マイクロコンピュータ 20－3
（1982. 3. 15）

16ビットマイコン68000用クロス型テストデバッグ支援システムHITS

中所武司，田中厚，岡本恵里，本田明徳，黒崎徹
（日立製作所システム開発研究所）　（同左 戸塚工場）

1. まえがき

ソフトウェアの生産性と信頼性の向上は，最近のマイクロコンピュータをはじめとする計算機応用分野の広がりに伴い，ますます重要な課題となってきている。[1]

なかでもテスト工程は，ソフトウェア開発費用の半分を占め，生産性向上に重要であるばかりでなく，品質保証に不可欠の重要工程である。[2] しかしながら，テストデータを用いてプログラムを実行し，その結果を調べるというプログラムの検証方法は，実用面で次のような問題がある。

(1) 効果的テスト項目[3]の効率的作成方法がない。
(2) テストの準備，結果の確認作業に手間取る。

これらのテストの問題は，最近の16ビットマイクロコンピュータの普及に伴うソフトウェアの大規模化によって，この分野でも重要になってきた。そこで，我々は，これらのマイクロコンピュータ用ソフトウェアを汎用大型計算機を用いて効率良く開発するための支援システムの一つとして，16ビットマイクロコンピュータ68000用クロス型テストデバッグ支援システムHITS[4,5]（Highly Interactive Testing-and-debugging System）を開発している。

従来，この種のツールとしては，データ値の表示や変更のための実行中断機能やトレース機能を主体としたデバッグツールが多い。汎用計算機の分野では，単体テストを支援するツールとして，MSP社のMTS[6]やGE社のTPL[7]などがあるが，前者は対象言語を限定しない方式などのため，また後者はFortranサブルーチンの引数に基づいたテストに限られることなどのため，先に述べたテストの問題はあまり解決していない。その他，プログラムのフロー解析や記号実行による誤りの自動検出，パス解析による構造テスト支援などのツールの研究[8,9]も多いが，デバッグも含め，テスト工程全体を系統的に支援するものは少ない。

HITSは，このような現状把握に基づいて設計されたシステムであり，16ビットマイコンHMCS68000用のアセンブラおよび高級言語Super-PL/H[10]で記述されたプログラムの機械語オブジェクトを汎用計算機HITAC Mシリーズ上でシミュレーション実行しながら，その効果的かつ効率的なテスト，デバッグを行うものである。本文では，その設計思想，機能，処理方式について述べる。

2. 設計思想

本システムの設計は次のような基本方針に基づいて行った。

(1) 汎用大型機によるクロスシステム化。
(2) 系統的テストと効率的デバッグ支援機能の一元化。
(3) テスト作業の極小化。
(4) 高級言語とアセンブラの複数言語支援。

第1項については，16ビットマイコン用大規模ソフトウェアの開発を実機で行

う場合、
・開発用大容量ファイルが使えない、
・多数の開発担当者間で実機の同時使用ができない、
などの欠点がある。そこで、本システムはクロスコンパイラ、クロスアセンブラなどと共に用いるクロス型システムとし、汎用計算機による一貫開発システムを実現する。そのシステム構成を図1に示す。

