ソフトウェア工学 81－1
（1991. 10. 4）

プログラムの設計レビューシステム

大場 克彦　二木 敬一　井上 信介　金戸 孝夫

（株）島津製作所 東京研究所

あらまし　プログラムの誤りには、アルゴリズムに関する誤りと、人間の不注意によって生じる単純な誤りがある。単純な誤りには、コーディング過程で発生するものと設計過程で発生するものがある。コーディング過程で発生する単純な誤りは、設計書から直接、ソースプログラムを自動的に生成することにより防ぐことができる。設計過程で発生する単純な誤りは、現状では人間が発見しなければならないが、単調で注意力を要するため、人間は見逃し易い。しかし、コンピュータを使えば、簡単に確実に見つけることができる。このような観点から、設計書に含まれる単純な誤りを発見し、ソースプログラムの生成可能なことを検証する設計レビューシステムを開発した。

Design Review System of Programs

Katsuhiko Ohba, Keiichi Futagi, Shinsuke Inoue and Takao Kaneto

TOKYO RESEARCH LABORATORY, SHIMADZU CORPORATION

Abstract　There are two types of errors in programs.One is errors concerned with algorithms and the other is errors caused by careless mistakes.The careless mistakes are produced in the coding process and the design process. If we use the source code generator,there will be no careless errors in the coding process.The careless mistakes in the design processes can be completely found out by the computer programs.

We developped a design review system which finds out the careless mistakes and guarantees the possibility of source code generation from the documents of program design.

1　はじめに

プログラムの誤りの中には、アルゴリズムに関するような本質的な誤り（以後、本質的ミスと呼ぶ）と、文字を書き間違えて別の意味に解釈されたとか、処理対象となるデータの名称、値、型を間違えて記述したとか、値の設定されていないデータを使って処理をしたというような、不注意による誤り（以後、単純ミスと呼ぶ）がある。現実には、本質的ミスより単純ミスの方が多い。このため、本質的ミスの原因が単純ミスの陰に隠れて発見が遅れるとか、単純ミスそのものが見逃されてしまうということが起こる。

単純ミスは、プログラムの設計の工程とコーディングの工程の両方で起こり得る。設計は人間が行うためミスは避け難い。これに対してコーディング工程で発生するミスは、設計書通りのソースプログラムを自動的に生成することができれば、避けることができる。

ソースプログラムを自動的に生成することが可能であるためには、設計書自身が、一意に解釈でき、かつコンピュータ処理できるように形式化されており、その形式に従って正しく記述されていなければならない。

設計工程で発生する単純ミスは、従来、人手による設計レビューにより取り除かれていた。ところが、人間が設計レビューを行っても、単純ミスを完全に取り除くことは困難である。単純ミスは、設計書の内容を一定の規則に基づいて１つ１つ検査することにより取り除かれる。このような作業は、単調な作業でかつ絶え間ない注意力を要するため、人間が苦手とするところである。一方このような作業は、コンピュータの得意とするところであり、コンピュータを使えば単純ミスを完全に取り除くことが期待される。

単純ミスを含まなければ、本質的誤りの原因を発見することも容易になる。その結果、信頼性の高いプログラムの開発が可能になる。

上に述べたような考察結果を基に、プログラムの設計レビューのアルゴリズムについて研究し、その結果を設計レビューシステムとして実現した。

この設計レビューシステムの目的は、設計書に含まれる設計上の単純ミスを指摘すると共に、設計書からソースプログラムが自動的に生成可能であること（以後、生成可能性と呼ぶ）を検証することである。ここで生成可能とは、設計書の中に未定義の内容がなく、設計書からコンパイル可能なソースプログラムが唯一通りに生成できることを言う。

この研究は、筆者らのグループで進めている、部品合成による自動プログラミング[1)2)]の研究の一環として行われた。なお、ここで取り扱う設計書は、筆者らの設計書の形式的表現に関する研究結果[3)-7)]に基づく記法を用いて記述される。

2　設計書

まず、レビューの対象となる設計書について述べる。

設計書は、名称部とデータ宣言部と手順部から構成される。

名称部は、設計レベルでのプログラムの名称と入出力データおよびプログラムのソースコード名を記述する。設計レベルでのプログラムの名称と入出力データは、①式の書式で記述する。

