論　文

大規模ソフトウェアシステムを対象としたファイルクローンの検出

佐々木裕介[†]　山本　哲男[††]　早瀬　康裕[†††]　井上　克郎[†]

File Clone Detection for a Large Scale Software System

Yusuke SASAKI[†], Tetsuo YAMAMOTO[††], Yasuhiro HAYASE[†††], and Katsuro INOUE[†]

あらまし　大規模なソフトウェア群中にはソースコードの等しいファイル，ファイルクローンが大量に存在する．発注先から納品されたソフトウェア製品や，新たに導入しようとしているオープンソースソフトウェアのシステム群に含まれるファイルクローンについて調べることは，システムの整理及び再構築の際やシステム群の全体像を把握するときなどに有用となる．そこで我々は大量のソースコードから高速にファイルクローンを検出できるツール FCFinder を開発し，大規模なソフトウェアシステムであり既存研究で多くのファイルクローンの存在が予測されている，FreeBSD Ports Collection に適用した．結果として，ports の約 68%のファイルがファイルクローンとなっていることが明らかになった．本論文では，FCFinder で用いているファイルクローン検出手法について説明し，FCFinder を様々なソフトウェア群に適用した結果について考察を行う．

キーワード　ファイルクローン，ハッシュ，べき乗則，オープンソースソフトウェア

1. まえがき

大規模なソースコード群からなるソフトウェアシステムではファイルの流用が頻繁に行われており，複数のプロジェクトにまたがるコードクローンが大量に存在することが予測される [1]．発注先から納品されたソフトウェア製品や，新たに導入しようとしているオープンソースソフトウェアのシステム群に含まれるファイルクローンについて調べることは，システムの整理及び再構築の際やシステム群の全体像を把握するときなどに有用となる．しかし，既存のコードクローン検出手法 [2], [3] はソースコードのミクロな部分に着目するものであり，大規模なソースコード群からなるソフトウェアシステム内にコードクローンがどの程度，どれくらいの規模で含まれているかを調査するにはコストが高すぎる．また，発見した大量のコードクローンをコード単位で整理・統合すると多大なコストを要する．

そこで我々は，より調査コストの低いファイル単位でのクローン，ファイルクローンを高速に検出するツール FCFinder を開発した [4]．ここでは，コメントを除外するなどの正規化を行ったときにソースコードが等しくなるファイルの関係をファイルクローン関係と呼び，それぞれのファイルをファイルクローンと呼ぶ．ファイルクローン関係は同値関係であり，その同値類となるファイル集合をファイルクローンセットと呼ぶ．ファイルクローンは既存のコードクローン検出ツール CCFinder [2] でも検出可能であるが，FCFinder は検出の対象をファイルクローンに限定しているため，より高速にファイルクローンを検出できる．ファイルクローンの分析結果は，

- 利用するシステム群の全体像を把握する
- 同様な修正が必要な箇所の発見を効率化する
- システムの整理や再構築の際の指標にする

などの用途に用いることができる．例えば，新たに利用しようとしているオープンソースソフトウェアのシステム群や発注先から納品されたソフトウェア製品などの一部で，

- どのようにサブシステムの間に依存関係があり，
- もし不具合が発見された場合には影響がどう波及するか

[†] 大阪大学大学院情報科学研究科，吹田市
Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan
[††] 日本大学工学部，郡山市
College of Engineering, Nihon University, Koriyama-shi, 963–8642 Japan
[†††] 筑波大学システム情報工学研究科，つくば市
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba-shi, 305–8573 Japan

といった情報を事前に調査しておくことができる．また，あるファイルで問題が発覚した場合には，同じファイルを探し修正する必要があるが，ファイルクローン情報に基づいて修正するファイルを知ることができる．更に，システムの整理や再構築を行う際には同じ部分をまとめる作業（リファクタリング）を行うことがあるが，そのための情報をファイルクローン分析の結果から得ることができる．ファイルクローンはファイルであるため，コード片であるコードクローンに比べ分析結果に基づいてリンクによりファイルを代替し整理するなどの応用を行いやすい．

本研究ではファイルクローンに関する以下の問題を詳しく調べる．

RQ1　ファイルクローンが大規模なソフトウェア群内にどれくらい含まれているのか

RQ2　それらのファイルクローン群はどのような性質をもっているか

本論文では，我々が開発した大規模なソースコードからファイルクローンを検出できるツール FCFinder について説明した後，FreeBSD Ports Collection（以降，ports と略記する）やそのソースコードアーカイブ，Linux Kernel，Open Office などのソースコードを対象として行ったファイルクローンの調査について記述する．ports は port と呼ばれるファイル群（ここではこれをプロジェクトと呼ぶ）の集まりであり，世の中で利用されている多くのソフトウェアのソースコードを利用している．また ports に含まれるプロジェクト間では多くのファイル共有が行われていることがあるが，その定量的，定性的な性質は知られていないため，解析対象としては非常に興味深い．更にファイルクローンの調査を行うことにより，ports 内のプロジェクトでもし不具合が発見された場合に，他のプロジェクトに影響がどう波及するかを予測することもできる．

2. では関連研究として既存のコードクローンに関する研究について取り上げる．**3.** では，開発したツール FCFinder がどのように高速なファイルクローン検出を実現しているかについて説明する．**4.** では，ports 内部に含まれるファイルクローンの割合や，その度数分布について調べた調査実験について記述する．**5.** で，本研究で対象とする問題についての考察を行った上で，我々の行った研究についてのまとめを行い，今後の課題について述べることとする．

