# 意思決定に基づくプログラム設計知識の解析と設計知識のチュータリング

鈴木 彦文*, 関本理佳**, 海尻 賢二***

# Analyze Program Design Knowledge based on Decision Making and Tutoring of Program Design Knowledge

H. SUZUKI and R.SEKIMOTO and K. KAIJIRI

In the past several years, the study of algorithm and the study of well-formed programming style have been under the surroundings where less instructors have been teaching . Educating each student should be originally man to man style where an instructor teaches each student individually . However , it is physically impossible .

In the respect of program design , this study focuses on the decision making on the design and it leads to the study of each phase of programming and the study given to the students . This can deepen the understandings between the standard decision making and the program structure .

キーワード: プログラム設計，意志決定，CAI

## 1.　はじめに

プログラム開発には多くの局面がある．しかしながらその局面すべてを指導することは困難であり，次の項目に対して指導が不足しがちである．

(1) アルゴリズムの学習
(2) アルゴリズムをプログラムへと具体化するステップの学習 (プログラム設計)
(3) Well-formed なプログラムスタイルの学習

特にプログラム設計に関してのチュータリングにおいては，プログラムの構文に関する情報だけでなく，設計に関わる幅広い知識を保持し，これを効率的に学習者に提供しなくてはならない[1]．

しかしながらこのような知識は定義し難くまた正しく習得することも難しい．このため実際に指導を行なう場合，各教官の経験や知識に大きく依存し，また多大な負担をかけることになる．そこでまず実際の指導局面 (シナリオ) をベースに，設計段階における意思決定[2]と実際のプログラムの関連付けを行なった問題解法木を作成する．そしてこれを形式的に表現することができれば，設計知識という形でそれを蓄積・利用することが可能になる．

今回の報告ではそのために必要な要素を取り上げ検討する．まず設計と指導について考え，ついで，指導にあたりどのような知識が必要であるかを確認する．そして，それをどのように分析・蓄積していくのかを考える．

* 電子情報工学科 助手,** 信州大学 工学部 情報工学科 助手,
*** 信州大学 工学部 情報工学科 教授
原稿受付 1998年10月30日

## 2.　開発と指導

プログラムの設計指導は次の基本的な条件を満たすものでなければならない[3]．

(1) 対象は初心者であるので，問題の解法を考える作業や，具体的なプログラミングなどの各作業において，その作業にのみ専念できること．過度な情報の提供は控えること．
(2) 各作業の状況に応じて，適切な診断・誤りの情報を提供できること．
(3) プログラムの完成よりも，問題を解くに至る各作業プロセスの学習が重要であるため，正解・誤りと結論付けるプロセスと的確な理由付けの提供をすること．

### 2−1　開発プロセスの把握

実際に指導を行なう場合，まずは設計プロセスを順序立てて説明しておかなくてはならない．最も単純なプログラム作成の課題ですら，次のようなステップを指導しないと現実的ではないといえる[3]．

**(1) 問題の解析** 問題の意図を分析し，必要な知識の収集や方針を計画する．
**(2) モデル化** モデルを作成し，問題の解決法を考える．このレベルでは，例えば数学や図など利用することにより，問題解決のための情報を獲得する．

ただし，この段階では特にプログラムやアルゴリズムと密接に関わっている必要はない．

**(3) アルゴリズム設計** 作成したモデルを実際にどのようにプログラムにするのかを考える．この段階では，ソフトウエアでどのようにモデルを実現するのかを考慮し，トランザクションや状態遷移を考える．

**(4) プログラム設計** この段階で，プログラム言語の種類を意識した設計を行なう．ただしC言語といった具体的なプログラム言語を意識したものではなくても構わない．ここで考慮するのは，手続き型や関数型，論理型といったレベルである．

**(5) プログラム化** プログラムの設計を基に，コード化を行なう．

このようなプログラム作成手順を踏むことにより，生産性を正確に把握したソフトウエア開発の指導が可能になる．最近では各種開発法が考案されているが，ほとんどのパラダイムではこのような手順を意識しているものである (正確には (5) プログラム化以降を考慮に入れての話が抜けているがここでは詳しくは説明しない)．

