特別研究報告

題目

利用関係に基づく類似度を用いた Java コンポーネント分類ツールの作成

指導教員
井上 克郎 教授

報告者
佐々木 裕介

平成 21 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 20 年度 特別研究報告

利用関係に基づく類似度を用いた Java コンポーネント分類ツールの作成

佐々木 裕介

内容梗概

私の所属する研究グループは，ソフトウェアの再利用を促進することを目的として，コンポーネント検索システム SPARS-J を開発した．コンポーネントとはソフトウェア再利用の単位であり， SPARS-J では Java のクラスをコンポーネントとして扱っている．SPARS-J の検索対象となるコンポーネント集合には，同じように使えるコンポーネント (類似コンポーネント) が複数含まれている．類似コンポーネントの例としては，同じコンポーネントの異なるバージョンや，他プロジェクトから流用されたコンポーネントが挙げられる．もし検索結果で大量の類似コンポーネントが別々に表示されると，開発者が適切なコンポーネントを選択するのが難しくなってしまう．そこで， SPARS-J には，類似コンポーネントをグループ化することを目的としたサブシステム Luigi がある．Luigi は多数のコンポーネントを高速に分類するために構造を表すメトリクス値のみを用いてコンポーネントの比較を行っている．しかし Luigi は構造が同じであれば類似コンポーネントでなくとも同じグループに分類してしまうなどといった問題を抱えている．

そこで本研究では，Luigi の抱える問題を解決することを目的として，コンポーネント間の類似判定手法及びそれを用いたコンポーネント分類手法を提案する．提案するコンポーネント分類手法では，Luigi とは違う類似判定基準として，コンポーネントの相互の利用関係の一致度と名前の一致度を用いることにした．さらに提案するコンポーネント分類手法を実装したコンポーネント分類ツール Twigi をコンポーネント検索システムに実装した．提案した類似判定により Luigi の問題が解決されていることを確認するため，Luigi と Twigi の比較実験を行なった．実験では Twigi が Luigi の問題を解決している例が多く見つかった．一方で Twigi が正しく分類できないコンポーネントも存在したため，改善の余地があることもわかった．

主な用語

利用関係
類似度

コンポーネント

目 次

1　まえがき

ソフトウェア開発の生産性を向上させるための技術の一つとして，コンポーネントの再利用に注目が集まっている．コンポーネントとはソフトウェアを構成する一つの単位であり，具体的にはソースコードやドキュメント，ライブラリなどのことを指す．既存の質の高いコンポーネントを再利用することができれば，短期間で高品質なソフトウェアを開発することができる．

そこで私の所属する研究室では，コンポーネントの再利用を促進するためのシステムとして Java コンポーネント検索システム SPARS-J(Software Product Archive，analysis and Retrieval System for Java)[1] を開発した．コンポーネント検索システムとは，コンポーネントの登録・検索ができるシステムのことである．SPARS-J は Java 言語で書かれたソースコード中のクラスをコンポーネントとして扱う．

再利用に向いているコンポーネントを優先して開発者に推薦するために，SPARS-J では登録されているコンポーネントに対して Component Rank(以下，CR) 法による順位付けを行っている．CR 法とは，コンポーネントのグループ化を行った後にコンポーネント同士の利用する・利用されるといった利用関係からグラフを作成し，他のコンポーネントからの利用が多いコンポーネントほど高い順位となるように順位付けをおこなう手法である．CR 法によって算出される CR 値は，開発者が利用関係に沿って参照を行うと仮定した場合の各コンポーネントの参照されやすさを表しており，よく利用されるコンポーネントや，重要なコンポーネントから利用されるコンポーネントの CR 値は高くなる．

SPARS-J ではバージョン違いのコンポーネントや，他プロジェクトの流用によって生成されるような同じように利用できるコンポーネント（以下，類似コンポーネント）をまとめて表示することを目的として，CR 値の計算を行う前にコンポーネントのグループ化を行い，グループ毎に CR 値の算出を行う．コンポーネントのグループ化には，既存の研究 [2] で作成されたコンポーネント分類ツール Luigi が利用されている．Luigi とは，SPARS-J に登録されているような多数のコンポーネントに対する類似判定を短時間で行うことを目的としたコンポーネント分類ツールである．高速化のための工夫として，Luigi はコンポーネントの構造を表すメトリクス値のみを用いることで，比較一回あたりの類似判定時間を削減している．そして Luigi の高速化の工夫が有効であることは既に実証されている [2]．一方で，Luigi の類似判定手法にはいくつか問題点があることが判明している．例えば Luigi は，構造に少しでも差異があるコンポーネントは類似コンポーネントであっても違うグループへ分類する．今回行った Luigi の評価実験でもこの問題は確認された．

そこで本研究では Luigi の問題を解決した分類を行うために，類似コンポーネントを分類することを目的とした類似判定手法を新たに提案する．本研究で提案する類似判定手法で

は，コンポーネントを利用する側から見た基準として利用関係の一致度を採用している．さらにコンポーネント間の役割の違いを判別するために，コンポーネントの名前の一致度も測定する．利用関係の一致度，名前の一致度を測定した後に二つの一致度を組み合わせて類似か類似でないかを判定する．

利用関係の一致度測定では，比較している 2 コンポーネント間のインタフェースの一致度と呼び出しているコンポーネント集合の一致度をそれぞれ計算する．ここで述べているインタフェースとは，あるコンポーネントが他のコンポーネントに対してアクセスする時のインタフェースとなるコンストラクタやメソッド，フィールドのインタフェースのことを指す．そして測定した 2 つの一致度の内小さい値を利用関係の一致度として採用する．

名前の一致度測定では，比較している 2 コンポーネントに対し，コンポーネントとなる Java クラスのパッケージ名と単純名，単純名に含まれるキーワードの一致具合から名前の一致度を決定する．

さらに提案した類似判定手法を実装したツールを SPARS-J の後継である SPARS/R 上に作成し，Luigi の問題が解決されているかどうか確認した．結果として，Luigi が問題のある類似判定を行っているコンポーネントに対し，提案手法を実装したツールが正しい判定結果を下している例を確認することができた．しかしその一方で，実装したツールが類似コンポーネントではないコンポーネントを同じグループへ分類している判定結果も確認された．

以降，2 節ではコンポーネント検索システム SPARS-J と，SPARS-J で使用されている順位付け手法 CR 法について簡単な説明を行う．3 節では SPARS-J が類似コンポーネント分類に用いている類似度測定ツール Luigi と，Luigi が類似コンポーネントを正しく分類できるかどうかを調べるために行った実験について説明する．4 節では類似コンポーネントを分類することを目的とした類似判定の手法を提案する．5 節では 4 節で提案した類似判定を用いたコンポーネント分類手法について説明する．6 節では提案した類似判定手法及びコンポーネント分類手法を実装したツール Twigi について説明する．7 節では Twigi を評価するために Luigi と Twigi に同じコンポーネント集合を入力として与え，分類結果が Luigi と比較して改善できているか評価する．8 節では関連研究として，既存の論文における類似度定義とその用途について具体的な例を挙げ説明する．9 節では 7 節で行った評価実験の結果も踏まえ，Twigi 及び提案した類似判定手法に関する今後の課題について述べる．

2　コンポーネント検索システム SPARS-J

本節では Java コンポーネント検索システム SPARS-J[1] について説明する．

2.1　コンポーネント検索システム

コンポーネント検索システムとは，コンポーネントの登録・検索ができるシステムのことである．

検索者が多数のコンポーネントから効率的に自分の目的としているコンポーネントを探し出すことができるよう，コンポーネント検索システムでは検索者の目的に応じたコンポーネント探索を可能にしている．

2.2　SPARS-J の概要

SPARS-J(Software Product Archive，analysis and Retrieval System for Java)[1] はコンポーネント再利用を促進するためのシステムとして，私の所属する研究室で研究・開発されている Java コンポーネント検索システムである．

SPARS-J は Java のクラスやインタフェースのソースコードをコンポーネントとして扱う．検索時には内部に登録済みのコンポーネントを対象として検索を行う．

SPARS-J ではコンポーネントの検索時に複数のコンポーネントが検索結果として見つかることを想定して，再利用に向いているコンポーネントを優先して開発者に推薦するために登録されたコンポーネントに対して順位付けを行っている．順位付けを行うことで，検索結果を表示する際には順位付けで上位にきたコンポーネントから優先して表示する．

