ソフトウェア工学 38－3
（１９８４．９．２６）

再利用支援ツールＳＰＩＳＥ－Ⅱの モジュール設計への適用評価

松村　一夫　　小尾　俊之　　山城　明宏　　大筆　豊
（（株）東芝　システム・ソフトウェア技術推進部）

１．はじめに

ＳＰＩＳＥ－Ⅱ（Software Production by Interactive Synthesis Engineering, version2：以下、本文ではＳＰＩＳＥと略記する）は、設計からコーディングまでを一貫支援し、(1) 設計ドキュメントとプログラムを対応させて編集管理する、(2) 設計・プログラムの標準パターンを部品として蓄積・再利用する、ことにより生産性・品質・保守性向上を目的とするツールである[1]。

ＳＰＩＳＥは大阪大学のπエディタと供に、標準パターンを用いる構造エディタに属する[2]。一方、カーネギーメロン大学のＡＬＯＥエディタに代表されるように、特定言語の文法知識を持った構造エディタがある[3]。後者が厳密な文法チェックを行うのに対し、前者は使い方の自由度が高く、使用者自らが定めた設計・コーディングの方法論や手法を支援できる。ＳＰＩＳＥも、特定の方法論・手法を限定するものではないため、使用を全く個人レベルにまかせたのでは、個人差により使用効果が半減する不安がある。従ってチームで開発する際は、方法論や手法の思想を反映した標準パターンを準備し、統一した適用法を定めてＳＰＩＳＥを用いている。

本報告では、モジュール設計の部品化・再利用を目標とした５０ＳＭ（50 Steps/Module)方法論と、それをベースにしたＳＰＩＳＥ適用法を紹介する。更に、適用効果について、従来手法との比較実験、及び実務への適用結果をもとに評価する。

２．ＳＰＩＳＥの概要

ＳＰＩＳＥは、人間と計算機のもつ特性を有機的に生かすことを設計思想とし、以下の特徴を持つ（詳細は文献[1])。

プログラム表示の詳細化と抽象化

ＳＰＩＳＥは、図１のようなＣＲＴ画面でテキストを編集する。

図１の例で、Ｂの部分はその上の行Ａの詳細内容を表わしている。ＡとＢのような親子階層のことをフレーム構造と呼び、各構造の各構成要素をフレームと呼ぶ。各フレームには詳細化の度合を表わすレベル番号を与える。フレーム内のある行を詳細化したものが、次のレベルのフレームとなる。フレームとして詳細化される行（図１のＡ、行末に“…＆”がつく）をフレームの「表題」、詳細化された行（図１のＢ）をフレームの「内容」と各々呼ぶ。さらにＣＲＴ画面を有効利用し、フレームの表示／非表示を行うことができる。すなわち、フレームの内容を非表示にし、その内容の抽象化された行（表題行）のみを表示する。また、このフレーム構造は更に下位に詳細部を追加作成しフレーム構造を構築することができる。

標準パターンの再利用

上記フレーム構造を含んだ定型パターンに名前（ＩＤ）をつけて、部品ライブラリに登録管理し、必要に応じてその部品を検索・使用できる。この定型のパターン部品をスケルトンと呼ぶ。ＳＰＩＳＥでは主として、ホワイトボックス部品（機能だけでなく、その内容を知り、一部変更あるいは穴埋め方式で利用する）の支援を目的としている。

レポート機能

フレーム、スケルトンあるいは各行に機能文字をつけることができる。機能文字はフレームあるいは行に特定の属性を持たせ、ドキュメントの編集・作成に利用することができる。すなわち、ＳＰＩＳＥのファイルから必要な設計情報のみを選択して、設計ドキュメントを出力したりソースプログラムのみを出力したりできる。

