形式的ソフトウェア開発

日本ユニシス株式会社　山崎　利治

1．形式的仕様記述の意味

形式的記述(Formal description)は仕様を書いたり、設計文書を作ったりするためにヒントが得られるので貴重だと思います。プログラム言語は覚えるのが大変で、自然言語で計算プロセスが書けるといいと云う話がありますが、それはかえってマイナスではないかと思います。と云うのは自然言語で計算手続きをちゃんと書くのはとっても難しいことです。ものすごく注意してもやっぱり分けわかんないねと云うようなことがでてきますから、むしろその計算過程を記述することに対して有効な概念を取り入れた簡潔な人工言語を使うことの方がはるかに易しくなります。数式を使うと楽な時に、数式の代わりに自然言語を使えと云うのは耐えられないわけです。

そういうことから、何も数式的とか形式的な言語を使うことが良いのではなく、形式的な言語で考えて自然語で話すことが大事なのです。何が根っこなのか、つまり何が大事だと云うことが分かるからです。仕様記述や設計記述をどう考えればうまく書けるかのヒントが、きちんと形式的開発法と云う立場で議論されているものの中に、見つけることが出来ると思います。

例えば、事務計算の現場で書いている多くの仕様や設計文書は必ずしもいいとはいえない状況だと思います。ほとんど仕様文書はなく、仕様と称しているものが設計文書です。この設計文書が、疑似コードといってＣＯＢＯＬ文を日本語にしたようなものを書いているのです。それならはじめからＣＯＢＯＬで書けばと思ってしまいます。仕様や設計文書の書き方をまだまだ検討し改善しなければなりません。

さて、形式的ということですが、よくできる人にとっては無意味です。問題なくしっかりソフトウェアを開発してしまいます。Knuthはそんな一人です。

ＴｅＸなんて云う、文書を作るシステムがありますが、これはKnuthが自分の本を書く為に作ったプログラムですが、最近はいろんな所でよく使われています。

この解説文書は疑似的なPascalコードと、そのコードの回りの一般文書からできています。そのPascalコードをKnuthの云うＷＥＢに通すと正式のパスカルコードになってコンパイルされてプログラムとして実行できるようになります。そしてそれをまた、開発文書として読む、つまり元のコードはドキュメントとしても読むと云ったことができるわけです。それはKnuthの文芸的プログラム、literateプログラミングといったりしてるものです。こういうふうにKnuthですと、なんか融通無碍に形式的とかなんとか云わずにうまくやってしまうわけです。しかし、我々というと問題があるのですが、我々はこんな風にはできない色々と課題を抱えて、例えば、耳にタコができる位聞いた図書館問題ですが、どう書いたら良いか、仕様が決まったとか出来たと云っても、どんなふうに書かれたかが問題になるわけです。

さっき云われていた、仕様が立派にできましたと云われた時に、どんなふうに出来ましたって、やっぱり聞かなければなりません。仕様は誤解の元にもなるし、書いたものに矛盾があると分かれば、それから作られたものはどんなプログラムでも正しいと云うことが云えます。そんなことがあるわけですから、我々は書き方を工夫しなくてはならない。そこで形式的なdescriptionがどんなものかと云う例をお見せして、渡辺さんからのご依頼をごまかそうと思います。

このFormal descriptionと云うのは欧州で結構はやっているらしくて、産業界の人を巻き込んでFormal development のための国際的な会議がずいぶんやられてます。そこで出てくる名前は、ＺとかＶＤＭとかＲＡＩＳＥです。まだありますけど、産業界が注目しているものとして、こういったものを良く目にしたり、聞いたりします。Zは Oxfordが育てたもので、産業界が注目して急速に普及してきました。

ＶＤＭは非常に伝統をもってまして、ＩＢＭがＰＬ／Ｉの意味記述をどうしたらいいかということで、Vienna Definition Language、ＶＤＬと云うのを作ったのが始まりです。ＶＤＬが気にいらないと云うので、ＶＤＭ、このＤはDevelopmentこのＭはMethodなのですが、ＶＤＬからＶＤＭに変えたわけです。こういうものがMethodと云っているからMethodらしくなっているのですが、仕様記述や設計記述のためにどういう言語を使えばいいかが、議論されて出来上がりつつあります。

また、最近ではこういったものを国際規格にしようとする動きがあります。そう云う面からも注目されています。

RAISE(Rigorous Approach to Industrial Software Engineering)はRigorous(厳格な)アプローチをソフトウェアインダスタリに適用しようとＶＤＭやＺをにらみながら、ＶＤＭやＺが、今持っていない考えを支援したいということで、いろんなアイデアを取り込んでいます。例えば、抽象データ型の代数的な仕様記述や、ＣＳＰの互いに協力しながらパラレルモードで動くプロセスなんかを組み入れたりしています。さて、サンドイッチが来る前にそのＺによる図書館問題について部分的にやってみましょう。

Ｚは、実はZermeloと云う哲学者、数学者がいて彼の集合論いわば公理的な集合論のことで、集合や関数という抽象的な概念によって、現実の課題の定義をやっていこうと云うものです。ちなみに、よく数学で、数学は集合論をもとに作られていると云われますが、現実の数学者の皆さんが使っている集合論はＺＦとかＺＦＣとか云っているものです。このＦとかＣとかは人や公理の名前で、特別な公理を入れているのですが、普通のアプリケーションの仕様を書く場合はＺＦＣなどは必要なくてＺだけで済むわけです。で、この辺の細かいことについては、集合論の本を見て頂きたいわけですけれど、そういう要する集合と関数、集合と関係を道具として仕様を書くと云うアイデアです。

2. Ｚによる図書館問題の記述例

さて、図書館問題ですが、よく数学などでは未定義の言葉を用意して、その意味はやがて分かるから、ちゃんと定義しないで使おうと云う癖があって、そのようにします。

2.1 基本型(basic type)、公理的記述(axiomatic description)

先ずCopyと云うのは物理的な本です。本と云うのはそれを抽象して情報があるものとすると同じ本が何冊もあることになります。例えば、千とか万とか、中には百万部も売れる本もあったりするわけです、その一冊の本をイメージする訳です。がたがたそれを定義するのも面倒ですから、やがて明らかになるという精神でほっておきます。で、Bookと云うのも考えます。これが何を意味するかと云うのは問題なんだけれど、例えばAuthor、Title、Publisher、発行年とかで抽象出来る一冊の本を意味します。で、今度は図書館のUserを未定義の集合だと思って導入し、Ｚの書き方はスキームといって、特別の括弧でくくって、特別の様式「スキーム」で何でも書いていきます。

[Copy、Book、User] ･･･ 基本型の定義

この図書館で、一人が借りられる最大冊数と云うのがありますから、例えばmaxloanを(公理的記述として)導入します(図1、(1))。いくつかのbuilt-inの集合を導入しておいた方が便利ですから、自然数をbuilt-inの集合(N)として持っているとしましょう。で、ある自然数maxloanなんていうのがあり、例えば２０冊、１０冊、５冊とか云うものとして考える。

maxloan : **N** ･････(1)

図 1. 公理的記述 (Axiomatic descriptions)

2.2 図書館スキーマ

その他に導入するものをこんな格好で書きます(図2)。

Lib

stock : Copy ⤖ Book ·····(2)
issued : Copy ⤖ User ·····(3)
shelved : **F** Copy ·····(4)
users : **F** User ·····(5)

---

shelved ∪ dom issued = dom stock ·····(6)
shelved ∩ dom issued = ϕ ·····(7)
ran issued ⊂ users ·····(8)
∀u: users ・#(issued ▷ full) ≦ maxloan ·····(9)

図2. 図書館スキーマ (library schema)

上がdeclarationで下がpredicateになります。上が宣言で、下がいわばそれを規定する述語になります。Libと云うのがスキームの名前になります。先程挙げたCopyとかBookがいわばタイプです。図書館全体の本をイメージすると、所蔵している(stock)本と云うのは、Bookへの写像だと思って下さい(2)。こういったものは最初から確定しているわけではなくて、時間と共に本が増えたり減ったりするわけですから、この写像は部分的なものpartial functionになるわけです。partial function は⤖のように表します。

貸出(issued)中と云うのも導入します。これは、CopyからUserへの写像になります(3)。

それから図書館の棚にある(shelved)これは、stockと云う登録された本のうち在庫中のものを指します。これは、Copyの一部です。

それから、図書館にはどうせ有限個の在庫しかないわけですからFを使ってshelved一棚にあるを表します(4)。Fは有限集合(Finite set)を表します。

さて、本を借りる人(user)を考え、これはUserの有限集合になります(5)。

こういうものを、図書館システムのデータベースと云うか状態として導入し、これらがどういう制約を持つかを下に書きます。例えばshelved、全てのものは棚にあるか利用者が持っている、つまり、棚にあって利用者も持っていると云う本はない。図書館が持っている本はどちらかの状態だと云うことを表したい。

そうしますと、shelvedと借りだしている本Copyと考えますとissuedへの写像なのですがこのドメイン(定義域)が図書館の本全てを表します(6)。これは図書館に登録した本は借りだされているか、棚にあるどちらかだと云うことを表しています。

それから、shelvedと貸出中issuedのドメインの共通部分はありません(7)。空です。

issuedのレンジ、つまり借りている人range(値域)はusersです(8)。ranはZが持っている予約語(reserved word)です。userと云うのは図書館に登録されている利用可能な人と云う意味です。

それから、貸出数に制限があります(9)。そうすると、利用者の全ての人が、全ての人と云う時、述語論理で使われる∀といった記号はそのまま使います。丸印は、一つの区切りを表し、#記号は、括弧内の集合の濃度即ち要素の数を表します。 ▷ は範囲の制限(range restriction)です。これは、issuedは関数で、usersでレンジの方を制限する、つまりあるユーザが借りだしている本の数をmaxloanで制限すると云うことを表しています。

こう云う具合に書いていきますと、自然語で書かれていましたことが、はっきり分かるようになります。

2.3 スキーマIssue

次に貸出と云うことがどういうことになるかを考えます(図3)。貸出をIssueと書くとしますと、Zでは色々なcon-

ventionがありまして、ΔLib(10)と書きますと、先程考えたLibが生きていまして、Issueと云うオペレーションの結果出てきた、変化が表されます。

Issue

△Lib	·····(10)
c? : Copy	·····(11)
u? : User	·····(12)
c? ∈ shelved	·····(13)
u? ∈ users	·····(14)
#(issued ▷ {u?}) < maxloan	·····(15)
issued' = issued ⊕ {c? ↦ u?}	·····(16)

図3. 貸出スキーマ (Issue schema)

プライムの記号のついたissued'は、それは、Issueした後の状態を示します。例えば、ここで、入力データを?で表します。Issueがどういうふうに規定できるかと云いますと、c?は、在庫している(shelved)本であり(11、13)、u?は登録されたユーザ(users)であると云うことです(12、14)。

それから、貸出制限を守っている(15)。その時に、issued'は、(16)のように書きます。issuedは写像です。c?に対してu?を対応させると云う要素をオーバライドする、オーバライドと云うのはoverride(重合)で、二つの関数f,g: X↛Yに対しf ⊕gと書き、関数でその定義域はfのそれとgのそれの合併であり、fのそれの上ではgに一致し、その他の上ではfに一致するもののことです。c?と云うのは棚にありますから貸し出されてはいない。Issueは今まで棚にあったものを棚から持ちだし、棚からなくなると云うことが起こる。このc?と云う本はu?と云うユーザに貸し出されると云うことをこういうふうに書くことによって示すことが出来ます。

ここで、棚にあったと云うshelvedは、棚からなくなるわけですから変化する。次のように書くことができます。

$$shelved' = shelved \setminus |c?|$$

しかし、これは示さなくても最初の2つの条件(6)、(7)から自動的に推論できるわけです。申し忘れているかもしれませんがこのプライムはIssueと云うオペレーションが終わったあとの状態を示します。 Issueをした後の棚の本は、なくなるわけですがそれは書き込まなくても、Libの二つの条件から導き出せると云うことが分かるわけです。

3. 最後に

あんまりこういうことをがたがた云ってても、仕方がないかもしれませんが一度例えば密かに、図書館問題を定義し直しましてこんな風に書いてみる。そして他の仲間やお客様と話ができるように書き直すといいかも分かりません。

何が大事なことなのか、何は云わなくても、知っていることから普通に推論出来ることなのかが分かりますから、なんでもかんでもごちゃごちゃ書かなくても済むともいえるわけです。

それで、こういった格好でどんどん書いていきまして、これからどう実現するかとがあるわけですが、実現したものが仕様を満たすんだと云うことを、いちいちチェックしていけば、まあ間違いのない実現ができるであろう、であろうと云うのはできる可能性があるだろうと云うぐらいの意味で本当にできるとは云えないわけです。どうせ人間は間違うものですから。

形式的開発法が現時点での、仕様技術とかソフトウェア開発の状況をより良くするヒントになるのではないかと思える。私等はそう思っているわけなのですけれども、こういった書き方をするのが全てではなく、こういった書き方もやればできるよと云う力を密かに持っていると云うことが、大事なんじゃないかと思っています。

形式的なソフトウェア開発法と云うことで、出発点になる仕様記述の一部と云うか雰囲気をお伝え出来てれば良いのですが。

(記録者 MHIエアロスペースシステムズ(株) 伊藤 昌夫)

記録者より

最初に、本記録の文責は全て記録者にあることをお断りしておきます。
号番は、記録者の方で振りました。章ごとに関連しそうな文献と感想を残しておきたいと思います。

1章について

(WEBについて)
WEBについては、最近様々なところで最近良く目にする。一番読みやすいのは「クヌース先生のプログラム論(有澤誠編、共立出版社)」で、TeXの話からWEBの話迄が含まれている。WEBは山崎氏の話にも有ったようにWEBコードからweaveとtangleにより、プログラムとしてのPascalコードと設計文書用のTeXプログラムを生成する。ただ個人的には、WEBコード自体は読みやすいとは思わない。
Knuth自身はTeXの開発用として、プロトタイプを作ったが(これはたいていTeXと同時に配布されている)、必ずしもそれだけではない。様々な所で同様のWEBシステムが作られている。例えば、"FWEB:A Literate Programming System for Fortran 8x(SIGPLAN vol.25,No.1)"などがその例である。
Literate programmingに言及した最近の記事としては他に、D.Cordes and M.Brown,"The Literate-Programming Paradigm (Computer,Vol.24 No.6,pp52-61)がある。この論文では、literateプログラムの良い点と悪い点について言及した後、改良のために、構造的な内容の目次、グラフィカルユーザインターフェイス、デバッキングツール、索引を追加する事を提案している。

(Z、VDM、RASIEについて)
Zによらず Formal specificationについて解説した日本語の文献は殆ど見たことがない。しかし、日本語以外の文献はかなり豊富で、Zの表記法について解説したものとしては、"The Z notation second edition(Prentice Hall, 1992)"がある。VDMについては、Systematic Software Development using VDM second edition(Prentice Hall,1990)"がある。ともに有名なHoare seriesであり、読みやすい。ただ読みやすいのとすぐ使えるのは別だが。

RAISEについては、入手容易な文献が少なく、纏まったものとして私が持っているのは昨年のSIGSOFTのコンファレンス"Software for Critical Systems"のプロシーディング中の論文位である(B.Dandanell "Rigorous Development Using RAISE",pp29-43)。山崎氏からは、次の文献を教えて頂いた。The RAISE Language Group,"The Raise Specification Language",Prentice-Hall,1992.

(ZFCについて)
ZFCのZを表すE.Zermeloは、1904年に有名な整列可能定理の証明を発表している。この中に含まれる概念に選択公理(AC、Axiom of Choice)と呼ばれるものがある。Fは置換公理等でZermeroの議論を補完したA.Frankelの頭文字であり、この公理的集合論を一般にZF集合論と呼ばれる。P.Cohenによって選択公理がZF集合論から独立であることが証明され、このZF+ACの集合論はZFCとも呼ばれる。
集合論に関する本は色々あるが、Cohen自身が連続体仮説を含む集合論を非専門家向けに解説した、「連続体仮説」(近藤他訳、東京出版)が復刊されており、一読に値する。

集合論はG.Cantorに導入されてから約100年の歴史を持つが、情報科学の歴史と比べ長いのか短いのか。

(形式的手法全般について)
Formal Methodについては、1990年9月号のIEEE computer/software/software transactionに特集がある。
この中の"Seven Myths of Formal Methods"では、次の7つが形式的手法の神話として挙げられている。
(1) ソフトウェアが完全であることを保証している。
(2) プログラム証明のみを行う。
(3) 安全性を要求されるシステムに対してのみ役立つ。

(4) 非常に高度に訓練された数学者を必要とする。
(5) 開発コストを増加させる。
(6) ユーザに受け入れられない。
(7) 実際の大規模なソフトウェア構築には使用されない。
もちろん、これらの神話は誤りであると云うのが論旨になっている。

(自然語仕様とプログラム言語)

自然語でかかれた仕様の問題点と関連付けた記事としては、"On Formalism in Specification"(IEEE software Jan.1985,pp.6-26)があり、日本語訳もある(「IEEEソフトウェア'88」、pp139-154)。

著者B.Meyerは、言語Eiffelの設計者であり、Eiffel中にはAssersionを初めとする幾つかの正当性保証のための機能が組み込まれており、個人的にはそういった点でEiffelを気にいっている。Eiffelの立場を、著書("Programming by contract",p202)から引用する。

「…Eiffelは、一階述語論理を用いることで、一人前の形式的仕様記述言語に拡張できるであろう。しかし、Eiffelは、その立場として、実際的なソフトウェア開発に用いられる事を意図したプログラム言語であり、表明機構を言語に組み込む場合、(言語)設計上のゴールの違いにより、技術的なトレードオフがなされ、異なった言語になる。一方は、(高い)信頼性を要求されるソフトウェア開発を支援することを目標とするものであり、他方は、効率の良いコードを生成し、言語そのものの構造を総体的に単純化することを狙っている。…」

2章について

以下に shelved' の導出を簡単に示す。

issued' = issued + { c? → u?}	----(a)	[(16)より]
shelved' ∪ dom issued' = dom stock		[(6)より]
shelved' ∪ dom (issued ⊕ { c? → u? }) = dom stock		[(a)より]
shelved' ∪ (dom issued ∪ dom { c? → u?}) = dom stock		[⊕ に関するルールより]
shelved' ∪ dom issued ∪ {c?} = dom stock	----(b)	

一方、(7)も同様にして、以下のように記述出来る。

shelved' ∩ dom issued' = ϕ
shelved' ∩ dom (issued ⊕ { c? → u? }) = ϕ
shelved' ∩ (dom issued ∪ { c? }) = ϕ
(shelved' ∩ dom issued) ∪ (shelved' ∩ {c?}) = ϕ

従って、次が成り立つ。

shelved' ∩ dom issued = ϕ ----(c)
shelved' ∩ {c?} = ϕ ----(d)

また、(7)と(13)より明らかに次式が成り立つ。

{c?} ∩ dom issued = ϕ ----(e)

再度、(b)について考える。(6)より、

shelved' ∪ dom issued ∪ {c?} = shelved ∪ dom issued
shelved' ∪ { c? } = shelved [(c)、(d)、(e)、(7)より]
shelved' = shelved \ { c?} [(d) より]

従って、shelved' に関する記述が不要であることが分かる。

全般について

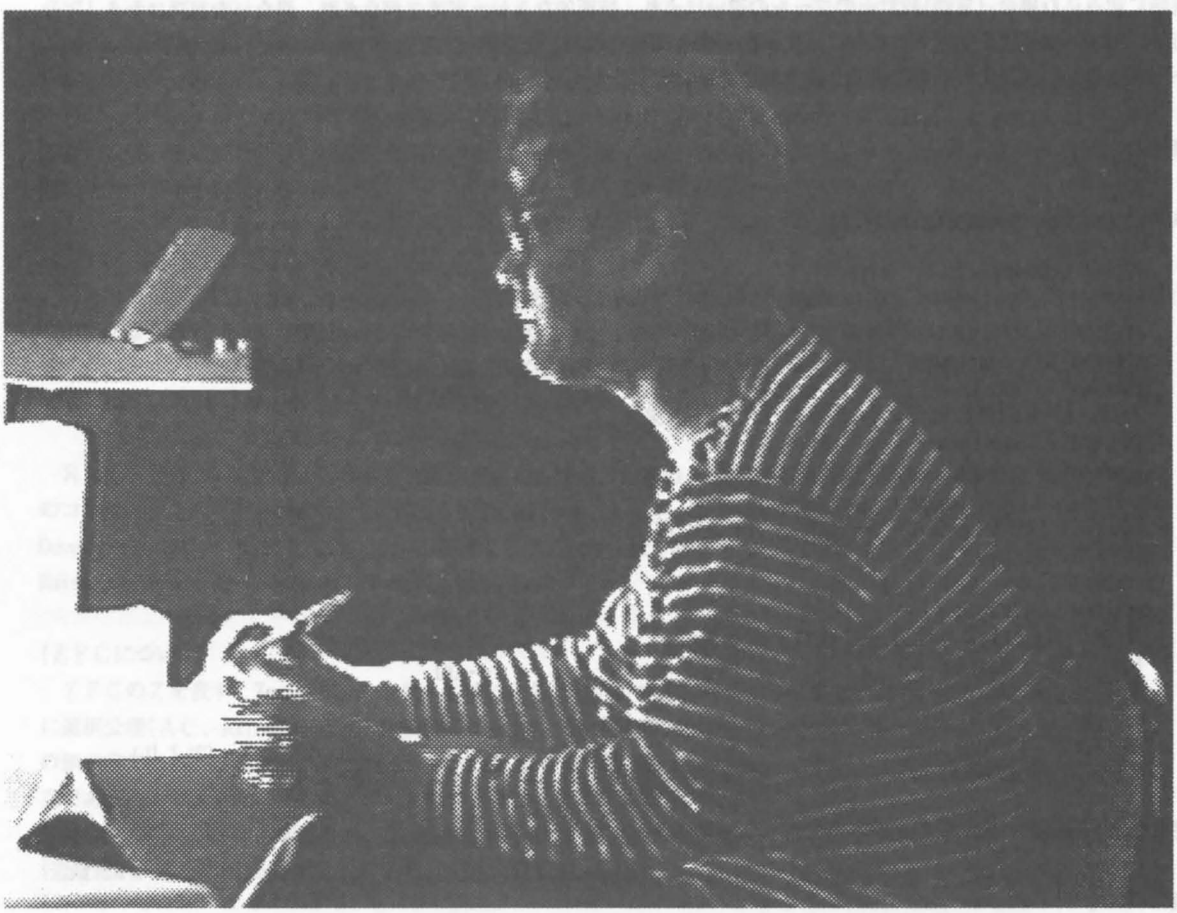
形式を主題とする開発法は、一般に内に向きやすい。つまり、その表現の強力さが、正しさや誤りを明確に指摘するため、気軽に使ってみようという気持ちを萎えさせてしまう。

しかし、山崎氏の話にもあったように、その良い部分を前向きに取り込むことにより、我々ソフトウェア技術者の仕事が楽になる可能性は高い。

先のEiffelの例で示したのも、その可能性を実現した一つの例であると思う。つまり、安全性と効率のはざまで、何もしないのでなく、その中間にも実は解があり、総合的に見た場合今迄よりもより良いソフトウェアが作れると云うことであると思う。

ところで、集合はある意味で、情報工学とは切っても切れない要素であるとも云える。外延的、或いは内包的集合の定義は、抽象データ型に通じるし、より狭い分野ではＡＩに用いられる述語論理等の論理の世界は公理的集合論を基盤としているのは明らかである。

そういった意味で、今回短い時間で分かりやすく、集合と情報工学の連関についてお話し頂いた山崎氏に深く感謝します。



Lecturing Yamazaki

付　録

- 参加者名簿 178
- プロフィール/ポジションペーパ/W.S. 終了後の感想 179
- オブジェクト指向設計 – 責任駆動のアプローチ – 241
- 読書案内 247
- グループ 2 作成の模造紙 249
- その他資料 255

  開催案内

  岡崎マップ

編集後記に代えて – 最近の SIGENV の活動について – 257

第10回夏のプログラミング･ワークショップ
@ホテル　レク･ワールド岡崎

	氏　名	会 社 名	その他
P01	渡邉雄一	アスキー	実行委員長G5
P02	塩谷和範	ＳＲＡ	プログラム委員長(G3,G4)
P03	伊藤昌夫	ＭＨＩエアロスペースシステムズ	G2
P04	岡本隆一	さくらケーシーエス	G1
P05	後藤浩	シスプラン	G4
P06	桜井修	池上通信機	G3
P07	田中慎一郎	ＳＲＡ	G2
P08	福良博史	ジェーエムエーシステムズ	G4
P09	豊田英利	セス	G3
P10	山崎利治	日本ユニシス	オブザーバ(G5)
001	古閉幸一	ヒラタソフトウェアテクノロジー	G4
002	東宮祐一	ネクストファウンデーション	G5
003	岡田安彦	オリンパス光学工業	G4
004	渡辺清文	オリンパス光学工業	G3
005	井上忠則	シスプラン	G3
006	酒井雅裕	ハイテック	G3
007	冨岡雅巳	ジェーエムエーシステムズ	G5
008	直江魅音子	ネクスト ファウンデーション	G2
009	江谷典子	富士ゼロックス情報システム	G5
010	森下葉子	ＳＲＡ	G4
011	岩城弘二	富士ゼロックス情報システム	G5
012	高橋友重	大和ソフトウェアリサーチ	G1
013	伊藤徹	アドバンスト･ソフトウェア･インスティチュート	G3
014	高巣英利	ＳＲＡ	G2
015	佐藤公	日本電気ソフトウェア	G1
016	木崎俊広	日本電信電話	G2
017	八百板司芝	ＳＲＡ	G1
019	工藤聡	岩手電子計算センター	G4
020	西岡啓和	さくらケーシーエス	G5
021	舟引秀光	さくらケーシーエス	G4
023	門田久好	新日鉄情報通信システム	G4
024	藤巻昇	東芝	G2
025	吉田恵	さくらケーシーエス	G3
026	寺井文康	ＭＨＩエアロスペースシステムズ	G2
027	沖さとみ	ＭＨＩエアロスペースシステムズ	G1
029	山口裕史	高岳製作所	G1
030	水谷公一	高岳製作所	G5

プロフィール/ポジションペーパ/W.S.終了後の感想

本章では, 参加者全員の

I. **プロフィール**

ポジションステートメントの 1. ～ 3.

II. **ポジションペーパ**

ポジションステートメントの 4.

III. **W.S.終了後の感想**

会場でのアンケート (メンバのみ)

を掲載する。掲載した順序は, 所属したグループ毎に, 報告書に名前が記載されて順である。

I.**プロフィール** と II.**ポジションペーパ**は, 申し込み時に**ポジションステートメント** という形で提出してもらったものである。ポジションステートメント に設けた質問項目は,

1. 現在までの主な仕事の内容について
2. 現在利用している開発環境について
3. これまで現場でのシステム設計に採り入れてきた技術・技法などについて
4. 以下のいずれかを選んで (A4 用紙に 1～2 枚:2000～4000 文字) に述べて下さい。いずれの場合も, 「何が」, 「どのように」, 「どうして」を可能な範囲で具体的に述べるように配慮して下さい。

特に b) の場合はその「改善の提案」も行なうようにして下さい。

a) 今までに経験した開発作業で, 設計に感じた「ありがたみ」について

b) 今までに経験した開発作業で, 設計に感じた「うらめしさ」について

の 4 つである。

4. の**ポジションペーパ**の部分は, こちらから示したものとは独自にタイトルを設定してきた人についてはそれを尊重した。また, その他の項目についても 個々の設問に対する回答記載は自由形式とし, そこでの表現形式を尊重したつもりである。

また, III.**W.S.終了後の感想**は, ワークショップの最終日 (9/5) に会場でアンケートとして記載/提出してもらった。用意した設問は

Q.1 参加する前にワークショップに期待していたことは何ですか?

Q.2 それは満たされましたか?

満たされたものについてはどのように満たされたかを, また満たされなかったものについてはその原因と感想をコメントして下さい。

Q.3 ワークショップに参加して, ソフトウェアの設計に関して何か意識が変わりましたか?

変わった場合は, その内容をできるだけ具体的に御紹介下さい。

Q.4 ワークショップに参加して, 未知の他人と短時間にあるレベルの成果を要求されるという経験をされました。これをもとに以下の点について具体的な意見を聞かせて下さい。

1) 他人とのコミュニケーションをとること

2) 短時間で問題を理解して, グループとして作業を進めること

Q.5 ワークショップでの経験は, 自分の仕事に対してどのような位置付けになったと思いますか?

□ 自分の仕事の進め方に自信が持てた。

□ 今後の仕事の進め方に対して, 新しい視点が持てるようになった。

□ ワークショップの議論と現実とはやはり違うと感じられた。

□ その他 : (具体的な内容を以下に記述して下さい)

Q.6 全体討論の内容に関して

Q.7 その他, ワークショップに実際に参加して良かったことはありましたか? 参加前の期待とは違う成果などがありましたら御紹介下さい。

Q.8 反対に残念なところ, 悪かったところはどんなところでしょうか? 進行上の問題も含めてコメントして下さい。

Q.9 今後の W.S. に取り上げて欲しいテーマ, 全体的な感想等があれば, お聞かせ下さい。

の 9 つである。 なお, 誌面の都合上, 個々の設問は省略した形で掲載してある。

アンケートについては, 特に集計することは差し控えた。全体で 30 枚にも満たないので, 安易に集計した結果を眺めて満足するのではなく個別に回答内容を見て その行間に秘められた思い[†1]を, 汲み取ることの方が重要と考えるからである[†2]。ただ, PC がざっと目を通した限りでは, 概ね[†3]好評だったみている。

---

†1 その本音を!

†2 忙しいので, 集計なんてことまで手がまわらないのも事実であるが 8-)

†3 参加者の御世辞/気遣いの部分を差し引いても

グループ 1

29 山口 裕史 (やまぐち やすし)
性別：男　年齢：29　血液型：A
会社：(株) 高岳製作所
所属：開発部システム技術開発課

I. プロフィール

(1) 現在までの主な仕事の内容について

- 68000 マイコンソフト開発
- UNIX 下プログラム開発 2.UNIX 下プログラム開発
- プログラム設計
- システム管理

(2) 現在利用している開発環境について

- UNIX(SUN) + X 端末
- Open Look & Motif

(3) これまでシステム設計に採り入れてきた技法など

- 構造化設計
- PAD
- HIPO
- 構造化プログラミング

II. ポジションペーパ

(4-a) 設計に感じた「ありがたみ」について

私も早いものでこの業界に入って 4 年目になります. 入社当初 2 年間は簡単なマイコンのソフトの開発を行ないました. ここでは, 基本的な設計から詳細設計, プログラミング, テストと一通りのことをやり, 一つの物を作る上では一連の作業, 全てができることができとてもよかったと思っています. また, マイコンということで, コンピュータの細かいレベルでの中味の動作とかが確認できたということで, とても勉強になったと思っています. その後, UNIX というものが脚光をあびて, それに伴う新しい部署ができ, そこに私が配属され, 現在にいたっています. この部署では UNIX をベースとしたソフトの設計・開発, 社内のソフトを開発する上での技術的な標準, 社内のネットワークの設計・構築などいろいろなことをやっています. そのため, ある仕事を行なう上でもしっかりと計画を立てて設計を行ない, テストをしっかりやっておかないとトラブルの原因になります. (実際には, 私の部署もトラブルは多いのですが) そこで私の部署では, 開発の作業をスムーズに行なうことにより, 少しでもトラブルを少なくしようと考えました. この項目が開発作業を行なう上での「ありがたみ」になったような気がします.

その項目については以下の通りです.

(1) 1 人 1 台に X 端末又は, EWS 等を設置し X ウィンド環境で作業ができる.

(2) 設計書等は文章管理ツールにより管理を行なう.

まず (1) の項目については, この開発環境を実現したことにより, 以下の利点がでてきました.

1. 端末が 1 人 1 台なので, 人目をきにしないで自由に作業できる. そのためテストしたいことがあれば, すぐにプログラムを作成してテストができる.
2. X ウィンド環境なので複数のウィンドが開くことができマルチに作業ができる. (汎用機を使用していた時は, 複数の端末を使用しての作業でとても不便でした.)
3. X ウィンドの各ツールが使用できるので便利です. (たとえば, 打合せや会議等のスケジュールを決める時等には, カレンダーマネージャーのツールを使用して, 各ユーザーのスケジュールを参照することにより, 打合せや会議等のスケジュールが決められます. この他にもいろいろ便利なツールがそろっています)

以上のことが (1) でいう開発環境を実現したことにより, 設計に感じた「ありがたみ」になるのでないかと思います.

次に (2) の項目については, この文章管理ツールを実現したことにより, 以下の利点がでてきました.

1. 各仕様書のドキュメントのフォーマットが標準化でき見やすくなった.
2. 各文章を共有できるので, 個人単位で文章を作成していたのがグループ単位でのドキュメントの作成が可能になった.
3. 文章の管理が整理され一括管理されているので, 恰も 1 つの整理棚に文章が項目単位にきれいに整理されているようなイメージです. そのため, 文章の読び出しもメニュー形式で簡単に検索できます. また, 紙に印刷することも少なくなり, 紙の節約になり一石二鳥というわけです.

以上のことが (2) でいう文章管理ツールを使用することにより, 設計に感じた「ありがたみ」になるのではないかと思います.

最後に, 前に述べた 2 つの項目の「ありがたみ」については, これからもどんどんと発展し, よくなっていくと思いますが, 特にこれからは, いろいろな開発作業をやる上でのツールが数多く出てくると思います. そのため, 無差別にいろいろなツールを購入するのではなく, そのツールが自分たちの開発作業をやる上で適切かどうかという判断力をみにつけることが, これからの私の課題だと考えています. この

力が 私についてくれば, 設計で感じた「ありがたみ」というものは, より増えていくのではないでしょうか.

III. W.S. 終了後の感想

(Q1) **期待していたことは?**

構造化分析設計の方法論の取得.

(Q2) **期待は満たされたか?**

日程が短いこともあり, 詳細部分が明確にされていないことがありもっと調査したい.

(Q3) **設計に意識の変化はあった?**

方法論に基づいてシステムの分析を行なっていけば, もれが少なくということが再認識された.

(Q4-1) **他人とのコミュニケーションについて**

意外に, テーマがはっきりしていれば, 他人と意見討論をやっていれば, レベルも そろってきて, そんなに異和感 なくコミュニケーションが取れた.

(Q4-2) **短時間での, 問題の理解と作業について**

色々な意見がでるのでまとめ上げるのに時間がかかる. 色々な方向から課題に対して検討できるので, もれが少なくなるのでは?

(Q5) **経験は, 自分の仕事に対してどのような位置づけ?**

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) **全体討論の内容に関して**

JSD 法が実際の業務に採用できるというのは疑問.

(Q7) **その他, W.S. に参加して良かったこと**

色々の技法が狭くではあるが知識のひとつとして, 少しではあるが取得できたと思う.

(Q8) **W.S. の悪かったところ**

(1) もっと余裕をもって作業をやりたかった.

(2) 事前にもっと勉強, 準備が必要だった.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

(記述なし)

17 **八百板 司芝** (やおいた もりしげ)

性別：男 年齢：28 血液型：A

会社：(株)SRA

所属：システム開発第三部

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 証券バックオフィスのシステムの開発・メインテナンス
- 銀行フロントオフィスのシステムの開発

(2) **現在利用している開発環境について**

- 現在は開発に携わっていない

(3) **これまでシステム設計に採り入れてきた技法など**

- 構造化設計らしきもの

II. ポジションペーパ

(4-a) **設計に感じた「ありがたみ」について**

今思えば, 'あの時' ほど設計のありがたみを感じたことはなかったような気がする。 'あの時' というのは, ワークステーションで初めて開発を行なった時である。会社に入ってから大きくわけて二つのプロジェクトを経験した。一つは証券会社で汎用機を使った PL/I での開発, もう一つは銀行でワークステーションを使った C での開発である。どちらも金融系のシステムであったが, 設計に対する考え方が全く変わった。金融系と言うとユーザの業務知識の習得が大切になる。汎用機をやっていた時は特にそれが顕著で, 業務知識さえあれば仕事ができた。設計についても, 何が必要でどう書けば良いかということはすべて決められていた。分からないことを聞ける人もたくさんいた。そのため, 設計で困るということはほとんどなかった。行なっていた手法は構造化設計に似てはいるものの, DFD や状態遷移図, ER 図はそれらしいものしか書かず, 多くを勘と経験に頼るもので, とても設計技法などと呼べるものではなかった。だが設計技法はその程度分かっていればよく, 業務知識こそ必要なものなのだと思っていた。

ワークステーションになっても金融系なんだし同じだと思っていたのだが大きな間違いであった。いちばん困ったのが, ワークステーションで開発する上でなにが汎用機と違うのかを分かる人がいなかったことである。汎用機で開発していた時は当たり前だと思っていたことが当たり前ではないと気づくことはしょっちゅうだった。例えば, DB の容量やパフォーマンスなどは以前は気にしたことがなかったがそういうことまで考えなければいけないということが後になって判明する。それが分かっていなかったため, 設計するために書くドキュメントが今まで書いていたものだけでは足りなかった。次の段階に進むための資料作りが全然できなかった。汎用機の場合歴史も古く, 設計の方法論やテンプレートなどもできているが, ワークステーションでの開発の場合, 比較的金融では歴史が浅く, きちんとした方法論, テンプレートがないため, こうなってしまうのだろう。また, 業務についてもほとんど初めてのことだったので, 今までのような勘と経験で設計するということができなかった。そこで, ちゃんと DFD を書いて分析をしようと思ったのだが, " らしきもの" しか知らなかったのでうまくできず (例えば, DFD を書くにしてもどうやってプロセスを発生させるのかとか全然知らなかったし, ER 図をなぜ書くのかも良

く知らなかった), この時ほど設計の大切さ, 必要性を感じたことはなかった。汎用機時代のあんないい加減な設計技法でも困ることがなかったのだから, きちんとした技法を身につければ, 後のフェーズでどんなに楽になるだろうと思った。全く新しいことの分析設計をやる場合には設計技法が分かっているのといないのとでは大きな差が出ることが分かった。ただ, もちろんこれだけでは片手落ちで, ユーザから要件を引き出す能力も必要になってくる。金融系の場合, 設計技法の知識と, ユーザの業務知識が設計には不可欠な要素なのだと思う。しばらく後, 新人研修の教育スタッフをやる機会に恵まれた。そのとき構造化設計を少し勉強したのだが, 設計の大切さが改めて分かったような気がする。まずデータのながれに注目して DFD を書く, 次にその DFD をもとに SC を作る, データの正規化を行ない ER 図を作成する, 上流工程での誤りは取り返すのが大変など, 分かっていたこともあったが知らなかったことも多く, ワークステーションのプロジェクトで痛い目を見ていたので身につまされる思いだった。もう少し早く分かっていればプロジェクトがうまくいったのではないかと思うと残念である。ただ, 設計の大切さを身に染みて分かったことはとても良かったと思っている。最後になるが, 設計の大切さを実感できたこの時期にワークショップが開催されるというのはとても嬉しく思う。これを機会に今主流の構造化設計とかオブジェクト指向設計などの手法を少しでも自分のものにし, 今後の仕事に役立てていきたいと思う。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) 分析設計を行なう際の作業の流れを習得する.

(2) いろいろな分野の仕事をしている方の話を聞く.

(Q2) 期待は満たされたか?

(1) について.

今回採用した手法が局面的にどんな作業を行なえばよいかという指針を与たえてくれたので, 期待していた事は満たされた.

(2) について.

普段の仕事と全く違う話が聞けてためになった.

(Q3) 設計に意識の変化はあった?

大きく変わった訳ではないが, 通常行なっていた方法がどういう位置付けのものかということを改めて認識できた.

(Q4-1) 他人とのコミュニケーションについて

ワークショップに求めるものが人によって違っているのでなかなか難しいと思う.

(Q4-2) 短時間での, 問題の理解と作業について

普段別の分野の仕事をしている人同士が作業をしていくわけで, 問題のとらえ方とか, 使っている用語の理解とか, 本当に同じ認識でいるのだろうかという不安は残った. これだけの短期間で, すりあわせをするのは難しいだろうと思う.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には : 作業の進め方の流れがつかめたので, 実作業の目安になると思う.

(Q6) 全体討論の内容に関して

本の分野名をどの DB にどういう形でもっているとか, DB の内容を聞きたかった

(Q7) その他, W.S. に参加して良かったこと

(1) PC の方の話を聞けたこと.

(2) 普段の仕事と違う分野の仕事を行なう時に, どう作業を進めるべきかが少しわかった.

(Q8) W.S. の悪かったところ

課題を解くにあたって, もう少し多くの可能性を検討したかった.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

自分にとっては結構有意義なワークショップだったと思う. 出来れば多くの人に参加してもらいたい.