コンポーネントの順位付けには利用例の多いコンポーネントが上位に来るような順位付け手法，CR 法を用いている．

2.3　Component Rank 法

Component Rank(CR) 法は利用例の多いコンポーネントから上位にくるようにコンポーネントの評価付けを行うコンポーネント評価手法である．

利用例の多いコンポーネントを上位にランク付けするような評価を行うために，CR 法ではコンポーネント間の利用関係を解析する．利用関係とはコンポーネントが互いに利用し，利用される関係のことをさす．

具体的には，コンポーネントを頂点，利用関係を有向辺としたグラフ（コンポーネントグラフ）を構築する．例えば図 1 のコンポーネント C がコンポーネント A,B を利用している場合は，C から A,B に対して有向辺を引くことができる．A,B が互いのコンポーネントを利用している場合には，A から B，B から A にそれぞれ有向辺を引く．

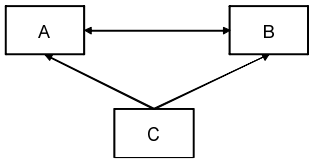

図 1: グラフの一例

CR 法ではこのグラフから読み取られる利用関係を解析し，利用例の多いコンポーネントから上位にくるようにコンポーネントの評価付けを行う．

具体的には，このグラフを表す隣接行列における固有値 1 の固有ベクトルを求め，ベクトルの各要素の値を CR 値とする．CR 法では この CR 値を部品の評価値としており，CR 値が高いコンポーネントほど重要なコンポーネントあるいはよく利用されているコンポーネントである．

2.4 コンポーネントのグループ化

SPARS-J が登録しているコンポーネントには，異なるバージョンのコンポーネントや，他プロジェクトから流用されたようなコンポーネントも多数含まれている．これらのコンポーネントが，検索結果で別々に表示されると，開発者が適切なコンポーネントを選択するのが難しくなってしまう．そこで SPARS-J では類似コンポーネントをまとめて表示することを目的として，CR 値の計算を行う前にコンポーネントをコンポーネント群としてグループ化する．

さらにグループ化の結果は CR 値の計算にも利用される．具体的には CR 値をこのグループ化の結果から得られるコンポーネント群グラフから算出する．例えば図 2 においてコンポーネント群 X に属しているコンポーネント A が他のコンポーネント群 Y に属しているコンポーネント B を利用している場合，コンポーネント群 X がコンポーネント群 Y を利用していると考えることができる．図 2 のように各群を頂点，その間の利用関係を有向辺としたグラフをコンポーネント群グラフと呼ぶ．CR 値を計算する際にコンポーネント群グラフを 2.3 のコンポーネントグラフの代わりに用いることで，グループ化した結果も CR 値に反映させることができる．

2.5 本研究との関連

本研究では，SPARS-J で利用されているコンポーネント分類ツール Luigi の問題点を解決するために，Luigi とは異なる基準で類似判定を行うコンポーネント分類手法を提案している．Luigi の具体的な問題点については 3.5 節において記述する．さらに提案したコンポー

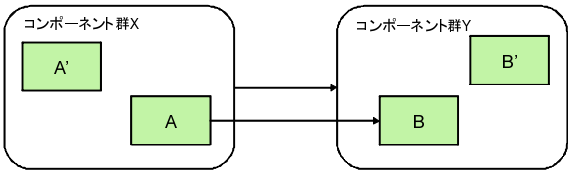

図 2: コンポーネント群の利用関係

ネント分類手法を SPARS-J の後継システム SPARS/R に実装し，Luigi との比較実験を行った．実験の結果については 7 節に記述する．

3　コンポーネント分類ツール Luigi

本節では SPARS-J がコンポーネントを分類するために用いているコンポーネント分類ツール Luigi[2] について説明を行う．

Luigi とは，SPARS-J に登録されているような多数のコンポーネントに対する分類を短時間で行うことを目的としたコンポーネント分類ツールである．Luigi は入力として渡された各コンポーネント間の類似判定を行い，グループ化すべきだと判定されたコンポーネントの集合を同じグループへ分類する．Luigi の類似判定は，ソースコードから抽出されたメトリクス値のみを用いて定義された，コンポーネント間の類似度メトリクスを測定することにより行われる．

今回報告者は Luigi の手法で類似コンポーネントを正しく分類できるかどうか評価するための実験を行った．結果として，Luigi の手法では一部の類似コンポーネントを正しく分類できないことが確認された．実験の詳細については 3.5 節で説明する．

3.1　Luigi で定義されている類似判定

Luigi は Java コンポーネント間の類似度メトリクス（以下，類似度）を測定し，類似判定を行う．類似度とは類似判定の定量的基準を数値として定義したものであり，Luigi で定義している類似度には Java プログラムのソースコードから抽出された様々な静的特性を表すメトリクス値が使われている．

具体的には，以下の二種類の数値メトリクスを用いて，比較している Java コンポーネントの対に対してそれぞれについて類似判定を行う．

- 複雑性メトリクス
- トークン構成メトリクス

まず複雑性メトリクスの類似判定について説明する．複雑性メトリクスには具体的には図 3 に表される数値メトリクスが用いられている．Luigi では各複雑性メトリクスに対して許容差を閾値として設定している．実際に開発者がコンポーネントを転用する場合には，転用先でコードを書き換えることが少なくない．そこで Luigi では変更を許容する範囲を閾値として定め，数値として指定できるようにしているのである．

例えば図 3 のメソッド呼び出し数の閾値は 2 であるため，ある 2 コンポーネントを比較したときにそのメソッド呼び出し数の差が 2 を超えている場合，この 2 コンポーネントは類似とはみなされない．

次にトークン構成メトリクスの類似判定について説明する．トークン構成メトリクスに使用されるメトリクスの構成は図 4 のようになっている．図 4 のトークンの具体例として予約

メトリクス名	閾値
サイクロマチック数	0
メソッドの宣言数	1
メソッド呼び出し数	2
ネストの深さ	0
classの数	0
interfaceの数	0

図 3: Luigi が用いている複雑性メトリクス一覧

トークン構成
メトリクス：ソースコードにおける各トークンの出現数
トークン ＝ 予約語 + 記号 + 演算子 + 識別子
（96種） （49種）（9種）（37種） （1種）

図 4: Luigi が用いているトークン構成メトリクス

語には例えば“ class ”というトークンが含まれる．

このトークン構成メトリクスの比較時に，Luigi ではコサイン尺度を用いて評価を行う．コサイン尺度とは，数学分野においてある空間における 2 ベクトル $\vec{A}, \vec{B}$ 間の角度を求める際に用いられる次の数式のことである：

$$\cos \vec{A}, \vec{B} = \frac{\vec{A} \cdot \vec{B}}{|A||B|}.$$

コサイン尺度は類似度として，しばしば二要素間の類似性を評価する際に利用される．図 5 に表されるように，Luigi ではあるコンポーネント A,B を比較したとき，A,B から抽出されたトークン構成メトリクスを構成する各メトリクス値をベクトルとみなしてコサイン尺度を計算し，それを用いて類似度 $D(A, B)$ を算出する．$D(A, B)$ が，独自に設定した閾値 97%以下の場合には非類似と判定，97%を越える場合には類似と判定する．

ある 2 コンポーネントを比較したときに，複雑性メトリクスによる類似判定，トークン構成メトリクスによる類似判定両方において類似と認められた場合のみ，Luigi はこの 2 コンポーネントを同じグループへ分類する．

3.1.1 Luigi の類似判定の性質

Luigi の使用している類似度計算法では大きさの小さなコンポーネント間で高い類似度を示す傾向がある．このため Luigi が実際に行う判定処理では，トークン数が 500 未満のコンポーネントはすべて単独のコンポーネントとして扱う．

さらに Luigi は，一部の複雑性メトリクスが一致した時点で完全限定名が一致するコンポーネントを類似であると判定する．完全限定名とは，そのコンポーネントが属するパッ

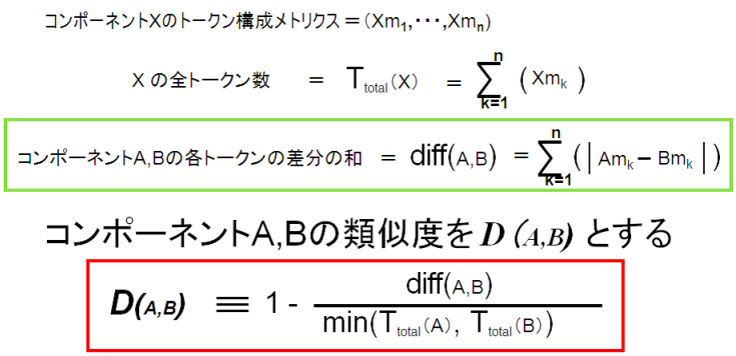

図 5: トークン構成メトリクスの類似度

ケージまで含めたコンポーネントの識別名のことを指す．Java の場合にはパッケージ名. 単純名で表される．完全限定名が一致するコンポーネントは，そのコンポーネントが含まれているパッケージごと転用されているかバージョン違いである可能性が高い．

