プログラム開発体験に基づくソフトウェア技術者育成カリキュラム†

竹田尚彦†† 大岩元†††*

システム設計をおこなうことを業務とするシステム・エンジニア（いわゆるSE）の育成は，従来プログラマに対して，OJT (On the Job Training) の実施によりおこなわれてきた．これはプログラミング教育とシステム設計は別のものであると考えられてきたからである．しかし，小さな練習問題のプログラムでも，独立したシステムとみなすことができる．われわれは，プログラムをシステム的にとらえることにより，『よいプログラマは，よいSEになる』という立場にたつ．こうした立場からSE育成を考慮した初級・中級のC言語カリキュラムを作成した．本カリキュラムは，C言語の文法を教授するだけでなく，1)プログラム書法や例外処理の扱いを徹底する，2)ミニ・プロジェクトにより漸進的なプログラム開発を学習者におこなわせることを主目的に開発した．1)では，プログラム書法や，ユーザにとって使いやすいプログラムについて理解させる．2)では，プロジェクトを通じて，実際の開発現場で生ずるようなソフトウェアの進化を体験させることにより，仕様設計，モジュール分割や部品の再利用について理解させる．本カリキュラムを，約75時間の講義-演習時間で約50人の全くの初心者に実施した．この結果，ミニ・プロジェクトをうまく纏められることのできる学習者には，短時間でモジュールや仕様に対する認識が確立され，初級アプリケーションSE程度の能力が身に付くことが分った．また本カリキュラムを教授するためには，教師が十分なシステム構築経験を持っていなければならないことも分った．

1. まえがき

近年，システム開発の中心を担うソフトウェア技術者の不足が深刻となり，その効果的な養成方法が望まれている．ここでソフトウェア技術者とは，システム設計業務を担当するシステム・エンジニア（いわゆるSE）と，システムの詳細化および実現を担当するプログラマを意味する．

一般的にSEは，プログラマとしての経験を何年か積んだ後，プロジェクトの管理とシステム設計を担当することにより養成される．この時，SE教育としてプログラマに対してOJT (On the Job Training) を実施するという方法をとることが多い．このOJTは，プログラミング技術の向上と，ソフトウェア設計の職務につくための基礎的訓練をおこなうことを目的として実施される[1]．

OJTによるプログラマのSE化の問題点として，

1) 部門ごとに一定した教育を受けられない

2) プログラマの安易なSE化

があげられる．

まず，1番目の問題は，OJTは各社・各部門ごとにおこなわれるため，教育内容が一定しなかったり，同一部門においても指導者の能力により異なる教育をされることがある．このため必ずしも効果的なSE教育が実施されているとは限らない．

2番目の問題点は，プログラマが経験年数により，SEへ昇格をはからねばならない会社組織に由来するものである．ソフトウェア分野においては，作業能率の高いプログラマが，よい設計者になるであろうと予測されている．しかし，作業能率の高いプログラマといえども，設計のセンスやコミュニケーション能力，マネジメント能力が備わっていなければ，よいSEになれない[2]．SEは，総合的な能力が必要とされるため，単にソフトウェア知識だけでは，問題解決ができない場合が多いからである．

一方，SEには事象を系統的にとらえ，仕様化でき，プロジェクトを管理する能力があれば，特にプログラマとしての経験を必要としないという立場を主張する人々もいる．しかし，現代のソフトウェア開発では，システム化したい対象の知識だけでソフトウェア設計をすることは難しい．なぜならば，実現段階のハードウェアに関する知識や効率的なアルゴリズムに関する知識を抜きにして，高性能のソフトウェアを実現することは困難だからである．

---

† A Curriculum for Developing Software Engineers' Discipline through Program Development Experiences by NAOHIKO TAKEDA (Computer Center, Toyohashi University of Technology) and HAJIME OHIWA (Department of Information and Computer Sciences, Toyohashi University of Technology).

†† 豊橋技術科学大学情報処理センター
††† 豊橋技術科学大学工学部情報工学系
* 現在 慶應義塾大学環境情報学部
Faculty of Environmental Information, Keio University

われわれは「よいプログラマは，よい SE になる」となるという立場をとる．つまり，もしも他人にも容易に理解できるように可読性が高く，かつ，妥当なモジュール分割もなされているようなプログラムを作成できたならば，彼らは実務経験とともに，よい SE になるであろうと，われわれは考えている．

本論文においてプログラマは「プログラミング技能のみならず，モジュール構造およびインタフェース定義，データ定義，制御フローなどの設計業務と開発文書の作成をするもの」と定義する．この定義は，花岡[3]によるソフトウェア技術者のキャリアパスの定義のうち，初級・中級プログラマの定義に相当する*．したがって，単に制御フローをプログラムに置き換える作業をするのだけのコーダとは，区別して考える．

花岡は，さらにアプリケーション SE を定義しており，このうち初級アプリケーション SE は，「利用者の要望を聞いて，簡単なシステムの構築，文書作成ができ，対象業務に関する知識があるもの」と定義されている．この定義において，システムの構築をモジュール単位での設計ととらえれば，先のわれわれのプログラマの定義とほぼ一致する．異なる点は，対象業務に関する経験と知識の有無ととらえることができる．

プログラマにはモジュール単位での設計能力が，初級アプリケーション SE には，小規模なシステム設計能力が求められるという意味において，「よいプログラマは，よい SE になる」とわれわれは考えているわけである．以下，本論文の SE は，この初級アプリケーション SE を意味する．

ところが大多数のプログラミング教育では，プログラミング言語の文法や機能を短期間で習得することを中心としており，よいプログラムを書くことは，OJT により習得させる方針をとっている．このため，プログラミングの演習課題は，文法事項や機能の動作確認程度で終ってしまっているものが多く，プログラム書法や実際にシステムを作成するための訓練が十分でない．この結果，文法や機能を理解させる程度の例題を中心としたプログラミング教育は，一般には SE 育成に直接役に立つものではないと考えられている．