プログラム名（入力データの並び）
　　=＞（出力データの並び）　　①

データの並びは、データをカンマで区切って並べたものである。

データ宣言部は、入出力データ宣言部と内部データ宣言部から構成される。入出力データ宣言部では、入出力データの名前とデータ型（以後、型と呼ぶこともある）を宣言する。内部データ宣言部では、その設計書の内部でだけ使用するデータの名前と型を宣言する。

手順部は、プログラムのアルゴリズムすなわち処理手順を記述した部分である。

2．1　データとデータ型

データは型を持つ。そこで、データの名前と型を対応づけて宣言する。型はデータの構造と値域を表すものである。データと型の対応づけは次のように表す。

データ名--データ型名

ここではデータ型をデータの属性を表すものと考える。データ型を特徴付ける基本的な項目として、名称と構造と値域がある。名称はデータ型の識別子の役割をすると共に、その意味も表す。構造は、データがどの様な構成要素から構成されるかを表す。データ型を構成する要素もまたデータ型である。値域は、データ型の取り得る値の範囲を表す。

データ型は、基本型と定義型に分類される。

基本型は、別のデータ型で表現できないデータ型である。基本型にはINT、REAL、STR、STATE の４種類がある。INT、REAL、STRは、それぞれ整数、浮動小数点数、文字列に対応するデータ型である。STATE は複数の条件があるとき、何番目の条件と一致するかを表すのに使うデータ型である。

定義型は別のデータ型を使って定義したデータ型である。定義型には次に示す３つの形式がある。この３つの組合せにより、ここで対象とする全てのデータ型を構成する。

（１）派生型

派生型は構成要素が１つで、かつ、その構成要素が基本型あるいは値域が基本型のサブセットになっているものである。派生型は次のように定義される。

型名::基本型名;

または

型名::基本型名　<値域>;

例えば、

社員番号::INT　<1‥9999>;

は、社員番号という型名を持つ派生型のデータ型で、INT という基本型の１から９９９９の間の整数を値域とすると言うことを表している。

（２）直積型

直積型は、複数の構成要素からなるデータ型である。

直積型は次のように定義される。

型名::型名１＋・・＋型名ｋ；

（３）反復型

反復型は、直積型の特殊な場合で、その構成要素の型が全て同じ場合である。反復型は次のように定義される。

型名::型名１［ｋ］；

２．２　文

２．２．１　文の形式的表現

手順部のアルゴリズムは文を用いて記述する。このとき、記述されたアルゴリズムにあいまいさが含まれないことが必要であるが、そのためには、アルゴリズムを記述する文の意味が一意に定義されている必要がある。ここでは文の意味を、その文が表すコンピュータの動作であると定義する。

アルゴリズムを記述するのに自然言語の文を用いると、文の種類が非常に多くなり、その意味を正確に定義することが困難である。文の意味を漏れなく正確に定めるためには、ある一定の規則に基づいて意味を定めることが必要である。そのためには、文の形式的表現が必要である。形式的表現という観点から、文を次の４種類に分類する

①代入式（代入文）
②条件を表す文（条件文）
③標識文
④関数形式文

代入式は、次に示す等式で表され、右辺の計算式の計算結果を左辺の変数に代入することを意味する。

変数＝算術式　　②

算術式の中で使用できる演算子は加（＋）、減（－）、乗（×または＊）、除（÷または／）、べき乗（＊＊）、剰余（％）である。また、ＳＩＮ、ＣＯＳなどプログラム言語で用意されている組込関数やカッコも使用できる。これらの演算子の意味は明確に定義できるので、代入式は、書式が正しければその意味は一意に定まる。

条件文は反復や選択の条件を表す文である。条件文は表１の文型の中に出てくる、ＩＦ文、ＥＬＳＥ－ＩＦ文、多分岐文（ＳＷＩＴＣＨ文に相当する分岐文）、前判定文、終了条件文、ＦＯＲ文である。条件文の条件を表す論理式の中で使われる比較演算子は、左辺は右辺より大きい（＞）、より小さい（＜）、以上（≧または＞＝）、以下（≦または＜＝）、等しい（＝）、等しくない（≠または！＝）の６つである。また、論理演算子は、ＡＮＤ，ＯＲ，ＮＯＴ，ＥＸＯＲの４つである。また、論理式の中に現れる算術演算子は、算術式の中のものと同じである。比較演算子および論理演算子もその意味を明確に定義することが出来るので、条件文の条件を論理式で表すことにより、その意味を一意に形式的に定めることができる。

