解　説

標準構造に基づく系統的ソフトウェア設計法†

片岡雅憲†† 金藤栄孝†††
宮本和靖†† 山野紘一†††

1. はじめに

生産技術の改革は「標準化」の基礎なしには達成できない．ここで，「標準化」は，対象物の「構造の標準化」と，これを作り出す「工程の標準化」からなり，両者は相補関係にある．構造の標準化への視点を欠いた工程の標準化は，徹底さを欠いた小手先のものになりがちである．逆に，工程の標準化への配慮を欠いた構造の標準化は実現が困難で，絵に描いたもちになりかねない．

ソフトウェア（直訳すれば「やわ物」[1]）を生産物ととらえるならば，上記の原則はそのままソフトウェアにも適用することができる．実際に，構造の標準化に関しては，構造化プログラミング[18]，階層化設計[10]，データ中心設計[14),15]，抽象化設計[7]，等々，多様な提案がなされてきている．また，これらの構造の標準化を前提とした工程の標準化についてもいろいろな議論がされてきている[9),10),13)~15),18]．このように，ソフトウェアの標準化については，多くの提案がなされてきてはいるが，未だ，成熟段階に達しているとはいえない状況にある．個々の技法はともかくとして，統合的観点からの標準化は，まだまだ不十分といわねばならない．どの技法が，どの構造要素に関連づけられ，構造全体にどのような影響を与えるのか．また，構造と工程はどのように関係するのか．というような統合的観点からの検討が不足している状況にある．

現実のソフトウェア開発プロジェクトで設計法を改善するに当たって，筆者等が直面した問題点は，このような統合的アプローチの不足であった．部分的な改善を図っても，それが全体にどれほどの効果を与えるのかを確信できず，説明できないでいた．本稿は，この問題を解決すべく，筆者等がこれまで取りくんできた[2]「ソフトウェアの構造と工程の標準化技法」の概要を設計技法を中心に述べるものである．本技法では，構造の標準化の徹底と，これにともなう設計手順の系統性を最大の特長としている．特に，構造の標準化では，①ユーザ要求定義における抽象データ型の利用，②データ中心型モジュール設計，③正規表現による階層構造の標準化，等の近年のソフトウェア工学の新手法を実用化レベルで統合的に取り入れている．

本稿では，まず，ソフトウェアの構造の概念とその標準化について述べ，次に，これを実現するための設計法の概要を説明する．特に，この中で設計結果を構造に基づいて客観的に評価するための設計評価基準についても概説する．そして，共通課題[17]を本技法で設計した例を示し，最後に，本設計法の特長を述べる．

2. ソフトウェアの構造とその標準化

2.1 ソフトウェアの構造概念

ソフトウェアは次の要素からなる構造体ととらえることができる．

① 論理・物理構造
② データ・プロセス構造
③ 階層構造

以下に，これらの各要素について説明する．

（1） 論理・物理構造

情報は論理的意味と物理的形式の2つの面をもっている．ソフトウェアの構造を規定する情報についても，その役割や働き等の論理的意味を規定する情報と，その物理的実現形状，表現形状を規定する情報とに分類できる．**表-1**はこれをまとめたものである．

ソフトウェアの設計に当たっては，論理構造と物理

† Software Specification and Design Method Based on Integrated View of Structure Standardization by Masanori KATAOKA and Kazuyashu MIYAMOTO (Software Works, Hitachi, Ltd.), Hidetaka KONDO and Koichi YAMANO (Systems Development Laboratory, Hitachi, Ltd.).

†† (株)日立製作所ソフトウェア工場
††† (株)日立製作所システム開発研究所

表-1 ソフトウェアの論理構造と物理構造

項番	項目	論理構造	物理構造
1	説明	論理的意味構造	物理的形状構造
2	データ構造(例)	データ要素の意味，要素間の意味的関係（排他関係，直積関係）	記憶形式，データ長，記憶媒体（主記憶，ビデオ端末，プリンタ等） 記憶域上のレイアウト，データ間のポインタ
3	プロセス構造(例)	機能要素，要素間の意味的関係（合成関数関係，直積関係等）	モジュール，モジュールの親子関係・制御関係，コマンド，コマンドプロシジャ
4	管理指標(例)	データ要素数，機能要素数，要素間関係演算子数，エントロピ（情報量）	ソースカード枚数，オブジェクトコード長，モジュール数，コマンド数，ドキュメント枚数

