修士学位論文

題目

業務システム理解のための
外部アクセスに着目したクラスタリング手法

指導教員
井上 克郎 教授

報告者
秦野 智臣

平成 27 年 2 月 6 日

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 ソフトウェア工学講座

平成 26 年度 修士学位論文

業務システム理解のための外部アクセスに着目したクラスタリング手法

秦野 智臣

内容梗概

現代社会で稼働している業務システムの中には，数十年以上前から保守開発が続けられているものが多く存在している．このような古いシステムでは，大規模化と複雑化による保守性と生産性の低下が問題となっている．古いシステムの構造を見直し，保守性と生産性を向上させるシステムの再構築が行われているが，再構築のための最初の作業である対象システムの構造を理解することが困難であることが多い．その理由の 1 つとして，古いシステムではソースコード中の識別子にその役割を表す名前が付けられていないことが多いという問題がある．

本研究では，識別子を利用せずに，ソースコード中の関数を動作の類似性によって分類するクラスタリング手法を提案する．手法によって形成されたクラスタに含まれるある 1 つの関数を理解すれば，同じクラスタに属する関数を理解する際に，その知識が利用できると考えられ，その結果として，システム理解に必要なコストの削減が期待される．提案手法は，システムの操作画面やデータベースへのアクセスといった業務システムの基本的な命令を各関数から抽出し，関数をそれらの命令の列として表現する．そして，各関数を表す頂点と，命令列間の類似度を重みとした辺を持つグラフを作成する．最後に，作成したグラフに対して，密接な関連のある頂点を検出するクラスタリングアルゴリズムを適用し，動作が類似している関数を 1 つのクラスタにまとめる．手法を評価するために，2 つの Java システムに対して適用実験を行った．その結果，提案手法は，人手によるクラスタリングに近いクラスタリングを行うことができることが分かった．

主な用語

リエンジニアリング
ソフトウェアクラスタリング
プログラム解析

目 次

1　はじめに

現在，多くの組織が情報システムを使って業務を遂行している．このようなシステムの中には，数十年以上前から保守開発が続けられているものも多く存在しており，継続的な機能追加や仕様変更により，システムの大規模化と複雑化が進み，保守費用の増加を招いている [26]．また，現代ではユーザーの要求の変化に合わせて迅速にシステムを変更することが求められているが，大規模で複雑なシステムが対象の場合，それは困難である．

大規模システムが抱えるこれらの問題を解決するために，システムの再構築が行われている．再構築では，まず，現行システムの構造を分析し，その改善を図るためにシステムの再設計を行う．そして，再設計に基づいてソースコードの変更を行い，それらのテストを行う．最終的に，保守性と生産性の高い新システムとして運用される．このようなシステムの再構築を行うための最初の作業として，現行システムの構造を理解することが必要である．

しかし，古いシステムでは次のような理由によりシステムの構造を理解することが困難であることが多い．1 つ目に，システムの設計書が残されていない場合や，最新の状態が記述されていない場合が多いという問題がある [7]．この場合は，対象システムの保守開発を行っている開発者に対してヒアリング等を行い，システムを理解することになる．しかし，その開発者でも最近の変更点しか理解しておらず，システム全体の構造や動作までは把握していないことがある．2 つ目に，現代のシステム開発における主要言語である Java ではパッケージ構造を用いてソースコードを階層的に整理することが行われているが，当時はそういったことが行われていなかったため，膨大な数のファイルが 1 つのディレクトリに存在しており，システム全体の構造が分からなくなっているという問題がある．3 つ目に，現在では，ソースコード中の識別子にその役割を表す名前を付けることが主張されているが [15]，当時は，設計時に機械的に割り当てた文字列や無関係な名前を使用する [20] ことが多かったため，識別子からその役割や機能を推測することが難しいという問題がある．

既存研究では，システムの構造を分析する作業を支援する手法として，ソフトウェアクラスタリングが提案されている．ソフトウェアクラスタリングは，システムのソースコードの構成要素（ファイルや関数など）を，ある観点に基づく類似性によって分類する技術である．これまでに提案されたクラスタリング手法の多くは，ファイル間の依存関係や識別子の類似度に基づいて，システムを凝集度が高く結合度の低いサブシステムに分類することを目的としている [14]．この分類によってシステムの複雑度を下げることで，大規模システムの理解を支援することができる [12]．また，Kobayashi らは，クラスタリングを用いてシステムの構造を可視化し，構造分析の支援を行っている [11]．

本研究では，ソースコード中の識別子を利用せずに，動作が類似している関数の集合に分類するクラスタリング手法を提案する．大規模システムでは，扱うデータは異なるが，類似

した動作をする処理が多く存在する．そのため，動作が類似した集合に分類することによって，集合に含まれるある 1 つの関数を理解すれば，同じ集合に属する関数を理解する際に，その知識が利用できると考えられる．結果として，システム理解に必要なコストの削減が期待される．理解を行うためにはソースコードの読解が必要であることを考慮すると，既存研究で行われているようなサブシステムへの分類だけでなく，動作の類似性に基づく分類も重要であると考えられる．

提案手法は，プログラムの各関数を，システムの操作画面やデータベースへのアクセスといった業務システムに共通する命令の列で表現する．そして，プログラムの各関数を表す頂点と，対応する列間の類似度を重みとした辺を持つグラフを作成する．最後に，作成したグラフに対して，密接な関連のある頂点を検出するクラスタリングアルゴリズムを適用することで，類似度の高い関数がクラスタとしてまとめられる．

手法を評価するために，Java で書かれたシステムに対する実装を行い，適用実験を行った．実験では，2 つのシステムに対して手法を適用し，クラスタリングの良さを評価するための指標を計測した．また，提案手法によるクラスタリングが，既存研究で行われている凝集度と結合度に基づくクラスタリングとは異なることを確認するために，関数の呼び出し関係によるクラスタリングとの比較を行った．

以降，2 章では，既存のクラスタリング手法とそれに関連する技術について述べる．3 章では提案手法を述べ，4 章では手法を適用した評価実験について述べる．5 章では，手法の実用化に向けた議論を行い，最後に，6 章でまとめと今後の課題を述べる．

2　背景

本章では，本研究の背景として，既存のソフトウェアクラスタリング手法とその評価方法について述べる．また，クラスタリング手法の関連技術としてソースコードの類似性を認識する手法について述べる．

2.1　ソフトウェアクラスタリング

ソフトウェアクラスタリングとは，システムのソースコードの構成要素（ファイルや関数など）を，ある観点に基づく類似性によって分類する技術である．この技術は，まず，ソースコードの構成要素からある特徴を抽出し，要素間の関連を数値化する．そして，機械学習において利用されているクラスタリングアルゴリズムを適用し，関連の強い要素を 1 つのクラスタとしてまとめる．クラスタリングの結果は，システムの構造を復元したり，構造の改善点を発見したりするために使われている [11, 14]．それぞれの目的によって，特徴の抽出方法は様々である．

Mancoridis らは，関数の呼び出し関係や変数の参照関係をファイル間の依存関係として抽出し，対象システムを凝集度が高く結合度の低いサブシステムに分類することを目的としている [13]．サブシステムに分類することによってシステムの複雑度を下げ，システムの構造理解を支援している．しかし，依存関係によるクラスタリングは，様々な関数から呼び出される関数が存在する場合，クラスタリング結果を開発者が手動で洗練する必要があるという問題があった [17]．そこで，そのような関数を除去する手法や，その影響を軽減する手法が提案されている [10]．Kobayashi らは，[10] の手法を用いてシステムの構造を可視化し，構造の理解を支援する手法を提案している [11]．

ソースコード中の識別子の類似性に基づくクラスタリング手法も提案されている．Anquetil らは，識別子に含まれる単語の類似度を用いたクラスタリングが有用であることを示しており [3]，Andritsos らは，単語の出現頻度による重みを考慮することが重要であることを示している [1]．また，Anquetil らは，ファイル名の命名規則を用いたクラスタリング手法を提案している [2]．これらの手法は識別子にその役割を表す名前が付けられている場合，有用であると考えられる．

Scanniello らは，オブジェクト指向言語で書かれたシステムから，クラスの継承関係とインターフェースの実装関係を用いたクラスタリング手法を提案している [19]．この手法は，クライアントサーバシステムでよく用いられる層構造を認識するために，オブジェクト指向言語の特徴を利用しており，認識された層ごとに識別子の類似性によるクラスタリングを適用している．

Tzerpos らは，大規模システムでよく見られるパターンに基づくクラスタリング手法を提

案している [23]．この手法におけるパターンとは，たとえば，C 言語で書かれたシステムにおいて fileA.c と fileA.h を同じクラスタにまとめる，デバイスドライバを同じクラスタにまとめる，様々な関数から呼び出される関数を同じクラスタにまとめるなどといったクラスタリングの方針を指す．これらのパターンを対象システムの特徴に合わせて開発者に定義してもらうことで，開発者にとって理解しやすいクラスタリングを行うことを目的としている．

2.2 クラスタリング手法の評価方法

クラスタリング手法の評価は，あるシステムに対して手法を適用し，以下の 3 つの指標を計測することによって行われている [25]．

信頼性 手法によるクラスタリングが人手によるクラスタリングにどのくらい近いか

クラスタの分布 クラスタの大きさが極端でないか

安定性 システムのソースコードが少し変更されても，クラスタリング結果が大きく変わらないか

2.2.1 信頼性

信頼性を計測するためには，手法の目的に合ったクラスタリングを人手で行う必要がある．人手によるクラスタリングは，対象システムの開発者に対するヒアリングやソースコードを読解することなどによって行われる．ただし，この作業はコストを要するため，ディレクトリ構造が適切に管理されているシステムを適用対象とし，各ディレクトリを 1 つのクラスタに対応させたものを人手によるクラスタリングとして用いる場合も多い [10, 19]．

2 つのクラスタリング間の距離を計測する指標として，MoJoFM [24] が使われている．MoJoFM は，あるクラスタリング A から別のクラスタリング B に変換するときに，以下の 2 つの操作が何回必要であるかを計測する指標である．

