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.

類似度メトリクスを用いたJavaソースコード間類似度測定ツールの試作

小堀　一雄† 　山本　哲男†† 　松下　誠† 　井上　克郎†

† 大阪大学大学院情報科学研究科　〒 560–8531 大阪府豊中市待兼山町 1–3
†† 科学技術振興事業団　〒 332-0012 埼玉県川口市本町 4–1–8
E-mail: †{k-kobori,t-yamamt,matusita,inoue}@ist.osaka-u.ac.jp

あらまし　プログラム開発において，生産性と品質を向上する手段としてしばしば既存のソフトウェアの再利用を行なう．この時，類似したソフトウェア部品に関する情報は，ソフトウェア間のコピー関係を理解するのに非常に有用である．類似測定の最も単純で効果的な手法はソースコードの文字列比較である．しかし，この方法は計算コストが非常に大きいので大量の部品を対象とした類似測定には不向きである．そこで，本論文では構成トークンや複雑度などのメトリクスを用いたJavaプログラム間類似度測定手法の提案を行う．この手法は文字列比較を用いた場合と同様の解析を，より低コストで行うことができた．

キーワード　メトリクス，類似度，Java

Implementation of Java Program Similarity Measurement Tool Using Token Structure and Execution Control Structure

Kazuo KOBORI†, Tetsuo YAMAMOTO††, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1–3, Machikaneyama-cho, Toyonaka-shi, Osaka 560–8531, Japan
†† Japan Science and Technology Corporation
4–1–8, Honmachi, Kawaguchi-shi, Saitama 332–8531, Japan
E-mail: †{k-kobori,t-yamamt,matusita,inoue}@ist.osaka-u.ac.jp

Abstract　In program development process, engineers often reuse components which have already been produced in past development by copying directly or minor modification. In these cases, information on similar software components is useful for understanding of copy relation among components. One of the most simple and effective methods of similarity measurement is a string comparison between two source codes. However, the analysis cost is generally high, so this method is inapplicable to large source codes. In this paper, We propose similarity measurement method for Java programs by using software metrics that are calculated from the structure of token and execution control in the target source program. We compares the resulting metrics values without using expensive string comparison. Therefore, using this method, we can reduce the comparison cost sharply.

Key words　Metrics, Similarity, Java

1. はじめに

現在，ソフトウェアの大規模化と複雑化に伴い，品質の高いソフトウェアを一定期間内に効率よく開発することが重要になってきている．これを実現するための工学技術の一つとして再利用が注目されている．再利用とは，あるソフトウェアを実現しようとしたときに，新しくソースコードを一から書いていくのではなく，既存のソフトウェア部品を同一のシステム内や他のシステムで用いることであると定義されている [2]．一般にソフトウェアの再利用は生産性と品質を改善し，結果としてコストを削減する報告が多く見られる [1], [4], [5]．なお，ここではソフトウェアのソースコードファイルのことを，開発者が再利用を行う単位としてソフトウェア部品，あるいは単に部品と呼ぶ．

一方でインターネットの普及により，SourceForge [12] などのソフトウェア開発に関する情報を交換するコミュニティが誕生し，大量のソースコードが簡単に入手できるようになった．これらの公開されている大量の部品の中から，開発者の必要と

している機能を持つ部品や，必要としているものと似た機能を持つ部品のような，再利用に有益な部品の情報を得ることができる．

ところで，再利用を効率よく行うには様々なソフトウェア部品が各々どれほどの重要度を持つかということを定量的に評価することが重要になる．個々のソフトウェア部品の重要度を評価する方法には様々なものがあるが，その中の一つとして，利用実績に基づいて各部品の重要度を測定し，順位付けし，評価する手法 (Component Rank 法, CR 法) [16] がある．また，この手法により測定された重要度を元に検索された部品を評価し順位付けし，選別して表示する部品検索システム (Software Product Archiving, analyzing, and Retrieving System ,SPARS) [17] なども開発されている．