構造の整合に十分に留意する必要がある．論理構造面での配慮を欠いたソフトウェアはユーザ要求を十分に満足させぬものになりがちである．また，物理構造面での配慮を欠いたソフトウェアは性能上の問題点を発生させる恐れがある．過去において，何よりもまず性能が優先される時代にあっては，物理構造の設計が重要であった．しかし，近年のハードウェアの性能・価格比の大幅な向上に支えられて，ユーザ要求への整合を目的として，論理構造がより重視されるようになってきた．このような時代の要請に合わせるためには，まず論理構造を設計し，次にこれに基づいて物理構造を設計するという論理構造優先の手順が必要である．

（2） データ・プロセス構造

ソフトウェアは，それを抽象的な情報処理機械ととらえれば，処理対象であるデータと，処理主体であるプロセスとに分けることができる．そして，そのおのおのについて，表-1に例示するような構造を考えることができる．したがって，ソフトウェアの設計方針として，データ中心の設計とプロセス中心の設計の2つが考えられる．現代の電子計算機ハードウェアは，ノイマン型の制御フローマシンであり，その中心は，いわゆる CPU (Central Processing Unit) であることから，プロセス中心のソフトウェア設計の方が馴染みやすく，ハードウェアの性能をひき出しやすかった．しかし，電子計算機のユーザである人間の情報処理系を考えると，これがデータフローマシンとしての性格も強く持っていることを無視し得ない[3),4),19)]．そして，電子計算機ハードウェアの設計もマンマシンインタフェースを含めたシステム全体の改善に大きな努力が払われるようになってきており，CPU 性能だけを重視する考え方は見直されつつある．ユーザたる人間の情報処理系との整合を考えるならば，今後，ソフトウェアの設計方針もプロセス中心の設計から，データ中心の設計も含めた統合的な観点をとるようになるものと考えられる．そして，このような統合的観点に基づいたソフトウェア設計は，ユーザ要求に整合した優れたマンマシンインタフェースを生み出し，データ局所性の高い保守しやすい内部構造を作り出す．

表-2 ソフトウェア（物理プロセス）の階層構造

項番	階層名称	大きさの目安（ソースコード行数）	説明
1	System	50,000以上	特定の問題解決に関連する Program の集まり
		÷5 ↓↑ ×5	
2	Program	10,000以上	計算機上で機能を独立に実行可能な単位
		÷5 ↓↑ ×5	
3	Component	2～3,000	特定のデータに関連する Section の集まり
		÷5 ↓↑ ×5	
4	Section	600	コンパイル単位，単体テスト単位
		÷5 ↓↑ ×5	
5	Unit	125	机上レビュの単位
		÷5 ↓↑ ×5	
6	Segment	25	最小単位のモジュール（Unit 中で共用可）
		÷5 ↓↑ ×5	
7	Basic Segment	5	最小の処理単位（PAD やフローチャートの1ボックス）
		÷5 ↓↑ ×5	
8	Source Code	1	ソースコードの1行

{Magic Number 7±2}
人間は 7(±2) つ以内のものの統合や，7(±2) つ以内のものへの分割は得手，しかし，7(±2) つを越えると誤りが激増する．ex. 電話番号 045-881-7161 (0458817161 と比べてどちらがおぼえやすいか)

（3） 階層構造

人間の頭脳の特性として「7つ（正確には7±2）以内のものの統合や，7つ以内のものへの分割は得手であるが，対象が7つを越えると誤りが激増する」ことが良く知られている[3),4)]．例えば表-2下部の電話番号の例でも簡単にこの事実を確かめられる．

ソフトウェアが人間の頭脳労働の所産であることを考えれば，上記の人間の頭脳特性はソフトウェアの設計においても十分に考慮されてしかるべきであり，階層構造設計が極めて重要である．表-2は，ソフトウェアの物理プロセスの例をとって階層構造を示したもので，各階層の大きさ（ソースコード行数で表示）の目安が下位階層の約 5(=7−2) 倍になっている点に注意されたい．もしも，この階層の2つ以上を1度に跳ぶと無理が生じる．例えば Unit の仕様から，いきなり Source Code を書くと，プログラマは1度に扱

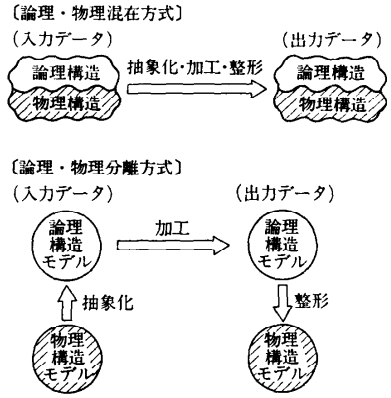


図-1　論理・物理構造を分離した設計方式

える情報の複雑さを越える難しさの作業を強いられることになり，不良が入り込みやすい．

2.2　ソフトウェア構造の標準モデル

（1）　論理構造モデルと物理構造モデルの分離（型と形）

前節で述べたソフトウェア構造を標準化するために，標準モデルを定める．モデルは論理構造と物理構造について，おのおの「型」モデルと「形」モデルが定められる．ここで「形」とは，物理的実現形式や外見を定めるモデルである．一方「型」は，形式や外見に依存しない本質的な意味や働きを定めるモデルである．

