論　文

ソフトウェアシステムの類似度とその計測ツール SMMT

山本　哲男[†]　　松下　誠[††]　　神谷　年洋[†]　　井上　克郎[††]

Similarity of Software System and Its Measurement Tool SMMT

Tetsuo YAMAMOTO[†], Makoto MATSUSHITA[††], Toshihiro KAMIYA[†], and Katsuro INOUE[††]

あらまし　二つのソフトウェアシステムが与えられたとき，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，二つのシステムからそれらの相違点を定量的な値として計測することは容易ではなかった．本論文では，ソフトウェアシステム間の類似度メトリックスとそれを計測するツール SMMT（Similarity Metrics Measuring Tool）を提案し，そして，SMMT を種々の UNIX 系 OS のソースコードに適用した．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，OS の分類を正しく行えるか検証を行った．その結果，類似度は OS の変遷を知る有効な指標となり，得られた派生図は OS の分類を派生どおりに表していることを確認した．

キーワード　コードクローン，ソフトウェアメトリックス，類似度

1. まえがき

二つのソフトウェアシステムが与えられたとき，そのシステム間の違いはどれぐらいあるのか，定量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間における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．また，システムを改変する際の有益な指針にもなる．更に，リリース版とその修正版といった少し異なるバージョンが多数存在する場合，その管理にも役立つ．例えば，あるソフトウェアで多くのバージョンを作成した際に，どのバージョンがどのバージョンから派生したかわからなくなる場合がある．このようなとき，バージョン間の相違の度合から派生の系統を調べ，系統樹を作成することで解決することができる．また，最新バージョンにバグがあり，ソフトウェアを作成し直す場合においても，バージョンの系統樹からどのバージョンから作成し直せばよいかの目安になる．

システムの改変時にドキュメントが作成されていれば，それを手掛りにして，システム間の相違点を知ることは可能であろう．しかし，システム間の違いを定量的な値として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，自動的に値を求めることが必須となる．

システム間の類似度を求める研究として様々な研究がある．Baxter らは抽象構文木（Abstract Syntax Tree）を利用したクローン検出手法を提案している [2]．しかしながら，類似度を求める定義はあるが，その値の有効性については述べられていない．また，実際にシステムに適用した結果はなく，定量的な評価を行っていない．[1], [12], [17] は，プログラムの類似度を自動的に計測するツールであるが，大規模システムに適用した結果はない．[10] では，提案した類似度を実際のソフトウェアに適用し，用途に応じてどのような類似度が考えられるかについて考察をしている．

我々は，ソフトウェアシステムの類似部分を見つけるだけではなく，計測した類似度を用い，ソフトウェアシステムの各バージョンの派生を調べる．得られた派生図をソフトウェアシステムの改良，保守に役立て

[†] 科学技術振興事業団，川口市
Japan Science and Technology Corporation, Kawaguchi-shi, 332–0012 Japan
[††] 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, 1–3 Machikaneyama-cho, Toyonaka-shi, 560–8531 Japan

ることを目的としている．本論文では，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を求めるためのメトリックスを提案し，それを自動的に計測するツール SMMT（Similarity Metrics Measuring Tool）の提案を行う．類似度メトリックスの形式的な定義を行い，次にそれを求めるための現実的な方法を示す．その方法を用いて実際にツール SMMT を種々のバージョンの UNIX 系 OS に適用した結果を示す．

そして，得られた類似度を基にして，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，ここで提案するメトリックスを用いた樹状図は，OS の開発者が示した系譜図にほぼ対応することがわかった．

以下，**2.** では，類似度の形式的な定義とそれを計測するメトリックスを提案する．**3.** では，メトリックスを現実的に求めるための方法と SMMT の実現について述べる．**4.** では，UNIX 系 OS への適用について述べ，適用結果に基づいたクラスタ分析を行い，提案する類似度の妥当性の検証を行う．

2. 類似度とそのメトリックス

2.1 類似度の定義

[2] では，二つの抽象構文木の類似度を定義している．定義している類似度は一致する節点数をすべての節点数で割った値である．また, [10] では，類似度を二つのプログラムの全行数の中で共通部分の行数の割合と定義している．我々も同様に，二つのソフトウェアシステムで等価な要素数を全要素数で割った値を類似度とする．以下に我々が提案する類似度メトリックスの形式的な定義を示す．