CR 法における重要度についての基本的な考え方は，以下の (1)〜(3) の通りである．(1) ソフトウェアを構成する部品間には相互に利用関係がある．(2) 一般に，時間が経過し，多くのプロジェクト開発で再利用などが行われるに連れて部品の利用関係は変化していく．(3) 十分な時間が経過した状態のもとで，被利用数が多い部品は重要である（再利用性が高い）．また，重要な部品から利用されている部品も重要である（再利用性が高い）．

このような評価手法は，様々な分野において採用されている．論文の引用解析の分野においては，1970 年代には実績に基づいて論文の重要度を評価する手法が提案されている．Narin らは，論文の重要度を論文がどの程度引用されているかという実績に基づいて評価する手法「Influence Weight」を提案している [8], [10], [11]．この手法では，(1) 多くの論文から引用されている論文は重要である，(2) 重要な論文から引用されている論文はまた重要である，という 2 つの考えに基づいて論文の重要度を評価している．また，よく知られている Web ページサーチエンジン Google では，多くのページ良質なページからリンクされているページはやはり良質なページであるという再帰的な関係をもとに，あらゆるページの重要度を評価している [9], [14]．

システムを構築する際には，過去に開発したシステムの部品の一部を再利用して開発が行われることがよくある．これらの部品は，ライブラリとして再利用される他に，コピーされたり，コピーされた上で大小様々な変更が加わった上で再利用されることが考えられる．そのため，様々なソフトウェアから部品を集めた場合，その部品集合にはコピーした部品や，コピーして一部を変更しただけの部品（類似部品）が数多く存在すると考えられる．

このとき，異なるシステムをまたいで類似部品が現れる場合を考える．再利用という観点から考慮すると，これらの類似部品は類似部品中のある一つの部品をコピーして作成されたと推測することができる．そのため，それらの類似部品間に利用関係を定義するのが妥当であると考えられる．ライブラリとして再利用された場合には，部品間の利用関係を定義するのは簡単であるが，コピーによる再利用がなされた場合はコピー元の特定が難しく，利用関係を定義するのは難しい．

そこで，類似部品をグループ化し，部品群それぞれに対して重要度を与える．このとき各部品の重要度は，その部品が属する部品群の重要度とする．その際，それぞれの類似部品への利用関係が，グループ化を行うことで一つの部品群への利用関係とみなされる．そのため，コピーして再利用される回数が多いほど，その部品群はたくさんの部品から利用されることになる．グループ化によりコピーされた部品の重要度が高くなるため，コピーされたという利用実績を重要度に反映させることができる．

グループ化を行う際には，2 つの部品がどの程度類似しているかを定量的に評価するために部品間の類似度 (Similarity) を評価する．

これまでは，ソフトウェアの類似度比較にはソースコードの最長共通部分列を発見するアルゴリズム LCS [6], [7], [13] を用いた diff [3] やそれを利用したツール [15] が用いられてきた．しかし，この手法は一回当たりの計算コストが高い上に，複数の部品の中から類似した部品を検出するには，全通りの部品対に対して類似比較を行わなければならないので，大量の部品の中から類似部品を探すときにはコストが現実的でない．そこで本論文では，類似度メトリクスを予め抽出しておき，その値の比較をすることにより部品間の類似判定を行う手法を提案し，ツール Luigi を試作した．この手法により，従来の文字列比較による類似判定と同様の解析精度を保ったまま，計算コストを大幅に下げることが可能となる．

以降，2. 節では，メトリクスを用いた類似度測定手法における類似の定義と判定方法について述べる．3. 節では，主メトリクスを用いた比較回数の効率化手法について述べる．4. 節では，適用実験について述べる．最後に，5. 節では，本論文のまとめと今後の課題について述べる．

2. メトリクスを用いた類似度測定ツール

本節では Java プログラムソースから抽出した類似度メトリクスに基づいたソフトウェア部品間の類似度測定について説明する．まず，類似度の定義を説明した後，具体的な類似度判定方法について述べる．