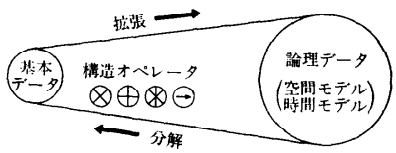
「型」はデータ型 (Data Type)，抽象データ型 ADT (Abstract Data Type) 等の名で呼ばれている．一方，「形」は，様式 (Form)，定規 (Template)，編成 (Organization)，等の名で呼ばれている．

モデルが論理構造モデルと物理構造モデルとに分離されていることは，当設計法の大きな特長である．このため，**図-1** に示すように，抽象化，加工，整形の各操作が分離され，標準化が徹底される．

（2）　論理構造モデル（型モデル）

データの論理構造モデルは **図-2** に示すようになっている．基本データを構造オペレータで組み合わせることにより，拡張された論理データが得られる．逆に論理データは基本データに分解できる．構造オペレータとしては，直積，直和，列[10]の正規表現オペレータとその拡張オペレータ（写像，合成写像）を用いており，単純であるため修得が容易である．

論理構造モデルは構造オペレータへの意味づけの違いによって，(論理)空間モデルと時間モデルに分けられる．空間モデルは，論理空間上での集合，及びその操作をモデル化するのに適している．一方，時間軸上に定義した方がわかりやすい構造や，因果関係をモデル化する場合には，時間モデルが適している．後者は，いわゆるオートマトンモデルになっている[5),6),11),12)]．実際の設計作業では，この2つのモデルを組み合わせて用いている．



- 基本データ：整数，実数，文字，ブール，数え上げデータ等
- 構造オペレータ[注]：直積⊗，直和⊕，列⊛，写像→
- 空間モデル：論理空間上に定義したモデル
- 時間モデル：時間軸上に定義したモデル

注）　プロセス(機能)の論理構造モデルではこの他に次の構造オペレータがある
合成写像◉，繰返し合成写像★

図-2　データの論理構造モデル

基本データ及び論理データに対して，これらを操作する関数群が定められる．基本データとこれを操作する関数群は，基本データ型として，プログラミング言語レベルで標準化される．また，これを組み合わせて拡張した論理データとこれを操作する関数群についても標準化することができる．例えば，行 (Record)，列 (Sequence)，表 (Table)，配列 (Array)，木 (Tree) 等の良く知られた論理データがあるが，これらは，おのおの，専用の操作機能群とひとまとめに考えることにより，おのおのの実現方式とは独立な抽象データ型として標準化できる．抽象データ型は，代数的に厳密な形式的仕様記述やその正当性の証明の手段として用いられるが[20),21)]，本稿に述べるように準形式的な仕様記述の手段として実用レベルでの効果も発揮している[7),22)]．

基本データ型と抽象データ型，及び構造オペレータによるこれらの組合せを用いれば，ほとんどの構造は容易にモデル化できる．

（3）　物理構造モデル（形モデル）

物理構造を規定する情報は次の3種類から構成される．

①論理構造情報，②形状情報，③装置情報．

このうち，①は論理構造が物理構造中に埋め込まれたものであり，既に本節の(2)で説明した．次に③は装置に関するもので，いわゆる仮想化により，装置独立性が実現できれば，不要な情報である．いずれにし

表-3 物理形状情報の例

分類	形状区分	形状情報の例
物理データ	表	タイトル，行 ID，カラム ID 他
	入出力パネル（端末画面）	タイトル，フィールド属性・配置他
	テキスト	種別（本・論文・その他），タイトル，著者名，目次，概要他
	グラフ	種別（棒，折れ線，層，円等），タイトル，軸情報他
	ファイル	編成（順，直接，区分等），ファイル名，レコード情報他
物理プロセス	プログラム	プログラム名，入口名，記述言語，入出力パラメータ他
	コマンド	コマンド名，コマンド形式
	メニュー（端末画面）	タイトル，メニュー機能 ID，機能 ID の配置他

