Vol.2017-SE-195 No.17
2017/3/12

ソースファイル群の類似性を用いたソフトウェア再利用元の検索

坂口 雄亮[1,a)]　石尾 隆[1,b)]　伊藤 薫[1,c)]　井上 克郎[1,d)]

概要：本研究では，開発中ソフトウェアが再利用するソフトウェアとそのバージョンを特定する手法を提案する．ソフトウェアはソースファイルの集合とみなし，その各ソースファイルについて類似するソースファイルを既存の高速検索手法を用いて，大量のソフトウェアのソースファイルから検索する．そして，与えられたソフトウェアがもつソースファイルの集合と類似度がより高いものが上位になるように，検索結果から得られたソフトウェア群の順位付けを行う．この操作によって，再利用したソフトウェアとそのバージョンを知ることが可能になる．実験として，Mozilla Firefox と Android が再利用するソフトウェア 72 件に対して手法を適用した．その結果，手法が出力するソフトウェア集合の中に再利用元が含まれるものが 71 件，その集合を絞り込んだ集合の中に再利用元が含まれるものが 66 件であった．その 66 件について，再利用元の順位が 1 位であるものは 50 件であった．

1. はじめに

ソフトウェア開発において，既存のソースコードを再利用することが盛んに行われており，ソフトウェアに必要な機能を開発するコストの削減に役立っている．再利用可能なソースコードとしてはオープンソースソフトウェアが活用されている．コードを再利用することによって，機能を開発するコストを大幅に削減することが出来る．オープンソースソフトウェアは，多くのユーザによって使用されているため，安全性が比較的高いというメリットがあることから，企業の製品開発にも利用されるようになっている [1]，[2]．

ソースコードの再利用の行われ方は様々あるが，C 言語でのソフトウェア開発では，既存のソフトウェアのソースコードをコピーして取り込むという方法で再利用が行われる．コピーして取り込むと，開発中のソフトウェアに合わせた変更が容易になるというメリットがある一方，再利用したソースコードに含まれる脆弱性を取り込んでしまうおそれがある．そのような場合，ソフトウェアの更新を速やかに行わなければいけないが，更新された部分と独自に変更した部分が競合する可能性があるため，再利用しているソフトウェアとバージョンの情報が必要になる．しかし，多くのプロジェクトにおいてそのような情報が記録されていないことが判明している [3]．

ソースファイルが一致するかどうかの単純な判定は容易であるが，実際には再利用後にソースコードの変更が加えられることがあるため不十分であり，ソースファイルの内容によって再利用元を検索する必要がある．ソースファイルに基づいて再利用元を検索する既存手法として，Kawamitsu ら [4] は，リポジトリ内の最も類似したファイルのリビジョンを再利用元のバージョンとして識別するコード比較手法を提案した．この手法は，あらかじめ再利用元ライブラリのリポジトリを用意しなければならない．また彼らは，locality-sensitive hashing[5] を用いた類似ファイルの高速検索手法を提案した [6]．これは，検索のクエリとして与えられたソースコードのファイルに対し，大量のソースコードのファイルの中からソースコードの内容がクエリに類似するものを高速に検索する．1 つのファイルについての正確性や有効性について述べられているが，入力として複数ファイルを与え，それらの情報を合わせて利用することが課題とされている．

本研究では，複数ファイルの検索結果であるソースファイル群を利用して，開発中ソフトウェアが再利用しているソフトウェアの再利用元を自動的に推定する手法を提案する．提案手法では，開発中ソフトウェアのソースファイルを入力とする．まず，各ソースファイルについて，locality-sensitive hashing を用いた検索手法を適用し，大量

[1] 大阪大学大学院情報科学研究科
Osaka University, Suita, Osaka 565-0871, Japan
[a)] s-yusuke@ist.osaka-u.ac.jp
[b)] ishio@ist.osaka-u.ac.jp
[c)] ito-k@ist.osaka-u.ac.jp
[d)] inoue@ist.osaka-u.ac.jp

のソフトウェアのソースファイルから成るデータベースから類似するファイルを検索する．データベースは，フリーなオペレーションシステムである Debian GNU/Linux が採用してきたほぼすべてのパッケージで構築されている．検索後，各入力ソースファイルの類似ファイル群をデータベースに登録されているソフトウェア毎にまとめ，ソフトウェア集合を得る．そして，入力ソースファイル集合により近いものが上位となるようなソフトウェア間の半順序関係を用いて順位付けられた再利用元ソフトウェアリストを得る．

手法の評価のために，Mozilla Firefox と Android のソースコードに再利用されているソフトウェアを対象として手法の妥当性を評価した．

以降，第 2 章では，本研究の背景について述べる．第 3 章で提案手法について説明し，第 4 章で手法の評価を述べる．第 5 章では妥当性への脅威について述べ，最後に第 6 章で，まとめと今後の課題を記述する．

2. 背景

2.1 ソースコードの再利用

ソフトウェア開発者は，既存のソフトウェアのソースコードを再利用してソフトウェアを開発する [1]．ソースコードの再利用は，機能を開発するコストを大幅に抑えるため，ソフトウェア開発の効率化につながる．Rubin ら [7] は，企業内での新規ソフトウェア開発において，既存のソフトウェアの再利用が行われていることを報告している．Mohagheghi ら [8] は，再利用されたコンポーネントはそうでないものと比べて安全性が高いと報告している．

