ソフトウェア工学 58－17
（１９８８．２．５）

ソフトウェア開発用インタフェース：ｓｉｆ

藤岡　卓、中島　毅、上原　憲二、高野　彰
三菱電機（株）　情報電子研究所

本報告では、分散型ソフトウェア開発支援システム Solonにおける人とワークステーションとの接点として、ソフトウェア開発時のコマンド入力を効率化することを目的としたソフトウェア開発用インタフェース sifについて述べる。

ソフトウェア開発作業を調べてみると、作業の1)局所性、2)反復性、3)並列性という特徴があげられる。これらの特徴をもとに、「作業環境」という考えに基づいたソフトウェア開発作業モデルを提案する。そして、作業目的毎に作業対象および操作を整理し、これらをメニューやウィンドウを駆使して視覚化することによって作業の効率化を図るツール sifの構成、特徴的機能、実現方式について述べる。

Sif : Man-Machine Interface for Software Development

Taku FUJIOKA, Tsuyoshi NAKAJIMA, Kenji UEHARA, Akira TAKANO
Information Systems and Electronics Development Laboratory
Mitsubishi Electric Corporation
5-1-1 Ofuna, Kamakura, Kanagawa, 247 Japan

We have developed an interface for software development called sif as one of tools of a distributed software development support system called Solon, so that we can communicate with the system easily and efficiently.

Observing a software development process, we found that it has such characteristics as 1)locally fixed, 2)repeating, and 3)parallel operations. Here we propose a model of the operation based on "operation context". And then we show a configuration, special features, and an implementation of sif, which holds objects (e.g. directories and files) and operations for each context and visualizes them elegantly by using menus and windows.

1．はじめに

ソフトウェア開発システムの形態は、大型計算機を複数の端末で共用する TSS型から、ワークステーションと計算機を LANで結合した分散処理型へ移行しつつある。従来のソフトウェア開発作業において、人が計算機と接するのは、主にコーディング、テスト、デバッグの各段階である。人は、大型計算機等の TSS端末に向かって、スクリーンエディタなどを用いてプログラムコードを修正し、再テストを行う。このテスト、デバッグ作業の繰り返しが、実にソフトウェア開発工程の半分を占めると言われている。従って、この部分の効率を上げることができれば、ソフトウェアの生産性はかなり向上するはずである。

さらに、ソフトウェアCAD という言葉が示すように、近年、ソフトウェア開発工程の前段階、すなわち設計段階を機械化し、支援するツール開発がさかんである。これらによって、単にソフトウェア開発工程における機械化の割合が増加したばかりでなく、その形態も、後段階のツールに見られるバッチ型から、人と計算機のかかわりの多い会話型のツールへと移り変わってきており、マンマシンインタフェースの研究がますます重要になってきている。幸い、ハードウェアの面でのマンマシンインタフェースの向上はめざましいものがある。ビットマップディスプレィやマウスを備えた高機能ワークステーションの出現がそれである。それに対して、それらを利用するソフトウェアの面が遅れており、研究が急がれるところである。

筆者等は、ソフトウェア開発作業の特徴を調べてモデル化を行い、このモデルに基づいて、主にソフトウェア開発時のコマンド入力を効率化することをねらったソフトウェア開発用インタフェース sifを、エンジニアリングワークステーション上に開発した。

本稿ではまず作業モデルについて述べたあと、これを視覚的な手段を用いて支援する sifの特徴的機能について述べる。

2．ソフトウェア開発作業の特徴

2.1 作業目的の遷移

ソフトウェア開発作業を行っているとき、ユーザは複数個の作業目的を持つ。例えば、仕様書作成、コーディング、単体試験、作業記録、メイル処理等である。

ユーザの作業目的は、ウォータフォールモデルに従い、図2.1 に示すように上から下へと段階的に遷移していく。しかし、実際の作業は単純なものではなく、並行する作業への遷移とフィードバックの遷移（前段階への参照と編集）が頻繁に起こる。

