ソフトウエアにおけるモジュール化

大筆　豊

1. はじめに

コンピュータの応用が広がるにしたがいソフトウエアの需要が増大するとともに，複雑化，巨大化の傾向にあり，品質の良いソフトウエアをいかに効率よく作成・保安するかが重要な課題になっている．ソフトウエア開発には，要求定義・設計・プログラミング・テスト・保守というライフサイクルの各工程を通して行なわれるが，いずれの作業も創造的な頭脳労働にもとづくものである．ハードウエア生産の場合，設計（抽象表現）が最終的には製品(物)の形に具体化され，対応関係が明確であるが，ソフトウエアの場合は，各工程ごとに情報の詳細化，あるいは追加が行なわれることから対応関係も曖昧になりやすい．特にソフトウエアは，頭脳労働の成果物ということもあり，同一の問題に対して，解が一意的に決まらないこと，換言すると，できあがったソフトウエアの良し悪しが，技術者の能力に大きく依存することになる．

最近では数百万ステップという巨大で複雑なソフトウエアシステムの開発もまれでなくなり，多数の技術者による共同作業をいかに管理するかということも重要な問題になっている．複雑な問題を解説するのに，全体を小さな部分に分割し，各々の小問題の解決に置き換えるという，モジュール化の概念は普遍的なものである．

おおふで　ゆたか
㈱東芝　システム・ソフトウエア技術推進部
〒105 港区芝浦1—1—1

2. モジュール化の背景

プログラムを作成していくとき，同じパターンのプログラムコードの列がいろんなところに現われることがある．このコード列に名前をつけて登録し，以後その名前を書くだけで，そのコード列を持ってこれるようにしたのが，マクロプロセッサである．名前だけでなく，コード列の中の可変部分（引数）を指定し，部分的な変更にも対処が可能になっている．多くのアセンブラにマクロ機能があるばかりでなく，FORTRAN, COBOLといった高級言語にもプリプロセッサとして取り入れられている．マクロプロセッサの意図は，同一パターンの記述を容易にすることにあり，あくまで「文字づら」としての同一性を前提としており，一連のコード列がどんな「機能」を持っているかには関わりがない．

FORTRANに代表される高級言語では，サブルーチン（あるいは相当するもの）が実現されており，ある特定の機能とサブルーチン名との対応により，サブルーチン名を記述することだけでその機能の実現が保証される．マクロとの違いはメモリ上の節約（マクロでは，呼ばれるたびにコード列がそのまま展開されるが，サブルーチンでは単にサブルーチンを呼び出すコードが発生するだけ）という実現上の違いの他に，プログラムの設

© 日本オペレーションズ・リサーチ学会．無断複写・複製・転載を禁ず．

計の仕方にも大きく影響を与えている．マクロの場合，プログラミングの過程でたまたま同じパターンが現われるのを便利にしようとする立場をとるのに，サブルーチンの場合，あらかじめ設計段階でサブルーチンとして区別しておこうとする立場を取っている．どういうサブルーチンを切り出すかがソフトウエアシステム全体の機能・性能・保守性に大きく影響を与える．これがまさにモジュール化の問題である．

よく使われそうなサブルーチンをあらかじめ作っておき，機能ごとにまとめて提供するのが，サブルーチンライブラリあるいはパッケージの考え方である．数値計算や統計計算，グラフィック処理のライブラリ，入出力の標準ルーチンなどオペレーティングシステムであらかじめ提供されるものも多い．事務処理で良く使われる，ファイルアクセス（マージ・ソート）や帳票出力のルーチンを充実，整備し，それらの組合せだけで必要なプログラムの開発も可能になっている．この考え方を進めると，アプリケーションごとのサブルーチンを蓄積し，その組合せだけでプログラムを開発するという，部品化プログラミングが可能になる．もちろん，プログラム部品としてのサブルーチンをいくら集めても，その組合せ方をいかにするかの課題は残っており，これは，与えられた問題をどのように分割するかのモジュール化の課題に関係する．