表-4 正規表現による構造オペレータの標準化

分類	構造＼オペレータ		⊗(乗算)	⊕(加算)	⊛(べき乗)
論理構造	空間モデル	論理データ	直積	直和	列
		機能	直積	直和	列
	時間モデル	論理データ	連接	選択	くり返し
		機能	状態の無条件遷移	状態の選択的遷移	同一状態のくり返し
物理構造	構造単位相互間	物理データ間	ポインタ	選択的ポインタ	チェインポインタ
		プロセス間	無条件コール	選択的コール	くり返しコール
	構造単位内	物理データ内	レコードデータ	選択データ	リストデータ
		プロセス内	連接	選択	くり返し
	ソースコード	データ	レコードデータ	選択データ	リストデータ
		プロセス	連接	選択	くり返し

ても，装置情報は OS (Operating System) や，デバイスドライバ (Device Driver) と呼ばれる基本ソフトウェアが取り扱うべき情報なので，ここではこれ以上の議論は省略する．

物理構造モデルの中心は，形状情報の標準化にある．そこで，第一に重要なことは，論理構造情報と形状情報の完全な分離である．分離ができないと標準化を達成し得ない．次に大切なことは形状情報自体の標準化である．表-3 は物理形状情報の例を示したものであり，これらの構造を把握し，標準パターン化することができる．また，形状情報の構造オペレータを表-4 の物理構造に示すように標準化することができる．

（4） 正規表現による構造オペレータの標準化

本技法では，構造オペレータを表-4に示すように正規表現で統一的に標準化している．これにより次のような利点が得られる．

①単純で設計者が理解しやすい（3構造がベース）
②階層化が徹底できる（必ず階層分割できる）
③論理・物理構造の対応が容易（すべて正規表現）

3. 本設計法の概要

3.1 標準設計手順

本設計法は，2章に述べた「構造の標準化」と，これを作り出すための「工程の標準化」たる標準設計手順からなる．前者については既に2章で述べたので以下では後者について説明することとする．当手順は次の原則の上に定められている．

㋑ 論理構造→物理構造（論理構造優先）
㋺ データ構造→プロセス構造（データ構造中心）
㋩ 階層構造（トップダウンまたはボトムアップ）

論理構造は，設計対象とするソフトウェアの本質的構造を規定するものであるから，物理構造よりも優先されなくてはならない．すなわち，論理構造を先に決定し，次にこれに基づいて物理構造を決定するのが良い．ユーザとの直接インタフェースである入出力データ（例えばビデオ端末画面による入出力やプリンタ出力）については論理構造の決定と同時に，物理的レイアウト構造を定め，早期にその妥当性をチェックすることが望ましい．また，基本的な論理構造（例えば，行構造や表構造）に対応する物理構造を部品群としてあらかじめ用意しておけば，論理構造だけを設計し，物理構造の設計が不要となり設計効率が向上する．

データ構造中心の設計を行うと，データ参照の局所性の良い，また，その結果としてモジュラリティの優れた構造が得られる．入力データと出力データが共に正規表現構造であって，相互間の対応がとれる場合は，データ構造からプロセス(機能)構造が機械的な操作で決定できる．論理データ構造と物理データ構造との間の変換プロセスでは，物理構造が論理構造に一致しないがための，境界不一致[14]が起る．これに関しては Jackson[14] のいう転換 (Inversion)手法により，状態ベクトル[14),23)]を導入すれば，プロセス（機能）構造はデータ構造から，やはり機械的な操作で決定できる．より複雑な構造不一致が起った場合には，個別にアルゴリズムを定めプロセス構造を決定せざるを得ない．しかし，この場合でも，良く用いられる基本的な機能（例えばソート機能）は標準部品化しておくことが望

ましい.

階層構造を設計するには，トップダウンとボトムアップの2つのアプローチがある．トップダウンアプローチでは上位の抽象的概念を分割してより下位の構造要素と対応づける．ここで必要とされる設計能力は，このような上位レベルの構造を的確にとらえ得る見識と経験にささえられたものでなくてはならない．一方ボトムアップアプローチでは，抽象データ型や標準部品等の既存の知的財産を組み合わせて利用してゆく．このアプローチでは，これら既存財産がどれほどに整備されていて使いこなし得るかが開発プロジェクトの成否を定める．トップダウン，ボトムアップのどちらを利用するかは，対象とするソフトウェアの性質と，また，その開発プロジェクトが置かれている環境によって定まる．標準部品が十分に整えられた環境下での中小規模のソフトウェアの開発ではこれら既存財産を活用できることからボトムアップの設計をベースにするのが着実である．しかし，大規模なソフトウェアの開発や，標準部品が備わっていない環境下での開発では，個人単位での担当が可能な適切な規模まで，仕様を分割するためのトップダウン設計に重点を置かざるを得ない.

