ソフトウェア工学　46－16
（１９８６．２．７）

C言語プログラム情報の統合化

横河北辰電機（株）
西岡健自，平田陽一郎，井上健，岡垣弘美

１．はじめに

ソフトウェア開発方法論のほとんどは各開発過程で出力されるプログラム，ドキュメントを分り易くすることを主張する。それはソフトウェアの信頼性，生産性を左右する要因として人間的要素が大きな比重を占めるからで，この人間的要素の克服には"人にとっての分り易さ"が必要条件となる。

了解性を高める方法の一つは情報の分割である。構造化プログラミングやモジュール化の考え方，water fall 型ライフサイクル・モデル，これらは部分的な情報を徐々に積み上げることによって開発を進める方法である。モジュール化などは主として機能の面からシステムを横方向に分割し，ライフサイクル・モデルは外部仕様からミクロな実現方式へと縦方向に分割する。こうして最下層フェーズのモジュール分割が終れば全体として見通しの良いソフトウェア・システムが完成する。

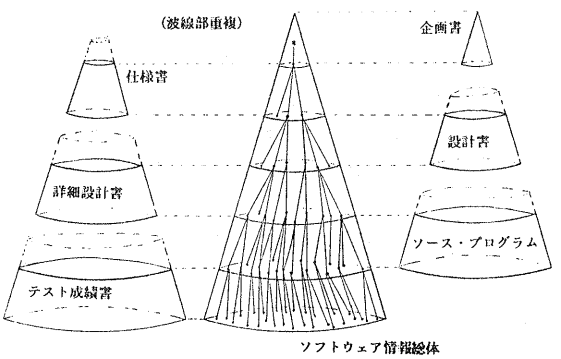
しかし，必ずしも上流から下流へと進行しないのがソフト開発の宿命で，実際にはこのようにうまくはゆかない，というのが，water fall 型モデルに対する最近の批判だが，ここでは，まず，上記のような情報の分割が機械化の不十分な現状では人為的エラーを誘発し易く，作業量も増大させるという問題を提起する。次いで，情報分割による了解性を保ちながらエラー混入を防止し，改造，修正を含めた後ろ向きの作業効率も向上させる方式を提案し，この方式に基いて構築したC言語の詳細設計の一部とコーディングを同時に支援するプログラミング環境について述べる。

２．了解性の追究に伴う問題点

情報分割による了解性の追究により二つの問題が生ずる。

第一は情報の重複に伴う，情報間の食違いや矛盾の発生である。これは人の了解性を高めるにはある程度冗長な情報を要することと，分割された情報は他の部分との相互関係を明らかとするために重複情報を必要とする場合があることによる。

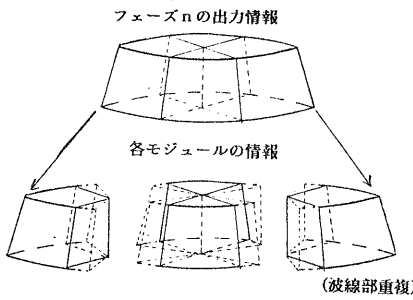
（図１）はライフサイクルに沿ったソフトウェア開発で出力される情報の概念図である。中央の円錐が仕様・設計・コーディングなど全フェーズを通しての重複のない情報の総体を表している。その左右が各フェーズ毎に出力される情報だが，それらは上位フェーズで出力された情報に重複する部分を持つ。たとえば，設計書には各実現方式がどのように外部仕様に結びつくかが述べられ，ソースプログラムは詳細設計書のほとんどの情報をプログラムの一部かコメントとして含んでいる。



（図１）フェーズ毎のソフトウェア情報の重複

モジュール分割の観点からも，（図２）のように各モジュールは他のモジュールと重複する情報を含んでいる。たとえば，大域変数や外部定義関数を使うモジュールは，それらの定義モ