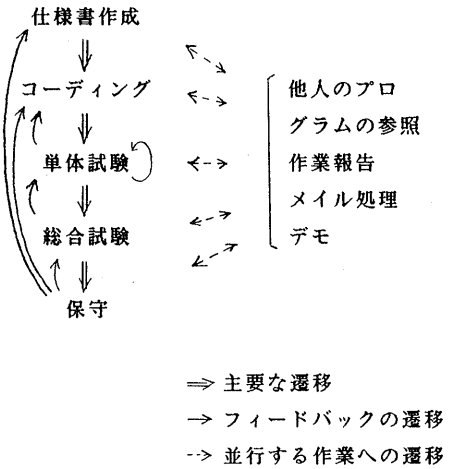

図 2.1　作業目的の遷移

2.2 各作業目的内の作業モデル

一つの作業目的に沿った作業は、主にある生産物を目標とする完成度まで作成していく作業であり、次のようにモデル化される（図2.2）。

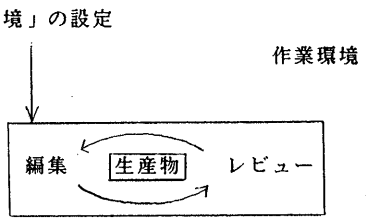

図 2.2　各作業目的内での作業モデル

作業は、まず「作業環境」の設定から始まる。これは、作業目的を達成するために必要なデータを設定し、局所化することである（ディレクトリを作り、そこに必要なファイルを複写すること等）。こうして作られた環境の下で、生産物を中心として編集とレビュー（コンパイル、実行、解析、確認等）を目標とする完成度まで繰り返す。

ユーザはここでエディタやデバッガ等のツールを使用するから、これらのツール内部（起動後）のマンマシンインタフェースも大変重要であり、インタフェース部分をツールと独立させて作る方法などが提案されているが、ここではツール内部には立ち入らず、ツールの起動までを含めたコマンド入力にのみ注目して、その効率化について考察する。

2.3 ソフトウェア開発作業の特徴

上の作業モデルに基づいてソフトウェア開発作業の特徴を以下に述べる。

1)作業目的の遷移が頻繁に発生する－バグの発生、仕様の変更、共同開発者の生産物の参照、作業の報告、メイル処理などのために、並行する作業が頻発する。

2)作業の局所性が高い－一つの作業目的内では、特定のディレクトリの特定の生産物群が対象になる。さらに、生産物の種類によって、使用するコマンドの種類とオプションの組み合わせは固定化する。

3)作業の反復性が高い－作業モデルは本質的に編集とレビューの繰り返しである。

3．sif の構成

3.1 基本構想

前章のモデルに基づき、上の三つの特徴を考慮して、sif は次のような支援を行う。

1)コマンド入力の最適化

マウス、キーボード、（ファンクションキー）を有効に利用し、コマンドを素早く構築する手段を与える。具体的には、次のような機能を提供する。

i)コマンド履歴再利用機能

以前に入力したコマンドを呼び出して、（編集して、）再利用できる。

ii)別名コマンド機能

よく使う（長い）コマンドを、（自分の気にいった）別の（短い）コマンドでおきかえる。

iii)ファイルの図的表示、選択機能

4）参照。

iv)文字列切り出し機能

画面上に表示されているファイル名等を引用できる。

これらの機能により、キーボードからの入力はより短くでき、ほとんどの場合はマウスで目の前の文字列を選択するだけで、コマンドの入力を行うことができる。

2)個人の作業環境の保持と利用

作業別にコマンド履歴、別名コマンド、環境変数（端末属性、ウィンドゥ属性等）などを「作業環境」として複数個保持できる。

3)並列作業の支援

作業は並行して進むことが多い。特に編集作業は、同時に複数のファイルを扱うことが多い。ツールをウィンドゥに対応させて、複数のウィンドゥを開くことでこれをサポートする。さらに、ツール（主に編集作業）の一時中断（棚上げ）を実現するために、これらをアイコンによって表現する。

4)ディレクトリ構造の表示とその利用

ユーザは常に自分がどこにいて、どういう状態におかれているのかを知りたがっている（ユーザは、ディレクトリの変更、現在作業の対象となっているディレクトリの表示、ファイル名リスティングなどのコマンドを多用する）ので、ディレクトリ構造を図示し、移動や現在位置表示を図的に扱えるようにする。さらに、ファイル名をリスティングし、そのファイル名を選択してコマンド入力に利用できるようにする。

