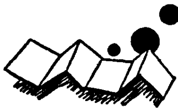
解説



複合設計†

久保未沙††

1. CONSTANTINE と MYERS

複合設計 (Composite Design) は Myers が使った用語である. 構造化設計 (Structured Design) は Constantine, Myers と Stevens が IBM Systems Journal に書いた論文[4]の標題である. 後に Constantine は Youdon と「Structured Design」という題名の本を出したが, 書き始められたのは '70 年の初めである. Constantine は IBM の System Research Institute (SRI) の先生であった (現在は違う) し, Myers はこの SRI で設計法を学んだ. ところが先生の方はなかなか出版に至らない. そこで, 真面目な生徒が先に本を書き, その本の題名に使った用語が冒頭の言葉である. これら手法が IBM の社内で使用されるようになり, 特に System Development Division (OS を開発した部門) で使われ, 後に Improved Programming Technologlies (Techniques ということもある) にモジュール分割の方法として取り入れられた. これの関係をデータフローに模して描くと図-1 のようになる.

二人の設計法は親と子ほども違ってないと筆者は考えている. これを理由にする訳ではないが, 例題を解くときの過程は, どちらの方法とも言えないことをお断りしておく. とは言え細く眺めると違いが目につくこともあり, 2, 3 取り上げ表-1 にまとめる. 性格的違いのためと思えるところもあり差自体をどうこう考える必要もないと思う. ただ弟子にあたる Myers の本が簡潔にまとめられているように思う. この点が差として以外に大きく特徴になっている. 自分流設計法をまとめるうえでこの差を考えることが参考になるかもしれない.

しかし本質的には同じ構想でできた設計法である.

特徴をまとめると次のようになる. ① 設計記法はバブル・チャートとモジュールの階層構造図の 2 種類である. ② 設計手順はバブル・チャートを使いデータフローを追跡し, これらを機能的にグループ分けをして階層構造図を構築する. もう 1 つの特徴が, ③ モジュールの評価基準である. ①, ②, ③ についてはもう少し詳しく後述する. ただ, 複合設計/構造化設計というとモジュールの評価基準が名高く, 設計手順は余り日本では評価されてないように思える. モジュールが簡単に作れる人には確かに設計手順は不要である. しかしこういう人は, かなりの経験者である. ところが設計技法を論ずるのは, 経験のない人にいかに上手く設計してもらい, しかも標準的に実行できるかということである. そのための作り方がまず必要である. こういう意味で設計手順を強調したい.

2. 複合設計による設計

設計の過程を ① 記法, ② 手順, ③ 評価基準の順番に述べていく. 設計はモジュール分割が主体であるため, モジュールの定義をまず述べておく.

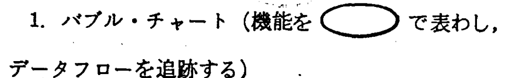
モジュールはプログラム文のグループで次のような特性をもつ.

1. 複数文が意味をもつ群としてまとまっている.
2. 文群は境界識別子 (たとえば, START と END 文) で区切られている.
3. 文群は名前 (モジュール名) によって, まとめてプログラムの他の部分から参照できる.

プログラムは, モジュールの集まったもので, OS から呼び出せるものとする.

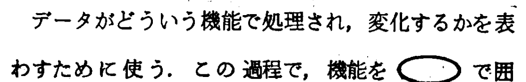
2.1 記法

設計には 2 種類の記法を使う.



1. バブル・チャート (機能を ◯ で表わし, データフローを追跡する)

データがどういう機能で処理され, 変化するかを表わすために使う. この過程で, 機能を ◯ で囲



---

† Introduction to Composite Design by Misa KUBO (Information Systems, IBM Japan Ltd.).

