設計自動化　５４－９
(1990.10. 8)

基盤市場向けCASEワークベンチ：その開発と実践

CASE FOR SMALL BUSINESS MARKET

大西 司
Tsukasa ONISHI
ビジネスシステム研究所
The Institute of Business Systems

あらまし

小規模事業体の事務処理（販売・仕入・在庫及び関連業務）用のアプリケーションを、各企業の特性に合わせて製作する手法（ＳＳＳ： Structured System for Small business computer = トリプルエスと呼ぶ）を開発した。ソフトウェアの部品化と資源の再利用により、開発工程の自動化が図られている。これは、営業・要求定義・設計・製作・テスト・納品・管理の全工程を統合したCASE Work-bench として、まとめられている。我々の実施事例では、中規模のアプリケーション（約100Kstep）で要求定義→テスト間の工程での工数は、平均３３時間で完了するに至っている。

[英 文 題 目] ＣＡＳＥ for small business market
[英文発表者名] Tsukasa Onishi
[英文勤務先] The Institute of Business Systems
[〃 住所] 1-6-19 Imabashi CHUO-KU OSAKA

Abstract

S S S (read triple S or Structured System for Small business computer) is a computer based method for free customization of small and medium-sized business accounting systems (general ledgers, reports, inventory, etc.)

The program engineering steps are automatized using the method based on reusage of program units. A CASE Work-bench has been developed, where it includes sales tools, end-user's requirement analysis, design, engineering, testing, set-up and after support tools.

An average medium-sized business oriented accounting system (100Ksteps) requires about 33 hours of engineer's time for the development from the end-user's requirement analysis to its testing.

1．開発の背景

1－1．市場ニーズ

年商1億円～10億円程度の小規模事業体が、全国に約100万社存在するといわれている。これら企業体へのコンピュータ導入率は、未だ、極めて低いのが現状である。その大きな要因の一つは、実用的で安価なアプリケーションソフトウェア（以下アプリと呼ぶ）が提供されていないことである。周知の如く、低価格・高性能のパソコンが登場するに及んで、これらの企業体でも、ハードウェアを入手することは不可能では無くなった。しかし、依然として、実用的なアプリを入手することは、困難である。

パッケージされたアプリも幾つか存在してはいる。しかし、これらのアプリは、一部の規格化された業務を除いて、なかなか各企業の特性に合いにくい。小なりとはいえ、各企業体は、受注・発注・販売・仕入・在庫等の一連の事務処理を、やはり必要としている。そして、これらの業務は、細部に至っては、各企業体でそれぞれ微妙に処理内容が異なっている。標準的なアプリを流用して済ますというには、無理があるというのが実情である。その上、これらのパッケージソフトは、操作指導料等の付帯費用を加算すると、それほど安価とは言い難い。

伝統的手法によって顧客の個別条件に合致したアプリを作るには、ハード価格の数倍のソフトウェア費用が必要である。そして、必ずしもその品質は保証し難いものになってしまう。

従って、顧客の個別条件に合致した、しかも安価で使いやすいアプリが強く要望されている。

ディーラー・ソフトハウス等のアプリ供給側にとっても、事情は深刻である。従来の手法では、高度に訓練されたＳＥやプログラマーが、多数必要であり、この方面の要員不足は、今更言うまでもない。従って、初心者に簡単なトレーニングを施せば、高品質のアプリを作り上げることが可能な、ソフトウェア開発手法が切望されている。

1－2．既存手法の紹介と評価

カスタマイズを前提とした顧客アプリの開発手法として、パソコン上で実現可能なアプローチとしては以下のものがある。（ここでは，いわゆるソフトウェア危機の元凶とされる伝統的開発手法はあえて評価の対象から外した。）

1）パッケージシステムのカスタマイズ

カスタマイズ要求の範囲をあらかじめ想定した上で、機能をパラメータ化しておき、顧客の要求に応じてこのパラメータを変更することでカスタマイズを実現しようという考え方。