Move あるクラスタの要素を別のクラスタに移動させる．移動させる要素を新たに 1 つのクラスタとすることも含む．

Join 2 つのクラスタを併合し，1 つのクラスタにする．

MoJoFM は，上記の操作回数が少ないほど A は B に近いとする指標であり，以下のように定義される．

$$\mathrm{MoJoFM}(A,B) = (1 - \frac{mno(A,B)}{max(mno(\forall X,B))}) \times 100\% \quad (1)$$

ここで，$mno(A,B)$ は，A から B に変換するのに必要な最小の操作回数である．
$max(mno(\forall X,B))$ は，B に変換するのに必要な最小の操作回数が最大であるようなクラス

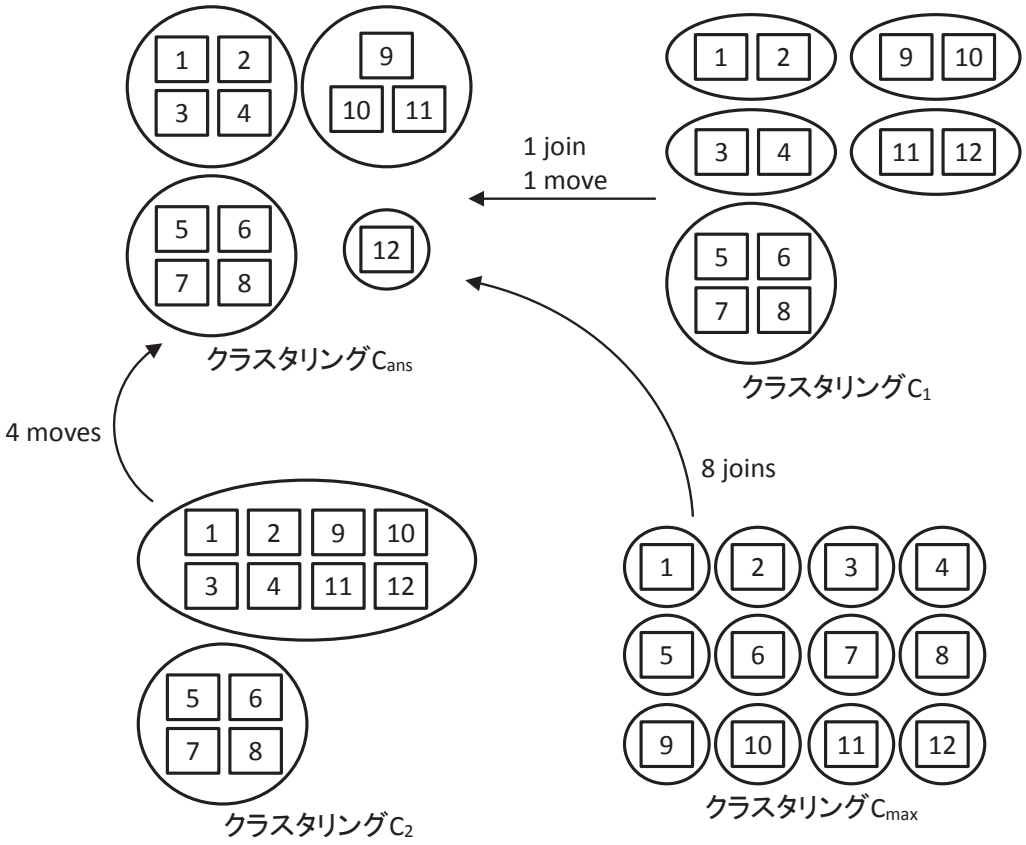

図 1: MoJoFM における操作回数の数え方

タリングの操作回数である．MoJoFM の値が高ければ高いほど，A は B に近いことを意味し，MoJoFM が 100% であれば，A は B に等しいことを意味する．MoJoFM は距離を計測する指標として提案されているが，MoJoFM(A, B) と MoJoFM(B, A) は一般に等しくないため，A を手法によるクラスタリングとし，B を人手によるクラスタリングとして評価を行う．

図 1 を用いて MoJoFM の計算例を示す．図 1 は，12 個の要素を分類する 4 つのクラスタリング（$C_{ans}, C_1, C_2, C_{max}$）について，$C_1$ から C_{ans}，C_2 から C_{ans}，C_{max} から C_{ans} にそれぞれ変換するのに必要な最小の操作を示している．たとえば，C_1 から C_{ans} へは，要素 1，2 を含むクラスタと要素 3，4 を含むクラスタを join し，要素 9，10 を含むクラスタに要素 11 を move することによって変換できる．C_{max} は操作回数が最大であるクラスタリングの 1 つであり，8 回の join 操作を要する．したがって，$\mathrm{MoJoFM}(C_{max}, C_{ans}) = 0\%$ であり，$\mathrm{MoJoFM}(C_1, C_{ans}) = 1 - 2/8 = 75\%$，$\mathrm{MoJoFM}(C_2, C_{ans}) = 1 - 4/8 = 50\%$ となる．以上より，MoJoFM では C_{ans} にもっとも近いクラスタリングは C_1 となる．

2.2.2 クラスタの分布

クラスタの大きさが極端であるかどうかは，Non-Extremity of cluster Distribution（NED）によって計測される．NED は，クラスタの要素数が決められた範囲に収まっているかを割合で表現したものであり，以下のように定義される．

$$\mathrm{NED}(C) = \frac{1}{N} \sum_{c \in C; 5 \leq |c| \leq \max(20, N/5)} |c| \tag{2}$$

ここで，c はクラスタリング C におけるクラスタであり，$|c|$ は c の要素数である．また，N はクラスタリング C におけるすべての要素数である．NED が 1 に近ければ，クラスタの大きさが極端でないことを意味する．

2.2.3 安定性

安定性は，同じシステムの連続した 2 つのバージョンに対してクラスタリング手法を適用し，得られた 2 つのクラスタリング間の距離を計測する．安定性における距離は，クラスリング A, B 間の両方向の距離が等しい MoJoSim [22] が使われる．MoJoSim は MoJoFM と同様に move と join の回数を数えるが，以下の定義のように A から B と B から A のうち，操作回数が少ない方向を採用する．

$$\mathrm{MoJoSim}(A, B) = (1 - \frac{min(mno(A, B), mno(B, A))}{N}) \times 100\% \tag{3}$$

連続した 2 バージョンに対する MoJoSim が高ければ，そのクラスタリング手法は安定性が高いことを意味する．ただし，2 バージョン間でクラスタリングの要素が異なる場合は，両バージョンに共通している要素に対して MoJoSim を計算する．複数の連続したバージョン間に対して MoJoSim を計算し，それらの平均値を求めることで安定性の評価を行う．

2.3 ソースコードの類似性を認識する技術

2.3.1 コードクローン検出

コードクローンとは，ソースコード中に存在する同一または類似した部分を持つコード片のことである．これまでの研究において，コードクローンを検出する様々な手法が提案されており，識別子の情報を利用せずに，大規模システムにおける類似したコード片や関数を検出する手法も提案されている [9]．[9] では，ソースコードをトークンの列に変換し，ある一定以上の長さで一致しているトークン列をコードクローンとして検出している．

しかし，これらのコードクローン検出手法は，本研究の目的に対しては適切ではない．コードクローン検出では，サブルーチンの呼び出しを考慮した類似処理の検出は行われていない

が，本研究ではそれを考慮する必要がある．また，コードクローン検出における主な目的は，類似処理の集約やクローンとなっている複数のコード片に対する一貫した修正を行うという作業を支援し，プログラムの保守性を向上させることである．そのため，コードクローン検出ではある程度長いコード片の検出を行うが，本研究では，短い関数であっても動作が類似していれば同じクラスタとして認識する必要がある．さらに，コードクローン検出では，データベースを操作する前後でのエラー処理といった動作の種類に関係なく行われる処理をクローンとして検出してしまい，動作が異なる関数を類似している関数であると判定してしまう可能性がある．これらの処理は，関数の動作の特徴を表すものではないため，本研究ではこれらの処理の類似性を無視する必要がある．

2.3.2 類似したアプリケーションの検出

McMillan らは，ソースコードの再利用を効率的に行うために，類似した Java アプリケーションを検出する手法を提案している [16]．この手法では，同じパッケージや同じクラスに属するメソッドを呼び出しているアプリケーションを類似したアプリケーションとして検出している．この検出方法は，同じ API を利用するものは似ているアプリケーションであるという考え方に基づいている．また，この手法は，パッケージやクラスといった概念で整理されたアプリケーションがオープンソースとして数多く公開されていることを利用している．

2.3.3 ソースコードの自動要約

プログラム理解の支援を行うために，ソースコードの自動要約を行う技術が提案されている．Sridhara らは，Java プログラム中のメソッドの動作を実装しているコード片を自動的に認識し，その動作を自然言語で表現する手法を提案している [21]．この手法では，メソッドにおける代入文，条件文，ループ文といった文の並びを解析することで，各メソッドの動作を認識し，類似した動作の検出を行っている．この手法は，ソースコード中の識別子を利用するため，識別子が適切に命名されているプログラムにおいて有用であると考えられる．

3　提案手法

本研究では，システムのソースコードを与えると，動作が類似している関数を 1 つのクラスタにまとめるクラスタリング手法を提案する．動作が類似しているとは，関数が扱うデータの内容によらず，データの検索，更新，削除といった処理の内容が類似していることを言う．本手法は，ソースコード中の識別子を利用しないため，識別子に意味のない名前が使われているシステムに対して適用することができる．また，本論文では，実験対象としたシステムで用いられている Java を例として提案手法について説明するが，本手法は一般的な手続き型言語に存在する概念のみを利用するため，業務用のプログラミング言語として広く用いられている COBOL などの他言語への適用も可能である．

