社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

大規模ソースコード集合を対象とした類似関数集合群の抽出

田中　健介†　　肥後　芳樹†　　楠本　真二†

† 大阪大学大学院情報科学研究科 〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{k-tanaka,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　近年ソフトウェア開発は大規模化の一途をたどっており，開発コスト削減のため，既存の資源を有効に活用することが望まれる．しかし，ソースコードの再利用を行うためには内容を理解しなければならず，特に，再利用を考慮して作成されていない資源の場合，その労力は大きい．本研究では，これまでに実装された機能を効率的に把握・再利用するため，コードクローン検出技術を用いて，複数のソフトウェアから同様の機能を実現した関数集合群を検出する手法を提案する．実験の結果，数千万行のソースコードから多くの類似した機能を持つ関数群を検出することができ，どのような関数が多数利用されているのか容易に調査することができた．
キーワード　プログラム解析，コードクローン，関数分類，再利用

Identifying frequent functionalities from large-scale source code

Kensuke TANAKA†, Yoshiki HIGO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita, Osaka, 565–0871 Japan
E-mail: †{k-tanaka,higo,kusumoto}@ist.osaka-u.ac.jp

Abstract　In recent years software development scale is steadily increasing. To reduce development costs, it is desirable to effectively reuse existing resources. However, it is necessary to understand the source code for reusing it. Especially if the resources are not created with reuse in mind, the greater effort is needed. In this study, to efficiently understand and reuse source code, we identify frequent functionalities from many software systems by clone detection technique. The experiment shows that many similar functions can be detected from tens of millions of lines of source code. This study makes it easier to find frequent functionalities.
Key words　Program Analysis, Code Clone, Function Classification, Code Reuse

1. まえがき

近年，ソフトウェアシステムの大規模化，複雑化に伴い，ソフトウェア開発に要するコストが増加してきている．ソフトウェア開発コストの削減方法として，これまでに作成したソフトウェアやオープンソースソフトウェアなど，既存資源の再利用が挙げられる．しかし，再利用を行うためには内容を理解しなければならず，特に，再利用を考慮して作成されていない資源の場合，その労力は大きい．

ソフトウェアの保守作業を困難にする要因の 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，ソースコード上に存在する同一もしくは類似したコード片のことであり，主にコピーアンドペースト等によって発生するといわれている [1]．コピーアンドペーストを行った際，コピー先に合わせた識別子（変数名や関数名）の変更や，文単位での追加および削除などの修正作業がしばしば行われる．この作業により，これまでに作成したソフトウェアに同様の機能を実装した関数が複数作りこまれる．

本研究では，コードクローン情報を用いて，これまでに実装された機能を効率的に把握・再利用するため，大量のソースコードから同様の機能を実現した関数群を検出する手法を提案する．コードクローンは類似したコード片であるので，その処理内容も類似している．つまり，既存のソースコードからコードクローンを検出，クローンセットを生成することによって類似した処理集合を得ることができる．クローンセットは互いに類似したコード片の集合を表し，関数をコード片の単位として検出を行うと，クローンセットは同様の機能を実現した関数集合と考えることができる．

しかし，既存のクローン検出ツールを用いてコードクローンを検出する場合，対象にできるソースコード量に制限がある．大量のソースコードを対象としたクローンペアの検出手法 [2] はこれまでに提案されているが，クローンセットの生成を行う手法は提案されていない．この問題に対応するため大量のソースコードを複数のグループに分割し，それぞれのグループにつ

いて関数単位のクローン (関数クローン) を検出，その結果を組合わせることによって関数単位のクローンセット (関数クローンセット) の生成を行った．また，複数の計算機で並列計算することにより，検出時間の短縮を行った．