2. 関 連 研 究

近年，ソースコード解析技術として，コピー&ペーストや類似ソースコード，コードクローンの検出技術が注目を集めている．特にソースコードの流用に関する情報は保守において重要であり，剽窃検出，関連バグの発見など様々な用途で用いられている [3], [5], [6]．コードクローン検出手法として，トークンベースの検出手法や関数を単位とした検出手法などが研究されており，そのためのツールもいくつか開発されている [2], [7]．Mayrand らは 21 種類のメトリクスを用い，類似した関数単位での検出を試みた [3]．ここでいうメトリクスとは，ソフトウェアの評価するための尺度であり [8]，特にソースコードから抽出される特徴に対する数値を指す．また，彼らはオブジェクト指向のプログラムに対する類似クラスの検出を行った [9]．

これらの研究では比較的大きさの限られたソフトウェアに対して解析を行っているが，Zhenmin らはより規模の大きいソースコードからコピー&ペーストを検出し，それに関するバグを発見するツール CP-Miner [6] を開発し，FreeBSD のソースコードに適用した．

更に大規模なソースコードに対するコードクローン検出として，Livieri らは分散処理を用いたコードクローン検出ツール D-CCFinder を開発し，実験を行った [1]．Livieri らの研究によって，大規模なソースコード群からなるソフトウェアシステムでは，複数のプロジェクトにまたがってソースコードの等しいファイルが存在することが判明した．しかし我々の調べた限りでは，大規模なソフトウェア群中からソースコードの等しいファイルを検出するような研究は行われていない．

3. FCFinder

この章では，我々が大規模なソースコードを対象としたファイルクローンの検出を高速に行うために開発した，File Clone Finder（以下，FCFinder）について説明する．

FCFinder は複数のプロジェクトのソースコード群を入力として受け取り，プロジェクト内のファイルクローン関係やプロジェクトをまたぐファイルクローン関係を検出し，ファイルクローンセットとしてデータベースへ出力する．データベースに出力したファイルクローンセットに関する情報は，他のツールから呼び出して利用できるようになっている．処理の概要を

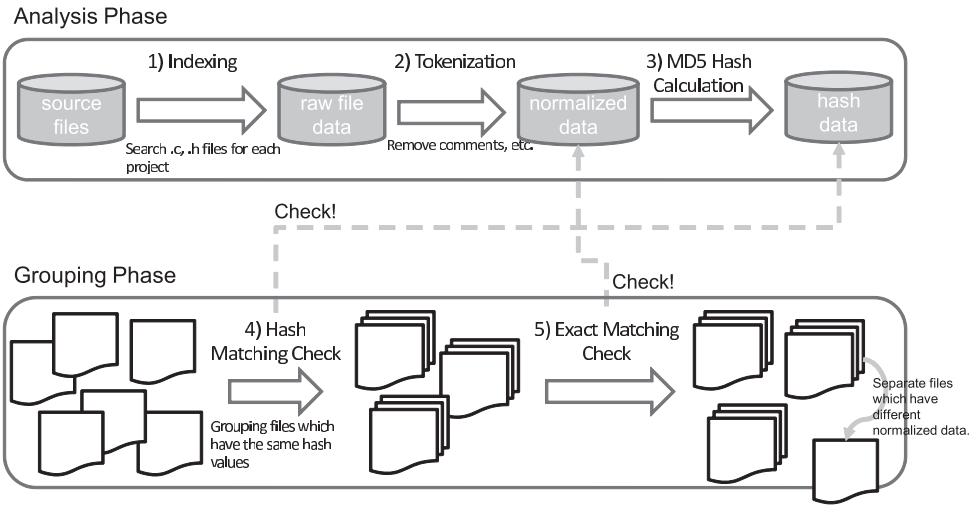


図 1　FCFinder
Fig. 1　FCFinder.

図 1 に示す．以下，図 1 に示す各フェーズを順に説明する．

3.1　Analysis Phase

3.1.1　Indexing

今回解析対象とする .c，.h ファイル及びプロジェクトを特定する．更に特定されたファイル群と各ファイルが属しているプロジェクトをデータベースへ格納する．

3.1.2　Tokenization

FCFinder はコメント文やインデントなど細かな変更を無視するために，コメント文の削除及び字句解析によりファイルを正規化する．このフェーズではまずファイル中のコメント文を削除し，その後に字句解析によりソースコードをトークンへ切り分ける．ソースコードをトークンへ切り分けることにより，インデントなどの連続する空白や改行記号は，同一の空白記号へ置き換えられる．

またこのフェーズでトークン数が 15 より小さなファイルを取り除く．トークン数が 15 より小さなファイル 100 ファイルを ports から無作為に選出し調査を行った結果，変数宣言や関数宣言，マクロ定義，ファイルのインクルードなどの処理しか記述されていなかった．このため本ツールではトークン数が 15 より小さなファイルをソースコードとしてあまり意味をもたないものと判断し，検出の対象外として解析対象から外している．

3.1.3　MD5 Hash Calculation

正規化を終えた解析対象となるすべてのファイルそれぞれに対して，MD5 [10] によりハッシュ値を算出する．ファイルから算出されたハッシュ値は，データベースへ格納しておく．

3.2　Grouping Phase

このフェーズでは，**3.1.2** で正規化した内容が同一であるファイル集合を，ファイルクローンセットとしてグループ化する．

すべてのファイルの組合せについてファイル内容を全比較すると時間がかかる．そこでファイル内容の全比較はハッシュ値の一致するファイルのみに対して行う．ハッシュ値が異なるファイルはそのファイル内容も必ず異なっているため，内容が同一であるファイル集合を検出する際には，ハッシュ値が異なるファイル同士の比較は不要となる．

3.2.1　Hash Matching Check

ハッシュ値の同じファイルをグループ化する．

3.2.2　Exact Matching Check

3.2.1 で作成したグループを，**3.1.2** で正規化した内容が同一であるグループに分割する．分割によって作成されたグループを，ファイルクローンセットとしてデータベースに格納する．

4.　実　　験

FCFinder を用いて, [1] の実験で用いられた，2007 年 9 月 6 日時点での ports のうち，拡張子が .c と .h

に対してファイルクローンの検出を行った．実験にはCPU 2.50 GHz，メモリ 16 GByte を積んだ 64 bit マシンを利用した．ファイルクローンの検出で FCFinder が使用した最大メモリ量は 4 GByte 未満であった．

