# ソフトウェア工学のＯＳ開発への寄与
# 開発経験と問題の提起

高橋延匡（東京農工大学工学部）

## １．はじめに

オペレーティング・システムの開発の歴史は、単なる計算機のハードウェア資源の有効活用から、ソフトウェアから人間までを含んだ情報システムの管理運営システムへと展開してきた。

その結果、汎用大型計算機システムを対象とした汎用オペレーティング・システムは大規模化、広範的利用に対処するため、益々多機能化している。したがって、その開発には、メインフレーマ各社は膨大なマンパワーを投入せざるを得なくなっている。当然その開発現場では、いわゆるライフサイクル・モデルの各段階で種々の問題が生じ、それぞれ「ソフトウェア工学」の知見の利用や各社独自の創意工夫がなされていると思われる。

一方、マイクロ・プロセッサを用いたパーソナル・コンピュータ用のオペレーティング・システムも、UNIXをはじめ、種々のものが市場に出荷されている。これ等は、従来の汎用オペレーティング・システムにくらべて、小回りのきく点もあり、評価が高くなりつつある。

以上のように、オペレーティング・システムと言っても、汎用大型計算機を対象とするものから、パーソナル・コンピュータ用まであり、それぞれの開発に関し「ソフトウェア工学」の主題も異なるものと思われる。

## ２．ＯＳ開発上の問題点

ここでは、筆者が関連した、初期のバッチ用オペレーティング・システムと、仮想記憶方式のタイムシェアリング・システムと、現在、研究室で開発中のパーソナル・コンピュータ用のオペレーティング・システムを例にとり、ソフトウェア工学の見地から眺めてみよう。以下、それぞれのＯＳの開発時点で考えたことについて、良かったこと、反省材料とを対比させ、その項目を中心に述べることにより問題の提起としたい。

### ２．１　HITAC 5020モニタ（1962/5～1966/3）

［Ｉ］　良かった点は以下の通り。

（１）HITAC 5020アーキテクチャの哲学とプログラミング技法の教育（島内剛一（立教大））

（２）治工具ソフトウェア（ソフトウェア・ツール）を並行して開発したこと（穂坂衛(東大)）

（３）クローズド・オペレーションを採用し、１回の使用時間を制限したこと

（４）ＯＳのモジュールは、すべて256語単位に設計したこと

（５）最初からＯＳは動くように設計したこと。何もしないでリターンをするモジュールを作り、出来上がり次第、置き換えていったこと

（６）FORTRAN(HARP 5020)の開発は最強メンバーを投入したこと

（７）コントロール・プログラムとFORTRANとの設計者の相互理解が良く、インタフェースの切り分けがうまくいったこと

（８）完成後、設計の評価のため私自身約一年間東京大学大型計算機センターに常駐したこと

［Ⅱ］　考えさせられた問題点は以下の通り。

（１）開発の後半のフェーズを技術移管の見地から管理体制を中央研究所から現在のソフトウェア工場に移したこと

- モラール・オリエンテッドから員数管理オリエンテッドへの管理方式へ
- 人の増加に伴い少数精鋭から平均的プログラマの集団へ

（２）夜中の計算機利用

（３）新人の投入と教育の問題

（４）ドキュメンテーション、マニュアルの問題

### ２．２　HITAC 5020 TSS（1966/4～1968/3）

ソフトウェア工場から中央研究所に戻り、HATAC 5020 TSS の開発プロジェクトが開始された。メンバーは少数精鋭に戻った。

［Ｉ］　開発開始時点における問題意識は、以下の通り。

（１）ソフトウェアの生産性をあげること。そのため、システム記述言語を採用すること

（２）仮想記憶方式のTSS を開発することによりソフトウェアの開発環境(夜間作業や残業など)を改善すること

（３）技術移管と新人の教育両面のニーズに対処するため、ドキュメンテーションのルールを確立すること

- プログラム・レポート制度
- 仕様作成などの設計検討会議の詳細な議事録を残すこと
- 「何故？」に答えるドキュメントにすること

（４）設計に徹底的に時間をかけること

- ２年間のプロジェクトでコーディングに集中したのは、最後の３ヶ月間であった

［Ⅱ］　開発の設計責任者として恵まれた点は以下の通り。

（１）MIT のproject MAC の発表による MULTICS の概念に刺戟を受けたこと

（２）HITAC 5020 TSSの目標を皆、熟知していたしソフトウェアの開発能力が高かったこと

（３）担当者がバッチ型のＯＳの開発経験を積んだこと

（４）ハードウェアのスペック（特に Dynamic Address Translator、端末制御用計算機）をハードウェア/ソフトウェア両グループで議論しながら進められたこと