標識文は、目印のための文で、ＥＬＳＥ文、状態文、ＤＥＦＡＵＬＴ文、後判定文、ブロック終端文がある。

関数形式文は、代入文、条件文、標識文以外の普通の文を形式的に表現したものである。普通の文は同じ内容に対して異なる表現が可能である。また、内容は異なるが表現形式が似ている場合もある。また、必要が情報が欠落していたりあいまいな表現がされていることもある。このように、普通の文は、その数が非常に多く、また、あいまいな表現がされていたり、必要な情報が欠落しているため、前もって全ての文について、その意味を一意に定めることは不可能である。普通の文については、次の２点が必要である。

①必要な情報を漏れなく含む形式的表現
②あらかじめ意味を定義してある文は少なくても、多様な表現ができる文の構成法

ここでは、必要な情報を漏れなく含む形式的表現を関数形式文という形で与える。②の要請については、定義文により実現している。これは、２．２．２節で述べる。

プログラムの処理を記述する文には、どの様なデータが入力され、どのようなデータが出力されるか、また、入力データにどのような操作を施すと出力データが得られるかが記述されていることが必要である。これを③式の書式で実現する。

操作（Ｘ１、・・、Ｘｉ）
　＝＞（Ｙ１、・・、Ｙｏ）　　③

ここで、Ｘ１、・・、Ｘｉは入力データ、Ｙ１、・・、Ｙｏは出力データである。③式の表現は、関数的表現である。例えば、「一課と二課の人数を加算して支店人数とする」という文は

加算する（一課人数、二課人数）
　＝＞（支店人数）　　④

と表す。

２．２．２　文の種類

文には、意味が既に定義されている文（これを既定義文と呼ぶ）と意味がまだ定義されていない未定義文がある。既定義文は、さらに定義文と基本文に分かれる。基本文は、意味が明確に定義されている文のセットで、代入文、基本関数、条件文、標識文より構成される。条件文には、選択条件文と反復条件文がある。選択条件文と反復条件文には、さらにいくつかの文があるが、これは文型のところで説明する。基本関数は、入出力や文字列の処理を行う文を、関数形式文の書式③で表したものである。

定義文は、意味が自明でないので、基本文を使ってその意味を定義する。既にいくつかの定義文がある場合には、基本文だけでなく既にある定義文を使って、新しい定義文の意味を定義することが出来る。定義文は、ユーザーが新たに作成した文である。定義文を導入することにより、限られた文のセットを基に、多様な文を作成して使うことが可能になる。

未定義文は、処理手順を分かりやすくするために導入した、疑似的な文である。未定義文は、その意味がまだ定義されていないので、その意味を、現在記述し

つつある処理手順の中で記述する。

２．２．３　文型

未定義文の意味を記述するとき、一定の文型に当てはめて記述する。この文型を表１に示す。この表の各文型の１行目の未定義文の処理手順を、２行目以降で記述している。二項選択、ＥＬＳＥ－ＩＦを含む分岐、ＳＷＩＴＣＨ文に相当する分岐、前判定反復、後判定反復、ＦＯＲ文に相当する反復について説明する。

（１）二項選択

ＩＦ文の論理式が真なら、ＩＦ文のブロックを実行する。これを未定義文１で表している。論理式が偽なら、ＥＬＳＥ文のブロックを実行する。これを未定義文２で表している。ＥＬＳＥ文は、ＥＬＳＥブロックの始めを示す標識文である。

最後から２行目は、ＥＬＳＥ文のブロックのブロック終端文である。最後の行は、未定義文に対応するブロック終端文である。

（２）ＥＬＳＥ－ＩＦを含む分岐

ＩＦ文の論理式が真なら、ＩＦ文のブロックを実行する。これを未定義文１で表している。論理式が偽なら、一番目のＥＬＳＥ－ＩＦ文を実行する。ＥＬＳＥ－ＩＦ文は複数個書くことが出来る。ＥＬＳＥ－ＩＦ文の論理式が真なら、そのＥＬＳＥ－ＩＦ文のブロックを実行する。これを未定義文２で表している。ＥＬＳＥ－ＩＦ文の論理式が偽の場合は、次のいずれかの動作が行われる。

