特別研究報告

題目

識別子名のタグクラウドを用いたコードクローン理解支援手法

指導教員
井上 克郎 教授

報告者
佐野 真夢

平成 26 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 25 年度 特別研究報告

識別子名のタグクラウドを用いたコードクローン理解支援手法

佐野 真夢

内容梗概

ソフトウェアの保守を困難にしている大きな要因の 1 つとしてコードクローンが指摘されている. コードクローンとは, ソースコード中に存在する互いに一致または類似した部分を持つコード片のことを示しており, 既存コードのコピーアンドペーストによる再利用や, 特定のコードを意図的に繰り返し書くことなどによって生じる.

コードクローンに対する保守作業の 1 つとして集約が挙げられる. 集約とは, 互いにコードクローンとなっているコード片を 1 つのメソッドやクラスなどにまとめることである. 集約を行うことで, 保守の対象となるコードクローンを除去することが可能となる.

ソフトウェアの保守を容易にするためには, コードクローンの集約を行うことが望ましいと考えられる. しかし, ツールによって自動検出されたコードクローンの中には, 悪影響を及ぼさないといった理由から, 開発者が意図的に除去していないコードクローンが数多く存在する. そのため, 全てのコードクローンが集約の対象になるとは限らない. コードクローンが集約の対象となるかどうか判断するには, 実際にソースコードを読む必要がある. しかし, 全てのコードクローンとなっているコード片を読むことは非効率的である.

上述の問題の解決策として, 着目するコードクローンを厳選することが挙げられる. 例えば, コードクローン可視化手法であるクローン散布図 (ソースコード上に存在するコードクローンの位置をプロットした散布図) を用いた場合, ソフトウェア中のどのディレクトリやコンポーネントにコードクローンが存在しているかを直観的に理解することが可能である. 従って, コードクローンが多く含まれるディレクトリやコンポーネントのみに注目し, 着目するコードクローンを厳選することが可能である. しかし, クローン散布図には, ソースコードの処理に関する情報は反映されていない. そのため, コードクローンとなっているコード片がソフトウェア中においてどのような役割を果たしているかを理解することは困難である. 短時間でコードクローンとなっているコード片の内容を直観的に理解することができれば, 重要なコードクローンを把握することができると考えられる. 結果として, 着目するコードクローンの数をより少なくすることが可能であると考えられる.

そこで, 本研究では, 識別子名を利用してコードクローンの処理内容を可視化する手法を提案した. この提案手法は, ソフトウェア開発者はプログラムを読む際, その中で使用されて

いる識別子名の意味からプログラム要素の役割を推測しているという研究報告に基づいている．提案手法では, ソースコード中で使用されている識別子名を抽出し, それをキーワードとしたタグクラウド (文書中の重要なキーワードを可視化する手法) を生成することで, コードクローンの処理内容の直観的な理解を図っている．また, 本手法を実装したコードクローン分析ツールの開発を行った．

さらに, 本研究では, 本手法により抽出された識別子名の評価を行い, 有用な識別子名を抽出できているかどうかを確かめることができた．また, 開発したツールと既存のコードクローン分析ツールの比較実験を行い, 識別子名のタグクラウドに有用性があるかどうかを確かめた．

主な用語

コードクローン
ソフトウェア保守
タグクラウド

目 次

1　まえがき

ソフトウェアの保守を困難にしている大きな要因の 1 つとしてコードクローンが指摘されている. コードクローンとは, ソースコード中に存在する互いに一致または類似した部分を持つコード片のことを示している [6]. コードクローンの主な発生原因としては, 既存コードのコピーアンドペーストによる再利用や, 特定のコードを意図的に繰り返し書くことなどが挙げられる.

コードクローンに対する保守作業として, 同時修正と集約が挙げられる. 同時修正とは, 互いにコードクローンとなっているコード片の集合 (以下, クローンセットと呼ぶ) に対して一貫した修正を行うことである. 例えば, コードクローンとなっているコード片に欠陥が存在している場合, 同じクローンセットに属する他のコードクローンにも同様の欠陥が含まれている可能性がある. そのため, そのクローンセットの全てのコードクローンに対して, 一貫した修正を加えることを検討する必要があると考えられる. また, 集約とは, 同じクローンセットに含まれるコードクローンを 1 つのメソッドやクラスなどにまとめることである. 集約を行うことで, 保守の対象となるコードクローンを除去することが可能となる.

ソフトウェアの保守を容易にするためには, コードクローンの集約を行うことが望ましいと考えられる. しかし, ツールによって自動的に検出されたコードクローンの中には, 悪影響を及ぼさないといった理由から意図的に除去されていないコードクローンが数多く存在する [9]. そのため, 全てのコードクローンが集約の対象になるとは限らない. コードクローンが集約の対象となるかどうかを判断するには, 実際にソースコードを読む必要がある. しかし, 全てのコードクローンとなっているコード片を読むことは非効率的である. このような問題を解決し, コードクローン分析を効率的に行うために, コードクローンの理解支援を目的としたコードクローンの可視化手法が提案されている.

既存のコードクローン可視化手法として, クローン散布図がある. クローン散布図とは, 分析対象となるソースコード上に存在するコードクローンの位置をプロットした散布図のことであり, コードクローン分析ツール Gemini[17] などで利用されている. クローン散布図を用いることで, ソフトウェア中のどのディレクトリやコンポーネントにコードクローンが存在するのかを直観的に理解することが可能である. また, 含まれているコードクローンの数が少ないディレクトリやコンポーネントを除外し, コードクローンが多く含まれているディレクトリやコンポーネントのみに着目することで, 分析するコードクローンの数を減らすことも可能である. しかし, クローン散布図には, ソースコードの処理に関する情報は反映されていない. そのため, コードクローンとなっているコード片が, ソフトウェア中においてどのような役割を果たしているかを理解することは困難である.

そこで, 本研究では, 識別子名 (変数名や関数名など) を利用した, コードクローンの処理

内容の可視化手法を提案した．この提案手法は, ソフトウェア開発者はプログラムを読む際, その中で使用されている識別子名の意味からプログラム要素の役割を推測している [14, 18] という研究報告に基づいている．提案手法では, ソースコード中で使用されている識別子名を抽出し, 文書中の重要なキーワードを可視化する手法であるタグクラウドを用いることで, コードクローンの処理内容の直観的な理解を図っている．また, 本手法を実装したコードクローン分析ツールの開発を行った．

さらに, 本研究では, 本手法を用いて抽出されたコードクローンに含まれる識別子名を, コードクローン分析の専門家が記述したコードクローンの説明文に基づく, 有用な識別子名のリストと照らし合わせることで, 抽出された識別子名が有用であることを確認した．また, 本ツールと既存のコードクローン分析ツールである Gemini のそれぞれを用いて, コードクローンの説明文を記述するという被験者実験を行った．それぞれのツールを用いて記述した説明文の正確さや, 記述に要した時間を比較することで, 識別子名のタグクラウドに有用性があるかどうかを確認した．

以降, 2 節では本研究の背景について説明し, 3 節では識別子名を利用したコードクローンの処理内容の可視化手法について説明する．また, 4 節では本研究で開発したコードクローン分析ツールについて説明し, 5 節では本手法, および, 本ツールを用いた評価実験について述べる．そして, 6 節で本研究のまとめと今後の課題について述べる．

2　背景

本節では, 本研究の背景として, コードクローンとその検出ツール, コードクローンの可視化手法, 及び, タグクラウドについて説明する.

2.1　コードクローン

コードクローン (Code clone) とは, ソースコード中に存在する互いに一致または類似した部分を持つコード片のことを示している. 一般的に, コードクローンの存在はソフトウェアの保守を困難にすると言われている [6]. 例えば, コードクローンとなっているコード片の 1 つに欠陥が存在している場合, そのコード片と類似または一致する他のコード片についても同様の欠陥が存在する可能性がある. そのため, ソフトウェアの保守のためにコードクローンの存在を知ることは重要である. しかし, ソフトウェアの規模が非常に大きい場合, 開発者がソフトウェアに含まれる全てのコードクローンを認識することは非現実的である. 従って, 一般的にソフトウェア開発では, ツールを用いた自動的なコードクローン検出が行われる [8].

一般的に, 互いにコードクローンとなるコード片の対をクローンペア (Clone pair) と呼び, クローンペアにおいて推移関係が成り立つコードクローンの集合をクローンセット (Clone set) と呼ぶ.

コードクローンに対する保守作業として, 同時修正と集約が挙げられる. 同時修正とは, クローンセットに含まれる全てのコードクローンを一貫して修正することである. 例えば, あるコードクローンに欠陥が含まれていた場合, 同じクローンセットに属する他のコードクローンについても, 同様の欠陥を修正しなければならない可能性がある. そのような場合には, コードクローンの一貫した修正が必要になると考えられる. また, 集約とは, 同じクローンセットに含まれるコードクローンを 1 つのメソッドやクラスなどにまとめることである. 集約を行うことで, 保守の対象となるコードクローンを除去することが可能となる.

2.1.1　コードクローンの発生原因

コードクローンがソースコード中に発生する原因として, 以下のものが挙げられる [2, 8].

既存コードのコピーアンドペーストによる再利用

一からソースコードを書くよりも, 同様の処理を行う既存コードを流用し, 必要に応じて部分的な変更を加えた方が信頼性が高い. そのため, 実際には, コピーアンドペーストによる既存コードの再利用が数多く存在する.

定型処理

定義上簡単で, 頻繁に用いられる処理はコードクローンになる傾向がある. 例として, キューの挿入処理や, データ構造へのアクセス処理などが挙げられる.

プログラミング言語における適切な機能の欠如

プログラミング言語が抽象データ型やローカル変数に対応していない状況において, 同じようなアルゴリズムを持つ処理を繰り返し書かなければならない場合がある.

パフォーマンスの改善

時間制約のあるシステムにおいて, インライン展開などの機能が提供されていない場合, 特定のコードを意図的に繰り返し記述することでパフォーマンスの改善を図ることがある.

コード生成ツールによる自動生成

コード生成ツールは, あらかじめ定められたコードをベースにして自動的にコードを生成する. そのため, 目的の処理が類似している場合, 識別子などを除いて類似したコード片が生成される.

複数のプラットフォームへの対応

複数の OS や CPU に対応しているソフトウェアでは, 各プラットフォーム用のコード中に重複した処理が存在する傾向がある.

偶然の一致

偶然, 開発者が同一のコードを書いてしまう場合がある.

2.1.2 コードクローンの定義

コードクローンには様々な検出方法が存在するが, そのどれもが異なったコードクローンの定義を持つ. そのため, 厳密で普遍的なコードクローンの定義は存在しない. Bellon はコードクローン間の相違の度合いに基づいて, コードクローンを以下のような 3 つの定義に分類している [3, 7].

タイプ 1

空白やタブの有無, 括弧の位置といったコーディングスタイルを除いて完全に一致するコードクローン.

タイプ 2

変数名や関数名といったユーザ定義名, 及び変数の型などの一部の予約語のみが異なるコードクローン.

タイプ 3

タイプ 2 における変更に加えて, 文の挿入や削除, 変更が行われているコードクローン.

2.2 コードクローン検出ツール

コードクローンの検出手法としては, ソースコードの字句解析に基づく手法 [1, 11, 15] や, 特徴メトリクスに基づく手法 [10, 12] などが存在する. ソースコードの字句解析に基づく手法では, ソースコード中にある同一の文字列を検索することでコードクローンの検出を行う. また, 特徴メトリクスに基づく手法では, クラスや関数, ファイルといったプログラム中のある種の単位ごとに特徴メトリクスを定義・算出し, それらのメトリクス値が類似したものをコードクローンとして抽出する.

本研究では, 字句解析ベースのコードクローン検出ツール CCFinder[8] を利用してコードクローンの検出を行う. CCFinder は高いスケーラビリティを有しており, 大規模なソフトウェアに対しても実用的な時間で解析を行うことができる. また, 表現上の差異があるコードクローンを検出することができるという特徴も備えている. 実際に, CCFinder は様々な大規模ソフトウェアに適用され, その有用性が確認されている [13].

