情報処理学会論文誌 Vol.53 No.3 1137–1149 (Mar. 2012)

# ソースコードと設計書を用いたソフトウェアの派生関係の抽出

岸本 康成[1,a)] 坂本 啓[1] 小林 透[2]

受付日 2011年3月23日, 採録日 2011年12月16日

**概要**：アプリケーション統合やプロダクトライン開発において，既存ソフトウェアプロダクトを分析し，再利用可能な共通機能を見極めることは重要である．共通性分析を行う場合，分析対象とするプロダクト群の派生関係を抽出し，共通性の高いグループに絞り込むことが効率的である．本研究ではプロダクトを絞り込むスコーピングに焦点を当てる．スコーピングを支援するためにマイニング技術によりプロダクトの分類を行う手法を提案する．本手法の特徴はソースコードを用いたソフトウェア分類手法をベースとして，さらに設計書を用いたソフトウェア分類手法を利用することにより正確なプロダクトの分類を可能としていることである．提案手法を実際に企業のソフトウェアプロダクト群に対して適用したところ，ソースコード単独による分類手法に比べより正確に類似したプロダクトのグループに分類でき，スコーピング支援に対する有用性が高いことが確かめられた．

**キーワード**：ソフトウェア類似性，コードクローン，テキストマイニング，クラスタリング，共通性分析

# Extraction of Derived Relations among Software Products Using Both Source Cord and Specification Document

Yasunari Kishimoto[1,a)] Akira Sakamoto[1] Toru Kobayashi[2]

Received: March 23, 2011, Accepted: December 16, 2011

***Abstract:*** It is important to find out the reusable common functions by the analysis of existing products on the application integration or the product line development. In the commonality analysis, it is efficient to extract the derived relations among them and narrow them down to the products which seem to be similar. We focus on the scoping that narrows products. We propose the method of clustering the products by applying mining techniques to support the scoping. The feature of the proposed method is that it is based on the clustering method using the source code and additionally uses the clustering method using the specification document to achieve more accurate clustering results. The proposed method is actually applied to the software products of an enterprise and it is confirmed that the method enables to achieve more accurate clustering results than the method using only source code. It is useful for the scoping.

***Keywords:*** software similarity, code clone, text mining, clustering, commonality analysis

## 1. はじめに

現在，企業において部門ごとに個別に最適化されたアプリケーションから企業全体のITシステムの最適化を目指すというアプリケーション統合の動きが進んでいる．しかし，既存アプリケーションをそのまま新しいコンピュータ環境上に載せただけでは，類似性の高い機能が分散実装されてしまう可能性があり，メンテナンス性の観点で問題がある．このような問題を解決するには，既存アプリケーションに対して共通性分析を行って共通機能を切り出し，共通基盤として利用可能とすることが重要である．また，

[1] NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories, Musashino, Tokyo 180–8585, Japan
[2] NTT サイバーソリューション研究所
NTT Cyber Solutions Laboratories, Yokosuka, Kanagawa 239–0847, Japan
[a)] kishimoto.yasunari@lab.ntt.co.jp

近年，再利用性を高めるための開発手法としてプロダクトライン開発 [1] が注目されているが，ここでも要求定義をプロダクトに共通な要求と可変な要求とに分けて管理することが重要とされており，共通性分析が行われている．

共通性分析を行う場合，分析対象とするプロダクト群の派生関係を抽出し，共通性が高いグループにあらかじめ絞り込むことが有効である．この作業のことを本研究ではスコーピングと呼ぶ．ここで，派生関係とは，clone-and-own 的にコードが再利用され，派生品の開発が行われ，進化してきたプロダクト群があった場合に，それらのプロダクト間の関係を意味している．本来，スコーピングとは，プロダクトライン開発において対象とするプロダクト群の範囲を決定することであり [1]，スコーピングで選ばれたプロダクト群に対して共通性分析や可変性分析が行われるが，本研究ではプロダクトライン開発に限らず共通性分析を行う対象を絞り込むことをスコーピングと定義する．Clements らはプロダクトラインにおけるスコーピングとは内側に含まれるものと外側にあるものを決めることであり，手短にいえば，スコーピングとはどのプロダクトが自分のプロダクトラインに含まれるべきかという質問に答えることであると述べている [2]．さらに，Schmid は，プロダクトラインにおけるスコーピングを Product Portfolio Scoping，Domain Scoping，Asset Scoping の 3 つのレベルに定義されると述べている [3]．Product Portfolio Scoping でプロダクトの範囲を決定し，それを基に Domain Scoping でドメインの範囲を決め，さらに Asset Scoping で再利用される資産の範囲を決定する．Product Portfolio Scoping はマーケティングの観点で対象とするプロダクトを決定することであり，これは本稿で定義するスコーピングの範囲ではない．本稿でのスコーピングは Domain Scoping 以降のレベルにおいて共通性分析の対象となるプロダクトの絞り込みを行うことに対応している．スコーピングのためのプロダクトの分析を人手で行う場合，特に企業においては，異動等でプロダクトに詳しい人がいない，プロダクトの管理がきちんと行われていない，プロダクトの数や規模が大きい等の様々な理由により分析コストが大きくなる傾向がある．

著者らはスコーピングにマイニング技術を用いて支援することができれば分析コストの低減につながるのではないかと考えた．そこで，著者らはスコーピングを支援するためにマイニング技術を用いてソフトウェアプロダクト群を派生関係がある類似プロダクト群に分類する手法を提案する [4]．マイニング技術を用いたソフトウェアの分類に関する既存研究はあるが，たとえば，ソフトウェアリポジトリにおけるソフトウェアの探索を容易にする等の目的で行われているものが多く，著者らが定義した共通性分析を行う対象を絞り込むスコーピングを目的とした先行事例はない．マイニング技術によって自動的にソフトウェアプロダクトを分類できれば，人手による分析を行わずに，多数あるプロダクト群から関連あるプロダクトを絞り込むことができる．もちろん，最終的な絞り込みには人手による詳細な分析が必要と考えられるが，第一次的な情報として自動分類の結果や類似度を活用できれば，分析コストの低減につながることが期待できる．具体的にはあらかじめ分析対象外のプロダクトを排除するのに用いたり，同じ分類グループの中でも，どのプロダクトが類似性が高いのかといった予備知識が得られることにより，類似性が高いプロダクトどうしから分析を進めていく等の効率的な分析が可能になると考えられる．ただし，スコーピングにマイニングを導入するにあたって，分析に利用可能なデータの量や品質が問題となる．分析に用いるソースコードや設計書等のデータについて十分なデータ量が得られない場合やデータ量が十分あっても設計書の記述レベルに統一性がない等のデータ品質に問題があれば，信頼性の高いマイニング結果は得られないと考えられる．

マイニング技術によりソフトウェアを分類する手法としては，ソースコードを用いる手法と設計書を用いる手法の 2 種類がある．ソースコードを用いる手法では，プログラミング言語という explicit な言語による類似性のため，プロダクトが機能的に類似していても設計やアーキテクチャが大きく異なる場合，類似性が抽出されにくいという問題があるが，プロダクト間の類似性が過大に抽出されてしまうという問題は生じにくい．一方，設計書を用いる手法では，設計書による類似性が単語の分布の類似性を利用するため，プロダクト間の類似性が過大に抽出されてしまうという問題があるが，設計やアーキテクチャの違いで類似性が抽出されにくくなるといった問題は生じにくい．従来，これらの 2 つの分類手法は存在するが，既存研究では同一対象に対して両手法の組合せによる分類手法に関する報告は少ない．そこで，本研究では上記の両手法によって得られる類似性の違いに関する仮説を構築し，その仮説に基づいて両手法を組み合わせたソフトウェアの分類手法を提案する．また，提案手法を実際の企業のプロダクト群に適用して検証した結果を示す．