3.2 構造評価基準とレビュ

ソフトウェアの設計作業は極めて複雑な頭脳労働であり，必ずしも手順通りに円滑に進捗するとは限らず，工程の途中で思考誤りが混入することが少なくない．このため，設計結果を客観的に評価して工程にフィードバックする「設計レビュ」が重要である.

設計レビュには大きく分けて，①ユーザ要求と設計仕様との整合と，②設計仕様自身の妥当性の2つの観点がある．前者については形式化することは困難であり，チェックリスト等によるレビュが一般的である．一方，後者については，構造に基づいた設計評価基準を標準化することができる.

表-5は，これをまとめたものであり，各基準は本稿で述べたソフトウェア構造と対応づけられている.

4. 設　計　例

当設計法による設計例を示すために，1984年3月情報処理学会設計技法シンポジウム[17]の共通問題をとりあげてみよう．この共通問題では，当設計法でいうところの論理構造上の条件だけしか示されていないので，ここでは，物理構造の設計は省略する．また，問題に明示されていない入力エラーチェック等の機能は省略した．以下，設計手順に従って，説明する.

表-5　ソフトウェア設計評価基準

項番	分　類	設計評価基準
1	論理データ階層	① 全データ要素が定義済 ② 全階層オペレータが定義済 ③ 子要素の数（≦7 が良い）
2	機能階層	① 全機能要素が定義済 ② 全階層オペレータが定義済 ③ 子要素の数（≦7 が良い）
3	論理データと機能との対応	① 全データが機能に対応 ② 全機能要素がデータに対応 ③ 論理データと機能の相互関係数（局所性）
4	物理データ階層	① 全データ要素が定義済 ② 物理データ間の全ポインタが定義済 ③ 上記ポインタは階層型 ④ データ要素数／物理データが適切
5	モジュール階層	① 全モジュールが定義済 ② モジュール間のすべての制御関係が定義済 ③ 上記制御関係はサブルーチンコール型 ④ 各階層のモジュールの大きさが適切
6	物理データとモジュールとの対応	① 全物理データがモジュールに対応 ② 全モジュールが物理データに対応 ③ 1つの物理データに対応するモジュール数（1対1対応が望ましい）
7	論理データと物理データの対応	① 全論理データが物理データに対応 ② 1つの物理データに対応する論理データ数（1対1対応が望ましい）
8	機能とモジュールの対応	① 全機能がモジュールに対応 ② 1つのモジュールに対応する機能数（1対1対応が望ましい）

（1）　系全体への入出力データの定義：**図-3, 4**

（2）　中間データの設計：中間データとして，①酒の在庫管理表，②コンテナの管理表が必要なことは明白であり，これらを設計する（**図-5, 6**）.

（3）　中間データへのアクセス機能の設計：図-5, 図-6 はおのおの，酒の在庫管理表，コンテナの管理表の表現データ構造である．この2つの中間データへのアクセス用の機能を以下のように選んで，設計することにより，これら2つの管理表を利用者側に対して抽象化する.