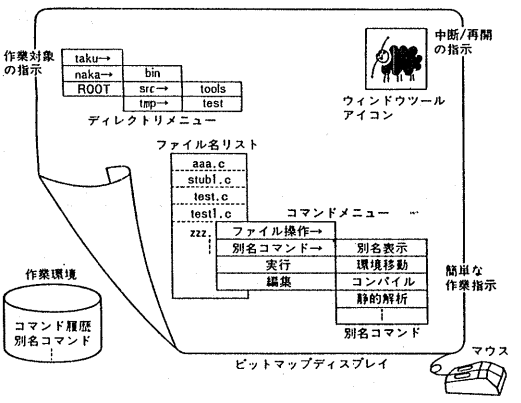

図 3.1 sif の概念図

3.2 sif の構成

sif の構成を図3.2 に示す。

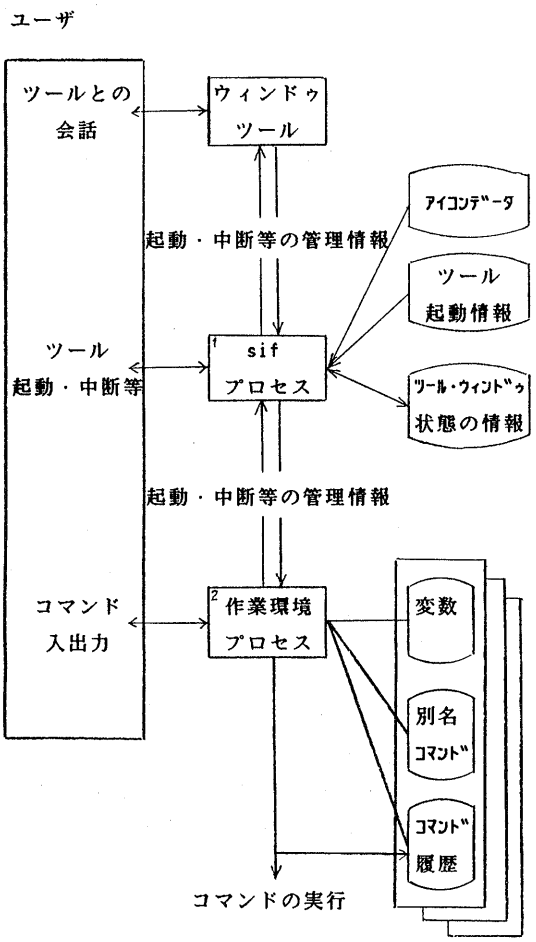

図 3.2 sif の構成

sif は、次の二つのプロセスから成る。

1)sif プロセス

作業環境プロセスおよびその他のツールの起動、終了、中断、再開等を管理するウィンドゥマネジャである。

2)作業環境プロセス

作業対象の表示やコマンドの起動をグラフィカルに行うコマンドインタプリタである。

4．特徴的な機能

4.1 ディレクトリメニュー

作業環境毎に対象とするディレクトリをいくつか定義しておくことによって、そのディレクトリ以下の木構造がポップアップメニューの形で表示される。これにより、作業を行う場所および場所間の位置関係が把握しやすくなり、その移動も容易になる。

メニューのあるアイテムを選択し、次にディレクトリ操作のメニューを選択すれば、移動やリスト表示等を行うことができる。

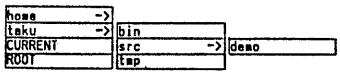

図 4.1 ディレクトリメニュー

4.2 リストウィンドゥ

ディレクトリ間のファイルのコピー等のコマンド入力時のファイル名の指定はわずらわしいものである。sif では、リストウィンドゥと呼ばれるウィンドゥによって、これを容易に、視覚的に行えるようにする。

リストウィンドゥはディレクトリ自身を図的に表現したもので、中にファイルや子ディレクトリを入れることができるいれものと考えることができる。

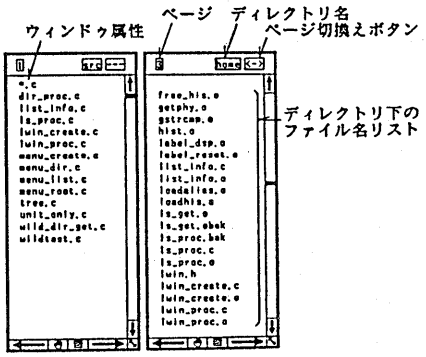

図 4.2 リストウィンドゥ

このウィンドウ上ではキーボードを全く使わず、マウスによってまずオブジェクトーいれもの（ディレクトリ名）やその中身（ファイル名）ーを選択し、次にメニューを出して操作を選択することでコマン