実験の結果，2,700 万行のソースコードから約 15 時間で関数クローンセットを生成することができ，どのような関数が多数利用されているのか容易に把握することができた．また，類似した関数の数，関数が存在するソフトウェア数などを基にしたメトリクスを利用して関数クローンセットを分類することにより，ソースファイル単位のクローンも発見することができた．本手法を利用して散在する類似関数の整理を行うことにより，ソースコードの内容理解やソフトウェアの保守性の向上支援が期待できる．

2. コードクローン

2.1 概　　要

コードクローン [1] とは，ソースコード中の同一，あるいは，類似したコードの断片を意味する．コードクローンは，コピーアンドペーストによるプログラミングや意図的に同一処理を繰り返して書くことにより，ソースコード中に作りこまれる．互いに類似する 2 つのコード片をクローンペア，互いに類似するコード片の集合をクローンセットと呼ぶ．

2.2 クローン検出ツール：CCFinder

CCFinder [3] は，単一または複数のファイルのソースコード中から全ての極大クローン検出し，それをクローンペアの位置情報として出力する．ある 2 つの字句列 (a, b) がクローンペアで，それぞれの字句列を真に包含する如何なる字句列も等価でないとき，(a, b) を極大クローンと呼ぶ．CCFinder は様々な特徴を持つが，本手法で特に重要な，ある程度の違いは吸収可能である特徴について説明する．コピーアンドペーストし，少々コードに変更を加えたとしても処理内容が変化しない場合がある．このような場合でもコードクローンとして抽出できれば，同様の処理部分を検出できる．下記にどのような違いを吸収可能であるかについて示す．

- ソースコード中に含まれるユーザ定義名，定数をパラメータ化することで，その違いを吸収できる．
- クラススコープや名前空間による複雑な名前の正規化を行うことで，その違いを吸収できる．
- その他，テーブル初期化コード，可視性キーワード（protected，public，private 等），コンパウンド・ブロックの中括弧表記等の違いも吸収できる．

以降，識別子をパラメータ化せずにコードクローンを検出することを完全一致検出，識別子をパラメータ化してコードクローンを検出することを名前変更検出と呼ぶ．完全一致検出は字句列が等価であっても，識別子が異なる場合，コードクローンであると判定されない．そのため，コピーアンドペーストされた後に識別子が変更されたものはコードクローンとして検出されなくなる．名前変更検出は識別子が異なっていてもコードクローンであると判定されるので，コピーアンドペーストされた後に識別子の変更があってもコードクローンとして検出できる．

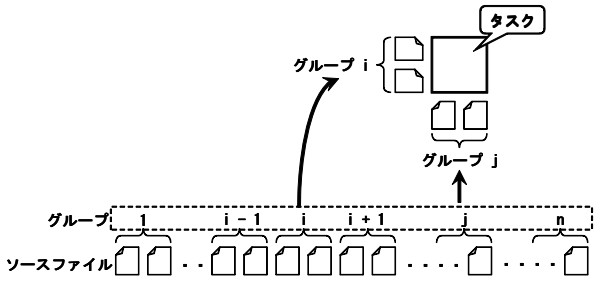

図 1　計算モデル

3. 関数クローンペア

3.1 定　　義

2 つの関数間で字句列が閾値以上等価であるとき，互いに関数クローンペアと定義する．関数 A，関数 B が式 (1)(2) を満たす場合，互いに関数クローンペアになる．

$$DUP(A,B) = \frac{CL(A,B)}{LOC(A)} \tag{1}$$

$$min(DUP(A,B), DUP(B,A)) \geqq THRESHOLD \tag{2}$$

$DUP(A,B)$:　関数 A の関数 B に対する重複度
$CL(A,B)$:　関数 A において関数 B とコードクローンになっている行数
$LOC(A)$:　関数 A の行数
$THRESHOLD$:　関数重複度の閾値

3.2 検 出 手 順

検出手順は以下の 3 つのステップからなる．

Step 1 : 関数情報の取得

関数クローンペアを検出するために，ソースコードから関数の位置や行数などの情報を抽出する必要がある．本研究では Ctags [4] を利用して，関数名と開始行，ソースファイル名を取得し，終了行を中括弧をカウントすることによって取得する．