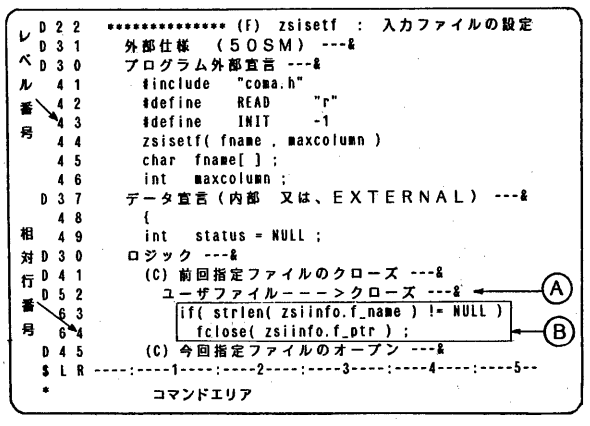


図１．ＳＰＩＳＥ－ⅡのＣＲＴ画面

3．適用方法

ここでは、モジュール設計のために設定した５０ＳＭ方法論の概要を説明し、それをＳＰＩＳＥで支援する適用法について述べる。

3.1　５０ＳＭ方法論

５０ＳＭ方法論の目的は、

(1)　モジュール設計・コーティング・テスト作業を作業者への能力依存度を減らし標準化する。

(2)　設計・プログラム情報の部品化・再利用で、生産性及び品質を向上させる。

(3)　設計文書とソースプログラムを、正しく対応づけて管理し保守性を向上させる。

である。この目的を実現するため、次のような考え方に基づき方法論を設定した。

プログラム構成要素は５０ステップ／モジュール

プログラムを構成する処理、データの単位は、コーディング時５０ステップ（ソースプログラム行数のうち、ヘッダーを除く）以内に制限する。これにより、一貫性、理解性を高め再利用しやすくするとともに、モジュール内部のコーティングのバラツキを小さくする。

言語依存しない設計概念と記述法

可能な限りプログラミング言語に依存しない設計概念と、その記述法を定める。

モジュール設計は、タスク（論理的な並行処理単位で、ビジコン系のジョブステッププログラム、プロコン系のファンクションプログラム、バッチ処理のアクティビティに相当する）内部を処理型５０ＳＭ、データ型５０ＳＭ、パッケージ型５０ＳＭの組合せとして表現する（図２）。ここで、５０ＳＭの種類は、

- 処理型５０ＳＭ：　サブルーチン、関数、手続マクロなどで実現される処理型のモジュール。
- データ型５０ＳＭ：　変数、定数やメモリ領域、ファイルやデータベース、外部入出力機器とのインターフェースデータなどのデータ型モジュール。
- パッケージ型５０ＳＭ：　Adaパッケージと同様。データ抽象化など、お互いに強く関連し合った５０ＳＭ群を、まとめて管理できるようにしたモジュール。外部から使える５０ＳＭ（図２のg,h,i）は区別して明記される。

である。また、各５０ＳＭは、全体の中での位置づけ（あるいは所属）により、“グローバル”（タスク間で共通に使用される）、“タスク内共通”、“パッケージ内共通”、“ローカル”（特定の１つの５０ＳＭに強く依存し、それからしか使われない）の４つの属性をもつ。

これら５０ＳＭの外部仕様、内部仕様の書き方については省略するが、概略は図３(a) ～(d) に示されている。

５０ＳＭ部品ライブラリ

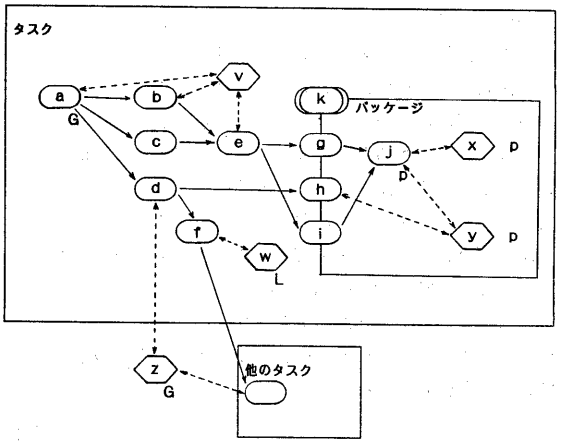
作成した５０ＳＭ自体をライブラリ化しておき、検索し利用できる。応用分野に依存した専用ライブラリと、応用分野に依存せず共通に使える共通ライブラリに蓄積していく。将来は、５０ＳＭ部品の組立てだけでプログラム作成できるようにする。これに関し、方法論としてはまだ不十分であるが研究中である[4]。

