ソフトウェア工学 95－1
（1993．11．30）

クラスタ分析を用いたプログラム構造解析の一手法

小田利彦
（株）リコー　情報通信研究所

概要

プログラムからその構造や設計レベルの情報を獲得するために、クラスタ分析を適用した解析手法について述べる。この手法では、モジュール内の外部参照の出現という事象に注目して、情報隠蔽の尺度をエントロピーの概念で表現した類似性測度を定義した。その結果、同じ外部参照を共有する性質を持つモジュールのクラスタの階層的樹形図と、外部参照同士がモジュールで同時に出現する事象に基づくタイプや大域変数等のクラスタの階層的樹形図を得た。本手法を実現するツールと、その実行結果を示す。そして、これらのクラスタ樹形図の有用性について議論する。

A Method to Capture Program Structure by Applying Cluster Analysis

Toshihiko Oda
RICOH Information and Communication Research and Development Center
3-2-3, Shin-Yokohama, Kouhoku-ku,Yokohama,222

Abstract

Our research interest is to capture design abstraction from program itself. For that, we pay attention to program feature regarding the occurrence of external reference in module. And cluster analysis is applied to investigate program structure by using the similarity measure based on information hiding in terms of the occurrence of external reference. The results are represented hierarchical cluster tree of program elements such as module, type, global variable. Prototype tool was implemented to evaluate our method. We discuss what helpful information the results can provide for particular activities of software process.

1 はじめに

今日、ハードウエアの進歩に伴い、ソフトウエアはより複雑にそして大規模になっていくため、その保守はより困難でコストを要することが問題となりつつある。ソフトウエア保守は、バグの修正やユーザの新たな要求を満足させたり計算機環境の変化に対応させるために、ソフトウエアを修正あるいは改良するという、ソフトウエア製品の寿命を伸ばしていくための継続的な作業である。

開発プロセスと保守プロセスの相違に関して多くの議論がある[LPLS78,Co89,BR91]。例えば、開発プロセスでは、ユーザの仕様を獲得した後に設計作業に移るが、保守プロセスではユーザの仕様を対象プログラムのデザインと照合し、新たな機能をどの個所に加えるのか、またそれがどのように他の個所へ影響が波及するのかについて明らかにしなければならない。それゆえ、保守プロセスにおいては、元のプログラムを十分に理解することが重要であり、保守者はそのために多くの時間を費やしている。

システムを理解するには、仕様や設計のドキュメントを参照しながら行なうが、多くの場合、こうしたドキュメントから十分な情報が得られず、また、システムが頻繁に保守変更されていると、ドキュメントとそのプログラムとの整合性が失われていることもある。その結果、保守技術者が、プログラムがどのように振るまいかつその構成要素がどのような役割を行なうのかを学ぶためには、ソースコードを直接に読むことが唯一の手段となる。

こうした問題に対する解決手段の一つとして、リバースエンジニアリングと呼ばれる技術が注目を集めている。これは、対象とするシステムの構成要素とそれらの間の関係を明らかにし、そこからより高い抽象レベルでシステムを表現するプロセスである[CC90]。具体的には、ソースコードから仕様や設計レベルの情報に還元することが挙げられる。

本研究の目的は、ソフトウエア保守におけるプログラム理解を支援するため、完成したプログラムを解析して、プログラムの抽象情報を自動的に抽出することである。

本稿では、特に大規模プログラムの大局的な構造を把握することを目指したアプローチをとる。そして、情報隠蔽という概念に基づいたクラスタ分析を適用してプログラムの解析を行なう一手法を考案し、それをC言語を対象に実現し、解析した結果について述べる。第2節では、関連する研究を述べる。第3節は、情報隠蔽をクラスタ化によるエントロピーの変化で捉えたクラスタ分析の手法を説明する。第4節では、本手法を実現したツールによる解析例を示す。第5節では、得られたクラスタ情報の性質や意味を述べ、第6節では、この情報の利用について述べる。