以降，2 章でスコーピングにおける問題について述べ，3 章で問題を解決するためにマイニング技術を用いる手法を提案する．そして，4 章で実際の企業のソフトウェアプロダクト群に対して行った実験およびその結果を示す．さらに，5 章で実験結果の考察，6 章で関連研究について述べ，7 章で本研究のまとめを行う．

## 2. スコーピングの問題点

本章では，スコーピングのための既存プロダクトの分析を行うにあたってのいくつかの問題点が，分析コストの増大につながり，そのことが本研究の動機付けになっていることを述べる．

企業においてスコーピングのために既存プロダクト群の

分析を行う場合，以下に示す問題がある．

(1) 人的な問題

企業では時間経過にともなう人事異動等によりプロダクトに詳しい担当者がそのプロダクトを分析できないことがある．

(2) 組織的な問題

企業，特に大企業では縦割り組織が多いため組織をまたがって企業全体の保有するプロダクト群について把握することは難しいという問題がある．

(3) プロダクト管理面の問題

プロダクトの管理において，ソースコードの修正が設計書には反映されずに設計書とソースコードの内容に乖離が生じることがあり，このことがプロダクトの分析を困難にする．

(4) 開発形態の問題

企業におけるソフトウェアの開発形態として，内製ではなく外注による開発形態となることも多い．その場合，実際の開発者へのリーチが困難になるケースがある．

(5) 量的な問題

プロダクトの規模が大きい場合，分析コストが増大する傾向がある．

このような問題があるため，既存プロダクトの分析をすべて人手で行うのはコストが大きい．そこで，著者らは，分析をサポートするための手法の1つとして，マイニング技術を用いてソフトウェアプロダクトを類似したグループに分類する手法を提案する．

## 3. 提案手法

### 3.1 前提

#### 3.1.1 分類の観点と粒度

ソフトウェアプロダクトを分類する場合，分類の観点や粒度を考慮する必要があるため，はじめに本研究が想定する分類の観点と粒度について述べておく．

分類の観点としては，様々なものがあるが，本研究ではスコーピング支援を目的とした分類を行うため機能的なものを想定する．ここでいう機能とは，関数レベルのそれ単独ではサービスにならないようなものではなく，たとえば電子メールサービスや勤務管理サービスといった人にとって意味のあるサービスレベルのものを意味する．

分類の粒度については，同じプロダクトのバージョン違い，もしくはそれに相等するような，機能的に類似，もしくは関連したプロダクトどうしがグルーピングされるような分類の粒度を想定する．たとえば，あるプロジェクトチームが時系列的に機能拡張により関連した一連のプロダクトをリリースしてきた場合にそれらのプロダクトは派生関係にあり同じグループに入る．本稿では今後，この粒度で同じグループに分類されるものを，シリーズと呼ぶことにする．また，ここで分類という用語の意味について説明しておく．分類といった場合，プロダクト群をあらかじめ決められたクラスに分ける場合と，あらかじめ決められたクラスはなく，類似したプロダクトどうしをクラスタリングする場合がある．本研究では分類という用語を後者の意味で用いることとする．

#### 3.1.2 類似性

類似性という言葉について定義しておく．本稿ではソースコードによる類似性と設計書による類似性という2つの類似性を取り扱う．前者はソフトウェアを実装したソースコードに含まれるトークン列が類似しているということを意味し，後者はソフトウェアの設計時に作成した設計書に含まれるテキストにおける単語の分布が類似していることを意味するものである．

### 3.2 仮説

マイニング技術によりソフトウェアの類似性を分析する手法として，分析対象種別の観点からソースコードを用いる手法と設計書を用いる手法の2種類がある．まず，ソースコードを用いた手法としてはコードクローンを利用する分析手法がある [5]．次に，設計書を用いて分析する手法については，テキストマイニング技術を用いた手法がある [6]．テキストマイニング技術を用いた類似性分析手法とは設計書に含まれる単語の出現頻度等の情報を用いてプロダクトの類似性を分析する手法を意味する．著者らは各手法単独よりも両手法を併用した方がよりプロダクトの類似性を正確に分析できるのではないかと考えた．

そこで，本研究では両手法から導かれる類似性に関して以下に示す2つの仮説を立て，その仮説に基づくソフトウェアプロダクトの分類手法を提案する．

- **仮説1**
  **ソースコードによる類似性が高いプロダクトは同じシリーズに入る可能性が高い．**
- **仮説2**
  **同じシリーズに含まれるプロダクトの中には，ソースコードの類似性は低いが，設計書の類似性はそれに比べると高いものが存在する．**

1つ目の仮説は，ソースコードが類似している場合，コードが頻繁に流用された可能性が高いため，同じシリーズに入る可能性が高いのではないかという考えに基づくものである．2つ目の仮説は，たとえば，同じシリーズに入るプロダクトであっても，構造整備や性能改善等で設計に大きな変更が生じたことにともないソースコードが大きく変わることが考えられる．このような場合，ソースコードによる類似性は低くなるが，設計書による類似性は単語の分布に基づくものなので，機能的に大きく変わらなければ，比較的類似性は保たれると考えられる．

### 3.3 提案する分類手法

3.2 節の仮説を基に，ソースコードと設計書を用いたソフトウェアプロダクトの分類手法を提案する．提案手法は以下に示す 3 つのステップで行われる．

(1) ソースコードを用いたソフトウェア分類
(2) 設計書を用いたソフトウェア分類
(3) 分類結果の合成

まず，ステップ (1) でソースコードを用いたソフトウェアの分類結果を求める．本研究では，第 1 の仮説に基づき，この分類結果を提案手法の分類のベースとする．設計書ではなくソースコードを分類のベースにするのは，第 1 の仮説以外にも 2 章で示したように，企業の場合，ソースコードの改変が必ずしも設計書に反映されない場合があることを考慮したからである．第 2 の仮説に基づき，このベースに対して，ソースコードは類似していないが設計書は類似しているようなプロダクトが同じシリーズに入るように，ステップ (2) で求めた設計書による分類結果を利用するのがステップ (3) である．ステップ (3) はソースコードによる分類結果と設計書による分類結果という 2 つの分類結果を合成するという意味で分類結果の合成と呼ぶことにする．各ステップの詳細については，以降の節で示す．

### 3.4 ソースコードを用いたソフトウェア分類

ソースコードを用いたソフトウェア分類は，以下に示す距離計算，クラスタリングの 2 つの手順により行う．

#### 3.4.1 距離計算

(1) ソースコードにおける類似度の定義

山本らの論文 [7] で用いられている類似度の定義を，著者らも同様に用いる．形式的には次のようになる．ソフトウェアプロダクト $P$ を，それを構成する要素の集合と考え，$P = \{p_1, \cdots, p_m\}$ と書くものとする．2 つのソフトウェアプロダクト $P = \{p_1, \cdots, p_m\}$, $Q = \{q_1, \cdots, q_n\}$ に対して，等価な要素の対応 $R_s \subseteq P \times Q$ が得られるとする．提案手法では本項 (2) で述べるように $R_s$ は具体的にはコードクローンを用いた対応関係を利用して定義される．$P$ と $Q$ の $R_s$ に対する類似度 $S$ $(0 \leq S \leq 1)$ を以下のように定義する．

$$S(P,Q) = \frac{|\{p \in P | \exists q \in Q, (p,q) \in R_s\}| + |\{q \in Q | \exists p \in P, (p,q) \in R_s\}|}{|P| + |Q|} \quad (1)$$

