研究速報

コードの静的特性を利用したJavaソフトウェア部品類似判定手法

小堀　一雄[†a)]　山本　哲男[††]（正員）
松下　誠[†]　井上　克郎[†]（正員）

Java Program Similarity Measurement Method Using Token Structure and Execution Control Structure

Kazuo KOBORI[†a)], *Nonmember*,
Tetsuo YAMAMOTO[††], *Member*,
Makoto MATSUSHITA[†], *Nonmember*, and
Katsuro INOUE[†], *Member*

[†] 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
[††] 立命館大学情報理工学部情報システム学科，草津市
College of Information Science and Engineering, Ritsumeikan Univerisity, 1–1–1 Nojihigasi, Kusatsu-shi, 525–8577 Japan
a) E-mail: koborik@nttdata.co.jp

あらまし　本論文では，Javaソースコードの静的特性を利用したソフトウェア部品類似判定手法を提案する．本手法を用いることで，大量のソフトウェア部品の中から互いに類似する部品群を高速に抽出することができ，コピーアンドペーストで再利用した部品の発見を迅速に支援することが可能である．

キーワード　類似，再利用，メトリックス，Java

1. まえがき

ソフトウェア部品の再利用は，生産性と品質を改善し，開発コストを削減できることが知られている[1]．ここで，どの部品間で再利用が行われたかという関係を把握することは，保守・運用の面から見て重要である．この関係は，ソースコードの文字列比較で抽出できるが，計算コストが高く，大量の部品に対しては不向きである．これに対し，ソースコードから抽出されるメトリックス値の比較による部品間類似判定の研究[2]も行われている．そこで我々は，高速に部品間の類似判定を行うことを目的として，Javaで記述された部品を対象に，予約語の出現頻度等に注目したメトリックス値比較と，ハッシュを用いた事前分類を組み合わせる類似判定手法を提案する．また実際に高速な部品間類似判定ができることを適用実験を通じて示す．

2. ソフトウェア部品類似判定手法

本手法は，Javaのクラスをソフトウェア部品と定義して，類似判定を行う．まず大量の部品を収集し，各部品ごとに類似度メトリックス計算を行う．類似度メトリックスとはのちに定義する，部品の類似判定に適した2種類の静的なメトリックスのことである．次に，我々が定義する類似判定の必要条件となっているいくつかのメトリックス値の組をキーとし，対応する部品群のIDを値とするハッシュを作成する．以上の環境を構築した後，ある部品に対して，その部品と同じハッシュキーをもつ部品群に対してのみ，すべての類似度メトリックス値の比較計算を行う．このハッシュの利用により，比較計算をしても類似と判定されない部品をあらかじめ比較対象から外すことができ，高速化が実現できる．最後に，両方の類似度メトリックスで類似と判定された部品を，類似部品として関係付けて出力する．

2.1 類似度メトリックス

本手法において，部品の類似判定に用いる二つの静的メトリックスについて説明する．

2.1.1 トークン出現回数メトリックス

トークン出現回数とは，Java言語仕様で定められた49種類の予約語，9種類の記号，1種類の識別子（合計59種類）がプログラム中に出現した回数をいう．これらは，ソースコードを構文解析することで得られる．

トークン出現回数を用いた類似度を次のように定める．49種類の各トークンの出現回数が部品P，Q間で異なったとき，その出現回数の差分の総延べ数をdiff(P, Q)とする．また，部品Pの全トークン出現回数の総延べ数をTtotal(P)とする．このとき，部品P，Q間の類似度D(P, Q)を以下の式とする．

$$D(P,Q) \equiv 1 - \frac{diff(P,Q)}{min(Ttotal(P), Ttotal(Q))}$$

このとき，D(P, Q) $\geq$ 0.97ならば，その部品対はトークン出現回数の観点から類似であると判定する．0.97というしきい値は経験的に得られた値である．

2.1.2 構造的複雑度メトリックス

構造的複雑度とは，部品内における条件分岐の数や，メソッド宣言の数といったプログラム制御構造の複雑さをいう．これらは，ソースコードを構文解析することにより抽出する．類似判定法については，表1において採用したすべての構造的複雑度メトリックスの部品間の差が，対応するしきい値を超えていなければ，類似とする．なお，各しきい値は経験的に得られた値である．

2.2 ハッシュの利用

本手法では類似判定の高速化のためにハッシュを利用する．まず，類似度メトリックスの中から類似判定

表 1　構造的複雑度メトリックス
Table 1　Structural complexity metrics.