また [2] によると，Luigi の複雑性メトリクスの類似判定に利用されている閾値及びトークン構成メトリクスの類似判定に利用されている閾値は，実験などにより経験的に決められたもので，合理的な説明はされていない．

3.2 高速化の工夫

次に Luigi で行われている高速化のための工夫について述べる．Luigi は，SPARS-J のような大規模なデータベースに登録されている多数のコンポーネントを対象とすることを目的としているため，類似判定を高速化するために様々な工夫を行っている．

3.2.1 比較回数の削減

Luigi は比較する対象となるコンポーネントを，メトリクス値を用いたハッシュテーブルを用いた前判定を行うことにより減らしている．図 6 に示すように，Luigi は予め各コンポーネントに対して以下の作業を行っておく．

1. メトリクス値をコンポーネントから抽出
2. 一部のメトリクス（主メトリクス）の値を利用してハッシュ値を作成
   Luigi ではハッシュ値の基となるメトリクスをサイクロマチック数 (CYC)，トークン数 (NOT)，メソッド数 (NOM) と定め，主メトリクスと呼んでいる
3. ハッシュ値をアドレスとしてコンポーネントを DB へ登録

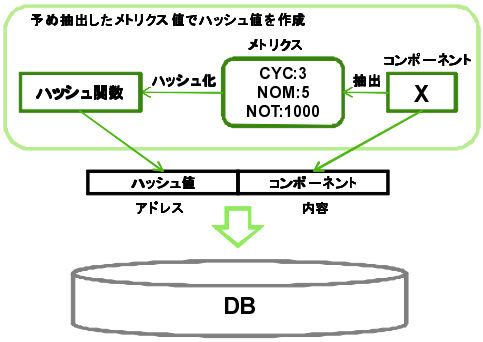

図 6: 主メトリクスを基にした DB へのコンポーネント登録

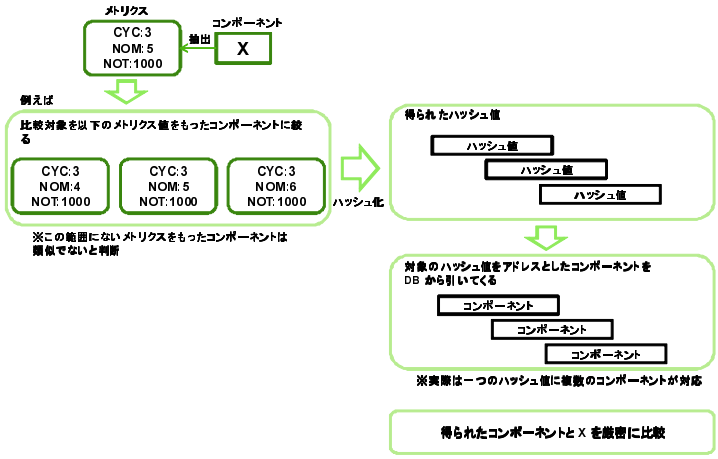

図 7: メトリクスによる比較範囲の範囲限定

そして図 7 に示すように，比較時には以下の作業を行い比較対象となるコンポーネントの範囲を絞る．

1. コンポーネントからハッシュ値の基となったメトリクス（主メトリクス）を抽出する
2. 抽出したメトリクスから，比較対象とするコンポーネントが持つべきメトリクス値の範囲を限定する
3. 範囲を限定したメトリクス値からハッシュ値を生成する
4. 生成したハッシュ値を基に比較対象のコンポーネントを呼び出す

実際にはコンポーネントから抽出したメトリクス値は予め別の DB に登録しておくため，Luigi はメトリクス値を抽出する作業の代わりに，登録しておいたメトリクス値を呼び出す作業を行っている．

このように，Luigi はメトリクス値をハッシュ値としたハッシュテーブルを利用することにより，比較対象となるコンポーネントの数を減らし，比較回数を削減している．

3.2.2 比較一回当たりの計算時間短縮

Luigi は数値メトリクスを用いることで比較一回あたりの計算時間を短縮している．
コンポーネント A,B を文字列による単純比較で比較する場合には最悪の場合，

(A に出現する文字列) $\times$ (B に出現する文字列)

に比例する時間がかかる．しかし Luigi の場合には，予め抽出されている数値メトリクスから類似度を計算するだけで良いため，

(各トークン構成メトリクスの比較時間) + (コサイン尺度の比較時間)

で一回の比較を終えることができる．各メトリクス値の比較時間は $O(1)$ であるため文字列比較より高速な比較が行える．

3.3 Luigi の処理手順

高速化を施した後の Luigi の比較処理概要と，Luigi のコンポーネント同士の単体比較における類似判定の手順を以下で説明する．

3.3.1 Luigi の比較処理概要

Luigi の比較処理概要を以下に示す．
すべてのコンポーネントについて

1. 対象のコンポーネントからメトリクス値の抽出，データベース登録
2. データベースからメトリクス値をハッシュ値として比較対象となるコンポーネントを選出
3. 対象のコンポーネントと選出した各コンポーネントを比較
   (a) データベースからメトリクス値を参照，類似度を算出
   (b) 類似性が認められれば比較しているコンポーネント同士を同じグループへ分類

2. において比較対象となるコンポーネントを限定することで，比較回数の削減を行っている．比較一回当たりの計算時間削減は 3.(a) により行われている．

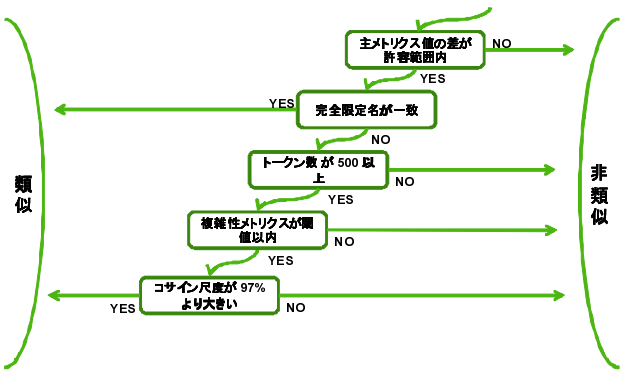

図 8: Luigi の類似判定手順

3.3.2 Luigi の類似判定手順

コンポーネント同士の単体比較における Luigi の類似判定手順を図 8 に示す.

まず比較対象となるコンポーネントの絞込みで，比較しているコンポーネントとの主メトリクス値の差が閾値以内でないコンポーネントは比較対象から除外されるため，非類似と判定される.

その後に，トークン数の判定，完全限定名の一致判定が他のメトリクス値による類似判定に先駆けて行われる.

最後に複雑性メトリクスに対する判定，トークン構成メトリクスに対する判定が行われ，類似・非類似が決定される.

3.3.3 高速化の有効性

前述した工夫により高速化が実現できたことを実証すべく，既存の研究 [2] では JDK [15] に使用されている 431 コンポーネントを対象として，文字列比較を用いた類似度測定ツール SMMT[3] との比較実験が行われている．結果，SMMT がすべての計算に 24.35 秒 を要したのに対して，Luigi は精度を落とすことなく 0.16 秒 で計算を終えるという結果を得ている．これにより Luigi に施された工夫が類似度測定ツールの高速化に有用であることがわかる.