これは 2 つのソフトウェアプロダクトで等価な要素数を $P$, $Q$ の総要素数で割ったものである．$R_s = \phi$ のとき，すなわち $P$ と $Q$ の間に等価な要素が 1 つもないとき，$S(P,Q) = 0$ となる．対応 $Rs$ に関係する $P$, $Q$ の要素が増えること，すなわち等価な要素が増えることによって $S(P,Q)$ は大きくなる．また $\forall p \in P$（$\exists q \in Q$, $(p,q) \in R_s$）かつ $\forall q \in Q$（$\exists p \in P$, $(p,q) \in R_s$）のとき，$P$ と $Q$ が等価であると定義する．$P$ と $Q$ が等価であるとき，$S(P,Q) = 1$ となる．

(2) コードクローンを用いたソフトウェアプロダクト間の類似度

ここではソフトウェアプロダクトに使用されたプログラミング言語が単一の場合を前提とする．ソースコード中に同一あるいは類似したコード断片があるとき，それらをコードクローンという．コードクローンが発生する原因としては以下のようなものが考えられている [8]．

(a) 既存コードのコピーアンドペーストによる再利用
(b) 定型処理
(c) プログラミング言語に適切な機能がない
(d) パフォーマンス改善
(e) コード生成ツールを用いたコードの生成
(f) 複数のプラットホームに対応したコード
(g) 偶然

(a) については，一からコードを書くよりも既存コードを流用して部分的な変更を加える方が信頼性が高いこともあり多く存在する．また (b) については，定義上簡単で頻繁に用いられる処理はコードクローンとなる傾向があることを意味する．たとえば，エラー処理，初期化，キューの挿入処理等があげられる．コードクローンの定義については，厳密で普遍的なものは存在しないが，本研究では，ソースコードをトークン列と見なしたときに，改行位置等のコーディングスタイルを除いて，連続して一致する部分列がある場合に，その部分列をコードクローンと定義する．ただし，変数名や関数名等のユーザ定義名だけが異なっている場合もコードクローンと見なす．本研究では，コードクローンを検出するツールとして CCFinderX [9]（windows 版 version 10.2.7.4）を用いる．CCFinderX では，ソースコードをトークンに切り分けた形で取り扱い，ユーザ定義名（識別子）の対応付けを行ったうえで個別のパラメータに置き換えて，ソースコードの比較を行う．この結果，改行位置が異なったり，変数名が書き換えられたりしているようなソースコードからもクローンを検出することができ，著者らが定義したコードクローンの摘出に適している．ここで，このコードクローンと (1) で定義した類似度を用いて，コードクローンを用いたソフトウェアプロダクト間の類似度を定義する．まず，ソフトウェアプロダクト $P$ に含まれるソースファイルを適当な 1 つの決まった順序で結合し，トークン単位で分割したときの各トークンを $p_i$ とし，同様にプロダクト $Q$ に含まれるトークンを $q_j$ とする．今，$p_i$ を含むトークンのシーケンス $a$ と $q_j$ を含むトークンのシーケンス $b$ がコードクローンとし

て検出された場合，$p_i$ と $q_j$ は等価な要素であるとする．このとき，等価な要素の対応 $R_s$ は，等価な要素対 $(p_i, q_j)$ を元とする $P \times Q$ の部分集合として定義される．ここで式 (1) に示す類似度の定義をあてはめると，類似度 $S(P,Q)$ は以下のようになる．

$$S(P,Q) = \frac{TC_p + TC_q}{T_p + T_q} \ (0 \leq S(P,Q) \leq 1) \qquad (2)$$

ただし

$TC_p$：$P$ 上にある $Q$ とのコードクローンに含まれるトークン数，

$TC_q$：$Q$ 上にある $P$ とのコードクローンに含まれるトークン数，

$T_P$：P の全トークン数，

$T_Q$：Q の全トークン数

(3) 複数プログラミング言語が混在した場合の類似度

(2) ではソフトウェアプロダクトが単一のプログラミング言語であることを前提にしていた．ここでは，プロダクトに含まれるソースコードが複数のプログラミング言語を含む場合の類似度について説明する．まず，プロダクトに含まれるプログラミング言語の集合を $L = \{L_1, L_2, \cdots, L_i, \cdots, L_{n_l}\}$ とする．プログラミング言語 $L_i$ で記述されたソースコードに対して式 (2) によって計算した類似度を $S_i(P,Q)$ とする．ここで，$S_i(P,Q)$ を成分とする類似度ベクトル $\vec{S}(P,Q)$ を以下のように定義する．

$$\vec{S}(P,Q) = (S_1(P,Q), S_2(P,Q), \cdots, S_i(P,Q), \cdots, S_{n_l}(P,Q)) \qquad (3)$$

プロダクト間の類似度 $S(P,Q)$ は，この類似度ベクトル $\vec{S}(P,Q)$ を用いて以下のように定義する．

$$S(P,Q) = \frac{|\vec{S}(P,Q)|}{\sqrt{n_l}} \ (0 \leq S(P,Q) \leq 1) \qquad (4)$$

類似度 $S(P,Q)$ は，$L$ に含まれるすべてのプログラミング言語の類似度が 1 の場合に最大値 1 となり，すべてのプログラミング言語の類似度が 0 の場合に最小値 0 となる．なお，上記定義は同じ仕様のものを異なる言語に書き換えた場合（たとえば，C 言語で書いていたものをパフォーマンス向上のためアセンブリ言語で書き換えたような場合）等には類似性を正しく抽出できない可能性がある．よって，本研究では，このようなソースコードの書き換えが発生した部分は分析対象にしないことを前提とする．

(4) 類似度から距離（非類似度）への変換

(2) もしくは (3) で定義した類似度に基づき，クラスタリングの際に用いる距離（非類似度）$D$ を定義する．類似度が高いとき距離としては近く（小さい），逆に類似度が低いとき距離としては遠く（大きい）なるように以下のように規定した．

$$D(P,Q) = 1 - S(P,Q) \ (0 \leq D(P,Q) \leq 1) \qquad (5)$$

ここで 2 つのソフトウェアプロダクト $P, Q$ がまったく同一である場合 $D(P,Q) = 0$，2 つのプロダクトで共通なソースコード部分がまったくない場合 $D(P,Q) = 1$ となる．

#### 3.4.2 クラスタリング

3.4.1 項で導出したソフトウェアプロダクト間の距離に対してクラスタリングを適用する．クラスタリングには代表的な手法である階層的クラスタリング法 [10] を用いる．階層的クラスタリング法は最初に $N$ 個の各プロダクト自身をクラスタとして，最も距離の近い 2 つのクラスタを結合して 1 つのクラスタを作るという処理を最終的に 1 つのクラスタになるまで繰り返して行う手法である．また，クラスタリングした結果は樹状図と呼ばれる階層構造を持った図形として可視化できる．階層的クラスタリング法はクラスタ間の距離の計算方法によって，単連結法，完全連結法，群平均法，ウォード法等様々な方法が提案されているが，本研究では単連結法を用いる．ソフトウェアプロダクトの分類は，階層的クラスタリングにおいて，結合するクラスタ間の距離の上限値を閾値として与えることにより行う．

### 3.5 設計書を用いたソフトウェア分類

本研究ではウォーターフォール型開発プロセスにより開発されたソフトウェアプロダクトを対象とする．設計書として要件定義書，基本設計書，機能設計書等のソフトウェア開発の上流工程で作成された設計書を用いる．また，比較するプロダクト間で同等な上流工程の設計書を用いることを前提とする．これは，比較に用いる設計書の記述の抽象度を統一するためのものであり，たとえば，各プロダクトから機能設計書に該当するものだけを抜き出して分類のための入力として用いる．設計書を用いたソフトウェアの分類は，以下に示す単語ベクトルの生成，単語ベクトル間の距離計算，クラスタリングの 3 つの手順により行う．

