特別研究報告

題目

類似バグの抽出を目的としたコード片検索結果の分類手法

指導教員
井上 克郎 教授

報告者
向井　康裕

令和 2 年 2 月 10 日

大阪大学 基礎工学部 情報科学科

令和元年度 特別研究報告

類似バグの抽出を目的としたコード片検索結果の分類手法

向井　康裕

内容梗概

ソフトウェア開発の工程において，開発者はすでに存在する他のソフトウェアからソースコードを再利用することがある．これにより，開発コストの削減や信頼性の高い機能の実装が可能となっている．しかし，再利用元にバグが含まれていたとしても気づかずに再利用している場合がある．このよう類似コードを検索するために，類似コード片検索ツールが数多く開発されている．その中の 1 つである NCDSearch というツールは，クエリとして与えたコード片に類似するコード片を高速に漏れなく検出することが可能である．これにより，開発者は類似バグを含むコード片の位置を迅速に把握することができる．しかし，類似バグを含むコード片の見逃しをなくそうとするあまり，数多くのバグを含まない類似コード片が検出される問題がある．

本研究ではクラスタリングに基づく分類手法を用いて出力結果を分類することで，人間が類似バグを含む可能性のあるコード片を確認する労力を減らすことを目的とする．クラスタリングについては，ディレクトリ単位での分類と階層型クラスタリングである最短距離法と最長距離法による分類を行った．さらに，バグを含む確率が高いコード片のみを出力するためのフィルタを分類後のコード片群に適用した．このフィルタにより，確認する必要のないコード片を除外することができる．また，フィルタにはパラメータが設定されており，各手法において最適な表示数の検証を行った．

評価実験として，3 つの提案手法を評価するために類似バグを含むデータセットを用いて NCDSearch との比較実験を行った．その結果，提案手法を用いることで高い再現率を維持したまま開発者が出力結果を確認する労力を削減できたことを確認した．

主な用語

ソースコード検索
クラスタリング
正規圧縮距離

目 次

1　まえがき

大規模なソフトウェア開発において，開発者はすでに存在する他のソフトウェアからソースコードを再利用することがある．これにより，開発コストの削減や信頼性の高い機能の実装が可能となっている．しかし，ソースコードの再利用元のコード片にバグがあった場合，再利用先においてもコード片に同様の修正を行う必要がある．しかし，開発者がすべての再利用を把握しておくことは難しい．この問題を解決するために，ソフトウェア開発において類似コード片検索手法がいくつか提案されている．開発者は類似コード片検索手法を用いることであるコード片に類似したコード片や再利用されたコード片を把握でき，複数の場所に潜んでいる同じバグの同時修正を低コストで行うことが可能となる．

類似コード片検索手法の 1 つとして，Ishio らが開発した NCDSearch[1] が存在する．検索クエリと調査対象のソースコードを NCDSearch に与えると，NCDSearch は調査対象のソースコードから検索クエリに類似したコード片を抽出し，類似度順に並べて表示する．NCDSearch は 21 億行のソースコードから検索クエリと類似したコード片を検索する作業を 8 分と高速に行うことができ，検索結果にはバグを含むコード片の検出漏れがない．しかし，その代わりに検索クエリに類似したバグを含まないコード片を多数誤検出してしまう．Ishio らの実験によると，NCDSearch を用いた 8107 件の検索結果において Precision は 0.010 となっており，検索結果から 1 個のバグを含む片を探すのに 100 個のコード片を調査することになる．実際の業務で使用することを考えた場合，エラーコード片がいくつあるかはわからないため，1 つのバグを含むコード片を修正するためにその 100 倍のコード片を調査するのは現実的でない．このため，調査対象を絞るために検索クエリと類似度の高い順にコード片にランキングを付け，順位の高いコード片のみ確認する手法が考えられる．しかし，ランキングの何番目まで確認すれば十分であるという指標を立てることは難しい．

本研究では NCDSearch の検索結果をバグを含む可能性が高いものとそうでないものにクラスタリングを用いて分類し，人間が確認する労力を削減する手法を提案する．分類したクラスタについて，バグを含む可能性のあるものは中身を詳しく確認するが，そうでないクラスタは検索クエリと最も類似するコード片のみ確認する．クラスタリングはバグを含むコード片，含まないコード片同士は類似した属性を持つという仮定のもと，類似した属性を持つコード片同士をまとめるものである．今回の研究では，コード片の持つ 2 つの属性に着目し，3 つのクラスタリング手法を適用した．1 つ目の属性はコード片同士の正規圧縮距離であり，この属性を用いたクラスタリング手法として最短距離法と最長距離法の 2 つのクラスタリングを使用した．この 2 つのクラスタリング手法はどちらも凝集型階層クラスタリングの 1 つである．2 つめの属性はコード片が存在するファイルのディレクトリパスであり，3 つ目のクラスタリング手法はこのディレクトリパスごとにコード片を分類する．これら 3 つの中で

どの手法が最も有効かを仕事の削減率や再現率をもとに調査した．

以降，2 章では本研究の背景として，NCDSearch の説明を行う．3 章では提案手法について詳しく説明する．4 章ではデータセットを用い，提案手法を評価した結果を述べる．5 章では本研究でのまとめと今後の課題を示す．

2 背景

2.1 類似コード片検索ツール：NCDSearch

NCDSearch[1] とは，バグ修正の水平展開向けの類似コード片検索ツールである．バグのあったソースコードの断片を入力（クエリ）とし，類似度が高いソースコード片から順に出力する．NCDSearch の特徴として，類似性の判定に正規圧縮距離 [2] を用いている点があり，変数名の変更や並び替えの変化に強く，多言語への対応も容易である．また，ファイル単位でのフィルタリングを用いているという特徴がある．これにより，他のツールと比較して高速であり，高い再現性を実現することが可能となっている．

現段階での NCDsearch の問題点として Precision(適合率) の低さが挙げられる．そのため実務で NCDSearch を使用して類似バグを含むコード片の検索に大きな労力が必要となる．例えば，1 個のバグを含むコード片を探すのに 100 個の類似コード片を調査することになる．表 1 に NCDSearch より検出される類似コード片群の様子を示す．このように，NCDSearch より出力される類似コード片群はすべてクエリとの類似度が高いものであるため，どのコード片がバグを含むコード片であるか即座に判断するのは難しい．また，類似度が高い順に調査する場合，どのコード片まで調査すればいいのかについては不明である．

