ソフトウェア保守のためのデータアクセス可視化技術の検討

矢野 啓介[1,a)]　松尾 昭彦[1,b)]

概要：ソフトウェア保守のために，ソフトウェアを構成するソースファイルの集合に対し依存関係に基づいてグラフクラスタリングを適用して機能を発見および可視化し，現状の分析に用いることが研究・実用化されている．従来研究ではプログラム間の依存関係を用いており，プログラムが操作・参照しているデータとの関係は用いていない．本研究ではプログラムが扱うデータとの依存関係を含めたクラスタリングを行い，プログラムとデータからなるクラスタを発見し可視化する．その上で，各データにアクセスするプログラムの所属クラスタを考慮することでデータの性質を自動的に判別する方法を提案する．これによりデータの使われ方の実態を明らかにし，当初の設計と比較することにより実態と設計との乖離を発見できる．ここで得られた情報はソフトウェアの保守・修正や再設計に役立てることが期待できる．

キーワード：ソフトウェア保守，プログラム理解，ソフトウェアクラスタリング，データベース

Data Access Visualization Technique for Software Maintenance

1. はじめに

企業等の組織で使用される情報システムは長年にわたって何人もの担当者の手を経て修正・改良を続けられることが少なくない．修正を繰り返してきた複雑なソフトウェアを保守していくあるいは進化させるには，現状のシステムがどのような状態にあるかを把握することが必要である．しかし，年月を経るうちに，設計ドキュメントが適切に更新されずに現状に合わなくなっていたり，開発者が異動していて設計意図が確認できなかったりすることがある．

ソフトウェアの現状の構造を理解するために，ソフトウェアを構成するプログラム間の依存関係に対してグラフクラスタリング技術を適用するソフトウェアクラスタリングが使用されている [3]．また，クラスタリングの結果に基づいて，ソフトウェアシステム全体の構成を視覚的に表現し理解を助ける技術も提案されている [4]．

こうした従来技術では，クラスタリングの対象としてはソースファイル，あるいはオブジェクト指向言語におけるクラス，また言語によっては関数など，単位をどのようにとるかという問題はあるものの，いずれにしてもプログラムを扱っている．

[1] 株式会社富士通研究所
[a)] yano@jp.fujitsu.com
[b)] a_matsuo@jp.fujitsu.com

しかし実際のシステムではプログラムだけでなくデータも重要な構成要素である．本稿において「データ」とは，システムの実行に伴ってプログラムから読み書きされるデータベースやファイルなどの総称とする．一口にデータといっても，ひとつのシステムの中には様々な性質のデータが存在する．長期にわたって様々な業務から参照・利用される製品情報や取引先情報のようなマスターデータもあれば，特定の業務の進行に応じて随時更新されていく注文情報のようなトランザクションデータもある [14]．また，プログラムの処理中に一時的に作成される作業ファイルもあれば，プログラム間で受け渡されるファイルもある．システムの設定情報を定義した大局的に参照されるファイルもある．そうした様々なデータの使われ方を知ることは，システムの維持や修正の上で重要である．

これらのデータは長年にわたるシステムの拡張や修正の間に，元々の設計意図から乖離した使い方がなされることがある．例えば，ある特定のプログラムの作業ファイルであったものを別のプログラムからも参照して利用したり，マスターデータに付随的な情報を持たせてトランザクションデータのように利用したりされることもある．こうした変更がなされていくと，データ構造やデータ処理プログラムの修正に際して思わぬ箇所へ影響が及んでしまい，修正の困難さが増し，障害の原因となり得る．したがってデー

タが実際にどのように使われているかを明らかにすることはシステムの保守や修正のために有用であると考えられる．とりわけ，当初の設計と乖離した使われ方，あるいは開発者の想定と異なる使われ方を発見することは，将来の障害を防いだりシステムをより良い形に再設計するために有効であると期待される．

本研究では，著者らの従来研究を拡張する形でプログラムとともにデータをクラスタリング対象とし，またクラスタリングの結果に基づいてデータの性質を判定する．これにより，データの意図しない使われ方を発見することを可能にする．

本稿は以下の構成をとる．第 2 節は関連研究として，本研究の前提となるいくつかの研究を挙げる．続いて第 3 節では提案手法として，データとプログラムの間の依存関係を含んだクラスタリングの結果を利用し，各データについてその特徴あるいは性質を表す指標を計算する方法を示す．また，得られた指標値を利用したソフトウェア可視化の手法を示す．第 4 節は本技術を適用した事例で有効性を示す．第 5 節はまとめである．

2. 関連研究

ソフトウェアシステムを構成するプログラムをクラスタに分割するソフトウェアクラスタリングは多くの研究がなされている．どのような情報に基づいてクラスタリングするかについて複数のアプローチが提案されている．プログラム間の依存関係や構造に基づいて行うものや，プログラム中の識別子やコメント等から得られる語彙に基づくものがある．前者には例えば Mancoridis らによる Bunch [7] や，Tzerpos と Holt による ACDC [11] があり，また静的解析と動的解析を組み合わせる研究も行われている [9]．後者には例えば Kuhn ら [5] による潜在意味解析を応用した研究が知られる．