3.4 類似判定の有効性

既存の研究 [2] では，複雑性メトリクス及びトークン構成メトリクスを用いた類似度を測定することにより，コンポーネントを数値メトリクスのみを用いて二つの側面から比較することができているとしている．また定義した類似度を用いて既存の類似度測定ツール SMMT[3]

の類似測定結果と高い相関を得られたともしている．

一方で，Luigi はトークン数が 500 未満のコンポーネントをすべて単体のコンポーネントとして扱っている．ゆえにトークン数が 500 未満のコンポーネントの中に類似コンポーネントが含まれている場合，Luigi はこの類似コンポーネントを分類することができない．

さらに 3.5 節の評価実験では，Luigi が類似コンポーネントを正しく分類できないケースがあることを確認している．

3.5 Luigi 評価実験

Luigi の類似判定手法で，類似コンポーネントを正しく分類できるかどうか評価するために行った実験について説明する．

本実験の目的は 3.3.2 の図 8 に示される Luigi の類似判定が類似コンポーネントの分類において妥当であるか評価することである．そのために，実際に Luigi にコンポーネントの集合を入力として与え，分類の結果を手作業で確認する．入力するコンポーネントの集合は JDK1.4, JDK5.0[15] の一部で使用されている 20569 個のコンポーネントである．また本実験で確認する内容は次の項目である：

- 入力の中に類似コンポーネントの組があった場合に同じグループへ分類しているか
- 同じグループの中に類似コンポーネントでないコンポーネントの組が存在していないか

3.5.1 実験結果

実験を行い次のような Luigi の分類結果を確認した：

- 類似コンポーネントを正しく分類できなかったケース 7 個
- 類似コンポーネントを正しく分類できたケース 9 個

今回は正しく分類できなかったケースを重点的に探したためこのような結果になっているが，実際には正しく分類できるケースの割合はもっと多いと考えられる．類似コンポーネントを正しく分類できなかったケースについて具体的な例を次に示す．

まず，類似コンポーネントとして同じグループへ分類すべきだが，Luigi が別のグループへ分類した例を示す：

- javax.swing.text.html.HTMLDocument の新旧 2 バージョン
  非類似とされた具体的要因：メソッドの追加によるサイクロマチック数の不一致

- javax.xml.namespace.QName の新旧 2 バージョン
  非類似とされた具体的要因：メソッド分離のリファクタリングによるサイクロマチック数の不一致

非類似とされた要因であるメソッドの追加・分離は，コンポーネントがコピーにより転用された先で容易に起こりうることである．このためこの例に示す入力は類似コンポーネントとして分類されるべきであるが，図 8 の主メトリクス値による判定で非類似と判定された．続いて Luigi が類似コンポーネントとして同じグループへ分類しなかった別の例を示す：

- org.omg.CORBA.IRObjectOperations と
  com.sun.org.omg.CORBA.IRObjectOperations

この 2 コンポーネントの内容は同一とみなせるが，図 8 の判定でトークン数が 500 未満であったため非類似と判定された．次に別のグループに分類すべきだが，Luigi が類似コンポーネントとして同じグループへ分類した例を示す：

- java.lang.Short と java.lang.Byte
  類似と判定された要因：名前は違うが，定数の数値などを除いて内容に違いがみられない

Short,Byte は意味的な違いから別のグループへ切り分けておくべきと考えられる．

3.5.2 評価実験まとめ

これらの分類例が存在することから，Luigi の類似判定では一部の類似コンポーネントを類似であると判断できないこと，また類似コンポーネントとして判断すべきでないコンポーネントを類似であると判断する場合があることがわかった．

4　類似判定手法の提案

本節では類似コンポーネントを正しく分類するための類似判定手法を提案する．

3.5 節の評価実験により，Luigi の類似判定では類似コンポーネントを正しく分類することができないことがわかった．そこで本研究では，Luigi の類似判定手法とは異なる方法で類似コンポーネントを判定する類似判定手法を提案する．

4.1　類似コンポーネントの特徴

提案する類似判定手法では，類似コンポーネントがもつ以下の特徴を利用している：

- 類似コンポーネント同士の利用関係は類似
- 類似コンポーネント同士の完全限定名は類似

これら二つの特徴が認められるコンポーネント対は，類似コンポーネントである可能性が非常に高い．それぞれの特徴については次に詳しく述べる．

4.1.1　利用関係が類似

類似コンポーネント同士の利用関係は類似である．前述したとおりコンポーネント間には利用関係が存在するため，類似コンポーネントにも利用関係が存在する．例えば図 9 の場合にはコンポーネント A を利用するコンポーネント，またコンポーネント A に利用されるコンポーネントが存在する．

もし図 9 においてコンポーネント B がコンポーネント A の類似コンポーネントであれば，コンポーネント A とコンポーネント B は同様に利用できるはずである．この時 A と B を比較すると以下の条件が満たされるべきである：

- 提供する機能及びインタフェースが類似
- 利用しているコンポーネント集合が類似

この時 A,B は利用関係が類似であると言える．このため利用関係が類似している 2 コンポーネントは，類似コンポーネントである可能性が高い．

4.1.2　完全限定名が類似

類似コンポーネント同士の完全限定名は類似である．類似コンポーネントが生成されるとき，完全限定名の変更度合によって類似コンポーネントを以下の三通りに分けられる：

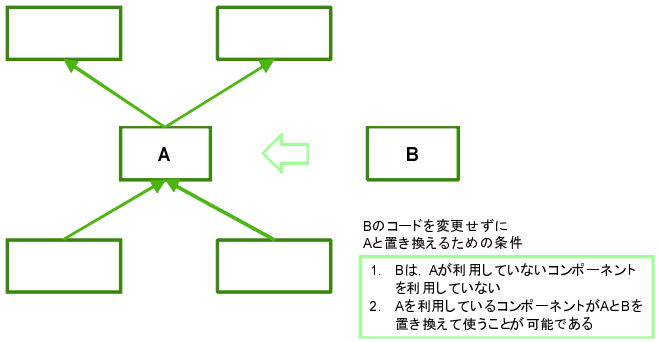

図 9: 類似コンポーネントの利用関係

完全限定名が変更されていない類似コンポーネント

異なるバージョンのコンポーネントであると考えられる．特に大きな変更がない限り，役割などが変化している可能性は低い．

単純名が変更されていない類似コンポーネント

単純名を変更せずに，他のパッケージから転用していると考えられる．

単純名が変更されている類似コンポーネント

転用先で単純名を変更したと考えられる．ただし単純名に含まれているキーワードは，そのクラスの持つ役割を表現していることが多いことから，単純名に含まれるキーワードが残されている可能性がある．

完全限定名の類似性が高くなるほど類似コンポーネントである可能性が高くなることがわかる．

4.2 類似判定手法の提案

前述した類似コンポーネントの特徴を踏まえた類似判定手法を提案する．

あるコンポーネントとあるコンポーネントを比較したとき，そのコンポーネント対が類似コンポーネントであるとする類似判定として，以下の手順を用いることを提案する：

- 完全限定名の一致度測定を行った上で，
- 利用関係の一致度測定
  - インタフェースの一致度測定
  - 提供する機能の一致度測定
  - 利用しているコンポーネント集合の一致度測定

各判定の詳細について以降で述べる．

4.3 完全限定名の一致度測定

本手法では比較しているコンポーネント対に対して利用関係の類似判定を行う前に，そのコンポーネント対の完全限定名の一致度を，次の 3 通りに判定する：

Level1 完全限定名が一致している場合

Level2 単純名のみ一致している場合

Level3 単純名が不一致であるが共通のキーワードを含む場合

Level1 から Level3 へ移るほど類似コンポーネントである可能性は低くなる．このため Level1 から Level3 へ移るほど利用関係の類似判定を厳しくする．例えば比較しているコンポーネントの一致度が Level2 であった場合に，利用関係の類似判定において最低一致度として用いる閾値に設定する数値を 50%と設定するこの場合，Level3 で用いる利用関係に用いる閾値には Level2 で用いている閾値 50%より高い値を設定する．利用関係に用いる閾値については後述する．

4.4 インタフェースの類似判定

外部のコンポーネントからアクセスできるのは private 以外の修飾がなされているコンストラクタ，フィールド，メソッド (以下，メンバと呼ぶ) のみであるから，インタフェースの類似判定にはコンポーネントに定義されている各メンバのインタフェースの一致度を用いる．

さらにスーパークラスとなるコンポーネントのメソッドなどは外部へ提供されている可能性がある．このためインタフェースが類似している条件にはスーパークラスとなるコンポーネントの一致も含む必要がある．

そこでインタフェースの類似判定では以下の判定を行う：

- 各メンバのインタフェースの一致度が閾値として定めた最低一致度以上である
- スーパークラスとなるコンポーネントの単純名が一致している

上記の全条件を満たしていればインタフェースが類似であると判定する．

4.4.1 各メンバのインタフェース一致度

各メンバのインタフェースの一致度については割合で算出する．割合で算出する理由は，閾値がメソッド数の大きさに左右されないようにするためである．メンバのインターフェースの一致度 p は下記のように定義する：

- 比較するコンポーネントを A，B としたとき，

$$p \equiv \frac{A,B \text{ を比較したときにインタフェースの一致するメンバ数}}{\max((A \text{ の定義しているメンバ数}),(B \text{ の定義しているメンバ数}))} \leq X$$

