Z 開発試行報告
―仕様言語の必要性と仕様言語 Z によるプログラム開発の実際―

李　春瑞, 染谷　誠

1. Z 開発試行の背景と目的

1.1 はじめに

当社のある社内システムの機能拡張の際に，ウィンドウ処理プログラムを形式仕様を使って開発した．本稿ではそれについて報告する．

仕様言語には Z 言語を採用した．ウィンドウ処理プログラムを取り上げたのは，その社内システムを担当する部署で取り扱うプログラムの代表的な類型だからである．したがって，ここで取り上げる課題は，そのウィンドウ処理プログラムであり，本特集号で取り上げている「三つの共通例題」とは異なる．

1.2 背　景

1.2.1 Z 開発試行の発端

当社は国内・国外の協力会社にプログラム・コードの開発を依頼している．コードの開発を依頼するには，どのようなコードを開発して欲しいのかを正確に伝えなければならない．そのためプログラムの要求仕様書を作って，これを協力会社に提示している．ところが，その要求仕様書に判りにくいものがあり，どのようなコードを開発すべきなのかが，コード開発者になかなか伝わらないことがある．そのため種々の弊害が生じ，要求仕様書の判りにくさがあらためて問い直されている．

国外の協力会社に発注する場合，要求仕様がコード開発者に滑らかに伝わらないのは，要求仕様書作成者とコード作成者の母語の違いに拠るのではなく，往々にして原因は要求仕様書にあると言う．

要は，コード開発の際に提示される要求仕様書に，しばしば記述不足や曖昧な表現があったり，時には依頼部署毎に形式が異なっている，などの理由から，仕様書の理解が難しく，理解するまでに時間がかかりすぎ，労力や通信費が嵩むため，開発費を押し上げる傾向がある．

一方，そのような仕様書は信頼性が低くなりがちで，その結果，仕様変更が多くなり，検収テストやコードの修正に手間がかかる．同時に，それに基づいて開発されたコードの信頼性も低くなり，この面からも，検収テストやコードの修正に手間がかかる．その上，リリーズ後も問題を起こし勝ちで，これが保守コストを押し上げる要因となる．

このように要求仕様書の品質が及ぼす影響は，開発から保守まで広い範囲にかかわる．

要求仕様書の品質について，ある協力会社（以下,「協力会社」という）と当社の双方が以上のように考え，ここから，この要求仕様書の品質問題の解決策として，次のような課題が設定された．すなわち「専用の仕様言語を使って仕様書を作成すれば，要求仕様書の品質問題は解決するのか．その可否および可能性を協力会社と協力して

探れ」．このような課題である．

これを受けて，先ず，仕様言語（あるいは形式仕様言語などとも言われる）として特に仕様言語Ｚを選んだ．そして当社側がこれを使ってプログラム仕様書を書き，「協力会社」側にはその仕様書からコードを開発してもらうことにした．その開発経験に基づいて仕様言語Ｚとそれによる開発について評価しようというのである．これがＺ開発試行の発端である．

では，なぜ仕様言語なのか．要求仕様の品質問題の解決に，仕様言語がどのように有効なのか．次にこの点を掘り下げておこう．

1.2.2　要求仕様書の品質問題

それを掘り下げる前に，要求仕様の品質問題が「協力会社」と当社間だけの問題ではないこと，Ｚ開発試行の狙いもそれなりの広がりをもっていることを指摘しておきたい．

1.2.2.1　問題の普遍性

要求仕様書の品質問題は，当社だけが抱えている問題ではなく，業界全体の問題と考えられる．実は，信頼するに足る確かな文書を仲立ちとして開発を進めることが業界全体に強く求められているからである．さらに我が田に水を引けば，Ｚ開発試行もそうであった．

1.2.2.2　問題の分析

なぜ仕様言語なのか．要求仕様の品質問題の解決に，仕様言語がどのように，どの程度に有効なのか．これに答えるため，要求仕様書の品質問題を分析してみよう．記述言語の役割が析出してくるはずだ．初めに，当社が「協力会社」に提示する要求仕様書がどのように拙いのか，次の三点

・記述不足
・表現が曖昧
・依頼部署毎に形式がバラバラ

が指摘されている．

1.2.2.2.1　記　述　不　足

第１の，要求仕様書が「記述不足」であるとはどういうことであろうか．仕様書作成者の不注意による記述漏れといった類のことではあるまい．コード開発者は，その種の記述漏れの有無とは関係なく，それが要求仕様書の理解のさまたげにならないかぎり，与えられた仕様書を満足するコードを作ればよい．第一，記述漏れの有無はコード開発者には気づきようがない．記述漏れは，後で仕様書作成者が気づいた段階で，仕様変更として処理するしかない．それで仕様書作成者もコード開発者も不満はないはずだ．

「記述不足」で問題とすべきは，仕様書作成者が「当然理解してくれるはず」と思って記述したことが，コード開発者には「理解できない」という場合だろう．言わばコミュニケーション・ギャップである．解決すべきはこれである．

このギャップは，言うまでもなく，記述の対象となる世界について，仕様書作成者は豊富な知識を持っているが，コード開発者は貧弱な知識しか持ち合わせていないことからくる．この知識の差は，仕様作成者とコード開発者との役割の違いそのもので

あり，あって当然であろう．

問題は，対象世界に関する知識が，それも要求仕様書を理解するための前提となる知識が，記述されていないことにある．まさに「記述不足」なのだ．これが書いてないから，要求仕様書が自己充足的な文書とならないのである．解決策は，この暗黙の前提知識を記述すること．それしかない．

前提知識が，暗黙のまま，どこにも記述されていないために生ずる問題は，要求仕様書作成者とコード開発社とのコミュニケーション・ギャップにとどまらないのである．実は，要求仕様書作成者の間にもギャップがあり得る．要求仕様書作成者たちが抱いているその種の暗黙の知識の間にも当然不整合があるかも知れない．不整合があるのに，暗黙のまま記述されることがないので，お互いにそのことに気づかないだけかも知れないのだ．仕様書作成者たちの間ばかりではない．一人の仕様書作成者の暗黙知のなかにも不整合はあるかも知れない．あっても記述されていないので気づかないのだ．勿論それで済めば何も問題はないわけだが，あとでその不整合が顕在化して大問題となることも，さほど稀とは言えまい．現実はそんなに甘くはない，と言うことだろう．巨大ソフトウェアの開発中に遭遇する**多くの問題の淵源がこの暗黙の前提にある**．