2.1.1 出力される情報

図 1 にクエリと NCDSearch より出力される類似コード片群の一部を示す．NCDSearch より出力される類似コード片の情報には，ファイル名，クエリとの距離，該当箇所の開始行番号もしくは終了行番号，トークン文字列がある．

2.1.2 正規圧縮距離

このツールで 2 つのコード片の比較を行うとき，正規圧縮距離を用いる．正規圧縮距離は以下の式で定義される．

$$NCD(q,s) = \frac{C(qs) - min(C(q), C(s))}{max(C(q), C(S))} \tag{1}$$

データ圧縮アルゴリズムを適用するために, トークン列 q と s を連結してバイト列に変換する. 圧縮アルゴリズムには zip と gzip の最も一般的なアルゴリズムである zlib の Deflate アルゴリズムを使用している. そのため, 使用者は自分の環境でアルゴリズムのプロパティを簡単に分析できる.

表 1: NCDSearch より検出される類似コード片群の一部

距離	検出したコード片
0.0	”scan=heap_beginscan(rel,SnapshotAny,0,NULL)”
0.105	”scan=heap_beginscan(rel,SnapshotNow,0,NULL)”
0.175	”scan=heap_beginscan(lRel,SnapshotNow,0,NULL)”
0.195	”scan=heap_beginscan(testrel,SnapshotNow,0,NULL)”
0.289	”scan=heap_beginscan(rel,SnapshotNow”
0.392	”scandesc=heap_beginscan(cstate->rel,ActiveSnapshot,0,NULL)”
0.44	”scanDesc=heap_beginscan(event_relation,SnapshotNow,0,NULL)”
0.44	”scan=heap_beginscan(pgclass,SnapshotNow,1”
	検出したバグを含むコード片
0.318	”scan=heap_beginscan(heapRelation,snapshot,0,NULL)”

query.json


```
.
.
"3": {
  "query": "3-1.c",
  "queryloc": { "path": "postgres-
dcb09b5","file": "src/pl/plperl/plperl.c", "sline":
2132, "eline": 2133 },
  "lang": "c",
  "type": 3,
  "path": "postgres-dcb09b5",
  "answers": [
  { "path": "postgres-dcb09b5",
    "file": "src/pl/plperl/plperl.c",
    "sline": 2088,
    "eline": 2088 },
  { "path": "postgres-dcb09b5",
    "file": "src/pl/plperl/plperl.c",
    "sline": 2720,
    "eline": 2720 } ]
},
.
.
```


result-3.json


```
.
.
{
  "fileName" : "C:¥¥Users¥¥y-mukai¥¥lab-
data¥¥src¥¥dcb09b5¥¥src¥¥pl¥¥plpython¥¥plp
ython.c",
  "startLine" : 1855,
  "endLine" : 1855,
  "distance" : 0.3469387755102041,
  "startChar" : 2,
  "endChar" : 41,
  "clusterID" : 2,
  "tokens" : "perm_fmgr_info(typeStruct-
>typoutput,&"
}, {
  "fileName" : "C:¥¥Users¥¥y-mukai¥¥lab-
data¥¥src¥¥dcb09b5¥¥src¥¥pl¥¥plperl¥¥plperl
.c",
  "startLine" : 2088,
  "endLine" : 2088,
  "distance" : 0.36538461538461536,
  "startChar" : 4,
  "endChar" : 50,
  "clusterID" : 3,
  "tokens" : "perm_fmgr_info(typeStruct-
>typinput,&(prodesc"
},
.
.
```


図 1: NCDSearch より出力される類似コード片の例

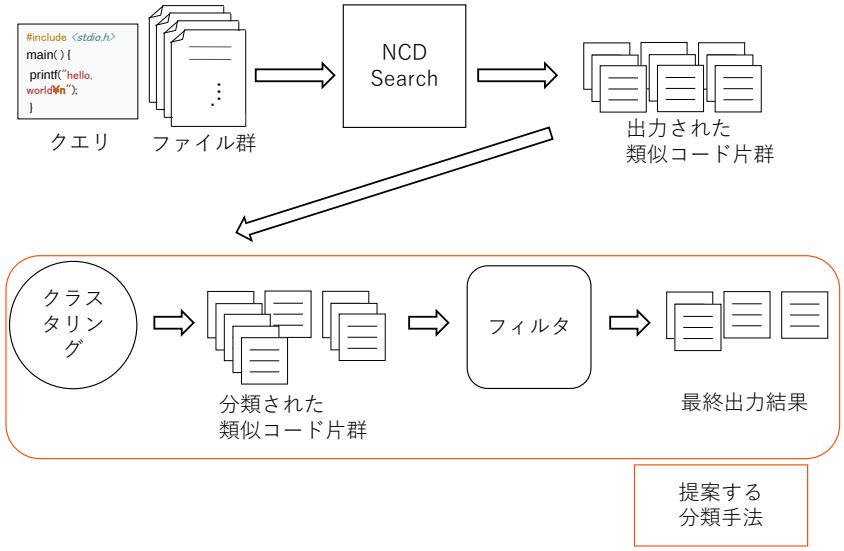

図 2: 提案手法の概要図

3 提案手法

本研究は，クラスタリングを用いたフィルタによって NCDSearch によるコード片検索結果の分類を行い，類似バグを含むコード片の抽出を行うことを目的とする．NCDSearch を使用する場合，検索クエリとの距離が小さいものから順に類似コード片の調査を行うが，その際に確認する必要ない類似コード片も調査しなければならない．そこでクラスタリングとフィルタを用いることによりバグを含まない類似コード片同士をまとめてクラスタにすることで極力，調査する必要のないものは表示させないようにしたいと考えた．

図 2 に提案手法の概要図を示す．まず，NCDSearch に検索クエリを与えて出力された類似コード片群に対して類似する属性を持つコード片同士を分類するためにクラスタリングを行う．次にクラスタリングによって分類された類似コード片群に対し，バグを含むコード片が存在する可能性高い場合は中身を詳しく確認し，そうでない場合は詳しく確認しないフィルタリングを適用することで，最終出力結果を得る．出力として，使用したデータセットより提示される類似コード片数，類似バグを含むコード片数のそれぞれの合計が与えられる．