2 関連研究

リバースエンジニアリングに関しては、様々な研究がある[CNR90,Bi89,KNE89,Cl89]。その中で、クラスタ分析を適用してプログラム構造を見つけだす研究に関して、R.Selbyたち[SB89]は、データ束縛のメトリックを用いてコンポーネント間のデータ送受の強さを計量することを行ない、これに基づくコンポーネント間の類似性測度からクラスタ分析した結果、システムの階層的なサブシステム構成を生成した。さらにサブシステムでのエラーの個数やそれに要した労力からサブシステムを特徴付けている。A.Delisたち[DB90]は、このデータ束縛のメトリックを用いてクラスタ分析を行なう、dbtというAdaプログラムを対象にしたツールを開発した。dbtの出力、階層的クラスタ樹形図は再利用性の高いコード部品を分離したり、またパッケージ化する方法を理解するために利用できることを示した。

他方、L.Briandたち[BMB93]は、Adaモジュール（ルーティン、データ、タイプ定義の集まり）に拡張したコヒージョンとカップリングの定義を提案し、モジュール間の関連性を計量化した。これにより、保守活動の困難さを事前に推定できることを示した。

情報量とシステムの複雑さの関連については、D.Paulsonたち[PW92]のシステムの分割方法を形式化した研究がある。この研究では複雑さとはインプリメントや保守に要するコストと関連する尺度であり、その定義はモジュールへの入力事象の情報量に基づいている。仕様設計時にシステムの分割案を求める場合、この複雑さの計量値を用いて分割案の候補をランク付けすることで、より良い分割が行なえる。

3　クラスタ分析によるプログラムの構造解析

クラスタ分析は、観測データを組織化して意味のある構造にまとめ、分類を発展させるという問題で広く利用されている手法である。その目的は、類似性あるいは距離の測度を使用して、対象を結合してクラスタを形成していき、最終的にクラスタの階層的樹形図を得ることである。我々は、この手法によりプログラムの大局的な構造を見つけだすことを試みる。

まず用語について述べると、モジュールとは、プログラムのプライマリ単位として定義され、独立にコンパイルが可能な閉じたサブルーチンである。C言語ではモジュールは個々の関数と対応する。次にモジュール外で定義されかつモジュール内に出現するプログラム要素のことを外部参照と呼ぶ。C言語では表1のプログラム要素が外部参照にあたる。

ところで、システム全体の複雑さの重要なタイプにモジュール間の関連性がある[My78]。このモジュール間の因果関係や依存関係による関連性は、こうした外部参照がモジュール内に出現することで引き起こされていると言える。また、モジュール内に出現した外部参照の意味を知るためには、コード上離れた場所にあるその定義を参照せねばならず、このことがモジュールを局所的に理解することを妨げている。

我々は、こうした外部参照の出現から導かれるプログラムの特性を、クラスタ分析を用いて解析し、システム全体のプログラム構造を見いだす。即ち、このプログラム構造は、プログラムに存在するプログラム要素、モジュール、タイプ、大域変数そしてマクロに対して、クラスタ分析を行なった結果得られる階層的樹形図として表現される。

クラスタ分析は、2つの方法で行なう。第1の方法は、情報隠蔽という概念に基づき、各モジュールでの外部参照の出現をクラスタの内部にできるだけ局所化するように、モジュールのクラスタを構成していく。そして、このクラスタの階層的構成から、情報隠蔽という特性を反映するプログラムの構造を調べる。情報隠蔽はプログラム設計の重要な特性であり、複合設計やオブジェクト指向設計等の設計手法では、データや操作の実現の詳細を隠すためにカプセル化と呼ばれる設計原則を導入している。それゆえ、このクラスタ分析の結果は、情報隠蔽の設計特性の観点から、プログラムの設計構造を表層化すると思われる。

第2の方法は、外部参照の各モジュールにおける出現の同時性という事象に注目する。同じモジュール内にある2つの外部参照が出現する現象は、それらの間に何らかの相互作用の可能性があることを見ることができる。例えば、大域変数AとBが複数のモジュールにおいて同時に出現していることが多い場合、AとBの間には設計上の関連性が存在する可能性が高いと考えられる。

表1　C言語におけるモジュール内の外部参照

1. モジュール呼び出し
2. モジュール内で用いられているタイプやそのメンバ
3. モジュール内でアクセスされた外部変数
4. モジュール内で使われているマクロ

3.1　エンティティと属性

エンティティとはクラスタの要素となるデータ単位のことであり、属性とはエンティティ間の類似性測度を計量するために用いられるエンティティの特質のことである。外部参照の出現を情報隠蔽するよ