これらのステップをどのように捕らえるべきなのか．大まかな目安としては，次のように考えられる．

- **(1) 問題の解析〜(2) モデル化**
  問題がどういうものかを正確に把握する段階であり，この段階で誤った認識を行うと，後のすべての作業に影響してくる．プログラム言語やプログラム言語の型に密接に関わっている必要はない．
- **(3) アルゴリズム設計〜(4) プログラム設計**
  プログラム言語の型とは密接な連携が必要であるが，プログラム言語そのものには依存してはいけない．例えばC言語に特化したフロー図などはこの範疇に入らない．またPADの様にプログラムそのものを表すものもこの範疇には入らない．つまり，手続き型言語で言う意味的な情報が欠落していない状態である．
- **(4) プログラム設計〜(5) プログラム化**
  具体的なプログラム言語を意識した設計を行うレベルである．設計だけでなく，インプリメントするレベルであるので，実際のプログラム言語と切り放すことはできない．ここでいうプログラム設計は言語依存が高いものであるので，厳密に言えば，本来1つ上の段階のプログラム設計とは区別した方がよい．

以上の様に，問題から解答であるプログラムを作成する際には，多くの段階を経るということそのもののトレーニングを行う必要がある．それはたとえプログラムの設計を目的としたものでなくとも最低限このプロセスを把握できるように指導する必要がある．

## 2−2 プログラム開発に必要な知識

学生は問題に対して解答であるプログラムを作成するわけだが，このプログラムを作成する際に必要な知識は図1の様にまとめることかできる．

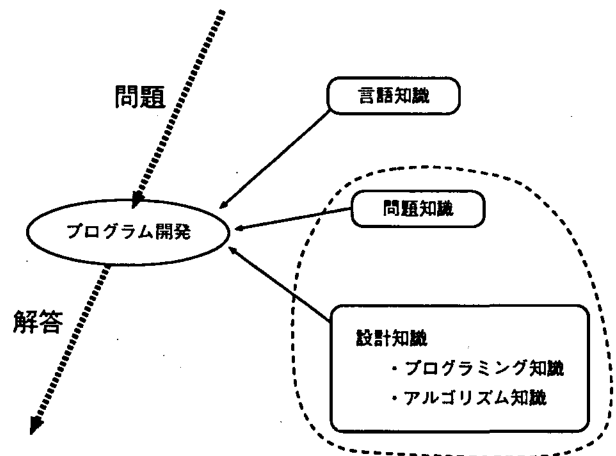


図1 問題を解く際に必要な知識

それぞれは次のような意味を表している．

**言語知識** 言語の構文などに関する知識．例えばC言語であれば，ループの作成方法や，変数の型などの知識がこれに相当する．

**問題知識** 「問題」を解く際に必要となる知識.「問題」は必ずしもプログラミングに適した用語などが使われているわけではない．また，計算に必要なすべての情報が盛り込まれているわけでもない．このような不足している情報を補う知識を指す．モデル化を行うためには必要な知識と言える．

**設計知識** プログラムをインプリメントするに当たって必要となる知識．アルゴリズムを作成するための知識や，プログラムを設計するに当たっての知識を表す．

これらの他にもツールに関する知識 (開発方法・環境に関する知識) が必要となるが，ここでは特にプログラムの設計に注目する．また，今回対象とするのは**言語知識の習得は終了している学生**を対象にする．すなわち，**問題知識**や**設計知識**とは具体的にどのようなものであり，それをどのように教授していくのかについてまとめる．

そして，本研究ではプログラム開発のプロセスにおける意志決定と段階的詳細化プロセスに注目し，これをベースにプログラムの設計知識を蓄積し，チュータ

リングに利用することを考えている.

### 2–3 プログラム設計のチュータリング

特にアルゴリズムの設計段階以降の開発プロセスにおいて, 特に必要となる指導は次のようなものがあげられる.