#### 3.5.1 単語ベクトルの生成

ソフトウェアプロダクト $P$ に含まれる設計書から単語 $t$ を抽出し単語ベクトル $\vec{W}(P)$ を生成する．単語ベクトル $\vec{W}(P)$ は次のように生成する．

(1) 設計書からテキストを抽出する．これは設計書が単なるテキストファイルではなく，MS-Word 等の文書作成ソフトウェアを使用している場合に必要となる処理である．

(2) (1) で抽出したテキストに含まれる半角文字はすべて全角文字に，小文字は大文字に変換する．これは，たとえば，“Web”，“Ｗｅｂ”，“ＷＥＢ” のようにまったく同じ概念を表す単語を異なる単語として取り扱われるのを防ぐためである．

(3) テキストを形態素解析し，単語を抽出する．ただし，抽出する単語は名詞のみを対象とする．また，“<” や “=” 等の記号を含む単語等の不要語を除去する．

(4) 単語ベクトル $\vec{W}(P)$ の単語 $t$ に関する成分 $w(t,P)$ を情報検索でよく用いられる TF-IDF ベースの重みを考慮し，以下のように計算する．

$$w(t,P) = tf(t,P) \times idf(t) \quad (6)$$

$$idf(t) = \log\left(\frac{n_p}{df(t)}\right) + 1 \quad (7)$$

ここで，

$tf(t,P)$：プロダクト $P$ に含まれる設計書における単語 $t$ の出現頻度，

$n_p$：全プロダクトの数，

$df(t)$：単語 $t$ が出現するプロダクトの数

#### 3.5.2 単語ベクトル間の距離計算

プロダクト $P$，$Q$ 間の距離 $D(P,Q)$ は以下に示すコサイン距離を用いて計算する．

$$D(P,Q) = 1 - \frac{\vec{W}(P)\cdot\vec{W}(Q)}{|\vec{W}(P)||\vec{W}(Q)|} \ (0 \le D(P,Q) \le 1) \quad (8)$$

#### 3.5.3 クラスタリング

3.5.2 項で導出したソフトウェアプロダクト間の距離に対してクラスタリングを適用する．クラスタリングにはソースコードによる分類の場合と同じ単連結法による階層的クラスタリング法を用いる．ソフトウェアプロダクトの分類は，階層的クラスタリングにおいて，結合するクラスタ間の距離の上限値を閾値として与えることにより行う．

### 3.6 分類結果の合成

ソフトウェアプロダクトの集合を $S_P$，$S_P$ に対するソースコードによる分類結果を $\mathcal{X} = \{X_1, X_2, X_3, \cdots, X_m\}$，設計書による分類結果を $\mathcal{Y} = \{Y_1, Y_2, Y_3, \cdots, Y_n\}$ とする．ここで，$\mathcal{X}$ については $X_i \cap X_j = \emptyset$，$X_i \subseteq S_P$，$\bigcup_{i=1}^{m} X_i = S_P$ を満たす．同様に $\mathcal{Y}$ については，$Y_i \cap Y_j = \emptyset$，$Y_i \subseteq S_P$，$\bigcup_{i=1}^{n} Y_i = S_P$ を満たす．

分類結果の合成ではソースコードの分類結果 $\mathcal{X}$ を基本にする．$\mathcal{X}$ に含まれるクラスタの中で，お互いに同じ設計書のクラスタに包含されるクラスタどうしを結合することで，新しい分類結果を生成する．

ソースコードの分類結果と設計書の分類結果を合成して新たな分類結果を生成するアルゴリズムの詳細な手順を以下に示す．

(1) 空の集合 $\mathcal{C} = \{\}$ を用意する．

(2) $\mathcal{X}$ の要素を 1 つ取り出し，それを $X_i$ とする．

(3) $\exists Y_j \in \mathcal{Y}$，$X_i \subseteq Y_j$ ならば，$A = \{k | X_k \subseteq Y_j\}$ とすると，

$$\mathcal{C} \leftarrow \mathcal{C} \cup \{\bigcup_{k\in A} X_k\},\ \mathcal{X} \leftarrow \mathcal{X} - \{X_k \in \mathcal{X} | X_k \subseteq Y_j\}$$

$\neg\exists Y_j \in \mathcal{Y}$，$X_i \subseteq Y_j$ ならば，

$$\mathcal{C} \leftarrow \mathcal{C} \cup \{X_i\},\ \mathcal{X} \leftarrow \mathcal{X} - \{X_i\}$$

(4) $\mathcal{X}$ が空になるまで (2)〜(3) の処理を繰り返す．

(5) $\mathcal{X}$ が空になったら，$\mathcal{C}$ を新たな分類結果とする．

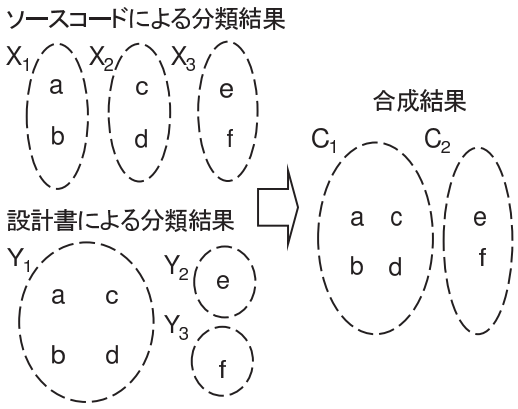



図 1 分類結果の合成
**Fig. 1** Combination of clustering results.

**図 1** に分類結果の合成の例を示す．図中の a〜f はソフトウェアプロダクトを表している．ソースコードによる分類結果における a，b を含むクラスタ $X_1$ と c，d を含むクラスタ $X_2$ は，設計書による分類では同一のクラスタ $Y_1$ に包含されているため結合されて，合成結果では a，b，c，d を含むクラスタ $C_1$ が生成されることが分かる．一方，ソースコードによる分類結果で e，f を含むクラスタ $X_3$ は，設計書による分類結果のどのクラスタにも包含されていないので，合成結果でもそのままクラスタ $C_2$ に引き継がれる．

## 4. 実験

提案手法により効果的にソフトウェアプロダクトを分類可能かどうかを検証するために，実際に企業で開発されたプロダクトを用いて提案手法による分類を行う実験を行い，人手で作成した分類の正解データと比較することにより評価を行う．

### 4.1 実験データ

実験では，2002 年以降に著者らの所属する部署で開発したソフトウェアプロダクトから異なる 5 つのカテゴリにわたる 27 個のプロダクトを対象として用いた．これらのプロダクトの開発形態は日本の企業でよく行われる担当部署が仕様をコントロールしつつもソフト開発ベンダに開発委託する形態であった．27 個のプロダクトのプロフィールを **表 1** に示す．

表 1 のカテゴリ A〜E について，以下に説明する．

- カテゴリ A
  デジタルコンテンツや無線 LAN 等の資源を一時的に利用可能とする権利情報を安全にネットワーク上で流通させる「権利流通プラットフォーム」に関するプロダクトカテゴリ

表 1 プロダクトのプロフィール
**Table 1** Poduct profiles.