4.1 ソースコードの収集方法

前述のように，本実験では ports を利用してソースコードの収集を行った．ports は，port と呼ばれる単位で，システムへのインストールやアンインストールを行う．この port に含まれるファイル群をここではプロジェクトと呼ぶ．プロジェクトは，カテゴリーに分けて管理されている．一般に，ports のプロジェクトをインストールする際には，ソースコードのアーカイブをインターネット上の外部サイトから取得し，必要に応じてカスタマイズやコンパイルを行った上で，その全部または一部をシステムにインストールする．そのため，異なったカスタマイズを行ったり，インストールする部分が異なるなどといった理由で，複数のプロジェクトが同一のアーカイブを用いる場合が数多く存在する．

例えば図 2 に示すように，アーカイブ F に対するバイナリがプロジェクト F1，プロジェクト F2 として別々の場所に展開，利用されることがある．この場合，F1，F2 に対して生成されるバイナリは別のものであることが多い．

本実験におけるソースコードの収集では，すべてのプロジェクトごとにソースコードのアーカイブを取得し，個別に展開を行った．そのため，異なるプロジェクトのソースコードとして，同一のソースコードアーカイブが多重に収集されている．

収集した時点での，カテゴリー数及びプロジェクト数を表 1 に示す．表 1 には，収集の結果得られた .c 及び .h ファイルの数とデータ量も記してある．なお，ファイル収集時の make config のデータはデフォルト値である．また移転によりダウンロードできなかったなどの理由で入手が困難だったファイルは除外した．

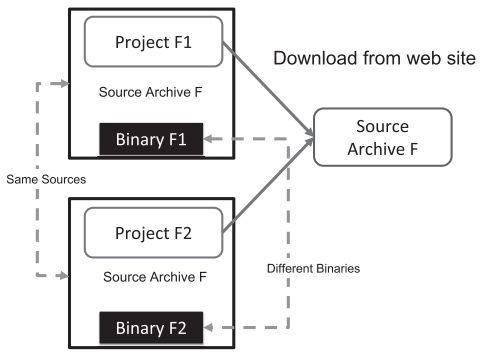


図 2 ソースコードアーカイブのダウンロード及び展開
Fig. 2 Source archives downloading and extracting.

4.2 解析対象に対する事前調査

ファイルクローンの分析を行う前準備として，解析対象に含まれているファイルについて簡単な考察を行うために，ファイルサイズとプロジェクトごとのファイル数について度数分布を計測した．

図 3 はファイルごとのサイズを集計した度数分布グラフである．X 軸はファイルサイズ，Y 軸はファイル数を示し，両対数軸をとっている（以降のグラフでは X 軸 Y 軸ともに対数軸をとっているものとする）．このグラフではサイズが増えるほどファイル数が線形に減少しており，べき乗則 [11], [12] と呼ばれる法則が成り立っている．どれくらいべき乗則に従っているか確認するため，ファイルサイズの累積度数分布に対し線形回帰分析を行ったところ，自由度調整済み寄与率 R^{*2} は 0.9024 となった（以降，自由度調整済み寄与率 R^{*2} は累積度数分布に対して計算したものとする）(注1)．一般的にこの値は 0.8 程度あれば回帰モデル

表 1 解析対象の構成
Table 1 Mining target characteristics.

Number of files (.c, .h)	1,355,098
Number of projects	7,209
Number of categories	45
Total size	11.2 GByte

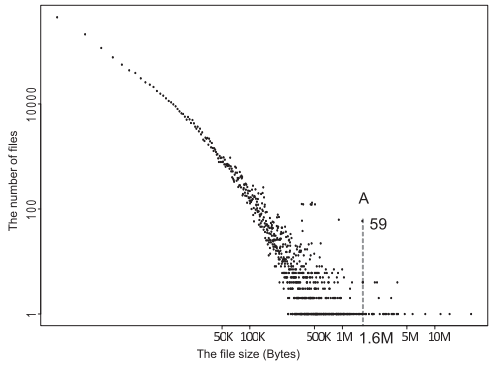


図 3 ファイルごとのサイズ度数分布
Fig. 3 File size distribution.

(注1)：自由度調整済み寄与率 R^{*2} とは，あるデータについて線形回帰分析を行ったとき，そのデータが回帰分析によって得られたモデルにどれくらい従っているかを示す数値であり，0 から 1 の値をとる [13] すべてのデータが回帰モデルに従っていれば R^{*2} は 1 になる．

に従っているといえる [14], [15]．そのため今回調査した ports におけるファイルサイズの度数分布は，高い割合でべき乗則に従っているといえる．

しかしその一方で，比較的ファイルサイズの値が大きい範囲で線形回帰モデルから外れている部分がいくつか見つかった．例えば図 3 A の中でもサイズが約 1.6 MByte であるファイル数は 59 という大きな数値を示している．より詳しく調べてみたところ，サイズが約 1.6 MByte であるファイルはすべて互いにファイルクローンとなっており，PHP5 の time zone に関するファイルであった．

図 4 はプロジェクトごとに .c，.h ファイル数を集計した度数分布グラフである．X 軸 Y 軸ともにどちらも対数軸であり，X 軸はプロジェクトごとに含まれている .c，.h ファイルの数を，Y 軸はプロジェクト数を示す．自由度調整済み寄与率 R^{*2} を調べたところ 0.8868 となり，高い割合でべき乗則に従っていることが分かった．

