried on through complementing the functionalities which were needed in applying a code clone tool to large-scale business software and depending on the application purposes in each project. As a result, our investigation stopped short of the stage where the utilization of the Code Clone Detection Tool is positioned as one of work to inevitably enhance the measurement and improvement of software quality through its effects. But, through several case studies, we confirmed that the utilization is capable of becoming a way of obtaining useful information for “planning, estimation and practice of the existing software modification and remake”, “acceptance inspection of the subcontract deliverables”, etc. This paper suggests some scenes where the code clone tool and several complimentary tools work effectively based on results of these investigations, and discusses the expectation effects that we will be obtained in those scenes.

## 1. は じ め に

情報システムが社会インフラとなった現在，「情報システムの信頼性・安全性」の確保は情報サービス・ソフトウェア産業のみならず，社会全体の関心事である．その意味でも，情報システムの構成要素の一つであるソフトウェアの「品質」の重要性はこれまでにも増して大きくなってきている．『JIS ソフトウェア製品の品質－第 1 部：品質モデル（JIS X 0129-1: 2003)』[1] では，ソフトウェアに関わる品質について，ソフトウェア製品の取得・供給・開発・運用・保守といった活動に関する品質である“プロセス品質”，ソフトウェア製品そのものの品質である“内部品質*1”，“外部品質*2”，“利用時の品質*3”の四つに分類して説明している．“プロセス品質”が“内部品質”，“外部品質”の向上に寄与し，さらにそれらが利用者*4 視点での品質である“利用時の品質”に寄与する関係にあると述べており，「製品を作るプロセスの品質が良ければ，その成果物である製品の品質も良くなる」という考えが根底にある．また，逆向きの関係として，“利用時の品質”の評価を製品品質の向上に反映させることができ，製品品質の評価を“プロセス品質”の向上に反映させることができるとも述べている．すなわち，製品品質の良し悪しは，プロセス品質の良し悪しと相互に影響あるいは依存し合う関係にあるということであり，プロセス自体も製品品質の評価により絶えず改善の対象となる．ソフトウェアの開発者の立場からみると，定められたプロセスに従って開発成果物自体の品質の向上を図る活動は“利用時の品質”の向上に寄与するために必要なことではあるが，成果物品質の新たな評価指標の有効性を検証し開発プロセスのさらなる改善を目指す活動も忘れてはならない．

近年，ソフトウェア開発における成果物の一つであるプログラム・ソースコードを「“コードクローン”と呼ぶ重複コード片」の存在状態に基づいて可視化することができる「コードクローン検出ツール」が登場してきた．これは，コード作成担当者のみならず，ソースコードのすべての内容に目を通すことが役割的にも時間的にも困難なプロジェクトマネジャのような管理者にとっても，ソースコードの状態を客観的に知り評価するための新たな手段を得たことを意味する．

本稿では，コードクローン検出ツールを日本ユニシスグループにおけるソフトウェアの開発・保守現場に適用し，プログラム・ソースコード自体の新たな評価手段として，上述したソフトウェアに関わる品質の向上の観点で効果を出せるか否かについて検証した内容を報告する．合わせて現時点で有効と考えるいくつかの適用場面を提案し，その期待効果について述べる．

まず 2 章でコードクローンに関する予備知識を与え，3 章でコードクローン検出ツールの適用事例を紹介し，その適用結果について考察する．最後に，3 章での考察を踏まえ，4 章でコードクローン検出結果の活用に対しての提案を行う．

## 2. コードクローンに関する予備知識

### 2. 1 コードクローンとは

ソフトウェア・プログラムのソースコード量は，保守性の観点からも，できるだけ少ない方が良い．しかし，大規模業務システムのソフトウェア開発プロジェクトともなれば，全体で数百万行といった膨大なコード量になり，その中には「全く同一のコード片」あるいは「類似したコード片」が少なからず存在する．このようなコード片のことを「コードクローン」と呼ぶ．

Bellon らは，コードクローンを表 1 に示す三つのタイプに分類している[2]．

**表1　コードクローンの分類**

| 種類 | 説明 |
|---|---|
| タイプ1 | 変更を伴わない完全な複製（空白や注釈の違いを除く） |
| タイプ2 | 変数，型，あるいは関数の識別名のみが変更された構文的に同一な複製 |
| タイプ3 | 文の変更，追加あるいは削除といった追加的な変更をともなった複製 |

コードクローンは，意図的な「標準化のパターン」として発生する場合だけではなく，新しい機能を追加する際に，既存のコードを安易にまるごと複製し必要部分を手直しすることで発生する場合がある．

### 2.2　コードクローン検出技術

コードクローンが存在するからといって必ずしもそのソフトウェアの品質が悪いとは断言できない．しかし，たとえば，あるプログラムの中で，不具合の原因となる箇所が発見され，かつその箇所を含むコード片とのクローンが存在していた場合，クローンとなっている他のコード片にもその不具合が存在していると見るべきである．不具合の原因を持つコード片のクローンを手作業で探すことは，ソフトウェアが大規模であればあるほど困難となってくるため，他の箇所に存在する同等の不具合を事前に発見し対応することが一層難しくなる．また，ファウラーは，リファクタリングと呼ばれる“ソースコードの内部構造の改善”の必要性を示す不吉な兆候の一つとして「重複したコード」すなわち「コードクローン」の存在を挙げている[3]．