本研究はそうしたソフトウェアクラスタリング技法のうち，静的な依存関係を入力に用いてグラフクラスタリングを適用する著者らの従来研究である SArF [3] を用い，それにデータの情報を付加することで拡張し利用する．ただし後に述べる本研究の提案手法はクラスタリングの方式として必ずしもこのアルゴリズムを前提とする必要はなく，システム上の業務あるいは機能をクラスタとして発見するものであれば異なるクラスタリング方式を用いてもよい．

プログラムとデータを含めた可視化の研究としては，Van Geet と Demeyer [12] によるものがある．これは，レガシーシステムのサービス指向アーキテクチャ (SOA) への移行のためにシステムの構造の理解に役立てることを目的とし，COBOL で書かれたシステムを対象として，プログラムとデータの依存関係を可視化している．ここではデータとプログラムの関係に着目して，視覚的に発見できるパターンとして 3 種類，すなわち，“Common Data”, “Internal Data Facade”, “External Data Facade” を挙げている．プログラムとデータの依存関係からパターンを見出してシステムの理解に役立てるという点では本稿の目的に近い．ただしデータ自体がどのような性質を持っているかを発見することまではなされていない．また自動的なクラスタリング技術の利用はしていない．

3. 提案手法

本節では提案手法を説明する．手法は大きく 3 つの段階に分けられる．最初はクラスタリングである．この際に，プログラム間の依存関係だけでなく，プログラムとデータの依存関係を含めて行う．次に，クラスタリングの結果に基づいて，各データについてそのデータの使われ方を表す指標を計算する．最後に，計算された指標をソフトウェアの可視化に適用する．以下，この 3 つの段階それぞれについて，詳細を説明する．

3.1 データつきのクラスタリング

最初に，データアクセスを含んだクラスタリングについて説明する．従来の依存関係に基づいたソフトウェアクラスタリングにおいては，プログラム（ソースファイルやオブジェクト指向言語におけるクラス等の何らかの単位を本稿ではプログラムと総称する）を頂点とし，プログラム間の依存関係を辺とする有向グラフを入力としてグラフクラスタリング技法を用いる．依存関係には，例えば，関数やメソッドの呼び出し関係や，変数によるクラスの参照，継承関係によるクラスの参照といったものが用いられる．

今回，データアクセスを含めるにあたって，プログラムと同等の単位，すなわちグラフの頂点として個々のデータを扱う．ここで個々のデータとは，関係データベースにおけるテーブルや，ファイルシステムにおけるファイル等，通常ハードディスク等の補助記憶装置に存在しプログラムから読み書きされる単位を指す．テーブル名やファイル名を，当該データを表す識別子として用いる．データに関する依存関係としては，プログラムからデータへの読み書きを対象とする．したがって，例えば，あるプログラムがデータベースのテーブルに対してレコードを書き込んだり，あるプログラムがファイルを読み出したりすることを，当該プログラムから当該データへの依存関係とみなす．こうした依存関係はプログラム間の依存関係とともにクラスタリングへの入力として用いる．

クラスタリングそのものは，従来のソフトウェアクラスタリングと同じ手法を用いることができる．本研究においては，SArF [3] を用いている．

クラスタリングの結果として得られたクラスタには，プログラムとデータが混在して含まれる．ある機能を実現しているプログラムとデータの集合がひとつのクラスタをなすことが想定される．例えば，請求情報のデータは，それ

を読み書きする請求処理のプログラム群と同じクラスタに含まれる．

3.2 指標計算

次に，クラスタリング結果を用いてデータの使われ方を表す指標を計算する方式を説明する．データとそれにアクセスするプログラムがどのクラスタに所属するかの情報を用いて，各データについて以下に述べる複数の指標を計算する．

指標の種類として基礎指標と応用指標の 2 種類を定義する．基礎指標は，応用指標の計算のために用いられるものである．データへの読み書きそれぞれについて，当該データと同じクラスタの中にあるプログラムから何件あるか，外部のプログラムから何件あるかをそれぞれ勘定する．具体的には，RI, RO, WI, WO という 4 つの基礎指標を設定する．これらの指標名はデータの読み書き Read/Write とクラスタ内外を表す In/Out の頭文字からなる．すなわち，RI は当該データと同じクラスタ内部のプログラムからの読み出しを，また，WO は当該データのクラスタの外部からの書き込みを意味する．これら 4 つの基礎指標を用いた計算によって，データの性質を表す応用指標を計算する．

応用指標はデータがどのように使われているかという性質を表すものである．応用指標はさらに，分類指標と逸脱指標という 2 つのタイプに分けられる．分類指標とは，データアクセスがある典型的なパターンに則っているかどうかを示すものである．一方，逸脱指標とは，典型的なアクセスパターンに則っているようでありながら一部にそのパターンから外れているアクセスが存在する場合に値が高くなることを意図した指標である．逸脱指標は，データが元々の設計から外れた使用法がされていないかを検出する意図で設けている．