（５）東京大学大型計算機センターとの共同研究を通じて外部仕様を固めたこと

［Ⅲ］　以上の開発の評価として、良かったことは以下の通り。

（１）システム記述言語として、PL/Ｉ サブセット(PL/ＩＷ)を開発して用いたこと

- ＯＳの核は、PL/ＩＷから人間がコンパイル

したが、プログラムの可読性は大幅に向上した

・ PL/ＩＷの完成後のプログラムの生産性向上には目をみはる結果を得た

（２）ディスク・ベースのファイル・システムと、セグメンテーション機構を用いたdynamic linkはプログラムの構造を柔軟にし、ＯＳの設計を極めて容易にした

（３）設計に時間をかけ、詳細な「何故そのような方式を採用したのか」の議事録は非常に有効であった

［Ⅳ］ 5020 TSSを、中央研究所の一般ユーザに解放した結果、発生した問題点は、以下の通り。

（１）大手ユーザとしてLSIのCADプログラムの開発が行なわれた。実記憶領域の数倍のプログラムやデータをのせる結果となった。これは、性能の問題を引き起こした

（２）実用に供しながら、一方でシステムの改善や機能追加をせねばならなくなった

・ このことは研究用計算機システムと実用化のシステムと、２系統のシステムの必要性認識させた

・ ＯＳはどんどん進化する。進化に対応できる環境作りが大切である

（３）やりたかった事で出来なかったことは、「日本語」の計算機による処理であった。「マニュアル」を計算機で管理したかったが、ハードウェア環境が伴わなかった

## ２．３ ＯＳ/ｏ

現在、MC68000を用いたマルチ・マイクロプロセッサベースのマルチ・タスク用オペレーティング・システムを開発中である。

我々のオペレーティング・システムの開発の狙いは、

（１）日本語情報処理や、人工知能の研究用の専用のシステムを構築したい

（２）安価で、高性能で、日本語をベースにしたものを将来のパソコンのレベルで実現したい

（３）現在研究開発中のオンライン手書き文字認識の研究用計算機として利用したい

などにあるが、これらを実現するには、自前のオペレーティング・システムを作成する以外に、我々のニーズを満たすものが存在しないからである。オペレーティング・システムはリソースの管理を目的にする以上、どんどん手を加えられなければ研究用には使えない。したがって、まず、コンベンショナルな手法でコンベンショナルなものを開発することにした。

ソフトウェア工学の知見は必ずしも十分に活かされていない。しかし、以下のことは重点に考えている。

（１）毎年、担当者が卒業し交替して行く環境では、まず第一にドキュメンテーションが大切である。使用できる可能性のあるソフトウェアやハードウェアに関する「仕様書」の類がなくては、拡張も出来ず、「賽の河原」になってしまう。

（２）文書化には日本語ワードプロセッサを積極的に利用する。

（３）学生と教官との参加による「設計仕様」の検討会と結果の議事録は、原則として、日本語ワードプロセッサで作成され、管理している。

（４）システム記述言語として、Ｃ言語を基本として採用した。ＣコンパイラもＣでインプリメントしている。制御プログラムもＣでインプリメントする。クロス・ソフトウェアは東京大学大型計算機センターを利用している。

## ３． オペレーティング・システムの開発への提案

ここ４～５年の間に、ハードウェアのCAD/CAMに匹敵するソフトウェアの設計自動化や、プログラムの自動生成技術は、オペレーティング・システムを対象には完成しまい。したがって、当面、コンベンショナルな方法をリファインしながら進めざるを得まい。

（１）設計は設計者の発想が大切である。発想は「発想の母体となる言語」をベースとする。日本人の場合、K-Ｊ法のような図と日本語で物を考える場合が多いだろう。このような、状態の改善には、まず第一に、「日本語ワードプロセッシング機能」をオペレーティング・システムの開発現場に完全に定着させることである。