本手法は，システムの操作画面に対する入出力処理と SQL 文を実行する処理（これらをまとめて外部アクセスと呼ぶ）によってメソッドを特徴づける．外部アクセスは，システムが様々な機能を提供するための基本的な処理であるため，メソッドの動作内容をよく表していると考えられる．また，システムの機能を実装するためには，外部アクセスの実行を条件分岐やループ文などによって制御する必要があるため，これらの制御文もメソッドの特徴として抽出する．

対象システム中のどのメソッドが外部アクセスに対応するかは，開発者に定義してもらう．外部アクセスを行う方法は限定されているため，対応関係を定義してもらうことは現実的であると考えられる．たとえば，Java では JDBC の API を呼び出すことによってデータベースを操作する方法が一般的であり，その他のライブラリを利用する場合も，データベースを操作するためには決められた API を呼び出す必要がある．

本手法は，以下の 3 つのステップからなる．

Step 1. 外部アクセスと制御文の抽出 各メソッドから，外部アクセスに対応するメソッド呼び出し文と，それらの文を本体に含む制御文を抽出する．抽出された文は，それぞれの種類に対応した記号に変換され，各メソッドは，それらの記号の列として表現される．

Step 2. 記号列間の類似度の計算 抽出した各記号列を N-gram による集合で表し，すべての集合間のジャッカード係数を計算する．この値をメソッド間の類似度とする．

Step 3. クラスタリングアルゴリズムの適用 各メソッドを表す頂点と，Step 2 で計算した類似度を重みとした辺を持つグラフを作成する．このグラフに対してクラスタリングアルゴリズムを適用し，クラスタリング結果を得る．

以降では，各ステップの詳細について説明する．

表 1: 抽出する文の一覧

カテゴリ	文	記号
SQL 文の実行	SELECT 文を実行するメソッドの呼び出し	Ss，Sm
	INSERT 文を実行するメソッドの呼び出し	Is，Im
	UPDATE 文を実行するメソッドの呼び出し	Us，Um
	DELETE 文を実行するメソッドの呼び出し	Ds，Dm
画面との入出力	画面から値を読み込むメソッドの呼び出し	Rs，Rm
	画面に値を書き込むメソッドの呼び出し	Ws，Wm
制御文	if 文の開始	If
	if 文の終了	If}
	else 節の開始	E
	for, while, do 文の開始	L
	for, while, do 文の終了	L}

3.1 Step 1. 外部アクセスと制御文の抽出

プログラムの各メソッドから，外部アクセスに対応するメソッド呼び出し文と，それらの文を本体に含む制御文を抽出する．抽出された文は，それぞれの種類に対応した記号で表現され，メソッドは，それらの記号の列で表現される．本手法で抽出する文と本論文における記号表現を表 1 に示す．どのメソッド呼び出しが外部アクセスに対応するかは，開発者が定義する．また，外部アクセスに対応するメソッド呼び出し文は，その文が単一のレコードを扱うか，複数のレコードを扱うかを区別する．本論文ではそれぞれの記号に，単一レコードの場合は s を，複数レコードの場合は m を添えることによって表現する．単一レコードと複数レコードのどちらを扱うメソッドであるかは，メソッドのシグネチャによって判断する．SELECT 文を実行するメソッドと画面から値を読み込むメソッドの呼び出しについては，そのメソッドの返り値が配列かリスト（java/util/List）である場合は複数レコードとし，そうでない場合は単一レコードとする．これら以外のメソッド呼び出しについては，そのメソッドの引数に配列かリストが含まれる場合は複数レコードとし，そうでない場合は単一レコードとする．

記号列抽出のアルゴリズムを Algorithm1 に示す．このアルゴリズムは，抽出対象のメソッド m と外部アクセスに対応するメソッドの集合 S, I, U, D, R, W （それぞれ表 1 の記号表現に対応する）を入力として受け取り，記号列 SS を出力する．SS は，m の抽象構文木を構築し，その深さ優先探索を行うことによって生成される．抽象構文木とは，プログラムの構文解析を行い，その結果を木構造で表現したものである．本手法の実装は，Cesare ら

の手法 [5] を参考にしており，Java development tools [1] の抽象構文木を利用している．

本アルゴリズムにおける探索では，メソッド呼び出し文と制御文についてそれぞれの処理を行う．メソッド呼び出し文に対応する頂点を初めて訪問した場合は，その呼び出し先メソッドが外部アクセスであれば対応する記号を SS に追加する．呼び出し先メソッドが外部アクセスでなければ，そのメソッドから再帰的に記号列を抽出し，それを SS に追加する．制御文に対応する頂点を初めて訪問した場合は，それぞれの開始記号を SS に追加し，その頂点以下の子頂点をすべて探索したら終了記号を SS に追加する．m の探索がすべて終了したら，SS における条件文の開始記号から対応する終了記号までの区間を走査し，その区間に外部アクセスに対応する記号が存在しない場合は，その条件文の記号を SS から削除する．

図 2，3 のコードを用いて，抽出例を示す．図 2 のコードは，あるシステムの顧客情報管理画面において，ユーザーが選択したすべての顧客情報を顧客データベースから一括で削除する機能を実装しているメソッドである．このメソッドは，選択された各顧客情報に対して，それを削除して問題ないかを 7，8 行目で確認している．7，8 行目では，注文テーブルの検索を行い，削除しようとしている顧客が何も注文していないことを確認している．注文がなければその顧客情報を削除し，そうでなければスキップする．同様に，図 3 のコードは，商品情報管理画面において，ユーザーが選択したすべての商品情報を商品データベースから一括で削除する処理を実装しているメソッドである．この処理は，図 2 のコードと削除確認の内容が異なるが，ほぼ同じ動作をしている．

図 2，3 のメソッドに対して，表 2 のようにメソッドと外部アクセスの対応付けを行う．そして，各メソッドの抽象構文木を構築し，その探索により記号列を抽出する．図 2 のメソッドを抽象構文木で表現すると図 4 のようになる．実際の抽象構文木の頂点は式や文などといった言語仕様によって決められているプログラムの構成要素であるが，図 4 では簡略化のため 1 行を頂点とし，行番号を頂点の数字で表している．この抽象構文木の深さ優先探索による訪問手順を赤色の破線で表している．この手順に従って記号列の抽出を行うと，表 3 のような記号列が抽出される．同様に，図 3 のメソッドについても，抽象構文木の探索によって記号列が抽出される．

3.2 Step 2. 記号列間の類似度の計算

Step 1 で抽出したすべての記号列間の類似度を計算する．記号列間の類似度は，各記号列を N-gram から構成される集合で表現し，集合間のジャッカード係数を計算することで求める．N-gram とジャッカード係数は，文字列間の類似度を計算する手法として広く使われており，同じく文字列間の類似度計算として使われている最長共通部分列と比較して計算コス

[1] http://eclipse.org/jdt/

Algorithm 1: メソッドから記号列を抽出する

Input : m : method; S, I, U, D, R, W : set of methods

Output: SS : list of symbol

```
1  M ← S ∪ I ∪ U ∪ D ∪ R ∪ W                          /* M : set of methods */
2  syntaxTree ← createSyntaxTree(m)                    /* syntaxTree : Tree */
3  SS ← [ ]                                            /* SS : list of symbol */
4  forall the v ∈ visitInDepthFirstOrder(syntaxTree) do
5      if isFirstVisit(v) then
6          if v is method call then
7              if v ∈ M then
8                  SS ← SS + getSymbol(v)
9              else
10                 SS ← SS + extractSymbolSequence(v, S, I, U, D, R, W)
11         else if v is IF statement then
12             SS ← SS + "If"
13         else if v is ELSE statement then
14             SS ← SS + "E"
15         else if v is LOOP statement then
16             SS ← SS + "L"
17     else if endVisit(v) then
18         if v is IF statement then
19             SS ← SS + "If}"
20         else if v is LOOP statement then
21             SS ← SS + "L}"
22 return removeEmptyControlStatements(SS)
```

```
1  void batchDeleteCustomer(CustomerDao cstDao,
2                           OrderDao ordDao, Vo vo) {
3     // 選択された顧客のIDを取得する
4    String[] idArray = vo.getSelectedCustomers();
5    for (String id: idArray) {
6        // 削除対象の顧客が注文中でないか確認する
7        List<OrderDto> ordList = ordDao.select(id);
8        if (ordList.isEmpty()) {
9            cstDao.delete(id); // 顧客情報をテーブルから削除する
10       }
11   }
12 }
```

図 2: 選択された顧客の情報を一括で削除するメソッド

```
1  void batchDeleteProduct(ProductDao prdDao,
2                          OrderDao ordDao, Vo vo) {
3    // 選択された商品のIDを取得する
4    String[] idArray = vo.getSelectedProducts();
5    for (String id: idArray) {
6        // 削除対象の商品が注文されていないか確認する
7        List<OrderDto> ordList = ordDao.select(id);
8        // 商品を保持しなければならない期間を過ぎているか確認する
9        ProductDto dto = prdDao.select(id);
10       Date limitDate = dto.getLimitDate();
11       Date currentDate = new Date();
12       if (ordList.isEmpty() && currentDate.after(limitDate)) {
13           prdDao.delete(id); // 商品情報をテーブルから削除する
14       }
15   }
16 }
```

図 3: 選択された商品の情報を一括で削除するメソッド

表 2: 図 2，3 のコードに対する外部アクセスの対応付け

外部アクセス	対応するメソッド
SELECT 文の実行	OrderDao クラスの select メソッド
	ProductDao クラスの select メソッド
DELETE 文の実行	CustormerDao クラスの delete メソッド
	ProductDao クラスの delete メソッド
画面からの入力	Vo クラスの getSelectedCustomers メソッド
	Vo クラスの getSelectedProducts メソッド