再利用のされ方として，ソフトウェアの機能を開発する際に，新しく開発するのではなく，その機能を持つオープンソースソフトウェア (以下 OSS とも表記) が一般的に利用される．その際に，再利用したコードを開発中のソフトウェアに合うように独自の変更が加えられることがある．Java プロジェクトでの OSS の利用は，バイナリファイルの再利用が一般的である．また，新たにソフトウェアを開発する際には，既存ソフトウェアを全て再利用し，細かな改変が加えられ類似したソフトウェアが開発する派生開発，ソフトウェアプロダクトで全体に共通するコア資産と独自性を持つ機能部品に分割し，これらを組み合わせてソフトウェアが開発するソフトウェアプロダクトラインエンジニアリング [9] もソースコードの再利用と言える．

2.2 オープンソースソフトウェアの管理状況

OSS は安全性が高いとは言え，再利用したソースコードにバグやセキュリティに関する脆弱性が発見されることもある．脆弱性が公開されると開発者は，脆弱性の含まれたコードを再利用しているのかを確認する必要がある．再利用している場合，開発者のソフトウェアにも脆弱性が含まれることになり，それに対する修正を手元のソースコードにも適用すべきである．脆弱性が含まれたコードを再利用しているのかを確認するためには，再利用しているソフトウェアを管理する必要がある．

しかし，多くのプロジェクトにおいて，再利用しているオープンソースソフトウェアの管理がなされていないことがわかっている．Xia ら [3] は，オープンソースライブラリを再利用しているプロジェクトについて，そのライブラリが脆弱性の含まれるソースコードを保有しているバージョンであるかどうかを調査した．その結果，zlib について，45 プロジェクト中 14 プロジェクト（31.1%）が脆弱性のあるバージョンを使用していることがわかった．同様に，libcurl については，28 プロジェクト中 24 プロジェクト（85.7%），libpng については，50 プロジェクト中 46 プロジェクト (92%) であった．また，計 123 プロジェクト中 27 プロジェクト (22%) で，ライブラリのソースコードを再利用後に編集しており，23 プロジェクト（18.7%）では，再利用しているライブラリのバージョンに関する情報が残っていなかった．さらに，6 プロジェクト（4.9%）で，ディレクトリ名や他のライブラリのソースコードが含まれており，管理が容易ではなかった．このように，再利用しているソフトウェアの管理について，そのソフトウェア名やバージョンなどの再利用元を推定することは有用であると考えられる．

2.3 起源分析

本研究では，再利用元の特定に関する研究として，起源分析が上げられる．起源分析とは，ソースコードの起源を複数のソフトウェアから特定する技術で，ソースコードの類似性を用いるため，コードクローン検出の一部と言える．コードクローンとは，ソースコード中に存在する同一または類似した部分を持つコード片のことであり，既存コードのコピーアンドペーストによる再利用等が原因で生じる．コードクローン検出技術の例として，Kamiya ら [10] は，ファイル間の類似したコード片を検出するために，CCFineder を提案した．また，Sasaki ら [11] が提案した FCFinder は，プロジェクト間でのファイルクローンを高速に検出する．

起源分析では，Inoue ら [12] は，Ichi tracker という名前のツールを提案した．これは，インターネット上の様々なリポジトリ間で，特定のソースファイルの類似ファイルを検索し，コード断片との類似度でクラスタリングを行う．そして，ソースファイルを時系列順に並べることによりソースコードの再利用の経緯を可視化する．このツールでは，1 つのファイルについて注目している．Kawamitsu ら [4] は，リポジトリに含まれているソースコードについて，ライブラリのバージョンを推定する手法を提案した．ソースコードの類似度として最長共通部分列に基づいた類

似度 [9] を用いて，最も類似度の高いソースコードが再利用元であるという仮定に基づきバージョンを提示した．この手法はあらかじめ再利用を行ったソフトウェアのリポジトリと再利用元のリポジトリが必要なため，再利用元が不明である場合は利用することが出来ない．

2.4 LSH アルゴリズムを利用した高速検索手法

本研究では，類似ソースファイルの検出に，川満ら [6] の locality-sensitive hashing(以下 LSH とも表記) アルゴリズムを利用した高速検索手法を用いる．この手法では，検索のクエリとして与えられたソースファイルに対し，データベース中のソースファイルからソースコードの内容がクエリに類似するものを検索する．LSH を用いることにより，実際の類似度ではなく，類似度の推定値を計算することで高速な検索を可能にしている．

2.4.1 ソースファイル間の類似度

ソースファイル f_1, f_2 の類似度は，各ファイルの 3-gram 集合間の Jaccard 係数を利用し以下のように定義される．Jaccard 係数は，2 つの集合間の類似度であり，文字列間の類似度を計算する手法としてよく利用される．

$$sim(f_1, f_2) = \frac{|3\text{-}grams(f_1) \cap 3\text{-}grams(f_2)|}{|3\text{-}grams(f_1) \cup 3\text{-}grams(f_2)|}$$