ジュールで記述されているデータ・タイプや引数の個数，タイプなどの情報を直接・間接に重複して持っている。



（図２）モジュール分割による情報の重複

しかも，従来の人手に頼る開発方式ではこれらの重複する情報の記述中に人為的エラーの混入する危険が高く，仕様と設計の食い違い，インタフェースの錯誤など多くのトラブルを招く要因となっている。

更に，修正や仕様変更も重複する情報の存在によって，複数の場所を同時に直す必要が生じ，変更もれや記入ミスが発生し易い。この結果，開発末期には上流側ドキュメントとソース・プログラムの間に大きなギャップが生じ，最悪の場合には設計書等による保守が不可能となることもある。

第二の問題はこの情報重複に伴う開発，及び，保守作業の増大である。フェーズやモジュールの違いはあれ，似たような情報の記述は担当者の意欲を削ぎ，作業効率を落とす。また，第一の問題を引き起こす要因でもある記述上のミスにもつながり，信頼性の劣化を招くことになる。

３．情報統合化モデル

従来のソフトウエア開発アプローチで以上の問題に対応するものとして，次の二つが考えられる。

一つは仕様からのプログラム自動生成である。これは機械化によって人手の介入を大幅に排除するもので，縦方向の情報の重複は改善され，作業効率も上るが次のような問題がある。

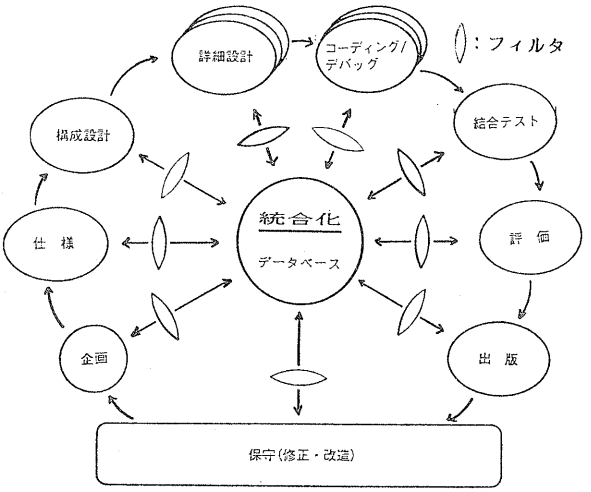
１）広い領域をカバーできるような，プログラム合成のための実際的仕様定義技術はまだ存在しない。

２）縦方向の情報重複は起り得ないが，横方向の情報重複は仕様書のレベルで改善されていない。

もう一つはソースプログラムへの情報一元化である。これは実現可能だが，仕様，設計といった段階的情報を排除することは人の了解性を損う恐れがあり，かえって信頼性を劣化させ，保守も困難となりかねない。モジュール・レベルの情報重複の解決にもならない。

上記二つの解決策はプログラムの全情報を最上流，あるいは，最下流へ一元化することによって生産性の向上を図る方式と見做すことができるが，ここで提案する方式は，いずれのフェーズにも属さない統合化情報と呼ぶデータベース（以下，統合化データベースと呼ぶ。）に全情報を重複のない形で集約させることを基本とする。また，(図３)のように各フェーズ毎にこの統合化データベースを参照・変更するための双方向のフィルタを設ける。

このフィルタはフェーズｎでユーザがモジュール名を指定すると統合化データベースの指定モジュール対応情報をフェーズｎの出力フォーマッ



（図３）情報統合化モデル

トに投影する。このフォーマットは仕様書，設計書，ソース・プログラムに対応し適度な情報重複によって，人の了解性を高めるよう構成されている。逆に，フェーズnのドキュメントに記述された情報はフィルタによって統合化データベースに反映される。

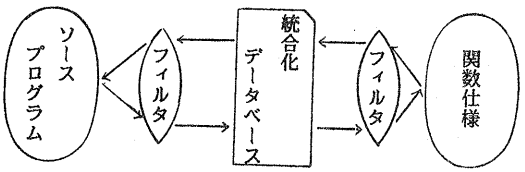
つまり，開発担当者は従来の各フェーズのドキュメントに基く段階的ソフトウェア開発を踏襲するが，彼の記述する情報はフィルタを通して統合化データベースに取り込まれ，一度記述された情報はフィルタによって各モジュールのドキュメント，あるいは，プログラムに自動的に投影される。