しかしこのグラフについても X の値が大きい範囲で線形回帰モデルから大きく外れている部分がいくつか見つかった．図 4 B では，ファイル数が 1,316 のプロジェクトが 59，ファイル数が 1,070 のプロジェクトが 61 存在していた．より詳細な調査を行ったところ，これらのほとんどは PHP4/PHP5 の拡張ライブラリを生成するためのプロジェクトであり，互いに大量のファイルクローンを共有していた．図 4 B に含まれるプロジェクトの具体例としては www/PHP5-tidy などがある．これらのプロジェクトは同一のソースコードアーカイブを利用しているが，それぞれソースコードアーカイブに含まれるファイルのうち異なる部分を用いているため，インストール時には異なるバイナリが生成される．

また図 4 C にはファイル数が 6,506 のプロジェクトが 14 存在していた．これらはアーキテクチャごとに用意された RTEMS OS のクロスコンパイラであり，多くのファイルを共有していた．これらのプロジェクトは同じソースコードアーカイブを利用しているが，インストール時にはそれぞれソースコードに異なるカスタマイズを加えコンパイルしている．これらのことから，収集したソースコードの中には，多くのファイルクローンが存在することが予測される．

また，図 5 に示すのはプロジェクトごとのファイルサイズ合計を集計した度数分布グラフである．X 軸はそのプロジェクトが含んでいる .c，.h ファイルのサイズ合計を表す．Y 軸はサイズの等しいプロジェクト数を表す．自由度調整済み寄与率 R^{*2} は 0.8214 となった．このことから，プロジェクトごとのファイルサイズについてもべき乗則が成り立っていると考えられる．

図 5 にも D のようにいくつか回帰モデルから外れている部分があるため，サイズが 50 MByte 以上のプロジェクトに対して詳細な調査を行った．表 2 はプロジェクトサイズが大きいものを，上から 10 プロジェクト表示したものである．この表に示されている emulators/xmame は emulators/xmess の後継プロジェクトであり同一組織で作られている．このように同じ組織で作成されたプロジェクトが順位の近い位置に見つかるものが，ほかにもいくつか発見された．このことから，対象とするファイル群の中には，同一組織によって作られたバージョンの異なるプロジェクトなどが存在することが予測される．

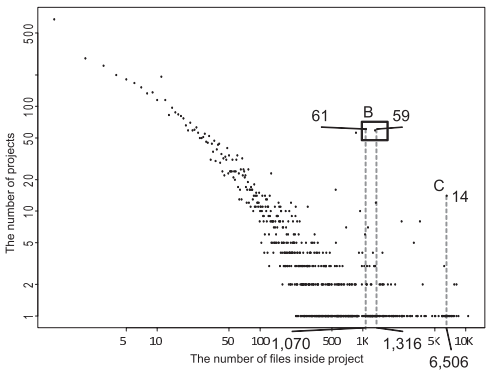


図 4　プロジェクトごとに含まれるファイル数の度数分布
Fig. 4　Distribution of the number of files inside project.

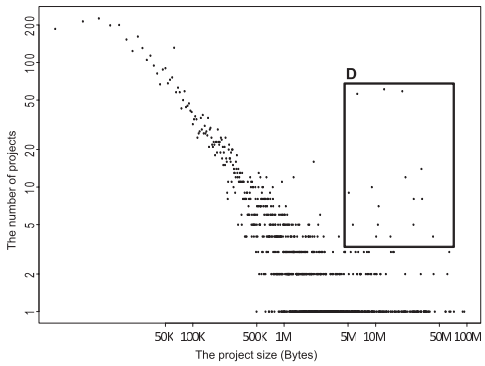


図 5　プロジェクトごとのサイズ度数分布
Fig. 5　Project size distribution.

表 2　プロジェクトサイズ上位 10 件
Table 2　Top 10 project sizes.

順位	project	project size (MByte)
1	palm/prc-tools	93.8
2	lang/nhc98	89.4
3	x11/xorg-clients	72.0
4	net/nxserver	66.0
5	net/vnc	64.1
5	x11/xorg-manpages	64.1
5	x11/xorg-libraries	64.1
5	devel/imake-6	64.1
9	emulators/xmess	61.1
9	emulators/xmame	61.1

4.3　ファイルクローンの解析

我々はファイルクローンについての解析を行うため，FCFinder を用いてファイルクローンを検出した．ファイルクローン数は 915,409，ファイルクローン合計サイズは 7.6 GByte となった．これは全体の約 68%がファイルクローンであることを示している．ports ではプロジェクトの大部分が共通のファイルを利用しているためと考えられる．

また見つかったファイルクローンのうち，コメントやインデントなどに違いのあるファイルの数は，全ファイルクローン中の 26.01%（238,154 ファイル）であった．具体的な事例としては以下のようなものがある．

- cad/cider と cad/spice の間に，インデントの異なるファイルクローンが見つかった．
- database/firebird-server と database/firebird-client の間に，異なるソースコード自動生成プログラムにより生成されたファイルクローンが見つかった．

cad/cider と cad/spice はともに回路・デバイスのシミュレーションを行うソフトウェアである．Firebird はオープンソースのデータベースであり，見つかったファイルクローンのコメント部分には，そのソースコードを生成したソースコード自動生成プログラムが記述されていた．

ファイルクローンが特殊なファイルサイズの分布をもっていないか確認するため，ファイルクローンのみを抽出しファイルサイズについての度数分布グラフを図 6 に構築した．この図 6 は，図 3 に比べあまり変化はない．自由度調整済み寄与率を計算したところ 0.9602 と上がっており，図 3 と比較するとよりべき乗則に近づいた形となっている．

次にファイルクローンのみを除外したファイルサイズについての度数分布グラフを図 7 に示す．図 6 より更に自由度調整済み寄与率が 0.9902 と上がっているが，こちらもグラフ全体の傾向はあまり変化がない．

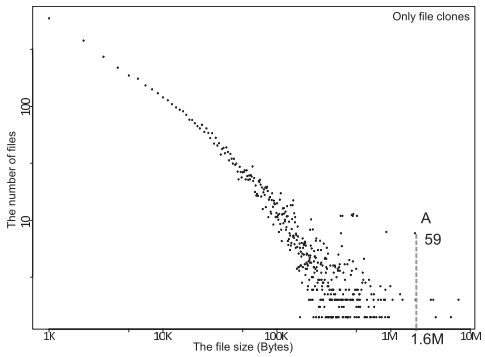


図 6　ファイルごとのサイズ度数分布（ファイルクローンのみ）
Fig. 6　File size distribution. (file clones only)