†† 日本アイ・ビー・エム(株)川崎事業所 KW-293 情報開発

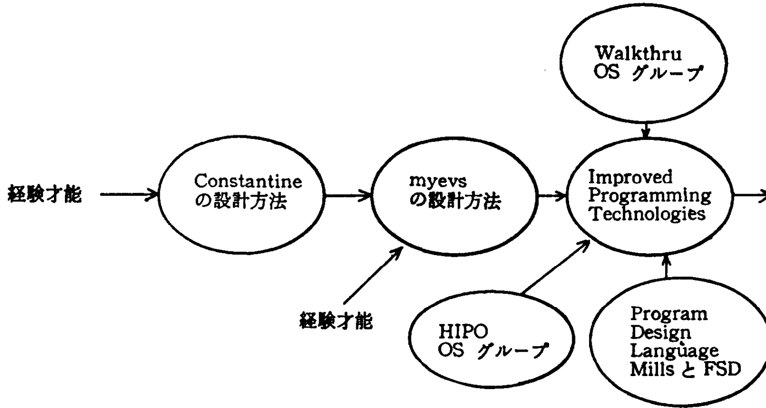


図-1 構造化設計から IPT へ

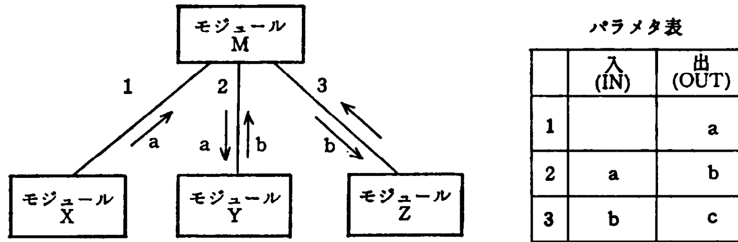
表-1 Constantine と myers の差

		Constantine	myers
記述，モジュール分割のレベル		細く分割，記述	それ程細くない
記法	パラメタの線		
	連結線		
	バブル図	データ 1, データ 2, OP, データ 3, OPerator, *: AND ⊕: OR	
	パラメタの値	a, b, c	1, 2; IN / OUT: 1 — a, b / c; 2 — c / a
用語		Afferent Efferent (生物学)	Source Sink (物理学)
Coupling のタイプ		Data coupling Control coupling Hybrid coupling	Contents coupling Data coupling まで 6 種類

む．この機能である◯から出るデータを⟶で表わし，機能間の相互関係を示す．たとえば，$\xrightarrow{a}$ (X) $\xrightarrow{b}$ (Y) $\xrightarrow{c}$ の図が描けると，データ'a'は(X)という機能により，'b'に変わり，(Y)の機能でさらに'c'になる，と読む．

2. 階層構造図（プログラム設計後のモジュール間従属関係を示す）

図-2 のモジュール M はモジュール X, Y, Z の機能に分解され，X, Y, Z は M の従属モジュールとなって



パラメタ表

	入 (IN)	出 (OUT)
1		a
2	a	b
3	b	c

図-2 モジュールの階層構造図

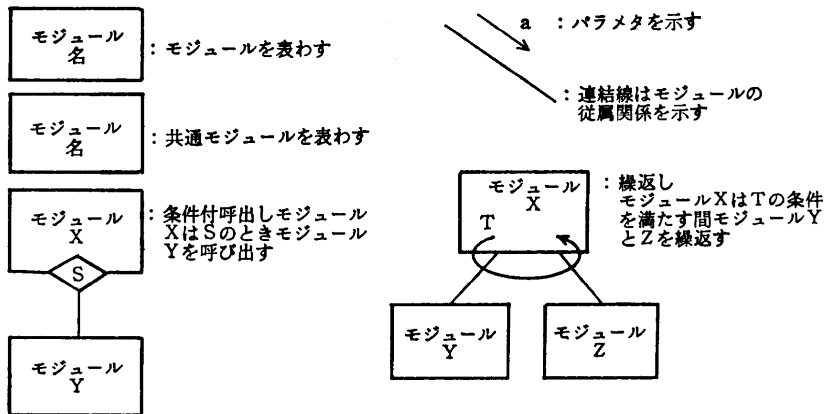


図-3 階層図の記法

いる. MとX, Y, Z の間にはパラメタ a, b, c が受渡しされる. これを図-2 モジュールの階層構造図として示す. この図にはパラメタも併せて描く. このとき ↗aとして直接に図の中に描いてもよいし, 図-2 のパラメタ表のようにまとめて書いてもよい. モジュール間のパラメタのやり取りが直接解るという点では前者の方がよいが, パラメタの受渡しの個数が多くなると図に書き込みにくくなり後者の方がよい.

この階層構造図を描くときに使う記法の中で代表的なものを図-3 に挙げる. 記法を細く区別して描きたい人は, 参考文献1)を参照されたい.

2.2 設計の手順

現在の適用業務システムは複雑で, 1個のプログラム作成に目を向けていただけでは設計できない. 特にオンライン・システムが普通になってきた現在, システム全体の流れを考え, 個々のプログラムの設計を考えるべきである. それには, まずシステムに何が入って来るかを考える必要があり, この設計のためにトランザクション分析法を述べ, 次に個々のプログラムを解析するために変換 (Transform) 分析を述べる.

2.2.1 トランザクション分析法

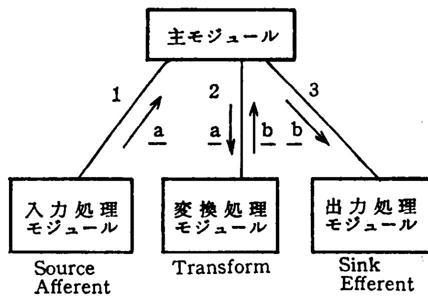
トランザクション分析法は, システム全体を分析する, ① トランザクション・センタの役割を定義することと ② トランザクションの解析することとに分かれる.

➡ トランザクション・センタの役割として次のことを定義する.

- トランザクションを受取る.
- トランザクションのタイプを決める.
- トランザクションをタイプ別処理に送り出す.
- 各トランザクションの処理を完了する.

➡ トランザクションの解析をする.

- どこからトランザクションが来るかを識別する.
- トランザクション中心のシステム構成を組み上げる.
- トランザクションのタイプを識別し行動を定義



		入	出
1	源泉モジュール	稀	有
2	変換モジュール	有	有
3	吸収モジュール	有	稀

パラメタの存在

図-4 従属モジュールの特徴とパラメタの存在

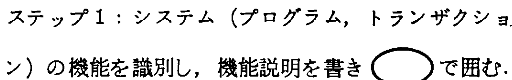
する.

● トランザクションごとに，これを処理するモジュールを規定する.