ただし，$3\text{-}gram(f)$ は，ソースファイル f から抽出された 3-gram の多重集合である．多重集合の要素は，ソースコードからコメント，空白行を除いたものから得た字句列から長さ 3 の部分文字列である．

2.4.2 Locality-Sensitive Hashing

クエリのソースファイルとデータベース中の大量のソースファイルの間の Jaccard 係数を求めるには，膨大な時間がかかってしまう．川満らの手法では，文献 [13] の方法を用いて LSH を構成した．各ソースファイルについて，3-gram の多重集合に $r \times b$ 個のハッシュ関数を用いて，r 次元の MinHash の値からなるベクトルを b 個求める．そして，クエリのソースファイルとデータベース中のソースファイル間の対応するベクトルを比較し，ベクトルが 1 つでも一致したものを出力する．この時，類似度を p とすると，b 個のベクトルのうち 1 つ以上のベクトルが一致する確率は，$1-(1-p^r)^b$ となる．パラメータ b, r を調整することで，出力するソースファイルを調整することが出来る．類似度 p に対する最尤推定量 $\hat{p}$ は，ベクトルが一致した個数を x とすると，

$$\hat{p} = \sqrt[r]{\frac{x}{b}}$$

となる．また，類似度 p がある程度低いと，$x = 0$ となり，類似度の推定値が 0 になる特性を持つ．川満らの手法では，64bit 長のハッシュ値を使用し，LSH のパラメータは $b = 120, r = 8$ を利用した．その時の $1-(1-p^r)^b$ のグラフを図 1 に示す．本手法においても，同様のパラメータを利用する．

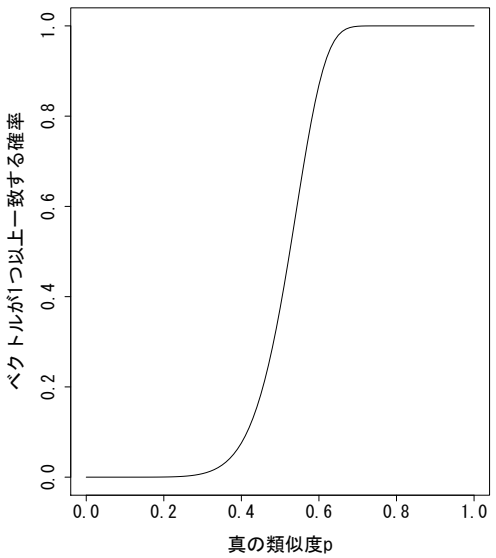

図 1 LSH のパラメータ

3. 提案手法

本研究では，開発中ソフトウェアが再利用しているソフトウェアの再利用元を推定する手法を提案する．既存の様々なソフトウェアが蓄えられたデータベースが存在することを前提とし，入力として開発中のソフトウェアのソースファイル集合が与えられると，その入力ソースファイル集合と類似するソースファイルを持つソフトウェア群をデータベースから抽出し，類似度の順に並べ替えたリストを出力する．

提案手法は，以下の 3 つのステップから構成される．

(1) 開発中ソフトウェアのソースファイル集合を入力し，各ソースファイルをクエリとして川満らの手法 [6] を用いて，類似するソースファイル集合を得る．
(2) 類似ソースファイル集合を，それらを保有するソフトウェア毎にまとめ再利用元候補ソフトウェア集合を得る．
(3) 再利用元候補ソフトウェア集合の各ソフトウェアに順位付けを行い，再利用元ソフトウェアリストを出力する．

3.1 類似ソースファイルの検索

入力された開発中ソフトウェアのソースファイル集合の拡張子を調べて，検索対象となるソースファイル集合 $F = \{f_1, f_2, ..., f_n\}$ を得る．そして，川満らの手法を用いて，F 中の各ソースファイルをクエリとして検索を行う．F 中のあるファイル f_i に対し，類似度の推定値 $\hat{sim}$ が 0 より大きい，すなわち，真の類似度 sim がある程度大きい類似ソースファイル集合 $\tilde{F}(f_i)$ を抽出する．

表 1　手法行列の例

入力ソースファイル	候補ソフトウェア					
	S_1	S_2	S_3	S_4	S_5	S_6
f_1	0.8	0	0.95	0.85	0	0.85
f_2	0.8	0	0.9	0.85	0	0.95
f_3	0	0	1.0	0.9	0	0.9
f_4	0	1.0	0	0	1.0	0
f_5	0	0.9	0	0	0.8	0

本手法では，検索を行い類似するソースファイルを得たあとに，真の類似度 sim を計算する．本研究では，真の類似度 sim による閾値を 0.8 とし，それ未満のものは類似ソースファイルとみなさず集合 $\tilde{F}(f_i)$ に含めない．また，あるソースファイル f_i の検索結果において，同じソフトウェアが持つ類似ソースファイルが複数ある場合，類似度が最も高いソースファイルをそのソフトウェアから再利用したソースファイルとみなして $\tilde{F}(f_i))$ に含める．同一の類似度のソースファイルが複数ある場合は，ソースファイル名がアルファベット順で先頭のものを $\tilde{F}(f_i)$ に含める．