機械的コーディング

設計仕様を見れば、誰でもほぼ一意にコーディングが進められるよう、設計記述とコーディング規則を対応づけている。特に、制御構造と変数定義・宣言は、プログラミング言語毎にコーディングへの変換規則を定める。

ツール支援でペーパレス化

設計とコーディングの全ドキュメントを、計算機内に貯え、モジュール毎に設計／ソースプログラムを常に一緒に保守・再利用できる。



：処理型５０ＳＭ　　パッケージ型５０ＳＭ
：データ型５０ＳＭ
G　：グローバル
無印：タスク内共通
P　：パッケージ内共通
L　：ローカル

→　処理型５０ＳＭ参照又は他タスク起動
←→　データ型５０ＳＭへのアクセス

図２．５０ＳＭの種類と位置づけ（概念図）

処理手続き記述のスケルトン化

処理型５０ＳＭで処理手続きを設計記述するために、約２０個のスケルトンを用意した。それぞれの記述用途に合せて

・２分岐処理用スケルトン「@Ｃ２」
・３分岐処理用スケルトン「@Ｃ３」
・多重分岐処理の追加用スケルトン「@ＣＣ」
・繰返し処理（無限ループ）用スケルトン「@Ｌ」
・　〃　（while 型）用スケルトン「@ＬＷ」
・　〃　（untile型）用スケルトン「@ＬＵ」
・　〃　（for 型）用スケルトン「@ＬＦ」
・　〃　抜け出し用スケルトン「@Ｘ」
・エラーチェック用スケルトン「@Ｅ」
・例外処理用スケルトン「@ＥＰ」

や、これらの変形パターンなどである。

図４は、２分岐処理用スケルトンと、その使い方の例である。例えば、「ファイルの初期化を行う」という処理手順を記述するのに２つの条件分岐があれば、図４(a) のスケルトンを「@Ｃ２（ファイルの初期化）」のように呼び出して使う。その結果は図４(b) のようなフレーム構造に展開される。設計者は、この中の「条件１」や「処理１」の部分を、それぞれ「既存ファイルの場合」や「○○する」のように置きかえていく（図４(c)）。この時点で、既にコーディングの一部が入っているので、コーディング工程では、「％」の部分をコーディングしていくことになる。

図４でもわかるように、分岐処理の設計表現には（Ｃ）の記号でシンボライズし、理解しやすくしている。同様に、繰返し処理は（Ｌ）（図５）、繰返し処理の抜け出しは（Ｘ）、エラーチェックは（Ｅ）などでシンボライズしている。

設計ドキュメントの分離出力

設計ドキュメントおよびソースプログラムをユーザに提出すべき場合は、ＳＰＩＳＥのレポート機能を用いて設計情報（機能文字“Ｄ”のついた行）のみを分離出力する。また、ソースプログラムリストは、設計情報を全てコメントに変換してリスティングする。

```
D 1 5 ************** (F) @1 : @2 ****** 所属 @3 ******* ---&
D 2 4     外部仕様  （５０ＳＭ） ---&
D 3 3         概要 ---&
D 4 2             %
D 3 1         インターフェース ---&
D 4 0             使用形式：  %
D 4 9             引数１  ：  %
D 4 8             共通データ名（ファイル）：  %
D 4 7             共通データ名（コモン）  ：  %
D 4 6             パッケージ名            ：  %
D 4 5             外部関数名              ：  %
D 3 4         使用上の制限 ---&
D 4 3             %
D 2 2     プログラム外部宣言 ---&
  3 1          @1 ( % )
  3 0          %
D 2 1     データ宣言（内部　又は、ＥＸＴＥＲＮＡＬ） ---&
  3 2          (
  3 3          %
D 2 4     ロジック ---&
D 3 5          %
D 3 6          RETURN------------------------------ ---&
  4 7              ret :
  4 8                   return % ;
  3 9          )
```

図３．（ａ）．処理型５０ＳＭ用スケルトン

```
D 1 0 ************** <D> @1 : @2 ****** 所属 @3 ******* ---&
D 2 1     概要 ---&
D 3 2         %
D 2 3     使用上の制限 ---&
D 3 4         %
D 2 5     データ定義表（実現区分，名前，型，配列，説明） ---&
D 3 6         f.r.i.v.c   name% , type% , array% , descrip.% ---&
  4 7                     %
D 3 8         f.r.i.v.c   name% , type% , array% , descrip.% ---&
  4 9                     %
```

