ソフトウェア工学 89－2
（１９９２．１２．１４）

# Ｚ言語によるソフトウェアの仕様開発

柴山武彦[*1] 内記美絵[*1] 篠木裕二[*2] 大槻繁[*3]
＊１：日立中部ソフトウエア㈱
＊２：㈱日立製作所ソフトウエア開発本部
＊３：㈱日立製作所システム開発研究所

近年，標準の記述やソフトウェア開発への適用事例が報告されている形式的仕様記述技術に注目し，実用的なソフトウェア生産性向上のために，同技術を利用した開発支援環境を考えている。形式的仕様記述言語の記述実験結果からＺ言語に注目し，ソフトウエア開発におけるＺ言語の効果を検証するため，実用的な規模のソフトウェア開発にＺを適用した。Ｚ記述は，データ間の論理的関係やデータに対する操作とデータ間の関係といった問題構造を明らかにでき，対応する処理方式を構築する点で，最も効果があった。しかし，インプリメント上の制限事項が仕様記述に漏れるなど，実用的なソフトウェア開発のための仕様記述としては課題もあった。今後，Ｚ言語による実用的なソフトウェア開発向けの記述法の開発が大きな課題である。

# Development of Specification for Practical Software using Z

Takehiko Shibayama[*1] Mie Naiki[*1] Yuji Shinoki[*2] Shigeru Otsuki[*3]
＊１：Hitachi Chubu Software, Ltd.
＊２：Software Development Center, Hitachi, Ltd.
＊３：Systems Development Laboratory, Hitachi, Ltd.

Recently, Formal Description Technique has been reported as method for describing standards and developing software. We focus on this technique. We are trying to make an environment for supporting practical software development with this method. We selected Z language through experimental description for standards with several kinds of formal description languages. Now we described specification of practical software with Z for the purpose of verifying its effect. With description using Z, we made main points of argument clear. Those points are logical relation between data, and relation between user's operation and data. But description using Z has problems awaiting solution such as missing of restriction for implementation. On the next stage, the most important work is development of descriptive manner for specification of practical software.

## 1．はじめに

形式的仕様記述技術によるソフトウェア開発は，古くから欧州を中心に盛んに研究が行われているが，実用性という観点からはまだ課題があった。しかし，近年，①ＯＳＩの通信規格が形式的仕様記述言語により記述され流通し始める[1,2,8]，②欧州の企業を中心に，製品への同技術の適用事例が報告される[3]，③高品質なソフトウェア作成が容易な同技術に注目が集まっている[4]等の動きがある。これらは，厳密な構文規則と一意な意味規定を持つ形式的仕様記述言語の能力に注目した動きと言える。

我々は，実用的なソフトウェアの生産性向上を目指す立場からこの技術に注目し，高品質なソフトウェアを効率よく開発するために，この技術を適用した開発支援環境の構築を目指している。

これまでに，形式的仕様記述言語を用いた記述実験を行い，オックスフォード大学のＺ言語[5]に注目した。具体的には，①仕様記述対象の持つ概念を整理できる点，②前提知識なしで修得，理解できる点，③数学的手法で検証ができる可能性，である。

しかし，記述実験が曖昧さのない，簡単な標準を題材とした実験であったため，Ｚの概念整理の能力を含めた，実用的なソフトウェア開発での効果を中心とする適用性に課題が残った。そこで，今回，Ｚ言語で，仕様検討，仕様記述を行い，ソフトウェアの機能仕様書を記述した。

その結果，Ｚ記述は，データ間の論理的な意味関係や，データ操作と対応するデータ間の関係等の問題構造を明らかする点，対応するソフトウェアの処理方式を構築する点で最も効果があった。しかし，インプリメント上の仕様記述漏れが発生するなど，Ｚ言語だけで実用的なソフトウェア開発のための仕様記述を行うには，課題があることも分かった。今後，Ｚを実用的なソフトウェア開発で使うためには，Ｚによる記述法の開発が大きな課題である。

## 2．Ｚ言語

Ｚ言語は，集合論と論理学に基づく言語で，数学的な理論を背景とした手法で，仕様を記述することを目指している[6]。また，ＯＤＰ(Open Distributed Processing)に関連して標準化の動きが出始めていたり[8]，抽象度を高く記述することにより仕様を再利用できる可能性もある。また，Ｚによるソフトウェア仕様の記述形態に関する議論もある[7]。