（２）上記の定着後、ニワトリとタマゴの関係の面もあるが、次のステップとしていわゆるオフィス・オートメーションの諸機能こそ、ソフトウェアの開発現場で最大限利用すべきである。この段階ではじめて、日本語情報処理とソフトウェア・ツールの有機的結合が果たされよう。また、これによって、オペレーテイング・システムのインプリメンタが一般ユーザの立場に立つことになるため、マンマシンインタフェースは改善されよう。

（３）以下、次のフェーズとして、日本語をベースとした要求記述言語などの研究を行なうべきである。

（４）オペレーティング・システムの諸機能を実現するアルゴリズムの開発と、それをデータベースに登録し、活用できる環境を研究する必要がある。このような研究を通じて、オペレーティング・システムの標準化された機能と、目的に合った機能とから好みのオペレーティング・システムを容易に生成する研究を行なうことが期待される。

## ４．おわりに

ソフトウェアの開発は人に依存する。ソフトウェア工学だけにとどまらず社会科学や人文科学の教育も、システム設計に大きく寄与すると確信している。

## ５．参考文献

［１］高橋延匡，土居範久，益田隆司：オペレーティング・システムの機能と構成，岩波講座情報科学１６，岩波書店，1983.

［２］高橋延匡，武山潤一郎，並木美太郎，中川正樹：ＭＣ68000 用小型ＯＳ：ＯＳ／ｏの開発，計算機システムの制御と評価研究会資料２１，情報処理学会，1983.12.

［３］中川正樹，篠田佳博，藤森英明，高橋延匡：ＭＣ68000 ユニ＆マルチ・プロセッサ・システム用記述言語Ｃ処理系の開発，計算機システムの制御と評価研究会資料２１，情報処理学会，1983.12.

# ソフトウエア工学のＯＳへの貢献

## ――ソフトウエアマネジメントの視点から――

東　基衛

日本電気（株）ソフトウエア生産技術研究所　管理技術開発部

はじめに

ソフトウエア工学とＯＳとの関連を考える時、二つの視点が考えられる。一つはソフトウエア工学がＯＳにどのような影響を与えるかということであり、他の一つは、本パネルのテーマでもある、ソフトウエア工学がＯＳ開発にどのような貢献できるかということである。

前者は、主としてソフト工学の成果である新しい言語やその支援系をＯＳの中に取り入んでゆくことであり、一般にＯＳ屋さんは積極的である。これに反して後者はソフト工学の成果をＯＳ開発過程に取り入れて生産性や品質向上を図ってゆくことであり、ＯＳ屋さんはどちらかというと懐疑的である。彼等は良くこういう。「成程ソフトウエア工学は良いですね。しかし、ＯＳは特別なのです。そう簡単には使えません。」

ここでは、ソフトウエア工学を普及推進してきた立場から、問題を整理してみた。

何が同じか、何が違うか

一般にソフトウエアをソフト工学の立場から見ると次の三つに分類される。

(1) 大学、研究室などで作られ使われる、論理的に正しさの証明が要求されるソフトウエア

(2) 少しづつ違うが全体として見ると類似のものが、いろいろな組織で大量に作られているビジネスアプリケーションなどのソフトエア

(3) 大規模かつ複雑で、類似品が少なく、常に改善されながら永く使われていくＯＳ、軍関係、航空管制などのソフトウエア

このように見ると、ＯＳはソフトウエア工学の重要な一対象分野であることがわかるが、そのソフトウエア工学から見た特色は次の３点に集約される。

１．大規模かつ複雑である。

２．現存するソフトウエアとの連帯性、親和性を保ちつつ大巾な機能追加、改善が行なわれる。

３．クリテイカルな性能が要求される。

何が問題か

これらの特色からＯＳのためのソフト工学は、検証や自動化などの分野で実用化することが如何に難しいか容易に理解される。しかし今日のソフトウエア工学はそれだけではない。ソフトウエア工学のＯＳへの適用を難しくしているのは人間の問題、テクノロジートランスファーの問題なのである。

ＯＳは歴史も長く、高度な技術を要するため、その開発者は誇り高い専門家集団である。このような集団にはいわゆるＮＩＨ（Not Invented Here）の原則が働く。彼等は外部から指導されることを潔しとしない。

ハードウエアの場合には、一般にインダストリアルエンジニアが工程の改善を行う。しかし高学歴社会である日本では、ＱＣサークル活動やＶＥ提案などを通じて改善を行って成功している。更に高知識労働者集団であるＯＳの場合には、