**意味付け・設計** プログラムそのものではなく, 抽象的な段階から設計してゆく. この時, 単にチャートではなく意味的な要素や構造的な要素を盛り込んで設計を行なえるようにする.

**意思決定・詳細化** 設計段階における意思決定・詳細化に対する指導を行なう. 学習者が実際に意思決定・詳細化を行なう. 詳細化とは段階的詳細化を表している. より抽象的な設計から, より具体的なコードへの refinement を行なわせるのである.

**意思決定の変更** 過去に行なった意思決定を変更する. この時変更に伴う影響がどのようなものかを指導する.

**トップダウン設計・ボトムアップ設計** 問題に対しての解であるプログラムを作成するまでの過程を指導する.

**診断** プログラムやその設計が正しいかどうかを判定し, 診断結果を学習者に返す.

## 3. サンプル

ここではサンプルとして, ソーティングについての決定, 詳細化, 決定の変更, それに伴う再利用がどのようになるのかを考えてゆく. ここで重要なのは, 学習者に対しての情報として, どのような情報をどこまであらかじめ獲得しておけば良いのかということである.

なお, ここでは対象をアルゴリズム設計以下の段階を考える. すなわち, すでに問題の解析とモデリングは終了していると考える. ソートとは「数の列を降順, あるいは昇順に並べ替える」ことである. このときソーティングのモデルとは, 自然言語で記述するならば, 例えば次のようになる.

表 1:ソートのモデル

| 要素数が $m$ である数の列の各々の要素を $n_{1\sim m}$ とする. $1 \leq i \leq m, 0 \leq j \leq m, i < j$ であるとき, $i, j$ がとり得る全ての範囲において, $n_i \geq n_j (n_i \leq n_j)$ であるような列をソートされた列とする. |
|---|

もちろん, ここまで形式的でなくても構わない. 表 1 ではできる限り曖昧性を排除するように書かれている.

### 3–1 意思決定

ソーティングプログラムを作成する場合, 開発の段階において表 2-a と表 2-b のような決定すべき項目がある.

表 2-a:設計方針の意思決定

- ソート方法
  どのようなアルゴリズムを利用してソートするのかの決定がある. 代表的なところでも「単純挿入」「2 分挿入」「Bubble」「Sell」「Heap」「Quick」及びその改良 (例えば「Shaker」) などがある[4]. もちろんそれぞれに効率・理解しやすさ・基本的な設計などといった情報があり, これが判断材料となる.
- データ構造
  データ構造は主に 2 種類あり「配列」と「リスト構造」が主なところになる.

表 2-a では, プログラム設計の大きな方針となる項目の決定を記している. ただし, 方法によってはさらに多くの決定が必要となるものもある. 例えば同じアルゴリズムでも, 実現方法 (設計) が大きく違うものもあるからである. さらにもう少し細かいレベルで見ていくと

表 2-b:インプリメント時の意思決定

- loop 文
  タイプによって分けられる. 主に 3 通りの基本的な構造が考えられる. (1) 条件前置,(2) 条件後置,(3) カウンタ, である. なお, loop 文に関してはその役割から, もう少し抽象的なレベルからのカテゴリ化が可能である.
- 値の交換方法
  大きく 2 つの方法になる. (1) テンポラリパラメータを 1 つ用意する, (2) テンポラリパラメータを 2 つ用意する, である. また, 値の交換に関して, 各ステートメントが構造的に離れた場所にあってもよい, ということを考慮しておく必要がある.

表 2-b では, 実際のプログラミングレベルでの方針の決定を表している.

このように, 意思決定を行なう箇所は, 開発段階のさまざまなところで現れる. そして, 表 2-a 及び表 2-b のような決定を学習者が行なうことにより, 設計指針を明確にすることができる.