● 各トランザクション・モジュールを下位モジュールに細分する.

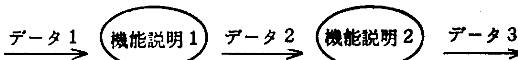
2.2.2 変換分析法

複合/構造化設計がデータフロー型設計と言われるのはこの分析法が，データフローの解析に基づいているからである.

ステップ1：システム（プログラム，トランザクション）の機能を識別し，機能説明を書き  で囲む.

これをバブルという.

ステップ2：機能の中で変換されていくデータを追跡し，関連するバブルを矢印で結ぶ.



これがバブル・チャートである.

ステップ3：機能をグループに分ける．第一レベルのグループ分けでは入力処理部分，変換部分，出力処理部分に分けていく．詳細化が進んだレベルでは個々の機能の関連性からグループをまとめることもある.

ステップ4：グループをモジュールにして階層構造に描く．入力処理部分はシステムにデータを提供する源であるということで源泉 (Source, Afferent) モジュールと呼ぶ．変換部分は入力を出力に変換する機能をもつので変換 (Transform) モジュールの性質をもつ．そして出力処理部分はデータを受け取り，システムの外に出して終るため吸収 (Sink, Efferent) モジュールの性質をもつ．このとき，① バブルの中に書いた機能説明をモジュール名に変える．② パラメタでデータの変化を記述する．これをまとめて図-4 に書く.

ステップ5：設計の停止の条件を満たしていないならば，ステップ1に帰り，より細かい機能に細分化する．停止の条件：これ以上細分化できないモジュールとして機能が表現されるとき詳細化を終る.

2.3 モジュールの評価基準

モジュールの階層構造を作る過程で，より良いモジュールをつくるために，モジュールを評価し，再評価

表-2 モジュールの尺度

凝 集 度

凝集度		特 徴
弱	論理的	1つのモジュールに複数機能を持たせたもの．パラメタも多くなり，制御要素（スイッチ等）が多くなる.
	時間的	論理的凝集度に時点によるまとまりを附加したものである．初期設定時の OPEN 文，終了時の CLOSE 文などを1カ所にまとめて書くときがあるが，これがこの代表である.
強・良	機能的	1つのモジュールに1つの機能のみを遂行するために各文が存在する．モジュールとしての凝集力は一番強い形である.

結 合 度

結合度		特 徴
強	共通外部	変数名がパラメタを通さず複数モジュールで使用できるもの．参照すべきでないモジュールもこれら変数を使用できてよくない.
	制御	パラメタとしてスイッチ変数などを受渡すモジュールをいう．このスイッチ変数によりモジュール実行の可否を決める場合は受側のモジュールは複雑になり独立性が弱くなる.
弱・良	データ	パラメタとしてデータ値（制御データでない）のみを受渡す．そのため独立性が高く，呼び出すモジュールをブラックボックスとして使える.

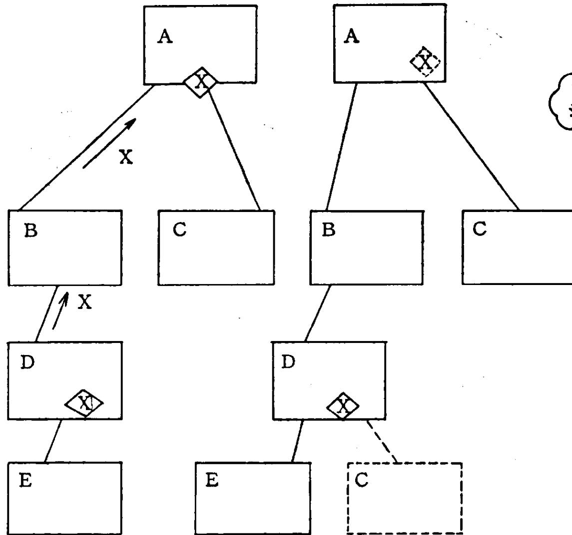


図-5.a　決定構造　　　図-5.b　修正した決定構造

するための基準である．この基準の代表的なものに，モジュール内ステートメントの関連度を示す尺度を測る凝集度，モジュール間の関連を示す尺度として結合度を使う．これらの尺度の中で代表的なものを表-2にしておく．詳しくは参考文献2) を熟読されたい．

凝集度や結合度にくわえて，モジュールを調整するために役立つ基準がある．モジュールの大きさとか，パラメタの個数，入力/出力の隔離などがある．ここでは一番使われると思われる'決定構造'について述べる．

決定構造とは判定結果の影響が下へ及ぶような構造を良しとする考えである．図-5.aでモジュールCはXの判定により実行すべきか否かが決まるものとする．これは決定構造の方針に合っていない．なぜならモジュールCはモジュールDの従属モジュールでないのにXの判定が及んでいるからである．そのためモジュールDは判定の結果Xを制御標識（制御結合による）としてBに返さなくてはならない．Bはこれを Aに返さなくてはならない．解決策としては，判定Xの位置を分析しAでは不可能かを考える．またはCをDの従属モジュールにできないかどうかを考える．この形を図-5.bで示しておく．