3.1 クラスタリング

クラスタリングでは，NCDSearch の結果の分類を行う．これを用いることでバグを含む可能性が高いものとそうでないものに分類することができる．

query.json


```
"1": {
        "query": "1-1.c",
        "queryloc": { "path": "postgres-2618fcd", "file":
        "src/bin/pg_dump/pg_dump.c", "sline": 2671, "eline": 2675 },
        "lang": "c",
        "type": 1,
        "path": "postgres-87d96ed",
        "answers": [
         { "path": "postgres-87d96ed",
           "file": "src/bin/pg_dump/pg_dump.c", "sline": 2672,
        "eline": 2676 }
        ]
}
                .
                .
```


result-1.json


```
[ {
  "fileName" : "C:¥¥Users¥¥y-mukai¥¥lab-data¥¥src¥¥postgres-
87d96ed¥¥src¥¥bin¥¥pg_dump¥¥pg_dump.c",
  "distance" : 0.0,
  "startLine" : 2672,
  "endLine" : 2676,
  "startChar" : 4,
  "tokens" :
"sprintf(q,¥"CREATE%sINDEX%son%susing%s(¥",(strcmp(indinfo[i
].indisunique,¥"t¥")==0)?¥"UNIQUE¥":¥"¥",fmtId(indinfo[i].indexre
lname),fmtId(indinfo[i].indrelname),indinfo[i].indamname)",
  "endChar" : 27,
  "clusterID" : 1
} ]
```


図 3: ディレクトリ名を調査する様子（左：クエリ，右：類似コード片）

3.1.1 ディレクトリに基づくクラスタリング

類似エラーコード片は単一のディレクトリに集中するという仮定のもと，クエリと同じディレクトリに含まれているかどうかで類似コード片を分類する．クエリと類似コード片が持つディレクトリパスの情報をもとにクエリと同じディレクトリパスを持つものと，そうでないものの 2 つのクラスタに類似コード片を分けることができる．図 3 にクエリと類似コード片群でそれぞれディレクトリ名に該当する部分を示す．クエリを含むディレクトリとその親ディレクトリや子ディレクトリの関係にあるディレクトリは異なるものとする．

3.1.2 階層型クラスタリング（最短距離法，最長距離法）

階層型クラスタリングを用いることで，類似コード片間の距離が近いもの同士を同じクラスタに，遠いものを別のクラスタにまとめる．この手法により，バグを含まないクラスタを作ることで効率的な類似バグを含むコード片の抽出ができると考えられる．今回，各クラスタに分類する手法として階層型クラスタリングである最短距離法と最長距離法を用いた．最短距離法とは，2 つのクラスタの要素間の距離のうち最小のものをクラスタ間の距離とし，各クラスタ組で一番小さい距離を成す 2 つのクラスタを統合する手法である．最長距離法とは，2 つのクラスタの要素間の距離のうち最大のものをクラスタ間の距離とし，各クラスタ組で一番小さい距離を成す 2 つのクラスタを統合する手法である [3]．階層型クラスタリングは初期状態で各要素がそれぞれクラスタを作り，終了条件を満たすまで先ほど述べた統合処理を繰り返す．提案手法では，上記のアルゴリズムに従って NCDSearch により出力された類似コード片群を入力として，距離尺度に正規圧縮距離を用いてクラスタリングを行った．出力として各コード片にクラスタの所属情報 (clusterID) が追加された類似コード片群が得られる．

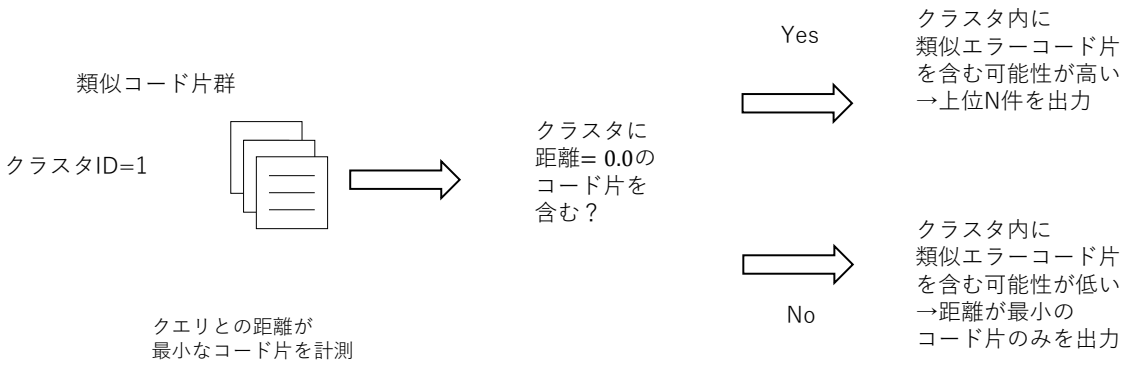

図 4: 階層型クラスタリングを用いたときのフィルタリングの様子

3.2 フィルタリング

各クラスタについて，クエリ，もしくはクエリとの距離が 0.0 であるコード片を含むかどうかを調査し，含んでいればコード片を詳しく調査し，含んでいなければ詳しく調査しない．この操作により，最終的にバグを含む可能性が高いコード片のみを出力することができる．これは各クラスタ内にクエリやクエリとの距離が 0.0 であるコード片を含む場合，同クラスタ内に類似バグを含むコード片が固まっているという仮定に基づいている．各手法により調査する基準が異なる．図 4 に階層型クラスタリングを用いたときのフィルタリングの様子を示す．

ディレクトリに基づくクラスタリングを用いる場合，各クラスタ内にクエリそのものを含むかどうかに基づいてフィルタリングする．クラスタ内にクエリそのものを含む場合，同クラスタ内の類似コード片をクエリとの距離が小さい順に上位 N 件を出力する．クラスタ内にクエリそのものを含まない場合，同クラスタ内の類似コード片のうちクエリとの距離が最短なコード片のみを出力する．