メンバのインタフェースが一致するとは，具体的には下記の 3 要素がすべて一致することである：

- インタフェースの名前
- 渡す引数
- 戻り値の型

ただしコンストラクタに関してはインタフェースの名前を考慮する必要はない．理由は Java においてコンストラクタの名前は単純名の名前と同じだからである．あるコンポーネントが定義しているすべてのメンバのインタフェースが一致する場合，そのコンポーネントのインタフェースは一致していると言える．

閾値 X が高くなるほど求められるインタフェースの類似性は高くなる．左辺の最大値は 1 であり，この時 A と B のインタフェースは一致する．

4.5 利用しているコンポーネント集合の類似判定

利用しているコンポーネント集合の類似判定では具体的に以下の判定を行う：

- 利用しているコンポーネントの単純名と，利用しているメソッド名 (以下，リファレンス名) の集合の一致度が定められた閾値以上である

4.1.1 節にて前述したとおり，類似コンポーネントの利用関係が類似している条件の一つとして，利用しているコンポーネント集合の類似が挙げられる．そして利用しているコンポーネントの単純名及びメソッド名は，ソースコードの解析により確実に読み取ることができる情報である．

4.5.1 リファレンス名集合の一致度

あるコンポーネントとあるコンポーネントを比較したときのリファレンス名集合の一致度 p は割合で算出している．具体的には下記のように定義する：

- 比較する単純名を A，B としたとき，

$$p = \frac{((A \text{ のリファレンス名集合}) \cap (B \text{ のリファレンス名集合})) \text{ の要素数}}{((A \text{ のリファレンス名集合}) \cup (B \text{ のリファレンス名集合})) \text{ の要素数}}$$

$p \leq X$ のとき，A,B は一致するという．

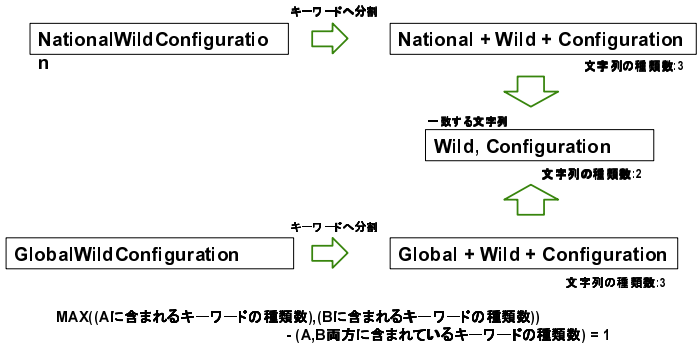

図 10: 単純名の類似判定例

4.6 機能の類似判定

機能の類似判定では具体的には以下の類似判定を行う：

- 単純名が類似
- リファレンス名集合の一致度が定められた閾値以上

単純名には機能がわかりやすい名前が推奨されるため，一般に単純名に含まれているキーワードはそのコンポーネントの機能を表している．そのため，単純名に含まれるキーワードの一致度が高ければ機能も類似しているものと考える．また実装する機能によって利用する機能も偏るため，リファレンス名集合も類似する．

単純名の類似条件は次のとおりである：

- コンポーネントの単純名に含まれるキーワードに一語以上の差異が認められない

例えば図 10 に示すように単純名 NationalWildConfiguration を一語だけ変更して GlobalWildConfiguration として転用されたコンポーネントは，この類似条件を満たすことになる．

4.7 類似判定を行うことができないコンポーネント

提案した手法では次のコンポーネントに関しては類似判定を行うことができない：

- メンバを全く含まないコンポーネント

メンバを全く含まないコンポーネントには比較できる要素が完全限定名以外に存在しない．よってこの種のコンポーネントは例外として，完全限定名のみ比較することにする．

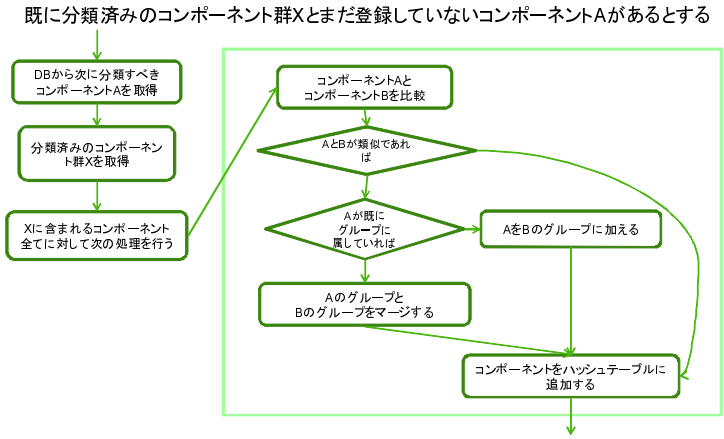

図 11: コンポーネント分類処理概要

5　提案手法を利用したコンポーネント分類手法

本節では前述した類似判定を用いてコンポーネントを分類する手法について説明する．

まずコンポーネント分類処理の概要について述べ，さらにその分類処理に対して Luigi で用いられている高速化手法を適用した場合の分類処理について述べる．

5.1　コンポーネント分類処理の概要

コンポーネント分類処理の概要を図 11 に示す．

図 11 においてコンポーネント A はグループへ未分類であり，新たに分類すべきコンポーネントである．A は類似判定を一度も行っていない状態である．一方コンポーネント群 X はすべてのコンポーネントが類似判定及びグループへの分類を終えている状態である．

A をコンポーネント群 X に含まれるすべてのコンポーネントと 1 対 1 で比較していき，もしコンポーネント A と類似と判定されたコンポーネントが X の中に見つかった場合には，見つかったコンポーネントが属しているグループへ A をマージする．この処理により A はすべてのコンポーネントと類似判定を行うことができ，類似と判定したコンポーネントを一つのグループへまとめることができる．

この手法では，A を分類済みのすべてのコンポーネントと比較しなければならないため時間がかかる．そこでこの手法に対し Luigi の高速化を適用する．高速化を適用する方法について説明する．

5.2 Luigi の前判定による高速化の適用

Luigi の高速化の工夫を前述した分類処理に対しどのようにして適用しているか説明する．Luigi では比較対象となるコンポーネントを削減するために，前判定による比較対象の絞り込みを行っていた．具体的には，コンポーネントから抽出したメトリクス値をハッシュ値としたハッシュテーブルを用いていることで，比較する対象となるコンポーネントを絞り込んでいた．ハッシュ値に数値メトリクスを用いることで，類似であると予測できるコンポーネントに当たりをつけることができるのである．

本研究で作成したツールでも Java 上でハッシュテーブルを利用することで比較回数の削減を図っている．Java には HashMap というクラスが存在し，このクラスを利用するとハッシュテーブルを簡単に実装することが可能である．ただし本研究で作成したツールは Luigi とは違い，二種類のハッシュテーブルを利用する：

- キーワードをハッシュ値とするハッシュテーブル
  単純名に少なくとも一つは対象コンポーネントと同じキーワードが含まれているコンポーネントのみを選出できる
- メトリクス値をハッシュ値とするハッシュテーブル
  メトリクス値の一致度が対象コンポーネントと比較したときに閾値以内であるコンポーネントのみを選出できる

各ハッシュテーブルの詳細は 5.2.1 節，5.2.2 節で説明する．またメトリクス値の一致度については 5.2.1 で説明する．これらのハッシュテーブルは具体的には以下のように使用する．今新たに分類すべきコンポーネントを A としたとき：

1. 対象コンポーネントを A と指定して，各々のハッシュテーブルからコンポーネント群を取得
2. 得られた二つのコンポーネント群の積集合をとる

積集合をとることで，A と比較したときに次の二つの条件を満たしたコンポーネント群を得ることができる：

- 単純名に少なくとも一つは A と同じキーワードが含まれている
- メトリクス値をハッシュ値とするハッシュテーブルにおいて利用されているメトリクス値の一致度が，A と比較したときに閾値以内である

後は取得したコンポーネント群に含まれているすべてのコンポーネントと A を，提案した類似判定により比較すればよい．

5.2.1 メトリクス値をハッシュ値とするテーブル

メトリクス値をハッシュ値とするテーブルでは，ハッシュ値にコンポーネントから抽出したメトリクス値を使用して，分類済みのすべてのコンポーネントを予め登録しておく．ハッシュ値とするメトリクス値には，類似コンポーネント対において大きく違ってはならない次のメトリクス値を利用する：