CCFinder のコードクローン検出手順は, 字句解析, 変換処理, 検出処理, 出力整形処理からなる. 字句解析では, ソースコードをトークン列に変換する. 変換処理では, 変数名や関数名などを同一のトークンに置換することで, 表現上の違いを吸収している. 検出処理では閾値以上の長さの共通トークン列の探索を行い, クローンペアとして検出する. その後, 出力整形処理で検出したクローンペアに関する, ソースコード上での位置情報のリストを出力する. 従って, CCFinder は 2.1.2 節で述べたタイプ 1, タイプ 2 のコードクローンを検出することが可能である.

2.3 コードクローンの可視化手法

既存のコードクローン分析ツールで利用されている可視化手法として, クローン散布図 (Scatter plot) が挙げられる. クローン散布図とは, 分析の対象となるソースコード上に存在するコードクローンの位置をプロットした散布図のことを示す. 図 1 は, コードクローン分析ツール Gemini[17] における実際のクローン散布図の例である. 図 1 のクローン散布図では, 左上端を原点に, 縦軸と横軸に, 解析するソースコードをトークン列として並べている. 縦軸と横軸のトークンが一致する場合には点が描画されている. 従って, コードクローンは線分 (斜めに連続する点) としてクローン散布図上に現れる. 線分の長さが長いほど, より大きなコードクローンであることを意味する. 縦軸と横軸のトークン列は, ともにソースコード上での出現順に並んでいるため, 散布図の左上端から右下端への対角線上に必ず点が描画

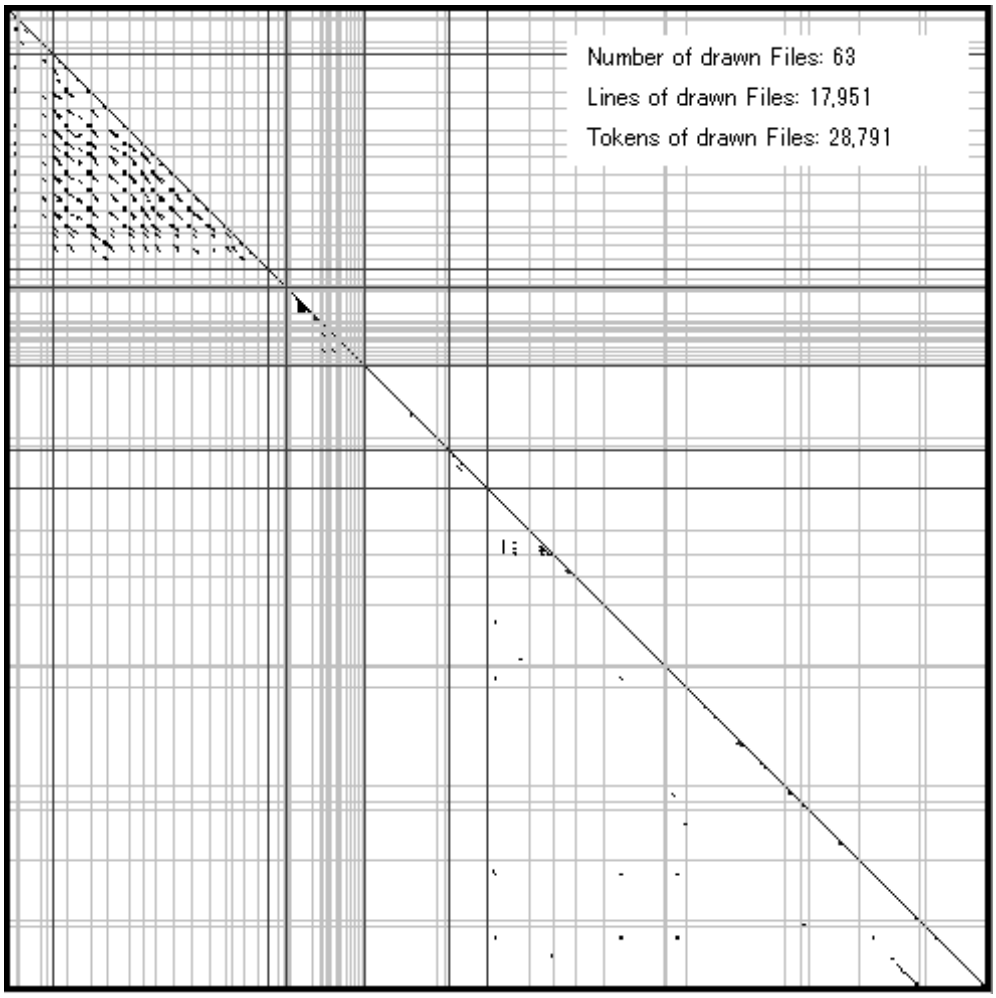


図 1: Gemini におけるクローン散布図の例

される．また, 一致するトークンの分布は対角線を対称軸として線対称となっている．そのため, 対角線から上の部分は省略されている．クローン散布図を見ることで, ソースコード (あるいは, ディレクトリやコンポーネント) のどの部分に, どの程度のコードクローンが含まれているのかを直観的に理解することが可能である．

コードクローンの集約を行う場合, 開発者は上述の Gemini などのコードクローン分析ツールを用いてソフトウェア中のコードクローンを分析し, 集約が必要なコードクローンを調査すると考えられる．しかし, ツールによって自動的に検出されたコードクローンの中には, 悪影響を及ぼさないといった理由から意図的に除去されていないコードクローンが数多く存在する [9]．また, 2.1.1 節で説明したように, パフォーマンスの改善などのために, 開発者が故意にコードクローンを作り込んでいる可能性もある．従って, 全てのコードクローンが集約の対象になるとは限らない．コードクローンが集約の対象となるかを判断するためには, コードクローンとなっているコード片がソフトウェア中でどのような役割を担っているかを, ある程度把握する必要がある．しかし, クローン散布図にはソースコードの処理に関する情報は反映されていない．そのため, コードクローンの処理内容を理解するためには, 実際にソースコードを読む必要がある．しかし, 全てのコードクローンとなっているコード片を読むことは非効率的であると考えられる．

本研究では, ソースコード中に出現する識別子名を利用することで, コードクローンの処理内容を直観的に理解することを試みている．この理由は, ソフトウェア開発者は, プログラムを読む際, その中で使用されている変数名や関数名といった識別子名の意味からプログラム要素の役割を推測しているためである [14, 18]．

2.4 タグクラウド

タグクラウド (Tag cloud) とは, ブログの記事などの文書からキーワードを抽出し, そのキーワードを画面上に表示する可視化手法を示す．様々なキーワード (tag) が雲 (cloud) のように見えることからタグクラウドと呼ばれているが, タグクラウドの形状や, 表示されるキーワードの色や大きさなどの計算方法に関する厳密で普遍的な定義は存在しておらず, 多種多様なタグクラウドが存在している．一般的には, 表示されるキーワードの大きさは, 対象の文書における重要度や関心度を表す指標に基づいて定められており, 相対的に大きく表示されているキーワードは, より重要度や関心度が高いキーワードであることを意味している．例えば, 図 2 は, ユーザが与えたテキストからタグクラウドを生成することができる Web サービス Wordle[1] を用いて生成した自然文に対するタグクラウドの例である．図 2 において, 他のキーワードよりも相対的に大きく表示されている “code” や “clone”, “source” といった

[1] http://www.wordle.net/ .

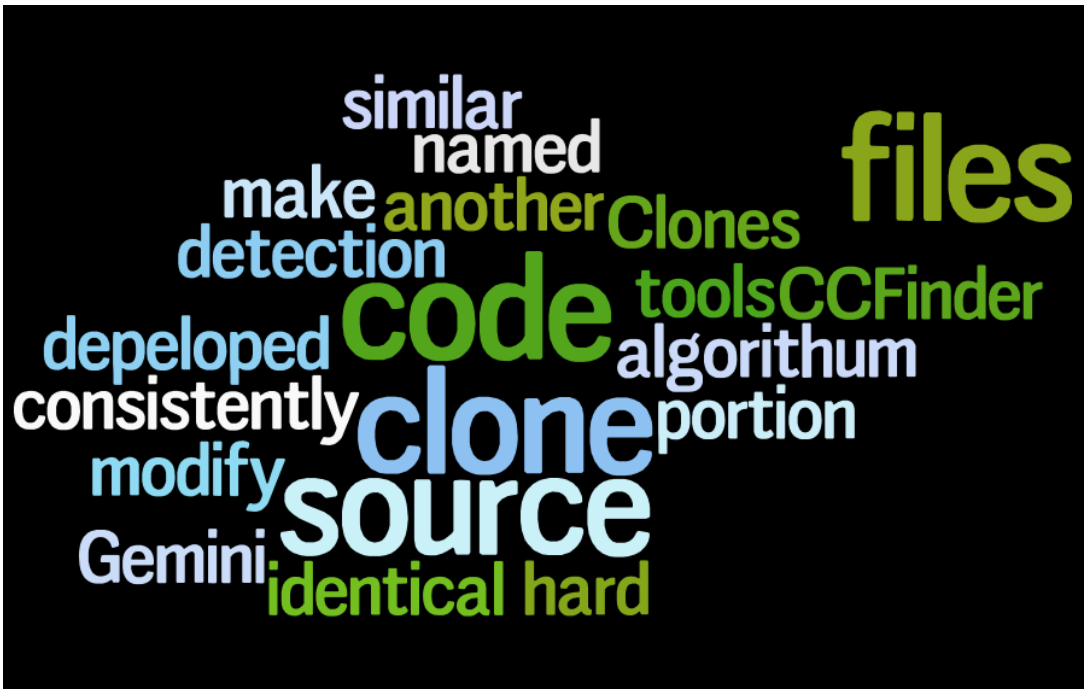


図 2: 自然文におけるタグクラウドの例

キーワードは, 元の自然文で頻出する単語であり, その文章の特徴を表すキーワードである可能性が高いことを示している.

本研究では, タグクラウドを用いてソースコード中の識別子名を可視化することで, コードクローンの処理内容の理解支援を図っている. タグクラウドを用いる利点として, 以下の 2 つが挙げられる.

小さな領域に多くのキーワードを表示できる

大規模プロジェクトへの適用を考えた場合, 多くの識別子名がキーワードとして抽出されることが予測される. そのため, 単純にリストなどを用いてキーワードを並べた場合, ディスプレイのサイズによってはキーワードが画面内に収まらない可能性が考えられる. しかし, キーワードの相対的な重要度を直観的に理解するためには, 全てのキーワードが画面内に収まっていることが望ましい. タグクラウドを利用することで, より多くのキーワードを 1 つの画面内に収める事が可能となる.

重要度を直観的に理解しやすい

タグクラウド上に表示されるキーワードの大きさは, その重要度が高いほど相対的に大きく表示される. そのため, どのキーワードがより重要であるのかを直観的に理解することが可能となる.

なお, 文書に付けられたタグをキーワードとする場合にのみタグクラウドと呼び, 文書中で使用されている単語をキーワードに利用する場合はワードクラウドと呼ぶ場合もある. 本論文では, 特に区別する必要が無い限り, どちらも統一してタグクラウドと表記している.

3　提案手法

本節では, 本研究において提案する識別子名を用いたコードクローンの可視化手法について説明する. 本手法では, ソースコード中に存在する識別子名を抽出し, それをキーワードとしたタグクラウドを生成することで, コードクローンの可視化を行う. 生成されたタグクラウドのキーワードの大きさは, TF-IDF[16] と呼ばれる重み付け手法に基づいて定めており, ソースコード中の特徴的な識別子名に対応するキーワードが相対的に大きく表示される. また, コードクローンとなっているコード片に含まれているキーワードには着色を行い, コードクローンに直接関係していることが直観的に理解できるようにしている.

3.1　生成するタグクラウドの仕様

本節では, 本手法を用いて生成されるタグクラウドについて説明する. 本手法により生成されるタグクラウドは, 以下のような仕様に基づいている.