| カテゴリ | プロダクト | 設計書 | ソースコード | | | | |
|---|---|---|---|---|---|---|---|
| | | | ライン数（Kline） | | | | |
| | | 単語数 | C/C++ | | | Java | 他の主な |
| | | | c* | cpp/cxx* | h* | java* | 言語 |
| A（権利流通） | A01 | 15,839 | 0 | 0 | 0 | 44.7 | なし |
| | A02 | 24,281 | 0 | 0 | 0 | 67.2 | なし |
| | A03 | 23,012 | 0 | 4.1 | 12.8 | 0 | なし |
| | A04 | 32,184 | 13.9 | 0 | 2.0 | 0 | なし |
| | A05 | 74,869 | 0 | 2.6 | 1.0 | 46.5 | なし |
| | A06 | 150,253 | 18.0 | 14.5 | 6.0 | 54.8 | なし |
| | A07 | 151,503 | 15.3 | 14.5 | 5.5 | 57.1 | なし |
| | A08 | 132,397 | 5.4 | 6.2 | 1.8 | 77.1 | なし |
| | A09 | 167,840 | 7.8 | 17.7 | 8.2 | 63.4 | なし |
| | A10 | 85,455 | 139.1 | 48.3 | 53.9 | 46.9 | PHP |
| B（SIP） | B01 | 30,588 | 0 | 439.9 | 93.3 | 0 | なし |
| | B02 | 29,252 | 0 | 0 | 0 | 20.6 | なし |
| | B03 | 46,986 | 21.2 | 0 | 2.1 | 0 | perl |
| | B04 | 85,575 | 13.0 | 0 | 14.9 | 0 | perl |
| | B05 | 129,449 | 48.0 | 0 | 5.2 | 0 | perl |
| C（グラフ検索） | C01 | 26711 | 0 | 0 | 0 | 12.2 | なし |
| | C02 | 42,251 | 0 | 0 | 0 | 27.3 | なし |
| | C03 | 40,942 | 0 | 0 | 0 | 39.7 | なし |
| | C04 | 40,018 | 0 | 0 | 0 | 50.2 | なし |
| | C05 | 25,652 | 0 | 0 | 0 | 10.4 | なし |
| | C06 | 10,690 | 0 | 0 | 0 | 11.1 | なし |
| | C07 | 34,950 | 0 | 0 | 0 | 23.0 | なし |
| D（認証局） | D01 | 802,689 | 1,528.5 | 0 | 631.2 | 0 | なし |
| | D02 | 793,281 | 1,221.6 | 10.4 | 431.4 | 0 | なし |
| | D03 | 907,987 | 1,456.7 | 0 | 458.8 | 0 | なし |
| E（研究管理） | E01 | 33,157 | 0 | 0 | 0 | 31.0 | なし |
| | E02 | 14,627 | 113.2 | 0 | 26.7 | 151.3 | なし |

* ファイルの拡張子を表す．

- カテゴリ B
  SIP（Session Initiation Protocol）と呼ばれる通信プロトコルを使用した VPN トンネルの接続，切断操作を行うシステム等の「SIP」に関するプロダクトカテゴリ
- カテゴリ C
  RDF（Resource Description Framework）を使用したグラフ構造を持つデータを管理し，検索を可能とする「グラフ検索」に関するプロダクトカテゴリ
- カテゴリ D
  暗号等で必要となる公開鍵証明書を発行する「認証局（CA）プラットフォーム」に関するプロダクトカテゴリ
- カテゴリ E
  研究所における研究テーマや研究成果に関する情報を管理する「研究管理」に関するプロダクトカテゴリ

本実験では設計書による分類では機能設計レベルのもの，ソースコードによる分類では，プログラミング言語が C/C++と Java であるソースファイルを対象とした．著者らの所属する部署では，準拠すべき開発標準が規定されている．そのため，開発プロジェクトが異なっても，設計書の記述粒度にぶれが少ないことが期待される．今回対象とした 27 個のプロダクトはすべてこの開発標準に準拠している．以下に機能設計書の構成の一例を示す．

- 機能概要
- 機能詳細
- コマンドインタフェース仕様
- 画面仕様
- ファイル仕様
- メッセージ仕様
- 関数仕様

表 1 には各プロダクトのデータの規模を示す値として設計書については，設計書から抽出したテキストに含まれる単語数，またソースコードについては対象としたプログラ

ミング言語ごとにファイルの拡張子ごとのライン数を示す．

### 4.2 提案手法の評価

本研究では提案手法によるプロダクトの分類の正確性を評価する．ここで，正確性とは人手で作成した正解の分類にどれだけ近い分類ができるかということである．提案手法はソースコードによる分類結果をベースとして，さらに設計書による分類結果も考慮することにより，ソースコード単独による分類よりもより正確な分類を目指したものである．よって，提案手法の評価は，4.1 節に示したプロダクト群に対する提案手法による分類結果をソースコード単独による分類結果と比較することにより行う．

分類結果を比較するための評価指標には Purity $P$，Inverse Purity $IP$，$F_{P-IP}$ を用いる．クラスタリングによる分類結果を $\mathcal{C} = \{C_1, C_2, \cdots C_M\}$，シリーズごとに分けた分類の正解データを $\mathcal{A} = \{A_1, A_2, \cdots, A_N\}$，プロダクトの総数を $n_p$ とすると，Purity $P$，Inverse Purity $IP$，$F_{P-IP}$ は以下のように計算される [11]．

$$P = \sum_{i=1}^{M} \frac{|C_i|}{n_p} \max_j P_r(C_i, A_j) \tag{9}$$

$$IP = \sum_{j=1}^{N} \frac{|A_j|}{n_p} \max_i R_e(C_i, A_j) \tag{10}$$

$$F_{P-IP} = \frac{1}{\alpha \cdot \frac{1}{P} + (1-\alpha) \cdot \frac{1}{IP}} \quad (0 \leq \alpha \leq 1) \tag{11}$$

ただし，ここで

$$P_r(C_i, A_j) = \frac{|C_i \cap A_j|}{|C_i|} \tag{12}$$

$$R_e(C_i, A_j) = \frac{|C_i \cap A_j|}{|A_j|} \tag{13}$$

とする．Purity は分類結果 $\mathcal{C}$ に含まれる各クラスタがいかに正解データ $\mathcal{A}$ に含まれる同じシリーズのプロダクトで満たされているかを示す値で，Inverse Purity は正解データ $\mathcal{A}$ に含まれる同じシリーズのプロダクトがいかに分類結果 $\mathcal{C}$ の同じクラスタに含まれるかを示す値である．Purity，Inverse Purity は情報検索分野でよく用いられる Precision と Recall の関係と同様にトレードオフの関係にある評価指標であり，総合的な評価指標として F-measure に相当する $F_{P-IP}$ を用いる．Purity，Inverse Purity は 0 以上 1 以下の範囲の値をとり，両方の値が同時に 1 に近い値を持つほど，すなわち，$F_{P-IP}$ が 1 に近い値をとればとるほど，より分類の正確性が高いという意味合いを持つ．したがって，これらの評価指標を用いた評価方法は分類の正確性を評価する評価方法としての妥当性がある．$F_{P-IP}$ の計算式に含まれる $\alpha$ の値は Purity と Inverse Purity に対する重み付けのための値である．本研究におけるプロダクトの分類はスコーピングを目的としており，できるだけ共通性を持つ可能性が高い同じシリーズのプロダクトが同じクラスタに含まれる方が好ましいと考え，Purity よりも Inverse Purity を重視し，$\alpha$ の値は 0.2 とした．

**表 2** には，評価指標を計算する際に用いる分類の正解データを示す．表中のシリーズの列に示すプロダクトのグループが本実験で正解とする分類である．本表は著者 3 名がプロダクトの開発に携わった関係者からのヒアリングや設計書の内容を比較参照することにより作成した．

### 4.3 実験環境

設計書の分析において単語ベクトルを生成する際に，設計書からのテキストの抽出には xdoc2txt [12] を，形態素解析器には MeCab [13] を用いた．また，階層的クラスタリングには統計解析ソフトウェアである R [14] を用いた．

