「オブジェクト指向プログラミング」，その起源からの検証

山　本　雅　昭

目　次

1　本論文について
2　Simula：オブジェクト指向言語の誕生
3　Smalltalk
4　C++
5　オブジェクト指向とクラス指向
6　オブジェクト指向プログラミングの方向性
7　結びにかえて

1　本論文について

「Object-Oriented」なる語が学術論文，一般論文，コンピュータ関連雑誌，及びソフトウェア工学書に頻繁に登場し始めた時期は，世界的には1986年以降からであり，日本国内では若干遅く，この翌年以降からである。「Object-Oriented」はSimulaが起源となって，そこから派生した初期のSmalltalkがその概念を総称したものであり，1970年代前半には既に存在していた。しかし，1986年にBoochの『Object-Oriented Development』に端を発して，世界的に注目を集めるようになった。この点において1986年は「Object-Oriented元年」であり，この年を境に，「Object-Oriented」

なる語がコンピュータ業界で話題を独占することとなる。

「1986年」から十年が経過した現在，ソフトウェア産業界においてオブジェクト指向は既に新語としての響きを完全に失い，技術的な側面ではある程度の普及したものと認識されている。ただし，これは「オブジェクト指向」が専門技術用語の一つとして一般でも認知され，普及段階への移行の兆候が現れ始めたレベルにすぎない（Yourdon, 1994）。

過去において，構造化開発の一般化が「構造化プログラミング」の普及からスタートしたのと同様に，オブジェクト指向も「オブジェクト指向プログラミング」から一般へ普及し始めた。ところが，現在においても，オブジェクト指向プログラミングの一般への普及に反して，オブジェクト指向開発の普及に大きな進展がみられない。この状況は，構造化プログラミングの普及と構造化開発の一般化が比例関係であったケースとは大きく異なる。企業レベルでのオブジェクト指向ソフトウェア開発が本格化する兆候も現れていないし，本格化するまでにまだかなりの時間を要する。(1)

オブジェクト指向開発とオブジェクト指向プログラミングの関係は，絶対的な相互関係ではない。構造化開発においても，オブジェクト指向プログラミングは選択肢の一つであり，ADT（Abstract Data Typing）の延長線上にオブジェクト指向プログラミングを応用すればよい（Meyer, 1988）。ただし，この現状を問題視するに至るのは，この現状が起こり得ると予測した複数の研究者が，既にその弊害を含めて警告を発していたことにある。(2)「オブジェクト指向」は単にプログラム・コーディングを技術的側面から効率化するためのものではなく，分析/設計段階における「非アルゴリスミック（non-algorismic）」なオブジェクト指向アプローチがシステム開発に齎す恩恵のほうがより重要なのである（Booch, 1994）。現在

(1) Yourdon (1994)は，COBOLの標準仕様にオブジェクト指向技術が拡張されるまで，企業レベルでのオブジェクト指向開発は本格化しないと予測している。

(2) Booch (1994), Yourdon (1994), Meyer (1988), Cox (1990) は，オブジェクト指向プログラミングが先行して普及した場合，オブジェクト指向設計/分析との間に概念的な溝が生ずる可能性を示唆していた。

では多くのプログラム言語がオブジェクト指向プログラミングをサポートするようにはなったが，プログラミング手法のみの習得で「オブジェクト指向」の真意が理解できるわけではなく，又，その真の目的が達成されるわけでもない。残念ながら，オブジェクト指向は構造化と同様にプログラミングから普及し始め，そのことが多くの矛盾と誤認を蔓延させている。

本研究の着手には，これら研究者の予測が，現実問題として浮上し始めたことが起因している。特に重大な現実問題として，以下の三点に総括される。

○入門書や技術解説書を参照し，オブジェクト指向プログラミングを習得しても，オブジェクト指向の概念を明確に理解できない。
○オブジェクト指向プログラミングでコーディングを行ったにもかかわらず，オブジェクト指向システムを開発できない。
○再利用性が向上しない。

これらの問題は実例として，『Pitfalls of Object-Oriented Development』と『オブジェクト指向狂詩曲』中において既にある程度詳細されている。しかし，このどちらも技術解説書の部類に属する実用書であり，学術的な見識には全く欠けている。オブジェクト指向は学術研究の結果から導き出されたものであり，数々の学術研究を経て実用段階レベルに到達した。この後詳細するが，オブジェクト指向は単に新しいプログラミング技法として，学術研究者に注目されたわけではない。一般普及の段階において，単なるプログラミング技法として技術解説を行えば，実際には「オブジェクト指向」そのものの意味が極めて曖昧なものとなる。そして現時点において，「オブジェクト指向」がいかに曖昧で，かつ難解なものとなったかを『Pitfalls of Object-Oriented Development』と『オブジェクト指向狂詩曲』が代弁している。

「オブジェクト指向」が注目を集め始めた1980年代後半，多くの研究者

達は「構造化開発」の普及時の失敗を繰り返さないために，オブジェクト指向分析/設計をオブジェクト指向プログラミングに先行させるべく努力してきたが，予想通り，ソフトウェア SE を代表とする技術者達の多くはオブジェクト指向プログラミングを，「オブジェクト指向」の入り口として選択している。プログラミングを「オブジェクト指向」の第一歩に選ぶことに否定的であるべきではないが，各技術者が構造化開発の経験の中で蓄積してきた「アルゴリズム重視」の思考と知識をベースにし，オブジェクト指向プログラミングの習得の過程において，オブジェクト指向の潜在的概念である「非アルゴリスミック」な思考を同時に習得できる技術者が数多くいるとも考え難い。

研究目的

厳密には，オブジェクト指向プログラミングは，「プログラム言語」，「プログラミング技法」，「オブジェクト指向設計」の三つが複雑に絡み合う。プログラミングを「オブジェクト指向」の第一歩とする場合，ここに大きな落とし穴がある。技術解説書はプログラム言語とプログラム技法を詳細するが，同レベルでオブジェクト指向設計を詳細していない。反対に，オブジェクト指向設計は，手法間での標準化が進まず，所謂「乱立状態」にある。

本論文の目的は，オブジェクト指向技術（オブジェクト，クラス，カプセル化，継承等のサポート技術）の技術論から離れ，オブジェクト指向プログラミングの習得に不可欠な基本概念と，その土台となる思考法について，未定義であった部分を含めて新たに明確にすることにある。これは，総括的には以下の三点を指す。

(a) 「オブジェクト指向プログラミング」の真意
(b) オブジェクト指向プログラミングの必要性
(c) オブジェクト指向プログラミングの方向性

ただし，本論文は研究対象として，オブジェクト指向開発上における「オブジェクト指向プログラミング」に重点を置いている。これは，「オブジェクト指向プログラミング」をオブジェクト指向開発上の一つのステージ（段階）として位置付け，「オブジェクト指向」をコンピュータ上に反映するための作業とツール（「プログラム言語」なる道具）にしかすぎないことを意味する。