階層型クラスタリングを用いる場合，クエリとの距離が 0.0 であるコード片を含むかどうかに基づいてフィルタリングする．各クラスタ内にある類似コード片群のうちクエリとの距離が最小なコード片を確認し，そのコード片の距離が 0.0 である場合，同クラスタ内の類似コード片をクエリとの距離が小さい順に上位 N 件を出力する．そのコード片の距離が 0.0 でない場合，同クラスタ内の類似コード片のうちクエリとの距離が最短なコード片のみを出力する．

また例として，クラスタ内のコード片のうち距離が小さい順に上位 3 件まで確認する様子を図 5 に示す．この場合だと，クラスタ内のコード片のうちクエリとの最小距離に 0.0 を含むので，クエリとの距離が小さい順に上位 N 件 (図 5 では 3 件) のコード片を出力し，それ以降のコード片は調査しない．

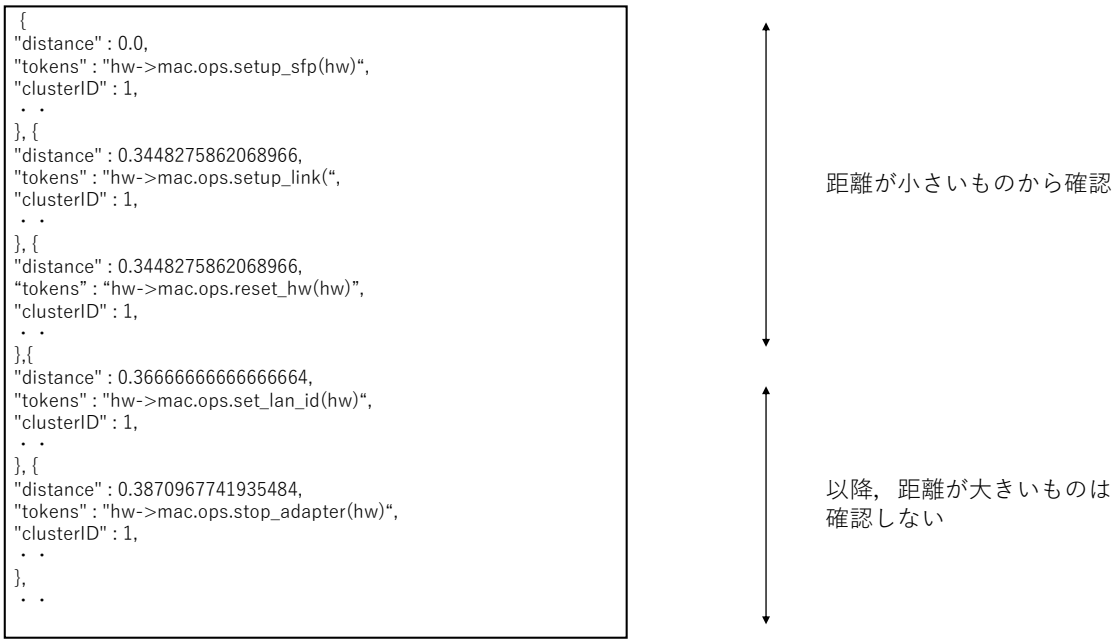

図 5: 例:確認するファイルを選択する様子 (上位 3 件を確認する場合)

4　評価

本研究では，提案手法の評価のため，以下のリサーチクエスチョンを設定した．

RQ1. 提案手法はどの程度有効か?また，3 種類の分類手法のうちどの手法が最も有効か?
4.1.1 節で述べる評価指標によりどの提案手法が最も有効かをパラメータを変化させながら調査する．

RQ2. パラメータの変化が提案手法に与える影響は?
各提案手法について，パラメータを変化させることで現れた特徴を 4.1.1 節で述べる評価指標に基づいて評価する．

RQ3. 分類手法適用後の類似コード片の総数は適用前と比べてどうか?
NCDSearch の出力結果における各データセットに含まれる類似コード片の総数は，今回提案する分類手法適用後の類似コード片の総数と比べるとどう変化するかを調査する．

RQ4. NCDSearch の出力結果の上位を調査した場合と比較して，提案手法は有効か?
既存研究である NCDSearch の出力結果の距離が小さいコード片から順に指定数だけ調査したものと，3 種類の提案手法を比較するとどうなるかを調査する．

これらのリサーチクエスチョンを考察するため，本研究では 4 つの評価指標と複数のパラメータを用いて実験を行った．本章ではまず実験設定について説明したのちに，各リサーチクエスチョンについて回答する．

4.1　評価設定

4.1.1　評価指標

評価では提案手法の有効性を測るため 4 つの指標を使用した．

Precision: 提案手法適用後に出力される結果のうち類似バグを含むコード片の割合．

Recall: NCDSearch より出力される類似バグを含む全コード片のうち提案手法適用後に出力される結果の割合．

削減率: NCDSearch より出力される全類似コード片の数と比較した提案手法適用後に出力される全コード片の数の削減割合．削減率は以下の式で定義される．

$$\text{削減率} = \frac{(\text{全類似コード片}) - (\text{提案手法が出力するコード片})}{(\text{全類似コード片})} \tag{2}$$

表 2: 本実験で使用したデータセット

Projects	クエリの数	類似バグを含むコード片数	ファイル数の中央値	LOCの中央値
PostgreSQL	14	39	1,058	277,959
Git	5	8	261	67,028
Linux	34	41	22,181	6,931,715
Total	53	88	792,432	241,074,652

調和平均: Recall と削減率の調和平均．調和平均は以下の式で定義される．

$$\text{調和平均} = \frac{2}{\frac{1}{Recall} + \frac{1}{\text{削減率}}} \tag{3}$$

本研究の目的はできるだけバグを含む可能性が高いコード片のみを分類し提示することであり，そのために削減率と Recall ともにバランスよく高い値を出すことが必要である．そこで 2 つの値について両者のバランスが考慮される調和平均を評価指標に採用した．

4.1.2 データセット

本実験では PostgreSQL[4]，Git[5]，Linux[6] という 3 つの OSS プロジェクトを基にした CBCD ツール [7] のベンチマークデータセットを使用した．

表 2 は分析したバージョンのプロジェクトにおける各プロジェクトのクエリ数，バグ数，ファイルのサイズを示す．クエリごとにそれぞれソフトウェアのバージョンは異なるため，ファイル数とソースコード行数は中央値を示している．この表より，ファイル数に対してバグを含むコード片の数が最も多いのは PostgreSQL だということがわかる．プロジェクトの主なプログラミング言語は C/C++である．またクエリには次の特徴がある．

- ほとんどのクエリは数行のソースコード片から成る．行数の中央値は 2 行であり, 最長のクエリは 14 行のコードから成る
- 53 個クエリのうち 42 個がソースコード内に類似コード片をただ 1 つ持ち, 残りのクエリはソースコード内に最大 13 個の複数の類似コード片を持つ.