ドを実行させることができる。ユーザはディレクトリ構造を意識しなくてよいし、長いパス名を打ちこむ必要がない（5章参照）ので、誤作業も少なく、大変効率がよい。

このウィンドゥには属性をつけることができ、これによって、同じディレクトリを、ファイル名のパターン（*.c、*.o等）、ファイルタイプ（ディレクトリ、実行可能ファイル等）といった異なる見かたで表示させることができる。

4.3 コマンドメニュー

ソフトウェア開発作業では、特定のコマンドが多く使われる。ここでは、編集、実行といった基本的なコマンドがあらかじめ用意されているほか、作業環境毎にその作業特有のコマンド（＋オプション）を別名コマンドとしてメニューに登録しておくことができる。

このコマンドはもちろんコマンドウィンドゥ上でキーボード入力可能であるが、従来の別名コマンドが、長いコマンドを短い文字列で置きかえる方向にあるのに対して、ここでは日本語を使った、よりわかりやすい文字列に置きかえるのがふつうである。

また、コマンドメニューに出したくない別名コマンドを定義したり、他の作業環境の別名コマンドを一時的に利用するといった、ユーザの様々な要求に対応した使いかたも可能である。

作業（コマンド）をうまく整理して環境を作っておくことによって、作業の把握が容易になり、ユーザがここで何をすればよかったかというガイド的な役割も果たすことができる。

```
仕様書編集環境
    編集        spec %1 &
    文書読込    tar xvf /dev/rfd0
    文書保存    tar cvf /dev/rfd0

テスト環境
    コンパイル  cc -tp -B/lib/k -c -g
    再生成      make_tool %1 &
    デバッグ    wsdb %1* &
```

図 4.3　別名コマンドの例

4.4 コマンドウィンドゥ

キーボードを用いてUNIXコマンドを入力するためのウィンドゥである。ここでは、通常のキー入力のほか、過去に入力したコマンド文字列を検索、表示し、修正して実行することができる。

コマンド履歴の検索は、コマンドの頭文字や番号による指定だけではなく、このウィンドゥ上に一つずつ出して見ることができる。頭文字を忘れた場合でもコマンドをさがすことができ、その作業の記憶をとりもどすのに役立つ。

コマンド履歴は作業環境毎に保持しているので、割込み作業などのために履歴が失われることがなく、コマンド再利用率を高めることができる。

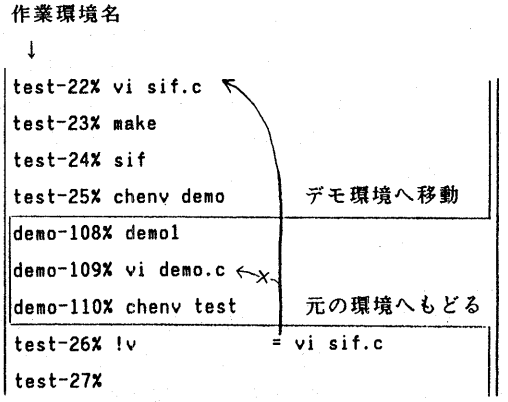

図 4.4　作業環境の移動とコマンド履歴の連続性

ほとんどの場合はキーボードに全く触れなくても作業ができるが、すべてのメニュー選択操作はコマンド文字列に変換されて実行されるので、マウスを使いたくない場合は、キーボードだけでも操作が可能で、ユーザは自分の好みに応じて使い分けることができる。

4.5 中断ボタンとアイコン

ソフトウェア開発ではいくつかの作業を並行してあるいは交互にくりかえして進める。ウィンドゥが何枚も重なって見づらいときは、ウィンドゥツールの中断ボタンをクリックすれば、そのツールのウィンドゥが消え、かわって中断中であることを示すアイコンが表示される。アイコンをクリックすると、中断中のツールのウィンドゥが再び表示される。こ

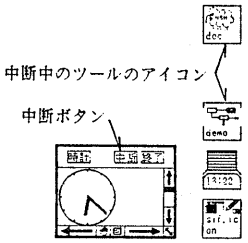

図 4.5 中断ボタンとアイコン

のように、時間のかかる作業や画面いっぱいにウィンドゥを出すツールをかたづけて、別の作業をすることができるので、大変見やすく、効率もよい。

さらに sifでは、作業中のウィンドゥ状態を保存し、再現することができる。ユーザが自分のお気にいりのウィンドゥの配置などを一度設定しておけば、ワークステーションを立ちあげる度にツールを起動したりウィンドゥを動かしたりすることなく、常に自分用のデスクができあがる。作業を途中でやめても、翌日その続きから始めることができる。

