Code Clone Analysis Environment for Supporting Software Development and Maintenance

Yasushi Ueda[†], Toshihiro Kamiya[††], Shinji Kusumoto[†], and Katsuro Inoue[†]

[†] Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560-8531, Japan
[††] PRESTO, Japan Science and Technology Corporation, JAPAN

Abstract

It is generally said that code clones are one of the factors to make software maintenace difficult. They are pairs or sets of code portions in source files that are identical or similar to each other. If some bugs are found in a code portion, it is nessesary to correct them in its all clones. However, in large-scale sofrware, it is very difficult to find all clones by hand. In this paper, we develop a code clone analysis environment, Gemini, that uses a clone detection tool, CCFinder. Also, we conduct a case study for the evaluation.

keyword

Software maintenance, Code clone, Software metrics

論　文

開発保守支援を目指したコードクローン分析環境

植田　泰士[†]　　神谷　年洋[††]　　楠本　真二[†]　　井上　克郎[†]

Code Clone Analysis Environment for Supporting Software Development and Maintenance

Yasushi UEDA[†], Toshihiro KAMIYA[††], Shinji KUSUMOTO[†], and Katsuro INOUE[†]

あらまし　ソフトウェアの保守作業を難しくしている要因の一つとしてコードクローンがある．コードクローンとは，ソースコード中の同一，または類似した部分を指す．あるコード片にバグが含まれていた場合には，そのコード片のコードクローン全てについて修正の是非を考慮する必要がある．しかし，大規模なソフトウェアの場合，それら全ての箇所を手作業で発見し，修正の是非を考慮することは非常に困難である．本研究では，コードクローン検出ツール CCFinder の検出結果を利用したコードクローン分析環境 Gemini の構築を行う．本システムは，開発保守におけるコードクローンの利用を支援するため，コードクローンの位置情報の視覚化，コードクローンに関するメトリクス値の算出，および対応したソースコードを参照する機能を実装する．適用実験の結果，本システムを用いることで，特徴的なコードクローンなどを抽出できることを確認した．

キーワード　ソフトウェア保守，コードクローン，ソフトウェアメトリクス

1. まえがき

ソフトウェア保守とは,「納入後，ソフトウェア・プロダクトに対して加えられる，フォールトの修正，性能またはその他の性質改善，変更された環境に対するプロダクトの適応のための改訂」であると定義されている [9]．近年，ソフトウェアシステムの大規模化，複雑化に伴い，ソフトウェアの保守・デバッグ作業に要するコストは増大しており，大企業では既存システムの保守に多くのコストを費やすようになった．

コードクローンは保守作業を困難にする要因の一つである．コードクローンとは，ソースコード中の同一，または類似した部分のことであり,「重複コード」とも呼ばれる．コードクローンがソフトウェア中に作り込まれる原因として，既存コードのコピーとペーストによる再利用，頻繁に用いられる定型処理，パフォーマンス改善のための意図的な繰り返し，コード生成ツールによって生成されたコードなどがある．

[†] 大阪大学 大学院情報科学研究科，豊中市
Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560-8531 Japan
[††] 科学技術振興事業団 若手個人研究推進事業
PRESTO, Japan Science and Technology Corporation, Japan

コードクローンの存在は保守作業を困難にする [7]．これは，修正されるコード片がコードクローンであれば，すべての類似コード片を何らかの手段で同様に修正する必要性が生じるためである．対策としては，

- コードクローン情報を文書化しておくことで一貫した修正を可能にする，
- コードクローンを必要に応じて検出する

の 2 つがある [10]．

しかし，前者の方法はコードクローン情報を常に最新の状態に保つ手間がかかり，実現困難である．後者の方法として，いくつかのコードクローン検出手法やツールが提案されている [1] [2] [3] [4] [5] [6] [12] [13] [14]．

我々が開発してきているコードクローン検出ツール CCFinder [11] は，大規模なソフトウェアから実用的な時間でコードクローンを検出することが可能である．しかし，(1)CCFinder の出力はコードクローンの位置情報（ファイル名，行，カラム）であり，直感的な理解が困難である．また，(2) 大規模なソフトウェアから検出された膨大なコードクローンを分類・整理する機能が提供されないなど，実際の保守作業に利用するには問題があった．