本稿において分類指標としては，以下のものを設定する: (1) グローバルデータパターン指標，(2) 内部専用パターン指標，(3) マスターデータパターン指標，(4) ログパターン指標．また，逸脱指標としては，以下のものを設定する: (5) 内部専用パターン逸脱指標，(6) マスターデータパターン逸脱指標．

これらの各指標の意味とその計算方法を以下に述べる．

(1) グローバルデータパターン指標は，プログラムにおけるグローバル変数と同じように，システム全体から読み書きされるデータアクセスパターンを表す指標である．グローバル変数がプログラムのメンテナンス性の観点から問題になるのと同じように，この指標値の高いデータはシステム全体に与える影響が大きく，変更には慎重な取り扱いを要する．この指標の計算方法は次の式として定義する．

$$I_1 = RO \times WO$$

すなわち，クラスタ外部からの読み書きが多いほど高い値として計算される．

(2) 内部専用パターン指標は，そのデータがある特定の機能に閉じて使われていることを示す指標である．この指標は内部専用か否かを表す二値をとる．1 が内部専用，0 が内部専用でないことを意味する．データが内部専用であるときは，システムの他の部分からの依存が存在しないので，当該データの形式や意味の変更が他の部分に直接影響を及ぼさず，保守性が良好であるといえる．この指標の計算方法は次の式として定義する．

$$I_2 = \begin{cases} 1 & (WO = 0 \text{ かつ } RO = 0) \\ 0 & (\text{それ以外}) \end{cases}$$

すなわち，同じクラスタ外のプログラムからの読み書きが存在しないときに 1 となる．

(3) マスターデータパターン指標は，当該データがマスターデータのように使われていると想定されるアクセスパターンを表す指標である．マスターデータとは典型的には顧客や製品等の情報を格納したものであり，随時業務ロジックから参照して使われる．Brunner ら [1] は，マスターデータは中核的な業務の実体であり，多くの業務プロセスやシステムにわたって企業が繰り返し使用するものであるとしている．また，マスターデータの特徴を Cleven と Wortmann [2] は，他のデータとは独立して存在すること，低頻度な変更，一定した分量と説明している．

こうした性質を踏まえると，アクセスのされ方から見たマスターデータの特徴は，様々な機能から読み出されることである．一方，書き込むものは少数に限られる．一般的にマスターデータのメンテナンス用のプログラムが存在する．マスターメンテナンスのプログラムは他のプログラムよりもマスターデータとの関係が強いため，依存関係に基づくクラスタリングによってマスターデータと同じクラスタに所属することが期待される．以上を考慮して，この指標の計算方法は次の式として定義する．

$$I_3 = \begin{cases} RO & (WO = 0 \text{ かつ } RO > 0) \\ RI & (WO = 0 \text{ かつ } RO = 0 \text{ かつ} \\ & \quad RI > 0 \text{ かつ } WI = 0) \\ 0 & (\text{それ以外}) \end{cases}$$

基本的には外部からの書き込みがないときはマスターらしいと判断し，そのときは外部からの読み出しが多いほど指標値を大きくする．ただし WO も RO も 0 であるときには同じクラスタ内部でのみ使用されているマスターらしい挙動とみなし得るために 2 番目の場合分けを設けている．

(4) ログパターン指標は，システムの様々な部分から書き込まれるが，読み出すのはごく少数または全くないような場合に値が高くなるよう設定される指標である．これは

動作ログに典型的なパターンである．読み出すプログラムは書き込まれたログのようなデータを分析するためのプログラムであることがある．この指標の計算方法は次の式として定義する．

$$I_4 = \frac{WO + WI}{RO + RI}$$

ただし，$RO + RI$ が 0 のときはこれを 1 として扱う．

続いて 2 種類の逸脱指標である．

(5) 内部専用パターン逸脱指標とは，前述の内部専用パターンに近いが少数の例外的なアクセスが存在する場合に高くなる指標である．ここで例外とはすなわち，アクセスの多くは同じクラスタ内からでありほぼ内部専用パターンに近く見えるが，しかしクラスタ外からのアクセスが少数だけあるような場合である．この指標の計算方法は次の式として定義する．

$$I_5 = \begin{cases} 0 & (WO = 0 \text{ かつ } RO = 0) \\ \frac{RI+WI+RO+WO}{RO+WO} & (\text{それ以外}) \end{cases}$$

(6) マスターデータパターン逸脱指標は，前述のマスターデータパターンに近いが少数の例外的なアクセスが存在する場合に高くなる指標である．ここで例外とはすなわち，所属クラスタ外からの読み出しが多数あるがその一方で，外部からの書き込みも少数存在するときに高くなるよう設定される．マスターデータパターン指標が外部からの書き込みが存在しないことを前提としているのに対し，外部からの書き込みが少数存在するとこの逸脱指標の値が高くなる．この指標の計算方法は次の式として定義する．

$$I_6 = \begin{cases} 0 & (WO = 0) \\ \frac{RO+WO}{WO} & (\text{それ以外}) \end{cases}$$