5．実現方式

5.1 作業環境

作業環境プロセスは、作業目的別に、変数、別名コマンド、コマンド履歴を保持している。これらのデータは、ユーザが与えた「作業環境名」によって管理される。作業目的を変えるときは、作業環境を切り換えるコマンドを入力すれば、その作業で使用する変数や別名コマンド、コマンド履歴が読み込まれ、利用可能になる。

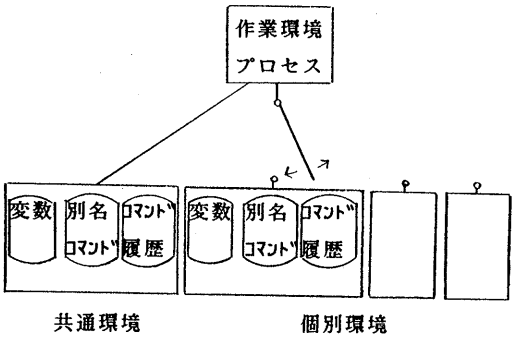

図 5.1 作業環境の切り換え

作業環境プロセスは、各作業目的別の作業環境の他に、名前を持たない「共通環境」を持っている。ここで定義した変数と別名コマンドは、各個別環境でも共通して用いることができる。また、この環境で割り込み的な作業をすることによって、個別環境のコマンド履歴をみだすことを防げるので、コマンドの再利用率をより向上させることができる。

5.2 コマンドの組み立て

作業環境プロセスは、文字列を積み上げておく、文字列管理テーブルと呼ばれるテーブルを持っており、コマンドの組み立てに使用している。図 5.2に文字列管理テーブルの概念図を示す。

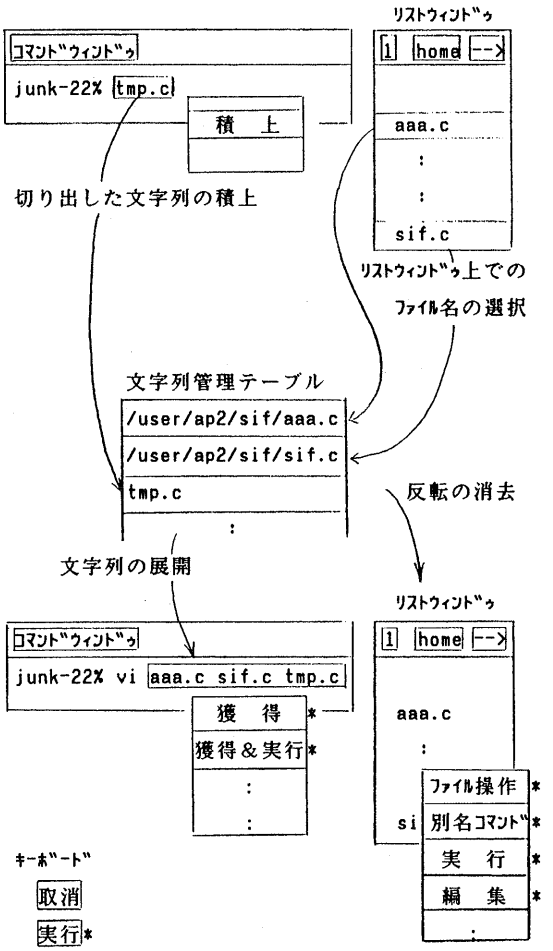

図 5.2 文字列管理テーブルへの積上・からの展開

リストウィンドゥ上でファイル名（あるいはディレクトリ名）を選択するか、コマンドウィンドゥ上のメニューの「積上」を選択することにより、文字列が順に積み上げられていく。この時、リストウィンドゥ上でのファイル名は絶対パス名に置き換えられて積み上げられる。

文字列管理テーブルからコマンド入力エリアへの文字列の展開は、図 5.2下に示した二つのメニューの *がついたアイテムの選択、あるいは実行キーのキーインによって起こる。展開される文字列は、空白で区切られた文字列となる。この時、'/' で始まる文字列（ファイル名）は、カレントディレクトリと比較して最も短いパス名に変換されて展開される。

このようにして、あるいは直接キーボードからの入力によって組み立てられたコマンドは、過去に実行したコマンドを再利用するコマンドであればコマンド履歴ファイルを検索し、別名コマンドであれば実際のコマンドに展開し、変数の記述があればその値で置き換えて、実行される。正しく展開・置換されたコマンド（のもとの形）は、コマンド履歴ファイルに追加される（図5.3）。