ソフトウェアシステム P はそれを構成する要素の集合と考え，$P = \{p_1, \cdots, p_m\}$ と書く．二つのソフトウェアシステム $P = \{p_1, \cdots, p_m\}$, $Q = \{q_1, \cdots, q_n\}$ に対し，等価な要素の対応 $R_s \subseteq P \times Q$ が得られるとする．P と Q の R_s に対する類似度 $S(0 \leq S \leq 1)$ を次のように定義する．

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

これは，図 1 のように対応 R_s 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_s 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．$R_s = \phi$ では，$S = 0$ となる．また，P と Q が等価であるとき，$\forall i(p_i, q_i) \in R_s$ となり $S = 1$ となる．

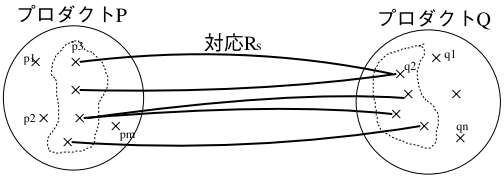

図 1 要素の対応 R_s
Fig. 1 Correspondence of elements Rs.

2.2 類似度のメトリックス

［等価なファイル名を用いたメトリックス S_{fn}］

ソフトウェアシステム P, Q に対し，p_i, q_j 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_s を同じ名前をもつファイル同士の対応として類似度を計測するメトリックスを S_{fn} とする．この場合，容易にシステム間の類似度を計測できるが，ファイル名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求めるため，直感に合わない場合もある．例えば，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，高い値になってしまう．

［等価な行を用いたメトリックス S_{line}］

ソフトウェアシステム P, Q に対し，p_i, q_j を P, Q それぞれの各ファイルの各行とする．直感的には P, Q の各ファイルを連結した一つのファイルをそれぞれ考え，その各行を構成要素とする．等価な行の対応を調べることによって対応 R_s が与えられる．この類似度メトリックスを S_{line} とする．これは，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．

そのほかに，いろいろ類似度のメトリックスを考えることができようが，求める手間やその効果を考え，ここでは，S_{fn} と S_{line} について議論する．S_{fn} は容易に求めることができるので，以降では，S_{line} の求め方について詳しく述べる．

3. S_{line} の求め方とツール SMMT

3.1 アプローチ

二つのソフトウェアシステム P, Q に対し S_{line} を

求めるためには，R_s を得ること，すなわち P の各ファイルの各行が Q のどのファイルのどの行に対応するか，または対応する行がないか，を知る必要がある．

このための簡単な方法としては，P の各ファイルを連結したファイル p_{all} と Q の各ファイルを連結したファイル q_{all} を作り，p_{all} と q_{all} の間の共通部分を求めて，そこに含まれる各行を R_s とすることが考えられる．共通部分の発見には，通常，最長共通部分列を発見するアルゴリズム LCS [8], [9], [16] を用いた diff [3] 等のツールが便利であるが，ファイル名が変わるなどしてファイルの連結順が変わった場合には，共通部分列として検出ができなくなる．

そこで，本研究ではコードの重複（クローンと呼ぶ）を求めるためのツール CCFinder [5] と diff とを組み合わせて R_s を求める．

CCFinder は，与えられたプログラムファイルの内に存在する同じプログラム文の系列を効率良く検出し，出力する．ただし，コメントや改行，空白の違い，また，変数や関数の名前（識別子）の違いがあっても同じものとして検出する．

まず CCFinder を用いて p_{all} と q_{all} の間に存在するクローンを検出する．クローン中の各行は R_s の要素とする．CCFinder は，後述のように保守作業にとって無意味なクローンを除去して出力する．しかし，除去されるものの中には類似度における対応としては有用なものもある．

そこで，検出されたクローンをもつファイル間に対し，共通部分列を diff によって求め，そこに含まれる各行も R_s の要素とする．2 種類のツールを組み合わせることで，意味のある行の対応を効率良く求めることができる．

CCFinder では，プリプロセッサ命令（C 言語の #include 行など）は検出対象から除外される．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し，より有効な行の対応が求められると考える．後節で述べる適用実験の結果，対応をもつ行は 1 割程度増加する．

また，diff だけを用いた対応の抽出の場合，二つのソフトウェアシステムの間で同一の構造をもつ必要があり，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．同一の構造をもたない場合，P と Q のファイルのすべての組合せについて diff を実行しなければならず，多くの時間を要する．そのため，ここでは CCFinder と diff を組み合わせて対応を求める方法を採用する．