表2　エンティティ・属性の組

Type	エンティティ	属性
C1-1	モジュール	そこで呼びだされたモジュール
C1-2	モジュール	そこで出現する外部変数
C1-3	モジュール	そこで出現するタイプとメンバ
C1-4	モジュール	そこで出現するマクロ
C1-5	モジュール	そのモジュール名に出現するターム

表3　エンティティ・属性の組

Type	エンティティ	属性
C2-1	モジュール	呼びだしたモジュール
C2-2	外部変数	それが出現するモジュール
C2-3	タイプとメンバ	それが出現するモジュール
C2-4	マクロ	それが出現するモジュール
C2-5	ターム	それが出現するモジュール

うなクラスタを生成する場合、エンティティはモジュール、属性は外部参照の出現事象となる（表2）。また、属性に対してその具体的な要素を属性インスタンスと呼ぶ。例えば、属性が大域変数なら、プログラム中に存在する個々の大域変数が属性インスタンスである。

ところで、外部参照に加えて、モジュールの名前を構成しているターム（文字列パタン）の出現についてもクラスタ分析の適用を試みる。これは、プログラムの可読性の重要な要因であるネーミングの側面から、モジュールのクラスタを調べるためである。

【C1-1】では、同じモジュール呼び出しを含んでいるモジュール同士からクラスタが形成される。

【C1-2】では、同一の大域変数を共有し合うモジュール同士からクラスタが形成される。

次に、各プログラム要素について、それが外部参照としてモジュール内に出現することの同時性に基づき、プログラム要素のクラスタを生成を行なう（表3）。ここでは、属性は外部参照が出現しているモジュールの集合となる。

【C2-3】では、異なったタイプ同士が、宣言やメンバの参照等として同じモジュールに出現するという事象から、タイプのクラスタを形成する。

【C2-5】では、モジュールの名前を構成しているターム同士が、同じモジュール名に出現し合うことに基づき、タームのクラスタを形成する。

3.2 類似性測度の計量

クラスタ分析では、その過程でエンティティの全ての2組に対して類似性測度を計量化する。そして類似性測度の定義は、クラスタ分析の結果がどのような意味を備えるかに大きく寄与するため、分析の目的を反映させた定義を与えねばならない。

我々は、情報隠蔽という設計上の原則に根ざしたモジュールのクラスタを生成するために、外部参照の出現を効率良くクラスタに局所化することを反映したモジュール間の類似性測度を定義する。

まず類似性測度の意味を述べる。エンティティ間の類似性は、それらをクラスタ化することでエンティティ空間における属性インスタンスの出現の乱雑さが減少することと関連づける。即ち、属性インスタンスの分布の度合いをエンティティ空間におけるエントロピーとして計量化し、2つのエンティティの類似性測度をクラスタ化することによるエントロピーの減少量と表現する。この定義により、クラスタ化過程は、エンティティ空間での属性の分布の乱雑さを最適に減少させていく過程と対応する。

まず、あるエンティティにおいてある属性インスタンスが出現するという事象に対して、その出現確率を次のように定義する。

$$P(atr\_ins_i,e_j) = \frac{NumOfOccur(atr\_ins_i,e_j)}{\sum_{k=1}^{NumEty} NumOfOccur(atr\_ins_i,e_k)}$$

```
atr_insi  ：i番目の属性インスタンス
ej        ：j番目のエンティティ
NumEty    ：エンティティの総数
NumAtrIns：属性インスタンスの総数
NumOfOccur(atr_insi,ej)：
          ejにおけるatr_insiの出現回数
```

ところで、NumOfOccurは、例えば、モジュールP内でモジュールMを何度も呼び出しても、Pにおける Mの出現回数は最大1としている。これは、モジュール間の関連の複雑さに、同一モジュール内での出現回数は依存しないからである。しかし、構造体タイプの属性の場合、構造体のメンバは個々にカウントするため1以上になることがある。

全エンティティにおける属性インスタンスのエントロピーは次のようになる。

$$H(atr\_ins_i,[e_1,e_2,\cdots,e_{NumEty}]) = \sum_{j=1}^{NumEty} P(atr\_ins_i,e_j)\log\left(\frac{1}{P(atr\_ins_i,e_j)}\right)$$