Step 2 : コードクローンの検出

これまでに様々なコードクローン検出手法が提案されている．今回は大量のソースコードからコードクローンを検出するので，適用対象が大きくなる．そのためスケーラビリティの高い CCFinder を用いる．

Step 3 : 関数クローンペアの検出

関数情報，コードクローン情報を式 (1)(2) に適応し，閾値を満たす関数のペアを関数クローンペアとして検出する．

3.3 並列計算を用いた実装

本研究では大量のソースコードを検出対象とするため，一度に対象全体からコードクローン検出を行うことは不可能である．このため，検出対象をソースファイル群の小さなグループに分割し，グループの組合せ (タスク) 単位で関数クローンを検出する (図 1)．検出対象のソースファイル群を n 個に分割しているとする．このとき，任意のタスクは (i, j) で表すことができる（ただし，$1 \leqq i, j \leqq n$）．タスク (i, j) に含まれるコードクローンはタスク (j, i) に含まれるコードクローンと同様であるため，後者については検出を行わない．これにより，関数クローンを

検出するタスク数は $n(n+1)/2$ となる．

本手法は，関数クローンペアの検出を複数のコンピュータで並列に実行することにより，計算時間の短縮を行う．各タスクの演算（関数クローンペア検出）は他のタスクの演算結果に全く依存しないため，タスクの割り当て処理は単純に行える．関数クローンペアが未検出のタスクを，アイドル状態のコンピュータに割り当て，検出結果を回収するだけでよい．

演算はタスク管理用計算機 1 台，ソースファイル保存用計算機 1 台，関数クローンペア検出用計算機複数台で行う．それぞれの計算機の役割を下記に示す．

タスク管理用計算機： 計算タスクを管理し，アイドル状態の関数クローンペア検出用計算機にタスクを割り当てる．また，関数クローン検出用計算機から演算結果を集約する役割も持つ．

ソースファイル保存用計算機： 計算対象のソースファイル群を保持する．関数クローンペア検出用計算機はこの計算機からソースファイルを取得する．

関数クローンペア検出用計算機： タスク管理用計算機から割り当てられたタスクに必要なソースファイルをソースファイル保存用計算機からダウンロードし，3.2 節で述べた手順で演算を行う．関数クローンペアの検出結果をタスク管理用計算機に送信し，次のタスクを受け取る．

4. 関数クローンセット

4.1 定　　義

関数 a_i と関数クローンペアとなる関数の集合を $\{b_1, b_2, b_3, ..., b_n\}$ をする場合，集合 $\{a_i, b_1, b_2, b_3, ..., b_n\}$ を関数 a_i をもとにした関数クローンセットと定義する．この集合は関数 a_i からコピーアンドペーストされた関数の集合と考えることができる．以降，関数 a_i のような関数を基底関数，$b_1, b_2, b_3, ..., b_n$ のような関数を派生関数と呼ぶ．

4.2 生 成 方 法

生成のイメージを図 2 に示す．ノードは関数，エッジは関数クローンペアの関係を表しており，ノードの数値の違いによりユニークな関数を表している．この方法で関数クローンセットを生成した場合，関数クローンセット間で基底関数は異なるが，要素となっている関数が全て一致することがある．例えば，関数 a_i が基底関数となる関数クローンセット $\{a_i, b_1, b_2, b_3, ..., b_n\}$ と関数 b_1 が基底関数となる関数クローンセット $\{a_i, b_1, b_2, b_3, ..., b_n\}$ が存在する場合を考える．この場合，関数 a_i と関数 b_1 が基底関数となる関数クローンセット $\{a_i, b_1, b_2, b_3, ..., b_n\}$ のようにマージする．このようにマージを行うことによって情報量を保ったまま関数クローンセット数を少なくし，機能の類似した関数の把握を容易にする．