対象世界に関する暗黙の前提を，暗黙のままにしておいてはいけない．どのような形であれ，それをきちんと記述しなければならない．そして，その「記述」を見さえすれば，対象世界のことは何も知らなくても，要求仕様書が理解できるようにしなければならない．つまり，対象世界の事柄がその「記述」の中にすっかり表現できている，ということである．そのような「記述」のことを，しばしば**対象世界のモデル**という．この対象世界のモデルを書き加えれば，要求仕様書は自己充足する．

対象世界のモデルを描くことの利点は計り知れない．第一に，要求仕様書作成者が抱いている暗黙の前提知識が公開されるので，多くの人がこれを客観的に批判し，吟味できるようになる．同時に，その「モデル」を現実と突き合わせることも可能となる．この批判，吟味，現実との突き合わせをとおして，要求仕様書が信頼するに足る文書となる．この利点がいかばかりか，言うまでもあるまい．

第二に，自己充足的な文書となるので，コード開発者とのコミュニケーションギャップも解消できる．だから，要求仕様書さえ渡せば，あとは黙っていてもコードが作ってもらえるようになる．文書だけで開発が進められるようになるわけである．

第三に，このコミュニケーション・ギャップの解消は，テストや保守も，要求仕様書とそれを実現したコードさえあれば，だれでも的確にできるようになることにも通じる．つまりは，開発の仕方を改革するだけでなく，保守の仕方を改革することにもつながるのだ．

1.2.2.2.2　表現が曖昧

第 2 の表現が曖昧と言うのも，よく指摘される問題点である．「記述不足」と言われることと重なる部分もありそうだ．「協力会社」に提示されたプログラム要求仕様書をいくつか見た限りでは，曖昧と言われる表現を構成しているものは，図や表，それに日常語による説明文である．「協力会社」に対してだけでなく，これが一般的な傾向と言ってよい．

まず，図，表，日常語による説明文には，それぞれ**表現が曖昧になる要因が内在している**ことを弁えておくべきだろう．

1） **図，表**：図や表をうまく使って，複雑なことがらを一目で分かるように整理してもらうと，嬉しくなる．そのような図・表ばかりならそもそも問題は起こらないわけだが，残念ながら現実はそうではない．**その見方を説明してもらわないと，理解できない**ような図や表が決して少なくないのである．口頭で説明するときのお手元資料ならそれでも差し支えないだろうが，要求仕様書はそれではいけない．

図や表が問題なのは，**その記述要素や配置に，書く人が場当たりに勝手な意味を付与することができる**ことである．曖昧さはここから生じる．もちろんその意味が自明なら問題は起こらないが，なかなかそうは行かない．例えば，データフローの矢線ひとつに要求仕様全体と等価な意味あいを付与することだってできるのだ．勿論これは極端だが，本質的にこれと同じようなデータフローが多いのも事実である．実をいうと，データフローが便利なのも矢線や丸に勝手な意味を込めることができる点にあるのだが，ただ用途がちがう．例えば，データフローを使って課題をトップダウンに整理するときは，この，矢線や丸に勝手な意味づけができる点を利用しているわけである．しかし，これは要求仕様記述には適していない．図や表には，要求仕様を記述するには無理があるのだ．

2） **日常語による説明文**：後述するように，**日常言語には自ずからその記述能力に限界がある**．と言うのも，日常言語はその名の通り日常生活の用を足すように自然発生的にできてきた言語だからである．日常生活の用を足す限りではまことに便利だが，例えば，記述に，日常必要とされる以上の正確さを求められても，応えきれない．要求仕様書には日常必要とされる以上の正確さが求められるので，日常言語だけで要求仕様を記述するには無理がある．また，**日常言語は，その運用能力を身につけるのが極めて難しい**，という事情がある．日常言語の使用を考えるときには，この点も弁えておいてほしい．

表現が曖昧となる理由を，**表現する対象**の面から考えてみよう．表現の対象としては（1）**データ**と（2）**データが充たすべき条件**について考えれば十分だろう．

1） **データ表現が曖昧となる理由**

① 図や表を使うときの問題点

・**データの種類**（あるいは**データの型**）を一つに限ってみても，その種類に属するデータの**具体例は無数**にあるのに，図や表では**その中の一つ**を示すだけである．

・**データの種類**は無数にあるが，そのすべて種類の描き方を決めておくことはできないから，**種類毎に ad　hoc な工夫が要求されるため，勝手な意味づけが避けられない**．その意味づけが自明なら問題ないが，なかなかそうは問屋が卸してくれないことは，日常体験に照らしてみれば，それこそ自明だろう．

② 日常語による説明の問題

・どう記述すればよいのか，**記述するための視点**が与えられていない．

・例えば，データの種類は無数にあるが，それらをどう表現するのか，記述するための視点も仕組みもない．

2）**データが充たすべき条件が曖昧となる理由**

①　図や表を使うときの問題点

・描き方が決まっていないので，**ad　hoc な工夫**が要求される．そのとき**勝手な意味づけ**をしてしまいがち．

②　日常語による説明の問題

・条件が複雑になると，説明文も複雑となり，その**構造を明示することが至難**．

要するに，曖昧さのない要求仕様を記述するためには，図，表，日常言語だけに依存する記述方法では本質的に間に合わない，と言うことである．だから，図，表，日常言語では明晰に記述できないことがらを，**明晰に記述するための記述法を補う**しかない．