一層外からの改善でなく、中からの自主的な改善活動を盛り上げてゆかなければならない。

何を行ってきたか

ＮＥＣでは１９７８年より情処グループのソフトウエア効率化に取り組んだ。これはＡＳＰ（応用ソフトウエアプロジェクト）ＢＳＰ（基本ソフトウエアプロジェクト）に分かれて行なわれた。ＡＳＰが主として構造的プログラミングの検討など、ソフトウエア工学の分野別に研究を行なったのに対して、ＢＳＰは顕著な差異を示した。キロステップ当りのバグ数や工数をいつまでに半分にするというように目標を立てて、何をするかはラインにまかせようというのである。

その後これは、ＳＰＣ（ソフトウエア効率化分科会）という全社活動へと発展して展開された。ここでは、コストモデルによる見積りと原価管理、作業環境の整備、パソコンによる管理ツールの開発とシステム化、教育の整備などがとり上げられた。

一方、１９８１年より全社的に推進しているＳＷＱＣ（ソフトウエア品質管理）グループ活動は、ＯＳ担当部門でも熱心に行なわれ大きな成果を納めてきている。

今後何ができるか

これまで行ってきた、ソフトウエア工学の普及推進のための活動のアプローチは、大きく分けると次の４つに分けられる。

(1) トップダウンによるもの
(2) ＳＷＱＣなどボトムアップによるもの
(3) 専門部会、委員会、研究会など組織間にまたがるＳＩＧ(Special Interest Group) によるもの
(4) 教育、論文大会、講演会などの形式によるもの

これらの形式は互に補完関係にあるため、バランスよく併用してゆくことが重要であり、そのためには中心となる専門組織の存在が不可欠である。ＮＥＣではソフトウエア生産技術研究所がその役割を果している。

技術分野から見るとこれまでの所、成果を収めているのは、ソフトウエアマネジメントの範ちゅうに属するものが中心を成している。この他にはＳＷＱＣによるミクロな技術が集積してきている。

仕様化技術はＨＩＰＯやＳＰＤなどが試みられ受け入れられてきている。しかし、正しさの証明のように数１０ステップのものに対してのみ成果が発表されているような技術は採用され難い。

今後急速に発表させてゆかなければならない技術は特に開発におけるコンピュータの活用であろうう。この分野では、ＣＡＲＥ、ＣＡＤ、ドキュメント管理、カバレージテストやダイナミックテストなどのＣＡＴなどが有望視されている。

# ソフトウェア工学のOS開発への寄与

大場　充
（日本アイ・ビー・エム　サイエンス・インスティチュート）

## 1. はじめに

ソフトウェア工学の歴史を再検討してみると，この約15年の間は，現実が理論よりも急速に発展してきた．そのため，つねに理論が現実の後を追うかたちで進歩してきている．J. レーダが今年3月に開催された第7回ソフトウェア工学国際会議の「ソフトウェアの技術移転に関するセッション」で，『ソフトウェア工学の問題は技術移転ではなく，われわれが移転すべき技術をもたなかったことにある』との指摘をしたことに，これがよく示されている．

## 2. ソフトウェア工学以前

ソフトウェア工学が出現する以前にも，比較的大規模なソフトウェアの開発が計画どうりにできないことは知られていた．例えば，F. ブルックスの「ソフトウェアの神話」で有名な，OS360の開発では，労力不足を解消するために新しい労力を追加投入しても，労力不足がなかなか解消できなかった例があった．特にそのテストでは，当初30名前後だったテストチームが，最後には200名以上の大部隊になっていたと記録されている．

このように，ソフトウェア工学以前のソフトウェア工学的課題は，ソフトウェア開発における「工数の見積り」と「正確なスケジューリング」，そして「進捗管理をどう行うか」であった．このため，要求定義から最終（受入れ）検査に至る，ソフトウェア開発の一般的プロセスが提唱され，また各工程での標準的工数の見積り法が研究されていた．

## 3. 構造化プログラミングとソフトウェア工学

E. ダイクストラによって，「goto無しプログラミング」が提唱されたのをきっかけとして，1970年以降様々なプログラミング技法やソフトウェア設計技法が研究されるようになった．例えば，G. マイヤーズの「複合設計法」や，「トップダウン設計法」，そして「段階的詳細化（stepwise-refinement）」や「情報隠蔽（Information hiding）」が有名である．

