はしがき

ハードウェア，ソフトウェアという概念は，単にコンピュータの分野のみにとどまらず，今日では一般のシステムないしは装置にも適用され，日常語として広く利用されるようになってきている．

しかし，いかなる場合においても，ハードウェアは機械そのものを指し，それを目的どおりに駆動させるための一連の制御指令（プログラム）ならびにそれを作り出すまでの思考過程を総称してソフトウェアとよぶことにおいては，変わりはない．

ソフトウェアは，時としてプログラムと同義語として用いられている場合がある．しかし，この両者は基本的に異なる．プログラムはソフトウェアの所産ないしは最終的な記述表現であり，直接ハードウェアに入力されるべきものである．オペレーティング システム ないしはデータベース管理システムのように，特定の目的をもって作られたプログラム集団を，ソフトウェア システムということもある．

近年，コンピュータのきわめて広範な普及により，プログラムの作成技法を学ばれる人が激増してきた．問題を単に プログラミング技法に 限定するならば，これは使用する言語，すなわち許容される単語と，それを用いた記述規則である文法を覚えることに終始し，一見，単純作業の積み重ねに尽きる．しかし，コンピュータを用いる人はすべて，このプログラミングができなくてはいけない．今日 広く用いられている FORTRAN, COBOL, C 言語，LISP, PROLOG などの諸言語に関する解説書が数多く出版されているのはこのためであり，当面する問題をいかに簡潔に，またいかに早く解答を得るようにプログラミングを行うかがこのレベルの主要因となる．

しかし，言語出現の源にふれ，より合理的なプログラムを作成しようと考え

る場合には，ソフトウェアの基本を学び，その本質に触れなければならない．コンピュータと人間との間の意思の伝達を行うのが前述のようなコンピュータ用言語であり，またそれを用いて書かれたプログラムである．したがって，より自然語に近く，また，より的確に問題を記述できるような言語をいかに作り出すかはソフトウェア論の重要な課題の一つでもある．

本書は，問題提起からそれをプログラムとして記述し終わるまでの思考過程，ならびにその一般的な技法について，ソフトウェアの本質を説きながら，できるだけ平易に記述したものであって，ソフトウェア論を正統的に学ばれようとする初心者の方々を対象に，大学・高専の教科書ないしは参考書として用いられることを意図して著したものである．

われわれの生活の中に，コンピュータは限りなく浸透してきており，この結果としてプログラムの作成に要する時間と経費は膨大なものとなりつつある．プログラム生産の合理化・適正化は，現代社会における重要な問題であり，理科系の学生はもちろん，文科系の学生にとっても，また社会で活躍されている方々にも，ソフトウェア論の基本を修得されていることが好ましい時代に到達している．

本書がソフトウェア論の入門書として，広くお役に立つことができれば極めて幸甚とするところである．

終りに本書を発刊するに当たり，多大のお手数を煩わしたコロナ社の方々に厚く御礼申し上げる．

昭和 62 年 2 月 1 日

手 塚 慶 一

目　　次

1 プログラムとソフトウェア

2 構造化プログラミング

3 プログラミング言語

4 プログラム設計

5 プログラミング言語の処理系

6 ソフトウェア工学

7 プログラミング言語とソフトウェアの将来

プログラムとソフトウェア

プログラムとソフトウェアという言葉は，特に区別せずに使われることが多い．しかしながら，両者はかなり異なるものとみるべきである．プログラムという言葉は計算機の出現以前から使われているのに対して，ソフトウェアという言葉は計算機がある程度使われ始めてから，プログラムの発想から作成までの過程を総括する言葉として作られたものである．すなわち始めにハードウェアとプログラムがあり，その後ソフトウェアが生まれたものであり，プログラムはソフトウェアの最終段階に位置するとみることもできる．その意味からプログラム作成において考慮すべき問題と，ソフトウェア作成において考慮すべき問題はかなり異質のものであり，まったく別の点からのアプローチが必要である．

本章ではそのような観点からプログラムとは何かについて述べ，その後プログラムからソフトウェアへの展開について述べる．

1.1 プログラム開発

ある処理を実現するために作成され，計算機を始めとする種々の情報処理システムによって解釈・実行されるシンボル列あるいは構文をプログラムとよぶ．

われわれが計算機に対して計算処理を行わせようとするとき，計算機が受理可能な言葉（プログラミング言語）を用いて計算の内容を表現し，これを入力

することによって計算機上に所望の機能を実現する．プログラミング言語がFORTRAN. PASCAL, BASIC, C などに代表される手続き型言語の場合，処理は手順またはその集合の形で表現される．このことから，プログラムの定義を論理的に，“目的とする処理を実現するための一連の手順”と考えてもよい．