1.2.2.2.3　形式がバラバラ

第 3 の「依頼部署毎に，形式がバラバラ」は，単に見かけの書式が統一されていない，と言うだけではないないだろう．要求仕様を記述する視点さえ同じなら，見かけの書式の違いは仕様理解の障害にはならないからだ．だから問題は，記述する視点がバラバラだ，と言うことである．これなら確かに理解の障害になる．この「記述する視点」の不統一もさまざまだが，基本的には

・What と How

・実世界用語と要求仕様世界用語の混同

この二つに対処すれば十分であろう．

第 1 の不統一の解消には，要求仕様である以上 **What** に統一すべきことは言うまでもない．

第 2 の不統一で問題とすべきは，要求仕様の話の中に実世界用語が混入する形のものだが，そうせざるを得ない原因は「記述不足」にある．前提となる知識が記述されていないため，実世界用語を借用せざるを得なくなるわけだ．つまり，この不統一は対象世界の「モデル」が書かれていないか，書いてあっても「抜け」があることに起因する．だから対象世界について「抜け」のない「モデル」記述し，「記述不足」が解消すれば，この不統一も解消する．

1.2.2.2.4　仕様言語による解決

以上から，先に提示した要求仕様書の品質問題を解決するためには以下を遂行すればよい．

①　対象世界の「モデル」を記述し，要求仕様書を自己充足的な文書とする．

②　どのような種類のデータであれ，どのように複雑な条件であれ，明晰に記述するための記述言語を用意する．

③　要求仕様の「記述の視点」を「What」とする．

④　対象世界について「抜け」のない「モデル」記述する．

そのためには，要求仕様書のための記述言語を用意することである．それは，どのような種類のデータであれ，どのように複雑な条件であれ，明晰に記述するための言語である．しかも，その「記述の視点」が「What」であり，対象世界の「モデル」が記述できるような言語である．当開発試行では，仕様言語のいくつかは Model based Approach を支援する仕様言語 Z を採用した．Z 言語を使って「抜け」のない要求仕

様書を記述すれば，要求仕様書の品質問題の解決に繋がるはずだ．

以上で，最初に掲げたわれわれの課題

この要求仕様書の品質問題の解決策として，仕様記述に仕様言語を使ったらどうか．その可否および可能性を「協力会社」と協力して探れ．

に連絡した．

ただし，仕様言語を使うだけでは問題は解決しないことを，心に留めておいてほしい．仕様言語はあくまで要求仕様を記述するための道具なので，その使い方に良し悪しの区別がある．そして良い使い方をしなければ，要求仕様書の品質問題は解決しない．例えば，「モデル」を書くにしても，ただ仕様言語を使って書くだけではすまない．「抜け」のない「モデル」を書かなければならないのである．だから「抜け」のない「モデル」を書くための技術，更には良い要求仕様書を記述するための技術，つまりは仕様言語の運用技術が必要なのだ．

さて，仕様言語の運用技術というのは，その性質上，実際の運用をとおして身につける類のものである．その習得には経験を積むのみではない．実際の運用に望むときの指針といった類のものがある．その指針を知ることによって，技術を身につけるまでの期間を短縮することができる．

1.2.3 仕様言語導入の秋

仕様言語として注目されているものに，LOTOS，仕様言語 Z，VDM 仕様言語，OBJ，PVS などがある（本特集号の「形式的方法概観」，「形式仕様のための読書案内」参照）．それぞれの仕様言語には開発環境や支援ツールが提供されている．編集系，型検査系，検証系，アニメーションやコード生成ツールなどがあるようだ．ところで，このようにさまざまな仕様言語があることに対して，「当社としてはどれか一つを選び，それにコミットしていくべきではないか」との意見があるかもしれない．だが，受諾開発のことを考えると，そうは行かないだろう．受諾開発では，発注者側がそちらの開発標準に定めている仕様言語を使うしかないからである．仕様言語を選ぶのは発注者であって，受諾する側ではない．当社としては，どのような仕様言語が要求されても応えられるようにしなけらばならない．もっとも，当社の社内標準としてなら，特定の言語を指定してもいいだろう．

1.2.3.1 仕様言語の必要性

ところで，なぜ仕様言語なのか．なぜわれわれが慣れ親しんでいる日常語ではいけないのか．先にも，簡単に触れたが，この点を確認しておきたい．

日常言語というのは，もともと日常生活の用を足すように自然発生的にできてきた言語である．だから，その記述能力にもその発生に由来する限界がある．つまり，日常言語は日常生活に便利にできているが，その反面，正確な記述，厳密な記述，それも日常生活で要求される程度を越えるような正確さ，厳密さを要求されても，これに応えられない．そのための洗練や工夫が凝らされていないからである．また，日常生活では遭遇しないものごとも的確に記述できなくて不思議はない．

例えば，物体の運動のことを考えてみよう．運動自体はまことに日常的なことがらである．人が走るのも運動だし，電車や自動車が走るのも，飛行機が飛ぶのもみな日々

見慣れた運動である．天体の運行にしても，これは自分で運転することも乗ることもできないので少々問題はあるが，少なくとも日々目にはしている．しかし，日々見慣れた運動について正確に語るとなると，日常語ではなかなか難しいのではないか．的確に，あるいは正確に語るためには微分方程式を介在させるしかないだろう．運動を的確に記述するためには解析学という，専用の記述言語が必要なのだ．

要求仕様でも同じことなのだ．ソフトウェアの設計図たり得るような要求仕様書を記述するには，日常生活で要求される以上の正確さ，厳密さが要求される．だから，これに応えるられるよう，特別の工夫を凝らした，専用の記述言語が必要となる．

日常言語の限界について，記述の対象（1）データ（2）データが充たすべき条件の観点から整理しておこう．この点は，先に，日常言語では「表現が曖昧」になることについて記したことがそっくり当てはまる．