図３．（ｂ）．データ型５０ＳＭ用スケルトン

```
D 1 0 **** (( P )) @1   @2 ***** ---&
D 2 1   パッケージ概要説明 ---&
D 3 2     %
```

図３．（ｃ）．パッケージ型５０ＳＭ用スケルトン

```
D 1 7 **** (( P )) zsi-pack : 文字列入力パッケージ ***** ---&
D 2 6     パッケージ概要説明 ---&
D 2 5     ********** <D> zsi_data  : パッケージ共通データ **** 所属 P ***** ---&
D 2 4     ********** (F) zsisetf   : 入力ファイルの設定   **** 所属   ***** ---&
D 2 3     ********** (F) zsirchar  : １文字入力           **** 所属   ***** ---&
D 2 2     ********** (F) zsisrchar : 読み飛ばし１文字     **** 所属   ***** ---&
D 2 1     ********** (F) zsiskip   : 指定文字まで読み飛ばし ** 所属   ***** ---&
D 2 0     ********** (F) zsicomp   : 文字列の比較         **** 所属   ***** ---&
```

図３．（ｄ）．パッケージ内部の例

3.2 ＳＰＩＳＥ適用法

上記方法論のサブセットに対し、ＳＰＩＳＥで運用支援する適用法を定めた。サブセットの主な制限は、

・プログラミング言語はＣとする。

・処理型５０ＳＭは、関数で実現する。

などである。なお、５０ＳＭ方法論は、フォームシートを用い、計算機支援なしでも運用可能である。

モジュール設計の記述構成のスケルトン化

３種類の５０ＳＭの設計記述形式を、それぞれ図３(a)～(C) のようにスケルトン化した。

・処理型５０ＳＭ用スケルトン「@Ｆ」（図３(a) ）

このスケルトンでは、一番左端に機能文字“Ｄ”の記号のついた行は設計情報である。従って、設計者は“Ｄ”のついた「％」の行を、適切な設計記述で置きかえていく。

・データ型５０ＳＭ用スケルトン「@Ｄ」（図３(b) ）

「@Ｆ」と同様である。“Ｄ”記号のついていない「％」行は、後でデータ定義文（コーディング）で置きかえられる。また、グローバルやタスク内共通のデータ型５０ＳＭは、データ宣言文（Ｃ言語ではextern宣言）でコーディングしたものを、その５０ＳＭ名で部品（スケルトン）登録しておき、コーディング時に必要な場所に再利用する。

・パッケージ型５０ＳＭ用スケルトン「@Ｐ」（図3(C)）

「％」行を、上記の「@Ｄ」および「@Ｆ」のスケルトンで置きかえながら設計を進める。その結果、ＳＰＩＳＥでパッケージ内を見ると図３(d) の例ようになる。すなわち、パッケージ内に含まれる５０ＳＭの一覧とその表題が同一レベルで、見ることができ、各々の５０ＳＭの詳細はその下位フレームに展開されている。なお、原則として１つのパッケージはＳＰＩＳＥの１ファイル（コンパイル単位に相当）で管理する。