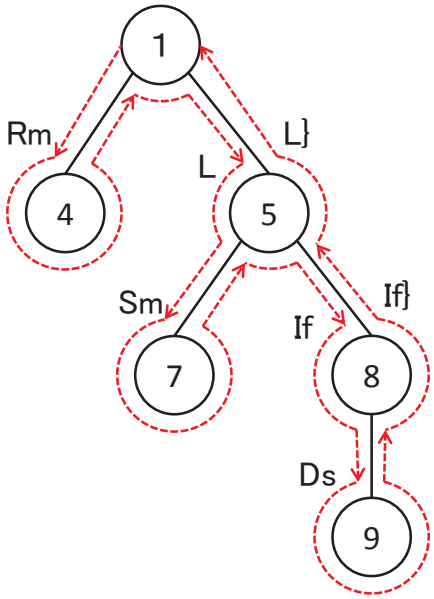

図 4: 図 2 のコードの抽象構文木の訪問順序

表 3: 図 2，3 のメソッドから抽出される記号列

batchDeleteCustomer メソッド	Rm, L, Sm, If, Ds, If}, L}
batchDeleteProduct メソッド	Rm, L, Sm, Ss, If, Ds, If}, L}

トが小さい．そのため，N-gram とジャッカード係数による類似度計算は，大規模システムの解析により適していると言える．

N-gram は，文字列を長さ N の文字列に分解して記述する手法である．N はパラメータであり，N = 3 とする tri-gram がよく用いられる．例として，文字列 ”ABCDE” を tri-gram による集合で表現すると以下のようになる．

$$\{\$\$A, \$AB, ABC, BCD, CDE, DE\$, E\$\$\} \tag{4}$$

’$’ は，文字列の先頭と末尾を表す特殊文字である．このように，N-gram を用いることで文字列の順序を考慮しつつ，それを集合で表現することができる．

集合間の類似度は，ジャッカード係数により計算する．ジャッカード係数は，2 つの集合 X, Y に共通する要素の割合を表すものであり，以下のように定義される．

$$\mathrm{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{5}$$

定義より，X と Y が集合として等しい場合，ジャッカード係数は 1 となり，X と Y に共通する要素がない場合は 0 となる．この定義にしたがって，各記号列を N-gram で表した集合間のジャッカード係数を計算し，その値をメソッド間の類似度とする．

Step 1 の抽出例を用いて，記号列間の類似度計算の例を示す．表 3 の tri-gram による集合をそれぞれ M_1，M_2 とすると，以下のようになる．

$$
\begin{aligned}
M_1 =& \{\$\_\$\_Rm, \$\_Rm\_L, Rm\_L\_Sm, L\_Sm\_If, Sm\_If\_Ds, \\
& \quad If\_Ds\_If\}, Ds\_If\}\_L\}, If\}\_L\}\_\$, L\}\_\$\_\$\} \\
M_2 =& \{\$\_\$\_Rm, \$\_Rm\_L, Rm\_L\_Sm, L\_Sm\_Ss, Sm\_Ss\_If, Ss\_If\_Ds, \\
& \quad If\_Ds\_If\}, Ds\_If\}\_L\}, If\}\_L\}\_\$, L\}\_\$\_\$\} \\
M_1 \cap M_2 =& \{\$\_\$\_Rm, \$\_Rm\_L, Rm\_L\_Sm, If\_Ds\_If\}, Ds\_If\}\_L\}, If\}\_L\}\_\$, L\}\_\$\_\$\} \\
M_1 \cup M_2 =& \{\$\_\$\_Rm, \$\_Rm\_L, Rm\_L\_Sm, L\_Sm\_If, Sm\_If\_Ds, L\_Sm\_Ss, \\
& \quad Sm\_Ss\_If, Ss\_If\_Ds, If\_Ds\_If\}, Ds\_If\}\_L\}, If\}\_L\}\_\$, L\}\_\$\_\$\}
\end{aligned}
$$

したがって，Jaccard(M_1, M_2) = 7 / 12 = 0.583 となり，これが 2 つメソッド間の類似度となる．

3.3 Step 3. クラスタリングアルゴリズムの適用

Step 2 で計算した類似度を用いて，メソッドのクラスタリングを行う．これまでに様々なクラスタリングアルゴリズムが提案されてきたが，本研究では Newman らが提案したアルゴリズム [18] を用いる．このアルゴリズムは，ソーシャルネットワークにおけるコミュニティを検出する目的で提案されたが，ソフトウェアクラスタリングにおいても有用であることが示されている [8, 10]．また，アルゴリズムの計算複雑度は，クラスタリングの対象となる要素の数を |V| としたとき，$O(|V|\log^2|V|)$ であり [6]，大規模システムのクラスタリングに適している．

Newman らのアルゴリズムは，クラスタリングの対象となる要素をグラフの頂点で表し，要素間の関連を対応する頂点間の辺で表す．辺には，関連の強さを表す重みを付けることができる．このグラフにおいて，以下に定義される Q 値が最大となるようなクラスタリングを求める．

$$
Q = \frac{1}{W}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{W}\right]\delta(c_i, c_j) \tag{6}
$$

ここで，A_{ij} は，頂点 i と頂点 j を接続している辺の重みを表す．W は，すべての頂点間の辺の重みの和であり，$W = \sum_{i,j} A_{ij}$ で表される．k_i は，頂点 i に接続されているすべての辺の重みの和であり，$k_i = \sum_j A_{ij}$ で表される．c_x は，頂点 x が属するクラスタであり，$\delta(c_i, c_j)$ は，$c_i = c_j$ のとき 1 をとり，そうでないときは 0 をとる関数である．Q 値は，クラスタ内の辺の重みの和が大きいほど高くなるように定義されている．したがって，Q 値が最大となるようなクラスタリングを求めることで，クラスタ内の関連が強い分類を求めること

ができる．しかし，Q 値の最大化は NP 困難であるため [4]，Newman らのアルゴリズムでは以下のような貪欲法で Q 値最大化の近似計算を行う．

1. すべての頂点がそれぞれ 1 つのクラスタであるとする．

2. Q 値が最も高くなるように 2 つのクラスタを併合する．

3. すべてのクラスタが 1 つに併合されるまで 2 を繰り返す．

4. 併合過程の中から最も Q 値が高い状態を選び，それをクラスタリングの結果とする．

提案手法では，プログラムの各メソッドをグラフの頂点とし，Step 2 で計算した類似度を頂点間の辺の重みとして Newman らのアルゴリズムを適用する．ただし，類似度の低いメソッドが同じクラスタに分類されることを防ぐため，類似度がある閾値未満であるメソッドについては，それらの間の辺の重みを 0 とする．したがって，2 つのメソッド m_i，m_j に対応する頂点間の辺の重み k_{ij} は，それぞれの N-gram による集合を M_i，M_j とすると，以下の式で表される．

$$k_{ij} = \begin{cases} \mathrm{Jaccard}(M_i, M_j) & \text{if } (\mathrm{Jaccard}(M_i, M_j) \geq threshold) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

閾値 $threshold$ は手法のパラメータとし，類似度の分布によって変化させるものとする．

クラスタリングアルゴリズムの動作例を示す．4 つのメソッド m_1, m_2, m_3, m_4 について，Step 1，2 より図 5 のような類似度が得られたとする．図 5 のグラフに対して類似度の閾値を設けずに Newman らのアルゴリズムを適用すると，図 6 の手順でクラスタリングが行われる．図 6 は，各メソッドがそれぞれ 1 つのクラスタである状態から始まり，2 つのクラスタが順次併合されていく過程を示している．初期状態では，クラスタ $\{m_1\}, \{m_2\}, \{m_3\}, \{m_4\}$ のすべての組から，それらを併合した場合に Q 値が最も高くなるような組を選び，その組を併合する．その結果，$\{m_1\}$ と $\{m_2\}$ が併合され，次の段階では，$\{m_1, m_2\}, \{m_3\}, \{m_4\}$ の組に対して併合を行う．その結果，$\{m_3\}$ と $\{m_4\}$ が併合され，最後の段階では，$\{m_1, m_2\}$ と $\{m_3, m_4\}$ が併合され，すべてのメソッドが 1 つのクラスタに含まれる．そして，この過程のうち最も Q 値が高いステップ 3 の状態がクラスタリング結果として選ばれる．以上のような併合の過程は，図 7 のような樹形図で表現することができる．図 7 は，クラスタの併合を線の合流で表し，それが何回目の併合であるかを縦軸で表している．赤色の水平線は，その線で樹形図を切断すると，Q 値が最も高い状態のクラスタリングに対応することを表している．