①銘柄入庫機能 AIN

入力：銘柄名⊗銘柄コンテナ情報⊗（酒の在庫管理表）＝銘柄名⊗（所在コンテナ番号⊗銘柄コンテナ本数）⊗（酒の在庫管理表）

出力：なし（更新された酒の在庫管理表）

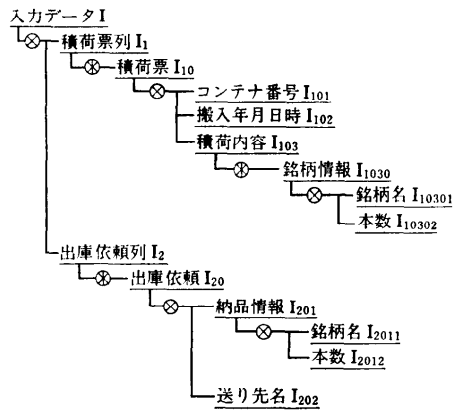


図-3　入力データの論理構造

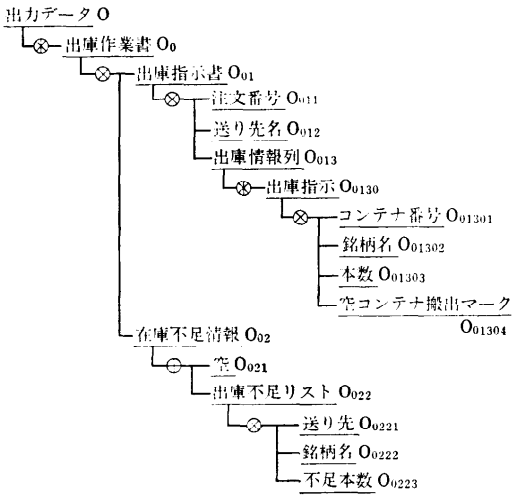


図-4　出力データの論理構造

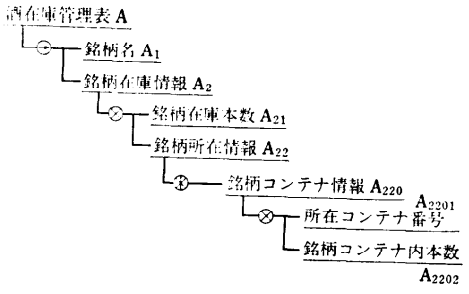


図-5　酒の在庫管理表の論理構造

機能：指定された銘柄名のコンテナ情報を在庫管理表に追加し，銘柄在庫本数を更新する．

②銘柄出庫機能 AOUT

入力：銘柄名⊗出庫要求本数⊗（酒在庫管理表）

出力：銘柄出庫情報m⊗不足本数 l

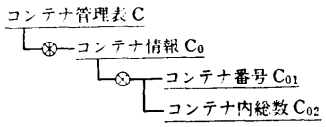


図-6　コンテナ管理表の論理構造

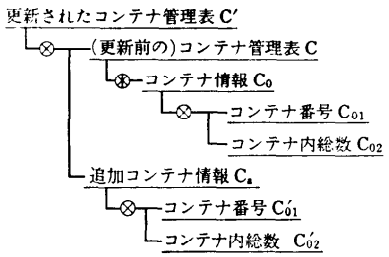


図-7　CIN による更新後のコンテナ管理表の論理構造

表-6　CIN の入出力・機能対応表

入　力	出　力	機　能
$C \otimes P$	C'	CIN
C	C	CIN_1
P	C_a	CIN_2
P_1	C'_{01}	CIN_{21}
P_2	C'_{02}	CIN_{22}

表-7　系全体の入出力機能対応表

入力	出力	機能	入力	出力	機能
I	O	F	m	O_{013}	F_{20132}
I_1	(C)	F_1	m_0	O_{0130}	F_{201320}
I_2	O	F_2	m_{01}	O_{01301}	$F_{2013201}$
I_{10}	—	F_{10}	I_{201}	O_{01302}	$F_{2013202}$
I_{101}, I_{103}	t	F_{101}	m_{02}	O_{01303}	$F_{2013203}$
I_{101}, t	—	CIN	m_{01}, m_{02}	O_{01304}	$F_{2013204}$
I_{101}, I_{1030}	—	AIN	m_{01}, m_{02}	emp	COUT
I_{20}	O_0	F_{20}	emp	O_{01304}	$F_{20132042}$
I_{20}	O_{01}, l	F_{201}	$I_{20}, l(\leqq 0)$	O_{021}	F_{2021}
I_{20}, l	O_{02}	F_{202}	$I_{20}, l(>0)$	O_{022}	F_{2022}
—	O_{011}	F_{2011}	I_{202}	O_{0221}	F_{20221}
I_{202}	O_{012}	F_{2012}	I_{2011}	O_{0222}	F_{20222}
I_{201}	O_{013}, l	F_{2013}	l	O_{0223}	F_{20223}
I_{201}	m, l	AOUT			