最後に2つのエンティティepとeqの類似性測度は、epとeqとクラスタ化した場合の属性インスタンスの事前エントロピーと事後エントロピーの差により求める（次式参照）。図1では、タイプに関してモジュールの類似性測度を求めた計算例を示す。

$$Distance(e_p,e_q) = \sum_{i=1}^{NumAtrIns} H(atr\_ins_i,[e_1,..,e_p,..,e_q,..,e_{NumEty}]) - \sum_{i=1}^{NumAtrIns} H(atr\_ins_i,[e_1,..,e_q \cup e_q,..,e_{NumEty}])$$

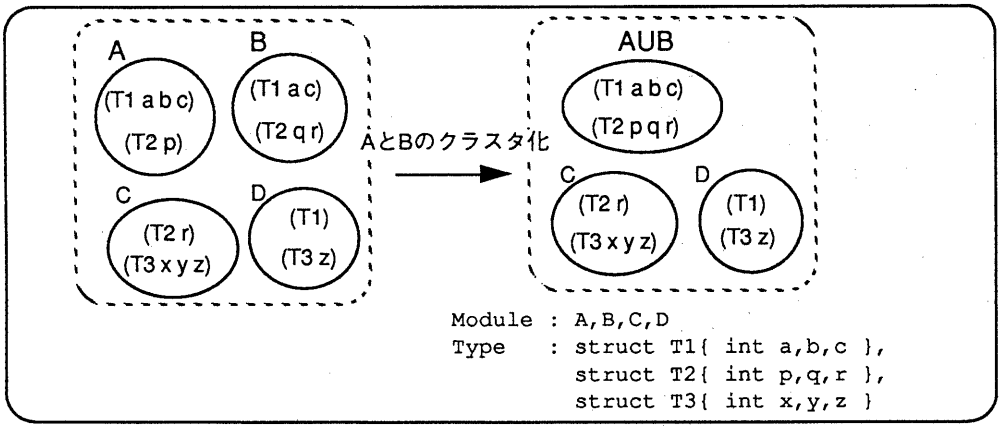
以上で述べた類似性測度は、情報隠蔽という概念に基づくモジュールのクラスタを生成するためのものであるが、表2のエンティティ／属性に適用すれば、外部参照の出現の同時性を共有し合う尺度として、外部参照間の類似性測度が得られる。

3.3 クラスタ化の手続き

クラスタの階層的樹形図を生成する手続きについて簡単に説明する。この手続きは主に、初期クラスタセットの生成、クラスタ化候補の選択、クラスタセットの更新の3つのステップからなる。第1ステップでは、クラスタリングの対象となる全てのエンティティを集めて初期クラスタセットを形成する。第2ステップでは、クラスタセットに含まれる要素（最初はエンティティ）の全ての2組を取り出し、その類似性測度を求める。そして、最も類似性測度が大きいクラスタの2組を選び出し、クラスタ化の候補集合に入れる。第3ステップでは、まず、この候補集合の中の同じ要素を含むクラスタ同士を一つのクラスタに併合した後、候補集合の内容をクラスタセットに反映させて更新する。そして、第2ステップに移り、候補集合が空になるまで繰り返す。クラスタセットをこの繰り返しの度に保持しておくと、最後に階層的クラスタの構成が得られる。

4 ツールの開発と実験

前述した手法を実現するための2つのツール、CEXTRとClusterViewerを開発した。CEXTRは、C言語プログラムを構文解析して、解析に必要な情報をASCIIファイルに出力する。ClusterViewerは、そのファイル情報を読み込み、クラスタ分析の実行や呼びだしグラフの生成等の解析機能とそれらを画面に表示する機能を持つ。このツールを使って解析した幾つかの既存プログラムの中で、ある通信プログラムの解析結果を示す。




```
D(A,B) = H([A,B,C,D]) - H'[AUB,C,D])
       = H(T1) + H(T2) - (H(T1)' + H'(T2))
       = -4/8*log(4/8) - 3/8*log(3/8) - 1/8*log(1/8) /* H(T1)  */
         -2/7*log(2/7) - 3/7*log(3/7) - 2/7*log(2/7) /* H(T2)  */
         - (-4/5*log(4/5) - 1/5*log(1/5))            /* H'(T1) */
         - (-4/6*log(4/6) - 2/6*log(2/6))            /* H'(T2) */
```