1） データの記述の限界

① どう記述すればよいのか，記述するための視点が与えられていない．

② 例えば，データの種類は無数にあるが，それらをどう表現するのか，記述するための視点も仕組みもない．

2） データが充たすべき条件を記述する場合

① 条件が複雑になると，説明文も複雑となり，その構造を明示することが至難．

1.2.3.2 仕様言語の記述の仕組み

では，仕様言語にはどのような工夫が凝らされているのか，Z 言語の記述の仕組みの要点だけ列挙しよう．

1.2.3.2.1 データ型の記述

データ型というのはデータの種類のことである．仕様言語はどのような種類のデータであろうと記述できなけらばならない．そのため Z 言語では集合論の記述の枠組みを取り入れている．つまりは，Z 言語は一種の集合論なのだ．集合論は現代の存在論と言われることがあるくらいだから，その記述能力は折り紙付きである．その意味で，Z 言語は現在望みうる最強のデータ型記述装置と言ってよい．

1.2.3.2.2 データが充たすべき条件

Z 言語は述語論理を採用している．通常の述語論理の記述装置にスキーマ記法を加え，巨大な論理式が取り扱えるよう，工夫している．これによって，どのように複雑な条件であろうと，実際に記述することができる．

1.2.3.2.3 仕様の構造化

仕様言語の要件は，「データ型の記述」，「データが充たすべき条件」が明晰に記述できるだけでは未だ足りない．**文書の構造化の仕組み**が必須である．巨大ソフトウェアの仕様書となると，その記述の分量も並大抵ではない．日常生活で要求される文書量をはるかに越えている．従って，秩序だった，文書の構造化の仕組みがなければ，首尾一貫した仕様を書くことなど，とてもできるものではない．まして，それを読み解くとなると，これはもう何と言えばよいのか，途方に暮れる．

1.2.3.3 それでも日常言語は有用

確かに，日常言語は，日常生活で要求される程度を越える正確な記述，厳密な記述には不向きだが，仕様書に書かれるのは，そのような記述ばかりではない．日常言語

で記述できることが一杯あるのだ．例えば，仕様の概要や情報処理システムの目標など，日常言語で的確に記述できる筈だ．このようなところには，もちろん日常言語で書くべきであり，仕様言語の記述対象ではない．

それに日常語はわれわれの母語である．仕様記述だろうと何だろうと，大枠は日常語を使って進めるしかないし，それでよいのである．作業が進むにつれて，やがて話が細かくなる．細かな区別をつけなければ話が分かりにくくなる．そういう段階になると，日常語でも表現はできるがそれでは記述が長くなるところとか，日常語では表現しきれないところがきっと出てくる．そういうところの記述に仕様言語の助けを借りようと言うのである．日常語が適しているところは日常語を使い，日常語が不向きなところ，日常語が想定していない領域，そのようなところの表現に仕様言語という道具の助けを借りようというのだ．あくまで日常語が主要な道具で，仕様言語はそれを補完する．これが日常語と仕様言語の役割分担である．

1.2.3.4　合意形成のために

専用の仕様言語を使って要求仕様書を書くことに対して，決まって出てくる反論は

> 「仕様言語で書いた要求仕様書は情報処理システムの発注者や利用者に読めるのか．読めなければ，開発の受諾契約のための合意形成ができないではないか．」

と言うものである．

仕様言語で書いた要求仕様書を発注者や利用者が読むのは一般には無理だろう．それは丁度，ビルの建築主の前に設計図の束を積み上げて，「これがあなたの注文通りのビルです．そのことに合意して下さい」と言うようなものだからである．要求仕様書はソフトウェアの設計図だから，基本的には開発者が見るためのもので，専門家以外の人が読めなくて当然なのである．

それでは，請負契約のための合意形成ができないではないか．こう反論するかも知れない．この点でも，ビルの建築が参考になる．ビルの建築を請け負う場合，設計図だけで合意形成するのであろうか．勿論違うだろう．設計者は，合意形成のために，説明用の資料を用意する．たとえば見取り図がそうである．それだけではない，実物大のモデルルームを作ったりする．あるいはビル全体の模型を作ったりもするだろう．つまり，合意形成のためには，設計図の外に説明用の資料や模型がひつようなのだ．ソフトウェア開発の受託契約でも同じことだあろう．合意形成のためにはやはり説明用資料やプロトタイプが必要なのだ．

では，ソフトウェア開発の受託契約の場合，合意形成のための説明用資料とは具体的にどういうものを指すのか．例えば，機能の概要をデータフロー図に描いたもの，や，端末操作を体験するためのプロトタイプなどである．要求仕様書だけでは無理なのだ．ビルのように，目で見，手で触れるような具体物ですら，説明用ツールが必要なのだ．説明用ツールは，ソフトウェアのような抽象物ではなおさら必要なはずだ．

1.2.4　管　理　以　前

ところで，ソフトウェアの開発にまつわる諸問題を，専ら管理技術によってカバーしようとする傾向が強いようだ．だが，到底開発管理だけでカバーしきれるものではない．

管理以前に，開発方法そのものに問題があるからだ．繰り返すようだが，要求仕様書を記述するための公共の言語がないのである．要求仕様書は書かれはするが，建築で言う「設計図」の役割を果たしていない．極論すれば，ソフトウェアの開発は，ちょうど，きちんとした設計図を描かず，素描や設計メモだけでビルを建てようとしているようなものである．これでは，いくら管理で頑張ってもうまくいかない．「設計図」を描くことが先なのだ．

1.3 目　　的

Z 開発試行は

・何を検証するためにやるのか

・何を獲るためにやるのか

など，その目的を整理しておこう．目的は，あとの評価項目に連絡する．

1) 要求仕様書の品質問題解決のため

先に検討したように，仕様言語を上手に使うことによって，要求仕様書の品質問題が解決できそうである．われわれは仕様言語 Z を選んだが，研究室では使えても，実際の開発には，規模の違いなどの理由で歯が立たない，と言ったことはないのか．Z 開発試行の第 1 の目的は，この仕様言語 Z の記述能力に関する検証実験である．さて，先の検討結果に拠れば，これには以下を確かめればよい筈だ．