特に，ダイクストラの「goto無しプログラミング」を発端とした構造化プログラミングの運動は，後にオペレーティング・システムの開発に大きな影響を与えた．1970年頃，オペレーティング・システムの多くはアセンブラ言語で書かれていた．しかし，1973年頃から構造化プログラミング運動の影響で，オペレーティング・システムの一部が，システム開発用特殊高級言語で書かれるようになった．

また，ダイクストラは「形式主義」をプログラミングへ導入することを提唱した．このことは，それまでテストに多大な費用をかけていたメーカに対し，テストではソフトウェアの品質を必ずしも向上させることができないことを警告したのと同等であった．しかし，この形式主義への過度の期待は，ソフトウェアの生産現場の職人達を混乱させただけに終った．

## 4. ソフトウェア・メトリクス

構造化プログラミング運動とほぼ時を同じくして，ソフトウェアの信頼性に関する議論，複雑性に関する議論，そして作業量および生産性に関する議論が，実務家達の関心を集めだした．これらの議論は当初，理論的なモデルに関するものが多かったが，序々に実務家達の協力で，現実のデータとの適合性が主題となるようになった．

ソフトウェア・プロジェクトにおいて様々なデータが記録・保存されるようになり，次のような事実が確認された．単位行数当りのエラーを含む行数の平均値のバラツキが小さいこと．総コーディング行数と複雑性尺度との間に大きな相関があること．そして，単位時間当りのコーディング行数の平均値はバラツキが大きいこと等が判明した．

ソフトウェア・メトリクスで提唱された理論やモデルは，一般のソフトウェアエンジニアが毎日の仕事に応用できるほど便利なものではなかったが，現実をデータの観点から問いなおす文化を育てたことは重要であろう．

## 5. 大規模化と保守：ソフトウェア工学10年の現実

ソフトウェア工学が生れて10年後，ソフトウェアは大規模化の急速な変化の過程にあった．大規模化は複雑化の原因であり，複雑化はテストの増大化の原因となる．従って，今日ではテストをいかに効率よく行えるかが開発のカギとなる．現在では，設計作業の2倍の労力がテスト作業に消費されていると言われている．

テストと同様，今日ではソフトウェアの保守も大きな問題である．ソフトウェアが大規模になると，1つのモジュールのもつインタフェースの数が増大する．従って，同一のモジュールの保守に必要となる知識の量も，大規模になるほど増加する．また，ソフトウェアが大規模になってくると，ソースコードの1つの変更が他の処理にどのような影響を与えるかを，予知し時前評価することも困難になってくる．

## 6. まとめ

ソフトウェア工学の約15年の歴史をみると，1つの大きな主題が底辺にある．それは，ソフトウェア技術の移転である．よい理論やアイデアも技術移転を失敗すると何も残らない．また，ソフトウェアは「見えない実体」であるため，その技術移転はハードウェアよりもむづかしい．このため，この15年間に成功した技術移転は比較的単純で明解な技術が多かった．また，実験と実証が比較的容易な技術が多かったことも注目に価しよう．

# オブジェクト指向言語と オペレーティングシステム設計の親和性について

上林　憲行
富士ゼロックス株式会社

## 1　オブジェクト指向言語について-Smalltalk-80の場合-

ここでは、オブジェクト指向言語として**Smalltalk-80** を前提に議論をする。Smalltalk-80は、厳密には、オブジェクト指向言語の定義にもよるが、オブジェクト指向ではなく、*Everything is an object*の考え方を徹底しているので、他のオブジェクト指向言語とは、本質的に一線を画すものである。Smalltalk-80が提供するソフトウエア工学(プログラミングの方法論)上のパラダイムは、以下のとおりである。

1)*Everything is an object*概念　→作用(communication)の対象の抽象化･一元化
2)*Message passing*概念　→対象間の作用(communication)の抽象化･一元化
3)*Class and Instance*概念　→抽象と実体の概念によるモデル表現能力
4)*Subclassing*概念　→Classを形成する環境･リソースの継承能力