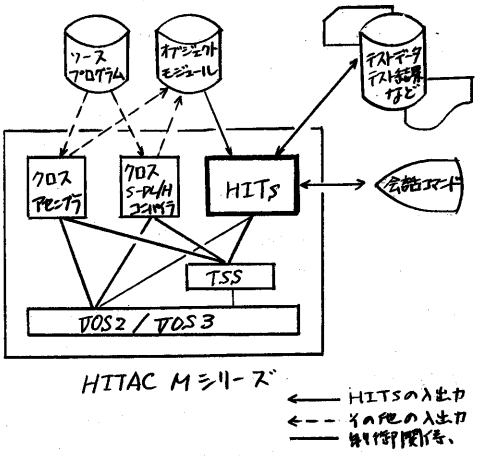

図1　システム構成

第2項については、テストがプログラムの誤りの検出を目的とするのに対し、デバッグは誤りの原因を除くのが目的であることから、従来のデバッグツールではテスト機能が不充分である一方、テストツールはデバッグ機能を有しないなど、お互い独立に開発されてきた。しかしながら、言うまでもなくテストとデバッグは表裏一体の関係にあるものであり、誤りを検出したテスト手続きに一時的なデバッグ用コマンドを付加しながらもう一度実行できれば、デバッグ効率が良い。そこでHITSでは、特定のテスト手続きを会話型実行しながら、任意の位置での中断点指定、命令やデータのトレース、データ値の表示や変更などができるようにして、臨機応変なデバッグを可能にした。

第3項については、テスト支援ツールの主な目的の1つはテスト環境の作成やテスト結果の確認のための作業を軽減することなので、ツールの使用のために必要となる作業は極力少なくしなければならない。そのため、テスト手続き記述言語をコマンド言語として単純化すると共にテスト環境作成用コマンドを充実させた。各コマンドのオペランドも被テストプログラム記述言語と同一構文とし、利用者の負担を減じた。

第4項については、本システムが主たる対象とするプログラムは、大部分を、PL/Iサブセットを母体にマイコン特有の機能を追加した高級言語Super-PL/Hで記述しているが、一部をアセンブラで記述している。そこで、単体テストに限らず、連動テスト、システムテストにも一貫して本システムを使えるように、両言語を各々のソースレベルで支援する。

3. 系統的テスト

プログラムのテストは、まず個々のモジュールを対象にした単体テスト、次にそれらを幾つか結合した連動テスト、最後に全体を結合したシステムテストと段階的に行われる。特に単体／連動テストの方法としては、そのシステムの特性に応じて、下位モジュールから順に上位モジュールを結合しながら行うボトムアップテスト法やその逆に最上位モジュールから行うトップダウンテスト法などがある。テスト支援システムは各段階のテストをそのテスト法に依らず系統的に支援することが望ましい。本文ではこの系統的テストを次の3要件で定義する。
・単体、連動、システムテストの各段階を支援する。

・各段階で同一のテストデータが使える。
・効果的なテスト項目作成を支援する。

以下、(1)～(3)でこれらう要件について述べ、(4)でそれに基づくテスト手順を示す。

（1）単体/連動テスト

単体/連動テストは次の理由でソフトウェアの信頼性と生産性の向上に重要である。

・誤り修正費用はその検出時期が遅いほど増大する。
・大規模ソフトウェアの構造テストの徹底。
・平行開発の促進による工程短縮。

さらに、最近定着しつつある構造化設計などのモジュール化技法との相乗効果も大きいと思われる。そこで、本システムでは、単体/連動テストを容易に行えるように、図2に示すような上位モジュール、下位モジュール、外部データ、入出力処理の模擬機能を備えた。

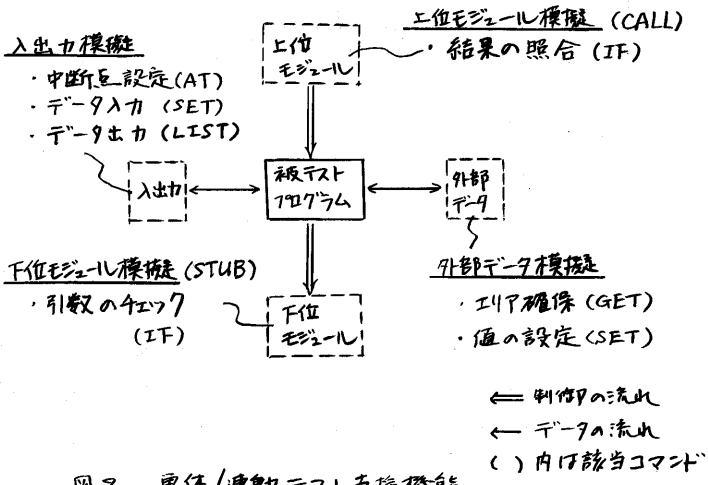

図2　単体/連動テスト支援機能

（2）テスト手続き

単体/連動テストでは、被テストプログラムの入力データの他に(1)で述べたような環境設定が必要である。そこでこれらをまとめ、さらに結果の確認を含めてテスト手続きとして記述する。そして、結合するモジュールを次第に増やしてテストする時には、不要になった部分を削除すれば前のテスト手続きを再利用できるようにする。