また本実験では NCDSearch の入力として，データセットとクエリを使用する．データセットを NCDSearch に適用した結果，類似コード片の総数は 2641，そのうち類似バグを含むコード片の総数は 88 であった．

表 3: 3 つの手法を比較した結果

手法	Precision	Recall	削減率	調和平均
最短距離法 (クラスタ数=6，上位 10 件)	0.112	0.852	0.746	0.797
最長距離法 (クラスタ数=5，上位 15 件)	0.116	0.852	0.755	0.801
ディレクトリ単位 (上位 15 件)	0.103	0.795	0.743	0.768

4.1.3 NCDSearch の設定

NCDSearch には Deflate，zstd，xz のアルゴリズムがある．本実験では，既存研究で用いられ，最も高い Recall を示した Deflate を使用する．また，各コード片とクエリとの類似度を測るための距離尺度として LZJD を用いた．距離は 0 から 1 の範囲で表され，2 つのコード片が類似しているほど値は 0 に近づく．既存研究との比較のため，類似度の閾値は既存研究で設定された値と同じく，0.5 とした．

4.1.4 パラメータの設定

各手法において設定したパラメータについて説明する．ディレクトリ単位でクラスタリングする手法では，調査するファイル数を距離が小さい順に調査する件数 N をパラメータとして設定（N=1,5,10,15）した．提案手法では，検索結果が多いものも少ないものも一定数検索する．階層型クラスタリング手法では調査する件数に加え，最終的なクラスタの数もパラメータとして設定した．

4.2 RQ1．提案手法はどの程度有効か?また，3 種類の分類手法のうちどの手法が最も有効か?

表 3 は各提案手法において調和平均が最高の値になったパラメータセットでの Precision，Recall，削減率，調和平均を示す．本研究の目的は効率的な類似バグの抽出である．つまり NCDSearch が出力する類似バグコード片をできるだけ類似バグを含まないコード片を排除した状態にすることが目的である．また，類似バグを含むコード片の検出をするうえで，見落としがあるのは望ましくない．そのために Recall と削減率がともに高い値であることが求められる．今回の提案手法において両者はある程度トレードオフの関係にある．3 手法の調和平均はどれも 0.8 に近い値となった．これは Recall をほぼ 0.8 で保ったまま，NCDSearch が出力する類似コード片の約 80%が削減できたといえる．3 手法の調和平均について比較したところ，ディレクトリ単位の手法と階層クラスタリングの手法を比べると階層クラスタリ

表 4: ディレクトリ単位の手法の結果

調査数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
上位 5 件	531	54	0.102	0.614	0.799	0.694
上位 10 件	614	62	0.101	0.705	0.768	0.735
上位 15 件	679	70	0.103	0.795	0.743	0.768
上位 20 件	733	72	0.098	0.818	0.722	0.767
上位 25 件	778	73	0.094	0.83	0.705	0.762
上位 30 件	815	73	0.09	0.83	0.691	0.754
all	1236	74	0.06	0.841	0.532	0.652

ングのほうが高い値であり，その中でも最長距離法が最も高い値であった．最長距離法で調和平均の最高値が 0.801 であり，そのときのパラメータの設定がクラスタ数=5，上位 15 件であった．実際に削減率=0.755，Recall=0.852 という値を出した．

4.3 RQ2. パラメータの変化が提案手法に与える影響は?

パラメータを変化させることによりわかった各手法における特徴を述べる．表 4 にディレクトリ単位で調査した結果を示す．調和平均について，クエリを含むディレクトリ内の調査数上位 15 件以降減少し続けたので，類似バグを含むコード片数が初めて増加しなくなった上位 30 件までの値を調査した．結果について，上位 15 件まで調査数を大きくするほど検出される類似バグを含むコード片数が大きく増加するので，類似バグを含むコード片はある程度クエリを含むディレクトリ内のうち距離が近い順にみて上位に固まっていることがわかった．調和平均について，調査数による変動が少ないことより，類似バグを含むコード片はある程度クエリを含むディレクトリ内のうち距離が近い順にみて下位にもいくつか存在することがわかる．原因として，大規模なデータセットにおいて類似コード片が多く含まれていた可能性がある．

表 5 に最短距離法を用い，クエリとの距離が 0.0 であるコード片を含むクラスタにあるコード片をクエリとの距離が近い順に上位 5 件調査し，それ以外のクラスタではクエリとの距離が最も近いコード片のみを調査し，クラスタ数を変化させたものを示す．クラスタ数を増加させるほど調査するコード片数が増加するとともに，Recall の値も増加している．クラスタ数を増やすことで，よりバグを含むコード片を検出できていることがわかる．また，クラスタ数=13 以降バグを含むコード片数は増加していないことがわかる．このとき検出し

きれていないバグを含むコード片はクエリとの距離が遠いものであることが考えられる．また，クラスタ数の変化による調和平均の変化はあまり見られなかった．

表 6 に最短距離法を用い，上位 10 件を調査したものを示す．Recall について，表 5 では増加し続けていたが，今回クラスタ数=7 のときに減少していることがわかる．クラスタ数=6 のときバグを含むコード片同士固まっていたが，クラスタ数=7 で各クラスタに分散したことが考えられる．表 7 に最短距離法を用い，上位 15 件を調査したものを示す．表 6 と比較すると調査するコード片数がより多くなるとともに，同クラスタ数において Recall の値もより大きいことがわかる．表 8 に最短距離法を用い，クエリとの距離が 0.0 であるコード片を含むクラスタにあるコード片をすべて検索したものを示す．クラスタ数=15 のとき Recall=0.909 であることからバグを含むコード片を検出しきれていないことがわかる．表 9 に最長距離法を用い，上位 5 件を調査したものを示す．最短距離法のときと同様に，クラスタ数が増加するとともに Recall は増加し，削減率は減少していることがわかる．表 10 に最長距離法を用い，上位 10 件を調査したものを示す．Recall について，クラスタ数=6 のときに減少している．表 6 と比べるとクラスタ数がより少ない段階で減少していることから，最長距離法によるクラスタへのまとまりは最短距離法より弱いとわかる．表 11 に最長距離法を用い，上位 15 件を調査したものを示す．最短距離法のときと同様な動きを示すことがわかる．表 12 に最長距離法を用い，クエリとの距離が 0.0 であるコード片を含むクラスタにあるコード片をすべて検索したものを示す．最短距離法のときと同様に，バグを含むコード片を検出しきれていないことがわかる．