Smalltalk-80は、『設計』にまつわる複雑さをいかに解決し管理するかといったソフトウエア工学の本質にせまる一つのアイデアを具現化したものとして評価すべきである。Smalltalk-80は、ソフトウエアの創造主である設計者から、この複雑さ(問題の、開発の、保守の)を管理する負担を軽減する点に関して、重大な配慮がなされている。

a)実世界のモデル化能力の基本的なエッセンスをプログラム言語に内包させたこと。
b)プログラミング、ユーザ･システムインターラクション等のソフトウエア設計のあらゆる過程で、統一的な概念モデルを提供したこと。
c)ユーザインターフェースにコストをかけ、デザイナサイドの実際的な作業時間を軽減するとともに、その心理的な負担を取り除いたこと。
d)システム自身が、デザイナの試行錯誤的なプログランミングプロセスを熟知(履歴を採っており)しており、必要に応じて、その情報および過去に逆のぼって状態を再現できること。

こうした、配慮の体系化によつて、Smalltalk-80は、Unified Programming Environmentとしての体裁が整ってきたわけである。Smalltalk-80には、従来のOSの概念的延長上のOSは存在しない。今後のパーソナルコンピュータの時代に、OSとアプリケーションと言う図式は、システムの階層概念として本質的であるのか、おおいに疑問である。

## 2　オペレーティングシステムの構造とオブジェクト指向言語の整合性

オペレーティングシステムの構造的な設計法は、1960年代後半のT.H.EやMuticus等、1970年代初めの、Hydra等の研究で一つの成果を挙げた。その当時のOS設計の複雑さの要因は、おもに、多重プログラミングに起因するリソース管理や雑多なI/O機器の制御等であった。T.H.EやMuticus等は、OSを数レベルの仮想マシンの階層構造として設計の抽象化が可能であるという点を示唆している。また、Hydra等では、OSのコンポーネントは、いわゆるオブジェクトとして体系化する方法が 採用されている。仮想マシンの考えは、ある機能性を実現する幾つかのオブジェクトをまとめて抽象化を図ったものであり、Hydra等では、OSのコンポーネントとしてのオブジェクトをさらに詳細に抽出を図ったものと理解できる。特に、OSのカーネルの設計に関しては、こうしたカーネルが提供するデータベースに関して、その操作を規定してゆくアプローチ(データエンカプセレーション)が有効であると評価されている。カーネルが対象とするオブジェクトは、例えば、プロセス、メモリ、シクロナイゼーションポート、ケイパビリテイ等が挙げられる。こうした、研究の一つの到達点が、インテル社のiAPX-432における、OSカーネルオブジェクトのファームウエア化である。iAPX-432では、こうしたOSカーネルオブジェクトに対しては、オブジェクトベース

のインストラクションを提供しており、この点に関しては、オブジェクト指向アーキテクチャが実現されていると考えてよい。このように、OSの構造、挙動のモデル化は、オブジェクトによるモデル化手法と一致する。この点は、非常に重要な事である。また、このオブジェクトの考え方を、単に、概念設計レベルのモデル化の方法論として使用するのでなく、直接、そのOSのモデルを、プログラミング言語レベルで表現できる点が大きな利点である。このように、OSの設計においては、オブジェクトプログラミング言語は、親和性が高いと評価できる。Cuncurrent-PascalとSolo、HydraとAlphardの関係に見られるように、OSと言語の一体感は、大きな潮流となってきている。その意味で、Smalltalk-80は言語とOSの枠を越えたシステムの在り方を示すものとして革新的なものである。

Smalltalk-80の、*"Everything is an object*概念*""Class and Instance*概念*"*は、OSにおけるオブジェクトの設計に有用である。OSにおけるオブジェクトは、Smalltalk-80では、まさにobjectとして表現できる、さらに、そのobjectの挙動の抽象的な表現は、Classとして、表現される。さらに、実際に、システムでactivateされる、挙動の主体は、instanceとして扱われ、そのinstanceに対するアクションは、messageによつて起動される。このように、Smalltalk-80のモデル化能力は、OSの構造の実際のモデルと高い親和性がある。

OS設計に関して、オブジェクト指向言語の親和性は、つまり、

1)オブジェクトの表現能力　→OSはまさに機能オブジェクトの集合
2)instanceの明示的な役割　→実際の挙動の主体はアロケートされたデータベースを核とするオブジェクト、またインスタンス生成(初期化)を基本とするのでプログラムの初期化、挙動する環境およびリソースが独立してアロケートされるので、そうした内部リソースの管理の単純化が図れる。