（3）テスト充分性評価

プログラムが仕様を満たすことを検証する方法としては、形式的に記述された仕様とプログラムの等価性を自動検証するのが理想的であるが、未だ実用には程遠い。そこで、実際には本文で対象としているようなテストデータを用いた動的な検証方法に頼らざるを得ない。従って、(2)で述べたテスト手続きはプログラムの機能仕様の代りとなる重要なものであり、テスト手続きに反映されたテスト項目の充分性がそのままプログラムの品質に反映されることになる。そこで、テスト手続きの充分性評価の1つの方法として、準備したすべてのテスト手続きを実行した時にプログラムのパスがどの程度網羅されたかを測定する。

（4）テスト工程の定式化

以上のような系統的テストを支援するHITSを用いた場合のテスト手順を図3に示す。テスト工程とプログラミング工程は仕様書ができた段階から独立に開始

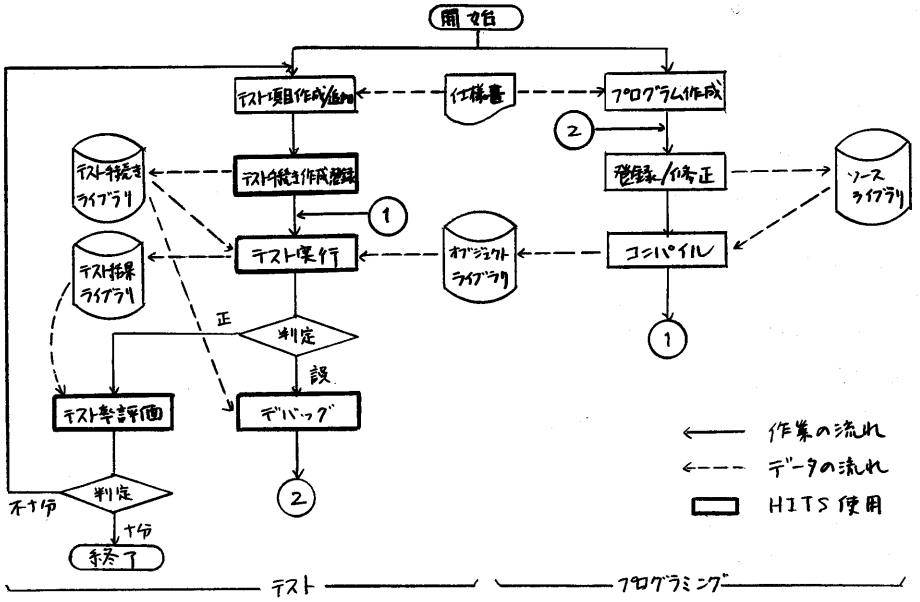

図3　HITSを用いたテスト工程

できる。まず、仕様書に基づいてテスト項目を作成し、それに対応したテスト手続きを記述する。そして、被テストプログラムが作成されると、両者を用いてテスト実行し、誤りが検出されると、簡単なものは机上で、また難しいものはHITSの会話型デバッグ機能を用いてデバッグ後，再テストする。

この間、テスト網羅率の測定も同時に行い，不充分であれば，それを補うためのテスト項目を追加してテストを繰返す。なお、HITSの使用形態としては、大量のテスト手続きの実行はバッチ処理、個々のデバッグは会話型処理を基本にしているが、作業環境次第でいずれも可能である。ソースの修正，リコンパイルについても同様である。

