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.
SS2001-15(2001-07)

クローン検出ツールを用いたソフトウェアシステムの類似度調査

山本 哲男† 松下 誠† 神谷 年洋‡ 井上 克郎†

† 大阪大学大学院基礎工学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3
Phone: 06-6850-6571 Fax: 06-6850-6574

‡ 科学技術振興事業団 若手個人研究推進事業

E-mail: t-yamamt@ics.es.osaka-u.ac.jp

あらまし 二つのソフトウェアシステムが与えられた時，そのシステムの間の違いはどれぐらいあるのか，客観的に知ることは重要である．しかしながら，二つのシステムのからそれらの相違点を定量的な値を計算することは容易ではなかった．本稿では，クローン検出ツール CCFinder を用いたソフトウェアシステム間の類似度メトリクス CSR(Corresponding Source-line Ratio) を提案する．類似度 CSR を種々の OS に適用した．また，それらの類似度からクラスタ分析を行い樹状図を作成し，CSR が OS の分類を正しく行えているか検証を行った．さらに，CSR と開発期間の相関を求め，高い相関があることが分かった．

キーワード コードクローン，ソフトウェアメトリクス，類似度

Similarity Metric CSR Using Code Clone Detection Tool

Tetsuo Yamamoto† Makoto Matsushita† Toshihiro Kamiya‡ Katsuro Inoue†

† Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka,
Osaka 560-8531, Japan
Phone: 06-6850-6571 Fax: 06-6850-6574

‡ PRESTO, Japan Science and Technology Corporation

E-mail: t-yamamt@ics.es.osaka-u.ac.jp

Abstract

It is important to know a differrence between two software systems objectively. However, it is hard to compute a quantitative difference from these systems. In this paper, we propose a similarity metric CSR(Corresponding Source-line Ratio) using code clone detection tool CCFinder. We apply CSR to many OSes, and construct an OS genealogy using cluster analysis. As a result, we can classify OSes using CSR. We also find out that there is a high correlation between CSR and development period.

key words Code Clone, Software Metrics, Similarity

1 はじめに

二つのソフトウェアシステムが与えられた時，そのシステムの間の違いはどれぐらいあるのか，客観的に知ることは重要である．いくつかあるシステムのバージョンの間の相違の度合を調べることによって，システムの保守の様子や進化の度合を知ることができる．さらにシステムを改変する際の有益な指針にもなる．例えば，プログラムの大幅な改変を行うよりも，新たに作り直す方がコストが安くなる場合がある．

システムの改変時にドキュメントが作成されていれば，それを手がかりにして，相違点を知ることは可能であろうが，定量的な値を得ることは容易ではなかった．システムが小規模で，全体の構造を人間が容易に把握できる場合は，そのシステムの個々の構成要素について定量的な値を調べ，システム全体の値とすることができよう．しかし，構造が複雑になり，数百，数千にも及ぶファイルから構成されるシステムでは，何らかの機械的な処理により，自動的に求めることが必須となる．

本研究では，ソースプログラムとして与えられた二つのソフトウェアシステムの類似度メトリクスCSR(Corresponding Source-line Ratio) を提案する．本稿では，CSR の形式的な定義を与え，それを求めるための現実的な方法を示す．そして，実際に CSR を種々の UNIX 系 OS のいくつかの版に適用した結果を示す．

さらに，得られた CSR を元にして，クラスタ分析を行い OS のバージョンを分類し，バージョンの樹状図を作成した．その結果，ここで提案する CSR を類似度メトリクスとしてを用いた樹状図は，UNIX の開発者が示した系譜図にほぼ対応することが分かった．また，他のいくつかのメトリクス（名前が一致するファイル数の割合，対応が取れるファイル数の割合など）を類似度として用いてクラスタ分析を行った結果との比較を行った．さらに，二つのバージョンの間の CSR と開発期間の間に相関のあることも分かった．