Smalltalk-80等では、仮想マシンの提供する機能性の制約の範囲であれば、通信プロトコルを処理するような内部状態の保持およびプロセス同期の能力が要求されるOSを、上記の理由で、非常に効率良く開発が可能である。

## 3　パーソナルコンピュータ指向のOS-プログラミングシステムについて

大型機におけるTSS･多重プログラミング指向OSと異なり、今後のパーソナルコンピュータベースのシステムでは、多重プログラミングおよびケイパビリテイのサポートは本質的ではなくなり、従来のOS設計の複雑さの要因がない、新しい概念のOSが期待できる。この場合、OSはプログラミング言語と独立に存在するのではなく、OSの仮想マシンを基底アーキテクチャとするOS自身の記述･表現を有したプログラミング言語とが融合したシステムを模索する必要がある。この場合でも、Smalltalk-80等が提示するパラダイムは、OS開発の強力な指針となると思われる。

参考文献

[1] Goldberg,Aet al.,.,"Smalltalk-80 :The Language and Its Implementation,"Addison-Wesley 1983

[2} Wulf,W.A et al.,"HYDRA: The kernel of aMultoprocessor Operating System", Comm.ACM, Vol.17, No.6, pp.337-345

[3} Kamibayashi.N et al.,"HEART: An Operating System Nucleus Machine Implemented By Firmware", Proceeding of Synposium on Architectural Support for Programming Languages and Operating Systems,1982

[4] Intel Coporation,"iAPX-432 mamuals'

# 擬似コードを用いた設計手法

山地克郎　富士通(株)

## テーマの設定と背景

この15年の間にソフトウェアの危機を乗り切るべく，幾多の工夫と努力がソフトウェア・エンジニアリングと呼ばれる分野で繰り広げられて来た。試行錯誤を含めた，それ等の試みの中には，消え去ったものも多いが，そうそう根を張り始めたものもある。

ソフトウェアの開発という場面において，決定的に重要なことの一つは，人間の思考方法とプログラムの表現方法の近接化にある。人間の論理的思考過程は母国語に依っているが，一方オペレーティング・システムの各コンポーネントのための現時点でのプログラミング言語による表現は，日本人が思考に用いる言語とはかなり異なる。ここにおいては，勿論文字そのものよりも文法の差が，より本質的である。構造化プログラミング論議以来，プログラムは連接，反復，選択程度の基本処理に限定すべしという考えが根づいている。ここに到って両者歩み寄りの余地が生れた。一般のソフトウェア・エンジニアにとって然程異和感を持つことなく，論理思考にかなりの程度は耐え得るであろう擬似コードを設定したことの意味は，そこにある。更に，このことにより，

- 設計書とプログラムの近接化（トップダウン・プログラミングへの一つの道でもある）。
- 設計書，プログラムの執筆，編集効率の向上。
- 管理データの精度向上，收集／分析の容易性。

等の効果も同時に得られることとなった。

## 擬似コードによる設計／プログラミング

オペレーティング・システムを構成するプログラムのバグ分析によれば，開発者が検出するバグの内，かなりの部分（4割弱）が設計論理をソース・コードに変換する際に作り込まれることが分っている。構造化プログラミングへの道という発想とは別に，上記状態の改善を試みんとするのは当然の成行きでもある。

## 擬似コード

擬似コード（pseudo code）は，プログラムの処理論理の記述表現の一つである。処理論理の設計においては単純かつ明確な論理の組合せに基づくことが重要である。擬似コードでは処理論理における制御フロー構造を連接，反復，選択の三つに限定し，この三種の基本型をソース・コードの記述言語に近く，かつ，計算機に容易に入力可能な簡易な言語で表現したものである。

## 擬似コードの利点

①人間の頭の中にもやもやとして発生する設計のイメージを，人間の思考をあま

り混乱させない程度の表現形式でドキュメント化できる。
②最終生産物と同じ形式での設計が可能。これにより，工程フェーズにより表現形式が変わることによるバグの混入を回避できる。
③ソースコード・イメージでの設計チェック／レビューが可能。
④計算機への入力，編集が容易。

## 擬似コードの効果

フローチャート，HIPO等の手法による設計に比べ，擬似コードの場合には詳細設計工程も含めて，よりソースコードに近い型になるため，以下の効果が得られる。

①品質の早期向上
コーディング以前の論理思考が，より自然で入念になるため論理ミスの発生率が減少し，かつ，設計論理をソースコードに変換する際の誤りの発生を未然に防止できる。ソースレビューにより検出したバグの内，設計論理が正確には反映されなかったものの割合は，