これによって，フェーズ間，モジュール間に渡って重複する情報を繰返し記述する必要はなくなり，ドキュメント，プログラム上で重複する情報の一致が保証される。つまり，信頼性，保守性が向上すると共に，記述量が減少して作業効率も上る。

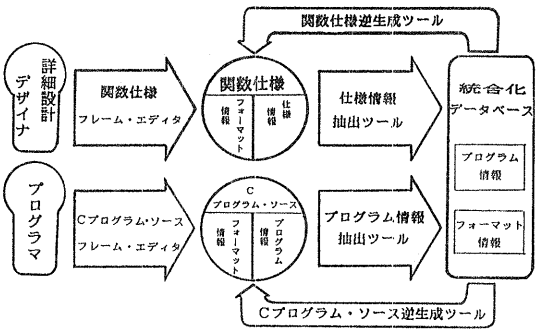
±．情報統合化プログラミング・システム

情報統合化モデルをC言語のソフトウェア開発における関数仕様定義と，コーディング・フェーズにあてはめるプログラミング・システムを開発した(図4参照)。

ここで，1ケの関数仕様書には複数の関数の仕様が記述され，対応するソースにはそれら関数のコード及びデータが記述される。このソース・プログラムがコンパイル単位にあたり，これをモジュールと呼ぶことにする。つまり，関数仕様書とソース・プログラムはモジュール毎に一対一対応する。



(図4) 情報統合化プログラミング・システム・モデル



(図5) 情報統合化プログラミング・システムの構成

(図5)に具体的情報統合化システムの構成を示す。このシステムでは統合化データベースとユーザとをつなぐフィルタを次の三つの独立なツールによって実現する。

1) 情報抽出ツール：関数仕様，ソース・プログラム各テキスト・ファイルからコメントを含むプログラム情報を抽出し統合化・データベースに格納するツールで，ユーザの記述情報をデータベースに反映させる役割を持つ。

2) 逆生成ツール：統合化データベースのデータに基いて関数仕様書，ソース・プログラムを合成するツールで，統合化情報を了解性の高いテキスト形式に再現する役割を持つ。(各出力は情報抽出ツール，フレーム・エディタの適用可能。)

3) フレーム・エディタ：情報抽出ツールによる機械処理可能な固定フォーマットを持つ関数仕様書，ソース・プログラムを効率的に編集するための構文型スクリーン・エディタで，システムの中枢的マン・マシン・インタフェースを司る。

以上のツールをつなぐのは関数仕様，ソース・プログラムのテキスト・フォーマットで，本システムでは過不足ないプログラム情報を高いreadabilityで配置できる標準的コーディング・スタイル，関数仕様書スタイルを設定して，こ

れに充てた。なお，このフォーマット情報自体も統合化データベース中に置かれ，逆生成時等に参照される。

5．構成ツールと処理の特徴

情報統合化プログラム・システムの主な構成ツールの処理と特徴について述べる。

5.1 ソース・フレーム・エディタ

通常のスクリーン・エディタの機能に加え，情報統合化システムの中枢的マンマシン・インタフェースとして次のようなねらいを持っている。

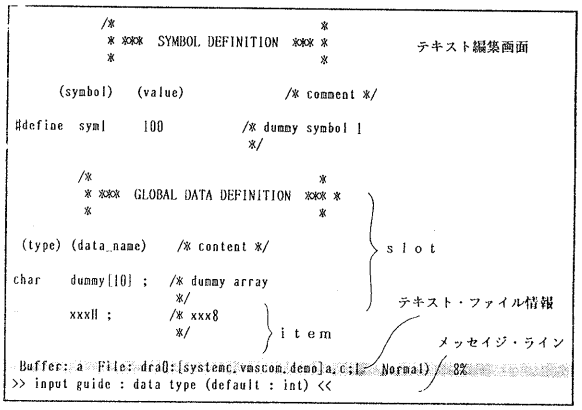
1）標準的コーディング・スタイルからの逸脱防止。

2）コーディング作業効率向上。

(1)はソース・プログラムのreadabilityの維持という意義の他に，パターンマッチングに基く情報抽出処理のための固定フォーマット保全のために必要である。(2)はこのフォーマットを遵守した上で従来以上の作業効率を実現することを意味する。これらの効果を実現させるため，ソース・フレーム・エディタはemacs上のパッケージとして実現され，次のような特徴を持つ。