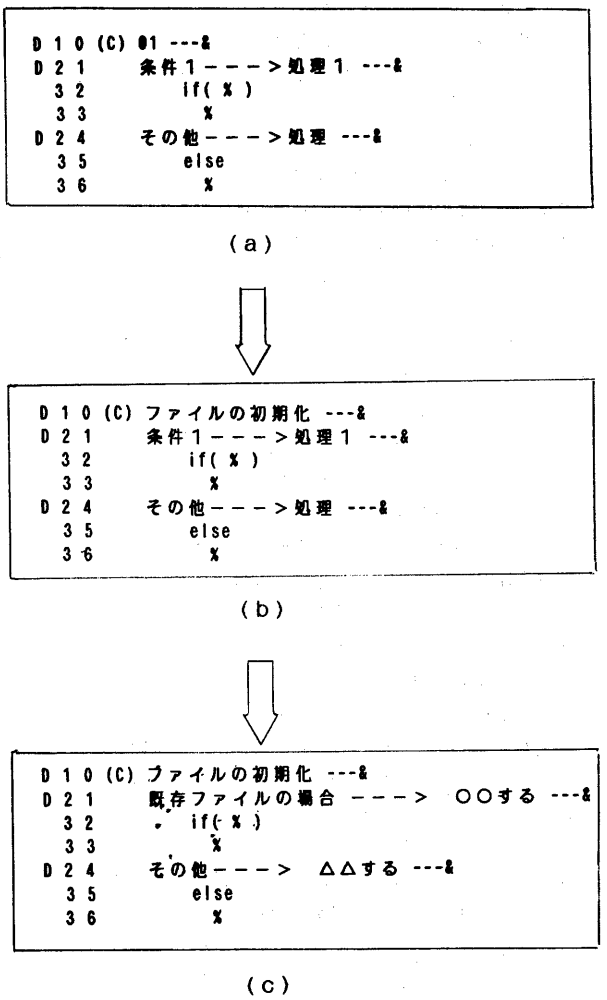


図４．２分岐処理用スケルトンと使用例

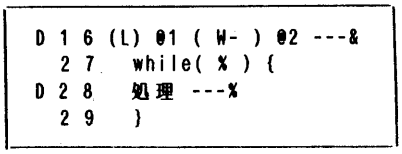


図５．繰返し処理用スケルトン

4．評価方法

ＳＰＩＳＥのモジュール設計・コーディングへの適用を通して評価実験を行なったので、その評価項目、実験方法について述べる。

4.1 評価項目

ＳＰＩＳＥを使用することによる効果は、つぎの項目の総合から評価するべきである。ただし、３章で述べた適用方法を前提とし、ＳＰＩＳＥ単独の評価ではなく、適用方法とペアでの評価である。

①生産性

ＳＰＩＳＥを用いることにより、他のツールと手法を用いた場合より、どの位、生産性が向上するのかを評価する。例えば、設計仕様書の記述、あるいはコーディング作業、デバッグ・テスト作業等にかかる工数により評価する。

②品質

ＳＰＩＳＥを用いることにより設計プログラムの品質がどの位向上するかを示すもので、例えばコンパイルエラーの数やテストに於ける、仕様との不具合点の数などから評価する。さらに、モジュールの制御構造が単純な程エラーが少ないという経験則に基づき、プログラム複雑度指標の McCabe メジャー[5] も測定し評価する。

③保守性

拡張、修正等の保守作業が生じた場合、それに対して対処し易いかを評価する。これは、設計・プログラムの理解のし易さ、前述のMcCabeメジャー、および標準化率で評価する。これらの項目で評価する理由は、McCabeメジャーは小さい程、すなわち制御構造が単純な程、追加・変更がしやすいことによる。また、各種の標準化は、保守員がかわっても同じスタイルで設計・コーディングを扱えることで保守効率が向上することによる。なお、標準化率は、モジュールごとのステップ数、コメント量などを調べる。

④操作性

ツールそのものの使い易さも重要な項目である。ツールの使用感、ツールによる入力作業の工数などから評価する。

4.2 従来手法との比較実験

実験は６人の被試験者にプログラムを作成してもらい、データを採取した。各被試験者は、次の３つの手法を用いて３つのモジュールについてそれぞれの手法で作成した。

①ＨＩＰＯで書かれた仕様書から、通常のエディタを用いてコーディング、テストをする（H-E と略す）。

②ＨＩＰＯで書かれた仕様書から、ＳＰＩＳＥを用いてコーディング、テストをする（H-S と略す）。

③ＳＰＩＳＥで書かれた仕様書から、ＳＰＩＳＥを用いてコーディング、テストをする（S-S と略す）。.