以下，2 節では，類似度メトリクス CSR を定義する．3 節では，CSR を現実的に求めるための方法について議論する．4 節では，UNIX OS への適用について述べ，5 節で，そのデータに基づいたクラスタ解析を行なう．6 節で考察し，参考文献との比較を行なう．

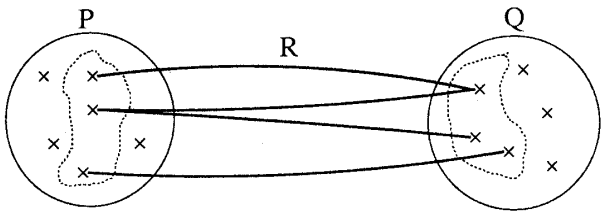

図 1: 対応

2 類似度メトリクス

本節では，ソースプログラムとして与えられた二つのソフトウェアシステムの類似度メトリクス CSR(Corresponding Source-line Ratio) を提案する．二つのソフトウェアシステムの類似度メトリクスには，多くの要素が考えられる．たとえば，ファイルの数や行数，ファイル名の違いなどである．以降，我々が提案する類似度メトリクス CSR の形式的な定義を行い，それを求めるための具体的な方法を説明する．

2.1 類似度の定義

一つのファイルや複数のファイルから構成されるソフトウェアシステムの間の関係を定義するために，ファイルやソフトウェアシステムを抽象的にあらわす“プロダクト”を用いる．ここでは簡単化のためにプロダクトは，その要素の集合と考え，$P = \{p_1, \cdots, p_m\}$ と書く．いま P をファイルとすると，各 p_i はファイルの各行，P をソフトウェアシステムとすると，ソフトウェアシステムを構成する各ファイルとなる．

二つのプロダクト $P = \{p_1, \cdots, p_m\}$, $Q = \{q_1, \cdots, q_n\}$ に対し，対応 $R \subseteq P \times Q$ が得られるとする．

今，P と Q の R に対する類似度 $S(0 \leq S \leq 1)$ を次のように定義する．

$$S(P,Q) \equiv \frac{|\{p_i|(p_i,q_j) \in R)\}| + |\{q_i|(p_i,q_j) \in R)\}|}{|P| + |Q|}$$

これは，図 1 のように対応 R に含まれる P, Q の要素数を P と Q の総要素数で割ったものである．R に関係しない P, Q の要素が増えることによって S は

下がる．$R=\phi$ では，$S=0$ となる．また，P と Q が同じものの時，$\forall i(p_i,q_i)\in R$ となり $S=1$ となる．

2.2 類似度の適用方法と CSR

定義した類似度を実際のソフトウェアシステムやファイルに適用する方法を考える．

1. P,Q をファイルとし，p_i,q_i をそのファイルの中の行とする．R は，ファイルの差分抽出ツール diff により与えられる関係とすると，S をファイル間の類似度を定義し計算で求めることができる．

2. P,Q をソフトウェアシステムとし，p_i,q_i をそのソフトウェアシステムを構成するファイルとする．R を同じファイル名を持つファイルの対応とする．この場合，容易にシステム間の類似度を計算できるが，名前を変更した場合や名前は同じだがファイルの中身を変更した場合などは，類似度は直感的な値とは異なってしまう．また，ファイルの大きさにかかわらず均等な重みで類似度を計算するため，直感に合わない場合もある．たとえば，小さな多数のファイルのみが対応にあり，少数の大きなファイルが対応していない場合，高い類似度になってしまう．

3. P,Q をソフトウェアシステムとし，p_i,q_i をそのソフトウェアシステムを構成するファイルとする．R をファイル同士の類似度が最も高いファイルへの対応とする．こうした場合，ファイル名の考えや中身の変更には対応できる．しかし，ファイルの大きさが反映されない．また，すべての組み合わせでファイル間の類似度を求めなければならず，大きな手間がかかる．