1）AUTO，MUNUALモード切替方式：任意の時点で両モード相互に切替えられ，MANUALモードではVAX/VMSのスクリーン・エディタがエミュレートされる。

2）自動ガイディング：(図6)のように通常画面は二つに分割され，AUTOモードでは下部メッセージ・ラインにテキスト編集画面上のカーソル位置に対応する入力ガイドが表示される。更に，入力終了後，カーソルは次の入力欄に自動的に移動する。



(図6) ソース・フレーム・エディタ画面の構成

3）slot，item挿入・削除：現カーソル位置附近の意味的ブロックであるitem，slot(図6参照)を必要に応じて一括削除・挿入する。

4）コンパイル処理：編集の終了したソース・プログラムに対するプリプロセシング(例外処理，動的解析用プローブ埋込等。[2，3]参照。)を含むコンパイル・サブプロセス起動。コンパイル中に別ファイルの編集も可能。

5）その他：コマンド実行，他ファイル参照等のための情報参照用ウィンド管理，他。

なお，関数仕様フレーム・エディタも同様の機能を持っている。

5.2 情報抽出とデータベース管理

フレーム・エディタで編集されたテキスト・ファイルは情報抽出ツールにかけられ，フォーマット情報を除く重複のないプログラム情報がprologのホーン節の形で取り出される。関数仕様書，及び，対応するソース・プログラムからの情報抽出例を(図7)に示す。なお，ソース情報抽出ツールは構文解析に基くテキスト変換ツールの生成系であるトランスレータ・ジェネレータによって開発されている([1]参照)。

```
main  ::  main routine

 [FACULTY]   1.        member check main routine.

 [PARAMETER] 1.        int    argc    :  parameter counter

             2.  char   *argv[] :  parameter pointer array
```

(関数仕様書)

情報抽出

(抽出情報)

```
is_a(main,function).
function_type_is(main,users).
belong_to(main,msr1).
title_is(main,'main routine').
description_is(main,[1,'member check main routine.']).
is_a([main,argc],parameter).
data_type_is([main,argc],['int','|',''']).
comment_is([main,argc],'parameter counter').
is_a([main,argv],parameter).
data_type_is([main,argv],['char','*','|','[]']).
comment_is([main,argv],'parameter pointer array').
```

情報抽出

(ソース・プログラム)

```
        /*                                  *
         * ***  TYPE & PARAMETERS  *** *
         *                                  *

/*>>> type of this function is ...

      (type)    (name)  /* parameters */

                main (argc,argv)

/*>>> parameters are ...

 (type) (parameters)          /* comment  */

int   argc ;              /* parameter  counter
                           */
char  *argv[] ;           /* parameter  pointer array
                           */
```

(図7) 情報抽出例

次に抽出されたプログラム情報の構造の特徴について述べる。プログラム情報は関数，グローバル・データ，シンボルといったプログラム構成要素とその属性及び他構成要素との関係という形でモジュール化される。(図8)にグローバル・データに関するデータ構造の形式とホーン節による内部形式例を挙げる。このモジュール構造によって，(図7)のように（陰影部の情報一致），対応する関数仕様書とソース・プログラムからの抽出情報を統合でき，この統合化された情報から関数仕様，ソース・プログラム両テキストを逆生成することができる。

統合化データベースは実行プログラムの単位でprolog上に構築される。ここでは，データベース更新の問題について触れるが，プログラムの変更により情報が減少するケース(たとえば，

(データ構成)