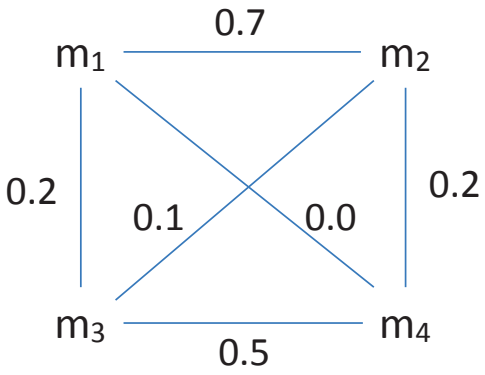

図 5: メソッド間の類似度を表すグラフの例

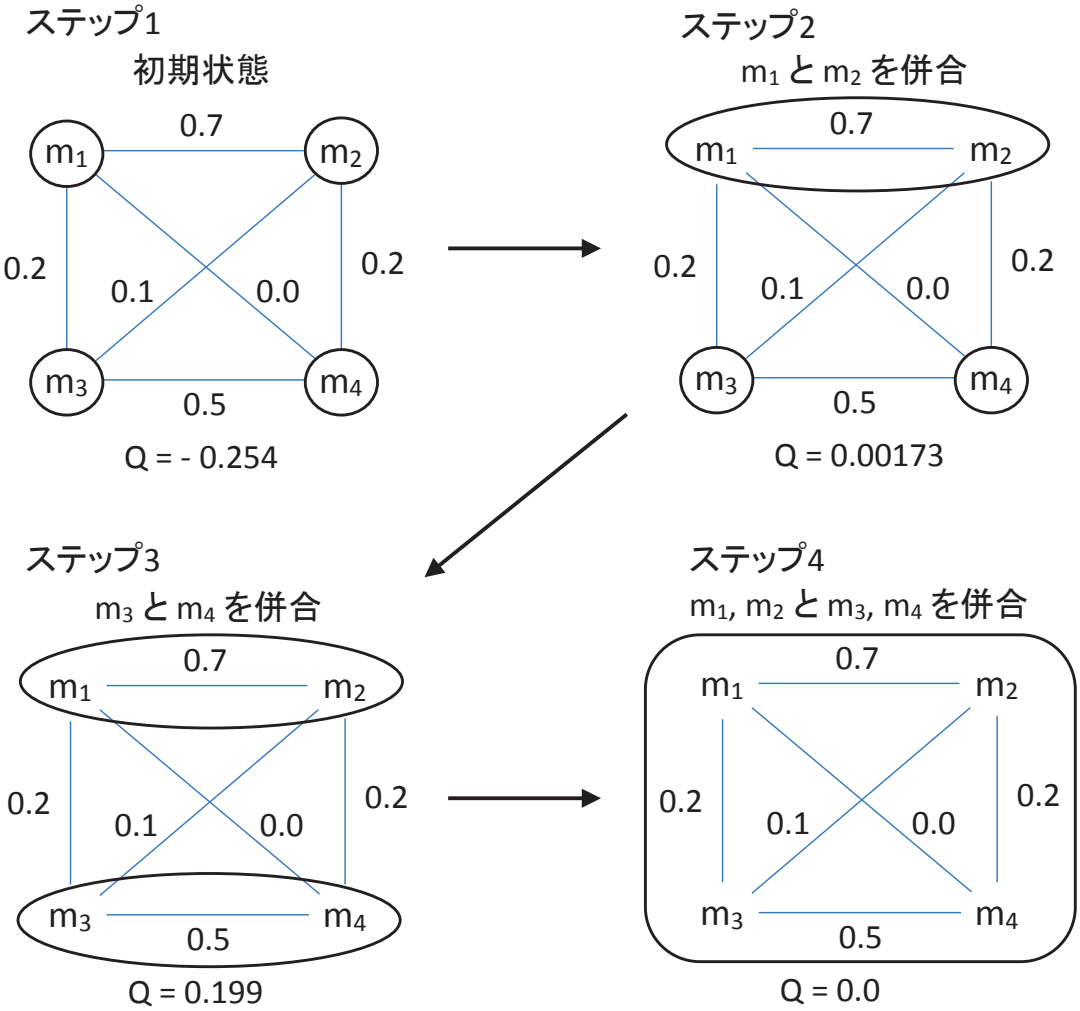

図 6: クラスタリングの計算手順

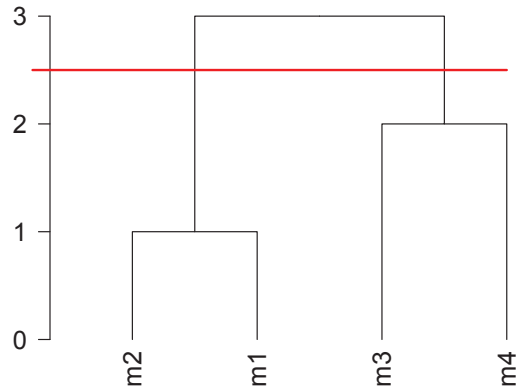

図 7: クラスタリングの過程を表す樹形図

4　評価実験

提案手法を実装し，2 つの Java システムに対して適用した．本実験では，2.2 節で述べた評価指標を計測し，手法の評価を行う．また，手法のパラメータを変化させた場合に，クラスタリング結果がどのように変化するかを調査するために，手法の Step 2 における N-gram の N を 1，3，5 と変化させ，Step 3 における類似度の閾値を 0.0 から 1.0 まで 0.1 刻みで変化させて各指標の計測を行う．N を 1 とすることは，記号列の順序を考慮せずに単純な集合として比較することを意味する．また，閾値を 1.0 とすることは，N-gram による集合が完全に一致しているメソッドのみを同じクラスタにまとめることを意味する．一般に，N を小さくすると類似度は高くなりやすく，N を大きくすると類似度は低くなりやすいため，N が小さい場合は閾値を高く設定し，N が大きい場合は閾値を低く設定するべきであると予想される．

4.1　準備

本実験では，勤怠管理システムである MosP 勤怠管理 V4 [2] と，ある企業におけるトライアル用の販売管理システムを用いて手法の評価を行う．それぞれのシステムの規模を表 4 に示す．表 4 における対象メソッド数とは，ユーザーからのボタン操作などのアクションによって呼び出されるメソッドのうち，データベースへのアクセスを行うものの数であり，これらのメソッドをクラスタリングの対象とする．テーブル数とは，対象メソッドからアクセスされるテーブルの数である．MosP においては，クラス名が Action で終わるクラスの action メソッドから呼び出されるメソッドを対象メソッドとした．テーブル数は，sql ファイルにおいて”CREATE TABLE” によって定義されているテーブルの名前を取得し，フィールド変数 TABLE にそれらの名前が定義されているクラス数によって計測した．表中の”-” は，MosP の 11 バージョン（4.0.0-4.4.2）における最小値と最大値を示しており，これらのバージョンを用いて安定性の評価を行う．信頼性とクラスタの分布は，最もメソッド数が多い最新バージョンを用いてそれぞれ評価を行う．また，本実験では，業務を遂行するための基本的な情報を保持するマスタテーブルを操作するメソッドはそれ以外のメソッドと異なる特徴があると考え，MosP におけるマスタテーブルを操作するメソッドのみをクラスタリングの対象とした場合と，すべてのメソッドを対象とした場合で，それぞれ評価を行う．

それぞれのシステムの人手によるクラスタリングは，企業の研究者に作成してもらった．MosP については，その操作マニュアルとソースコードを参考にし，販売管理システムについては，その仕様書とソースコードを参考にして，動作が類似しているメソッドに分類した．

提案手法を適用するには，外部アクセスに対応するメソッドを定義する必要があるため，

[2] http://sourceforge.jp/projects/mosp/releases/62164

表 4: 対象システム

対象システム	対象メソッド数	テーブル数
MosP（マスタ）	38-43	7-8
MosP（全体）	253-334	56-75
販売管理システム	9	6

表 5: MosP におけるメソッドの対応付け

外部アクセス	対応するメソッド
SELECT 文の実行	各種 Dao クラスの findFor で始まるメソッド
INSERT 文の実行	各種 Dao クラスの insert メソッド
UPDATE 文の実行	各種 Dao クラスの update メソッド
DELETE 文の実行	各種 Dao クラスの delete メソッド，logicalDelete メソッド
画面からの入力	各種 Vo クラスの get で始まるメソッド
画面への出力	各種 Vo クラスの set で始まるメソッド

著者がソースコードとコメントを読み，表 5 のような対応付けを行った．MosP では，クラス名が ”Dao” で終わるクラスがデータベースを操作する機能を提供しており，クラス名が ”Vo” で終わるクラスが操作画面とシステムの間で値の受け渡しをしている．販売管理システムについても，ソースコードと仕様書を読み，メソッドの対応付けを行った．

提案手法が，凝集度と結合度に着目したクラスタリング手法とは異なるクラスタリングを行うものであることを確認するために，メソッドの呼び出し関係を用いたクラスタリングを実装した．呼び出し関係を用いたクラスタリングは，各メソッドを頂点とし，呼び出される可能性のあるメソッド間を辺で接続したグラフに，提案手法と同様に Newman らのアルゴリズムを適用することによって行った．また，提案手法の適切な比較対象となる既存研究が存在しないため，メソッド名が同じであるものを 1 つのクラスタとするクラスタリングを行い，メソッド名を利用しない本手法を評価するための目安とした．

4.2 結果と考察

4.2.1 MosP（マスタ）

MosP のマスタに対して手法を適用した結果を表 6 に示す．表における MoJoFM の列は，手法によるクラスタリングが人手によるクラスタリングにどのくらい近いかを MoJoFM で計測した値である．NED は，クラスタの分布が極端でないかを計測した値であり，MoJoSim は，連続したすべてのバージョンに対するクラスタリング間の MoJoSim の平均値であり，

表 6: MosP（マスタ）に対する指標

	N=1			N=3			N=5		
閾値	MoJoFM	NED	MoJoSim	MoJoFM	NED	MoJoSim	MoJoFM	NED	MoJoSim
0.0	71.05	**1.0**	98.7	**92.11**	**1.0**	**100**	92.11	**1.0**	**99.76**
0.1	71.05	**1.0**	98.7	**92.11**	**1.0**	99.2	**94.74**	0.95	99.73
0.2	71.05	**1.0**	98.7	86.84	0.93	99.47	84.21	0.88	99.74
0.3	50.00	**1.0**	98.93	86.84	0.88	99.73	81.58	0.84	98.73
0.4	71.05	**1.0**	96.82	81.58	0.84	98.97	68.42	0.63	98.97
0.5	52.63	0.98	**100**	76.32	0.65	98.78	57.89	0.49	97.95
0.6	52.63	0.98	**100**	63.16	0.63	97.69	47.37	0.33	97.96
0.7	71.05	0.93	97.15	52.63	0.44	97.69	36.84	0.16	96.65
0.8	68.42	0.74	98.49	36.84	0.16	96.14	28.95	0.12	96.89
0.9	71.05	0.65	97.45	28.95	0.12	96.12	26.32	0.12	97.16
1.0	**78.95**	0.63	97.45	26.32	0.12	97.16	26.32	0.12	97.16

呼び出し関係によるクラスタリングの MoJoFM：10.53

メソッド名によるクラスタリングの MoJoFM：92.11

手法の安定性を表している．表の各列において最も高い値を太字で表記している．また，表の最下段には，メソッドの呼び出し関係によるクラスタリングとメソッド名によるクラスタリングにおける MoJoFM の値をそれぞれ示している．