パラメータの範囲内での変更が容易であり、品質が安定して，短期間で完成し、低価格であるという長所がある。短所としては、最大公約数的なニーズによって作られているため、顧客にとっては不必要な機能が多く、逆にカスタマイズ可能な範囲が限られているので、細かな調整が必要な場合などは、通常使いものにならない。また，範囲外のカスタマイズを行うためには、ソースコードを直接修正する必要があるが、この手法が明確でないと結局のところ、作業者の能力・経験に左右され、メンテナンスも難しいという従来同様の問題が露呈する。

2）簡易言語の利用

最近では、ＣＡＳＥ的な統合環境も部分的に付加されるなど簡易言語の能力も向上し、十分に候補の一つと考えられるようになってきた。

長所は、当然のことながら開発が容易ということである。しかし完成物をアプリとしてパッケージ等と比較すると、その機能が貧弱であるということや、高性能を要求される大量のデータ処理には向かないことが大きな欠点として浮かびあがり、コマーシャルベースのアプリ開発に採用するには問題が多い。

3）ジェネレーション手法

定義された仕様から必要なプログラムの高級言語ソースコードを自動生成する手法。ＣＡＳＥツールを利用し、要求定義から保守に至る全ての工程の開発作業を統合的に行う。

顧客要求を実現する本格的なアプリが構築可能であり、プロトタイピングに適している他、メンテナンスが容易になるという長所がある。ただし、エンドユーザの細かい注文を完全にジェネレートすることは困難であるため、結局はやはり中間生成物のソースコードを直接修正する必要があることや、過去の類似システムの資産を再利用する機能がいまだ十分でない等の欠点があるが、なによりも決定的な問題は、パソコン環境上では重すぎて、実現が難しいということである。

4）ソフトウェア資源の再利用手法

システム仕様を構造的に体系化するとともに標準的に利用可能なソフトウェア資産を部品化して管理し、これを利用してアプリを作成する手法。また、カスタマイズ過程で新たに開発された部品は、フィードバックして再利用する。

オーダメイドに近いアプリを構築可能であり、類似システムを有効に利用すれば、開発期間が極めて短く、品質も安定している他、標準部品を利用したプロトタイピング作成に適して、メンテナンスも容易等の長所がある。

ただし、似かよったシステム内容が繰り返し発生するような対象にしか向いていないのと、用意されていない機能を再利用可能な部品として新規作成するのが難しいという欠点がある。

2．ＳＳＳの開発方法論

2－1．ＳＳＳの開発目標

ＳＳＳはその対象とする基盤市場の特性から、「ソフトウェア資源の再利用手法」を基本的な開発方法論に採用した。再利用手法は、同一システム内容の多い基盤市場の事務処理用アプリ開発手法として最適であり、他の欠点（部品化等の準備作業の負荷）を補って余りあると判断したからである。

そして、統合開発環境の考えを取り入れたＣＡＳＥワークベンチとしてアプリケ作成工程を総合的に支援するものとした。

ＳＳＳの目標は、以下のようにまとめることができる。

1）顧客ニーズに合致するアプリを短期間に、かつ高度な品質で作成すること
2）過去のソフトウェア資産を部品という形で再利用可能とすること
3）顧客（業種）別のノウハウを仕様という形で蓄積可能とすること
（これは営業上の大きな武器となる）
4）プロトタイピングにより、仕様確認を上流工程で行えること
5）アフターサポート（メンテナンス）の負荷を最小にすること
6）過去の資産の継承が容易で、継続的レベルアップを保証できる（提供者による付加価値のないパッケージにはマネができない）こと
7）営業からの一貫したツールによる支援体系をとる（工程別のツールの集合ではなく、開発方法論に裏付けられた統合開発環境とする）こと
8）ベテランＳＥレベルの作成ノウハウをツールが支援する（初心者でも短期間に高度なアプリを作成可能とする）こと

2－2．ＳＳＳの構造概要

再利用手法によるアプリ開発を実現するために、ＳＳＳではシステム全体の仕様からプログラムの内部構造、ファイル構造、機能部品、画面、帳票、ウィンドウ等の画面に至るまで、全てのソフトウェア資産が再利用を前提として標準化・体系化されている。