### 4.4 実験手順

ソースコードによる分類を行う際の結合するクラスタ間の距離の上限値による分類の閾値を $h_s$，設計書による分類を行う際の結合するクラスタ間の距離の上限値による分類の閾値を $h_d$ とする．ソースコードによる分類結果は閾値 $h_s$ に，設計書による分類結果は閾値 $h_d$ に，提案手法による分類結果は閾値 $h_s$ と $h_d$ の両方に依存する．提案手法を評価する場合，これらの閾値を決めないとならないが，後に 4.5 節で示すようにソースコードによる分類結果，設計書による分類結果は閾値によって異なる．そこで，本実験では閾値 $h_s$ と $h_d$ を各々 3 点，つまり組合せで 9 点をとり，提案手法による分類結果とソースコード単独による分類結果を比較する．この時の閾値のとり方は，0（各プロダクトがバラバラな状態）から，すべてのプロダクトが 1 つのクラスタになる閾値までの間でほぼ均等になるようにとることとする．

以下にソースコード，設計書および提案手法による分類の手順について示す．

- ソースコードによる分類
  実験対象のソフトウェアプロダクトの中から 2 つを選ぶすべての組合せについて以下を実施する．
  (1) CCFinderX を用いてコードクローンを検出する．
  (2) コードクローン検出結果をプレーンテキストで出力する．
  (3) コードクローン検出結果からプロダクトをまたがるクローンペアを取り出し，それを基にソフトウェアプロダクト間の距離（非類似度）を計算する．
  (4) 単連結法による階層的クラスタリング法により樹状図を生成し，閾値 $h_s$ により分類する．
- 設計書による分類
  設計書による分類は以下の手順で行う．
  (1) ソフトウェアプロダクトに含まれる設計書から 3.5.1 項で示した方法に基づき単語ベクトルを生成

表 2 人手による分類
Table 2 Clustering by humans.

| カテゴリ | シリーズ | 説明 |
|---|---|---|
| A（権利流通） | A01, A02 | A01 は，最初に発案されたコンセプトの基本部分を実装したもので，A02 は，それを他のサービスに柔軟に適用可能なようにプラットフォーム化したもの．Jini という技術を使っている点で独自性がある． |
| | A03 | 出張時に出先のネットワーク環境でファイルサーバ等のコンピューティング資源を利用する場合のゲストアカウントに対する権利の管理を行うシステムであるが，IC カードの利用を前提としない人を介した対面認証である点でカテゴリ A の他のプロダクトと大きく異なる． |
| | A04～A07<br>A09 | 権利情報や属性情報の格納に IC カードを用いた権利流通プラットフォームに関するものであり，カテゴリ A の中で中核的な位置づけになるプロダクト群である．A04, A05 はデジタルコンテンツの権利流通のためのプロダクトで，A04 は IC カード上のアプリケーションプログラム，A05 は端末，サーバ部分である．特に A06，A07，A09 は A04，A05 から派生したもので無線 LAN 接続サービスの権利流通に関するプロダクトのバージョン違いであり，非常に関連性が高いものである． |
| | A08 | A04，A05 と同様に IC カードを用いた権利管理ではあるが，プライバシを考慮した設計である点やローカルだけでなくネットワーク越しのリモート資源の利用も可能とするシステムである点で独立性が高い． |
| | A10 | 端末に認証のためのクレデンシャル情報を持たせる機能に特化しており独立性が高い． |
| B（SIP） | B01 | セキュアメッセージ通信を行うためのプロダクトで SIP サービスにおける認証機構に SAML を適用したシステム |
| | B02 | SIP を用いて双方向で属性情報を流通するシステム |
| | B03～B05 | SIP プロトコルを用いて VPN トンネルの接続，切断操作を行うシステムが徐々に拡張されたもので，非常に関連性が高い． |
| C（グラフ検索） | C01～C04 | グラフ検索の検索エンジン部分 |
| | C05 | 検索エンジンを利用したグラフ検索アプリケーションの 1 つ |
| | C06, C07 | 検索エンジンを利用したグラフ検索アプリケーションの 1 つ |
| D（認証局） | D01～D03 | 電子認証のための認証局用プログラムのバージョン違い |
| E（研究管理） | E01 | 研究テーマの計画情報を管理するシステム |
| | E02 | 研究成果を管理するシステム |

する．ただし，単語ベクトルの次元数が約 12,000 と高次元になったため，単語ごとにベクトルの成分の値の最大値を求め，その値が上位 200 に入る単語の成分以外は除去して，200 次元のベクトルにする．

(2) (1) で生成した単語ベクトル間のコサイン距離を計算する．

(3) 単連結法による階層的クラスタリング法により樹状図を生成し，閾値 $h_d$ により分類する．

- 提案手法による分類
  ソースコードによる分類結果と設計書による分類結果から 3.6 節で示したアルゴリズムにより分類結果を生成する．

### 4.5 実験結果

まず，ソースコードおよび設計書による階層的クラスタリング結果の樹状図を**図 2**，**図 3** に示す．ここで縦軸はクラスタどうしを結合した距離を表しており，下側で結合しているものほど距離が近い，すなわち類似度が大きいことを示している．

次に，ソースコードによる分類の評価指標値の閾値 $h_s$ に対する依存性，設計書による分類の評価指標値の閾値 $h_d$ に対する依存性を示すグラフを**図 4**，**図 5** に示す．

最後に，**表 3** に $F_{P-IP}$ を提案手法による分類とソースコードによる分類とで比較した結果を示す．ここで閾値 $h_s$，$h_d$ は図 2，図 3 より，4.4 節で述べた考え方に従って，各々，0.25，0.50，0.75 の 3 点，0.15，0.30，0.45 の 3 点をとった．

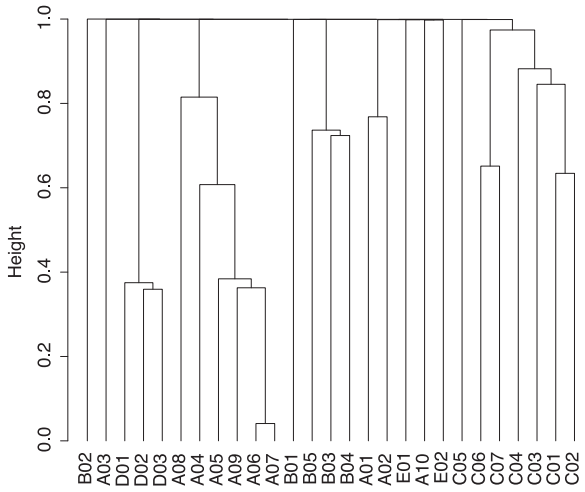



図 2 樹状図（ソースコード）
**Fig. 2** Dendrogram (source code).

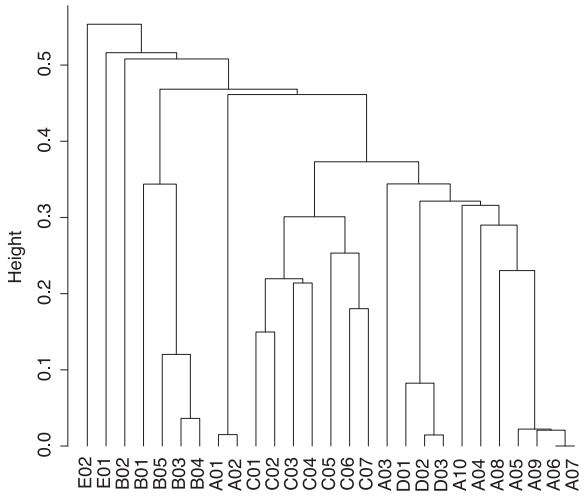



図 3 樹状図（設計書）
**Fig. 3** Dendrogram (specification document).