我々は，記述実験を通してＺ言語の持つ３つの能力に，特に注目した。第１は，記述対象の持つ概念を整理できる能力である。例えば，データの集まりがあった場合に，それがデータ要素の重複を考慮するか，データ要素の順序性まで規定するかによって，Ｚでは記述の仕方が変わってくる。重複を考慮しない概念はSet(鮨)であり，重複を考慮する概念の場合はBagである。また，順序性を規定する概念を著す場合はSeqである。

この様な記述の区別により，記述対象の持つ概念(上の例ではデータの集まりの持つ性質)を整理できると考えた。しかも，データ構造を記述できるので適用範囲が広い点も評価した。

第２に注目したのは，特別な前提知識なしでＺ言語を理解できることである。一般に難解と言われる形式的仕様記述言語だが，Ｚ言語は，初等の集合論や論理学で，充分，理解できる。今回，Ｚによる仕様検討，仕様記述を行った担当者も特別な数学の知識を持っていなかった。

第３には，数学的な検証を行える可能性である。Ｚは数学的な理論を背景とした言語であり，論理学的な手法で仕様の検証を行える可能性がある。仕様の検証は，実用的なソフトウェアを開発する中で，大きな課題のひとつである。

## 3．Ｚ言語による仕様記述

### 3．1 概要

今回，仕様記述の対象としたのは，インタフェーステーブルを編集対象とするエディタである。インタフェーステーブルとは，既存のデータ生成ソフトがインプリメント上の必要から作成する中間テーブルである。データ生成ソフトとインタフェーステーブル，エディタ，画面制御ソフト間の関係を図3.1-1に示す。画面制御ソフトは，画面

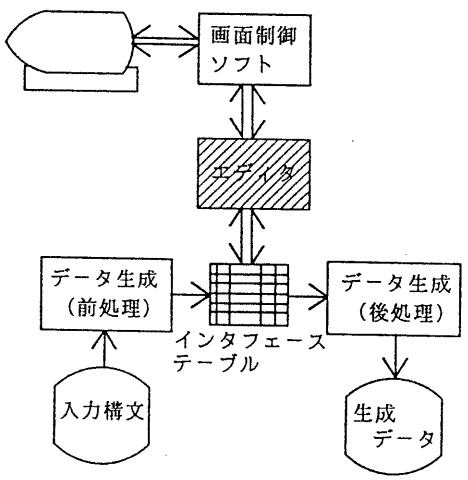


図3.1-1 開発対象

定義やデータのマッピング，入出力を行う。

インタフェーステーブルは既存だが，エディタは外付けで新規開発し，独立性が強い。規模も約7ksteps程度で，適当な大きさだと判断した。

インタフェーステーブルの仕様は，データ生成ソフトのテーブル仕様書として既に存在していた。また，エディタのユーザインタフェースに関連した機能部分について，第３者から要求仕様書が提示される予定であった。それらから，Ｚを用いてエディタの機能仕様書を記述した。作業は，①Ｚ言語によるインタフェーステーブルの記述，②要求仕様書の提示，③Ｚ言語によるエディタの仕様検討，機能仕様書の記述の順で行った。この作業は，２名で分担しながら進めた。

### ３．２ 想定効果と確認項目

我々は，形式的仕様記述言語の評価のために，曖昧さのない，簡単な標準を題材とした記述実験を行った経験があった。その経験からＺ言語による仕様検討，記述による効果等を想定した。以下では，その想定項目と，ソフトウェア開発への適用性評価のために，確認すべき項目として挙げた項目を示す。

（１）Ｚ言語による想定効果と確認項目

①記述対象の概念整理能力

我々は，Ｚ言語の持つ記述対象の概念整理能力に，最も注目している。これは，Ｚに，記述対象の持つ各種概念を厳密かつ詳細に区別して表現する能力があることによる。Ｚで記述対象を表現する場合，まず，それが持つ概念をＺの記号に従って整理し，明らかにする。その上で，その結果をＺ言語の持つ厳密な構文規則と一意な意味規定で形式的に記述する。我々は，これらの作業を通して，Ｚにより記述対象の持つ概念を整理できると考えている。

ソフトウェア開発では，開発対象(＝仕様記述対象)の持つ概念を整理しないまま開発し，仕様に不良を作り込む例がある。Ｚ言語で，記述対象の持つ概念を整理することで，これらの不良の作り込みを防げる。しかも，Ｚは，特別な前提知識なしで修得，理解できる。