3.2　候補ソフトウェア集合の取得

抽出された各類似ソースファイルは，それを保有するソフトウェアがただ 1 つ存在する．本手法ではそのようなソフトウェアを候補ソフトウェアと呼ぶ．集合 $\tilde{F}(f_i)$ の各類似ソースファイルから候補ソフトウェア集合 $SW = \{S_1, S_2, ..., S_m\}$ を得る．

集合 $\tilde{F}(f_i)$ 内のある類似ソースファイル f からは，入力ソースファイル f_i と類似度 $sim(f_i, f)$ で類似しており，候補ソフトウェア S_j が保有しているという 3 つの情報が得られる．そこで，入力ソースファイル f_i を行，候補ソフトウェア S_j を列，成分を類似度 $sim(f_i, f)$ とするような $n \times m$ 行列 M を作る．行番号 i は，入力ソースファイル f_i の番号，列番号 j は，候補ソフトウェア S_j の番号である．ただし，$f \in \tilde{F}(f_i) \wedge f \in S_j$ となるような類似ソースファイル f がない場合，行列の成分となる類似度を 0 とする．この行列から各候補ソフトウェア S_j は，行列の j 列目の列ベクトルを得て，$S_j = \left(M_{1j}, M_{2j}, ..., M_{nj}\right)$ と表す．表 1 に，入力ソースファイル 5 つ，候補ソフトウェア 6 つで構成される行列の例を示す．入力ソースファイル f_1 の類似ソースファイルのうち，候補ソフトウェア S_1 が保有するものの類似度は 0.8 と読むことができる．

3.3　候補ソフトウェアの順位付け

類似するソースファイルを 1 つでも持てば候補ソフトウェアとなる．そのため，候補ソフトウェア集合 SW は，非常に大きくなる可能性があり，そこから再利用元ソフトウェアを見つけ出すことは困難である．そこで，本手法では以下の手順により，候補ソフトウェアを絞り込み集合 F に類似しているように順位付けを行う．

(1) 候補ソフトウェア集合 SW に順序関係を定義し，半順序集合を得る
(2) 有力ソフトウェアを抽出し，集合 F との距離を設定し順位付ける
(3) その他の候補ソフトウェアをトポロジカルソートをして順位付けしリストを出力する

3.3.1　順序関係

候補ソフトウェア集合 SW に以下のような順序関係を定義し，候補ソフトウェア間に順序を付け，候補ソフトウェアの半順序集合 SW_{po} を得る．

定義. 候補ソフトウェア S_a, S_b において，すべての入力ソースファイルについて S_a よりも S_b のほうが類似度が高いソースファイルを保有するとき，すなわち任意の i について $S_a[i] \leq S_b[i]$ が成り立つとき，S_b は S_a よりも，再利用しているソフトウェアの再利用元である可能性が高いとし，$S_a \leq S_b$ と表す．

表 1 から得られる候補ソフトウェア集合 $S = \{S_1, S_2, ..., S_6\}$ に，順序関係を定義すると，$S_1 \leq S_1$, $S_1 \leq S_3$, $S_1 \leq S_4$, $S_1 \leq S_6$, $S_2 \leq S_2$, $S_3 \leq S_3$, $S_4 \leq S_3$, $S_4 \leq S_4$, $S_4 \leq S_6$, $S_5 \leq S_2$, $S_5 \leq S_5$, $S_6 \leq S_6$ という順序関係を持つ候補ソフトウェアの半順序集合 SW_{po} が得られる．

3.3.2　有力ソフトウェアの順位付け

候補ソフトウェアの半順序集合 SW_{po} 内のどの候補ソフトウェアよりも小さくない候補ソフトウェアは，他の候補ソフトウェアよりも再利用元ソフトウェアである可能性が高いと言える．本手法では，このような候補ソフトウェアを「**有力ソフトウェア**」と呼ぶ．定義した順序関係では，有力ソフトウェア間の順序はつかない．そのため，有力ソフトウェア S と入力ソースファイル集合 F 間の距離を距離関数を使って求め，その長さで順位を付ける．本研究では距離関数として，以下のようなマンハッタン距離を採用する．

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

入力ソースファイル集合 F の各要素はソースファイルであるため，類似度ベクトル F_{sim} を求める．入力ソースファイル f_i 自身との類似度であるため，類似度 $sim(f_i, f_i) = 1$ である．よって，F_{sim} の全てのベクトルの大きさは 1 である．有力ソフトウェア S と入力ソースファイル集合 F 間の距離 $d(S_j, F_{sim})$ は，上の式に当てはめると，

$$d(S_j, F_{sim}) = \sum_{i=1}^{n} |S_j(i) - 1|$$

となる．距離関数からわかるように，入力ソースファイルとの類似度が高いほど，また，類似ソースファイルを多く持つほど，距離 $d(S_j, F)$ は短くなる．よって，有力ソフトウェアにおいては，距離が短い順に順位を付ける．ただし距離が同じ場合は，入力によるものとする．

例の半順序集合 SW_{po} における有力ソフトウェアは，

表 2　有力ソフトウェアの順位付け後の行列

入力ソースファイル	候補ソフトウェア					
	◎S_3	◎S_6	◎S_2	S_1	S_4	S_5
f_1	0.95	0.85	0	0.8	0.85	0
f_2	0.9	0.95	0	0.8	0.85	0
f_3	1.0	0.9	0	0	0.9	0
f_4	0	0	1.0	0	0	1.0
f_5	0	0	0.9	0	0	0.8