3. 例題を設計する

例題を解く前に，前提条件を追加しておく．① データ構造は完全に決定されていない．② プログラムの形態も特に指定されていない．③ コンテナ数を最小

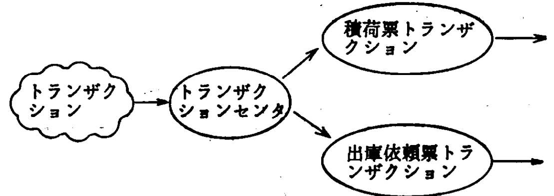


図-6　トランザクション・センタのデータフロー

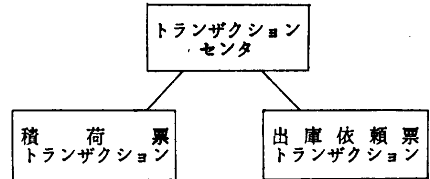


図-7　トランザクション・センタの構造図

にすることは1つの基準であっても絶対ではない．また，設計の途中で必要があれば適宜に条件は追加するものとする．

このシステムはトランザクションとして端末から指令が来るとして，まずトランザクション分析法を行う．

段階1：トランザクション・センタの役割を考える．

トランザクション・センタでは，積荷票トランザクションと出庫依頼票トランザクションのタイプに分ける．システムとしての形は次の構造，図-7になる．

段階2：トランザクションの解析

段階1でトランザクションのタイプは2つあることが解った．またトランザクション・センタとしてのプログラムはオンライン・システムとしてすでにあるとすれば，適用業務プログラムとしてはこの2つのタイプのトランザクションを解析すればよい．これらのトランザクションの解析は，変換分析法により行う．

A．積荷票トランザクションの解析

ステップA.1：機能の洗い出しをする．

図-A.1

ステップA.2：データフローを示す．

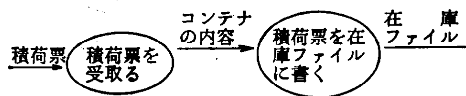


図-A.2

ステップA.3: 機能のグループ分け.

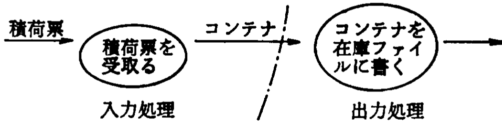


図-A.3

ステップA.4: 階層構造に描く.

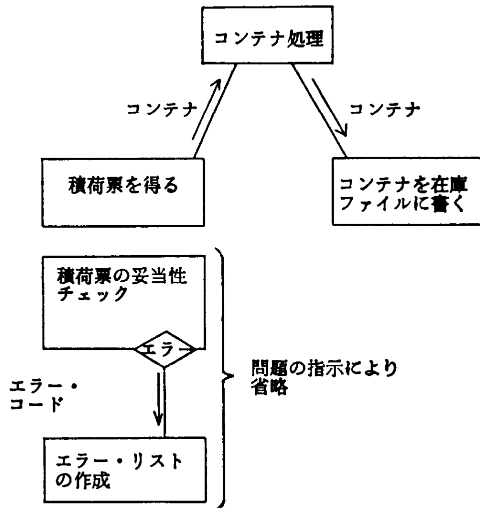


図-A.4

ステップA.5: 停止.

積荷票トランザクションのモジュールとしては, 単一機能になると考えられるので, ここで, このトランザクションのモジュール分割は止める.

B. 出庫依頼票トランザクションの解析.

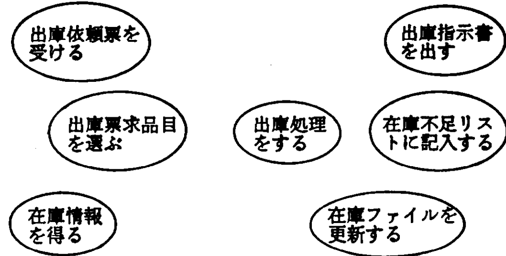


図-B.1

ステップB.1: 機能の洗い出しをする.

一番始めのレベルでの機能の洗い出しで, どの程度に細く機能を描くかが問題である. それは1つには仕様書にどれ位詳細に書き込まれているかということ, そして設計者にどんな経験があるかということで異なるからである. ただ手順としては, ステップ 2, 3, …, 4, があり, 次のレベルでまた各ステップがありそこで精練していけばよいわけで, 余り厳しく考え込む必要はない. 大体7個から10個位にまとめるつもりで機能を書き出して行けばよい. ただ機能記述は後にモジュール名に使うため解りやすく書く.

ステップB.2: データフローを矢印で関連性をつける.

ステップB.3: 源泉 (Source), 変換 (Transform), 吸収 (Sink) の機能に大きくグループ分けする. 参考文献2) では中央最大抽象点を決定しこれを起点に分けると記述されている. 詳しくはそちらを参照されたい. しかし, 筆者は具体的分け方で次のようにする. ① 出力処理に関係するもの, ② 入力処理に関係するもの, ③ どちらにも入らないもの, という思考で分けていく. これを**図-B.3** に示す.