3.3 計算例

以上の指標の計算例を簡単な例題を用いて説明する．例題としてソフトウェアシステム全体が p1 から p5 までの 5 つのパッケージ（サブシステム）からなり，各パッケージには 9 つのクラスが含まれるプログラムを想定する．各パッケージ内のクラス間は全結合しており，パッケージ間にまたがる依存関係が p1 から p2，p2 から p3，……，p5 から p1，という形で存在する．パッケージ p1 のあるクラスはデータベースのテーブル（名称 Goods_Master）を読み書きしており，p2 から p5 に属するクラスそれぞれひとつが，Goods_Master を読み出している．また，同じくパッケージ p1 のあるクラスは作業用のデータに読み書き両方のアクセスを行う．このデータは他のクラスからはアクセスされない．このデータの名称は Working_Data とする．この事例を SArF によってクラスタリングすると，p1 から p5 までのパッケージは内部が全結合していることからそれぞれひとつのクラスタとしてまとまり，合計 5 つのクラスタが得られる．データ Goods_Master ならびに Working_Data は p1 のクラスタに所属する．名称から推測されるように，Goods_Master はマスターデータを，Working_Data はプログラムが作業中に読み書きするデータを想定して設定している．

表 1 例題の応用指標値

指標	Goods_Master	Working_Data
グローバル	0.0	0.0
内部専用	0.0	1.0
マスター	4.0	0.0
ログ	0.5	1.0
内部専用逸脱	1.5	0.0
マスター逸脱	0.0	0.0

この例題の 2 つのデータすなわち Goods_Master と Working_Data それぞれについて，上述の各指標値を計算する．Goods_Master は，自分が所属する p1 相当のクラスタ内から読み書き両方の依存関係をそれぞれ 1 ずつ持ち，他の 4 つの各クラスタから読み出しのみの依存関係を 1 ずつ持つ．したがって，既述の 4 つの基礎指標は $RI = 1$, $WI = 1$, $RO = 4$, $WO = 0$ となる．また，Working_Data は同じクラスタのクラスから読み書きそれぞれ 1 件ずつのみであるので，基礎指標は $RI = 1$, $WI = 1$, $RO = 0$, $WO = 0$ となる．これらの基礎指標に基づいて応用指標を求めると，**表 1** となる．Goods_Master はマスターデータ指標が 4 と高くなっている．一方 Working_Data においては同指標値は 0 であり，このデータにはマスターデータらしさはないと判断される．また，Working_Data は内部専用指標が 1 となっており，自分の所属するクラスタ内に閉じて使用されていることを示している．この指標は前述の通り 0 か 1 の二値であり，Goods_Master においては外部からの参照が存在することを反映して 0 の値となっている．ログパターン指標は共に 1 以下の値となっている．この指標値は 1 より大きいほどログらしさを表す．この例ではどちらのデータもログらしいとはいえない．

この例題でマスターデータらしさと特定機能内部の作業データらしさが指標に反映されることを示したが，ここで用いた説明は文章と数値のみであるため直感的に理解しやすいとはいい難い．これをより理解しやすい形で表現するため，次節ではクラスタと指標値を地図形式により視覚的に表現する手法を導入する．

3.4 指標に基づいた可視化

3.4.1 地図形式の可視化

ソフトウェア構造の理解のために，構造を視覚的に表現する技法がしばしば用いられる．ソフトウェアの可視化にソフトウェアメトリクスを組み合わせて表現する Polymetric

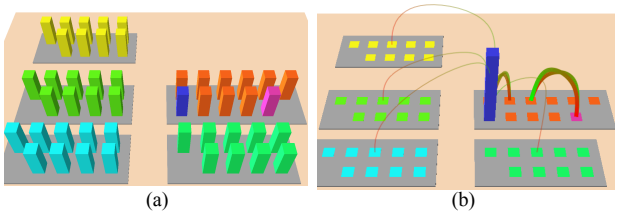

図 1 小さな例題による可視化の例

View [6] のアプローチは，各データに指標値を求めている本研究においても，指標値をメトリクスとみなすことで有効であると考えられる．本研究では，SArF Map [4] の方式によってプログラムとデータを含んだシステム全体を都市の地図を模した形式で表現する．この方式により，クラスタに相当する地図上の区画にプログラムとデータが共に配置され，どのようなプログラムとデータがシステム上の業務や機能を実現しているかを直感的に表すことができる．直感的な可視化はプログラムコードの読解と組み合わせて使ったり [6]，プログラムの詳細を知らないプログラマ以外の利害関係者と共に議論するために使ったり [4] することができる．CodeCity [13] に代表されるこうした地図形式の可視化手法は今日では研究のみならず産業界への適用もなされており有用性が認識されている [10][16]．