ソフトウエア設計の段階では，与えられた問題をどのようにモジュールの構造として分割するか(Programming in the large) と，モジュールとして切り出されたものをいかにプログラムとして実現するか(Programming in the small)に分けられる．後者については構造化プログラミング[1] (Structured Programming) や段階的詳細化法[2] (Stepwise refinement) の手法が提唱されており，プログラムの制御3構造（順次，反復，分岐）あるいはGOTO lessプログラムといった具体的な指針によりプログラム作成に大きな影響を与えてきた．前者についても，抽象データ型，情報隠蔽，機能分割等のモジュール分割のための基準が提唱されている．次章ではこれらの考え方の代表的なものを紹介する．

3. モジュール化の基準

3.1 目的

なぜモジュール化するかという単純な問いかけを行なうことから始める．数十ステップのプログラムなら下手に分割するより一気にプログラム化する方が良いだろう．しかし何百万ステップもあるプログラムを1本のプログラムで実現することあるいは1人で作成することは不可能である．すなわち，問題を分割し，多数の技術者の共同作業を可能にすることがモジュール化の第1の目的である．ソフトウエアシステムは，開発され運用に移されても，機能の拡張や変更，あるいは開発時には見つからなかった不具合点の修正が，いわゆる保守という名のもとに引き続き行なわれる．保守作業を適確にすばやく行なえるようにすることがモジュール化の第2の目的である．すなわちモジュール化の目的は，巨大で複雑なソフトウエアシステムの開発と保守を可能にすることである．

この目的を達成するため，各モジュールは次の性質を有する必要がある．

- 単純性：各モジュールは明確な目的を有し，単独の機能を実現する．同一モジュールに複数の機能を含ませると複雑になり，理解性に欠けることになる．
- 独立性：他のモジュールに対する影響を少なくするため，モジュール間のインターフェースを極力少なくする．また，共通データのアクセスなど予期しない効果の表われる可能性をなくす．
- 階層性：モジュール間の関係はトップダウンの階層構造にまとめられる．

これらの性質を有することにより，分担開発を可能にするとともに，インターフェースの変更をともなうことなく機能拡張や修正ができる，各モ

© 日本オペレーションズ・リサーチ学会．無断複写・複製・転載を禁ず．

ジュール間の関係が明確になり，システムの理解性が増すことになる．

3.2 モジュール分割の指針

次に，どうすれば上記のような性質を持つモジュール分割ができるか，あるいはモジュール分割のための手順があるかについて考えてみよう．個々のモジュールが特定され，その実現を行なう場合，構造化プログラミングの制御3構造のみで行なっても，性能を含めてほとんど問題が起こらないことが経験的にわかっており，特定の枠組みのみで設計する標準化の効果が大きい．モジュール分割という，抽象レベルの高い上流工程では，単一の設計基準（原理）だけで行なうことが良いかどうかに疑問がある．システムの規模，目的，性質に応じて選択すべきである．

• データの抽象化［3］：いろいろのモジュールから共通データに直接アクセスすると，共通データの変更にともない，関係するモジュールをすべて変更する必要が生じる．データを直接各モジュールに開放せず，データに対する操作（オペレーション）の集合として，ひとまとまりのモジュールを定義する．こうすると，データの変更があっても，関連する操作モジュールの変更だけですませるようになり，変更に強い設計が可能になる．たとえば，ファイルに対するアクセスを考えるとき，各モジュールから直接ファイルとの入出力を行なうのではなく，レコード単位の入出力を操作する抽象データ型モジュールを経由することにより，デバイスの変更に容易に対処できるようになる．直接データにアクセスするのに比べて，中間に共通のオペレーションが介在するので，性能は必然的に低下するが，将来の変更を考慮するとデータの抽象化を行なう利得は大きい．特に，プログラム内で使用する各種のテーブル，入出力機器に依存する部分，オペレーティングシステムに依存する部分，ファイルへのアクセス，主要データへのアクセス部分はデータの抽象化を行なえば効果が大きい．