表示するキーワード

タグクラウドのキーワードは, 分析対象となる各ソースコード中に含まれる識別子名である. ただし, 識別子名のスペルにおいて, その違いがアルファベットの大文字・小文字のみである場合 (file と File など) は, 同じ識別子名として扱う. これは, 大文字・小文字の違いは識別子名の意味に全く影響を及ぼさないと考えられるからである. また, 識別子名の中には, 複数の英単語で構成されている場合 (addElement, getValue など) も多数存在している. これらの単語をそれぞれ分割してタグクラウドのキーワードに用いる方法も考えられるが, 本手法では, 単語の分割は行わず, そのまま各識別子名をキーワードとしている. これは, 1 つの識別子名を英単語ごとに分割した場合, 元の識別子名と異なる意味を導く可能性があると考えたためである. 例えば, addElement と getValue という識別子名があった場合, 英単語ごとに分割してタグクラウドを生成すると, “add”, “element”, “get”, “value” の 4 つのキーワードが表示されることになる. しかし, この時点で元の識別子名は判断できないため, “addValue” といった存在しない識別子名を推測する可能性がある.

キーワードの色

タグクラウドのキーワードの内, コードクローンに含まれている識別子名に対応するキーワードは, コードクローンに含まれていないものと区別できるように着色を行う. これにより, コードクローンと直接関係のある識別子名を直観的に判断することが可能である. また, コードクローンに含まれていない識別子名も, 分析対象全体の処理内容を示しているため, コードクローンが分析対象全体の中でどのような役割を果たし

ているのかを理解するために有益であると考えられる.

キーワードの大きさ

タグクラウドのキーワードの大きさを定める指標には, TF-IDF と呼ばれる重み付け手法を用いている. これについては, 3.1.1 節で詳しく述べる.

不要なキーワードの除去

分析対象の全ての識別子名をタグクラウドとして表示するのは無駄であり, また, あまりに多くのキーワードを表示することはタグクラウドの可視性を失うと考えられる. そこで, 本手法では, 本手法の目的であるコードクローンの処理内容の理解に有益である可能性の低い識別子名の除去を行っている. これについては, 3.1.2 節で詳しく述べる.

3.1.1 キーワードの大きさ

3.1 節で述べたように, 本手法によって生成されるタグクラウドのキーワードの大きさは, TF-IDF[16] と呼ばれる重み付け手法に基づいて決定している. TF-IDF とは, ある文書中に出現する単語に対する重み付けを示しており, 単語の出現頻度 (TF: Term Frequency) と逆文書頻度 (IDF: Inverse Document Frequency) の 2 つの指標を掛け合わせることで, その単語の文書に対する重要度の指標としている.

例として, ある文書の集合 D に属する 1 つの文書 d において, d 中に存在する単語 w の TF-IDF を求める場合を考える. 単語 w の出現頻度 (TF) とは, d 中に存在する全ての単語に対する w の出現頻度を示しており, 以下の式で計算される. 以下の式において, N は d における重複を許した単語の総数, n_w は d における w の出現回数を意味する.

$$TF = \frac{n_w}{N}$$

また, 単語 w の逆文書頻度とは, D に属する文書全体に対する, w を含んでいる文書の割合の逆数を指しており, 以下の式で計算される. 以下の式において, $|D|$ は D に属する文書の総数, $|d_w|$ は D に属する文書の内, w を含んでいる文書の総数を意味する.

$$IDF = \log \frac{|D|}{|d_w|}$$

一般的に, ある文書において頻出する単語は, その文書を特徴付ける重要な単語である可能性が高いと考えられる. 従って, TF の数値が高い単語ほど重要な単語であると言える. 一方で, 冠詞や代名詞といったどの文書でも頻出するような単語は, たとえ出現頻度が高くても重要であるとは言えない. そこで, IDF を利用することで, このような単語の除去を図ることができる. IDF は, 単語を含んでいる文書が多いほど数値が低くなるため, 単語の一般

性を測る指標として利用することができる. すなわち, IDF の数値が高いほど単語は非一般的であり, 重要な単語である可能性が高いと考えられる. これら 2 つの指標を掛け合わせることで, 単語の重要度の指標である TF-IDF を計算することが可能である.

ここで, 本手法で用いる, タグクラウドのキーワードの大きさを定めるための TF-IDF について説明する. 本手法においては, TF-IDF の文書をソースコード, 単語を識別子名と考える. 識別子名 i の TF とは, 分析対象のソースコード群 s 中に出現する全ての識別子名に対する識別子名 i の出現頻度を指す. また, 識別子名 i の IDF とは, あるソースファイルの集合 S に対する, i を含んでいるソースコードが記述されたソースファイルの割合の逆数を指す. S は s を内包した集合であり, 例として, ソフトウェアの全てのソースファイルなどが挙げられる. また, s の例としては, ソフトウェアのあるコンポーネントやディレクトリに属するソースコードなどが挙げられる. なお, 本研究において開発するツールでは, s を 2 つのディレクトリ (同じディレクトリでも可) に属するソースコードとしており, その 2 つのディレクトリ間におけるコードクローンを検出対象としている.

本手法における, 識別子名 i に対応するキーワードの重み $TF\text{-}IDF_i$ を求める計算式を以下に示す. 以下の式で使用する記号は表 1 にまとめている.

$$TF\text{-}IDF_i = TF_i \times IDF_i$$

$$TF_i = \frac{n_i}{N}$$

$$IDF_i = \log \frac{|D|}{|d_i|}$$

3.1.2 不要なキーワードの除去

3.1 節で述べたように, 単純に分析対象の全ての識別子名をタグクラウドのキーワードとして表示するのは非効率的であると考えられる. そこで, 本手法では, コードクローンの処理内容の理解に有益である可能性の低い識別子名の除去を目的とした, いくつかの指標によ

表 1: 提案手法における TF-IDF の計算式の記号の意味

記号	意味
N	分析対象のソースコード群における, 重複を許した識別子名の総数
n_i	分析対象のソースコード群における識別子名 i の出現回数
$\|D\|$	分析対象のソースコード群を内包するソースファイルの集合
$\|d_i\|$	$\|D\|$ に属するソースファイルの内, 識別子名 i を含むファイルの数

るキーワードのフィルタリングを行っている. 以下に, 本手法で行われるキーワードのフィルタリングとその説明を示す.

識別子名の長さによるフィルタリング

“o”, “it”といった 2 文字以下の識別子名は, ほとんどの場合その意味を推測できないと考えられる. そこで, 本手法ではある一定の長さ未満の識別子名は, タグクラウドのキーワード候補から除外している.

識別子名の **IDF** によるフィルタリング

3.1.1 節で説明したように, 識別子名の IDF はその識別子名の非一般性を表している. すなわち, IDF の低い識別子名は, より一般的な識別子名であり, 多数のソースコード中で見られるということを意味する. 従って, IDF が非常に低い識別子名は, 汎用的によく使われる識別子名であり, コードクローンの処理内容を特徴付けるには不向きであると考えられる. そこで, 本手法では閾値未満の IDF を持つ識別子名は, タグクラウドのキーワード候補から除外している.

また, 大規模ソフトウェアを対象にする場合, 多数の識別子名が抽出されることが予想される. その場合, たとえフィルタリングを行ったとしても, 全ての識別子名をタグクラウドに表示することは非現実的であると考えられる. そこで, 本手法では, 分析対象のソースコード群中における識別子名の出現回数により, 識別子名に対して順位を付け, 出現回数の多いものを優先的にタグクラウドとして表示するようにしている.

3.2 可視化手順

図 3 は, 本手法の可視化手順に関する概要図である. 本手法において, 識別子名のタグクラウドが生成されるまでの可視化手順を以下に示す. 本手法において, 入力は分析の対象となるソースコードの集合である. 入力の具体例としては, あるソフトウェア全体のソースコードの集合などが挙げられる.

STEP1:コードクローンの抽出

分析の対象となるソースコードの集合に CCFinder を適用し, そこに含まれるコードクローンの集合を抽出する. さらに, 分析対象のソースコード集合に属する, あるコンポーネントやディレクトリ間のコードクローンに関する識別子名のタグクラウドを生成する場合は, コンポーネントやディレクトリ間におけるコードクローンのみで構成されるコードクローン集合を生成し, 以降のステップではこの集合を利用する.

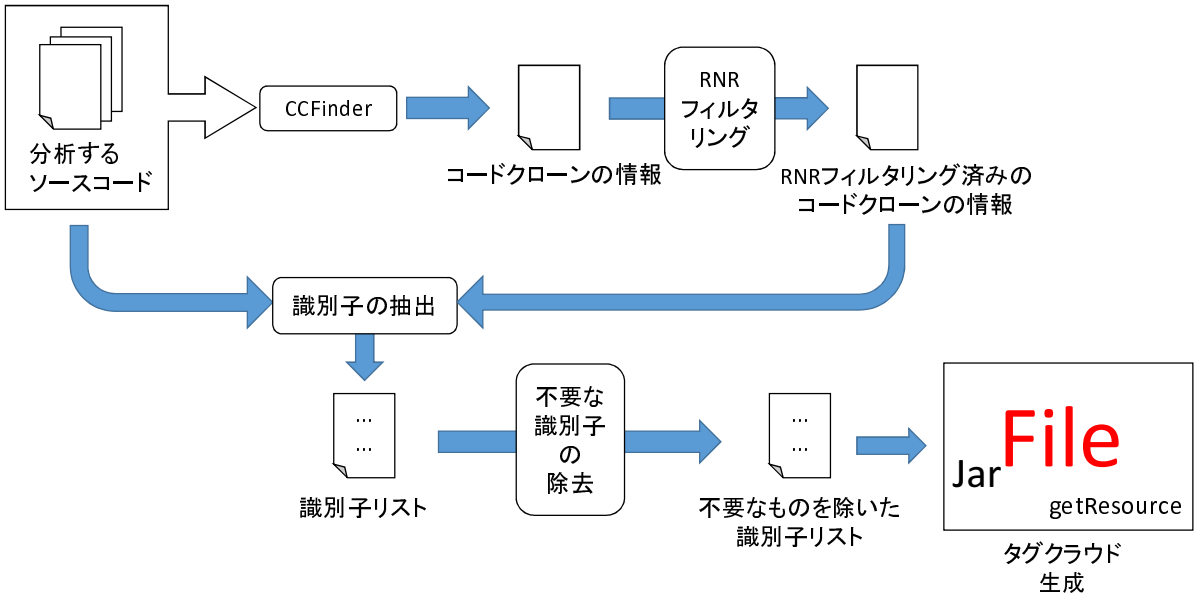


図 3: 本手法の可視化手順の概要図

STEP2:RNR によるコードクローンのフィルタリング

RNR とは, コードクローンに関するメトリクスの 1 種であり, あるクローンセットに属するコードクローンとなっているコード片の非繰り返し度数を表している. あるクローンセットの RNR の数値が低い場合, そのクローンセットに属するコード片は, print 文の連続などの同じような処理の繰り返しが多く含まれているコード片であることを示している [5]. 一般に, RNR の数値が低いコードクローンは前述のような重要度の低いコードクローンである場合が多いため, 本ステップでは, 閾値以下の RNR を持つクローンセットに属するコードクローンの除去を行う. ここでは, STEP1 で生成したコードクローンの集合から閾値以下の RNR を持つコードクローンを除去した集合を生成する.

STEP3:識別子名の抽出

分析対象となるソースコード集合に含まれる識別子名の抽出を行う. このとき, 各識別子名が STEP2 で生成したフィルタリング済みのコードクローン集合に属するコード片に含まれているか否かを調べ, 保持しておく. 3.1 節で述べたように, 本手法では識別子名をキーワードとしたタグクラウドを生成する際, そのキーワードがコードクローンに含まれているかどうかを把握するための着色を行う. 従って, 各識別子名が STEP2 で生成したコードクローン集合に属するコード片に含まれているか否かを保持しておく必要がある.

また, 3.1.1 節で説明したタグクラウドのキーワードの大きさを決定する重みを計算するた

めに必要な, 各識別子名の分析対象となる各ソースコードに対する出現回数や, その識別子名を含んでいるソースコードが記述されたソースファイルの総数を求めておく. なお, 本研究においては, 大文字・小文字のみの違いしか持たない識別子名は同じ識別子名として扱っている.

STEP4:不要なキーワードの除去

3.1.2 節で説明した不要なキーワードの除去を行う. すなわち, STEP3 で抽出した識別子名の集合から, 名前が一定の長さに満たない識別子名と, 閾値未満の IDF を持つ識別子名を取り除いた識別子名の集合を生成する.