①ＥＬＳＥ－ＩＦブロックの次の文がＥＬＳＥ－ＩＦ文なら、そのＥＬＳＥ－ＩＦ文を実行する。

②ＥＬＳＥ－ＩＦブロックの次の文がＥＬＳＥ文なら、そのＥＬＳＥブロックが実行される。これを未定義文ｋで表している。

最後から２行目は、ＥＬＳＥ文のブロックのブロック終端文である。最後の行は、未定義文に対応するブロック終端文である。

（３）ＳＷＩＴＣＨ文に相当する分岐

多分岐文の変数が１なら、状態文１（状態：１）のブロック（これを未定義文１で表している）、２なら状態文２（状態：２）のブロック、・・、ｍなら状態文ｍ（状態：ｍ）のブロックを実行する。対応する状態文がない場合にはＤＥＦＡＵＬＴ文のブロック（これを未定義文ｋで表している）を実行する。

状態文、ＤＥＦＡＵＬＴ文は、各々、状態ブロック、ＤＥＦＡＵＬＴブロックの始まりを示す標識文である。

（４）前判定反復

前判定文の論理式が偽になるまで、前判定反復ブロック（これを未定義文１で表している）を繰り返す。

（５）後判定反復

終了条件文の論理式が真になるまで、後判定反復ブロック（これを未定義文１で表している）を繰り返す。後判定文は、後判定反復ブロックの位置を示す標識文である。

表１　　文　　型

順次処理

```
１　順次処理
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　　：
　|　|－□未定義文ｋ（入力並び）＝＞（出力並び）
　|
```

選択処理

```
２　二項選択
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－◇－ＩＦ：論理式
　|　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　|　★－ＥＬＳＥ：
　|　|　|－□未定義文２（入力並び）＝＞（出力並び）
　|　|
　|
```

```
３　ＥＬＳＥ－ＩＦを含む分岐
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－◇－ＩＦ：論理式
　|　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　|　◆－ＥＬＳＥ－ＩＦ：論理式
　|　|　|－□未定義文２（入力並び）＝＞（出力並び）
　|　|　　：
　|　|　★－ＥＬＳＥ：
　|　|　|－□未定義文ｋ（入力並び）＝＞（出力並び）
　|　|
　|
```

```
４　ＳＷＩＴＣＨ文に相当する分岐
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－◇－多分岐：変数
　|　|　|－※状態：１
　|　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　|　　：
　|　|　|－※ＤＥＦＡＵＬＴ：
　|　|　|－□未定義文ｋ（入力並び）＝＞（出力並び）
　|　|
　|
```

反復処理

```
５　前判定反復
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－○－前判定：論理式
　|　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　|
　|
```

```
６　後判定反復
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－○－後判定：
　|　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　|　▲－終了条件：論理式
　|　|
　|
```

```
７　ＦＯＲ文に相当する反復
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－○－ＦＯＲ：式１；論理式；式２
　|　|　|－□未定義文１（入力並び）＝＞（出力並び）
　|　|
　|
```

既定義文

```
８　代入式
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－・変数＝算術式
　|
```

```
９　基本関数
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－：基本関数（入力並び）＝＞（出力並び）
　|
```

```
１０　定義関数
　|－□未定義文（入力並び）＝＞（出力並び）
　|　|－@定義関数（入力並び）＝＞（出力並び）
　|
```

（6）ＦＯＲ文に相当する反復

ＦＯＲ文の中の論理式が偽になるまで、ＦＯＲブロック（これを未定義文１で表している）を実行する。ＦＯＲ文の中の式１は最初に一度実行される。式２はＦＯＲブロックが実行されるたびに実行される。

3　生成可能性の検証

設計書の中で必要事項が完全に定義されている（以後、設計の完全性と言う）ことと、過剰な定義がされていない（設計の無矛盾性）ことを検証する。これらの検証を行う前に、設計書が正しい書式で記述されていること（以後、構文的な正しさと言う）を確認しておく必要がある。ここでは、構文的な正しさを前提として、設計の完全性と無矛盾性について論じる。