メトリックス名	説明	しきい値
cyclomatic	サイクロマチック数	1
declamethodnbr	メソッド宣言の数	1
callmethodnbr	メソッド呼出しの数	2
nestingdepth	ネストの最大深さ	0
NOclass	継承するクラスの数	0
NOinterface	実装するインタフェースの数	0

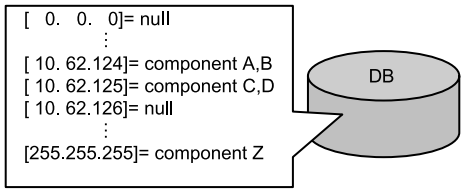

図 1　主メトリックスと部品を対応させたハッシュテーブル
Fig. 1　Hash table of major metrics and components.

の必要条件となっているいくつかのメトリックスを選択する（以降，選んだメトリックスを主メトリックスと呼ぶ）．次に，それらの主メトリックス値の組をキーとし，そのような主メトリックス値をもつ部品の ID を値とするハッシュテーブルを作成する（図 1）．このテーブルの例では，三つの主メトリックスを選択し，それぞれの値の組合せ（例えば，各主メトリックス値を 0 から 255 に正規化したとき，0.0.0〜255.255.255 になる）をキーとしている．

このハッシュを利用して，類似判定の対象をしきい値以内のキーに属する部品に限定することにより，詳細な類似判定を行っても類似とは判定されない部品を事前に判定対象から除外する．これにより，計算コストを減らすことができる．

3. 評価実験

3.1 実験内容

提案手法を実現した Java プログラム部品間類似判定ツール Luigi [4] を用いて適用実験を行い，その有効性を評価した．実験対象は，Sun JDK 1.3（java.lang, java.io, org.omg.CORBA などのパッケージ）内の 431 個のクラス群である．ハッシュについては，以下のような 4 通りの主メトリックスの組合せを試した．なお，サイクロマチック数とはプログラムコードの処理分岐の総延べ数のことを表す．

- 未使用
- サイクロマチック数（以降 C）
- C 及びメソッド宣言数（以降 M）
- C, M 及びトークン出現回数メトリックス（以降 T）

表 2　適用実験結果（速度）
Table 2　Experiment result (speed).

主メトリックス	ハッシュクラスタ数	最終クラスタ数	計算コスト (s)
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

表 3　適用実験結果（精度）
Table 3　Experiment result (accuracy).

全部品数	類似部品数 (Luigi)	類似部品数 (SMMT)	類似部品数 (Luigi ∩ SMMT)
431	201	199	156

3.2 結果と考察

適用実験を行った結果を表 2，表 3 に示す．本節では，まず事前分類の結果について考察した後，類似判定の精度と速度について既存の類似度計測ツール SMMT [3] と比較することで評価する．特に適合率と再現率について評価を行った後，適合あるいは再現しなかった部品に対して考察を行う．最後に，類似判定の速度を比較する．

3.2.1 事前分類について

本手法では，任意の主メトリックス値の組合せを用いて事前分類を行い，その結果をもとにハッシュテーブルを構築する．表 2 中のハッシュクラスタ数とは，この事前分類において同じグループに分類された（同じキーの組合せをもった）部品群のグループ数を示している．ここでは，主メトリックスの数を増やすに従って事前分類が進んでいることが表 2 から見て取れる．また，最終クラスタ数とは，Luigi が最終的に類似と判定し，まとめられた部品群のグループ数を示している．主メトリックスは類似判定の必要条件の一部であるので，ハッシュクラスタ数に関係なく最終クラスタ数は同じになっている．