図-B.3で入力/変換/出力の3部分に分ける境界は,

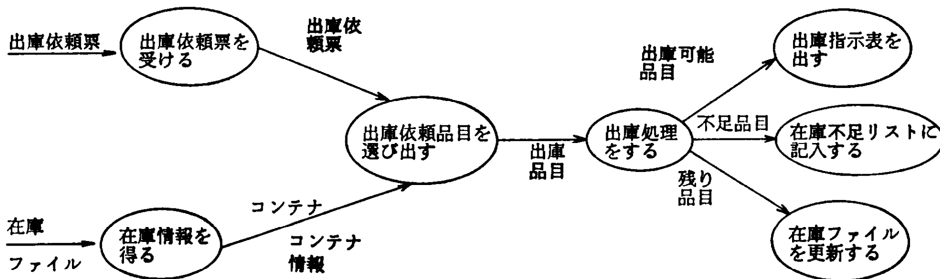


図-B.2

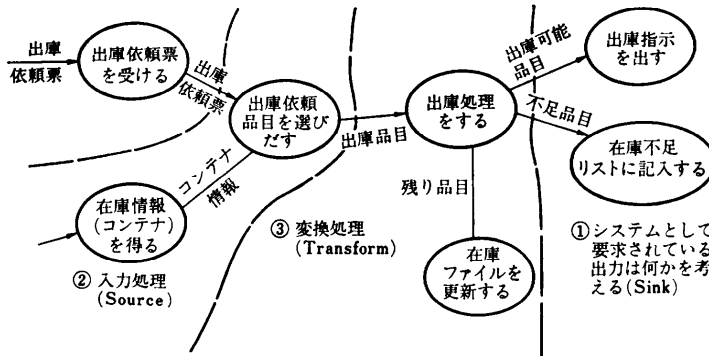


図-B.3 機能のグループ分け

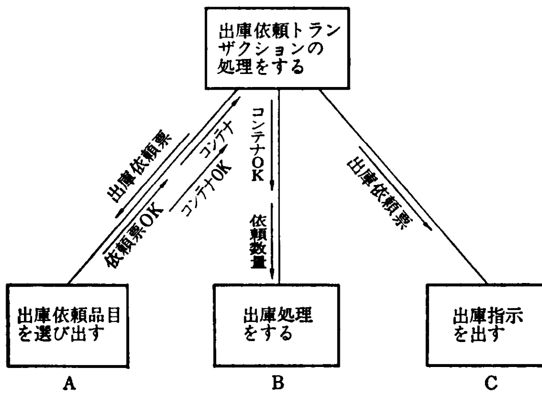


図-B.4 第1レベルの階層構造図

システムとして〝どういう機能を持つ〟かということで決める．この例題のシステムの機能は，〝依頼品目の注文に応じた出庫処理をする〟という解釈にした．そのため在庫から依頼品目を選び出す過程は，例題の機能に入力を用意するという機能部分として分けた．出庫依頼品目を選択する機能は，システムへの入力をつくる部分と判断したためである．当然他の解釈もあり〝出庫依頼票を受ける〟機能のみが入力処理部と考える人もある．

ステップB.4：階層構造に描くステップで，これを図-**B.4**に示す．これは第1レベルの分割であるから，源泉，変換，吸収の役割のモジュールがはっきり存在する．レベルが下がるにつれ，3つの機能がすべて出て来るとは限らない．この3つの機能で表現するのは，機能には入力があり結果を出力していることに基づいている．すなわち，変換にいたる前で入力を得る何らかの処理があればまとめる，また結果の出力も，何か出力に手をほどこしてないかを見極め，各々をモジュールにして，階層とする．各モジュールが決まるとこれに記述文を入れ，パラメタを記入する．

ステップB.5：ステップの繰返しをする．図-B.4の階層構造図では，まだモジュールが1つの機能のみを実行する単位とは言えず，更に分割の必要があり，変換分析法のステップを繰返す．ステップの番号の表示を以下 X. A. …Z. n と使う．Xはトランザクションのタイプ（A, B），AからZは，階層構造図の連結子に左から右につけた記号とする．nは変換分析法のステップ番号とする．またステップ1，ステップ2，ステップ3はバブル・チャートが基本になり，これをステップ1で描くと，ステップ2，ステップ3は発展させるだけであるから，次の様にまとめて記述する．

ステップB. A. 1：〝出庫依頼品目を選び出す〟のモジュールの機能を更に細く洗い出し，⬭ のバブルを描いていく．

ステップB. A. 2：機能の中のデータフローを追跡する．⟶矢印でバブルを結ぶことでこのステップを表わす．

ステップB. A. 3：機能(バブル)をグループに分ける．この3ステップを図-**B. A. 3**にまとめて描く．機能分けで〝○○を得る〟といった機能記述は単純に入力処理機能，〝○○を書き出す〟というと出力処理機能と考えるのは少し早計である．考えるときは，今分割しようとしている〝機能〟を遂行するために，どの機能を分担しているかを考える．そのため機能のグループ分けで間違いやすいグループ分けを・—・—・—・で示し，最終的にグループ分けしたところを— — — —で示す