3.2 対応の求め方

CCFinder と diff を用いた対応の求め方の例を述べる．図 2 に示すようにソフトウェアシステム P, Q があり，P は二つのファイルから構成され，Q は四つのファイルから構成されているとする．また，Q は P の後継バージョンとする．Q を構成するファイルの中でファイル A とファイル B は，P のファイル A とファイル B をもとにしており，ファイル C はファイル A をもとに新しく作成されたファイルである．更に，ファイル D は新規に作成されたファイルとする．

この二つのソフトウェアシステム P, Q の R_s を求めるために，まず CCFinder を用いて P と Q の間に存在するクローンの検出を行う．検出されたクローンからクローンを含む行の間に対応を結び R_s の要素とする．更に，クローンが一つでも存在するファイルの組を探す．図 2 の場合，矢印で結ばれた P のファイル A と Q のファイル A，P のファイル B と Q のファイル B，P のファイル A と Q のファイル C の間にクローンが一つ以上検出されたとする．

次に，これらの三つの組に対してのみ diff を用いてファイル間の差分を求める．差分の結果から共通行と判断された各行も R_s の要素とする．この方法では，まずすべてのファイルを入力として CCFinder を実行し，その結果を使用して一部のファイル間の組合せについてのみ diff を実行する．CCFinder は $O(n)$ の手間で，高速に処理を行う．例えば，ファイル数が 1648 個で全行数が 50 万行のソフトウェアシステムの処理を行った場合の実行時間は約 3 分である [5]．また，C

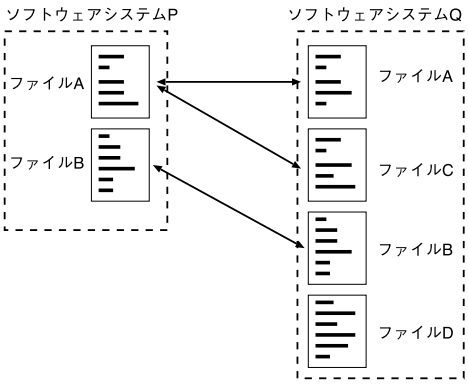

図 2　対応の求め方
Fig. 2　How to find a correspondence.

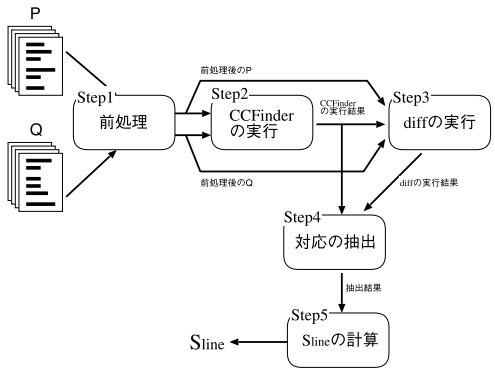

図 3　処理の流れ
Fig. 3　Similarity measuring process.

言語の#include 行といった多くのファイルに存在するような行の対応もクローンが検出されたファイルの組に対してのみ付けられる．そのため，単純に共通行であるとしても対応はせず，意味のある行のみが対応する．

3.3 ツール SMMT

S_{line} を求めるためのアプローチに基づき，S_{line} を計測するツールを作成した．以下に，ツールの入出力と処理の流れを述べる．図 3 に処理の流れを表す．

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 S_{line} $(0 \leq S_{line} \leq 1)$
Step 1:　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．例えば，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 を実行したときの類似度の精度を向上させる．

Step 2:　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．CCFinder の実行にあたって，最低一致トークン数を 20 とした．最低一致トークン数とは，一致するクローンを含むトークンの長さの最低値を表す．ツールのオプションとして，この値は変更可能である．

Step 3:　diff の実行

CCFinder の実行の結果，一つでもクローンが発見された P と Q のファイルの組のすべてに対して diff を実行する．

Step 4:　対応の抽出

CCFinder で検出されたクローンのトークン列から，一致している行同士を求める．更に，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の組を R_s に入れる．

Step 5:　S_{line} の計算

S_{line} 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．

ツール SMMT は Windows2000 上で稼動し，その対象言語は C，C++，Java，COBOL である．例えば，二つのソフトウェアシステムの総行数が 503884 行の S_{line} を計測する場合，PentiumIII 1 GHz，メモリ 2 GBytes のシステムで Step 1 から Step 5 までの処理にかかる時間は 329 秒である（CCFdiner と diff の実行時間を含む）．