### 3−2 詳細化と設計

決定に基づき，段階的詳細化を行なう．実際には表2-a に示した意思決定の,「ソート方法」と「データ構造」が決定すれば，大まかな構造は決まってくる．大まかな構造が決まったあとは，これを段階的に詳細化する．例えば「単純挿入」を実現すると選択した場合，

表 2-c:「単純挿入」に実現すべき処理

- 初期化
- データ入力
- 2 重ループによる比較対象の指示 (2 個所)
- 指示された値の比較
- 値の交換
- 結果出力

表 2-c に示した処理をすべて実現しなければ,「単純挿入」によるソーティングは実現できない．そして,「2 重ループによる比較対象の指示 (loop 文)」「値の交換」には表 2-b で示したような決定がある．他にも，データ入力はファイルから行うのか，それともキーボードからの入力なのか．結果の出力方法はどういうものなのかを決定する必要がある．決定したら，それに基づき，より具体的な処理へと詳細化していく必要がある．つまり，表 2-b の「loop 文」や「値の交換方法」に関する決定は,「ソーティング」を実現するための詳細化の過程でどのように実現するのか決定して行くことになる．

### 3−3 決定の変更と再利用

すでに行なった意思決定を変更すると，どのような影響があるのかを考えていく．

表 3-a:設計方針の変更と再利用

- ソート方法
  同一カテゴリに分類されるソートでは，基本的な設計にかなり共通点が多い．また，改良などを行なう場合も同様である．
- データ構造
  基本的なソートアルゴリズムや設計の変換は行なわなくても良い．具体的なプログラムレベルでの変更が必要になるため，設計自体は再利用できるが，プログラムコードの再利用は難しい．

表 3-b:インプリメント時の
決定の変更と再利用

- loop 文
  基本的には条件文とプレフィックスステートメント (Pre-fix statement:前置処理)，およびポストフィックスステートメント (Post-fix statement:後置処理) の操作で再利用できる．ループのインステートメントシーケンス (In-statement sequence) についてはそのまま利用できるため，新たに作り直す必要はない．
- 値の交換方法
  多くは代入ステートメントの交換ですむ．

ソートについてもうすこし詳しく説明しておくと，ソーティングの場合多くは次の 3 つのカテゴリに分類される[4]．

表 4:ソーティングの種類 (カテゴリ)

(1) 挿入によるソーティング
(2) 選択によるソーティング
(3) 交換によるソーティング

これらのカテゴリ内では，その多くが似たような構造をとることがあり，ケースバイケースで再利用できるステートメントシーケンスが変わってくるので注意が必要である．

実際に指導を行なう場合，以上の情報を具体的に保持している必要がある．再利用が可能な場合は，どの程度再利用が可能であるのかを判断しなくてはならない．逆に変更にともなう影響や再利用についての指導ができれば，学習者は意思決定変更にともなう状態を的確に把握できると考えられる．

## 4. 意思決定と詳細化

以上述べてきたようなキメの細かい指導を行なおうとすると，非常に多くの知識が必要となる．このような知識を獲得する場合には，特に次の点を考慮に入れなければならない．

- 全ての開発段階において，意思決定を意識したモデル・設計・プログラミング知識の収集と分析 (開発者の意図が識別できるような情報)
- 粒度の高い設計プロセスの把握

学習者が行なう意思決定を念頭においた指導を考えるならば，まず意思決定に関する情報を収集・蓄積し

なくてはならない．そのなかでも特に，アルゴリズム設計以降の開発プロセスにおける意思決定を考える必要がある．そして，設計から具体化までのプロセスを切れ目なく考えていくためには，プロセスプログラミングよりはるかに粒度の高いプロセスの把握が必要となる．加えて，指導を行なう場合，ケーススタディの学習のためにも開発者の意図に関する情報を保持しておかなくてはならない．

このとき意思決定・詳細化に関する情報を蓄積・表現する方法として，2 つの表現を考える必要が出てくる．それは**問題解法木**と**設計木**である．

### 4–1 問題解法木

課題からその解であるプログラムに至るまでの知識を表した木構造．同一カテゴリに入る課題 (処理) を，実現方法毎に保存するのではなく，関連付けを行ない同一カテゴリのデータとしてまとめる．これは問題解法木と呼ばれ，解のパスが決まれば，任意のプログラム (またはその設計図) が決定する性質を持っている．