研究方法

本研究は上記に従い，オブジェクト指向プログラミングをオブジェクト指向開発の１ステージとして位置づける。つまり，道具の効率的な使用方法を追求するのではなく，「オブジェクト指向」において，この道具が必要となった背景を検証し，この本質を追求している。

以下に，本研究過程の手順の要約を記しておく。

(1) Simula と Smalltalk 誕生当時まで溯り文献調査を行い，「オブジェクト指向誕生」の背景を検証する。
(2) 初期の Simula と Smalltalk において，「オブジェクト指向」は何を意味するものであったか，研究開発プロジェクトの真意を探る。
(3) 現時点で一般的に総称される「オブジェクト指向プログラミング」，及び「オブジェクト指向プログラミング言語」と，初期の Simula 開発チームと Smalltalk 開発チームが創出したオブジェクト指向プログラミングを比較し，相違点を明確にする。
(4) オブジェクト指向プログラミングとオブジェクト指向言語の発展すべき方向性を追求する。

本研究は先ず，過去のオブジェクト指向誕生まで溯り，「オブジェクト指向」の本来の意味を再確認する。オブジェクト指向の起源である Simula

と，「オブジェクト指向」の総称を創り出した Smalltalk は，曖昧となってしまった現在の「オブジェクト指向」を，原点から見据えるためにも必要不可欠な存在である。特に Simula は「非アルゴリスミック」な「オブジェクト思考」を前提にして，このオブジェクト思考をオブジェクト指向としてプログラム可能な「オブジェクト・アセンブリ」の手法を創出した。詳細は本文中にて行うが，オブジェクト指向プログラミングによってコーディングされたソフトウェアが，オブジェクト指向ソフトウェアと呼べるかどうかは，この「オブジェクト・アセンブリ」に代表されるマルチプル・ビュー（Multiple Views[3]）の有無が唯一の判断材料となる。

2 SIMULA：オブジェクト指向言語の誕生

Simula 誕生

Simula 誕生は1960年代まで溯り，シミュレーション用言語の Simula 1 がその産声を上げる。Simula 1 の「Simula」はシミュレーションを指しており，正確には当時の OR（Operational Research）と離散事象シミュレーション（Discrete Event Simulation）を背景に誕生する。

Simula 1 の最大の特徴は，ALGOL 60 をベースにオブジェクトとクラスの概念をプログラム言語中に取り入れた点であり，Simula-67 はその当時で既にオブジェクト指向プログラミングを可能としたことにある。ただし，Simula はオブジェクト指向言語として誕生していない。厳密に言えば，当時において「オブジェクト指向」なる名称と共に Simula は誕生していない。「Object-Oriented」の総称は Smalltalk の基本概念として初めて用いられた語であり，Simula 1 誕生当時にはまだ存在してはいなかった。

---

(3) Simula の「オブジェクト・アセンブリ」と Booch の MV (Multiple Views)は厳密には異なるが，基本的には「静的（Static)」と「動的（Dynamic)」の二つのビューを同様に持つ。

Simula 誕生の背景

Simula は ALGOL ベースであれ，全く新しい概念をプログラム言語に取り入れたのは偶然ではなく，それを要する背景があった。1960年代に利用可能だったプログラム言語は現在と比較すれば，極めて少なく，Fortran に代表される科学技術計算用言語を用いてシミュレーション処理を行っていた。

○第一世代言語（～1959）：Fortran Ⅰ, ALGOL 58, IPL V 等
○第二世代言語(1960年初頭)：Fortran Ⅱ, ALGOL 60, COBOL, Lisp 等

しかし，各プログラム言語とも適正があり，この適正外においては，プログラム言語そのものが大きな制約となることもある。Simula 誕生には正にこの問題が背景にあった。

ALGOL や Fortran に代表される従来のプログラム言語とその設計手法においては，言語特有の処理工程をプロセスとして記述するが，言語構造と記述形式はコンピュータの内部処理に直接的に影響されている。ここで問題となるのは，これらの従来言語では設計とプログラミングが困難なケースに遭遇した場合である。コンピュータの内部処理よりも，人間の思考や認識に沿って設計とプログラミングを行いたいケースもある。特にシミュレーション分野では，一般的な計算シミュレーションではなく，観察目的の再現シミュレーションを行うケースがある。当時，Nygaard (1981) は原子炉設計に取り組んでおり，従来言語と手法を用いての発展性を含めた再現シミュレーションは困難であると判断を下した。Simula 誕生の背景には，離散事象シミュレーションに適したプログラム言語を創造する必要性があった。

Simula の特徴

Simula はオブジェクト指向言語ではあるが，プログラム言語として他

の言語と比較すれば，現在では別段突出した存在ではない。C++ と比較しても，オブジェクト指向プログラミングに対するサポート機能で劣るし，言語自体としても ALGOL からの影響度が高い。

Simula の最大の特徴は言語構造でも，記述形式でもなく，オブジェクト指向プログラミングを可能にしたことそのものにある。ソフトウェア開発は最終的にはコンピュータ（計算機）上で行われるため，単に哲学的な概念や思想では，コードを生成するプログラムに適用することは困難であり，システム思考に準ずるオブジェクト思考をコンピュータ上で直接動作させることは根本的には不可能である。しかも，オブジェクトの創発特性（Emergent Properties）を損なわず，これらをコンピュータ上でそのまま動作させるとなると，オブジェクト思考で動作するコンピュータか，又は，これをエミュレートする仮想実行環境が必要である。[4] そこで，Simula はオブジェクトの抽出と定義する静的なビューと，要求された処理を実行する動的なビューの二つに切り離す手法を採用した。[5] Simula を経て，現在では Booch によってマルチプル・ビューは一つの確立された手法となったが，相対的な二つの思考的概念を結び付ける発想なしに「オブジェクト指向」が現実のものとなることはなかった。現実に，マルチプル・ビュー不在のソフトウェア開発はオブジェクト指向開発ではなく，単に従来のプロセス思考にクラス指向プログラミングを組み合わせたものにしかならない。

オブジェクト思考から Simula へ

Schmucker が行った調査から，オブジェクト指向プログラミング言語の派生とその進化の過程は既に明確にされている。この調査結果において，

(4) 実際に初期の Smalltalk は，このエミュレーション環境に極めて類似する実行環境を開発し，この環境上でのみ動作できた。

(5) 分析と設計の初期段階は静的ビューを基にドメイン（Problem Domain）からオブジェクトの抽出し，その本来の創発特性から各オブジェクトを定義する。実際の処理プロセスはこのオブジェクトを一部利用して行う。簡単に言えば，オブジェクトを幾ら集めたところでプロセスにはならないが，プロセスは各オブジェクトの創発特性を利用する。