本論文では，大規模ソフトウェアにおけるコードクローンの分析を行うための統合環境 Gemini を構築す

る．Gemini は内部的に CCFinder を利用してコードクローンの検出を行い，様々なユーザインタフェースを通して，ソフトウェア技術者に対して有益なコードクローン情報を視覚的に提示する機能を備えている．また，本システムを大学におけるプログラミング演習で開発されたプログラムに適用するケーススタディにより，その有用性を評価する．

以降，2. 節では，コードクローン検出ツール CCFinder，および Gemini において用いられるコードクローンに関するソフトウェアメトリクス等について説明する．3. 節では，Gemini の設計と実装について，4. 節では，適用実験の結果とその分析と考察について述べる．最後に 5. 節で，まとめと今後の課題を述べる．

2. 準　　備

2.1 コードクローン検出ツール CCFinder

本節では，Gemini がコードクローン検出に利用するツールである CCFinder に関する説明を行う．

2.1.1 コードクローンの定義と関連用語

あるトークン列中に存在する 2 つの部分トークン列 α, β が等価であるとき，α と β は互いにクローンであるという．また，ペア (α, β) をクローンペアと呼ぶ．α，β それぞれを真に包含するいかなるトークン列も等価ではないとき，α，β を極大クローンという．また，クローンの同値類をクローンクラスと呼ぶ．ソースコード中でのクローンを特にコードクローンという．

2.1.2 コードクローン検出処理手順

CCFinder は単一または複数のソースファイル中から，極大クローンのクローンペアを検出し，その位置情報を出力する．処理は以下の 4 つのステップからなる．

ステップ 1(字句解析): ソースファイルを字句解析 (lexical analysis) によってトークン列に変換する．入力ファイルが複数の場合には，個々のファイルから得られたトークン列を連結した単一のトークン列を生成する．

ステップ 2(変換処理): 実用上意味を持たないコードクローンを取り除くこと，および，些細な表記上の違いを吸収することを目的とした変形ルールによって，トークン列を変換する．例えば，この変換によって変数名は同一のトークンに置換されるので，変数名が付け替えられたコード片もコードクローンであると判定することができる．

ステップ 3(検出処理): トークン列の中から，指定された長さ以上一致しているような部分クローン列をすべて検出する．

ステップ 4(出力整形処理): 検出されたクローンペアについて，元のソースコード上での位置情報を出力する．

2.2 コードクローンに関するメトリクス

Gemini ではコードクローンを識別するために，以下の 6 つのメトリクスを利用する．

LEN(C) (Length) [11]:
クローンクラス C 内に含まれるコード片の最大トークン長を表す．

POP(C) (Population) [11]:
クローンクラス C 内のコード片の数を表す．

DFL(C) (Deflation) [11]:
クローンクラス C を再構築した場合に減少するコードの量の予測値を表す．ここで，再構築とは，クローンクラス C 内のコード片集合から抜き出した共通のロジックを実装するサブルーチンを作り，各コード片をそのサブルーチンの呼び出しに置き換えることである．$DFL(C)$ は，再構築前のコードの大きさ (すなわち，クローンクラス C 内のコード片の大きさの和) から，構築後のコードの大きさ (すなわち，サブルーチン呼び出しの大きさの大きさの和+共通ロジックを実装するサブルーチンの大きさ) を引いたものとして定義される．実際には，クローンクラスのなかには再構築が不可能なものも存在するので，DFL は再構築を行うべきクローンクラスの優先順位を付けるために用いることを想定している．

RAD(C) (Radius of clone class) [11]:
ソースファイルが階層型ファイルシステムに格納されている場合に，クローンクラスのファイルシステム内での「広がり」を表す．クローンクラス C 内のコード片を含むソースファイルの集合を F として，F の各要素のファイルパス全てに共通した最も下位 (ルートから遠い) ディレクトリから，各要素のファイルまでの距離の最大値と定義する．

直観的には，RAD 値が低ければ，同一ファイル内や同一ディレクトリに閉じているため，そのクローンクラスをサブルーチンに置き換えることは比較的容易であると考えられる (修正を行う場合も同様)．逆に，RAD 値が高ければ，それらのコード片が様々なモジュールに分散している，あるいは，(特に複数の開発者が関わる場合は) 異なる開発者のモジュールに含