S_2, S_3, S_6 である．各有力ソフトウェアと集合 F との距離は，それぞれ 3.1，2.15，2.3 となる．有力ソフトウェアを距離が短い S_3, S_6, S_2 の順に 1 位，2 位，3 位と順位を付ける．k 位である候補ソフトウェアが k 列目となるように列を入れ替えた行列を表 2 に示す．順位が確定した有力ソフトウェア名には◎印を付けており，残りの候補ソフトウェアは順位がついていない．

3.3.3　トポロジカルソート

本手法では，ソフトウェアの再利用元を順位付けて出力することを目的としている．再利用元が有力ソフトウェアでない可能性もあるため，残りの候補ソフトウェアも順位をつける．残りの候補ソフトウェアについては，半順序集合 SW_{po} の各要素を半順序関係に矛盾しないように並べ替えて順位付ける．そこで本手法では，トポロジカルソートを利用する．

半順序集合は，$S_a \leq S_b$ のとき，S_a から S_b に向かう辺をつくることで有向グラフとみなすことができる．ただし，反射律 $S_a \leq S_a$ と，反対称律 $S_a \leq S_b$ かつ $S_b \leq S_a$ を満たすような辺はつくらない．また，$S_a \leq S_b$ かつ $S_b \leq S_c$ のとき推移律である $S_a \leq S_c$ を満たすような辺もつくらない．反射律と反対称律を満たす辺が存在しないということは，グラフ上では閉路が存在しない，すなわちグラフが 有向非巡回グラフ であることに対応する．

半順序集合を表現した 有向非巡回グラフ に対してトポロジカルソートを適用する．半順序関係にトポロジカルソートを使用し得られる解は，半順序関係の順序を満たすような全順序の 1 つに相当する．有力ソフトウェアは有向非巡回グラフにおいて，出力辺が 0 であるノードで極大元にあたる．有力ソフトウェアを除く候補ソフトウェアの半順序集合 SW_{po} に以下の手順でトポロジカルソートを行い順位付ける．

(1) DAG の極大元であるノードをすべて選択
(2) 選択したノードを選択した順に順位付け
(3) DAG から選択したノードとそのノードへの入力辺を削除
(4) DAG にノードが残っていれば 1. へ戻る

例の半順序集合 SW_{po} から生成された有力ソフトウェアを含む DAG を図 2 に示す．有力ソフトウェアであるノードは◎で囲っており極大元であることがわかる．有力ソフトウェアを除く DAG の極大元は，S_4 と S_5 であり，この

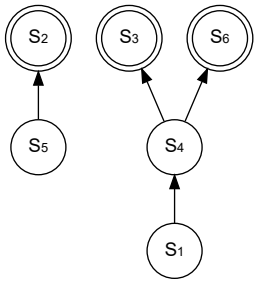

図 2　有向非巡回グラフの例

表 3　トポロジカルソート後の行列

入力ソースファイル	候補ソフトウェア					
	◎S_3	◎S_6	◎S_2	◎S_4	◎S_5	◎S_1
f_1	0.95	0.85	0	0.85	0	0.8
f_2	0.9	0.95	0	0.85	0	0.8
f_3	1.0	0.9	0	0.9	0	0
f_4	0	0	1.0	0	1.0	0
f_5	0	0	0.9	0	0.8	0

順に選択しそれぞれ 4 位，5 位と順位付ける．そして DAG からノード S_4, S_5 とそのノードへの入力辺を削除し，再び極大元を選ぶ．残ったノード S_1 が極大元であるので選択し，6 位と順位付ける．有力ソフトウェアの順位付けと同様に k 位である候補ソフトウェアが k 列目となるように列を入れ替え，すべてのソフトウェアの順位が確定した行列を表 3 に示す．

手法では最後に例のような候補ソフトウェアを順位付け並べ替えた行列をリストとして出力する．リストを先頭から見ることでソフトウェアの再利用元を推定することが出来る．この例では，入力ソースファイル $f_1, f_2., f_3$ の再利用元はソフトウェア S_3，入力ソースファイル f_4, f_5 の再利用元はソフトウェア S_2 であると推定することが出来る．

4.　実験

手法が出力する候補ソフトウェアリストと有力ソフトウェアから再利用元ソフトウェアを推定できるか確認するために，提案手法を Java で実装して実験を行った．対象言語は C/C++，Java とした．本節では，はじめにデータベースに登録するソフトウェアおよびファイルについて説明する．そして，Mozilla Firefox が再利用しているソフトウェア 20 個と，Android が再利用しているソフトウェア 52 個をデータセットとして実験を行い評価した．本稿ではスペースの都合上，データセットの説明は省略するが，ソフトウェアのリストは [14] に収録している．

4.1　データベース

データベースに登録するデータセットとして，フリーなオペレーションシステムである Debian GNU/Linux が使

用しているアーカイブのスナップショット [*1] を利用する．このアーカイブには 2005 年以来の使用してきたほぼすべてのパッケージのソースコードが含まれている．本研究では，1 パッケージを再利用元ソフトウェアの 1 つとみなす．アクセスした日付は，2016 年 8 月 19 日である．本手法では独自の変更が加えられていない，“*.orig.*” というパターンにマッチしたもののみをダウンロードした．