4. P,Q をソフトウェアシステムとし，p_i,q_i を P, Q それぞれの各ファイルの各行とする．直感的には各ファイルを連結したファイルの各行を考える．何らかの方法で各行の対応 R が与えられたとする．類似度は，ファイル名やファイルの大きさに影響されず，直感的に近い値が得られることが期待される．本稿では，この方法を用いたソフトウェアシステム間の類似度メトリクスを CSR(Corresponding Source-line Ratio) と呼ぶ．

以降では，どのようにして，R を決め，CSR を求めるかを詳しく述べる．

3 CSR の求め方

本節では，前節で定義した類似度メトリクス CSR を求めるための対応 R の具体的な求め方にについて説明する．さらに，与えられた二つのソフトウェアシステムから類似度を計算するアルゴリズムについて述べる．

3.1 アプローチ

すべてのファイルのすべての行に対しての対応を求めるためには，各行に対してその行と同じ行が存在するかどうか調べればよい．

このようなコードの重複を求めるための手法はすでにいくつか存在する．

クローン検出ツール CCFinder[2] は，複数のソースコードを入力としてコードクローンを出力するツールである．CCFinder はソースコードをプログラミング言語の文法に沿ってトークン列に変換する．その際，複数のソースも一つのトークン列に連結される．また，ソースコード中の空白とコメントは生成されるプログラムの機能に影響しないので無視される．さらに，実用的に意味のあるクローンのみを検出するために，そのトークン列の変換を行う．これは，パラメータ置き換え（名前が異なっても等価にする）などである．そして，そのトークン列を比較し，ソースコードが一致しているかどうか調べる．一致した部分トークン列をコードクローンと呼ぶ．

UNIX の diff コマンドは，二つのファイルの各行に対して発見アルゴリズム LCS[3, 4] を利用して，そのファイル間の行単位の差分を求める．この差分は，一方のファイルからもう一方のファイルを生成可能なファイルとなっている．diff コマンドの特徴は構文解析などを必要とせず差分の計算が行えるところにある．

ここで，CCFinder と diff を用いて，対応を求める．まず CCfinder を用いてコードクローンを検出する．検出されたクローンを持つファイル間に対して diff を実行する．CCfinder と diff によって同一行と判断された行の間に対応があるものとする．2 種類のツールを組み合わせることで，類似度の正確性を向上させる．

3.2 アルゴリズム

CSR を求めるためのアルゴリズムを以下に示す.

入力：二つのソフトウェアシステム P と Q

出力：P と Q の類似度 CSR ($0 \leq S \leq 1$)

Step1: 前処理

生成されるプログラムの機能に影響を与えない部分を取り除く．この処理は，用いられているプログラミング言語によって異なる．たとえば，C 言語で記述されたファイルの場合，コメント部分，空行をすべて取り除く．これにより，diff を実行した時の類似度の精度を向上させる.

Step2: CCFinder の実行

与えられた二つのソフトウェアシステムを入力として CCFinder を実行させる．実行させる際のオプションとして，最低一致トークン数を 20 とする．最低一致トークン数とは，出力すべき一致するトークン列の長さの最低値を表す.

Step3: diff の実行

CCFinder の実行の結果，一つでもクローンペアがみつかったペアのファイルにすべてに対して diff を実行する.

Step4: 対応の抽出

CCFinder で検出されたクローンのトークン列から，実際のファイルの一致している行同士を求める．さらに，diff で求まった差分情報から一致している行同士を計算する．CCFinder か diff のどちらかで一致している判断された行同士に関係を定義する.

Step5: CSR の計算

CSR の定義より計算する．ただし，二つのソフトウェアシステムの全行数は前処理後の行数を用いる.

対応を求めるにあたって，CCFinder だけでなく，diff も用いる理由は対応の正確性の向上である．C 言語のプリプロセッサ命令（#include 行など）は CCFinder では除外される．そのため，diff を用いた差分情報を加えることにより，同一行と判断できる行が増加する．そのため，類似度はより正確に類似をあらわす値となると考える．後節で述べる適用実験の結果，対応をもつ行は一割程度増加する.