OJT を中心とした SE 教育の問題点は，プログラミング教育とシステム設計に関する教育を分離していることであるとわれわれは考えている．

前述したように，プログラマにもモジュール設計などの SE 的な能力が要求されるのであるから，プログラミング教育の段階から，このことを考慮し，よいプログラムを書く技能を習得するための教育をおこなうべきである．

われわれの提案するカリキュラムでは，小さな演習課題の段階から可読性が高く，ユーザ・インタフェースを考慮し，モジュール化された，いわゆるよいプログラムが書かれるような指導を徹底しておこなう．

次に，モジュール化にとって重要な手段の 1 つである関数を導入する．その後，ミニ・プロジェクトを実施する．このミニ・プロジェクトでは，学習者が自ら仕様を決定し，モジュール設計をおこない，プログラミングをする．

さらに，プロジェクトの開発過程において，現実の開発現場でおきるような，ソフトウェアの改造や変更などの，いわゆるソフトウェアの進化を疑似的に体験させる．この体験を通じて，モジュール化，設計の重要性や仕様変更の困難さを理解させる．これをわれわれは“体験によるプログラミング教育”と呼ぶことにする．

ミニ・プロジェクトのような実際に即した教育は，従来，OJT によりなされていた．これをカリキュラム化することにより，先に述べたような OJT の指導者ごとの実力による指導内容のばらつきを抑制することができる．

われわれは，これらの点を考慮したカリキュラムを作成した．このカリキュラムでは，C 言語の基本的な部分を習得することも目的としており，初級と中級に分けられている．初級では，一通りの制御構造と関数を導入し，特にプログラム書法やマン・マシン・インタフェースを意識させる．中級では，ミニ・プロジェクトを実施する．

本論文では，まず SE に要求される能力を分析し，この能力を身につけるために必要な要素をカリキュラムにどのように盛り込むかを検討し，決定した指導方針について述べる．次に，カリキュラムの構成について述べたあと，カリキュラムの各部で習得すべき SE 能力について述べる．最後に，このカリキュラムを用いたプログラマ養成講座を，全くの初心者約 50 人に実施した結果について述べる．

* 花岡は，プロジェクト管理までも業務の範囲内に含むものとして，上級プログラマを定義している．しかし，われわれは業務管理は SE の業務であるととらえているため，われわれのプログラマの定義中に上級プログラマを含めていない．

2. プログラミングによる SE 能力の養成

2.1 SE に要求される能力

SE に要求される基本的な能力は，

1) 問題を発見し解決する能力
2) システムを構築する能力
3) コミュニケーション能力
4) プロジェクトを運用・管理する能力

の4つに大別することができる．

1)は，現状のシステムを分析および問題発見をし，新しい情報システムの企画・提案をおこなっていく能力である．2)は，いわゆるシステム設計技法で，トップダウンなシステムの詳細化やデータ構造とプロセスの関係の設計，ユーザ・インタフェースおよび例外処理の設計をする能力である．3)は，仕様書の作成のための文書力，プレゼンテーション能力，インタビュー能力である．4)は，予算や工程の見積，プロジェクト計画・運用をおこなう能力である．

なお初級アプリケーション SE には，2)システム構築能力，3)コミュニケーション能力と，システムを実現していく上での問題解決という意味における問題解決能力が要求される．

2.2 プログラミングによる SE 教育

現在，プログラミング教育は文法および機能，操作方法を教えるための，いわば言語教育としてとらえられている場合が多い．一方，SE 教育はシステム設計論やマネジメント論を教えることが中心で，プログラミング教育は SE 教育にならないという考え方が一般的である．実際，前節で述べた SE 能力のうち，マネジメント能力や複雑なシステムに対する問題解決能力は，プロジェクト内での業務経験なしに身につけることは難しい．

しかし，われわれはシステム構築技術やコミュニケーション能力のような，いわゆる初級アプリケーション SE に必要とされるような能力は，プログラミングの初歩の段階から教育していくことが可能であると考えている．本節では，小さなプログラムをシステム的にとらえること，学習者にソフトウェア進化を体験させるミニ・プロジェクトの実施，およびプログラミング演習によるコミュニケーション技法の習得について述べる．

2.2.1 システムとしてのプログラム

前節でみたシステム構築技術をさらに詳細化すると

- ボトムアップなシステム構築技法
- トップダウンなシステムの詳細化
- モジュール構成とデータの関係の設計
- データ構造と設計
- ユーザ・インタフェースの設計
- 例外処理とエラーハンドリングの設計

などの要素に分類することができる．われわれの提案するカリキュラムでは，これらの要素を小さな演習課題の段階から，学習者に意識させることによりプログラムに反映させるように指導する．

ここで，重要な点はどのような小さな演習課題でも，ユーザを意識した小さなシステムとしてとらえさせることである．

例えば「2数を入力し，足し算の結果を出力する」プログラムについて考える．これはC言語の教科書や解説書で最初のプログラムとして，しばしば取上げられる例題である．従来の教科書では，単に入力して計算結果を出力するだけの図 1-a）のようなものが多い．このプログラムは，C言語の単なる動作確認をするだけであり，入力プロンプトも出力されないため，第三者は，実行画面を見ただけでは，どんな動作をするプログラムなのか予測することもできない．

本カリキュラムでは，このような簡単なプログラムであっても，ユーザの利用を想定し，ユーザが使いやすく，予測されない誤入力に対する処理もできているような加算演算システムを構築するという態度でプログラミングをさせる．