オブジェクト指向プログラミング言語の起源が，ALGOL から Simula への派生にあり，Simula から更に Smalltalk へと派生し，以降登場するオブジェクト指向プログラミング言語は「Simula 派生」と「Smalltalk 派生」に大別可能である。この調査結果と分類方法を用いれば，全てのオブジェクト指向プログラミング言語の起源は Simula にあり，以降，派生的に進化し，その数が増加してきたことになる。

現時点において，Simula は単純にオブジェクト指向言語の一つと認識されがちであるが，Simula と「オブジェクト指向」の間に「オブジェクト思考」があることを忘れてはならない。他のオブジェクト指向言語は「オブジェクト指向」を基に発展するが，Simula はゼロからスタートし，結果として現在のオブジェクト指向の基本原理を創造したことになる。目標であった離散事象シミュレーションに適したプログラム言語の創造は，「オブジェクト思考」不在では不可能であり，これをプログラム言語上で実現する努力がなければ達成できるものではない。

3 Smalltalk

Microsoft Windows の Version 3.1（及び3.11）が世界市場を独占する以前，Apple Macintosh は PC にマウスと GUI (Graphic User Interface) を導入し，多くの一般ユーザの指示を得た。この Macintosh の成功を背景に，現状の PC-OS の大半は GUI ベースとなり，又マウスは今や標準としてどの PC にも付属している。

Apple 社は PC 市場における「マウス＋ GUI」の先駆者であるが，実際にはこのどちらも Apple 社が研究/開発したものではない。Apple 設立者の一人である Steve Jobs は，XEROX 社の研究所を訪れ，当時そこで研究/開発中のマウスと GUI に魅了されたのである。この研究所内（略称 PARC: Palo Alto Research Center）において心理学研究者（一部，哲学研究者含む）チームが，幼児の学習能力と学習過程を参考にし，コンピュー

タ上でのポインティング・デバイスと GUI のコンビネーションを創出した。ただし，この研究は「子供達にも扱えるコンピュータ」なる大目的の基に行われた。そして，Smalltalk もここから創造される。

Smalltalk 誕生の背景

Smalltalk は「オブジェクト指向言語」として初めて誕生したプログラム言語である。当時，Simula は既に実用されていたが，Smalltalk の言語構造も記述形式も Simula とは大きく異なり，直接の影響を匂わせる点は極めて少ない[6]。上記において Smalltalk は Simula から派生していると述べたが，表面的に見れば，プログラム言語としての Smalltalk が Simula から派生したとは想像し難い。この違いは，Simula がシミュレーション目的なのに対して，Smalltalk はユーザ・インターフェイス重視であることから生じる。Smalltalk を開発したメンバーの一人は，コンピュータ上でグラフィックをインターフェイスに採用することに，ユタ大学の学生であった時点から興味を示し，Simula の経験を踏まえて，Smalltalk の研究を XEROX において行うことになる[7]。Simula が離散事象シミュレーション用に研究/開発されたのに対して，Smalltalk の研究/開発はユーザ・インターフェイスに注目したコンピュータの進化そのものであった。

Smalltalk 開発プロジェクト・チームが目指したのは，「オブジェクト指向」の確立ではない。このプロジェクト・チームが試みたのは，「数値と文字」のみのコンピュータ上に，「サウンド」と「イメージ」をユーザ・インターフェイスに応用することであり，一つの結果として Smalltalk が創造されたのである。Simula は「数値と文字」中心の典型的なシミュレーション・プロ向けであったが，反対に Smalltalk 開発プロジェクト・チームのターゲット・ユーザは子供達であった。Macintosh は皮肉にも成

---

(6) Smalltalk は Simula 以外の言語（例，FLEX）にも強く影響されている。

(7) Smalltalk が Simula から大きな影響を受け，Simula から派生していることは，Goldberg 自身が明確にしている。

人層からの強い支持を得るが，現在の一般普及型 PC のユーザ・インターフェイスの全てがこの影響を受けている。

Smalltalk の特徴

Simula と Smalltalk の間には，オブジェクト思考上において大きな違いがある。Smalltalk のマルチプル・ビューは，Simula とは多少異なり，静的ビューの比率が極めて高い。Smalltalk は全てをオブジェクトとして取り扱う。これは Smalltalk のプログラム言語としての特徴の一つではあるが，コンピュータ上で動作可能にするため，厳密にはプロセスであるものまでオブジェクトとして利用するよう強制している。この点が，Smalltalk がオブジェクト指向理想主義の象徴と称される最大の要因であろう。しかし，これは記述上においての問題であり，Smalltalk であってもマルチプル・ビューを応用しなければ，コンピュータ上で実行可能なソフトウェアを構築することは不可能である。

Smalltalk が Simula から受けた影響は極めて大きいが，言語構造と記述形式等は Simula から離れ，独自の世界を築くに至っている。例えば，以下に Smalltalk の実行環境の特徴を上げるが，これらを参照しただけでも Smalltalk が何故独自の世界を築くに至ったかは十分に察することが可能であろう。

(a) ユーザ・インターフェイス
(b) 多重 Windows
(c) アイコン
(d) プルダウン・メニュー
(e) ポインティング・デバイス（マウス）

上記の特徴は，Macintosh のユーザ・インターフェイスと，現在の主要 PC-OS が標準でサポートする機能である。結果として，Macintosh を含

めて大多数の PC-OS 及びワークステイションのユーザ・インターフェイスは Smalltalk の実行環境が OS として進化したものであり，コンピュータの一般普及への基盤を創造したことになる。又，Smalltalk はオブジェクト指向プログラミングの観点においては，その GUI 構築に対する有効性を実証したことにもなる。ただし，オブジェクトとクラス，オブジェクト思考等，概念的な部分を創造したのは，Simula であることには間違いない。仮に Simula が創造されていなければ，Smalltalk も存在せず，Macintosh も生まれていなかったかもしれない。この点を熟考すれば，Simula から Smalltalk への派生が，コンピュータの短い歴史の中でも極めて重要なことであることが分かる。

Smalltalk のプログラム言語としての特徴は，ターゲット・ユーザを子供達に設定したことにあった。このユーザ設定は他のプログラム言語とは大きく異なり，言語構造と記述形式を従来のプロセス指向から，より直感的，かつ簡素化された新たな指向性を必要とした。このため，開発チームは単なる技術者の集団ではなく，心理学者，哲学者，医学者等の研究者も含めた研究チーム編成で，十年にも及ぶ結果の一つとして，「オブジェクト指向」から Smalltalk を創出した。

Simula から Smalltalk へ

Simula は離散事象シミュレーションに特化したプログラム言語であった。Simula はシミュレーション用の道具（ツール）であり，この観点上においては，Smalltalk を GUI 重視の子供向けに開発された道具であった。ここにおいて再確認すべきは，この道具（Simula と Smalltalk）が「オブジェクト指向」であったのではなく，オブジェクト指向をコンピュータ上に反映させる道具として，オブジェクト指向言語が必要であった事実である。この点を見落とせば，「オブジェクト指向プログラミング」の意味は極めて不透明なものとなる。Simula と Smalltalk の開発/研究過程を振り返れば，その目標が「再利用性」ではないことは明白であり，「再利用性」