4. HITSの機能

4.1　テスト手続き記述言語

HITSは会話型処理形態もサポートすること、および一般的な手続き向き言語は複雑で熟練を要することなどから、テスト手続き記述言語はコマンド言語形式とした。

1つ以上のオブジェクトモジュールから成る被テストプログラムに対す

```
PROC　テスト手続き名
    {各テストケース共通のコマンド列}
  CASE　1番目のテストケース名
    {1番目のテストケース固有のコマンド列}
  END CASE
  CASE　2番目のテストケース名
    {2番目のテストケース固有のコマンド列}
  END CASE
    :
    :
END PROC
```

図4　テスト手続きの構造

るテスト手続きは、図4に示すように、ＰＲＯＣコマンドから始まり、ＥＮＤ　ＰＲＯＣコマンドで終る一連のコマンド列で構成される。通常テストケースは複数個なので、各テストケースに固有のコマンドはＣＡＳＥコマンドとＥＮＤ　ＣＡＳＥコマンドで囲む。このテスト手続きは、ライブラリ化しておいてＥＸＥＣコマンドで実行できる他，直接端末に入力して実行することもできる。

本章の以下の節では，２章の設計方針を満たすために設けたコマンドの機能について述べ、構文等の言語仕様の詳細は次章に譲る。なお、アセンブラ用と高級言語用の相違部分についてはその都度述べる。

４．２　環境模擬機能

（１）ドライバ定義

上位モジュールの模擬は、ＳＥＴコマンドによる入力パラメータの値設定、ＧＯコマンドによる被テストプログラムの起動，ＩＦコマンドによる出力パラメータの照合によって行うが，さらに高級言語用としてＣＡＬＬコマンドを設けた。

（２）スタブ定義

下位モジュールの模擬は、ＳＴＵＢコマンドのサブコマンドにドライバ定義と同様のＳＥＴ，ＩＦコマンドを用いて行う。

（３）外部データ模擬

被テストプログラム内で宣言されていない外部データは、ＧＥＴコマンドでそのエリアを確保しておけば他のデータと同様に値の設定や表示ができる。

（４）入出力模擬

被テストプログラムが入出力処理を含む場合は、ＡＴコマンドでその位置に中断点を設定し、ＳＥＴコマンドによる値の設定、ＬＩＳＴコマンドによる値の表示を行う。

４．３　環境設定作業削減機能

ＨＩＴＳのような開発支援ツールは、その使用のための作業が多くなると普及が難しくなるため、次のような作業削減機能を充実させた。

(1) 繰返して用いるコマンド列をマクロ化しておいてそのマクロ名で引用したり，ＱＵＡＬＩＦＹコマンドで修飾名の省略時解釈を指定しておいて修飾付きアドレスの指定を簡略にすることによりテスト手続きの記述量を軽減できる。さらにアセンブラ用として、複雑なアドレス式の代りとなる名称をＥＱＵＡＴＥコマンドで定義できる。

(2) 値の設定や照合に用いる定数をそのコマンドの実行毎に変えたい場合は、その定数リストをＤＡＴＡコマンドで定義しておけば、以後その定数名を定数の代りに用いることができる。

(3) 上位/下位モジュールや外部データで既にオブジェクトが存在するものは、それをLINKコマンドのサブコマンドINCLUDEを用いて結合できるため、その模擬が不要になる。

4.4 デバッグ機能

(1) ATコマンドを用いて中断点を設定でき、LISTコマンドでデータの値をチェックできる。その際、中断点で入力するコマンドが決まっている場合はそれらをサブコマンドとしてあらかじめ指定できる。また、中断点の解除はOFFコマンドで行う。

(2) 分岐、副プログラムの引用など実行順を表示する制御トレースや、データの値変更の時点を検出するデータトレースおよびその解除はTRACEコマンドで行う。その際、BREAKオプションを指定しておけばトレース対象の事象生起毎に中断するため、トレース情報の出力を必要最小限にできる。

(3) 会話型処理時にLIST, TRACEコマンドを用いて大量の情報を出力する場合、OUTオプションを指定すれば出力先をTSS端末からユーザファイルに変更できるので、後でオフライン出力すれば良い。