例えば，データの集まりを記述する場合，まず，記述対象とするデータの集まりが，要素の重複を許すか，要素の順序性を規定しているかといった概念を整理する。そして，その結果を，Set，Bag，Seqで表す。

今回の開発では，Ｚにより，データとデータ相互間の論理的関係，エディタに対する操作とデータの関係を明快にした仕様記述を進められるので，信頼性の高い仕様を効率よく記述できる。この結果，機能仕様書の開発や，それに続くインプリメント時に，不良発生数が減少したり，開発時間を短縮したりできる。特に，インタフェーステーブルをＺ記述に含めたことにより，エディタで編集した後のインタフェーステーブルのデータ整合性に関して，不良や手戻りは発生しないと考えた。

②その他の想定効果

Ｚ言語は，記述対象の抽象度を高くした記述が可能である。抽象度の高い記述は，再利用できる可能性がある。今回は，始めてＺを使うので再利用できる記述はない。そこで，記述結果により再利用可能性の検討を行い，再利用による開発促進の可能性を検討する。

（２）形式的仕様記述言語のもつ想定効果

①厳密な仕様定義

形式的仕様記述言語は，厳密な構文規則と一

意な意味規定により，曖昧さのない仕様を定義できる点に特徴がある。今回は，作業を担当した２名の間でＺ言語の解釈に相違は発生しないと考えた。従って，仕様の解釈誤りによる不良や手戻りは発生しないと考えた。

②開発時間

一般に形式的仕様記述言語による仕様記述は，自然言語によるそれよりも多くの時間を必要とする[3]。仕様を厳密に規定する分，多くの時間が必要だからである。今回は，記述実験の経験から自然言語だけによる機能仕様書の開発と比べ，約３割程度の時間増加を見込んだ。

しかし，仕様のインプリメントまで含めて考えると，インプリメント時に仕様変更を伴う不良が少なくなり，ソフトウェア開発完了までの時間は短くて済むと考えている。これは，今後エディタのインプリメントを行い，不良の数や種類等のデータと共に検証していきたい。

（３）確認項目

実用的なソフトウェア仕様を，Ｚで記述するのは今回が始めてであった。そのため，実用的なソフトウェア仕様の検討，記述にＺ言語を用いた効果を検証する観点から，以下の項目を確認することとした。

・機能仕様書開発に要する時間とインプリメントに要する時間の変化。
・不良の数や種類。特に仕様変更を伴う不良についての検証。
・仕様の再利用の可能性

次に，Ｚ言語が実用的なソフトウェアの機能仕様を記述する言語として通用するかという観点から，以下の項目を確認することとした。

・記述量
・保守ドキュメントとしての有効性，理解のしやすさ。

### ３．３ 記述方針

（１）記述方針

エディタの仕様開発では，ｉ)Ｚ記述によりインタフェーステーブルのデータ整合性を保つこと，ii)Ｚ記述によりエディタを構成する機能間の関係を明確にすること，iii)Ｚ記述により前提条件を明確にした仕様を記述すること，iv)画面フォーマットはＺ記述に含めない，ｖ)概要等の項目は別に記述すること，vi)Ｚ記述をなるべく判り易くすること，の６つの方針をたてた。

まず，データ生成ソフトとのインタフェース関連の不良の発生を防ぐため，インタフェーステーブルのデータ整合性を保つことが最も重要な項目であった。そこで，Ｚが持つ記述対象の概念整理能力を，テーブルデータの概念整理に適用し，データ整合性を保つこととした。そのため，以下の作業手順でＺによる仕様検討，記述を進めた。

①インタフェーステーブル内でのデータ間の整合性を明らかにする(＝テーブルデータの概念を整理する)ため，Ｚでテーブル内データの論理的意味関係を記述する。
②要求仕様をもとに，抽象度の高いエディタの仕様記述を行う。
③②からエディタの抽象度を低くしていき，最終的に①とマージし，インタフェーステーブルを含めたエディタ記述を完成させる。
④レビューにより仕様を検証する。

以上の手順で，インタフェーステーブルの持つ概念を明らかにし，それをエディタ仕様の一部に含める。それにより，エディタで最も重要であったインタフェーステーブルデータの整合性を保つことができると考えた。

