情 報 シ ス テ ム 50－5
（1994．7．19）

情報技術者の育成 と コンピュータ科学

大 岩 元
慶応大学環境情報学部

情報技術者の絶対的不足から、企業においてはコンピュータの未経験者を数ヵ月教育して、実務につかせることが一般的に行なわれている。この結果、技術レベルが低く、生産過程を定量的に捉えることが出来ず、他の分野に較べて納期や費用の見積が甚だ不正確である。

この事態を少しでも改善するためには、諸外国と同じコンピュータ科学の教育を徹底し、要員の質を保障する必要がある。このための企業内教育として、「プロジェクト体験を目的とするシステム教育」と、それに続くデータ構造、計算理論の具体的な教育案を提案する。

Fostering of Information Technology Engineers in Japan in View of Computer Science

Hajime OHIWA
Department of Environmental Information
Keio University

Because of the shortage of Information Technology Engineers in Japan, employment of novice computer engineers is quite common. They are taught only syntax of a programming language at the employment, and then real software production starts.

Technical standard of Japanese software industru is therefore very poor. The appointed date of delibery is often ignored or ultra heavy overtime work is forced to the engineers. The cost estimation is also very poor.

To overcome these situations, we propose a shortened curriculum of computer science for the employee education. It is composed of three teaching steps:

1) system concept by programming experience,
2) data structure concept through mathematical form compilation,
3) finite automata and regular experssion.

1 はじめに

ハンフリーによれば、日本には2つのソフトウェア産業があるという[1]。1つは少数大企業によるソフトウェア工場であり、もう1つはその他の群小ソフトウェアハウスである。前者は米国の最高の実状に匹敵するように思われるが、後者は彼が見た中の最低品質であるという。

このように評価が極端に分れても、両者に共通しているのはコンピュータ科学の軽視である。プロセス管理の行き届いたソフトウェア工場においても、そこで使われているソフトウェア技術は必要最低限のものだけである。

しかし、物の生産で培われてきた生産管理技術は世界最高であり、これがソフトウェア工場にも流用されて、ソフトウェア工場の一流の成果は世界に誇れるものとなっている。これに対して、生産管理技術の導入されていない小規模ソフトウェアハウスの状況は想像を絶するものとなっている。

日本では情報専門学科においてさえも、コンピュータ科学がほとんど教育されていない場合が多い。この結果、ソフトウェア作成を必要とする企業においては、コンピュータの未経験者を数ヵ月教育して、すぐに実務につかせることが一般的に行なわれている。この結果、日本を代表する企業であっても、悲惨の一語につきる生産現場が多々見うけられる。

能力差がほとんど現われない cobol による事務計算ソフトだけを対象とする場合は、これでもなんとかなるかもしれないが、ワークステーションが技術計算のみならず事務計算にも作われる時代が到来した現在、このような企業内教育のあり方も考え直す必要がある。

一方、日本のソフトウェア技術として優れたものに、要求分析がある。大規模ソフトウェアの場合、関係者が全員参加して仕様を決定していくプロセスは、極めて日本的である。西欧の場合、このプロセスは有能な個人の力量にまかされるので、しばしがユーザーが欲しいものに似てはいるが、使えないシステムが出来てしまう。

こうした要求分析プロセスの方法論として、ＫＪ法がある。我々はＫＪ法のコンピュータ支援ソフトウェア（ＫＪエディタ）を作成し[2],[3],[4]、要求分析等の上流工程に使用してきた。その結果、ＫＪ法のＡ型図解を作成するには、高度のシステム分析能力が要求されることが分った。

こうした能力をどのようにして養ったらよいかは、ソフトウェア業界にとって緊急の課題である。そこで本稿では、情報技術者の人材育成カリキュラムについて一つの提案を行なう。