27 沖 さとみ (おき さとみ)
性別 : 女　年齢 : 26　血液型 : B
会社 : MHI エアロスペースシステムズ
所属 : 技術部制御システム一グループ

I. プロフィール

(1) 現在までの主な仕事の内容について

- 戦闘機の能力を評価するシミュレーションシステムの設計。
  担当は, システム全体のプログラム全体の制御を行うシステム制御系統。システムの概念設計, 基本設計を経て, 現在はコーディングを行っています。

(2) 現在利用している開発環境について

- HP シリーズ 400 : プログラム基本設計, 詳細設計
- ENCORE91 : プログラムコーディング
- PC-98

(3) これまでシステム設計に採り入れてきた技法など

- 特になし

II. ポジションペーパ

(4-a) 設計に感じた「ありがたみ」について

私が現在仕事をしているのは, 戦闘機能力評価システムというリアルタイムシミュレーションシステムです。本システムは, 私が入社した頃にちょうど立ち上がったプロジェクトで, システムの概念設計に始まって現在はプログラムの詳細設計及びコーディングを行っています。本システムでは, いわゆる「設計手法」と呼ばれているものを直接には使っていません。しかし, ソフトウエアのことを何も知らずに今の仕事を始めた私にとって, 「設計というのは, このようにやっていくんだな。」というような, 「仕事の仕方」というか「設計の流れ」みたいなものがわかってきたのが, 私にとって大きな収穫だと考えています。これまでの設計の流れを振り返ってみます。

1 システム概念設計

最初の1年は, システム全体の概念的な設計としてサブシステムの分割, タスクの制御方法, 共有データの管理方法等について基本的な処理方式の検討を行いました。
【例】データベース設計及びデータの管理方式の検討

本システムは60Hz周期で処理を行うリアルタイムシステムで, マルチプロセッサ (4CPU) のターゲットマシン4台構成となっています。従って, 複数のタスクが複数のタスク上で同時に動作するため, 以下のような問題が発生することが予想されます。

a 複数タスクによるデータの競合
b データを更新するタスクと, そのデータを参照するタスクが非同期に動作することによるデータの不整合

また本システムには過去に行ったシミュレーション試験の再現機能があるため,

c 実際に行った試験を忠実に再現するため, 処理として不確定要素がないこと

が要求されました。そこで,

1. ある1つのデータを更新するタスクは必ず1タスクとする。
2. 共有データを2バッファとし, 更新用と参照用を別バッファとする。
3. 1フレーム (1/60秒) 単位で更新/参照用バッファの管理を行う。

という方針で, データベースの設計を行うことにしました。

またタスクの分割, スケジューリング方法等についても, システムとしての制約, 要求を配慮し基本的な方針を決定しました。

2 基本設計

システムの概念的な設計が終了し基本設計のフェーズに入ると, プロジェクトの人数も増えてにぎやかになってきました。

サブシステム間のインターフェースの設計, タスクの分割等の上位の設計を, サブシステム (系統) のリーダーが設計チームとして行いました。まず各々が現状での問題点, 要処置事項を挙げ, 担当を決め, 担当者がたたき台となるものを作成します。その後, 設計チーム全員でそれを論議して決定し, 実際の作業を他のメンバーに展開していきました。
例えば, タスク分割を行う際は,

1. 機能単位でタスクを分割する
2. 処理周期に対応してタスク分割する
3. ハードウエアに対応してタスク分割する

の3方針についてそれぞれタスク分割案を作成し, 処理負荷を見積ってタイムチャートを作成しました。その上で各タスク分割案について, メリット/デメリットを挙げトレードオフを行いました。

システムの概念設計で漠然と考えていたものが, プログラムイメージでどの様に実現されるのか最初のうちは殆ど解りませんでしたが, 基本設計のフェーズでようやく形が見えてきて, 「次はどうなるのだろう?」となんだかわくわくしていました。また, プログラム側から使いやすく, デバッグ, 改修時に便利なデータベースの設計の大切さを実感しました。

3 詳細設計/コーディング

そして今, 詳細設計とコーディングを行っています。コーディングしながら感じることは, システムの概念設計はプログラム言語にとらわれない部分もありますが, よりよいシステム設計を行うためには, 実際にプログラムのコーディングを経験しなければいけないのではないかということです。今回のプロジェクトで得たことを, この次に担当するプロジェクトでうまく生かしていければよいと思います。

4 現状での問題点

今回のシステムはかなり規模が大きいため, 多少複雑になってしまったところもありますが, 制御/管理方式をシステムとして統一するという方向で, これまで設計を進めてきました。しかし, プロジェクトとしてもかなり多人数となり, 末端のメンバーまで基本方針, 処理方式が行き渡っていないところがあります。プロジェクトの途中から参加したメンバーに, 「さあ, コーディングして下さい。」と設計書

を渡すのではなく, システムの概念, 特徴を理解してもらうということが大切だと思います。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

設計手法の概要を理解したい.

実際の仕事をすすめていく方法みたいなものをつかんでみたい.

全く違う分野で仕事をしている人の話しをきいてみたい.

(Q2) 期待は満たされたか?

問題を与えられた時に, どこからとりかかるかみたいな糸口のつかみ方が少しわかったような気がする.

グループ単位の行動が多く, 他の人と話す機会があまりなかった.

(Q3) 設計に意識の変化はあった?

普段行なっている業務の中で「ものをつくる」ということにこだわりすぎ, 見直して修正するという後戻りは許されないという意識が潜在的にあった. でも, その後戻りしながら見直していくということも含めて, 分析という作業は行なわれるもので, 大切なことだということが認識できた.

(Q4-1) 他人とのコミュニケーションについて

相手のパーソナリティがわからないとコミュニケーションに遠慮があり, キャッチボール的なコミュニケーションをするのが難しいと思った.

(Q4-2) 短時間での, 問題の理解と作業について

職場によって, 同じものを表現する言葉が違っており, また逆に同じ言葉でも認識が違ってたりというギャップがあり, 作業をすすめていく上でかなり支障があった.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対し て, 新しい視点が持て るようになった.

具体的には: 今回使った SSADM を, まず小さな設計作業の中で試してみようと思います.

(Q6) 全体討論の内容に関して

設計方法により, もののとらえ方や, 問題点のアプローチの仕方がずいぶんちがうのはおもしろいと思った.

(Q7) その他, W.S. に参加して良かったこと

いろいろな設計手法があるけれども, それぞれが問題を解決する為のアプローチの方法として, その状況に適したアプローチしていけばいいのかもしれないと思った. だから, 食わず嫌いにならずいろいろ勉強しようと思います.

(Q8) W.S. の悪かったところ

私のグループでは図書館問題にとらわれすぎて, 現場での設計手法における問題点とか, そういう討論ができず残念でした.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

(記入なし)

15 **佐藤 公** (さとう こう)

性別: 男　年齢: 25　血液型: B

会社: 日本電気ソフトウェア (株)

所属: 応用ソフトウェア事業部 アプリケーション開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

- 機械系 CAD/CAM 製品 CAE-NC(E) の開発 (金型加工 NC データ作成支援)
- 機械系 CAD/CAM 製品 CAE-NC/MC(E) の開発 (機械加工 NC データ作成支援)
- 内容: システム設計 (機能設計) から総合テストまで, リリース物件作成, ユーザサポート, 開発環境整備, 等

(2) 現在利用している開発環境について

開発マシン EWS

開発言語 C 言語

環境 フロア内の EWS と PC で LAN を構築している. 約 50 台 (うち EWS30 台)

ネットワークを用いて, 特定マシン (EWS) での SI 作業を行ない, 集中管理している. また, 各 CAD/CAM 製品の開発はスタンドアローンで行ない製品共通部は, ネットワークを介し共用している. (EWS1 台/2 人)

(3) これまでシステム設計に採り入れてきた技法など

- CAE プロジェクト内で統一した作業標準 (設計からテストまで) を設け, それに沿った開発を行なっている.
- SPD ツール (SPD: 日本電気 (株) 内でのフローチャート相等のもの)

  フローチャートをプログラムに変換するツール. また, その逆も可.
- rflow ツール

  上位モジュールを検索するツール.

II. ポジションペーパ

(4-a) 設計に感じた「ありがたみ」について

私が, 開発作業の設計において, 「うらめしい」と感じることが二点あります.

一点めは, プログラムのコーディング形式が統一されていないために, 先人の作成したプログラム等のメンテナンスが大変に困難であるということです. 私の所属する

CAD/CAM プロジェクトは，「作業標準」と呼ばれる設計工程からテスト工程までの各工程を標準化した規則を持っています．基本的には，この規則である「作業標準」に従って開発作業を進めていくわけですが，しかし実際は，コーディングの際に SPD ツール (詳細設計からプログラムコーディングを行なうツール．SPD：日本電気内でのフローチャート相当) を使用していなかったり，「作業標準」におけるコーディング規則を守っていなかったりする場合が多々あります．その様なプログラムがメンテナンスの困難な対象となるわけです．メンテナンスが困難であることの原因としては，関数名や引き数名が必要以上に長すぎるものや，わかりにくいもの，引き数の処理内容 (Input,Output) がわかりにくいもの，コメント等が少ないため処理内容がわかりにくいもの，プログラムが読みにくいもの，等々さまざまです．この様なプログラムのメンテナンス作業は，どうしても後回しになりがちで，また作業も面倒です．これらの改善内容としては，先人の作成したものを再び作成し直すには，工数がかかりすぎるので，これらの対処方法として

1. SPD ツールの使用を心掛ける．
2. コーディング標準に従ったプログラムを設計する．

ということを実施していきたいと考えています．

二点めは，プログラムが構造化設計されているために，メンテナンスが困難であるということです．例えば，コモンを修正または改造した場合，そのコモンを使用しているソースを洗い出し，修正していかなければなりません．プログラムが構造化になっているため，一箇所を修正しただけでかなりのソースをメンテナンスしなければならないのは大変です．修正にもれがあった場合などは，テスト工程で後戻りの工数が発生します．一部分の修正により様々なところに影響するため，その影響範囲を考慮したメンテナンスが必要となります．その影響範囲の調査は，修正したコモンを使用している全てのソース及びコモンが調査対象となるため，余分な工数及び労力となっています．対処方法としては，オブジェクト指向寄りの設計を考えていかなければならないと思います．

私は現在まで，主に詳細設計以降を担当してきましたが，今後基本設計，機能設計などの上流工程を担当していくにあたって，CAD/CAM 分野におけるノウハウをどれだけ設計に生かせるか，また，どのようにして製品に反映させていくかということを，今回の研修で学びたいと考えています．

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) WS で紹介，使用される設計法の理解

(2) システム設計の上流工程の習得

(Q2) 期待は満たされたか?

(1) グループ毎にそれぞれの設計手法で設計していった為，理解しるまでには到らなかった．全ての設計手法で何らかのアプローチをしたかったと思う．

(2) 習得したかどうかは解らないが，システム分析から設計できてとても新鮮に感じられました．

(Q3) 設計に意識の変化はあった?

システム分析工程での重要性を認識しました．

(Q4-1) 他人とのコミュニケーションについて

はっきりした自分の意見を持ち，その意見を発言すること．かつ，他人の意見を真剣に聞き入り，それに対して意見を発せられることが大事と思われる．

(Q4-2) 短時間での，問題の理解と作業について

グループとして作業を進めるため，積極的に組んでいけば自然と他の人もついてきて，討議も進んでいくと思われる．

(Q5) 経験は，自分の仕事に対してどのような位置づけ?

仕事に対してどうこうといったことはまだよくわかりませんが，今後の仕事の上で何らかの影響があったと感じられたら良いと思っています．

(Q6) 全体討論の内容に関して

設計手法の違いによる差異を討論できる様な議題が上がれば良かったと思われます．

(Q7) その他，W.S. に参加して良かったこと

他社の方と討議することができ，とても刺激を受けることができました．

(Q8) W.S. の悪かったところ

出来れば，他の手法の理解及び運用について，習得まではいかなくとも少しでも触れられる場が欲しかったと思う．

(Q9) 今後 W.S. に取り上げたいテーマ，全体的感想

普段，システム分析から行なうことのない私にとって，とても有意義な 4 日間であったと思います．PC の皆さんいろいろとありがとうございました．ご苦労様でした．

12 **高橋 友重** (たかはし ともしげ)
性別：男　年齢：23　血液型：A
会社：(株) 大和ソフトウェアリサーチ
所属：第二システム部 運用二課

I. プロフィール

(1) 現在までの主な仕事の内容について

- プログラム作成
- システム移行 (コンバージョン)

(2) 現在利用している開発環境について

- 富士通 M360(OS—FSP)
- 富士通 M1800(OS—FSP,MSP)

(3) **これまでシステム設計に採り入れてきた技法など**

JCL 構造解析ツールの作成

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

今年の 1 月より, M360 から M1800 へのシステム移行 (コンバージョン) を行なっている. 各サブシステムごとの処理がどのようになっているのか調べようにも, 即存ドキュメントが不完全であって使いものにならないのが現状である.

JCL 構造を解析するツールを最終的に作ってみても, その JCL がどのように呼ばれているのかは誰もわからず, 結局 5 階層にも 6 階層にもなっているものもあった.

JIS COBOL から COBOL85 への変換作業にしても, 業務ごとにながれている JCL より, ロードモジュールを特定して, それをツールにかけて, 構造図を出力して, プログラムソースを特定する, という方法をとっている.

このように, 生きている資産というものを把握するのに, 多大の時間と労力をとられてしまい, 設計がどうこうというよりも, 手作業がほとんどという状態である.

どこの会社でも, 多かれ少なかれ, このようなものなのであろうか. ドキュメント類, DASD 内等の管理はいったいどのように行なっているのだろうか.

当社でもこれからは, ライブラリの管理をおこなうためにライブラリアンをおいて, DASD 内になにかを作成するときは, 申請書を提出するようにしているが, 果たして何人の人が実施しているのであろうか.

また, 現在 MSP の交換のために, JOB 構造図等のドキュメント類も作成されているが, これは, ちゃんと引き継がれていくのであろうか.

このように, 私が入社して 1 年, 即存ドキュメント不備のためにふり回されている毎日である.

III. W.S. 終了後の感想

(Q1) **期待していたことは?**

実業務の中にどうすれば, 設計をとりいれるか理解すること.

(Q2) **期待は満たされたか?**

満たされたかどうかは今後にかかっていると思う.

(Q3) **設計に意識の変化はあった?**

(記述なし)

(Q4-1) **他人とのコミュニケーションについて**

各人の持っているイメージを統一するのに時間がかかる.

(Q4-2) **短時間での, 問題の理解と作業について**

(記述なし)

(Q5) **経験は, 自分の仕事に対してどのような位置づけ?**

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) **全体討論の内容に関して**

(記述なし)

(Q7) **その他, W.S. に参加して良かったこと**

(記述なし)

(Q8) **W.S. の悪かったところ**

各設計方法に対してそれを使用したことにある人の講義 (利点・難点等) を聞きたかった.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

もっと参加対象者をしぼった会 (例えば, UNIX 使用者のみ等) も行なって欲しい

グループ 2

08 **直江 魅音子** (なおえ みねこ)
性別：女 年齢：27 血液型：A
会社：(有) ネクストファウンデーション
所属：

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 証券会社 外為管理システム, 株式管理システム (COBOL)
- 進捗状況管理システム, 資料管理システム等
  PC 上で簡易データベースソフト (dBASEII, R:BASE5000) による管理システムの構築
- 網性能評価システム (FORTRAN)
  APOLLO 上で GUI 作成ツール Dialogue による国際電話網シミュレーションシステムの構築 (GUI 及びデータ操作部担当)
- 光学設計支援システム (C)
  X-Window 上で Athena Widget による光学設計の為の表形式 (スプレッドシート) データ入力システム及び収差図, 光路図描画システムの構築
- オンラインマニュアルの翻訳
- OSF/Motif の日本語化 等

(2) **現在利用している開発環境について**

- SUN SparcStation 1+ (SunOS4.1.1, X11R4, OSF/Motif1.1.4,GNU/Emacs)

(3) **これまでシステム設計に採り入れてきた技法など**

- 特になし。

II. ポジションペーパ

(4-a) 設計に感じた「ありがたみ」について

システム設計に対する「ありがたみ」も「うらめしさ」も特に感じたことはないのですが, ここでは, 「うらめしさ」をまじえつつも「ありがたみ」に関して述べさせていただきます。

システム設計と一口にいっても, 「要求を分析し, 実現させる」というその目的は同じでも, 環境, 言語, 目標等によってその内容はまったく違うと思います。
私の場合, 入社当時は COBOL による情報処理システムに携わっておりました。当時の仕事は既存のシステムを各証券会社に合わせてカスタマイズするといったもので, ここでのシステム設計はほとんどデータを変更するだけのはめ込み式のものでした。

次に経験したのが, やはり COBOL なのですが, 数社による数十人の大きなプロジェクトでした。ここでは, 各社がおのおの 1 つのシステムを担当し, 毎朝ミーティングを行っていたのですが, プログラミングの段階で週に 1 度は必ずといってよいほどの仕様変更が出ました。原因はデータ設計が甘く, あちらのシステムがたてば, こちらのシステムが立たず, といった状況になってしまったのでした。プロジェクトが大きくなればなるほどこういう問題は生じます。よく, 大きなプロジェクトでは担当部分でせいいっぱいで全体はわからないということがありますが, 要であるデータに関しては設計の段階ですべてを把握しているのが当然であるべきです。(理想ではなく)

その後, 移動に伴い, FORTRAN 等の技術計算系を経て, そのころはまだマイナーだった UNIX 環境での X-Window の仕事に携わりました。

この X-Window の仕事で, 私のシステム設計は変わりました。

X-Window の Widget はそれ自体がオブジェクト指向の考えを取り入れていますので, 設計も自然とオブジェクト指向的になります。このため, システム設計の要である客先の要求分析がより詳細になり, 設計モデルやプロトタイプを作成することにより, 客先とのコミュニケーションも取りやすくなり, 相互理解に役立ちます。(惜しむべくは加えて上流 CASE ツールがあるとさらに設計が容易かと思いますが, いかんせんまだまだ高過ぎて手がでません。)

そんなわけで, X-Window での開発は良い経験となりました。

今回はシステム設計のなかでも, 要求分析, 基本設計を中心に述べてしまいましたが, 実際のシステム設計ではターゲットマシン等の開発環境やコスト, 開発期間によってかなり左右されてしまいます。かくゆう私も前述の X-Window で, 限定された Athena Widge をいかに使用してスプレッド・シートを実現させるかに七転八倒致しました。

本来ならば, システム開発の技術や技法について述べるべきなのでしょうが, 恥ずかしながら語るべく経験を持ち合わせておりません。今後, システム設計を正確にかつ効率良く行うための手法を学んでいきたいと思います。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

・種々の設計手法に関する知識を身につける.
・他社の人達の設計方法を知る (意見を交わす)
・自分の設計方法についての確認をとる.

(Q2) 期待は満たされたか?

(満たされたもの)
・他社の人達との意見交流 (グループ内のディスカッションによって)
・ SA/SD の SD の記述法に関する理解
(満たされなかったもの)
・ジャクソン法等 SA/SD 以外の設計手法に関する知識
・ SA/SD の SD に関する詳細

(Q3) 設計に意識の変化はあった?

・世の中には様々な設計手法がある→自分の知識の少なさの再認識
・しかし, あまり手法にこだわる必要はないのではないだろうか.
→対象となるシステムによって Case by Case で各手法をとりいれればよい.

(Q4-1) 他人とのコミュニケーションについて

このてのワークショップは初めてで, 性格上 (人みしりが激しい) かなり不安がありましたが, ふたをあけてみればメンバが良かったのか とてもスムーズにうまくいきました. 5 人もの人間がいれば意見がばらばらなのは当然ですが, みんなで言いたいことを言いあって問題点を一つ一つクリアしていったのが良かったと思います.

(Q4-2) 短時間での, 問題の理解と作業について

私達の場合, 例えばともすればあいまいになりがちな仕様の詳細なども, 人間シュミレーション (:-)) 等を 行なって一つ一つ決めていくなどして, メンバ全員の意思の統一ができていたと思います. この意思 (システム・イメージ, 目標等) の統一が グループとして作業していく上で重要なポイントだったと思います. また PC の方々のグループの統制力にも多いに助けられました.

(Q5) **経験は，自分の仕事に対してどのような位置づけ?**

■今後の 仕事の進め方に対して，新しい視点が持てるようになった.

具体的には：(記述なし)

(Q6) **全体討論の内容に関して**

全体討論としてはもう少し議題を上げていただきたかったです. やはり手法によって取り組み方や考え方，視点等が違うので最終発表をもう少し長く取り，質疑応答などで相互理解を深めたかったです.

あと一つだけ主張したいのは，G5 の方から先生と父親というたとえで職員が借りられるのはおかしいという意見がありましたが，私達の間でもそういう話しはありましたが最終的に職員の方から「わざわざカードを作るのが面倒だから」という意見があってああしたまでです. あれはあくまでも仕様 (客先要求) であることを誤解しないで欲しいです.

(Q7) **その他，W.S. に参加して良かったこと**

私の場合，普段人とは話さないという特殊な環境で仕事をしているので，人と議論し，一つのものを作っていくということは，それだけでも大変有意義な経験になりました. 自分の中で固まっていたものが リフレッシュ(再び活性化) しだしたような気がします.

(Q8) **W.S. の悪かったところ**

進め方としてはとても良かったと思うのですが，結果的に設計の段階まで行けなかったのが残念です. あと 1 日はいりませんからせめて半日は欲しかったです.

(Q9) **今後 W.S. に取り上げたいテーマ，全体的感想**

特にテーマはありませんが，次回もぜひ参加させていただきたく思います. (今度はまじめにポジションペーパーを書きますので... (_ _) peko)

14 **高巣 英利** (たかす ひでとし)
性別：男　年齢：26　血液型：?
会社：(株)SRA
所属：中部支社

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 金融業の第三次オンラインシステムのメンテナンス
  (夜間センター一括処理) 勘定系
- 第三次オンラインシステムパッケージソフトの導入のカスタマイズ
  (夜間センター一括処理) 勘定系要求定義，基本設計，詳細設計，テスト etc

(2) **現在利用している開発環境について**

- マシン： IBM3090 大型汎用機
- OS 等： IMS
  SAIL(IMS オンライン・アプリケーション開発支援)
  TSO
  客先常駐による作業環境

(3) **これまでシステム設計に採り入れてきた技法など**

これまでメインテナンスの作業がほとんどであったため，何等かの設計技法等を積極的に使っての開発自体は，行えていないのが現状です。

しかし新規モジュール等の作成の際には，若干ではあるが，本で読んだ程度の「複合設計」を意識して行った事があるくらいです。

II. ポジションペーパ

(4-b) **設計に感じた「うらめしさ」について**

あまりうらめしいとは考えたくはないのですが，私の今までの経験は，既存システムに対するメインテナンスが主だった作業なのでその中からの事について感じた事を述べます。

まず，「何が」

既存の環境に拘りすぎて，新しい技術等を導入するより既存の物と似通ってしまう。

「どの様に」

既存のシステムの稼動環境がかなり頑固に有るため，アプリケーションの機能を追加する際に，設計上納得がいかなくとも，ある程度既存と同等のサブ・システムに成ってしまうところ。また，実際に本番環境に於て稼動しているため，できるだけ修正・変更の箇所を極力最小限にとどめる傾向にある。

「どうして」

モジュールの共通化がかなり行われているのだが，その共通モジュールに call する側の機能を’ふんだん’に意識してしまったため，それらの共通モジュールを使用する際には，かなりその内容を把握しておかなければならない。当初はモジュール機能をきちんと分割していたかもしれないが，現状は，共通モジュールでどれを使おうか，とゆうより，使って他に影響がないかどうかを検討したうえで設計を行わなければならなくなってしまっている。また，そのような変更の繰り返しのため，基本的なモジュールの構造・機能自体がゆらいでしまう悪循環を繰り返しているように思える。システムが成熟してゆくにつれて，モジュールの強度が強くなるのではなく複雑極まりなく変化してゆくように思われ

る。どうしたらよいか

現在の環境では，あまり今後の発展は望めないものの，これから先システム開発に付きまとうであろうメインテナンスの時期にある程度柔軟性のある構造を持ったシステムを構築してゆかねばならないのではなかろうか。また，設計の基本コンセプトを尊重しつつその場限りの対応ではなく，恒常的に設計を繰り返すことも大切な事ではないだろうか。現在のカスタマイズにおいても，かなり複雑であり「知る人ぞ知る」面もかなりある気がしている。しかも，私が今たずさわっているシステムにおいては，1000 万ステップ程あると言われているが，既に基本的なコンセプト等は忘れ去られ，よりマン・パワーの少ないメンテナンスだけを行っているに過ぎないのではないかと思う。

設計は変化してゆくものであるから，モデルチェンジもある，この時，今の設計と，今のベースでどこまで変化に耐え得るか。モデルチェンジ毎に，図面を 1 からひいていては，よりよいフィードバックは出来ないと思う，現在のシステムの評価も行ってメインテナンスするべきか?

以上ですが，本題とは離れているかもしれませんが，何卒経験のない私書きたい事を書かせて頂きました。

III. W.S. 終了後の感想

(Q1) **期待していたことは?**

・設計手法で今後役に立ちそうなことを身につけたい.

・他社の人とのコミュニケーションを図りたい.

(Q2) **期待は満たされたか?**

満たされた. 基本的な考え方をまず身につけられたことと，PC の方に手法にこだわらない意見や経験による

(Q3) **設計に意識の変化はあった?**

色々な設計手法に有効なアプローチのやり方があり，多く知った上で自分自身の方法を確立して論議できるいきに達したい.

(Q4-1) **他人とのコミュニケーションについて**

すごく良くできたと思う. 特にテーマよりグループで行なう内容の方針のプレゼンテーションに魅力を感じて集まったメンバーだったので，私達のグループは全員で徹夜もやったし，PC の伊藤さんのおっしゃるところの『“美しい”の意味』を理解できたし，共感できた. また，他の人も口々に“美しい”と発言するまでになった.

(Q4-2) **短時間での，問題の理解と作業について**

短時間で出来なかったが，プロセスを大切にして模造紙を使うことで深くまでかなり仕様固めができたのは良かった.

(Q5) **経験は，自分の仕事に対してどのような位置づけ?**

■自分の仕事の進め方に自信が持てた.

■今後の 仕事の進め方に対し て，新しい視点が持て るようになった.

具体的には： 「明日から役立つ“小さな発見”の意識」を持って今後の仕事に役立てたい. こつこつやっていくことの大切さがわかった.

(Q6) **全体討論の内容に関して**

(記述なし)

(Q7) **その他，W.S. に参加して良かったこと**

(記述なし)

(Q8) **W.S. の悪かったところ**

出来ればメンバ (グループ) とゆっくり酒を飲んで話したかった.

(Q9) **今後 W.S. に取り上げたいテーマ，全体的感想**

(記述なし)

16 **木崎 俊広** (きざき としひろ)

性別：男　年齢：27　血液型：B

会社：日本電信電話 (株)

所属：九州支社 技術開発センタ

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 社内業務改善 OA 化
- 開発したシステムの保守及びバージョンアップ

(2) **現在利用している開発環境について**

- ワークステーション (sony NEWS, sun) と pc9801 等のパソコンを Ethernet で接続し，ディスクの共有化を図っている。
- 開発言語は C, アセンブラ
- 自社開発の設計ツールを利用
  (CASE ツール NTT ソフト研究所開発)

(3) **これまでシステム設計に採り入れてきた技法など**

- ワークステーション上でのツールを利用し，そのヘッダー部分を整形し，プリントアウトしてモジュールの二度手間を省いた。
- ワークステーション上で，C 言語で作成した関数を切り出し一覧表で見れるようにし，ドキュメントの保守に役立てた。

II. ポジションペーパ

(4-b) **設計に感じた「うらめしさ」について**

1 ユーザ要求をシステムに的確に反映できない

ソフトウェア開発作業において, 幾度かの苦い経験がある。それは, ユーザ要求をシステムに的確に反映できなかったことである。

システムを開発する場合, まず必要なことは, 基本検討でユーザの求めているニーズは何かを正確に理解することであるが, 開発スキル (折衝能力, チームリーダの力量) が不足な為に, ユーザが求めているニーズと開発側が受けとめたこととの間に, ギャップを生じたまま開発を進めることがあった。それがそもそもの間違いである。その時点では, ユーザもシステムイメージが不鮮明であり, こういった機能があればよいというイメージしかない。開発側にとっても, これだろうといった想定のシステムイメージしかないだろう。この食い違ったイメージのまま, 開発を進めることで, 後でこんなシステムではなかったと意見の相違が現われる。

そこで, この意見の相違をなくすために, 基本検討段階で十分な対ユーザとの会合を実施しベクトルを確認しあうことや, トップとの確認をとることが, 開発工程を円滑に進める重要なポイントだと考える。

この他にも, オブジェクト指向というものがあるが, 私は, まだオブジェクト指向というものが, どの程度ユーザ要求を分析し, ユーザとのインターフェース設計に役に立っているかが, まだよくわからない。もし, 今後このオブジェクト指向が, ユーザの真のニーズは何かを捕らえることに役立つならば, ソフトウェア開発作業で, 開発工程を短縮できるように今後取り入れて行きたい。

2 品質管理がおろそかになってしまう

ソフトウェア開発工程の中で, もう一つ重要なことがある。それは, 品質管理である。品質を高めると, 進捗も早くなり生産性も向上するといわれている。しかし, 現状の開発ではシステム納期に間に合うようにと必死になって開発している。その為, 品質がおろそかになり, テスト工程で多くのバグが発生したり, 機能落ち, 設計ミスがあり後工程において多大な稼働を要する結果となっている。

私達の職場でも, なんとか品質を高めようと, レビュー (インスペクション・ウォークスルー) を行っているが, どうも経験不足, スキル不足のため活発な意見がでてこない。そのため, レビューの方法を考えたり, 様々な品質の本や手法を取り入れたりして工夫している。

3 開発レベルに合った設計方法はないものか

私の職場では, 主にパソコンレベルの開発が多い。開発工程のライフサイクルとして, 基本検討, 基本設計, 詳細設計, プログラム設計, コーディング, 単体試験, 結合試験, 総合試験, 総合運転試験, 移行, 運用の工程をとっている。しかし, パソコンレベルでは基本設計と詳細設計とが重複している点があり, 最低どの部分までを基本設計で行えばよいのかの作業配分が明確になっていない。また, コーディングと単体試験を同時に行っているのが現状である。

世の中の動きとして, 今後 UNIX が主流になってくると, UNIX での開発工程というのが存在してくるのだろうか?また, 大型システムとパソコンレベルとでは開発設計の方法が違うはずだが, どんな点に違いがあるのか今までの設計経験の中で疑問に思う。

コンピュータは日進月歩性能がアップしているが, システム設計手法の進歩はスローテンポである。そんな中最も効率が良く, 無駄のない設計方法ができないものか, 私は日々考えている。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) 他社と自社の設計方法はどのような違いがあるか, またどの点を工夫すれば分析/設計においてスムーズな流れとなるか.

(2) オブジェクト指向を取り入れた設計方法では, どんな知点があるか.

(3) ユーザーニーズの分析において, 後戻りがなく確実な分析が行なえるか.

(Q2) 期待は満たされたか?

(1) 設計方法については, 他社も自社も違いはなく初期段階でシステムイメージを考え, 問題点の洗い出しを確実に行なうことが工程を進めていく上で大切だ.

(2) 他グループの発表, 話を聞く事によりオブジェクトの指向の考え方がわかった.

(3) 1つ1つを押えフィードバックをする事で確実な分析が行なえる.

(Q3) 設計に意識の変化はあった?

ソウトウェア の設計において仕様を固め, メ ンバー内のコミュニケーションギャップをなくし, 1つ1つ工程において確認していくことの重要性がわかった.

(Q4-1) 他人とのコミュニケーションについて

他人の考えている事を素早く理解し, 目で見える形で残し討論を行なうことでコミュニケーションがとれる.

(Q4-2) 短時間での, 問題の理解と作業について

(1) グループの日別目標を立て, スケジュールで行動する.

(2) 問題が生じたらどうするのか討論し素早く解決策を立てる.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■自分の仕事の進め方に自信が持てた.
■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) **全体討論の内容に関して**

SA/SD, オブジェクト指向, JSD, SSADM 等の手法があるが, この手法はこのシステムに適するということが解らなかった.

(Q7) **その他, W.S. に参加して良かったこと**

SA/SD 法の修得ができ, 分析/設計において質問表, 模造紙等を利用して行なった方が短時間で問題を解決できると思った.

(Q8) **W.S. の悪かったところ**

日数/時間的余裕がなく, SD の手法はやれなかった. 手法を確実に理解しながらいくともう少し時間が必要だ.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

いい経験をさせていただきました.

24 **藤巻 昇** (ふじまき のぼる)
性別：男　年齢：25　血液型：B
会社：(株) 東芝
所属：システム・ソフトウェア技術研究所

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 状態遷移図エディタの開発〜保守
- 金融機器制御ソフトウェア向け CASE の開発

(2) **現在利用している開発環境について**

マシーン SUN ワークステーション TOSHIBA AS シリーズ, SPARC LT
パソコン TOSHIBA J-3100
OS UNIX, MS-DOS
エディタ nemacs, vi, mifes

(3) **これまでシステム設計に採り入れてきた技法など**

状態遷移モデルを用いた仕様記述法

II. ポジションペーパ

(4-a) **設計に感じた「ありがたみ」について**

1 はじめに

上記の状態遷移図エディタの開発を通し, 対象分野に適した設計手法を用いると開発・保守が非常にやり易くなるということを体験した。

状態遷移図エディタの開発作業は, 以前に開発されたエディタの機能拡張にともなうバージョンアップを行うことであった。この開発でまず行ったのは, 6000 ステップ余りのソースコードを読みながら全体の構成を把握することであった。

エディタの Ver.1.0 には各種仕様書やマニュアルなどが存在したが, この作業はより詳細な内部仕様を理解するために行った。ソースの読み込みと同時に, 理解した仕様は東芝で採り入れているモジュール設計記述法 TFF(Technical description Formula for Fifty steps/module design) でまとめていった。

この作業中から, モジュール化がうまい単位で行われていて読み易いソースであるとは感じていた。そのときはコーディングのうまい人が作ったのだろう, ぐらいに思っていた。しかし, エディタに新しい機能を付け加える段階になり, このエディタの設計の偉大さに驚愕することになった。

2 開発期間短縮の理由

それは機能拡張が予定の半分ほどの期間で終了するといった形で現れた。この機能拡張作業には事前の見積で 3 人月, それでも多少後ろ倒しになる危険性ありと考えられていた。それは, 当時新人で配属されたばかりの経験の浅い人間が, 初めて目にするシステムに携わるという要素があったからだ。私も他人の作ったプログラムに手を入れることの難しさはよく承知していたので, かなりの困難を予想していた。しかし, 実際には 1. 5 人月でほぼ開発を終了する事ができた。開発期間が縮小された理由は何だったのであろうか。それをまとめると,

1. 対象システムに分野特化された設計手法の採用
2. その設計手法と対応の取れたソフトウェア・アーキテクチャの採用
3. 設計手法と相性の良い開発言語の採用

にあった。

ここで言う設計手法とは, 仕様記述法のことである。この設計手法は世間に流布する著名な手法によるものではなかったが, その仕様記述は状態遷移モデルをベースにしたもので, 「エディタ」というユーザインタフェースに力点を置くソフトウェアの仕様を明確に記述するには, 有効な手法であった。また, 作成された仕様書は拡張性の高いものとなった。

ソフトウェア構成はこの状態遷移図ベースの仕様書に忠実に対応づけられていた。そのおかげで, 仕様上で機能拡張したとき, どのモジュールに何を付け加えればいいのかが仕様上からハッキリと見えていた。後は実際にモジュールに機能を追加していくだけであった。

エディタ作成にはウィンドウプログラミングを行い，ウィンドウシステム SunView を利用した。SunView 上で開発することで，プログラムをイベント・ドリブンに容易に記述でき，これはイベントとアクションで記述する状態遷移による設計によくマッチしていた。そのおかげで，上記のソフトウェア構造が素直に実現されていた。このことは，ソースコードそのものを理解性の高いものにしている。

以上のような理由で機能の拡張作業をスムーズに行うことができた。

3 まとめ

この経験を通して設計に感じた「ありがたみ」とは，良い設計の提供する開発作業の「快適さ」である。ここでの「良い設計」とは，流行の設計手法をそのまま使うことではなく，対象となるシステムに特化された手法を採り入れた設計のことである。

設計の良さが開発作業に快適さ生み，その結果として開発効率，ソフトウェアの保守性，品質を高めることになる。Peter Coad と Edwrd Yourdon は著書"Object-Oriended Design"の中で，「設計は，他人の人生を左右する」"Design affects the everyday life of other humans."と述べている。まさに，その通りではないだろうか。設計の良さが開発者の作業効率に影響し，ソフトウェアの品質を決め，さらに利用者の作業に影響し，ひいてはその利用者から生み出される「何か」の品質にも影響してくる。

私自身はまだ開発の経験は浅く，今回紹介したシステムも大規模なものではないが，担当する業務が CASE の推進を目的としていて，設計手法を考える立場にある。 現在携わっているのは「金融機器制御ソフトウェア向けの CASE の開発」というテーマであり，ソフトウェア・アーキテクチャを含めた仕様記述方式の検討と，ソフトウェアの自動合成といった研究・開発を行っている。その過程で現在感じているのは，「本当に効果的な設計支援とは何か?」ということである。仕様記述方式を検討する過程でいくつかの案を検討したが，ともすると流行の方法論に走り実状に合わないこともあった。変更や保守性まで考え，かつ現場の実状にあった設計手法とはどんなものであるのか。まだ漠然とした間であるが，その荷な立場から，今回のワークショップに臨みたいと思う。また，色々な方法論で共通の課題がどのように解かれるか，その点にも非常に興味がある。そして，このワークショップで得るものを糧に開発者に「快適さ」を与えるような設計手法を考えていきたいと思う；

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) 同じ問題が別々の手法でどう設計されるか．それを参考に手法の特徴を理解すること．

(2) 他社技術者との交流

(Q2) 期待は満たされたか?

(1) ほとんどのグループは設計までいっていなかった．仕様の分析に手惑っていたのが原因．しかし，分析の段階でも手法ごとの観点の違いが良く出ていた

(2) 友好は十分に深められた．

(Q3) 設計に意識の変化はあった?

方法論にしばられずに“出来ることからやっていく”ことが方法論の適用には大切．要求仕様をフィックスする作業が重要である．

(Q4-1) 他人とのコミュニケーションについて

自分の意見をうまく主張することと，他人の考えをハッキリと理解すること

(Q4-2) 短時間での，問題の理解と作業について

試行錯誤のサイクルを早く廻すことが必要．

(Q5) 経験は，自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して，新しい視点が持てるようになった．

具体的には：模造紙を使った方法．

明日からできる “小さな発見” 等を仕事に役立たせることができる．

(Q6) 全体討論の内容に関して

利用者，職員の問題は，これが正解という定義は 存在しないと思う．方法論により要求分析のアプローチが違い，その違いが大きく定義の違いに現れ るのだと考えられる．一つ疑問に思ったのは OO の手法には 具体的なデータ構造のイメージは必要ないのか? ということだった．

(Q7) その他，W.S. に参加して良かったこと

参加前には，手法を学ぶことを大きな目標としていたが，「手法にこだわらない」というグループで活動したことで，問題への取り組み方も学ぶことが出きたのは，うれしい誤算であった．

(Q8) W.S. の悪かったところ

積極的なメンバーのいるグループに参加でき，非常に有意義で楽しくもあり，SD まで終了した時の成果が期待できたが，時間的な理由で SA の途中で終了したのは残念であった．

(Q9) 今後 W.S. に取り上げたいテーマ，全体的感想

オブジェクト指向/分析/設計/プログラミング等のオブジェクト指向に関する手法適用．

26 寺井 文康 (てらい ふみやす)
性別：男　年齢：26　血液型：B
会社：MHIエアロスペースシステムズ
所属：技術部開発グループ

I. プロフィール

(1) 現在までの主な仕事の内容について

- ニューラルネットワークを用いたパターン認識システムの開発
- ソフトウェア開発支援ツールの開発

(2) 現在利用している開発環境について

マシン HP ワークステーション
OS UNIX
使用言語 C

(3) これまでシステム設計に採り入れてきた技法など

これまでシステム設計に採り入れてきた技術・技法は特になく，プログラムを書きながら設計するという方法を採ってきました。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

1 今までの設計のやり方

まず最初に，今までの自分の設計のやり方について述べたいと思います。作るものの仕様をおおよそ理解すると，すぐに端末に向かってプログラムを書き始めるというやり方を採ってきました。モジュールをどのように分割しようか?，とかデータ構造をどのような形にしようか?といったことは最初に大ざっぱに決めておき，プログラムをどんどん (?) 書いていきます。途中で行き詰まることもありますが，そういう場合はあっちこっとをいじくりながら最後まで作り上げてしまいます。ドキュメントに関しては，きちんとした形で作るというわけではなく，プログラムの中に雀の涙程度のコメント文を書くぐらいです。

2 設計に感じた「うらめしさ」

こういうやり方を採ってきたのは，1つには，時間がないからとりあえず作ってしまおうという気持ちが強かったからです。時間がない場合には，設計という作業とプログラムを書くという作業をまとめてやってしまうということは仕方がないことだと思います。そのほかの理由としては，今までに経験した開発作業が，比較的小規模なものだったからというのが挙げられます。ある程度，全体を把握しながら作業を進めることができるし，また，これぐらいだったら特別設計という作業をしなくても，頭の中で考えながらできる，という自信を持っていたのかもしれません。しかし，できあがったシステム，プログラムをみてみると，それほどできがいいものではありませんでした。それは不具合が多いものになってしまっていたのです。また，その修正のための作業も，大変労力を必要とするものになってしまいました。時にはどこになにが書いてあったのかを忘れてしまうことがあり，そういう場合は，貧弱なコメント文を頼りに1行1行追って理解していかなければならず，非常に時間がかかる作業になってしまいました。仕様の変更という作業についても同様のことが言えました。こういったプログラムをいくつか繰り返して作ってくると，今までの設計のやり方に対して「うらめしさ」を感じるようになってきました。では，その「うらめしさ」をなくすために，当然何らかの「改善の提案」が必要になってきます。

3 「改善の提案」

「設計」とは，簡単に言ってしまうと，問題をいかに (how) 解決するかということです。今まで私は，プログラムを作りながら経験とか勘のようなものを頼りにして，その「how」という作業を行ってきました。こういった，経験とか勘を頼りにしてプログラムを作りながら作業を進めるというやり方は，間違っていないどころかひょっとすると理想的なやり方なのかも知れません。でも，このようなやり方は，「優秀な」ソフトウェア技術者でなければできないことです。しかし私自身，とてもそんな「優秀な」ソフトウェア技術者ではありません。それは，今までに開発したシステム，プログラムからわかります。ということは，私はまだ「未熟な」ソフトウェア技術者ということでしょう。では，その「未熟な」ソフトウェア技術者が「優秀」なソフトウェア技術者になるためのよい手助けとなるものがないのでしょうか?