また，diff だけを用いた対応の抽出の場合，ディレクトリ構造を保ったまま二つのソフトウェアシステムを入力とする場合と，一つのソフトウェアシステムのすべてのファイルの全行を巨大な一つのファイルにまとめたのを入力にする場合の二種類が考えられる．前者の場合，二つのソフトウェアシステムの間で同一の構造を持つ必要があり，ファイル名の変更やディレクトリ構造の変化に追従できない問題が生じる．後者の場合，結合するファイルの順番に問題が生じる．diff で利用しているアルゴリズムでは，文字列の入れ替えに対応できないためである．つまり，ファイル A，B が存在するとき，A，B と結合したファイルと B，A と結合したファイルに対して，diff を実行したとき，これらの二つのファイルは同一と判断できない．そのため，CCfinder と diff を組み合わせて対応を求める方法を採用している.

4 適用

本節では，類似度メトリクス CSR を，実際のソフトウェアシステムに適用する．適用するソフトウェアシステムとして，最近の UNIX 系 OS を用いた.

4.1 適用するシステム

適用するソフトウェアシステムは，BSD 系 UNIX である 4.4-BSD Lite，4.4-BSD Lite2 と，これらから派生した OS である FreeBSD[8]，NetBSD[9]，OpenBSD[6] である．FreeBSD，NetBSD，OpenBSD はオープンソースとして開発が現在も進められている．この三つの OS からは，4.4-BSD Lite 以降にリリースされたバージョンから現在までのバージョンまでの中から主要なバージョンを選びだした．その結果，FreeBSD は 6 バージョン，NetBSD は 6 バージョン，OpenBSD は 9 バージョンを選び出した．適用した OS は総数 23 個となる．これらの各 OS に対し，すべての組み合わせを考えて類似度 CSR を計測した.

表 1: 各 OS のファイル数と行数

FreeBSD

バージョン	2.0	2.0.5	2.1	2.2	3.0	4.0
ファイル数	891	1018	1062	1196	2142	2569
行数	228868	275016	297208	369256	636005	878590

NetBSD

バージョン	1.0	1.1	1.2	1.3	1.4	1.5
ファイル数	2317	3091	4082	5386	7002	7394
行数	453026	605790	822312	1029147	1378274	1518371

4.4BSD

バージョン	Lite	Lite2
ファイル数	1676	1931
行数	317594	411373

OpenBSD

バージョン	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
ファイル数	4200	4987	5245	5314	5507	5815	6074	6298	6414
行数	898942	1007525	1066355	1079163	1129371	1232858	1329293	1438496	1478035

表 2: 類似度一覧（一部）

	FreeBSD 2.0	FreeBSD 2.0.5	FreeBSD 2.1	FreeBSD 2.2	FreeBSD 3.0	FreeBSD 4.0	4.4BSD-Lite	4.4BSD-Lite2	NetBSD 1.0	NetBSD 1.1	NetBSD 1.2	NetBSD 1.3
FreeBSD 2.0	1.000	0.833	0.794	0.550	0.315	0.212	0.419	0.290	0.440	0.334	0.255	0.205
FreeBSD 2.0.5	0.833	1.000	0.943	0.665	0.392	0.264	0.377	0.266	0.429	0.348	0.269	0.227
FreeBSD 2.1	0.794	0.943	1.000	0.706	0.421	0.286	0.362	0.258	0.411	0.336	0.265	0.225
FreeBSD 2.2	0.550	0.665	0.706	1.000	0.603	0.405	0.226	0.179	0.291	0.254	0.225	0.201
FreeBSD 3.0	0.315	0.392	0.421	0.603	1.000	0.639	0.138	0.133	0.220	0.193	0.190	0.208
FreeBSD 4.0	0.212	0.264	0.286	0.405	0.639	1.000	0.101	0.100	0.140	0.152	0.158	0.179
4.4BSD-Lite	0.419	0.377	0.362	0.226	0.138	0.101	1.000	0.651	0.540	0.421	0.331	0.259
4.4BSD-Lite2	0.290	0.266	0.258	0.179	0.133	0.100	0.651	1.000	0.450	0.431	0.436	0.366
NetBSD 1.0	0.440	0.429	0.411	0.291	0.220	0.140	0.540	0.450	1.000	0.691	0.553	0.445
NetBSD 1.1	0.334	0.348	0.336	0.254	0.193	0.152	0.421	0.431	0.691	1.000	0.783	0.622
NetBSD 1.2	0.255	0.269	0.265	0.225	0.190	0.158	0.331	0.436	0.553	0.783	1.000	0.769
NetBSD 1.3	0.205	0.227	0.225	0.201	0.208	0.179	0.259	0.366	0.445	0.622	0.769	1.000