STEP5:タグクラウドの生成

STEP4 で生成した不要なキーワードを除去した識別子名の集合に含まれる各識別子名をキーワードとして, タグクラウドを生成する. キーワードの大きさを決定する重みは, STEP3 で必要な値を求めているため, それを利用して計算する. また同様に, キーワードの着色についても, STEP3 で調べたコードクローン集合に属するコード片に含まれているか否かの情報を利用して行うことができる.

4　開発ツール

本研究では, 3 節で説明した識別子名のタグクラウドをユーザに対して提供するためのコードクローン分析ツールの開発を行った. 本節では, 本ツールにより提供されるコードクローンの分析環境, 及び, 本ツールの利用手順について説明する.

4.1　コードクローン分析環境

本ツールが提供するコードクローン分析環境は, コードクローンの密度を示したクローン散布図, 提案手法により生成される識別子名のタグクラウド, 及び, タグクラウドの対象範囲における, ユーザが選択した識別子名が含まれているコードクローンを表示するコードビューの 3 つの環境で構成されている. 本節では, これらのコードクローンの分析のための環境について説明する.

4.1.1　ディレクトリ単位のコードクローン密度を示した散布図

本ツールにおいて提供されるクローン散布図は, Live scatterplot[4] と呼ばれる散布図を参考にしている. Live scatterplot は, 2.3 節で述べたような一般的なクローン散布図とは異なる特徴を持っている. 最も大きな違いは, 一般的なクローン散布図が, ソースコード中の各トークン対の一致・不一致を示しているのに対し, Live scatterplot は, ディレクトリ対 (あるいは, コンポーネントといったソフトウェア要素の単位) ごとのコードクローンの密度を示しているという点である. また, 大規模なソフトウェアを対象にする場合, 一般的なクローン散布図は, トークン単位で大量の点を打たなければならないという仕様上, 散布図の描画に時間がかかる, 点が密集する場所では可視性が悪くなるといった問題があると考えられる. しかし, Live scatterplot では, ディレクトリ, あるいは, それ以上のソフトウェア要素の単位で描画を行うため, 上述のような問題を大きく緩和し, より大規模なソフトウェアに適用が可能になると考えられる.

図 4 は, Apache Ant[2] に本ツールを適用した場合に, 実際に表示されたクローン散布図の例である. 図 4 における各番号が指す場所の説明を以下に示す.

A. コードクローンの密度を示したクローン散布図

分析対象全体に関するコードクローンの密度を示したクローン散布図である. 縦軸・横軸は, それぞれ同じ順序で並んだ分析対象となるソフトウェアのディレクトリの列であり, 散布図の左上端が原点となっている. また, 縦軸と横軸のディレクトリ間にお

[2]http://ant.apache.org/ .

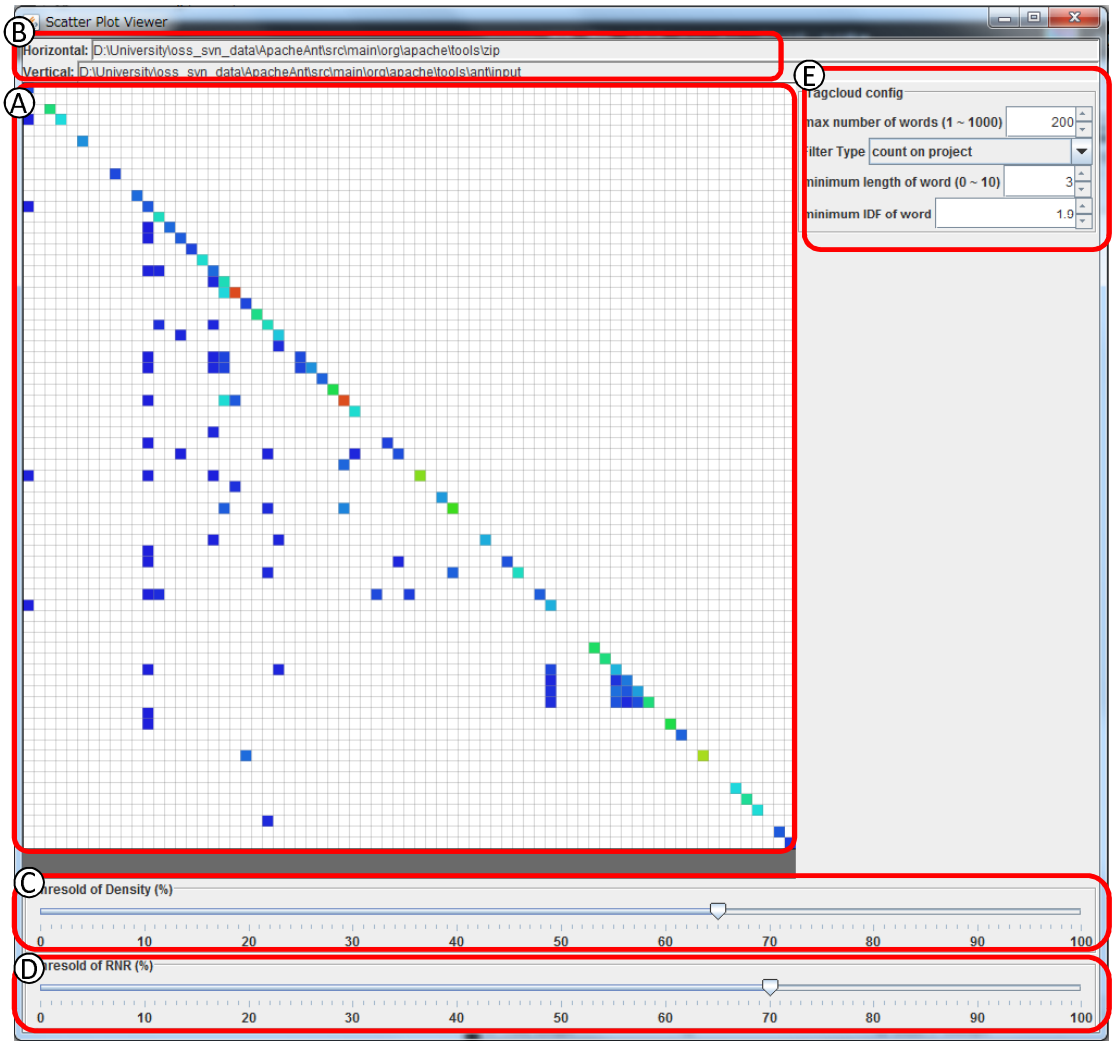


図 4: 本ツールにおけるクローン散布図の例

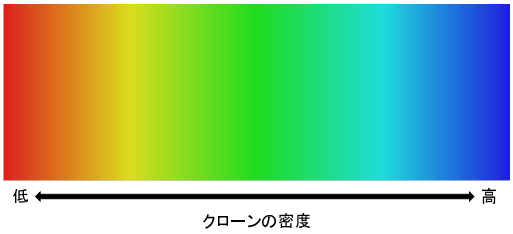


図 5: コードクローンの密度に対する散布図の色の変化

いてコードクローンが存在する場合, 対応するマスをコードクローンの密度に基づいて色付けしている.

本ツールにおいて, コードクローンの密度はソースコード全体のトークン数に対する, コードクローンとなってるコード片に含まれるトークン数の割合としている. すなわち, 縦軸と横軸の 2 つのディレクトリに属する全ソースコードのトークン数を N, そのうち, コードクローンとなってるコード片に含まれるトークン数を $n_c(\leq N)$ とした場合, それら 2 つのディレクトリ間におけるコードクローンの密度 d は以下の式で計算される.

$$d = \frac{n_c}{N}$$

また, 本ツールにおいて提供されるクローン散布図では, コードクローンの密度が色で表現されている. 図 5 は, 本ツールにおけるクローン散布図上で, 各コードクローンの密度に対して着色する色を示したものである. 本ツールでは, コードクローンの密度が低い場合は青色で着色され, コードクローンの密度が高くなるに従って, 色相が赤色に変化していく. コードクローンの密度が最も高いと判断される数値, すなわち, 赤色で表示されるために必要な密度の下限は, 図 4 の 3. の場所で設定可能であり, 設定した数値以上のコードクローンの密度は, 全て赤色で描画される.

B. 縦軸・横軸方向のディレクトリの絶対パス

図 4 の 2. のクローン散布図において, マウスカーソルがあるマスに対応した, 縦軸・横軸方向のディレクトリの絶対パスを表示する場所である.

C. クローン密度の下限を設定するスライダー

図 4 の 2. のクローン散布図のマスの色を決める際に使用する, 赤色で表示されるために必要な密度の下限を設定する場所である.

D. RNR の閾値を設定するスライダー

3.2 節で説明した手順の STEP2 において, RNR によるコードクローンのフィルタリン

グを行う際に使用する RNR の閾値を設定する場所である.

E. 識別子名のタグクラウドのパラメータ設定

4.1.2 節で述べる識別子名のタグクラウドを生成する際に使用する様々なパラメータを設定する場所である. 本ツールにおいて, 設定可能なパラメータを以下に示す.

表示上限数

識別子名のタグクラウドに表示できるキーワードの上限数を設定できる.

表示優先順位の指標

識別子名のタグクラウドに表示するキーワードの優先順位を定める指標を指定できる. 本手法では, 識別子名の出現回数を優先順位を定める指標としているが, 本ツールでは, 優先順位を定める指標を TF-IDF にすることもできる.

表示する識別子名の長さの下限

3.1.2 節で説明した, 不要な識別子として除去される識別子名の長さの閾値を設定できる.

表示する識別子名の IDF の下限

3.1.2 節で説明した, 不要な識別子として除去される識別子名の IDF の閾値を設定できる.

本ツールにおいて, クローン散布図 (図 4 の 2.) の主な用途として, 注目するディレクトリの削減が挙げられる. 本ツールにおけるクローン散布図を見ることで, どのディレクトリ間のコードクローンの密度が高いのかを直観的に把握することができる. 一般的に, コードクローンの密度が低いディレクトリ対では, 実際に含まれているコードクローンの量も少ないと考えられる. そのため, コードクローンの密度が低いディレクトリ対は, 特に大規模なソフトウェアを対象にする場合, そのコードクローン分析における重要度は低いと考えられる. 従って, コードクローンの密度が高いディレクトリのみに着目することで, 分析するコードクローンをより重要度の高いものに限定することが可能になると考えられる. 本ツールにおけるクローン散布図は, 上述のような方法により分析するコードクローンの数を減らすことで, 分析における時間的効率をより良くすることを主な目的としている.

なお, このクローン散布図の全体を通した利用手順については, 4.2 節の手順 2, 3 で説明する.