3．1　設計の完全性

ソースプログラムは、識別部とデータ宣言部と手続き部から構成される。各部を生成するのに必要な情報について述べる。

3．1．1　識別部

識別部は、設計書の呼び出しの書式から作成される。識別部を生成するために必要な情報は、

①呼び出しの書式が名称部に記述されていること
②呼び出しの書式の中の入出力データが、入出力データ宣言部で宣言されていること
③プログラムのソースコード名が名称部に記述されていること

である。

3．1．2　データ宣言部

データ宣言部は、設計書の入出力データ宣言部と内部データ宣言部から生成される。この２つの部分で宣言されているデータの書式が正しいことを確かめる。

3．1．3　手続き部

手続き部は、設計書の処理手順部から生成される。処理手順部では、処理内容が手順として与えられているので、処理手順部の文を上から順にソースコードに変換することにより、ソースプログラムが生成できる。このことを前提にして検証を行う。

プログラムそのものを、未定義文と見なし、未定義文の意味を一定の文型に当てはめ、文を用いて記述する。こうして記述された文の中に未定義文があれば、さらにその意味を記述する。このようにして未定義文の意味を次々に記述して、図１に示すような木構造で階層的に表現したものが、処理手順である[7]。もし、この階層構造の末端の文が未定義文であると、その未定義文の意味が、まだ定義されていないことになる。その結果、処理手順そのものの意味が定義されていないことになる。このような理由で、階層構造の末端の文は既定義文でなければならない。

この検査は次のようにして行う。

まず、末端の文の文記号を調べる。文記号が未定義文ならばエラーである。

末端の文の文記号が未定義文でない、すなわち既定義文ならば、文記号が基本関数か定義文か調べる。基本関数ならば、その文の操作が基本関数辞書に登録されているか調べる。登録されていなければ存在しない基本関数を使ったことになりエラーである。文記号が定義文ならば、その文の操作が設計書に定義されているかどうか調べる。定義されていなければエラーである。文記号がその他の既定義文の場合は、構文的に正しいことを確かめる。文記号を表２に示す。

表２　文記号

文	文記号
開始文	◎ー
終了文	●ー終わり：
未定義文	｜ー□
代入式	｜ー・
基本関数	｜ー：
定義文	｜ー@
ＩＦ文	｜ー◇ーＩＦ：
ＥＬＳＥ文	｜　★ーＥＬＳＥ：
ＥＬＳＥ－ＩＦ文	｜　◆ーＥＬＳＥ－ＩＦ：
多分岐文	｜ー◇ー多分岐：
状態文	｜　｜ー※状態：
ＤＥＦＡＵＬＴ文	｜　｜ー※ＤＥＦＡＵＬＴ：
前判定文	｜ー○ー前判定：
後判定文	｜ー○ー後判定：
終了条件文	｜　▲ー終了条件：
ＦＯＲ文	｜ー○ーＦＯＲ：
ブロック終端文	｜

次に、データの検査を行う。データの検査では、既定義文の中に出てくるデータが、データ宣言部で宣言されていることを確かめる。宣言されていないデータがあればエラーである。

以上の検査が通れば、手順部の内容をソースコードに変換することが出来る。すなわち、完全性が検証されたことになる。

3．2　設計の無矛盾性の検証

完全性の検証により、設計書からソースコードが生成可能なことは保証されるが、一意に生成可能なことが保証されたわけではない。一意に生成可能なことを検証するのが無矛盾性の検証である。

一意に生成できない原因として

①設計書から処理手順が一意に定まらない。
②文の意味が一意に定まらない。
③データが一意に定まらない。

が考えられる。

この内、①については考える必要がない。設計書に記述されている処理内容が、処理手順として一意に定まるように与えられているからである。ここでは、②と③について考える。

3．2．1　文の操作が一意に定まらない

定義文の操作が一意に定まらない原因は２つ考えられる。１つは定義文の操作名が重複して定義されている場合である。もう１つは、定義文のソースコード名が重複して定義されている場合である。いずれの場合も、定義文を一意にソースコードに変換できない。こ