本カリキュラムでの加算プログラムの例題を図 1-b）に示す．これは，初級・第1回目の講義で扱う例題である．この例題では

- プロンプトを出力する
- 本処理を繰返し，ユーザの入力値によって停止する
- 出力を見やすく表示する

ことなどを，学習者が考慮するように指導する．学習者は，例題を学習した後，演習課題をおこない，例題で考慮した事項を踏まえた上で，ユーザ・インタフェースを設計しプログラミングをおこなう．さらに，非数値以外の入力があった場合の処理や，0+0を扱うための終了条件の設計，関数導入後は入力部の部品化などの議論をカリキュラムの進行とともに，徐々に導入していく．

このように例題でシステムとしてのプログラムを示し，それをもとにして演習をおこない，実際にシステムとしてのプログラムを学習者自身が作成すること

```
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

a) a program in a popular text book

```
/*
**      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

b) a program used in the proposed curriculum

図 1　加算プログラムの例
Fig. 1　Programs for addition.

は，システム設計の演習課題として，意味のあることだと考えられる．なぜならば，仕様書や設計図式を作成するだけのシステム設計演習では，実システムを簡単に実現できないため，机上演習となりがちである．しかし，本カリキュラムでは，学習者が自ら設計したシステムを実際にプログラムすることにより，実際の動作を確認し問題点を発見したり，システムを改造したりすることが可能だからである．

2.2.2　ミニ・プロジェクトによるシステム設計の体験

本カリキュラムでは，トップダウンな設計技法として，処理の階層的な表現ができ，データと処理の関係を表示することのできる HCP チャート[4]を用いて，段階的詳細化をおこなわせる．

従来のC言語の解説書では，文法事項や機能の技術的な解説に終始するものが多く，設計方法について詳しいものが少ない．一方，Pascal においては，D. Price「Pascal・思いやりプログラミング」[5] のようにプログラム書法に詳しいものや，W. Findly, D. A. Watt「Pascal プログラミング」[6] のように段階的詳細化の方法について事例を挙げて説明したものもある．

しかし，単にトップダウンな方法の習得のみでは，SE 能力としては十分でない．なぜならば，適切なモジュールの設計や再利用可能な部品の設計は，モジュールの相互関係で決められることが多く，単にトップダウンな方法では一義に決められないからである[7]．また，Findly, Watt の教科書では学習者自身が仕様を作成し，プログラムを書くという演習がない．

本カリキュラムではモジュール設計，データ構造，部品*の再利用などの技術の習得は，ミニ・プロジェクトにより漸進的にシステムを作っていくことによって習得させることにした．

ミニ・プロジェクトは学習者自らが仕様を決め，データ構造，画面設計，コマンド体系などのさまざまな仕様を決定した上で，実際のインプリメントをおこなわせる．学習者は，最初から大きなシステムを作ることは無理なので，単純な機能の実現からはじめて，順次，機能を拡張していくような演習課題の構成になっている．学習者は，課題の提示により，仕様を決定し実現する．次の課題では，前の段階で作成したプログラム自身を問題として用い，機能拡張や設計変更をおこなわせる（図 2）．このため，課題で取扱う対象業務は，学習者が熟知した分野であることが望ましい．

ミニ・プロジェクトでは，開発現場で実際におきるソフトウェア進化を，学習者に疑似的に体験させることを目的としている．ソフトウェアの開発現場では，ユーザ側からのソフトウェア進化要求，すなわち，改

* 本論文中で部品という場合，プログラム片やモジュール，文書など再利用可能なすべてのものを指す．

良，改造，仕様変更，設計変更などが日常的に生じている．進化要求に対する対処は，基本設計がよいものならば迅速におこなうことができる．ミニ・プロジェクトでは，課題を漸進的なプログラムの改造という形で構成することにより，意図的に進化要求をおこし，学習者に要求に対する対処を経験をさせる．

このミニ・プロジェクトで扱うプログラムは，システムといっても，高々，数百行のプログラムであるため，変更が比較的容易で，レビューの上，モジュールやデータ構造の変更をさせてみることにより，モジュールやモジュール関係の変更による利害得失を把握させ，適切なモジュール化について理解させることができる．

開発現場でのソフトウェア進化を体験させる方法は，従来，ほとんどの場合，OJT によりなされていた．しかし，OJT では，指導者の能力や指導方針に左右されるため，必ずしも効果的になされているとはかぎらない．ミニ・プロジェクトのようにカリキュラム化することによって，ソフトウェアの進化要求とその対処方法を，効果的に体験することができる．また，改造要求が出された時の，学習者の振舞い，例えば組織的に洩れのない変更が全体に渡って可能かどうか，効率のよいデータ構造を採用できるか，部品の再利用をうまくできるかなどを観察することで，経験のある SE ならば，学習者の SE 適性の有無を判断することができる．

2.2.3 プログラミングによるコミュニケーション能力の養成

本カリキュラムでは，コミュニケーション能力の養成のために，演習課題で動作するプログラムを書くだけでなく，レビューをおこなうことと，設計文書を書かせることに重点をおく．

レビューにより，学習者の理解度が確認でき，自分のプログラムを分りやすく説明できる能力，他人の説明を理解する能力，および適切な質問をする能力が習得される．

設計文書は，簡単な仕様書や HCP チャートを書かせる．また，ソースプログラムも他人がソフトウェアを理解する上での，重要な情報となりうるので，特に，プログラムの可読性に十分注意をはらうように指導する．