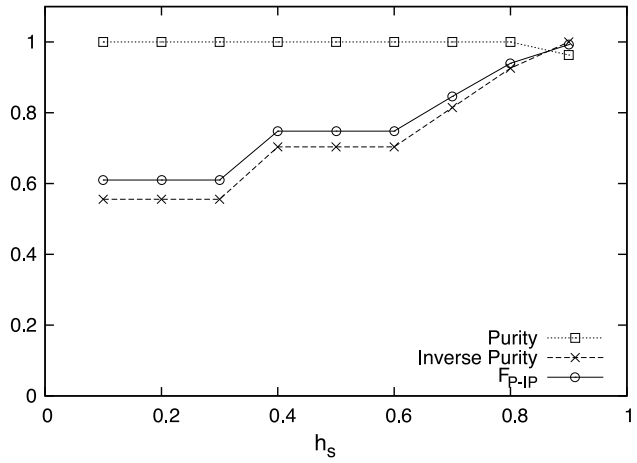



図 4 評価指標値の閾値依存性（ソースコード）
**Fig. 4** Impact of tuning threshold value for clustering performance (source code).

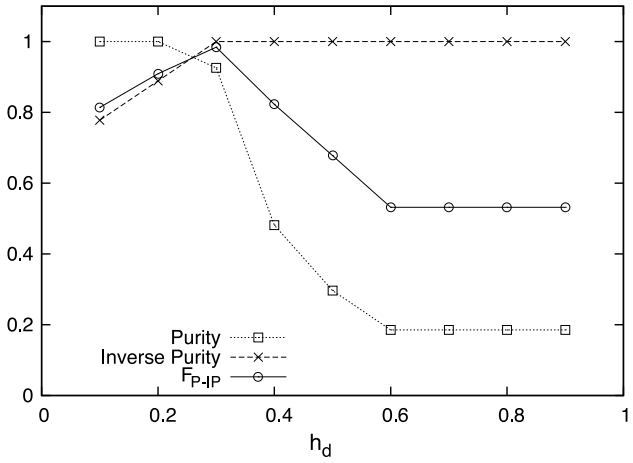



図 5 評価指標値の閾値依存性（設計書）
**Fig. 5** Impact of tuning threshold value for clustering performance (specification document).

表 3 $F_{P-IP}$ の比較
**Table 3** Comparison of $F_{P-IP}$.

| $h_s$ | $h_d$ | ソースコード | 提案手法 |
|---|---|---|---|
| 0.25 | 0.15 | 0.61 | 0.88 |
| 0.50 | 0.15 | 0.75 | 0.88 |
| 0.75 | 0.15 | 0.91 | 0.94 |
| 0.25 | 0.30 | 0.61 | 0.98 |
| 0.50 | 0.30 | 0.75 | 0.98 |
| 0.75 | 0.30 | 0.91 | 0.98 |
| 0.25 | 0.45 | 0.61 | 0.82 |
| 0.50 | 0.45 | 0.75 | 0.82 |
| 0.75 | 0.45 | 0.91 | 0.82 |
| average* | | 0.76 | 0.90 |

* すべての閾値の組合せについての平均値を表す

## 5. 考察

### 5.1 単独手法の閾値依存性

ここでは，ソースコード単独による分類手法および設計書単独による分類手法の閾値依存性について述べる．まず，ソースコード単独による分類手法の場合について述べる．図 4 よりソースコード単独による分類手法の場合，Purity の値が閾値 $h_s$ にあまり依存せず，ほぼつねに 1 になっていることが分かる．Purity が 1 であるということは，異なるシリーズのプロダクトを一緒のクラスタにしない，すなわち，同じクラスタ内にある類似性が高いプロダクトは同じシリーズに入るということを意味しており，3.2 節で示した提案手法の基となる仮説の 1 つである仮説 1 を支持しているものといえる．一方，Inverse Purity の方は，階層的クラスタリングという手法の性格上，閾値 $h_s$ の増加につれて単調的に増加するため，$F_{P-IP}$ の値も同様に単調的に増加しているといえる．

次に設計書単独による分類手法について述べる．図 5 よ

り，設計書単独による分類手法の場合，Purity の値は閾値 $h_d$ を増加させた場合，0.3 のあたりから急に小さくなっていくことが分かる．また，その影響で，$F_{P-IP}$ の値も同様の傾向を示すことが分かる．このことは，図 2，図 3 を見ても分かるように，設計書による分類では，ソースコードに比べ，低い閾値でクラスタどうしの結合が進んでいくことを意味するものといえる．

### 5.2 提案手法の評価

ここでは，提案手法とソースコード単独による分類手法に対する分類精度の比較について述べる．

まず，表 3 より，提案手法の $F_{P-IP}$ の値は，閾値 $h_s$ と $h_d$ の組合せの総数 9 個の中で 8 個の組合せにおいて，ソースコード単独による分類手法の値を上回っていることが分かる．また，提案手法は閾値の組合せについての平均値においてソースコードによる単独手法を上回っていることが分かる．このことは，今回の実験データにおいて閾値の組合せを適当に 1 つ選んだ場合，提案手法がソースコード単独による分類手法よりも高い確率で正確にプロダクトを分類可能であることを示したものであるといえる．ここで，$F_{P-IP}$ の値について提案手法がソースコード単独による分類手法を上回っている 8 つの閾値の組合せの中の 1 つである閾値 $h_s = 0.75$ と $h_d = 0.15$ の場合を考える．A01，A02 というプロダクトに注目すると，表 2 により，この 2 つのプロダクトは 1 つのシリーズを形成していることが分かる．図 2，図 3 を見ると，この閾値の組合せにおいて A01，A02 はソースコードによる分類では，各々が 1 つのクラスタとなっているのに対して，設計書による分類では両プロダクトのみを含む 1 つのクラスタを形成している．よって，提案手法による分類結果は，3.6 節で示した分類結果の合成のアルゴリズムより容易に分かるように，設計書による分類結果と同じく両プロダクトのみを含む 1 つのクラスタを形成する．この現象は同じシリーズである A01，A02 のプロダクトが，ソースコードでは類似性が低いが，設計書ではそれに比べ類似性が高くなっていることを示しており，3.2 節で示した提案手法の基となる仮説の 1 つである仮説 2 を支持するものであると考える．提案手法は仮説 2 に基づいて設計書による分類の情報をアルゴリズムに取り込むことにより，ソースコード単独ではできなかった A01，A02 を含むシリーズの抽出を可能にしたといえる．

一方，表 3 より，閾値 $h_s$ と $h_d$ の組合せの中で $h_s = 0.75$ と $h_d = 0.45$ の場合のみ，提案手法はソースコード単独による分類手法より $F_{P-IP}$ の値が下回っている．これは，提案手法が設計書による分類の情報を取り込むことで，関係性が強くないプロダクトどうしを結び付けてしまうという単語の分布による類似性の負の側面が現れ，異なるシリーズのプロダクトを同じクラスタに取り込んでしまう作用が働いたために生じたものではないかと考えている．このような負の側面がどの程度影響するかについて，今後さらに多様なプロダクト群に本提案手法を適用して調べる必要がある．

また，本稿では提案手法を 3.1.1 項で定義した機能的な観点でソフトウェアプロダクトを分類できるかで評価しているため，この機能の定義が異なると，評価結果が変化する可能性がある．よって，本稿とは異なる機能の定義で提案手法を評価する場合は，この点に留意し，分類の正解データを新たな機能の定義に合わせたものにする必要がある．