まれるなど，修正の難易度は高くなると考えられる．

RSA(f) (Ratio of similarity to all other files):
RSA は，ファイル f と，ファイル f 以外の全てのファイルそれぞれとの間に存在するクローンペアによって，ファイル f のコードがどれだけカバーされているかの割合を表し，次の式で定義される．

$$RSA(f) = \frac{1}{Len(f)} \sum_{cf \in CF(f)} Len(cf)$$

($Len(f)$: ファイル f のトークン長)

($Len(cf)$: コード片 cf のトークン長)

ここで，$CF(f)$ は，ファイル f 以外のファイルとクローンの関係を持つファイル f 内のコード片の集合を表し，cf は 1 コード片を表す．ただし，本総和においては，重複しているコード部分は，1 度しか考慮しないものとする．

高い RSA を持ったファイルが生成する原因の一つに，開発者が，あるソースファイル全体をコピーした上で，一部を修正して再利用するといった作業を行うことがある．Gemini において RSA は，コードクローン情報を視覚的に表示するために利用される (3.2.2 節参照).

RST(f_1, f_2) (Ratio of similarity between two files):
$RST(f_1, f_2)$ は，ファイル f_1 がファイル f_2 とのクローンペアによってどれだけカバーされているかの割合を表し，次の式で定義される．

$$RST(f_1, f_2) = \frac{1}{Len(f_1)} \sum_{cf \in CF(f_1, f_2)} Len(cf)$$

ここで，$CF(f_1, f_2)$ は，ファイル f_2 とクローンの関係を持つファイル f_1 内のコード片の集合を表す．RSA 同様，本総和においては，重複しているコード部分は，1 度しか考慮しないものとする．

RST は，2 ファイル間の類似度として用いることができる．もし f_1，f_2 が，あるソースファイルの 2 つのバージョンであれば，RST の計測値はバージョン間の類似度を意味する．計測値が小さければ，バージョンアップの際に多くの変更が加えられた可能性がある．

Gemini において RST は，RSA 同様，コードクローン情報を効果的に視覚化するために利用される (3.2.2 節参照).

3. コードクローン分析環境 Gemini

3.1 設計方針

これまでに様々なコードクローン検出ツールが開発されてきている．中でも，Duploc [6] はコードクローン情報を，クローン散布図を用いて視覚化する機能を持つ．クローン散布図はクローンペアを視覚的にする図であり，プロットされたそれぞれの点は，その座標に対応する行が一致していることを表す (詳細は 3.2.2 節で述べる)．Duploc では，クローン散布図がウィンドウ内に表示され，点をクリックすることで対応したソースコードを参照することができる．

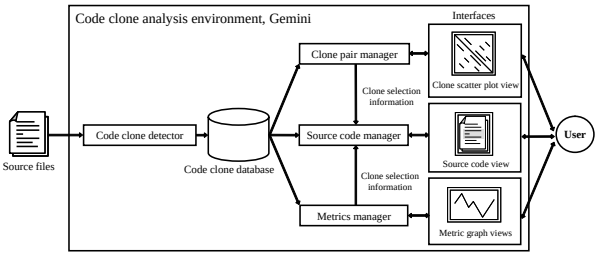


図 1　システムの構成
Fig. 1　Architecture

クローン散布図を使うことで，クローンペア全体の分布状態を俯瞰的に表示し，ソースコードを素早く参照することが可能となるなど，対話的操作に有効な視覚化手法であるため，Gemini においても，クローン散布図を GUI の一部として利用している．

ただし，本研究が想定するような大規模なソフトウェアのコードクローン分析においては散布図は膨大な数の点を含むため，単にクローン散布図を表示するだけではなく，着目すべき部分を強調する，フィルタリングを行うなど，利用者を補助するためのさらなる方法が必要である．本課題に対して提案する我々の手法は，3.2.2 節で述べる．

また，Gemini はクローン散布図以外にも，2.2 節で定義したコードクローンメトリクスのグラフを表示する機能を持つ．これらのグラフは，クローンクラスやファイルを対象としており，クローンペアを表示するクローン散布図とは異なった対話的な分析機能を提供する．詳細は 3.2.3 節で述べる．

3.2 システム構成

本システムは，内部的に CCFinder を実行し，CCFinder から得られた解析結果を基に分析環境を提供する．システムの構成を図 1 に示す．

Gemini は主に 5 つのコンポーネント: (1) コードクローン検出部 (Code clone detector)，(2) クローンペア管理部 (Clone pair manager)，(3) メトリクス管理部 (Metrics manager)，(4) ソースコード管理部