3 節の類似度を計測するアルゴリズムの Step1: の言語依存部分の処理は，以下の通りである．

- 計測対象は OS のカーネル部分のみとし，カーネルを生成するのに必要なファイルだけを取り出す．
- 対象言語は C 言語のみとし，拡張子が .c .h のファイルのみを計測対象とする．
- コメントと空行は全て消去する．

4.2 適用結果

各 OS の総ファイル数と行数を表 1 に示す．この表の値は，Step1: を行った後の結果に測定した値である．類似度を計測した値の一部を表 2 に示す．

表 2 の FreeBSD だけに注目すると，各バージョンの中で，最も類似度が高い値を持つのはそのバージョンの前後のどちらかである．リリースした時期により前のバージョンか後のバージョンのどちらかになる．たとえば，FreeBSD 2.2 と他のバージョンとの CSR を図 2 に示す．自分自身を除くと，最も類似度が高いバージョンは 0.706 の FreeBSD 2.1 である．次期バージョンである FreeBSD 3.0 との類似度は 0.603 であり，これは三番目に近い値となっている．二番目の類似度は FreeBSD 2.0.5 の 0.665 である．FreeBSD 3.0 で，大幅な変更が加えられていることが読みとれる．表 1 のファイル数と行数の変化を見ると，ファイル数は約 1.8 倍増加し，行数は約 1.7 倍増加しており，それ以前の変更量とは異なることが分かる．

FreeBSD と NetBSD の間の CSR を図 3 に示す．FreeBSD 2.0 から FreeBSD 2.2 までは，NetBSD 1.0 と最も類似度が高く，NetBSD のバージョンが上がるにつれ類似度は減少していく．しかしながら，FreeBSD 3.0，FreeBSD 4.0 に関しては NetBSD 1.3 と類似度が他の NetBSD のバージョンと比べ高くなっている．あるバージョンを基準に考え，比較するバージョンを上げていくと類似度の変化は減少する．FreeBSD と NetBSD という基本的に異なる開発（開発者や開発ポリシー）で行われている場合，類似度が上がる原因