⊗（更新された酒の在庫管理表）

銘柄出庫情報m＝(所在コンテナ番号 m_{01} ⊗銘柄コンテナ取出数 m_{02})＊

機能：指定された銘柄に関する出庫情報を出力，在庫不足の場合は不足本数を出力する．

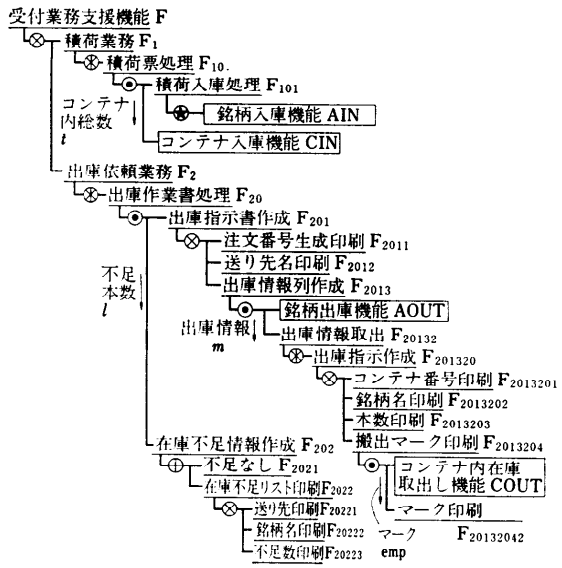


図-8　系全体の機能階層構造

③コンテナ入庫機能 CIN

入力 P：コンテナ番号 P_1 ⊗コンテナ内総数 P_2 ⊗（コンテナ管理表 C）

出力：なし（更新された管理表 C'）

機能：指定されたコンテナ情報をコンテナ管理表に追加する．

④コンテナ内在庫取出し機能 COUT

入力：コンテナ番号⊗在庫取出数⊗（コンテナ管理表）

出力：空コンテナ搬出マーク emp ⊗（更新されたコンテナ管理表）

次に上記の4機能と入力，出力との対応関係を厳密に定義する．ここでは上記の CIN を例にとると，更新されたコンテナ管理表 C' の構造を**図-7**のように定義し，**表-6** の入出力・機能対応表が得られる．

（4）系全体の機能の設計：前記（1）での入力データと出力データの構造の対応に基づいて，機能の構造をトップダウンに定義する(**表-7**)．この際に，前記（3）で定めた抽象データへのアクセス機能は定義済の基本機能として扱う．この結果，**図-8** の機能構造を得る．

5. 本設計法の特長

本設計法の特長は，ソフトウェアの構造を統合的に把握し，その標準化を徹底し，また，これに基づき設計手順を系統化している点にある．以下にその詳細を説明する．

（1）　構造の標準化を徹底

ソフトウェアを，①論理・物理構造，②データ・プロセス構造，③階層構造，の3構成要素からなる構造体としてとらえその標準化を図っている．

論理・物理構造を別々の構造として分離して扱えるようにしている点は本技法の大きな特長の1つである．Myers 等の複合・構造化設計法[9]は物理プロセス（モジュール）の実践的な設計法であり，モジュール化の良さの基準として「強度」と「結合度」が定義されている．これらの基準は，本稿表-5の「機能とモジュールとの対応」，「物理データとモジュールとの対応」基準と同等のものである．つまり，複合・構造化設計でも暗黙に論理構造を意識している．一方，本技法では，論理構造を明示的に意識していて，むしろ，積極的に利用している．具体的には，従来ともすれば，正当性の証明等の学術レベルで用いられていた抽象データ型[7),20)-22)]を実用レベルで取り入れている．これにより，抽象データ型の持つ，構造的単純さや，論理的厳密さという利点を活用している．

データ・プロセス構造のとらえ方は，Jackson 法のそれとほぼ同じである．相違点の1つは，当技法では，Jackson 法にはないモジュールという概念が用いられていることである．また，もう1つの相違点は，Jackson 法では境界不一致を一般的な問題として扱っているが，当技法では，これを図-1に示す変換過程での論理構造と物理構造の不一致によるものと意味づけている点である．

階層構造を標準化するために，当技法では表-4に示すように正規表現構造を統合的にとり入れている．正規表現の設計での活用についてはこれまでも二村[13)]，Jackson[14)]，Warnier[15)]等による優れた業績があり，当技法での正規表現の活用法はこれらと同等である．ただし，これらの技法では区分していない直積オペレータ⊗と合成関数オペレータ⊙を区分して，時間的順序の必要性の有無を明示できるようにしている．

（2）　ユーザ要求に整合した優れた機能を実現

本技法では，抽象データ型や正規表現を用いて，ユーザ要求仕様を簡単に，しかし，正確に記述できる．また，表-4に示すように（論理）空間モデル，時間モデルの両方が記述できることからその適用範囲が広い．したがって，ユーザ要求と整合した優れたソフトウェアを実現しやすい．

また，論理構造と物理構造の分離により，ハードウェアや OS に依存しない論理的なユーザインタフ

ェースを実現できる．さらには，抽象データ型の概念は形式に依存しないデータ構造を定義することができるため，1つのデータを表，グラフ，ファイル等の多様な表現形式に簡単に変換できる．これは今後のソフトウェアに求められている重要な機能であり，当技法はそれを実現するのに極めて適している．

（3） 生産技術を大幅に改善する

当技法では構造の標準化を徹底しているため，これを前提とした設計支援ツールの活用により[16]設計工程の早い時期に不良を摘出できる．そして，作りやすく，保守しやすい構造を作り出す．特に，論理構造と物理構造の分離は標準部品群を作ることを容易にする．また，正規表現構造の徹底は各種のソフトウェア開発支援ツールによる自動化に適している．そして，これらの特長は品質と生産性の改善に大きく寄与する．