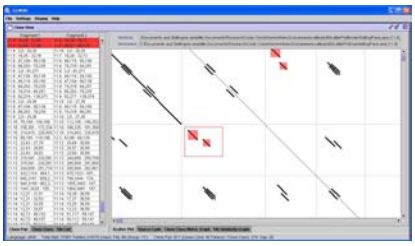


図 2　クローン散布図ビューの表示例
Fig. 2　GUI snapshot of clone scatter plot view

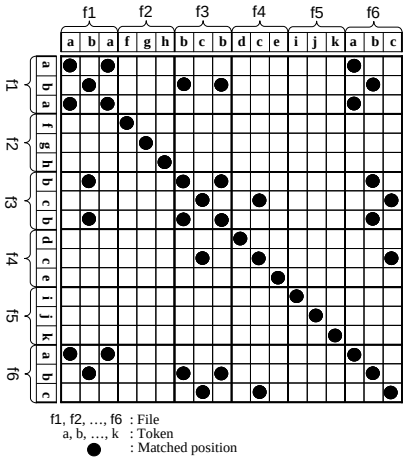


図 3　クローン散布図モデル
Fig. 3　Simple model of clone scatter plot

(Source code manager)，(5) ユーザインターフェースで構成されている．

まず始めに，コードクローン検出部にソースファイルが入力され，全てのコードクローンを検出する．次に，クローンペア管理部と，メトリクス管理部がその解析結果であるコードクローン情報を各インターフェースを通して視覚化する．それらのインターフェース上では，ユーザは任意のクローンペア (あるいは，クローンクラス) を選択することができ，その選択によって，実際のソースコードをソースコード管理部とそのインターフェースを通して参照することができる．

次節から図 1 における各コンポーネントについて順に述べる．

3.2.1　コードクローン検出部

解析対象のファイルや，コードクローン検出のパラメータを管理する．CCFinder を利用して検出したコードクローンの位置情報も管理する．

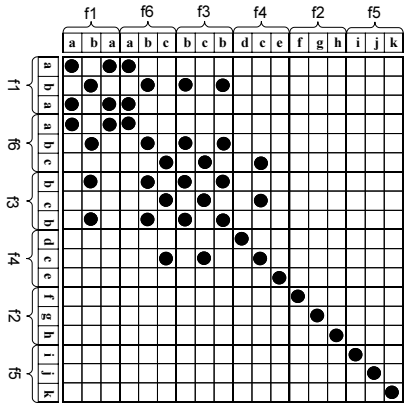


図 4　図 3 のクローン散布図のソート後
Fig. 4　Sorted model of clone scatter plot in Figure3

3.2.2　クローンペア管理部

クローンペア位置情報をもとにして，利用者の要求に応じて，クローン散布図を表示する．クローン散布図上での GUI による操作として，拡大・縮小，および，任意のクローンペア集合を「選択状態」にすること，がある (選択状態は，後述する他のサブシステムと連携した操作で使われる)．図 2 に選択状態のクローンペア集合を含んだクローン散布図の例を示す．矩形で囲まれたクローンペアが選択状態になっている．

クローン散布図

Gemini が表示するクローン散布図の簡単なモデルを図 3 に示す．散布図の原点は左上隅にあり，水平軸，垂直軸は，それぞれソースコードの並びに対応している．両軸上で，原点から順に，それぞれのソースファイルに含まれるトークンが並んでいる．座標平面内に点がプロットされている部分は，その両軸の対応するトークンが一致することを意味する．したがって，散布図の主対角線は，両軸の同じ位置のトークンを比較することになり，すべての点がプロットされることになる．また，点の分布は主対角線に対して線対称となる．一定の長さ (CCFinder で設定される最小一致トークン数) 以上の対角線分が，検出されたクローンペアである．図 3 では，`f1` から `f6` までのファイルが，それぞれ 3 つのトークンを含んでいる．ファイルの並びはファイル名の辞書順となっている．検出されるクローンペアは `f1` と `f6` に含まれる “ab” というトークン列と，`f3` と `f6` に含まれる “bc” というトークン列である．