4.3 関数クローンセットメトリクス

1 つの関数クローンセットはある 1 つの機能と考えることができる．その機能がどのような特徴を持っているのかメトリクスを用いて数値化する．以下に利用したメトリクスとその定義，意図を述べる．

POP (Population)： 関数クローンセットの要素数を表す．ある関数クローンセットを同様の機能を持った関数の集合であると考えると，POP はその関数がどの程度利用されているのかを表す．

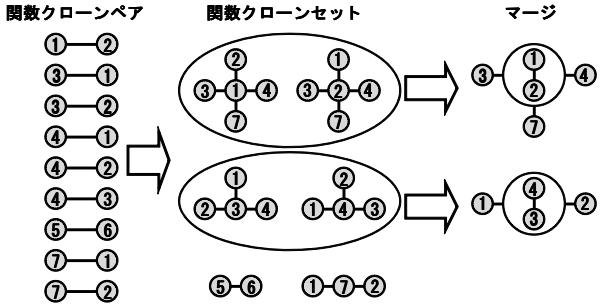

図 2　関数クローンセットの生成方法

LEN (Length)： 関数クローンセットに含まれる関数の平均行数を表す．LEN の値が大きいほどより多くの行数を要する機能であるといえる．

FOF (Frequency Of Functions)： 関数クローンセットに含まれる関数がいくつの異なるソフトウェアに存在するか表す．この値が高いほど複数のソフトウェアで利用されている機能であるといえる．

DOS (Dispersion Of Software)： 関数クローンセットに含まれる関数がどの程度異なるソフトウェアに分散しているか表す．定義式を以下に示す．

$$DOS = \frac{FOF - 1}{POP - 1} \tag{3}$$

同じソフトウェアで何度も利用されている関数の場合は値が低く，そうでない場合は値が高くなる．

GVOF (General Versatility Of Functions)： 関数クローンセットに含まれる関数がいくつの異なるドメインに存在するか表す．ドメインとはソフトウェアの種類を表す．

DOD (Dispersion Of Domains)： 関数クローンセットに含まれる関数がどの程度異なるドメインに分散しているか表す．

$$DOD = \frac{GVOF - 1}{POP - 1} \tag{4}$$

同じドメインで何度も利用されている関数の場合は値が低く，そうでない場合は値が高くなる．

5. 実　　験

5.1 目　　的

提案手法の有効性を確かめるため，実際のソースファイル群への適用実験を行った．実験の目的を下記に示す．

- 検出できる関数クローンと関数クローンペアの閾値，完全一致・名前変更検出との関係を確認
- 関数クローンセットメトリクスと関数クローンセットの関係を確認

5.2 実 験 環 境

3.2.2 節で述べた関数クローンペアを検出する並列計算の環境を下記に示す．

タスク管理用計算機

CPU : Intel(R) Xeon(R) CPU 5150 2.66GHz

メモリ：8.00GB

OS : Windows Vista Business

ソースファイル保存用計算機

CPU : Dual-Core AMD Opteron(tm) Processor 290 2.8GHz

メモリ : 16GB

OS : Solaris10

関数クローンペア検出用計算機 (13 台)

CPU : Dual-Core AMD Opteron(tm) Processor 2210 HE 1.80GHz

メモリ : 6.00GB

OS : Windows Vista Business

関数クローンセットの生成，関数クローンセットメトリクスの計算は関数クローンペアを全て収集した後，タスク管理用計算機で行う．

5.3 実験対象

本稿でのコードクローン検出対象は，オープンソースオペレーティングシステム FreeBSD 用のソフトウェア集合 Ports システムに含まれているソースファイルであり，各ソースファイルは 1 つのプロジェクトに属している．全てのプロジェクトは，bind9，emacs，apache など，一意に特定可能な名前を持っている．共通の特徴を持ったプロジェクトは同じドメインに所属している．例えば，emacs や vim，gedit などは editors ドメインに所属している．