その手助けの1つになり得るものが「SA/SD」や「OOA/OOD」と呼ばれる設計手法だと思います。設計手法が「優秀な」ソフトウェア技術者になるための案内人となってくれるような気がするのです。ほかに手助けとなってくれそうなものが見当たらないから，とりあえず設計手法に手助けをお願いしたい，それを「改善の提案」ということにします。しかし，どの手法を手助けの手段として選択するか，という難しい問題が残ります。これについては，「若手の会」を通じて見つけ出すということにしておきたいと思います。いかんせん，設計手法といえば「SA/SD」を若干知っている程度ですから。

では，最後になぜ設計手法というものがいくつか存在するのか?について考えてみたいと思います。歴史が浅いから，

まだこれというような絶対的な設計手法が確立していないという理由を最初に考えました。では, これから先年月がたてば, 絶対的な設計手法というものができるのでしょうか? 私は決して不可能なことではないと思います。しかし, できたとしても, それはとても複雑で習得するのに時間がかかるような, とても使いづらいものになると思います。ということは, 特定の問題を解決する設計手法がいくつか存在するのが自然な形であると思います。どの手法が優れているのかということを議論するのではなく, どの手法を選択するのかということを考えなければならないと思います。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) ある 1 つの設計手法 (SA/SD, OOA/OOD 等) を修得する.

(2) 自分自身が行なっている分析, 設計方法について良い点悪い点を見つけ出す.

(Q2) 期待は満たされたか?

(1) SA/SD を一応知っているつもりでした. しかし, ただその記述方法を知っているだけということを知らされました.

(2) あまりにも設計の前段階で行なう作業に手をぬいているかがわかりました.

(Q3) 設計に意識の変化はあった?

設計にはいる前に行なう作業 (仕様の理解, 要求の分析) がとても重要であることがわかりました.

(Q4-1) 他人とのコミュニケーションについて

他人とのコミュニケーションをとるということは, 問題に対する共通の理解を持つと いうことで, とても大切である ことが解りました.

(Q4-2) 短時間での, 問題の理解と作業について

短時間で理解するということはとても難しいと思っていました. しかし, 今回の ワークショップを通じて (方法と しては人間シュミレーションという方法を採った) 意外に簡単だなと感じました.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には : 少なくとも自分の行なっている分析・設計のやり方に対して何か変えなければならないという気持ちが出てきました.

(Q6) 全体討論の内容に関して

我々が使った模造紙というやり方について, 他の人の意見を聞きたかった.

(Q7) その他, W.S. に参加して良かったこと

あまりにも設計手法というものにこだわりすぎていたと思います. それよりも今すぐからでも出来るようなことを見つけて, 一歩一歩積み重ねていく努力が必要であると思います.

(Q8) W.S. の悪かったところ

人数が多いので仕方ないと思うのですが, やっぱり開始時間とか終了時間をもう少し守って進行すれば良かったと思います.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

旅館みたいなもっと at home な環境の場所でやれればと思います.

グループ 3

06 **酒井 雅裕** (さかい まさひろ)
性別 : 男　年齢 : 30　血液型 : B
会社 : ハイテック (株)
所属 : 開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

- 自社 CAD/CAM システムの一部の設計, コーディング
- FA 関連商品の請負開発
- 現在もプログラマかつ管理職も少し

(2) 現在利用している開発環境について

- NEC9801 シリーズ, その互換機。ネットワーク環境はなし
- 将来では, AT 互換機
- 可能性としては, 特別製ハードウエア (クロス開発?)
- 開発言語は C

  統合環境を使わないように指導されて, 部下にも指導しています。メモリが足らないことを経験してから, 私自身はソースデバッガも使いません。

(3) これまでシステム設計に採り入れてきた技法など

- 上司+私+部下+外注＝1＋1＋2＋4 位の規模で, PC がターゲットですから, 特にこれといった技法も技術も設計のポイントとしてはありません。
- しいていうなら, 個々人の良心。

  わかりやすい構造にすることを心掛けて設計。

  可読性を重視するとか, 適切なコメントを入れるとか, ドキュメントを残すとか。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

1 設計なんてやったことがない

いまさら, 管理職にもなっていうのもなんですが, 設計なんて気にかけて思ったことはございません。適当にデータ構造を考えて, 適当に機能構造を処理単位に分けて, 画面設計は, 美しくないと営業におこられますから, 多少真面目にやって, 入力のインターフェイスは一度は真面目に考えてみたことがありますが, これなら使いやすいだろうとか思うと, まったくの独り善がりだよっていわれて, なまじ, 真面目にやってるぶんだけがっかり度が大きくそれ以来考えなくなりました。

すると自分が今真面目にやってる設計は画面設計ぐらいということになってしまいます。あと, 問題解決方法の設計ですか。

2 なぜ, やらないのか

画面設計にしても, ユーザーに納めた途端, 「あーっ!ここが足りんかったぎゃあ」といわれると, 血の気が引いていくのがわかります (もうさすがにそんなことにはなりませんが)。仕様変更の前には, 「設計」とは一体なんなのか, 何を一生懸命やってきたのかと無力感に襲われます (もうさすがにそんな風にはなりませんが)。

私の経験で, 30 ページほどの図解で機能のみ示してある CAD の仕様書をプログラミングすると 50 本 900KB 位のソースコードになりました。当然それも不具合以外に仕様落ち, 仕様変更等あって, そのたびに「設計」をしてコーディング・テスト・デバッグするわけですから, 莫大に時間を食い潰していくわけです。

そうすると, 過激かもしれないのですが, 身を入れて設計せずともしょせんは仕様変更の憂き目にあうのだから, 身を入れない分, 成果物へのこだわりがなく仕様変更も楽にできて, 精神衛生上いいんじゃないかと思ったりするわけです。

3 「設計」をまなぶ必要

ではなぜ, 設計を学ぶためのワークショップに参加するかといえば, 生産性を向上したいという問題の切り口のひとつとしての, 「設計」を吸収したいという「渇望」です。PC プログラマというポジションに身を置いているからか, 「設計」などせずドカッとコーディングというほうが生産性があがるんじゃないか, という意見も多いのも事実です。また確かに, そんなことをしなくてもできる規模のソフトウエアも多いのかも知れません。

しかし, 成果物の規模が大きくなってくると, 「それでいいのか?」思ってしまうのです。「基本の考え方」「定石」というのを知らなくてはいけないのではないかと自問してしまいます。

コンピュータスクールでは, 設計しなさいと教えますが, 実習規模が小さかったりして, 設計なんかしなくてもできあがったりしてしまい, 結果, 学生に徹底しないのだと思います。

また, 「設計」を残しておかないと, ソースコード以外に資産が会社に残らず, 個人のみにノウハウが蓄積するという現象もおこっているわけです。中小企業はどこも同じなんですかね?

4 「改善の提案」

というようなわけですから, 「改善の提案」は「まずは, 管理職が勉強します」ということになります。勉強をして「やっぱりいいや」になるか, 「だめだ, 設計論採りいれよう」になるかは, 現在のところではよくわかりません。

問題になるのは, 「設計」をとりいれることで, どれだけ生産性が向上するか, どのくらいの対費用効果があるのかという, 経営的な根拠のあるアピールができるかということにつきるでしょう。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

分析設計の方法論の習得.

(Q2) 期待は満たされたか?

満たされた.

(Q3) 設計に意識の変化はあった?

変わった. お気軽でやってみるところに価値がある. わかる部分からとりかかる.

(Q4-1) 他人とのコミュニケーションについて

CRC カードというものを中心にストーリーを作成することでコミュニケーションが『発生せざるを得ない』状況におちいるのは驚きにあたいする.

(Q4-2) 短時間での, 問題の理解と作業について

1) とも関連するが, ある方法を導入することで容易に解決するのだということ.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には: 実験的にならCRC カードの分析, オブジェクト指向言語のによるインプリメントを小さいシステムでなら行なってみたい.

(Q6) 全体討論の内容に関して

ある意味で CRC を全グループに試していたら, もっと面

白い結果になったかもしれない (SA/SD と CRC の比較という意味で弱かったのでは)

(Q7) **その他, W.S. に参加して良かったこと**

良かった. 本当にありがとうございました.

(Q8) **W.S. の悪かったところ**

ご飯が少なかった. 贅沢言ってすみません.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

いちばん 知りたいことは満たされたので, 今のところはありません.

25 **吉田 恵** (よしだ めぐみ)
性別：女　年齢：24　血液型：O
会社：(株) さくらケーシーエス
所属：オープンシステム部 システム第二課

I. プロフィール

(1) **現在までの主な仕事の内容について**

- '90/6 社内ソフトウェア情報 DB 化
- '90/7-9 見積作業支援システム
- '90/10- 営業支援システム

(2) **現在利用している開発環境について**

- Macintosh IIcx
  - Hyper Card(Hyper Talk)
  - Wing2(Hyper Script)

(3) **これまでシステム設計に採り入れてきた技法など**

幸いなことに, 入社後丸 2 年間, 設計者に恵まれ, あまりこれといった苦労もなく過ごしてきましたので, 特にありません.

II. ポジションペーパ

(4-a) **設計に感じた「ありがたみ」について**

私立の女子大の文学部を卒業し, ワープロをホンの少し触っていたくらいで, 他は全く何も知らずにほとんど興味本位でこの世界に入った。私が入社して, 約 2 年半がたとうとしている今, 仕事もプライベートも充実した日々を送りたいとかねがね思っているだけで, 未だ実行できずにいる。

ここではプライベート面は抜きにして, 仕事の方はどうかというと, 現在ある企業との共同開発を進めているため, ずっと同じ相手と同じ系列のシステムの開発を行なっている。商売相手 (関わってきたユーザ) の少なさでは, 部内でも下から数えて何番目だろうか...

入社して 3 年目といっても, 新人研修期間が長かったため, 実務経験はちょうど 2 年と 2 カ月ぐらいなので, このワークショップにおける参加資格を, ぎりぎりクリアしている状態である。

さて, そんな私の乏しい開発作業経験から今回の題について述べたいと思う。最初にはっきり言えることが 1 つある。それは, 「私には設計をしたという経験がない」と言うこと。そして, 私のようなものが設計に関わらなくても, 社内には優れた設計者が大勢いて, 私達はこれといった苦労もなく開発作業に携わっていられるのである。そういう意味では設計にありがたみを感じていると言える。しかし, 設計の経験もなく今回参加資格があるのかとも思ったが, 今はそういう状態であっても, これから先もしも自分が設計を行なっていく立場になった場合に, 今回参加して吸収したものを充分に生かした設計を行なうよう心掛けたいと思っている。であるから, そういう視点でしか書いていないこと, また, 「ありがたみ」とも「うらめしさ」ともどちらかに絞られた文面ではないことを重ねて始めにおことわりいたします。

まず, 配属されたのが 6 月で, 1 カ月間は OJT(On the Job Trainning) があり, のうのうと過ごしていた。実務についたのは 7 月で, それから 3 カ月間 PC(FM-R) と共に, 大変な思いをして C 言語と付き合った。その頃は, 女性上司のもと (バリバリ仕事をこなすミセスキャリアウーマン) で, 設計書, 仕様書共に完璧に近いものが手渡され, 私達は, ただ仕様どうりにプログラミングするだけだった。初めての実務でそういう経験をしたので, 全てはそうして進められていくものだという認識を持った。結局, 設計に関してはほとんど修得しないまま 3 カ月が終わった。というより, とてもそこにまで至らなかったのである。

次は, 現在進行形の開発中のシステムに加わった。同じ PC でも, FM-R と Mac の違いに大いに驚いた。'小学生になろう' が実にぴったりだった。

ここでも, システム設計に関しては既に完全に終わっている状態であったので, PD 作業から入ったが, 初めての実務で持った私の認識は覆された。今まで仕様書など他人から渡されたことのない人との作業だったので, 設計書なし, 仕様書なしで, 口頭で" これだけの機能を持つプログラム" といわれ, 自分で考えることになった。机上で一通りの流れを考え, みてもらったあと PG 作業にはいる。2〜3 本後には全て一人で作業する。要領が解かってきたころ, だんだんと楽しいと思うようになってきた。システムが新聞, 雑誌に取りあげられるとどこかしら嬉しく思った。

しかし, やはり設計に関しては, 縁遠いものであり, 得たものはほとんど何もない。

インタプリタ型言語を使用していたこともあり, 設計書

を作成する間にPG 作業がほぼ見え, また, 当初予定ではドキュメント不要ということであったので, 設計書作成や, 仕様変更に伴う設計書変更などに関わっている時間の短縮で, 開発効率は大幅に Up していた。そして今, 話が一転し, ドキュメント作成が必要になったり, メンテナンス面での苦労の中でつくづく実感している。'設計は充分に行ない, 最新のものを形に残すべきである' と...

III. W.S. 終了後の感想

(Q1) **期待していたことは?**

設計作業というものに, 実際に触れることによる現場につながる成果.

(Q2) **期待は満たされたか?**

設計作業というものがかなり理解できた. 現場に戻ってからの利用度はかなり高いものがあると思うし, 使ってみたいと思う.

(Q3) **設計に意識の変化はあった?**

"設計" という 言葉に対するイメージは "難しいもの" であったが, 自分の中で, ある程度できているのではないかということ.

(Q4-1) **他人とのコミュニケーションについて**

むずかしい面は確かにあるが, 自分というものを素直に出していくことにより, 自分も他人も理解できるようになる.

(Q4-2) **短時間での, 問題の理解と作業について**

はじめは他人の集まりということでよそよそしい面が見られるが, 目標が明確にな ると, それに向かってみんなで進んでいこうとする.

(Q5) **経験は, 自分の仕事に対してどのような位置づけ?**

■自分の仕事の進め方に自信が持てた.

具体的には : CRC 記述法というものに対する理解が深まることにより, 今まで自分が進めていた仕事にある程度近いものであることがわかった.

(Q6) **全体討論の内容に関して**

・最終発表で何が言いたいのかを押えていなければ討論の論点からは外されてしまう.

・自分達の成果に対 しての評価を G4 グループとの比較の中で, 全体的に得られたことは意義のあることだと思う (塩谷さんの同意も得られた)

(Q7) **その他, W.S. に参加して良かったこと**

もっと苦しいものを想像していたが, そうではなかった.

(Q8) **W.S. の悪かったところ**

・自分のグループのことで手一杯になって, 他のグループのメンバとの交流の場が少なかったように思う.

・全体討論の場で, 今なぜそんな論議をしなければならなかったのか理解できない部分があった. (レベルのひくい論議をしている時間がかなりあった)

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

・人数的にこれよりも多くならない方がよいと思う.

・今回参加できたことは大変有意義だったと思う (あんなサマツなポジションペーパを拾っていただいた PC の方に感謝します)

04 **渡辺 清文** (わたなべ きよふみ)
性別 : 男　年齢 : 24　血液型 : O
会社 : オリンパス光学工業 (株)
所属 : システム開発部

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 顕微鏡画像伝送システムのプログラミング・設計。

(2) **現在利用している開発環境について**

- PC-9801DA
- MS-DOS 3.3D
- HD 100MB
- Turbo C++ 1.0 (C コンパイラとして使用)

(3) **これまでシステム設計に採り入れてきた技法など**

- 構造化プログラミング
- 簡単なイベント駆動

II. ポジションペーパ

(4-b) **設計に感じた「うらめしさ」について**

1 自分の立場

私は (1) に書いたように, 顕微鏡画像伝送システムのプログラミング・設計を担当しています。システム自体が実験段階であり, ユーザーも我々もどのようなシステムにすれば良いのかを模索中です。実際に使っていただいて, ユーザーと打ち合わせしながらの徐々に仕様変更という形をとっています。よって, 今のところ割合いい加減な進め方でも何とかなります (というと言い方が悪いですが)。

私の担当ですが, 仕様書にまとめるのは以前は直接の上司がやっており, プログラムの詳細設計は私がつくり, 自分ともう一人のものでコーディングという形でした。最近は機能仕様書も私が作成し, それを元にレビューで修正するわけです。プログラム作成以外の仕事が多いため, プログラム担当者が自分でやる方が効率が良いのでこうなりました。よって, 仕様書を受け取る側と, 作成する側のふたつの面か

ら「うらめしさ」を語ることができます。

- 機能仕様書を渡されたとき

  紙にプリントされて渡されたものは，再利用できない。要するに，テキストファイル化されていない。人がワープロソフトで書いたものはエディタ上では利用できません。もちろんワープロ文書をフロッピーでもらい，テキストファイルにコンバートすれば良いのですが，常にそこまでやるほどの必要性も実はないので，書類だけ受け取っています。

- 機能仕様書の設計上の問題

  さて，自分で仕様書を書くときは全てエディタで書きます。その方が速く書けますし，テキストファイルの方があとで利用しやすいからです。しかしこれにも問題はあって，グラフィックをドキュメントに混在させることはできません。文章だけでは説明が不十分ですし，画面設計やPADなどのチャートはどうしても必要でしょう。

- 詳細設計・プログラム仕様のレベルでの問題

  自分でプログラムを書いているので，頭の中でわかっていれば良いからと，詳細設計レベルのドキュメントが，ついついおろそかになってしまいます。ドキュメントを苦労してつくるよりも，プログラムを書いた方が速いというのもい事実です。ただし一人で全てのプログラミングをやっているわけではないので，結局問題が発生しやすくなります。

- 理想の仕様書・ドキュメントとは。理想の開発環境とは何か。

  仕様書はもちろん相手にその仕様を伝えるのが目的ですから，「必要な情報が過不足なくはいっている」というのが理想であり，必要な条件でしょう。ここで言いたいのは，そういう意味ではなく，開発中に参照しやすいという意味です。それは「開発中にコンピュータ上にいつでも呼び出せて，書類としても同じものが準備されている」という状況ではないかと私は考えています。他の世界では知りませんが，98上ではこれは非常にむずかしいと思います。むろん，仕様書をしっかりしたものにするためにはハード・ソフトの環境よりも，標準化やグループとしての問題の方が大きいでしょう。しかし環境の問題は消してばかにはできないと思います。手書きで仕様書を書いていたりしたら，開発効率は確実に下がりますから。

- 設計しやすく，開発し安い98上での環境

  MACとの併用

  最近マッキントッシュが入りました。きれいな仕様書はこれで書けます。この際モニター上で見たいという問題はあきらめます。 その他

  なんとかします。

- 結論

  一番大事なのはしっかりと設計して，周りのメンバーに正しく伝えるということでしょう。チームワークが一番大切ではないでしょうか。

III. W.S.終了後の感想

(Q1) 期待していたことは?

(1) 外部の人との交流

(2) オブジェクト指向への理解を深める

(3) 仕事の進め方を今すぐ改善するアイデアを得る

(Q2) 期待は満たされたか?

(1) ～ (3) とも満たされた．オブジェクト指向に関しては，単なる勉強でなく，実際に使ってみようという気になった．

(Q3) 設計に意識の変化はあった?

自分の 設計方法が OOA/OOD 的な面を持っていることが確認された．

(Q4-1) 他人とのコミュニケーションについて

普段の仕事でも，関係者一同を集めたディスカッションはどんどんやるべきだと思った．

(Q4-2) 短時間での，問題の理解と作業について

とりあえず始めてしまうのが早い．やってしまったもの勝ちだ．

(Q5) 経験は，自分の仕事に対してどのような位置づけ?

■自分の仕事の進め方に自信が持てた．

■今後の 仕事の進め方に対して，新しい視点が持てるようになった．

(Q6) 全体討論の内容に関して

議論がかみあっていないのが残念．

他のグループが何故そんなに一所懸命分析を細かくやっているのが不思議でした．

(Q7) その他，W.S. に参加して良かったこと

名古屋の名物料理を食べた．どうも名古屋名物は怪しい．

(Q8) W.S. の悪かったところ

暑くて遊べなかった．

他のグループのやっていることが，良くわからなかった．

(Q9) 今後 W.S. に取り上げたいテーマ，全体的感想

マルチメディア関連

OOA/D/P とからめて

13 伊藤 徹 (いとう とおる)
性別：男　年齢：29　血液型：O
会社：アドバンスト・ソフトウェア・インスティチュート
所属：

I. プロフィール

(1) 現在までの主な仕事の内容について

(株) 豊田自動織機製作所にて 織機の制御プログラムの開発を主として行なってきました。現在は織機自身の制御プログラムでは無く, 織機の表示器のプログラムを作成しています。

(2) 現在利用している開発環境について

UNIX

(3) これまでシステム設計に採り入れてきた技法など

構造化設計 (プログラムの独立性を重視した設計)。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

現在私は派遣業務で豊田自動織機製作所に勤務しています。こちらに来て既でに 5 ～ 6 年経ち, 昨年の 10 月までは織機の制御系の設計, プログラミング及び, 出荷業務を担当していました。現在は, 同じ織機関係ではありますが, 情報表示器 (ファンクションパネルと呼んでいます) の担当を行なっています。織機と言うと, 今一つパットしない機械の様に思われますが, これがなかなかどうして, 結構凄いのです。

さて, ここの業務の流れとして, 新しい開発業務は, 開発設計課から依頼が来て, その仕様に基ずいてプログラム設計を行います。従って開発設計課が全体を見渡す SE 的な役割を果たしている訳ですが, 実際には織機を一番理解しているのはプログラム設計をしている我々で, SE 的な仕事もこちらがリードして進めていかなくてはならない場合も少なくありません。多くのソフトハウスの様に委託業務として, 自社の中で仕事を請け負う形で業務を行なっているプログラマ, SE の方には理解しにくいと思いますが, 仕様書自身も, 悪く言えば "いいかげん" な訳です。もっとも, すぐに連絡出来る位置にお互いがいるため, 口頭で疑問を定義, 確認出来るために仕様書の重要度が低下してしまっている部分も有りますが...。仕事を始めた頃にたびたび悩まされたのが, この仕様書の存在と実際に作成する (された) プログラムのギャップです。仕様書どうりにソフトを作成したのでは, 仕様書を作成した人の要求を満たしてい無い事が多いのです。もちろん中には難しい仕様で, 取り合えず "えいやー" でやってみなければ次のステップに進めないためにやってみる。と言った場合も有ります。しかし, 今までの経験から見ると, ほとんどの場合は仕様書を作成している人の頭の中の仕様が, 紙に表現されてい事が一番の原因と思われます。仕様書を作成する人も人間です, 文章能力のある人もいれば, 私の様に苦手な人もいます。例えば自分が逆の立場に置かれた場合を考えれば大変な仕事である事は一目瞭然です。

しばらくは仕様書にだまされながら, プログラムを作っては SE に確認してもらい, 仕様修正されては又作り直す。を繰り返す様な仕事の進め方をしていましたが (仕事の経験も無く, そう言う物だと思い込んでいた), 常に 1 つの仕事だけを受け持っていられるほど, 会社は甘くありません。2 つ 3 つの仕事が並行してプログラム作成の要求を求めて来る事も発生してくるのです。(訳のわからん雑務もあるし) そうした時に, それまでと同じ, 方法でプログラムを作っては壊し, 作っては壊しをやっていたのでは, とんでも無い工数を取られてしまい, 土日を潰して出勤しても間に合わない様な状態に陥ってしまったのです。ある時上司からこう言われました。"仕様書どうりに作るな, 仕様書を書いた人の思ったとうりの動きをするソフトを作れ。"と。言われてみれば当たり前の事です。今考えれば, 以前からそれに似た様な言葉を聞かされていた様にも思いますが, 人間自分が本当に追い詰められて, どうにもなら無い状態になってみないとその言葉の意味に気づか無い事も有るものです。

仕様書に目を通すだけで無く, 本人とよく話し合い, 何がしたいのか, どうしてそうしたいのかを, 正確に把握する必要が有るのです。時にはメカの構造にまで首を突っ込んで確認しなくてはならない場合も有ります。そうする事によって, SE の実現したい物の実像がハッキリ見えてくるのです。こちらが勘違いしていた部分が見えてくるのです。これをしっかりやっておくだけでその後の仕事の流れのスムーズさが全然違うのです。又, この作業のために費やされる工数は後でプログラムを作り直す工数に比べれば, 取るに足らない物です。

"石の上にも 3 年"とはよく言ったものです。この事に気づくのに 2～3 年かかりました (すいません)。最近は職場に慣れた事も有りますが, 話しを聞いてソフトの都合でやり易い様に仕様を修正してしまう時も有ります。もちろん SE の了解の基でですが。基本的には SE の意見, 要望を尊重しますが仕様の許す範囲であれば, なるべく単純なソフトにした方がバグも出にくく又出た場合にも解決し易いのです。

"設計に感じた「うらめしさ」について" と言う事でしたが, どちらかと言うと "設計手順のうらめしさ" になるかと

思いますが, 他に書く事が思い当たらなかったので, 御了承下さい。

III. W.S. 終了後の感想

(Q1) **期待していたことは?**

オブジェクト指向の意味が理解できるか?

(Q2) **期待は満たされたか?**

当然満たされた訳ではないが, 確認できた部分も多い.

(Q3) **設計に意識の変化はあった?**

皆でやるのもおつなもの.

(Q4-1) **他人とのコミュニケーションについて**

きらいじゃない.

(Q4-2) **短時間での, 問題の理解と作業について**

力業

(Q5) **経験は, 自分の仕事に対してどのような位置づけ?**

■自分の仕事の進め方に自信が持てた.

■ワークショップの議論と現実とはやはり違うと感じられた.

(Q6) **全体討論の内容に関して**

(記述なし)

(Q7) **その他, W.S. に参加して良かったこと**

予想以上に楽しかった (もっと固いかと思っていた)

(Q8) **W.S. の悪かったところ**

設計時に, もう少し PC の意見を聞きたかった.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

(記述なし)

05 **井上 忠則** (いのうえ ただのり)

性別：男　年齢：27　血液型：B

会社：(株) シスプラン

所属：技術部

I. プロフィール

(1) **現在までの主な仕事の内容について**

- MS-WINDOWS 上で動作する専用エディタの作成
- UNIX 上で動作する通信ソフト
- MS-DOS 上で動作する自動車生産管理システムのユーザーインターフェース部分
- Macintosh 上で動作する絵画データベースシステム

(2) **現在利用している開発環境について**

UNIX(NEWS) マシン 3 台, DOS マシン 6 台 + ノート型 2 台, Mac 1 台

(3) **これまでシステム設計に採り入れてきた技法など**

フローチャート, データフロー, HIPO 等を使ってモジュール分割を行なっていたが, 最近は小さなシステム, あるいはシステムの一部分を作成するというのが多いため, システム設計の技術・技法という大げさなものを上げることができない。

II. ポジションペーパ

(4-a) 設計に感じた「ありがたみ」について

ソフトウェア開発に携わるようになって, はやくも 5 年が過ぎました。この間に, 行ってきた仕事の内容は, (1) に上げたような制御系の C 言語によるソフトウェア開発のプロジェクトがほとんどでした。

プロジェクトの規模としては, 4 人で行ったプロジェクトが一番多人数で, あとは 2, 3 人でしか仕事を行った事がありません。

今までに経験した開発作業で, 感じた設計の「ありがたみ」ということですが, はっきりいって「ありがたみ」とかを, あまり強く感じたことはありません。

勉強としての設計技術・技法ということでは, システム開発 JSD 法 (M.Jackson 著) とか, 構造化分析とシステム仕様 (Tom DeMarco 著) 等を読んだり, 前前回の若手の会「Case study of CASE Tools」で, オブジェクト指向ソフトウェア開発 OOA, OOD, OOP (青木淳氏) に参加をしたりと, 知識としてはとりあえず蓄積してきているつもりなのですが, どうもうまく実作業に応用できていません。

その理由として, 少人数のプロジェクトで, まして, 小規模であるため, 高額な C ASE ツールを購入することは, 現実問題として許されないのです。もし購入してそのツールの設計技法に乗っ取って使うことができれば, 設計者は能率よく作業が進められるでしょうが, そして, 設計技法をマスターする前に, 作業に取り掛かった方が早く仕事が終わってしまうだろうというのも, CASE を導入できない原因の 1 つかもしれません。

また, オブジェクト指向などの設計方法を学び, プログラミングに応用できても, 規模が小さいため, 何か自己満足で終わってしまっているというのもあります。

今後, パワーのあるマシンで, 大きな仕事を行ってこそ, 「ありがたみ」というものが実感できるのでしょう。「ありがたみ」を感じられるような大きな仕事をいつでも行えるよう, 日頃から設計技法についてアンテナをはっておきたいと思っています。

III. W.S. 終了後の感想

(Q1) **期待していたことは?**

今後のソフトウェア設計に何かヒントとなるものが得られるのではないか.

(Q2) 期待は満たされたか?

得られた.

CRC の分析, 設計過程が体験できた.

(Q3) 設計に意識の変化はあった?

ソフトウェア設計をグループで行なった場合, 他のメンバーの意見が重要であるということが再認識させられた.

(Q4-1) 他人とのコミュニケーションについて

メンバーによって作業の進み方が異なる. コミュニケーションは重要である.

(Q4-2) 短時間での, 問題の理解と作業について

自分の意見をうまく他の人に説明できなかった.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■その他：今後の仕事に具体的にできるのではないかと思えた

(Q6) 全体討論の内容に関して

グループごとに異なる図書館問題の認識があり, 不要な議論があったと思う.

(Q7) その他, W.S. に参加して良かったこと

岡崎城に行けた.

半分しか食べられなかったが, 味噌煮こみうどんが食べれた.

(Q8) W.S. の悪かったところ

毎日酒びたりになっていた.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

楽しかった.

グループ 4

01 古閑 幸一 (こが こういち)
性別：男 年齢：27 血液型：A
会社：ヒラタ ソフトウェア テクノロジー
所属：システム開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

ここ 2, 3 年の内容です。

FA 生産管理システム開発

某 M 社の生産システムの開発を行った。そこで私は人質という捕らわれの身にとなった。まぁ, おもに人質が担当 (?) だったが, その他に生産計画, 実績に関するシステム設計～インストール迄を行っている。この経験で品質の大事さを嫌という程教えられた。

PLC 開発ツールの移植

某 M 社 (上の M 社とは違う) が自社パソコン用に開発した PLC 開発ツールをワークステーション上に X-Window で動作するように移植した。私は, 移植元のソースの解析, X-Window に関する調査, 移植時の方針決定～実際のインストールまでほぼ全体の作業を経験した。(但し, プロジェクト管理はやっていない) このプロジェクトで X-Window に関する (特に Toolkit) 知識をある程度得る事ができた。

大規模ビル管理システム開発

某 M 社のビル管理システムの一部を某 K 社から受注し, 開発を行った。このプロジェクトには途中参加であったが, 私は主に GUI に関するアプリケーションの開発を行った。このプロジェクトは, 4 社での共同開発であったが, チーム開発の難しさを色々な観点 (技術, 商売) から再認識させられた。

(2) 現在利用している開発環境について

あまり触れて欲しくないが

OS

・Unix(4.2 BSD)

マシン

・News-830(news-os2.0 で使用)

ウィンドウ

・(ナント) X10

ネットワーク

・Ethernet + TCP/IP + NFS

・E-Mail

言語

・C + SDB

・GCC + GDB

であり, 客先から借りたマシンの環境の方が良いことが多い今日この頃。

(3) これまでシステム設計に採り入れてきた技法など

あまり突っ込んで欲しくないが

- データ・フロー・ダイアグラム
- E-R 図
- 状態遷移図
- オブジェクト指向設計

を取りいれたことがある。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

1 はじめに

私は, どうも人間的に不十分らしく, 色々なありがたみ (親のありがたみ, 空気のありがたみ, 人の親切のありがたみ, etc..) を適宜に感じることが少ない。逆に色々な「うらめしさ」を感じることは多々ある。こんなことでは, これからの長い人生 (人として, エンジニアとして) 有意義な時間を過ごせそうにないので「うらめしさ」を想起し, これか

らの「ありがたみ」につなげて行きたい。

まず何ゆえ設計は必要なのか (話が長くなるので) 簡単に整理し, それゆえに実際の開発作業の中で感じた「うらめしさ」を中心に書いていく。

要求されている仕様が明らかにされる間にも設計は行われと思うが, 今回の "若手の会" で扱う「設計」は要求仕様がある程度明らかになり, 要求仕様分析が終了した次から始まる (と思われる) プログラム構造に対する設計と考えている。以下ではプログラムの構造設計を単に設計と呼ぶ。

2 設計はなにゆえ必要か?

設計は要求仕様として出された, ある意味で架空的 (?) なプロダクトをプログラムとして具現化する為に必要な準備作業だとする。(準備作業といっても担う役割は大きいが) その為に私達はシステム全体の中での役割を最も大事にした上で

イ) 構成物 (機能) の抽出を行い

ロ) 構成物 (機能) を心地よく制御 (または管理) する仕組み

を苦労して考えているように思う。

a) その際, 要求仕様書は架空的なプロダクトの為 :-), 構成物の拡張性を重視しなければいけない。

b) また, 開発者と言ってもサラリーマンなので ;-), 高生産性を考えた内容を目指さなければならない。

c) さらに, 開発者と言っても人間なので ;-<, 仮にバグった時などのメンテナンス性についても考えなければならない。

3 設計に感じる「うらめしさ」

私が設計で感じる「うらめしさ」を想記すると, 自ずと上記を満たしていない設計がなされている時である。特に, モジュールの独立性が低い設計が行われていると「うらめしさ」を強く感じる。これは上記の a), c) に特に効いてくるからである。モジュールの独立性は使用するデータとそれらの操作が一つのパッケージとして考えられているかどうかによると思う。これを満たしているかどうかの一つの判断材料に

- グローバルな変数が存在するか?
- 一つの型が定義されると同時に, 操作方法も提供されているか?

が考えられると思う。もう少し, 型と操作のパッケージ化を意識した設計を行うことを提案する。その為に, イ) の作業時に適切なモジュール (クラス) の抽出を行うことが非常に大事だと感じている。これを行うにあたって, オブジェクト指向分析方法等を使ってデータの関係を分析することが有効な手段ではないかと感じているが, これは現在の私の課題でもある。

次に「うらめしさ」を感じるのは 99 全く別のモジュールを作成しなければならないような設計が行われているときである。これは a), c) にも効くが, 特に b) に効いてくるからである。改善提案としては, ソースの再利用という考え方から, 前述した型と操作のパッケージ物を取り込むという考え方にシフトすればよいと思う。より良いのはこれらの仕組みがある環境を開発環境として揃えることだが, 現実はなかなか厳しい。

以上, 私が感じる「うらめしさ」を簡単に記述した。実を言うと私が日頃感じていた「うらめしさ」は全て,

『オブジェクト指向入門』 ASCII 出版局

の前半 100 ページまでに整理されて書かれている。

現場で私は C 言語を使って開発を行っている。その中で私は設計時になるべくオブジェクト指向的な設計を行う様心がけている。オブジェクト指向分析を実践して感じたのは, 混沌としている要求仕様から一つのデータを抽出し, リレーションを張って一つの型を定義することの難しさであった。

『オブジェクト指向分析入門』 啓学出版

を片手にやってみたが, 結局, 型の定義が不適切で使用できないものであったりした。

今回の "若手の会" は設計について実践的な作業を取り入れた形で討論するようであり, 日頃私が抱えている問題点になんらかの光を与えてくれそうな場である。

若い人が集う場である。大いに議論し, 盛り上がり, 何かしらの新しいものが生まれてくることを期待し, 参加できることを切に希望する。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) オブジェクト指向に対する疑問に対する解答

(2) 技術者間の交流

(Q2) 期待は満たされたか?

(2) についいてはある程度満たされた.

(1) については質問していないため, 満たされたとは言えない.

(Q3) 設計に意識の変化はあった?

設計にブレーンストーミングがどの位有効かという事を再度考え直す必要があると感じた. 今後の設計時に大事にしたい.

(Q4-1) 他人とのコミュニケーションについて

言いたい事を言うのが大事で, 言いたい事を言ったら, 聞来たくない事を聞く事が大事だと感じた.

(Q4-2) 短時間での，問題の理解と作業について

ディスカッションして意見を出しつくし，共通のイメージを持つために視覚化する事は大事だと感じた．

(Q5) 経験は，自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して，新しい視点が持てるようになった．

具体的には：

役割という見方でクラスを探す．

(Q6) 全体討論の内容に関して

もう少し全体討議で作業の進め方自体について話したかった．

(Q7) その他，W.S. に参加して良かったこと

知らない人とのコミュニケーションの難しさと，やり方の少しのコツがわかった．

(Q8) W.S. の悪かったところ

時間で切って，深夜作業をやらない様にすれば良かった．

(Q9) 今後 W.S. に取り上げたいテーマ，全体的感想

(記入なし)

03 **岡田 安彦**　(おかだ やすひこ)

性別：男　年齢：24　血液型：AB

会社：オリンパス光学工業 (株)

所属：システム開発部 開発 1G

I. プロフィール

(1) 現在までの主な仕事の内容について

栄養診断システムの開発

(2) 現在利用している開発環境について

機種 98 ノート

言語 C

OS MS-DOS

(3) これまでシステム設計に採り入れてきた技法など

特になし。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

栄養診断システムが私がいままで開発したシステムである。このシステムは 3 日間の食事内容と一日の生活パターンを入力として，栄養の摂取量やバランス，エネルギーの摂取量や消費量を計算し，受診者の栄養診断を行うものである。

このシステムはある栄養診断の受注をしている会社からの依頼で開発することになった。その会社で使っているシステムに不便なところがあるので，その改良版を作ってほしいという依頼だった。

開発チームは三人で，私は詳細設計や，プログラミングを主に担当した。

このシステムの設計に当たって考えたことは次のようなことである。

1. 従来のシステムよりも入力の効率を上げる

   入力の効率を上げるために，食品の入力の方法を数種類用意した。特に料理ごとの入力ができるようになったことで入力の効率が大幅に改善された。また，生活パターンの入力では入力画面に工夫を加えることによって入力の効率を上げることができた。
2. 受診者に渡す判定表を一般の人にも分かりやすく，親しみやすいものにする。

   判定表の印刷には，カラーでイラストなどがプレ印刷された専用の用紙を使用した。診断結果の表示には，グラフやレーダーチャートを使って，一目で分かるようにした。また，診断コメントを自動的に出すようにした。
3. そのほかの部分に付いては従来のシステムどおりとし，開発コストをおさえ，開発期間を短くする。

   結果的に開発期間を短くすることはできなかった。

この設計に「うらめしさ」を感じることは，ユーザーインタフェイスの部分で設計時に考えてたより，使いにくいところできてしまったことである。たとえば一覧表から食品を入力するのが思ったより面倒だったり，ファンクションキーの使い方がわかりにくかったことなどである。

それに対する改善策としては，自分が実際操作している姿をよく想像してユーザーインタフェイスの設計をするべきだ。また，市販されているソフトを参考にするのもいいと思う。そのほか，共通ルーチン化が完全にできなかったことも「うらめしさ」を感じることである。

設計以外の開発作業では次のことに「うらめしく」感じた。

1. テストの不完全さ

   テストが不十分なままユーザーに渡してしまったため，そのあと延々とバグ対応に追われることになってしまい，開発期間が大幅に延びてしまった。
2. 開発計画の不完全さ

   コーディングに時間がかかってしまいテストに費やす期間がほとんどなくなってしまった。

改善の案としては，

- 組織的なテスト計画をたてる
- デバッガーを使う

などが考えられる。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) オブジェクト指向分析/設計とは何か?

(2) 設計した手法を用いるとどうなるか (何も手法を用いないときに比べて)

(Q2) 期待は満たされたか?

(1) オブジェクト指向についての説明はほとんどなかったが, 設計を進めるにつれてだんだん分かりかけてきた.

(2) シュミレートがしやすいと思った.

(Q3) 設計に意識の変化はあった?

設計について特に手法を使ったことがなかったが, 今後機会があれば使ってみたいと思う.

(Q4-1) 他人とのコミュニケーションについて

自分の知識不足から他の人の発言の意味がわからないことが多かった.

(Q4-2) 短時間での, 問題の理解と作業について

議論がわき道にそれた時や, 無限ループに入った時に, それを元に戻す人が必要だと思った.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) 全体討論の内容に関して

各々のグループ発表の時間を長くして欲しかった.