計算機システム開発の黎明期においては，計算処理の記述は機械語が唯一のプログラミング言語として使用され，続いてアセンブラ言語がこれに加わった．これらの言語は，基本的には，命令語がハードウェアの実行命令単位と1対1に対応しているのが特徴であり，プログラムの記述は非常に繁雑になる。このため，人間のもつ認識能力になじみにくい点が問題である．このころは，ハードウェアのコスト高が問題にされた時期でもあり，メモリ空間を節約しながら，いかにして高効率なプログラムを実現するかという職人芸的プログラミングセンスが重要視された．

開発されるプログラムが次第に大規模化するに伴い，この種のプログラムのもつ問題点が明らかになってきた。例を挙げれば，可読性の低さ，拡張性の乏しさ，信頼性の低さ，開発期間の長さ（コスト高）などである．当時，集積回路技術などの発達に伴うハードウェアの低廉化傾向を受け，人間が読みやすく，また，計算機の命令単位とは独立したより高度な処理を厳密に記述できる言語の開発が進められた．1960年代は，いわゆる第一世代高級言語の時代であり，FORTRAN, COBOL, PL/I, BASIC などの手続き型言語とその受理系（インタプリタ，コンパイラ）が相次いで開発されている．これに伴い発達したプログラム解析ツール，デバッギングツールなどのプログラム開発支援系と合わせ，プログラムの開発運用に関する問題は解消するかに見えた．しかしながら，これらを利用した莫大なプログラム開発の結果，次の2点が明らかにされたのである．

1. 高級言語を用いても，可読性，保守性，信頼性などについての問題は基本的には解決されない。理解が容易に思えるのは，程度の問題にすぎない。
2. 巨大で複雑なプログラムを開発するための技術は，依然として熟練プロ

グラマのノウハウに依存している．一般のシステム設計者が使用できる明確な設計指針が必要である．

さらに，高級言語を用いたプログラム開発の要求が指数関数的に高まる傾向を見せたことから，将来のプログラムの需給関係の破綻，すなわちプログラム危機が唱えられ始めた．

これらの経験的事実を受け，当時のプログラミング技法についてさまざまな問題点が指摘され始めた．これらの一部を要約すれば，

「処理内容の記述は，プログラミング言語による処理手順の記述としてだけ与えられるのでは不十分であり，処理の内容あるいは意味のプログラムへの変換を含めた一連の過程を記述し，さらに，プログラム上にもこれを反映させることが必要である」

との主張である．これが，ソフトウェアの概念の必要性を広く認識させる結果となった．

Dijkstra による構造化プログラミング，Wirth によるトップダウン設計法の二つは，プログラミングの指針をソフトウェア論の見地から明らかにした代表的技法といわれている．前者はプログラムの可読性の向上を目差し，後者はプログラム開発の設計指針を明らかにしているが，いずれも．“ソフトウェアの開発を通じたプログラム設計の必要性”を主張していることに変わりはない．本書で展開されるソフトウェア設計論は，これら二つの技法をもとにしたものである．

その後，開発された第二世代の手続き型言語，PASCAL, C, ADA などは，いずれも上記の主張の影響を受け，近年は，開発支援環境の著しい発達と相まってプログラムの開発効率は次第に高まりつつある．

以下の節では，プログラムおよびソフトウェアの言葉でよばれる概念あるいは実体の詳細についてさらに議論し，それらの関連を明確にするとともに，本書を通じて述べられるソフトウェアの開発，運用法の意義を明らかにする．

1.2 プログラムの要件

人間に命令を与える場合を考えてみよう．たとえば，次のような表現が可能である．

新聞を取ってこい！

風呂をわかせ！

人間ならば，このようなあいまいな表現に対しても適当に目標を設定し，またその場の状況に応じて臨機応変に処理を進める．しかし，計算機にはそのような判断・処理がほとんどできない．このため，計算機に対して命令を与えるプログラムでは，このようなあいまいさを含む表現はいっさい許されていない．現在，人間との対話を通じて計算機が命令を解釈し実行する試みはなされているが，実用のレベルには達しておらず，人工知能研究の分野において精力的に開発が行われている段階である．

プログラムにおいては，このため，一つ一つの動作にあいまいさがなく，動作のつながりにもあいまいさを含まない処理手順の表現を与えることが重要となる．このような処理手順の表現を**手続き**（procedure）という．

以下に手続きの例を二つ示す．

例 ***1.1*** 2^n-1 の形の素数であるメルセンヌ数を小さいものから順に m 個求める．

1. 2個のカウンタ A, B を用意する．
2. カウンタ A の値 a を 0 とする．また，カウンタ B の値 b を 1 とする．
3. $a<m$ である間，以下の操作を繰り返す．
   - 3.1 2^b-1 が素数であるかどうかを判定する．素数であればカウンタ A に 1 を加える（a の値を 1 だけ増やす）．
   - 3.2 カウンタ B に 1 を加える（b の値を 1 だけ増やす）．

例 ***1.2*** 与えられた実数 a, b, c を 3 辺の長さとする三角形が存在するか否かを yes, no で判定する．

1. a, b, c の少なくとも一つが零または負の数であれば，no.

2. $a+b<c$ ならば，no.