クローン散布図のソート

一般に，ソースファイルにはコードクローンを含まないものがあり，クローンペアのクローン散布図内で疎に分布する可能性がある．大規模なソフトウェアのクローン散布図を操作する場合，疎な分布であることは，すなわち，広大なクローン散布図を全体に渡って参照しながら分析を行うことを意味し，分析の手間が増えることになる．

これを解決するため，クローン散布図のファイル順序を調節することで，分布を密にする方法 (クローン散布図のソート) を提案する．ソートにより，クローン散布図内の点の分布が密になり，大規模ソフトウェアのコードクローン分析労力を軽減することができる．

ソートにより，類似したファイルほど近くに配置され，また，原点周りにクローンを集めることができる．

本ソートの手順は次のようになる．

ステップ 1: 対象ファイルのうち，RSA 計測値が最も高いファイル f を先頭のファイルとする．

ステップ 2: 順序が決定されていない対象ファイルのうち，f との間で RST が最も高いファイル f' を次のファイルとする．f' を f として，すべての対象ファイルの順位が決定するまで，ステップ 2 を繰り返す．

図 3 のファイルでは，`f1` が最も高い RSA を持つので，`f1` が先頭のファイルになり，以下，`f1` との RST が最も高い `f6`，`f6` との RST が最も高い `f3`，等々という順番になる．ファイルソート後にクローン散布図を求めたものは図 4 であり，原点近くに点が集まっていることがわかる．

3.2.3 メトリクス管理部

クローンペア位置情報から，6 種のメトリクスを算出する．利用者の要求に応じて，メトリクスグラフビューによってメトリクス計測値を表示する．クローンクラスに関するグラフ (RAD, LEN, POP, DFL を表示) とファイルに関するグラフ (RSA, RST を表示) がある．利用者は，GUI による操作を行い，グラフの一部を「選択状態」にすることができる．

3.2.4 ソースコード管理部

指定されたソースファイルの内容を表示する．選択状態の部分は強調表示される (図 2 参照)．

3.3 サブシステム間の連携

クローンペア管理部，ソースコード管理部，メトリクス管理部の間で「選択状態」を用いた連携が可能になっている．この連携により，利用者は例えば (1) クローン散布図でクローンペアを選択した後，そのソースコードを表示する，(2) メトリクスグラフで一部のクローンクラスを選択した後，そのソースコードを表示する，(3) メトリクスグラフで長い (LEN が大きい) クローンクラスだけを選択した後，クローン散布図ではそれらがどこにあるのか調べる，といった対話的操作を行うことができる．

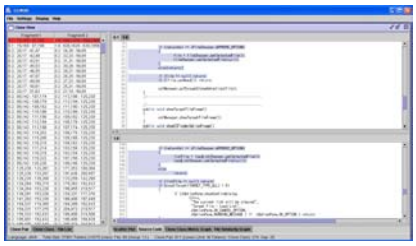


図 5 ソースコードビューの表示例
Fig. 5 GUI snapshot of source code view

3.4 実　　装

本システムは，Java で実装されており (約 13,000 行)，JDK1.3VM が実行可能な環境で動作する．表示画面の例を図 2，5 に示す．

4. 適 用 実 験

4.1 概　　要

大阪大学のあるプログラミング演習で作成されたプログラムに対し，Gemini の適用実験を行った．

本演習において，各学生は，Pascal 言語 (のサブセット) でかかれたプログラムを CASL 言語 (アセンブリ言語) に変換するコンパイラを C 言語で作成する．作成にあたり，学生にはコンパイラの仕様が掲載された指導書が与えられた．また，学生は，他の講義で，簡単なコンパイラのサンプルプログラムが記載されたテキストを用いてコンパイラの設計方法についての説明を受けている．本演習は，次の 3 つのステップ (課題) で構成されている．

ステップ 1(課題 1): 構文解析器 ($Parser$) の作成．

ステップ 2(課題 2): 意味解析器 ($Checker$) の作成．

ステップ 3(課題 3): コンパイラ (SPC) の作成．

また，$Checker$ と SPC を作成するにあたり，各課題の前課題のプログラムを拡張して作成することが課題として要求されている．つまり，$Checker$ は $Parser$ を，SPC は $Checker$ を拡張することで作成される．