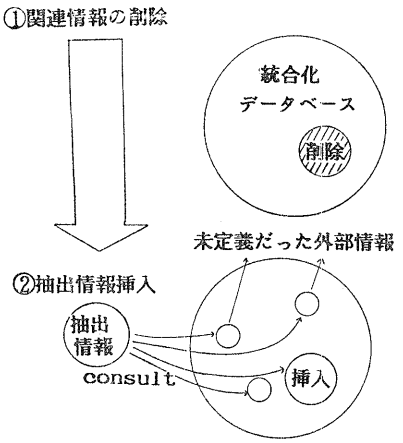
＜構成要素名＞		＜データ項目＞	＜データ＞
グローバル・データ名	属性	データ・タイプ	タイプ名
		内容（コメント）	ストリング
		初期値	データ
		サイズ	配列長
	関係	定義の所在	モジュール名
		参照関数	関数名
		変更関数	関数名
		extern指定	モジュール名

(内部形式例)

```
is_a(erflag,global).
data_type_is(erflag,['short','|']).
initial_value_is(erflag,'{0}').
belong_to(erflag,msr1).
comment_is(erflag,'error flag').
refered_by(erflag,main).
```

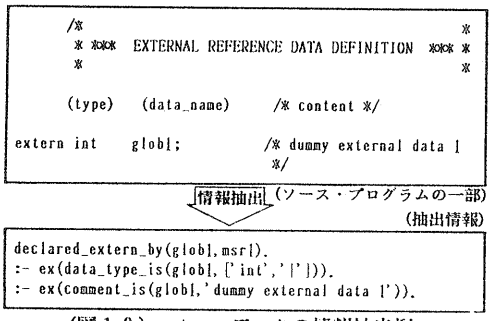
(図8) グローバル・データのデータ構成と内部形式

あるグローバル・データが削除される等。)を考えると，抽出データのreconsultでは統合化データベースに更新内容を完全な形で反映できない。そこで，(図9)のように，まず，更新モジュールに対する関連情報をデータベース上から削除した後に抽出情報をconsultする更新方式を取った。(consult,reconsultはprologのプログラム統合ユーティリティ，詳細は[6]参照。)



(図9) 統合化データベースの更新手順

更に，include file，externデータ，呼出し関数に関する情報には(図10)の例のように情報抽出時に"ex"のパラメータの形式をとらせ，各定義モジュール側の情報を優先するものとした。すなわち，"ex"はcounsult時にデータベース中にパラメータ情報が既に存在するか否かを



(図１０) extern データの情報抽出例

判断し，存在しない時のみパラメータ情報をデータベースへ格納する（図９②参照）。これによって，参照側に誤った情報が混入されていても，逆生成時には定義側の情報がとって変わる。

5.3 逆生成

逆生成は関数仕様，あるいは，ソース・プログラム各テキスト・フォーマットに統合化データベース上の指定されたモジュールのプログラム情報を埋込んで出力する処理である。

ここで，テキスト・フォーマット情報はプログラム情報と共に統合化データベース中にあり，フレーム・エディタによって形成されるテキスト・フォーマットに一致する。これは，フレーム・エディタに最初に読みこまれる各モジュールの骨格となるテキスト・ファイル，及び，item，slotの挿入で取り込まれるテキスト・パターンはすべてテキスト・フォーマット情報から合成されることによる。

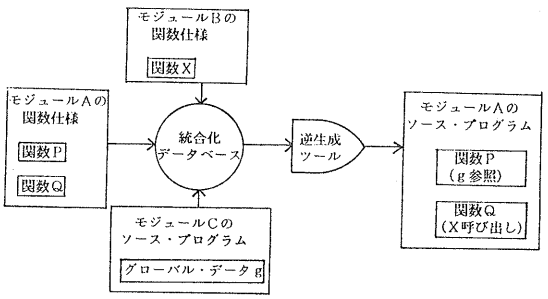
逆生成ツールは統合化データベース中の未定義の情報について出力テキスト中に空欄(具体的にはxxx_パターン)を生成する機能を持っており，これによって，関数仕様書の情報のみから，ソース・プログラム形式のテキストを生成することができる。従って，情報統合化プログラミング・システムにおけるコーディングは関数仕様から逆生成されたテキストをソースフレーム・エディタによって未定義情報を記述する作業となる。つまり，ユーザは関数仕様書のパラメータ情報などをソース・プログラム上に書き写す作業から解放され，写し違いの発生する恐れもなくなる。逆に，ソース・プログラムの変更を関数仕様書に自動反映することもでき保守性が向上する。