のようなことがないことを確かめる。

3．2．2　データが一意に定まらない

データ名が重複して定義されている場合、及びデータ型が重複して定義されている場合に、データが一意に定まらなくなる。このようなことがないことを確かめる。

4　設計上の単純ミスの検出

設計上の単純ミスとして次のようなものがある。

①文字の書き間違い

文の操作名やデータ名などを書き間違えた。

②間違ったデータを処理の対象とした

Xというデータを処理するべきところを、勘違いをしてYというデータを処理の対象としてしまったと言う例である。この結果、構造の違うデータを処理の対象にしてしまい、プログラムやデータを破壊するといったことが起こる。

③インタフェースの間違い

操作名、引数の数、あるいは受け渡すデータそのものが間違っている。

④値が設定されていない変数を使用している。

4．1　設計上の単純ミスを除く方法

設計上の単純ミスの原因には、次の4つがある。

（1）文字の書き間違い

文字の書き間違いは、文法的なエラー、処理手順の中で使われている定義文や基本文が定義されてない、定義されていないデータを使った、定義文やデータ名が重複して定義されているなど、様々なエラーの原因になる。このように文字の書き間違いは、単独に検出するのは難しいが、文法的なエラーの検出、生成可能性の検証、設計上の単純ミス（文字の書き間違いを除く）の検出などを通じて取り除くことが出来る。

（2）間違ったデータを処理の対象とした

Xというデータを処理の対象とするべきところを、Yというデータを処理の対象としたというエラーは、次のようにして検出する。まず、データにデータ型を定義しておく。次に、文が処理するデータにも許されるデータ型を決めておく。そして文が実際に使われたとき、文の中に書かれたデータのデータ型が許されたデータ型かどうか検査する。

（3）インタフェースの間違い

インタフェースの間違いには3つのケースがある。1つ目は操作名の間違いである。このエラーは、生成可能性の検証によって取り除くことが出来る。2つ目は、引数の数の間違いである。これは文法エラーの検査により取り除かれる。3つ目は、受け渡すデータそのものが間違っている場合である。これは、「間違ったデータを処理の対象とした」という場合に含まれる。

（4）値の設定されていない変数を使用している

文の入出力変数の間の関係が、以下に述べる条件を満足しているか検査する。もし満足しておれば、文の入力変数には値が設定されていることになる。そうでなければ、文が値の設定されていない変数を使用していることになる。

①ある文の入力変数が、それまでに実行された文の出力変数の中にある。

②ある文の入力変数が、プログラムの入力変数の中にある。

文の書式が、入力データ、出力データを明確に識別できるように定められているので、上に述べた検査を行うことは容易である。

以上の考察結果から、設計上の単純ミスは次のいずれかの方法により取り除くことができることが分かる。

①生成可能性の検証（文法エラーの検出を含む）

②データ型の一致を検査する

③文の入出力変数の間の関係の検査

この内①と③については既に述べた。次に②について述べる。

4．2　データ型の一致の検査

4．2．1　文とデータ型

文の中に現れるデータには、許されるデータ型が定められている。以下にそれを示す。

（1）定義文

定義文に許されるデータの型は、次のように定義される。ここで定義文の中のデータ型は定義型である。

操作名（入力型1、・・）＝＞（出力型1、・・）

例えば次のように表される。

平均濃度計算（濃度集合、サンプル数）＝＞（濃度）

この式は、操作名が平均濃度計算、入力データのデータ型が、濃度集合、サンプル数、出力データのデータ型が濃度であることを示している。

定義文のデータ型は定義文辞書という形にまとめられている。

（2）基本関数

基本関数に許されるデータ型の定義は、データ型が基本型である点を除けば定義文と同じである。基本関数のデータ型は、基本関数辞書という形にまとめられている。

（3）式

代入式と論理式をあわせて式と呼ぶ。式の中のデータは、INT かREALに変換可能でなければならない。これは、データ定義を左から右への変換規則とみなし、この変換規則を適用した時、式の中のデータが、INT かREALに変換出来なければならないということである。