次に，エディタの持つユーザインタフェースに関連して，エディタを構成する各機能間の関係を明確にする必要があった。すなわち，各機能を，必ず最初に呼ばれる機能，必ず最後に呼ばれる機能，必要に応じて呼び出される機能に分け，それらの関係を明確にする必要である。

そこで，エディタ全体を状態遷移の考え方で４つの状態に分けた。各状態は，エディタの持つ４つの機能に対応する。そしてその機能順序を，Ｚ記述に含めて記述した。つまり，各ユーザ操作でエディタの状態が保たれるのか，遷移するのかを明記し，かつ，状態の順序性についても条件の中に含めて記述した。これによって，エディタを構成する機能間の関係を明確にできた。

また，仕様記述での前提条件の記述漏れを原因とする不良がしばしば発生することと，Z言語による記述スタイル[6]を参考に，前提条件を明確に記述することを方針とした。そこで，前提条件をレビューの重点項目とした。記述結果をレビューするだけの仕様検証に留めたのは，数学的手法による検証方法や支援系がなかったためである。

画面フォーマットは，Z記述と別に検討した。これは，Zで仕様を記述する目的が主に前出の3点だったこと，エディタの開発目的から画面上の出力形式は，あまり重要でなかったことによる。

概要(目的，適用範囲等)，前提となるハードウェアやソフトウェア，入出力データの形式，性能に関する事柄等は，別に日本語で記述した。これらは，後にマニュアルに記載したり，ソフトウェア開発や保守時に指針としたりする項目として重要であり，仕様検討の段階で明らかにし，機能仕様書に記載しておく必要があった。

記述作業時には，記述結果を判り易いものとするため，各行にコメントを付けると共に，スキーマ(＝Zの言語要素)にもコメントを付けた。また，

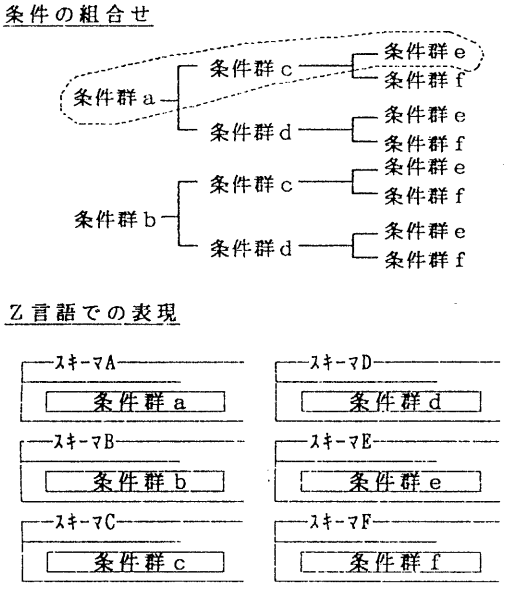


図3.2-1 インタフェーステーブルのグループ分け

変数名やスキーマ名にはできるかぎり日本語を用いてわかりやすくし，さらに，スキーマの参照順序を見やすくするために，参照順序に従った番号を含めたスキーマ名称とした。

これらの方針で，Zによる仕様検討，仕様記述を進めた。

(2) インタフェーステーブルの記述

インタフェーステーブルはデータの論理的意味関係だけでなく，インプリメント上の制約を含んでいた。そのため，データの論理的意味関係のみを明らかにし，誤解釈による不良の作り込みを防ぐ必要があった。そこで，Zによりインタフェーステーブル仕様を，整理，記述することとした。

まず，インタフェーステーブル全体は，各テーブルの複雑な結合で構成されていた。そこで，全体を入力構文の持つ要素でグループ分けできる点に目を付け，整理した。各要素の持つ意味に従った，部品となるスキーマを作成し，それらスキーマの結合(要素の組合せ)により各グループを表現