① 自己充足した要求仕様書となるよう，「対象世界のモデル」が十全に記述できるか

② 曖昧さのない，明晰な要求仕様書を書くために

・ⅰ どのようなデータでも記述できるか

・ⅱ 複雑な条件でも明晰に記述できるか

③ 巨大ソフトウェアの要求仕様が記述できるよう，強力な構造化の仕組みを備えているか

④ 要求仕様書だけから，コードが開発できるか

2) 仕様言語 Z を「当社の記述言語」とするために

Z 開発試行は，仕様言語 Z を要求仕様書のための「当社の記述言語」とすることにも貢献する．どのような貢献か．

① 要求仕様記述例の獲得

「当社の記述言語」とするには，多くの人に仕様言語 Z を理解してもらわなければならない．なんであれ言語を理解するには記述例を見るのが早道だ．Z 開発試行を実施すれば，必然，仕様言語 Z で記述した要求仕様書が産出される．そこで，この，要求仕様書の，記述例を見てもらうと，仕様言語 Z がどのようなものか，多くの人が速やかに理解できる．

② 魅力的な開発実績の獲得

仕様言語を「当社の記述言語」とするには，多くの人に理解してもらうだけでなく，その良さも分かってもらわなければならない．そのためには，魅力的な開発実績を示すことが有効である．魅力的な開発実績とは，仕様言語を使うことによって，「従来の開発法法と比べて，信頼性の高いコードが開発できた

とか，生産性が高まった」と言えるような開発事例のことである．Z 開発試行によって，そのような開発事例が得られるかも知れない．

③ **運用技術とそれに関する知見の獲得**：

その他にも，仕様言語を実際に使用する経験から，その運用技術に関する知見が得られるはずだ．

2. Z 開発試行とその進め方

Z 開発試行は，当社と「協力会社」が協同して実施した．

2.1 準　　備

1） 課題は，社内システムの機能拡張のための開発案件の中から選んだ．事務処理分野の課題である．

2） 協力会社の要員 2 名を，仕様言語 Z が読めるように教育した．しかし，実際に開発が始まる時点では，両人は別の作業に忙殺され参加できなかった．代わりに，ソフトウェアの開発には豊富な経験はあるが，Z 言語や VDM 仕様言語の予備知識が全くないメンバーが参加した．

2.2 基 本 方 針

2.2.1 三つの仕事

Z 開発試行とは，次の三つの仕事である．

・仕様言語 Z を使って当社がそのプログラム仕様書を作成する．

・「協力会社」側がその仕様書からコードを開発する．

・その開発経験に基づいて仕様言語 Z とそれによる開発について評価する．

を遂行することである．

2.2.2 試行の規模

諸般の事情から，今回は，プログラム 1 本の開発を Z 開発試行に充てることになった．したがって，1.2 節に掲げた，Z 開発試行の意味合い

・何を検証するのか

・何を獲得するのか

もそれだけ制限される．この意味で，この度の試行はあくまでその第 1 歩と位置づけしたい．今後，第 2，第 3 の試行を積み重ねることによって，Z 開発試行の目的を達成して行きたい．

2.2.3 危 険 分 散

このように分割して行うことには危険分散の意味がある．大規模な開発試行は危険が大きい．試行であるだけになおさらである．だから，Z 開発試行は，危険を分散する意味でも，小規模な試行を積み重ねる形で行うべきであろう．今回は，その第 1 歩である．

2.3 課題の選定

2.3.1 選 定 方 針

課題の選定は，社内システムの開発担当部署が行った．仕様言語 Z の都合で選ぶことはしなかった．当面する開発業務の中から勝手に選んだ．その「記述能力」や「使い勝手」を試すことにした．

2.3.2 課　　題

課題は，当社の営業支援システムのなかの一プログラム「契約書作成プログラム」を取り上げた．同部署で手がけるプログラムの代表的な類型の一つを選んだ．

① 言語：C 言語
② ライブラリ：XWindow，WND（Motif に皮を被せたもの）
③ 機器環境：U 6000（UNIX 搭載のワークステーション）

規模などについては実績値を示す．

① 規模：3344 ステップ
・制御部：2851 ステップ
・画面定義部：493 ステップ
② 開発期間：15 人日

2.3.3 選 定 理 由

選定理由とは，販売システム室がこれを選んだ理由だが，

1） これが販売システム室で取り扱うプログラムの代表的な類型である．
2） この類型の仕様には，次のような仕様記述上の懸案があり，この点が前から気になっていたこと
① Event Driven 型の仕様をどう書くか
② ウィンドウ処理プログラムの仕様をどう書くのか
・ウィンドウの生成・消滅をどう表現するか
・ウィンドウを介する入力やボタン操作をどう表現するのか

大きくは，この二つの理由からである．この第 2 の理由は，「この類型の仕様がきちんと書けるのか」という挑戦でもある．

2.4 作 業 手 順

最終的な作業手順は次の通りである．

① 現行様式の仕様書を作成する．
② Z 言語の仕様書を作成する．
③ Z 言語の仕様書からコードを開発する．
④ 検収テストする．
⑤ 評価する．

Z 開発試行のためには，「現行様式の仕様書」は必要ではない．いきなり「Z 言語の仕様書」を作成しようとしたが，うまく進まなかった．その打開策として，まず「現行様式の仕様書」を作成し，それから「Z 言語の仕様書」を作成することにした．

このように二つの仕様書を作ることになった経緯については，3 章の「開発過程で遭遇した問題とそれへの対応」を参照されたい．

実は，「現行様式の仕様書」を作ることになった理由はもう一つある．そこにも記したが，保守のために「現行様式の仕様書」が必要だ，との意見があり，これに応える意味もあった．

3. 開発過程で遭遇した問題とそれへの対応

3.1 仕様書作成過程

仕様書作成段階で気づいた，特筆すべき問題点は以下のとおりである．

3.1.1 「データベース」仕様記述の困難

3.1.1.1 「データベース」と「共有メモリ」

このプログラムは「データベース」にアクセスする．加えて，「共有メモリ」にもアクセスする．「共有メモリ」と言うのは「データベース」の一部をメモり上に展開したものである．「データベース」を一々読み書きしていたのでは即時処理要求に応えきれない．そのため，営業支援システムでは，必要に応じて「データベース」の一部をメモり上に展開している．それが「共有メモリ」で，言わば第 2 の「データベース」である．