(Q7) その他, W.S. に参加して良かったこと

もっと勉強しなければならないと思えた.

(Q8) W.S. の悪かったところ

予習が不足だった.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

オブジェクト指向についてのセミナーを開いて欲しい.

10 **森下 葉子** (もりした ようこ)
性別：女　年齢：27　血液型：O
会社：(株)SRA
所属：先端技術開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

今までの主な仕事は 大きく分けて「社内ツールの開発」, 「ソフトウエアの品質管理およびトラブルサポート」の 2 つです。それぞれの内容は下記のとおりです。

- 社内ツールの開発
  社内 (または部内) で必要なツールを開発。
  主に X-Window 上で稼働する GUI 部分を作成。
- ソフトウエアの品質管理およびトラブルサポート
  商品としてリリースするソフトウエアの品質評価。(品質テストは 基本機能テストではなくて複合機能テスト)
  リリース後のトラブルサポート。(ソフトの不具合管理と, 顧客のトラブルに対応する社内 SE <System Engineer> のサポート)

(2) 現在利用している開発環境について

マシン (OS) SONY NEWS マシン (UNIX)
共通に使うコマンドと各自のデータは, 別のマシンにおいて一括管理しています。
共通コマンド, データは, 自分のマシンから NFS でマウントして使っています。
X-Window 上で NEmacs を使用。

プログラム言語 C 言語 (最近 B-シェルも少し覚えました)

デバッガ dbx

エディタ NEmacs の C-mode

ソース管理 RCS

(3) これまでシステム設計に採り入れてきた技法など

「〜法」のように名前がついた技術・技法を使って設計をしたことはありません。(もしかすると名前は知らなくて使っていたかもしれませんが) 今までの設計手順としては, システム設計とプログラム設計と同時におこなった, 次のようなパターンになります。

<パターン>

1. 必要とする機能をもとに外部仕様を決める
2. 外部仕様書 (Window のレイアウトのような外部デザイン) を作る
   ※ この段階で 「ボタンを選択したら 内部処理で〜を実行する」といった処理を考えます
3. 外部仕様書をもとにレビューをおこなう
4. プロトタイプを作り, 機能の実装が可能かの確認をおこなう
5. 外部仕様が決まったら コーディングを始め, つくりながら内部処理の詳細を決める

※ 4. のプロトタイプは作らない場合があります

II. ポジションペーパ

今までの開発作業のほとんどは個人ベースの開発作業なので, 商品開発のような理想的な設計作業を経験したことはありません。また, , 特別な設計技法を使ったことはないの

ですが, ソフトウエアのライフサイクルとして PDCA サイクル

P … 計画 (Plan)
D … 製造 (Do)
C … 検査 (Check)
A … 運用 (Action)

の概念が使われるとすれば, 個人ベースの開発では P(Plan) ～ C(Check) までを自分で担当しました。

「～の機能があるものがほしい」と要求があってから (この機能要求は自分で考える場合が多いですが), 最初に操作状況をイメージし, 外部設計からとりかかります。

私の場合は X Window 上の GUI の開発をいくつか行ないました。その時の設計は次のような手順です。

1. 必要とする機能をもとに外部仕様を決める
2. 外部仕様書 (Window のレイアウトのような外部デザイン) を作る
   ※ この段階で 「ボタンを選択したら 内部処理で～を実行する」といった処理を考えます
3. 外部仕様書をもとにレビューをおこなう
4. プロトタイプを作り, 機能の実装が可能かの確認をおこなう
5. 外部仕様が決まったら コーディングを始め, つくりながら内部処理の詳細を決める

外部設計は Window 等の画面レイアウトを考えるところからはじまりました。

まず, 「これらのボタンや結果を表示する出力表示 Window があれば, この機能要求を満たすことができる」というように, 機能を満たすのに必要な部品を考えます。

必要な部品が揃えば, あとはその部品を組み合わせて Window Layout を決めていけばいいのですが, 自分が欲しい機能をもった既存の部品が用意されているかが問題になりました。

XView や Motif のように既存の部品をあらかじめ調べて, 必要な機能をあてはめていく方法をとれば, 高機能で短期間に開発できたのですが, どちらも勉強から始めなければいけなく, また, ツールキットが動かないマシンで稼働させる必要もあったため, 1 つ 1 つの部品を Xlib で自分でつくることになりました。

Xlib を使ってつくるといっても, 大きなプログラムをつくるのははじめてだったので, 必要とする機能を本当に実装できるかわからず, 確認のためにプロトタイプをつくることからはじめました。

このプロトタイプをつくることが, 設計段階での「ありがたみ」でもあり「うらめしさ」でもありました。

「ありがたみ」としては, プロトタイプをつくることにより, 実現できるか懸念があった要求機能の実装確認ができたことです。また, プロトタイプとはいえ動作確認ができるので, 外部仕様書よりも完成状態をイメージすることができて, あと何を追加すればいいかの項目抽出と, 開発期間の見積りがしやすくなりました。

「ありがたみ」の例で出した"完成状態がイメージできる"の反面, 「うらめしさ」としては, プロトタイプの作成に時間がかかったこと, また, 次のように機能要求が増えるという場合がありました。

外部仕様書 (この場合は Window レイアウト) をもとに機能要求者達と仕様のレビュー済みなのですが, プロトタイプをもとにレビューをおこなうと, 完成状態をイメージしやすいためか, "この機能も欲しい"というように, 新たな機能要求が出てくるのです。

プロトタイプをつくることにより P(Plan) → D(Do) へライフサイクルの流れが動いていたものが, この新たな機能要求の検討をすることにより, P(Plan) ← D(Do) と要求分析まで戻ることになりました。

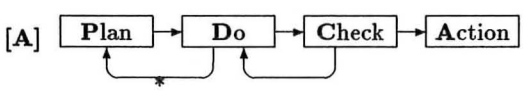
ここでまた最初の開発過程と同じように, 要求分析からはじまり,

1. 要求機能をもとに外部仕様を決める
2. プロトタイプの作成

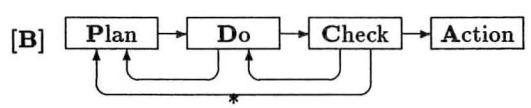
を行ないました。

P(Plan) – D(Do) の間の繰り返しは 1 回だけだったので, 大幅にスケジュールに変更はでませんでしたが, 全体のスケジュールを見積りなおすという作業が加わりました。

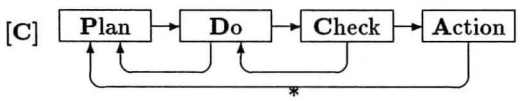
今回例であげた開発過程は,
プロトタイプをつくったあとでの機能の見直し
テストの時に Bug を見つけて修正
という前工程への 2 回のフィードバックを含んだ
[A] の開発サイクルパターンになりました。(* … 機能の見直し)



今回の場合は D(Do) の段階での機能の見直しでしたが, 設計が不十分だと, D(Do) の次の工程 C(Check) の段階で機能不備が見つかり, P(Plan) にフィードバックする [B] のパターンが考えられます。



最悪の場合を考えると, 最終段階の運用段階 A(Action) で機能不備が見つかり, [C] のようにつくりなおしになる可能性もないとは言えません。



このように見ると, 最初の段階の設計を十分に行なわないと, そのあと全ての開発作業に影響を与えると感じました。

設計段階でプロトタイプをつくりながら開発していく方法は, 前述したようによい点もあり, 設計方法の 1 つの手段だと思います。

ただ, プロトタイプをつくる時期と, どこまでの機能をもったプロトタイプをつくるか, また, どのような環境でプロトタイプをつくるかを検討しないと, 開発作業に悪影響を及ぼしかねません。

機能実装を確認したいだけなのに, プロトタイプをつくる開発環境の選択を誤ると, プロトタイプをつくるだけで時間がかかってしまいます。もしもプロトタイプの結果が期待はずれなものだったら, 大幅な時間のロスになります。

「改善の提案」としては, 機能要求段階から考えて,

- 機能要求の分析を正確におこなう
- 機能を満たすためにどのような技法をつかうか十分に検討する

を行なえる作業環境で設計をおこなうと, 効率があがると思います。

具体的には, 機能要求にそった全体の流れをモジュール別に図等で描いていき, その図をもとに自動的にできるプロトタイプで機能確認をしながら詳細設計ができるような環境をイメージします。(Window ベースの GUI の開発であれば, 部品を基本 Window にコピーして, 内部処理だけ記述すれば動作確認ができるツール環境のイメージです)

また, 効率のよい設計技法を使うこと以外に, 私がよく忘れてしまうことなのですが, 開発するソフトのモジュール全てを新しく作るのではなく, 今まで作ったもの等, 既存の処理の組み合わせの検討も設計段階で必要だと思います。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

・設計手法を使ってみる.

・設計手法を使って効果があるか/ないか.

(Q2) 期待は満たされたか?

・設計手法を使う事に関しては, 今回 CRC を使うことで満たされた.

・設計手法に効果があったかどうかは開発を行なう 1 つのプロセスとして意義はあると思うが, 効果があったかどうかは疑問です.

(Q3) 設計に意識の変化はあった?

設計に対するエンジニアの取り組み方と顧客サイドとのかかわりが重大だと実感しました.

(Q4-1) 他人とのコミュニケーションについて

バックボーンが違う人同士で, 言葉の表現が難しかったです.

(Q4-2) 短時間での, 問題の理解と作業について

作業の役割分担をうまく行なわないと進まないと思いました.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■ワークショップの議論と現実とはやはり違うと感じられた.

具体的には： 現実は実装条件がもっとシビアでは?

(Q6) 全体討論の内容に関して

他のグループの作業内容が発表内容だけからでは理解しにくく, 比較が難しかったです.

(Q7) その他, W.S. に参加して良かったこと

いろいろな会社の人と話せて良かった.

(Q8) W.S. の悪かったところ

グループ内では討論の時に話せましたが, 今回参加した方全員とコミュニケーションをとる時間がなかったです.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

インプリ後のデバッグ法等.

19 **工藤 聡** (くどう さとし)
性別：男　年齢：26　血液型：A
会社：(株) 岩手電子計算センター
所属：行政システム部 行政第 4 課

I. プロフィール

(1) 現在までの主な仕事の内容について

- 社会保険庁オンラインシステムの業務システムの徴収関係の開発。期間：3 年。主として詳細設計～総合テストを担当
- 安田信託銀行短期 ALM システムの開発。期間：2 年。要求分析～運用テストを担当
- 岩手県財務会計システムの開発。期間：3 年。要求分析～運用を担当
- 森林資源管理システムの開発。期間：2 年 (予定)。要求分析～

(2) 現在利用している開発環境について

- 大型機
  日立の汎用機 (M660 シリーズ)。

VOS3,EAGLE2,XDM

- WS/PC

  日立 2050。SEWB。

  Apple Macintosh

(3) **これまでシステム設計に採り入れてきた技法など**

- CASE

  大型機用の日立の下流 CASE ツール EAGLE2 を導入。また, 開発方法論として同社の HIPACE を採用。ワークステーション用の CASE ツール SEWB を導入。データフローダイヤグラムなどの作成に利用。標準的なプログラムのパターンを用意し利用。

- 方法論

  システム開発の方法論として, 構造化設計, データ中心アプローチを採用。構造化設計はプログラムの設計に, データ中心アプローチはデータベースの設計に適用。また, データ中心アプローチをサポートするシステムとして Macintosh の ERAS を利用。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

1 あるオンラインシステムの開発

オンラインシステムの開発においてのことである。このシステムはエンドユーザが直接端末機を操作するもので, ある種の情報の登録, 情報の検索とその情報を用いたシミュレーション, シミュレーション結果の登録, 各種統計情報の出力などがメインであった。システムの位置づけとしては, 基幹業務システムを構成するサブシステムである (私の担当はその中の 1 サブシステム (サブ-サブシステム?) であった)。動作/開発環境は大型汎用機, 端末はパソコンベースのダム端末, DBMS としては構造型データベースを利用するものであった。CASE ツールは使用しない。開発期間はおよそ 2 年で, 私は 1 年半にわたって開発に携わった。私はこのシステム開発における設計に, 最初は「うらめしさ」を, 後半では「ありがたみ」を感じた。

2 要求定義のミス

エンドユーザはシステムの性能として, 「速い分にはいくら速くてもよい, ただし遅くても 3 秒以内」というレスポンスを要求, これ以下のレスポンスでは使い物にならないということであった。1 秒を争うような業務で使用するシステムに対する要求としては, 妥当なものであろう。しかしながら, 要求定義の段階では 3 秒という具体的数値は記述されず, 「できるだけ早く」というなんとも曖昧な表現となった。今思えば, ユーザの要求を無視したともいえるこの要求仕様書に要求分析, システム要件決定への認識の甘さが現われていた。結果的にはこの要求定義仕様書が, 設計に対して「うらめしさ」を感じさせる諸悪の根源となった。

3 設計のミス

曖昧な要求仕様書をベースに行なった基本設計, 機能設計は自らその内容が制限される結果となった。システムの仕様, 機能などを明確に定義するべき基本/機能設計は, その役割を果たさなかったといえる。たとえば機能設計では, もとめられる機能を一つの処理単位に分割し, それぞれの処理単位の機能の詳細を定義する必要があった。しかし, 実際に仕様書に記述された内容は, その時点で判っていたことの半分にも満たないものであった。プログラムあるいは処理単位の機能を記述すべき機能設計書に, 機能が明確に記述されておらず, 次のステップに進んでも, 仕様を見直さないと作業ができないものも多数存在した。たとえば, 実現すべき機能については記述があるが, 機能の実行条件が明確ではない, など, そのままでは使用に耐えられないというものがあった。ここでのミスは, これ以降の作業で役に立つ (かも知れない) 情報を, 設計書あるいは仕様書という「モノ」で残さなかったことにあると判断する。また, 設計書についてものその重要性が認識されておらず, 「こんなものでいいだろう」という安易な発想があったことは否めない。また, 構造化設計も名ばかりで, このシステムの設計段階での私の認識は, 「構造化設計/プログラミング=とにかくモジュールを分ける」ということであった。

4 うらめしい

このまま設計, 製造を進めていたならば, このシステム開発は失敗, 設計は「設計」と呼べるものではなかったと言える。まさしく設計に感じた「うらめしさ」である。しかし, ここで要求仕様のまずさがクローズアップされ, エンドユーザの強い要望により, 是が非でも「3 秒の壁」を突破しなければならなくなった。データベース構造, プログラムの作り, どれをとってみてもユーザの要求に応えられず, また, プログラミングに耐えられるレベルの設計書ではなかった。実際, 数本のモジュールについてはプログラミングを終了していたが, まともに動くものではなかった。

5 再設計

最終的には, 設計の悪さも手伝って (できることならもう一度設計し直したいという意識はあった), 私の担当した特定の業務だけの再設計を行なうこととなった。データベースの構造から見直しを図り, システムを再設計するのであ

る。これにより全てのドキュメントを再度作成する羽目になったわけだが, (同じシステムを設計し直すという担当者の意識は別として) システム上の問題としてみた場合はよいことではあった。

6 ありがたみ

同一システムの再設計はいやなものである。しかもそれがつい先日設計したものを破棄にして行なう場合は尚更である。しかし, この再設計は私にとって「ありがたく」感じた。

6.1 フィードバック

既にこの段階では, 前の設計で犯した前述のミスを認識していたため, どこをどうすればよいかははっきりしていた。再設計で課せられた要件自体は要求レスポンスの実現, データベースの構造変更に伴う新しい方式の策定など, 容易なものではなかった。しかし, 直前の作業で失敗した, ドキュメンテーションの徹底, 構造化設計に対する反省点についてのフィードバックがすぐさま行なえたのである。
再設計後はシステムの内容は別として, 設計内容としてはまずまずのできと判断した。

通常であればこのフィードバックは次回の開発作業まで持ち越されるが, この開発では, 「まずい」と感じた直後に行なえた。プログラム製造時に設計書に感じた統一の無さ, 機能が明確でない, 果ては機能要求を満たしていない, といった製造者から見た欠点は解消することができた。

7 設計における問題点と解決策

しかし, このようなことは通常の開発では起こりえない。普通は一回だけである。一回きりの開発を成功させるためにはどのような設計を行なえばよいのか?

7.1 ドキュメンテーション

この開発ではドキュメントの重要さを認識した。システムを構築するための設計書として, 必要な情報を過不足なく記述することがいかに重要であるかを, 身をもって体験した。それでは, どのようなドキュメントを作成すればよいのか?実をいえば今以て判らない。試行錯誤の最中である。作る側からいえば「小量」, 見る側からいえば「単純明解」といったところであろうか。今年度から発足した開発チームでは, ドキュメントの標準化を図り, 誰が見ても内容が理解でき, 誰もが作れるような仕様書を検討中である。現在はDFD をベースとした分析を行なっているが, その理論をプログラム設計のレベルにも応用しようと検討中である。

今までにとった手法の中では, トップダウン (名称は定かではない) による設計が一番効率がよいように思えた。システムをいくつかの機能に分け, 更にその機能を構成する機能に分け... という手順を繰り返して, 機能毎に入出力を明確に定義していく方法である。

この方法の場合, 「機能」と「ロジック」の切り分けが人によって異なるという問題が生じた (機能で分割するため, データ中心アプローチ, オブジェクト指向といった概念とは多少かけ離れ, 「よい設計」はしづらいと感じた)。私自身も, 「これは機能だ」, 「これはロジック」だとはいうものの, 明確な理由付けや, その証明はできた例がない。

7.2 理論武装

個人のスキルを高め, 構造化設計, データ中心アプローチ, オブジェクト指向といった理論を熟知し, 実際の設計に適用していくことも有効であろう。しかしながら, どれも「理論」であり意義は認識できても, それを適用し実践する方法, それも間違いのない「こうすれば必ずうまくいく」といったものはないため, 難しいことも事実である。前回の開発では, データ中心アプローチについては途中放棄, 構造化設計は単なる「モジュール分割」に留まったものも多く, オブジェクト指向については論外であった。

7.3 プログラミング

構造化プログラミングを適用したときの問題として, モジュール数の増加とモジュール間インタフェースの複雑化があるように思う。モジュールを適切な機能単位に分割し, モジュールの独立性を高めることは有効であると判断する。設計においても, 限られた機能を持つ, 限られた大きさのモジュールの設計は楽なものである。

7.4 オブジェクト指向

数年ほど前からオブジェクト指向プログラミングやオブジェクト指向データベースといったものが注目されている。つい最近まで携わっていた開発と, 現在開発中のシステムにオブジェクト指向の技法を取り入れ, 設計を試みたことがある。しかし, 現在の大型機の環境では難しく, 構造化設計の延長で取り入れているに過ぎない。大型機の環境下で稼働する SmallTalk などがない限り「オブジェクト指向の香がする構造化設計」が精一杯であろう。

7.5 CASE ツール

要求分析やコーディングで使用できるツールは多数あるが, システムの設計で役に立つものは見たことがない。精々

狭い範囲の作業のサポートである。構造化設計をサポートするようなものがあれば有効であろう。現状ではCASEツールも成長過程であり，ある特定の理論を応用したものが大半で，理論を知らなければ役に立たないものである。ツール内で理論を管理できればよいのだが。

8 最後に

設計については各種の理論があり，どれも一長一短があると思う。また，実際の作業に適用できるものも限られる。個人の資質に依存する設計には限界があるということであろうか。

- 設計がきちんとできれば理論なぞ不要である。
- 理論を理解しなければまともな設計にならない。

どちらも真であると思う。「よい」設計はあるのだろうが「正しい」設計は...

ワークショップのタイトルに「テクノロジ」という言葉があるので，夢物語を一つ。

分析作業で用いるDFDに情報を付加し成長発展させ，最終的には一つのシステムが構築できるようなCASEツール，あるいはHAL 9000の開発が待たれる。

III. W.S.終了後の感想

(Q1) 期待していたことは?

(1) 設計手法を現実の作業にいかに適用できるかの判断.

(2) 未知のグループでどこまで自己の主張ができるか.

(3) 未知のグループでどこまでチーム作業ができるか.

(Q2) 期待は満たされたか?

(1) CRCがある程度有効と感じたが．オブジェクト指向の手法をメインフレーム環境でいかに使うか→課題

(2) 活動方針「かくしごとをしない」，「発言にはレスポンス」を実践した (と思う)

(3) 意見が収束しないことが多々あった．時には妥協も必要.

(Q3) 設計に意識の変化はあった?

構造化設計しかやったことがなかった→オブジェクト (クラス) の見方が新鮮

マシン，言語にとらわれない設計が重要という再認識.

(Q4-1) 他人とのコミュニケーションについて

難しい．が，3日あれば何とかなる.

(Q4-2) 短時間での，問題の理解と作業について

自己の主張だけではいけない.

よい意見であれば，自分の考え方と合わなくとも，認め吸収する姿勢が大切と感じた.

(Q5) 経験は，自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して，新しい視点が持てるようになった.

(Q6) 全体討論の内容に関して

図書館コンピュータクラスはいらない．やっぱりメニューでしょう.

G3のクラスよりは，よさそうなクラスができたと思いたい.

質問に対する明確な解答のない場合があった．→意志の疎通は難しいと感じた.

(Q7) その他，W.S. に参加して良かったこと

Zがおもしろそう.

(Q8) W.S. の悪かったところ

最終発表について

当グループの発表者が，全体 (チーム) としての経過，結果の報告だけでなく，個人の意見をメンバの合意なしに発言したことが残念である．信念，意見を持つことはよいが，チームとしての活動と関係のない部分をチームの報告として欲しくなかった．バカヤロー! あるグループで，血液型によりチームの行動を説明していたが，グループの活動結果としての行動，状態を実証もされていない理論まがいのもので説明したことにハラがたった (人間をクラス分けしようとする行動に).

同じ意味で，各自のポジションペーパーに血液型の項目が必要なのであろうか? 何らかの事故を想定し，輸血の可能性を考えたのであれば，PCが管理すればよいことなはず．なぜ 血液型を載せたのか? SEAとは血液型に より人を判断する組織なのか? 是非 PC メンバの意見をうかがいたい.

SEA の会員になろうかと思ったが，そのような人たちの構成ならば考えてしまう.

(Q9) 今後 W.S. に取り上げたいテーマ，全体的感想

メインフレームの COBOL 上で CRC の結果をインプリメントする方法を考察したい

21 **舟引 秀光** (ふなびき ひでみつ)

性別：男　年齢：25　血液型：B

会社：(株) さくらケーシーエス

所属：研究開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

- オブジェクト指向分析に関する調査
- オフィス情報システムへの OODB 適用の準備
- HyperText システムへの OODB 適用の準備

(2) 現在利用している開発環境について

マシン Sun SPARCstation2

オペレーティングシステム SunOS 4.1.1

コンパイラ Sun C++ 2.0

DBMS ONTOS DB 2.1

UI ツール ONTOS Studio 0.51Beta

(3) **これまでシステム設計に採り入れてきた技法など**

オフィス情報システム設計の初期段階でのモデル化において, Coad/Yourdon による OOA 手法を利用。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

1 はじめに

現在我々は, オフィス情報システムやハイパーテキストシステムへのオブジェクト指向データベースの適用に取り組んでいる。その作業として当然, 分析/ 設計/プログラミングの順で行なっていく。RDB においての分析・設計には, 呼称こそ様々であるが, 外部データモデル・概念データモデル・論理データモデルという概念があり, これらを使ってモデリングを行なっていく。また, 実際に当社においても就業管理を中心とするシステムが RD B において稼働している。

しかし, このような既存のシステムに OODB を適用していく際, または新しく構築していくシステムに OODB を適用していく際に問題が出てくる。それは, OODB の分析・設計においては, RDB の様な定式的手法が確立していない点である。現在提唱されている OOA/OOD の手法は, 20 種以上あるが, 構造化技法を発展させたものや, 全く独自の定義を行なっているものなど様々である。いずれも, 根本的な部分では共通点が多いが, 記法, 手順が異なっている。また, クラス, オブジェクト等の概念的な事項に対しても, それぞれ意味づけが微妙に異なっているものもある。

そこで, システムの試作の初段階として, 対象領域を把握するために, またより正確にオブジェクトを抽出し, 属性, 振る舞いを定義していくために, 幾つかの OOA,OOD の手法についての調査を行ない, 適用を試みた。ここで述べていくのは, 日本においても, 雑誌, セミナー等で紹介されている, Coad/Yourdon による手法と Rumbaugh らによる手法 (OMT: Object Modeling Technique) である。

なお, ここではこの文章の主旨上, 特に各手法の問題点を中心にとりあげていく。それから, 私なりの考えを述べていく。

2 基本的表記法

Coad/Yourdon 法, Rumbaugh 法ともクラス表現の仕方は, 全く同じである。つまり, 四角の中を 3 層に区切り上から順に, クラス名, 属性名, 手続き名というふうに定義している。

しかし, オブジェクト表現においては, 両者は異なっている。Coad/Yourdon 法ではクラスをさらに 4 角で囲むことによって, そのクラスが実際の値としてのオブジェクトを持つことを示している。これに対して, Rumbaugh 法では, クラスとは別の 4 角を用い, その中にクラス名, 属性値を記述することによりオブジェクトを表現している。

しかし, この Rumbaugh の表記法では, システムの構造が複雑である場合, 一目見ただけでは, クラスとオブジェクトの区別がつかない可能性がある。まして, クラス表現に, 属性名と手続き名を省く簡略記法を用いる場合, 抽象クラスとオブジェクトを持つサブクラスの区別がより困難となる可能性がある。

我々がオフィス情報システムのモデル化において適用してみたのは, Coad/Yourdon の手法であり, 実際, 抽象クラスとサブクラスの区別においては, 不具合な点はなかった。

3 クラス・オブジェクト間の関連

クラス・オブジェクト間の関連性に対する Coad/Yourdon のとらえ方と, Rumbaugh らのとらえ方ではかなり異なっている。

Coad/Yourdon 法では, 関連を大きく, Gen-Spec(一般化一特殊化関係, is-a) 構造と, Wh ole-Part(部分一全体関係, has-a) 構造として定義し, その他の関連をインスタンスコネクション, メッセージコネクションにより定義している。これに対し, Rumbaugh 法では, 全ての関連を association という概念でとらえ, その中に generalization, aggregation やその他の関連を位置づけている。さらに association のインスタンスという位置づけで link という概念を定義している。つまり association と link の関係はクラスとオブジェクトの関係に対応しているといえる。

確かに Rumbaugh 法では, 従来あまりすっきりと説明されていなかったオブジェクト間の関連を association と link を用いてきれいに説明している。

しかし, 私が思うに, オブジェクト間に成り立つ関連とオブジェクトの集合間に成り立つ関連, さらに抽象クラス間に成り立つ関連などを別々に捉えるべきであると考えている。

また, 実際に適用してみた Coad/Yourdon 法では, 問題領域の主なモデル化以外に, システムを Human Interaction, Task Management, Data Management という 3 つの構成要素によってモデル化している。しかしそれらを全て, OMT でいう Object Model によって記述している点が問題である。特に, 関連性こそシステムで重要であるので, 詳細な表記法を用いるべきである。また, 時間的前後関係にっ

いての考慮がない点も問題である。

4 おわりに

現在適用している手法は, Coad/Yourdon 法である。それは, 現段階においては, オブジェクト間の関連, 手続き等が詳細に決定していないからである。対象の種類に関わらず, このような初期段階においては, まず問題領域に何があるかを把握することが重要であるという意味からも, Coad/Yourdon 法が適用しやすい。そして, 関連, 手続きが具体化してくれば, Rumbaugh 法に移して分析・設計していくのが良いだろう。

ただ, 両手法について指摘できることは, 制約表現においての未熟さである。制約という概念を如何に機能的にモデル化していくかが 1 つの課題である。また, 既存の RDB からの移行のし易さという点についての考慮も必要である。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) 各設計手法の特徴の評価

(2) 他の人が現在どのような仕事をしているかの理解

(3) 協同作業におけるポイントのチェック

(Q2) 期待は満たされたか?

(1) について : 自分のグループの手法については, TEXT を読む暇はなかったものの, 自分なりに評価できた.

(2) について : 概要は聞く事ができたが, もう少し詳細に聞く時間が欲しかった.

(3) について : かなり議論が出たりして作業がストップしたりしたが, 短期間での協同作業をどのように進めていけばよいかを少し理解できたように思う.

(Q3) 設計に意識の変化はあった?

特に変わった点はないが, 改めて認識できた事は, 特定の手法に頼りすぎてはいけないということ. アプリケーション, 規模により各段階で良い点を取り込む事が大切.

(Q4-1) 他人とのコミュニケーションについて

自分の意見をはっきり述べる事も重要であるが, 一方で協調していくという事も重要.

(Q4-2) 短時間での, 問題の理解と作業について

目的, 目標, 仕様に対する全員の認識の一致が一番重要であるが, 途中の意見は, くい違っていた方がより良い物が出来るかも知れない. ただし, 結果を出す段階では全員の認識を一致させる.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■ワークショップの議論と現実とはやはり違うと感じられた.

(Q6) 全体討論の内容に関して

質問に対して, みんなもう少し積極的に発言して欲しかった.

(Q7) その他, W.S. に参加して良かったこと

設計に対する他人の考え方を聞く事が出来た.

(Q8) W.S. の悪かったところ

(1) 各手法に対して簡単な内容を事前にレジメ等で送って欲しかった.

(2) ホテルは良かったが, もう少し周りの環境も考えて欲しい (アウトドアで楽しめる所←北海道希望)

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

(記述なし)

23 **門田 久好** (もんた ひさよし)
性別 : 男　年齢 : 28　血液型 : A
会社 : 新日鉄情報通信システム (株)
所属 : 鉄鋼システム本部 企画推進部

I. プロフィール

(1) 現在までの主な仕事の内容について

入社以来, 製鉄所で生産管理システムの開発・保守業務を担当してきた. その中でも主に, 受注処理, 材料計画システムを中心に, プロジェクト開発および保守 (メンテナンス, トラブル対応等) の業務経験がある.

(2) 現在利用している開発環境について

メインフレーム (IBM. 日立) を利用した開発を中心に行なっている. ソフトウェアの開発においては, 社内で独自開発した, ソフトウェア開発のためのツールを使用し, 一貫して開発を行なうことができる.

(3) これまでシステム設計に採り入れてきた技法など

トップダウン設計を中心に行ない, 機能の概要を順次詳細化していく手法を採っている.

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

1 設計に感じた「うらめしさ」の提起

設計段階で, 開発システムの利用者 (以降, 利用者と表記) との十分なレビューを実施して仕様を決定したにもかかわらず, 後工程 (製作, テスト等) で仕様変更が発生し, 設計の一部やり直しが発生した点である.

2 仕様変更の影響

後工程で仕様変更により, 次のような影響がある.

1. 設計において, 利用者の要求機能の洗い出しを行なうために, 利用者の中の代表にインタビューの実施, 業務に関する資料の調査, システムの開発者 (以降, 開発者と表記) による仕様のまとめ, 利用者と開発者間でのレビューの実施等を行なう必要があり, 業務をモデル化するためにかなりの時間がかかる. そのため, 後工程での仕様変更による設計のやり直しは, 納期, 開発要員, 品質などの面で回復するのに大変な作業となる.
2. 特に仕様変更が発生しやすい機能として, 例外的な処理が多い. この例外的な処理は, 開発後に保守を実施する時に, 理解しにくい箇所になりやすい. そのために, 仕様変更に即応して設計に作成したドキュメント類も変更しないと保守時に使えない.

3 仕様変更の発生原因

このような後工程での仕様変更発生の原因として, 設計段階について次の点が考えられる.

1. 業務のモデル化を進める上で, 利用者と開発者との間で共通認識できる, よい表現方法があまりない. 表現方法としては, 文章, 各種の図表現などがある. 開発者が作成するシステム化のための仕様書で, 利用者へ説明をしても十分な理解が得られない.
2. 利用者が, あまり意識していない常識的な業務, あるいは逆に頻度が少ない業務のモデル化が漏れやすい. 開発者が, 資料調査等である程度業務に関する知識をもっていたとしても, 利用者から直接のインタビューを基本に業務のモデル化をする. その時に, このような業務はあまり話題にならず, 開発者が見逃してしまう.

4 改善の提案

1. 利用者用の説明資料により, 利用者と開発者との間で共通認識ができるようにする.
   図表現を中心とした利用者用の説明資料を作成し, 利用者とのイメージ合わせ後に, 開発者が業務のモデル化を実施する.
2. 利用者が早期にシステム化のイメージができるようにする.
   開発システム対象にもよるが, プロトタイプ手法を用いて試作品を作り, 利用者がシステム化のイメージを早期にとらえられるようにする.

III. W.S. 終了後の感想

(Q1) 期待していたことは?

設計方法を実習を通し身につける.

(Q2) 期待は満たされたか?

CRC 手法について, 概要を理解でき, 一応満たされました.

(Q3) 設計に意識の変化はあった?

カードなどの道具を使うことで設計に有効なことがわかった.

(Q4-1) 他人とのコミュニケーションについて

グループ内の指針ということもあり, グループ内ではとれた.

(Q4-2) 短時間での, 問題の理解と作業について

各自の業務経験から種々の考え方があり, まとめるのに時間がかかった.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) 全体討論の内容に関して

構造化設計でどの時点で I/F を作れば良かったのか?

(Q7) その他, W.S. に参加して良かったこと

グループ内が中心ではあったが, 他社の人たちと意見交換できたことが大きかった.

(Q8) W.S. の悪かったところ

他のグループとの意見交換する機会が少なかったと思います.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

設計手法 にすぐに到達できるように, 問題設計 を工夫しておけばもっと手法理解に専念できたと思う.

グループ 5

20 **西岡 啓和** (にしおか ひろかず)
性別：男　年齢：30　血液型：B
会社：(株) さくらケーシーエス
所属：研究開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

- RDB 関連
- オブジェクト指向 DB 関連

(2) 現在利用している開発環境について

- Sun SPARC station2
- Open windows

(3) これまでシステム設計に採り入れてきた技法など

- RDB の社内教育用テキスト作成 (概要, スキーマ理論, SQL)
- 動的 SQL 処理用サーバ作成 (informix,TCP/IP)

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

私が所属しています研究開発部内では開発設計を行なっていないということでは必ずしもありませんが, 現在, 私は具体的な開発作業を担当しておりません。また, 上記題目としてあります (a), (b) 共に「設計に感じた」という内容が何を指しているのか私にはよく分かりませんので, かつて経験したことのあるマイコンのソフト開発での仕事を行なっている際に「これは何とかならないものか」と思われた内容について述べたいと思います。

ある製品に搭載するマイコンのソフト開発を行なった際に, 「これは何とかならないものか」と思われたことは, スペック変更と納期 (スケジュール) であったといえます。それは, 変更に対応する処理が面倒だとか納期がきびしくてしんどいといった事よりも, 出来上がったソフトを改めて見直した時そのプログラムのできばえにスペック変更や納期といった要因があまり好ましくない影響を与えているのではと思ったからです。

開発途中に起こるスペック変更の根本的な問題は, ソフト開発担当者にとってみれば全く別の問題とも言えます。しかし, 現実として開発スタートから開発エンドまでの間に, スペック変更なしといった事例は少なくとも私にはありませんし, 一般的に考えてもそのような善例はないでしょう。従って, 開発担当者にとってもその問題は, もはや自分の仕事をうまく行なっていくうえでの課題となってきます。当然私も, まとまった処理はルーチン化しておき, 全体のフローをはっきりとそして出来る限り簡潔にするなど, 変更に備えるべき対応をとりながら仕事をしておりました。

しかし, 次々に入ってくる変更内容に対応する処理を加えていくに従って, せっかくルーチン化し全体の構成をフロー化していたその姿が少しづつくずれて行き, 最終的に出来上がったプログラムを見ると, 作った本人以外の人から見て, フローが分かりにくい, 見直しにくい, 従って, version up(再利用) しにくいものとなっていました。

そのような哀れなプログラムにはならないように気を配りながら当初は対応していたのですが, スペック変更はいとも簡単に起こっても納期変更 (延長) はないといった切羽詰まったスケジュールに迫られているうちに, とにかく, 一番簡単に (手軽に) その変更内容を実現できる処理 (対応) をとってしまうようになり, 一度, そのような対応をとってしまうとその後はもう言うまでもなくといった状態です。

しかも, 変更内容によってはフローの構成自身も変更しなければ対応できないようなものもあり, それじゃ最初から作り直した方が早いと何度も叫びたくなったような気がします。

確かに, スペック変更などに速やかに対応することも開発担当者の仕事ではあり, それがうまくできるか否かはその担当者の能力, 心構えなどにもよるでしょうが, いずれにせよ, 出来上がったソフトのできばえ (機能が実現できるとかいったことではなく, そのソフトの信頼性とか品質とかいったもの) にとってあまり良い作用はしないといえます。

一度完成したソフトを見て, もう一度それを最初から作り直したら, もっとうまく, もっとすっきりとプログラムできるのにと思った経験のある人は多いと思いますが, そのことは, 実際開発中に満足できる作業ができなかった (能力的に, あるいは大部分の担当者にとっては時間的に) ということにほかならないといえるでしょう。

このようなことを意識して設計という作業を見てみると, 機能の実現の為にどんな技術, 手法が有効かといった検討は重要, 必要なものではありますが, ただその点だけを考慮するだけの設計では, 実際の開発段階で起こる問題には対応できていないといえるように思えます。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

「構造化分析」, 「設計」に対して, 何か 1 つの方法論 (考え方, 捉え方) というものの概要を見てみたいというようなこと.

(Q2) 期待は満たされたか?

満たされたもの :

今回の JSD という手法で, 共通問題を考えて行くことで, 充分とは言えないが, 1 つの現実世界の捉え方といった ものは把握できた.

(Q3) 設計に意識の変化はあった?

特に変わっていない.

(Q4-1) 他人とのコミュニケーションについて

特別な意識はないが, 話題 (話の主題) がバラバラにならないのが大事ではないでしょうか?

(Q4-2) 短時間での, 問題の理解と作業について

認識を共通に維持していくことが大事ではないでしょうか?

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には : 今回の JSD 手法で 設計過程というものを疑

似体験はできたが, だからといって職場に戻って, 即 JSD 手法ができるかというものでなく, 改めて JSD 手法の考え方を見直す場合に, 認識しやすくなっているだろうと思える.

(Q6) **全体討論の内容に関して**

特になし

(Q7) **その他, W.S. に参加して良かったこと**

個人的には, 山崎利治さんのアドバイス・お話しが聞けたことが今回のワークショップでの一番の成果であったと言える.