不具合の内包やプログラム構造の問題は品質（“内部品質”，“外部品質”，および“利用時の品質”）にも関わる事項であり，コードクローンの存在がこれらに関係する場合もある．このような観点からも「コードクローン検出」は，ソフトウェア開発・保守において密接に関係する信頼性，使用性，あるいは保守性といった品質特性の向上に対して有用な情報を与える一つの手段と考える．

肥後らは，これまでに提案されているコードクローン検出技術を表2に示す五つに分類して紹介している[4]．

**表2　コードクローン検出技術の分類**

| 種類 | 説明 |
|---|---|
| 行単位の検出 | ソースコードの行そのものをコードクローン検出の単位とする |
| 字句単位の検出 | ソースコードを字句の列に変換し，しきい値以上連続して一致する部分列をコードクローンとして検出する |
| 抽象構文木を用いた検出 | ソースコードから抽象構文木を構築し，しきい値以上の大きさを持つ同形の部分木をコードクローンとして検出する |
| プログラム依存グラフを用いた検出 | ソースコード内のデータフローと制御フローからプログラム依存グラフを構築し，しきい値以上の大きさを持つ同形グラフをコードクローンとして検出する |
| メトリクスやフィンガープリントなど，その他の技術を用いた検出 | ソースコード内のモジュール単位でメトリクスを計測し，メトリクス値が類似しているモジュールをコードクローンとして検出する |

### 2.3　コードクローン検出ツール

今回使用したコードクローン検出ツールは，現在フリーウェアとして提供されている『CCFinderX』である[5]．本ツールは次のような特長を持つ．

- 採用しているコードクローン検出技術の種類は「字句単位の検出」である[*5]．
- 検出できるコードクローンの種類は「タイプ1」および「タイプ2」である[*6]．

- コード片を構成する字句数および字句種類数をしきい値として，しきい値以上の字句で構成され，かつ字句列として一致[*7] あるいは類似する[*8] コード片をコードクローンとして検出する．
- クローンとして認識されたコード片に含まれる最も外側のブロック[*9] の内部を最終的なコードクローンとして抽出することができる[*10]．
- 検出対象ソースコード・ファイルごとに“ファイルメトリクス”，“行ベースのメトリクス”を，検出されたコードクローンセット[*11] ごとに“クローンセットメトリクス”を算出できる（各メトリクスの具体的な内容については，『CCFinderX ver. 10.2 ドキュメント』[6]を参照）．
- 対象としたソースコード・ファイル群におけるコードクローンの存在箇所をプロット表示するコードクローン散布図（図 1）で視覚的にクローンの存在状態を把握できる．
- プログラミング言語は，COBOL，C++/C，C#，Java，Visual Basic に対応しており，言語ごとにユーザカスタマイズ可能な字句・構文解析用のプリプロセス処理スクリプトを標準添付している．


**図 1　コードクローン散布図例**

図 2 は，CCFinderX によるコードクローン検出の流れを示したものである．まず，ソースコードを入力に，プログラミング言語別に用意されたプリプロセス処理スクリプトに従って字句解析と構文解析を行い，各字句の使用文字列・種類・位置やブロック情報などをソースコード・ファイルごとにプリプロセス処理結果ファイルとして出力する．この各プリプロセス処理

結果ファイルの情報を基に一致あるいは類似する字句列部分を洗い出し，コードクローン情報としてクローンデータ・ファイルに出力する．さらに，クローンデータ・ファイル（およびプリプロセス処理結果ファイル）から各種メトリクスを算出できる．これら一連の操作はすべて，コマンドラインからCCFinderXのコマンド（ccfx.exe）を実行することにより行えるが，付属のフロントエンドGUIツール『GemX』を使用することで，各種処理の実行やコードクローン散布図等を含む実行結果の表示が対話的に行える．