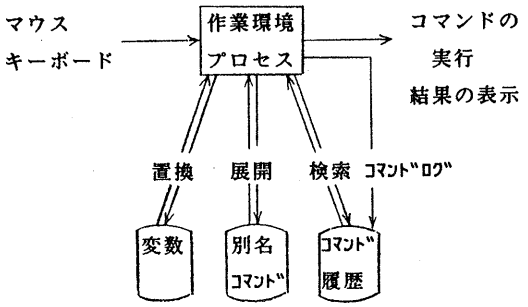

図 5.3 コマンドの実行過程

図5.4 に別名コマンドを使用した場合のコマンドの組み立ての例を示す。複写元と複写先のディレクトリ名をリストウィンドゥ上で選択した後、別名コマンドメニューの「ディレクトリ複写」を選択すると、文字列管理テーブルに積み上げられていた二つのディレクトリ名が正しいパス名に変換されて展開される。別名コマンドの定義は穴埋め式になっており、別名コマンド定義ファイル中から「ディレクトリ複写」の定義が見つかると、さきほどのパス名が順に埋め込まれてコマンドが完成する。

ディレクトリ名選択の前後でディレクトリの移動を行っても、文字列管理テーブル上では絶対パスで管理しており、展開する時点でのカレントディレクトリと比較して正しいパス名に変換するので、ユーザはディレクトリの位置関係を全く意識する必要がない。

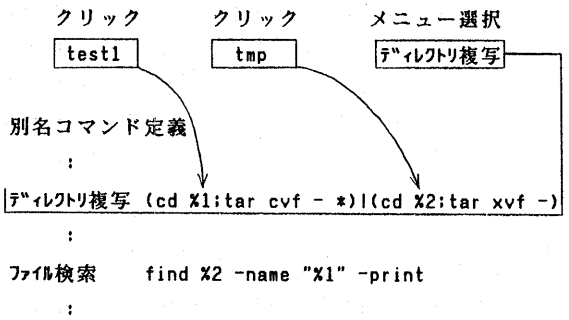

図 5.4 コマンド組み立ての例

6．おわりに

ソフトウェア開発用インタフェースsif について、その考え方と特徴的な機能および実現方式を述べた。

sif の大きな特徴は、「作業環境」の考え方である。従来のUNIXの C-shellにもコマンド履歴、別名コマンド、変数を利用する機能があるが、 C-shell の場合は、これらのデータの集合がユーザ毎にただ一つしか保持できない（ sifでいう「共通環境」だけを持つ）ため、作業目的が変わった時にコマンド履歴を再利用しにくくなり、また作業目的毎にわずかずつ異なるコマンドや変数を一カ所に登録すると効率も悪く、覚えるのも大変である。さらに、 sif では別名コマンドをメニューによって視覚化することによって、コマンド入力のストローク数を減らすこととコマンドのわかりやすさを保つことという相反する要求を満たしている。

sif のアイディアは、まず、ソフトウェア開発作業者のコマンド入力の実態調査から始まった。そのデータをもとにして作業モデルを想定し、その支援を行うことに主眼をおいて機能設計を行った。

sif 開発のユニークな点は、自分自身を使って開発を行う形態をとっていることである。つまり、基本的な機能が動作するようになった時点で実機上にインストールし、それ以後の作業は、sif 自身を使

用しながら評価・改良をくり返している。こうすることによって、設計時には気づかなかったアイディアや使い勝手の悪さ、思いがけない使い方などが早期に発見された。

リストウィンドゥやメニューといったユーザインタフェースのキーとなる部分の使いやすさ、わかりやすさの点については、やはり実現してみなければなかなかわからず、改善要求の最も多いところである。開発者たち自身による約一年間にわたる試使用によって、これらの項目が洗い出され、優先度の高いものについては既に改良を終えた。現在、より親しみやすいインタフェースへ向けて調整中である。

参考文献

[1] 中島他：UNIX上でのコマンドインタフェースの入力効率化，情処学会第33回全国大会，7F-3，1986
[2] 藤岡他：分散型ソフトウェア開発支援システム(5)，情処学会第34回全国大会，3T-5，1987
[3] 高野他：分散型開発システムで仕様書作成からプログラム生成までを支援する，日経エレクトロニクス No. 425，1987.7.13