4.1.2 提案手法を用いた識別子名のタグクラウド

本ツールでは, 4.1.1 節で説明したクローン散布図における 1 組のディレクトリ対を対象に, 3 節で説明した手法を用いて識別子名のタグクラウドの生成を行っている. 従って, 本

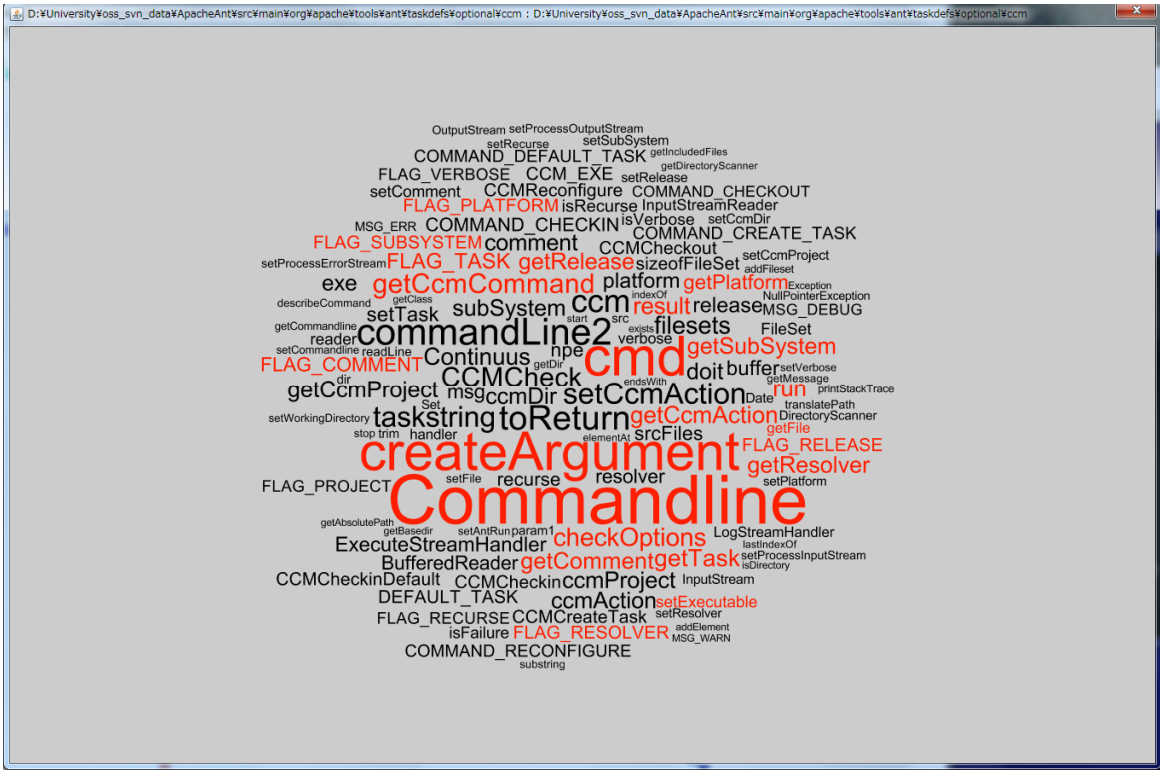
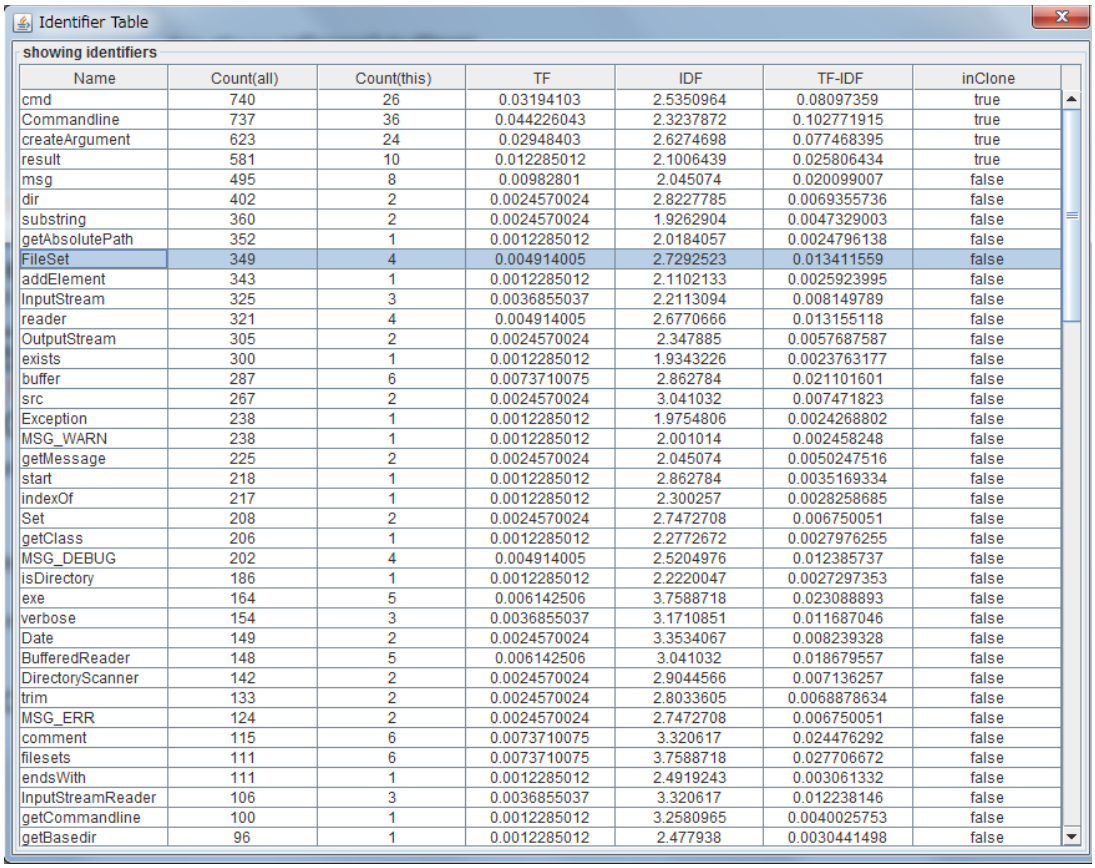


図 6: 本ツールにおける識別子名のタグクラウド



Name	Count(all)	Count(this)	TF	IDF	TF-IDF	inClone
cmd	740	26	0.03194103	2.5350964	0.08097359	true
Commandline	737	36	0.044226043	2.3237872	0.102771915	true
createArgument	623	24	0.02948403	2.6274698	0.077468395	true
result	581	10	0.012285012	2.1006439	0.025806434	true
msg	495	8	0.00982801	2.045074	0.020099007	false
dir	402	2	0.0024570024	2.8227785	0.0069355736	false
substring	360	2	0.0024570024	1.9262904	0.0047329003	false
getAbsolutePath	352	1	0.0012285012	2.0184057	0.0024796138	false
FileSet	349	4	0.004914005	2.7292523	0.013411559	false
addElement	343	1	0.0012285012	2.1102133	0.0025923995	false
InputStream	325	3	0.0036855037	2.2113094	0.008149789	false
reader	321	4	0.004914005	2.6770666	0.013155118	false
OutputStream	305	2	0.0024570024	2.347885	0.0057687587	false
exists	300	1	0.0012285012	1.9343226	0.0023763177	false
buffer	287	6	0.0073710075	2.862784	0.021101601	false
src	267	2	0.0024570024	3.041032	0.007471823	false
Exception	238	1	0.0012285012	1.9754806	0.0024268802	false
MSG_WARN	238	1	0.0012285012	2.001014	0.002458248	false
getMessage	225	2	0.0024570024	2.045074	0.0050247516	false
start	218	1	0.0012285012	2.862784	0.0035169334	false
indexOf	217	1	0.0012285012	2.300257	0.0028258685	false
Set	208	2	0.0024570024	2.7472708	0.006750051	false
getClass	206	1	0.0012285012	2.2772672	0.0027976255	false
MSG_DEBUG	202	4	0.004914005	2.5204976	0.012385737	false
isDirectory	186	1	0.0012285012	2.2220047	0.0027297353	false
exe	164	5	0.006142506	3.7588718	0.023088893	false
verbose	154	3	0.0036855037	3.1710851	0.011687046	false
Date	149	2	0.0024570024	3.3534067	0.008239328	false
BufferedReader	148	5	0.006142506	3.041032	0.018679557	false
DirectoryScanner	142	2	0.0024570024	2.9044566	0.007136257	false
trim	133	2	0.0024570024	2.8033605	0.0068878634	false
MSG_ERR	124	2	0.0024570024	2.7472708	0.006750051	false
comment	115	6	0.0073710075	3.320617	0.024476292	false
filesets	111	6	0.0073710075	3.7588718	0.027706672	false
endsWith	111	1	0.0012285012	2.4919243	0.003061332	false
InputStreamReader	106	3	0.0036855037	3.320617	0.012238146	false
getCommandline	100	1	0.0012285012	3.2580965	0.0040025753	false
getBasedir	96	1	0.0012285012	2.477938	0.0030441498	false

図 7: 本ツールにおける識別子名の情報テーブル

ツールが識別子名のタグクラウドを生成する際に対象となるコードクローンは, ユーザが選択したディレクトリ対の間におけるコードクローンとなる. 図 6 は, 図 4 のクローン散布図 (図 4 の 2.) において, あるディレクトリ対を選択した場合に, 実際に生成された識別子名のタグクラウドの例である. 図 6 において, コードクローンとなっているコード片に含まれている識別子名は赤色のキーワードとして表示されており, コードクローンに含まれていない識別子名は黒色のキーワードとして表示されている.

本ツールにおける識別子名のタグクラウドは, そのタグクラウドと対応するディレクトリ間のコードクローンの内容や, 対象のディレクトリ対におけるコードクローンとなっているコード片の役割を大まかに把握することを主な目的としている. 識別子名のタグクラウドによりコードクローンの大まかな内容を直観的に把握することで, 関心の無いコードクローンを除外するといったことが可能であると考えられる. また, これにより, 実際に読む必要のあるソースコードの量を減らすことができると考えられる.

また, 本ツールでは, 識別子名のタグクラウドと共に, 識別子名の情報テーブルを表示して

いる. 図 7 は, 図 6 と共に表示される実際の識別子名の情報テーブルである. 識別子名の情報テーブルは, 同時に表示された識別子名のタグクラウドに表示されている識別子名に関する様々な情報を記載した表である. 本ツールにおける識別子名の情報テーブルに表示されている情報を以下に示す.

- タグクラウドに表示されている識別子名.
- 分析対象のソースコード全体における識別子名の出現回数.
- 識別子名のタグクラウドが対象としているディレクトリ対における識別子名の出現回数.
- 識別子名の出現頻度 (TF), 逆文書頻度 (IDF), 及び, TF-IDF の数値.
- 識別子名がタグクラウドが対象としているディレクトリ対におけるコードクローンに含まれているか否か.

本ツールにおいて, この識別子名の情報テーブルは, 識別子名同士の厳密な比較を行うために利用することを想定している. 例えば, 識別子名のタグクラウドでは, TF-IDF によって各キーワードの大きさが定められており, 相対的に大きく表示されているものが重要なキーワードであることが直観的に理解できる. しかし, 同じ程度の大きさの 2 つのキーワードを比較する場合, 目視では正確さに欠ける可能性がある. そのような場合, 識別子名の情報テーブルに記載された具体的な数値を参考にして, キーワード (識別子名) の厳密な比較を行う.

なお, この識別子名のタグクラウド, 及び, 識別子名の情報テーブルの全体を通した利用手順については, 4.2 節の手順 4, 5 で説明する.

4.1.3 識別子名を含んでいるコードクローンの表示

本ツールにおいて, 4.1.2 節で説明した識別子名のタグクラウドに表示されている識別子名のうち, コードクローンに含まれている識別子名に関しては, その識別子名が含まれている実際のコードクローンを表示することが可能となっている. 図 8 は, 図 6 のタグクラウドに表示されている識別子名 “createArgument” に対する, 実際のコードクローンの例である. 図 8 における各番号が指す場所の説明を以下に示す.

A. ディレクトリ対の絶対パス

識別子名のタグクラウドが対象としているディレクトリ対の絶対パスを表示する場所である.

B. 選択した識別子名

ユーザが選択した識別子名を表示している.