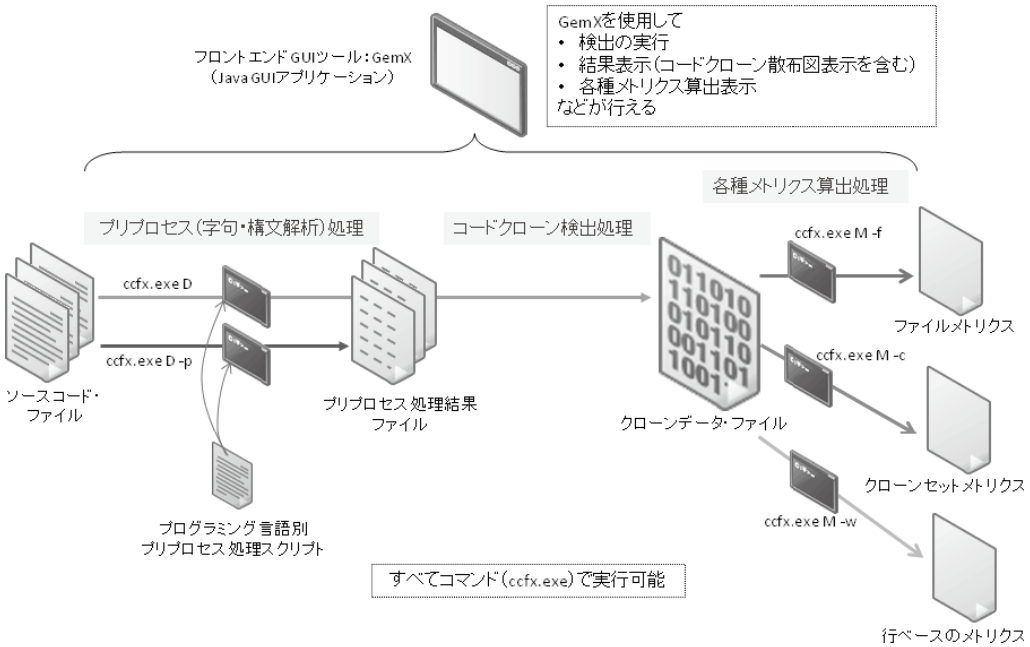


**図2　CCFinderXによるコードクローン検出の流れ**

CCFinderXを使用する上で，筆者が特に重視している利点は，次の3点である．

- 見える化：コードクローンの検出結果を散布図という直観的なイメージ図として表現することにより，ソースコードの内容を直接確認しない/できない場合でも，ソースコードの状態を概観可能としている（“ソフトウェア・プログラムのレントゲン写真”）．
- 高速：コードクローンの検出処理が高速であり，大規模業務システムのソフトウェア・プログラムに対しても十分実用に耐えうる処理速度である[*12]．
- 充実＆柔軟：主要なプログラミング言語に標準対応している．さらに，各プログラミング言語の字句・構文解析用のプリプロセス処理スクリプトが実行プログラムと分離しているため，カスタマイズや新たな言語用のスクリプト作成をユーザ側で行うことができる[*13]．また，プリプロセス処理結果やコードクローン検出結果のデータファイル書式が公開されているため，ユーザ側で独自の分析を行いたい場合の入力として活用できる．

### 2. 4　大規模業務ソフトウェア適用に向けた補完ツール

大規模業務システムのソフトウェア・プログラムへのCCFinderXの適用において認識した課題を解決し，CCFinderXを補完するためのツールを試作した（表3）.

**表3　CCFinderX適用時の課題とそれに対する補完機能**

| 課題 | 補完機能 |
|---|---|
| CCFinderX で検出したソースコード上のクローン位置の確認・共有に手間がかかる | クローン位置情報を一覧化する機能 |
| 自動生成コードやテンプレートコードの存在がクローン内容検証の際のノイズとなる | 特定コード部分をクローン検出対象から除外する機能 |
| 毎回手作業によるクローン検出では負担となる | コードクローン検出およびその他の関連情報出力を 1 回の操作で実行する機能 |

なお，この補完ツールは，新たに必要と考えたその他の機能を加えた上で「コードクローン情報編成ツール“Athanor（アタノール）for CCFinderX”」という名称で一般に公開し，誰でも自由にダウンロードして利用できるようにしている[7]．当該補完ツールの主な機能を表4に示す.

**表4　コードクローン情報編成ツール「Athanor」の主な機能**

| 機能 | 説明 |
|---|---|
| コードクローン位置情報一覧出力 | 各コードクローンがどのソースコード・ファイルのどの位置に存在しているかの情報を一覧ファイルとして出力 |
| コードクローン散布図画像出力 | 各コードクローンが対象ソースコード・ファイル群にどのように広がっているかを視覚的に確認可能とする散布図を画像ファイルとして出力 |
| コードクローンに基づくファイル間類似度情報出力 | コードクローン検出対象とするソースコード・ファイル群の任意の 2 ファイル間に存在するクローンがそれぞれのファイルに占める割合を算出（「コードクローンに基づくファイル間類似度」と定義）し，次の情報を出力：<br>● 類似度によるグルーピング情報<br>● 画像ベース類似度マトリクス<br>● 文字ベース類似度マトリクス |
| 類似コード片検索 | 指定した任意のコード片と類似（あるいは一致）するコード片を検索し，それらがどのソースコード・ファイルのどの位置に存在しているかの情報を一覧ファイルとして出力 |
| One オペレーション実行 | コードクローン検出の自動実行<br>以下の処理を 1 回の操作で実行するためのスクリプト群で構成：<br>● コードクローン検出に必要となる各種準備（“自動生成コード”や“テンプレートコード”といったクローンとして検出されても興味の対象となりにくいコード部分を検出の対象から除外するための機能を含む）<br>● CCFinderX によるプリプロセス，クローン検出，各種メトリクスデータ出力<br>● Athanor によるコードクローン位置情報一覧出力，コードクローン散布図画像出力<br>● クローンメトリクスに基づくコードクローン位置情報の分類処理<br>● クローンセット間の包含関係に基づくコードクローン位置情報の正規化<br>● コードクローンに基づくソフトウェア検診結果出力（対象ソフトウェアのプロファイル，コードクローン情報のサマリ，各種出力データファイルの一覧等を含む） |

## 3.　コードクローン検出事例の紹介と考察

### 3. 1　適用目的別事例紹介

これまでに日本ユニシスグループにおけるソフトウェア開発・保守プロジェクトにてコードクローン検出ツールを適用した事例の中から，適用目的別に主なものを紹介する[*14].