表 1 に各ケースの実験対象規模，表 2 に各ケースで得られた関数クローンペア数，表 3 に各ケースで得られた関数クローンセット数を示す．次節以降で各ケースについて述べる．

5.4 ケース 1：mail ドメイン，完全一致検出

1 つ目のケースでは，mail ドメインを実験対象とし，完全一致検出で関数クローンの検出を行う．関数クローンペアの検出時間は約 13 時間半であった．この実験の目的は，FOF，DOS 値と関数クローンとの関係を調査することである．

表 1　実験対象規模

ケース	ドメイン数	プロジェクト数	総行数
1,2	1	717	27,076,351
3,4	8	1,306	18,339,890

表 2　関数クローンペア数

ケース	THRESHOLD(%)				
	60	70	80	90	100
1	475,022	446,592	421,502	403,447	383,091
2	1,064,941	929,141	819,848	763,538	718,048
3	88,913	80,417	72,405	65,927	56,832
4	407,860	335,802	259,790	221,538	201,565

表 3　関数クローンセット数

ケース	THRESHOLD(%)				
	60	70	80	90	100
1	70,796	69,910	69,024	67,941	67,064
2	72,196	70,971	69,313	67,881	66,455
3	36,980	34,798	32,717	30,468	28,016
4	49,290	45,966	41,818	38,941	36,378

表 2，3 に示す通り，関数クローンペア数，関数クローンセット数共に関数クローンペア閾値が上昇するにつれて減少している．関数クローンペア閾値が高くなるほどより多くの行がコードクローンになっていなければならないので，関数クローンペア数が減ると考えられる．結果はこの予測に従っているといえる．以降のケースにおいても，この傾向に変化はなかった．

次に関数クローンセットメトリクスと関数クローンセット数の関係の結果について述べる. POP 値と関数クローンセット数の関係は，反比例のような関係になっていた. LEN 値は，15 付近で関数クローンセット数が最も多く，LEN 値が上がるにつれて反比例するように減少していた．FOF 値，DOS 値，関数クローンペア閾値 100 %の関数クローンセット数の関係を表 4 に示す．表に示す通り，DOS 値 0.9〜1.0，FOF 値 2〜6 に関数クローンセットが集中していることがわかる．最も関数クローンセット数の多い，DOS 値 0.9〜1.0，FOF 値 4 の関数クローンセットの要素を調べたところ，大部分が lightning，thunderbird，enigmail-thunderbird，enigmail-seamonkey プロジェクト間，続いて postfix，postfix23，postfix24，postfix-current プロジェクト間，続いて mutt，mutt-lite，mutt-devel，mutt-devel-lite プロジェクト間，続いて bogofilter，bogofilter-qdbm，bogofilter-sqlite，bogofilter-tc プロジェクト間の関数クローンセットであった．さらに，これらのクローンはほぼ全て修正されずにソースファイル単位で利用されていた．また，関数クローンペアの閾値を下げた場合においても，関数クローンセット数の増加が小幅であるため，傾向の変化は無かった．

5.5 ケース 2：mail ドメイン，名前変更検出

2 つ目のケースでは，名前変更検出でその他はケース 1 と同条件で行う．関数クローンペアの検出時間は約 14 時 12 分であった．ケース 1 に比べ検出時間を必要としたのは，完全一致検出より名前変更検出のほうがより多くのコードクローンを検出するためである. この実験の目的は，完全一致検出と名前変更検出の結果がどの程度異なるか調査することである．

完全一致検出より名前変更検出のほうがより多くのコードクローンが検出されるので，ケース 2（名前変更検出）のほうがケース 1（完全一致検出）より多くの関数クローンが検出され