(Q8) **W.S. の悪かったところ**

グループでの活動も大事であるが, そこで議論されたことを一人で考え直すというような時間がほとんどなかったのが残念.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

(記述なし)

07 **冨岡 雅巳** (とみおか まさみ)
性別：男　年齢：26　血液型：A
会社：(株) ジェーエムエーシステムズ
所属：西葛西事業所

I. プロフィール

(1) **現在までの主な仕事の内容について**

- X-window を用いたユーザインターフェース
- データベースアクセス言語処理系 (構文解析部)
- バッチ自動運転の環境作成/運用管理の支援システム

(2) **現在利用している開発環境について**

現在, 業務として開発には直接携わっていない。(工程管理などを担当)

'92 年 3 月までは, 開発環境として UNIX(Sun OS 4.1.1) を使用。

(3) **これまでシステム設計に採り入れてきた技法など**

構造化設計をこれまでの開発で採り入れてきたと, 私は思っている。構造化設計について私自身正しく理解, 実践しているかどうかは怪しいものであるが, ... ((4) を参照のこと)

II. ポジションペーパ

(4-b) **設計に感じた「うらめしさ」について**

私は, この (4) b) の意味を充分には理解できないでいる。おそらく, 設計工程や, 設計工程の結果について経験を通して感じたことの記述が, ポジションステートメントとして求められているのだと思う。そこで私が業務で体験した”前任者の設計結果”に対する「うらめしさ」, 及びその問題点, 改善案について述べることとする。また, (3) に関連して, 今後採り入れたい技術についても述べたい。

私はこの 3 月まで, 言語処理系の開発に携わっていた。このプロジェクトに私が参加したのは第一バージョン完成直前のことで, 本来ならテストの最終段階であるべき時期であった。ところが, 各モジュールの機能分担が不明瞭, データ構造が複雑怪奇なため, プログラミング工程が遅れに遅れていた状況であった。私が参加してからも状況は変わらず, テストもろくに行えず完成予定日を迎えることになった。バグの多発 (試用中に 1 日に何個もバグを発見される始末) は当然のことである。更に, この処理系の開発に当初から関わっていた人物が人事移動でいなくなってしまったのである。結局, 数ヵ月後に全て設計からやり直すことになった。この再設計の時には, 構造化設計の手法を用いたつもりである。(構造化設計について, きちんとは勉強していないので, 構造化設計としてあるべき形態を成していたかどうかは確信がない。)

何が問題であったのか。聞けば, このプロジェクトは, まずプロトタイピング的に構文解析器を試作するところから始まったそうである。そして, 構文解析器としての仕様が決定された。問題はこの後である。モジュール構成やデータ構造の見直しを行わず, 試作物をそのまま継続して開発の基盤として用いてしまったのである。機能が不明瞭なモジュール, 充分な検討が施されていない難解で複雑な構造のデータが存続することになった。その結果が, プログラミング工程の遅れや, 品質の異常な低さになったのである。前任者の設計工程に対する認識の甘さは, この時, 実に「うらめしく」感じられたものであった。只, このプロジェクトに参加することで, 設計の重要性を実感できたことは, 私にとって大きな経験であったと思う昨今である。

プロトタイピングを行う。これは, ユーザー要件を引き出し, 仕様を明確にする手段として有効な方法である。しかし, 手間がかかるといった不安がある。場合によっては, 設計, プログラミング工程が 2 倍になるかもしれない。上のプロジェクトの前任者もこの点を心配していたようであった。仮に, 再開発を容易に行える確証があったなら, 彼が再設計に躊躇する事もなかったであろう。容易に再開発を行える環境を用意することが, このプロジェクトの設計工程で発生した問題の改善になるのではなかろうか。以下で, この実現について考察する。

一般にプロトタイプ試作時の仕様と開発本番での仕様には差がある。試作時 (設計, プログラミング, テストでの) の成果物が再利用可能かどうか。プロトタイピングでの成果物で再利用可能なものがなければ, プロトタイピングの手法はそのまま開発者の負担増をもたらす。再開発の容易性は成果物の再利用性が鍵である。再利用性の実現はそのまま, 再開発の容易性の実現となる。

バージョンアップについても考えたい。既に存在するシステムのバージョンアップを行うとき, システム全体の再設計, 再プログラミング, 再テストを行うのであろうか。もし全工程の再施行がバージョンアップの度に必要ならば, バージョンアップを経る毎に開発の量は肥大化し, 期間は長期化し, システムの進歩は緩慢なものとなる。ここでもシステムの再利用性が重要である。充分に再利用可能なシステムならば, 追加/変更部の開発だけで, バージョンアップに対応できる。設計, プログラム, テスト結果すべてについて再利用可能なシステム開発が望まれる。

ここでは, 設計の再利用について考える。再利用可能な設計が行われているならば, 新たなシステムの設計においても, 新規に設計を行うこと無く, 過去の設計を再利用することで設計を完了することが出来る。これは, 設計工程を容易にし, 工数の削減に繋がる。しかも品質の高い設計が実施可能となる。

再利用性の高い設計とはどの様なものか。システムの部品としてのサブシステムにおいて, 再利用性はいかに実現されるか。システム内部でのサブシステムの独立性を高くすることで, サブシステムの設計の再利用性を高められはしないか。また, サブシステムの設計が独立性の高いものならば, サブシステムの部分的な再設計も容易である。

サブシステムの設計でありがちな方法に, 最初にサブシステムの機能と入出力とを決めることがある。サブシステムの設計で, 入出力を最初に決定するとどうなるか。すでにこの時点でそのサブシステムの独立性は失われている。つまり, 入出力の決定により, 入力元, 出力先のサブシステムと切っても切れない関係が生みだされている。機能単位中心の設計で, サブシステムの独立性を獲得することは永遠に不可能である。つまり, 構造化設計では再利用性の実現は絶望的である。

どうやら, 設計の視点を変革する必要があるようだ。この点に, オブジェクト指向設計 (OOD) が答えてくれそうである。オブジェクトとして高度に抽象化されたデータや手続きは部品として再利用性の高いものであるに違いない。設計の段階からシステムの各部をオブジェクトとして捉え, 内部の隠蔽化を行えば, 設計工程での成果物は再開発や他の新規システムでの設計で再利用が可能なはずである。私は今現在, 社の勉強会を通じオブジェクト指向につて学習中であり, その本質を充分に理解している状況ではない。オブジェクト指向技術の設計の再利用性について, その有効性を論じ切れないのが残念である。しかし, オブジェクト指向の技術の可能性は充分に期待できるものであると確信している。OOD は今後採り入れたい, いや, 採り入れるべき技術・手法である。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) OOA/OOD の修得.

(2) 他社, 同年代の人との交流.

(Q2) 期待は満たされたか?

(1) JSD 法に触れることで, 未知の視点を体験できた. この経験は 今後の OOA/ OOD の修得に効果があると期待できる.

(2) 当初の期待以上にコミュニケートがとれたと思う.

(Q3) 設計に意識の変化はあった?

モデル中心の設計で, 機能の追加/変更が生じた場合の, モデルの変更の少なさは驚きであった. 設計手法の選択が与えるシステムのライフサイクルへの影響を認識した次第である.

(Q4-1) 他人とのコミュニケーションについて

未知の他人とはいえ, 同年代なため打ち解けることの困難はなかった.

(Q4-2) 短時間での, 問題の理解と作業について

G5 内部の問題として, WS の期間について認識が甘かったと思う.

明確に予定を立てず, 結果として会期の有効利用が不充分だったと言えよう.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には : モデル中心の設計とは, どういったものか, その有用性はなにかを感覚的とはいえ知る経験となった.

■その他 : 思い出 (仕事とは無関係)

具体的には : 久々に熱中した時間を持てた (青春の思い出)

(Q6) 全体討論の内容に関して

頭がボーとしている状態で, 積極的な参加ができず誠に申し訳ありませんでした. おわびします.

(Q7) その他, W.S. に参加して良かったこと

実務者集団の SEA が, 予想以上にアカデミックな雰囲気があり, 嬉しさすら覚えた.

(Q8) **W.S. の悪かったところ**

G5 の JSD 法の取り組みで, 山崎氏に頼り過ぎたといった反省がある. “発見” を自ら出来ればよかった.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

山崎氏のレクチュアであった形式的な記述の修得/活用に, 私個人は非常に興味を感じている. 取り上げていただきたいテーマである. 取り組み難いテーマではありますが...

09 **江谷 典子** (えたに のりこ)
性別：女　年齢：29　血液型：A
会社：富士ゼロックス情報システム (株)
所属：技術部

I. プロフィール

(1) **現在までの主な仕事の内容について**

- パーソナルコンピュータの開発
  基本 OS 開発, ノート型パソコンの調査・企画・開発
- ヒューマンインタフェースの研究開発
  トラックボール・ペンの操作性調査
  MS-DOS 標準搭載型統合環境ソフトウェア (GUI) の企画, 開発
- CSCW の研究開発

(2) **現在利用している開発環境について**

- Macintosh IIfx
- smalltalk-80

(3) **これまでシステム設計に採り入れてきた技法など**

- 開発環境： MSC, MASM
  オブジェクト指向分析を採り入れ, 従来性能, 機能重視型デザイン設計から, 操作手順や使いやすさという点から取り組み開発を進めた.

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

【問題点】

1. 開発に入るまでの段階で, システムの対象となる人・集団・環境を調査していないため開発途中で何度も仕様変更がはいったり, 製品完成後に大幅な修正が必要となり, 開発者にとっては限られた工数の中でストレスをかかえている。
2. プログラミング手法および開発環境は, 変化への対応を前提に考えられていない。開発者と使う人における,「人的変化 (人事異動など)」「思考の変化」「好みの変化」, そして開発者や使う人を取り巻く「環境 (集団・組織・社会) の変化」に対応できないために, 以下のような歪みが起きている。
   (a) 仕様作成から完成までの間に「どういう処理にしていいのかわからない」というあいまいな部分が発生するが, とにかく決めてしまわないと次の工程に進めないなどの理由によりあいまいな部分を無理やり方式を決めてしまう。こういう部分からバグ, 障害, ユーザークレームが起きている。人間は常に「あいまいな考え」を持っており, 間違っていれば訂正ができる特性を持っているにもかかわらず, 開発環境はこの人間の柔軟性に対応できていない。
   (b) 操作性評価のために, 社内評価を繰り返し実施し, 製品に反映させていた。しかし, 完成度が上がるにつれて, 構造化プログラミング手法の下ではモジュール修正による他モジュールへの影響が見えにくくなり, 開発者側はテスト項目の増加とバグ発生の可能性増加によりストレスをかかえ, 評価反映を拒みだしてしまう。
   (c) 開発が始まった時点と終った時点では対象市場やユーザー層が変ってしまって, 売れない製品になってしまう。開発過程において, 環境の変化を吸収できない, あるいはしたくても出来ない開発環境である。
3. 使いやすさの評価基準がないために, コンピュータの専門家であり多少の使いにくさも克服できる開発者の価値基準により開発され, ユーザーには複雑で使えない商品となっている。また, 開発者側でも画面設計において, どのような言葉や表現にすれば理解していただれるのか判断できない。

【改善案】

1. 開発環境や開発工程についてフィードバックを機能させる
   (a) 変化する環境にあって使い手の状況と開発者の状況を調整し, 両者間のコミュニケーション・ギャップによる差異を小さくするための調整役が必要。使い手にとって本当に必要なシステムは何であるかを分析することや, 仕様の間違いやあいまいな点を明確にして使い手と開発者の間のコミュニケーションを頻繁に行わせる。
   (b) 開発中に何度もプログラムの手直しがきるような開発工程や手法を取り入れる。

(c) 商品を販売または納入後も使う人とのコミュニケーションを継続し, 使い手からの評価やニーズを吸い上げて次回の開発や社内評価基準に反映させる。

2. 使う人の好みや使いやすさを満足させるため使う人が態動的に操作環境を調整できたり, また使用する環境に応じて変化するインタフェースがつくれるプログラミングモデルを開発する。オブジェクト指向→ヒューマン指向→環境指向 へとプログラミングパラダイムは変化していくべきではないかと考えている。

【疑問点】

オブジェクト指向技術は開発環境における最適化技術であり, 使う人の環境への最適化ではないと考えている。しかし, ヒューマン指向や環境指向への移行は, このオブジェクト指向技術の上で進化していくものであると推測している。そこで, 今回のワークショップ参加により, 以下の課題を探るヒントが得られることを期待している。

1. オブジェクト指向技術は, どれだけ開発環境を改善できるのか? 変化にどれだけ対応できるのか?
2. 使う人の環境に応じる場合, オブジェクト指向技術の限界は何か?

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) 現場の人たちは, 複数の視点をどう調整して, 1 つにしているのかを知りたかった.

(2) 手法に関して

・問題領域の不明点を明らかにする方法.

・現実世界の変化にどれだけ対応できるか.

・複数の視点を一つにしていく方法.

(Q2) 期待は満たされたか?

(1) 模造紙に書き出す, カードに書き出す, イメージを話し合うという方法 (CSCW 風) によって視点を 1 つにしているのが, OO グループの特徴であった.

JSD 法の場合, イメージを調整するのではなく, 判断基準を 1 つの視点へと導いている. よって, この点について手法にも含まれるべきであるが, 現状 では付加された現場のアイデア でおぎわれていた.

(2) 不明確な点を残して, 次にすすむ. 機能 unit で扱っているので, Feedback して設計追加をするのが容易にしてある. 一気に明確にするのではなく, 何度もやり直して明確にしている.

(Q3) 設計に意識の変化はあった?

人間の視覚認識には, “事象自体をみる” “事象と環境をみる” という視点により実世界の情報を取り込んでいる. また, 思考については「発散」「収束」がある. このような人間の視点の違いによって, さまざまな手法が存在していると考えている. 今後, 手法の統一理論をまとめみたいと考えた. (→こういう立場, 視点において XXX 法で問題を解決する etc.)

(Q4-1) 他人とのコミュニケーションについて

「まずは自分から話しだす」という考えを持って いたので実行した.

初対面の場合, 相手がどんな人かわからないと安心ができない. その結果, 自己開示がそこなわれ自分の考えを表出しにくくなる. そこでまず「自分はこういう考えを持っている」を表出し, 相手からも引き出す.

(Q4-2) 短時間での, 問題の理解と作業について

・短時間で目的 (目標) が明らかな場合, あえて各人のグループ内役割を決めなくても, 自然にできあがった.

・リーダを決めなかったので各個人間に構造をつくらない関係となり, 全員があるシーンではリーダになったりできた. その結果, 士気の高いグループ活動が行なえた. つまり, “グループ活動” を意識しなくても良い活動ができた.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) 全体討論の内容に関して

(記述なし) **(Q7) その他, W.S. に参加して良かったこと**

「プログラマたちの視点は実世界よりも実装段階への執着があること」を再確認した.

・細部を見過ぎて全体が見えていない

・ Keypoint の絞り込みができない

という特性が往々にしてプログラマにはある.

(私は, 実世界での使いやすさを考えた設計を, なかなかプログラマが納得しないので, 多々困ることがあったため→プログラマが良いと思う システムは実世界での使いやすさとは ならないことが多い.)

全員が手法を知らないということから, この手法をどうやって知っていくのかという問題解決もできて面白かった.

(Q8) W.S. の悪かったところ

Q7 の理由により, プログラマは実世界を分析する目を持つことが必要である. そういう視点を与えてくれる分析・設計手法を期待していた. しかし, 設計～実装に近い手法であるため, プログラマの視点や発想が変わらなかったのではないかと思う.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

(1) 先の理由により, 1 グループ内に, 上流工程経験者と下流工程経験者を配置し, 視点の違いの大きさを経験させるグ

ループ活動にした方が良いと思う．手法にはあらゆる立場の視点を一つにする方法論を付加する必要があるから．

(2) ヒューマンインタフェース設計をテーマにして下さい．GUI の乗り込えて，設計思想，適応時例と結果など困ったこと．

11 岩城 弘二 (いわしろ こうじ)
性別：男　年齢：28　血液型：B
会社：富士ゼロックス情報システム (株)
所属：技術部

I. プロフィール

(1) 現在までの主な仕事の内容について

- 溶鉱炉知識ベース作成
  エキスパートシステム構築ツール「NX-7」により，某社溶鉱炉知識ベース (プロダクション・ルール) 開発。
- 「NX-8」の開発
  Xerox Common Lisp により，「NX-7」の拡張版である「NX-8」の，主にインタフェース部分開発。
- 科学技術庁科学技術振興局
  「創造的研究開発支援のための自己組織型情報ベースシステムの構築に関する調査」調査および報告書執筆。
- 社内プロジェクト「Blizzard-92」
  オブジェクト指向技術の実用化を目指す社内プロジェクト Blizzard-92 に配属。現在，主にプロジェクト管理とリポジトリについての研究・開発に従事。

(2) 現在利用している開発環境について

マシン Macintosh IIci,fx,quadra
言語 Objectworks/Smalltalk
CASE ツール ObjectCast

(3) これまでシステム設計に採り入れてきた技法など

現在，分析・設計の手法としてオブジェクト指向方法論 (特に Coad/Yourdon 法，OMT 法など) を採用している。

II. ポジションペーパ

(4-b) 設計に感じた「うらめしさ」について

私が「うらめしい」と感じた開発の経験は，設計作業のみの問題ではなく，開発過程全体に対する問題である。したがって，ここでは設計に感じた「うらめしさ」ではなく，開発過程全体で生じた「うらめしさ」について述べる。

1 「うらめしさ」を感じた仕事の概要

数年前，某社より以前に Common Lisp により開発されたエキスパートシステム構築ツールに対する拡張の依頼がなされた。このシステムの知識ベースは最初はプロダクション・ルールのみであった。新システム開発の作業項目として，フレーム・システムの追加，推論機構の変更，C 言語インタフェース部分の開発，新ユーザ・インタフェース環境の開発等が要求された。私の担当部分は新ユーザ・インタフェース環境の開発であった。また要求された開発期間は 4 カ月であった。開発は，クライアント側と我々との共同開発の形で進められた。また，以前のシステムを拡張する予定だったが，変更点や追加点が多すぎるために，結果的にはほとんど設計をやるなおすことになった。開発の進め方としては，まずクライアント側から要求仕様 (このシステムはクライアント側にとっては研究・開発色が多少あり，最初から要求が完全に定義されているものではなかった。したがって，これは要求仕様というよりはたたき台と言える程度のものであった) が提出され，我々開発者はそれに対しての説明を受ける。ただし，この要求仕様にはある意味では設計仕様とも言える部分がかなり含まれている。また，この開発では，分析と言える作業は行なわれなかった。次にそれに対するモジュール毎の設計仕様を提出し，クライアントとの協議によりユーザ要求を明確化し，その設計仕様を訂正する。この設計仕様の作成と協議の過程を繰り返すことにより，最終的に設計仕様に対するクライアントの承認を受ける。次にコーディグを行ない，更にその過程で判明した設計の不備をクライアントとの協議により訂正する。コーディング終了後単体テスト，結合テストを行なった。システム完成後，プログラム詳細仕様書を作成した。最終的にこのシステム開発に要した工期は 6 カ月であった。

2 「うらめしさ」を感じた点

- 工期が短い。作業量的にはおよそ 1 年と見積っていたが，クライアント側の強い要望により 4 カ月と設定された。そして予算面での制約があった。
- 開発者毎の担当部分のモジュール化はされているが，時間的余裕のなさから各開発者は自分のモジュールの開発だけで手一杯となり，モジュール間の関係が明確化さず，インタフェースの取り決めがきちんとなされなかった。また，取り決めがなされても，コーディング途中の口頭による取り決め等でしかなく，きちんと仕様化もされなかった。結果的には，お互いの誤解などが生じて，インタフェース部分の不備が生じた。

- 特に Lisp 言語による開発にとっての有効なドキュメント・フォーマットがなかった。その為にドキュメントのフォーマット決めから始めなければならなかった。しかし, 他の開発要員と協議する余裕もなく, その為設計仕様に関しては開発モジュール毎に異なったフォーマットが出来てしまい, 結果的には統一がとれないものになってしまった。
- 仕様変更や追加が頻繁に起こり, そのため設計のやりなおしなどが頻繁に起きた。
- 設計仕様の訂正は, 問題が出た時点で行なわれたのであるが, 局所的な部分に関してのみ行なわれ, 全体的な整合性をとらなかった。結果的にいびつなシステム構成になってしまった。

3 「うらめしい事柄」が発生した原因と解決策

以上の点により工期が延びた原因は, 短すぎる工期によるものである。しかし, 以下の 2 点が的確に遂行されていたならば, もっと開発がスムーズに進み, 被害は最小限に食い止められていたはずである。

・プロジェクトの計画や管理が的確でなかった。

この開発においては, 個々の開発者に担当モジュールの開発がまかされていた。プロジェクト・リーダは存在したが, 開発者と兼任のため自分の担当部分の開発に手一杯で管理する余裕もなかった。しかし, このシステムを開発するうえでの的確なプロジェクト計画と管理は存在すべきであった。プロジェクト管理がうまくいかなかった原因は, 開発は必ずウォータフォール型で進めなければならないという慣習にしばられてしまったことによる (しかし, 基礎調査, 分析, 設計, 作成, テストという一連の作業をきちんと行なった訳ではないが)。この開発では, クライアント側から仕様の記述が完全に終了したと認められるまでに報告を何度も行なわされ, 次の段階に進むという事は行なわれなかった。しかし, 特にこの手のクライアント側の要求項目が完全ではないものにおいては, 途中の変更や追加が頻繁に行なわれるものである。元々, 開発途中の仕様を完全に記述することは不可能であり, ウォータフォール型で行なうには無理があった。そこで実際に後戻りが頻繁に起こってしまったのである。ならば, 初めからこのシステムの開発では後戻りが行なわれることを前提するべきであり, そのような見積もりにより計画や進行が行なわれなければならなかった。また, 製品が最終的に完成した段階でなければクライアント側が要求しているものかどうかがはっきりしないような方法ではなく, 早い次期にプロトタイプレベルのシステムを構築し, それを修正して完全なシステムを徐々に開発していくスパイラル方式を採用すべきであった。

・仕様書フォーマットの明確化計画の段階で仕様書のフォーマットを明確にしておくべきであった。

ある特定のフォーマットにより仕様書を記述するという事が行なわれずに, 個人個人の書き方にまかせられていた。その為にモジュール毎の記述が違っており, 全体的な構成などをうまく把握できなくなってしまったのである。その結果, 正確にモジュール間のインタフェースを記述することができずに, 後でモジュール間の食い違いが生じたのである。lisp のようなあまり使用されていなく, 決められた仕様書フォーマットが存在していない言語であったとしても, クライアント側が要求している項目とこちら側が提出できる項目をあげ, 統一的なフォーマットを規定する必要がある。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

現在オブジェクト指向を利用しているが, 他の手法と比較することにより, 相対的にオブジェクト指向を眺めたかった. そこで今回は JSD 法を選択した. そして, OO と比べて良い点があれば取り込むことができればよい.

(Q2) 期待は満たされたか?

まだ完全に JSD 法について理解できた訳ではないが, 取っかかりが相当つかめたと思う.

(Q3) 設計に意識の変化はあった?

1 つの手法にばかりこだわるのは良くない.

(Q4-1) 他人とのコミュニケーションについて

どっちみち他人とコミュニケーションを取らなければならない. コミュニケーション技術をみがく必要が私の場合はある.

(Q4-2) 短時間での, 問題の理解と作業について

問題を完全には理解できないが, その為にかえって新しい視点が出る可能性がある.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の仕事の進め方に対して, 新しい視点が持てるようになった.

(Q6) 全体討論の内容に関して

(記述なし)

(Q7) その他, W.S. に参加して良かったこと

ある特定の手法を身につけることのみが重要ではない. どう自分の仕事に取り組むかが大事.

(Q8) W.S. の悪かったところ

(記述なし)

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

眠い. つかれた.

30 水谷 公一 (みずたに こういち)
性別：男　年齢：26　血液型：A
会社：(株) 高岳製作所
所属：開発部システム技術開発課

I. プロフィール

(1) **現在までの主な仕事の内容について**

- UNIX 下プログラム開発 (C 言語)
- プログラム設計
- システム管理

(2) **現在利用している開発環境について**

- UNIX(SUN) + X 端末
- Open Look & Motif

(3) **これまでシステム設計に採り入れてきた技法など**

- 構造化設計
- PAD
- HCP
- 構造化プログラミング

II. ポジションペーパ

(4-b) **設計に感じた「うらめしさ」について**

1 距離のうらめしさ

現在自分は UNIX をベースにしたコンピュータを対象にしたあるシステムの構築に従事しております. 業務内容としては, 同システムの設計・開発, ワークステーション 2 台・X 端末 10 台のシステム管理, 外注さんの管理・開発依頼等です. 私どもの開発チームは社員 6 名という小さなチームです. その割には大きなシステムを開発しておりますので外注さんを使わざるを得ません. そこで現在私どもは, ある東京のソフトハウス (以後便宜上 A 社と呼ぶ) に仕事のお手伝いをお願いしております. A 社さんからのお手伝いは 3 名ですが名古屋の自社には派遣してもらうという形はとっておらず, 必要に応じてこちら側より出向くという形をとっております. なぜなら開発予算の上限や社内に A 社さんが入ってもらうための物理的なスペースがないためです. というわけで A 社さんの 3 名は A 社内で仕事のお手伝いをしてもらっております.

さて, ここでなぜ距離がうらめしいか書きます. 先程申し上げたとおり, 我社が名古屋にあるのに対し, A 社は東京にあります. この距離の差がいろいろ問題を引き起こします. たとえば距離の差による意識合わせの遅延化があります. 自分の所属するチームは各自専用の端末を持っており, 常時稼働しておりますので電子メール等でたとえ小さな仕様変更でも瞬時にチーム全体へ連絡することができます. しかし A 社へは距離の差があるために, そう簡単に実現できるものではありません. 連絡をとるのには, 普通どちらかが移動・FAX・電話等を利用しますが距離の差がちゅうちょさせたり, 小さな連絡ほど「ついでの時でいいや」と考え連絡が遅れたり, もしくは, 連絡内容自体を忘れる等が少なからず発生します. その結果, 両社の意識の整合いがとれないという現状があります. 先日もこんな例がありました. 自社内ではホスト内 プロセス間通信にメッセージキューを利用することを当然のように考えておりました. しかし, A 社の担当分の仕様の説明を受けたところ, ホスト内プロセス間通信にはメッセージキューを利用せず UDP/IP を利用することを前提としており, A 社はそれに対して何の疑問を感じていなかったようです. つまり長い間かけた労力が意識合わせがしっかりしていなかったため水の泡となったわけです. では, どのとうにしたら上記の問題が解決できるか考えたいと思います. 以下に意識合わせの改善提案を 4 例あげます.

1. A 社の 3 名を名古屋常駐にする.
2. 自社社員の東京出張.
3. まめな FAX, 電話.
4. A 社内に自社専用端末を置き常時ログインしてもらう.

ではそれぞれ批評します. (1) の案は不可能である. というのは A 社が名古屋常駐にならなかったのは物理的に常駐してもらう場所の問題や金銭面の問題があったためで, それが解決しない限り (1) の案は無理である. (2)(3) 案は, 今までの状況であり, 自分らはこれで十分だと思っていた. しかし, 先程説明した意識合わせができていなかった例から考えるとこれらではまだ不十分であると言えよう. そのうえ (2) の案の場合, 全ての開発のメンバーが動くわけにはいかない. どうしても 2〜3 人の代表者が出向くわけだが, その会議では, 仕様などがすぐに決定になるという場合は, 希である. 大抵の場合その会議で生じた検案事項を持ち帰り, 自社内で検討し, 再び繰り返す. これではタイムラグが生じ仕事が予定通り進むわけがありません. (4) の案は金銭面の問題が解決すればよい提案であるといえる. 現在, 自社内ではペーパレス化が促進しており仕様書, 説明書等は全てハードディスクに保存してある. 逆から言えばログインさえできれば最新の情報が手に入れられる. それにより意識の不整合いの問題も解決するであろう. 金銭的な問題が解決すればの話だが.

いづれにせよ, 開発と予算は切っても切れない関係にあるのは言うまでもない. もし採算が合うのなら会社側もどんどん予算を増やすであろう. しかし, 予算が決められたのなら, その予算内で最大の利益を得なければならない. だから金銭的な問題が解決しない場合, 結局は現在の状況のまま

運用していくしかないと言える. しかし, ただ出張の繰り返しや電話・FAX を利用するのではなく, 少し方法を変えようと思う. 出張して会議を行なう場合, 必ずお互い議題等をFAX で交換し, それぞれ十分吟味し, あらかじめ不鮮明な所は電話等で確認し, 実際の会議では内容の確認だけにする. そうすれば, 検討内容の持ち返りという事例は減り, それによるタイムラグもなくなり, 意識の不整合いもなくなるのではないだろうか. それにより, 自分の距離のうらめしさの気持ちも十分減るように思う.

III. W.S. 終了後の感想

(Q1) 期待していたことは?

(1) 設計手法を 1 つは取得し, 自分なりにその方法で 1 つシステムを開発してみる (→ プログラミングまで)

(2) 他の会社の方の考え方を知る.

(Q2) 期待は満たされたか?

(2) については大体満たされました.

(1) については, 我が 班は JSD について正確なレクチャーを受けず, 文献等の読み合わせにより 各自学習した. よって, JSD については大体の概要は把握できたと思いますが, 初期の目的は達成できなかった.

(Q3) 設計に意識の変化はあった?

JSD 法は現在職場で行なっている手法とは全く正反対である. よって, 現在の手法では解け なかった問題もこの JSD の考え方を部分的に導入すれば解決できるような気がしてきた.

(Q4-1) 他人とのコミュニケーションについて

他人の意見をまつのではなく, 自分から発言しなくては他人とのコミュニケーションがとれないということを改めて実感した.

(Q4-2) 短時間での, 問題の理解と作業について

時間が決められている以上, 心ゆくまで問題について論議できない. だからどの程度で問題を打ち切るか, どのように進めるか決定するグループリーダは必要である.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には : JSD という現在職場とは全く異なった手法であるため, 現行のシステム開発で壁にぶつかった 時でも JSD の視点から考察すれば解決すると思う.

(Q6) 全体討論の内容に関して

特になし

(Q7) その他, W.S. に参加して良かったこと

設計について新たな感激が得られた.

(Q8) W.S. の悪かったところ

あらかじめテーマ等を公開し, 参加すべきグループが分かっていたら JSD に関し予習でき, こんなにも山崎さんに頼らなくてもよかったと思う.

(Q9) 今後 W.S. に取り上げたいテーマ, 全体的感想

もう一度 JSD をテーマにあげ, 山崎さんとじっくり話しあってみたい.

02 東宮 祐一 (とうみや ゆういち)
性別 : 男 年齢 : 32 血液型 : B
会社 : (有) ネクストファウンデーション
所属 :

I. プロフィール

(1) 現在までの主な仕事の内容について

- ビジネス用統合型ソフトウェアの設計, 開発, メンテナンス
  (簡素なウインドウシステム, マクロ言語処理系, ソースレベルデバッガ, DOS 上でのオーバレイシステムなど)
- オブジェクト指向言語の設計とインタプリタ開発
- Z80 用構造化アセンブラ
- オーバラップウインドウシステムの設計と開発
- ソフトウェアマニュアルの記述

(2) 現在利用している開発環境について

- SPARC STATION, NCD 製 X 端末
- SUN OS, X11R5, MOTIF
- ASCII 版日本語 LaTeX

(3) これまでシステム設計に採り入れてきた技法など

- プログラムの構造化
- 機能単位でのモジュール分割
- オブジェクト指向設計
- オブジェクト指向言語

II. ポジションペーパ

(4-a) 設計に感じた「ありがたみ」について

私が以前に関わったソフトウェアについてお話しする。このソフトウェアは DOS 上で走行する, いわゆるビジネス用の「統合型ソフトウェア」などと呼ばれていたもので, スプレッドシートを中核とし, その回りにグラフ作成機能などをつけたものである。当初から C 言語で開発され, 第 1 バージョンは'84 年に発表された。

多くのソフトウェアの例に洩れず, これも単なるスプレッドシートであったものが, 次第にグラフ機能, マクロ言語等の大幅な機能拡張をするようになった。しかしながら, その過

程で設計上の大きな欠陥, というよりもこれは設計以前の問題かもしれないが, 開発やメンテナンスを妨げる要因があった。

このソフトウェアは, ユーザがシェル上でコマンドを投入し, そのコマンドを解釈実行することによりスプレッドシートに対して様々な操作 (データ入力, 計算, グラフ化など) を行なうものである。

数名のチームで行なわれた当初の開発では, システマティックな考え方がなく, 「設計」されていたのは, データファイルの構造と各コマンドの機能だけであった。各人は割り当てられたコマンドを実現すべく, 勝手な関数をいくつも作成し,

1. 同様な機能を持つ関数が複数個存在することになった
2. 関数の呼びだし関係がいわゆるスパゲッティ状になった
3. 拡張と共にメモリがパンクし, 一部の機能を諦めざるを得なくなった

という状況に陥ったのである。

プロトタイプでは, C 言語でなくアセンブラを使用したような小規模なプログラムであるから, 明解な設計方針などがなくても当然のような雰囲気があったに違いない。しかしながら, 結果的には誰も全体像を把握できず, メンテナンスも困難であるようなプログラムが作成されてしまった。

上記の点を改善すべく, 第 2 バージョンの計画を練っていた頃, ちょうど MACINTOSH が出始めており, 内部構造は「マネージャ」と呼ばれる機能単位に分割されていることを知り, 他の有効な方法論も目につくことはなかったので, それを真似することにした。

具体的には, 機能を以下のように切りわけ, これらのモジュールの機能を呼び出すためのエントリポイント (グローバル関数) のみを公開する。各モジュールごとに開発者を割り当て, 責任を持たせることにする。また, コマンド処理プログラム中で, 特定の機能が必要になった場合には, 「どの機能モジュールにいれるべきか」を協議決定する。

- スプレッドシート操作部
- ユーザー入力 (マウス, キーボード) 管理部
- マクロ言語解析実行部
- グラフィック描画部
- オーバレイ管理部
- ファイル入出力部
- 共通ライブラリ etc...

各モジュール中の関数の名前として

- どのモジュールに属するか
- モジュールの外で使用してよい関数か

を示す付け方を決めた。

また, 各コマンドについてもコマンドごとの実行モジュールを作成し, 機能モジュールの集合体である「核」部分と, それを「システムコール」する形で動作するコマンド郡とに分けることにした。

これらの方策に従って, 開発を進めたことにより, 以下のメリットを生み出すことができたと思う。

- 全体構造の見通しがよくなり, メンテナンスやバージョンアップの際に役だった。
- バグ (タチのいいものに限って) の特定や, その責任をはっきりさせることができた。
- メモリに一度には入り切らない数のコマンドを使用することができ, コマンドの増設も自由に行なえた。

書籍の執筆でさえ「章」や「節」といった機能単位で分割するのに, ソフトウェアで何故それが行なわれなかったのか不思議でならないが, この体験がソフトウェア開発に於ける「設計」の重要性について開眼することになった最初のものである。そして, そのインパクトは私にとっては強力なものであった。

III. W.S. 終了後の感想

(Q1) 期待していたことは?

「設計」という作業を行なったことがなかったので, どのような手法でもよいから, 一つ経験したかった.

(Q2) 期待は満たされたか?

手法についての理解が困難を極めたので, 結果的には当初の目標を達成することが出来なかった. しかし, 設計手法の一つのさわりだけでも経験することが出来たことで, 自分としては満足している.

(Q3) 設計に意識の変化はあった?

自己流のいわゆる「設計」で今までやってきたが, 「手法」にのっとってとは言わないまでも「手法」の考え方を参考にし, する必要があるのではないかと思う.

(Q4-1) 他人とのコミュニケーションについて

言葉が合わないということをまず第一に感ずる. お互いやってきていることがかなり異なるため用語とその上に成り立っている了解事項が一番の問題であった.

(Q4-2) 短時間での, 問題の理解と作業について

手法を理解しつつ問題を解決するという目的はむしろ明確であり, それに向かって邁進したので特に問題は感じない.

(Q5) 経験は, 自分の仕事に対してどのような位置づけ?

■今後の 仕事の進め方に対して, 新しい視点が持てるようになった.

具体的には : 仕事の上でもこれから「設計」ということをやらなければならないので, 大変参考になった.

(Q6) 全体討論の内容に関して

くたびれて他のグループの内容を聞く余裕がないことが残念.

(Q7) **その他, W.S. に参加して良かったこと**

予想していたより全然待遇が良かった (ホテルの部屋割り等).

試行錯誤したりするのは自分だけではない.

(Q8) **W.S. の悪かったところ**

寒い. 遊びは必要ないと思う. 宴会だけで十分.

仕様の拡張などをする必要がないのではないか? 課題を忠実にこなすことで十分に思える.

(Q9) **今後 W.S. に取り上げたいテーマ, 全体的感想**

既に「設計」ということを行なっている人には不満だったかもしれないが, 私には 4 日間有意義に過ごせたと思う.

「設計」というのは重要なテーマだと思うので何度でもやって欲しい

プログラム委員

P01 **渡邊 雄一** (わたなべ ゆういち)
性別：男 年齢：33 血液型：A
会社：(株) アスキー
所属：OA 推進部

I. プロフィール

(1) **現在までの主な仕事の内容について**

- 自社内向け事務アプリケーション開発
- その運用 (含む, 通信機, 周辺装置の維持管理)

(2) **現在利用している開発環境について**

OS Sun OS 4.1.1 Rev.B
マシン X-terminal + SPARC Station 2
ウィンドウ X11
ネットワーク Ethernet + TCP/IP
言語 informix SQL,4GL, C-shell

(3) **これまでシステム設計に採り入れてきた技法など**

あまり突っ込んで欲しくないが **データ・フロー・ダイアグラム**を取りいれたことがある。

II. ポジションペーパ

"設計以前" に "うらめしさ" を抱きつつ

1 はじめに

設計の方法論が, いくつかあることはなんとなく知っているが, いずれもきちんと適用したことがない。理由はいくつかあると思う。現場ゆえの悲しさと言えなくもない。しかし, すべてそれを「現場に設計方法論なんて･･･」と言って片付けるのも後ろめたさを感じてしまう。

そこらへんの自分なりの分析も含めて, 日頃 うらめしく思っていることを述べてみたい。

2 なぜ 設計方法論 に躊躇する?

自分自身で, 今まで仕事をしてきて, 新しい技術を素直に受け入れられる部分とそうでない部分があることに気づく。前者は, いわゆる上流工程におけるものがほとんどだし, 後者はやはり言語処理系などだろうか。

まあ, はっきり言えば, 今の仕事が楽に出来るものなら, 多少の苦労はあっても導入するが, そのうちいずれ役に立つかもしれないというものは, なかなか真正面から取り組む勇気も湧いてこない。下手に勉強を始めたところで「えっ, まだあんなことやってるの〜。もうそんな話しはすっかり廃れてしまったよ」なんていう目にあうのが恐くて, おいそれと貴重な時間を割くのは, どう考えても馬鹿らしいからある。だからこそ, 偽物ではなくて本物ならば, きちんと勉強してみたいという気持ちが心の隅に無い分けではない。

また, それと同じくらいの動機付けとして, 自分のやっていることを, 気軽に確かめることが, その習得に大きく関係して, その結果それを勉強したいという意欲がなんとか保たれることが少なく無いだろう。勉強/習得してみたいということは, 以上のようなことが壁になってなかなか進まないと言えそうだ。だから逆に, そのような習得を助けてくれるようなものがあれば, そのことの理解は比較的容易に可能だとも言えよう。

さらに, たとえある程度の理解が出来たとして, それが自分の仕事にうまく適用出来るかとなると, これがまた難しい。

3 設計 に拘 (こだわ) る理由

しばらく前 (3 年くらい前かな?) に SIGENV(環境分科会) で, プロトタイピングをテーマに勉強会をしていたが, 息詰まってしまった。

その理由の一つに, やはりプロトタイピングは方法論として固定的なものではなく, 種々の解釈/拡張が出来てしまうことが, その話題を続けていくことの壁になっていた。

これは, 何も SIGENV だけの問題ではなくて, 実際に自分の開発の問題に当てはめてみても同様である。自分の扱っている自社内向けの事務アプリケーションを考えた場合, プロトタイプの位置付けは, 実際に作成するプロトタイプに大きく左右されることは明白だからだ。具体的には, もうプロトタイプを作り出す時点で, それをモックアップとして位置

付けたくても出来ないターゲット環境の制約が少なからず存在するからだ。

いろいろプロトタイピングについて考えて思ったことは,

> 上流工程の位置付けをきちんとして, 当座の自分の仕事にふさわしいプロセスを支援する1つとして, プロトタイピングと云う行為があってもそれは構わないが, プロトタイプに固執することは, 理論的な裏付けがあるわけでもないので危険である

ということだ。

余談だが, この「プロトタイピング」という言葉を「再利用」という言葉と置き換えても何ら問題無いと思っている。この2つは80年代後半から90年代初めのキーワードとして少なからず議論されたように思うのは皮肉だろうか?