はオブジェクト指向（正確には「オブジェクト思考」）そのものから得られる二次的な効果でしかなかった。

4　C++

PC 上で利用可能なプログラム言語は年々その数を増している。その中でも最もポピュラーなのは VB (Visual Basic), C++, Pascal であろう。VB は Basic の言語特性とその容易な GUI インターフェイス構築を武器にし，一般企業に広く普及，プロトタイピングを含む業務用応用ソフト開発を中心に広く利用されている。Pascal 系プログラム言語は欧米で高い人気を維持し，従来の DOS ベースの応用ソフト開発の主要言語の一つであったが，ここ数年においては Delphi を筆頭にし，本格的に GUI ベースの PC-OS に移行し始めた。

C++ は上記 VB や Pascal 系言語とはその存在自体が大きく異なる。C 言語そのものは低水準言語であり，強力なタイプ・チェックと厳格な構文法がセールス・ポイントである VB や Pascal 系言語と比較すれば，タイプ・チェックも構文法も極めてルーズであり，どちらかと言えばアセンブリ言語の代用的な存在と受け取られやすい。反面，強力なエディタと開発環境から得られるデバッグ性の高さから，制約の多い高水準言語を嫌う技術者達が C 言語を利用し続けている。

C++ の背景

C++ という言語自体は一般に誤認されているケースが多いため，ここにおいて解説するが，C++ はオブジェクト指向言語ではない。[8] C++ の

(8) 混乱を引き起こす要因の一つに Objective-C の存在がある。Smalltalk 派生の Objective-C に関する限り，オブジェクト指向の概念と関数ライブラリーは Smalltalk 派に属すが，スーパーセットのオブジェクト識別子とメッセージ転送をプリプロセッサで処理する以外は従来の C と同じであり，C++ とは正反対のプログラム言語となっている。

開発者である Stroustrup (1986) が最初に開発したのは「C with Classes」である。Stroustrup が注目したのは，Simula や Smalltalk のオブジェクト指向の概念や「オブジェクト」ではなく，クラスと継承 (Inheritance) の応用性と実用性の高さにあった。Stroustrup はこの「C with Classes」を「C++」へと呼び名を変更したが，オブジェクト指向開発を前提としたオブジェクトの概念定義と C 言語自体をオブジェクト思考に沿う形式には変更していない。単純に言えば，C++ はオブジェクト指向開発用言語ではなく，ADT (Abstract Data Typing) とクラス及び継承を組み合わせ，クラス指向[9]の拡張に重点を置いていた。勿論，オブジェクト指向開発のコーディング段階でオブジェクト指向プログラミングを行なうことも可能である。

C++ の特徴

C++ はオブジェクト指向プログラミングに必須である技術は一通りサポートしている。GC (Garbage Collection) は基本的にサポートされていない点と，平行処理に関してサポートの乏しい点を除けば，技術的な側面においてはオブジェクト指向言語の理想主義を代表する Smalltalk よりも優位に立っている。

C++ は，Smalltalk を理想主義とすれば，象徴的な実用主義のプログラム言語である。オブジェクト指向言語とは異なり，C++ は大量に既存する C コードをそのまま流用することも可能にし，又必要とあれば，既存 C コードを強引にオブジェクト指向プログラミングに併用することも可能にする。つまり，複合型のプログラミング・スタイルを可能にすることで，従来のスタイルを好むプログラマー達には拡張機能的にオブジェクト指向スタイルのプログラミングを可能にできる。この C++ の手法は，

(9) スーパークラス不在でオブジェクト指向プログラミングを行なうケースをクラス指向と総称とする場合もあるが，ここではオブジェクト指向開発に限定されず，従来のプログラミングにクラス・ライブラリーと継承等を拡張的に利用するスタイルを指す。

高水準言語を使用せず，敢えて低水準の C 言語を使用するプログラマー達にとっては，オブジェクト指向プログラミングの所謂「美味しい所」だけを拝借できる重宝な代物なのである。(10)

Simula から C++ へ

「Simula～Smalltalk」と「Simula～C++」，この二つの派生を比較した場合，一つの明確な違いを見つけることができるはずである。この違いとは，C++ は「オブジェクト指向」を何ら発展させてはいない。C++ は貪欲にオブジェクト指向を吸収し，現時点においては，オブジェクト指向技術のサポート度では，Simula は勿論，Smalltalk ですら完全に圧倒している。ただし，これらオブジェクト指向技術の全ては必ずしも必須ではないし，ADA, Modula-2, Mesa, Sue 等の特に貧弱なオブジェクト指向技術しかサポートしていない言語であっても，オブジェクト指向プログラミングは可能である。(11) オブジェクト指向技術のサポート度が，プログラム言語の優劣を決めるわけではない。

C++ の「オブジェクト指向」に対する最大の貢献は，C が汎用言語であり，C 言語でオブジェクト指向プログラミングを可能にしたことである。(12) Simula は本質的にシミュレーション用言語であり，Smalltalk はその独自性と実行環境（Virtual Machine）から，汎用言語と呼べる存在では

(10) C++ や Object Pascal に代表される複合プログラミング・スタイルを可能とするプログラム言語は，「ADT+ クラス及び継承」を応用することによって，従来の構造化開発でも強力な拡張機能となる。ただし，オブジェクト指向開発以外においてオブジェクト指向プログラミングを利用したところで，オブジェク指向分析/設計の基本概念との間に矛盾が生じる。クラス・ライブラリーと継承の利点のみを拡張的に応用する場合は「クラス指向プログラミング」とすべきであろう。

(11) 機能的な不足や技術の不備は，拡張的に付加すればよい。例えば，Rumbaugh et. al.（1991）は，従来言語上でオブジェクト指向プログラミングを可能にする多数の拡張テクニックを紹介している。ここにおいて問題視している点は，言語自体の特性である。

(12) Turbo Pascal も同様に，この点においては大きく「オブジェクト指向」に貢献したことになる。

なかった。この観点においては，C++ はPC/ワークステイション上に，汎用言語として新しい可能性を一般ソフトウェア技術者に与えたと言えよう。

C++ とオブジェクト指向プログラミング

Simula は離散事象シミュレーションを目的としたシミュレーション用言語であったがため，観察目的の仮想現実をシミュレートする作業自体が「オブジェクト指向（思考）」であった。つまり，「観察目的の仮想現実のシミュレーション」上に，オブジェクト指向（思考）の原点があるといっても過言ではない。これは言い換えれば，Simula と同様に，離散事象シミュレーション上において C++ が使用された場合，C++ のサポートするオブジェクト指向技術は，必然的にオブジェクト指向プログラミングに方向づけられる。ところが，C++ や Object Pascal（ここでは，Smalltalk も含む）に代表される PC/ワークステイション上での汎用言語は，オブジェクト指向技術（特に，カプセル化，クラスと継承）を拡張的にサポートしたが，汎用目的において，この道具をどのように「オブジェクト指向」に対して応用するかには，全く無関心である。