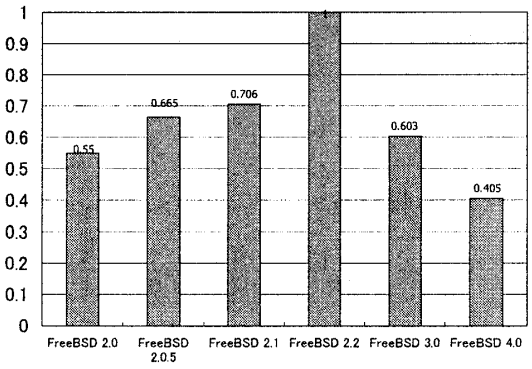

図 2: FreeBSD 2.2 との CSR

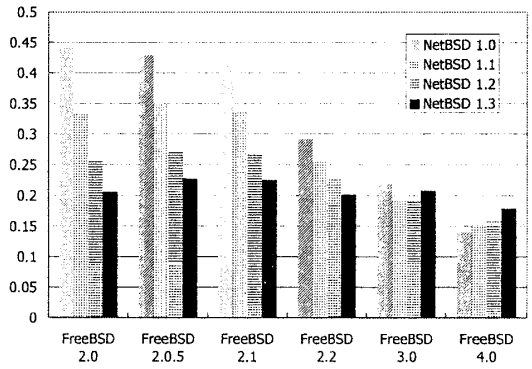

図 3: FreeBSD と NetBSD の CSR

としては，他方の OS にある機能を追加するために，ソースコードをコピーした場合や，両方の OS に同一ファイル取り込んだ場合が考えられる．

図 4 に BSD 系 UNIX の派生ツリーを示す [7]．この図は，FreeBSD，NetBSD．OpenBSD といった OS がどのように派生し，いつリリースされたかをあらわしている．FreeBSD 3.0，NetBSD 1.3 は，共に 4.4BSD Lite2 を取り込んだ最初のバージョンであることを示している．つまり，FreeBSD，NetrBSD に取り込まれた 4.4BSD Lite2 のソースコードの行が対応し，類似度が増加した．

これらのことから，CSR がソフトウェアシステムの類似度を正しく表しているといえる．

5 分析

本節では，4 節で適用して得られた類似度 CSR を元にして，クラスタ分析を用いてバージョンを分類し，バージョンの樹状図（デンドログラム）を作成する．作成した樹状図と OS の系譜図を比較し，似ている点，異なる点についての議論を行う．

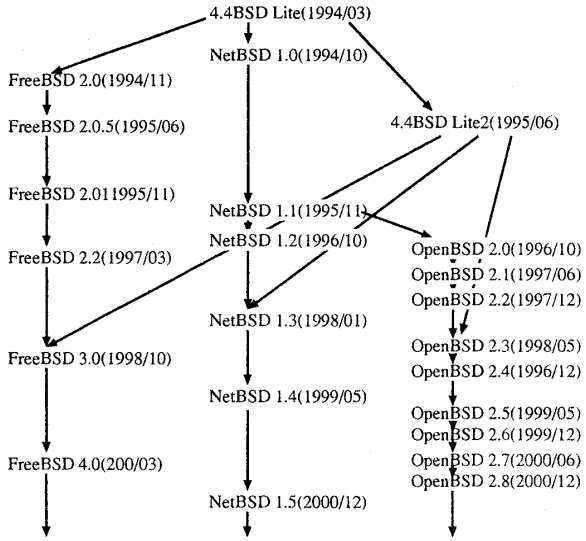

図 4: BSD 系 UNIX 派生図

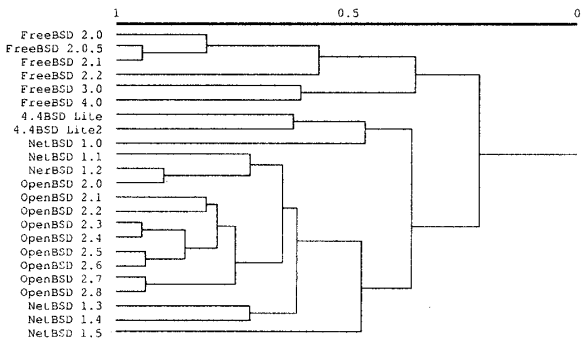

図 5: 類似度 CSR を用いた樹状図

5.1 樹状図

類似度 CSR を各個体間の距離として，クラスタ分析を行う．クラスタ間の距離には平均距離を用い計算する．クラスタ分析を行い，その結果から得られた樹状図を図 5 に示す．横軸は結合距離を表している．

図 5 から，FreeBSD，NetBSD，OpenBSD がどのように分類できるか考える．最も大きな分類としては，最終的なクラスタ統合が行われている部分で，FreeBSD と NetBSD + OpenBSD という分類である．類似度 CSR を用いたクラスタ分析による分類が正しく OS の種類の分類を反映している．