表 4　ケース 1:関数クローンセット数の分布

FOF	DOS									
	0.0 〜 0.1	0.1 〜 0.2	0.2 〜 0.3	0.3 〜 0.4	0.4 〜 0.5	0.5 〜 0.6	0.6 〜 0.7	0.7 〜 0.8	0.8 〜 0.9	0.9 〜 1.0
1	766	0	0	0	0	0	0	0	0	0
2	7	45	21	127	149	0	0	0	0	10,884
3	2	23	21	249	6	0	65	0	0	8,481
4	2	78	133	14	1,092	10	0	353	0	41,759
5	1	15	29	68	84	31	8	197	0	844
6	0	4	3	3	36	1	0	0	31	1,237
7	0	0	0	0	0	0	0	0	12	33
8	0	0	0	0	0	0	0	0	0	32
9	0	0	0	0	0	0	0	4	2	9
10〜	0	0	0	0	0	0	5	1	0	87

ると予想できる．実験の結果，予想通りケース 2 のほうがより多くの関数クローンペアが検出された．しかし，関数クローンセット閾値 90 %と 100 %の関数クローンセット数はケース 2 に比べケース 1 のほうが多くなっている．これはケース 1 では識別子が異なっていたために別の関数クローンセットになっていたものが字句をパラメータ化することによって 1 つの関数クローンセットになったことが原因である．

次に関数クローンセットメトリクスと関数クローンセット数の関係の結果について述べる．POP，LEN 値と関数クローンセット数との関係はケース 1 と同様であった. FOF 値，DOS 値，関数クローンペア閾値 100 %の関数クローンセット数の関係を表 5 に示す．大まかな傾向はケース 1 と同じであるが，若干の違いが見られる．ケース 1 に比べ，DOS 値 0.9〜1.0，FOF 値 3〜6 付近の関数クローンセット数が減少し，DOS 値 0.9 未満の関数クローンセット数の増加が確認できる．これは，名前変更検出により関数クローンペアがより多く検出され，関数クローンセットの POP 値が増加し，DOS 値が減少したためであると考えられる．

5.6 ケース 3：複数ドメイン，完全一致検出

3 つ目のケースでは accessibility，archivers，benchmarks，converters，finance，ftp，irc，misc の 8 ドメインを実験対象とし，完全一致検出で関数クローンの検出を行う．関数クローンペアの検出時間は約 6 時間半であった．この実験の目的は，GVOF，DOD 値と関数クローンとの関係を調査することである．

関数クローンセット数に関して，同ドメイン内の関数クローンセットの場合に比べ，減少する割合が大きいことがわかる．これは，ドメイン間の関数クローンのほうがドメイン内の関数クローンに比べ，コードの変更が多いことを表している．

次に関数クローンセットメトリクスと関数クローンセット数の関係の結果について述べる．POP 値と関数クローンセット数との関係はケース 1，2 と同様であった．LEN 値は 10〜40 付近で最も関数クローンセット数が多く，LEN 値が上がるにつれ関数クローンセット数が減少していた．GVOF 値は 1 で最も関数クローンセット数が多く，2 になると急激に関数クローンセット数が減少し，GVOF 値が上がるにつれ関数クローンセット数が減少していた．DOD 値は 0 の関数クローンセットがほとんどでその他の DOD 値の関数クローンセットは非常に少なかった．これは，ドメイン間の関数クローンが非常に少ないことを表している．

表 5　ケース 2:関数クローンセット数の分布

FOF	DOS									
	0.0 〜 0.1	0.1 〜 0.2	0.2 〜 0.3	0.3 〜 0.4	0.4 〜 0.5	0.5 〜 0.6	0.6 〜 0.7	0.7 〜 0.8	0.8 〜 0.9	0.9 〜 1.0
1	3,405	0	0	0	0	0	0	0	0	0
2	40	108	36	307	252	0	0	0	0	9,942
3	38	57	66	411	34	0	148	0	0	7,643
4	164	510	537	263	2,661	19	0	920	0	35,891
5	7	91	58	90	88	40	11	199	0	812
6	1	16	22	17	36	3	2	0	15	53
7	0	13	4	1	8	3	2	0	15	53
8	0	4	5	0	4	0	0	1	1	30
9	0	1	1	0	0	0	1	4	5	17
10〜	0	0	2	0	1	1	10	13	4	79