このあとすぐに, CASE ブームが到来する。この CASE で, ことごとく登場するのがデータフロー図であり, 状態遷移図だ。当たり前のことだが, 上流工程の CASE ツールは, その裏に方法論があり, それを実践してゆく過程で必要な「もの」作りを支援してくれる道具でなのだ。つまり, その「もの」が状態遷移図だったりするわけである。だから, 単にデータフローを書けるだけでは, 何も意味がなくて, その背景にあるもの (理論, 方法論) を理解した上でそれらを作成していかなければいけないのだ。

だが, 先に述べた理由で CASE ツールを使いながら, そこで作らねばならないものを理解して, 背後のエッセンスを汲み取るということがあながち出来ないわけではないようだ。

4 方法論 にすがってみたい

で, そうはいっても, 方法論がそれぞれに提唱されるくらいだから, 皆それぞれに悩み苦労しているのが『設計』なんだろう。だから CASE ツールが出現してきたとはいえ, その周辺の含めて問題は山積みされているように思う。

だが, CASE ツールの出現により, 少なくともそのツールを適用すれば, かなりの確からしさで (本来の方法論の誤解/誤用をしないだろうという意味で) その背後の方法論そのものの見定めをすることが期待される。

何故, 方法論に拘るのか?。それは, 現場にはスンナリと受け入れられる「プロトタイピング」や「再利用」ということは, 自由度が高すぎて, 体系的に物事を組み立てていかなければならないであろう設計ということには, 補間こそすれど柱になるものではなかろうと思うからである。個人的な反省ともオーバラップするが, 要は, 直観だけに頼ってプログラムの構造化だとかを決めてゆくだけで, 信頼性や生産性を伴うような仕事は, いつまで立っても出来まいという気持ちから, まずあるものを勉強して, (現場に普及しない) 疑問を見定めてから, 次のステップを決めたのでも遅くはなかろうと思うからである。その疑問とは, 以下の2つである。

1. 設計方法論をきちんと理解せずに適用して, 期待される効果をうけていないための誤解なのか?
2. 設計方法論の前提が現場の問題には, 不十分なため, 所詮, 机上の空論に終っているのか?
   → 例えば,
   (a) 好ましいエラーリカバリの設計とは?
   (b) トランザクションとデータの排他処理の関係などの問題

いずれにせよ, その答の1部でも得られれば, それは自分にとって大きな進歩である。ワークショップにおいては, その議論をしたいと思っている。で, 私のポジションを説明する前に, 方法論のうちでも最近話題に挙がることの多いオブジェクト指向への期待を, 自分のプログラミングの経験を通して簡単に述べたい。

5 設計で 問題を感じさせぬ部分

過去に開発/運用を経験してきたプログラムのほとんどが, 中規模の事務アプリケーションなので, 考察としては狭いかも知れないが, そこでのプログラム開発の量と内容を分析してみたことがある。そのときにわかったことは, プログラムの制御に関しての処理は, (事務アプリケーションとしての) 画面エディタを除いてはほとんど入力データの特定のフィールドに依存して処理を分岐させ, データを編集しているだけだった。そして, 80年代後半には, これらにターゲットを絞ったパッケージソフトが少なからず現れてきたことは, 承知の事実である。そのようなもののプログラムの工数は, 従来の Cobol ベースのものと比較して, せいぜい3倍程度しか開きがないが, 実際の信頼性と保守性は, プログラマの精神的な徒労をも考えれば10倍以上の開きがある。

すなわち, 中規模のシステムであれば, 全体を大まかに捉えてから, 要所を機能で分けてみる。それらを比較的容易にバッチ的にデータの受渡しが行なえるように出来ればしめたもので, それに設計方法論を論じてもしょうがないという気持ちが少なからずした。

6 オブジェクト指向 への誉れ

これに対して, エディタを作成するようになると状況は一辺した。特にエディタは,

1) そのターゲットとして使える環境によって, 提供される機能が千差万別で, そのことをきちんと調査/理解したうえでなければ, そのプログラムの制約を設定することが出来ず

2) その具体的な制約によって, 使いやすいユーザインタフェースを備えた画面は設計出来ない

ということがわかる[†4]。さらにこれに加えて実際にプログラミングを始めると,

3) 画面の遷移を, エラー/割り込みを考慮しながら記述することが, 簡単に機能分割出来ない
4) ある程度 3) が出来たところで, いわゆるデバッグや修正に対して個々の画面に依存したところと, そうでないところとの切り分けが難しい
5) 処理によっては, 排他制御を考慮せねばならず, それを怠ると, 最悪の場合にはシステム全体を停止させることにもなりかねない

という問題に直面した。

このような状況を, ちょっと冷静に振り返ってみると画面に直接現れないが, 画面の遷移 (=処理の遷移) を制御する情報の管理が, プログラム全体を極めて不透明なものにしている。逆に, その部分を, 素直に記述できるメカニズムとして通常の手続き型のプログラムには, 持ち合わせておらず, 拡張性などの面から限界を感じる。それに対してオブジェクト指向では, メッセージを持ってある種のプロセスが通信しながら全体を管理しているようなメカニズムは (そのスーパバイザ的な部分の記述を除けば) 極めて容易に出来るように直感した[†5]。

7 オブジェクト指向設計 に拘る理由

だが, 実際にそのような言語や環境が実際の運用環境に早急には望めない以上, そのメカニズムをよりメタな設計の場で利用出来ないだろうか?というのが, 本音である。

その勉強を SIGENV の場で始めたのが今から 2 年位前からである。当時さほど議論されていなかったオブジェクト指向設計/分析について, Meyer の本のペーパバックも入手できたので勉強を開始した。また, その後から Yordon の本を始めとして, 多くの文献が登場してきた。以来, 言葉では良く聞くようになったし, 本も良み進んでいくものの, 今1つピンと来ない。オブジェクト指向の話しを聞くと, 聞いたその時は感激し, すごいと思うのだが, いざ会社に返って自分の机で仕事を始めるとそんなことはどこへやら…。ことのつまりは, オブジェクト指向を理解出来ていないから, 設計が出来ないのである。

このような, 個人的な葛藤はあるもの, やはりオブジェクト指向は魅力的である。そして私自身の勉強を助けてくれるようなガイドブックや教科書も少なからず登場してきた。具体的には, 4 月位から読み始めた Budd のオブジェクト指向プログラミングの教科書 [1] では, 具体的にその分析/設計を進める方法として CRC カードの利用が具体的に紹介されている。

8 現実に立ち返ってみると…

さて, 今の仕事に立ち返ってみると (つい先ほども, Makefile を書き換えそこなって make clean なんかしたもんだから, 主要な source を見事に消してしまって, 回りの人たちに多大な迷惑をかけて落ち込んでいることも含めて), おいそれと夢物語を話していても仕事は進まない。

だが, 字数の制限もすくなからずあるので, ここでは, 今まで述べたことに関連して重要と思われる事柄をあげるにとどめる。

a) 大変なものは, 限りなく大変だ

一言で, 要求仕様などと気易く言うが, 実際のところそれを完全に行なったらプログラムの開発など, いつまで立っても出来ない場面が少なくない。

明らかに, 現実には「時間/費用との戦い」という試練があるので 1) と関連して完全な設計が出来なくとも, どの程度までいったら良しとするのか?

b) 保守というツケ

保守ということを, どうとらえどう位置付けて設計にフィードバックさせるのか誰も議論していない

c) 拡張性という高望み

利用者が変更/カストマイズ出来るアプリケーションをサポートしたい

d) 設計という抽象的な作業を, 具体的にどう表現すれば良いのか?

まあ, 端的に言えば『仕様書』である。

これは, プログラミングまでの 設計内容を他人にどう伝えるのか?という一種のコミュニケーションの手段としてとらえるのか?, システム完成後も尊重しなければいけないバイブルのような存在とみるのか?

e) 制約の表現

ターゲットへの依存

制約 (画面のカーソルの動き) と好ましい MMI

トランザクション/排他などの制御 と アクセス権の管理

9 個人的な うらめしさ

設計方法論の大前提は, 要求定義がされたものを素直にモデリングして実態 (object) あるいは機能 (function) 毎に

†4 この部分については, プロトタイピングが少なからず有効であることは賢明な諸氏の指摘を待つまでもないだろう

†5 だが, 排他の問題はオブジェクト指向で, きちんとうまいこと問題解決しうるのか, 自分自身で見究められていない

分けて, プログラミングするための情報に整理していることだと思っている。

設計方法論を論じるには, このような前提を設けた考察/議論は致し方あるまい。だが, 自分が 1 番不満に思っているのもそこである。

すなわち, 要求定義を誰がやるのかで, そのシステムの良否が決定されることも少なくない。これはシステムの大小には関係無い問題だろう。多くの場合, 具体的には良いシステムか否かの別れ目は, 要求定義の裏にあることがらを, いかに整理して仮想的なメカニズムをきちんを組み込んだモデルを考えて要求定義を考えるか?, そしてそれをシステム設計に反映するかにかかっているように思う。

にも関わらず, 要求定義/機能設計に関して残されるのは, その検討の結果のみであって, 過程ではない。それが, それら作業を評価することを阻害している重要な要因であることに, 多くの人が気付き始めた。

システムの出来の如何は, 設計だというが, その多くは (プログラミングのレベルで言えば構造化された言語の積極的な利用やライブラリなどの充実によって) 解決の手当がされているようにも思える。たまたま悪いプログラミングを出会っても, それをすべてに渡って書き直すことは稀であり, そのプログラムの何割かは, 利用するものである。その意味で, 本当にうらめしく思うのは, そのようなことをする気力すら奪い取ってしまうほど, ひどい設計であり, そのようなことは往々にしてきちんと要求のモデルを確立していないことにあると思われる。その意味で, 私個人の興味の対象は, 設計方法論よりも, 要求分析の段階から要求定義までなのかもしれない。

10 まとめ

本文で十分に表現/説明出来なかったが, 私のポジションをまとめると以下の通り

1. 方法論を持ち出せる場面を作る努力が必要だ
2. そのためには要求仕様のモデリングこそ重要な問題だ。
3. それが出来たとして仮定して, 複数の手法/アプローチを個々の分解した機能単位に適用するための判断基準を明確にする必要があろう
4. そのための勉強/経験不足は素直に認めよう
5. だが, 問題の難易度によっては,
   - きちんと方法論を適用して効果を得ることを期待する

   場合と
   - ある程度まで方法論を適用しても, 諸般の制約からそれに振り回されることなく, 物作りを始めなければならない

   があろう。後者は, 概念的には邪道だが, 現実には無視出来ないので経験を積んで, 出来る限り多くのファクタを規定出来るように努力が必要だろう
6. 構造化などの多くは, 従来のプログラミング言語でも出来ているので, そのさきの技術を適用していかない限り問題の解決はおぼつかないだろう。
7. そのためには, 現実のシステム化の対象であることがらと, 直感的にも素直に採り入れられモデルを採用している, オブジェクト指向を設計にいかせると良さそうだと思われる場面が少なくないので, これに期待している。
8. 現実の設計の問題の多くは, 仕様の変更などから, それを設計に反映させるのではなく, 設計の成果であるプログラムに対して行なっている。このことが, 設計を空虚なものにしている。

【参考文献】

[1] Timothy Budd : An introduction to Object-Oriented Programing, Addison-Wesley, 1991 (ISBN:0-201-54709-0)

P02 **塩谷 和範** (しおや かずのり)
性別：男　年齢：41　血液型：O
会社：(株)SRA
所属：ソフトウェア工学研究所

I. プロフィール

(1) 現在までの主な仕事の内容について

- 数値計算アプリケーションの保守
- 広域分散ネットワークの通信プロトコルの設計
- オンラインバンキング・アプリケーションの開発/環境整備
- Unix ツールの開発
- プロトタイピングツールの研究開発

(2) 現在利用している開発環境について

- ハードウェア関連
  - メインフレームコンピュータ (DIPS-II, 富士通 M, 日本電気 ACOS, 他)
  - ミニコンピュータ (沖電気, DEC)
  - ワークステーション (Sun, SONY, PFU, Tektronix, Fuji Xerox, NeXT)
  - パーソナルコンピュータ (Macintosh)
- 言語 (fortran, Asm, COBOL, Sys-L, Shell, C, Smalltalk, 他)

- 開発支援ツール関連
  - Smalltalk-80 は開発環境を含んでいる。
- それ以外はほとんど手作りでやってきた。
- オンラインシステム開発の時期には, データジェネレータとかオンライン環境シミュレータなどを使った。
- Unix 上には有用なツールが沢山ある。
- NeXT のバンドルされている開発環境は特に優れている。

(3) これまでシステム設計に採り入れてきた技法など

- フローチャート
- ディシジョン・テーブル
- 疑似コード
- HIPO
- HCP(NTT が作成したフローチャート手法)
- 状態遷移図
- タイミングチャート
- データ・フロー
- イベント駆動設計
- 素朴な構造化設計
- 素朴なオブジェクト指向設計 など

II. ポジションペーパ

現在設計について感じていること

設計が恨めしいと感じるのは, たいていの場合現在行なおうとしているバグ修正あるいは機能変更が, 設計時に考慮されていない事に気づいた時である。これは限られた時間で要求分析・定義を行なわなければならず, その結果, 不完全な要求定義をもとに設計を行なうことに起因する。また, 設計に要する時間も限られる為, 設計書に必ずしも記述されない仮定と制約条件をもとに設計が行なわれることにも起因する。この不完全さを減らす為に, きちんとした設計手法を導入し, それに従った分析設計を行なう事は事態をかなり改善させる。当然その手法を支援するツールなり環境なりと併せて用いる事により更に大きな改善が期待出来る。しかし, ツールと環境を用意すれば事足れりとするのは甘い。実際に考え, ツールを使うのは人間であるから, 設計者が対象システムを完全に理解し, かつ適用しようとする手法について熟知していなければ場合によっては逆効果にもなり得る。
同様に, 要求分析・定義時にも同じ事が言える。しかし, 設計時にもまして難しいのは, これを支援する適当なツール・環境がほとんど無いし当然普及もしていないと言う点にある。いずれにしても要求分析という過程は,
要求を出すユーザの考えを,

(1) 引き出し・分類し・関連付け・類推し・分解し・抽象化 (モデル化) し,
(2) ユーザに理解出来る形に形式化し,
(3) 提示して確認を求め, 追加・修正があれば更に分析 (1-3) を行ない,

確認が得られたら,

(4) 設計者に理解出来る形に文書化する。

という過程となる。

この (1) の過程では, 適当な分析手法を用いることができ, (2), (3) の過程では適当なツールを使って効果を上げることが出来る。しかし一番重要な (1) の分析過程では, 大部分の作業が分析者の思考そのものによって行なわれ, 実際の所, 分析手法はその思考の為のガイドラインにすぎない。実際の思考過程は, 一定の決まった処理をしているのではなく, (1) に上げた作業を過去に蓄積した知識と経験及び勘を駆使して, 半ば無意識の内に同時並行的に行なっていると考えられる。
であるとすれば, 優れたガイドラインとしての方法論・手法は, それを体得するまでに, 大きな誤りをなるべく避け, 体得するまでの期間を短縮するという効果はあるが, この時期を過ぎてしまえば特別意識せずにこれを使いこなすことが出来る様になる。しかし, その様になっても (1) の過程が複雑でありかつ間違いを犯し易い, という事態は変わらない。これを解決する為に, 色々なツール・システムが提案・開発されているが, 未だ古典的な紙を使ったツール・ブレーンストーミングなどコンピュータを主体におかない手法に及ばない。加えてこの過程は自動的に行なわれる為, 記録として残す事が難しい。したがって, この様な経過情報無しにシステムを理解する事はしばしば極めて困難な作業となる。

結局, 良く言われている人間の思考を支援する環境・システムが実用化されないと, 大きな飛躍は望めない様である。しかしながら先にも述べた様に, 個々の過程に適切な手法を導入し, かつこれを支援するツールと併せて使う事によって, 更にこれらの過程が, 一貫した手法と環境によって一体化される事によって現実的な効果は得られる。
したがって, 分析・設計者が習得し易く, 使い易い, 小回りの利くツールと, 分析からプログラミングまでを支援する方法論・手法が, より完全な分析・設計を行なう為に役立つと考える。

この為の手法として, オブジェクト指向手法に注目している。また, (1) の過程を支援する可能性のあるツールとして, ハイパーメディアの紙を越えた実用化に期待している。これにより, 従来残す事が難しかった分析・検討過程の記録と検索 (もしツールを使う方が楽になるならば), 及び関連付けが出来る様になると期待する。更にこの機能に加えて類推・分解・抽象化 (モデル化) 機能が有機的に組み合わさる

事によって, 人間の指向支援ツールとして使える様になる事は, そんなに先の事ではない様に思われる。

P03 伊藤 昌夫 (いとう まさお)
性別：男　年齢：33　血液型：AB
会社：MHI エアロスペースシステムズ (株)
所属：技術部 開発グループ

I. プロフィール

(1) 現在までの主な仕事の内容について

- 数値流体力学関連の研究 (4 年)
- フライトシュミレータの開発 (3 年)
- 社内情報システムの開発 (2 年)
- CASE ツールの設計 (2 年)

この世界に来る前は, 自動車の駆動系の設計をしていました.

(2) 現在利用している開発環境について

- ハードウェア：Next(平均 8H/日) HP9000/750(平均 1H/日)
- 言語：状況に応じていろいろ.
- ツール：自作のものと, HP に関しては IA(インターフェイスアーキテクチャ)

(3) これまでシステム設計に採り入れてきた技法など

特に名前のついたものは使っていない. 上記 2. について云えば, SA・S D 的なアプローチであるが, 同じではないし, 3. について云えば, ER 的なやり方である.

II. ポジションペーパ

ソフトウェア設計に対する一考察

– "設計手法"が浮遊する或る知識の海 –

1 ソフトウェア設計

ソフトウェア設計を考える上で, 設計手法から入るのは望ましくない。それは, ソフトウェア設計と云う主題に対して, 考えようとする力を弱くしてしまうからである。

我々は, 漠たるシステム要求から, プログラムコードによって細部迄記述されたソフトウェア製品とドキュメントを作る。この活動をソフトウェア設計と, ここでは定義する。

ここで, 問題にすべき点が 2 点ある。第一は, 対象が何かをどのようにして把握するかと云う要求定義上の問題。二番目は, どうしたら正しく細部記述レベルまで落とすことが出来るのかと云う本来の設計に関わる事柄である。

(注) ここでは, システムの把握である要求定義も又, 設計の問題として捉える。一般に「設計」という場合, システムを特定し要求を明確化することも含まれると考える。又, 対象を明確化する過程なしに, その実現方法を検討・決定する設計過程を説明しにくいので, ここでは両者を取り扱う。

2 如何に対象を定義するかと云う問題

いわゆる「何を (what)」を定義する過程である。

最も良く用いられる方法が, 分割統治の考え方である。人間個人の短期的な思考能力には限界があり, 一時に考える量を制限したい。従って, 問題を把握出来る量まで分割していくことによって, 分割された部分について正しく対象を定義する (要すれば更に細分化する)。最終的に部分の和としての全体が, 構築しようとするシステムを表しているという考え方である。

この方法には次の様な注意点がある。一つには, Jackson が云う様に我々は本当に分割が正しく出来るのだろうかと云うことである。分割出来るのはシステムが完全に分かって初めて出来るのではないかと云うことである。

比喩を用いると, 人間の体は手や足や頭と云った部分からなっている。それらを更に分解し続けると細胞レベルまで落とすことが出来る (もっと分解することも勿論可能だが)。しかし, そのように分解出来るのは, 詳細が, つまり細胞が何らかの手段をとることによって, 実際に目で見えるからである。従って, 分割を行うためには, 対象が見えてなければならない。見えてない場合は, いったん何らかの形を, 作り上げてその差分で, 繰り返し分割をやり直す必要がある。

もう一つは, 分割された部分を組み上げると, 本当に元のシステムを理解出来るのかと云うことである。例えば, 「SA」では上位の DFD を眺めれば, 少なくとも対象システムの機能が分かり, 大きなデータの流れが分かる。しかし, それは作り上げようとしているシステムを完全に表現し, 誰でも容易に理解出来るものではない。

ここでは, 分割統治について考えた。「SA」や機能分割法では, これらは階層構造により表現される。他の手法では, 必ずしも階層構造を用いない。例えば, オブジェクト指向分析では, 多くの場合構造があるだけで, 階層表現はなされない。(coad/ yourdon 法に於ける subject は階層構造に類似しているが)

階層構造は, 一枚が理解しやすい数 (たいていはマジックナンバ) で表現されるが, 一時に全体が見通せない。逆に階層構造を用いない場合, 対象が大規模なシステムであると表記は確実に複雑になると云う問題がある。

システムは一つの視点から捉えると, 捉えられ表現された

ものについては良く理解出来る。しかし, 作り上げようとするシステムは本来, 多次元の世界に浮く超球であり (データフローの軸, 機能の軸, 制御の軸, 関係の軸 ‥), 一面が理解出来ただけで全体が理解出来る訳ではない。

3 如何に細部記述レベルに落とすかと云った問題

いわゆる「如何に (how)」を決定する過程である。

定義された what に対して, 如何に作り上げるかを決定する。そのための, 手引き書が白紙の状態からスタートすることは少なく, 過去の開発事例や自らの開発経験を参考にすることになる。(敢えて, 設計手法や開発標準を前面に出さない) この過程のうち下位に相当するモジュール分割以降については, 比較的整理されている。例えば, ローカルなアルゴリズムの選択や, データ構造の決定がそれにあたる。ここでは, モジュール分割に至る迄の, 抽象度の高いレベルでの「如何に」の検討について考える。我々は, どういった点に注意すべきであろうか。

1. 開発しようとする対象が殆ど同じである場合
   基本的には同じ方法を踏襲すれば, システムは出来上がる。しかし, そこでとられている方法が最適かと云う観点からすると必ずしもそうではない。反面教師としての参考にしかならない場合も多い。しかし, 考える上での一つの軸になることは確かである。
2. 対象が異なる場合
   一般的に, 相違点が大きくなるにつれて, 流用の困難性は非線形に増加する。何故ならば, 機能で分割するにしろ, データで分割するにしろ, オブジェクトで分割するにしろ完全に独立の部分を作り上げることは, 殆ど不可能であり, 一部の変更は, 程度に差こそあれ, 他の部分に影響するからだ。
   例を挙げる。ソフトウェアの構造は, 必ずしも物理的制約から完全に切り離された所で, 設計出来ない。そうした場合, 例えばユーザから見た品質特性の一つである効率性を向上させるためには, ソフトウェアの構造もある程度, 対象に特有の部分が大きくなる。(この例に関して云うならば, ハードウェアの向上が, 物理依存性を弱くすると云う議論もあるが, 現実に於けるユーザ要求はそれに比例する現実がある。)

従って, 我々が「如何に」を決定する時, 過去の設計結果が必ずしも完全に役立つ訳ではなく, 何らかの別のアプローチが必要になることが分かる。

その最も有効なアプローチが, 様々な局面での思考過程を抽出することである。設計のそれぞれの局面での前提条件と内部状態から, 結論に至る過程が動的な知識として蓄えられる。この知識こそが, あらゆる設計過程に於て適用可能な汎用的な知識となり得る。

ここで云う知識の集合体は, "設計手法"ではない。もっとダイナミックなものである。この知識を設計手法が持つ静的で表象的な知識と区分するために, 設計手法としてのレシピ (調理法) から料理を生みだすテクニックと考えても良い (或はベルクソンの云う, イマージュ的なものである)。勿論, レシピなしに知らない料理を作るのは困難である。しかし, レシピがあるからと云って, 良い料理が作れる訳ではない。ある料理を実際に作る時には, 様々な制約がある。例えば香辛料が一つ足りないとか, 名古屋コーチンを使う所を, ブロイラでごまかしたい云ったことである。更に, これら制約がない最高の条件で作るにしても, やはり作る人間によってはひどいものが出来たりする。

つまり, レシピとしての設計手法は明らかに有用であるが, 実際に現場でそれを適用するにはもっと別の, 経験等からダイナミックに得られるイマージュ的な知識が必要となり, 恐らくその知識こそが, 最も基本的なものである。

4 我々にとっての設計手法とは

前章で, ダイナミックな知識の重要性について考えた。では, 再度「SD」や「JSD」と云った設計手法について考えてみる。

これらは, ダイナミックな知識の内, 共通項としての最低限のものを抽出し乾燥させた干物のようなものと考えることが出来る。従って, その干物を生かすためには, 再度細かな我々の持つダイナミックな知識をつけ加え, 生きたものとする必要がある。

それが, 前章で考えたイマージュ的知識となる。

設計手法を現場に適用するには, その設計手法が成熟していなくてはならない, 幾つかの利用経験が必要だとか云われる。前者は, 設計手法が, 干物として不十分だと云うことであり, 後者は, 干物を戻すのに, 戻し方をイマージュ的知識 (その一部は経験でしか得ることが出来ないが) を知る必要があると云うことだ。

5 現時点での現実的な対応

前章までで, 現在の設計に於ける幾つかの問題点を挙げてきた。その内大きな次の二つの問題をとりあげる。

(ア) システム全体を表現するのは難しい。

(イ) 乾いた設計手法をどう適用するかが難しい。

これらの問題点に対して, 完全な解は現時点ではない。しかし, 現実的には以下の手段をとることで, ある程度の対応が可能と考えている。

5.1 模造紙の利用

初期の段階では, 模造紙の様な大きな紙を使い, 階層構造を同一平面上で表す。これは (ア) に対する対応策である。

模造紙を使うメリットは, 単に面積が大きいということの他に, 数人の人間で議論出来る (イマージュ的知識の共有), 手書きすることで, 画面上で描く以上により豊かな表現 (濃淡, 線種等) が可能となる。それらにより, 頭の中の考えをある程度素直に写像しやすい。

このやり方の問題点は, 第一に複写がとりにくいこと。次に, 最終的には, 標準語としての何らかの表記法に変換することになるが, 手書きと電子媒体では変換が困難であることである。

5.2 ガイドブックの作成

次に, 設計活動そのものについては, 細かな事例を集める。どういう条件で, どういう問題に行き当たったのか。どうやって解決しようとしたのか。これらを収集・整理してして, 我々の, 未知のソフトウェア世界を歩き回るガイドブックを作る。これは, (イ) に対する対応策である。

この方法の問題は, 如何に収集し, 如何に整理するか, 如何に検索するかと云うことである。しかし, 躊躇からは何も生まれないし, 書き留める行為そのものが, 少なくともその個人にとってはイマージュ的知識の獲得につながる。(イマージュ的知識とは本来は表象によって完全に捉えられない知識であるが, その一部は何らかの形, 例えば, 細かな事例-エピソード-と云う形で整理可能と考えている。)

6 最後に

何らかの設計手法を使っていますか, と問われた時に現場の多くの人は「いいえ」と答えるのではないかと思う。このことには, 二つの側面があると思う。一つには, 会社で標準として定められたやり方があり, それが満足出来ないものにしろ, 使っているが, 特に名前がついていないので設計手法を利用していないと考える場合。

もう一つは, 著名な設計手法が, エキスのみを提示していることによって無味乾燥で使いたくないと最初に思ってしまう場合である。良く云われるこの, 「研究」と「実践」のギャップはソフトウェア生産工学と云うべき生産の技術についての議論を深めるか, もっと気軽に (例えば, CASE を買わないとS A・SD できないと云う意識をなくして), 設計手法を利用し成熟化させる方法論のようなものが議論されても良いと思う。

更に云うなら, Song らが取り組んでいる設計手法の分類と云ったものが, 問題領域との関係の中で明確になれば, より使いやすいものになっていく。勿論そのためには, 最低限必要な各問題領域に適した設計手法が存在し, 明確に定義されていなければならない。

レシピとしての設計手法が, あらゆる問題に対してたった一つしかないと云うことはないし, 最適なレシピをなるべく短い時間で捜し当てたい。

ソフトウェア工学は若い分野である。単純に他の工学の分野と比較するのは危険であるが, 例えば機械設計と比較すると, ある意味でのイマージュ的な知識の集積が不足している。しかし, 成熟していない部分だけ, 設計と云った工学に於ける基本的な活動に対して, 我々はかなり前向きに取り組んでいるのではないかと思う。

P04 **岡本 隆一** (おかもと たかかず)
性別：男　年齢：36　血液型：B
会社：(株) さくらケーシーエス
所属：オープンシステム部 システム第一課

I. プロフィール

(1) 現在までの主な仕事の内容について

- 受託計算センターの運用管理
- 受託計算センターの運用管理システムの設計/開発
- 上水道事業管理システムの設計/開発
- NC 群管理システムの設計/開発
- 地方税オンラインシステムの設計/開発
- 都銀オンラインセンターの運用管理システムの設計/開発
- 社内 LAN, 対外ネットワークの環境整備
- 情報処理教育支援システムの設計
- 社内開発環境の整備
- ソフトウェア設計支援環境の設計/開発 (プロトタイプ)
- 文書処理環境とデータベースの連携ユティリティの設計
- 配電設計支援システムの設計 (プロトタイプ)
- 地図処理システムの設計
- 諸プロジェクトの管理, およびシステム営業

(2) 現在利用している開発環境について

- ハードウェア関係
  - 大型メインフレーム
    : FACOM 230-58
    : FACOM M シリーズ
  - ミニコンピュータ
    : PFU U シリーズ
    : PFU A ファミリ： FACOM S3000 シリーズ

- ワークステーション
  : Sun, PFU, Xerox, SONY
- パーソナルコンピュータ
  : FM R シリーズ
  : Macintosh

- 言語
  : ASM
  : Fortran, COBOL
  : C
- 開発支援ツール関係
  - 大型メインフレームでは,
    : データジェネレータ
    : チャータ (フローチャート, ランチャート)
    : ハードウェアトラブルシミュレータ
  - ミニコンピュータでは,
    : データジェネレータ
    : デバッガ
    : クロスコンパイラ
  - ワークステーションでは,
    : UNIX 上のコマンドツール等
    : DTP 環境
  - パーソナルコンピュータでは,
    : エディタとワードプロセッサくらい....
  - メインフレームやミニコンピュータ上では, ほとんど手作りまかないでした。 私用のジャンクが役にたった。

(3) これまでシステム設計に採り入れてきた技法など

- メソッド等
  : JSP/JSD
  → SA/SD
  → 富士通 (SDEM)
  → プロトタイピング
  → 何でもあり...!
- 技法等
  : フローチャート, HIPO, PAD, YAC チャート
  : ディシジョンテーブル
  : データフローダイアグラム, 状態遷移図
  : タイミングチャート
  : プロトタイピング
  : 等...,

II. ポジションペーパ

現在, 設計について感じていること

1 設計に関する基本的な考え方

プログラミングやシステム設計に関する手法・技法のスタンダードとして「JSP」, 「JSD」は過去に専門教育を受ける機会に恵まれました。 「3. 」に前述したその他の技法は独学...., というより (飯を食うために, 徹夜を少なくするために) 我流で身に付けたものです。

オブジェクト指向分析・設計がシステム設計・ソフトウェア設計の技法として注目されている昨今ですが, 自分の「設計」に対する基本的な考え方は構造化分析・構造化設計がベースになっています。 また, システム開発手法としては, ウォータフォール方式にプロトタイピングをミックスしたやり方に馴染んでいます。

どこにでもいる...., メーカ系の技術屋という位置付けになるのでしょうか。 ただ, これはあくまでベースで, 現実の「設計」に対する取り組みでは「何でもあり」を信条としています。

「設計」という仕事に対して私は次の「a」～「c」のような意識で取り組んでいます。

a. システム構築の要件・要望を分析し, その結果をユーザが理解できる様式に形式化する。
   : 要求分析・要求定義
   : ユーザインタフェース設計

b. 「a」の成果物を他の設計者が理解できる様式に抽象化する。
   : システム構造設計 (処理設計→機能設計)

c. 設計者は, システムの仕様をプログラマが理解できる様式にマッピングする。
   : プログラム構造設計

「a」では, これから設計するシステムに要求される機能とその実現方式, 要求機能の実現に対するユーザの期待効果, ユーザによるシステムの利用像等, 主としてシステムの全体像を設計者が理解・把握し, かつ要求システムに対する設計者の理解度・把握度をユーザが確認するための作業として行います。

「a」では, 機能実証, 実現様式の検討等のためプロトタイピングを多用しますが, 最終プロダクトは殆どドキュメントです。 システム完成後の総合解説書をイメージして文書構成を行うようにしていますが, もの書き中心で, かつ定型的な文書処理でなく, しかも書き物のアップデートを繰り返し繰り返し行う作業なので若手には苦痛のようです。

100%を目指すのは難しい作業ですが, 私にとっては「設計」という仕事の中で一番好きな領域です。

「b」と「c」は主としてモデリングとマッピングの作業

として行います。 本設計作業は, 同業各社それぞれに社内標準等を設け, 設計書・仕様書の標準化を推進し, 取り組んでいることと推察します。

さくらケーシーエスでも同様ですが, 私の所属部門ではまだまだ標準化への対応が遅れています。 ガイドラインの提示が充分に行われていないわけですから, 若手を苦しめている現状です。 私本人にとっては, 採用するモデリング技法・マッピング技法が決まれば後の作業が単調になるので, あまり好きな作業領域ではありません。

標準化が遅れる諸悪の根源は岡本だという説もあります。

「b」でもプロトタイピングを適用します。 適用目的は性能実証が中心です。 ところで「b」と「c」は, まる3年くらい自分自身でやっていません。

これまで(自分で)行ってきた「設計」を振り返ると, かなり色々と入れ込んで試行錯誤した時期がありました。 例えば, 全てのシステム仕様をモデリング技法とマッピング技法だけで表現しようと試みたり, 関数型仕様記述表現によって同じような試みを行ったり...., 現在では, 直接に現場作業に携わることも稀になり「a」～「c」に落ちついています。

2 設計関する悩み

現在, 私のところで直面している「設計」に関する最も大きな悩みは, 設計に対応できる技術者が少ないことです。

私の担当部署の作業分掌は「主に UNIX ワークステーション・ベースのエンジニアリング系業務システムの設計および開発の取り扱い...., 」ですが, 若手中心の組織でもあり, 前述の「(1)-a」, 「(1)-b」が辛くなっています。

「設計」を行うためには, そのための技法・技術を修得しなければなりませんが, 技法・技術を修得しただけでシステム設計に従事することは困難です。

「(1)-b」, 「(1)-c」では, 必要と思えるレクチャ等も含めて作業のガイドラインを用意すれば, かなり悩みの解消につながるかなと考え, 標準化の推進に意欲は感じていますが「(1)-a」の場合, 場数も含めて設計者の経験と情報収集量と情報整理の能力に依存するところもあり, 頭の痛いところです。

3 設計にうらめしさを感じるとき

ソフトウェアの仕事ではメンテナンス作業の増加が顕著になっています。 過去の設計の不行き届き (設計者の不行き届きもある) から再設計が難作業となり辛い目にあった技術屋さんは多いと思います。もちろん, こんな時には「うらめしい」。

膨大な紙の山を作りだすような設計作業を当たり前とする設計標準にも「うらめしさ」を感じます。

システム設計は, 本来, 何のために行うのか....?

この問いに対して答えにならない現場が見えたとき, 設計そのものではなく, 設計をそのように位置付けてしまう何かに「うらめしさ」を感じます。

一例ですが,

最近では設計支援のためのツールが数多く製品化され, 使える (使えれば...., と思う)CASE ツールも幾つか見当たります。

本来「(2)」の悩みの解消のためにツールの適用を考えたいところですが, こんなところに (変なところですが)「設計に対するうらめしさ」を感じることがあります。

※コストだけが「うらめしい」わけではありません。

高価ではあっても, そのコストに見合った効果が期待できると考えている CASE ツールは見当たります。 が, CASE ツールの生成した設計プロダクトを認めて頂けるお客様がとっても少ない。

高いお金をかけて CASE を導入しても設計成果物の書き方の違い等で成果物が認めて頂けない (成果物の品質の良否を問わずに)「設計」本来の目的とは...., と講釈しても通じません。

ちょっと脱線しますが,

システム設計に対するコンピュータ支援は, CASE ツールに到らないまでもワードプロセッサや DTP のレベルでは広く普及しています。 しかし, 考えようですが, 僅か 10 年以前 (自分自身がシステム設計を手掛け始めた頃) では, 設計者は, 自分の体ひとつと設計定規・鉛筆と消しゴムで仕事になっていたのに, 最近ではワープロ仕立てが当たり前。何と高価な鉛筆と消しゴムなんだろうと思うのは私だけでしょうか。

こんなことを考えている設計屋さんも少しはいるんじゃないかと思いました。

4 設計支援環境に対する期待

テクニカルなステートメントに欠けるポジションペーパになりましたが, そのあたりは岡崎で話題にできるかな...., と考えています。

システム分析からプログラミングまで一貫した考え方のもとで手法と環境の実現に期待できるオブジェクト指向技術には手法・技法習得の容易性と設計環境の高度化, ポータビリティの向上等に期待できそうですが, 私の場合, これから勉強です。

手法・技法の革新による「設計」を取り巻く問題解決のアプローチに平行して, 設計技術者の社会的地位の向上に努

力することも間接的ではありますが設計技術を高度化する大事なアイデアじゃないでしょうか。

最後になりますが, 5〜6 年前, SDA と呼ぶソフトウェアの設計支援環境の統合化技術に取り組むプロジェクトに参加していた時期がありました。

当時は「まず勉強」のスタンスで参加していたこともあり, プロジェクトにはあまり貢献しませんでしたが, 同プロジェクトで盛んに検討されていた設計作業プロセスの支援技術, 設計者の思考過程を支援する環境が実現すれば, システム設計・ソフトウェア設計の領域における諸事情も飛躍的に様変わりするものと期待しています。

P05 **後藤 浩** (ごとう ひろし)
性別：男　年齢：27　血液型：O
会社：(株) シスプラン
所属：営業技術部

I. プロフィール

(1) 現在までの主な仕事の内容について

- MS-DOS パソコン上のターミナルエミュレータ作成
- MS-DOS, UNIX 上の各種フィルタ作成
- コンピュータシステムの営業活動

(2) 現在利用している開発環境について

現在は, 基本的に開発作業は行っていない。単に使用している計算機環境ということであれば,

- NEC PC-9801
- SONY NEWS
- Macintosh IIfx 他

(3) これまでシステム設計に採り入れてきた技法など

- 状態遷移図
- 顧客を納得させることのみが目的のプロトタイピング

II. ポジションペーパ

PC メンバとして参加するにあたって

1 不安/自信/期待

(2) でも記したが, 現在私は, ソフトウェアの開発に携わっていない。故にソフトウェアの設計もしていない。ハードウェアを含めたコンピュータシステムの設計とその営業活動が主な仕事である。

縁あって SEA の環境分科会に参加するようになり, オブジェクト指向設計の勉強というかたちで活動を続けている。

そんなわけで, 「現場でのシステム設計を改革するテクノロジを求めて」という今回のテーマにはふさわしくない PC という意味で不安が無いわけではない。

しかし, PC の役割はメンバ諸氏のグループ活動 (今回の場合は設計作業) を円滑に進めるための世話役であると理解しているので, この点には自信がある。

また, 過去にメンバとして参加した経験からグループ活動の面白さを身に染みて分かっているつもりなので, 私も含めて参加者全員が大きな成果を得ることを期待している。

2 盛岡/小淵沢/岡崎

「夏のプログラミング ワークショップ」, いわゆる「若手の会」は今回で 10 回目を数えるという。私は, 過去にメンバとして 2 回参加している。通常開催は盛岡で行われている。盛岡は, ご存知の通り岩手の県庁所在地で街も大きい。事情は知らないが, 1 回だけ小淵沢で開催されたことがあった。で, 今回は岡崎で開催される。

何が言いたい訳でもないが, 開催地だけを考えれば, 皆勤賞かとふと思った。

3 ワークショップ/セミナー/シンポジウム

「ソフトウェア技術者協会」主催のイベントに限らず, イベントには様々な形態があり, 今回はワークショップという形態のイベントである。セミナー, シンポジウムなどとの形態の違いを明確にし, 私を含めた参加者の意識を統一したい。

広辞苑によれば, といきたいところであるが手元に大辞林しかない。手元の大辞林によれば,

ワークショップ

1. 仕事場。作業場。
2. 研究集会。講習会。

セミナー → ゼミナール

1. 大学で, 少人数の学生が集まり, 教師の指導の下に自ら研究し, 発表・討論を行う形式の授業。ゼミ。セミナー。
2. 少人数を対象とし, 討議などまじえた講習会。セミナー。

シンポジウム

([共に飲む] 意のギリシア語から) 一つの問題について, 数人の人が意見を発表し, それについての聴衆の質問に答える形で行われる討論会。公開討論会。

とある。

手元の大辞林と私の認識は少し違うが, ワークショップは, 参加者が, 作業をしながら行う研究集会のことのようである。

ましてや, 今回の場合は「若手の会」というサブタイトルが示しているよう, 30 歳以下の若者が集う会であるから, 参加者主動のイベントであることの確認をお願いしたい。

4 C/R/C

私を含めた数人の PC で, 設計の方法論として, CRC カードを使用したオブジェクト指向のアプロ ーチを紹介する。

CRC は, それぞれ, Class, Resposibility, Collaborator のことである。 この先は, 当日に譲る。

5 ソフトウェア/設計/方法論

現在はソフトウェアの設計をしていないとはいっても, 営業に移る前の数年間は, 数人のグループもしくは一人でソフトウェアの開発を行っていたので当時の経験をもとに, 設計方法について感じていることを記す。

確立された手法を実践したという意識は無いが, 考えてみれば「構造化手法」を模索していたように思う。実現しようとしているシステムの複雑さに着目し, モジュール分けを行い, モジュール毎に, 機能, 論理, 用法を明確にした。一人の場合は, この段階で製造に取りかかった。

数人で開発作業を分担する場合は, お互いの入出力を決め, あとは各々が設計/製造を行っていた。

製造過程は, モジュールの機能, 論理, 用法を実現すべくひたすら, コーディング (エディタでの入力) を行った。

今思えば, 設計で苦労したことはほとんどなく, 苦労するのはデバック時のみであった。ソフトウェア開発の大半をデバックに費やしていたということになる。

恥ずかしい限りであるが, この状況で開発されたソウフトウェアが一時期ではあれ, 稼働, 流通したのだから世の中不思議なものである。

誰もが感じることとは思うが, 反省点をあげれば,

- 実は, プログラムの構造が設計されているのはコーディング時である。
- 設計時の技術的な問題を議論していない。

ことに気づく。

今現在, 私は営業職で, 「利益追求の為には設計方法論など問題にしない」という立場にいる。しかし, 「利益追求」＝「生産性の向上」であり, いろいろな意味での「生産性の向上」の為に, 「オブジェクト指向」が必要であると思っている。

現在, SIGENV の活動を主体に「オブジェクト指向設計/プログラミング」を勉強中である。

6 リーダ/マネージャ/スーパーバイザ

蛇足になるが, 最近ぼんやり考えることがよくある。プロジェクトを推進していく上で重要な存在となるリーダ, マネージャと呼ばれる人達の役割に関してである。

有意義なグループ活動の合間に, この辺の話題で脱線した討論が出来ることを少しだけ期待している。

P06 **桜井 修** (さくらい おさむ)
性別：男　年齢：31　血液型：O
会社：池上通信機 (株)
所属：ソフトウェア推進室

I. プロフィール

(1) 現在までの主な仕事の内容について

- 機器制御マイクロプロセッサのソフトウェアの作成
  パラボラアンテナ制御ソフトウェア
  放送運行システム
  映像送出卓スイッチャ制御ソフトウェア
  など放送関連機器の制御ソフトウェアの作成

(2) 現在利用している開発環境について

- NECPC98 シリーズパソコン上でのクロス開発環境
  8 ビット (Z80)/16 ビット (80X86)/32 ビット (680X0) プロセッサなど
- 各プロセッサー対応のインサーキットエミュレータ/ROM ライターなど。

(3) これまでシステム設計に採り入れてきた技法など

構造化手法を取り入れたことはあるが, データフローダイアグラムやモジュール構造図, モジュール仕様書ぐらいでプログラミング工程に入ってしまっている。データディクショナリーや状態遷移図など書かないときが多い。

II. ポジションペーパ

(4) PC としてのポジションペーパ

1 設計でのありがたみ

僕らが目標とするものは, 短絡的にプログラムコードであり, その動作している環境である。大抵のものが目的どおり動作してくれれば, メシが食えるわけだし, 依頼してくれた人を喜ばすことができる。しかし, 問題はそこまで行く過程とその後のための布石である。その方法を教えてくれるのが「設計」なるものと期待するわけである。つまり「設計」において, その対象となる問題領域をプロセッサの動作環境へ投影できるようにモデル化できればよいのである。しか

し, いったいどうすれば良いのであろうか。どのように分析すれば, 「モデル化」したことになるのだろうか。そう, それらのことを教えてくれる意味で, 設計手法を学ぶ必要があると感じ, ありがたみが得られるはずである。

現場では, しかし, いままでの経験からどうもうまく行っていない感がある。プログラム設計において, 大抵はデータとモジュール構成とモジュール間のインタフェース仕様, それにそのモジュールのフローチャートで「設計」を終えてしまっている私のいままでの作業そのものにも問題があるが, 設計ドキュメントを作成していく過程で, すべて対象領域の事象やふるまいを分析することはなかなか難しいと実感している。できないしても, それを見つけ出す手法を得たいのである。実際, 最終段階において「設計」への後戻りが多く発生してしまっている。それは設計作業において対象を見る側面の不十分さからくるもので。ちょっとした注意さえあればいい問題がほとんどである。たとえば,

- ユーザーの常識としてある機能が盛り込まれていなかった。(仕様分析の不充分さ)
- 機能を実現されるための操作が不充分であった。(操作フローの設計の不十分さ)
- 時間軸に対して解析されなく, 応答速度に問題が発生する。(制御タイミングの設計)
- 制御対象となるもののふるまいが予想できないものであった。(ハードウェアインタフェース)

など現われてくる。それらを一つ一つ解析して, ある部分はふるい落とし, 意味のとり違えをただす。

この間, 対象領域をいろいろな側面から見てるため, 関連のない細かな設計ドキュメントは発生するのであるが全体としてのドキュメントは, ますます分かりづらいものとなってしまうのである。そして, 最終チェックが完了した時点でのプログラムコードで作業が終了してしまっている。その後の保守で, 上記の問題の繰り返しが発生するのである。

2 設計手法/環境に望むこと

機械的に対象領域をモデル化することは可能であろうか。いろいろ提案されていると思われるが, 仕様の記述文から名詞。動詞からモジュールを抽出するなど, どうも期待出来ない感がある。個人の能力や経験も設計技法の中に入れ込むことを望むのも無理な話しなのだろうか。せめて, プロセスの側面/制御タイミングの側面/データの側面/エラー処理の側面/操作仕様の側面, を包括的に表現できる図でもテキストでも, 「これ1つあればあとは枝と葉」なるものがほしいと期待する。さらに, その先の工程「プログラミング」をも「その実」として出来あがるソフトウェア環境を作れればなお良い。もう, 追加仕様や保守作業においてプログラムコードからスタートするのではなく設計ドキュメントから検討/修正していきたいものである。最近 CASE ツールなるものに期待するのもこの点である。

3 オブジェクト指向技法への期待

そんな日々の設計作業のなかでオブジェクト指向技法へ目が向くのである。私自身, 以下の項目など設計からプログラミングまで透過的に見通せるものとして, 大いに期待するべきものと感じている。

- モジュール性：あるまとまったものとして表せられる性質。
- 情報の隠蔽：データを操作するものの側に公開すべきもの。必要のないものとを明確に意識させる。
- 抽象化：対象のもつ細かな構造やふるまいにとらわれず問題を解く上で必要最小限の性質で表す。

従来もいろいろ論じられてきたであろうが, 「設計」で思考するときに上記の項目は, その指針を与えてくれるので助かる。素朴にすぐに取り入れられる考え方であり, プロセスの側面とデータの側面だけでも包括的にモデル化できるように思われる。さらに制御タイミングやエラー処理についても表現できるものがあれば大いに助かるものとなる。

最後に, 今回のワークショップにおいて, どれだけのものを参加者と討論できるか分からないが, 多くの人たちと現場での問題点, 実際の設計作業を通して気づく点などの発掘に期待して, 参加することにします。

P07 **田中 慎一郎** (たなか しんいちろう)
性別：男　年齢：33　血液型：A
会社：(株)SRA
所属：ソフトウェア工学研究所

I. プロフィール

(1) 現在までの主な仕事の内容について

- 国土数値情報メインテナンス・パイロットシステムの開発
- ソフトウェア保守技術開発計画の管理, 関連調査など

(2) 現在利用している開発環境について

- Sony NEWS(NEWS-OS 4.1c)
- Sparc LT(SunOS 4.1.1+Emeraude PCTE) 他

(3) これまでシステム設計に採り入れてきた技法など

- 構造化設計の雰囲気を採り入れた先輩の設計法 (記述法) の真似.
- 設計技法ではないが開発時に意外に役だったのが, ド

キュメント/コーディング基準.