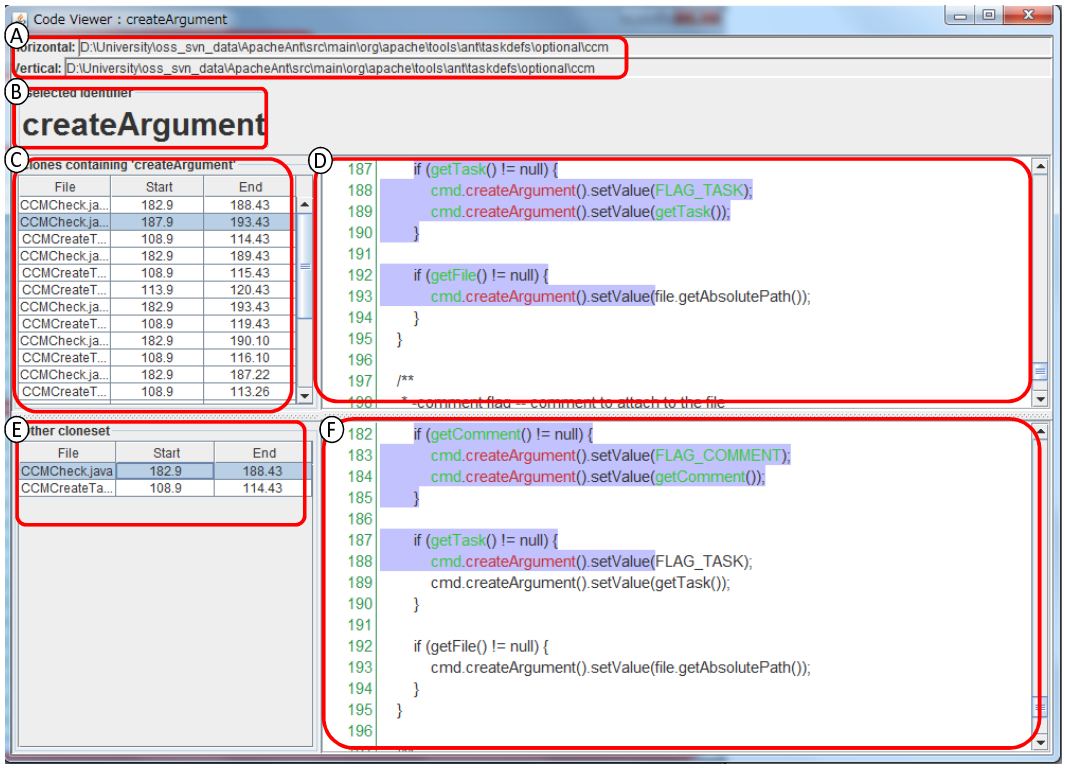


図 8: 本ツールにおける実際のコードクローンの表示例

C. 識別子名を含むコードクローンのリスト

選択した識別子名 (図 8 の 2.) を含んでいる, 対象ディレクトリ対におけるコードクローンのリストを表示している.

D. 識別子名を含むコードクローンを含むソースコード

図 8 の 3. で選択したコードクローンを含むソースコードを表示する場所である. また, 選択したコードクローンに該当する場所を青色の背景で強調表示している. さらに, 選択した識別子名を赤色, その他の識別子名のタグクラウドに表示されている識別子名を緑色でそれぞれ強調表示している.

E. 同じクローンセットに属する他のコードクローンのリスト

図 8 の 3. で選択したコードクローンと同じクローンセットに属する他のコードクローンのリストを表示している.

F. 同じクローンセットに属する他のコードクローンを含むソースコード

図 8 の 5. で選択したコードクローンを含むソースコードを表示する場所である. 図 8 の 4. と同様, 選択したコードクローンに該当する場所や選択した識別子名を強調表示している.

本ツールにおいて, ユーザが選択した識別子を含んでいるコードクローンを表示する目的は, 関心のあるキーワードを含んでいるコードクローンの, 実際のコード片を確認することである. すなわち, 識別子名のタグクラウドを見て, 関心のあるキーワードがあった場合, そのキーワードに関するコードクローンの詳細を見ることが可能である.

なお, この識別子名を含むコードクローンを表示するコードビューの全体を通した利用手順については, 4.2 節の手順 6 で説明する.

4.2 利用手順

図 9 は, 本ツールを利用する手順の概要を表した図である. 本ツールは, CCFinder の出力結果を入力として動作する. そのため, ユーザは分析対象のソースコードを CCFinder に適用した結果をあらかじめ準備する必要がある.

本ツールを用いてコードクローン分析を行う場合の利用手順を以下に示す.

1. CCFinder を用いて, 分析対象のコードクローン抽出を行い, 出力結果を得る.
2. 手順 1. で得た出力結果を入力として, 本ツールを実行する. 本ツールを実行すると, 最初に, 4.1.1 節で説明した分析対象全体におけるクローン散布図が表示される. 4.1.1 節でも述べた通り, 本ツールにおけるクローン散布図は, 一般的なクローン散布図をとは

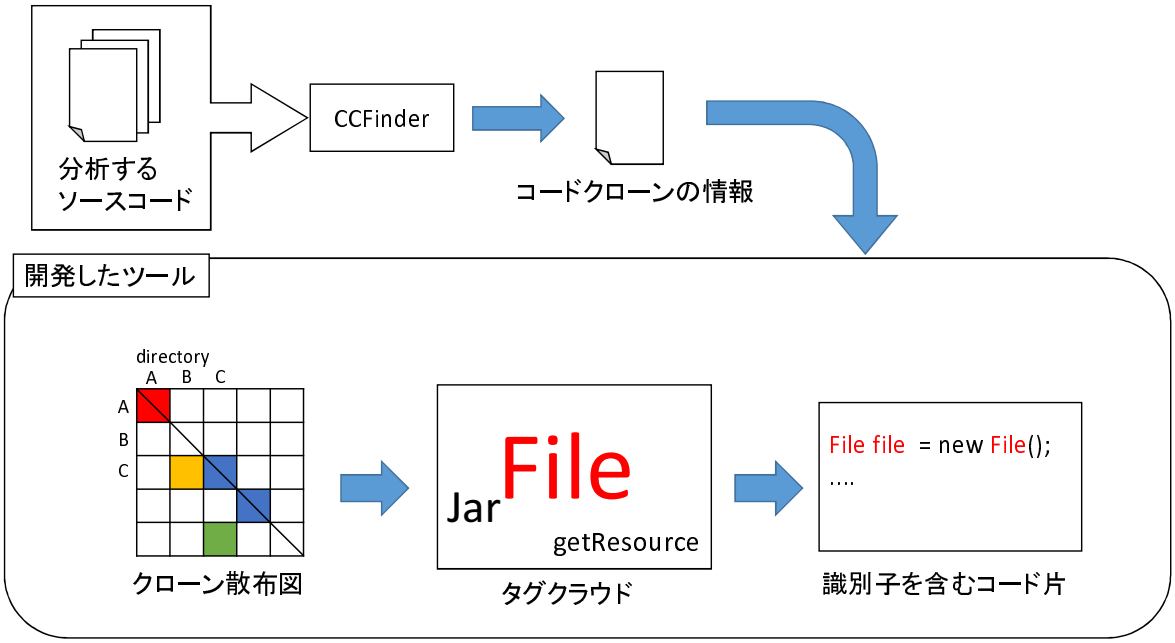


図 9: ツールの利用手順の概要図

異なり, 2 つのディレクトリ間におけるコードクローンの密度を示した散布図となっている.

3. クローン散布図を利用して, 関心の高いディレクトリ対を選択する. 関心の高いディレクトリ対の例としては, コードクローンの密度が高いディレクトリ対が挙げられる.

4. 手順 3. で注目したディレクトリ対を選択し, 4.1.2 節で説明した, 提案手法を用いた識別子名のタグクラウドを表示する. 本ツールにおいて提案手法が適用される範囲は, 注目したディレクトリ対に属するソースコードである. また, このとき, 4.1.2 節で説明した識別子の情報テーブルも同時に表示される.

5. 手順 4. で表示されたタグクラウドを確認し, 注目したディレクトリ対の大まかな処理内容を把握する. また, 関心のあるキーワードが存在しているか探索する. 必要であれば, 手順 3. でタグクラウドと共に表示された識別子の情報テーブルを利用して, キーワードの厳密な大小関係を把握する.

6. 手順 5. で関心のあるキーワード (識別子名) が表示されている場合は, そのキーワードを選択し, 4.1.3 節で説明した, 識別子名を含んでいる実際のコードクローンを確認できるコードビューを表示する. このコードビューを用いて, 実際のコードクローンとなっているコード片を読むといった, 詳細な調査を行うことが可能である.

5　評価実験

本研究では, 提案手法の有用性を評価するための実験を行った. 本節では, 本研究で行った評価実験とその結果について説明する.

5.1　提案手法により抽出されたキーワードの有用性評価

本節では, 提案手法により抽出された, タグクラウドに表示するキーワードの有用性に関する評価実験について説明する.

5.1.1　実験方法

本実験では, あらかじめ有用と考えられるキーワードを定めておき, 提案手法を用いて生成された識別子名のタグクラウドに表示されているキーワードがそれらと一致するか, あるいは推測が可能であるかを調べることにより, その有用性の評価を行う. あらかじめ定めておくキーワードは, コードクローンの分析経験を持つ専門家が書いたコードクローンの説明文 (自然文) を元に決定する. 今回の実験では, あらかじめ定めておくキーワードは, 実験対象となるディレクトリ対におけるコードクローンの説明文に出現する名詞と動詞に限定している. また, 代名詞や be 動詞といった明らかにキーワードにはならないと考えられる単語はキーワードから除外している.

本実験においては, 説明文のキーワードが, タグクラウドに表示されているキーワードと全く同じであるか, あるいは, タグクラウドに表示されているキーワードの一部に含まれている場合, キーワードが一致したとみなす. なお, 動詞の現在形と過去形や, 名詞の単数形と複数形といった文法的な変化については, 同じキーワードであるとみなしている. また, 例えば, “Windows”というキーワードから “OS”というキーワードが推測できるように, 説明文のキーワードかタグクラウドに表示されているキーワードが推測できる場合, 抽出されたキーワードは推測可能であるとみなす. “utility”と “util”のように一般的によく見られる省略語や, “load”と “loader”のように品詞が異なる場合も, 推測可能とみなしている. この理由は, 前者については, 万人が省略語と 1 対 1 対応の単語を確実に推測できるという保証が無く, 後者については, 英単語の中には品詞によって意味が変わってしまうものが存在するため, どちらも一致としてしまうには問題があると考えられるためである.

本実験では, 以下の 2 つの指標を元にキーワードの有用性を評価した. 以下の 2 つの指標では, タグクラウドに表示された “コードクローンに含まれている”キーワードのみが対象となっている. これは, 説明文がコードクローンのみを対象に書かれているため, タグクラウドのキーワードも説明文と同じ範囲に限定するべきであると考えられたためである.

1. 説明文のキーワードのうち, 識別子名のタグクラウドに表示されている, コードクローンに含まれているキーワードと一致, または, タグクラウドのキーワードを推測するヒントとなったキーワードの割合. この指標は, 実際にタグクラウドに表示されている説明文のキーワードの割合を示しており, 提案手法によるタグクラウドが, 有用なキーワードをどの程度取得し, 表示できているかの指標となる.

2. 識別子名のタグクラウドに表示されたコードクローンに含まれているキーワードにおいて, 説明文のキーワードと一致, または, 説明文のキーワードから推測可能と考えられるキーワードの割合. この指標は, タグクラウドに表示されているキーワードが説明文のキーワードとなっている割合を示しており, 提案手法によるタグクラウドに表示されているキーワードのうち, 有用なキーワードがどの程度存在しているかの指標となる. また, 逆に, この指標が低い値を示していることは, タグクラウドに表示されているキーワードの中に, 不要なキーワードが多く存在していることを意味する.

本実験では, Apache Ant の 10 個のディレクトリ対を対象に評価を行った. 表 2 に評価実験の対象とした Apache Ant のソースコードのディレクトリパスを示す. 表 2 の各行は, 1 つ目のディレクトリと 2 つ目のディレクトリを組としたディレクトリ対を表している. また, 表 2 における各ディレクトリパスは, 先頭の `org/apache/tools/ant/`を省略したものとなっている. これらのディレクトリ対は, 原則, Apache Ant の各ディレクトリ対におけるコードクローンの密度の高さが上位のものを選択しているが, 上位 10 組が選ばれている訳ではない. これは, 第三者にコードクローンの説明文を書いてもらうという都合上, 多数のコードクローンが存在するディレクトリ対を対象にすることが難しかったためである.

5.1.2 実験結果と考察

表 3 に, 本実験における実験結果を示した. 表 3 において, 指標 1 は, 説明文のキーワードのうち, 識別子名のタグクラウドに表示されているキーワードの割合である. すなわち, 本手法における有用なキーワードの取得率を意味している. 指標 2 は, 識別子名のタグクラウドに表示されているキーワードのうち, 説明文のキーワードと一致する, あるいは説明文のキーワードから推測可能であるキーワードの割合である. すなわち, 識別子名のタグクラウドの全てのキーワードに対する有用なキーワードの割合を示している.