(4) 誤りを検出したテスト手続きをデバッグのために再実行する時、BREAKオプションを指定しておけば被テストプログラムの実行開始直前に一時的コマンドを追加できるため、テスト手続きをデバッグ用に変更する必要がない。

(5) テスト実行中にシミュレータが不当命令コードやアドレスエラーなどの異常を検出した時のダンプ出力や逆トレースなどの指示をSTUBコマンドのINTERRUPTオプションを用いてできるため、デバッグのための再実行回数が減少する。また、無限ループ対策として、バッチ用には実行命令数の最大値指定機能、会話型では割込キーによるコマンド入力機能がある。

5. コマンド言語仕様

HITSのコマンドの構文は表1に示すが、通常のコマンド言語と異なる特徴として、

(1) 手続き的概念の導入
(2) 対象言語依存性
(3) コマンド名などの短縮記法

などがある。まず(1)については、コマンドは本来個々に独立したものであり、コマンド間にめたるお互いの制約は少ない方が良いが、本言語はテスト手続きの記述に用いられるために、最小限の手続き的規則を導入した。その主なものは次の7種類のブロック構造である。

- テスト手続きブロック (PROC ～ END)
- テストケースブロック (CASE ～ END)
- コマンドリスト定義ブロック (CLIST ～ END)

表1　コマンドの構文と機能

コマンド名	オペランド 構文	機能
AT	<実行アドレス指定並び> [<コマンド>]	中断点、設定とサブコマンド指定
CALL	<エントリ名> [({ <データ指定> / <定数> } ...)]	被テストプログラムの実行開始
CASE	<テストケース名>	テストケースの開始
CLIST	<コマンドリスト名>	コマンド列のマクロ定義の開始
DATA	<定数名> <定数> ...	定数列の定義
DO		サブコマンドの開始
END	[<種別キー>]	システム，各種ブロックの終了
EQUATE	<名前> <実行アドレス指定>	新しい記号名称の追加
EXEC	<テスト手続き名> [(<テストケース名>)]	ライブラリ化されたテスト手続きの実行
GET	{ <名前> [<長さ指定>] [<絶対アドレス定数>] }...	データエリアの確保
GO	<実行アドレス指定> [<実行アドレス指定>]	被テストプログラムの実行開始
IF	<条件式> <コマンド>	条件判定とサブコマンド指定
INCLUDE	<モジュール名> ...	被テストプログラムのローディング
LINK	[<セクション番号指定>...<絶対アドレス定数>]... <コマンド>	各セクションのローディングアドレスとサブコマンド指定
LIST	<拡張データ指定>...	主メモリ，レジスタの値，メッセージの表示
OFF	[* / <実行アドレス指定並び>]	中断点、の解除
OPTION	<オプション指定> ...	各種オプションの指定
PROC	<テスト手続き名>	テスト手続きの開始
QUALIFY	<修飾名>	修飾名の省略時解釈指定
RESUME		中断点からの実行再開
SET	<代入文> ...	主メモリ，レジスタへの値の設定
STOP	[<種別キー>]	テスト手続き，テストケースの強制終了
STUB	{ <エントリ名> / INTERRUPT } [<コマンド>]	スタブ又は要素処理の定義開始
TRACE	{ PROC / BRANCH / DATA (<データ指定>...) / BACK [<整定数>] PROC / OFF }	制御，データのトレースと制御の逆トレース およびその解除

(注) 記法の説明：[ ] は省略可，{ } はいずれか1つ選択，... は繰返し

・リンケージブロック（LINK ～ END）
・スタブブロック（STUB ～ END）
・条件ブロック（IF ～ END）
・中断点ブロック（AT ～ END）

これらのブロックを構成する最初のコマンドは、いずれも複数のサブコマンドの指定が可能なものであるが、これらのサブコマンド列を主コマンドのオペランドに記述させるような構文はテスト手続きとしての読み易さに欠け、TSS 端末からのキーイン入力も複雑になる。そこで、このようなブロック構造の導入により、1行1コマンドの簡潔なコマンド構文とした。但し、後の4コマンドについては、サブコマンドが1個の場合に限りオペランドに直接書けるようにして、記述の簡便さも考慮した。