そのプログラミング作業に対し、下記データを採取した。

ｉ）コーディング時間（仕様書解読から入力が終了するまでの時間）

ii）コンパイル時間（Syntax エラーがなくなるまで）

iii）テスト時間（テスト仕様が全て満足されるまで）

iv）コンパイルエラーの数（エラーの延べ数）

ｖ）テスト仕様に対するエラーの数

なお、用意したＨＩＰＯ仕様書、ＳＰＩＳＥの仕様書の記述量（情報量）は同等量となる様に努めた。また、用意した３つのモジュール（３０～４０ステップ程度の基本ソフト的内容のもの）間の差は、各方法について同数の人が実験を行ない、データ処理の際、差が出ない様にしている。被試験者の言語・計算機等についての知識は同レベルと見ているが、ＳＰＩＳＥについての知識、経験については若干の差がある。

この他、作成されたプロジェクトのリストの分析や被試験者への面接アンケートを実施した。

4.3 実務への適用実験

実務のプログラム開発に、ＳＰＩＳＥと５０ＳＭ方法論を適用した。このプログラム開発の概要は、

・対象プログラム：ソフト開発ツール（会話型）
・言語　　：Ｃ
・ステップ数　：約9000
・開発人員　　：４名　　設計のみ　　１名
　　設計／コード　２名
　　コードのみ　　１名

であり、システムプログラムの分類に入る比較的難しいプログラムである。　現在、単体テストに入った時点であるので、コーディング終了（コンパイルエラーなし）までの結果をまとめた。

5．結果と評価

5.1 実験の結果と評価

実験結果のまとめを図６～図８に示し、その評価及び考察を項目ごとに述べる。

プログラム作成の作業時間について（図６）

・Ｓ－Ｓは従来のＨ－Ｅと比べて作業時間の差はない。

・Ｈ－Ｓが他に比べて悪いのはＨＩＰＯからＳＰＩＳＥのフレーム構造を作るのに定まったやり方がないため、プログラマがプログラミングスタイルに迷ったり、あるいは、その構造化の不統一によりプログラムの質を落しているものと考えられる。

従って、ＳＰＩＳＥ－Ⅱを用いる時はその使い方を定めてから使うべきである。

エラー件数について（図６）

・コンパイルエラー数はＳ－Ｓが最少である。

・コンパイルエラーの種類として、Ｈ－Ｅ、Ｈ－Ｓには変数等の宣言の忘れ、｛　｝の不一致があったが、Ｓ－Ｓにはこの種のものはすくない。これはスケルトンを用いたことによる効果と考えられる。

アンケートによる好感度について（図７）

・総合では、１位Ｈ－Ｅ、２位Ｓ－Ｓの順である。主にテキストエディタとしてのＳＰＩＳＥへの不満が差になっている。（なお、現状でＳＰＩＳＥに不慣れなため、２位と答えた人が２名含まれている。）

・読み易さではＳ－Ｓが１位である。しかし、１行の表題が長くなり、横スクロールで画面を見る必要がある時、理解の妨げとなるという意見もあった。

・コーディングのし易さでもＳ－Ｓが１位である。スケルトンで組込まれた“if(%) % ”などのコーディングパターンがガイドとなり便利であったという人、あまり詳細すぎるフレーム構造はかえって足かせになるという人などもあった。また、ある被試験者は今回のスケルトンを使い慣れており、Ｈ－Ｓにおいても迷わずスケルトンを使っていた。部品が完全に手法の一部として身につけている例だろう。

モジュールの複雑さについて（図８）

実験データ数は少ないが参考までに、プログラムの複雑さ（McCabeメジャー）の結果を示す。

・Ｓ－Ｓの方が若干ではあるが単純構造（メジャー値がい）のプログラムとなっている。

H-E	H-S	S-S
5.2	4.6	4.4

（１モジュール当りのMcCabeメジャーの平均値）

図８．モジュールの複雑度比較

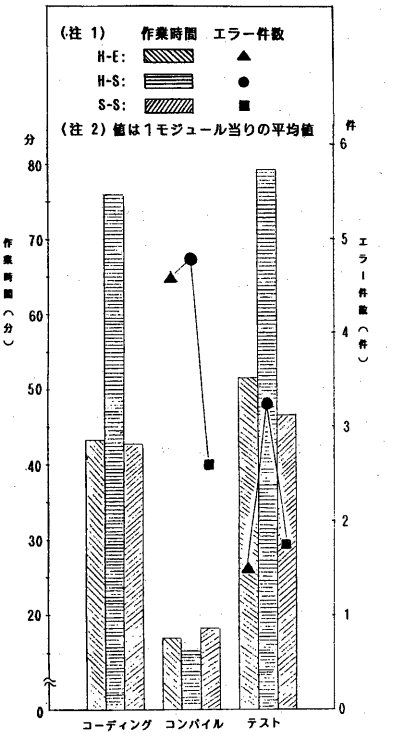


図６．プログラム作成の作業時間、エラー件数

手法	H-E	H-S	S-S
総合印象	1.6	2.5	1.9
読み易さ印象	2.1	2.9	1.5
コーディングし易さ印象	2.0	2.9	1.1

エディタ種類	ed	vi	sp
エディタの使い易さ	2.6	1.3	2.1

SP=SPISE, ed,vi はUNIXのエディタ

(1位=1点,2位=2点,3位=3点とした時の平均点)