### 3.1.1　ソースコード全体のスリム化

【事例Ⅰ】

1)　適用対象：メインフレームCOBOLで開発された既存ソフトウェアをオープン系COBOLに変換した上で再開発中のソフトウェア・プログラム

2)　適用概要：検出したコードクローンセット群の中から部品化が見込めるセットを見つけ出す目的で，クローンセットメトリクスを用いた内容確認のための優先度付けを実施した．

ⅰ）コード量の削減効果が高くなる候補として，「“クローン長が大きく”かつ“該当するクローン箇所数が多い”クローンセット」を対象に調査[*15]を実施した．上位を占めたコードの内容が「WORKING-STORAGE SECTION部分」，「MOVE命令・CALL命令の繰り返し部分」であったこともあり，部品化に適するコード片は発見できなかった．

ⅱ）“データ項目定義”や“単純な命令”の繰り返しのみで構成されているクローンセットを除外するために，「“条件分岐やループなどの制御構造を多く含む”クローンセット」を対象とした調査[*16]を実施した．上位100件を調査した結果，部品化することでコード削減可能な候補を確認できたため，さらに件数を増やして調査を実施した．

3)　適用結果：コードクローン検出結果からの上記ⅱ）のサンプリング調査により，全体としては，コードの共通化によって最大10%のコード量削減が見込めるとの予測結果を得た．

【事例Ⅱ】

1)　適用対象：メインフレームで稼働中の既存ソフトウェアのオープン化対応のためにC#で新規開発中のソフトウェア・プログラム

2)　適用概要：事前にコード内容を確認し改善必要箇所を調査済みのソースコード・ファイル群を対象にコードクローン検出を行い，改善必要箇所とコードクローン検出箇所とを比較した．

ⅰ）「“あるまとまった処理の繰り返しを含む（ただし，単なる変数宣言や代入式等の連続コードは除く）”クローンセット」に着目して調査[*17]した．

ⅱ）「“ディレクトリ階層上近く”かつ“より多くのファイルに含まれている”クローンセット」に着目して調査[*18]した．

ⅲ）「“ディレクトリ階層上遠く”かつ“より少ないファイルに含まれている”クローンセット」に着目して調査[*19]した．

3)　適用結果：上記ⅰ），ⅲ）で着目したコードクローンセットが事前調査にて改善ポイントとして挙げていた箇所とほぼ一致していることが判明し，コードクローン検出結果がソースコード改善箇所の発見に役立つ場合があることを確認できた．なお，それらの該当箇所については，必要な部品を揃え対応することになった．

【事例Ⅲ】

1)　適用対象：C#で新規開発され，統合テスト中のソフトウェア・プログラム

2)　適用概要：今後保守を行う上での問題となり得るコードクローンの存在有無を確認するために，検出したコードクローンセット群のコード内容を調査した．

ⅰ）初回のコードクローン検出実行にかなりの時間を要した．コードクローン散布図から

判断しても，相当量のコードクローンが存在しており，対象ファイルの多くが互いにクローンを持ち合っていることが確認できた．さらに該当箇所のコード内容を調べたところ“自動生成コード”や“コーディング規約による固定的な実装コード”の存在がクローンの主要原因であることが判明した．

ⅱ）Athanor を適用し，自動生成コードおよび固定的な実装コードを除いた業務処理部分のみを対象としたコードクローン検出を実施した．

ⅲ）対象ソフトウェアを構成する各コンポーネントのうち，特に多くのコードクローンを検出したコンポーネントを対象にサンプリング調査を実施した．

3） 適用結果：サンプリングしたコードクローンセットの半数近くについてそれぞれなんらかの方法でコードの共通化が可能であることが確認できた．これらについては，いずれも現時点ですぐにでも改修を必要とするものではなかったが，今後の保守においてコードの精緻化のために留意しておくべき情報となった．しかし，コードの共通化が大幅なコード削減やプログラム構造の大幅な改善につながるなどの理由により開発段階で対処を迫られるような場合，テスト工程に入った状況においては，進捗や要員リソースの観点からも影響度合が非常に大きくなる．このことからも，より早い時点での調査が必要であることをあらためて認識できた．

### 3.1.2 外注納品物の検証

【事例Ⅳ】

1） 適用対象：外部に開発を発注したソフトウェア・プログラム

2） 適用概要：納品されたソースコード・ファイルの中でソースコード行数などのメトリクスがほぼ同一のファイル群が存在していたため，コードクローン検出を実施しその結果を検証した．