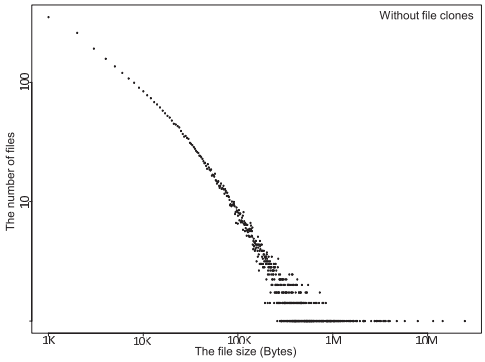


図 7　ファイルごとのサイズ度数分布（ファイルクローン除く）
Fig. 7　File size distribution. (without file clones)

これらのことから，ファイルクローンについても一般的なソースコードと同じように，べき乗則という性質が成り立っていることが分かる．

ファイルクローンについてより詳しい解析を行うため，図 8，図 9 の二つの度数分布グラフを作成した．

図 8 はファイルクローンセットごとにそのファイルクローンが全ソースコード中に登場する回数を集計した度数分布グラフである．X 軸は一つのファイルクローンセット中に含まれるファイルクローン数を，Y 軸はファイルクローンセット数を示す．

全体として，ファイルクローン数が大きいファイルクローンセットほど数が少ないことが分かる．自由度調整済み寄与率 R^{*2} は 0.8508 となり，ファイルクローンセット当りのファイルクローン数についても高い割合でべき乗則が成り立っていることが判明した．

しかし外れ値となっている部分がいくつか見られる．

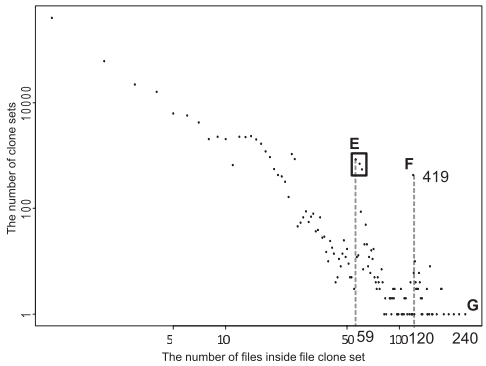


図 8　1 ファイルクローンセット当りに含まれるファイルクローン数の度数分布
Fig. 8　Distribution of population of file clone set.

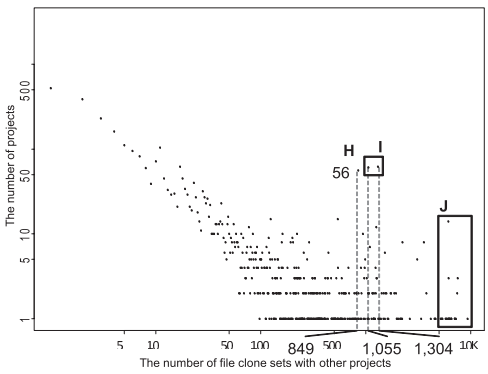


図 9　1 プロジェクト当りに含まれるファイルクローン数の度数分布
Fig. 9　Distribution of file clones in project.

図 8 E では図 4 B に対応するファイル群が見つかった．例えばファイルクローン数が 59 であるファイルクローンセットの中には，www/PHP5-tidy に含まれるファイルが確認された．

図 8 F では，ファイルクローン数が 120 の部分だけファイルクローンセット数が 419 となっている．より詳しく調査してみたところ，これらのファイルクローンセットは 419 ファイルを一組として PHP4，PHP5 関連の 120 箇所のプロジェクトで利用されていた．これら 120 のプロジェクトのうち，61 が PHP4 をベースとしているプロジェクト，59 が PHP5 をベースとしたプロジェクトであった．

加えて図 8 G に示すファイルクローン数が 240 であるファイルについて調査してみたところ，図 8 F に出現する PHP4/PHP5 関連の 419 ファイルとセットになっていることが分かった．これらのファイルは 1 プロジェクトにつき二つずつ含まれていたため，出現回数も 2 倍になっていた．

図 9 は，プロジェクトごとに他のプロジェクトとファイルクローンをいくつ共有しているかを集計した度数分布グラフである．X 軸は 1 プロジェクト当りに含まれる，他のプロジェクト内のファイルとファイルクローン関係となっているファイルクローンセットの数を表す．Y 軸はプロジェクト数を表す．

図 9 の全体の特徴として，他プロジェクトと共有のあるファイルクローン（以下，共有されているファイルクローン）数が多いプロジェクトほど，数が少ないことが分かる．自由度調整済み寄与率 R^{*2} は 0.8263 となった．

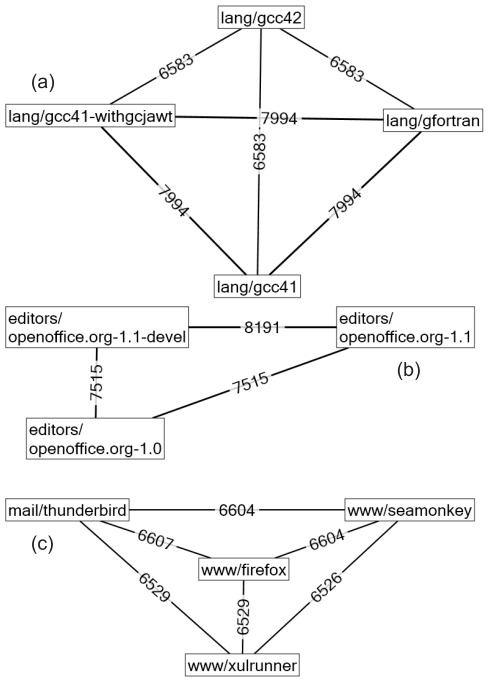
図 9 H では，共有されているファイルクローン数が 849 のプロジェクトが 56 見つかった．これらはマルチメディアフレームワークである GStreamer 用プラグインのソースコードアーカイブをベースとして作成されたプロジェクトであり，PHP4/PHP5 と同様に大量のファイルを共通して利用していた．