ダウンロードしたデータセットは，33,496 種類のパッケージを含んでおり，パッケージの総数は 188,212 である．データベースに登録するファイルは，C/C++のソースファイルは拡張子が，.c，.h，.cpp，.hpp であるもの，Java は.java であるものを対象とした．登録対象となるソースファイルの総数は，50,903,100 ファイル（18,569,351,349 LOC）である．LOC は，コメントや空白のみの行も含む．C/C++のソースファイルは，46,699,686 ファイル (17,722,308,883 LOC) である．また，Java のソースファイルは，4,203,414 ファイル (847,042,466 LOC) である．登録したファイルの合計ファイルサイズは，約 566GB である．

なお，検索クエリとなる Firefox 自身とその関連プロジェクトは検索から除外した．

4.2 評価方法

実験は，対象プロジェクトが再利用しているソフトウェアのうち，バージョン番号がわかっているものについて行う．そのようなソフトウェアを入力ソフトウェアと呼ぶ．入力ソフトウェアに対応するデータベース内の同名同バージョンのソフトウェアを正解ソフトウェアとする．ただし，正解ソフトウェアとなるものがデータベースにないような入力ソフトウェアは用いない．入力ソフトウェアに手法を適用し，以下の項目について調査し，評価と考察を行う．

- 正解ソフトウェアが出力された候補ソフトウェアリストに含まれているか．含まれているなら，正解ソフトウェアが有力ソフトウェアとして識別されているか．
- 正解ソフトウェアが有力ソフトウェアであるものについて，正解ソフトウェアを第何位に出力したか．

4.3 評価結果

入力ソフトウェアに対する正解ソフトウェアが，出力された候補ソフトウェアリストに含まれるかどうか，さらに有力ソフトウェアであるかどうかについて調べる．結果より再利用しているソフトウェアの再利用元の推定を行えるかどうかを評価することができる．手法は入力ソフトウェア単位に適用している．Firefox では，正解ソフトウェアが候補ソフトウェアリストに含まれる結果となるものは，20 件中 19 件 (95%) であった．また，正解ソフトウェアが有力ソフトウェアであるのは 17 件（85%) であった．Android では全ての正解ソフトウェアが候補ソフトウェアリストに含まれた．また，正解ソフトウェアが有力ソフトウェアである結果となったものは 52 件中 49 件（94.2%）であった．

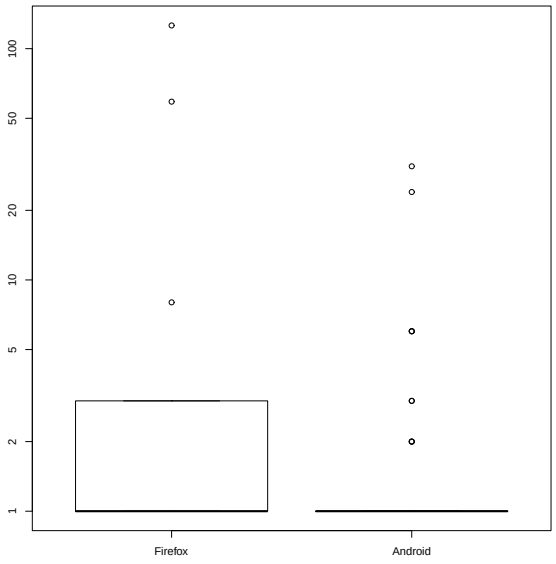

図 3 正解ソフトウェアの順位

結果より，Firefox，Android 共に正解ソフトウェアが候補ソフトウェアリストに含まれていた．これより，利用しているソフトウェアの情報がわからなくとも手法を用いることで，データベースに再利用元であるソフトウェアがあれば高い確率でそれが含まれるソフトウェアのリストを得ることが可能であると言える．さらに，ほとんどの正解ソフトウェアが有力ソフトウェアであるという結果から，有力ソフトウェアから再利用元であるソフトウェアを推定することが可能であると言える．

Firefox と Android における正解ソフトウェアの順位の箱ひげ図を図 3 に示す．図 3 より，ほとんどの正解ソフトウェアは 5 位以内である．よって，手法を用いることで有力ソフトウェアの上位にあるソフトウェアが再利用元と推定することが可能であると言える．多くの 1 位ではないソフトウェアは，同距離を同順位としてみなすと 1 位であった．つまり，距離が同じソフトウェアが多くある（多くの場合，入力ソフトウェアが持つファイルと同内容のファイルをもつソフトウェアが多数ある) 状態であり，これらについては再利用元を推定することは難しい可能性がある．これはソフトウェア再利用時，ソースコードに独自の変更を加えない場合に多く起こると考えられる．

提案手法で用いた順序関係について評価を行う．順序関係を用いると絞り込まれた有力ソフトウェアから再利用元を推定することが容易になると考えられる．また複数のソフトウェアを入力し順序関係を用いず距離のみを用いて順序付けた場合，ソースファイル数の少ないソフトウェアの