前節で用いた例題を本方式により可視化した例を，**図 1** に掲げる．図中の (a) と (b) は，同一のソフトウェアの地図に対し表示の仕方を変えたものである．地図上のひとつひとつの建物はプログラムまたはデータに相当する．建物の色はプログラム（クラス）の所属するパッケージ (p1, ..., p5) によって分けられている．ただしデータはプログラムと異なる色で表現されている．Goods_Master は青，Working_Data は紫色である．クラスタは建物の乗っている区画として表されている．区画が 5 つあるのは，パッケージ p1 から p5 までに相当する（なお，各区画それぞれで全てのプログラムが同じ色すなわち同じパッケージであるのは，そのようなクラスタリング結果が得られるように例題データを作ったためであり，通常は各区画=クラスタの中が同じ色=パッケージで揃うとは限らない．複数のパッケージのクラスが 1 機能をなす場合は複数の色が区画に現れる [4]）．p1 相当のクラスタ（地図の 2 段目右側の区画）にはデータ Goods_Master と Working_Data が共に所属している．これらのデータが他のクラスタではなく p1 に所属したのは，このデータへの読み書き両方を行うクラスが p1 に存在することにより他のクラスタよりも結び付きが強いためである．

図 1 の (a) は著者らが用いている可視化ツールで表示した初期状態であり，高さに何もメトリクスが割り当てておらず全て同じ高さで表現されている．一方 (b) は指標を高さへ割り当てた例である．ここで割り当てた指標は上記マスターデータ指標である．表 1 に示されているようにデータ Goods_Master は当該指標値が 4.0，また Working_Data は 0.0 であることが高さに反映されている．すなわちマスターデータらしいと指標値により判断されるものが目立つよう表示されている．なおプログラムの建物は当該指標値が存在しないので高さ 0 として表示している．このように指標値を建物の高さに割り当てることにより，特定の特徴を持つデータを目立たせることができる．

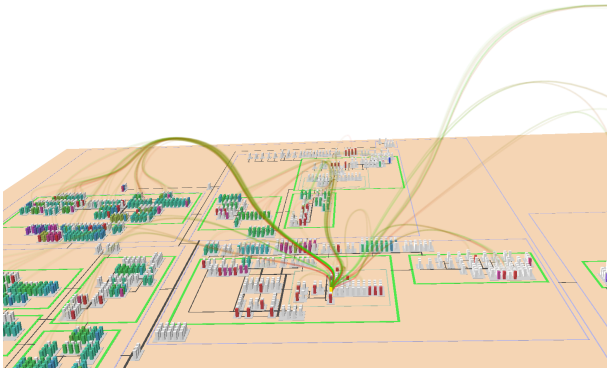

図 2 様々なプログラムからアクセスされるデータの可視化例

(b) においてはさらに，プログラムがデータへアクセスする関係を，建物の間に描画される曲線で表している．データ Goods_Master へは自分と同じクラスタ内のプログラム（当該データの右上の建物）から読み書き両方の線がある一方，他の各クラスタのそれぞれ 1 つのプログラムから読み出しのアクセスがあることが分かる．各クラスタから読み出しの線があることはこのデータのマスターデータらしさを反映している．また Working_Data に対しては自分と同じクラスタ内から読み書き両方のアクセスがあるだけで外部からのアクセスはないことが見てとれる．

曲線は緑から赤へのグラデーションとして色付けされている．依存元の側が緑，依存先が赤となる．この表現方法は SArF Map [4] におけるプログラム間の依存関係の表現を踏襲している．プログラムとデータの間のアクセス関係の表現でも基本的に同じだが，データを読み出す関係を表す線においては，線の向きを反転させてもよい．この反転させた表示を用いると，緑から赤への向きが読み書きにおけるデータの流れと一致する．すなわち，プログラムがあるデータから読み出して，別のデータに書き込むという流れが，線の緑から赤へという流れとして可視化される．データの流れを追いたい場合や，データを中心とした見方において読み書きのアクセスを区別して見たい場合にはこの反転させた表示が有用である．本稿の図ではこの表現を採用している．なお，曲線の太さの違いは，関係の重みを表している．本稿の例における重みは SArF [3] における dedication score を用いている．

プログラムとデータ間の依存関係を可視化する実際的な例として，**図 2** を挙げる．この表示例は，ある業務システム内の多数の様々なプログラムから読み書きいずれもされているデータに着目し，プログラムからこのデータへのア

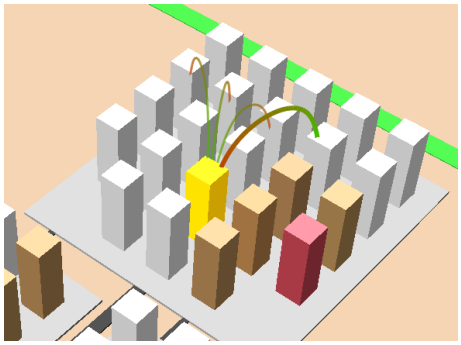

図 3 内部データの可視化例

クセスを地図上の建物の間に曲線として表示したものである．着目したデータは図の中央やや下の方，緑と赤の曲線が集中している箇所にある．この図から，問題のデータが，地図上の様々な領域のプログラムから使用されていることが，一見して分かる．地図上の様々な領域ということはシステム内の様々な機能ということを意味する．このようなデータは，依存関係がシステム全体に広がっており，修正による影響が広範囲にわたるため注意を要することが推測できる．このデータは前述のグローバルデータパターン指標が大きな値をとるものを選択したものである．