4．2．2　データの誤りの検出

ここでは、定義文に渡されたデータの誤りを、データ型を用いて検出する例を示す。図1に設計書の処理手順をを示す。

図2に設計書のデータ宣言部を示す。入出力データは、入出力データのデータ及びデータ型の定義を行っている。（1）－（6）は、サンプル、数、危険、上値、下値、状態というデータのデータ型が、各々、測定値、個数、危険率、実数、実数、整数であることを表している。（7）－（11）は、測定値、個数、危険率、実数、整数というデータ型を基本型で表している。内部データは、処理手順の中に現れるデータの中で、入出力データ以外のデータを定義している。

```
◎－平均値の区間推定（サンプル、数、危険）＝＞（上値、下値、状態）
｜－◇－ＩＦ：数＜＝２
｜　｜－□サンプル数が少ない（）＝＞（状態）
｜　｜　｜－・状態＝１
｜　｜　－
｜　◆－ＥＬＳＥ－ＩＦ：数＞＝３１
｜　｜－□サンプル数が多すぎる（）＝＞（状態）
｜　｜　｜－・状態＝２
｜　｜　－
｜　★－ＥＬＳＥ：
｜　｜－□区間推定（サンプル、数、危険）＝＞（上値、下値、状態）
｜　｜　｜－@ｔ分布のＰＰ計算（数、危険）＝＞（ｔ）
｜　｜　｜－@平均値計算（サンプル、数）＝＞（平均値）
｜　｜　｜－@標準偏差の計算（サンプル、数）＝＞（ｓ）
｜　｜　｜－□推定値の計算（平均値、数、ｔ，ｓ）＝＞（上値、下値）
｜　｜　｜　｜－・上値＝平均値＋ｔ＊ｓ／√平方根（数）
｜　｜　｜　｜－・下値＝平均値－ｔ＊ｓ／√平方根（数）
｜　｜　｜　－
｜　｜　｜－□正常状態の設定（）＝＞（状態）
｜　｜　｜　｜－・状態＝０
｜　｜　｜　－
｜　｜　－
｜　－
｜
●－終わり：
```

図１　設計書の処理手順部

定義文辞書には、定義文の入出力データのデータ型と定義文の操作名のソースコードが定義されている。「ｔ分布のＰＰ計算」の入出力データのデータ型は、⑤式のような書式で定義文辞書に登録されている。

```
１　入出力データ
　（１）サンプル--測定値；
　（２）数--個数；
　（３）危険--危険率；
　（４）上値--実数：
　（５）下値--実数；
　（６）状態--整数；
　（７）測定値::REAL[30]；
　（８）個数::INT；
　（９）危険率::REAL；
（１０）実数::REAL；
（１１）整数::INT；
２　内部データ
　（１）ｔ--実数；
　（２）ｓ--実数；
　（３）平均値--実数；
```

図２　設計書のデータ宣言部

ｔ分布のＰＰ計算（自由度、危険率）＝（実数）　⑤

これは、「ｔ分布のｐｐ計算」の入力データの第１引数が自由度、第２引数が危険率、出力データが実数というデータ型であることを示している。

設計書の処理手順（図１）の中で「ｔ分布のｐｐ計算」という定義文が使われている。この中の入力データの第１引数「数」が誤りである。これは、次のようにして検出される。

①処理手順の中の定義文「ｔ分布のｐｐ計算」の入出力データのデータ型を求める。これは設計書の入出力データから求める。

②定義文辞書に登録されている「ｔ分布のｐｐ計算」の入出力データ型を求める。

③得られた２つのデータ型を比較し、一致しないものがあればデータの指定間違いである。

「ｔ分布のｐｐ計算」について、図１、図２の設計書と定義関数辞書から得られたデータ型を表３に示す。

表３　設計書と辞書のデータ型の対応表

	設計書	定義関数辞書
入力引数１	個数	自由度
入力引数２	危険率	危険率
出力引数	実数	実数

この結果から入力データの１番目の引数の型が、辞書では「自由度」になっているのに、設計書では「個数」になっており、間違っていることが検出される。

5　設計レビューシステム

以上の検討結果を基に設計レビューシステムを開発した。設計レビューシステムの機能（図３）は、大きく分けると、文法の検査、名前の検査、入出力データの検査の３つになる。文法の検査では、ブロック構造の検査と書式の検査を行っている。ブロック構造の検査では、階層構造と文の並びの検査を行っている。