次に，NetBSD と OpenBSD の分類を見てみる．

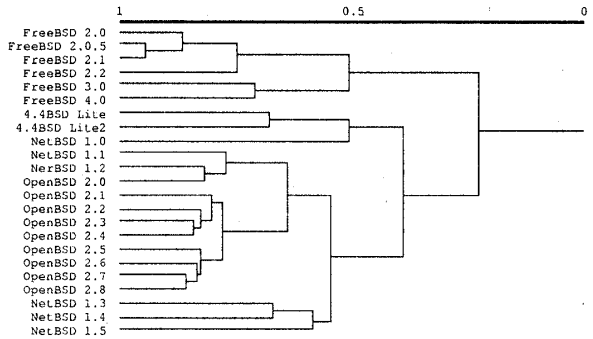

図 6: 一致したファイル数を用いた樹状図

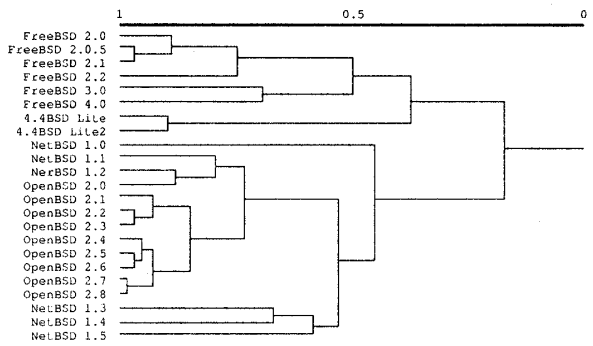

図 7: 同一ファイル名を用いた樹状図

OpenBSD 2.0 を除いたすべての OpenBSD のバージョンが統合され，NetBSD 1.1 と統合されている．OpenBSD 2.0 に関しても NetBSD 1.2 と統合された後，NetBSD 1.1 に統合されている．図 4 から分かるように，OpenBSD は NetBSD 1.1 から派生してできた OS である．樹状図でも，OpenBSD が NetBSD 1.1 から派生していることを示している．また，図 4 から NetBSD 1.2 と OpenBSD 2.0 は同時期にリリースされたバージョンであり，また，OpenBSD 2.0 は最初のバージョンであることも分かる．これらの原因から NetBSD 1.2 と OpenBSD が高い類似度を示した．

5.2 他の類似度

CSR による類似度の有効性を調べるため，各個体間の距離を CSR 以外の類似度を用いてクラスタ分析を行った．用いた類似度は，全ファイル数のうちの CCFinder でクローンが検出されたファイル数の割合と，全ファイル数のうちのディレクトリ名を含んだファイル名が一致するファイル数の割合である．これら二つの類似度を用いて作成した樹状図を図 6，図 7 にそれぞれ示す．共に図 5 と同様の結果が得られることが分かる．この理由は，これらの OS すべてが BSD 系由来のファイル構造，ファイル名を持つためだと考えられる．ファイル名命名規則やファイルの階層構造が決まったポリシーで開発が行われている場合は，単純にファイル名を用いた分析も可能であることがいえる．

表 3: FreeBSD 4.0 と Linux 2.2.15 の類似度

CSR	ファイル数の類似度
0.031	0

表 4: バージョン間のリリース間隔（ヶ月）

	2.0	2.0.5	2.1	2.2	3.0	4.0
FreeBSD 2.0	0	7	12	28	47	64
FreeBSD 2.0.5	7	0	5	21	40	57
FreeBSD 2.1	12	5	0	16	35	52
FreeBSD 2.2	28	21	16	0	19	36
FreeBSD 3.0	47	40	35	19	0	17
FreeBSD 4.0	64	57	52	36	17	0

さらに．由来が異なる二つの OS であり同時期にリリースされた FreeBSD 4.0 と Linux2.2.15 の類似度を計算した．計算する類似度は CSR と全ファイル数のうちのディレクトリ名を含んだファイル名が一致するファイル数の割合である．結果を表 3 に示す．ファイル数の割合は 0 であり，一つも同一のファイル名がみつからかった．しかし，CSR の値は，0.031 となった．異なる開発形態では，ファイル数の割合だけでは，必ずしも類似度を表せるとは限らないと考えられる．