3. $c<|a-b|$ ならば，no.

4. 上の三つの条件で no とならなかったものは，yes と判定する．

これら二つの手順の記述はあいまいさなく記述されており，余分な判断を加える余地がない．計算や動作が間違いなく行われれば，だれが行っても同じ結果を得る．したがって，これらは手続きである．

しかし，次の例はそうではない．

例 ***1.3*** 風呂をわかす．

1. 風呂が汚ければ，きれいに洗う．

2. 風呂に水が入ってなければ，適量の水を入れる．

3. 風呂のバーナを点火する．

4. 風呂の温度を監視し，適温になれば火を止める．

この例では，まず，“洗う”，“水を入れる”，“監視する”といった動作の表現があいまいである．人間ならば日ごろの経験から解釈し実行できることも，風呂の特性を知らない計算機はまったく理解できない．さらに，人間が行う場合でさえ，この手順に従ってわかした風呂は結果が異なってくる．これは，“汚い”，“適量”，“適温”といった判断のための条件があいまいなためである．このようなわけで，この処理手順は，手続きではない．

処理手順に関しては，無あいまい性以外にも要求される重要な特性がある．その特性は，停止性とよばれている．すなわち，処理手順は有限時間のうちに終了することが求められる．

たとえば，例 *1.1* の手続きについて考えてみよう．メルセンヌ数とよばれる 2^n-1 の形の素数が何個存在するか（有限個か無限個か）は，知られていない．ある個数（m 個）のメルセンヌ数を求めようとするとき，それだけのものが存在するかどうかは不明である．したがって，この手続きの停止性は保証されない．

停止性の保証されている手続きを**アルゴリズム**（algorithm）とよぶ．有名な

アルゴリズムの一例として，二つの自然数の最大公約数を求めるユークリッドの互除アルゴリズムがある．

例 *1.4*　ユークリッドの互除アルゴリズム

二つの自然数を A, B とする．

1. 以下のステップを $A \neq B$ の間，繰り返す．

1.1　$A > B$ ならば，A の値を $(A - B)$ に変更する．

1.2　$A < B$ ならば，B の値を $(B - A)$ に変更する．

2. $A = B$ のとき，この値を最大公約数とする．

なんらかのプログラミング言語を用いてアルゴリズムを**表現**（implementation）したものがプログラムである．したがって，プログラムは無あいまい性と停止性を満足していなければならない．一般に，プログラミング言語は命令にあいまいさを生じないよう構文を制限した設計がなされているので，これに従って書かれた命令は，計算機によりあいまいさを含むことなく実行される．この意味で，プログラムの無あいまい性はプログラミング言語に依存する．しかし，これに対して，停止性はプログラム自体に起因する．ここで主として対象としている手続き型の言語では，処理制御はプログラムに任せられており，言語レベルでは停止性を保証していない．これに対して，PROLOGに代表される論理型言語など停止性を保証したものも開発されているが，そのままでは言語のもつ記述の柔軟性が損なわれてしまうことが知られている．

そのような意味から，現状では停止性を犠牲にし，あるプログラミング言語によって記述された「手続き表現」（アルゴリズムではない）をプログラムと見なしているのが実状である．しかし，目標とする処理の完了を意図してプログラムが作成される以上，ソフトウェアの開発に当たっては，停止性の保証についてつねに留意しなければならない．

問1　読者の作ったプログラムについて，停止性はどのようなかたちで保証されているか考えてみよ．

問2　プログラムにおいては必ずしも停止性は保証されていない．ユーザーはそのようなプログラムをどのようにして止めるか．

索　　引

〔L〕

〔M〕

〔N〕

〔O〕

〔P〕

〔R〕

〔S〕

——著者略歴——

手塚慶一（てづか よしかず）

1951 年　大阪大学工学部通信工学科卒業
1961 年　大阪大学助教授
同 年　工学博士
1972 年　大阪大学教授
1992 年　大阪大学名誉教授
1993 年　逝去

海尻賢二（かいじり けんじ）

1972 年　大阪大学工学部通信工学科卒業
1977 年　信州大学助手
同 年　工学博士
1978 年　信州大学助教授
1995 年　信州大学教授（工学部情報工学科），現在に至る

計算機ソフトウェア

Fundamentals of Software Design

1987 年 2 月 15 日　初版第 1 刷発行
2001 年 5 月 10 日　初版第 9 刷発行

検印省略

著　者　手塚慶一
　　海尻賢二

長野市中御所 5-5-3
発行者　株式会社　コロナ社
　　代表者　牛来辰巳
印刷所　富士美術印刷株式会社

112-0011　東京都文京区千石 4-46-10
発行所　株式会社　コロナ社
CORONA PUBLISHING CO., LTD.
Tokyo Japan
振替 00140-8-14844・電話（03）3941-3131（代）
ホームページ http://www.coronasha.co.jp

ISBN 4-339-00130-9　（清文社，愛千製本所）
Printed in Japan