2.1 類似度の定義

本研究では，プログラムの類似度を比較するのに，構文解析時に予め取得しておいた静的特性メトリクスのみを用いる．それらを用いて以下の 2 つの視点から Java プログラム部品の類似度を定義する．

- 構成トークン類似度

Java プログラムの表層的特徴の一つとして，そのソースコードに記述されているトークンの種類別使用頻度が考えられる．本手法におけるトークンとは，ソースコード上での意味のある単語，つまり言語仕様にづく全ての予約語，演算子，記号，識別子を指す．変数名やメソッド名は全て一つの識別子トークンという範疇に集約される．具体的な名前は変更されてもプログラムの動作に影響しないため，本類似度比較において名前の違いは考慮しない．ソースコードとはトークンの集合であると考えられるので，構成しているトークンの数と種類が良く似ているプログラム同士は類似度が高いといえる．構成トークン類似

度の分析方法として，トークンの各出現頻度をメトリクスとして用いる．

- 複雑性類似度

プログラムソースコードの構造的複雑さの特徴を表すために，サイクロマチック数，クラス当たりのメソッド宣言数，ネストの深さなどをメトリクスとして採用する．

2.2 類似度メトリクス

前節で定義した 2 種類の類似度を算出するため，Java プログラムにおける類似度メトリクスとして，Java ソースコードにおける各トークンの出現頻度を用いた構成トークン類似度メトリクスと複雑性類似度メトリクスを定義した．以降，この 2 つのメトリクスについて説明する．

- **構成トークン類似度メトリクス**

Java ソースコードのトークンは大きくわけで 4 つに分類でき，それぞれ以下の (1)〜(4) に定義する．そして，各メトリクスの値の合計を，トークンの総出現回数を用いたメトリクス **Ttotal** として保存する．この時，メトリクス **Ttotal** は，プログラムのサイズのメトリクスとして捉えることができる．既存のメトリクス測定手法では，サイズのメトリクスに LOC(Line Of Code) が使われるのが一般的であったが，本手法ではコメントや空行，改行位置などのプログラマによる違いは，プログラムの動作には何の影響も及ぼさないので無視し，それらの影響を全く受けないトークンの総出現回数のみをサイズのメトリクスとして採用する．これにより，性質上小さなサイズのプログラムが多く，本来のプログラムサイズに対するコメント文や空行，改行位置の影響を受けやすかったオブジェクト指向言語プログラムに対しても，精度の高い解析が可能となる．

（1） **予約語の出現回数を用いたメトリクス**

Java ソースコードに対して，Java の言語仕様で定められた 49 種類の予約語の各出現回数をメトリクスとして記録する．

なお，Java の言語仕様で定められた予約語の中にはこの 49 種類以外に const と goto の 2 つの予約語があるが，これらには機能が実装されていないため，本手法においてはスペースや改行記号などと同じくプログラムに影響を及ぼさないものとみなし類似度メトリクスには含まない．

（2） **記号の出現回数を用いたメトリクス**

Java ソースコードに対して，プログラム中で使用される 9 種類の記号の各出現回数をメトリクスとして記録する．対象となる記号は表 1 に示されるものである．

（3） **演算子の出現回数を用いたメトリクス**

Java ソースコードに対し，Java の言語仕様に基づいてプログラム中で使用される 36 種類の演算子の各出現回数をメトリクスとして記録する．

（4） **識別子の出現回数を用いたメトリクス**

Java プログラムに対し，識別子（変数名やメソッド名など）の延べ出現回数を，メトリクス **NOidentifier** として記録する．なお，違う変数名やメソッド名もそれぞれ区別することなく，ただ一種類の識別子としてカウントする．これは，プログラムをコピーした際に識別子の名前だけを変えても，プログラムとしての動作に何ら影響を与えないので名前に関する差異はプログラムの類似度という視点からは意味を持たないと考えるからである．

表 1 構成トークン類似度メトリクス (記号)