第1段階はプログラミングを題材とするシステム教育で、可読性の良いプログラムを書かせてから、システム分析を実際にプログラミングを通じて体験させる。第2段階では、プログラミング技術としてのデータ構造を、数式のコンパイルを例にとって教える。第3段階では、計算の1つのモデルとして、有限オートマトンと正規表現の関係を数学的に論じる。この3段階でコンピュータ科学のエッセンスを学ばせようとするものである。

2．コンピュータ科学を基礎とするシステム教育

前節に述べたようなシステム教育は、入社時から行なうべきであると考えて、次に述べるような3段階の教育を入社時に行なうことを提案した[7]。

1．プログラミングを題材とするシステム教育（１００時間）
　　プログラム書法の徹底
　　ソフトウェアの進化を体験させる
　　モジュール化、再利用のためのシステム分析と実現
2．数式の翻訳とデータ構造（５０時間）
　　プログラム内蔵型コンピュータとスタック
　　スタックを用いた数式の翻訳
　　優先順位表を用いた翻訳
　　構文図による数式の定義
　　再帰呼出しを用いた構文解析
　　データ構造の工夫による翻訳機能の強化
3．状態遷移図と正規文法（５０時間）
　　順序回路の抽象化としての状態遷移図
　　正規文法
　　状態遷移図と正規文法の等価性証明

1．は新入社員教育として行なうものである。2．と3．は、入社後1年目と2年目に行なうのがよいと考えている。次節以下でこれらの詳細について述べることとする。

3．プログラム開発体験に基づくシステム教育

「プログラム開発体験に基づくシステム教育」[5]においては、プログラミング言語の文法のみでなく、良いプログラムとは何であるか討論し、ミニ・プロジェクトを通じてソフトウェアの進化を体験させることにより、モジュール化・再利用の効果を実際に体験させる点が特長である。

従来のプログラミング教育の内容はプログラミング言語の文法を教えるだけが精一杯で、良いプログラムを書く事まで教育することは、教師の能力、時間的制約等から不可能な状態である。

我々の提案するプログラミングによるシステム教育は、実際のソフトウェアが時間とともに進化する様子を体験させることによって、システムとしてのソフトウェアの本質を教えようとするものである。まずタイピング教育[6]から始めてC言語の制御構造、関数を教える。この段階では、プログラム書法について強調し、簡単な例外処理やユーザー・インターフェースの設計も行なわせる。続いてミニ・プロジェクトを設定し、学習者が自から仕様を決定し、プログラミングも行なわせる。さらに作成したプログラムの仕様を追加し、ソフトウェアを発展さ

せることによって、ソフトウェアの進化を体験させる。

本カリキュラムの指導方針は以下の通りである。

1．プログラム書法を徹底する。（図１）

他人がプログラムを読む必要性を理解させ、他人に分り易いプログラムはどうあるべきかという観点から指導する。また、関数宣言やプログラムの先頭には、コメントとしてその機能を説明する簡単な文書を書かせる。

2．ユーザー・インターフェースの設計をさせる。

インタラクティブなプログラムを題材として、ユーザーが使い易いプログラムを書かせるように指導する。ユーザーにプログラムの挙動が分るようなメッセージの出力や、見易い出力形式の設計を行なわせる。誤入力の検出も行なわせ、正しい入力が行なわれるまで繰返し入力を行なわせる。使い易さを評価させるために、作成したプログラムは、他の学習者が作ったものと交換させ、互に実行してみることを試みさせる。

3．レビューを行なう。

作成したプログラムについて、公開レビューを行ない、プログラムの仕様、内容、動作等について討論する。

4．設計を行ってからコーディングさせる。

実際にコーディングする前に、ＨＣＰチャートによって「何をどう実現するか」の関係を検討させ、概要設計を十分に行ってからコーディングさせる。

5．漸進的ミニ・プロジェクトを実施する。（図２）

学習者が問題を把握しやすいテーマ（実際に行ったのは金銭出納帳）を選び、仕様の設計と実現を行なわせる。出来上ったプログラムに仕様の追加を行ない、ソフトウェアの進化を体験させる。その際、データ構造、モジュール化、部品の再利用、プログラムの拡張性などの概念を理解させる。