この2つを徹底することによって，言葉によるコミュニケーション能力と文字によるコミュニケーション能力を養成する．

2.3 指導方針

以上のことから，次のような指導方針を定めた．

Ⅰ．プログラム書法を徹底する

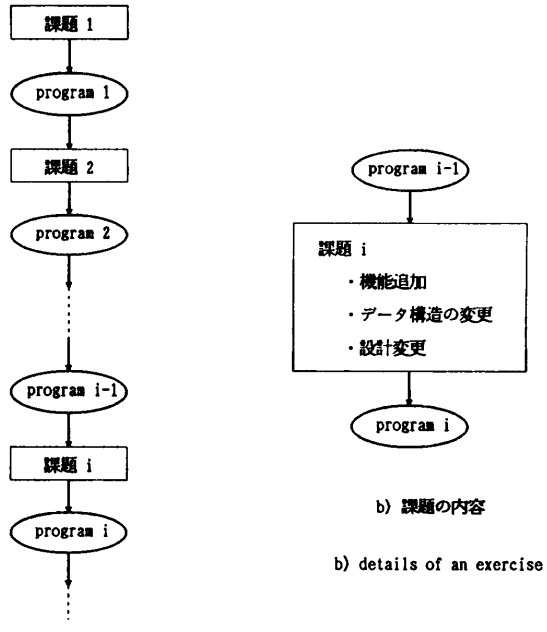
他人がプログラムを読む必要性が生ずることについて理解させ，他人に分りやすくプログラムとはどうあるべきかという観点から指導する．

Ⅱ．ユーザインタフェースの設計をさせる

本カリキュラムでは，インタラクティブなプログラムを例題としている．これは最初からユーザの存在を意識させ，ユーザが使いやすいプログラムを書かせることを目的としているためである．このため，簡単なプログラムでも，一度だけ本処理を実行するのではなく，終了条件が満たされるまで，何度も繰返し入力するようなプログラムを書かせ，ユーザにプログラムの挙動が分るようなメッセージの出力や，見やすい出力形式の設計をおこなわせる．使いやす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し，互いに実行してみることを試みさせる．

Ⅲ．例外処理の設計をさせる

ループからの中途脱出の方法や，不適当な値の入力時の処理を設計させる．これによって，処理が複雑に



a) 課題とプログラムの関係

a) relationship between the exercises and the example programs

b) 課題の内容

b) details of an exercise

図 2 ミニ・プロジェクトによる漸進的学習
Fig. 2 An incremental learning by small projects.

なるが，反面，使いやすいソフトウェアの設計のためには，例外処理の設計が重要であることを理解させる．

Ⅳ．学習者自ら仕様を設計し，実現する漸進的ミニ・プロジェクトを実施する

学習者が把握しやすい対象領域の問題を用い，実際に仕様の設計と実現をおこなわせ，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について理解させる．

Ⅴ．簡単な設計文書を書かせる

プログラム中の関数宣言や，プログラムの先頭に，コメントとしてその機能を説明する簡単な文を書かせる．また，ミニ・プロジェクトの課題について，簡単な仕様書，プログラムの動作解説書を書かせる．

Ⅵ．レビューをおこなう

作成したプログラムについて，公開レビューをおこないプログラムの仕様，内容，動作等について討論をする．

Ⅶ．概要設計をおこなってから，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」の関係を煮詰めさせ，概要設計を十分におこなってから，コーディングさせる．

Ⅷ．ANSI Cに基づいて，C言語を習得させる

同時に構造化プログラミングの重要性と制御構造についての理解を深める．

Ⅸ．基本的なデータ構造とその操作アルゴリズムを理解する

キュー，スタック，リスト構造，二分探索，ソート，ハッシュなど一般によく利用される基本データ構造とアルゴリズムを，ミニ・プロジェクトの進行にともなって解説し，実際にプログラムとして実現させる．

3. カリキュラムの内容

3.1 概　　要

本カリキュラムは，初級，中級と，全く計算機に触れたことのない初心者のため入門編の3つの部分からなっている．

各部分の内容と，時間配分は表1のようになっており，全体として 75 時間のカリキュラムである．入門は，3日間連続して開催する．一方，初級，中級は，隔日（月，水，金）で開講することにより，演習の積み残しを，講義のない日に消化できるようになっている．

表1 カリキュラムの構成
Table 1 Organization of the proposed curriculum.

区分	内　　容	講座回数	時間数
入門	タッチタイプ練習 MS-DOS とエディタの使い方 C言語の簡単な導入	3回	15時間
初級	制御構造と配列・関数 プログラム書法 ユーザ・インタフェースと例外処理	6回 (隔日開催)	30時間
中級	ポインタ・2次元配列 構造体・ファイル ミニ・プロジェクト	6回 (隔日開催)	30時間

講義1回の単位は5時間で，午前中，2時間を講義，午後3時間をプログラミング演習とした．これに午前中の講義の前，1時間を質問タイムとして，前日の講義や演習課題に関する質問時間にあてている．

なお，本カリキュラムではC言語に MicroSoft-C ver. 5.1，計算機は東芝 Dynabook (J 3100-SS 01)，OS は MS-DOS ver. 3.10 を使用した．

3.2 入　　門

入門は，全くの初心者にCを教えるための導入部である．

入門では，MS-DOS やエディタの基本的な使用方法を習得させることが目的となっている．また，タイプ速度が遅いと，効率的に演習課題のプログラムを作成することができないため，タッチタイプ練習[8]を実施した．

C言語については，出力だけ，入力-出力，入力-計算-出力の基本コンポーネントだけでできた3つのプログラムについてのみ講義した．ここでは例題を講義し，実際に動作させることをおこない，エディタとコンパイラの基本的な使用方法を憶えるのみである．

入門は，他の言語によるプログラミングの経験者には不必要な部分である．

3.3 初　　級

初級では，Cの基本的な機能の習得をおこなう．

まず，繰返し構造，次に条件分岐を導入し，制御構造について講義する．ここでは，構造化プログラミングの考え方を徹底する一方で，break 文を用いた中途脱出による柔軟なプログラムについても講義する．その後，配列，関数，ポインタなどを導入する．初級で