しかし，逆生成ひいては情報統合化の注目すべき特徴は，逆生成の出力が対応する関数仕様，あるいは，ソース・プログラムの情報を継承するだけでなく，その時点の統合化データベースの全内容を多元的に反映する点である。たとえば(図１１)のようなモジュールＡのソース・プログラムの逆生成を考えると，関数Ｐ，Ｑに関する機能，パラメータ等の情報は対応する関数仕様書の情報を継承するが，関数Ｐで参照されるグローバル変数ｇのコメント，関数Ｑで呼び出される関数Ｘのデータ・タイプなどはモジュールＢ，Ｃの関数仕様，ソース・プログラムの情報を継承する。

逆に，モジュールＡのソースが先行して作られていれば，その情報はモジュールＢ，Ｃの関数仕様書，あるいは，ソース・プログラムの逆生成時に継承される。

このような，情報継承の多元化，双方向性が開発中の人為的エラー混入を防止し，全モジュールに渡っての情報の一致を保証する。

なお，逆生成ツールは指定関数を独立のコンパイル単位ファイルとして切り出す機能も持っており，関数単位のソフトウェア再利用の便宜も図られている。



(図１１) 統合化データベースにもとずく逆生成例

６．情報統合化運用システム

以上のように，情報統合化システムは複数のツールと特殊なデータベース管理方式から構成

され，運用上の支援が必要になる。これを(図１２)のようなメニュー画面に基く運用システムにまとめた。適用範囲は(図１３)のようにモジュール構成設計からシステム生成までで，ユーザは統合化データベースをあまり意識することなくプログラミングを行える。

```
                    BASIC  PHASE  MENU

   100.      Program Information (renew or reference)
   200.      Module construction Design
   300.      Function specification Design
   400.      Coding & Compile
   500.      Executable image Generation
-------------------------------------------------< conversation area >--
- 100
     current program name : msr
     current module name  : msr|

key in new program(task) name [quit:,default:msr] :


-------------------------------------------------------------------------
C PROGRAMMING | > next,>> exit,< return,<< home,. remenu,% maintenance
```

（図１２）情報統合化運用システム画面構成例

ここで，(図１３)の全関数仕様，あるいは，全ソース・プログラムに関する処理はモジュール構成設計で定義したモジュール全体に渡る逆生成，データベース更新処理である。また，本運用システムでは，関数仕様とソース・プログラムのバージョン番号を一元化することによって両者の対応まで明かにできるバージョン管理も提供される。

７．効果と課題

情報統合化運用システムは現在試用段階にあるが，次の効果が確認されいてる。

１）関数仕様とソース・プログラムとの内容の一致が保証されており保守性が高い。

２）ソースの変更内容を自動的に関数仕様書に反映できるなど，後戻りの作業に効率的に対応できる。

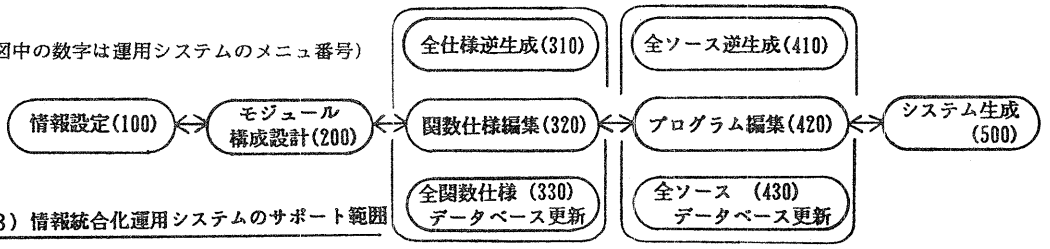
３）コーディング済の他のモジュール，及び，対応する関数仕様からソース・プログラムが部分的に自動生成され，開発効率が高い。