図1 タイプに関するモジュール間の距離計算の例

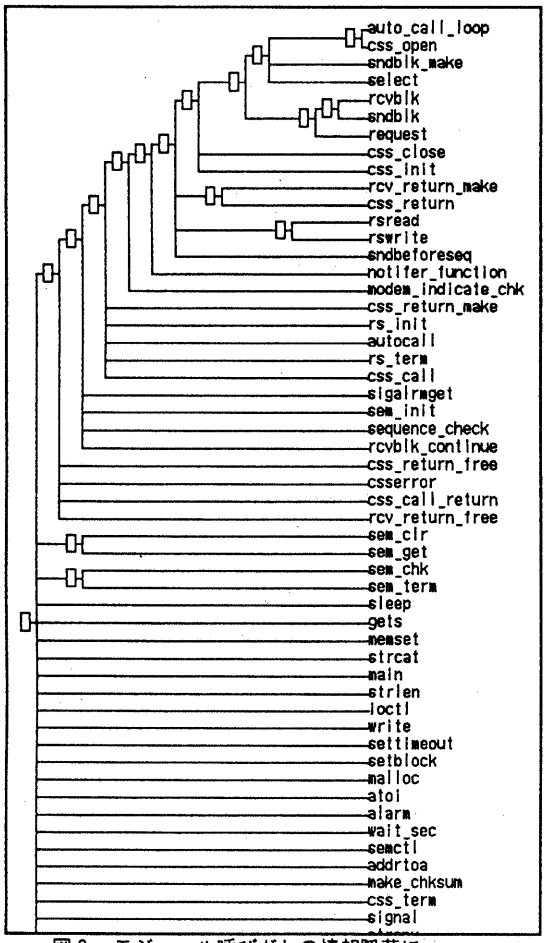


図2　モジュール呼びだしの情報隠蔽に
基づくモジュールのクラスタ階層【C1-1】

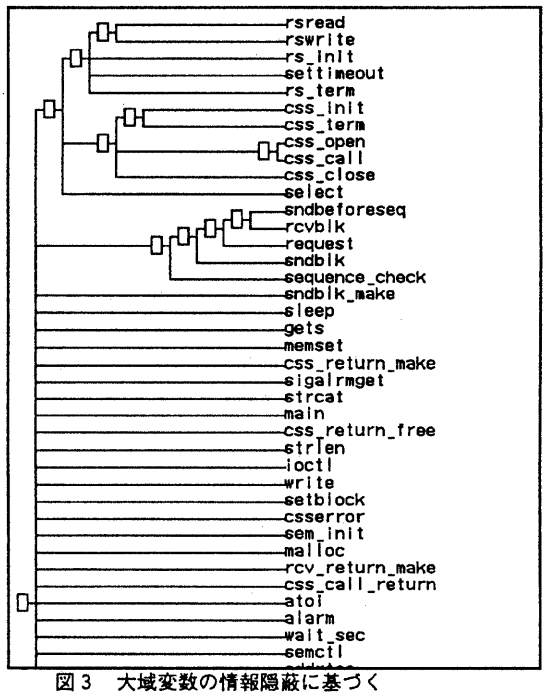


図3　大域変数の情報隠蔽に基づく
モジュールのクラスタ階層【C1-2】

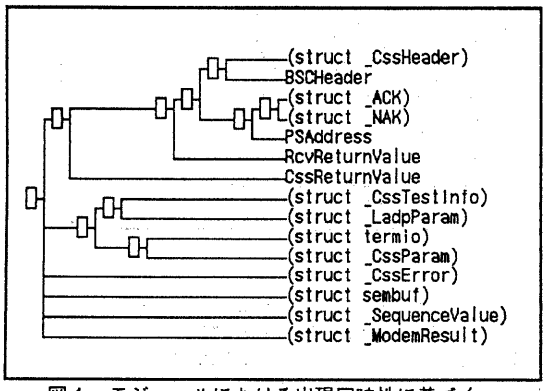


図4　モジュールにおける出現同時性に基づく
タイプのクラスタ階層【C2-3】

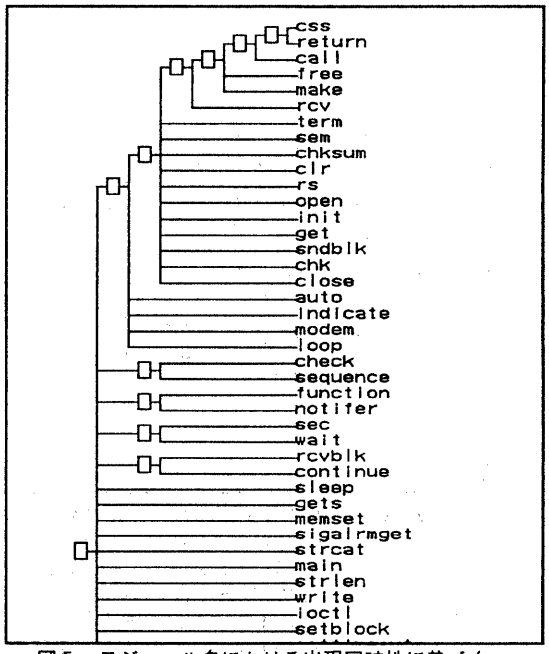


図5　モジュール名における出現同時性に基づく
タームのクラスタ階層【C2-5】

5 各クラスタの持つ性質と意味

クラスタ分析を行なった結果、表1、2にあるエンティティ／属性の組のそれぞれから、異なる意味を持つクラスタ樹形図が得られる。ここでは、それら樹形図のクラスタが有する性質と意味について述べる。

その前に、階層クラスタ樹形図上で、葉（leaf）の近くに位置するクラスタ、即ちクラスタ化手続きの過程で早期に形成された、類似性が高い要素の集まりを強いクラスタと呼び、逆に樹形図の根（root）に近いクラスタを弱いクラスタと呼ぶとする。

5.1 情報隠蔽クラスタ

表1のエンティティ／属性の組から、外部参照の出現の分布を効率良く囲い込んでいった、モジュールの階層的なクラスタの構成が得られる。

●モジュール呼び出しを隠蔽するクラスタ樹形図

強いクラスタに含まれるモジュール同士は同じモジュールの呼び出しを多く共有し合うため、それらの間に機能的な類似性が示されることがある。また、ある特定の外部ライブラリを呼び出しているモジュールの集まりを同定することができる。

●大域変数を隠蔽するクラスタ樹形図

大域変数を介してデータ束縛するモジュール間のカップリングは、外部カップリングと呼ばる。これは、互いの変更が敏感に影響するという保守性が良くない性質を持つ。強いクラスタでは、モジュール同士がより強い外部カップリングで結びつきあっているため、こうした性質を持ちやすい。

●タイプの情報を隠蔽するクラスタ樹形図