C と C++ の間に言語としての本質的な違いはなく，当たり前の事ではあるが C++ は C にしかすぎない。プログラミング言語としての C++ はオブジェクト指向プログラミングの必要機能を備えるが，この点だけをもって，C++ をオブジェクト指向言語と定義することは本質的に不可能であることは，これまでに検証をおこなった Simula と Smalltalk から明白である。どの言語にも誕生の背景があり，言語構造及び記述形式はその背景から創り出される。言語に拡張機能を追加したところで，その言語の本質は容易に変更可能なものではない。又，「オブジェクト指向プログラミング」はオブジェクト指向技術とプログラミング技法が創り出すものではなく，「オブジェクト指向」をコンピュータ上に反映させるための手段にしかすぎない。最終的には，C++ をオブジェクト指向プログラ

ミングの道具として使用できるかどうかは，設計仕様そのものが問題となる。

5　オブジェクト指向とクラス指向

「オブジェクト指向」は，英単語「Object Orientation」の日本語訳である。[13] しかし，この「オブジェクト指向」がこれほどまでに難解な専門用語となってしまった背景はどこにあるのか？

この疑問を解く鍵に，ソフトウェア開発に対する「オブジェクト指向効果」への期待の高さがある。そこで，期待される「オブジェクト指向プログラミング効果」を以下の二つのポイントに要約した。

(I)　生産性の飛躍的向上。
(II)　品質の大幅な向上。

ここで重要なのは，上記二つのポイントは，オブジェクト指向が潜在的に有する「高い再利用性」を前提において達成されると定義されていることである。

純然たる矛盾が生ずるのは，Simula と Smalltalk，そのどちらも「再利用性の追求」したプログラム言語として誕生していない点にある。Simula と Smalltalk，このどちらもが「結果」として創出されたものであり，単なる目的に要する道具（ツール，またはユーティリティ）にしかすぎない。「オブジェクト指向言語」という道具を，「コードの再利用」の方向に利用するのは，純粋な「オブジェクト指向」の目的ではない（正確には，「目的ではなかった」）。C++ や Object Pascal に代表される，「コー

(13)　「Object-Oriented」の日本語訳を「オブジェクト指向」とするほうが一般であるが，「Object-Oriented」は形容詞であり，厳密にはオブジェクト指向の概念的な総称である。

ドの再利用」を大前提としてオブジェクト指向技術（クラス，オブジェクト，カプセル化，継承等のサポート技術）を吸収してきたプログラム言語を用いて，「コードの再利用」を最大の目的とした「オブジェクト指向プログラミング」を解説したところで，オブジェクト指向が難解なものとなるのは極めて自然な結果である。

結論として引き出されるのは，「オブジェクト指向プログラミング」と「再利用」を切り離し，先ず「オブジェクト指向」の概念を徹底することが必要である。ただし，これは純粋に「オブジェクト指向プログラミング」を習得することが目的の場合に限られる。[14]「コードの再利用」を完全に無視し，オブジェクト指向を純粋に遂行した結果，再利用性が副産物として生じないならば，「オブジェクト指向」とその「潜在的な再利用性」を発見することは不可能であろう。

しかし，現時点において純粋にオブジェクト指向の概念を習得するのは，この数年に出版された技術書，発表された文献ともに，「クラス指向」の汚染度が極めて高く，極めて困難である。そこで，Simula と Smalltalk 誕生まで遡り，その背景にあった「オブジェクト指向（思考）」の基本概念を，オブジェクト思考を中心として以下の五点に要約する。

「指向（Oriented）」の意味

「オブジェクト指向」の概念を習得する過程において，先ず，英語の「Oriented」に相当する「指向」が極めて不明瞭であり，この「指向」なる語に翻弄されてしまう。そして，この不明瞭さが，「オブジェクト指向」に対して，何か霧に包まれたようなイメージを抱かせてしまう。

しかし，現実には，この「指向」の部分に然したる意味はない。「Oriented」の語訳は，「指向」，「志向」，「至高」，「重視」等の内どれであっても，「オブジェクト」と組み合わせ，一語となり得れば問題とはなら

(14)　「コードの再利用」が目的であれば，「オブジェクト指向」を習得する必要はなく，クラス指向プログラミングをその目的に合わせて習得すれば事足りる。

ない。結果的に，その語が「オブジェクト」を強調していれば十分である。ただし，英語の「Oriented」，日本語の「指向」にしても，問題となるのはその前にある「オブジェクト」の意味にある。つまり，「オブジェクト指向」の概念の全ては，この「オブジェクト」にある。簡単に言えば，「オブジェクト指向」の呼称をやめ，単に「オブジェクト」だけで済ませることも可能である。

「オブジェクト（Object）」の意味

実際には，「オブジェクト」の意味は極めてシンプル，かつ明解なものであり，辞書に掲載されている程度の意味しか持たない。

ここにおいて，Simula 誕生の背景と，Simula が「オブジェクト指向」をコンピュータ上に反映させえた手法が重要となる。Simula 開発チームが離散事象シミュレーションの分析と設計上で応用したのは「オブジェクト思考（Object Thinking）」であって，Simula でのコーディング技術の部分が「オブジェクト指向」に該当する。補足的な解説をするならば，Simula 開発チームは「全体から部分の役割を定義する」従来の思考では，「部分」に役割が与えられるために，離散事象シミュレーションのような仮想現実の観察目的には応用性の問題が大きい。そこで「部分」の概念を捨て，「オブジェクト」の概念を取り入れた。ここでは，「オブジェクト」を「全体の一部分」として定義しないことが，必要最低限の概念となる。つまり，オブジェクト思考とは，オブジェクトをドメインから抽出し，この定義を行うための概念でしかない。

「オブジェクト」の利用（再利用）

オブジェクト思考上で注意すべきは，オブジェクトに役割を与えてはならないことだけである。ところが，現実には，ソフトウェア技術者にとって，この点が極めて難解なのである。

当然ながら，ソフトウェアが提供するサービスをオブジェクトに割り振

ることができなければ，単なるオブジェクトの集合だけではソフトウェアとして動作可能な構造を形成しない。現実に，オブジェクト思考だけでソフトウェア開発は不可能である。

この解決方法は極めて単純で，オブジェクトに何かさせるのではなく，オブジェクトを「利用」すれば済むことである。つまり，全てにおいて「オブジェクトに〜させる」発想を捨て，オブジェクトの創発特性がプロセス中から利用可能であれば，これを活用すれば事足りる。