6 考察

CSR は類似度を表す値であるが，CSR を計算する過程で各行の対応を求めている．この各行の対応を用いると，実際にファイルのどの行が他のファイルの行と同じであるか知ることが可能である．CSR の値だけでなく，これらの対応する行の情報も得ることが出来，類似度を用いソフトウェアシステムを参照，改変する場合に役に立つ．

図 4 に示したリリース時期と類似度との間に相関があるかどうか計算をした．FreeBSD のリリース間隔を計算した結果を表 4 に示す．CSR とリリース間隔

の相関は，-0.973 であった．表 1 の行数の単純な増減数とリリース間隔との相関は，0.528 であった．CSR とリリース間隔との間には強い負の相関があることがわかる．また，行数の増減数より CSR がリリース間隔との相関が高い．このことから，ある二つのバージョンの類似度を計測することで，そのリリース間隔を知ることが可能である．CSR は，言い換えるとソースコードの変更していない部分の割合を示しているため，相関が高いということは，開発が滞りなく一定に行われているといえる．

類似度を求める研究としてさまざまな研究がある．Baxter らは AST を利用したクローン検出手法を提案している [1]．しかしながら，類似度を求める定義はあるが，その値をどのように用いるかや実際にシステムに適用した結果などなく，定量的な評価を行ってはいない．また，[5] では，提案した類似度を実際のソフトウェアに適用し，類似度が妥当であるか検討しているが，対象ソースコードが COBOL であり，提案する類似度も他のプログラミング言語に対応することが難しいといった問題がある．

7 まとめ

本稿では，ソースプログラムとして与えられた二つのソフトウェアシステムの類似度メトリクス CSR(Corresponding Source-line Ratio) を提案した．具体的には，類似度 CSR はクローン検出ツール CCFinder と diff コマンドを用い計算される．そして，実際に CSR を FreeBSD，NetBSD，OpenBSD といった種々の UNIX OS のいくつかのバージョンに適用した．適用結果から，類似度 CSR は二つのソフトウェアシステム間の類似性を正しくあらわしていると考えることが出来る．さらに，得られた類似度 CSR を元にして，クラスタ分析を行い，各 OS のバージョンを分類し，バージョンの樹状図を作成した．その結果，提案する CSR を類似度としてを用いた樹状図は，OS の分類を派生通りにあらわしていることがが分かった．また，二つのバージョンの間の CSR と開発期間の間に強い相関があることも分かった．

今後の課題としては，さらなる類似度の妥当性の検証，類似度と再利用プログラミングとの関係の計測などが挙げられる．

参考文献

[1] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M. and Bier, L.: Clone Detection Using Abstract Syntax Trees, *Proceedings; International Conference on Software Maintenance* (Koshgoftaar, T. M. and Bennett, K.(eds.)), IEEE Computer Society Press, pp. 368–378 (1998).

[2] 神谷年洋, 楠本真二, 井上克郎: "コードクローン検出における新手法の提案及び評価実験", 電子情報通信学会技術研究報告、SS2000-42～52, Vol. 100, No. 570, pp. 41–48 (2001).

[3] Miller, W. and Myers, E. W.: A File Comparison Program, *Software- Practice and Experience*, Vol. 15, No. 11, pp. 1025–1040 (1985).

[4] Myers, E. W.: An $O(ND)$ difference algorithm and its variations, *Algorithmica*, Vol. 1, pp. 251–256 (1986).

[5] 長橋賢児: "類似度に基づくソフトウェア品質の評価", 情報処理学会研究報告 2000-SE-126, Vol. 2000, No. 25, pp. 65–72 (2000).

[6] OpenBSD: . "http://www.openbsd.org/".

[7] Schneider, W.: The UNIX system family tree: Research and BSD. "ftp://ftp.freebsd.org/pub/FreeBSD/branches/-current/src/share/misc/bsd-family-tree".

[8] The FreeBSD Project: . "http://www.freebsd.org/".

[9] The NetBSD Foundation Inc.: . "http://www.netbsd.org/".