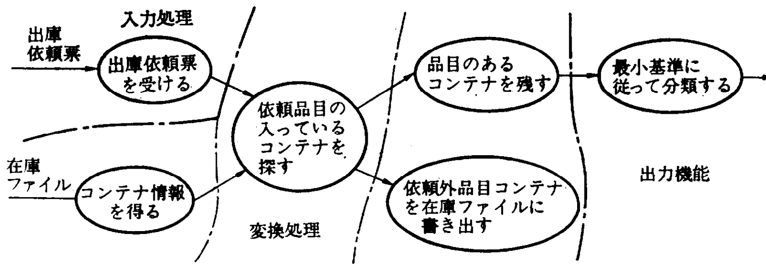


図-B.A.3 バブル・チャートと機能グループ

ことにする．'コンテナ情報を得る' ことは，確かに入力処理であるが，'出庫依頼の品目を選びだす' 機能としてはどういう機能が起動点となり，最終的に欲しいものは何かを考えると，入るべきグループは決まる．

ステップ B.A.4：階層構造図を描く．

この構造では，'コンテナを最小にする基準に従って分類をする' モジュールをこの位置におくと，該当コンテナが有ったかどうかの標識を上位モジュールに返さなくてはならない．そのため，これは該当品目のコンテナの従属モジュールとする．

図-B.A.4 の該当品目のコンテナを選ぶモジュールは，在庫ファイルにあるコンテナを1つずつ読み，依頼品目があるかどうかを確認する．依頼品目が入ってないコンテナはすぐ在庫ファイルに書き出す．これで，図-B.A.3 の変換処理の部分は，該当品目コンテナを選ぶモジュールとして全体をまとめ，個々の機能はその従属モジュールとして構成する．これを図-**B.A.B.4** に描いておく．この中で '該当品目を残す' モジュールは親の中に吸収させる．

親の中に吸収させるのは，ステップ数が少なく，モジュールとして独立させたくない時に行う．

図-B.A.B.4の階層図を実際に描くかどうかは，文書化のレベルによる．コンテナ情報を得るというモジュールは，高水準言語などでは READ ステートメント1つで済むかもしれない．そのためモジュールとして分割するには図-B.A.4 のレベルで充分だと考える．

出庫依頼票を受けるモジュールは，一般に妥当性チェックの従属モジュールがつながる．ここでは省略する．

出庫処理をするモジュールを更に分割する．

ステップ B.B.1：'出庫処理をする' の機能の洗い出し．

ステップ B.B.2：データフローを追跡する．

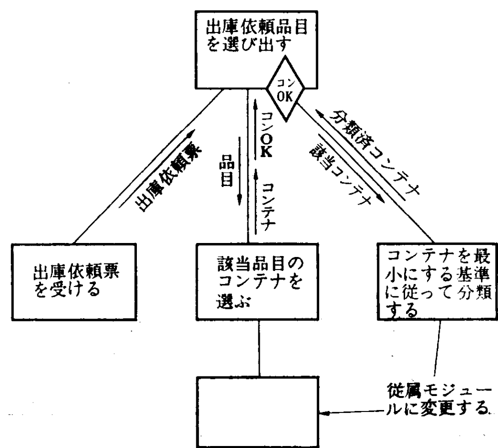


図-B.A.4 出庫依頼品目を選ぶ構造

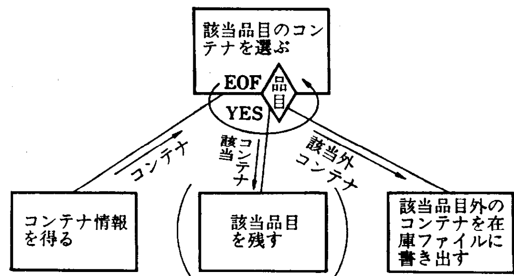


図-B.A.B.4 該当品目のコンテナを選ぶ構造

ステップ B.B.3：機能のグループ分けをする．

3つのステップをまとめて図-**B.B.3** に描く．

ステップ B.B.4：階層構造を描く（省略．全体構造図図-8 出庫依頼トランザクション処理を参照)