カバーする範囲はカーニハン，リッチーの「プログラミング言語C」[9]の第5章・関数までである．

初級の講義では，新しい事項を教えるために，必ず例題を提示し解説をおこなう．演習課題は，講義したプログラムの一部を改造・変更することと，アルゴリズムを工夫することにより容易に解答可能なものを用意した．

なお，初級では2.3節で述べた指導方針のうち，「Ⅳ．ミニ・プロジェクトの実施」はしない．この段階では関数の文法事項と引数の渡し方など使用方法の導入のみで，モジュール化の考え方やデータ構造の知識などを得ていないためである．

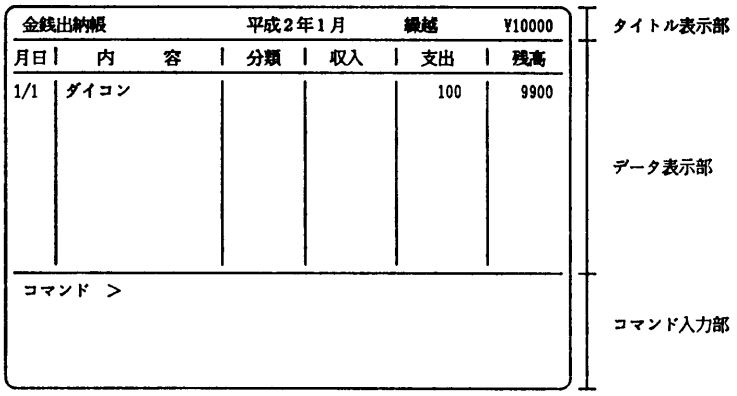


図3 金銭出納帳プログラムの画面設計例
Fig. 3 An example of display design for a cash-account program.

3.4 中　　級

中級では，初級に引続いてC言語の学習をすることとともに，ミニ・プロジェクトを実施し，ソフトウェア開発を体験することをおこなう．

C言語に関しては，新たに構造体，ポインタの使用方法，ファイルのオープン／クローズなどを学習する．

ミニ・プロジェクトでは，学習者に初級アプリケーションSE的な能力を身に付けさせることを目的とし，実際に小さなシステムを漸進的に開発させる．ただし，学習者は，実際の業務を経験していない*ため，課題で扱う対象領域を，学習者が熟知している領域（ここでは住所録と金銭出納帳）にする．

本節では，ミニ・プロジェクト課題と進め方，およびSE的能力の訓練項目について述べる．

3.4.1 ミニ・プロジェクト課題

中級編ではミニ・プロジェクトを設定し，漸進的にプログラム改良していくようなミニ・プロジェクト課題を与える．ここで与えた課題は，**図3**のような「金銭出納帳プログラムを実現せよ」というものである．

この金銭出納帳プログラムは図3のような画面表示で，タイトル表示部とデータ表示部，コマンドおよびデータ入力部からなる．

1) タイトル表示部：プログラム・タイトル，バージョン，扱っている年月と繰越残高の表示をおこなう．

2) データ表示部：いわゆる金銭出納帳のフォーマットで，金銭の出入を表示する．入出金の入力により，残高の計算は自動的におこなう．データが表示しきれない場合は，ページ・スクロール機能によりページングをして表示をおこなう．

3) コマンド入力部：コマンドを選択し，データの追加，削除，検索，ファイルへのロード／セーブなどの諸機能を実行できる．

ミニ・プロジェクト課題では，学習者には図3のような画面の設計仕様は提示せず，学習者自らが設計して，実現していくものとする．

3.4.2 例題提示型漸進的ミニ・プロジェクト

2.2.2項で，ミニ・プロジェクトにより，ソフトウェア進化を学習者に体験させる方法について述べた．ソフトウェアの現場において，ソフトウェア進化への対応もさることながら，他人のプログラムを使用し，ソースを読んで理解したり，部品を再利用したり，自分のプロジェクトに合うように変更したりすることもまた重要な技術である．

本カリキュラムで実施するミニ・プロジェクトは，単にシステムをプリミティブな要素から積み上げていくのではない．まず，よく似たソフトウェア構造を持つ例題を学習者に提示する．学習者は例題プログラムを実際に使用し，そのプログラムを理解する．これによって，自分の課題を効率よく，消化することができる．

また，例題を提示することの1つの理由として，初級の演習課題の経験だけで，画面設計をすることが困難なことがあげられる．その理由は，

- 最初の仕様をどう作ってよいか判断できない
- 画面制御などの問題が入ってくるとプログラミングができない

* 学習者が対象領域に関する知識がないという意味では，本カリキュラムは，先に述べた中級プログラマを養成しているともとらえられる．

からである．例えばある学習者は，前者について表計算型の金銭出納帳プログラムを想像するし，他のものはコマンド入力型の金銭出納帳プログラムを想像するといったように，最終的なプログラムの様相が違ってくる可能性もある．どのような形式を選ぶかによってプログラムの手間や複雑さは大きく異なってくるため，学習の進度をコントロールすることが難しくなる．また，後者のように画面設計を反映させるための画面制御の方法は計算機の内部動作に立入るため，初心者が自らインプリメントすることは難しい．

そこで，われわれは非常によく似た構造を持った例題プログラムとして住所録を選び，これを学習者に提示して講義をし，そのプログラムを実際に学習者に使用させ，さらに，その例題を利用することにより，比較的容易にプログラミングできるようにカリキュラムを構成した．