1）パターン別部品

プログラムは処理内容パターン別（入力、出力、問合せ、更新等）に分類され、そのパターン別基本部分に、具体的な個々のニーズに応じた処理機能部品を結合することにより作られる。

2）イベントドリブン形式

プログラムは、パターン別に定義された標準イベントによって、個々の処理機能部品が起動されるイベントドリブン形式の構造をとっている。標準イベントとは、入力項目への入力操作、ポップアップメニュー選択、ウィンドウ項目選択、メッセージ選択指示等をいう。

3）フレーム

部品を結合してプログラムを作りあげるためのパターン別の基本構造のことを、ＳＳＳでは「フレーム」と定義する。これはイベントドリブン形式のプログラムを実現するための部品の上位構造で、下位の機能部品を系統的に結合できるような標準化されたインターフェイスを持つメインループ構造のことである。

4）スロット

フレーム構造の知識なしに、誰にでも簡単にカスタマイズを実現可能にするためには、カスタマイズの内容別に機能部品を分類・体系化しフレームの下位構造として結合する手法が必要となる。「スロット」とは、フレーム上に用意された概念上の部品取付口を意味し、フレームに組み込むべき部品群を結合するための機能別インターフェイスのことである。

つまり、あるパターンのフレームにはパターン固有の機能を要求するいくつかのスロットがあり、このスロットには所定の機能の部品しか組込めないような構造にしてあるのである。

このスロットの目的とするところは、部品の選択作業を容易にし、プログラム合成の自動化を可能とすることである。（図１）

５）ユニット

プログラムは「スロット」に固有の機能を有する部品を組込んで作り上げるもので、この部品のことを「ユニット」と定義する。

「ユニット」は、カスタマイズに際して変更を必要とされる対象を単位に、一つまたは複数のモジュールを整理・統合して作られたものであり、いわゆる機能モジュールではない。

実際のカスタマイズ作業を考えてみよう。例えば顧客から『売上伝票の合計欄に合計消費税の項目を追加してほしい』というような変更依頼があった場合、変更を必要とする部品（この例では、過去の伝票内容を表示するモジュール、入力内容をチェックするモジュール、伝票を登録するモジュール等）があちこちに散在するのでは、これらをすべて間違いなく変更することは容易なことではない。

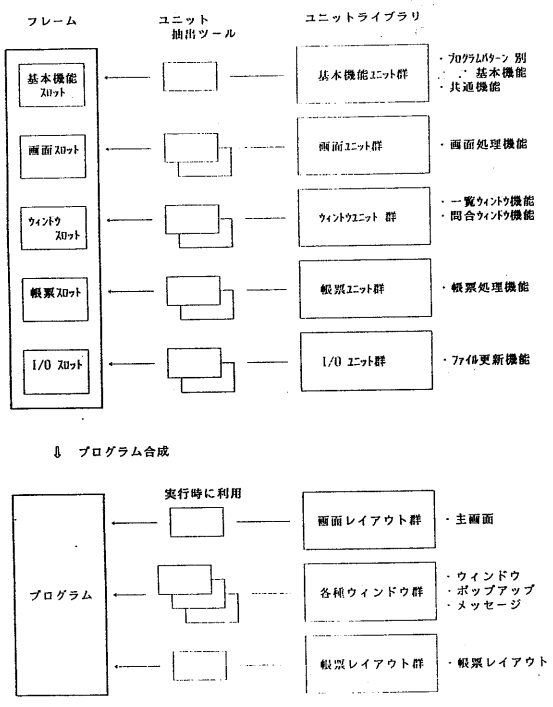
このようにカスタマイズの要求は、ある機能に対してではなく、対象に対して発生するため、モジュールの上位構造となる部品は対象別となっている必要があるのである。

６）その他の部品

プログラム実行時に呼び出される主画面や、帳票レイアウトの他、ウィンドウ、メッセージ、メニュー等の画面部品、ファイル定義体も「ユニット」とは別に部品として管理される。