「データベース」は巨大だが，「共有メモリ」の方はその一部なのだから小さいはず，と思いがちだが，そうとは言い切れない．確かに，ある 1 時点に展開されるのは「一部」だが，また別の時点では別の「一部」が展開される．必要に応じて必要な「一部」が展開される仕組みになっている．その意味で「データベース」より複雑な構造をもっている．

3.1.1.2 説明が発散

「データベース」と「共有メモリ」の仕様，と言っても，当面の開発に必要となるのは，その一部分の仕様だが，その説明がなかなか収束しなかった．収束しない理由は，当面の開発で必要なのは確かにその中の一部分にすぎないが，その中だけでは説明が完結しない，という事情にある．当面必要となる箇所を説明すると，その説明の中にそれと関連のある周辺部分がでてくる．すると今度はその周辺部分を説明しなければならない．周辺部分を説明すると，その説明の中にさらにその周辺部分がでてくる．すると今度は…，という具合に，「データベース」や「共有メモリ」全体の説明へと拡大していく．

3.1.1.3 自己充足した仕様書がない

そうなる大きな原因は「データベース」や「共有メモリ」に関する自己充足した仕様書がない，という事実である．説明を収斂させるには，「データベース」や「共有メモリ」の中から，当面の開発に必要な部分とその周辺を含み，しかも概念上ほどほどのまとまりのある部分を切り出す必要があるが，口頭でそれを行うのは容易でない．「ほどほどのまとまり」が曲者なのだ．説明に，完璧なまとまりを要求すると当然「データベース」や「共有メモリ」の全体へと発散してしまう．そこで，完璧なまとまりはあきらめて，「ほどほどのまとまりのある部分」を探すわけだが，自己充足した仕様書なしで，口頭でやるのは難しい．それで，口頭での説明は中断し，ひとまず「現行様式の仕様書」の形にまとめてもらうことにした．

3.1.1.4 「現行様式の仕様書」の作成理由その 1

2.4「作業手順」にあるとおり，

① 現行様式の仕様書を作成する．

② Z 言語の仕様書を作成する．

初めはその積もりではなかったが，二つの仕様書を作成することになった．「現行

様式の仕様書」と「Z 言語の仕様書」である．そうなった経緯は，前述したように，「データベース」や「共有メモリ」の説明を「ほどほどのまとまりのある部分」に絞るためである．

3.1.1.5 「現行様式の仕様書」の作成理由その 2

もっとも，これだけなら「現行様式の仕様書」である必要はない．ほどほどに筋さえ通っていれば形式はなんでもよい．「現行様式の仕様書」を作ることにしたのは，保守のことを配慮した意味もある． 仕様書が， この 1 本だけ Z 言語で書かれていて，あとは全部現行様式で書かれている，と言うのでは，保守が難しい．だから「現行様式の仕様書」も必要だと言うのである．と言うわけで，「現行様式の仕様書」を作成することになった．

その結果，「データベース」と「共有メモリ」の説明に関する限り，最小限度にとどめることができた．

3.1.2 「現行様式の仕様書」ではコードが作れない

3.1.2.1 「質問およびコメント」を作成

「現行様式の仕様書　第 1 版」は読んで分かる文書ではない．少なくとも，この仕様書だけでコードは作れない．その主な理由は「記述不足」（暗黙の知識を無原則に前提している）や「表現が曖昧」なことである．

その「記述不足」や「表現が曖昧」が具体的にどういうものなのか．それについては「質問およびコメント」に纏めたので，ここでは詳述しない．乞参照．

ただし，この文書は，一通り目を通した限りで分からないところや明らかに書き方の拙いところをリストアップし，社内システム開発担当部署のメンバー氏に質問するためのもので，分かり難い箇所すべてを尽くしているわけではない．

なお，「質問およびコメント」を反映して，「現行様式の仕様書　第 2 版」を作ってもらった．

3.1.2.2 記述不足への現行の対応策

ところで，今度書いてもらった仕様書が特に分かり難いわけではないようだ．当部署で作成する仕様書は, 人により差はあるにせよ同じような弱点をもっているという．このままではコードが作れないわけだが，これは次のようにして凌ごうとしている．

① コードの開発依頼先を固定する．

② 固定した開発依頼先に，こちらと暗黙の前提知識を共有する要員（キーマン）を育てる．

③ キーマンを通して開発を依頼する．コード作成者の疑問には，そのキーマンに対応してもらう．

これで，前提知識が「暗黙のまま」であることによる仕様書の分かり難さはある程度解消されるが，それも限度があるという．暗黙の前提知識なら，それである程度は解消できそうだが，「表現の曖昧さ」となるとキーマンにも答えようがないだろう．

要求仕様書の品質問題を解決するには，やはり，読んで分かる要求仕様書を，自己充足した要求仕様書を書かずにすますことはできない．

3.2 コード作成過程

担当者は情報処理の専門学部出身で，ソフトウェア開発の経験は豊富ではあるが，

Z 言語は全く初めて，当社の社内システム開発担当部署の仕事も初めてであった．当然，同部署の開発環境や共通ルーチンの類も不案内，加えて，X-Window や Motif も初めてという．これほどの不利な状況にもかかわらず，成果物の品質もその生産性も，きわめて上乗だった．

3.2.1 代替要員への支援

3.2.1.1 Z 言語に関する教育

担当者に対して Z 言語の基礎について 2 日ほど解説し，さらに 2 日ほど，「契約書作成支援プログラム」の Z 仕様を読むために必要な知識を補いつつ，同仕様書を読む練習をした．

3.2.1.2 現行様式の仕様書

「現行様式の仕様書」は日常語で記述してある．「現行様式の仕様書」は，あくまで概要を知るためのもので，「現行様式の仕様書」だけではコードは作れない．それには「記述不足」や「表現が曖昧」なところがあるからだ．