メトリクス名	説明
NOLPAR	記号”(”の出現数
NORPAR	記号”)”の出現数
NOLBRACE	記号”{”の出現数
NORBRACE	記号”}”の出現数
NOLBRACK	記号”[”の出現数
NORBRACK	記号”]”の出現数
NODOT	記号”.”の出現数
NOSEMICOLON	記号”;”の出現数
NOCOMMA	記号”,”の出現数

表 2 複雑性類似度メトリクス

メトリクス名	説明	閾値
cyclomatic	サイクロマチック数	0
declamethodnbr	メソッド宣言の数	1
callmethodnbr	メソッド呼び出しの数	2
nestingdepth	ネストの深さ	0
NOclass	クラスの宣言数	0
NOinterface	インタフェースの宣言数	0

- **複雑性類似度メトリクス**

Java プログラムに対し，表 2 に示すメトリクスを，ソースコードの複雑さの指標として記録する．さらにそれぞれのメトリクスに対して，類似判定に関する閾値を設定する．

その他にも様々な類似度メトリクスは考えられるが，ここでは以上のメトリクスを使用した類似度定義について議論する．なお，これらのメトリクスは Java ソースコードの構文解析を行えば容易に求めることができるものばかりである．

2.3 類似の判定方法

- 構成トークン類似度メトリクス

ソースコード P はそれを構成する要素 $\{p_1, \cdots, p_m\}$ の集合であると考え，$\mathrm{P} = \{p_1, \cdots, p_m\}$ と表現する．各 p_i はソースコード P における各トークンの出現数となる．前節で定義した 96 種のトークンメトリクスを基に，二つのソースコード $P = \{p_1, \cdots, p_{96}\}$, $Q = \{q_1, \cdots, q_{96}\}$ に対し，類似の判定方法を説明する．

ソースコード P の全トークン数を Ttotal(P) とすると，

$$\mathrm{Ttotal}(P) \equiv \sum_{k=1}^{96} p_k$$

と表せる．

さらにソースコード P とソースコード Q の各トークン数の差分の和を diff(P,Q) とすると，

$$\mathrm{diff}(P, Q) \equiv \sum_{k=1}^{96} |p_k - q_k|$$

と表せる．

diff(P,Q) が 0 なら，全く同じ種類のトークンを，全く同じ

回数用いてプログラムを構成していることになるので，コピーした部品であると推測できる．

また，diff(P,Q) が同じ 30 トークンだとしても，全トークンが 40 トークン中の差分が 30 トークンであるのと，10000 トークン中に差分が 30 トークンあるのとでは，その性質が全く違うため，非類似度を求める際には全トークン数で正規化した値を求めるべきである．そこで，部品 P と 部品 Q の非類似度 D(P,Q) を次のように定義する．

$$D(P,Q) \equiv \frac{\mathrm{diff}(P,Q)}{\min(Ttotal(P), Ttotal(Q))}$$

本手法では，$D(P,Q) < 0.03$，つまり部品 P,Q の各要素の差分の合計が全体の 3%以下であるものを構成トークン類似度において類似とみなす．この 0.03 という値は適用実験において，希望する類似部品判定が行えた際の経験則から設定してある．そもそも類似という概念は直感的なものであるので，プロジェクトごとにこの値は随時変更されるべきものであると考える．

- 複雑性類似度メトリクス

複雑性類似度メトリクスに関する類似判定法は，各メトリクスに対する部品 P,Q 間の差分が全て表 2 の中で設定した閾値以内に収まっていれば部品 P,G は複雑性類似度において類似であると判定する．

そして，この 2 種類の類似判定のどちらにおいても類似と判定された部品 P,Q のみを最終的に類似部品であると判定する．

3. ハッシュを用いた効率化手法

前節ではメトリクスを用いた類似度判定の手法を述べたが，この手法だけでは複数の部品に対して類似部品を検出するのに，全部品と類似度比較をしないといけない．そこで，まず類似度メトリクスの中から類似判定の必要条件となっているメトリクスを主メトリクスとして採用し，図 1 のように各部品とその部品が持つ主メトリクス値を対応付けたデータベースを作成し事前分類することにより，類似判定をする際にハッシュキーが閾値以内におさまる部品だけを対象とすることにより比較回数の効率化を図る．