これらは完全に独立した部品として標準化されており、すべてのプログラムから自由に呼び出すことができる。

図１　フレーム・スロット・ユニットの概念図



２－３．ＳＳＳの設計方法論

ＳＳＳでは、外部仕様の項目をカスタマイズの対象の単位とすることによって、要求仕様の確定とアプリの作成を容易に行うことができる。

この外部仕様項目に注目して仕様定義をする手法は、項目指向設計（item oriented design）と呼ばれ、ＳＳＳの開発方法論の根幹をなすものである。

１）項目指向とは？

伝統的なウォータフォール型モデルの開発手法では、アプリは上流から下流へと工程を経ることによって作成される。各工程では、上流で作られた成果物を受け取り、これを変換して下流へ渡す作業を行うのである。

従来の開発手法は、開発の出発点として確定したニーズ（要求仕様）を必要としてきた。ＤＦＤやプロトタイピング等によって、要求定義を支援する手法もあるが、エンドユーザの抱いている曖昧なニーズと分析ツールで利用可能な要求仕様の間には、依然として大きなギャップが存在する。如何に統合化されたツールであろうと、最上流のニーズ定義が困難であったり、使用する者によって異なる要求仕様が定義されるようでは大いに問題がある。

対象を基盤市場の事務処理用アプリに限定した場合においては、『カスタマイズ要求は、常に外部仕様の変更を伴う』という我々の経験から生まれたものが、項目指向設計という手法の考え方である。

この手法は、カスタマイズの対象となる単位としてエンドユーザでも認識が容易な外部仕様（画面レイアウト、帳票レイアウト）の具体的な項目一つ一つを対応させることにより、

a)最上流でのニーズ定義を容易にし、

b)上記工程間のギャップを極小化し、

c)仕様からアプリを自動作成する

という分析、設計、製造の各工程を総括した開発手法である。

分析時には、エンドユーザと分析者が、画面や帳票の実物を見ながら具体的な項目についての要求や問題点を明確にし、細部にいたるまでのカスタマイズ要求の仕様を確定する。こうしてアプリの動きを直接確認することによって、分析工程でのヒアリング作業において、常に大きな問題であったコミュニケーションギャップを限りなくゼロに近づけることができる。

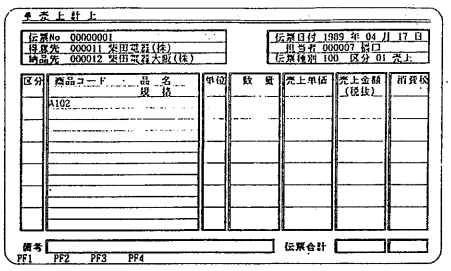
また項目別にユニットを定義することにより、定義された要求仕様を、そのまま必要な部品仕様へと利用することができ、以降の工程で発生する工程別仕様間（要求仕様←→機能仕様←→部品仕様）のギャップを解消することができるのである。

2）項目指向と部品

項目指向手法においては、要求仕様≒設計仕様≒部品仕様であるから、エンドユーザの要求からいきなり目的とする部品を定義することすら可能である。ここでは、項目指向設計により定義された仕様を、部品がどのような構造で実現しているかを概説する。