3.2.2 判定精度の比較について

本手法の類似部品判定精度に関しては，SMMT と Luigi とで同じ部品群に対して類似判定を行い，SMMT の判定結果に対する Luigi での判定結果の適合率と再現率を算出することで評価した．SMMT は既に高い類似判定精度をもつことが評価済みなので，上記の適合率と再現率が高い値になれば Luigi の精度が高いことに対する評価になると考える．

3.2.3 適合率について

3.1 で述べた 431 個の部品群を対象に実験したとこ

ろ，表 3 に示すように，SMMT が分類した類似部品は全部で 199 個あった．このうち，Luigi の分類結果と完全に一致した部品は 156 個であった．このとき，適合率は 156/199 = 0.783 となる．類似した部品を検索する目的においては，SMMT が示す類似部品のうち，約 8 割が同じようにヒットすることとなり，この適合率は十分高い値と考える．

3. 2. 4 再現率について

表 3 に示すように，431 個の部品の内，Luigi が分類した類似部品は全部で 201 個（最終クラスタ数は 278 グループ）であった．よって，再現率は 156/201 = 0.776 となり，こちらも高い値を示した．

3. 2. 5 適合外の部品の扱いについて

適合しなかった部品を調査したところ，コピーしてメソッドを増やしたような部分的な高い類似性をもっているものが多いことが分かった．この原因として，Luigi では部品全体のメトリックス値のみを扱うという高速化を図ったため，このような部分的に高い類似性を評価できないというトレードオフがあることが判明した．本研究ではクラス単位のコピーアンドペーストの補足を目的としているため，この問題には対応しなかったが，今後更にメソッド単位のコピーアンドペーストを補足するためにはメソッド単位でのメトリックス比較が必要になると考える．

3. 2. 6 未再現の部品の扱いについて

再現しなかった部品について調査したところ，非常に規模の小さい（総トークン数が 10 数個以内）部品が多いことが分かった．このような小さい部品に関しても Luigi は類似判定の対象としていたのに対して，SMMT は類似判定の対象としていなかった．これは純粋に，小さい規模の部品に対する類似判定基準が違っていることが原因であった．

以上の評価結果から，コピーアンドペーストを用いた Java ソフトウェア部品の再利用関係を補足する目的においては，本手法で提唱したメトリックスを用いた比較を行うことで十分精度の高い結果が得られることが分かった．

3. 2. 7 判定速度の比較について

表 2 に示すように，主メトリックスとして C，M，T の組を用いることで，計算コストを 0.16 秒まで軽減できた．これは，同じ実験対象に SMMT を適用した際のコストである 24.35 秒に比べ約 1/150 に短縮されており，本手法が従来手法に比べ，十分高速であることが確認できた．

4. むすび

本研究では，Java ソフトウェア部品の類似度メトリックスを用いた部品間類似判定法を提案した．更に，本手法に基づく類似判定ツール Luigi を用いて適用実験を行った．その結果，従来の文字列比較法より高速に類似判定が行えることが分かった．今後の課題として，類似判定の精度に関する詳細な分析や，Java 以外の言語への適用などが挙げられる．

文　献

[1] C. Braun, “Reuse,” in Encyclopedia of Software Engineering, ed. J.J. Marciniak, vol.2, pp.1055–1069, John Wiley & Sons, 1994.

[2] K.A. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M. Bernstein, “Pattern matching for clone and concept detection,” Automated Software Engineering, vol.3, pp.77–108, 1996.

[3] 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,” 信学論（D-I），vol.J85-D-I, no.6, pp.503–511, June 2002.

[4] 小堀一雄，山本哲男，松下　誠，井上克郎，“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,” 信学技報，SS2003-2, 2003.

（平成 18 年 5 月 23 日受付，8 月 21 日再受付）