6. まとめ

ソフトウェアは，ユーザにとっては知的活動の道具であり，メーカからすると知的生産の成果物である．このようにソフトウェアは人間の知的活動と密接にかかわっている．近年のハードウェア技術の急速な発展を背景としてソフトウェアの設計技法の視点は，ハードウェアの性能重視から人間の情報処理系とのインタフェース重視へと大きく転換されつつあるといえよう．そうであれば，本稿に述べたようなソフトウェアの論理構造と物理構造の両方を意識した統合的な構造概念に基づく設計方法は今後のソフトウェア開発において大きな効果を発揮するものと期待できる．

ソフトウェア要求定義論理モデルへの正規表現オペレータの適用に関しては，情報処理振興事業協会(IPA)技術センタの要求定義プロジェクトワーキンググループで議論した．IPA 技術センタ川合所長，倉橋研究員をはじめ関係諸氏のご指導とご助言に深く感謝する．

また，日頃，当技法の活用と改善に貴重なご助言をいただいている(株)日立製作所システム開発研究所及びソフトウェア工場の関係各位に心より感謝する．

参考文献

1) 森口繁一：やわ物の品質管理の考え方，第1回ソフトウェア生産における品質管理シンポジウム発表報文集，日本科学技術連盟（1981）．
2) 片岡雅憲他：ソフトウェア構造設計技法，日立評論，No. 62, pp. 7-10, 日立評論社（1980）．
3) Klatzky, R. L.（箱田他訳）：記憶のしくみ，サイエンス社（1982）．
4) Rumelhart, D. E.（御領訳）：人間の情報処理，サイエンス社（1983）．
5) Salomaa, A.（北川他訳）：オートマトン論，共立出版（1984）．
6) Hopcroft, J. E. and Ullman, J. D.(野崎他訳)：言語理論とオートマトン，サイエンス社（1980）．
7) Shankar, K. S.: Data Structures, Types, and Abstractions, IEEE Computer, Vol. 13, No. 4, pp. 67-77 (1980).
8) 菅野文友他：ソフトウェアデザインレビュー，日科技連出版社（1982）．
9) Myers, G. J.（国友訳）：ソフトウェアの複合/構造化設計，近代科学社（1979）．
10) Hoare, C. A. R.（川合訳）：データ構造化序論，『構造化プログラミング』サイエンス社（1975）．
11) 倉橋昭他：正規表現に基づいた要求定義モデル，第25回（昭和57年度後期）情報処理学会全国大会講演論文集，pp. 509-510（1982）．
12) 倉橋昭他：正規表現に基づくソフトウェア要求モデルについて，情報処理振興事業協会技術センタ（1983）．
13) 二村良彦他：PAD(Problem Analysis Diagram)によるプログラムの設計および作成，情報処理学会論文誌，Vol. 21, No. 4, pp. 259-267(1980)．
14) Jackson, M. A.（鳥居訳）：構造的プログラム設計の原理，科学技術出版社（1980）．
15) Warnier, J. D. and Flanagan, B. M.（鈴木訳）：ワーニエ方式によるプログラミング学習（上，下），日本能率協会（1973）．
16) 片岡雅憲他：ソフトウェア開発支援システム(CASD システム)，日立評論，No. 62, pp. 33-36, 日立評論社（1984）．
17) 「プログラム設計技法の実用化と発展」シンポジウム論文集，情報処理学会（1984）．
18) Dijkstra, E. W.（野下訳）：構造化プログラミング論，『構造化プログラミング』サイエンス社(1975)．
19) Bindra, D.（富田訳）：知的行動の脳モデル，誠信書房（1980）．
20) Goguen, J. A., Thatcher, J. W. and Wagner E. G.: An Initial Algebra Approach to the Specification Correctness, and Implementation of Abstract Data Types *in* Current Trends in Programming Methodology, Vol. 4, Prentice-Hall (1978)．
21) 稲垣康善，坂部俊樹：抽象データタイプの代数的仕様記述法の基礎(2) 抽象データタイプ，情報処理，Vol. 25, No. 5, pp. 491-501（1984）．
22) Archibald, J. L., Leavenworth, B. L. and Power, L. R.: Abstract Design and Program Translator: New Tools for Software Design, IBM Syst. J., Vol. 22, No. 3, pp. 170-187 (1983).
23) 大野侑郎：ジャクソンシステム開発法，情報処理，Vol. 25, No. 9, pp. 955-962（1984）．

（昭和59年9月3日受付）