「クラス指向プログラミング」と「オブジェクト指向プログラミング」

「オブジェクト指向」と「クラス指向」はプログラミング技法上においては，全く同一である。故に，コーディング段階においてプログラマーが，オブジェクト指向プログラミングとクラス指向プログラミングを使い分けることは不可能である。又，詳細仕様書に従いプログラムをコーディングするプログラマーに，各オブジェクトの仕様を書き換えることは許されない。最終的には，設計者に全てが委ねられる。オブジェクト思考とマルチプル・ビューが設計仕様に反映されてこそ，プログラマーがオブジェクト指向プログラミングでコーディングしていることになる。反対に，この反映がなされていなければ，結果的にはクラス指向プログラミングでコーディングしていることになる。

「オブジェクト」と「創発特性」

オブジェクト思考上において，「創発特性」は必ずしも不可欠な概念ではない。しかし，「オブジェクト指向」と「クラス指向」の違いを明確に把握できていない設計者には必要となる。

複合体である全体においては，その全体性における特性がみられる。オブジェクト思考上では，オブジェクトは「全体」に対して「部分」に該当することになるが，この「部分」の潜在的な特性は，「全体」の特性の一部ではない。例えば，「100 M 走の陸上選手」の足の特性を「走る」と定

義すれば，このオブジェクト「足」は，「走る」という役割を与えられたことになる。しかし，熟考すれば容易に矛盾を発見できるはずである。「足」は走りはしない，「走る」のはその陸上選手でしかない。足の創発特性は，「走る」，「歩く」，「座る」等の人間の動作ではなく，これらの動作を可能にする構造的な特性と一致しなければならない。人間の動作を機能的に分割し（partitioning），オブジェクトに各役割を分担させることは，本質的にはトップダウン方式の構造化設計と何ら変わらない，「（オブジェクト応用型）構造化設計」でしかない。

6　オブジェクト指向プログラミングの方向性

Simula と Smalltalk から創造されたのは，単にプログラミング・スタイルとオブジェクト指向技術だけでないことを忘れてはならない。「オブジェクト指向プログラミング」がオブジェクト指向であり続けるためには，「分析〜設計〜プログラミング」が思考方法論上で一つとなることが前提条件となる。この前提条件が歪められれば，Smalltalk をプログラム言語に用いたところで，コーディング段階で強引に「オブジェクト指向」に引き戻すことは不可能である。オブジェクト指向言語にはオブジェクト指向（正確には，オブジェクト思考）がその背景にある。プログラム言語は結果として，コーディング段階においてオブジェクト指向をコンピュータ上に反映させるための道具にしかすぎず，その道具のみを振り翳したところで，コーディングがオブジェクト指向化されるはずもない。たとえ技術解説書であれ，この点を疎かにはするのは，極めて危険である。

本研究では現状を踏まえ，特に以下の二点が，「オブジェクト指向」の習得において，重要な鍵となるとの結論に至った。

(i)　クラス指向プログラミングの概念の普及

オブジェクト指向とクラス指向はプログラミング・スタイル上にお

いて同一のものであるが，オブジェクト指向分析/設計を経ていなければ，例外なくクラス指向プログラミングとなる。マルチプル・ビューが完全に満足されなければ，オブジェクト思考はコンピュータ（計算機）上には反映されない。コーディング段階で実用主義的に効率を重視する場合には，「クラス指向」の名称を使用し，オブジェクト指向とは別の方向性を持たせた上で，発展的にクラス指向プログラミングを利用すべきである。

(ii)マルチプル・ビューの徹底

オブジェクト指向開発において，マルチプル・ビューはオブジェクト思考をソフトウェア開発に反映できる唯一の手法である。残念ながら，学術研究以外では，このマルチプル・ビューの重要性に対する認識が極めて低い。クラス指向開発の場合を除き，開発方法論上において更なるマルチプル・ビューの徹底が必要である。

総括的には，この上記二点は，オブジェクト指向を本来の軌道に引き戻すことを意味しているが，同時に「クラス指向」も発展的に肯定している。クラス指向は実用主義的な見地に立てば，必ずしも否定されるべきものではないし，現時点ではオブジェクト指向よりも実益を上げている。特に，GUI が必須のサポートとなった現在の PC 環境においては，OS を含めて全てをオブジェクト指向に拘束するよりも，クラス指向のルーズさが逆に技術者には好まれるであろう。つまり，クラス指向は「クラス指向」として今後大きく発展すべき手法の一つであると考えるべきだろう。

反対に「オブジェクト指向」を更に発展的なものとしていくには，開発プロジェクト全体に適用できる「オブジェクト思考方法論（Object Thinking Methodology）」の確立が不可欠である。[15] マルチプル・ビューにおけ

(15) 極めて少数ではあるが，Booch (1994), Yourdon (1994), Jacobson (1992)等の開発方法論は，「分析～設計～プログラミング」を既に単一のオブジェクト思考上にほぼ統一している。しかし，これら総合的な開発方法論であっても，分析段階でのオブジェクト思考と，設計段階でのオブジェクト思考が完全に同一ではない。

る，「オブジェクト思考（静的ビュー）」と「プロセス指向（動的ビュー）」の二層階層は，構造化開発手法に依存してきたソフトウェア技術者達にとって，必ずしも容易に理解可能なものではない。これはマルチプル・ビューが難解であるのではなく，マルチプル・ビューについて詳細な解説を行なっているのは，厳密に Booch (1994), Ingalls (1981), Yamamoto (1993) 程度に限定されており，多くの学術論文でさえ，このマルチプル・ビューには敢えて言及していない。これの理由の一端には，オブジェクト指向ブームの初期において，構造化開発手法を拡張した「クラス指向開発手法」が多数登場したことが影響している。[16] Booch (1994), Meyer (1988), Yourdon (1994), Jacobson (1992) 等のオブジェクト指向開発方法論上においても，この点に関する矛盾に触れているが，「クラス指向」の実用性を考慮し，「オブジェクト指向」の枠から排除しなかった。しかし，上記の結論を下したように，この弊害が表面化している以上，「オブジェクト指向」と「クラス指向」を明確に差別する必要があろう。

しかし，これは逆に，自然淘汰の法則に置き換えれば，Simula と Smalltalk が創出した「オブジェクト指向」は，現在においては最早必要とされていないのかもしれない。企業レベルでのオブジェクト指向開発を本格的な普及段階に導くためには，この点に関する研究も方法論上の課題の一つとして残ろう。現実に，Smalltalk も初期の独自性の追求から，「クラス指向」でも利用可能とする方向に徐々に移行しつつある。[17]

---

(16) この代表的な例は，構造化開発上で利用されていた ER モデリングのエンティティ（Entity）の概念とそのデータ・モデリングに，ビヘビアー（振る舞い）を追加し，オブジェクト・モデルとして利用し，強引に「オブジェクト指向」と称した手法である。

(17) これは Smalltalk を，例えば，Windows 上での開発環境に適応させるためには，Smalltalk の実行環境やクラス・ライブラリー等を Windows 側の構造に適合させる必要性から起こっている。本来の Smalltalk の実行環境である VM（Virtual Machine）を，非オブジェクト指向 OS の Windows 上へ移行するのは，理論的にも困難であり，賢明な方法とも考え難い。この意味では，Smalltalk も生き残りをかけ，「理想主義」から「実用主義」へと方向転換し始めた。