ⅰ）問題のファイル群を含め全体を対象にコードクローン検出を実行した．

ⅱ）問題のファイル群については，コードクローン散布図においても相互にクローンを持ち合っていることが認められ，コードクローン含有率も他のファイル群に比べ高い値を示した．

ⅲ）問題のファイル群におけるクローン箇所のコード内容を確認した．

3） 適用結果：共通化可能なクローンが存在していることが判明し改修を行った結果，より望ましいプログラム構造になるとともに，ファイル数およびコード量の削減につながった．

### 3.1.3 マイグレーション計画立案・見積・検証

【事例Ⅴ】

1） 適用対象：オープン系 COBOL で再構築を予定している，メインフレーム COBOL で開発された既存ソフトウェア・プログラム

2） 適用概要：ソースコード・ファイル間の類似関係がソフトウェア・マイグレーションの生産性を見積もる際の有用な情報となり得るかを検証するために，コードクローン検出結果からファイル間の類似度算出を試みた．

ⅰ）任意の二つのソースファイルの類似度合を「互いに共有するコードクローンセットに

属するコード片がそれぞれのファイルに占める割合」として定義し，これを算出する機能，および算出した類似度（百分率）によりファイルをグループ分けする機能を実装し，Athanor に組み込んだ．

ii）対象ソースファイル群に適用し，その結果を検証した．

3） 適用結果：コードクローンに基づくファイル間類似度の算出結果は，プロジェクト側で期待していたものに近い情報であることが確認できた．類似性の高いグループのファイルに対しては，そのマイグレーション作業自体も類似性が高いと仮定できるため，初期段階での見積作業において当該類似度情報は，分類済みの各グループからサンプリングしたファイルを調査することで，全体の見積や作業分担等の立案を補助する情報として活用できる可能性が高まった．

【事例Ⅵ】

1） 適用対象：オープン系プログラミング言語で再構築を予定している，メインフレーム FORTRAN で開発された既存ソフトウェア・プログラム

2） 適用概要：ソフトウェア・マイグレーションにおける開発の規模見積，実行可能性検証の一環として適用を試みた．既存プログラムには，コードを複製して一部だけを変更したような類似プログラムが多数あるとの事前情報があったが，コードクローン検出結果を用いて，実際にどの程度の類似コードが存在しているのかを調査した．

i）CCFinderX が FORTRAN に対応していないため，FORTRAN 用のプリプロセス処理スクリプトを作成し，コードクローン検出を実施した．

ii）Athanor によって出力したコードクローン位置情報一覧に基づいて，1000 ステップ以上からなるコードクローンについて該当するコード内容を調査した．

3） 適用結果：的確に類似コード部分が抽出できていることが確認でき，これらの部分については，再開発の際にうまく共通化することにより開発工数を削減することが可能との見込みができた．

【事例Ⅶ】

1） 適用対象：メインフレーム COBOL で開発された既存ソフトウェアをオープン系 COBOL に変換・再構築し，テスト段階に入っているソフトウェア・プログラム

2） 適用概要：ソフトウェア・マイグレーション実施過程でのオープン系 COBOL への一括変換後の個別修正に関して，その実施漏れ・実施誤りを検出するためにコードクローン検出結果が利用できないかを検証した．

i）個別修正は，類似コード部分に対しては同一の修正パターンで実施するため，個別修正前のプログラム群においてコードクローンとして検出した箇所は，修正後もコードクローンとして検出される可能性が高い．そこで，「個別修正前後での各プログラム群に対する“コードクローンに基づくファイル間類似度算出”の結果を比較することで，その差異から修正漏れ・修正誤り候補ファイルの洗い出しができる」という仮説に基づいて調査を実施した．

ii）さらに，修正前コード・イメージと修正後コード・イメージがパターンとして揺れがないものについては，Athanor の類似コード片検索機能を使用して，個別修正前後のプログラム群での検索結果を比較することで，その差異から修正漏れ・修正誤りの洗い出しを試みた．

3） 適用結果：上記ⅰ）においては，修正前と修正後で類似度に大きな差があるファイルをサンプリングし調査したところ，修正漏れ・修正誤りを疑えるファイルを発見することができた．上記ⅱ）においては，CCFinderX のクローン認識方法に従った検索を行うため，コード片を構成する字句の種類と出現順序が一致してさえいれば，コード片に含まれている手続き名やデータ項目名のようなユーザ定義識別子が文字面として一致しているか否かにかかわらず，検索結果に含まれる．しかし実際には，手続き名は文字面として一致させたいがその引数であるデータ項目名は異なっていてもよいといった検索を行いたい場合も多く，利用の範囲が限られてしまうことが判明した[*20]．

### 3.2 適用事例とその結果に対する考察

ソースコード全体のスリム化を目的に適用した事例においては，コードクローン検出結果から実際のコード内容を確認する際に，確認対象を選出するために用いた指標には違いがあったものの，共通化等により削減可能なコード片の候補を発見することができた．ただし，これは次のことに大きく起因している．