図７．各手法の好感度

5.2 実務適用の結果と評価

実務への適用結果を図 9～図10に示し、その評価及び実務での感想を含めた考察を述べる。

設計・コーディング作業の生産性について（図９）

生産実績を図９に示す。生産性については、対象ソフトの種類、人月の作業日数、非生産的時間（休暇、訓練、会議など）の扱い方など種々の違いがあり、一般的に論ずることは非常に難しいが、ほぼ次のことがいえる。

コーディングの生産性は、この種のプログラムとしては従来の生産性と同等以上である。設計作業に関しては従来よりやや工数が多いが、そのうち２１％はＳＰＩＳＥへの設計情報（日本語）入力作業である。この作業量は、デバッグ以降のドキュメント修正や、保守用設計ドキュメント作成の工数削減でかなりカバーされる。また、ＳＰＩＳＥでは、設計情報が計算機に入力されている強みがあり、今後レポート機能をより充実させることで、さらに生産性を向上できる。

今回の事例では、５０ＳＭ部品ライブラリは全くない。この条件で、従来と同等以上の生産性が得られたことは目標を達成したといえる。

操作性について

設計情報の純粋な入力作業工数は、平均１．５Ｈ／モジュールであった。従来の作業（ＨＩＰＯ等によりモジュール外部仕様と内部設計書を手書き清書する）に比べて、 1.5～ 1.8倍の工数が必要であった。これは、日本語入力の不慣れと難しさによる。従って、設計情報の初期入力は、日本語ワープロの専門作業者が行えるような機能をＳＰＩＳＥに追加中である。

設計・コーディングをペーパーレスで行うツールという観点からすると、５０ＳＭの関連図や、自由に書いた説明図も設計情報として不可欠である。これらの図やイメージ情報を作成、参照するツール組込みが必要である。

標準化について（図10）

設計記述の形式については、スケルトン使用によりチーム内で完全に統一できた。以下は、設計・コーディングの内容的な標準化について述べる。

設計・コーディングの結果、処理型５０ＳＭは合計１５３個になった。これらのステップ数の分布を図10に示す。さらに、コメント率（コメント行数／５０ＳＭステップ数）は、平均30.4%（行数換算で約１５行）、標準編差 8.4である。

ＳＰＩＳＥの端末（現状ＲＴ－ＵＸ）で表示できる行数は40行（うちコマンドエリア８行）であり、コメント、すなわち処理手順の設計情報は確実に一覧できる大きさにおさまっている。ヘッダー、すなわち５０ＳＭの外部仕様情報を含めても１画面で見れることになる。更にソースプログラムを全て表示しても、ほぼ２画面以内で見れる大きさであり、理解するのが容易である。

	設　　計	コーディング
SPISE 適用の生産性	0.77 人月／kステップ SPISE の純粋入力作業の割合い　２１％	0.43 人月／kステップ* 107 ステップ／日