本適用実験に際し，69 人の学生 ($S_1..S_{69}$) のプログラム ($Parser$, $Checker$, SPC) を収集した (計

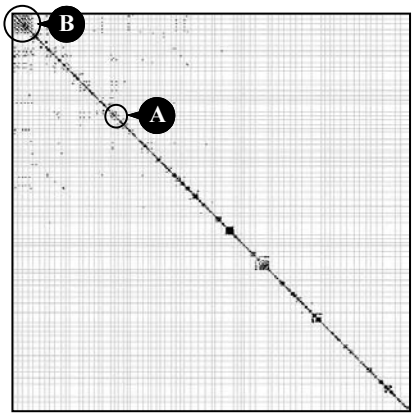


図 6　全学生の SPC を比較したクローン散布図 (ソート済み)

図 7　Sorted scatter plot of all students 'SPC

360KLOC).

4.2 分　　析

本適用実験において，次の項目について分析を行った．

(1) 全プログラムの間での類似性: プログラミング演習では，作成期限に間に合わせるなどのために，他の学生のプログラムをコピーおよび修正することで，プログラムを作成するといったことが起こり得る．そこで，RSA 値等を調べることで，学生が作ったプログラム間の類似性を確認する．

(2) メトリクスグラフの有効性: いくつかのモジュールに再構築できるような特徴的なコードクローンを抽出できるかどうかを調べることで，メトリクスグラフの有効性を確認する．

4.2.1　全プログラムの間での類似性

RSA 値の計測結果を表 1 に示す．表 1 における $Parser$，$Checker$，SPC の RSA 値の平均値は，それぞれ 0.089，0.032，0.019 となっており，課題が進むにつれ，各個人間での類似性は低くなっていくことが確認された．これは，後の課題 (課題 2 や課題 3) の方がより学生のオリジナリティを必要としているため，自然な結果と考えられる．

しかしながら，幾人かの学生は，課題 3 においても高い RSA 値を持っていた．そこで，クローン散布図を用い，学生間のコードクローンがどのように分布しているのか調べた．図 7 のクローン散布図は，全学生の SPC のソースコードを比較した結果であり，格子は各個人間の区切りを表している．本散布図上では，初め，コードクローンが散布図全体に広く分散していたため，Gemini に実装されたソート機能を用いて，ファイル順序の並べ替えを行った．ソートによってコードクローンの分布は，図 7 のように原点近くに密集した．図 7 中の A が付されたコードクローンが密集している部分では，S_1 と S_2(以降，S_1，S_2 を合わせて A グループと呼ぶ) の SPC 間の比較が行われ，B が付されたコードクローンが密集している部分では，S_3，S_4，S_5(以降，S_3，S_4，S_5 を合わせて B グループと呼ぶ) の SPC 間の比較が行われていた．

まず，S_1，S_2，S_3，S_4，S_5 の RSA 値を参照したところ $RSA(SPC)$ と $RSA(Checker)$ の値については，全学生の中で上位 5 位を占めていた．さらに B グループの学生の $RSA(Parser)$ 値は，上位 10 位内に入っていた．実際に，ソースコードビューを用いて対応したコードクローンを調べてみたところ，まず A グループにおける類似コードのほとんどは，テキストに記載されたサンプルコンパイラのコードであることが確認された．一方，B グループにおける類似コードは，変数名やコメントまで類似が見られた．つまり，B グ

表 1　RSA　値
Table 1　Values of RSA

	Parser	Checker	SPC	ave.
S_1	0.029	0.244	**0.159**	0.144
S_2	0.162	0.271	**0.148**	0.194
S_3	0.326	0.211	**0.137**	0.224
S_4	0.297	0.244	**0.160**	0.234
S_5	0.348	0.151	**0.142**	0.214
S_6	0.028	0.009	0.011	0.016
S_7	0.000	0.000	0.000	0.000
≈		...		≈
S_{69}	0.032	0.004	0.003	0.013
ave.	**0.089**	**0.032**	**0.019**	0.046
max.	0.407	0.271	0.160	0.234
min.	0.000	0.000	0.000	0.000

表 2　各プログラムにおける DFL 値の最大値
Table 2　The maximum values of DFL in each program

	Parser	Checker	SPC
S_1	79	131	131
S_2	145	199	199
S_3	75	97	199
S_4	75	75	391
S_5	75	119	233
S_6	**3538**	163	189
S_7	100	211	**3439**
≈		...	≈
S_{69}	223	211	258
max.	3538	603	3439
min.	0	47	51
ave.	196	183	311