この問題解法木を，特に意思決定と段階的詳細化プロセスを考慮して作成したのが**意思決定木 (Decision tree)**[5]である．

また，可能な限りモデリングなどの情報も保持しておかなくてはならない．

### 4–2 設計木

ステートメントとフローが記述されているものである (以下チャート)．しかし従来のチャートは，それが構造化されたプログラム言語用のものであったとしても，プログラマの明確な意図や方針を入れることは不可能である．これはチャートはかなりプログラムよりの表現方法であり，プログラムの中にプログラマの方針に関わる情報を盛り込むことはまず不可能であるということからも分かる．このことからも実際にはプログラマの意図や方針を取り込めるチャートの必要性がある．

この様なことを配慮し，プログラムの抽象的な構造を表現するのが**プログラム木 (Program tree)**[5]である．

## 5. 意思決定木 (Decision tree)

これまでに述べてきたように，プログラム設計では多くの詳細化と，意志決定 (選択) を行う必要がある．そこで，このような知識を形式的に蓄積する問題解法木として本研究では意志決定木[5]を考案した．

この意思決定木では，プログラムの設計から作成段階における意思決定と詳細化の情報を表現する．意思決定木は基本的には AND/OR 木である．図 2はその様子をあらわしている．ノードには**問題 (Problem)** が配置されている．問題はさらに幾つかの**部分問題 (Sub-Problem)** に分割されている．

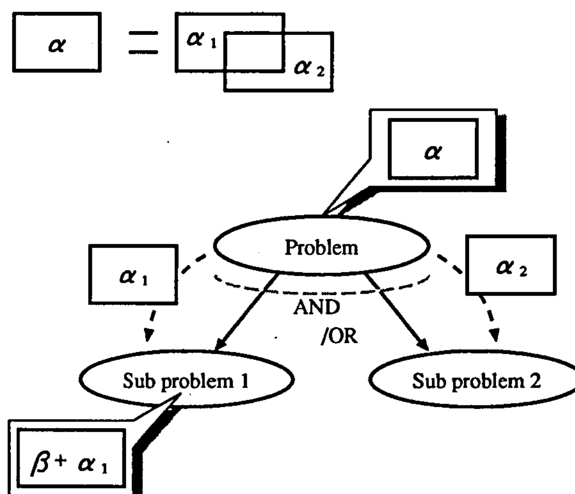


図 2 意思決定木におけるノードと問題分割

問題はそれぞれが設計情報を保持している．図 2では，Problem が$\alpha$という設計情報を保持している．この$\alpha$はそれぞれの Sub problem に対する設計情報の和 $(\alpha_1 + \alpha_2)$ である．Sub problem へ分割されたときに，必要な情報がそれぞれに伝達し利用されることとなる．これにより上位で行った意思決定詳細化の情報が伝播することになる．

このように，問題を部分問題に分割することにより，意思決定と詳細化を表現する．ノードには設計やプログラムに関する情報が保持されており，決定や詳細化に伴い部分問題の情報と合成されてゆく．したがって，意思決定や詳細化が反映したプログラムが作成できる．この時，プログラムに関する情報は主としてプログラム木の形で保持しておく．このため，ある程度意思決定・詳細化を行うことによりプログラムの構造が確定する．*clichés*[6]の形で具体的なプログラムを保持しておくことにより，具体的なプログラムを完成することができる．

また，途中で意思決定を変更した場合に，プログラムがどのように変化するのかだけでなく，過去に行った意思決定・詳細化の情報がどこまで再利用できるのか，また，具体的にコード化を行っていた場合，それがどこまで再利用できるのかを示すことができる．