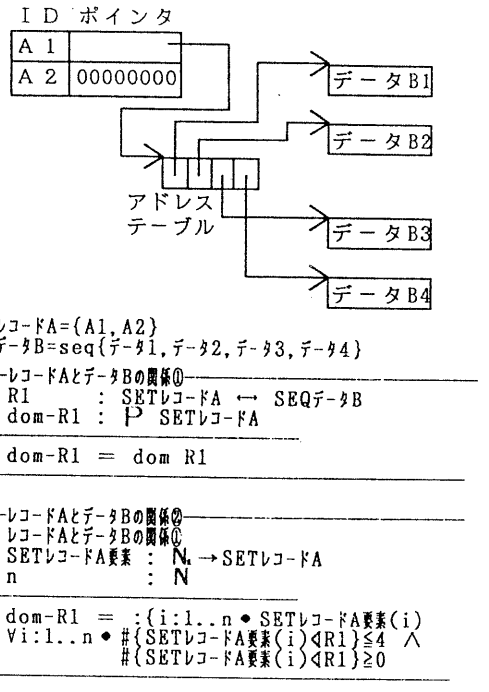


図3.2-2 Zによるデータの論理的関係の表現

し，全体を記述した(図3.2-1)。

また，部品となるスキーマの記述においても，データの論理的な意味関係のみを記述する様，留意した。例を図3.2-2に示す。この例で，アドレステーブルは，インプリメント上の必要から存在するテーブルである。データは，識別子レコードＡから０個以上４個以下の順序列データＢに対し，論理的関係がある。Ｚ記述では，この論理的関係のみを記述した。

## ４．結果と分析

### ４．１ 要求仕様書

エディタのユーザインタフェースに関連した機能部分についての要求仕様書は，Ｚ言語によるインタフェーステーブルの記述がほぼ終了する頃，第３者から我々に提示された。提示された要求仕様書は，具体的で，機能仕様書に近い内容も記述されており，以下の項目に関する要求が記載されていた。

・目的等，全体の概要
・機能構成と各機能の概要
・各機能に対応した画面フォーマット概要
・コマンド名の一覧と各コマンドの機能概要
・データ生成ソフトとのインタフェース
・規模，性能，他

我々は，Ｚの概念整理能力を生かすため，抽象度の高い記述から開始し，順に検討を進めようとしていた。ところが，要求仕様書が具体的で機能

```
CMD::=A|B|C
KIND::=ADD|DEL
┌─TABLES──────────────────────────┐
│known : ℙ ID                     │
│f     : ID→DATA                  │
│ids   : ℕ₁→ID                    │
│datas : ℕ₁→DATA                  │
│hum   : ℕ                        │
├─────────────────────────────────┤
│known = dom f                    │
│known = {i: 1..hum • ids(i)}     │
│∀i:1..hum • f(ids(i))=dates(i)   │
└─────────────────────────────────┘
┌─SAMPLE──────────────────────────┐
│処理対象名？：ID                 │
│コマンド名？：CMD                │
│詳細コード？：KIND               │
├─────────────────────────────────┤
│∃ j:ℕ₁ • f(ids(j))=処理対象名？  │
│(コマンド名？=A)∧(詳細コード？=ADD)│
│   ⇒f(処理対象名？)='△'          │
│(コマンド名？=A)∧(詳細コード？=DEL)│
│   ⇒f(処理対象名？)='＊'         │
└─────────────────────────────────┘
```

図4.1-1 データ削除のＺ記述

仕様書に近い内容も含まれていたため，順に検討する必要がなくなった。そこで，インタフェーステーブルの検索，更新方法を含めた，抽象度の低い，エディタに特化した記述のみを行った。例えば，「データ要素の削除は，指定された処理対象がインタフェーステーブルに存在する時，対応するデータにフラグ'＊'をたてる。(図4.1-1)」といった記述スタイルである。

抽象度の低い記述にのみ留めたことにより，記述結果の再利用は難しくなった。従って，検討項目のうち，再利用性の検討は今後の課題とした。

### ４．２ 結果と分析

Ｚ言語で記述することにより，信頼性の高い仕様を効率よく記述できると考えていた。しかし，実際は，エディタに特化した抽象度の低いＺ記述を行ったため，データ間の論理的関係やデータ操作とデータ間の関係といった問題構造を明らかにでき，対応する大まかな処理方式を見通すのに役立った。以下，詳細に示す。

（１）データ構造とその論理的概念の整理

インプリメント上の必要から誕生したインタフェーステーブルは，その中にインプリメント上の制約を含んでいた。そのため，データ間の論理的な関係がわかりにくかった。

これを前述の記述方針で記述した結果，テーブル内のデータ間の論理的意味関係を明らかにできた。その上で，その記述を含めたエディタ仕様をＺで検討，記述したため，データ操作とデータ間の関係を明らかにでき，対応する大まかな処理方式まで見通せた。