4．ミニ・プロジェクトの教育効果

システム設計の経験が無い人間が、はじめから金銭出納帳のプログラム設計を行なうのは難しい。そこで、よく似た構造を持った住所録を例題として提示し、そのプログラムを学習者に使用させる。続いて学習者は住所録のプログラム構造を理解し、これを金銭出納帳プログラムに改造するためには、どの部分を変更すればよいか、どの部分は利用できるかを考える。

続いて機能の追加を行ない、追加モジュールの設計を行なわせる。その際、出来るだけ局所的な変更だけですむようなモジュール設計を訓練する。

これとは逆に、金銭出納帳のデータ構造を配列から構造体に変更すると、変更点がプログラム全体に及び、かなりの工数がかかる。このような体験を行なわせることによって、初期設計の重要性を学ばせることが出来る。

また、よく似た構造の住所録と金銭出納帳であっても、進化にともなって、モジュール構造を変更した方が都合がよい場合も出てくる。

こうした事態は、ソフトウェア開発では日常的に起こるが、従来の教育では取

り上げられず、オンザジョブ・トレーニングに任されていた。

このカリキュラムで最終的に不合格となる人達は、コーディングは出来ても、システム分析が出来ない人達である。こうした人達は本来、ソフトウェア技術者としての適性が無いと考えるべきであろう。

もう一つ、この教育で重要なのは、教師の質である。C言語の文法に詳しいだけでは、このような教育を行なうことは出来ない。システム設計についての十分な経験がないと、ミニ・プロジェクトの指導は不可能である。

教育に良質のシステム技術者を当てるだけの余裕が、ソフトウェア業界には無い。また、それだけの効果が期待されるカリキュラムも無かったため、新入社員教育は文法が分かるだけの人間によって行なわれてきた。この結果、システム教育にプログラミング教育は役立たないという常識が業界には定着している。しかしながらプログラムほど、小規模であってもシステムとしての性格を持つものはなかろう。この意味から、プログラミングほどシステム教育に適した教材はない。

そこで問題となるのが、教師の確保である。これには、我々が提案するようなカリキュラムを修了した程度の技術者に、リーダー研修として教師をやらせるのが良い。3年目の教育として、新入社員教育の教師をやらせるのである。

教師をつとめるには、学習者との議論を指導する能力が要求される。また、学習者の能力を判断して、適切なアドバイスを与えなければならない。これらの能力は、プロジェクト・リーダーとなるために必要な能力である。これらの能力を、実務でなく教育できるメリットは大きい。

また、教えることによって、自分の持っている知識を確実にすることも出来る。実際、教える人間が一番勉強することになる。この意味でも、新入社員教育の教師を勤めさせることは、良い社員教育となる。

5．データ構造と計算理論の教育

前節のプログラミングによるシステム教育の欠点は、プログラミングの技術的な詳細については議論出来ない事である。こうした事は良いプログラムが書けるようになれば、十分自習できると考えているからである。しかし、アルゴリズムとデータ構造のような深い内容は自習は困難である。

データ構造については多くの教科書があるが、それらはソフトウェア技術として独立にあつかわれていて、実際の応用において、その技術がどのように使われるかを議論したものは少ない。

我々の提案するデータ構造論は数式のコンパイルを題材とする。情報技術者はコンパイラーを日常的に使用しているが、それがどのような原理で実現されているかを理解しているのは、コンパイラーの作成経験がある技術者に限られる。自分の使っている道具がどのような原理に基づいているかを理解していない技術者というのは、日本の情報技術者だけである。