1つの階層構造にまとめるとき，次の条件を加える．

- 評価基準の結合度，凝集度を確認する．
- 決定構造で制御データを受渡ししないようにする．

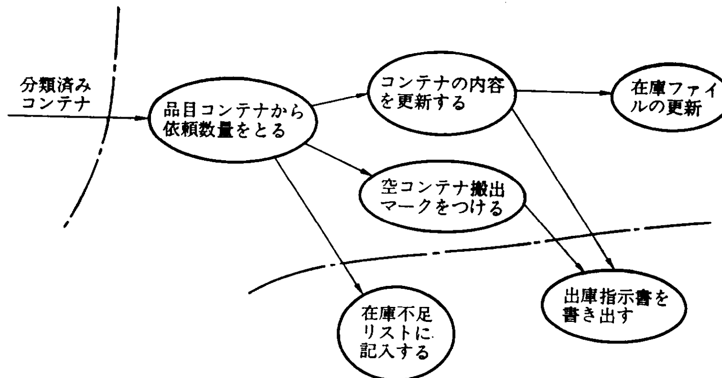


図-B.B.3 '出庫処理をする' の機能の細分化

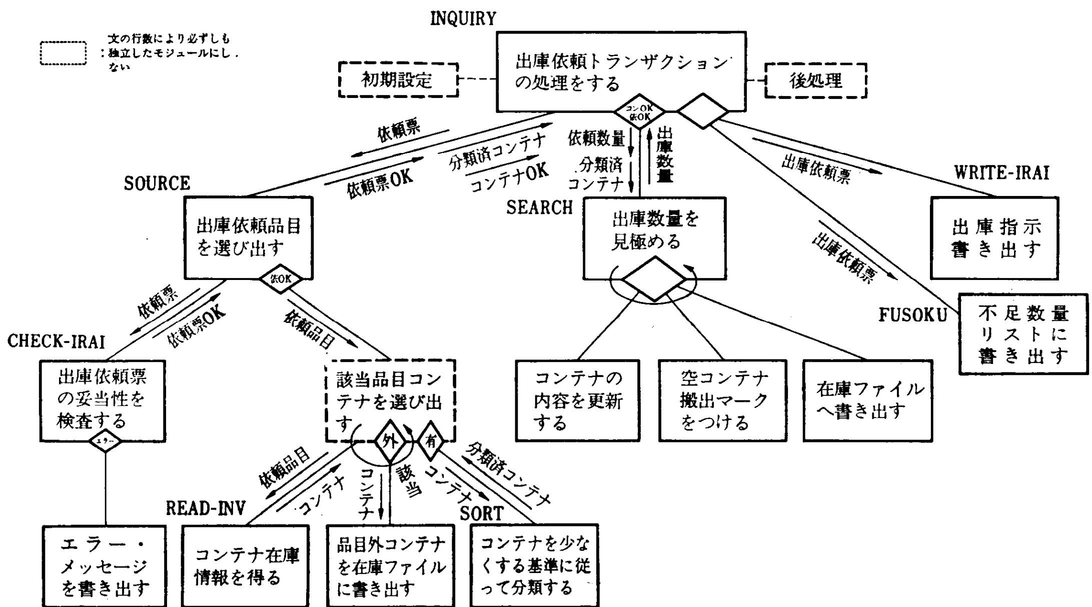


図-8 出庫依頼トランザクション処理

o 初期設定モジュール，後始末モジュールを確認する．

o モジュールの説明を書くときは，モジュール名に使えるように書く．または箱にモジュール名を追加する．

o 各モジュールへのパラメタを確認する．モジュールのプログラム論理を考えながら推敲すること．

4. コーディングへの橋渡し

トップダウン・プログラミングを考えるとき，上位モジュールが出来ればすぐコーディングするのか，全モジュールが分割できてからコーディングすべきか，

```
INQUIRY : PROCEDURE (PARM) OPTIONS (MAIN);
% INCLUDE DCL;
% INCLUDE INIT;
  CALL SOURCE (IN_IRAIFL, IRAI_OK, TBL_
  CONT, INV_OK);
  IF IRAI_OK THEN
    IF INV_OK THEN
      DO;
        CALL SEARCH (IRAIQNT, TBL_CONT,
        XIRAIQNT);
        IF IRAIQNT=XIRAIQNT THEN
          CALL WRITE_IRAI (IN_IRAIFL);
        ELSE
        DO;
          TEMP=IRAIQNT;
          IRAIQNT=XIRAIQNT;
          CALL WRITE_IRAI (IN_IRAIFL);
          IRAIQNT=TEMP-XIRAIQNT;
          CALL FUSOKU (IN_IRAIFL);
        END;
      END;
    ELSE
      CALL FUSOKU (IN_IRAIFL);
  ELSE;
% INCLUDE FINE;
END INQUIRY;
```

図-9 主モジュール

```
SOURCE : PROCEDURE (IN_IRAIFL, IRAI_OK,
TBL_CONT, INV_OK);
% INCLUDE DCLSRC;
  CALL CHECK_IRAI (IN_IRAIFL, IRAI_OK);
  IF IRAI_OK THEN
    DO;
      NAME=IN_IRAIFL・PRODNAME;
      CALL READ_INV (NAME, TBL_CONT, INV_
      OK);
      IF INV_OK THEN
        CALL SORT (TBL_CONT);
      ELSE;
    END;
  ELSE;
END SOURCE;
```

図-10 源泉処理モジュール

問題になる．分割したモジュールと必要なパラメタの定義を見極めきれるときは，上位モジュール分割後すぐコーディングできる．

とにかく主モジュール 'INQUIRY' からコーディングを始める．図-9 に示す．'% INCLUDE DCL;' 文は別のところでコーディングした文をとり入れる文である．DECLARE 文や，初期設定文をまとめて定義したものを使う．

また複合設計による階層構造図から間に何の手段も入れずすぐコーディングできるかどうかは，その時の状況により異なる．次のことが考えられる．

```
SEARCH : PROCEDURE (IRAIQNT, TBL_CONT,
XIRAIQNT);
% INCLUDE DCLSCH;
  ICTR=DIM (TBL_CONT, 1);
  SRCH 1: DO II=1 TO ICTR;
    TCONT=TBL_CONT (II) ・CQUANT (1);
    IF IRAIQNT<=TCONT THEN
      DO;
        TBL_CONT (II)・CQUANT(1)=
          TBL_CONT(II)・CQUANT(1)-IRAIQNT;
        OUT-CONTFLN=TBL_CONT (II),
        BY NAME;
        XIRAIQNT=XIRAIQNT+IRAIQNT;
        IF IRAIQNT=TCONT THEN
           EMPTYFLG=EMPTYFLG-1;
        IF EMPTYFLG¬=φ THEN
           WRITE FILE (CONTFLN) FROM(OUT_
           CONTFLN);
        ELSE
          WRITE FLE (NULLFL) FROM (OUT_
          CONTFLN);
        LEAVE SRCH 1;
      END;
    ELSE
      DO;
        TBL_CONT (II)・CQUANT (1)=φ;
        OUT_CONTFLN=TBL_CONT (II),
        BY NAME;
        XIRAIQNT=XIRAIQNT+TCONT;
        IRAIQNT=IRAIQNT-TCONT;
        EMPTYFLG=EMPTYFLG-1;
        IF EMPTYFLG 7=φ THEN
           WRITE FILE (CONTFLN) FROM (OUT_
           CONTFLN);
        ELSE
           WRITE FILE (NULLFL) FROM (OUT_
           CONTFLN);
      END;
    END SRCH 1;
  END SEARCH;
```