中級カリキュラムの構成を，図4に示す．

3.4.3 SE 的能力の訓練

a）部品の再利用を訓練する

例題プログラム「住所録」と課題プログラム「金銭出納帳」のモジュール構造と，その関係を図5に示す．この図から分るように，大きなプログラム構造は同じである．学習者は「住所録」のプログラム構造を理解し，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，また，どの部品は流用可能かということの判断をし，どのような設計をすれば最小の手間で金銭出納帳が実現可能かという選択をしなければならない．ここで，部品の再利用の訓練をおこなう．

b）プログラムを使う／読む訓練をする

a）をおこなうためには，まず，そのプログラムを使用させ動作を理解させる．その上で，例題プログラムを読ませ，どのように実現がなされているかを把握させる．

c）独立したモジュールを作る訓練

新しい機能を追加した際に，でき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．

d）大規模な変更をおこなわせる

c）とは逆に，全域にわたるような変更を経験させる．例えば，金銭出納帳のデータ構造を配列から構造体に変更させることにより，プログラム全体に変更点が波及し，かなりの工数がかかる．このような体験をさせることにより，初期の設計の重要性を学ばせることができる．

e）適切なモジュール化をおこなう訓練

「住所録」と「金銭出納帳」は，よく似た構造で実現できるものの，先に進んでいくにつれて，モジュール構造を変更したほうがうまくいく場合が出てくる．例えば，「金銭出納帳」で，金額を編集するコマンドを追加することを考える．「金銭出納帳」の場合，あるレコードの金額を変更すると，その後，すべての残高の再計算が生ずる．したがって，残高の再計算関数などのような，「金銭出納帳」特有のモジュールが必要になる．さらに，場合によっては，処理の都合のよいように，例題とは，異なったモジュール関係にしたほうがよい場合も出てくる．

4. カリキュラムの評価

4.1 講座の実施

本カリキュラムを在宅スタッフとしてのプログラマ養成講座で実施した．在宅スタッフとは，ある程度まとまった単位のモジュールやツールの開発をおこなうプログラムのことである．在宅スタッフは，社内スタッフから仕様書を受取り，打合せをした上で，自宅で

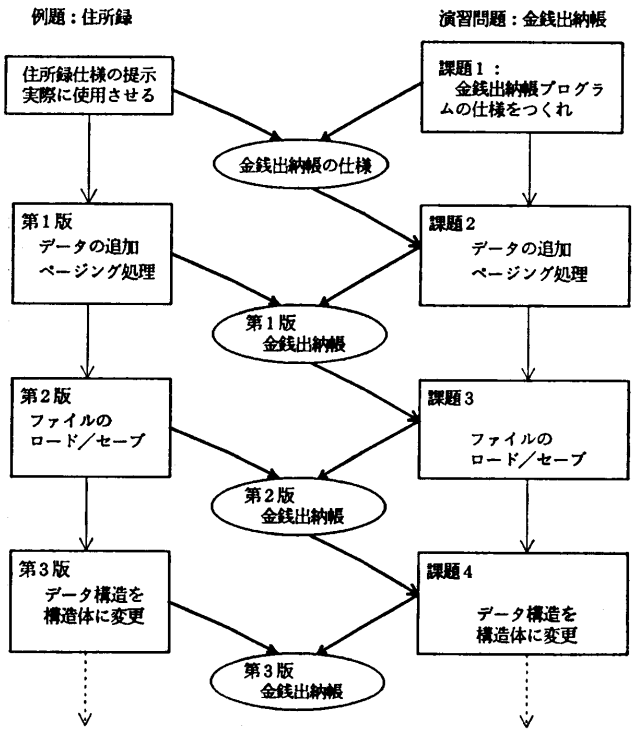


図4 例題と演習問題の関係
Fig. 4 Relationship between the examples and the exercises.

仕様書，HCP チャートなどの制御フロー，テスト報告書およびプログラムを作成し提出をおこなう．このため在宅スタッフには，初級アプリケーション SE 的な能力が要求される．

この講座の受講者は，20 代後半から 40 代にかけての女性約 50 名で，ソフトウェアに関しては，いわゆる素人ばかりであった．この講座の受講資格は，ムンツァートのプログラマ適性検査[10]を実施し，テストA (計算機の操作性と論理思考に関するテスト) で85点，テストB（分析的思考およびプログラミング能力に関するテスト）で 60 点を獲得することであった．この適性検査の結果，百数十名のうちから約 60 名のものが合格し，うち約50名が受講した．

本講座では，中級まで受講しミニ・プロジェクトの最終課題を完成して，レビューの上，提出すれば合格となり，在宅スタッフとしての資格を得ることができる．その理由は，ミニ・プロジェクトを通じてソフトウェア開発を一通り体験し，小さなシステムを自力でまとめたことになり，中級での SE 的要素の訓練項目を習得したと判断できるからである．

この講座を実施したところ，表2のような結果となった．全体の合格率は約 1/4 であった．

なお受講者は，この時点で実際の対象業務に関する経験を積んでいないので，初級アプリケーション SE というよりも，中級プログラマといったほうが適切である．この後，対象業務に関する小さく有用なツールを作成するなどの OJT を通じて経験を積めば，容易に初級アプリケーション SE として稼働することが可能であると考えられる．

4.2 カリキュラムの効果

本カリキュラムを実施したところ，次のような効果があることが分った．

1) プログラム書法の定着

例題で示されているプログラム書法が自然と学習者に定着していることが分った．特に名前の付け方やインデントの方法などは，最初はとまどうものの，初級のうちにほとんどの学習者に身に付くことが分った．

2) ミニ・プロジェクトにより効果的に学習できる

今回のミニ・プロジェクト課題は，受講者の実生活での体験に合致するように，金銭出納帳（講義では家計簿として解説）にしたため，興味が持続しやすく，学習者自らが仕様を作成する課題として適当であった．

表 2 受講者の推移
Table 2 Change of the number of attendants to the proposed curriculum.