## 6. 関連研究

ソフトウェアの分類はソフトウェアの類似性に基づいて行われる．ソフトウェアの類似性分析に関連する研究には様々な研究がある．

まず，ソースコードを利用して類似性を分析した研究として以下のようなものがある．川口らは MUDABlue という自動分類システムを提案している [15]．MUDABlue ではソースコードに含まれる関数名，変数名，型名等の識別子を用いて分類を実現している．山本らはコードクローンと DIFF を組み合わせて用いることで，行単位での類似性を求め，UNIX 系 OS のバージョンの派生関係調査に適用している [7]．吉村らはレガシーシステムにソフトウェア・プロダクトラインを導入するための共通性分析手法として，異なるソフトウェア間でコードクローンによる解析を行っており，評価のためのメトリックスとして製品間クローンカバレッジを定義している [16]．吉村らは，共通性分析に関連してソフトウェアの類似性について論じているが，吉村らの研究がプロダクトライン開発におけるコア資産の候補を抽出することを目的としているのに対し，本研究は著者らが定義した共通性分析を行う対象を絞り込むスコーピングを目的としており，目的が異なる．

次に，設計書等の開発文書を利用してソフトウェアの類似性を分析した研究として以下のようなものがある．Maarek らは開発文書に階層的クラスタリング法を適用し，GURU というソフトウェアライブラリの検索システムに応用している [17]．また，Ugurel らは SVM（Support Vector Machine）を用いた教師あり学習によりソフトウェアをトピックに分類している [18]．

このように，すでにソースコードや設計書等の開発文書を用いてソフトウェアの類似性分析を行った研究は報告されているが，スコーピングを目的としている点や同一のプロダクト群に対してソースコードを用いる手法と設計書を用いる手法を組み合わせた手法を提案している点で本研究は既存研究とは異なる．提案手法はソースコードによる類似性をベースにしながらも，さらに設計書による類似性を取り込むことで，既存手法よりもソフトウェア資産を無駄なく活用し，より正確な分類を行うことを可能にする．

## 7. おわりに

本研究では企業のソフトウェアプロダクトに対して共通性分析の対象となるプロダクト群を抽出するスコーピングを支援するために，マイニング技術によりソフトウェアを派生関係がある類似グループに分類する手法としてソースコードによる分類手法と設計書による分類手法の両者を組み合わせた手法を提案した．また，提案手法を評価するために，実際の企業のソフトウェアプロダクトを用いた実験を行った．提案手法による分類結果は閾値のとり方に依存するため，閾値を変化させて評価を行った結果，提案手法はソースコード単独の既存の分類手法に比べ，閾値の全パターンの中の約 9 割において，より正確に類似性の高いソフトウェアプロダクト群の抽出が可能であることが分かった．このことは閾値に対する知見がなく閾値を適当に選んだ場合において，提案手法は高い確率でソースコード単独の分類手法より正確に分類可能であり，スコーピング支援への有用性が高いことを意味する．

一方，本研究では提案手法の分類を行う際の閾値をどのように決定するかについては言及しておらず，汎用的に高い分類の性能が得られる閾値の決定方法について，今後，検討していく．そのために，より多様で規模の大きいプロダクト群へ本手法を適用する予定である．

### 参考文献


[1] 岸　知二：プロダクトライン開発の全体像と要求工学，情報処理，Vol.50, No.4, pp.268–273 (2009).

[2] Clements, P. and Northrop, L.: *Software Product Lines: Practices and Patterns*, Addison-Wesley (2001).

[3] Schmid, K.: Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines, Ph.D. Theses in Experimental Software Engineering, Fraunhofer IRB Verlag (2003).

[4] 岸本康成，坂本　啓，市川裕介，佐藤宏之，小林　透：スコーピング支援のためのソフトウェア類似性分析手法の提案，ソフトウェアエンジニアリング最前線 2010（ソフトウェアエンジニアリングシンポジウム 2010 予稿集），pp.51–56 (2010).

[5] 坂本　啓，岸本康成，佐藤宏之，小林　透：コードクローンを用いたソフトウェア間の類似度の分析，電子情報通信学会技術研究報告，Vol.109, No.456, pp.73–78 (2010).

[6] 岸本康成，坂本　啓，佐藤宏之，小林　透：テキストマイニング技術を用いたソフトウェアの類似性分析，電子情報通信学会技術研究報告，Vol.109, No.456, pp.79–84 (2010).

[7] 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).

[8] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌 D，Vol.J91-D, No.6, pp.1465–1481 (2008).

[9] 神谷年洋：CCFinder ホームページ，CCFinder (online)，入手先 ⟨http://www.ccfinder.net/ccfinderx-j.html⟩（参照 2011-03-23）.

[10] 神嶌敏弘：データマイニング分野のクラスタリング手法 (1)—クラスタリングを使ってみよう！，人工知能学会誌，Vol.18, No.1, pp.59–65 (2003).

[11] Artiles, J., Gonzalo, J. and Sekine, S.: The SemEval-2007 WePS Evaluation: Establishing a benchmark for the Web People Search Task, *Proc. 4th International Workshop on Semantic Evaluations* (*SemEval 2007*), pp.64–69 (2007).

[12] hishida: xdoc2txt，xdoc2txt (オンライン)，入手先 ⟨http://www31.ocn.ne.jp/~h_ishida/xdoc2txt.html⟩（参照 2011-03-23）.

[13] 工藤　拓：MeCab: Yet Another Part-of-Speech and Morphological Analyzer，MeCab (オンライン)，入手先 ⟨http://mecab.sourceforge.net/⟩（参照 2011-03-23）.

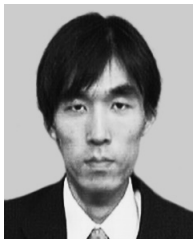
[14] R Project: The R Project for Statistical Computing, R Project (online), available from ⟨http://www.r-project.org/⟩ (accessed 2011-03-23).

[15] 川口真司，パンカジガーグ，松下　誠，井上克郎：MUDABlue：ソフトウェアリポジトリ自動分類システム，電子情報通信学会論文誌 D-I，Vol.J88-D-I, No.8, pp.1217–1225 (2005).

[16] 吉村健太郎，ガネサン・ダルマリンガム，ムーティック・ディルク：プロダクトライン導入に向けたレガシーソフトウェアの共通性・可変性分析法，情報処理学会論文誌，Vol.48, No.8, pp.2482–2491 (2007).

[17] Maarek, Y.S., Berry, D.M. and Kaiser, G.E.: An Information Retrieval Approach for Automatically Constructing Software Libraries, *IEEE Trans. Softw. Eng.*, Vol.17, No.8, pp.800–813 (1991).

[18] Ugurel, S., Krovetz, R., Giles, C.L., Pennock, D.M., Glover, E.J. and Zha, H.: What's the Code? Automatic Classification of Source Code Archives, *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD 2002*), pp.632–638 (2002).


**岸本 康成**　（正会員）

昭和 42 年生．平成 3 年九州大学大学院総合理工学研究科修士課程修了．同年日本電信電話株式会社入社．以来，ディレクトリ・システム，課金システム，データマイニング等に関する研究開発に従事．現在，NTT 情報流通プラットフォーム研究所研究主任．



**坂本 啓**　（正会員）

昭和 42 年生．平成 5 年北海道大学大学院工学研究科情報工学専攻修士課程修了．同年 NTT 入社．ソフトウェア生産技術，データマイニング等の研究開発に従事．現在，NTT 情報流通プラットフォーム研究所主任研究員．

小林 透 （正会員）

昭和 60 年東北大学工学部精密機械工学科卒業．昭和 62 年同大学大学院工学研究科修士課程修了．同年 NTT 入社．以来，ソフトウェア生産技術，ユビキタスコンピューティング，情報セキュリティ，データマイニング等の研究開発に従事．現在，NTT サイバーソリューション研究所主幹研究員．電子情報通信学会，IEEE 各会員，博士（工学）．