4. S_{line} の妥当性検証

4.1 UNIX 系 OS への適用

対象のソフトウェアシステムは，BSD UNIX である 4.4BSD Lite，4.4BSD Lite2 [7] と，これらから派生した OS である FreeBSD [14]，NetBSD [15]，OpenBSD [11] を用いた．FreeBSD，NetBSD，OpenBSD は現在もオープンソースとして開発が進められている．本研究では，これら三つの OS のうち 4.4-BSD Lite のリリース以降に発表された主なバージョンを選んだ．その結果，FreeBSD は 6 バージョン，NetBSD は 6 バージョン，OpenBSD は 9 バージョンの 21 バージョンを選んだ．そして，4.4BSD Lite と 4.4BSD Lite2 を加え，総数 23 個の OS に対し，すべての組合せで類似度 S_{line} を計測した．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．更に，それらのファイルの中から拡張子が .c .h のファイルのみをツールへの入力とする．**3.3** の Step 1 における言語依存部分では，入力されるファイルを C 言語とみなしコメントと空行はすべて消去する処理を行う．

各 OS の総ファイル数と総行数を表 1 に示す．この表の値は，Step 1: を行った後に測定した値である．計測した類似度の一部を表 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もつのはそのバージョンの前後のどちらかである．リリースしたとき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．例えば，FreeBSD 2.2 と他のバージョンとの S_{line} を

表 1　各 OS のファイル数と総行数
Table 1　The number of files and LOC of OS.

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
総行数	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
総行数	453026	605790	822312	1029147	1378274	1518371

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
総行数	898942	1007525	1066355	1079163	1129371	1232858	1329293	1438496	1478035

4.4BSD

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

表 2　類似度一覧（一部）
Table 2　Similarity of OS(partial).

	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

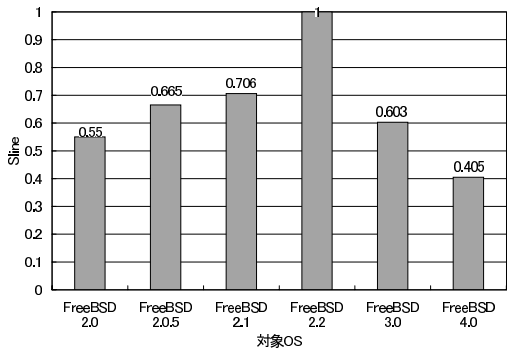

図 4　FreeBSD 2.2 に対する S_{line}
Fig. 4　S_{line} with FreeBSD 2.2.

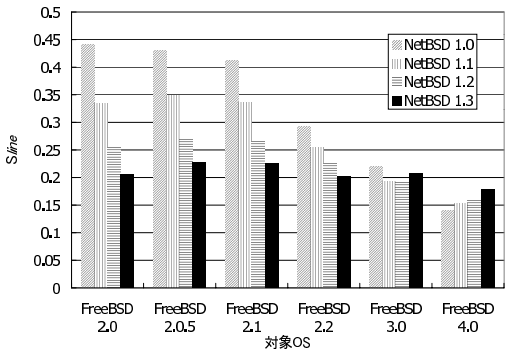

図 5　FreeBSD と NetBSD の S_{line}
Fig. 5　S_{line} between FreeBSD and NetBSD.

図 4 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，大幅な変更が加えられていることが読み取れる．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わかる．

FreeBSD と NetBSD の間の S_{line} を図 5 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 との類似度が他の NetBSD のバージョンと比べ高くなっている．FreeBSD と NetBSD という基本的に異なる開発環境（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．これを調べるために図 6 に示す BSD UNIX の派生図を調べた [13]．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表している．FreeBSD 3.0，NetBSD 1.3 は，ともに 4.4BSD Lite2 を取り込んだ最初のバージョンであることがわかる．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，類似度が増加したと考

えられる．これらのことから，S_{line} は OS の変遷を知る有効な指標といえる．

更に，例えば，4.4BSD Lite2 に致命的なバグが発見されたり，使用禁止になったりしたとした場合，4.4BSD Lite2 から取り込んだコードを取り除くか，取り込んだ以前のバージョンから最新バージョンを作成する必要がある．この際，取り込んだ以前のバージョンの特定は類似度を用い容易に検索可能であるため，類似度はソフトウェアを改変する際にも役立つ．