表 6 から，どのようなパラメータを設定しても，MoJoSim が高いことがわかる．これは，一般にマスタテーブルに対する機能に変更が加えられることが少ないためであると考えられる．MoJoFM と NED については，N を 3 あるいは 5 とし，閾値を 0.1 以下に設定すると，それぞれ高い値が得られている．メソッド名によるクラスタリングの MoJoFM は 92.11 であったため，提案手法の信頼性は十分高いと言える．提案手法によるクラスタリングが，メソッド名のよるクラスタリングと同等の信頼性が得られた理由として，マスタテーブルに対する操作がテーブルの種類に関係なく類似しやすいという特徴が考えられる．たとえば，勤務地マスタと所属マスタのそれぞれに対する検索処理は，入力フォームから受け取る検索項目は異なるが，SELECT 文によってテーブルを検索し，その結果を一覧にして画面に表示するという処理は共通である．提案手法は，外部アクセスの抽出によってこれらの共通した処理を認識し，類似したメソッドを正確に分類することができたと考えられる．また，N を 1 とし，閾値を低く設定した場合は，人手によるクラスタリングでは異なるクラスタに属するメソッドが同じクラスタにまとめられてしまい，MoJoFM が低くなっている．逆に，閾

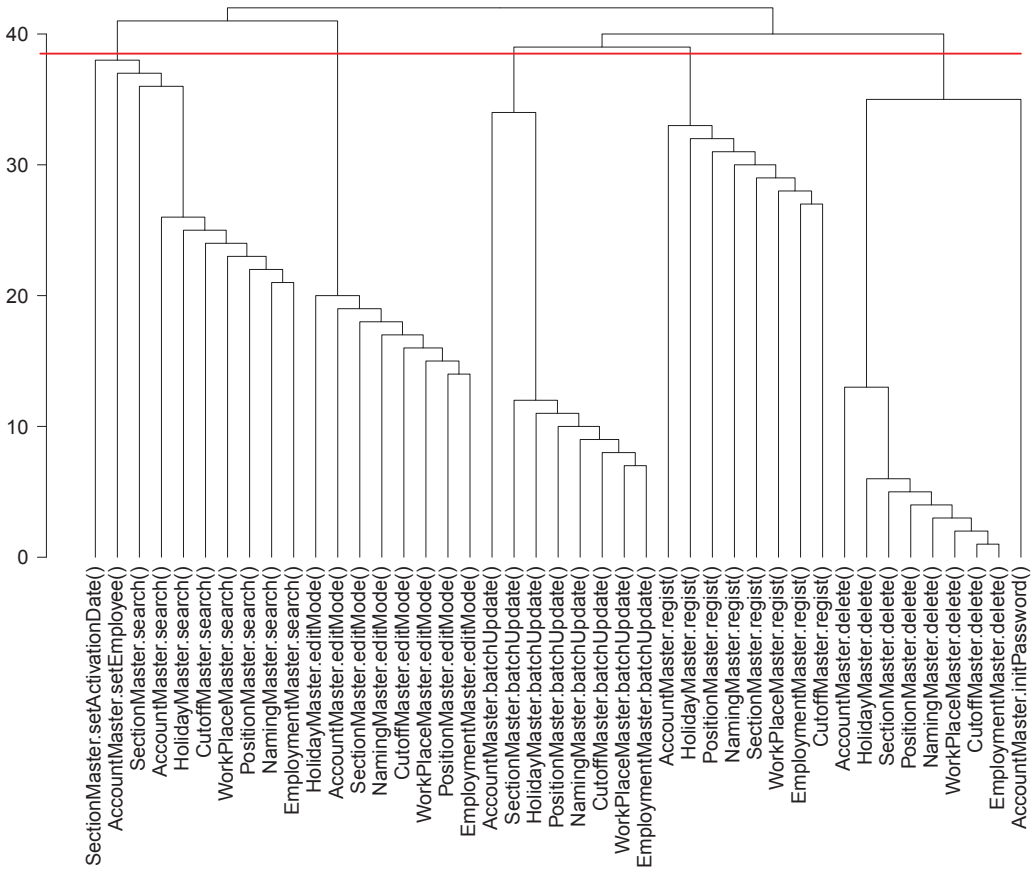

図 8: MosP（マスタ）に対する提案手法によるクラスタリングの樹形図

値を高く設定した場合は，人手によるクラスタリングでは同じクラスタに属するメソッドが異なるクラスタに分断されてしまい，MoJoFM が低くなっている．このことから，各メソッドの外部アクセスを列として比較することが有効であったと考えられる．

また，呼び出し関係によるクラスタリングの MoJoFM は 10.53 であり，提案手法とは大きく異なるクラスタリングを行っていることが分かる．実際に，提案手法と呼び出し関係によるクラスタリングはそれぞれ図 8，9 のような結果であった．図 8 は，N を 3 とし，閾値を 0.1 とした提案手法によるクラスタリングの過程を樹形図で表したものである．図 8 では，操作の対象となるテーブルによらず，同名のメソッドが 1 つのクラスタとしてまとめられている．一方で，図 9 は，呼び出し関係によるクラスタリングの過程を樹形図で表したものであり，図 9 では，操作対象が同じテーブルであるメソッドが 1 つのクラスタとしてまとめられている．以上の結果から，提案手法は動作が類似しているメソッドをクラスタとしてまとめており，凝集度と結合度に着目した手法は扱うデータが類似しているメソッドをクラスタとしてまとめていることが分かる．

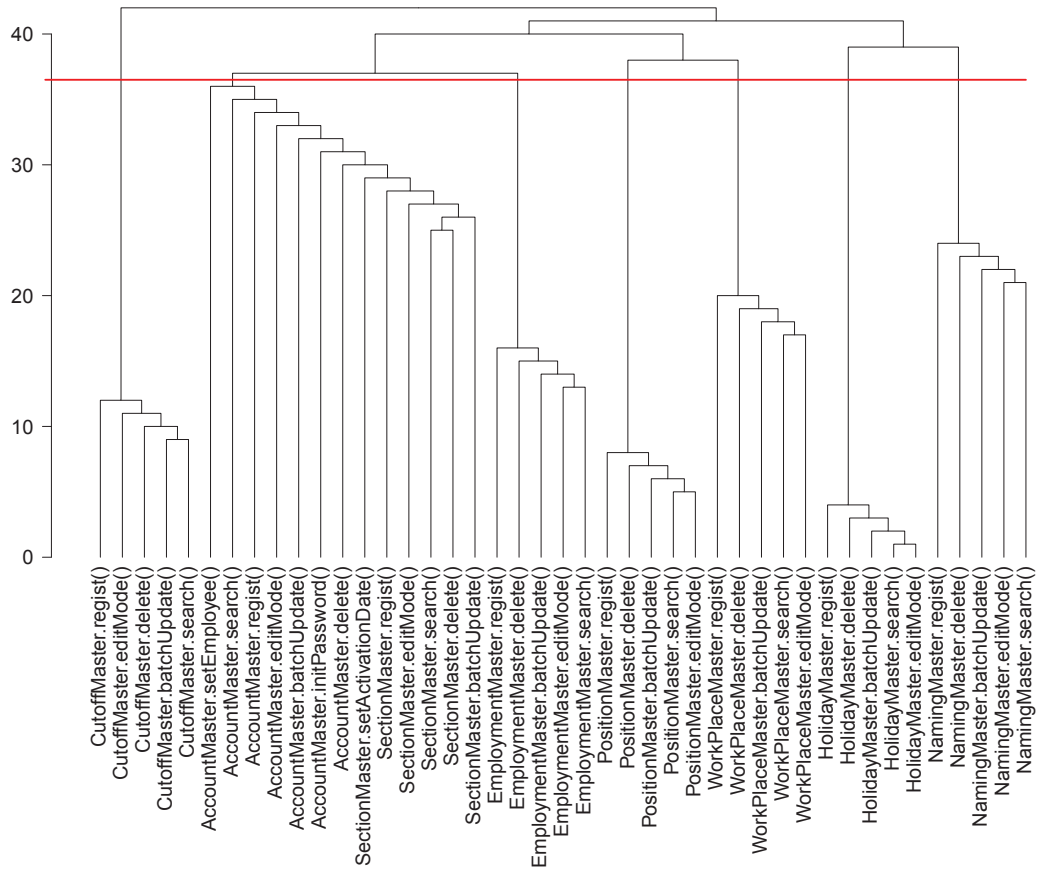

図 9: MosP（マスタ）に対する呼び出し関係によるクラスタリングの樹形図

4.2.2 MosP（全体）

MosP 全体に対して手法を適用した結果を表 7 に示す．表 7 から，MoJoSim は，マスタに対する結果と同様にパラメータによらず高いことが分かる．これは，機能の実装が変更されても，外部アクセスが大きく変わることが少ないためであると考えられ，手法の安定性が高いことを示している．NED は，N を 5 とし，類似度の閾値を 0.1 としたときが最も高い値となっており，クラスタの大きさが適切であることを示している．一方で，MoJoFM はマスタに対する結果と比べて低い値となっており，特に，N を 3 あるいは 5 とした場合は大きく減少している．これらの結果から，MoJoFM と NED はパラメータによって大きく変動することが分かったため，これら 2 つの指標に着目した分析を行った．

表 8 は，MoJoFM と NED のそれぞれの値によってパラメータの順位付けを行い，2 つの順位の和の上位 10 件を一覧にしたものである．括弧内の数字は，それぞれの指標の値を表す．表 8 から，N を 1 とし閾値を 1.0 とすると MoJoFM は最も高いが，NED は N を 3 や

表 7: MosP（全体）に対する指標

	N=1			N=3			N=5		
閾値	MoJoFM	NED	MoJoSim	MoJoFM	NED	MoJoSim	MoJoFM	NED	MoJoSim
0.0	36.70	0.0	98.26	38.23	0.19	91.11	39.14	0.21	92.87
0.1	36.70	0.0	98.33	32.72	0.17	88.67	**50.76**	**0.94**	92.29
0.2	36.70	0.0	97.88	45.57	0.70	94.67	48.62	0.74	95.73
0.3	35.78	0.0	96.81	**46.48**	**0.73**	95.01	46.18	0.51	97.05
0.4	36.70	0.0	98.15	44.95	0.54	97.23	42.51	0.43	96.78
0.5	42.81	0.11	96.81	44.04	0.42	97.18	35.47	0.34	96.69
0.6	37.61	0.0	94.60	36.70	0.36	96.88	29.97	0.27	97.13
0.7	45.26	0.18	95.98	31.19	0.26	97.41	27.22	0.24	97.14
0.8	49.85	0.23	**98.34**	25.69	0.20	97.55	20.49	0.16	97.91
0.9	50.46	0.21	97.82	18.35	0.10	98.27	16.51	0.10	98.72
1.0	**55.35**	**0.59**	96.94	15.60	0.08	**98.98**	14.98	0.08	**99.01**