例として，ある変数名をキーにデータ値を検索する処理を図4.2-1に示す。図では変更前テーブルにデフォルト値が存在し，データ値が変更されると変更後テーブルに新しい値が登録される。検索対象として入力した変数名が変更後テーブルにあればその値を，なければデフォルト値を，両テーブルになければエラーメッセージを出力する。図では，対応するテーブルの仕様を省略しているが，インタフェーステーブルでは，変数名(ID)とデータ値(DATA)，変更前テーブル，変更後テーブ

ルの関係を複雑なテーブル関係で実現している。特に，変数名とデータ値は，幾つかのテーブルポインタをたどる，複雑なテーブル変換が必要であった。また，エディタ向けに新規作成した変更後テーブルと，エディタ開発以前から存在した変更前テーブルの関係も不明確であった。

```
TABLEA(変更前テーブル)
  knowna : ℙ ID
  fa     : ID → DATA
  idsa   : N1 → ID
  datasa : N1 → DATA
  huma   : N
  knowna = dom fa
  knowna = {i: 1..huma • idsa(i)}
  ∀i:1..huma • fa(idsa(i))=datesa(i)

TABLEB(変更後テーブル)
  knownb : ℙ ID
  fb     : ID → DATA
  idsb   : N1 → ID
  datasb : N1 → DATA
  humb   : N
  knownb = dom fb
  knownb = {j: 1..humb • idsb(j)}
  ∀j:1..humb • fb(idsb(j))=datesb(j)

SAMPLE
  TABLEA
  TABLEB
  名前？   : ID
  結果！   : DATA
  メッセージ！ : MSG
  ∃ k:N1 • f(idsb(k))=名前？
     ⇒結果！=fb(名前？)
  (¬(∃ k:N1 • f(idsb(k))=名前？))∧
    (∃ k:N1 • f(idsa(k))=名前？)
     ⇒結果！=fa(名前？)
  (¬(∃ k:N1 • f(idsb(k))=名前？))∧
  (¬(∃ k:N1 • f(idsa(k))=名前？))
     ⇒メッセージ！="ERROR"
```

図4.2-1 データ値検索の記述

しかし，Z記述はデータの論理的関係が明確であり，変数名とデータ値が1対1に対応すること，検索は変更前テーブルよりも変更後テーブルが優先であることが一目で分かる。

しかも我々は，インタフェーステーブルをZで記述した知識から，Z上でのデータ操作を，複雑なテーブル変換を必要とするインタフェーステーブル上へ，難しくなく対応付けることができた。その結果，ユーザ操作に対応してインタフェーステーブルデータを検索する処理を見通す場合，Z記述をもとに，それをインタフェーステーブル上でどう実現するか，簡単に見通しを付けられた。つまり，大まかな処理方式迄を見通せた。

以上の例の様に，インタフェーステーブルの検索，更新方法を含めた，抽象度の低い，エディタに特化したZ記述を行った。それにより，そのZ記述とインタフェーステーブルをZで記述した知識から，大まかな処理方式まで見通せた。すなわち，Zで，インタフェーステーブルデータの論理的関係や，ユーザ操作に対応するデータ操作といった問題構造を明確にした結果，対応する処理方式の構築が容易にできた。

（2）前提条件の記載

今回，前提条件の記載を記述方針の1つに掲げたことは先に述べた。前提条件とは，例えば，インタフェーステーブルの検索において，検索データが既に存在するといった条件である。これによる効果例を，図4.2-1で示す。

この例は，ある変数を検索する場合，変更後テーブルにそのデータ値が存在すれば変更後の値を，存在しなければ変更前テーブルを検索し，変更前テーブルにも存在しない変数はエラーとなる記述である。この記述の中で，データ値を検索する場合には，変数名が変更前テーブルか変更後テーブルのどちらかに存在することが前提条件であり，なければエラーであることを明確にしている。

この例の様に，前提条件を明確にする記述を行った結果，どんな事態が正常でどんな事態が異常(エラー)かという問題構造を，明確にできた。

（3）一部条件の抜け

機能仕様書までの開発を終えた時点で，Z記述に1つの不良が発見されている。データ値の編集で，削除や追加を繰り返すとデータ長は±∞までの値を取る。一方インタフェーステーブルでは，0～256バイトの値しか許していない。従って，編集後データ長の制限が必要である。しかし，実際のZ記述ではデータ長の制限がなかった，という不良である。これは，インタフェーステーブルに関する仕様書で，データ長の記載が不明確であったために発生したと考えられる。