- インタフェース関連のメトリクス
  - コンストラクタ数
  - フィールド数
  - メソッド数
  - スーパークラスとなるコンポーネントの単純名から得られるハッシュ値
    実際には単純名の文字列 (String) に相当する数値 (Int) をハッシュ値として利用している．
- 機能関連のメトリクス
  - トークンの種類数
    トークンの種類数はコンポーネントに実装されている機能によって偏る．トークンの種類数が大きく違うコンポーネントは類似コンポーネントとは言えない
- トークン数
  トークン数は規模を表す．規模が大きく違うコンポーネントは類似コンポーネントとは言えない

上記すべてのメトリクス値をハッシュ値に含めることで，上記すべてのメトリクス値の一致度が閾値以上であるコンポーネントを比較対象として取得することができる．これによりすべてのコンポーネントを比較対象とする場合に比べて，大幅に類似判定の回数を減らすことができる．

比較対象とするコンポーネントを A として与えると，このハッシュテーブルは A と上記のメトリクス値が類似と判断されるコンポーネントをすべて取得してくる．ハッシュテーブルへのコンポーネントの登録・取得については 3 節の図 6，図 7 を参照のこと．

なお，メトリクス値の一致度 p が閾値 X 以上であるとは以下を満たすことと同意であると定義する：

- 比較するコンポーネントを A,B としたとき，

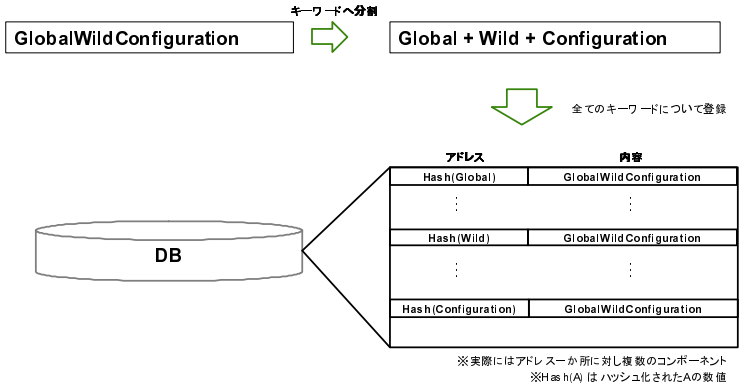

図 12: キーワードをハッシュ値にしたハッシュテーブルへの登録

$$p \equiv \frac{\min((A\text{ のメトリクス値}),(B\text{ のメトリクス値}))}{\max((A\text{ のメトリクス値}),(B\text{ のメトリクス値}))} \leq X$$

A,B のメトリクス値が一致する場合，左辺の値は最大値の 1 となる．

5.2.2 キーワードをハッシュ値とするテーブル

キーワードをハッシュ値とするテーブルはキーワードを用いた前判定を行うために使用する．具体的には対象コンポーネントと比較したときに次の条件を満たすコンポーネントを取得するために用いる：

- 単純名に少なくとも一つは対象コンポーネントと同じキーワードが含まれている

キーワードをハッシュ値とするハッシュテーブルでは，ハッシュ値にコンポーネントの単純名から抽出したキーワードを使用して，分類済みのすべてのコンポーネントを予め登録しておく．

具体的にはハッシュ値は以下のものとなる：

単純名に含まれるキーワードから得られるハッシュ値

単純名に含まれているキーワードが複数であれば，その単純名を持つコンポーネントは複数箇所に登録される．例えば図 12 の場合，Global,Wild,Configuration それぞれからハッシュ値を生成し，それらをアドレスとするところすべてにコンポーネントを登録しておく．

コンポーネント取得時には，各キーワードから計算したハッシュ値すべてからコンポーネントを取得する．例えば図 12 の場合，Global,Wild,Configuration それぞれからハッシュ値を生成し，それらをアドレスとするところすべてからコンポーネントを取得する．

このハッシュテーブルを利用することで，単純名に含まれるキーワードが一つ以上一致するコンポーネントを取得できる．

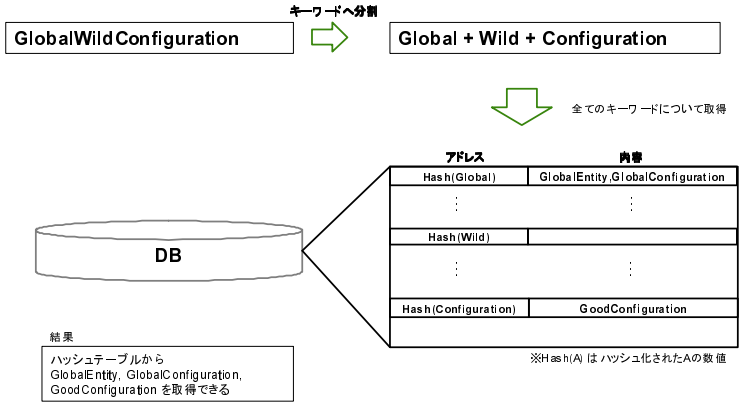

図 13: キーワードによるハッシュテーブルからのコンポーネントの取得

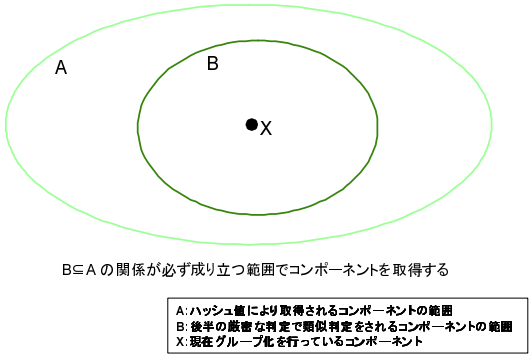

図 14: ハッシュにより取得するコンポーネント群と類似判定されるコンポーネント群の包含関係

5.2.3 前判定となる類似判定に課される制限

図 14 に示すとおり，前判定で取得できるコンポーネントの集合は必ず提案した類似判定により取得できる類似コンポーネントの集合を包含するようにしている．

厳密な類似判定とは提案した類似判定のことをさす．この包含関係を守った前判定を行わなければ，後半の厳密な判定でグループ化されるはずのコンポーネントがグループ化できなくなってしまう可能性がある．

5.3 高速化適用後のコンポーネント分類

以上の工夫を施した結果，分類処理の概要は図 15 のようになる．

コンポーネント A と比較すべきコンポーネントを 2 つのハッシュテーブルにより絞り込むことで，類似判定の回数を減らすことができる．

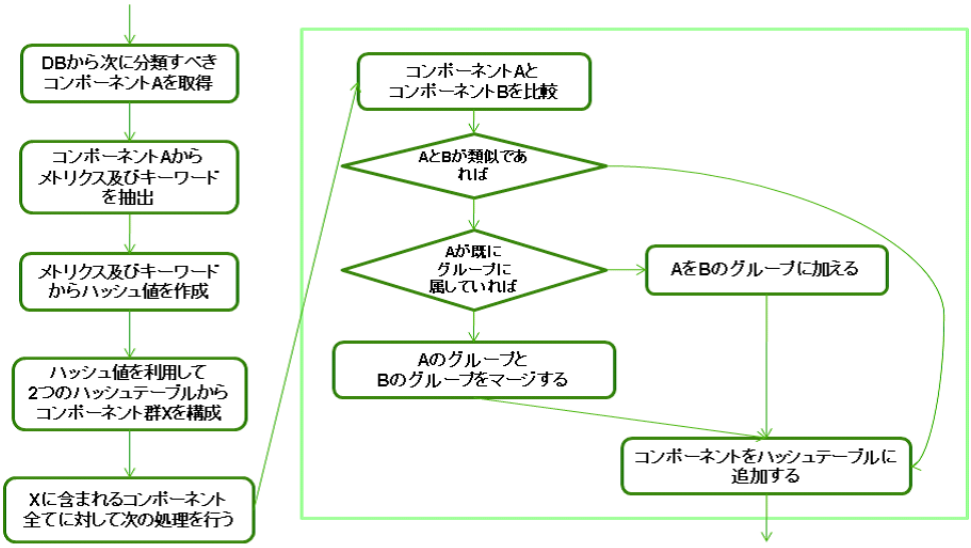

図 15: 高速化適用後のコンポーネント分類処理

6　コンポーネント分類ツール Twigi の実装

本節では SPARS-J の後継 SPARS/R 上に作成したコンポーネント分類ツール Twigi について説明する．前述の類似判定手法を実装したことで，Twigi は Luigi より理想に近い類似コンポーネントの分類を行うことができると考える．

6.1　SPARS/R

SPARS/R は私の所属する研究室で SPARS-J の後継として開発されているコンポーネント検索システムである．SPARS-J は C 言語で実装されていたが，SPARS/R は Java 言語で書かれている．今回作成したツールはこの SPARS/R 上に Twigi という名前でコンポーネント分類ツールとして実装した．記述言語は Java である．