	入門		初級		中級		最終合格者
	1日目	3日目	1日目	6日目	1日目	6日目	
一期	21	21	20 (2)	19	17 (1)	17	6
二期	16	16	15 (2)	14	11	11	4
三期	13	12	15 (4)	14	15 (1)	15	7

( ) 内は，中途参加者数

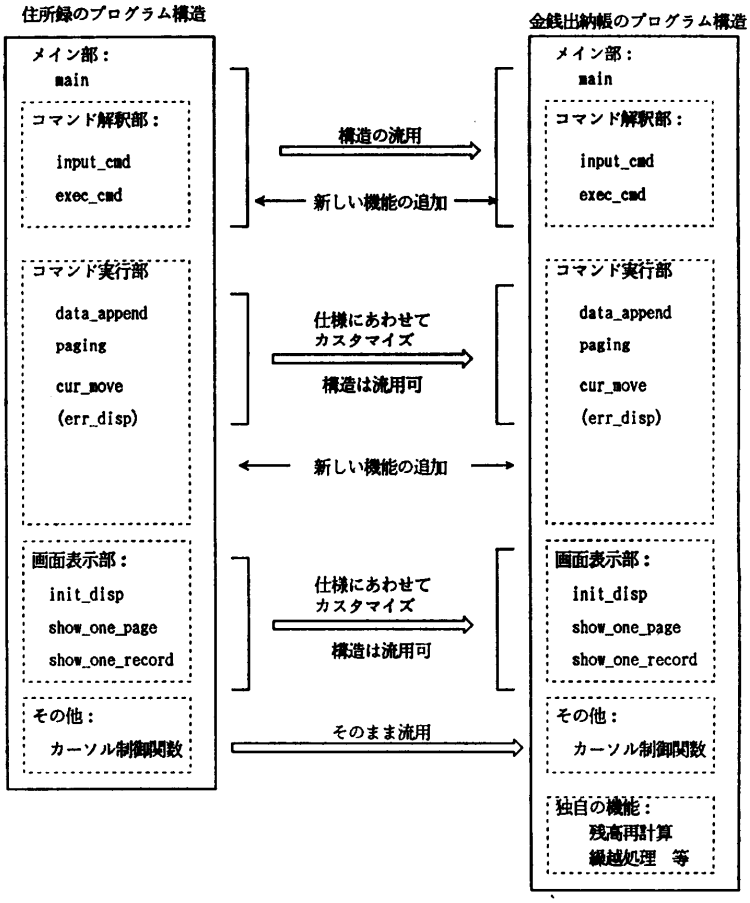


図 5 例題と演習問題のプログラム構造
Fig. 5 Program structures of the examples and the exercises.

3) ミニ・プロジェクトでの学習者の挙動がSE的素質を判断する材料になる

本カリキュラムを実施した結果，最終合格率は約1/4であった．合格に達しない理由には，a）演習時間が十分でなく，自宅に計算機を所有していなかったり，開講日以外に出社することができず，自習時間が十分にとれなかったために最終課題を仕上げることができなかった，b）SE的要素が要求される部分に関して理解が十分でなかったこと，の2つがあげられる．

a）に該当する人は，環境が整い演習時間が十分にあれば，最終課題に到達可能なレベルにあることが観察されている．このような学習者も含めれば，受講者全体の約半数は合格可能であると考えられる．

b）に該当する人は，多くの場合，次のような点でつまづいていることが観察された．

イ）例外処理や処理の繰返しループを定型パターンとしてとらえており，当該の処理が目的としている本質を理解していない．

ロ）部品の再利用がうまくいかない

ハ）処理を論理的に関係するものという観点からまとめることがうまくない

ニ）例題の理解が浅く，例題を演習課題にうまく応用することができない

これらの不合格者は，応用力が足りずミニ・プロジェクトの段階で非常に演習に手間が掛かってしまい，結果的に脱落してしまった．

また，ハ）に該当する人は，初級において適切なインデントや空白行の挿入の仕方がなかなか理解できないという傾向も観察された．これらの傾向は，処理を概念的にまとめる，モジュールの論理関係の把握，処理の意味の本質的な理解などが十分でないことを示している．

従来の，例題を中心としたプログラミング教育では，OJTや大規模プロジェクトに投入するまで，SE適性の有無を判断することができなかった．しかし，ミニ・プロジェクトを実施すると，上記b）のような学習者の振舞いから，SE適性に不安があることを，プログラミング教育期間中に，ある程度の判断を下すことができる．

4.3 問題点と今後の課題

本カリキュラムを実施した際に，次のような問題点が生じた．

1) 演習時間が不足であった

演習課題の消化のためには，時間不足であった．また，演習時間の不足のために，指導方針でかかげた，設計書の作成，レビューの励行などを十分におこなうことができなかった．

講習会後におこなった受講者に対するアンケートから，講義時間75時間に対して，受講者は自宅で平均20時間程度，演習課題に取り組んでいることが分った．このことから，演習課題を余裕をもって消化するためには，全体で100時間程度のカリキュラムに組みなおす必要がある．

さらに文書作成やレビューを徹底するためには，さらに30〜50時間が必要であると考えられる．

2) C言語のテクニカルな使用方法に対する解説が不足であった

受講者アンケートで「C言語についてさらに勉強したい内容はなんですか？」という問に対して，55%がポインタ，22%が関数（具体的にはポインタがらみの引数の使用方法）についてであった．

C言語のポインタは，C言語習得の上で最も重要な部分であるが，受講者が全員，計算機の素人であったため，実際のメモリ上のイメージとしてポインタ操作がとらえられないため理解することが難しかった．