そこで、コンパイルの原理を数式の翻訳を例にとって、教えようというものである。数式の翻訳はＦＯＲＴＲＡＮが提案された時、誰もその実現を信じなかっ

た程、難しい問題であったが、再帰の概念を用いれば現在では簡単に実現できる。そこで、これを例題として再帰の概念とスタックの関係を理解させ、さらに翻訳を実現するプログラム中に代表的なデータ構造とそれを使ったアルゴリズムを紹介することによって、プログラミング技術を向上させようとするものである。

さらに、数学的な概念を教えるために、第３段階として有限オートマトンと正規表現の等価性を数学的に証明する。現在３０代より若い世代は高校でユークリッド幾何を学んでいない。このため、高度な計算は出来ても証明が何を意味するかをよく理解していない。しかし、今後ソフトウェアに対する信頼性の要求はますます高まるものと思われる。こうした要求に答えるには、プログラムの正しさを証明する必要が生じてこよう。こうした目的を達するために、ソフトウェア分野で応用範囲が広い有限オートマトンの理論をとりあげる。

6．おわりに

ソフトウェアプロセスを定量的に捉え得るものにするには、まず要員の質が確保されなければならない。システム概念を教育するのに、プログラミングを用いる方法を提案した。これに続く段階として、数式の翻訳を題材としたアルゴリズムとデータ構造の教育、有限オートマトンの教育を提案した。

これらの教育には、ソフトウェア作成の経験を持つ有能な教師が必要となる。そこで、プログラマーからプロジェクト・リーダーに昇格する段階の技術者に対する教育としてこの役目を担当させることを提案した。

参考文献

[1] W.S.Humphrey、藤野喜一監訳：ソフトウェアプロセス成熟度の改善、xii頁、日科技連、1991年.

[2]小山、河合、大岩：カード操作ツールＫＪエディタの実現と評価、コンピュータソフトウェア、第９巻、第５号、33-53頁、1992年.

[3] 竹田、河合、大岩：ＫＪエディタを用いたソフトウェア設計者の思考過程分析の一方法、第10回ソフトウェア・シンポジウム（ソフトウェア技術者協会）、1990年.

[4] 土屋、塩見、竹田、河合、大岩：カード操作ツールを用いた要求分析と機能設計の事例研究、「利用者指向の情報システム」シンポジウム（情報処理学会）、1991年.

[5] 竹田尚彦、大岩　元：プログラム開発体験に基づくソフトウェア技術者育カリキュラム、情報処理学会論文誌、第３３巻、第７号、944-954頁、1991年.

[6] 大岩　元：ＴＵＴタッチタイピング（キーボード５時間速習ソフト）、岩波書店、1991年、　大岩　元（監修）：５時間10分キー入力習得法、マグロウヒル出版、1990年.

[7] 大岩：ソフトウェア技術者の基礎教育、「情報専門学科のコアカリキュラム」シンポジウム（情報処理学会）、1991年.

```
main()
{
        int     a,b,c ;

        scanf( &a, &b ) ;
        c = a + b ;
        printf( "%d+%d=%d", a,b,c ) ;
}
```

a)　一般的な教科書での加算プログラム

```
/*
**      講座・C言語　　初級
**
**      例題3 :
**        2つの数をキーボードから読み込み、足し算の結果を表示する。
**        2数とも0であったら、終了する。
**
**              composed by Naohiko Takeda
**
**                      ver. 1.00.      1990.4.30.      N.Takeda
*/

#include        <stdio.h>

main()
{
        int     a, b ;
        int     result ;

        /* 2数を入力する */
        printf( "¥n2数の和を求めます。¥n" ) ;
        printf( "    2つの数を入力して下さい >> " ) ;
        scanf( "%d %d", &a, &b ) ;

        while( a != 0 || b != 0 ){      /* 2数とも0なら終了 */
            /* 計算する */
            result = a + b ;

            /* 結果を表示する */
            printf( "¥n¥n ----- %d + %d = %d になりました。¥n", a, b, result )

            /* 2数を入力する */
            printf( "¥n2数の和を求めます。¥n" ) ;
            printf( "    2つの数を入力して下さい >> " ) ;
            scanf( "%d %d", &a, &b ) ;
        }
}
```