＊：ソースプログラムからヘッダーを除いた行数
１日人＝８Ｈ，　１人月＝２２人月

図９．設計・コーディング作業の生産性

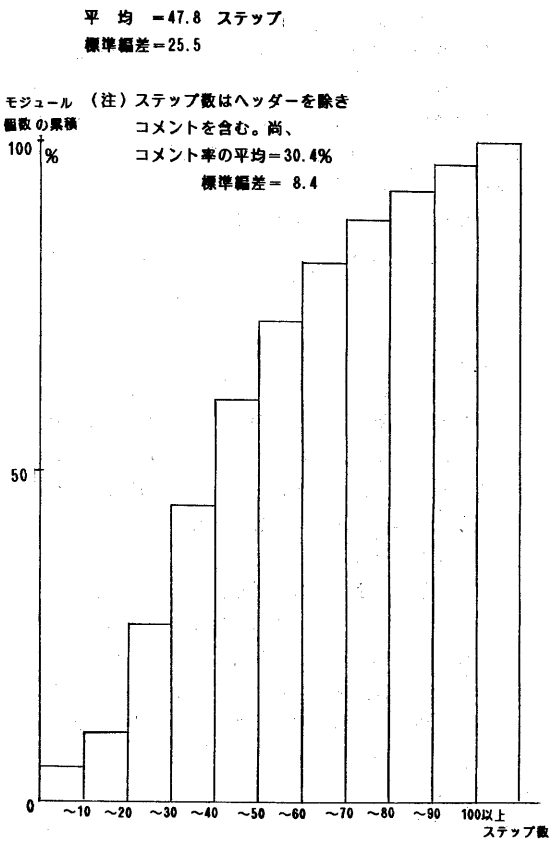


図１０．モジュールサイズの分布

コメント率は、通常のプログラミングより多めである。これはＳＰＩＳＥで表題行を全て自動的にコメントにしているからで、展開・縮退を基礎とした構造エディタではある程度避けられない。いずれにしろ、プログラマが機械的にコーディングできるためには、詳細な設計が必要なのは当然で、その意味でこのコメント率は良いと思われる。ただし、一旦コーディングをし、内部を理解しているプログラマがデバッグ作業を行っている時点では、詳細すぎる設計情報を自動的に間引いて表示するなどのツール支援が必要であろう。

５０ＳＭのステップ数を５０以内におさえる事に関し、それ程強く意識しなくとも、平均47.8ステップとなった。小さく再分割することで、良いモジュールが設計できたという印象はある（後述の「複雑さについて」参照）。今回の結果では、設計の個人差まで立入った評価はできないが、今後、５０ステップを越えたモジュールにつき、更に分割できないか、あるいは何故分割できないかなどの理由を分析して、ツールや方法論の改良に役立てる。

モジュールの複雑さについて

１５３個の処理型５０ＳＭの、McCabeメジャーは、平均 5.1、標準編差 3.3となった。これは、５０ステップ以内を目標にしたことと、処理記述用パターン部品の使用により、設計者の意図（制御構造）がそのままコーディングに反映された効果と思われる。McCabeメジャーが２０を境にして、エラー率が急増するという報告もあり、今回の結果は単純構造の良いモジュールの部類に入る。

６．まとめ

構造エディタＳＰＩＳＥがモジュール設計・コーディングの生産性、品質、保守性の向上に効果があることを、適用実験により示した。前提として、５０ＳＭ方法論に基づくＳＰＩＳＥ適用法を設定し、従来手法との比較実験や実務への適用実験により評価した。

その結果を総合すると、

(1) ＳＰＩＳＥは適切な適用法を定めて使うべきである。
(2) 従来手法と比べて生産性は同等以上である。
(3) モジュールの標準化、理解性、単純化などが促進されること、及び設計ドキュメントとソースプログラムが対応して計算機管理されることにより、再利用や保守に役立つ。

などである。

これらの結果は、ＳＰＩＳＥがホワイトボックス部品を扱う環境として有効であることを示唆するものである。今後は、部品化・再利用を更に推進すべく、方法論、ツールの改良・拡張を計画している。

参考文献

[1] 会田他，“標準パターンによるプログラミング支援ツール：ＳＰＩＳＥ－Ⅱ”，情報処理学会、ソフトウェア工学研究会資料、33-3, 1983.11.

[2] 大筆他，“標準パターン活用の構造エディタ”情報処理学会誌　解説　Vol.25, No.8,1984.

[3] Feiler, P.H. and Medina-Mora, R., "An Incremental Programming Environment", Proc. 5th Int. Conf. on Software Eng., pp.44 -53,1981.

[4] 小島他，“ジェネレータ用ソフト部品の分析法”，情報処理学会第２８回全国大会予稿集、pp.507-508, 1984.

[5] McCabe,T.T., "A Complexity Measure", IEEE Trans. on Software Engineering, Vol .SE-2, No.4, pp.308-320, Dec.1976.