4.2 類似度メトリックスを用いたクラスタ分析

類似度メトリックス S_{line} を二つの OS の間の距離として，クラスタ分析 [4] を行う．クラスタ間の距離には平均距離を用いて計算する．クラスタ分析を行い，その結果から得られた樹状図（デンドログラム）を図 7 に示す．横軸は結合距離を表しており，左側で結合しているものほど類似度が高く，近い OS であることを示しているが，図 6 によってこの分析結果が正しいことがわかる．

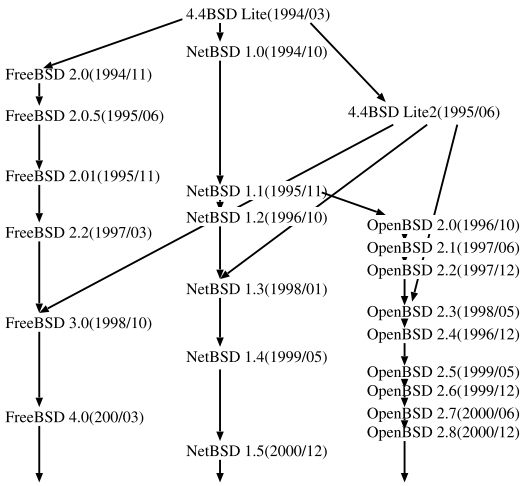

図 6　BSD 系 UNIX 派生図
Fig. 6　The BSD UNIX system family tree.

図 7 では，最も大きな分類としてクラスタ I と II に分けられている．FreeBSD はすべてクラスタ I に含まれ，OpenBSD はすべてクラスタ II に含まれており，お互い類似度が高くない，遠い系統であることを示している．図 6 の派生図を見ると OpenBSD は NetBSD から分岐しており，FreeBSD とは違う系統として開発されてきたことがわかる．

次に，NetBSD と OpenBSD の分類を見てみる．OpenBSD 2.0 を除いたすべての OpenBSD のバージョンが同一クラスタに統合され，NetBSD 1.1 と統合されている．これは，OpenBSD が NetBSD 1.1 から派生していることを示している．

このように，類似度を用いたクラスタ分析を行うことでバージョンの系統図が作成可能である．多数のバージョンを作成してそれらの派生の特定が困難になった場合，それらのバージョンの類似度を計測しクラスタ分析を行うことで派生関係がわかる．

各個体間の距離を S_{line} 以外の類似度を用いてクラスタ分析を行った．ここでは，前述した S_{fn} を類似度として用いて作成した樹状図を図 8 に示す．この結果より図 7 と同様の結果が得られることがわかる．この

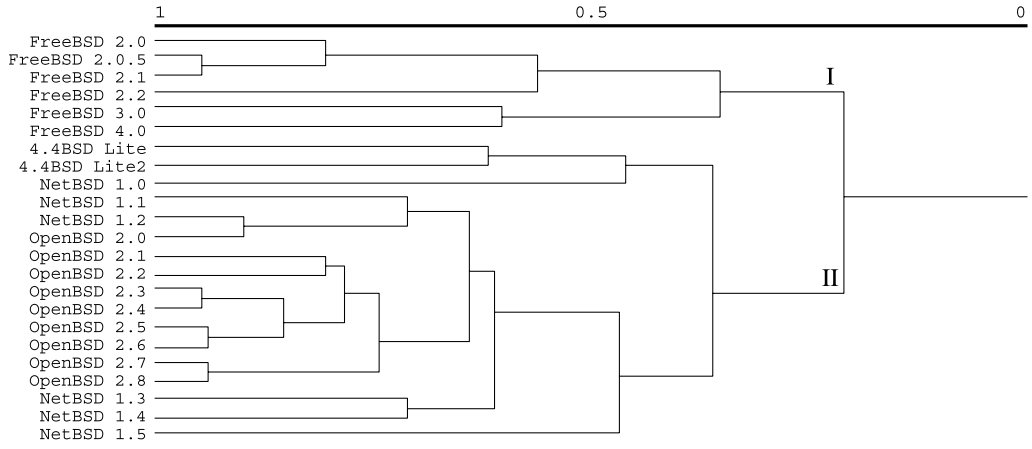

図 7　類似度 S_{line} を用いた樹状図
Fig. 7　Dendrogram using similarity S_{line}.