表 5: 最短距離法 (上位 5 件) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	491	61	0.124	0.693	0.814	0.749
6	532	65	0.122	0.739	0.799	0.768
7	562	67	0.119	0.761	0.787	0.774
8	606	68	0.112	0.773	0.771	0.772
9	633	69	0.109	0.784	0.76	0.772
10	658	70	0.106	0.795	0.751	0.772
11	701	72	0.103	0.818	0.735	0.774
12	816	74	0.091	0.841	0.691	0.759
13	840	75	0.089	0.852	0.682	0.758
14	870	75	0.086	0.852	0.671	0.751
15	891	75	0.084	0.852	0.663	0.746

表 6: 最短距離法 (上位 10 件) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	589	70	0.119	0.795	0.777	0.786
6	617	73	0.118	0.83	0.766	0.797
7	636	69	0.108	0.784	0.759	0.771
8	672	70	0.104	0.795	0.746	0.770
9	696	71	0.102	0.807	0.736	0.770
10	715	72	0.101	0.818	0.729	0.771
11	758	73	0.096	0.83	0.713	0.767
12	872	75	0.086	0.852	0.67	0.750
13	896	76	0.085	0.864	0.661	0.749
14	926	76	0.082	0.864	0.649	0.741
15	946	76	0.08	0.864	0.642	0.737

表 7: 最短距離法 (上位 15 件) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	653	73	0.112	0.83	0.753	0.790
6	672	75	0.112	0.852	0.746	0.795
7	691	71	0.103	0.807	0.738	0.771
8	727	72	0.099	0.818	0.725	0.769
9	751	73	0.097	0.83	0.716	0.769
10	770	74	0.096	0.841	0.708	0.769
11	810	75	0.093	0.852	0.693	0.764
12	924	77	0.083	0.875	0.65	0.746
13	948	78	0.082	0.886	0.641	0.744
14	978	78	0.08	0.886	0.63	0.736
15	998	78	0.078	0.886	0.622	0.731

表 8: 最短距離法 (all) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	2196	77	0.035	0.875	0.168	0.282
6	2191	78	0.036	0.886	0.17	0.285
7	2199	74	0.034	0.841	0.167	0.279
8	2195	75	0.034	0.852	0.169	0.282
9	2170	75	0.035	0.852	0.178	0.294
10	2172	76	0.035	0.864	0.178	0.295
11	2172	77	0.035	0.875	0.178	0.296
12	2176	79	0.036	0.898	0.176	0.294
13	2191	80	0.037	0.909	0.17	0.286
14	2213	80	0.036	0.909	0.162	0.275
15	2227	80	0.036	0.909	0.157	0.268

表 9: 最長距離法 (上位 5 件) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	509	63	0.124	0.716	0.807	0.759
6	550	66	0.12	0.75	0.792	0.770
7	588	67	0.114	0.761	0.777	0.769
8	623	69	0.111	0.784	0.764	0.774
9	646	70	0.108	0.795	0.755	0.774
10	665	71	0.107	0.807	0.748	0.776
11	694	72	0.104	0.818	0.737	0.775
12	809	74	0.091	0.841	0.694	0.760
13	838	75	0.089	0.852	0.683	0.758
14	870	75	0.086	0.852	0.671	0.751
15	903	75	0.083	0.852	0.658	0.743

表 10: 最長距離法 (上位 10 件) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	595	72	0.121	0.818	0.775	0.796
6	623	67	0.108	0.761	0.764	0.762
7	655	68	0.104	0.773	0.752	0.762
8	687	70	0.102	0.795	0.74	0.767
9	701	71	0.101	0.807	0.735	0.769
10	716	72	0.101	0.818	0.729	0.771
11	741	73	0.099	0.83	0.719	0.771
12	856	75	0.088	0.852	0.676	0.754
13	883	76	0.086	0.864	0.666	0.752
14	915	76	0.083	0.864	0.654	0.744
15	947	76	0.08	0.864	0.641	0.736

表 11: 最長距離法 (上位 15 件) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	646	75	0.116	0.852	0.755	0.801
6	668	69	0.103	0.784	0.747	0.765
7	695	70	0.101	0.795	0.737	0.765
8	727	72	0.099	0.818	0.725	0.769
9	741	73	0.099	0.83	0.719	0.771
10	754	74	0.098	0.841	0.715	0.773
11	779	75	0.096	0.852	0.705	0.772
12	894	77	0.086	0.875	0.661	0.753
13	919	78	0.085	0.886	0.652	0.751
14	950	78	0.082	0.886	0.64	0.743
15	982	78	0.079	0.886	0.628	0.735

表 12: 最長距離法 (all) の結果

クラスタ数	コード片数	類似バグを含むコード片数	Precision	Recall	削減率	調和平均
5	1689	78	0.046	0.886	0.36	0.512
6	1693	72	0.043	0.818	0.359	0.499
7	1577	72	0.046	0.818	0.403	0.540
8	1538	74	0.048	0.841	0.418	0.558
9	1536	75	0.049	0.852	0.418	0.561
10	1532	76	0.05	0.864	0.42	0.565
11	1544	77	0.05	0.875	0.415	0.563
12	1654	79	0.048	0.898	0.374	0.528
13	1499	80	0.053	0.909	0.432	0.586
14	1530	80	0.052	0.909	0.421	0.575
15	1560	80	0.051	0.909	0.409	0.564