表 3 からわかるように, 実験の結果では, 指標 1 と指標 2 は共に平均値が約 80%であった. 従って, 平均的に見ると, 本手法を用いて生成された識別子名のタグクラウドは, ほとんどの有用なキーワードが表示されており, また, 表示されているキーワードの中に不要なキーワードはほとんど存在しないということになる. しかし, 表 3 の一部の結果には, ディレクト

表 2: 実験対象 (抽出されたキーワードの評価)

1 つ目のディレクトリ	2 つ目のディレクトリ
taskdefs/optional/clearcase	taskdefs/optional/clearcase
taskdefs/optional/javah	taskdefs/optional/javah
taskdefs/optional/sos	taskdefs/optional/sos
types/spi	types/spi
types/optional/image	types/optional/image
taskdefs/optional/depend/constantpool	taskdefs/optional/depend/constantpool
util/regexp	util/regexp
taskdefs/launcher	taskdefs/launcher
taskdefs/optional/ccm	taskdefs/optional/ccm
types/resolver	types/resolver

表 3: 実験結果 (抽出されたキーワードの評価)

ディレクトリ対		指標 1 (%)	指標 2 (%)
taskdefs/optional/clearcase	taskdefs/optional/clearcase	88.0	84.3
taskdefs/optional/javah	taskdefs/optional/javah	90.0	75.0
taskdefs/optional/sos	taskdefs/optional/sos	95.0	100.0
types/spi	types/spi	66.7	100.0
types/optional/image	types/optional/image	92.9	78.8
taskdefs/optional/depend /constantpool	taskdefs/optional/depend /constantpool	71.4	100.0
util/regexp	util/regexp	47.8	66.7
taskdefs/launcher	taskdefs/launcher	80.0	66.7
taskdefs/optional/ccm	taskdefs/optional/ccm	100.0	90.9
types/resolver	types/resolver	66.7	70.6
平均		79.8	83.3

リ対 {util/regexp, util/regexp} といった, 極端に低い数値を示しているものが存在していることがわかる. このような結果が生じた要因としては, 以下のような可能性が考えらえれる.

- 本実験では, あらかじめコードクローンの説明文を第三者に書いてもらっているが, コードクローンの中には何らかの処理の途中から始まり, 途中で終わるような中途半端なものが存在している. このようなコードクローンを自然文で説明することは難しく, 結果として説明文に基づいて作成した有用なキーワードのリストが不正確なものとなってしまった可能性がある. これは, 指標 1 と指標 2 の両方の結果に影響を与える可能性があると考えられる.

- 本手法では, 出現回数の多い識別子名が優先的に識別子名のタグクラウドに表示されるようになっている. 従って, 出現回数の少ない識別子名は, 識別子名のタグクラウドに表示されない. ゆえに, このような識別子名が深く関係しているコードクローンが存在する場合, 有用なキーワードを取得できていないことになってしまう. これにより, 指標 1 の数値が低くなってしまった可能性が考えられる.

- 本手法では, 一般性の高いキーワードはタグクラウドに表示されないようになっている. ゆえに, コードクローンの処理が, 文字列の操作といった低水準で汎用的に利用される処理であった場合, このような処理に関するキーワードがタグクラウドに表示されない可能性がある. これにより, 指標 1 の数値が低くなってしまった可能性が考えられる.

- コードクローンとなっている部分の処理が, そのコードクローンを含むソースコード全体と直接的に関係の無い処理の場合, 説明文への反映が難しく, 結果として直接的な説明が書かれない可能性がある. この場合, 例えば, 正規表現に関するソースコードにおいて, 文字列の比較を行う処理を “正規表現のパターンマッチに関する処理” と説明するように, 処理を直接的に説明しない可能性があると考えられる. このような場合, 説明文に基づくキーワードリストに, タグクラウドに表示されているキーワードは含まれていないことになるため, 結果として指標 1 が低くなってしまうと考えられる. また, 本来コードクローンのキーワードとなるべき識別子名が, 有用なキーワードのリストに含まれていないので, 指標 2 の数値も同様に, 低くなってしまうと考えられる.

- 本実験の対象はコードクローンの密度に基づいて選択したため, 実際に含まれているコードクローンの絶対数が少ないという対象も存在している. そのような対象においては, 有用なキーワードや, タグクラウドに表示されているキーワードの数が少ない可能性がある. そのため, 1 つのキーワードに掛かる重みが大きくなり, 指標の数値が低

くなってしまう可能性がある. 例えば, 有用なキーワードが 3 つしか挙がらなかった場合, 1 つ取得できなかっただけで, 指標 1 の数値が 2/3 にまで下がってしまう.

本実験ではコードクローン分析の専門家が記述した説明文を利用して, 有用なキーワードを選定した. しかし, 例えば, 本実験の対象である Apache Ant に詳しい専門家が記述した説明文を利用する場合, 説明文の詳細の程度が異なる可能性があるため, 結果に変化が生じる可能性が考えられる. また, 説明文は自然文であるという仕様上, 記述する人によって個人差が出る可能性が高い. ゆえに, 様々な人が記述した説明文を用いて, 実験を行う必要があると考えられる. 同様に, 実験対象をコードクローンの密度が低いディレクトリ対や, Apache Ant 以外のソフトウェアに変更した場合にも結果に変化が生じる可能性があると考えられる.

また, 本研究では, コードクローン検出に CCFinder を利用しているが, 異なる検出手法を実装しているコードクローン検出ツールを利用することでも, 結果に変化が生じると考えられる. 例えば, 識別子名の類似度に基づいたコードクローン検出手法 [19] を利用しているツールを用いた場合, 内部に出現する識別子名が類似しているメソッドが同じクローンセットのコードクローンとして検出される. このような検出ツールを利用した場合, 識別子名ベースの検出手法であることが, 結果に影響を与えるかもしれない. また, メソッド単位のコードクローンであれば, 説明文を記述することが容易になると考えられるため, 上述の結果が悪くなる要因の一部が解決される可能性が考えられる.

5.2 識別子名をタグクラウドで表示することの有用性評価

本節では, 本研究における提案手法を実装したコードクローン分析ツールを用いた, 識別子名のタグクラウドに関する有用性の評価実験について説明する.

5.2.1 実験方法

本実験では, 本研究において開発したコードクローン分析ツールを利用し, 2 つのディレクトリ間のコードクローンに関する説明文を被験者に記述してもらう. そして, 被験者が記述した説明文の正確さと, その説明文の記述に要した時間を計測することで, 識別子名のタグクラウドに関する有用性の評価を行った. また, 比較対象としてコードクローン分析ツール Gemini[17] を利用して, 被験者に同様の作業を行ってもらった.

本実験において, 被験者はコンピュータサイエンスを専攻する大学院生 4 名である. また, コードクローンの説明文を記述する対象には, Apache Ant の 2 組のディレクトリ対を選択した. 表 4 に評価実験の対象とした Apache Ant の 2 組のディレクトリ対のそれぞれのディレクトリパスを示す. 表 4 の各行は, 1 つ目のディレクトリと 2 つ目のディレクトリを組としたディレクトリ対を表している. また, 表 4 における各ディレクトリパスは, 先頭の

src/main/org/apache/tools/ant/を省略したものとなっている．これらのディレクトリ対は, 表 2 に示されているコードクローンの密度が高いディレクトリ対の中から選んだものである．コードクローンの絶対量が多く, かつ, 類似した傾向のコードクローンを持つディレクトリ対にならないことを基準にして, これら 2 組のディレクトリ対を選んでいる．

本実験における, 具体的な手順を以下に示す．

1. 被験者に対して, 実験で使用するツール (本ツールと Gemini) の使用方法を説明する．また, それぞれのツールに関して, 自由に利用できる時間を 5 分間設けている．さらに, 本実験の対象以外のディレクトリ対を対象にして, 説明文を記述する時間を 10 分間設けている．なお, これらの時間中に, 被験者が実験対象のディレクトリ対に関する情報を得ないように, 本手順では, Apache Ant のソースコードから実験対象となるディレクトリと, その直下に属するソースコードをあらかじめ取り除いたものを配布して行う．本実験で使用する対象ディレクトリ対を含む, 全ての Apache Ant のソースコードは, 本実験の開始直前に配布する．

2. 被験者 4 名のうち, 2 名には Gemini を利用して, 表 4 の 1 行目のディレクトリ対におけるコードクローンの説明文を記述してもらう．残りの 2 名には, 本ツールを利用して, 同様に説明文を記述してもらう．本実験におけるコードクローンの説明文とは, どのような処理がコードクローンになっているのかに関する説明文を意味している．本実験では, 説明文の記述における制限時間を 30 分としている．また, その時間よりも早くに説明文が完成した場合は, その時点で報告をするように被験者に指示をしている．被験者から完成の報告を受けた場合は, そこまでの経過時間を記録する．なお, 本実験の目的は識別子名のタグクラウドの評価であるため, 本ツールや Gemini を用いて対象のディレクトリ対を探すまでの手順は計測時間には含めない．

   説明文は, 英語能力の差による記述時間の違いが出ないようにするため, 日本語で記述するように指示をする．ただし, 本実験における説明文の正確さの評価では, 説明文が英語で書かれている必要があったため, 実験終了後に, 英語に翻訳してもらう．

3. 手順 2 で Gemini を利用した被験者は本ツールを, 本ツールを利用していた被験者は

表 4: 実験対象 (識別子名タグクラウドの評価)

1 つ目のディレクトリ	2 つ目のディレクトリ
types/optional/image	types/optional/image
taskdefs/optional/clearcase	taskdefs/optional/clearcase

Gemini を利用して, 手順 2 と同様に, 表 4 の 2 行目のディレクトリ対におけるコードクローンの説明文を記述してもらう.

4. 手順 2, 3 で記述した説明文を英語に翻訳してもらう. また, 同時に, 本実験に関するアンケートを行う. これらに関しては, 制限時間は設けていない.

本実験では被験者が記述した説明文の正確さを, 5.1 節の評価実験で利用した説明文と比較することで定量的に評価する. 具体的には, 以下の 2 つの指標に基づいて評価を行う. 以下の説明において, 正解の説明文とは, 5.1 節の評価実験で利用した説明文のことを意味している. また, 説明文のキーワードは, 5.1 節の評価実験と同様, 動詞と名詞のみを対象にしている.

1. 被験者が記述した説明文のキーワードのうち, 正解の説明文のキーワードと一致する, または, 正解の説明文のキーワードを推測できるものの割合.
2. 正解の説明文のキーワードのうち, 被験者が記述した説明文に含まれるものの割合.

さらに, 本実験では, 被験者に対してアンケートを実施した. 実施したアンケートの内容を以下に示す.

Q1. Java の経験ついて

- 講義で習ったことがあるか.
- 研究の分析対象として利用しているか, あるいは, ツール開発に利用しているか.
- アルバイトで利用しているか.
- その他, Java を利用する場面があるか.
- Java の経験年数はどの程度か.

Q2. コードクローンの知識について

- コードクローンに関する研究を行っているか.
- コードクローンの分析経験があるか.
- コードクローンに関する保守経験があるか.

Q3. Apache Ant の知識について

- ユーザとして使用したことがあるか.
- ソースコードを読んだことがあるか.

Q4. 比較対象のツールについて

- Gemini を使用したことがあるか.

Q5. 提案ツールに関して, Gemini と比較して良かった点・悪かった点.

Q6. 実験対象の各ディレクトリ対に関して説明文を書く際, 何に時間が掛かったのか.

Q7. 自由記述.

Q1, Q2, Q3 は, 本実験においてコードクローンの説明文を記述する時間に影響を与える可能性がある事前知識に関する質問である. また, Q4 は, 本ツールとの比較対象である Gemini の事前知識に関する質問である. Q5 は, 実際に使用したユーザ視点での, 本ツールの利点・欠点を調査するための質問であり, Q6 は, コードクローンの説明文を記述する際に時間のかかった工程を調べるための質問である. Q7 は自由記述であり, 被験者が実験やツールの感想等を自由に記述することができる項目である.

5.2.2 実験結果と考察

本実験において, 被験者が説明文を記述するのに掛かった時間を表 5 に示す. 表 5 には, 各ツールを利用した場合における説明文の記述に掛かった時間, 及び, 各ツールにおける平均記述時間が示されている. なお, 本実験では, 制限時間が 30 分であるため, 時間内に説明文が完成しなかった場合は, “30 分 00 秒” と記載されている.