- クローン箇所のコード確認作業にあらかじめ時間と要員を割り当てていた
- コード内容にある程度精通している要員が確認にあたることができた

一般的には，ある指標で絞り込みを行ったとしても絞り込み後のコードクローンセットは依然としてかなりの数量になり得る．その中の数割が共通化可能なものだとしても残りのクローン部分も確認してはじめて洗い出せるため，全体としてはそれ相応の手間をかけることになる．さらに，適切な判断を行うには，単にプログラミング・スキルを有するだけではなく，対象とするソフトウェアに精通した要員による確認が必要となる．紹介した事例の中で用いた各指標は，採用しているプログラミング言語やソフトウェアの開発方法によってその有効性に違いがでてくる可能性がある．いずれにせよ，現時点では，常に効果を発揮する指標がないため，コードクローン検出結果を一つのきっかけとして地道に確認作業を行うことが必要となる．

また，多くの事例は，既にソフトウェアを構成するプログラムが存在している状態であった．プロジェクトの内容と工程によっては，コードクローンの検出結果からコードの改善を行うことは，プロジェクト進捗や要員リソースに大きな影響を与えかねない場合がある．たとえば，事例Ⅲのようにすでに統合テストに入っている場合には，ソフトウェアが正しく動作する限り，その時点での改善には踏み切らないと考える．事例Ⅰ，Ⅱのように既存ソフトウェアの再構築というタイミングであれば，そのような問題はなく，再構築後の保守を考えれば，この時点で見直して改善しておくべきである．新規開発プロジェクトの場合には，初期の段階から定期的にコードクローン検出を行うことで，意図していない類似コード発生の早期発見により手戻りを最小限に抑え，継続的な検査により改善すべきコードクローンか否かを効率的に判断できるようにしておくことが必要である．

外注納品物の検証を目的に適用した事例Ⅳでは，問題点の気づきの発端は通常的に行われているソースコード・メトリクスを確認することで得られたが，コードクローン検出結果を合わせて確認することで，冗長なコード部分の発見を効率的に行えた好事例である．このように他の情報とコードクローン散布図などのクローン検出結果を一緒に見ることで，通常では気づかない事象を効率的に発見できる可能性があるため，特に，外部に発注したものを検証する際の道具の一つに加えることは有効と考える．この事例以外にも，対象ソフトウェア・プログラム

の構造をある程度知っている技術者がコードクローン散布図を見ることよって気づきを得る場合があるとの意見があった．これは，設計や開発方針を把握しているプロジェクトマネジャ，アーキテクトなどにとって，ソースコードの内容まで踏み込まなくとも，問題のありそうな箇所に気づく可能性を与えてくれる情報であることを意味するものと考える．

マイグレーション計画立案・見積・検証を目的とした事例においては，事例Ⅵのようなケースでは，対象とする情報システムが「古くから運用されているもの」あるいは「開発・運用に携わっていなかったもの」であった場合，現状に則した仕様書が揃わないことがある．そのような場合，実際のソースコードから仕様を解析する作業が必要となるが，コードクローン情報を活用することにより，同一内容のコードを複数回解析するといったことを回避できることにもなる．

また，事例Ⅶは，修正前と修正後の二つの時点でのコードクローン検出情報を利用するという点で，他の事例とは異なる．さらに，その利用目的も修正漏れや修正誤りの検出という新たな視点での試行である．サンプリング調査の結果としては一部その有効性を示す結果が得られたが，全体として目的に対してどの程度貢献可能であるかについては，調査の手順や確認方法も含めその正当性をさらに検証する必要がある．

## 4. コードクローン検出結果活用に対する提案

3章で紹介した事例をはじめとしてこれまでの適用結果を踏まえ，現時点において，コードクローン検出結果の活用に有効性があると考える場面とそこでの期待効果を表5にまとめた．

**表5　コードクローン検出結果の有効な活用場面と期待効果**

| 活用場面 | 期待効果 |
|---|---|
| マイグレーション | 冗長な類似コードの発見とそのリファクタリングによるコード量の低減，プログラム構造の精緻化 |
| | ソースコード・ファイルの類似度合による分類を活用した計画立案・見積の効率化と精度の向上 |
| 構成管理<br>品質管理<br>外注納品物検収<br>プロジェクト管理 | 成果物の状態把握による問題点の早期発見・早期対応（予期せぬ冗長な類似コードの混入の発見とそのフィードバックによるコード肥大化の防止，プログラム構造の精緻化など） |
| 改造・不具合修正 | 改造・不具合発生時に修正が必要となる類似コード箇所の洗い出しによる対応の迅速化 |

検出結果から有益な成果を生み出すためには，対象となるソフトウェア・プログラムに精通した要員のスキルに依存しなければならない部分が大きい．しかし，直観的な“コードクローン散布図”や具体的な“コードクローン位置情報”による可視化によって，これまで「ソースコードの内容を詳しく確認しなければ気づかなかった問題」あるいは「ソースコードを見ても気づかずに終わっていたかもしれない問題」に対しても，目を向けるきっかけとなる新たな機会を得たとの見方もできる．