7　結びにかえて

最後に，現時点でオブジェクト指向開発に移行すべきか，本研究の検証の範囲内において結論を引き出す。

最終的には，開発プロジェクト全体が移行する場合，「オブジェクト指向」と「クラス指向」の二つの選択肢が含まれ，また，ユーザの想定も必要であるため，ポイントを五つに絞って，以下の質問形式に置き換えて行う。

(a)　オブジェクト指向開発を選択する理由は何か？

1986年を境にして1980年代後半から始まったオブジェクト指向ブームが災いして，オブジェクト指向開発とオブジェクト指向ソフトウェアは驚異的な勢い偶像化されてしまっている。このブームの間に様々な比較データ等が数多く発表されているが，大多数のものは構造化プログラミングに対するクラス指向プログラミングの有効性を部分的に立証できたにすぎない。[18]

企業レベルでオブジェクト指向開発が本格化するためには，現在まで COBOL でコーディングされていた専門業務処理部分，又は Fortran で計算していた処理等に対しての比較が必要となる。しかし，汎用機を土俵にして，現在汎用機上で動作しているアプリケーションを比較対象にした場合，現時点ではオブジェクト指向プログラミングは間違いなく惨敗する。忘れてはならないのは，C も Pascal も汎用機版が存在しているが，汎用機上では COBOL と Fortran, 又は 4GL が主要言語なのである。つまり，C や Pascal といっ

(18)　構造化プログラミングは特に GUI 等のインターフェイス構築に不向きであり，Smalltalk がオブジェクト指向言語として誕生した背景にこの問題が大きく関係している。GUI をベースにした比較データは，GUI に不向きな構造化プログラミングと，GUI を含むインターフェイスにおいて最大の効果を発揮するクラス指向プログラミングを比較する，極めて偏ったものである。

た PC とワークステイションの主要言語は，汎用機上では COBOL と Fortran との比較対象とはなりえない。C++ や Object Pascal で Fortran と計算処理速度を競うことは無意味であり[19]，汎用機上の主要言語争いで COBOL に敗退した C や Pascal を，単にオブジェクト指向化しただけでは何の優位性も生じない。

重要なことは，オブジェクト指向開発を過信してはならない。オブジェクト指向開発されたソフトウェアが，全てにおいて高い再利用性を発揮するわけではない。又，オブジェクト指向ソフトウェアの再利用性を飛躍的に向上させる手法が確立されているわけでもない。オブジェクト指向開発よりも現状では，構造化開発のほうが，プロジェクト管理，ツール，手法，方法論，技術者等の点において成熟度で優る。現時点では，PC やワークステイション上を除き，オブジェクト指向開発は構造化開発が困難を生じるケースに限定し，主要言語にオブジェクト指向プログラミングをサポートされるのを待つべきであろう。

(b) オブジェクト指向プログラミングが必要な根拠は何か？

オブジェクト指向プログラミングにのみ興味が生じた場合，制約の少ないクラス指向プログラミングを採用する方向を先ず優先させるべきであろう。特に，PC やワークステイションが対象機である場合，主要な選択肢であるプログラム言語はクラス指向プログラミングを採用すべきである。PC-OS とワークステイション OS の多くは，オブジェクト指向ではなく，クラス指向ソフトウェアである。SDK や DDK を利用し開発する必要があるソフトウェアであれば，クラス指向プログラミングが必須である。

(c) 設計手法はオブジェクト指向なのか？

ウォータ・ホール型のオブジェクト指向開発を採用する場合，オブ

(19) オブジェクト指向プログラミングによって速度面の向上を期待してはならない。メッセージ転送のボトルネックが大きく，速度的にはむしろ低下する。

ジェクト指向設計が非常に重要なものとなる。例えば，RDD (Responsibility-Driven Design) はオブジェクト指向設計手法であると位置づける研究者もいるが，RDD は典型的なクラス指向設計手法である。RDD はオブジェクトが本来有する創発特性は無視し，トップダウン方式でオブジェクトの役割を定義する動的ビューのみのシングル・ビュー手法である。オブジェクト指向開発の設計段階において RDD を用いれば，必然的にクラス指向開発に転換されてしまう。オブジェクト指向開発を厳密に行おうとするならば，RDD に代表されるクラス指向設計を排除する必要がある。

(d) 採用するオブジェクト指向開発方法論に矛盾はないか？

一部の統合型開発方法論を除いて，オブジェクト指向の手法同士で矛盾が起こらないかどうか検証しておく必要がある。[20] 上記の RDD が典型的な実例となるが，マルチプル・ビューを前提とした分析手法では，オブジェクトの抽出とその定義に，システム中での役割をオブジェクトに負荷することはないが，RDD では逆にオブジェクトに役割を分担させるアプローチをとるため，分析結果が正しく反映されない。

(e) コード再利用を過信していないか？

オブジェクト指向プログラミングと構造化プログラミングを「コードの再利用性向上を促進する支援技術」の視点から比較すれば，オブジェクト指向プログラミングが優る。しかし，Yourdon (1994, PP. 74−79)の調査を参考にすれば，開発プロジェクト中においてコーディングが占める割合はわずか10〜15%である。[21] つまり，コー

---

(20) 分析手法と設計手法の間に，思考方法論的な矛盾が生ずるケースがある。

(21) この割合は開発プロジェクト・チームの総合的なレベルによって大きく異なる。しかし，Boehm (1983) の調査が裏付けするように，分析と詳細な仕様書を含む設計を重視した開発プロジェクトのほうが，デバッグを含むコーディング時間とコーディング・コストを大幅に圧縮できる。Yourdon (1989) は，このコーディングが占める割合から，開発チームの総合力をある程度評価可能と指摘している。

ド再利用性だけの開発プロジェクト全体に対する貢献度は極めて低い。又，開発プロジェクト全体においてコーディングが三割を大きく超えるようなケースでは，オブジェクト指向開発に移行する以前に，開発プロジェクトそのものを効率化する必要がある。

ここにおいて，今一度熟考すべきは，「オブジェクト指向プログラミングとは何を意味するのか？」であろう。オブジェクト指向プログラミングとクラス指向プログラミングの間に明確な境界線を引くならば，オブジェクト指向プログラミングの必要性すら見えてこない。プログラミングの効率を追求するのであれば，現在と同様にクラス指向プログラミングで事足りる。

オブジェクト指向プログラミングが，「クラス指向」ではなく，厳密に「オブジェクト指向」である意義は，オブジェクト指向開発の中にしかない。オブジェクト指向設計で書き出される詳細仕様書がオブジェクト指向であれば，使用するプログラム言語においてオブジェクト指向プログラミングが可能であればよい。使用するプログラム言語の違いが，開発プロジェクト全体に及ぼす影響は，現実的にはわずかなものでしかない。Biggerstaff と Lubars (1991) が指摘するように，真の意味でオブジェクト指向の再利用性が必要なのは，分析と設計段階においてである。

参 考 文 献

Biggerstaff, T. J. and Lubars, M. D. (1991) Recovering and Reusing Software Design: Getting More Mileage from Your Software Assets, *American Programmer*, March.