具体的に図を用いて説明する．図 1 は，主メトリクスとして複雑性類似度メトリクスの中から nestingdepth(ネストの深さ)，cyclomatic(サイクロマチック数)，declamethodnbr(宣言メソッドの数) の 3 つを主メトリクスとして採用し，その 3 つの値の組をハッシュキーとして構築したデータベースの内部構造を表している．また，表 2 からそれぞれの主メトリクスの閾値は（0.0.1）となっている．そして，部品 A の主メトリクスが (10.62.124) なら，データベースでは (10.62.124) のハッシュキーに対応した値がアドレスに部品 A へのポインタとなるようにそれぞれの部品に対してデータベースを構築している．そして今，図 1 のようなデータベースが構築された状態で，新しく主メトリクス値が (10.62.125) の部品 P に対して類似比較を行う場合に，全部品に対して類似比較を行うのではなく，主メトリクスが閾値 (0.0.1) 以内にある部品，つまりハッシュキー

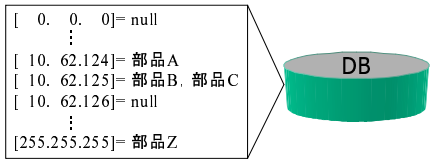

図 1　主メトリクスによるデータベース構築

が (10.62.124)，(10.62.125)，(10.62.126) のいずれかである部品 A, 部品 B，部品 C に対してのみ類似比較を行うことによって，類似比較回数の効率化を図っている．この A,B,C の 3 つの部品以外はたとえ類似比較をしたとしても，主メトリクスに採用した 3 つのいずれかのメトリクスによる類似比較で「類似ではない」と判断されてしまうので，比較しなくても最終的な類似判定の結果に影響は現れない．つまり，類似比較精度を落とすことなく計算コストのみを下げることが可能となる．

主メトリクスの候補としては，まず一定の閾値をもつメトリクスである複雑性類似度メトリクスが挙げられる．複雑性類似度メトリクスの値の差分が閾値以内であるかどうかは，類似判定の必要条件であるので，複雑性類似度メトリクスはその閾値と共にそのまま主メトリクスとして採用できる．

しかし，複雑性類似度メトリクスだけを主メトリクスに採用すると，単純な構造の部品がすべて同じメトリクス値をもってしまい，ハッシュキーによる分類が上手くできずに大きな偏りができる．そこで，部品のサイズを表すメトリクス **Ttotal** を主メトリクスとして採用することにより，効率的に部品を分類できる．

そこで，まず **Ttotal** が関係する類似判定の条件について考えてみる．前節の類似度判定条件の一つに相違トークン数の合計が **Ttotal** の 3%以内であると定義した．この時，Ttotal(P) と Ttotal(Q) の差が 3%を超えるもの (ここでは Ttotal(P) ≧ Ttotal(Q) と仮定する) は，たとえ Q の全トークンの各値が P の各値と同じか小さかったとしても，Ttotal の差の数は全て差分となるので，最終的に差分は全体の 3%を超え，類似判定結果は「類似ではない」となる．

このことから，効率化を図るには Ttotal の差が 3%以内の部品だけを対象とすれば良いと考えられる．

しかし，閾値が 3%ということは Ttotal が 30 の時には図 2 のように 29〜31 までのハッシュキーにアクセスすればよいが，Ttotal が 150 の時には 145〜155 までの 11 回のアクセスが必要になる．さらには Ttotal が 10000 の時には 9700〜10300 まで，実に 601 回のアクセスが必要になる．つまりアクセス回数が O(n) になるので，効率化の視点から見ると不適切であり，データベースアクセス回数は定数回が望ましい．