本2の特徴については、HITS がアセンブラおよび高級言語 Super PL/H を対象とするため、コマンドの構文はそれらの言語とは独立した共通の構文にすることが考えられた。しかし、アセンブラと高級言語では言語構造が非常に異なること、およびコマンドの構文が対象言語と異なると本システムの使用者にその習得のための余分の労力を強いることなどの理由から、各コマンドのオペランドの構文はできるだけ対象言語に合せた。

例えば表1の構文規則において、<実行アドレス指定>は、高級言語用ではテスト用としては手続き名、デバッグ用としては文番号を用いるが、アセンブラ用ではロケーションにつけた記号名称および16進表現の相対／絶対アドレス指定を用い、さらにそれからの変位として16進定数の加減算もできる。また、<データ指定>は、前者では変数名、後者では<実行アドレス指定>と同形式の他、レジスタ指定や間接アドレス修飾、インデックス修飾形式も可能である。

本3の特徴として、まずコマンド名は2文字目以降の仕

```
PROC TP21                                  } 被テストプログラムの
 LINK 7:1000H;9:3000H, DO                  } リンケージ方法指定
  INCLUDE SUB1,SUB2
  END
 GET COMDATA,0A000H
 SET STATE=10110000B
 STUB SUB3, DO                             } 手続き SUB3 の
  LIST P                                   } スタブ定義
  IF STATE=0F0H, SET P=40
  END
CASE C01

   〈

CASE C07                                   } 7番目のテストケース
 CALL SUB1(5,V)
 LIST V,STATE
 IF V=0, LIST 'ERROR:V=0'
 END CASE

   〉

END PROC
```

(a) テスト手続き

```
[Ready 状態]
OPTION BREAK                               } テストケースC07の
EXEC TP21(C07)                             } 実行指示
[テストケースC07の開始後 手続きSUB1の
 実行開始直後に中断。コマンド待ち。]
QUALIFY SUB1                               } デバッグ用コマンドの
AT 17, LIST STATE                          } 入力
TRACE DATA(STATE)
RESUME
[最初に STATE の値を変えた所で
 中断。コマンド待ち。]
STOP PROC
[テスト手続きの強制終了。Ready 状態]
```

(b) 会話型デバッグ

図5 HITS の使用例

意の位置から後を省略できるようにして、会話型でデバッグする時に端末から入力する文字数を減らせるようにした。但し、例えばSのように3コマンドのいずれか識別できない場合は、あらかじめシステム側で使用頻度の多いもの(SET)に決めている。

最後に、高級言語用のテスト手続きの記述例とその会話型デバッグ時の使用例を図5に示す。

6. 処理方式

6.1 システム構成

本システムは、図1に示したように汎用計算機HITAC Mシリーズのオペレーティングシステム上で稼動するが、会話型で使用する時はそのTSSシステム上で用いる。被テストプログラムは、クロスアセンブラまたはクロスコンパイラの出力するオブジェクトモジュールを対象とするため、16ビットマイクロコンピュータHMCS 68000用ソフトウェアはプログラミングからテストまで一貫して汎用計算機上でのクロス処理可能になる。なお、ソースプログラム内で用いた名前やロケーションに関する情報は、クロスアセンブラおよびクロスコンパイラがオブジェクトモジュールの付加情報として受け取る。

6.2 プログラム構成

プログラムの静的構造は、論理的には最上位にモニタ、その下にコマンド処理、最下位にシミュレータという階層構造を基本としているが、コマンドによってはその解析と実行の間に時間的にずれがあるため、実際の動的制御構造は、図6に示すような状態遷移図になる。中断点処理は、後で実行するコマンドの制御のためにコマンド処理から分離したもので、各コマンドの実行ルーチンはコマンド処理と共有している。