この結果から，Z言語は，それだけでソフトウエアの仕様を完全に記述できるものではないことがわかった。インプリメント可能な記述にするために，インプリメント上の制限等の記述漏れを防止するための記述法の開発や，何らかの検証のための手段構築が大きな課題である。

（4）その他

Z言語による仕様検討，記述は，2名の担当者が特別な前提知識なしで，行うことができた。約7ksteps程度の規模のソフトウェア仕様の検討，記述を，特別な前提知識なしで行えたことは，Zを実用的に用いる上で大きく評価できる。

開発時間を，ほぼ同規模のソフトウェアの機能仕様書をZ記述なしで開発する時間を1とした比で表わすと，約2.07であった。しかし，今回は，記述方針の検討等に時間を費やしたので，慣れ等によって約0.57の時間を削減できると考えており，約1.50程度にできる。また，将来，記述の再利用が可能になれば，さらに削減できる。しかし，現状で，Zを用いない場合よりも仕様記述時間が長い。これを，インプリメント工程で短縮できるか，今後確かめたい。

記述量を，ほぼ同規模の日本語によるソフトウェアの機能仕様書の記述量を1とした比で表わすと，約1.33であった。両者を比較することに意味はないが，Z言語で記述したために，爆発的に記述量が増えるという事態だけは起きなかった。

Z記述には，コメントを多く付ける等理解しやすい工夫をした。しかし，完成したZ記述だけからエディタ仕様を理解するのは易しくない。今後，より理解しやすいZ記述の工夫と共に，必要となる補助ドキュメント類の検討等が必要である。

Z記述の保守実験として，小規模な変更を試みてみた。変更内容が簡単な内容であったために，変更そのものはうまくできた。しかし，Z記述は記述量が多く，スキーマの参照の仕方によっては，変更の影響範囲が大きい場合が予測される。この場合でも保守が可能か，検討する必要がある。

Z記述は，厳密な構文規則と一意な意味規定を持っている。今回の記述は，かなり抽象度の低い記述を行ったが，2名の担当者間で解釈に相違が発生していない。担当者間でレビューをしながら記述を進めた効果もあると思われるが，実用的なソフトウェア仕様の解釈に相違が発生しなかった点は評価できると考えている。

## 5．おわりに

我々は，実用的なソフトウェアの生産性向上を目指す立場から，記述実験で形式的仕様記述言語を評価し，Z言語に注目した。そして，Z言語による仕様検討，仕様記述を経て，ソフトウェア開発のための機能仕様書を記述した。その結果，以下のことが分かった。

- Zで前提条件を重視した，抽象度の低い記述を行った結果，問題構造を明確にできた。問題構造とは，データ間の論理的意味関係，ユーザによるデータ操作と対応するデータ間の関係，異常事態(ｴﾗｰ)である。その結果，対応する大まかな処理方式を容易に構築できた。
- Z記述中に，インプリメント上の仕様記述漏れが存在した。これは，Z言語それだけでは仕様を記述するために課題があることを示している。インプリメント上の制限の記述漏れを防ぐ記述法の開発や，検証する手段の構築が，大きな課題であることを示している。

今後，記述した機能仕様書から，エディタのインプリメントを行い，仕様に含まれる不良の数や種類の変化，開発時間の変化等を検証したい。また，実用的なソフトウェア開発のため，Z言語による仕様記述法の開発をはじめ，開発環境について検討していきたい。

**謝辞**

最後に，本研究に関して御指導，御助言頂いた日立製作所システム開発研究所來間啓伸氏に感謝いたします。

[1] 二木厚吉，ISOにおける形式記述技法の標準化動向，情報処理 1990 VOL.31 No.1

[2] 水野忠則，標準仕様記述言語の概観，情報処理 1990 VOL.31 No.1

[3] Anthony Hall,Seven Myths of Formal Methods,IEEE Software,September 1990

[4] Jeannette M.Wing,Specifier's Introduction to Formal Methods,IEEE COMPUTER, September 1990

[5] J.M.Spivey,The Z notation -A Reference Manual-,Prentice Hall,1989

[6] Ian Hayes,Specification Case Studies, Prentice Hall,1987

[7] 來間啓伸，集合に基づく形式的言語を使ったソフトウェア仕様の記述形態について，情報処理学会研究報告 92-SE-83,1992

[8] 情報規格調査会，1991年度の情報規格調査会の活動について，情報処理 1992 VOL.33 No.8