[*1] http://snapshot.debian.org/

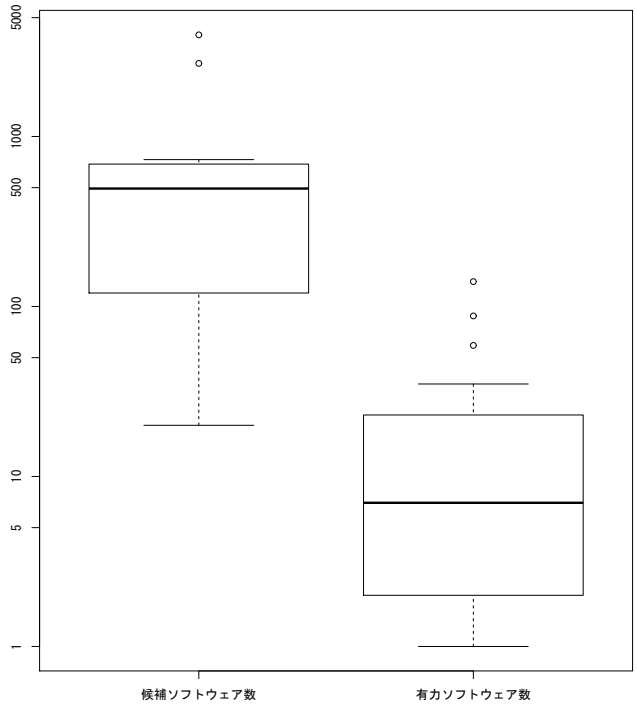

図 4 入力ソフトウェア単位の場合の各ソフトウェア数

表 4 入力ソフトウェアをまとめた場合の各ソフトウェア数

候補ソフトウェア数	有力ソフトウェア数
5969	598

表 5 順序関係の有無による正解ソフトウェアの順位

ソフトウェア名	順序関係なし	順序関係あり
cairo	359	130
graphite2	572	222
gtest*	1177	1855
hunspell	3963	558
libav	643	264
libevent	461	160
libffi	341	123
libjpeg-turbo	579	223
libopus*	269	1004
libpng	2642	481
libsoundtouch	2620	479
libvorbis	657	268
nspr	99	35
nss	17	8
snappy	3495	521
stlport	228	78
zlib*	1042	1653

再利用元はリストの下位になるが，順序関係を用いることで有力ソフトウェアとしてリストの上位になると考えられる．そこで候補ソフトウェア数と有力ソフトウェア数の比較を行い推定することが容易になるか調査する．また複数のソフトウェアを入力し，順序関係の有無による正解ソフトウェアの順位の変動を調べる．

Firefox の入力ソフトウェアに対する正解ソフトウェアが有力ソフトウェアとなった 17 件に対して評価を行う．入力ソフトウェア単位で手法を適用する場合と入力ソフトウェア 17 件全てをまとめて手法を適用する場合を考える．前者については，候補ソフトウェア数と有力ソフトウェア数の比較のみ行う．入力ソフトウェア単位の場合の 候補ソフトウェア数と有力ソフトウェア数は，箱ひげ図で示すと図 4 のようになる．入力ソフトウェア 17 件をまとめた場合の候補ソフトウェア数と有力ソフトウェア数は，表 4 に示す．図 4 より，順序関係を用いることで候補数を大幅に削減できることがわかる．また表 4 より，全てのソフトウェアをまとめて手法を適用した場合，有力ソフトウェア数が個別の適用結果の合計よりも多い結果となったが候補ソフトウェア数の約 10%であり，比較的少数の候補だけを抽出していると評価することができる．入力ソフトウェア単位の有力ソフトウェア数の合計よりも多くなった理由としては，2 つ以上の入力ソフトウェアのソースファイルを持つソフトウェアが有力ソフトウェアとなったと推定できる．

次に，順序関係の有無による正解ソフトウェアの順位について調べる．全てのソフトウェアをまとめた入力に対して，順序関係の有無によって得られる 2 つのリスト中の 17 つの正解ソフトウェアの順位を表 5 に示す．ソフトウェア名に “*” がついているものは，入力ソフトウェアをまとめて入力した場合の出力では正解ソフトウェアが有力ソフトウェアとならなかったものである．入力ソフトウェアをまとめた場合の出力でも正解ソフトウェアが有力ソフトウェアものは，14 件であった．その 14 件に対し，正解ソフトウェアの順位は全て上がっていることが確認できる．しかし，順位が 50 位以内に入るものは 2 件という結果であった．これは距離関数の定義より，多くの類似ソースファイルを持てば距離が短くなり，より上位になるという性質から複数ソフトウェアのソースファイルをもつソフトウェアが上位に位置しているためである．正解ソフトウェアが有力ソフトウェアにならなかった gtest，libopus，zlib は，特定のソフトウェアが再利用元ソフトウェアのソースファイルを完全に含んでおり，そのソフトウェアが上位に提示されていた．以上のことから，順序関係を用いることで再利用元候補の絞り込みは可能であるが，入力に含まれるソフトウェアの数が増えるほど有力ソフトウェアとなる精度が落ち，順位の上昇率が減る可能性がある．