更に図 9 I では，共有されているファイルクローンが 1,304 のプロジェクトが 59，共有されているファイルクローンが 1,055 のプロジェクトが 61 見つかった．これらはすべて PHP4/PHP5 のソースコードアーカイブを利用しており，図 4 B 及び図 8 E と対応関係にあった．図 9 I に含まれるプロジェクトには **4. 2** で述べた www/PHP5-tidy プロジェクトも含まれていた．www/PHP5-tidy プロジェクトの全ファイル数は 1,316 であったが，このうち共有されているファイルクローン数は 1,304 であった．一部のファイルが共有されていないのは，同じソースコードアーカイブを利用していても，一部のファイルに修正を行っているためと考えられる．

共有されているファイルクローン数が 5,000 以上の図 9 J でも，共有されているファイルクローン数が多いにもかかわらず，プロジェクト数が突出して多い部分がある．より詳細な調査を行ったところ，同じ組織で作成されており，共有されているファイルクローン数が近いプロジェクトが複数見つかった．表 3 は，図 9 上の H に示す範囲のプロジェクトを共有されているファイルクローン数が大きいものから順に表示したものである．例えば共有されているファイルクローン数が 7,415 から 7,033 のプロジェクトには，OpenOffice.org に関連するものが集中している．ほかにも Mozilla.org のプロジェクトが集中している箇所や GDB 関連のプロジェクトが集中している箇所，

表 3　共有されているファイルクローンの多いプロジェクト上位 10 件
Table 3　The number of file clone sets with other projects top 10.

順位	project	# of file clones
1	palm/prc-tools	9,332
2	lang/gfortran	7,500
2	lang/gcc41-withgcjawt	7,500
2	lang/gcc41	7,500
5	editors/openoffice.org-1.1-devel	7,415
5	editors/openoffice.org-1.1	7,415
7	editors/ooo-build	7,232
8	editors/openoffice.org-2.0	7,062
9	editors/openoffice.org-1.0	7,051
10	editors/openoffice.org-2.0-devel	7,033



- Nodes show the projects.
- Edges between projects show the number of file clone relations between two projects.

図 10　プロジェクト間のファイルクローン関係
Fig. 10　Project relations with file clones.

PHP 関連のプロジェクトが集中している箇所などが複数発見された．

図 10 は図 9 H 中に含まれるプロジェクト間のファイルクローン共有関係を示している．各ノードはプロジェクトを，エッジはそれぞれのプロジェクトに含まれているファイル間に存在している，ファイルクローン関係の総計を示している．

図 10 (a) は GNU Compiler Collection プロジェクトのうち gcc41 と gcc41-withgcjawt (java)，gcc42, gfortran を示す．これらはおおよそ 7,000 のファイルクローン関係を有している．図 10 (b) は OpenOffice.org の 3 プロジェクトを示しており，7,515 から 8,191 のファイルクローン関係を有している．図 10 (c) では Thunderbird と Firefox，Firefox-devel，Seamonkey (integrated network tool)，XULRunner (application development kit) がおよそ 6,000 のファイルクローン関係を有していることを示している．これらは異なるアーカイブをダウンロードして利用しているが，同じ組織内で管理されている．

表 4　ports で利用されているアーカイブ群に対するファイルクローン検出結果
Table 4　File clone detection for archive files used in ports.

# of archives	11,020
# of files	1,313,187
# of file clones	563,547
% of file clones	42.91

4.4 考　　察

図 10 から，多くのプロジェクトの間でファイルクローン関係をもつファイルが存在することが分かる．また図 9 に示すとおり，プロジェクト間で共有されているファイルクローン数は大部分がべき乗則に従っているが，一部ファイルクローン数が大きく突出している部分が見受けられる．そこでファイルクローン数が突出している部分についてより詳細に調べたところ，同じソースコードアーカイブがプロジェクトごとに何度も展開され，一部修正して利用されているケースがいくつか見つかった．

4.4.1　ports が利用しているソースコードアーカイブの分析

4.1 で示したソースコードの収集方法では，見つかるファイルクローンが非常に多くなる．そこで我々は，各プロジェクトが必要に応じてダウンロードを行う対象であるソースコードアーカイブ（以降，単にアーカイブと呼ぶ）自体についても分析を行った(注2)．取得されたアーカイブ数は 11,020 で，その中に含まれるファイル数は総計 1,313,187 であった（表 4）．ファイル数が非常に多いのは，前述の実験で収集したとき（表 1）よりも ports のプロジェクト数が増えたためである．

表 4 によるとファイルクローンの割合が 42.91%と非常に高くなっているが，これは同じソフトウェアの

(注2)：この実験では 2010 年 11 月 4 日現在 ports で利用されているアーカイブ群を対象とした．

アーカイブでも，プロジェクトによっては異なるバージョンをダウンロードするケースがあることが大きな要因として挙げられる．例えば，gcc についてはバージョン 2.7.2 及び 4.1.0，4.4.2 のアーカイブが含まれていた．なお，最も多くのファイルクローンを含んでいたのは gcc 4.4.2 であった．

図 11 は，ファイルクローンセットごとにそのファイルが全ソースコード中に登場する回数を集計した度数分布グラフである．X 軸は一つのファイルクローンセット中に含まれるファイルクローン数を，Y 軸はファイルクローンセット数を示す．図 8 と比較すると外れ値が非常に減っており，自由度調整済み寄与率も高い ($R^{*2} = 0.9722$)．