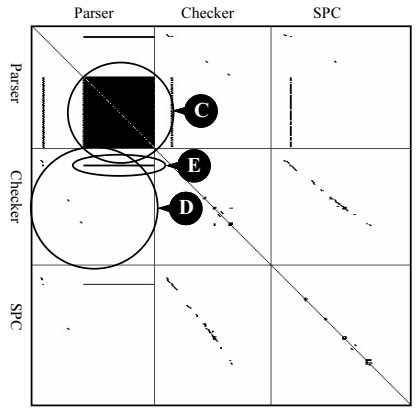


図 8　S_6 のクローン散布図
Fig. 8　Clone scatter plot for S_6' programs

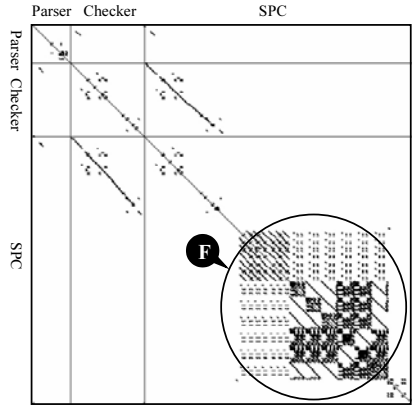


図 9　S_7 のクローン散布図
Fig. 9　Clone scatter plot for S_7' programs

ループ内において互いに何らかの参照が行われた可能性が高いと考えられる．

4.2.2　メトリクスグラフの有効性

ここでは，各学生の各プログラム内での比較を行い，特に DFL 値の有効性確認を行った．

ここで DFL を算出するに当たって，各サブルーチン呼び出しは 5 トークン (“サブルーチン名”，“(”，“引数”，“)”，“;”) とし，共通ロジックを実装するサブルーチンの大きさはコードクローンの各コード片と同じとする．したがって，

$$DFL(C) = LEN(C) \times POP(C) - (POP(C) \times 5 + LEN(C)).$$

表 2 に各プログラムにおける DFL 値の最大値を示す．メトリクスグラフより，S_6 の $Parser$ (DFL=3528) および S_7 の SPC (DFL=3439) は，DFL 値に関して特に突出した値を持っていることが確認された．

表 2 から，S_6 の $DFL(Parser)$，$DFL(Checker)$，$DFL(SPC)$ の値は，それぞれ 3538，163，189 となっていることが分かるが，$DFL(Parser)$ は非常に高い値であるのに対し，$DFL(Checker)$，$DFL(SPC)$ の値は，ほぼ平均値に近くなっている．これは，$Parser$ から $Checker$ へ拡張が行われる際に，コードクローンを除去する再構築が行われたことを意味する．

一方，S_6 のプログラムに関するクローン散布図を図 8 に示す．本散布図では，両座標軸上に $Parser$，$Checker$，SPC が課題の順に並んでいる．$Parser$ は，DFL 値が突出したクローンクラスを含んでいた (図中 C の部分)．$Parser$ と $Checker$ の比較 (図中 D の部分) においては，このような密集したクローンは存在せず，代わりに，一本の線分が現れている (図中 E の部分)．これは，S_6 が $Checker$ を作成する際に，$Parser$ に多く存在したコードクローンを一つのサブルーチンにマージしたことを意味している．実際，ソースコードを参照すると，そのような再構築が行われたことが確認できた．

次に，図 9 に S_7 のクローン散布図を示す．C と同様の，コードクローンが密集した部分が存在した (F)．ソースコードを参照することで，これらのコードクローンが，S_6 の場合と同様に再構築可能かを確認した．これらのコード片で異なっている部分は，ある 2 箇所の定数名のみであり，それらをパラメータ化することで，容易に一つのサブルーチンにまとめることが可能であった．

このように,「パラメータを追加して共通ロジックをサブルーチンにまとめる」場合は，パラメータの追加がモジュールの結合を増大させるため，保守容易性の観点からはトレードオフを考慮する必要がある．S_7 の場合には，追加されるパラメータは定数 2 つであり，モジュールの結合はそれほど増大しないと考えられる．