強いクラスタは、情報コヒージョンと呼ばれるモジュール同士の結合が高い特性を有している。また一般に、モジュールとそこで用いるデータタイプの種類の関係は、システムにおけるサブシステム構成と関連が深い。この場合、この樹形図からサブシステム構成を見いだす手掛かりになると言える。

●モジュール名のタームを隠蔽するクラスタ樹形図

共通のタームから構成された名前に基づく、モジュールのグループが得られる。

5.2 出現同時性クラスタ

表2のエンティティ／属性の組から、外部参照としてモジュールに出現する事象の分布を最小限にする各エンティティのクラスタの構成が得られる。

●モジュール呼び出しのクラスタ樹形図

強いクラスタには、呼び出し頻度が高いことから、汎用なモジュールが存在している可能性が高い。クラスタ樹形図の最上位のクラスタは、一つのモジュールだけから呼び出されるモジュールの集まりとなっている。

●大域変数のクラスタ樹形図

強いクラスタの中の大域変数は、互いにデータ束縛等の関連性が強いと考えられる。クラスタ樹形図の根の部分には、互いにデータ束縛がない大域変数のグループの分割が現われる。

●タイプのクラスタ樹形図

大域変数の場合と同じように、強いクラスタの中のタイプは、それと対応するデータ概念間に強い関連が存在といえる。また、互いに関連し合わないタイプのグループの分割も得られる。

●モジュール名のタームのクラスタ樹形図

モジュール名に同時に現われる場合が多いターム同士は、それと対応する概念や対象間にも関連性が存在する可能性がある。

6 クラスタ分析情報の利用

ソフトウエアの開発・保守プロセスの活動において、クラスタ分析を用いたプログラムの解析情報がどのような利用価値があるのかについて述べる。

まず、プログラム理解への利用については、前述したように、各クラスタは設計レベルの意味情報を含んでいるため、それを知ることによりモジュールを集合的に理解することができる。樹形図は、システムに存在するサブシステム構成を示唆することができる。これらは特に、プログラム理解の初期に行なう、トップダウン的に全体構造を把握することに有意義な情報であると言える。更に、モジュール名のターム同士の関連を知ることで、システム内に存在する設計上の概念や対象を知る手掛かりとする。