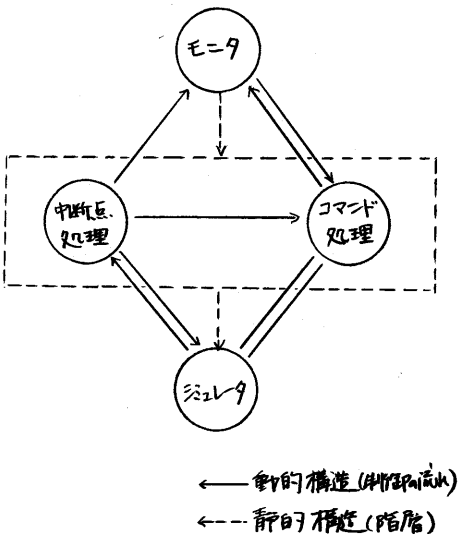

図6 HITSのプログラム構造

7. むすび

以上、16ビットマイクロコンピュータ用ソフトウェアのテストを汎用大型計算機上で効率良く行うための支援システムHITSについて述べた。

機能面では、単体、連動、システムテストの各段階を支援し、かつテスト手続きを統一的に記述するなど、テストを系統的に行えるようにした。そして、このテスト手続きは、コマンド言語として簡潔な文法にすると共に、テスト作業の極小化のためのコマンドを充実させ、使い勝手の良いシステムとした。

本システムは、高級言語Super PL/H用をアセンブラ用との結合版として開発中であり、現在、そのテストデバッグ段階である。

なお、適用段階において、HITSによる

単体/連動テストをより効果的に行うために、設計工程においても、モジュール間インターフェイスを簡潔にするような構造設計を行うことや、各モジュールの仕様を明確にドキュメント化することが望ましい。

HITSは、テスト作業の効率化を実現したが、ソフトウェアテストのもう1つの課題である効果的テスト項目作成のための機能については、テスト網羅性評価機能だけであり、今後、これらの機能を充実させていきたい。

最後に、本研究の機会を与えていただいた日立製作所戸塚工場道家浩太郎部長ならびに、有益な御討論をいただいた同工場山本利昭、同社システム開発研究所渡辺坦、山野統一の諸氏に感謝します。

参考文献

1) Boehm, B.W.: Software Engineering, IEEE Trans. Computers, Vol. C-25, No. 12, pp. 1226-1241 (1976).
2) 中所武司：ソフトウェアのテスト技法、信学会誌、Vol. 64, No. 5, pp. 549-552 (1981).
3) Goodenough, J. B. and Gerhart, S.L.: Toward A Theory of Testing: Data Selection Criteria, Current Trends in Programming Methodology, Vol. 2, Prentice-Hall, New Jersey, pp. 44-79 (1977).
4) 中所，渡辺，黒崎，山本：マイコン用クロス型テスト・デバッグ支援システムHITSの設計思想、情処学会才23回全国大会、pp. 419-420 (1981).
5) 田中，中所，岡本，本田：マイコン用クロス型テストデバッグ支援システムHITSの機能と処理方式、同上、pp. 421-422 (1981).
6) Myers, G.J.: Software Reliability, p. 360, A Wiley-Interscience, New York (1976).
7) Panzl, D.J.: Automatic Software Test Drivers, Computer, Vol. 11, No. 4, pp. 44-50 (1978).
8) Miller, E.F.: Program Testing Technology in the 1980s, Proc. Conf. on Computing in the 1980s, IEEE Computer Society, pp. 72-79 (1978).
9) 宮本勲：プログラム・テスト支援ツール：サーベイ、情報処理、Vol. 20, No. 8, pp. 688-693 (1979).
10) 神野，西野，加藤，横畑，渡辺，尾石：16ビット・マイクロコンピュータ68000用高級言語S-PL/Hの開発、情処学会マイクロコンピュータ研究会資料17-3, pp. 1-8 (1981).
11) 迫田行介：テストプログラム記述言語、情報処理、Vol. 22, No. 6, pp. 520-524 (1981).