４）その他副次的に次の効果がある。

- 関数仕様・コーディング各スタイル標準化による信頼性。保守性の向上。
- フレーム・エディタによる編集作業効率向上。
  （キー・ストローク：従来の１／３。）
- 関数仕様，ソース・プログラム対応明示を含むバージョン管理実現。
- 関数切出しによるソフトウェア再利用可能性向上。

以上，本システムは情報統合化モデルの適用例としてはわずか２つのフェーズにまたがるもので，統合化データベースの内容も十分ではないが，主としてエラー混入防止，及び，作業量の減少の面で信頼性，保守性，作業効率向上を実現できた。

しかし，技術的問題も少なくなく，全ソフトウェア開発フェーズを一貫して支援するシステムの実現には次のような課題がある。



（図１３）情報統合化運用システムのサポート範囲

1）統合化データベースの知識情報化，分散化。

2）統合化データベース内容の充実・拡大。（仕様からコードの意味情報までの統合化，及び，管理情報等の組込み。）

3）統合化データベース活用領域の拡大。（製品管理，プロジェクト管理など。）

4）ソフトウェア再利用システムへの展開。

8．まとめ

了解性と信頼性を両立させるソフトウェア開発方式として情報統合化モデルを提案し，これに基づき，現在，仕様段階にある関数仕様定義，コーディング両フェーズにまたがるC言語向情報統合化プログラミング・システムを紹介し，評価した。

情報統合化モデルの特徴は次の通り。

1）フェーズ，モジュールに渡り重複のない形でプログラム情報を集約する統合化情報データベースを前提とする。

2）統合化情報はフィルタを通して各フェーズのドキュメントとして投影され，ユーザは従来通り，ライフ・サイクルに基いて段階的にソフトウェアを開発できる。

3）フィルタを通して統合化データベースに格納された情報は多元的，双方向的に他のモジュールに自動継承されるため，作業効率が上り，システム全体に渡る情報の一致が保証されて，信頼性，保守性が向上する。

このモデルにもとずく情報統合化プログラミング・システムは統合化データベースとしてprologを採用し，フィルタをフレーム・エディット，情報抽出，逆生成の３段階で実現したことに特徴があるが，所期のエラー混入防止効果をあげ，プログラミングの信頼性・保守性・作業効率向上を実現できた。

今後は，技術的課題の解決を図りつつより上流側への展開，プログラムのよりミクロな情報レベルでの統合化など統合化データベースの高度化，充実化を進めてゆくが，開発管理等の方面への統合化データベースの活用など多様な可能性の追究にも目を向ける必要がある。

特に，情報統合化によって信頼性の高い再利用可能なソフトウェアを高い開発効率で提供できれば，ソフトウェア再利用システムの実現性も高くなると考えられる。

参考文献]

[1] 西岡，平田　"ソフトウェア・ツールを自動生成するトランスレータ・ジェネレータの開発"　情報処理学会第30回全国大会講演論文集 7T-1 (1985)

[2] 西岡，一瀬　"例外処理機構付C言語の実現"　情報処理学会第29回全国大会講演論文集 1D-6 (1984)

[3] 井上，平田，西岡，他　"sdbを利用したCプログラム動的解析システムの開発"　情報処理学会第31回全国大会講演論文集 6F-7 (1985)

[4] 西岡，平田　"ソフトウェア移植用ＦＯＲＴＲＡＮ翻訳ツールの半自動生成"　ソフトウェア工学研究資料40 (1985)

[5] J.Gosling　"EMACS User's Reference Manual"　UniPress Software,Inc.

[6] W.F.Clocksin,C.S.Mellish　"Prolog プログラミング"　(中村克彦 訳　マイクロソフトウェア)