## 6. プログラム木 (Program tree)

プログラム木[5]は具体的なプログラムの構造を表現する．ただし，抽象的な構造を示す表現も含まれているため，そのままではプログラムへマッピングすることはできない．

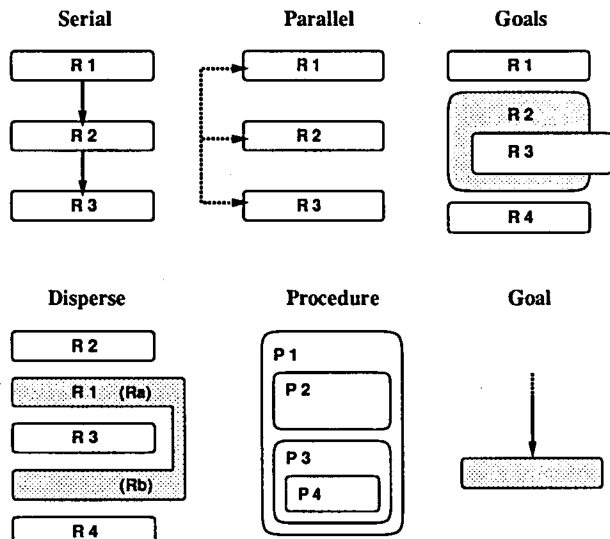


図 3 プログラム木の構造

図 3がその種類をあらわしている．基本的には次の6 つの構造から成っている．

**Serial:** 複数の処理順序が固定している構成を表現する

**Parallel:** 複数の処理の順不同な構成を表現する

**Goals:** ステートメント中にステートメントを定義する構造を表すためのもので，主にループ処理に対応する

**Disperse:** 場所的には分散しているが，実際にはひとまとまりの処理であることを表す

**Procedure:** 手続きを利用する場合に用いる

**Goal:** 具体的なプログラム断片の情報を持つ事を表す

Goal を葉とした抽象的なプログラム構造から構築される木をプログラム木と呼ぶ．特に Goal は具体的なプログラムに関する情報 (プログラム断片) を保持している．厳密に言えば，プログラム木の葉である Goal では，プログラム断片を蓄積 (goal/plan 法) してあり，実際のプログラムにマッピングするときには，その情報が利用されるのである．

この Goal で保持すべきプログラム断片の情報を取り替えることにより，多種類のプログラム言語に対応できる．つまり同一カテゴリに分類される言語，この場合で言えば手続き型言語であれば，プログラム言語毎に Goal 用意しておけば C や Pascal といった言語に対応できるのである．

## 7. まとめ

プログラム開発に関して，開発段階と設計知識，意思決定といった情報がどのようなものであるのかを大まかに説明した．実際にプログラム作成の指導を行なう時には，多少時間や手間がかかっても正しいプログラミングスタイルを学習させる方がよい．2 つの課題を出して 2 つとも適当に作成されるよりは，1 つの課題に対してより工学的に作成するように指導した方が良いと考える．これが最終的には良品質のプロダクトに結び付く．多くの開発手法でも結局は名称と手段が多少異なっているだけで，基本的には従来の設計プロセスを踏襲している．すなわち，少なくとも従来から (良いと) 言われている開発モデルを把握しておかなくては，最新の開発手法の理解も片手落ちになる可能性がある．

本研究では現在以上のような事柄を踏まえた上で，チュータリングシステムの開発を行なっている．

## 参考文献

1) Richard.C.Waters :*Programmer's Apprentice*, IEEE Transactions on Software Engineering, SE-11, no.11, (1985)

2) S. Rugaber and S.B.Ornburn and R.J.LeBlanc jr :*Recognizing Design Decisions in Programs*, IEEE SOFTWARE, Vol 7, No.1 (1990).

3) 手塚 慶一・海尻 賢二 :計算機ソフトウェア, コロナ社, (1987).

4) Niklaus Wirth :*PASCAL* アルゴリズム+データ構造=プログラム, 日本コンピュータ協会, (1979)

5) 鈴木 彦文:初心者プログラマーに対するプログラム設計支援と設計知識, 電子情報通信学会技術研究報告 KBSE96-26 (1997-01).

6) Charles. Rich and Richard.C.Waters :*A Research Overview*, COMPUTER, Vol 21, No.1 (1988).