対照的な例として，同じシステムから，特定のクラスタに閉じて使われるデータの可視化例を**図 3** に掲げる．この図の中央に黄色くハイライトしたデータは，同じ区画内の 1 つのプログラムから書き込まれ，3 つのプログラムから読み出されている．このデータは他のクラスタのプログラムとの関わりはない．したがって，このデータはクラスタの機能内に閉じて使用されているものであり，データの変更が直接には他機能に影響せず，保守性が良好であるとみなせる．このデータは前述の内部専用パターン指標が 1 となる．なお，図中に見える緑色の直線はクラスタ階層を括る枠の一部である．

3.4.2 指標値の地図上の表現方法

続いて，指標のソフトウェア地図上の表現方法を説明する．前節で述べたデータアクセスの指標は，ソフトウェアメトリクスと同様に扱い，地図上の建物の表示に反映させることができる．割り当てることのできる地図上の建物の表現は，高さ，色，建物の乗っている土地の高さ，建物の並びの乱雑さ，それに明るさである．本研究の指標値の表現として使用が適当であると考えられるのは，高さと色である．建物の高さは最も視認しやすい表現方法である．高さに割り当てる例は既に図 1 の (b) に示した．

色には三通りの使い方がある．第一は，事前に知られている分類によって色分けする方法である．プログラムにおけるディレクトリやパッケージに応じた色分けと同じように，データベーステーブルの命名規約やファイルの属するディレクトリによって色分けする．ふたつめは，利用者が選択したメトリクス（指標）を割り当て，値の大きいほど赤く表示されるなどして目立たせる．例えばグローバルデータ指標を割り当て，同指標値が高いほど赤く表示する方法である．もうひとつは，前節の計算によって得られた指標に基づいてデータをマスターデータやログデータ等に分類したうえで，その分類結果に応じて色分けする．本稿の図では，第一の方法によって色を用いている．

建物の表す対象がプログラムかデータかによって建物の形状を変えることも可能である．例えば，プログラムに相当する建物は直方体，データの建物は円柱形にすることもできる．これによりプログラムとデータの区別を付けやすくなることが期待される．ただしこの方式は，可視化対象の規模の小さなときには有効であるが，実際の業務システムのような規模になると，全体を俯瞰したとき個々の建物が小さく表示されて形状の違いが視認できないという問題がある．ソフトウェア地図全体のどの領域にデータが多いかといった分布を見る目的には形状よりも色の方が視認しやすく有効である．本稿の図では形状の違いを用いず，色分けによってデータを示している．

4. 事例

本節では事例として，実際に企業で運用されている業務システムに本提案手法を適用し，データの性質の判断に用いる例を挙げる．なお，秘密保持のため，ソースファイル名やテーブル名，業務内容等の詳細は開示しない．

4.1 事例 1

最初の事例は，保険業務に関するあるシステムを分析した例である．COBOL で開発されており，ソースファイル数は 2,131 である．それらのプログラムからアクセスされているデータベースのテーブル数は 114 ある．

このプログラムとデータの依存関係を，本稿の手法により分析した．プログラムとデータからなるクラスタ群が得られ，また各データについて上述の各指標を計算した．

指標計算の妥当性を調べるため，提案手法の指標計算によりこのシステムのデータベースのテーブルからマスターデータらしいものを抽出し，その結果をこのシステムの設計情報と比較して，設計者によってマスターデータとされているかどうか検証することを試みた．

前述の計算方法により得られたマスターデータパターン指標ならびにマスターデータパターン逸脱指標のそれぞれについて，0 より大きな値をとるデータのうち大きな方から 20 件ずつを抽出し，これら 40 件を，本手法によりマスターデータらしいと判断されたものとする．なお，この 2 つの指標は，あるデータについて片方の指標値が 0 より大きな値であるときにはもう片方は 0 になっており，ここで 20 件ずつ抽出したデータ名に重複はない．今回，マスターデータパターン指標値の第 20 位が同じ値で 2 件存在したため，21 件のデータを抽出した．マスターデータパターン

逸脱指標は第 20 位に同値のものがなかったためちょうど 20 件を抽出した．したがって合計 41 件のデータが本手法によりマスターデータらしいと判断された．

抽出した合計 41 件を設計情報に照らし合わせると，マスターデータと記されているものはそれぞれ 14 件と 15 件，合計 29 件だった．したがって，指標に基づいてマスターデータらしいとして抽出したもののうち，実際にマスターデータであったものの割合は，$(14+15) \div (21+20) \fallingdotseq 0.71$ である．