表 5 を見ると, 対象が異なるため厳密ではないが, Gemini を利用した方が早かった被験者が 2 名, 本ツールを利用した方が早かった被験者が 1 名であった. また, 説明文の記述に掛かった平均時間についても, Gemini の方が短いという結果であった. 従って, Gemini を利用した方が説明文の記述時間が短い傾向にあると考えられる. しかし, t 検定を用いて, 説明文の記述に掛かった平均時間の統計的な有意差を判定した結果, 有意水準 0.05 のもとで有意

表 5: 説明文の記述に掛かった時間

被験者	Gemini	提案ツール
A	30 分 00 秒	30 分 00 秒
B	14 分 25 秒	19 分 35 秒
C	30 分 00 秒	28 分 18 秒
D	19 分 39 秒	30 分 00 秒
平均	23 分 31 秒	26 分 58 秒

差は無いことがわかった. 従って, Gemini と提案ツールの説明文の平均記述時間の差は, 統計的な有意な差ではないと考えられる.

表 5 を見ると, 約半数の結果が時間切れとなっていることがわかる. 本実験では, 制限時間を 30 分としていたが, この制限時間が少なかった可能性が高いと考えられる. 従って, 被験者が説明文の記述に掛かった正確な時間を計測できなかったため, 有意な時間差が生じなかったと考えられる. また, 制限時間をより長くするか, あるいは, 無制限にして, 被験者が説明文の記述に掛かった正確な時間を計測する必要があると考えられる.

また, 5.2.1 節で説明した説明文の正確さを評価するための指標を表 6 に示す. 表 6 における, 指標 1, 2 はそれぞれ 5.2.1 節で説明した指標 1, 2 と対応している.

表 6 は, 表 6 に示された説明文の正確さの指標の平均値を示したものである. 表 6 を見ると, 指標 1, 指標 2 の両方について, Gemini よりも提案ツールの方が高い数値を示す結果になった. しかし, Student の t 検定では, 有意水準 5%の下で有意な差ではないという結果になった. このような結果となった要因として, 実験データの不足が考えられる. 従って, さらに多くの被験者に対して同様の実験を行い, より多くの実験データを得る必要があると考えられる.

以下に, 本実験で行ったアンケート結果を示す.

Java の経験

Java の経験年数は, 多くの被験者は 2, 3 年であったが, 半年と答えた者もいた. また, 被験者が全員が講義で習ったことがあると答え, 3 名がツール開発, そのうちの 2 名は分析対象としても利用したことがあると回答した. 従って, 被験者の Java の経験には若干の差があったと考えられる. また, このことが実験結果に影響を与えている可能性が考えられる.

コードクローンに関する知識

表 6: 説明文の正確さの指標

	Gemini		提案ツール	
	指標 1(%)	指標 2(%)	指標 1(%)	指標 2(%)
A	22.2	28.6	50.0	10.0
B	50.0	57.1	50.0	8.0
C	50.0	12.0	41.7	50.0
D	53.3	18.0	42.9	57.1
平均	43.9	28.9	46.1	31.3

被験者のうち, 2 名がコードクローンに関する研究を行っており, そのうち 1 名は保守経験もあると回答した. 残りの 2 名はコードクローンに関する経験は無いと回答した. 従って, コードクローンに関する知識には差があったと考えられる. また, このことが実験結果に影響を与えている可能性が考えられる.

Ant に関する経験

3 名が使用経験があり, そのうち 1 名はソースコードを読んだことがあると答えた. 従って, 被験者の Ant に関する知識には差があったと考えらえる. また, このことが実験結果に影響を与えている可能性が考えられる.

Gemini の経験

被験者全員が, 使用したことがないと回答した. そのため, 本ツールと Gemini に関する被験者の知識の差が, 実験結果に影響は与えている可能性は無いと考えられる.

本ツールの良い点・悪い点

良い点としては, 識別子名のタグクラウドにより, 対象のディレクトリ対が何を扱っているのかが理解しやすいという意見があった. また, 悪い点としては, クローンセットを全て探し出すのに時間が掛かる, 識別子名の強調表示をコードクローン以外のソースコードにも適用した方が良いという意見があった.

時間が掛かった作業

単純に, コードの読解に時間が掛かったという回答が多かった. また, 本ツールの悪い点でも指摘されているが, クローンセットを全て探し出すのに時間が掛かったという回答も見られた.

本実験では, 大学院生 4 名を被験者としているが, 被験者の能力が大きく異なる場合, 実験結果に変化が生じる可能性がある. 例えば, コードクローンの保守経験が長い人物を被験者とした場合, ツールの違いによる説明文を記述する時間やその正確さの差が, より顕著に現れるかもしれない. また, 実験対象のソースコードが, “a”, “b” といった意味を持たない識別子名が使用されていたり, 本来の役割とは無関係な識別子名を使用している場合, 本ツールに関する実験結果は悪くなる可能性が高いと考えられる. 5.1 節の実験と同様, 正解の説明文の詳細の程度が変わることでも, 実験結果に変化が生じる可能性があると考えられる.

また, 5.1 節の実験と同様, 本ツールが利用するコードクローン検出ツールを変更した場合にも, 実験結果は変化すると考えられる. 例えば, 識別子名の類似度に基づいたコードクローン検出ツール [19] を用いた場合, 内部に出現する識別子名が類似しているメソッドが同じクローンセットのコードクローンとして検出されるため, 識別子名のタグクラウドによるコードクローンの理解支援がより有益に働くかもしれない. また, CCFinder には, コードクロー

ンであるとみなすトークン数の長さの下限を設定する機能が備わっている. 本研究においては, この設定はデフォルトのまま使用しているが, この設定を変更することでも, 実験結果が変化するかもしれない. 例えば, より長いトークン数のもののみをコードクローンとみなすようにした場合, 説明することが難しい, ある処理の一部といった小さなコードクローンが除去され, 被験者が記述する説明文の正確さが向上する可能性が考えられる.

6　まとめと今後の課題

本研究では, ソースコード中に含まれる識別子名を抽出し, 重要な識別子名をタグクラウドとして表示することで, コードクローンの内容を直観的に理解する手法を提案した. また, 本手法により生成される識別子名のタグクラウドを開発者に提供するために, 本手法を実装したクローン散布図ベースのコードクローン分析ツールの開発を行った. そして, コードクローンの説明文に基づいた有用な識別子名リストと照らし合わせることで, 本手法により抽出される識別子名が有用なものであることを確認できた.

さらに, 被験者実験で既存のコードクローン分析ツールである Gemini と本ツールを利用し, コードクローンの説明文を書いてもらい, その正確さと要した時間を比較することで, 識別子のタグクラウドの有用性を評価した. しかし, 説明文の記述時間については有益な結果が得られず, また, 正確さについても有意な結果は得られなかった.

今後の課題として, まず, 識別子のタグクラウドの有用性の再評価が必要であると考えられる. また, 本実験はコードクローンの説明文という個人差が顕著に現れるものを対象にしているため, コードクローンの経験や実験対象に使うソフトウェアの経験などが異なる多くの被験者に対して実験を行う必要があると考えられる. 同様に, 実験対象とするソフトウェアについても, 様々なソフトウェアに対して実験を行う必要があると考えられる. これは, 本手法により抽出される識別子名の評価についても同様のことが言える.

ツールについては, 被験者実験におけるアンケートで指摘されたように, コードクローンが多く表示されているため, 全て調べるのに時間が掛かるという問題がある. 本ツールでは, 識別子名を含むコードクローンが全てリストとして表示されている. そのため, 同じクローンセットに属するものが同時にリスト内に複数存在することが起こりうる. これが原因で, 調べる必要のあるコードクローンが増加してしまっていると考えられる. この問題への対処としては, コードクローンのチェック機能が考えられる. すなわち, ユーザがチェックを施したコードクローンは, チェックを施したことが分かるように強調表示をするなどの対策を行うことで, この問題に対応できると考えられる. また, 選択した識別子名やタグクラウドに表示されている識別子名の強調表示の範囲を, ソースコード全体に広げるなどの細かい改善も必要である.

謝辞

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 井上克郎 教授には, 御多忙の中，常に適切な御指導及び御助言を賜りました. 井上 教授の適切な御指導のおかげで, 本論文を完成させることができました. 井上 教授のもとで研究生活を送ることができたこと, 本論文を執筆できたことに厚く御礼申し上げます.

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 松下誠 准教授には, 研究における問題点の提示など, 多くの御助言を賜りました. それらの御助言は, 本研究を遂行する上で非常に役立ちました. 多くの御指導及び御助言を頂いた 松下 准教授に心より深く感謝いたします.

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 石尾隆 助教には, 研究に関してだけではなく, 発表における細かな誤りなど, 様々な御指導を頂きました. それらの御指導は, 非常に勉強になりました. 様々な御指導及び御助言を頂いた 石尾 助教に心より深く感謝いたします.

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学講座 吉田則裕 助教には, 本研究の構成から, 本論文の執筆に至るまで, 終始適切な御指導及び御助言を頂きました. 本論文を執筆することができたのは, 吉田 助教の御指導のおかげであると, 心より深く感謝しております.

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 崔恩瀞 氏, 山中裕樹 氏には, 研究の相談や実験の手伝い, 本論文の修正など, 研究の様々な段階で御協力して頂きました. また, 研究生活においても, 様々な場面で支えて頂きました. 研究生活を有意義に過ごすことができたのは, 両氏のおかげであると心より深く感謝しております.

また, 本研究における被験者実験において, 長時間の実験にも関わらず被験者を快く引き受けてくださった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井上研究室の先輩方に心より深く感謝いたします.

最後に，様々な御指導，御助言等を頂き, 研究生活を支えてくださった, 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井上研究室の皆様に深く感謝いたします.

参考文献

[1] H. A. Basit and S. Jarzabek. Detecting higher-level similarity patterns in programs. In *Proceeding of the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 156–165, 2005.

[2] I. Baxter, A. Yahin, L. Moura, M. Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings of the 14th International Conference on Software Maintenance*, pp. 368–378, 1998.

[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *IEEE Transaction Software Engineering*, Vol. 31, No. 10, pp. 804–818, 2007.

[4] J. R. Cordy. Live scatterplots. In *Proceedings of the 5th International Workshop on Software Clones*, pp. 79–80. ACM, 2011.

[5] 肥後芳樹, 吉田則裕, 楠本真二, 井上克郎. 産学連携に基づいたコードクローン可視化手法の改良と実装. 情報処理学会論文誌, Vol. 48, No. 2, pp. 811–822, 2007.

[6] 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. J91-D, No. 6, pp. 1465–1481, 2008.

[7] J. H. Johnson and E. Merlo. Substring matching for clone detection and change tracking. In *Proceedings of the 10th International Conference on Software Maintenance*, pp. 120–126, 1994.

[8] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transaction Software Engineering*, Vol. 28, No. 1, pp. 654–670, 2002.

[9] C. Kapser and M. W. Godfrey. Cloning considered harmful considered harmful. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pp. 19–28, 2006.

[10] K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M. Bernstein. Experiment on the automatic detection of function clones in a software system using metrics. *Automated Software Engineering*, Vol. 3, pp. 77–108, 1996.

[11] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: finding copy-paste and related bugs in large-scale software code. *IEEE Transaction Software Engineering*, Vol. 32, No. 3, pp. 176–192, 2006.

[12] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the 12th International Conference on Software Maintenance*, pp. 244–253, 1996.

[13] 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, pp. 2178–2188, 2003.

[14] N. Pennington. *empirical studies of programmers: 2nd workshop*, pp. 100–113. Ablex Publishing Corp., 1987.

[15] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with jplag. *Journal of Universal Computer Science*, Vol. 8, No. 11, pp. 1016–1038, 2002.

[16] 徳永健伸. 情報検索と言語処理. 東京大学出版会, 1999.

[17] 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. 86-D-I, No. 12, pp. 863–871, 2003.

[18] A. von Mayrhauser and A.M. Vans. Identification of dynamic comprehension processes during large scale maintenance. *Software Engineering, IEEE Transactions on*, Vol. 22, No. 6, pp. 424–437, 1996.

[19] 山中裕樹, 吉田則裕, 崔恩瀞, 井上克郎. テキストマイニング技術を応用したメソッドクローン検出手法の提案. 情報処理学会研究報告, 第 2013-SE-182 巻, pp. 1–8, 2013.