6.2　コンポーネント分類処理

実装したコンポーネント分類処理のフローチャートについては前節の図 15 に示した通りであり，Luigi で用いられていた高速化手法も実装済みである．

6.3　コンポーネント間の類似判定処理

コンポーネント分類処理における図 15 の類似判定部分の詳細は図 16 のようになった．

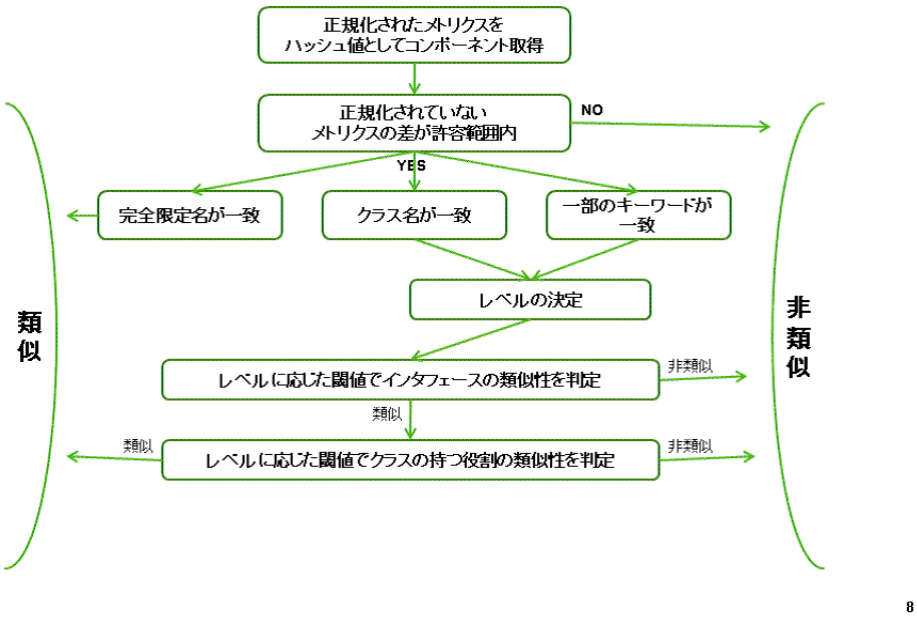

図 16: Twigi 類似判定手順

この判定手順により類似と判定されたコンポーネントは，類似コンポーネントとして同じグループへ分類される．各類似判定については前述の類似判定の提案において既に詳細を述べたため省略する．

7　評価実験

本節では Twigi の類似判定が類似コンポーネントの分類において Luigi の問題を解決できているか確認するために行った，Twigi と Luigi の比較実験について説明する．

7.1　測定する内容

本実験では次の内容を測定する：

3.5 節の実験において Luigi に対して入力したデータセットを Twigi に与えたとき，

- Luigi が類似コンポーネントとして正確に分類できなかったコンポーネントを Twigi は類似コンポーネントとして分類できるか
- Luigi が類似コンポーネントとして正確に分類できていたコンポーネントを Twigi は類似コンポーネントとして分類できるか

7.2　実験環境

使用する PC のスペック及びデータセットは次の通りである：

CPU Intel(R) Pentium(R) 4 CPU 3.00GHz

物理メモリ 300MB

仮想メモリ 2.6GB

入力とするデータセット JDK1.4, JDK5.0 に含まれている 20569 コンポーネント

ハッシュ値に利用するメトリクス値の閾値はそれぞれ次の値に固定する：

- コンストラクタ数の一致度 50%
- フィールド数の一致度 50%
- メソッド数の一致度 50%
- トークン数の一致度 50%
- トークン種類数の一致度 50%

インタフェースの閾値は次の値に固定する：

- 比較しているコンポーネント対の完全限定名の一致度が Lv2 のとき 55%

- 比較しているコンポーネント対の完全限定名の一致度が Lv3 のとき 60%

リファレンス名集合の閾値は次の値に固定する：

- 比較しているコンポーネント対の完全限定名の一致度が Lv2 のとき 55%
- 比較しているコンポーネント対の完全限定名の一致度が Lv3 のとき 60%

7.3 実験結果

Luigi で誤って違うグループへ分類されていた以下のコンポーネント対は，Twigi では類似コンポーネントとして同じグループへ分類されていた：

- javax.swing.text.html.HTMLDocument のバージョン違い 2 コンポーネント
  Luigi でうまくいかなかった要因：メソッドの追加によるサイクロマチック数の不一致
- javax.xml.namespace.QName のバージョン違い 2 コンポーネント
  Luigi でうまくいかなかった要因：メソッド分離のリファクタリングによるサイクロマチック数の不一致
- org.omg.CORBA.IRObjectOperations のバージョン違い 2 コンポーネント
  Luigi でうまくいかなかった要因：トークン数が 500 未満

また Luigi で誤って類似コンポーネントとして同じグループへ分類されていた以下のコンポーネント対は，Twigi では違うグループへ分類されていた：

- java.lang.Short と java.lang.Byte
  Luigi でうまくいかなかった要因：名前は違うが，定数の数値などを除いて内容に違いがみられない

逆に以下のコンポーネント対は Luigi では正しく分類していたが Twigi では正しく分類できなかった：

- java.net.Inet4AddressImpl と java.net.Inet6AddressImpl
  Twigi でうまくいかなかった要因：利用関係に少量の違いしか見られない

このコンポーネント対は Luigi では構造に微量の差異が見られたため類似でないと判定されていたが，この構造の微量の差異は，同様に利用できるコンポーネントの間でもよく見られるため，偶然正しい判断となったケースと考えられる．

7.4 評価実験考察

以上の実験結果から，Twigi は多くの場合 Luigi でうまく分類できなかったコンポーネントを正しく分類できていたが，正しく分類できないコンポーネントも確認されたため，提案した類似コンポーネントの類似判定には改善の余地もあることがわかった.

8　関連研究

類似度はテキスト検索やソフトウェア剽窃検出など幅広い分野で使用されている．本節では関連研究として類似度を利用している既存の論文をいくつか紹介し，最後にそれらの論文で利用されている既存類似度について考察する．

8.1　テキストに関する類似度

増井らは [4] において，テキストに関する類似度として，tf/idf 法に基づく類似度が使用している．tf/idf 法はテキストファイルの類似度を比較する際によく用いられる手法である．この類似度は検索システムにおける類似ファイルの探索に利用されている．

また岸本らは [5] において数式に関する特徴を並べたものをベクトルとして扱い，このベクトル間のコサイン尺度を類似度として利用している．この類似度は数式を検索するシステムの作成に利用されている．

8.2　ソフトウェアに関する類似度

SMMT[3] では diff とコードクローンを用いた類似度を利用して，ソースコード間の類似度を計測するツールを実装している．diff とはソースコード中の行間差分，コードクローンとは 2 ソースコード間に現れる重複した系列のことである．コードクローン及び diff は複数のソースコードから成り立つソフトウェアシステム間で測定できるため，[3] で利用されている類似度は複数のソースコードから成り立つソフトウェアシステム全体に対して適用が可能になっている．

MUDABlue[6] では LSA を用いた類似度が利用されている．LSA はベクトル空間モデルを用いてテキスト中に含まれている単語間の潜在的関連を解析する解析手法である．潜在的関連を解析することで，単語間の類似度を算出することができる．[6] においてこの LSA を用いた類似度はソフトウェア文書中に出現する単語の分類に利用されている．

長橋 [7] は diff を用いた類似度とベクトル空間を用いた類似度二つを定義している．ベクトル空間を用いた類似度では，ソースコード中から文ベクトルを抽出しベクトル間のコサイン尺度を求めることで類似度を測定している．[7] ではこの二つの類似度を，ソースコード保守の際にソースコードから重複部分を削減するときの見積もりに利用できるとしている．

また大野らは [8] において学生のレポートプログラムのソースコードの剽窃検出を行うための類似度を定義し，類似度を測定するためのフレームワークを作成している．[8] の類似度測定手法では 2 ソースコード間のコードクローンから共起マトリクスを作成して画像に見立て，特徴量を抽出して類似度算出に用いている．

北川らは [9] においてメトリクス値によるベクトル空間に対して自己組織化マップを適用した類似度を提案している．自己組織化マップは生体のニューラルネットワークをモデル化したものである．[9] で提案されているこの類似度ではメトリクス値全体をファジーに比較することを可能にしているとしている．