次に，設計情報においてデータベースのテーブルの中でマスターデータとされているもの全部のうちどれだけがこの手法で抽出できたかを検証する．設計情報においてマスターデータとされているものは全体で 37 件であった．上記の手法によってマスターデータらしいと判断されたもののうち，設計情報においてもマスターデータとされているものは，マスターデータパターン指標の 14 件とマスターデータパターン逸脱指標の 15 件で合計 29 件であった．したがって，設計情報によって分かるマスターデータ全体のうち，この手法によってマスターデータとして抽出されたものの割合は $(14+15) \div 37 \fallingdotseq 0.78$ となる．

以上の結果から，完全ではなく改良の余地がありそうではあるものの，本手法の指標計算により，プログラムとデータの依存関係だけからマスターデータを自動的に抽出することが可能であることが分かった．

続いて，マスターデータパターン逸脱指標に着目し，値の高いデータが実際に何であるかを調べた．この指標値の高いものは，マスターデータらしいが例外的なアクセスも一部にあるということを意味する．本事例においてどのようなものがこの指標値が高くなっているかを確かめる．値の最も大きなデータは，業務上の請求情報を格納するものであった．名称には「マスター」という言葉がついていた．請求情報という性格を考慮すると，当該データはマスターという名称がついているものの，業務の進展によって更新されるトランザクションデータとしての性質が強いものであると考えられる．こうした例は，マスター逸脱指標の高いものにほかにも見られた．例えば履歴情報を格納するテーブルにも「マスター」という命名がされていたが，履歴というものは随時増加していくものであり，トランザクションデータ的な性質と考えられる．

この事例から，設計時の命名や分類がマスターデータとされていても，更新頻度がそれほど低くないトランザクションデータ的性質を持つものも存在することが，逸脱指標の高いデータを調べることで明らかにできた．これは担当者の異動などで新たな開発者がシステムを正しく理解することに役立つと期待できる．マスターデータとトランザクションデータの区別は必ずしも明確でないことが指摘されている [8][14] ことから，データの実際の使われ方を証拠に基づいて可視化し，用語の解釈の違いによる齟齬を避けることは，システムの理解や修正・再設計の助けになると考えられる．

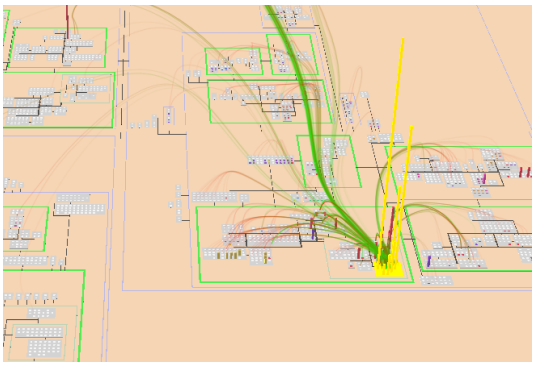

図 4　事例 2 のデータアクセスの可視化例

4.2　事例 2

次の例は，不動産管理業務に関するあるシステムを分析した事例である．このシステムは COBOL で開発されている．ソースファイルが 4,563 あり，それらのプログラムからデータベースのテーブルが 134，ファイルが 1,210 使用されている．この事例によって，指標を可視化技法と組み合わせることの有用性を示す．

このシステム全体をデータアクセスの情報を含めてクラスタリングして地図形式に可視化したものの一部が**図 4** である．なお，実際の使用上は地図上のそれぞれの区画の上に，その区画に含まれるプログラムやデータの特徴語をラベルとして浮かべて当該区画がどのような業務を実現しているかの理解に役立てることができるが [15]，この図では秘密保持のため非表示としている．

図 4 では建物の高さとしてマスターデータパターン逸脱指標を割り当てている．この図の右下の領域に高い建物が複数集中しているのが目を引いた．図 4 の表現では当該領域を黄色でハイライトしている．また，ハイライトしたプログラムとデータに関するデータアクセス関係を，緑から赤のグラデーションの曲線として表示している．図から読み取れるようにこの領域には他のクラスタとの依存関係が多く存在する．とりわけ多いのは緑色の線が左上方向に伸びている線である．これは，ハイライトしている領域に含まれるデータを他の領域のプログラムから参照している線の集まりである（前述のとおり，プログラムがデータを読む関係はデータ側が緑，プログラム側が赤のグラデーションの曲線で表現している）．

調べてみると，データの実態が，設計情報に記載されていたテーブル名称から想定されるものと異なることが分かった．当該指標の高いこれらのデータは，契約に関するデータベースのテーブルであった．システムの設計情報にはテーブル名として契約についてのマスターデータと記されており，実際，異なる業務の多数のプログラムから参照

されておりマスターデータらしい性質を持ってはいた．しかし一方，これらのテーブルはいずれも 10 以上のプログラムから書き込み処理が行われており，更新の少ないマスターデータというよりも業務の進展に応じて随時更新されるトランザクションデータに近い性質をも持ち合わせていることが分かった．このクラスタに含まれるプログラムとデータを調べると，業務で重要な役割を果たす契約情報ならびにそれを管理するプログラムが集まっていることが分かった．結果として，ここに属するプログラムとデータはシステム全体への影響が大きく，メンテナンス上十分な注意を要する部分であるといえる．