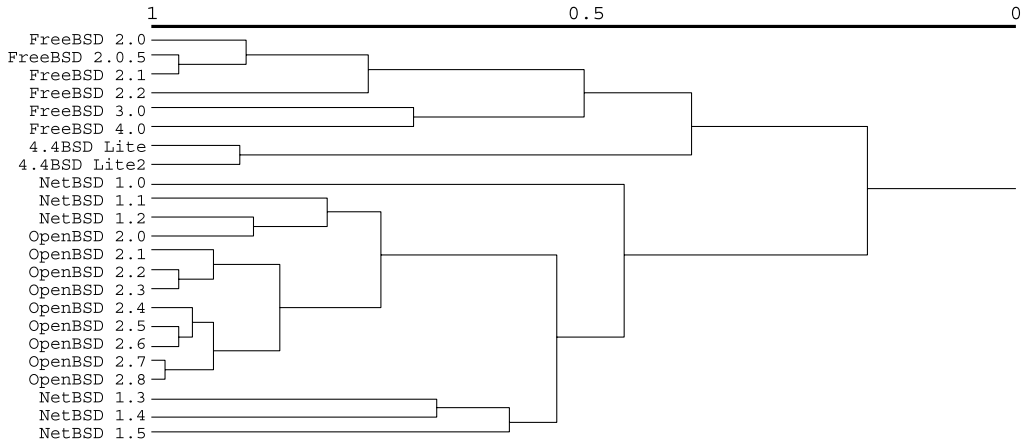

図 8　類似度 S_{fn} を用いた樹状図
Fig. 8　Dendrogram using similarity S_{fn}.

理由は，これらの OS すべてが BSD 系由来のファイル構造，ファイル名をも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て派生図を作成することが可能である．

4.3　開発系統が異なる OS の類似度

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 [6] の S_{line} を求めたところ，0.031 と低い値を示した．対応する行はデバイスドライバがほとんどである．異なる開発系統の OS は S_{line} によって容易に区別することができる．

4.4　S_{fn} との比較

S_{line} の簡便な代替メトリックスとして S_{fn} が用いられるかどうか検討した．今回は，大阪大学基礎工学部情報科学科の Pascal コンパイラの開発演習において作成されたシステムに対し，S_{line} と S_{fn} を計測した．本演習では，指導書によって提出すべきシステムの実行ファイル名が指定されているが，ソースファイルの名前は指定されていない．収集できた計 8 人分のシステムに対する総当りでの S_{line}，S_{fn} の計測結果をそれぞれ表 3 と表 4 に示す．

S_{line} が高いシステムは他の学生との同一コードが多数含まれていることを意味する．また，S_{fn} が高いシステムは同一ファイル名が多数含まれていることを意味する．各 S_{line} と S_{fn} の相関は -0.004 と低い．本演習では，提出する実行ファイル名が決められているため，ソースファイルのファイル名も実行ファイルと同じファイル名を付けている学生がおり，それらの学生同士では，S_{fn} が高い値を示す．つまり，学生のファイル命名規則が同じであれば，高い類似度を示すことになり，S_{fn} は必ずしもシステムの類似度を表しているとはいえない．表 3 に示す S_{line} の値を見ると，学生 G と学生 H の間の類似度が 0.797 であり他の学生同士の類似度と比べると非常に高い値である．しかし，S_{fn} は 0.190 と低い．実際，学生 G と学生 H のソースコードを見比べると非常に似ており，S_{line} がシステムの類似度を反映しているといえる．

表 3　学生実験の S_{line}
Table 3　S_{line} of student experiment.

	A	B	C	D	E	F	G	H
A	1	0.009	0.024	0.038	0.034	0	0.054	0.047
B	0.009	1	0.040	0.001	0	0	0	0.023
C	0.024	0.040	1	0.060	0.042	0.088	0.118	0.170
D	0.038	0.001	0.060	1	0.010	0.040	0.069	0.039
E	0.034	0	0.042	0.010	1	0.022	0.172	0.237
F	0	0	0.088	0.040	0.022	1	0	0
G	0.054	0	0.118	0.069	0.172	0	1	0.797
H	0.047	0.023	0.170	0.039	0.237	0	0.797	1

表 4　学生実験の S_{fn}
Table 4　S_{fn} of student experiment.