呼び出し関係によるクラスタリングの MoJoFM：26.91

メソッド名のよるクラスタリングの MoJoFM：77.37

5 とする場合に比べて低い値になっている．これは，手法の分類が非常に細かく，人手によるクラスタリングで異なるクラスタに属するメソッドが同じクラスタに含まれてしまうことは少ないが，要素数 1 のクラスタが数多く形成されてしまうためである．実際に，88 個のクラスタのうち，73 個のクラスタが要素数 5 個未満の非常に小さいクラスタであった．一方，N を 3 あるいは 5 とし閾値を低く設定すると N を 1 とした場合に比べて NED は高いが，MoJoFM の値が低くなっている．これは，手法のクラスタリングにおいて，人手によるクラスタリングで異なるクラスタに属するメソッドが同じクラスタに含まれてしまったためである．実際に，N を 5 とし，閾値を 0.1 としたクラスタリングにおけるあるクラスタでは，27 個のすべて要素が search メソッドであったが，別のクラスタでは，13 個の batchUpdate メソッドと 8 個の delete メソッドが混在していた．また，クラスタリングの要素数は図 10 のよう分布となった．図 10 から，小さいクラスタは存在しているが，N を 1 とした場合と比べて，クラスタの大きさが適切であることが分かる．

これらの結果から，人手によるクラスタリングで異なるクラスタに属するようなメソッドを提案手法は同じクラスタに含めてしまうことが分かった．そこで，表 8 においてクラスタ数が少ない 3 つのクラスタリングについて，各クラスタに対してクラスタリングを適用し，サブクラスタへの分類を行うという追加調査を行った．その結果を表 9 に示す．表 9 か

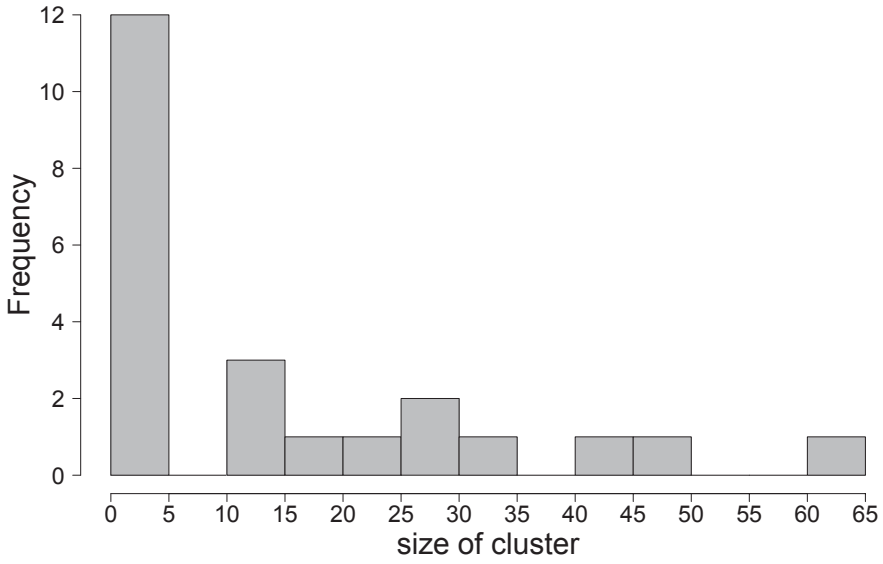

図 10: N=5，閾値 0.1 としたクラスタリングの要素数の分布を表すヒストグラム

表 8: MosP（全体）に対するパラメータの順位付け

N, 閾値	MoJoFM 順位	NED 順位	順位の和	クラスタ数
N=5, 0.1	2（50.76）	1（0.94）	3	23
N=1, 1.0	1（55.35）	5（0.59）	6	88
N=5, 0.2	5（48.62）	2（0.74）	7	79
N=3, 0.3	6（46.48）	3（0.73）	9	84
N=3, 0.2	8（45.57）	4（0.70）	12	25
N=5, 0.3	7（46.18）	7（0.51）	14	135
N=3, 0.4	10（44.95）	6（0.54）	16	126
N=1, 0.8	4（49.85）	15（0.23）	19	25
N=1, 0.9	3（50.46）	16.5（0.21）	19.5	57
N=3, 0.5	11（44.04）	9（0.42）	20	152

ら，サブクラスタへの分類によって MoJoFM が向上しており，特に N を 3 あるは 5 とした場合は，NED も高い値となっていることが分かる．メソッド名によるクラスタリングの MoJoFM は 77.37 であったため，これらの値はメソッド名を利用しない手法としては高い数値であると考えられる．

手法を実際のシステムに適用することを考えると，N-gram と閾値の適切な値を開発者

表 9: サブクラスタへの分類

N, 閾値	MoJoFM	NED	クラスタ数
N=5, 0.1	59.94	0.88	48
N=3, 0.2	59.63	0.83	50
N=1, 0.8	56.27	0.66	32

が判断する必要がある．MoJoFM の計測には人手によるクラスタリングが必要であるが，NED の計測には不要であるため，複数通りのクラスタリングを試行することができる場合は，NED の値によって判断することが考えられる．表 7 において MoJoFM と NED の相関係数は 0.62 であり，正の相関があることが確認されたため，NED の値が高いクラスタリングを選択すれば，それは信頼性の高いクラスタリングであることが期待できる．また，システム開発では複数人で作業を分担することが一般的であるため，クラスタ数を考慮して判断することも考えられる．ただし，大規模システムでは複数通りのクラスタリングを試行することが困難であることも考えられる．そのような場合は，表 9 の結果から，N を 3 あるいは 5 とし，閾値を 0.1 あるいは 0.2 などの低い値に設定し，段階的にクラスタリングを適用することで信頼性が高く，クラスタの大きさが適切なクラスタリングが得られることが期待できる．

4.2.3 販売管理システム

販売管理システムは規模が小さく，パラメータによる差異が小さいため，N を 3 とし，閾値なしでクラスタリングを行った結果を示す．図 11 は，手法によるクラスタリングの結果を樹形図で示したものである．update1，search1 などの要素名は，人手によるクラスタリング結果を表しており，数字を除いて同じ名前のものが同じクラスタに分類されたことを意味する．図 11 から，検索を行うメソッドについては手法と人手による分類が一致しているが，その他のメソッドは両者の分類が異なっている．これは，発行する SQL 文が異なるメソッドでも，類似した動作を実現していることがあるためだと考えられる．たとえば，batch1 と batch2 はマスタテーブルの情報を一括で更新する処理であるが，batch1 はテーブルのレコードを全件削除した後，新たなレコードを INSERT 文によって登録することで更新処理を行っており，batch2 は UPDATE 文によって既存のレコードの情報を書き換えて更新処理を行っている．手法では，これらのメソッドが発行する SQL 文が異なるため，メソッド間の類似度は小さくなるが，人手によるクラスタリングでは，両メソッドとも複数のレコードを一括で更新する処理であることから似ているメソッドであると判断された．このように，人手によるクラスタリングでは，発行する SQL 文が異なっていても，それらの文が実現し

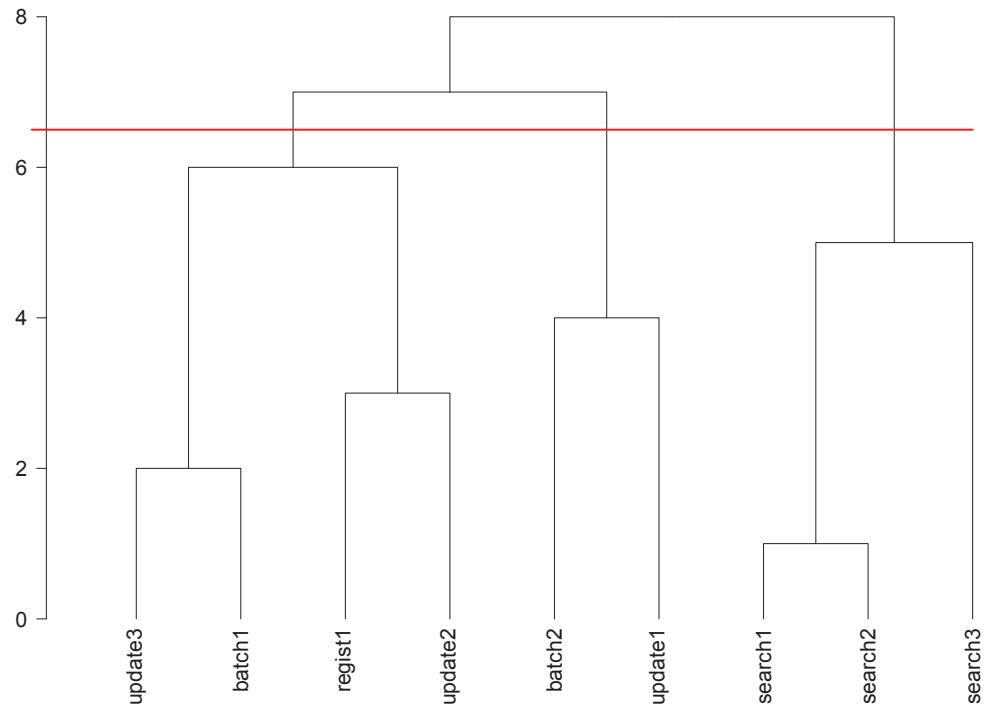

図 11: 販売管理システムのクラスタリング

ている処理の内容を考慮するため，提案手法によるクラスタリングと異なる結果になったと考えられる．

4.3 妥当性への脅威

本実験では，企業の研究者が作成したクラスタリングを用いて手法の評価を行った．研究者は，対象システムのソースコードを見て分類を行ったが，ソースコード中のメソッド名を参考にしたため，命名が不適切なものがあった場合は，目的と異なる分類を行っている可能性がある．また，異なる開発者が作成したクラスタリングを用いた場合は，本実験と結果が異なる可能性がある．