このように，計算によって得られた指標値をソフトウェア構造の上に重ねて可視化することで，システムの保守上注意すべき点に気付くことができる．問題のありそうに見える箇所は，設計情報やプログラムそのものを詳しく調べることで，本当に問題が存在するかどうかを吟味する必要がある．可視化だけで問題の詳細までが分かるわけではないが，複雑化したシステムの調査をどこから着手すべきかを発見するために本手法を用いることができる．

5. まとめ

本研究では，既存のソフトウェアシステムを理解するために，プログラムとデータを含めたクラスタリングを行い，その結果に基づいてデータの性質を表す指標を求めることを行った．依存関係とクラスタリングの結果を用いることで，データの性質を自動的に識別することが可能であることが分かった．また，得られた結果を地図形式によって可視化し，直感的に理解しやすく表現できるようにした．こうした指標計算や可視化は，長年維持されてきたシステムの現状を，担当者の異動やドキュメントの陳腐化といった状況において理解するために役立てることができる．

今後の課題としては，指標計算の更なる検証や改良，得られた結果のソフトウェア保守改良活動へのより広い応用が挙げられる．

参考文献

[1] Brunner, J.-S., Ma, L., Wang, C., Zhang, L., Wolfson, D. C., Pan, Y. and Srinivas, K.: Explorations in the use of semantic web technologies for product information management, *Proceedings of the 16th international conference on World Wide Web*, ACM, pp. 747–756 (2007).

[2] Cleven, A. and Wortmann, F.: Uncovering four strategies to approach master data management, *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, IEEE, pp. 1–10 (2010).

[3] Kobayashi, K., Kamimura, M., Kato, K., Yano, K. and Matsuo, A.: Feature-gathering dependency-based software clustering using Dedication and Modularity, *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pp. 462–471 (online), DOI: 10.1109/ICSM.2012.6405308 (2012).

[4] Kobayashi, K., Kamimura, M., Yano, K., Kato, K. and Matsuo, A.: SArF Map: Visualizing software architecture from feature and layer viewpoints, *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pp. 43–52 (online), DOI: 10.1109/ICPC.2013.6613832 (2013).

[5] Kuhn, A., Ducasse, S. and Gírba, T.: Semantic clustering: Identifying topics in source code, *Information and Software Technology*, Vol. 49, No. 3, pp. 230–243 (2007).

[6] Lanza, M. and Ducasse, S.: Polymetric Views—A Lightweight Visual Approach to Reverse Engineering, *Transactions on Software Engineering (TSE)*, Vol. 29, No. 9, pp. 782–795 (online), DOI: 10.1109/TSE.2003.1232284 (2003).

[7] Mancoridis, S., Mitchell, B. S., Chen, Y. and Gansner, E. R.: Bunch: a clustering tool for the recovery and maintenance of software system structures, *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, pp. 50–59 (online), DOI: 10.1109/ICSM.1999.792498 (1999).

[8] Otto, B.: How to design the master data architecture: Findings from a case study at Bosch, *International Journal of Information Management*, Vol. 32, No. 4, pp. 337–346 (online), DOI: 10.1016/j.ijinfomgt.2011.11.018 (2012).

[9] Patel, C., Hamou-Lhadj, A. and Rilling, J.: Software Clustering Using Dynamic Analysis and Static Dependencies, *Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on*, pp. 27–36 (online), DOI: 10.1109/CSMR.2009.62 (2009).

[10] Seerene: Seerene web site, Seerene GmbH (online), available from ⟨https://www.seerene.com/⟩ (accessed 2016-07-14).

[11] Tzerpos, V. and Holt, R. C.: ACDC: An Algorithm for Comprehension-Driven Clustering., *wcre*, pp. 258–267 (2000).

[12] Van Geet, J. and Demeyer, S.: Lightweight visualisations of COBOL code for supporting migration to SOA, *Electronic Communications of the EASST*, Vol. 8 (2008).

[13] Wettel, R. and Lanza, M.: Visualizing software systems as cities, *VISSOFT 2007 - Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 92–99 (online), DOI: 10.1109/VISSOF.2007.4290706 (2007).

[14] Wieczorek, S., Stefanescu, A. and Schieferdecker, I.: Test Data Provision for ERP Systems, *2008 1st International Conference on Software Testing, Verification, and Validation*, pp. 396–403 (online), DOI: 10.1109/ICST.2008.53 (2008).

[15] Yano, K. and Matsuo, A.: Labeling Feature-Oriented Software Clusters for Software Visualization Application, *2015 Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, pp. 354–361 (2015).

[16] 富士通：アプリケーションの見える化を強化した「業務・アプリケーション選別サービス」を提供，富士通株式会社（オンライン），入手先 ⟨http://pr.fujitsu.com/jp/news/2013/05/13-3.html⟩（参照 2016-07-14）．