• 情報の隠蔽［4］：print-message（“入力データに誤りあり”）という文を考えてみよう．これはprint-messageというモジュールの“入力データに誤りあり”という引数による呼び出しを表わしている．将来，メッセージを英語に変更しようとすると，print-messageを使っているすべての部分を見直す必要がある．print-message（1）のように，メッセージの内容を直接見せないで，print-messageというモジュール側で番号により選択させるようにする．こうすると，英語への変更も，モジュール側の対応したメッセージの変更だけですますことが可能となる．print-messageモジュールの実現方法は，前述のデータ抽象化モジュールと同様であり，メッセージ群というデータの扱いをprint-messageという操作のみで定義することになる．

• 強度と結合度［6］：1つのモジュール内の機能間の関係を分析したとき，同一データを参照するなど関係が強いほど強度が高いと呼び，同一モジュールにまとめる必然性が高い．逆に無関係な複数機能があるとき単一モジュールに入る必然性はなく，分割して別モジュールとすべきである．複数のモジュール間の機能を分析し，関係が少なく弱いほど，結合度が低いと呼び，望ましい．たとえば，複数のモジュールから同一データにアクセスしたり，相互にモジュール内部を直接参照し合うのは結合度が高く，いったん同一モジュールにまとめて，再分割を検討した方が良い．単独のモジュール内の強度は高く，複数のモジュール間の結合度は低くなるようにモジュール分割を行なう．この基準は，モジュール分割の結果を評価するのには有効であるが，どうすればこの基準を満足するようなモジュール分割が可能かについては触れていない．最終的には設計者の経験や能力に依存する．

• 最大抽象点［6］：分割の対象となっているモジュール内で，主要な入力データが出力データに変換を受ける状況を考える．具体的な入力データは

© 日本オペレーションズ・リサーチ学会．無断複写・複製・転載を禁ず．

モジュール内では，抽象的な情報になり，出口に近づくにしたがい，また具体的な出力データに変換される．たとえば入力ファイルがモジュール内では内部形式の中間データやテーブルの形で実現され，出力ファイルへ変換する対象となる．この抽象的な情報に変化するところを最大抽象点といい，この点を境界にして，入力処理部，変換処理部および出力処理部の各モジュールに分割することが望ましい．この分割法は，事務処理プログラムのように，主要なデータがモジュールを通過しながら変換を受ける問題に適している．

• ジャクソン法［8］：入力および，出力データのデータ構造をあらかじめ設計し，データ構造の各々の要素に対して，モジュールを対応させる分割方法である．データ構造として，構造化プログラミング制御3構造に対応した連続，繰り返し，選択の3種で表現する．入力および出力データのデータ構造の対応が明確な場合はほぼ機械的にプログラム構造を決めることが可能である．データ構造が一致しない場合は中間にデータ構造を入力，出力データのギャップをうめるように定義し，同様の操作を行なう．ジャクソン法は，事務処理プログラムのように，入出力のデータ構造が明確に定義できる問題に向いている．

• 機能分割［7］：与えられた機能（モジュール）をそれと等価なサブ機能の構造として分割する．元のモジュールへの入出力データは，分割された各サブモジュールに正確に引き継がれるが，この時，データの詳細化も行なわれる．1つの機能が3〜6個のサブ機能に分割されることが望ましい．結果として，与えられた問題は，機能の階層構造で表現されることになり，システムの理解性に優れたものになる．分割の指針として，前述の強度・結合度などを用いることになる．

• その他の分割の指針・モジュール分割を，データフローあるいは制御フローのいずれに着目して行なうかも重要な分かれ目である[10]．複雑なシステムでは，処理よりデータの方が具体的に決めやすく，まず入出力データあるいは主要な内部データを決定する方が設計が容易に進む．プロセス制御システムの場合，制御対象としてのプロセス側から入出力信号により制約をうけることが多く，タイミング情報などによる制御フロー中心の分割をする必要がある．リアルタイムシステムの設計のように性能を最優先するような場合，モジュール分割に先だち，あるいはモジュール分割の初期の段階で，性能上の影響が大きい部分を切り出し，その部分だけプログラム作成を行ない，性能上の確認を行なう，あるいはモデル化・シミュレーションにより性能予測を行なうなどして確認することが必要となる．