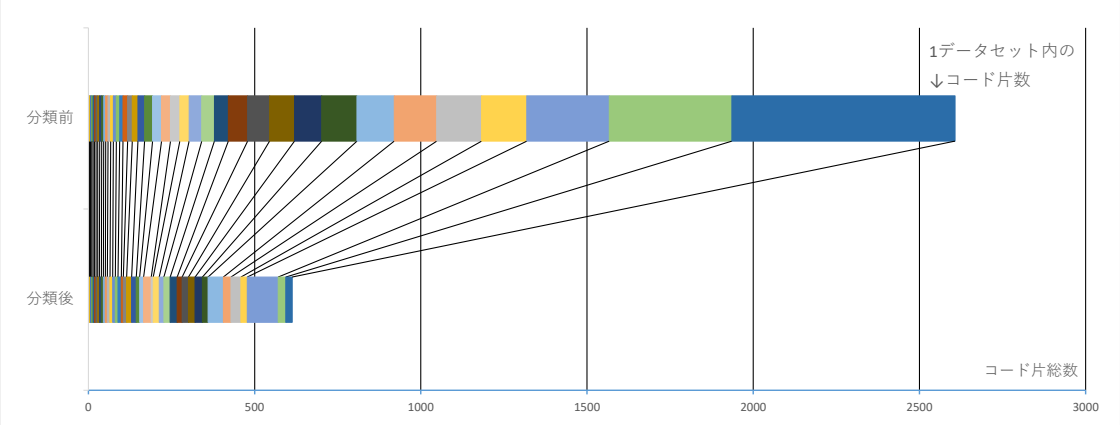

図 6: 分類前と分類後における各データセットに含まれる類似コード片数の変化

4.4 RQ3. 分類手法適用後の類似コード片の総数は適用前と比べてどうか?

図 6 に NCDSearch の出力結果と提案手法適用後それぞれに含まれる各データセットに含まれる類似コード片の総数を比較したものを示す．適用後のグラフには，3 つの提案手法の中で最も有効な最長距離法（クラスタ数=5，上位 15 件）を用いたときの値を使用した．図 6 から提案手法適用後は各データセット内のコード片数は全体的に減少していることがわかる．中央値の変化はあまり見られなかった．つまり，もともとコード片数が多く含まれるデータセットについては大きく変化し，もともとコード片数が少ないデータセットについてはあまり変化しなかった．これは手法の特徴として，NCDSearch からの出力結果が多くても少なくても一定の数だけ検索する特徴があるため，もともと含まれるコード片数が少ないデータセットについては変化が少なかったと推測される．

4.5 RQ4. NCDSearch の出力結果の上位を調査した場合と比較して，提案手法は有効か?

NCDSearch の出力結果を上から順に調査したものと，3 つの各手法で調和平均において最高の値を出したときの点を図 7 に示す．図を見てわかることは，コード片調査数 N について，調査する件数を増やせば増やすほど，Recall は上昇するが削減率は減少すること，つまり両者はトレードオフの関係にあることである．3 つの提案手法と N の関係について，各 N の点同士を結んだグラフと 3 点との位置関係について調べた．図 7 から，3 点ともグラフの左下の領域，つまり 3 つの各提案手法の Recall と削減率の調和平均は，NCDSearch の出力結果を上から順に調査したときのそれより低く，NCDSearch の出力結果の上位 10 件を調査したときの値と近いことが判明した．よって本実験では単純に NCDSearch の結果を上位 N 件調査するときの方が 3 つの提案手法より有効であることが判明した．表 13 に NCDSearch

表 13: 単純に上位を確認する手法

上位 N 件	1	5	10	15	20	25	30	35	40	50
Precision	0.264	0.251	0.183	0.157	0.133	0.115	0.104	0.096	0.089	0.078
Recall	0.159	0.648	0.784	0.875	0.898	0.898	0.909	0.920	0.920	0.920
削減率	0.979	0.912	0.854	0.810	0.770	0.734	0.702	0.673	0.645	0.598
調和平均	0.274	0.757	0.817	0.841	0.829	0.808	0.792	0.777	0.759	0.725

表 14: クラスタに含まれる類似バグの推移

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	⋯ 57	2	2	12	2										
	3	2	⋯ 54	2	12	2									
	3	9	2	⋯ 54	2	3	2								
	3	1	2	⋯ 54	8	2	3	2							
	3	⋯ 50	1	2	⋅ 4	8	2	3	2						
	3	⋯ 50	1	7	2	⋅ 4	1	2	3	2					
	3	⋯ 50	1	7	2	⋅ 4	1	2	1	2	2				
	3	⋯ 50	1	7	2	⋅ 4	1	1	2	1	1	2			
	3	⋯ 50	1	3	2	⋅ 4	1	4	1	2	1	1	2		
	3	⋯ 50	3	1	3	2	⋅ 4	1	1	1	2	1	1	2	
	3	⋯ 50	3	1	1	3	1	⋅ 4	1	1	1	2	1	1	2

の出力結果を上位から調査したものを示す．Recall の値から，バグを含むコード片は上位に固まっていることがわかる．実行時間について，NCDSearch のみを用いた場合が約 380 秒で，分類を用いた場合が約 629 秒であった．調査するコード片数が多ければ多いほど提案手法を用いることによる時間の短縮が見込まれる．表 14 に各クラスタに含まれる類似コード片の数と類似バグコード片の数を示す．数字の横の点はバグを含むコード片がいくつ存在するかを示す．このように類似バグを含むコード片が 1 つのクラスタに固まっている場合，既存研究との差ははっきりと表れない．