もし，「Z 言語の仕様書」と「現行様式の仕様書」からコードが開発できれば，少なくとも，両者を併せれば自己充足した文書であることが検証できる．明らかに，「Z 言語の仕様書」は「現行様式の仕様書」の内容を含んでいるので，もし両者を併せると自己充足するなら，実は「Z 言語の仕様書」だけで自己充足していることになる．

結果は，両仕様書だけから，きちんとコードができた．

3.2.1.3 雛形コードの利用

情報システムの開発環境に不案内である．どのような共通ルーチンがあるのか全く知らない．この点どう克服したか．

3.2.1.3.1 雛形利用の利点

実は，当部署の標準的なやり方が有効であった．コードの雛形を利用する方法である．つまり，すでに開発済みの類似したプログラムのソース・コードを渡して，それをテキスト編集系を使って改編する，というやり方である．

こうすれば，どのような共通ルーチンを使えばよいのかが一目で分かり，分厚いライブラリ集を抱え込んで，ページを繰り，前後参照する必要がない．

3.2.1.3.2 雛形利用の欠点

今回の雛形を利用するやり方は，残念ながら，良いことばかりではなかった．雛形として渡した，開発済みのコードには，共通ルーチンの引数の説明がなかったことだ．そのため，実引数の設定が不確かになりがちであった．

今回使った雛形でもう一つまずい点は，ステータス検査が甘かったことだ．このため，コードの規模見積もり（ステップ数）が大幅に狂ってしまった．

3.2.1.3.3 雛形コードの質の改善

現行では，開発済みのコードを雛形として使っている．そうではなく，専用の雛形のライブラリを用意すべきだろう．そのポイントは

・共通ルーチンやその引数に丁寧な説明を付けること

・決まり切ったステータスの異常チェックコードをキチンと組み込むこと

など．

3.3 成果物一覧

この度の Z 開発試行の出力文書は次のとおりである．

① プログラム仕様書（現行の記述様式）　第 1 版

② 契約書作成処理に関する質問およびコメント

③ プログラム仕様書「機器賃貸契約書発行画面 MIK 570」（現行の記述様式）第 2 版

④ プログラム仕様書（Z 言語による仕様）

⑤ プログラムコード（C 言語）

・制御部　　：2851 ステップ

・画面定義部：493 ステップ

4. 評　　価

この度の開発試行はプログラム 1 本の開発である．その限りでの評価・検証であることに留意してほしい．

4.1 評価項目の源泉

4.2 第 1 の源泉

その源泉は，言うまでもなく，1.3 節の Z 開発試行の「目的」にある．そのために，この度の開発試行を実施した．そこには大きな目的が三つあった．

1） **要求仕様書の品質問題解決のため：**

2） **仕様言語 Z を「当社の記述言語」とするために**

3） **運用技術とそれに関する知見の獲得：**

ただし，上述したような規模の制約から，例えば「対象世界のモデル」の記述などは，この度の試行には馴染まない．

4.3 第 2 の源泉

開発試行の評価の視点はもう一つある．課題選定理由の第 2 に掲げた，記述上の挑戦がそれである（2.3.3 項参照）．

では，それぞれについて吟味していこう．

4.4 第 1 の源泉の吟味

1.3 節「目的」の構成にしたがって見ていこう．

4.4.1 要求仕様書の品質問題解決のため

要求仕様書の品質問題解決のために，以下を吟味すればよい．

1） 自己充足した要求仕様書を書くために

① 「対象世界のモデル」が記述できるか

本来なら「対象世界のモデル」の記述は注目すべきところだが，この度の試行ではプログラムを 1 本開発するだけなので，これには頓着しない．ただ自己充足した要求仕様書が書ければよい．

開発結果の経験から，勿論，自己充足した要求仕様書が記述できたと，言える．更に，仕様書だけからコードが作成できたことからもそう言ってよい．

2） 曖昧さのない，明晰な要求仕様書を書くために

① どのようなデータでも記述できるか

ここで記述したのは，「データベース」と「共有メモリ」，それとウィンドウだが，**十全に記述できた．**集合論が現代の存在論と言われるのももっともである．

② 複雑な条件でも明晰に記述できるか

論理記号を用いて，明晰に記述**できる**．スキーマ記法もなかなか行き届いている．

3） 巨大ソフトウェアの要求仕様を書くために，

① 強力な構造化の仕組みを備えているか

十分期待できる．

4） 要求仕様書だけから，コードが開発できるか

できた．

4.4.2 仕様言語 Z を「当社の記述言語」とするために

1） **要求仕様記述例の獲得**

身近な課題について，記述例が獲られた．仕様言語普及のためになくてはならないものである．

2） **魅力的な開発実績の獲得**

獲得できた．コード開発の生産性，信頼性については後出．

4.4.3 運用技術とそれに関する知見の獲得

未検討

ここで，バグ件数とは検収時の検査で検出したバグの件数のこで，修正件数とは検収時に気づいた仕様変更の件数を指す．

4.5 生産性と信頼性

4.5.1 生　　産　　性

次に示す通り，生産性の高さが証明された．

1） コードの作成条件

① 言語：C 言語

② ライブラリ：XWindow，WND（Motif に皮を被せたもの）

③ 機器環境：U 6000（UNIX 搭載のワークステーション）

2） 規模と工数

① 規模：3344 ステップ

・制御部：2851 ステップ

・画面定義部：493 ステップ

② 開発工数：15 人日

4.5.2 信　　頼　　性

情報システム部・販売システム室では，平成 7 年度に，類似のアプリケーション「任意約定契約登録 2」（MIK 520 から MIK 532）をすでに開発している．その検収時の実績とこの度の開発（MIK 570）の検収時の実績を表 1 に対比して示す．

ここで，バグ件数とは検収時の検査で検出したバグの件数のこで，修正件数とは検収時に気づいた仕様変更の件数を指す．

表 1　検収時におけるバグ・修正依頼数一覧

任意約定契約登録２開発 (平成７年６月)

	バグ件数	修正件数	step 数	難易度
MIK520(処理選択)	24	18	8527	A
MIK521(H/W 明細指定)	20	5	4453	B
MIK522(S/W 明細指定)	14	5	4030	B
MIK523(受諾明細指定)	10	2	2375	C
MIK524(SE 明細指定)	10	2	2551	C
MIK525(その他明細指定)	9	2	2765	C
MIK526(新サコード指定)	9	8	3922	A
MIK527(新サ明細指定)	4	3	6197	B
MIK528(契約登録)	26	37	8097	A
MIK532(契約キャンセル)	1	8	3442	C
Z 開発試行 (平成８年４月)				
MIK570(賃貸契約書発行)	1	0	3344	B,C

4.6　第 2 の源泉の吟味

課題選定理由にある，記述上の問題についてはどうか.