ポインタおよび関数の解説には，動作確認のための例題や演習課題も追加し，十分な理解をさせるように，カリキュラムを改変する必要がある．

3) SE経験豊かな教師の必要性

3回開いた講習会は，それぞれ別々の教師が担当した．最初2回はSE経験者であり，最後の1回は，システム設計の経験のないプログラマであった．ミニ・プロジェクトにおいて，プログラマの教師は，学習者が提示する仕様を初期の段階で十分にコントロールをしなかったために，プロジェクトが進んでから，カリキュラムの目的に合うように学習者に大幅な設計変更を迫る場面が生じた．

本カリキュラムのようなSE育成を考慮したものでは，指導にも経験豊かなSEがあたる必要がある．また，教師に対して，カリキュラムの各々のセクションがどのような意図をもって作られているかをオリエンテーションするための，指導のための手引書を用意する必要がある．

5. むすび

本論文では，プログラミング教育の段階からSE的能力の向上を意識して教育するためのプログラミン

グ・カリキュラムについて述べた．

本カリキュラムの作成にあたって，まず，SE 的能力の検討をおこない，この能力をプログラミング教育に反映するための設計をおこなった．この結果，カリキュラムを初級と中級の2段階に大きく分け，初級では，従来の文法や機能を確認するためだけのプログラミング教育よりも，プログラム書法およびユーザ・インタフェースの習得に重点をおいたものにした．中級では，実際のソフトウェア進化要求とその対処を，疑似的に体験させ，モジュール化や部品化の重要性を理解させるために，例題提示型ミニ・プロジェクトを中心とする構成にした．

このカリキュラムを初心者50名に対して75時間の講習会を実施したところ，全課程を理解し終了することができた学習者には，初級アプリケーション SE 的な能力が備わっていることが確認できた．

この結果，従来は OJT あるいは大規模プロジェクトに関与するまで，判断できなかった SE 適性が，ミニ・プロジェクト実施時のモジュール化や部品の再利用方法の理解のようすから，比較的容易に判断できることが分った．

しかし，75時間程度の講習では，コミュニケーション能力や言語に関するテクニカルな面まで，徹底した教育をするためには不十分で，さらに演習に1.5倍から2倍程度の時間をかける必要があることも分った．

一方，本カリキュラムのような体験を中心としたプログラミング教育においては，教師の側に，十分な SE 経験が求められることが，実施上重要な点であることが分った．

本カリキュラムを教えられるような教師の育成方法の開発と，本カリキュラムによって育成されたプログラマの実務上での，より客観的，定量的な効果の測定をおこなうことが今後の課題である．また，上級段階として，アルゴリズムとデータ構造に関する技術を習得するカリキュラムを開発する必要がある．

謝辞 本研究にあたって，多大な協力をいただいたナカシャクリエイティブ株式会社・テクノメッセ豊橋の佐藤紀世久部長はじめ社員の方々，本講座を受講していただいたスタッフ候補生の方々，本論文を何度も読み返していただき適切な助言をいただいた豊橋技術科学大学知識情報工学系・河合和久講師，ならびに貴重な助言を多数いただいた査読者の方々に感謝いたします．

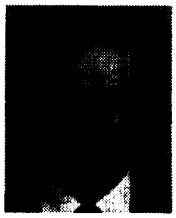
参考文献

1) 戸塚秀夫，中村圭介，梅澤 隆：日本のソフトウェア産業 経営と技術者，pp. 127-129，東京大学出版会 (1990)．
2) 三浦大亮，橋本茂司：システム分析，pp. 186-203，共立出版 (1987)．
3) 花岡 菖：システム・エンジニアの養成と管理，pp. 71-125，日刊工業新聞社 (1987)．
4) 花田收悦：プログラム設計図法，企画センター (1985)．
5) Price, D. (角田 (訳))：Pascal 思いやりプログラミング，近代科学社 (1984)．
6) Findly, W. and Watt, D. A. (牛島 (訳))：Pascal プログラミング，pp. 261-304，日本コンピュータ協会 (1981)．
7) Meyer, B. (二木 (訳))：オブジェクト指向入門，pp. 59-68，アスキー出版 (1990)．
8) 大岩 元，高嶋孝明：TSS によるタッチタイプトレーニングシステム，電子通信学会教育技術研究会技術報告，ET 79-12, pp. 37-42 (1979)．
9) Kernighan, B. and Ritche, D. M. (石田 (訳))：プログラミング言語C・第2版，共立出版 (1989)．
10) Munzert, A. W. (渡辺 (訳))：プログラマ適性テスト，東京図書 (1984)．

(平成3年9月12日受付)
(平成4年5月14日採録)

竹田 尚彦（正会員）

1958年生．1982年名城大学工学部電気工学科卒業．同年，(株)金陵入社．メカトロニクス関連ソフトウェアの開発に従事．1984年より社員資格のまま豊橋技術科学大学大学院へ進学．1990年同大学博士後期課程単位取得退学．工学修士．現在，同大学情報処理センター助手．HCI，ソフトウェア工学に関する研究に従事．ソフトウェア開発における人間的要因について興味をもつ．



大岩 元（正会員）

1942年生．1965年東京大学理学部物理学科卒業．1971年同大学院博士課程修了．理学博士．同年東京大学理学部助手．1978年豊橋技術科学大学情報工学系講師．1980年同助教授．1985年同教授．1992年慶應義塾大学環境情報学部教授．1974年～1976年英国ケンブリッジ大学キャベンディッシュ研究所客員研究員（ブリティッシュ・カウンシル・スカラ），1979年～1980年米国コーネル大学応用物理学科客員準教授．キーボード入力，情報教育，ソフトウェア工学などの研究に従事．