5.7 ケース 4：複数ドメイン，名前変更検出

4 つ目のケースでは名前変更検出でその他はケース 3 と同条件で行う．関数クローンペアの検出時間は約 14 時間 20 分であった．この実験の目的は，完全一致検出と名前変更検出の結果がどの程度異なるか調査することである．

完全一致検出と名前変更検出の違いについては，関数クローンペア数は名前変更検出のほうが多く，同ドメイン内の関数クローンの場合と同様であったが，関数クローンセット数は同ドメイン内とは異なり，常に名前変更検出のほうが多かった．これは，ドメイン間の関数クローンはドメイン内の関数クローンに比べて様々なものが存在し，名前変更検出でより多くの関数クローンペアを検出したが，類似した関数が少なかったためだと考えられる．

次に関数クローンセットメトリクスと関数クローンセット数の関係の結果について述べる．POP 値と関数クローンセット数との関係はケース 1，2，3 と同様であった．LEN 値はケース 3 とは異なり，LEN 値 4 で最も関数クローンセット数が多くなっており，名前変更検出では，4 行程度の非常に小さな関数クローンが多く検出されていた．GVOF，DOD の傾向はケース 3 と同様であった．

6. 考　　察

6.1 ケース 1，2

ケース 1，2 では mail ドメインを対象に実験を行った．表 6 に示す通り，関数クローンペアの閾値が増加するにつれて関数クローンペアの数が減少しているため，関数クローンとなる関数も減少している．また，コピーアンドペーストされた後に識別子が変更された関数も検出しているため，名前変更検出のほうがより多くの関数クローンを検出している．さらに，閾値 60 %以上の関数クローンのほとんどが閾値 100 %の関数となっている．mail ドメインには 617,713 個の関数が存在するので，名前変更検出で関数クローンペア閾値 100 %の関数クローンセット群は，617,713 個の関数から 253,455 個の関数を選択，66,455 種類に分類したものと考えることができる．次にこの関数クローンセット群から再利用性の高い関数を抽出することを目指す．

ここでは再利用性の高い関数を多くのプロジェクトで利用されている関数であるとする．実験の結果から，DOD 上位の関数クローンセットは完全一致検出より名前変更検出のほうが少

表 6　関数クローンセットに含まれるユニークな関数の総数

ケース	THRESHOLD(%)				
	60	70	80	90	100
1	255,050	252,275	249,466	246,275	242,868
2	269,463	265,787	261,757	257,541	253,455
3	83,070	78,311	73,912	69,443	64,122
4	113,387	107,011	99,980	94,238	88,918

```
File:hashcash/workhashcash-1.22/getopt.c

static char *
my_index (str, chr)
      const char *str;
      int chr;
{
      while (* str)
            {
                  if (*str == chr)
                        return (char *) str;
                  str++;
             }
      return 0;
}
```

図 3　ケース 2:フィルタリング後の関数クローン例

なかった．これは名前変更検出のほうが多くの関数クローンが得られるため，POP 値が大きくなってしまうことが原因であった．しかし，FOF 値に関しては増加することはあっても，減少することはない．よって，FOF 値を基準とし，3 以上も関数クローンセットを選択する．また，完全一致検出に比べ名前変更検出のほうが識別子の変更に対応することができ，類似した関数をまとめるのに効果的であるので，関数クローンペア閾値が 100 %の関数クローンセット群から選択を行う．表 7 に関数クローンペアの閾値 100 %の関数クローンセット群から FOF 値別にフィルタリングした関数クローンセット数，ユニークな関数の総数を示す．例えば，FOF 値 8 以上でフィルタリングした場合，617,713 個の関数から 184 種類 1,909 個の関数を得ることができる．関数クローンの例として，文字列において指定した文字へのポインタを返す関数などが得られた（図 3）．このように FOF，DOS を利用することによって，ドメイン内の再利用性の高い関数を得ることができた．