5.　まとめと今後の課題

本論文では，開発保守支援を目指したコードクローン分析環境 Gemini の構築を行った．Gemini を用いることで，クローン散布図やメトリクスグラフ等のインターフェースを介して，特徴的なコードクローンを識別することができ，それらに対応した実際のソースコードを参照することができる．また，本システムの有効性の確認のため，大学におけるプログラミング演

習で作成されたプログラムに適用した．クローン散布図においては，ソート機能の有効性等が確認された．また，メトリクスを用いた分析においては，特徴あるコードクローンが容易に識別できることを確認した．

今後の課題としては，大規模なソフトウェアや実際の開発現場へ Gemini を適用し，評価を行うことが挙げられる．

文　献

[1] B.S. Baker, "A program for identifying duplicated code", *Computing Science and Statistics*, vol.24, pp.49-57, 1992.

[2] B.S. Baker, "On finding duplication and near-duplication in large software systems", *Proc. the 2nd Working Conference on Reverse Engineering*, pp.86-95, Tronto, Canada, July 1995.

[3] M. Balazinska , E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis, "Measuring clone based reengineering opportunities", *Proc. 6th IEEE International Symposium on Software Metrics*, pp.292-303, Boca Raton, USA, Nov. 1999.

[4] M. Balazinska , E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis, "Partial redesign of Java software systems based on clone analysis", *Proc. 6th IEEE International Working Conference on Reverse Engineering*, pp.326-336, Atlanta, USA, Oct. 1999.

[5] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees", *Proc. IEEE International Conference on Software Maintenance-1998*, pp.368-377, Bethesda, USA, Nov. 1998.

[6] S. Ducasse, M. Rieger, and S. Demeyer. "A language independent approach for detecting duplicated code", *Proc. IEEE International Conference on Software Maintenance-1999*, pp.109-118, Oxford, UK, Sept. 1999.

[7] M. Fowler, *Refactoring: improving the design of existing code*, Addison-Wesley, 1999.

[8] D. Gusfield, *Algorithms on strings, trees, and sequences*, Cambridge University Press, 1997.

[9] *IEEE Std 1219: Standard for software maintenance*, 1997.

[10] 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, pp.47-54, 2001.

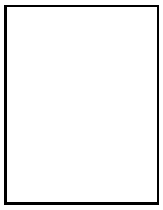
[11] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code", *IEEE Trans. Software Engineering*, vol.28, no.7, pp.654-670, 2002.

[12] R. Komondoor, and S. Horwitz, "Using slicing to identify duplication in source code", *Proc. 8th International Symposium on Static Analysis*, pp.40-56, Paris, France, July 2001.

[13] J. Krinke, "Identifying similar code with program dependence graphs", *Proc. 8th Working Conference on Reverse Engineering*, pp.562-584, Stuttgart, Germany, Oct. 2001.

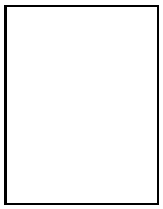
[14] J. Mayland, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics", *Proc. IEEE International Conference on Software Maintenance-1996*, pp. 244-253, Monterey, USA, Nov. 1996.

（平成年月日受付，月日再受付）



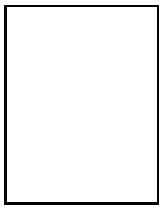
植田　泰士

平 13 阪大・基礎工・情報卒．平 15 同大大学院博士前期課程修了．現在，宇宙開発事業団所属．在学中，コードクローン分析の研究に従事．



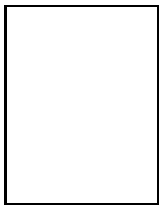
神谷　年洋

平 8 阪大・基礎工・情報中退．平 13 同大大学院博士課程了．現在，科学技術振興事業団 PRESTO 研究員．博士 (工学)．オブジェクト指向関連技術，ソフトウェア保守 (メトリクス，コードクローン)，認知科学に関する研究に従事．情報処理学会，IEEE 各会員．



楠本　真二　（正員）

昭 63 阪大・基礎工・情報卒．平 3 同大学院博士課程中退．同年同大・基礎工・情報・助手．平 8 同大講師．平 11 同大助教授．平 14 阪大・情報・コンピュータサイエンス・助教授．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．情報処理学会，IEEE 各会員．



井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜昭 61 ハワイ大マノア校・情報工学科・助教授．平 1 阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．工学博士．平 14 阪大・情報・コンピュータサイエンス・教授．ソフトウェア工学の研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE，ACM 各会員．