『項目の処理は項目部品自身が知っている』という表現が、項目指向における処理の考えを言い表している。

図2　売上伝票入力画面



ここでいう項目部品とは、以下の標準イベントにより起動される項目別の部品のことである。

a)自身への入力操作

b)ウィンドウ操作

c)他の部品からの処理要求

図2の売上伝票の入力プログラムを例に、項目部品の動きを説明しよう。

例えば、商品コードの項目部品は、入力操作というイベントに際して以下のような処理をする（エラー処理は省いて）ものとする。

a)商品コード入力イベント前処理

・入力された文字種をチェックする

・入力された文字列を指定形式に編集する

・以前に表示されていた文字列と異なるかをチェックする

b)商品コード項目処理

・商品マスタファイルを、入力された文字列で検索する

(*)・商品内容を表示する

－商品名を表示

－単位を表示

－売上単価を表示

c)商品コード入力イベント後処理

・項目を抜ける際の使用キーをチェックする（リターンキーか矢印キー、その他）

・キーにより次の入力項目へ移動する

顧客からのカスタマイズ要求は、『(*) 商品内容を表示する』の部分がほとんどである。例えば、

・商品名の他に規格（色、サイズ）を表示

・単位マスタから、単位名称を検索し表示

・単位の他に入数を表示

・標準売上単価ではなく顧客ランク別の単価を表示

・標準売上単価ではなく顧客別の最新単価を表示

・売上単価の他に外税単価を表示

・売上単価の他に原単価を表示

というような要求が単独で、または複合して発生する。

このような要求の一つ一つを個別に修正するのでは、部品化の意味がないといえるし、また全ての要求の組み合わせを事前に準備するようなことは、事実上不可能である。

ここで『項目の処理は項目部品自身が知っている』という言葉を思い出してほしい、項目部品は関連する項目の処理を、その項目部品に依頼することで処理を実現しているのである。

つまり、『A項目を表示する』とするのではなく『A項目部品に処理を依頼する』というように定義するのである。

例えば、『標準売上単価ではなく顧客別の最新単価を表示する』というカスタマイズ要求の場合には、商品コード部品の (*)以下の部分の内容を

(*)・商品内容を表示する
　　－商品名部品に処理を依頼する
　　－単位部品に処理を依頼する
　　－売上単価部品に処理を依頼する

というように定義しておけば、売上単価の項目部品を、『標準単価を処理する』から『得意先別最新単価を処理する』というものに交換するだけで（商品コード部品は一切変更しないで）カスタマイズが実現できるのである。

ところが、『売上単価の他に外税単価を表示』という項目を追加する場合には、

(*)・商品内容を表示する　という部分に
　　－外税単価部品に依頼する

という機能を追加する必要があり、部品を変更しなければならなくなってしまう。

そこで、依頼する個々の項目を一括して『必要な部品に依頼する』というように抽象化すれば、依頼項目に変更があっても部品の変更は不要となる。そして、個々の依頼項目の内容は、別途ツールによって指定することにすればよいのである。

同様に、項目部品の前処理と後処理の部分も処理を抽象化して、外部仕様の定義時にツールで属性、処理を定義するようにすれば、本質的にカスタマイズが必要な項目処理は、ファイル検索の部分だけとなり極めて定義が容易となる。従って、外部仕様を決定するだけで、項目入力イベントのための部品仕様はすべて確定可能となる。

このように、項目固有の処理と抽象化を追求し、処理定義作業の標準化、ツール化を徹底することによって、項目指向部品によるアプリ作成が実現されるのである。

3．アプリの作成手順

3－1．作成手順と統合環境

ＳＳＳによるアプリの作成は、従来の工程を経ることなく、システム全体を再利用によって定義→構築する手法に基づいた、合理的な作成手順とその実行を一貫して支援する統合開発環境によって実現される。（図3参照）

開発工程は、標準仕様（マザースペックと呼ぶ）との差異を確認しつつ、カスタマイズ仕様を定義するだけで完結し、必要な部品がそろっていればそのまま自動的にアプリを作成することが可能である。

3－2．要求分析

顧客ニーズをヒアリングする作業はアプリ作成工程の最上流にあり、カスタマイズ要求を決定する最も重要な部分である。

1）顧客業務要件のヒアリング

顧客の業務概要、必要サブシステム・プログラム、データ量の確認等の全般的要件をヒアリングし要件定義ツールへ登録する。

2）画面（項目）仕様の確認

画面検索ツールにより、顧客と類似業種、業務の画面サンプルを照会しつつ、カスタマイズ要求を項目単位で確認する。その結果に基いて、確認用画面レイアウトを画面エディタで作成し、再度、項目配置と内容及び動きをチェックする。

3）帳票（項目）仕様の確認

画面と同様に帳票サンプルを提示後、カスタマイズ要求に基いて、確認用帳票レイアウトを作成し、項目配置と内容をチェックする。

3－3．画面・帳票仕様設計

1）画面仕様設計

画面仕様定義ツール（スクリーンリファイナーと呼ぶ）に分析段階で作られた画面イメージを読込み、個々の項目別に項目属性と項目部品を定義する。定義作業は、マザースペック内の項目辞書（画面、ファイル、帳票の項目属性を管理する）と部品辞書（部品仕様を管理する）に支援され、対話形式ですすめることができる。項目部品は、項目別に候補となる部品が複数個準備されていて、これがウィンドウに一覧表示されるので、該当する機能をもつ部品を容易に選択することができる。画面から呼ばれて項目データを取得するウィンドウ部品等も、項目に