b) 本カリキュラムでの加算プログラム

図1　加算プログラムの相違例

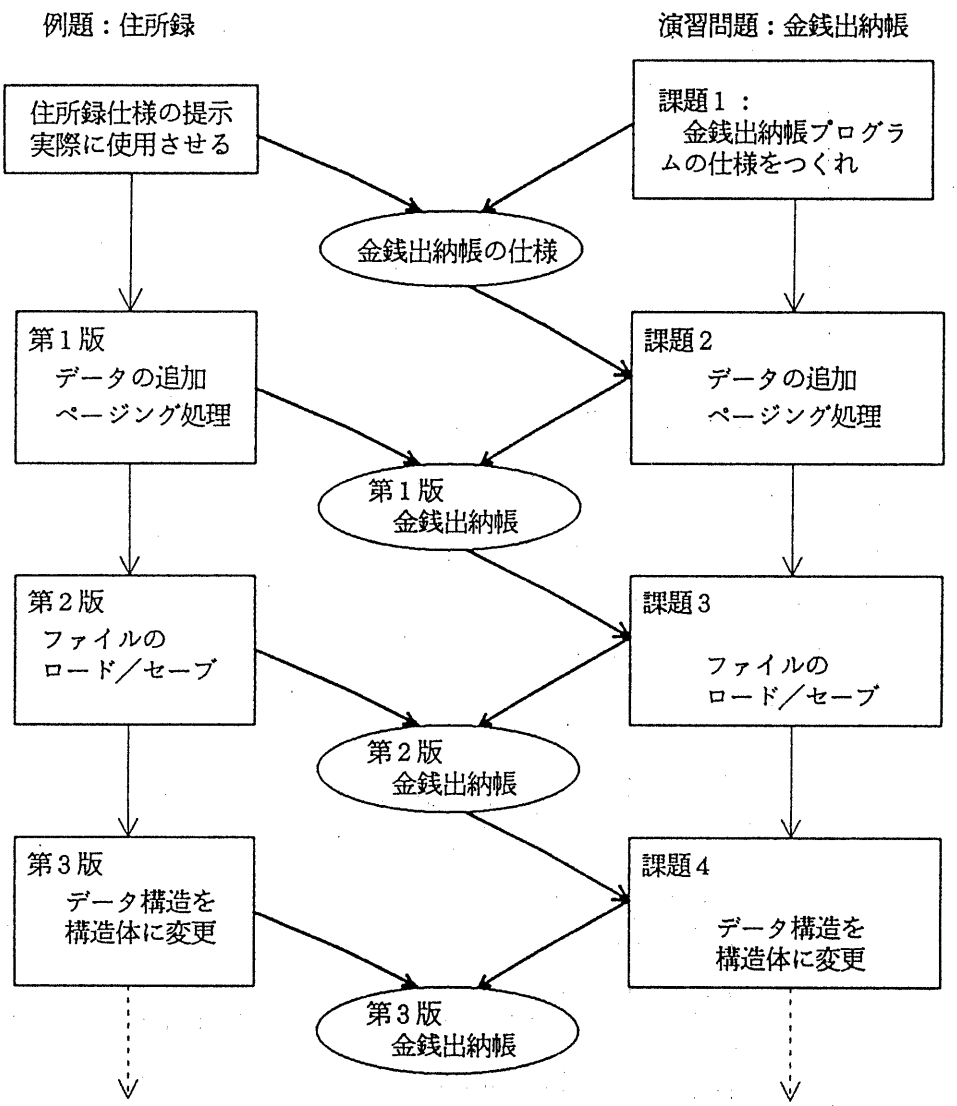

図2　例題と演習問題の関係