- 技法とは言えない試行錯誤レベルのプロトタイピング・アプローチ.

II. ポジションペーパ

PC メンバとして参加するにあたって

1 はじめに

文学部を卒業した. 実験でデータ処理をする必要があった. プログラミングの真似ごとみたいなものはやってきた. たまたま一年上の先輩がいた会社があった. 入ってしまった.

言ってみれば, これが私がこの業界に足を踏み入れることになったきっかけだ. こう書いてしまうとあまりにも安直ではあるが, . .

基礎もない, 勉強が好きなわけでもない, 一人で仕事をすることが多い. その結果, 他人のやり方を見る機会がなく, 仕事の性格を考えることもなく, かなりの時が流れてしまった. これは実にもったいないことだ. 結局, ソフト開発を業としていくために, 長期的に見て自分が何をしていかなくてはならないかがよく分かっていないのだ. PC メンバとして何とも情けない話とはいえ, これが私の実態だ.

しかし多少の違いこそあれ, このようなポジションに自分をおいたままの若手は結構いるはずだ. そんな風に思えてならない. 与えられた狭い世界だけがそのすべてのように感じてしまっているこのような若手に, いろいろな刺激を得られる場を提供すること, これこそが「若手の会」の本来の目的であるように私は思う. 彼らが私のようにならないためにも, . .

なお, だからと言って, すでに自分で活動できる参加者にはこの手のワークショップが面白味に欠けるものになるとは思われない. そういう人は PC メンバと同様に, 自分が楽しむと同時にほんの少しだけ, 他の参加者の自立に手を貸していただきたい. それもまた将来的に意味のある行動だと思うので, . . どうかよろしく.

以下では, 今回のワークショップに参加するに当たって考えていることを簡単に書き並べる. ごく当たり前のことで, たいしたことは何も並べていないが, そのたいしたことではないことが重要なのだということを明確にしたいのだ.

2 他人のためにプログラムを書くということ

現場でアプリケーションの開発をしているということは, すなわち「他の誰かのために開発をしている」という側面が強いはずだ. が, 一体どのくらいの人間がそれを明確に意識して活動しているのだろうか. 皆はどうだろうか.

また, チームを組んで開発を当たるということは, チーム内のコミュニケーションがきちんととれていなければ問題が発生しやすくなるはずだ. これも, 特にマネジメントすべき人間が, どのくらい明確に意識して指導しているだろうか. 皆はどうだろうか.

いろいろと考えなければならないことがあり, 対処しなければならないことがある. だがある一面から見れば, 「他人のために, 他人と一緒に開発を行なっている」, そういうことを常に意識することが基本として挙げられるだろう.

他人のために開発を行なうということは, 他人の要求を実現するということ, それもできれば当人の想定しているものよりもより効率的に実現すべきことだ. 他人と一緒に開発するということは, 他人に自分の書いた設計仕様を読ませる, また他人のソース・コードを読む, そういうことでもあるはずだ.

こういうことが何を意味するかを考えれば, 本来すべきことは明らかだとも言える. 形式だけ整えても何ら意味はないのだ.

3 設計技法を採り入れるということ

ここで問題となるのは, 設計技法は単なる技法であってそれ以上のものではないということだ.

設計技法にしても, はやりの CASE ツールにしても, 採用することによって, 即, 何らかの問題が解決するわけではない. こんなことは本来はごく当然のことなのに, どうしても期待の方が高くなってしまうきらいがある. 現実にはむしろ, 採り入れるにあたっての初期投資とでも言おうか, 新たな試みに対する負荷というものが厳然と存在する. したがって, そのための混乱/手間の増大などが発生し, しかも「形」から入るために本来の御利益が授かれるようなものになっておらず, 大変なだけということが多い. そのためか, 現場の作業者の反応は「設計技法は役に立たない. そもそも設計自体が無用とすら思える」などとなってしまう.

しかし, 本当にそうなのだろうか? それでいいのだろうか?

4 設計に関する恨めしさについて

設計に関する恨めしさというものは, 誰しもそれなりに経験しているのではないだろうか. 始めは「他人が行なった設計」に対して, その後は「自分が設計を行なうということ」に対して, . .

設計が恨めしく思える時, それは恐らくは単に「設計がなされていない」あるいは「設計以前に要求定義がなされて

いない」ということに過ぎないと思う．そしてそれは恐らく，「ユーザは自分とは違う」，「他の開発者は自分とは違う」というごくあたり前のことが見落とされている結果だと思う．

他人の書いた設計を恨めしく思う時，それは通常，「なんていい加減なんだ」というような反応として現れるだろう．だが逆に自分が設計をおこなう時には，真面目にやればやるほど，「一体どう書いたら他人に理解可能な設計になるのだろう．どうやったらユーザの確認を確実にとれるのだろう」というような反応として現れるのではないだろうか．

これは恐らくは単に「設計はとても難しい作業である」という事実を表しているだけと考えられる．が，一方，その認識はあまりに低く，設計工程以前への工数のかけかたは一般に少ない．何かあると簡単に削られてしまっているようにさえ見受けられる．どうもプログラミング以降で対処する方が現実的だととらえられているようだ．しかし本当は，単にやり方がこなれていないだけと思えてならない．

これは何も，ウォーターフォール型のガチガチの設計手順を推奨しているのでも何でもない．むしろ，「何がまだ確定していないのか」を明らかにしないで作業を進めるのは馬鹿げているというだけのことだ．まだ確定していないということがはっきりしてさえいれば，また別の対処のしようもあるはずだからだ．

しかし，では，一体それをどうやって?

私はここに「設計技法」というものが入り込む余地があると考える．同じ言葉で話し，同じ思考のガイドラインを持ち，その上で未確定部分を明らかにする．設計技法とは言ってみればそのためのものだとして捉えてみたいのだ．

私は今回，そういう点に焦点を当てて討論したいと希望する．

P08 **福良 博史** (ふくら ひろふみ)
性別：男　年齢：45　血液型：AB
会社：(株) ジェーエムエーシステムズ
所属：神谷町事業所

I. プロフィール

(1) 現在までの主な仕事の内容について

- 開発支援ツールの開発 (設計支援用言語，製造用言語，ドライバ/スタブ)
- ミニコン間の通信プロトコルの開発
- 事務系アプリケーション・システムの開発/保守

(2) 現在利用している開発環境について

- ハードウェア関連
  - ホストコンピュータ (IBM，富士通 M，日本電気 ACOS，他)
  - ミニコンピュータ (日本電気)
  - パーソナルコンピュータ (日本電気 98xx)
- DB/DC(CICS，IMS，AIM，他)
- 言語 (ASM，COBOL，PL/I，他)
- 開発支援ツール関連
  - MTS
  - PANVALET
  - ROSCOW
  - DATAMANAGER
  - DATA-EXPERT　etc.

(3) これまでシステム設計に採り入れてきた技法など

- フローチャート
- ディシジョン・テーブル
- 疑似コード
- HIPO
- HCP
- NS チャート
- 状態遷移図
- データ・フロー　etc.

II. ポジションペーパ

現在設計について感じていること

1 設計したものを後任者に引き継ぐ場合にしてきたこと

従来，色々な設計方法があり，自分で本や雑誌を読みあさり，どのような設計方法が良いのかを，その都度実践してみた。しかし，現場で何時も問題になるのは，以下のようなことである。

1. 後任者に引き継ぐ時に，理解出来る仕様書か。
2. 頭の中にしか残っていない事で，引き継ぐ必要のある事は何か。
3. 後任者への，設計上の制約条件等の一覧。

つまり，どの様な設計方法を行っていても，いつも頭を悩ますことに変わりが無い。そして，以下のようなドキュメントがあれば，理解出来るであろうと考えている。

- リソースの一覧 (プログラム，コピー句，画面定義… etc)
- 機能仕様書 (含む，モジュール構造図)
- 詳細仕様書
- ファイル定義

- 画面定義
- 帳票定義　　etc.

そしてこのようなドキュメントだけではどうしても理解しづらいと思われる場合には, データ・フローや状態遷移図などを付加する。

他人に誤解を受けない用にするためには, 設計時の前提, ユーザの方針, 開発環境などについて, 文書化されていない事柄で気が付いた点についてメモを残す。

最後の手段としては, 電話なり, FAX なりで当方に質問してもらう。

2 自分自身が行ってきた設計についての見解

機能による構造化を行ってきているが, 自分自身ではそれなりにうまく設計していると自負していた。これは, 今考えると, 自分が設計したものは, その考え方を理解しているために, 批判的にものを見ることが出来なかったためかも知れない。

数カ月前に以下のような話を聞いた。

> 「10 年程前に造ったシステムが, ドキュメントもしっかり残っていて, しかもバグも殆どでない。1 年に 1 回くらいの割合で制度変更にともなう修正が生じる。数年に一度, 担当者が変わるが, バグも出ないために仕組みが理解出来ていない。このため, システムがよく理解できていないので何処を修正したら良いのかを調査するのに時間がかかる。」

この話は, バグが少ないという点については自慢になるかもしれないが, システムの保守担当者に対する理解の促進という観点では問題がないか? 通常, ソフトウェアの設計というものが, 高度な知的生産物であるから, 元来理解は困難である。と簡単に言い切ってしまい過ぎていることはないだろうか。

設計自身はそれなりに知的な生産活動であると考える。しかし, その出来上がったものを理解させることは別の次元の話のように思える。

3 まったく知らないシステムの保守を行ってみての感想

以上に述べたことは, 設計したものを引き継ぐ立場での経験を現在ながめてみた上での感想である。この反対に全く内容を知らないシステムを私が引き継いだときの感想を述べる。

はじめにした事は, アプリケーション自身の理解をする事。その後, 設計書等のドキュメントを通読した。しかし部分部分を記述した詳細な資料をいくら読んでみても理解できなかった。当然上記 a. に示したたぐいのドキュメントが揃っていたのである。しかしこのようなモジュール構造図などは自分の頭にイメージを浮き上がらせる役にはたたなかった。そこでなにをつくって理解の促進をはかろうとしたか。一言で言うなら, それは

> 「一つのトランザクションがシステムの中をどの様に動くのか, またその時に DB はどのような影響を受けるのか」

を理解する事であった。

つまりデータの動きとプログラムの係わりあいを横断的に描いたデータ・フローを作成し, はじめてシステムのイメージが自分の頭に出来上がり, どのように保守していけば良いのかが掴め, 自信がついてきた。

4 設計書とは何か

設計書には, まずそのシステムを造る担当者への仕事の割り振りに必要である。これは当然のことであるが, この観点だけで考えると, 担当者は全体を理解出来ている必要はない。このために機能分担された一部分に専念していれば良いという事になる。

しかし, こうして出来上がった設計書は, 保守担当者には判りづらいものになってしまっている。保守担当者に理解しやすいドキュメントとは, 一枚の伝票がシステムの中をどの順序でどのプログラムを通るか, どの DB がどの時点で更新されるのか, などがハッキリと把握出来るものである。

従来, 設計書はここまでの記述を要求されていないのが一般的なようである。

5 設計書のあるべき姿

設計書の理想像は, システム開発の担当者に理解してもらうと同時に, 保守にも役立つドキュメントとなっていることが望ましい。

しかし, 従来の機能分割による設計はデータの流れを分断してしまう傾向にある。またデータ・フローによる設計を行っても最終的なインプリメント用言語が, データとモジュールとを分断してしまう傾向にある。(COBOL など) 良い解決方法があるのだろうか?

現在, 個人的には, 以下のような方向を試行していきたい, と考えている。

ア　抽象データ型による, データと手続きの一体化をはかる。

イ　うまくいけば, 部品化/共通化をはかる。

現在このような方法論としてオブジェクト指向による分析/設計/プログラミングの方法論がいろいろ出てきている。こ

のオブジェクト指向には, 今までに無かった有益な考え方がある, と感じており現在勉強中である。

6 保守の立場から見た設計ドキュメント

ア 人によりドキュメントの記述レベルの格差 (詳細か, 粗いかなど)

- プログラム・ソースがドキュメントと考える Eiffel の作者 B. Meyer 氏の考え方もこの方向
- CASE ツールの発達により変化が期待出来るのかも知れない

イ 時の経過と共に, ドキュメントとプログラム・ソース等との間の乖離が拡大

- 保守を行う過程で人が入れ替わるために発生。ア. と同じ状況になる

ウ 制度変更等に伴う機能の変更により, プログラム構造に揺らぎが発生

- トップダウンによる従来の設計方法を改める必要があるのかも知れない

エ 設計時の前提条件, 複数の可能性から或る一つの方向を採用した経緯の忘却

- 設計時のプロセスの記録が本来の設計ドキュメントではないか?
  この詳細を残す事が可能か?

7 設計/言語等を含めた開発環境全体の評価用の具体的なモデルが必要と考える

例えば, 郵便番号が 5 桁から 7 桁に変わった場合にどのような影響が生じ, どのような手順で修復するのか。 etc。

P09 **豊田 英利** (とよだ ひでとし)
性別：男　年齢：39　血液型：B
会社：(株) セス
所属：応用システム部

I. プロフィール

(1) 現在までの主な仕事の内容について

主な開発経験

1. **プラント制御**
   (a) セメントプラント PDP11/24 Fortran '83.4～
   (b) Management System VAX/VMS Fortran
2. **編集支援システム**
   (a) テキスト入力 Apollo W.S AEGIS/C '86.9～
   (b) 写植変換
   (c) 文字管理
   (d) ワープロ原稿入力 PC-98/MS-C
3. **ネットワークシステム**
   (a) ニュース速報伝送システム SUN W.S UNIX/C (SNA,TCP/IP) '89.2～
   (b) 3270 エミュレータ DG C (SNA)
4. **専門学校 講師**
   (a) C プログラミング '90.4～
   (b) 計測制御 理論・実習

私は, 現在の会社に入ってからリアルタイムシステムの開発を中心にしてきました。プラント制御では, ミニ Computer(PDP11/24) を利用してシーケンス制御及びフィードバック制御をメインにシーケンサー, プロセス I/O, X-RAY(PC) 等から運転状態を取り込み最適制御をした。このシステムは, 4 人のプロジェクトで構成されていた。この開発では, システム設計をする上でのプロトタイピングの重要性をひじょうに感じた。また, プロトタイピングの目的, 利用についてよい勉強をした。

編集支援システムでは, W.S.(Apollo) を Talken passing Ring で LAN 接続し, サーバー/クライアント型のアプリケーションを開発した。この開発では, 30 人のプロジェクトで構成されており, 各サブシステム間のインターフェイスの調整がひじょうに困難であった。私は, このシステムで始めて W.S./ C 言語を利用してシステムを開発した。プログラム設計では, HCP チャートを利用した。

私は, HCP を利用してひじょうに気に入っている。HCP は, レビューの能率が向上する, データと処理との関係を明示出来るため, 設計者自身の思考の整理ができる, フリーハンドのため保守が容易等利点が多い。

ネットワークシステムでは, ターゲトマシーンが DG(W.S.) で英語版 3270 エミュレータを日本語版エミュレータにコンバートし, グラフィック機能も追加した。この開発は, 3 人のプロジェクトで構成された。私は, この開発で英語システムを日本語システムに変換するのがやっかいなものだと実感した。また, ドキュメントもソースリストしかなくエミュレータも始めての経験であり, ひじょうに苦労した。アメリカのソフトハウスのプログラムリストを見て, プログラムの質がよかったことと, プログラムがドキュメントになっていたこと。しかし, やはりシステムのドキュメントは必要であり重要である。

(2) 現在利用している開発環境について

- フローチャート
- HIPO
- ディシジョン・テーブル
- タイミングチャート

- プロトタイピング (システムで採用するための方式, 調査)
- HCP
- データ・フロー・ダイアグラム
- 構造化設計
- リアルタイムシステム構造設計 (マルチタスク制御)

II. ポジションペーパ

PC メンバとして参加するにあたって

1 はじめに

私は, 情報処理業界にはいって, はじめてコンピュータのシステム開発を習得し, 汎用機を利用してビジネス系のシステムを作成していたが, どうしてもハードに密着した制御系のシステムの開発をしたくて会社も変わった。それ以降 COBOL とは縁がなくなり, ミニコン, ワークステーション, パソコンを利用して Fortran, C で, 制御系又は通信系のシステムを開発してきた。現在ワークステーション, パソコンを利用してビジネス系のアプリケーションも増えてきているが, 私の設計のなかではリアルタイムのイメージで考えがちである。しかし, アプリケーションも LAN を導入しているケースも多く, いいのではないかと思っている。

2 システム設計について感じること

設計論を考えてみた場合, ビジネス系 (主にバッチ) と制御系 (リアルタイム) の設計内容は異なるように思うが, システム設計の意義は当然同じ筈である。仮に, システム設計 (設計書として残せるもの) をしなくてもシステムは作成でき, 動作するだろが, 質が問題になるだろう。頭の整理ができにくいためテストが不十分であったり, 仕様のミスが多々でてくるだろう。また, システム設計をしても人間がするので個人差は当然あり頭ではわかっていても実行していない場合も多々ある。私自身, 設計作業に於いて今まで作成されたシステムを参考にして, 一般的な設計技法も取り入れ, 仕様によりまとめやすい様に, 色々な手法をとりいれて自分なりに工夫してきた。設計をしていると理想と現実の妥協のギャップをよく考えさせられる。これは, 開発期間, 開発コスト, ハードウェア, OS 等さまざまな要因がある。

そこで, 当然理想に近い設計を目指すわけであるが人間がする作業になるわけである。ここで, 人の意識が非常に重要となる。

- 何のために設計するのか
- 設計書をどう活用するか
- 設計上の規約
- 設計書の信ぴよう性　　etc.

活用

- Review の資料 (内部, 外部, サブシステム間)
- 本人の頭の整理
- 仕様変更時の見直し
- バグ対応の見直し
- システム拡張時
- システム流用時又はモデルケース　　etc.

ほとんどの会社は, 標準化するためにシステム設計規約が定められている。

システム設計論を考えて見た場合, 標準化, 生産性が重要であり, 色々な手法があるが, いずれも長短があるのが現状ではないだろうか。

P10 **山崎 利治** (やまさき としはる)
性別：男　年齢：59　血液型：B
会社：日本ユニシス (株)
所属：研究開発部

I. プロフィール

(1) 現在までの主な仕事の内容について

種々雑多なプログラミング

(2) 現在利用している開発環境について

現在はプログラミングを行なっていません.
したがって利用している開発環境もありません.

(3) これまでシステム設計に採り入れてきた技法など

特にありませんでした. ソフトウェア工学の出現以前にプログラミングをやめていました.

II. ポジションペーパ

(4) PC としてのポジションペーパ

老兵ですがオブザーバとして参加したく, 申し込みます.

オブジェクト指向設計：責任駆動アプローチ

Rebecca Wirfs-Brock
Tektronix. Inc

Brian Wilkerson
Instantiations. Inc

概要

オブジェクト指向プログラミング言語はカプセル化を支援しており，これにより，再利用，リファイン (refine：洗練)，テスト，保守そして拡張といったソフトウェアの能力を改善している。この支援の十分な恩恵は，設計段階においてカプセル化が最大限になされた場合にのみ与えられる。

本稿では，データ駆動 (data-driven) のアプローチが，オブジェクトのインプリメンテーション (implementation：実装) を早い段階で行なおうとして，カプセル化を最大限に行なうことに失敗していることを論ずる。そしてそれに代わるオブジェクト指向設計として，責任駆動 (responsibility-driven) のアプローチをとる設計方法論を提唱する。また，後の段階までインプリメンテーションの問題を持ち越すことで，そのアプローチが如何にカプセル化を向上させるのかを示す。

1 はじめに

オブジェクト指向プログラミングの主な恩恵とは，それがいくつかのソフトウェア尺度 (software metrics) のとる値を向上させる能力にある。これら尺度とは，コードの再利用，リファイン，テスト，保守そして拡張の容易性などである。しかしアプリケーションの大きさ，すなわち複雑度が増え続けているのに応じて，これらの尺度の値は減り続けている。

オブジェクト指向プログラミングは，複雑性をきちんと管理することでこれらの尺度の値を向上させている。複雑性を扱う最も効果的な手段は抽象化である。抽象化の形態はたくさんあるが，オブジェクト指向プログラミングで複雑性を管理する主な抽象化の形態はカプセル化である。

しかしながら，オブジェクト指向言語でプログラミングを行なえば，アプリケーションの複雑性が旨くカプセル化されると保証されるわけではない。良いプログラミング技法を適用すればカプセル化は向上する。しかしオブジェクト指向プログラミングの十分の恩恵は，カプセル化が設計段階の目標であることをはっきりと認識した場合にのみ与えられる。

どのようなアプローチを設計者が採るかにより，設計段階でカプセル化が具体化される度合は深刻な影響を受ける。以降では，設計におけるデータ駆動のアプローチと何故それが最善のカプセル化に失敗したのかを述べる。そしてそれに代るものとして責任駆動のアプローチと呼ばれるものを述べ，併せてそれが何故設計においてより高度のカプセル化をもたらすのかを説明する。

2 データ駆動設計

データ駆動設計 (data-driven design) は，抽象データ型設計手法をオブジェクト指向プログラミングに適用した結果である。その適用方法はじつに単純なものである。これは，クラスというものが抽象データ型に極めて類似しているためである。

純粋に実用的な観点からすると，オブジェクトは振舞い (オブジェクトの責任のインプリメンテーション) と構造 (そのオブジェクトから直接分かる他のオブジェクト) をカプセル化する。これは，抽象データ型の定義と良く似ている。

データ駆動設計を述べる前に，抽象データ型設計を簡単におさらいしてみよう。

2.1 抽象データ型設計

抽象データ型は，データとそのデータを操作するアルゴリズムのカプセル化である。抽象データ型は以下の質問をすることにより設計される。

- どういうデータをこの型は包含するのか?
- どういうアルゴリズムをこのデータには適合させれば良いのか?

これらの質問の主たる着眼点は，どういうデータがシステムで表現されているかを決定することにある。これは，プロ

この論文のすべてもしくは一部のコピーは，そのコピーが直接の商用利用のために作成もしくは配布されるものではなく，ACM の著作権告示と出版物の名称および発行日付が表示され，コピーは Association for Computing Machinery の許可によるという提示がなされるという条件においては無料で許可される。そうでないコピー，もしくは再配布は，費用および/もしくは特別な許可を必要とする。

原題:Object-Oriented Design:A Responsibility-Driven Approach, ACM OOPSLA'89 Proceedings (October 1-6, 1989) pp.71-75
©1989 ACM (Association for Computing Machinery)
ACM の左記許可に基づいて翻訳掲載する。
翻訳はソフトウェア技術者協会・環境分科会有志のものによる。ACM の許可に反しない範囲での再配布は自由とする。

グラム (時としてその一部だけ) から要求されるデータを明らかにすることによりまず行われる。次にこのデータは結合力を基準として型にグループ分けされる。(結合力とは一群のデータに適応されるもので, グループのその部分がどのくらい強く関連しているかを示す尺度である)。最後に, これらのデータの型に関係するアルゴリズムを明らかにすることにより, 要求される他の型がしばしば発見される。

2.2 データ駆動設計の定義

データ駆動設計では, オブジェクトは以下の質問することにより設計される。

- 何 (どういう構造) をこのオブジェクトは表すのか?
- どういう操作をこのオブジェクトは行なえば良いのか?

ここでもまた, システムで表現されるデータ構造に基本的な焦点が当てられている。

2.3 データ駆動設計の例

以下の例でデータ駆動アプローチの結果を説明する。この例はラスターイメージ・オブジェクトのクラスである。**RasterImage** は, 視覚イメージを 2 次元の bit 配列に格納するオブジェクトである。**RasterImage** オブジェクトを規定するためには, **RasterImage** によって表現される構造と **RasterImage** が実行できる操作について記述する必要がある。

RasterImage の構造は以下を包含する。

- その bit 配列 (バイトの配列として表現されている)
- bit 配列の大きさ (巾と高さの両方)

RasterImage の操作は, 回転や拡大縮小 (scaling) といった基本的なイメージ操作を包含する。これは bit 配列の中の個々の bit へのアクセスと変更, そしてイメージ操作を用いた **RasterImage** の結合によって行なわれる。ここではイメージ操作をこれ以上規定しない。そういったものは, より特化されたイメージング・オブジェクト (例えば Smalltalk [3] の **BitBlt**) もしくは他の多くの標準のビット・イメージ操作によって行えるはずである。

以上の **RasterImage** に関する構造および操作のあらましから, 以下に示す形式構造を持つ **RasterImage** を定義することができる。

```
class RasterImage is
  bitArray:ByteArray;
  width:Integer;
  height:Integer;
```

それぞれ必要な操作のためにプロトコルを定義する。プロトコルのリストの 1 部は次のようになる。

```
  rotateBy(degree:Integer):void;
  scaleBy(x:Integer, y:Integer):void;
  bitAt(index:Point):Integer;
  bitAtPut(index:Point, bitValue:Integer):void;
```

同様に **RasterImage** 構造の各部分にアクセスし, 変更するプロトコルも定義する。

```
  bitArray():ByteArray;
  bitArray(bits:ByteArray):void;
  width():Integer;
  width(aNumber:Integer):void;
  height():Integer;
  height(aNumber:Integer):void;
end RasterImage;
```

ShapedImage と呼ばれる **RasterImage** の特殊形は, bit 配列と同様に clipping mask を包含する。clipping mask は bit 配列中の個々の bit が不透明 (その場合の clipping mask は 1) であるか透明 (その場合の clipping mask は 0) であるかを規定する。**ShapedImage** は矩形でないビットパターンの表示を可能とする。**ShapedImage** は **RasterImage** のサブクラスである。ここで, clipping mask を表現するために追加構造を **ShapedeImage** に定義する。

```
class ShapedImage inherits RasterImage is
  clipMask : ByteArray;
```

そして clipping mask にアクセスし, 変更するプロトコルを定義する。

```
  clipMask() : ByteArray;
  clipMask(bits : ByteArray) : void;
end ShapedImage;
```

2.4 データ駆動設計の長所

データ駆動アプローチの主な利点は, 伝統的な手続き型の言語を経験したプログラマにとって親しみやすいプロセスだということである。このようなプログラマにとっては以前の経験をオブジェクト指向システムの設計に適応することは比較的容易である。

2.5 データ駆動設計の短所

データ駆動設計の目的はデータとアルゴリズムのカプセル化ではあるが, オブジェクトの構造をそのオブジェクト定

義の一部とすることで, そのカプセル化は本質的に乱されてしまう。そしてそれは次にその構造を反映する操作の定義へとなっていく (これは構造を念頭において設計されるためである)。オブジェクトの構造を分かりやすく変更する試みは失敗する運命にある。それは他のクラスがその構造に依存することによる。これはカプセル化に反することである。

3 責任駆動設計

責任駆動設計 (responsibility-driven design) の目標は, カプセル化の向上にある。それは, クライアント/サーバ・モデルの観点で, プログラムを見ることによって行なわれる。

3.1 クライアント/サーバ・モデル

クライアント/サーバ・モデルは, クライアントとサーバという 2 つの実体 (entity) の間のやりとりの記述である。**クライアント** (*client*) は, サービスを実行するためにサーバへ要求 (request) を出す。**サーバ** (*server*) は要求に応じてサービスの集まり (set) を提供する。クライアントがサーバとやりとりできる方法は, **コントラクト** (*contract*: 契約) により記述される。コントラクトとはクライアントからサーバに出される要求のリストである。双方とも次のようにコントラクトを守らなければならない。すなわちクライアントはそれが規定する要求だけを出し, サーバはこれらの要求に応じる。

オブジェクト指向プログラミングでは, クライアントおよびサーバはいずれもクラスもしくはクラスのインスタンスである。どのオブジェクトも, 時に応じて, クライアントとしてもサーバとしても振る舞うことができる。

クライアント/サーバ・モデルの長所は, サーバがクライアントに対して**どのように** (*how*) するのかではなく, **何を** (*what*) するのかに注目している点である。サーバのインプリメンテーションはカプセル化される。つまり, クライアントから切り離されるのである。

クラスには, 以下の 3 つの異なるクライアントの型がある。

- 外部クライアント (external clients)
- サブクラス・クライアント (subclass clients)
- クラスそれ自身 (self client)

各々のクライアントの型について以下に述べる。

3.2 外部クライアント

クラスの外部クライアントとは, クラスのインスタンスやクラスそれ自身にメッセージを送るオブジェクトである。受取側 (receiver) は**サーバ**とみなされ, 送り側 (sender) は**クライアント**とみなされる。どのようにメッセージが返されるかということは, 送り側にとっては重要なことではない。オブジェクトが返すメッセージの集まりは引数の型と戻り値の型を含んでおり, これがクライアントとサーバとの間のコントラクトを定義する。

3.3 サブクラス・クライアント

あるクラスのサブクラス・クライアントとは, そのクラスから継承される全てのクラスである。スーパクラスが**サーバ**とみなされ, サブクラスは**クライアント**とみなされる。サブクラスは, それが継承する振舞いのインプリメンテーションについては気にかけるべきではない [9]。サブクラスによって継承されメッセージの集まりがクライアントとサーバとの間のコントラクトを定義する。

多くのオブジェクト指向言語では, サブクラスはスーパクラスで定義された振舞いだけでなく, 構造をも継承する。これは不幸なことである。というのは, このことが再利用されるコードを増やす目的でカプセル化の破壊をプログラマに奨励する傾向を持つためである。

3.4 自己 (self) クライアント

カプセル化を最大限に行なうためには, クラスもそれ自身のクライアントとしてみなされるべきである。クラス構造に直接依存するようなコードはできるだけ書くべきではない。言い換えれば, クラスの構造は最少の数のメソッドにカプセル化されるべきである。こうすれば, あるオブジェクトが自分の構造に依存する度合を下げ, その構造の変更を可能な限り分かりやすくすることができる。

3.5 責任駆動設計の定義

責任駆動設計は, クライアント/サーバ・モデルによってもたらされたものである。これは以下の質問をすることでコントラクトに着目する。

- どんな振舞いにこのオブジェクトは責任があるのか?
- どんな情報をこのオブジェクトは分担するのか?

重要なことは, オブジェクトに割り当てられる情報はそのオブジェクトの構造の一部である場合もあるし, そうでない場合もあるという点である。つまり, オブジェクトは情報を

計算 (compute) するかもしれないし, あるいは情報要求を他のオブジェクトに委託するかもしれないのである。

3.6 責任駆動設計の例

それではここで, 責任駆動設計の結果を先ほどと同じ RasterImage と ShapedImage のクラスを使用して説明する。RasterImage オブジェクトを規定するためには RasterImage が実行に関して責任を持つ行為と, クライアントと分担する情報について記述する必要がある。

RasterImage は以下のものに責任を持っている。

- そのイメージを完全に含む最少の矩形を知り, 保持すること
- そのイメージ内の個々のビットの値をすべて知り, 保持すること
- そのイメージを回転し, 拡大縮小を行なうこと

このような責任を以下のプロトコルに変換する。

```
class RasterImage is
  boundingRectangle():Rectangle;
  boundingRectangle(bounds:Rectangle):void;
  bitAt(index:Point):Integer;
  bitAtPut(index:Point, bitValue:Integer):void;
  scaleBy(x:Integer, y:Integer):void;
  rotateBy(degress:Integer);void;
end RasterImage;
```

ShapedImage は, さらにそのイメージ内の個々のビットの可視性を知り, 保持する責任を持つ。この責任を果たすために, 以下のプロトコルを定義する。

```
class ShapedImage inherits RasterImage is
  isVisibleAt(index:Point):Boolean;
  makeVisibleAt(index:Point):void;
  makeInvisible(index:Point):void;
```

ShapedImage の形を規定するための方法には多くのものがある。例えば, すべての可視ビットを包含する多角形を規定するなどである。ここでは ShapedImage の形を記述するために以下のプロトコルを追加する。

```
  shape():Polygon;
  shape(aPolygon:Polygon):void;
end ShapedImage;
```

もしやりたければ, 閉じた領域を規定する関数を通じて ShapedImage の形を記述することを考えることさえもできる。

ShapedImage に対するプロトコルを見ると, ある点が可視か否かは RasterImage に確認するのが理にかなっていることが分かる (矩形の境界を含む内部のすべての点は可視であり, その他のものは不可視である)。このため, ついでに isVisibleAt() プロトコルを RasterImage に移す。

この責任駆動アプローチに従うと, RasterImage と ShapedImage のオブジェクト構造については, まだ規定していないし, そればかりかなんら関わってもいない。各々のオブジェクトに対して望まれる振舞いと関連する責任の組を十分に規定することの方をはじめに選ぶ。構造についてはインプリメンテーションの段階で注目する。

3.7 責任駆動設計の長所

オブジェクトの構造の詳細がそのオブジェクトのインターフェースの 1 部となるときには, カプセル化は危険にさらされる。これは設計者がこのような構造的な詳細の知識を利用する場合にのみ起こり得る。つまり責任駆動設計でカプセル化が最大限になされるのは, 設計者が故意にこの情報を無視した場合である。

ポリモルフィズム (polymorphism) はカプセル化を向上させる。なぜなら, オブジェクトのクライアントは, 要求されるサービスをオブジェクトが**どのように** (*how*) インプリメンテーションするかのみならず, **どの** (*which*) クラス (型) がその要求に応答するのかまでをも知る必要がないためである。責任駆動のアプローチは, 設計者にインプリメンテーションとは無関係に責任に対して焦点を絞らせることで, 標準のプロトコル (メッセージ名) を決定 (identify) することを支援する。これはポリモルフィズムを促進する。

構造の知識を持たずにクラスの設計をすると, インヘリタンスの階層の設計が型の階層に従うようになる。何故なら, クラスの型だけが知られているためである。インヘリタンスの階層が型の階層に従っている場合, それぞれのクラスの型はそのクラスのスーパクラスの型のサブタイプ (subtype) となる [1] 。これには 2 つの利点がある。

- サブクラス・クライアント関して, カプセル化を向上させる [4]。これは, すべての継承された振舞いはサブクラスのコントラクトの一部であることが保証されるためである。
- 抽象クラスの決定を容易にする。抽象クラスを明らかにする際の難しさは, 一つには既存のクラスのプロトコルのどの部分が抽象クラスの型の一部であるのか, そしてどの部分がインプリメンテーションの詳細であるのかを決定する点にある。しかし, あるクラスのプロトコ

ルはそのクラスの型を決めるメッセージしか含んでいないので, この問題は考慮する必要がない。

4 責任駆動設計とデータ駆動設計との比較

責任駆動設計はオブジェクトの**構造** (structure) やその他のインプリメンテーション方法についての決定をする前に, オブジェクトの**振舞い** (behavior) を規定する。この方法は大きな設計変更の際に必要とされる再作業を最少化することも分かっている。

責任駆動設計を用いると抽象クラスの定義を早い段階で行なうことができる。例えば, `RasterImage` オブジェクト階層の責任駆動設計を見ると, `ShapedImage` および `RectangularImage` の双方のスーパークラスとして, `RasterImage` という抽象クラスを規定することを考えるのは理にかなっている (新たなクラスが `RasterImage` の以前の振舞いをする)。こうすると, すべてのラスタ・イメージが期待する振舞いを抽象的に規定でき, イメージの新しい型をより素直に付け加えることができる。

オブジェクトの構造は, 責任駆動設計のインプリメンテーションの段階で規定される。この時に実行に関する検討をしてオブジェクトの持つ責任が調整される。インプリメンテーション段階の目標は, クライアント/サーバ・コントラクトの再検討の際にカプセル化を保持することである。

他方, データ駆動設計では, 早い段階でオブジェクトの構造を規定する。つまり, 抽象的なクラスを構築することは, 抽象概念を見い出して具体的なクラスの集まりからそれを抜き出すことを意味する。データ駆動で設計されたオブジェクトのクライアント・オブジェクトは, そのオブジェクトの特別な構造に依存することができる (そしてしばしば依存させる) ので, クライアントとサーバの間のコントラクトを保持しながら抽象概念を生み出す作業はより困難なものとなる。

5 デメテル (Demeter) の法則との比較

デメテルの法則もクラス間の依存度を引き下げ, カプセル化を向上させることを試みている [5][6]。この法則は以下のものに対してのみメッセージを送ることができると述べている。

- メッセージの引数
- インスタンス変数
- メッセージ送信によって返される新たなオブジェクト
- 広域 (global) 変数

この法則の強形式 (strong form) では継承されたインスタンス変数にメッセージを送ることを認めていない。この法則の弱形式 (weak form) では継承されたインスタンス変数を参照することを認めている。