属するユニットとしてここで定義される。

2）帳票仕様設計

帳票仕様定義ツール（レポートリファイナーと呼ぶ）は、スクリーンリファイナーと同様に、帳票外部仕様と部品仕様を帳票種別、項目別、印字グループ 別に定義するツールである。

3－4．仕様チェック

外部仕様を登録後、それぞれの仕様に以下の問題がないかをチェックする。

a)項目と項目辞書間との属性等の差異

b)全外部仕様項目間での関連性の矛盾

3－5．システム設計

ＳＳＳは、従来ＳＥの主な作業とされていた内部仕様定義（ファイル設計、プログラム設計）をほとんど完全に自動化することを実現した。

1）ファイル設計

従来の手法では、外部仕様が固まると次に行われるのがファイル設計である。ＳＳＳでは、外部仕様でユニット定義が完了すれば、ユニットで扱うファイル項目はあらかじめ分かっているので、必要なファイルとファイル項目は自動的に決定される。

また外部仕様にない項目は、登録された業務要件やプログラム別必須情報からファイル情報が特定される。

これらの情報からファイル設計は自動的に行われる。従って、ファイルとファイル項目は常に必要十分の内容を持ち、不必要なファイルや余白項目は一切存在しない。

2）プログラム設計

画面仕様と帳票仕様は定義されたが、その他の機能は未定義のように感じられるであろう。（例、ファイル更新処理、頁初期化処理）

実は、これらの機能は最初にプログラム種別（入力、出力など）が決まった時点で抽象機能として定義されている。個々の画面帳票仕様はシステム全体で一括して管理され、必要なインターフェイス部品（抽象機能と具体的な項目・ファイルに関連する部品とを対応させる）は、自動生成されるのである。

3－6．新規部品開発

ところで、あらかじめ用意されていない例外的な項目や処理が発生したらどうするのか。

項目属性は、外部仕様から自動的に既存ファイルのカスタマイズ仕様に取り込まれるか、または自動的に（例えばコードという項目種別をもつ新規項目は、必ず新規マスタファイルを必要とするといういような規約に基づいて）設計される。とはいえ、部品は新規に作成しなければならない。新規部品は、ユニットリファイナーというツールによってモデルとなるユニットをカスタマイズして作成、定義される。

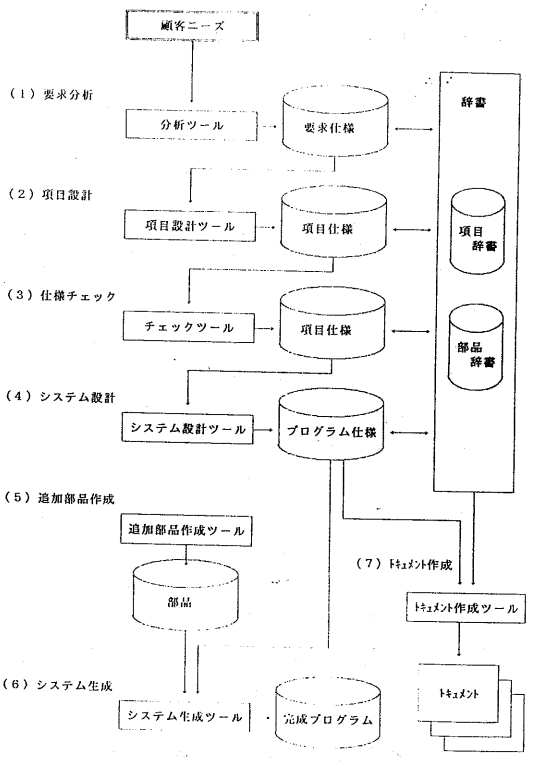
3－7．システム生成

以上の手順を経て定義された仕様に基づいて、部品のライブラリ、そして自動的に生成された部品を合成することで、カスタマイズされたプログラムとファイルを自動生成する。

3－8．ドキュメント

ＳＳＳにおいては、全ての開発支援情報はツールによって視覚的に取得可能であり、コミュニケーション（対人、対自分）のためのドキュメントは必要が無い。従って、ドキュメントはアプリ完成時にメンテナンス用としてツールより打ち出しておけば十分なのである。