Birtwistle, G., Dahl, O-J., Myhrhaug, B. and Nygard, K. (1979) *Simula Begin*. Studentlitteratur, Sweden.

Boehm, B. W. (1983) *Software Engineering Economics*, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ.

Booch, G. (1986) Object-Oriented Development, *IEEE Software*, Vol. 12, 2, February.

Booch, G. (1987a) *Software Components with ADA: Structure, Tools, and Subsystems*. Benjamin/Cummings, California.

Booch, G. (1987b) *On the Concepts of Object-Oriented Design*. Rational, Denver: CO.

Booch, G. (1989) What Is and What Isn't Object-Oriented Design, *American Programmer*, Vol. 2, 7-8, Summer.

Booch, G. (1994) *Object-Oriented Design with Applications*, 2nd ed. Benjamin/Cummings, California.

Coad, P. and Yourdon, E. (1991) *Object-Oriented Analysis*, 2nd ed. Prentice-Hall, Englewood Cliffs: NJ.

Constantine, L. L. (1990) Object-Oriented and Function-Oriented Software Structure, *Computer Language*, January, pp. 34-56.

Cox, B. J. (1987) *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, U. S. A.

Cox, B. J. (1990) There Is a Silver Bullet, *BYTE*, McGraw-Hill, October.

Cox, B. J. and Novobilski, J. A. (1991) *Object-Oriented Programming: An Evolutionary Approach*, 2nd ed. Addison-Wesley, U. S. A.

Dahl, O-J. and Nygaard, K. (1966) Simula—an ALGOL-based Simulation Language, *Communication of ACM*, No. 9, pp. 671-678.

Dahl, O-J., Myrhaug, B. and Nygaard, K. (1968) *SIMULA 67 Common Base Language*. Norwegian, Computing Centre, Oslo.

Dahl, O-J. (1987) Object-Oriented Specifications. In Schriver, B. and Wegner, P. (eds) *Research Directions in Object-Oriented Programming*, MIT Press, Cambridge: MA.

Dijkstra, E. W. (1979) Programming Considered as a Human Activity. In Yourdon, E. (ed.) *Classics in Software Engineering*, Yourdon Press, NY.

Goldberg, A. and Kay, A. (1976) *Smalltalk-72 Instruction Manual*. Xerox Palo Alto Research Center, California.

Goldberg, A. and Kay, A. (1977) *Methods for Teaching the Programming Language Smalltalk*. Xerox Palo Alto Research Center, California.

Goldberg, A. (1978) *Smalltalk in the Classroom*. Xerox Palo Alto Research Center, California.

Goldberg, A. and Robson, D. (1983) *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, Reading.

Goldberg, A. (1984) Smalltalk: The Interactive Programming Environment. Addison-Wesley, Reading.

Goldberg, A. and Pope, S. (1989) Object-Oriented Programming Is Not Enough, American *Programmer*, Vol. 2, No 7-8.

Goldberg, A. (1990) Taming Object-Oriented Technology, *Computer Language*, Vol. 7, 10.

Ingalls, D. (1981) Design Principles Behind Smalltalk, *BYTE*, August.

Ingalls, D. (1983) The Evolution of the Smalltalk Virtual Machine. In Kranser, G. (ed.) *Smalltalk-80: Bits of History, Words of Advice*, Addison-Wesley, U.S. A.

Jacobson, I. (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-wesley, U. S. A.

Kadie, C. (1986) *Refinement Through Classes: A Development Methodology for Object-Oriented Languages*. University of Illinois, Urbana: IL.

Knudsen, J. and Madsen, O. (1988) Teaching Object-Oriented Programming is more than teaching Object-Oriented Programming Languages, *Proceedings of ECOOP '88: European Conference on Object-Oriented Programming*, Springer-Verlag, New York.

Kranser, G. (1981) The Smalltalk-80 Virtual Machine, *BYTE*, August.

Meyer, B. (1987) Reusability: The Case for Object-Oriented Design, *IEEE Software*, Vol. 4, 2, March.

Meyer, B. (1988) *Object-Oriented Software Construction*. Prentice-Hall, New York.

Meyer, B. (1989) From Structured Programming to Object-Oriented Design: The Road to Eiffel, *Structured Programming*, Vol. 10, 1.

Minsky, M. A. (1975) A Framework For Representing Knowledge. In Winston, P. (ed.) *The Psychology of Computer Vision*, NY: McGraw-Hill.

Nygaard, K. and Dahl, O-J. (1981) The Development of the Simula Languages. In Wexelblat, R. (ed.) *History of Programming Languages*, Academic Press, NY.

Nygaard, K. (1986) Basic Concepts in Object-Oriented Programming, *SIGPLAN Notes*, Vol. 21, 10.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen (1991) *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ.

Schmucker, K. (1986) *Object-Oriented Programming for the Macintosh*. Hayden, NJ.

Standish, T. (1983) Software Reuse, *Proceedings of the Workshop on Reusability in Programming*, ITT, Stratford.

Stroustrup, B. (1986) *The C++ Programming Language*. Addison-Wesley, Reading: MA.

Webster, B. F. (1995) *Pitfalls of Object-Oriented Development.* Prentice-Hall, Englewood Cliffs, NJ.

Wirfs-Brock, R. and Johnson, R. (1990) Surveying Current Reseach in Object-Oriented Design, *Communication of the ACM*, Vol. 33, 9, September.

Wirfs-Brock, R. and Wilkerson, B. (1989) Object-Oriented Design: A Responsibility-Driven Approach, *SIGPLAN Notices*, Vol. 24, 10, October.

Wrifs-Brock, R., Wilkerson, B. and Wiener, L. (1990) *Designing Object-Oriented Software.* Prentice-Hall, Englewood Cliffs, NJ.

Yamamoto, M. (1993) *Potential Object Analysis.* Ph. D. Thesis, University of Warwick.

Yourdon, E. (1989) Object-Oriented Observations, *American Programmer*, Vol. 2, 7-8, Summer.

Yourdon, E. (1990a) Auld Lang Syne: in STATE OF THE ART, *BYTE*, McGraw-Hill, October.

Yourdon, E. (1990b) Object-Oriented COBOL, *American Programmer*, Vol. 3, 2, February.

Yourdon, E. (1990c) In Structured Analysis and Object-Oriented Analysis, *OOPSLA'90*, ACM Press, New York.

Yourdon, E. (1994) *Object-Oriented System Design: An Integrated Approach.* Prentice-Hall, Englewood Cliffs, NJ.

吉田弘一郎（1994）「オブジェクト指向狂詩曲」技術評論社。