この問題に関する具体的な解決方法として，図 3 のように Ttotal を $f(Ttotal) = \lceil log_{1.04} Ttotal \rceil$ (ただし $f(0) = 0, f(1) = 1$) を満たす関数 $f(Ttotal)$ を用いて Ttotal の値が小さいときには狭く，大きいときには広くマッピングしておく．この関数はある値 X を代入したときの値を n とすると，X-3% < Y < X+3%の範囲のどの値 Y を代入しても，$f(Y)$ の値が $f(X)$ と同じか差がたがだか 1 になるような関数である．つまり，ある

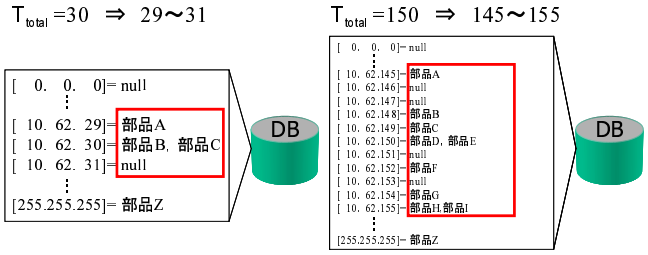

図 2　閾値とデータアクセス回数

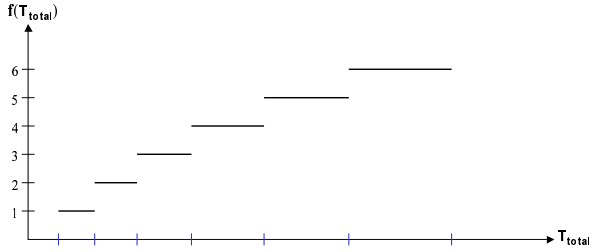

図 3　Ttotal の値をマッピングする関数

部品 X の Ttotal の値を Ttotal(X) とすると，部品 X と類似と判定される可能性をもつあらゆる部品 Y について，$f(Y)$ と $f(X)$ の差は高々 1 に収まる．このようなマッピングを行えば，$f(Ttotal)$ を閾値 1 の主メトリクスとして採用することが可能となり，データベースアクセス回数は 3 回に固定することができる．

これにより，構成トークン類似度に関するメトリクスも主メトリクスに採用することができるようになり，偏りの少ない効果的な事前分類が可能となった．

4. 適 用 実 験

本節では，提案手法を実現した Java プログラム間類似度測定ツール Luigi を用いて適用実験を行い，提案手法の有効性を評価する．実験においては，Sun JDK 1.3 内の 431 個のクラス群に対して Luigi を適用する．主メトリクスを定義することで適用対象を絞り込むことができるが，実験においては，次のような 4 通りの絞り込みに関する定義を用意し，それぞれの条件において Luigi を適用した．

（1） 主メトリクスを定義しない

この場合，一つの部品について全てのクラスに対して類似判定を行う．

（2） サイクロマチック数による分類

（3） サイクロマチック数およびメソッド数による分類

これらの場合，制御構造類似度の一部の判定を満たしている部品のみが事前抽出される．

（4） サイクロマチック数およびメソッド数およびクラスタによる分類

この場合，制御構造類似度による基準に加えて，構成トークン類似度に関する類似判定を満たす可能性がある部品のみが事前抽出される．

以上の 4 つの条件に対するする適用実験の結果を表 3 にしめす．表中の主メトリクス欄における記号 C はサイクロマチック数，記号 M は宣言メソッド数，記号 T はトークン数による事前分類が行われたことを指し，それぞれ前述した分類方法に対応している．

表 3　適用実験結果

主メトリクス	事前分類クラスタ数	最終クラスタ数	計算コスト (sec)
未使用	1	278	05.02
[C]	21	278	00.56
[C,M]	85	278	00.29
[C,M,T]	232	278	00.16

事前分類クラスタ数とは主メトリクスを用いて事前分類したときにいくつの部品群に分かれたかを示す．未使用の場合は，主メトリクスによっては分類されなかったとし，1 としている．主メトリクスに C を用いた場合は，431 のクラスが 21 の部品群に分類され，各は部品群は平均 21 個の部品から構成されている．以下，分類条件が細かくなるほど，細かく分類されることがわかる．