図３　作成手順と統合環境



4．実施効果

4－1．納品実績

昨年4月より現在（90年8月末）までの一年余で、139システムを納品した。顧客の評価は上々である。やはり、自分に合った、使いやすい、洗練されたソフトを顧客は求めている。

4－2．所要工数分析

当初は、新システムにありがちなことであるが、バグの発生や、操作の不慣れのために、設計・作成工数は、かなり変動した。

図4に、直近20例での工数分布図を示す。工数算出の対象作業は、直接的なシステム作業、即ち、営業マンの打ち合わせや資料作り、設計作業、作成作業、テスト作業の合計時間である。営業作業及び納品指導作業は、本システムとは関係のない要素、即ち、顧客の所在地の遠近や、受け入れ体制の有無等で、大きく変動するので、意識的に除外した。

図4によれば、対象期間中もなお、諸種の改善や習熟の過程にあることがうかがえる。しかし、直近の10例程度では、ほぼ一定値を示しており、このあたりが極限値かと思われる。

表1に、直近10例での平均値として、工程別所要時間を示す。

作成とテストの工数は、時に何らかの誤りが発生し、修正する時間を含んでいる。現在のバージョンでは、これらの誤りの発生は防止されるようになっており、従って、この両工程の工数は、更に短縮され、多分1／2にはなるであろうと期待されている。それは別としても、既に実績として達成されている範囲で、工数は33時間という短時間になっている。筆者等は永年ディーラーとして、類似のシステムをオフコンで作ってきた。その場合の所要工数に比べれば、凡そ1／10に短縮されている。

ここでの詳論は避けることにするが、営業工程での効率追求や、仕様定義の正確さの訴求においても、この手法は格段の進歩を示していることを付言しておく。

他方、顧客への納品・操作指導は、通常3日間を必要としている。つまり、この工程を含む全システム工数は、一週間が目安となる。かつて、システムの設計・作成に数ヶ月を要した段階では、操作指導工程は、付随的存在でしかなかったが、今や全工数の半数を占めるに至っている。そして、この工程は、必ずしも合理化の対象とはせずに、充分に時間をかけるべき対象と、我々は考えている。

4－3．アフターサポート分析

全ユーザについて、納品後のアフターサポートを記録してみた。

例えば、今年6月の1ヶ月間において、全ユーザ112社（当時）に対して、アフターサポートは、表2の如く28件発生している。例月この程度である。

各ユーザ別に見ると、納品後1～2ヶ月間にアフターサポートのピークがある。大半が、操作の問い合わせであり、その後は稀にしか発生しない。従って、ユーザ数が累積されても、月毎のサポート件数は、殆ど上昇しない。

従来、筆者等がオフコンの納品客にサポートした経験から言えば（正確な比較は不可能であるが）、サポートの負荷は1／10以下に減少しているという実感を持っている。

4－4．要員の早期養成

こうして、全工程がCASEワークベンチとして統合されたことによって、工数が大幅に短縮され、品質が向上すると同時に、要員不足を解消する目処も立ちつつある。全工程が高度に体系化された結果、4月入社の新人に、一ヶ月余のトレーニングを実施し、即戦力として登用することが可能となりつつある。豊富な経験を積んだベテランSEの存在がなくても、新人だけで、オーダーメイドソフトを完全に作れると考えられる。当社の新人を対象に現在試験中であり、その結果については改めて報告したい。

5．まとめ

筆者等が開発したSSSについて、その内容と実施例を紹介した。世に、CASEの紹介記事は数多いが、寡聞にして、基盤マーケットを対象にした、実践的な成果物を知らない。筆者等自身、実践家としてこのマーケットに関わって来ただけに、SSSの実際的意義は極めて大きいと考えている。

今後は、アプリそのものを、より高度で巾広い対象に拡張していく予定である。

御批判、御指摘を期待している。