“コードクローン散布図”におけるクローン箇所の広がりや集中の度合い，“コードクローン含有率”などの各種メトリクスが，プログラムの品質を測る際の有効な指標となればという期待を持っていたが，今回は，成果物の品質を直接的・客観的に測る手段として「コードクローン検出」を位置付けるまでには至らなかった．しかし，表5に挙げた場面での活用はそこで扱

うソフトウェアに関わる品質を向上させるための活動に少なからず貢献できることが，いくつかの事例を通して確認できた．したがって，まずは表5に挙げた場面での活用から着手することを提案したい．そうすることで，そこで得られた成果や経験を踏まえた新たな活用場面や活用方法についての検討につながるものと考える．

## 5. お わ り に

プログラム・ソースコードは，開発活動における出力（成果物）であり，保守活動においてはその入力ともなる．また，ソフトウェア・マイグレーションにおいては，開発活動のみならず企画活動の重要な入力の一つとなる．3章，4章を通じて，コードクローン検出結果を活用することで，それらの活動を効果的に支援できる場面があることを述べた．このことは，コードクローン検出という新たな手段でソースコード自体を評価することが，1章で述べた「ソフトウェアに関わる品質」の向上に貢献できる場面があることを意味している．ただし，現時点でコードクローン検出結果をうまく活用するには，活用側にも対象とするソフトウェアに対する知見を有することが必要である．これをどのようなプロジェクトでも容易に活用できるようにするためには，目的に合致したクローンを優先的に見つけ出すような指標が必要になる．ただし，これは，コードクローン検出結果だけでは得られないかもしれない．障害情報，修正情報などのプロジェクトで管理している他の情報との組み合わせやクローン情報も含めた各情報の時系列推移を組み合わせることで得られる可能性もある．それを検証できるようにするためにも，現時点での具体的な利用目的の有無にかかわらず，ソースコードさえあれば容易に実行できるコードクローン検出を継続的に実施し，メトリクスの一つとして結果を蓄積しておくことが必要であると考える．

コードクローンについては，その検出方法から検出結果の有効活用まで様々な研究が行われている．コード作成やコード修正時にタイムリーにコードクローン情報を利用する研究[11]も行われており，今回対象にできなかったコーディング段階での活用にも期待が持てる．今後は，そのような外部の研究動向や成果を注視しつつ，より多くのプロジェクトでの適用を通じて有効な活用方法について検証するとともに，補完ツールの拡充を行いたい．

最後に，コードクローン検出ツールの適用に協力をいただいた各プロジェクトの担当者，適用時に発見した不具合や要望に迅速に対応いただいたCCFinderXの開発者である神谷年洋氏[*21]，そして本稿の執筆にあたり適切な助言をくださった関係者の方々に感謝申し上げたい．

---

＊1　JIS X 0129-1: 2003[1]では，“内部品質”を『特定の条件下で使用される場合に，明示的及び暗示的必要性を満たす製品の能力を決定する製品の属性の全体』と定義し，ソフトウェアを実行する前の中間製品（例えば，プログラム・ソースコードなど）に関する品質を指している．

＊2　JIS X 0129-1: 2003[1]では，“外部品質”を『製品が指定された条件下で利用された場合に，明示的及び暗示的必要性を満足させる程度』と定義し．ソフトウェア実行時のプログラムの振る舞いに関する品質を指している．

＊3　JIS X 0129-1: 2003[1]では，“利用時の品質”を『指定された利用者が仕様化された特定の仕方で製品を利用したとき，仕様化された目的を達成するために，有効性，生産性及び満足度を伴い必要性を満たしている程度』と定義し．特定の環境および特定の利用状況で利用されるときの，利用者の視点でのソフトウェア製品の品質を指している．

＊4　JIS X 0129-1: 2003[1]では，“利用者”を『特定の機能を遂行するために，ソフトウェア製品を使う個人（利用者には，オペレータ，ソフトウェアの結果の受入者，ソフトウェアの開発者又は保守者を含めてもよい）』と定義し，『“利用者”は，オペレータ及び保守者の両方を