	A	B	C	D	E	F	G	H
A	1	0	0	0.113	0	0	0	0
B	0	1	0.666	0.125	0.600	0.666	0.105	0.428
C	0	0.666	1	0.137	0.857	1	0.125	0.545
D	0.113	0.125	0.137	1	0.133	0.137	0.102	0.117
E	0	0.600	0.857	0.133	1	0.857	0.235	0.500
F	0	0.666	1	0.137	0.857	1	0.125	0.545
G	0	0.105	0.125	0.102	0.235	0.125	1	0.190
H	0	0.428	0.545	0.117	0.500	0.545	0.190	1

5. むすび

本論文では，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を定義し，それを計測するメトリクス S_{line} の提案を行った．実際にツール SMMT を作成し，種々のソフトウェアシステムに適用し，その有効性を確認した．更に，得られた類似度を基にして，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，提案する類似度メトリックスを用いて作成した樹状図は，OS の分類を派生どおりに表していることがわかった．

S_{line} は類似度を表す値であるが，S_{line} 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どの行と同じであるか知ることが可能である．S_{line} の値だけでなく，これらの行の対応の情報もソフトウェアシステムを参照，改変，保守する場合に役に立つと考えられる．

今後の課題として，更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．

文　献

[1] A. Aiken, Moss (measure of software similarity) plagiarism detection system, “http://www.cs.berkeley.edu/ moss/.”

[2] I.D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” Proc. International Conference on Software Maintenance, pp.368–378, Montpellier, France, 1998.

[3] Diffutls, “http://www.gnu.org/software/diffutils/diffutils.html.”

[4] B.S. Everitt, Cluster Analysis, 3rd ed., Edward Arnold, London, 1993.

[5] 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,” 信学技報，SS2000-42〜52, 2001.

[6] Linux Online, “http://www.linux.org/.”

[7] M.K. McKusick, K. Bostic, M.J. karels, and J.S. Quarterman, The Design and Implementation of the 4.4BSD UNIX Operating System, Addison-Wesley, 1996.

[8] W. Miller and E.W. Myers, “A file comparison program,” Software - Practice and Experience, vol.15, no.11, pp.1025–1040, 1985.

[9] E.W. Myers, “An $O(ND)$ difference algorithm and its variations,” Algorithmica, vol.1, pp.251–266, 1986.

[10] 長橋賢児，“類似度に基づくソフトウェア品質の評価,” 情処学研報，2000-SE-126, vol.2000, no.25, pp.65–72, 2000.

[11] OpenBSD, “http://www.openbsd.org/.”

[12] L. Prechelt, G. Malpohl, and M. Philippsen, “Jplag: Finding plagiarisms among a set of programs,” Technical Report 2000-1, Fakultat fur Informatik, Universitat Karlsruhe, Germany, 2000.

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

[14] The FreeBSD Project, http://www.freebsd.org/.”

[15] The NetBSD Foundation Inc., “http://www.netbsd.org/.”

[16] E. Ukkonen, “Algorithms for approximate string matching,” INFCTRL: Information and Computation (formerly Information and Control), vol.64, pp.100–118, 1985.

[17] K.L. Verco and M.J. Wise, “YAP3 : Improved detection of similarities in computer program and other texts,” Proc. 27th SIGCSE Technical Symposium on Computer Science Education, pp.130–134, New York, 1996.

（平成 13 年 8 月 13 日受付，11 月 14 日再受付）

山本　哲男

平 9 阪大・基礎工・情報卒．平 11 同大大学院博士前期課程了．平 14 同大大学院博士後期課程了．現在，科学技術振興事業団研究員．博士（工学）．ソフトウェアプロセスの研究に従事．

松下　誠

平 5 阪大・基礎工・情報卒．平 7 同大大学院博士前期課程了．平 10 同大大学院博士後期課程了．平 10 同大大学院基礎工学研究科・助手．平 14 同大大学院情報科学研究科・助手．博士（工学）．ソフトウェアプロセス，オープンソースソフトウェア開発の研究に従事．

神谷　年洋

平 8 阪大・基礎工・情報中退．平 10 同大大学院博士前期課程了．平 13 同大大学院博士後期課程了．現在，科学技術振興事業団さきがけ研究 21 研究員．博士（工学）．オブジェクト指向関連技術，ソフトウェアメトリクス，コードクローン，認知科学の研究に従事．

井上　克郎　（正員）

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 1 阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．平 9 同大大学院基礎工学研究科・教授．平 14 同大大学院情報科学研究科・教授．工博．ソフトウェア工学の研究に従事．