図 12 は，アーカイブごとに他のアーカイブとファイルクローンをいくつ共有しているかを集計した度数分布グラフである．X 軸は 1 アーカイブ当りに含まれる，他のアーカイブ内のファイルとファイルクローン関係となっているファイルクローンセットの数を表す．Y 軸はアーカイブ数を表す．こちらも図 9 と比較すると外れ値が非常に減っており，自由度調整済み寄与率も高い ($R^{*2} = 0.955$)．

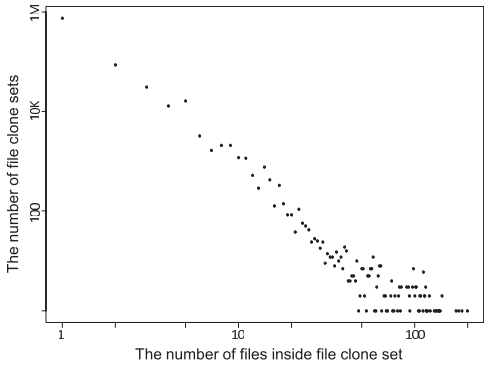


図 11　1 ファイルクローンセット当りに含まれるファイルクローン数の度数分布
Fig. 11　Distribution of population of file clone set. (archives used in ports)

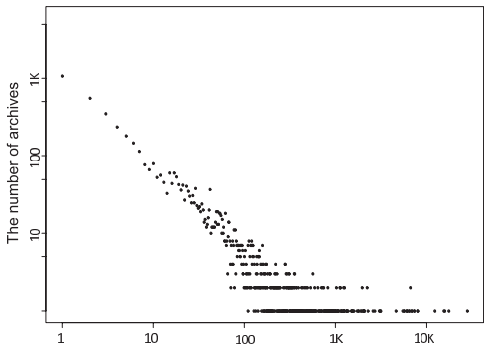


図 12　1 アーカイブ当りに含まれるファイルクローン数の度数分布
Fig. 12　Distribution of file clones in archives. (archives used in ports)

4.4.2　その他のソフトウェアシステムの分析

また，検出されるファイルクローンが ports ほど多くないと考えられる次のソフトウェアシステム群についてもファイルクローンの分析を行った．

- Linux Kernel 2.6.35.7
- OpenOffice 3.2.1

それぞれの構成を表 5 に，ファイルクローン検出結果を表 6 に示す．表 5 の file types は，解析対象としたファイルの拡張子を示すものである．

表 6 のとおり，Linux Kernel 2.6.35.7 でも割合は少ないがファイルクローンが検出された．ファイルクローンセットごとにそのファイルが全ソースコード中に登場する回数を集計したところ，そのべき乗分布に対する自由度調整済み寄与率は 0.8414 となった．

OpenOffice 3.2.1 でもファイルクローンが検出された（表 6）．ファイルクローンセットごとにそのファイルが全ソースコード中に登場する回数を集計したところ，そのべき乗分布に対する自由度調整済み寄与率は 0.8301 となった．

これらの調査から，ports 以外のプロジェクトにもファイルクローンが含まれており，分布もべき乗則に従っていることが判明した．

4.4.3　アーカイブ間で共有されているファイルクローン

また **4.4.1** の調査により，ports でダウンロードされたアーカイブ群について，異なるアーカイブ間で共有されているファイルクローンがいくつか見つかった．

(1)　データの圧縮/展開を行うライブラリ zlib のファイルが Mozilla や Python などをはじめとする多

表 5　解析対象の構成
Table 5　The target characteristics.

software system	file types	# of files
Linux Kernel 2.6.35.7	.c, .h	26,247
OpenOffice 3.2.1	.c, .h, .java	6,858

表 6　ファイルクローン検出結果
Table 6　File clone detection results.

software system	# of file clones	% of file clones
Linux Kernel 2.6.35.7	332	1.26
OpenOffice 3.2.1	305	4.45

くのアーカイブで利用されている．

(2) libpng や libjpeg といった画像処理ライブラリのファイルが Mozilla や OpenCV などをはじめとする多くのアーカイブで利用されている．

(3) Mozilla の JavaScript 実行エンジンである OSSP のソースコードが FreeSWITCH でも使われている．

(4) Boehm GC というガーベッジコレクレターが Mozilla や GCC，Kaffe など多くのアーカイブで利用されている．

(5) libffi のファイルが gcc や smalltalk などで利用されている．

これらの共有が起こっている理由として，例えば (1) に関しては，Firefox などのウェブブラウザではプラグインインストールの際に zlib を利用しており，また，Python ではデータ圧縮/展開の機能を提供するために zlib を用いているためであった．その他の共有も，システムの一部の機能を実装するために他のシステムやライブラリを利用していることが理由で起こっていた．

4.4.4 解析精度

検出されるファイルクローンセットが正しいものであるかどうか，実際に検出されたファイルクローンセットについて 50 セット確認を行ったが，すべて **1.** において定義したとおりのファイルクローンセットであった．

また CCFinder [2] が検出したファイルクローンのうち無作為に選んだ 50 個のファイルクローンの対について FCFinder でも検出可能か調査したが，すべてのクローンペアが検出できることを確認した．

以上の調査では，FCFinder は再現率・適合率ともに 100%であることが確認された．しかしながらより厳密に解析精度を評価するには，今後もサンプル数を増やしていく必要がある．

4.4.5 解析速度

CCFinder では 10.8 GByte のソースコードを解析する場合，1CPU (2.8 GHz)，メモリ 2 GByte を積んだ PC で約 40 日を要し．また同じソースコードを D-CCFinder で解析するのに PC 80 台 (CPU 1.0 GHz) で約 51 時間かかっている [1]．

今回の実験では，データベースアクセスも含めてすべての解析を 1CPU (2.50 GHz)，メモリ 4 GByte で約 17.16 時間で終えている（表 7）．実行環境が異なるため厳密な比較は困難であるが，CCFinder と比較すると約 50 倍以上の速度である．D-CCFinder と比べても約 3 倍の速度向上である．

表 7 解析時間
Table 7 Analysis time.