図-11 変換処理モジュール

① 分割したモジュールの大きさ．

② コーディングに使用する言語．

分割されたモジュールの大きさが 50 文位の大きさで，高水準言語を使用してコーディングするのならば，モジュール分割からすぐコーディングができる．しかし，設計とコーディングを別の人にやらせるとか，一定の形にコーディングしてほしいときは，複合設計から，コーディングの間に，HIPO 図や Program Description Language で記述したものを入れる．この例題では，モジュールは小さく分割してあるので，直接コーディングできるレベルである．参考として，源泉処理モジュールの '出庫依頼品目を選び出す' と，変換処理モジュール '出庫数量を見極める' のコーディングを挙げておく．

5. 複合設計の特徴

筆者の属する部門の1プログラムの大きさは250文である．それに基本的な合計や組合せプログラムでは土台になるプログラムを修正するだけであるため，誰も設計などを行わず，直接コーディングをしている．ただ新しいテーマのプログラムで，かなりの大きさになるプログラムになると，何らかの形でプログラム設計を行い，プログラムの構造を明にする必要がある．ところがどの手法で設計しろと決められてないため，各種各様の設計文書が残っている．これらを見ると，設計と文書化の実行は ① プロジェクト・リーダの強制力と分析力，② プログラマの性格と好みで決まるようである．こういう状況の中で経験の長い優秀といわれるプログラマのシステム流れ図は，複合設計で特徴づける機能に分割されている．彼らは多分，複合設計の勉強はしていないと思われるけれども上手く行っている．逆に新人はプログラマ教育後でも上手く機能分割できない．新人は機能分割の後，機能グループにまとめ，従属モジュールを構成する過程が理解しにくいようである．モジュールを機能による従属関係をつけることと，モジュールを実行する手続の流れとを混合させることが原因の1つである．この設計法の利点，欠点がここらあたりにあるように思う．これら自分の経験したことや，一般的に言われているこの設計法の特徴をまとめとして挙げておく．

欠　点

o　よく言われるのは，厳密な設計過程が規定されていないため，人が変わると異なるモジュール構造になるのではないかということである．そのため適用業務の形態で，その標準形を設定しておく方がよい．

o　経験のない新人には入りにくく，使いにくい．自習書だけでは会得しにくいのではないか．

利　点

o　欠点と裏腹のことで，厳密でないことは，応用分野が広く限定されない．プログラム以外の分野でも，データフローから機能別階層構造が構成できる．

o　モジュール分割にはやりやすい技法である．

o　モジュールの階層構造からプログラムの機能や特質が理解しやすい．

o　データ構造の制約は受けにくい．モジュール分割をする段階で，プログラムの論理を考えながら，モジュール間で受渡されるパラメタを記述していく．

o　細かい規則がなく覚えやすい，というより経験のあるプログラマは自然とやっているモジュール分割の手法である．

他の設計技法の導入

プログラミング言語も設計技法も何種類もあり，プログラマはこれらの何種類かを学ばなければ，良いプログラムが書けないのだろうか．1つの設計技法であらゆる設計ができないのが設計の設計たる由縁だろうか．それならば上手く融合できないだろうか．そして欠点といわれる個所を改良していけないだろうか．等等考えて例題にとりかかったが，日本語の中に外来語をカタカナで組み入れるようにはいかない．複合設計はモジュール分割のために，これから先はジャクソン法や，ワーニエ法でと設計法を使えばよいものだろうか．既存の設計法を包含した新しい概念で見直さなくてはならないのかもしれない．とにかくこのシンポジウムがこれら設計法革新の契機となればよいと考えている．

参考文献

1) Edword Youdon/Larry L. Constantine: Structured Design, PRENTICE-HALL (1979).
2) Glenford, J. Myers: Reliable Software through Composite Design, MASON/CHARTER PUBLISHER (1975).
高信頼性ソフトウェアー複合設計，久保・國友訳，近代科学社 (1976).
3) Glenford, J. Myers: Composite/Structured Design, VAN NOSTRAND REINHOLD(1978).
ソフトウェアの複合/構造化設計，國友・伊藤訳，近代科学社 (1979).
4) Constantine, L. L., Myers, G. J. and Steven, W. P.: Structured Design, IBM Systems Journal, Vol. 13, No. 2, pp. 115-135 (May 1974).

(昭和59年7月25日受付)