階層構造の検査は、未定義文、ＩＦ文、ＥＬＳＥ－ＩＦ文、多分岐文、前判定文、後判定文、ＦＯＲ文に対応する各ブロックについて行う。

文の並びの検査は、文がその文型に応じて表１に示したのと同じ順番に並んでいるかを検査する。

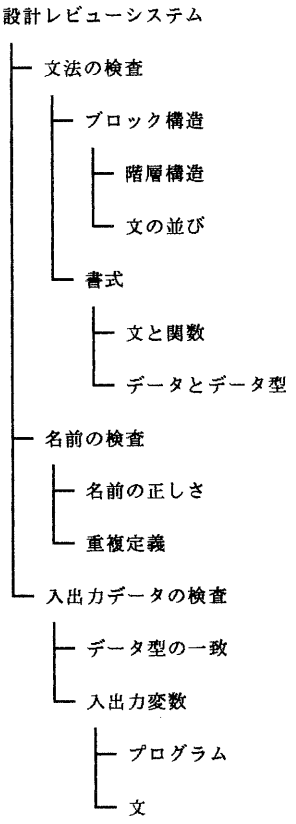

図３　設計レビューシステムの機能構成

書式の検査は、文と関数およびデータとデータ型の宣言が、正しい書式で記述されていることを検査する。

名前の検査では、基本関数、定義文、データ、データ型について、その名前が正しいか検査する。また、定義関数、データ、データ型について、重複定義がないか検査する。

入出力データの検査では、データ型の一致と入出力変数に関する検査を行う。

6　まとめ

設計書に含まれる設計上の単純ミスを指摘すると共に、設計書からソースプログラムが自動的に生成可能であることを検証する、設計レビューシステムを開発した。

６種類の評価用の設計書を作成し、この中に意図的にエラーを組み込み、それが、この設計レビューシステムで、全て検出されることを確かめた。組み込んだエラーは４２項目１９７種類で、その全てを検出した。

設計レビューシステムでレビューが完了した設計書から、ソースプログラム生成システムを用いてソースプログラムが生成できることを確かめた。ソースプログラム生成システムの基本的な考え方については、文献８）で述べた。ソースプログラム生成システム全体については、機会を改めて報告する予定である。

今回開発したプログラムの設計レビューシステムをプログラム開発の場で実際に使用し、使いやすいものにしていくことが今後の課題である。

参考文献

１）荻本浩三、奥山哲史、西伸彦、大場克彦、金戸孝夫：プログラム部品合成法によるソフトウエア開発支援環境、情報処理学会第４２回（平成３年前期）全国大会講演論文集、ｐｐ．５－２８７　－　５－２８８（１９９１）

２）荻本浩三、下田宏、杉本武和：プログラム部品合成システム、ソフトウエア工学研究会資料、７７－１２、ｐｐ．６９－７４（１９９１．２．７）

３）大場克彦、金戸孝夫：代数的仕様の実用プログラムへの適用に関する考察、情報処理学会第４０回全国大会講演論文集、ｐｐ．１０２９－１０３０（平成２年前期）

４）大場克彦、金戸孝夫：抽象データ型に基づくプログラム設計（１）、情報処理学会第４１回全国大会講演論文集、ｐｐ．５－１８５　－　５－１８６（平成２年後期）

５）大場克彦、井上信介、二木敬一、金戸孝夫：抽象データ型に基づくプログラム設計（２）、情報処理学会第４１回全国大会講演論文集、ｐｐ．５－１８７　－　５－１８８（平成２年後期）

６）大場克彦、金戸孝夫：形式的仕様の木構造表現、情報処理学会第４２回全国大会講演論文集、ｐｐ．５－１８９　－　５－１９０（平成３年前期）

７）大場克彦、金戸孝夫：プログラム論理図の形式的表現、ソフトウエア工学研究会資料、８０－２０、ｐｐ．１５１－１５８、（１９９１）

８）大場克彦、井上信介、二木敬一、金戸孝夫：知識ベースを利用したソースプログラム生成システム、１９９１年度人工知能学会全国大会（大５回）論文集、ｐｐ．６７１－６７４（１９９１）