5. 妥当性への脅威

本研究の実験において，プロジェクトが再利用してい

るソフトウェアの正解として Debian GNU/Linux のパッケージとして配布されているソフトウェアを利用した．各ソフトウェアの公式なリリースの配布ではなく，Debian GNU/Linux プロジェクトによる再配布という形であり，ソフトウェアの名前，バージョン番号が情報が正しくない可能性がある．

本研究の実験において，入力ソフトウェアのバージョン番号は各プロジェクトのバージョン番号が記述されているコミットログやファイルの情報を利用した．背景でも述べているように，再利用しているソフトウェアの管理は正しく行われていないという報告があるため，実際のバージョン番号と記述されているバージョン番号が異なる可能性がある．

6. まとめと今後の課題

本研究では，開発中ソフトウェアが再利用しているソフトウェアの再利用元を推定する手法を提案した．開発中ソフトウェアのソースファイル集合を与えると，入力ソースファイル集合と類似するソースファイルを持つソフトウェアを類似する順に並べ替たリストを表示する．この類似順に並んだソフトウェアリストを見ることで，どのソフトウェアを再利用したのか推定することが可能となる．提案手法を用いて実験を行い，プロジェクトが再利用するソフトウェア 72 件の内，71 件で再利用元を含むリストを出力した．また，手法では再利用元である可能性が高いソフトウェアを絞りこんでおり，絞り込んだソフトウェアの中に再利用元が含まれるものは 66 件であった．そのうち順位が第 1 位のもの 50 件であり，再利用元を推定することが可能であることを確認した．

今後の課題として，再利用元である可能性が高いソフトウェアの絞り込みを類似度による順序関係を用いたが，ソフトウェア全体が持つファイルの総数などによる絞り込みを用いることで精度の向上が期待できる．さらに，絞り込み得られた有力ソフトウェアのよりよい順位付けを提案することができれば，複数ソフトウェアを入力した場合の順位の向上が期待できる．また，再利用元の特定を行った後，どの程度ソースファイルを再利用し変更したのか，独自に新しいソースファイルをどの程度追加したかなどの調査を行うことが容易になると考えられる．

謝辞 本研究は JSPS 科研費（課題番号 JP25220003，JP26280021）の助成を受けたものです．

参考文献

[1] Dubinsky, Y., Rubin, J., Berger, T., Duszynski, S., Becker, M. and Czarnecki, K.: An Exploratory Study of Cloning in Industrial Software Product Lines, *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, pp. 25–34 (2013).

[2] Ebert, C.: Open Source Software in Industry, *IEEE Software*, Vol. 25, pp. 52–53 (online), DOI: doi.ieeecomputersociety.org/10.1109/MS.2008.67 (2008).

[3] Xia, P., Matsushita, M., Yoshida, N. and Inoue, K.: Studying Reuse of Out-dated Third-party Code in Open Source Projects, *Computer Software*, Vol. 30, No. 4, pp. 98–104 (2013).

[4] Kawamitsu, N., Ishio, T., Kanda, T., Kula, R. G., De Roover, C. and Inoue, K.: Identifying Source Code Reuse Across Repositories Using LCS-Based Source Code Similarity, *Proceedings of the 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, SCAM '14, Washington, DC, USA, IEEE Computer Society, pp. 305–314 (online), DOI: 10.1109/SCAM.2014.17 (2014).

[5] Indyk, P. and Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality, *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, New York, NY, USA, ACM, pp. 604–613 (online), DOI: 10.1145/276698.276876 (1998).

[6] 川満直弘，石尾　隆，井上克郎：LSH アルゴリズムを利用した類似ソースコードの検索，Vol. 2016-SE-191 (2016).

[7] Rubin, J., Czarnecki, K. and Chechik, M.: Managing Cloned Variants: A Framework and Experience, *Proceedings of the 17th International Software Product Line Conference*, pp. 101–110 (2013).

[8] Mohagheghi, P., Conradi, R., Killi, O. and Schwarz, H.: An empirical study of software reuse vs. defect-density and stability, *Proceedings of the 26th International Conference on Software Engineering*, pp. 282–291 (online), DOI: 10.1109/ICSE.2004.1317450 (2004).

[9] Kanda, T., Ishio, T. and Inoue, K.: Extraction of product evolution tree from source code of product variants, *Proceedings of the 17th International Software Product Line Conference*, Tokyo, Japan, ACM, pp. 141–150 (online), DOI: 10.1145/2491627.2491637 (2013).

[10] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multilinguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670 (online), DOI: 10.1109/TSE.2002.1019480 (2002).

[11] Sasaki, Y., Yamamoto, T., Hayase, Y. and Inoue, K.: Finding file clones in FreeBSD Ports Collection, *MSR* (2010).

[12] Inoue, K., Sasaki, Y., Xia, P. and Manabe, Y.: Where does this code come from and where does it go? – Integrated code history tracker for open source systems –, *Proceedings of the 34th IEEE/ACM International Conference on Software Engineering*, pp. 331–341 (online), DOI: 10.1109/ICSE.2012.6227181 (2012).

[13] Leskovec, J., Rajaraman, A. and Ullman, J.: *Mining of Massive Datasets*, Cambridge University Press (2011).

[14] 坂口雄亮：ソースファイル群の類似性を用いたソフトウェア再利用元の推定，修士論文，大阪大学 (2017).