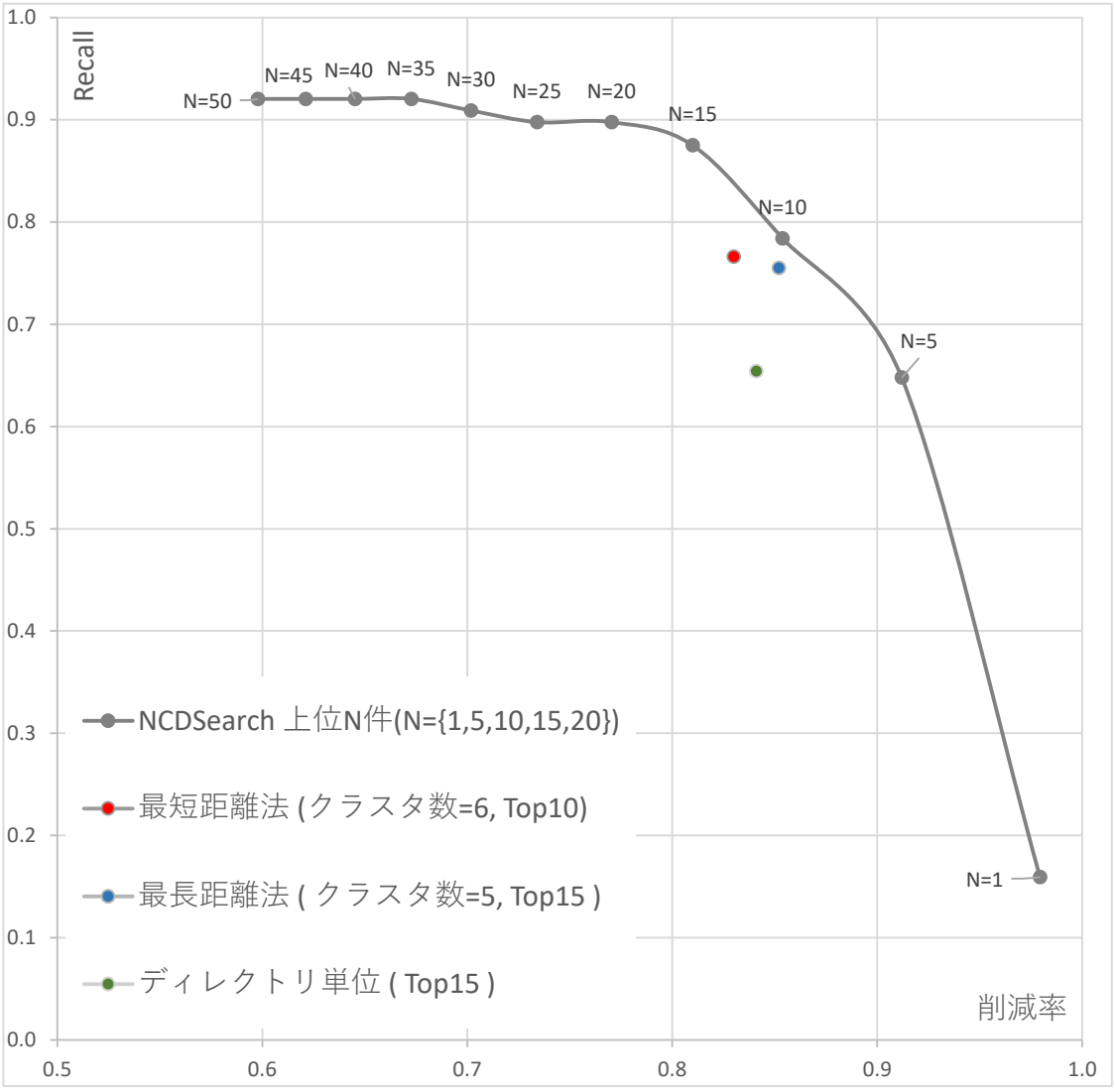

図 7: 単純に順位を確認する方法と 3 つの手法 (縦軸:recall，横軸:削減率)

5　まとめと今後の課題

本研究では類似バグの効率的な抽出を実現する類似コード片の分類手法を 3 つ提案した．評価では 3 つの分類手法全てで高い Recall を維持したまま確認すべき類似コード片を大きく削減することができた．3 つの提案手法の中では最長距離法 (クラスタ数=5，上位 15 件) が一番有効だった．ついで最短距離法（クラスタ数=6，上位 10 件）が有効な手法だった．3 つの提案手法の調和平均は，NCDSearch の出力結果の上位 10 件を調査したときの調和平均に近い値となった．今回挙げた提案手法はいずれも既存研究の結果を上回ることができなかった．

今後の課題として，より精度をあげるためにデータセットのサイズごとにパラメータに変化をつけて調査を行うことが挙げられる．また本手法ではパラメータによる精度の変動が少なかったので，距離尺度以外のメトリクスを採用や，新たなより精度が高い分類手法の検討が挙げられる．

謝辞

本研究ならびにに研究室生活，研究への心構えなど，多岐にわたってご指導およびご助言を賜りまし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克郎教授に心より深く感謝いたします.

本研究において，研究の方針や論文の書き方など，様々な御指導を頂きました奈良先端科学技術大学院大学先端科学技術研究科石尾隆准教授に深く感謝いたします.

本研究にならびに研究室生活において，主に発表スライドや発表の仕方について様々な御指導および御助言を頂きまし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 神田哲也 助教に深く感謝いたします.

本研究ならびに日常生活において，同じ高校出身ということもあり並々ならぬお心遣いをいただき，研究の進捗が滞るといち早く本研究に必要なツールを用意していただいたり，日頃より進捗を気に掛けていただいたり研究に対して意欲が持てるような助言をいただい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 伊藤薫氏に深く感謝申し上げます.

本研究ならびに研究室生活において，定期的にしばしば様々なご指導およびご助言をいただき，さらに本研究に必要なツールを作成していただいたり，論文を常に添削していただいたり，研究の進捗が滞ると心配していただい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 嶋利一真氏に深く感謝申し上げます.

最後に，その他様々な御指導，御助言等を頂い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] T. Ishio, N. Maeda, K. Shibuya, and K. Inoue. Cloned buggy code detection in practice using normalized compression distance. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 591–594, Sep. 2018.

[2] Ming Li, Xin Chen, Xin Li, Bin Ma, and P. M. B. Vitanyi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, Dec 2004.

[3] Daniel Müllner. “modern hierarchical, agglomerative clustering algorithms.”. *[stat.ML]*, 24(11):999–1006, 2011.

[4] Postgresql. https://www.postgresql.org/.

[5] Git. https://git-scm.com/.

[6] The linux kernel archives. https://www.kernel.org/.

[7] J. Li and M. D. Ernst. Cbcd: Cloned buggy code detector. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 310–320, June 2012.