従来の設計手法　：　30％
擬似コーディング：　5％

であった。

②歩留りの向上
詳細設計がソースコードに近い型で行われるため，ソースコードへの変換が容易であり，歩留りが向上する。

従来実績の平均　　：　56％
擬似コーディングの平均：　70％

(注)歩留り＝(完成時ステートメント数)／(投入ステートメント数)＊100

③開発規模見積り精度の向上
詳細設計完了時点での見積りと，実現されたプログラムの規模の差は，

従来の設計手法　：　20％
擬似コーディング：　5％

④端末利用の設計
全ての詳細設計が端末からライブラリ管理システムを用いて可能となる。この結果，編集／更新の効率は格段に向上する。

## ソフトウェア生産管理システム

ソフトウェアの開発過程で発生する多くの管理対象データを自動的に収集・管理し，必要な時点で分析容易な形式(多くはグラフ)で計算機により編集・出力し，品質および工程進捗の予測精度が向上した。また，これによる管理工数の削減も無視し得ないものである。

ソフトウェア・エンジニアリングの技術の進歩は決して速くはないが，止っている訳でもないというのが筆者の実感である。なお，本稿は筆者同僚の益本調査役の研究と協力が多大であったことを申し添えて結言とする。

以上

# パーソナル・コンピュータのＯＳへの影響

鈴木則久（東京大学工学部計数工学科）

今のパソコン（本体５０万円以下のもの）の売れ行を考えると、近い将来大部分の計算はパソコンで行なわれるであろう。そういう意味から、パソコンのＯＳを正しく作ることは社会的影響が大変大きい。私も最近16ビット・パソコンを買って使っているが、まだまだ改良すべき点が多い。パソコンの上にSmalltalkを動かすというような革命的な要求はここではしないことにして、現在のＯＳ上で、どうしたら使いやすくなるであろうか。

現在のＯＳでは次の３点が最も不便である。

1．フロッピー・ディスク・システムが遅い（ＣＰ／ＭとかＭＳ－ＤＯＳで）。
2．いろいろな事をやると、フロッピーを沢山持っていなければならず、管理が大変。
3．他の機種、特に大・中型機、とのファイルの転送が困難。

フロッピー・ディスク・システムが遅いのは、フロッピー・ディスク・ドライブが遅いからではない。現にパソコンの上で走る、ワープロ専用ソフトで専用のファイル・システムを持っているものには、大変速いものがある。これはパソコンの標準ＯＳのファイル・システムが手を抜いて作られているからに違いない。ファイル・システムが遅いのは、多くのページへのアクセスが起こるからで、ディレクトリの情報を主記憶にキャッシュしたり、ファイルのページ・マップをうまく作れば、アクセス数は減るはずである。

パソコンを持っているが故に、何１０枚ものフロッピーを持っていて、管理していなければならないのは大変骨が折れる。ハード・ディスクかオプティカル・ディスクが解答かもしれないが、これでは高くなるし、ファイルの共用はできない。やはりイーサーネットの様な高帯域ネットワークにつなげるのが最良の解であるが、これもまだ高いし、家庭用には役に立たない。

そこで、ファイル転送の問題も同時に解決する、次のようなシステムを提案する。まずひとつは、ネットワーク・コントローラをネットワーク側に含めたようなネットワーク網の実現である。現在ネットワーク機能のついたパソコンでは、コントローラはパソコン側についていて、ネットワーク側には、トランシーバーがついている。これだと、パソコン側の値段が高くなる。そこで、コントローラまでもネットワークに含め；パソコン側にはバスへ直接つなげるようにして、時々高速・高容量の情報転送が必要なときに、このようなコントローラを持った所・（ガソリン・スタンドのように町中にあって，パソコンを持っていって充電するように、情報を交換できる。）へ行くようにしたらどうだろう。このときパソコンは携帯型で、メモリーはＣＭＯＳで、ネットワークにつないでいない時は、つないでいた時にすい込んだシステムやファイルを電池で常時記憶しておく。また普通家庭で使う時は、電話線を夜寝ている間に使って、変更したファイルをどこか自分の大ファイル・システムがあるセンターに書き込み、必要なシステム・プログラムをすい込むようにする。