1)　Event Driven 型の仕様をどう書くか

2)　ウィンドウ処理プログラムの仕様をどう書くのか

①　ウィンドウの生成・消滅をどう表現するか

②　ウィンドウを介する入力やボタン操作をどう表現するのか

すべて適切に表現できている. これについては，本稿付録に Z 言語による 「仕様書」からそれぞれの該当部分を抜粋するので，そちらを参照していただきたい.

4.7　結　　論

きわめて限定された規模の試行ではあるが，以下の目的について

- **要求仕様書の品質問題解決のため**
- **仕様言語 Z を「当社の記述言語」とするために**
- **運用技術とそれに関する知見の獲得**

その規模なりの検証や成果物が獲得できた．特に，仕様言語 Z の「記述能力」や「使い勝手」の検証では，それを使うことが，要求仕様書の品質問題解決のために有効である，との結論が獲られた.

この結論は，課題選定理由の記述上の疑問にも肯定的に答えられたことによって，さらに強化されよう.

5.　これからの進め方

この度の Z 開発試行では，幸い，仕様言語の使用を支持する評価が得られた．これに乗じて，その普及をどう進めるかについて私案を述べたい.

5.1　他の国外協力会社との Z 開発試行を実施

「協力会社」との間で実施したような Z 開発試行を，他の国外の協力会社と協同して実施したい．他の協力企業に先行して，国外の協力会社に発注する開発に形式仕様言語を適用する理由は以下の通りである.

・遠隔地であるため，高品質な仕様書の必要性が高い．
・母語を異にするので，Z 言語や VDM 仕様言語など，中立的な標準言語を使うことが望ましい．

5.2 形式仕様言語普及に関する提言

5.2.1 ソフトウェア開発の現状

本稿のはじめで，要求仕様書に判りにくいものがあり，どのようなコードを開発すべきなのか，コード開発者になかなか伝わらないことがある．そしてその原因が従来の仕様書の記述不足，表示のあいまいさ等にあると述べてきた．その理由は，現行のソフトウェア開発法には「設計図」がないことであり，筆者等はソフトウェア開発における「設計図」とは，仕様言語で記述した要求仕様書であると考える．

5.2.2 改善の目標

そこで，

1） 仕様言語を導入することによって，ソフトウェアの開発方法そのものを改革する．
2） この改革によって，直面している行き詰まりを打開し，ソフトウェアの開発を，品質と採算の両面で，魅力ある仕事に変革する．
3） 同時に，この仕事に専門性と創造性をとりもどし，ソフトウェア技術者であることに誇りをもてるようにする．

5.2.3 改善の方策

これを実現するため，クリーンルーム方式による，ソフトウェア開発を実施したい．ただし，当面は，その準備として，次の点に注力する．

・新しい開発方法とその支援ツールを整備し，調達する．
・ソフトウェアの受諾開発を行い，その優位性を立証する．
・代表的な情報処理系類型について，開発事例集を作成する．
・その優位性をバネに，仕様言語の社内への普及・教育を推進する．
・クリティカルシステムの開発や安全解析など，新しい需要を発掘する．
・形式的開発法に関する調査・研究を行う．
・支援ツールを調査，研究し，開発する．

ここでクリーンルーム方式と言っているのは，仕様言語を使用し，相互にレビューしあうことによって信頼性の高いソフトウェアを効率よく開発するための方法のことである．コードの開発には，必ずしも形式的な証明には頓着しないことが一つの要点である．

6. おわりに

Z 開発試行の成果物のひとつ，Z 言語で記述した仕様書から，ウィンドウ処理プログラムの仕様記述について，課題選定理由で挙げた問題点に該当する部分を抜粋する．ウィンドウや Video の表現， 打鍵入力やボタン操作の取り扱いに関する部分である．

ウィンドウ処理プログラム．これが今最もポピュラーなタイプのプログラムであろう．この代表的類型の仕様の記述例であること，加えて，それが実用に供されるプログラムの仕様記述例であること，この二つの点で，当記述例は広く興味をもっていた

だけるのではないか．

「これからの進め方」の中心は「クリーンルーム方式による開発をはじめよう」という提言である．その精神は，今日の科学技術文明を支えてきた 2 大動因，つまり「ロマンチックな好奇心」と「経済的インセンティブ」を，ソフトウェア開発の推進力にしようというもの．ソフトウェアが科学技術文明の所産であることを忘れてはなるまい．

以上の報告が，仕様言語への関心を広く呼び起こし，仕様言語そのものについて，あるいは，われわれの仕様記述例について，あるいは，仕様言語を公共の言語とすることについて，あるいは，その普及の進め方について，議論を喚起する契機となれば幸いである．

執筆者紹介　李　　春　端（Li Chunduan）
1959 年北京生．1982 年北京大学コンピュータ科学技術部卒業．1982 年～1991 年中国国際旅行総社のコンピュータセンター SE．1991 年～1998 年科優軟件開発有限公司課長．現在，北京東方陸創科技開発有限公司社長．

染　谷　　誠（Makoto Someya）
1942 年生．1966 年立教大学理学部数学科卒業．1969 年同大学院修士課程中退．（株）日本ユニバック総合研究所入社．その後日本ユニバック（株）（現日本ユニシス（株））に移籍．現在，情報技術部技術研究開発室．