4. おわりに

本解説では，ソフトウエアのモジュール化についての主要な概念の紹介を行なった．たとえば，データの抽象化という概念は，その言葉で表現される前にも，優秀な設計者はすでに実行していたものである．しかし，明確な言葉で表現されることにより普遍化した概念となり，すべての技術者の共有の技法として定着することになる．

構造化プログラミングという言葉は，プログラムの構造を明確なものにするという概念を表わすとともに，GOTO lessあるいは順次・反復・分岐という制御3構造のみでプログラミングをすべしという具体的な枠組みをはめており，初心者でも実行可能である．

ジャクソン法は，入出力データの構造のみに着目すれば，ほぼ機械的なモジュール分割が可能ではあるが，事務処理用アプリケーション以外では適用が困難で普遍的な手法とは言いがたい．その他の指針は，おおむね，モジュール分割の良し悪しを判定する基準にはなるが，構造化プログラミングのように直接的な手段を与えてはいない．いかにすれば良いモジュール分割ができるかという命題に対して，より具体的な解が生まれることを期待したい．

© 日本オペレーションズ・リサーチ学会．無断複写・複製・転載を禁ず．

最後に，本文をまとめるにあたり，筆者の同僚松村一夫氏が社内用にまとめた資料を参考にした．ここに感謝の意を表わす．

参考文献

[1] Dijkstra, E. W.: Netes on structured programming. Structured programming, A.P.I.C. Studies in Data Processing No.8, Academic press (1972).

[2] Wirth, N.: Program development by stepwise refinement. Comm. ACM Vol.14, No.4 (1971).

[3] Liskov, B., Synder, A., Atkinson, R. and Schaffert, C.: Abstraction mechanisms in CLU. Com. ACM Vol.20, No.8 (1977).

[4] Jackson, M. A.: Principle of program design, Academic Press (1975).

[5] Parnas, D. L.: On the criteria to be used in decomposing systems into modules, Comm. ACM, Vol.15, No.12 (1972).

[6] Myers, G. J.: Characteristics of composite design, Datamation, Vol.19, No.9 (1973).

[7] Ross, D. T.: Structured analysis (SA): a language for communicating ideas, IEEE Trans. on Software Engineering Vol. SE-3, No.1 (1977).

[8] Jeichroew, D. and Herskey III, E. A.: PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing system, IEEE Trans. on Software Engineering, Vol. SE-3, No.1 (1977).

[9] ワーニェ，フラナガン（鈴木君子訳）：ワーニェ方法によるプログラミング学習上/下，日本能率協会（1973）

[10] 紫合 治，岩本莞二，藤林信也：統一的設計方法論に基づくソフトウエア設計システム，情報処理，Vol.21, No.5 (1980).

[11] 前川 守：情報処理システムのモジュール化，情報処理，Vol.21, No.7 (1980).

[12] 湯浅太一，小島哲二，中島玲二：モジュラー・プログラミングのための支援環境，情報処理，Vol.23, No.5 (1982).

[13] 高橋延匡：モジュール化の重要性—モジュール化の秘訣，品質，Vol.16, No.1 (1986).

[14] 松村一夫，古谷克二，山城明宏，小尾俊之：IMAPシステム(2) 設計・コーディング支援技術50SM，情報処理学会第31回全国大会予稿集（1985）.

次号予告

特集 企業の国際化

企業の国際化戦略……唐津 一（東海大学）

エンジニアリング産業の国際化戦略
——途上国の自律的発展を考慮した
プロジェクト形成の重要性について……北川 博（日本プラント協会）
……泉 一雄（NTT）

日本電気の海外事業要員育成施策……西村 捷敏（日本電気）

研究レポート ローカル・コンテント規制への適応……小野 桂之介（慶応大）

連載 EQUATRAN-M ……宮原昱中，小口梧郎（三井東圧化学）

© 日本オペレーションズ・リサーチ学会．無断複写・複製・転載を禁ず．