含む想定できるあらゆる種別の利用者であり，かつその要求は異なることがある』と述べている．

＊ 5　コードクローン検出技術の分類については 2.2 節の表 2 を参照のこと．
＊ 6　コードクローンの分類については 2.1 節の表 1 を参照のこと．
＊ 7　「字句列として一致する」とは，コード片を構成する字句の並び順で互いに対応し合う字句の文字面がそれぞれ一致することを指す．
＊ 8　「字句列として類似する」とは，コード片を構成する字句の並び順で互いに対応し合う字句がユーザ定義識別子（クラス名，メソッド名，関数名，手続き名，変数名，データ項目名，リテラルなど）の場合にはその字句種類が，それ以外の場合にはその文字面がそれぞれ一致することを指す．
＊ 9　ブロックとは，関数定義ブロック，条件分岐ブロック，ループブロックなどを指す．
＊10　ブロック内のコード部分に制限することで，より意味のあるまとまりをクローンとして提示できる．
＊11　コードクローンセットとは，同一のクローンとして検出されたコード片の集合．
＊12　もともと非常に多くのコードクローンを持つプログラム群においては，検出処理にかなりの時間を要することがある．その場合，検出結果を付属のフロントエンド GUI ツール『GemX』で参照する際の操作処理速度が極端に低下するため，注意が必要である．
＊13　プリプロセス処理スクリプトの詳細な記法についてのマニュアルがないため，誰もが容易にカスタマイズや作成を行えるわけではないが，標準提供のスクリプトを見本として対応することは可能である．
＊14　これら事例の多くは，筆者がコードクローン検出ツールの適用支援を行い，検出したコードクローンの内容調査についてはプロジェクト側で実施した．なお，プロジェクト側で自主的に適用し，事後にその成果について報告を受けたものも一部含まれている．
＊15　クローンセットメトリクスの LEN（クローン長）と POP（クローンになっているコードの断片数）の乗算値（LEN × POP）の大きい順に調査．
＊16　クローンセットメトリクスの McCabe（クローンのコードに含まれるループおよび条件分岐の数）が大きいものを対象とした調査．
＊17　クローンメトリクスの RNR（繰り返し部分に含まれない字句の割合）が小さい（すなわち，コードを構成する字句のほとんどが繰り返し部分に含まれている），McCabe が 0 より大きい（ループや条件分岐を少なくとも一つ含む，すなわち，単なる変数宣言や代入式のみからなるコードではない）ものを対象とした調査．
＊18　クローンメトリクスの RAD（該当クローンのコード片を持つファイルがディレクトリ階層上どれだけ広がっているか）が小さい（すなわち，同一サブシステム内に存在している），NIF（該当クローンを持つファイルの個数）が大きい（すなわち，該当クローンを持つファイルが多い）ものを対象とした調査．肥後・服部らは，この対象を「リファクタリングを検討すべき候補」として分類している[9][10]．
＊19　クローンメトリクスの RAD が大きい（すなわち，異なるサブシステム間に存在している），NIF が小さい（すなわち，該当クローンを持つファイルが少ない）ものを対象とした調査．肥後・服部らは，この対象を「設計情報との一貫性を確認すべき候補」として分類している[9][10]．
＊20　この課題を解決するために，新たな類似コード片検索ツールを試作した．なお，当該ツールでは，一部の字句が挿入・置換・削除されたような類似コード片の検索も可能となっている[8]．
＊21　現在，「独立行政法人 産業技術総合研究所」に所属．

**参考文献**
［1］ 日本工業標準調査会 審議，「JIS ソフトウェア製品の品質―第 1 部：品質モデル JIS X 0129-1: 2003」，日本規格協会，2003 年
［2］ Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo, “Comparison and Evaluation of Clone Detection Tools”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL.33, NO. 9, SEPTEMBER 2007, 577-591
［3］ マーチン・ファウラー（児玉公信 他訳），「リファクタリング プログラミングの体質改善テクニック」，ピアソン・エデュケーション，2000 年，pp.75-76
［4］ 肥後 芳樹，楠本 真二，井上 克郎，「コードクローン検出とその関連技術」，電子情報通信学会論文誌，D Vol. J91-D，No.6，2008 年，pp.1465-1481
［5］ “CCFinder ホームページ”，http://www.ccfinder.net/ccfinderx-j.html
［6］ “CCFinderX ver. 10.2 ドキュメント”，http://www.ccfinder.net/doc/10.2/ja/index.html
［7］ “コードクローン情報編成ツール Athanor for CCFinderX”，http://dev.tyzoh.jp/trac/info-extraction/wiki/CodeCloneInfo
［8］ “AthanorEX”，http://dev.tyzoh.jp/trac/info-extraction/wiki/AthanorEX

［9］ 肥後 芳樹，吉田 規裕，楠本 真二，井上 克郎，「Geminiを用いた効果的なコードクローン分析方法」，ソフトウェア工学工房 第7回セミナー コードクローン検出技術とその応用，2006年，
http://sel.ics.es.osaka-u.ac.jp/~lab-db/betuzuri/archive/573/573.ppt
［10］ 服部 剛之，肥後 芳樹，楠本 真二，井上 克郎，「コードクローンの分布情報を用いた特徴抽出手法の提案」，ソフトウェア信頼性研究会 第3回ワークショップ論文集，pp.9-17，2006年，http://sel.ics.es.osaka-u.ac.jp/~lab-db/betuzuri/archive/590/590.pdf
［11］ 山科 隆伸，上野 秀剛，伏田 享平，亀井 靖高，名倉 正剛，川口 真司，飯田 元，「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」，電子情報通信学会技術研究報告，Vol.108，No.64，2008年，pp.65-70

**執筆者紹介** 暮 井 豊（Yutaka Kurei）
1989年日本ユニシス(株)入社．メインフレーム系ミドルウェアの開発に従事し，1995年よりオープン系ミドルウェアの企画・販促・開発に携わる．その後，海外製ソリューションの国内顧客への導入支援などを経験し，現在は総合技術研究所先端技術部に所属．