手法を対象システムに対して適用するために，著者が外部アクセスとメソッドの対応付けを行った．著者は，システムのソースコードを読解した上で，表 5 のようなクラス名とメソッド名によるルールで外部アクセスとメソッドの対応付けを行ったが，すべてのソースコードを目視で確認することは行っていない．そのため，実際には外部アクセスとして相応しくないメソッドを対応付けていたり，対応付けるべきメソッドを見逃している可能性がある．これらの対応付けの方法によって，実験結果が異なる可能性がある．

5　手法の実用化に向けた議論

提案手法は，識別子が有用でないシステムに対して適用することを想定している．そのため，実際のシステムで外部アクセスと関数の対応付けを行うことは，本実験用いたような Java システムにおける作業ほど容易ではない．しかし，一般的に，外部アクセスを行うためには決められた API や READ/WRITE などのシステムコールを呼び出す必要があるため，外部アクセスに対応する可能性のある関数や命令は限定される．また，対象システムの保守を行っている開発者であれば，部分的な理解しかできていない場合でも，どの関数によって外部アクセスが行われているかを判断できると考えられる．保守開発者が判断を行う作業は必要であるが，その結果を用いて提案手法によるクラスタリングを行うことで，その他の開発者との知識の共有が可能になり，保守開発者へのヒアリングが集中することを避けることができる．また，1 つのクラスタに属する関数を類似処理としてまとめて理解することで，システム全体の理解が効率化されると考えられる．

また，本手法は，ファイル数が 10 万を超えるような大規模システムに対して適用することを目指している．そのため，時間的な計算コストが小さいアルゴリズムが要求される．本手法では，Step 1 における構文解析，Step 2 における N-gram によるジャッカード係数の計算，Step 3 における Newman らのクラスタリングアルゴリズムが主な処理となる．構文解析については，プログラミング言語によって異なるが，一般に高速に行うことができるように設計されていることが多い．N-gram によるジャッカード係数の計算では，すべての集合間で類似度を計算する必要があるが，並列計算による高速化が可能である．Newman らのクラスタリングアルゴリズムは，頂点数が 40 万を超えるグラフに対しても適用されており [6]，大規模システムへの適用可能性が高い．

6　まとめと今後の課題

本研究では，ソースコード中の各関数からシステムの操作画面やデータベースへのアクセスを行う命令を抽出し，動作が類似している関数の集合に分類するクラスタリング手法を提案した．評価実験では，2 つ Java システムに対して手法を適用し，クラスタリング結果を評価する指標を計測した．その結果，マスタテーブルを操作するメソッドのみを対象とした場合は，人手によるクラスタリングと非常に近い結果が得られた．また，システム全体を対象とした場合は，クラスタリングを段階的に適用することで，人手によるクラスタリングに近づくことが分かった．

今後の課題としては，企業で古くから開発が続いている大規模なシステムに対して手法を適用することが挙げられる．大規模システムでの適用結果の分析や，開発者からのフィードバックを得ることなどによって，手法の改善を行うことが考えられる．特に，非常に古いシステムでは，リレーショナルデータベースが使われていないこともあるため，ファイルの読み書きを行う命令を外部アクセスに対応付けてクラスタリングを行うことなども発展的な課題として挙げられる．また，提案手法は，外部アクセスに対応するメソッドを開発者に定義してもらう必要があるため，メソッドの呼び出し関係などから，これらのメソッドを自動的に推定することも考えられる．さらに，手法によるクラスタリング結果を利用すると，ソースコードの読解にどのくらい影響するかといったような，システム理解における開発者の作業を定量的に分析することも考えられる．

謝辞

本論文を執筆するにあたり，多くの方々に支えていただきました．ここに謝意を添えて御名前を記させていただきます．

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 教授には，ご多忙であるにも関わらず，居室時に声をかけてくださり，研究活動全般にわたってご指導を賜りました．また，私の将来についても時間を割いて考えてくださり，多くのご相談をさせていただきました．本研究室で 3 年間にわたりご支援を頂いたこと，また，新たな挑戦の機会を頂いたことに大変感謝しております．

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 准教授には，研究室内での中間報告や発表練習において，積極的にご意見を頂き，活発な議論へと導いてくださいました．頂いたご意見は，研究の新たな視点となり，よい研究発表をするための糸口となりました．深く御礼を申し上げます．

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 助教には，私が研究室に配属された当初から常々ご指導を賜りました．研究テーマの決定，研究の遂行，論文執筆，研究発表などの様々な場面でご助言を頂き，大変お世話になりました．本研究室での生活は，私にとって非常に貴重な経験であり，自身の成長につながりました．心より深く感謝しております．

株式会社 NTT データ 師 芳卓 氏，坂田 祐司 氏には，研究課題の策定から評価実験に至るまで様々な議論をさせていただきました．師氏には，実験を行うためのデータを提供していただきました．このデータは，本研究の進展に欠かせないものです．深く御礼を申し上げます．また，同社 岡田 譲二 氏には，本研究に取り組むきっかけとなる貴重な経験をさせていただきました．心より感謝しております．

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 鹿島 悠 氏には，研究を進めるにあたり，様々な場面でご相談に乗っていただきました．論文執筆や発表練習などにおいて，多くのご意見を頂き，研究活動の支援をしていただいたことに深く感謝しております．

最後に，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のおかげで，充実した研究生活を送ることができました．心より感謝いたします．

参考文献

[1] P. Andritsos and V. Tzerpos. Information-theoretic software clustering. *IEEE Transactions on Software Engineering*, Vol. 31, No. 2, pp. 150–165, 2005.

[2] N. Anquetil and T.C. Lethbridge. Recovering software architecture from the names of source files. *Journal of Software Maintenance*, Vol. 11, No. 3, pp. 201–221, 1999.

[3] N. Anquetil and T.C. Lethbridge. Comparative Study of Clustering Algorithms and Abstract Representations for Software Remodularization. *IEE Proceedings-Software*, Vol. 150, No. 3, pp. 185–201, 2003.

[4] U. Brandes, D. Delling, M. Gaertler, G. Robert, M. Hoefer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20, No. 2, pp. 172–188, 2008.

[5] S. Cesare and Y. Xiang. Malware Variant Detection Using Similarity Search over Sets of Control Flow Graphs. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 181–189, 2011.

[6] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, Vol. 70, No. 6, 2004.

[7] S. Ducasse and D. Pollet. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE Transactions on Software Engineering*, Vol. 35, No. 4, pp. 573–591, 2009.

[8] Ural E., Umut T., and Feza B. Object Oriented Software Clustering Based on Community Structure. In *Proceedings of the 18th Asia-Pacific Software Engineering Conference*, pp. 315–321, 2011.

[9] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder : A Multi-linguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670, 2002.

[10] K. Kobayashi, M. Kamimura, K. Kato, K. Yano, and A. Matsuo. Feature-gathering dependency-based software clustering using Dedication and Modularity. In *Proceedings of the 28th IEEE International Conference on Software Maintenance*, pp. 462–471, 2012.

[11] K. Kobayashi, M. Kamimura, K. Yano, K. Kato, and A. Matsuo. SArF map: Visualizing software architecture from feature and layer viewpoints. *Proceedings of the 21st International Conference on Program Comprehension*, pp. 43–52, 2013.

[12] R. Koschke and D. Simon. Hierarchical Reflexion Models. In *Proceedings of the 10th Working Conference on Reverse Engineering*, pp. 36–45, 2003.

[13] S. Mancoridis and B.S. Mitchell. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 50–59, 1999.

[14] O. Maqbool and H. Babri. Hierarchical Clustering for Software Architecture Recovery. *IEEE Transactions on Software Engineering*, Vol. 33, No. 11, pp. 759–780, 2007.

[15] S. McConnel. *Code Complete, Second Edition.* Microsoft Press, 2004.

[16] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *Proceedings of the 34th International Conference on Software Engineering*, pp. 364–374, 2012.

[17] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl. A Reverse Engineering Approach To Subsystem Structure Identification. *Journal of Software Maintenance: Research and Practice*, Vol. 5, No. 4, pp. 181–204, 1993.

[18] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, Vol. 69, No. 6, pp. 1–5, 2004.

[19] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico. Using the Kleinberg Algorithm and Vector Space Model for Software System Clustering. In *Proceedings of the IEEE 18th International Conference on Program Comprehension*, pp. 180–189, 2010.

[20] H. M. Sneed. Object-oriented COBOL recycling. In *Proceedings of the 3rd Working Conference on Reverse Engineering*, pp. 169–178, 1996.

[21] G. Sridhara, L. Pollock, and K. Vijay-Shanker. Automatically detecting and describing high level actions within methods. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 101–110, 2011.

[22] V. Tzerpos and R.C. Holt. MoJo: a distance metric for software clusterings. In *Proceedings of the 6th Working Conference on Reverse Engineering*, pp. 187–193, 1999.

[23] V. Tzerpos and R.C. Holt. ACDC: An algorithm for comprehension-driven clustering. In *Proceedings of the 7th Working Conference on Reverse Engineering*, pp. 258–267, 2000.

[24] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *Proceedings of 12th IEEE International Workshop on Program Comprehension*, pp. 194–203, 2004.

[25] J. Wu, A.E. Hassan, and R.C. Holt. Comparison of clustering algorithms in the context of software evolution. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pp. 525–535, 2005.

[26] 一般社団法人 日本情報システム・ユーザー協会（JUAS）. ソフトウェアメトリックス調査 2012. 2012.