最終クラスタ数は，本ツールを用いて解析を行った結果，構成トークン類似度および制御構造類似度の両方の判定において類似と判定されて最終的にいくつの類似部品群に分かれたかを示している．主メトリクスとして採用された事前分類条件が構成トークン類似度および制御構造類似度の判定において，類似であるとみなすのに必要な条件を破綻させるものではなかったので，最終クラスタ数は全て同じとなった．計算コストは，本ツールの開始から終了までの時間を計ったものである．

以下では，解析結果の妥当性および解析コストの考察を行う．

4.1 妥当性の考察

本節では，解析結果の妥当性について考察する．

まず，最終クラスタ 278 個を手作業で調べたところ，コピーした部品，またはコピーして一部変更した部品が部品群としてまとまっていることが確認された．また，文字列比較を用いた類似度測定ツール SMMT [15] による分析結果との比較においても，精度の差は多少あるものの，高い相関が得られた．これにより，ソースコード部品がコピーされたかどうかを判定するためには，文字列を直接比較する手法を用いなくとも，ソースコード中の構成トークンの類似度および制御構造の類似度を測ることで十分であることが確認できた．

次に表 3 から，主メトリクスの種類を増やすに従って事前分類クラスタ数も増加していることが確認できる．これは，主メトリクスが事前分類の条件になっているので，主メトリクスの種類が増えるに従って事前分類条件は細かくなり，結果として事前分類クラスタ数が増えていくからであると考えられる．

さらに，事前分類クラスタ数が変化しても最終クラスタ数は変化していないことがわかる．ここで，全ての主メトリクス条件におけるそれぞれ 278 個の最終クラスタの構成部品を手作業で調べたが全て同じ構成であることがわかった．これは，主メトリクスにおいて採用された分類条件が構成トークン類似度および複雑性類似度の両判定において，類似であるとみなすのに

必要な条件を破綻させるものではなかったからであると考えられる．実際には，主メトリクスを細かくしていくことは，主メトリクスによる絞込みにおいて前倒しで類似判定を行っていることに他ならない．以上のことから，主メトリクスの判定条件を十分考慮することで計算コストのみを下げることができ，解析結果を変化させることは無いことを確認した．

4.2 解析コストの考察

本節では，解析コストについて考察する．

まず，今回の適用実験の対象として採用した JDK1.3 の 431 個の同じクラス群に対して文字列比較を用いた類似度測定ツール (SMMT) を用いて解析した時のコストが 24.35(sec) であった．ツール SMMT は解析対象の部品に対し，全通りの組み合わせについて比較計算を行っている．

一方で，主メトリクスを用いなかった場合のツール Luigi による解析コストは 5.02(sec) であり，十分計算コストが小さくなっていることがわかる．主メトリクスを用いなかった場合のツール Luigi は，SMMT と同じように解析対象の部品に対し，全通りの組み合わせについて比較計算を行っている．このことから，本手法は文字列比較を用いた手法と比べ，類似比較一回当たりの計算コストが小さくなったことがわかる．

さらに，主メトリクスを用いたツール Luigi の解析結果を見ると，主メトリクスによる分類条件を増やすに従って事前分類クラスタ数は増えていき，計算コストは小さくなっていくのがわかる．これは主メトリクスを設定することにより，解析対象の部品が事前分類され，比較計算をする部品数が絞り込まれたからである．細かく事前分類すればするほど，比較計算の対象となる部品数は絞り込まれるので，結果として比較計算回数が減少し，計算コストは小さくなる．実際には，各事前分類クラスタの構成部品数にばらつきがあり，きれいな比例関係とはならないが，主メトリクスによる事前分類クラスタ数と計算コストは反比例している．以上のことから，主メトリクスを用いた事前分類によって，類似比較計算回数を減らすことができることがわかる．