Step	Time
Indexing	9.42h
Tokenization	
MD5 Hash Calculation	1.13h
Hash Table Creation	
Exact Matching Check	6.61 h
Total	17.16 h

解析速度に関して改善すべき点としては，データベースのアクセス時間削減が挙げられる．データベースへのアクセス回数はキャッシュを有効に利用することで削ることができるが，使用できるキャッシュには限りがある．FCFinder のキャッシュの利用の仕方には改善の余地があり，今後も速度向上が見込まれる．

また表 7 に示すように，今回の実験では Exact Matching に 6.61 時間を費やしているが，今のところ MD5 値の衝突は起こっていない．よって MD5 値の比較だけでも高い精度を出すことは可能だと予測されるが，**4.4.4** のような解析精度を維持するためには Exact Matching は必須である．

5. むすび

1. において述べた問題に対して得られた考察を述べる．

まず RQ1 について，今回対象とした ports には，ファイルクローン数 915,409（合計サイズ 7,6 GByte）が含まれていることが確認された．これは全体の約 68%であり，ソフトウェアパッケージ内に含まれるファイルのほとんどがファイルクローンであることを示している．

RQ2 については以下のような結果が得られた．

- ports から検出されたファイルクローンのうち 26.1%がコメントやインデントに違いの見られるものであった．
- またファイルサイズの分布については，ファイルクローンのあるなしに関係なくべき乗則に従う傾向があることが分かった．

ファイルクローンはコードクローンに比べ，検出が容易であり，応用も行いやすい．その一方でファイルクローンに関する研究はあまり行われてこなかったため，今後はファイルクローンについてより詳しく研究し，FCFinder を使った大規模パッケージの管理や品質評価，FCFinder より使い勝手の良い検索システム

の開発といった応用につなげていきたい．

謝辞　本論文の執筆中，Simone Livieri 氏，市井誠氏に多くの有益なコメントを頂いた．本研究は一部，文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた．また，日本学術振興会科学研究費補助金基盤研究（A）（課題番号：21240002）の助成を得た．

文　献

[1] 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,” 29th IEEE International Conference on Software Engineering, pp.106–115, Minneapolis, MN, 2007.

[2] T. Kamiya, S. Kusumoto, and K. Inoue, “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, 2002.

[3] J. Mayrand, C. Leblanc, and E.M. Merlo, “Experiment on the automatic detection of function clones in a software system using metrics,” 12th IEEE International Conference on Software Maintenance, pp.244–253, Monterey, CA, 1996.

[4] Y. Sasaki, T. Yamamoto, Y. Hayase, and K. Inoue, “Finding file clones in FreeBSD ports collection,” Proc. 7th IEEE Working Conference on Mining Software Repositories, pp.102–105, ACM, May 2010.

[5] R. Koschke, et.al, 3rd International Workshop on Software Clones, Kaiserslautern, Germany, 2009. http://www.informatik.uni-bremen.de/st/IWSC/

[6] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, “CP-Miner: Finding copy-paste and related bugs in large-scale software code,” IEEE Trans. Softw. Eng., vol.32, no.3, pp.176–192, 2006.

[7] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, “Comparison and evaluation of clone detection tools,” IEEE Trans. Softw. Eng., vol.33, no.9, pp.577–591, 2007.

[8] N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd ed., Course Technology PTR., 1998.

[9] E. Merlo, G. Antoniol, M.D. Penta, and V.F. Rollo, “Linear complexity object-oriented similarity for clone detection and software evolution analyses,” 20th IEEE International Conference on Software Maintenance, pp.412–416, Illinois, CA, 2004.

[10] R. Rivest, “The MD5 message-digest algorithm,” RFC 1321 (Informational), April 1992. http://www.ietf.org/rfc/rfc1321.txt

[11] M. Mitzenmacher, “A brief history of generative models for power law and lognormal distributions,” Internet Mathematics, vol.1, no.2, pp.226–251, 2003.

[12] G. Concas, M. Marchesi, S. Pinna, and N. Serra, “Power-laws in a large object-oriented software system,” IEEE Trans. Softw. Eng., vol.33, no.10, pp.687–708, 2007.

[13] 久米　均，飯塚悦功，入門統計的方法 2 回帰分析，岩波書店，1987.

[14] P. Louridas, D. Spinellis, and V. Vlachos, “Power laws in software,” ACM Trans. Softw. Eng. Methodol., vol.18, no.1, pp.1–26, 2008.

[15] M.E.J. Newman, “Power laws, pareto distributions and Zipf’s law,” Contemporary Physics, vol.46, pp.323–351, Dec. 2005.

（平成 22 年 8 月 2 日受付，23 年 4 月 14 日再受付）

佐々木裕介

平 21 阪大・基礎工・情報卒．平 23 同大大学院博士前期課程了．修士（情報工学）．在学中ソフトウェア類似性に関する研究に従事．現在，富士通研究所に勤務．

山本　哲男　（正員）

平 9 阪大・基礎工・情報卒．平 14 同大大学院博士課程了．同年科学技術振興機構研究員．平 16 立命館大学・情報理工学部情報システム学科・講師．平 20 同大准教授．平 23 日本大学・工学部情報工学科・准教授．博士（工学）．ソフトウェア開発支援ツールの研究に従事．情報処理学会，IEEE 各会員．

早瀬　康弘

平 14 阪大・基礎工・情報卒．平 19 同大大学院博士課程了．同年同大特任助教．平 22 東洋大学・総合情報学部・助教．平 23 筑波大学・システム情報工学研究科・助教．オープンソースソフトウェア開発，ソフトウェア保守の研究に従事．博士（情報科学）．情報処理学会，IEEE Computer Society 各会員．

井上　克郎　（正員：フェロー）

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