黒木らは [10] でプログラムのソースコードを，プログラムの制御構造とプログラムで使用されている演算命令の観点から類似性を比較する類似度を定義している．[10] においてこの類似度は入力されたソースコードと類似しているコードを検索するために用いられている．

Ichii らは [11] において検索システムに協調フィルタリングを適用するため，検索システムを利用するユーザ間の振る舞いに対する類似度を測定している．協調フィルタリングとは，検索システムなどを利用する個々のユーザの振る舞いを用いて，ユーザの嗜好の推論を行う方法論である．

鷲崎らは [12] において，コンポーネント間の差異を測定する際に継承関係に基づく類似度を使用している．

また Baxter らは [13] において構文解析によって得られる抽象構文木の親子関係から類似度を算出している．この類似度はコードクローンを検出するために利用されている．

8.3 類似度の整理

以上で述べた類似度について整理する．調査した類似度を整理するために，次の二つの観点から類似度の分類を行った：

- 類似度の測定対象
- 類似度の測定目的

分類した結果を表 1 に示す．比較個数は一回あたりの類似判定において比較する対象が複数であるか単一であるかを表す．前処理は類似判定の手法を適用する前に比較する対象に対して行うべき処理である．利用要素は比較時に対象から抽出して利用する要素である．

表 1 から測定対象がテキストであってもソフトウェアであってもメタデータやメトリクス値から類似度を測定するための手法にはコサイン尺度がよく用いられていることが読み取れる．また類似度を使用する目的が探索，分類，評価など多岐にわたっているが，測定対象がテキストでもソフトウェアでも利用される手法は測定目的によって多様であることがわかる．

表 1: 既存の類似度分類結果

対応論文	比較対象	比較個数	目的	目的詳細	前処理	手法	利用要素
[4]	テキスト内容 x2	総当たり	探索		単語抽出	tf-idf	
[5]	テキスト：付随するメタデータ	単一：複数	探索			コサイン類似度	テキスト, メタデータ
[2]	コード全体：分類済みコード全体	単一：複数	分類	改変コピー部品分類		コサイン類似度	メトリクス値
[6]	コード全体 x2	総当たり	分類		単語抽出	LSA	
[7]	コード全体	単一：単一	予測			一致行数割合	
					単語抽出	LSA	
[8]	コード全体	単一：単一	剽窃検出		共起マトリクス化	内積類似度	特徴量
[9]	コード全体	複数*1	相関理解			ユークリッド距離	メトリクス値
[10]	コード全体 x2	単一：複数	探索	アルゴリズムの類似性考慮		データ重みづけ	HIR, データ依存情報
[3]	複数ファイルのソフトウェアシステム x2	単一：単一	分類	バージョン違い分類		一致行数割合	
[11]	プロジェクト x2	単一：複数	予測			コサイン類似度	メトリクス値
[12]	コンポーネント構造 x2	単一：複数	探索			有効置換距離＋α	

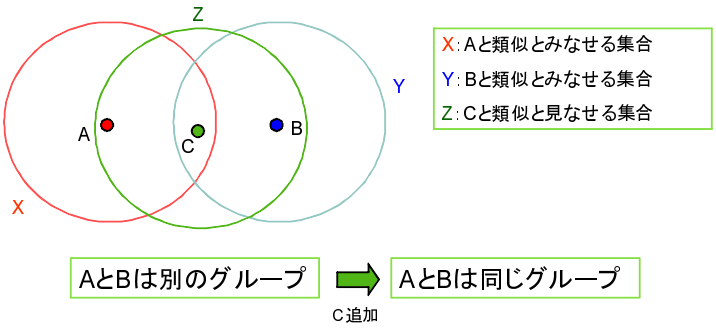

図 17: 類似範囲の拡大

9 今後の課題

Twigi の改良点に関する今後の課題を述べる．

まず今回行った実験は定性的なものであるため，ツールの性能が厳密に評価できているとは考えにくい．そこで今後はより定量的な実験を行い，ツールの性能を詳細に評価する必要がある．

また Twigi では前判定による分類処理の高速化を行っているが，前判定に使用する類似条件を追加することで分類処理に掛かる時間をさらに短縮できるはずである．

さらに本研究で提案した分類手法では，類似判定において類似の許容差異が大きくしすぎるとグループに含まれるコンポーネントの類似の範囲が広がってしまう可能性がある．例えば図 17 において，コンポーネント A とコンポーネント B のみが SPARS へ登録されている場合には A と B は別々のグループである．しかしこれらに加えてコンポーネント C が登録されると，類似でないはずの A と B は同じグループに分類されることになる．Luigi でもこの欠点は存在していたが，類似判定における類似の許容差異を狭くすることでこの欠点は今まで露呈していなかった．今後はこの欠点を解決するためのコンポーネント分類手法を考える必要がある．

最後に，類似判定に使用されている閾値は経験的に決められるため，閾値を決定する個々人の主観に依存してしまうという問題がある．この問題を解決するには閾値を個人の主観に依存せず自動的に決定するようなアルゴリズムを考える必要がある．

謝辞

本研究において，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本研究において，適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授に深く感謝いたします.

本研究において，逐次適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾 隆 助教に深く感謝いたします.

本研究において，逐次適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 早瀬 康裕 特任助教に深く感謝いたします.

本研究において，様々な御指導および御助言を頂きました立命館大学総合理工学院情報理工学部情報システム学科 山本 哲男 准教授に深く感謝いたします.

SPARS/R の開発者としてツールを実装する環境を提供してくださった大阪大学大学院情報科学研究科コンピュータサイエンス専攻 市井 誠 氏に深く感謝いたします.

過去，SPARS の研究および開発に携わった皆様に深く感謝いたします.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします.

参考文献

[1] 横森 励士, 梅森 文彰, 西 秀雄, 山本 哲男, 松下 誠, 楠本 真二, 井上 克郎.: “ Java ソフトウェア部品検索システム SPARS-J ”, 電子情報通信学会論文誌 D-I, VolJ87-D-I, No.12, pp1060-1068, 2004.

[2] 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 ”, 電子情報通信学会技術研究報告, Vol.103, No.102, SS2003-2, pp.7-12, 2003.5.

[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 ”, 電子情報通信学会論文誌 D-I, Vol.J85-D-I, No.6, pp.503-511, 2002.6.

[4] 増井俊之, 塚田浩二, 高林哲.: “ 近傍関係にもとづく情報検索システム ”, 第 11 回ソフトウェア科学会インタラクティブシステムとソフトウェアに関するワークショップ論文集, pp.79-86, 2003.12.

[5] 岸本貞弥, 村方衛, 中西崇文, 櫻井鉄也, 北川高嗣.: “ 数理分野を対象とした問題解決支援システム ”, 電子情報通信学会第 18 回データ工学ワークショップ, B8-7.

[6] Shinji Kawaguchi, Pankaj K. Garg, Makoto Matsushita, Katsuro Inoue.: “ MUDABlue: An automatic categorization system for Open Source repositories ”, The Journal of Systems and Software, Vol.79, Issue 7, pp.939-953, June 2006.

[7] 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.

[8] 大野麻子, 村尾元.: “ 参照ベクトルを用いたソースコード間の類似性検出 ”, 電気学会論文誌部門誌 C , Vol.128-C, No.1, pp.133-142, 2008.

[9] 北川 俊広, 杉山 安洋.: “ 自己組織化マップを用いた Java ソースコード間類似度測定ライブラリの試作 ”, 電子情報通信学会技術研究報告, Vol.104, No.723, pp.19-24, 2005.

[10] 黒木さやか, 上田高徳, 平手勇宇, 山名早人.: “ プログラムコードの抽象化を利用した類似ソースコード検索システム ”, 電子情報通信学会第 19 回データ工学ワークショップ, B10-2.

[11] Makoto Ichii, Reishi Yokomori, Katsuro Inoue.: “ Application of Collaborative Filtering for Software Component Retrieval System ”, International Workshop on Computer Supported Knowledge Collaboration

[12] 鷲崎弘宜, 深澤良彰.: “ 有向置換性距離に基づくコンポーネント検索システム ”, 第 4 回プログラミングおよび応用のシステムに関するワークショップ, pp.21-23, 2001.3.

[13] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier.: “ Clone Detection Using Abstract Syntax Trees ”, Proc. IEEE Int'l Conf. on Software Maintenance (ICSM) '98, pp. 368-377, Bethesda, Maryland, Nov. 1998.

[14] “ SourceForge.net ”, http://sourceforge.net/

[15] “ Sun Developer Network (SDN) ”, http://java.sun.com/