6.2　ケース 3，4

ケース 3，4 は複数のドメインを対象に実験を行った．ケース 1，2 同様，関数クローンペア閾値が増加するにつれて関数クローン数が減少しているが，その割合が大きいことがわかる．これは，ドメイン間の関数クローンはコードが変更されているものが多く，関数クローンペア閾値の影響が大きかったためだと考えられる．完全一致検出と名前変更検出の差が大きいこともこれが理由であると考えられる．以上のことから，ドメイン間の関数クローンは完全一致クローンのほうがより安全に類似した関数を検出できると考える．次に完全一致検出で関数クローンペア閾値 100 %の関数クローンセット群から再利用性の高い関数を抽出することを目指す．

表 7　ケース 2:FOF 値の閾値別関数クローンセット，関数の総数

FOF 閾値	3	4	5	6	7	8
関数クローンセット数	52,365	43,968	3,003	1,607	283	184
関数の数	222,635	197,077	17,812	10,591	2,520	1,909

表 8　ケース 3:FOF 値の閾値別関数クローンセット，関数の総数

FOF 閾値	3	4	5	6	7	8
関数クローンセット数	1,862	423	304	206	141	77
関数の数	6,591	2,422	2,007	1,499	1,162	726

ここでは再利用性の高い関数を多くのドメイン・プロジェクトで利用されている関数とする．実験の結果から GVOF を利用することにより再利用性の高い関数を得ることができると考えられる．名前変更検出で関数クローンペア閾値 100 %の関数クローンセット 28,016 個から GVOF が 2 以上のものを選択すると 2,863 個の関数クローンセットが得られた．この関数クローンセット群を FOF 値でフィルタリングすることにより再利用性の高い関数を得る．表 8 に FOF 別にフィルタリングした関数クローンセット数，ユニークな関数の数を示す．例えば，FOF 値 8 以上でフィルタリングした場合，421,008 個の関数から 77 種類 726 個の関数を得ることができる．関数クローンの例として，メモリ領域を解放する関数などが得られた．このようにドメイン間の再利用性の高い関数は GVOF，FOF を利用することによって得ることができた．

7. あとがき

本研究では，大量のソフトウェアから同様の機能を実装した関数群を検出する手法を提案した．まず，対象のソースコード群を複数のタスクに分割し，複数のマシンで並列に関数クローンペアを検出する．そして，その結果を一台の計算機に収集して，ある関数と関数クローンペアとなる集合を関数クローンセットとして検出した．さらに，関数クローンセットメトリクスを用いたフィルタリングを行った．

本研究では，クローン検出ツールに CCFinder を用いた．CCFinder はスケーラビリティ，再現率が高いが，誤検出が多く，適合率が低いというデメリットがある．計算の工夫をすることにより，他のツールを用いても高速で，適合率の高い関数クローンを検出できる可能性がある．クローン検出ツール別に比較を行うことで興味深い結果が得られるであろう．

謝　　辞

本研究は，一部，文部科学省「次世代 IT 基盤構築のための研究開発」（研究開発領域名：ソフトウェア構築状況の可視化記述の開発普及）の委託に基づいて行われている．また，文部科学省科学研究費補助金基盤研究 (C)(課題番号:20500033) の助成を得て行われている．

文　　献

[1] 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 D, vol.91-D, no.6, pp.1465-1481, 2008.

[2] S.Livieri, Y.Higo, M.Matushita, and K.Inoue. Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder. Proceedings of the 29th International Conference on Software Engineering, pp.106-115, 2007.

[3] T.Kamiya, S.Kusumoto, and K.Inoue. CCFinder : A multi-linguistic token-based code clone detection system for largescale source code. IEEE Transactions on Software Engineering, Vol. 28, No. 7, pp. 654-670, 2002.

[4] Ctags. http://ctags.sourceforge.net/.