今回の適用実験では，主メトリクス [C,M,T] による事前分類を行った後にメトリクス比較により類似度を判定する手法は，文字列比較を用いた類似度測定を行った場合に比べて，計算コストを 1/150 にまで下げることができている．さらにこの手法の場合，分類を行うべきソフトウェア部品の数が膨大になった場合にも，主メトリクスの採用数を増やすことによってより細かい事前分類が可能になるので，大規模な部品の集合にも対応が比較的容易である．このことから，本提案手法は文字列比較を用いた類似度測定手法ではコストが現実的でなかった大規模なシステムにおいても使用することが可能となり，大きな威力を発揮すると期待できる．以上のことから，本手法は文字列比較を用いた類似度測定手法よりも十分低コストかつ効率的な手法であることを確認した．

5. ま と め

本論文では，Java ソースコードの部品情報を数値化した類似度メトリクスの比較を用いた類似度測定方法を提案した．さらに，複数のソフトウェア部品の中から類似した部品対を検出し，部品群にまとめる機能を実装したツールの試作を行った．その中で，まずメトリクス値のみを用いて類似比較を行うことにより比較一回当たりの低コスト化を行い，さらに主メトリクスによる事前分類を導入し，比較計算回数の効率化も行った．最後に JDK 1.3 のソースコードから 431 個のクラスを対象とした適用実験により既存の文字列比較を用いた測定手法と比較して，同様の解析を低コストで実現できることを示した．今後の課題としては，さらに多くの視点から類似度メトリクスを測定し，解析精度を高めていくことと，解析に要する計算コストをさらに下げること，また，Java 以外の言語への対応などがある．

文　献

[1] V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: “The software engineering laboratory - an operational software experience”, Proc. of ICSE14, pp. 370-381 (1992).

[2] C. Braun: Reuse, in John J. Marciniak, editor, Encyclopedia of Software Engineering, Vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).

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

[4] S. Isoda: “Experience report on a software reuse project: Its structure, activities, and statistical results”, in Proceedings of 14th International Conference on Software Engineering (ICSE14), pp.320-326, Melbourne, Australia,(1992).

[5] B. Keepence and M. Mannion: “Using patterns to model variability in product families”,IEEE Software, Vol. 16, No. 4, pp. 102-108 (1999).

[6] W. Miller and E. W. Myers: A file comparison program. Software- Practice and Experience, Vol. 15, No. 11, pp. 1025–1040,(1985).

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

[8] F. Narin, G. Pinski, and H. H. Gee: “Structure of the Biomedical Literature,” Journal of the American Society for Information Science, Vol. 27, No. 1, 25-45, (1976).

[9] L. Page, S. Brin, R. Motwani, T. Winograd: “The PageRank Citation Ranking: Bringing Order to the Web”, http://www-db.stanford.edu/ backrub/pageranksub.ps

[10] G. Pinski and F. Narin: “Citation Influence for Journal Aggregates of Scientific Publications: Theory, with Application to the Literature of Physics,” Information Processing and Management, Vol. 12, No. 5, pp. 297-312, (1976).

[11] G. Pinski and F. Narin: “Structure of the Psychological Literature,” Journal of the American Society for Information Science, Vol. 30, No. 3, pp. 161-168, (1979).

[12] SourceForge: “http://sourceforge.net/”.

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

[14] 馬場肇: “Google の秘密 - PageRank 徹底解説”, http://www.kusastro.kyoto-u.ac.jp/ baba/wais/pagerank.htm

[15] 山本哲男, 松下誠, 神谷年洋, 井上克郎: “ソフトウェアシステムの類似度とその計測ツール SMMT”, 電子情報通信学会論文誌 D-I, Vol. J85-D-I, No.6, pp.503-511, (2002).

[16] 横森励士, 藤原晃, 山本哲男, 松下誠, 楠本真二, 井上克郎: “利用実績に基づくソフトウェア部品重要度評価システム”, Technical Report of SE Lab, Dept. of Computer Science, Osaka University,(2002).

[17] 横森励士, 山本哲男, 松下誠, 楠本真二, 井上克郎: “利用実績に基づくソフトウェア部品検索システム SPARS-J”, 第二回クリティカルソフトウェアワークショップ（WOCS2003）, NASDA 筑波宇宙センター, 筑波市, (2003).