次に、保守による変更の波及を調べることについ

て述べる。あるモジュールを変更する場合、それが及ぼす影響範囲を、モジュールのクラスタ樹形図から知ることができる。そこでは、内部結合が強いクラスタ内のモジュールの変更は、よりリスクやコストが高いことも推定できる。大域変数やタイプを変更する場合も、それらのクラスタ樹形図からその変更が及ぼす影響範囲を把握することができる。

プログラムの再利用については、部品候補の再利用性を評価することが重要となる。それを判断する情報として、モジュール呼び出しのクラスタから部品候補の汎用性に関する情報が、またタイプや大域変数の出現を隠蔽するモジュールのクラスタから部品候補の独立性に関する情報が得られる。

7 おわりに

本稿では、プログラム構造をクラスタ分析により解析する方法について述べた。残された課題として、各クラスタ樹形図の情報を相互に比較し、また他の情報も取り入れることで、より妥当な設計構造と対応したクラスタを同定することであるが、現在$\chi 2$乗検定等の統計検定法を利用して、各クラスタの関連性を調べることを検討している。

謝辞

本研究に対して有意義な示唆やコメントを頂いたメリーランド大学のV.R.バジリ教授とL.ブライアンド博士に感謝します。また日頃から御指導いただいている江尻公一博士に感謝します。

参考文献

[CC90] E.J.Chikofsky, J.M.Cross, Reverse Engineering and Design Recovery:A Taxonomy, IEEE Software, January, 1990

[My78] G. Myers. Composite-Structured Design. Van Nortrand Reihold Company, 1978.

[CNR90] Y. F. Chen, M.Y. Nishimoto, C. V. Ramamoorthy, The C Information Abstraction System, IEEE Transactions on Software Engineering, SE-16(3), March, 1990

[DF87] R. Prieto-Diaz, P. Freeman, Classifying Software for Reusability, IEEE software, January, 1987

[Bi89] T. J. Biggerstaff, Design Recovery for Maintenance and Reuse, IEEE computer, July,1989

[SB89] R. W. Selby, V. R. Basili, Error Localization During Software Maintenance: Generating Hierarchical System Descriptions from the Source Code Alone, Proceedings of the Conference on Software Maintenance, Phoenix, October, 1989

[DB90] A. Delis, V. R. Basili, Data Binding Tool: a Tool for Measurement Based on Ada Source Reusability and Design Assessment, Technical Report, Dep. of Computer Science, University of Maryland, May, 1990

[BSI92] L. R. Brothers, V. Sembugamoorthy, A.E.Irgon, Knowledge-Based Code Inspection with ICICE, 1992

[Co89] T. A. Corbi,Program Understanding: Challenge for the 1990s, IBM System Journal, Vol.28, No.2, 1989

[Cl89] L. Cleveland, A Program Understanding Support Environment, IBM System Journal, Vol.28, No.2, 1989

[SM79] B. Shneiderman, R. Mayer, Syntatic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results, Internal Journal of Computer and Information Sciences, Vol.8, No.3, 1979

[LPLS87] D. C. Littman, J. Pinto, S. Letovsky, E. Soloway, Mental Model and Software Maintenance, The Journal of System and Software, 7, 1987

[An73] M. R. Anderberg, Clustering Analysis for Applications, Academic Press, 1973

[KNE92] W. Kozaczynski, J. Ning, A. Engberts, Program Concept Recognition and Transformation, IEEE Transactions on Software Engineering, SE-18(12), December, 1992

[BMB93] L. C. Briand, S. Morasca, V. R. Basili, Measuring and assesing Maintainability at the End of High-Level Design, CSM93.

[MB90] T. Mason, D. Brown, lex & yacc, O'Reilly & associates, Inc. 1990

[ASU86] A. V. Aho, R. Sethi, J. D. Ullman, Compilers: Principles,Techniques and Tools,Addison Wesley, 1986

[PW92] An Automated Approach to Information Systems Decomposition, D. Paulson, Yair Wand, IEEE Transactions on Software Engineering, SE-18(3), March, 1992