デメテルの法則は, オブジェクトへのアクセスを制限することでカプセル化を達成する。我々の観点からすると, この法則はオブジェクト間の連結の可能性を過度に制限している。デメテルの法則は, オブジェクト指向プログラミングをデータ駆動の観点から見た成果である。メッセージを送ることにより返ってくるすべてのオブジェクトは, カプセル化を破壊する可能性をあるとみなされる。それは, これら返されたオブジェクトが受取側の内部構造を反映する可能性を持つことによる。

すべての返されるオブジェクトをトロイの木馬として扱うこの観点は, 我々の持つクライアント/サーバ関係からの観点とはまったく対照的である。責任駆動アプローチでは, 返される値はクライアント/サーバ・コントラクトの一部である。従ってあるオブジェクトの構造とメッセージによって返されるオブジェクトの間の相互関係は必要とされない。

デメテルの法則はカプセル化を向上するものではあるが, 同時にクラスから提供されるサービスの集まりの設計もしくはインプリメンテーションを混乱させるものでもある。返される値 (受取側の構造の一部であることが分かっているもの) にメッセージを送ることを避けるために, Lieberherr, Holland そして Riel はすべての仲介するサーバ・オブジェクトにメッセージを付加することを提案している [5][6]。それらメソッド (method) は, オブジェクトの構成要素にメッセージを中継するために存在する。このカプセル化の技法が過度に使用されると, あるオブジェクトの持つどこからでも (publicly) アクセス可能なインスタンス変数によって理解され得るすべてのメッセージは, それに対応する中継メソッドを持つことになってしまう。オブジェクトにこのようなメッセージを付加することは, ほとんど確実に型の階層を破壊することになる。これにより保守性と拡張性が危険にさらされる。

デメテルの法則の弱形式は, オブジェクトの外部クライアントに対するカプセル化を提供する。この法則の強形式は, サブクラス・クライアント間のカプセル化を提供する。しかし自己 (self) のカプセル化を実現することに相当する形式はこの法則には存在していない。責任駆動設計では, 自己, サブクラスそして外部クライアントのカプセル化を変数のアクセスの制限 [13][8] という同一の技法を通じて行なう。

6 言語支援

設計段階で達成された高度のカプセル化を維持できる言語支援の形式は数多くある。この中から, 変数へのアクセス制限と振舞いへのアクセス制限の 2 つを論ずる。ある言語

がこれらのどちらかあるいは双方への直接の支援を提供していない場合には, 書式の規約で同じ結果を達成するように補われているはずである。

6.1 変数へのアクセス制限

オブジェクトの状態 (変数) へ直接アクセスすると, クライアント/サーバ・モデル[12] の節で定義したコントラクトの 3 つの型のうち, 少なくとも 1 つに関してはクラスのカプセル化を壊すことになる。このコントラクトの型は, 状態にアクセスするクライアントの型に依存する。

数ある中でも Smalltalk-80 [3] と Objective-C [2] が特に, 外部クライアントがサーバの構造にアクセスすることを禁止している。Trellis/Owl [8], Self [11] および Modular Smalltalk [12] といった言語は, インスタンス変数を完全に排除することでクライアントのすべての型のカプセル化を実現している。

6.2 振舞いへのアクセス制限

状態へのアクセス制限に加えて, ある言語ではクラスの振舞いに対するアクセス制限をプログラマに認めている。外部クライアントに認められるすべてのアクセスはサブクラス・クライアントにも認められると仮定することは理にかなっていると考える。同様に, クラスそれ自身に認められるアクセスは, サブクラス・クライアントに認められるすべてのアクセスを包含すべきである。

Modular Smalltalk はこうしたアクセス・レベルのうちの 2 つだけを支援する。すなわちすべてのクライアントに対する可視性 (public なもの) と自己 (self) に対する可視性 (privete なもの) である。他方, C++ [10] はメンバ (変数もしくは関数) に対する 3 つのレベルの保護をすべて支援する。すなわち, すべてのクライアントに対する可視性 (public なもの), サブクラスと自己に対する可視性 (protected なもの) そして自己に対する可視性 (privete なもの) である。

Eiffel [7] では, よりきめの細かいアクセスの記述を認めている。それぞれの特徴 (変数もしくは関数) は, それにアクセスできるクラスのリストを持つ。このリストはクライアントを包含し, そのクライアントのコントラクトはその特徴を包含する。

7 おわりに

カプセル化は, 再利用性, リファイン容易性, テスト容易性, 保守性, さらに拡張性といったソフトウェア尺度の値を向上させるキーである。オブジェクト指向言語はカプセル化を向上させる多くのメカニズムを提供するが, 最大の効果が得られるのは設計の段階においてである。

オブジェクト指向設計へのデータ駆動のアプローチは, システム内のデータ構造に焦点を当てている。この結果, クラス定義のなかに構造の情報が混入することになる。そうするとカプセル化が破壊される。

責任駆動のアプローチは, オブジェクトの構造と振舞いの双方のカプセル化を強調する。クラスのコントラクト上の責任に着目することで, 設計者はインプリメンテーションの検討をインプリメンテーションの段階まで延ばすことが可能になる。

責任駆動設計がこの問題を扱う唯一の技法ではないが, 他の多くの技法はインプリメンテーション段階でのカプセル化に力を入れる傾向にある。しかし, ソフトウェアライフサイクル上, 最大の恩恵を得るためにはこれでは遅すぎるのである。

参考文献

[1] Luca Cardelli and Peter Wegner, "On Understanding Types, Data Abstruction and Polymorphism", *Computing Surveys*,17(4), December, 1985, pp.471-522

[2] Brad Cox, *Object-oriented Programming, An Evolutionary Approach*, Addison-Wesley, Reading, MA. 1986

[3] Adele Goldberg and David Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, MA. 1983

[4] Daniel Hlbert and Patrick D. O'Brien, "Using Types and Inheritance in Object-Oriented Languages", Technical Report DEC-TR-437, Degital Equipment Corporation, April, 1986

[5] K. Lieberherr, I. Holland and A. Riel, "Object-Oriented Programming: An Objective Sense of Style", *SIGPLAN Notices*, 23(11), November 1988, pp.323 334

[6] Karl Lieberherr and Ian Holland, "Formulations and Benefits of the Law of Demeter", *SIGPLAN Notices*, 24(3), March, 1989, pp.67-78

[7] Bertrand Meyer, *Object-oriented Software Construction*, Prentice Hall, Englewood Cliffs. NJ. 1988 (酒匂 [訳] オブジェクト指向入門, アスキー, 1990)

[8] Craig Schaffert, Topher Cooper, Bruce Bullis, Mike Kilian and Carrie Wilpolt, "An Introduction to Trellis/Owl", *SIGPLAN Notices*, 21(11), November 1986, pp.9-16

[9] Alan Snyder, "Encapsulation and Inheritance in Object-Oriented Programing Languages", *SIGPLAN Notices*, 21(11), November 1986, pp.38-45

[10] Bjarne Stoustrup, *The C++ Programming Language*, Addison-Wesley, Reading, MA. 1987

[11] David Unger and Randall B. Smith, "Self: The Power of Simplicity", *SIGPLAN Notices*, 22(12), December 1987, pp.227-242

[12] Allen Wirfs-Brock and Brian Wilkerson, "An Overview of Modular Smalltalk", *SIGPLAN Notices*, 23(11), November 1988, pp.123-134

[13] Allen Wirfs-Brock and Brian Wilkerson, "Variables Limit Reusability", *Journal of Object-Oriented Programming*, 2(1), May/June, 1989, pp.33-40

読書案内

福良 博史　桜井 修　渡邉 雄一

[1] Kent Beck, Ward Cunningham : A Laboratry for Teaching Object-Oriented Thinking, Proceedings of the 1989, OOPSLA †1 reprinted in ACM SIGPLAN Notices,24(10):1-6,October 1989.

オブジェクト指向による「思考」方法を身につけるために有効な方法として, 著者達が導入した CRC †2カードの紹介と, CRC を用いた用いたオブジェクト「思考」の教育を行った結果の報告である。(言語に依存しない方法論である) 特筆すべき事として, 最終的に固まった CRC カードを, 顧客へ納入される設計ドキュメントの一部になっているケースがあるとのこと。簡単な CRC を利用したサンプルがついている。

個人的な印象としては, オブジェクト指向設計の世界の KJ 法のような感じがする。設計時には, このカード以外にも適宜必要に応じて, 各種ドキュメントが必須となるであろう。

[2] Rebecca Wirfs-Brock, Brian Wilkerson : Object-Oriented Design: A Responsibility-Driven Approach, Proceedings of the 1989 OOPSLA, reprinted in ACM SIGPLAN Notices,24(10):71-75,October 1989.

Responsibility-Driven による設計方法を提唱している。従来の Data-Driven による設計方法は何がいけなかったのか, それを改めるために, この方法の考え方は, どこが従来の方法と考え方を異にしているのかを述べている。

単的に言えば, 「そのオブジェクトが占有している情報は何か?」と言うことをハッキリさせることにある。当然この情報をいじる事が出来るのは, これを占有しているオブジェクトのみである。

[3] Daniel Keller : A Guide to Natural Naming, ACM SIGPLAN Notices,25(5):95-102,May 1990.

名前の付け方のガイドである。ソフトウェアの設計においては, 名前のつけかたの良し悪しによって, 人に理解し易すくもなるし, 誤解を招き易くもなる。このために, どのような基準で名前をつけるのが良いのかを論じ, いくつかのヒントを与えている。

[4] Timothy Budd : An Introduction to Object-Oriented Programming, Addison-Wesley Publishing Company,Reading, MA,1991.†3

CRC カードを用いた設計の紹介とインプリメントを扱った教科書である。ここで紹介されている, 銀行の ATM のサンプルは, CRC カードを用いており, このカードの利用法を理解するために格好の材料となっている。インプリメント用の言語は C++, Objective-C, ObjectPascal, Smalltalk を扱っている。コーディング例は必ずしも, プログラミングの作法からいうと, お進め品というものでもない。各種の OOL の理解には役に立つ。なお, 著者は Little-Smalltalk †4 の作者として著名である。

[5] G. Masini, A. Napoli, D. Colnet, D. Leonard, K. Tombre : Object-Oriented Languages, The A.P.I.C. Series No.34, ACADEMIC PRESS Ltd.(英訳 1991 P. Hoggan, G. Masini)

オブジェクト指向言語 (OOL) の過去の歴史的な経過, 及び各種の OOL の考え方を紹介している。第 2 章のクラスの考え方の説明が, 一つの例を用いてクラスの考え方の色々な概念を説明しており, 有益である。各種のＯＯＬについては, コーディングや図を多用してそれぞれの言語の特徴の理解の促進に努めている。最後の章に各種ＯＯＬの比較表が出ており, 各種言語を概観するのに役立つ。Lisp の使用者には, 付録の ObjVlisp の Common Lisp 版に書き直したものが参考になるかも知れない。

[6] J. マーチン, C. マックルーア [國友 義久, 渡辺 純一 (訳)] : ソフトウェア構造化技法 ダイアグラム法による, 近代科学社

構造化技法中心であるが, 設計ドキュメントを図表形式で表現するための各種のアイデアがある。筆者の評価には, 個人的に賛成出来ない面もあるが, 色々と考えさせられるものがあり有益である。既存の設計手法は, 殆ど網羅されている (しかし日本人のアイデアには, 触れ

†1 Conference on Object-Oriented Programming Systems, Languages and Applications

†2 CRC は Class, Responsibility, Collabolation の略

†3 [編集部注] 翻訳あり。
羽部 正義 (訳):オブジェクト指向プログラミング入門, トッパン, ISBN 4-8101-8048-4

†4 [編集部注] 翻訳あり。
吉田 (監修), 長谷川, 太田 (訳):Little Smalltalk 入門, アスキー, ISBN 4-7561-0003-1

られていない)。

[7] マイヤー [酒匂 寛, 酒匂 (訳)] : オブジェクト指向入門, アスキー

この本は, オブジョクト指向設計について, その根底に流れる発想や哲学を学びたい読者にとっては, なかなかの良書である。第 1 部では, オブジョクト指向設計を「構造化された抽象データ型の集合としてソフトウェアシステムを構築することである」と定義するまで, ソフトウェアのモジュール性についての要因を上げ, その再利用性や拡張性を達成するためにデータ中心の抽象化が重要であると述べ, オブジョクト指向設計へと読者を導いてくれる。

第 2 部では, Eiffel を用いながらその詳細を説明している。設計の観点をシステムが何をするかでなく, システムが何を対象にするかである点を強調するためのケーススタディやクラスの見つけ方, 継承についての技術にも説明を加えている。第 3 部では, 他の環境におけるオブジョクト指向的概念の実現について触れている。

Fortran, Pascal, C さらに Ada, Modula-2 そして Simula, Smalltalk など Eiffel 以外の言語を対象にしているので読者のそれぞれの立場で読み進めることができる。

著者 Meyer 氏は, オブジョクト指向設計がソフトウェアの品質を高める可能性を持っていると確信していると言う。さらに従来の設計手法を異なるばかりではなく, ある程度両立していけるものであるとも言っている。その意味からもオブジョクト指向の入門としてだけでなく, ソフトウェア工学の入門書としても適していると思う。

[8] M. Jackson (大野, 山崎 [監訳]) : システム開発 JSD 法, 共立出版, 1989 (ISBN:4–320–02500–8)

今更説明の必要は無い。JSD の原典の訳書。これを読まずして JSD を語れない。

[9] 有澤 誠:ソフトウェア工学, 岩波書店, 1988 (ISBN:4–00–007693–0)

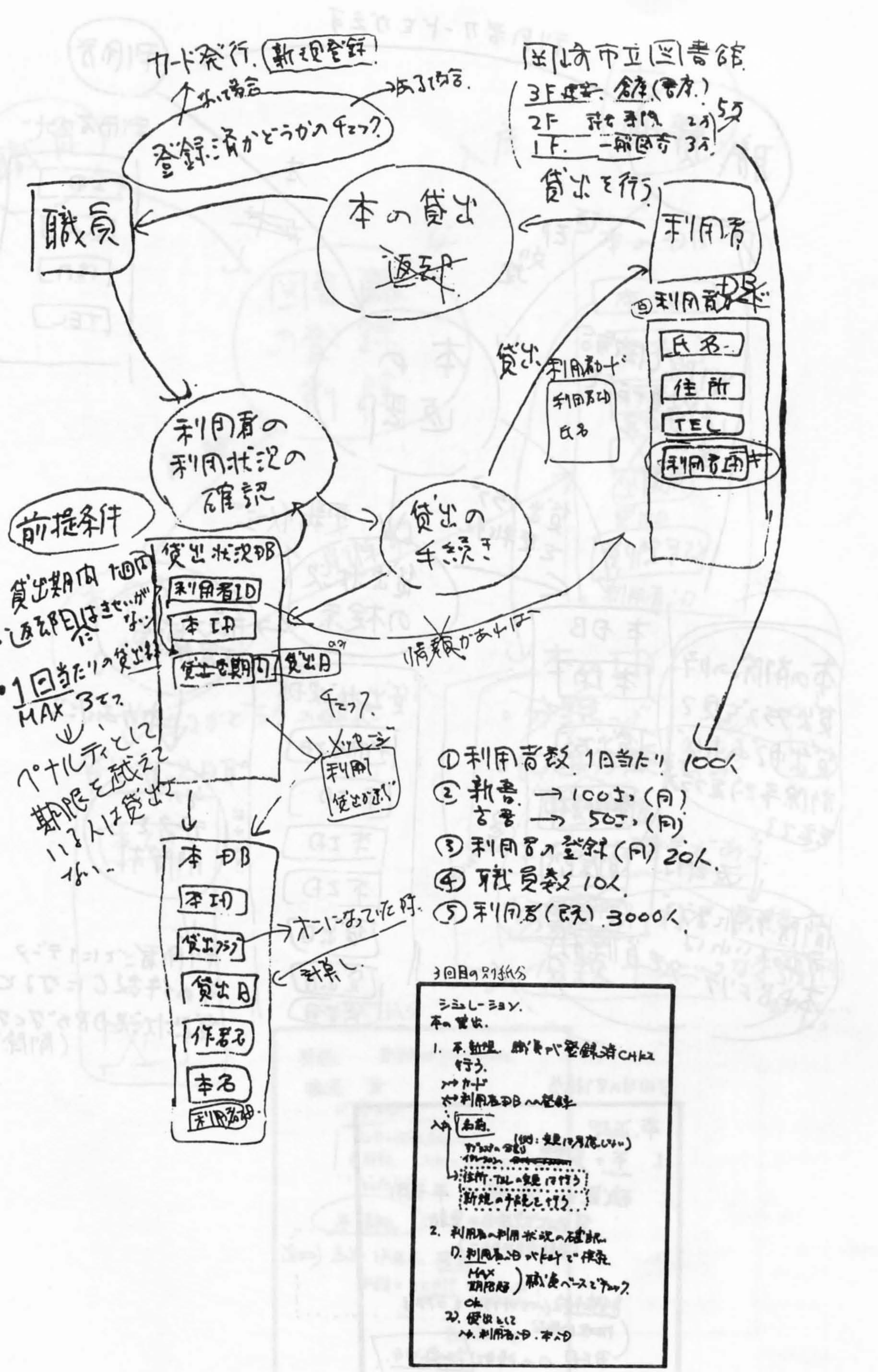
ちょっと, 古い本なので, 必ずしも熟読の必要は無いが, ソフトウェア工学の全般に関することを適当な分量でまとめてある。オブジェクト指向以前まで, すなわち JSD までの設計法に関して簡単に知りたい人には適当。

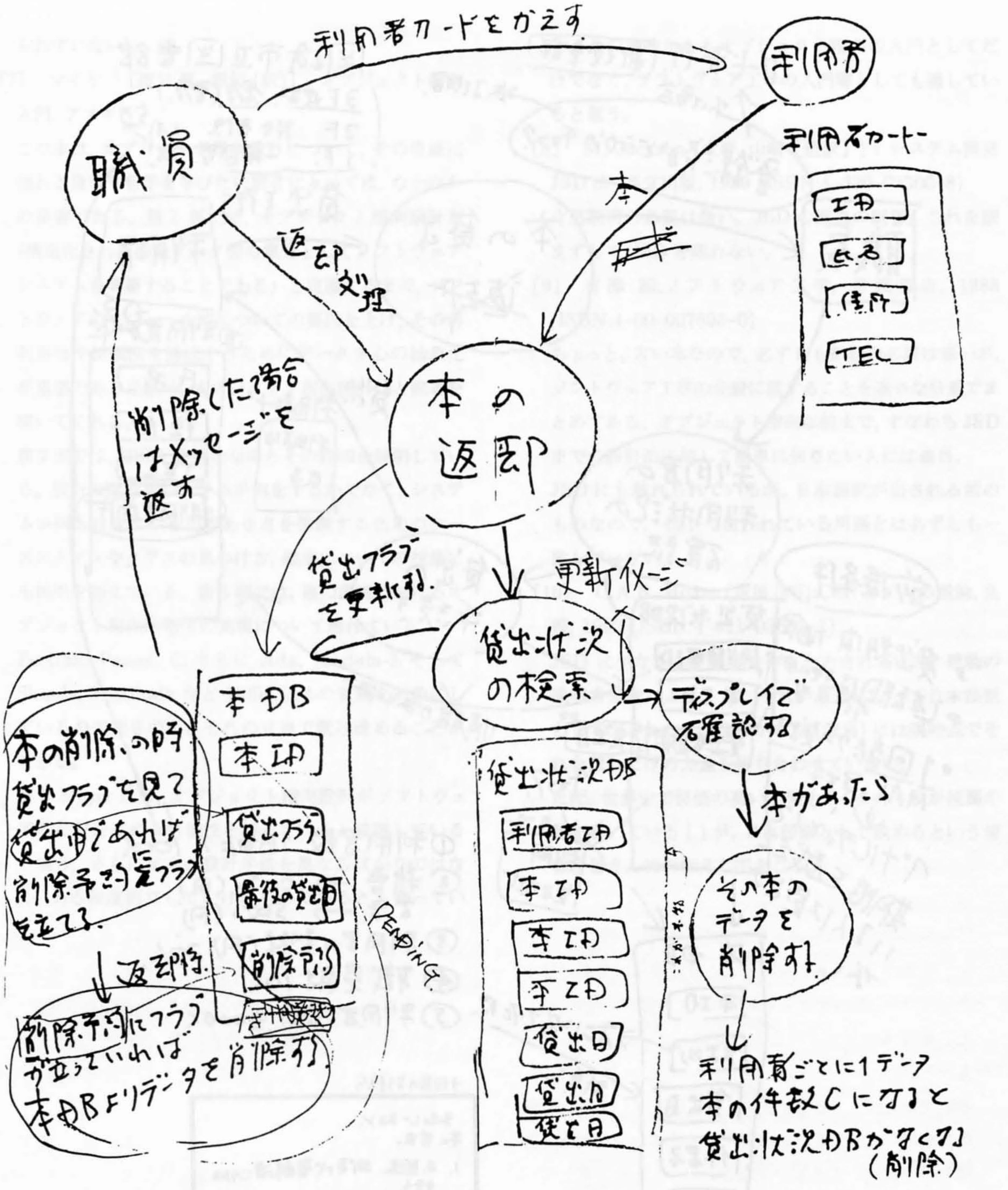
JSD にも触れられているが, 日本語訳が出される前のものなので, [8] で使われている用語とは必ずしも一致していない。

[10] C.A.R. Hoare (吉田 [訳]):CSP モデルの理論, 丸善, 1992 (ISBN:4–621–03683–1)

JSD に少なからず影響を与えたとされる CSP 理論の提唱者自身による 1 冊。それが最近になって日本語訳として出された。残念ながら私 (渡邉) には現時点でそれを読むだけの力量を持ち合わせていない。

ただ, 世界中で評価の高い 1 冊 (ダイクストラが推薦の辞を書いている！) が, 日本語訳として読めるという情報を皆さんにお伝えしておこう。


カード発行
新規登録
登録済かどうかのチェック
岡崎市立図書館
1F. 一般図書 3万
5万
貸出を行う
職員
本の貸出
返却
利用者
氏名
住所
TEL
利用者カード
利用者ID
氏名
利用者の利用状況の確認
貸出の手続き
前提条件
貸出状況DB
利用者ID
本ID
貸出期間
貸出日
情報があれば
チェック
本DB
本ID
貸出フラグ
貸出日
作者名
本名
オンになってた時
①利用者数 1日当たり 100人
②新書 → 100こ (月)
古書 → 50こ (月)
③利用者の登録 (月) 20人
④職員数 10人
⑤利用者(既) 3000人
シミュレーション
本の貸出



3回目の打ち合わせ分

本.返却

1. 本+利用者

職員が (本ID)で 本DB.

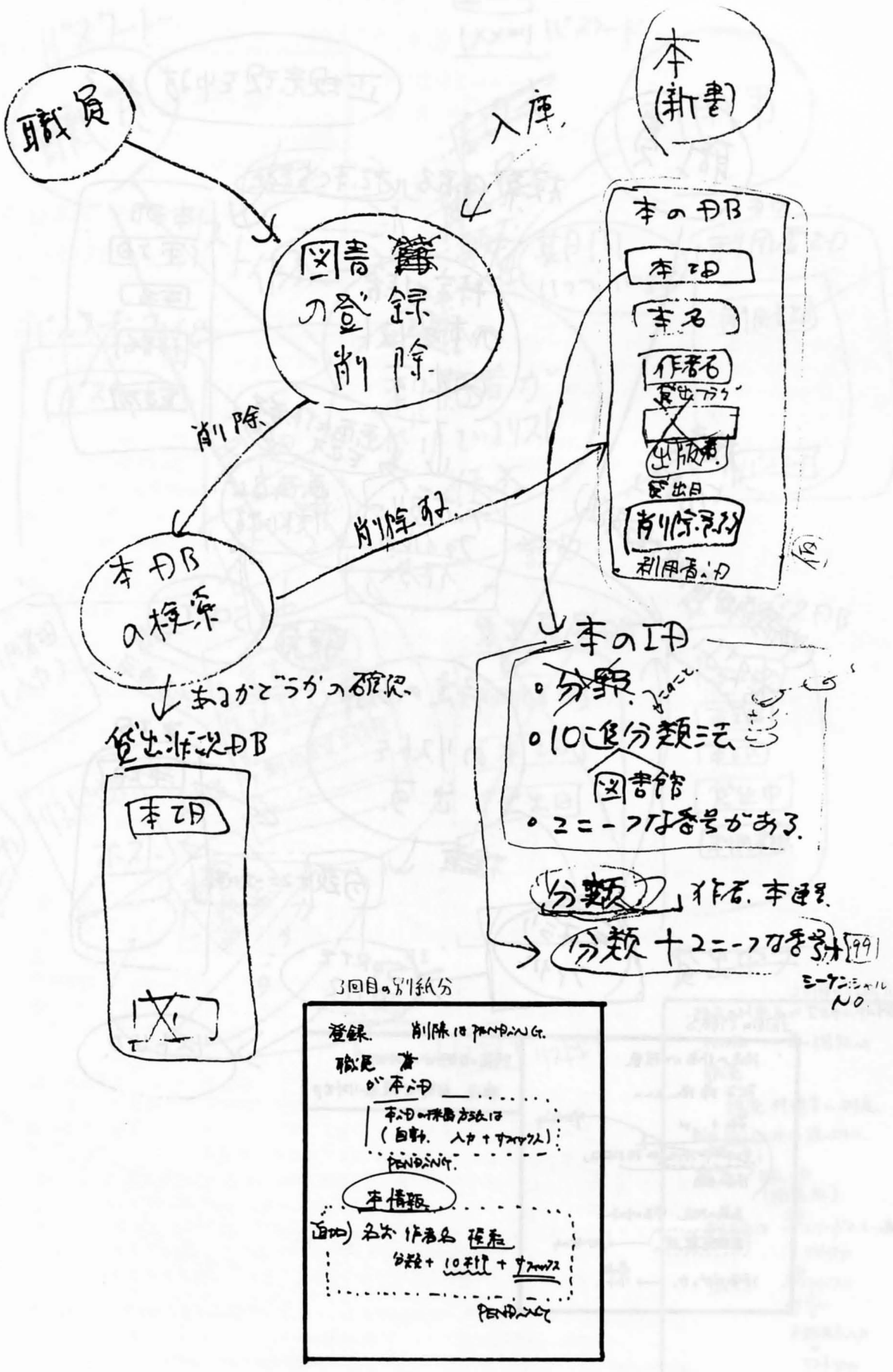
貸出状況DBの更新.

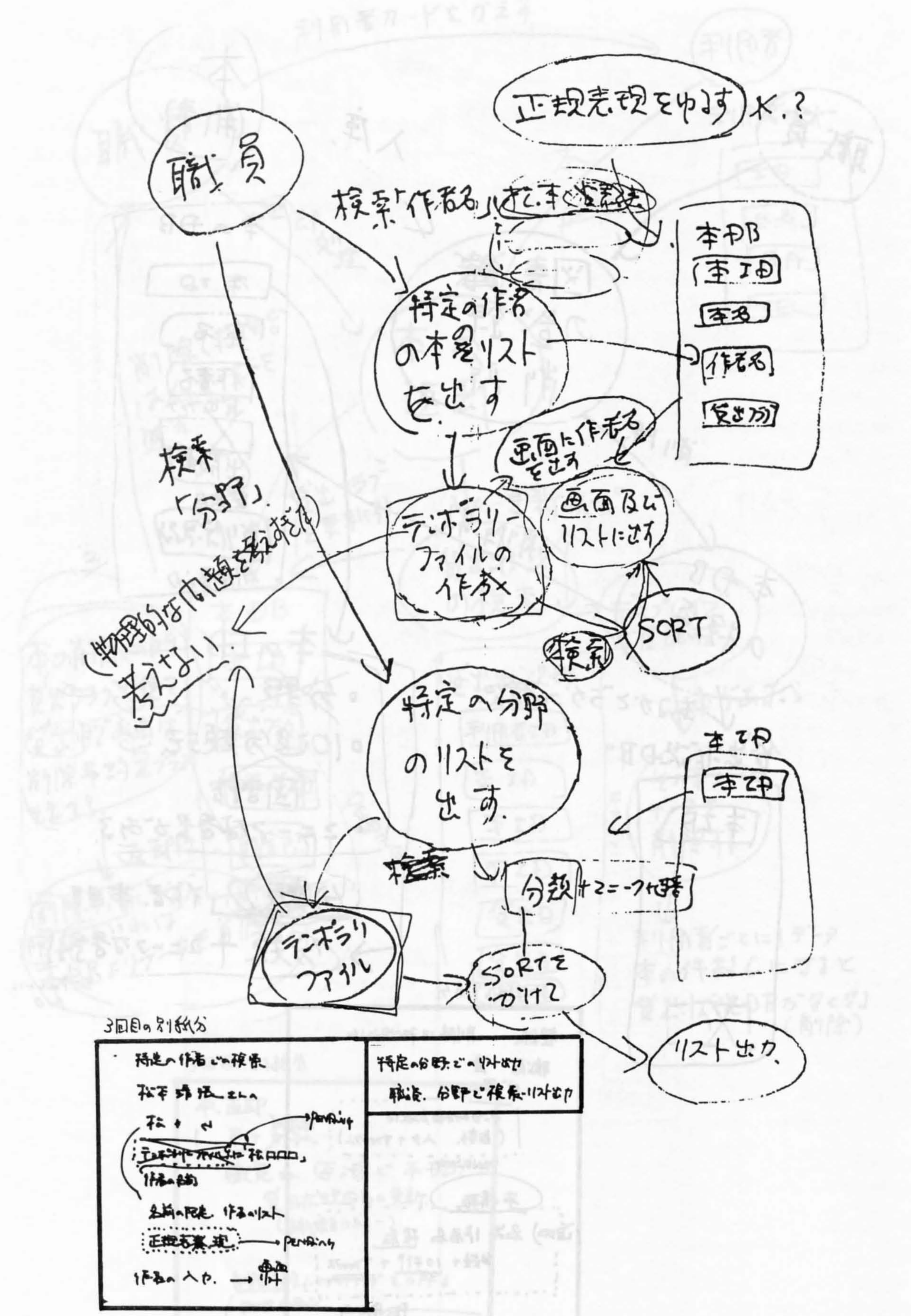
(更新項目はある..)

削除予約(バリッドフラグ)をOFFする.

物理的削除.

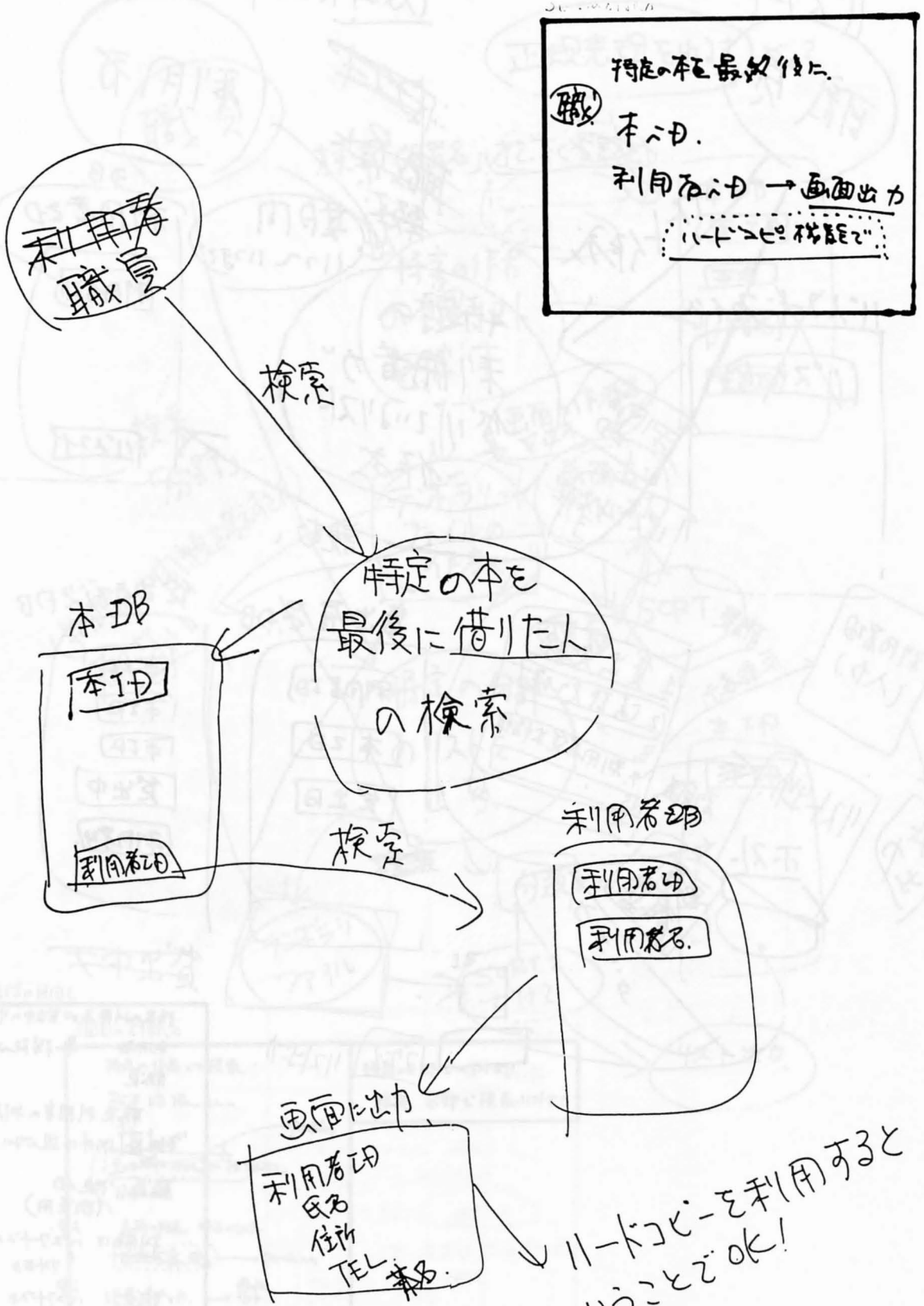
DFD.0の時まで PENDING.


職員
本(新書)
入庫
図書の登録削除
削除
本のDB
本ID
本名
作者名
貸出フラグ
出版者
貸出日
削除予約
利用者ID
削除する
本DBの検索
あるかどうかの確認
貸出状況DB
本ID
本のID
分類
10進分類法
図書館
ユニークな番号がある
分類
作者、本連号
分類＋ユニークな番号
99
シーケンシャルNO
3回目の別紙分
登録 削除はPENDING
職員 が本ID
本ID の採番方法は
(自動 入力＋サフィックス)
PENDING
本情報
追加) 名前 作者名 検索
分類＋10年代＋サフィックス
PENDING



職員
正規表現をやるか.?
検索「作者名」
特定の作者の本をリストを出す
画面に作者名を出す
画面及びリストに出す
テンポラリーファイルの作成
SORT
検索
検索「分野」
（物理的な問題を考えない
特定の分野のリストを出す
検索
分類ナンバーで
テンポラリファイル
SORTをかけて
リスト出力
3回目の別紙分
特定の作者での検索
特定の分野でのリスト出力

ID.
パスワード
職員
リスト作成
利用者
期間
いつ〜いつまで
特定の利用者が貸りているリスト作成
パスワードファイル
パスワード
利用者ID
パスワード
読む
貸出履歴DB
貸出状況DB
利用者ID
本ID
貸出日
本ID
本ID
本ID
貸出中
利用者ID
利用者ID
職員
一般
1. 貸出処理
2. 返却処理
3. 〃
4. 利用者貸出情報
5. 〃
利用者ID(入力)
リスト出力
ホスト
画面確認
カウンタ
マミー
プリンタ
リスト出力
貸出中フ
3回目の別紙分
特定の利用者が貸出中のリスト出力.
利用者 一単一機能のみ
職員
職員利用者の判定
利用者 4 以外は選べない.
職員. 特定ID
(職員用)
・利用者は パスワード入力→画面→プリント
リスト出力
・職員. パスワード入力
メニュー
利用者名入力
リスト出力


特定の本を最後に借りた人
職
本ID.
利用者ID → 画面出力
ハードコピー機能で
利用者
職員
検索
特定の本を
最後に借りた人
の検索
本DB
本ID
利用者ID
検索
利用者ID
利用者ID
利用者名.
画面に出力
利用者ID
氏名
住所
TEL
書名.
ハードコピーを利用すると
いうことでOK!

第10回 夏のプログラミング･ワークショップ

現場でのシステム設計を改革するテクノロジを求めて

― 設計における問題点とはなにか？ ―

参 加 者 募 集

主催：ソフトウェア技術者協会

夏のプログラミング･ワークショップは，リーダ役のスタッフのもとに２０代半ばの「若手」が数名のグループに分かれて集中討論／発表準備／成果発表／報告書作成などを実践的に行なうもので，企業，大学の壁を越えた積極的な活動姿勢を身につけるのに大変良い機会だと定評のあるものです．ＳＥＡの内部では「若手の会」と呼ばれるこのワークショップも，早いもので10回目になりました．例年は盛岡で行なわれることが多いのですが，今回は「1年おきくらいに西の方でも開催したい」と考え，暫定的に場所を愛知県は岡崎に移してみました．

今回のテーマは「設計」です．今回の目標の１つは共通の課題をグループ毎に異なった方法論(オブジェクト指向，ジャクソン法，構造化設計など)で解くことにあり，これを足がかりに設計作業に関する討論を行なっていきます．

ワークショップの前半(9/3)は，グループ毎の勉強と共通課題の回答に取り組むことになります．個々のグループは全く別の活動を行ない，実際にグループがどのような結論を導くかは，すべてグループのメンバに任されます．

さて，せっかく複数のグループが存在するのですから，ただ独立に活動したのでは意味がありません．そこで，他のグループが一体どのような討論を行なっているのかを知るための場として，期間途中に「中間発表」を設けます．今回は形式を変え，中間発表もグループ構成を変えた小グループ内で並行して行なうことにします．このため，参加者全員が最低１回の発表機会を得ることになります．

後半(9/4)は，中間発表を踏まえての軌道修正を中心に，「設計作業」に関連して抱いている各人の問題点，採り入れている技術なども合わせての討論を行ないます．

最終日には各グループの最終成果をグループ毎に発表します．そして，その後の質疑／全体討論結果などをふまえて，解散後に「グループ活動報告書」を作成するところまでがこのワークショップの作業です．参加者が自主的に活動することを望みます．

この４日間の活動を通じて，参加者全員が自分自身の現在のポジションを確認し，それぞれの会社の将来を担う中核技術者としてのビジョン/スタンスを確立することに役立てられれば何よりです．

例年どおり，意欲あふれる多くの若手エンジニアの方々の参加をお待ちしています．

開 催 要 領

期　　日： 1992年９月２日（水）から９月５日（土）まで（２日13:00開始，５日14:00解散）

場　　所： ホテル　レク･ワールド岡崎（〒444-21 愛知県岡崎市鴨田町所屋敷8-3; 0564-25-1311）

定　　員： 25名

参加資格： 年齢30歳以下，経験２年以上のソフトウェア技術者

費　　用： 55,000円（SEA正会員） 70,000円（SEA賛助会員） 80,000円（一般）
現地集合･現地解散とし，期間中の宿泊費（２日夕食から５日昼食まで９食付き）および資料代を含みます．

実行委員長： 渡邊雄一（アスキー）

プログラム委員長： 塩谷和範（ＳＲＡ）

プログラム委員： 岡本隆一（さくらＫＣＳ），桜井修（池上通信機），
後藤浩（シスプラン），福良博史（ＪＭＡＳ），豊田英利（セス）他の予定

申込方法： 裏面の参加申込用紙の必要事項を記入の上，1992年７月10日（金）までに，郵便またはE-mailで，SEA事務局宛お送り下さい．(入力の手間を省くためにE-mail等の電子媒体を歓迎します．フロッピの場合は，MS/DOSもしくはMacintoshのテキストファイルに限定させて頂きます.)

なお，申込者多数の場合には，プログラム委員会でポジション･ステートメントの内容を審査の上，８月初旬までに参加者を選定させていただきますので，あらかじめ御了承ください．

申込み先： 〒160 東京都新宿区四谷３-12丸正ビル5F　ソフトウェア技術者協会
Tel: 03-3356-1077　　Fax: 03-3356-1072
E-mail: wakate92@sea.or.jp (junet), MHG01727(NIFTY), shioya(mix)

岡崎市東岡崎駅前飲食店地図

うどん　喫茶　タイ料理　鳥料理　レストラン
鰻　寿司　中国料理　日本料理　その他
お好み焼き　スパゲティ　定食　焼肉
割烹　そば　田楽　ラーメン

編集後記に代えて – 最近の SIGENV の活動について –

渡邉 雄一

なんとか, 報告書をまとめあげることが出来ました。『今回はさっさと編集作業を切り上げて 11 月には出すんだ!』などと言っていたのですが, なんだかんだで結局年を越すはめになりました。みなさんどうもすみません。もうほぼ出来たと思ってからが大変なもんなんですね〜, 編集作業ってのは。

今回のこの報告書で, ちょっとだけ残念なのは, 体裁が整わないために, 積み残した材料が若干あることです。具体的には

1. 事前のポジションペーパの審査評 や
2. 中間発表のレポート

がそうです。†1

ところで, 私が SIGENV の世話人をするようになって随分と時間がたったように思います。何故か, 大昔†2のように華やかな活動をするわけでもないのになんとか続いてます。まあ, 実質 2 人しか集まらないなんていうときもありますが。折角ですので近況報告と今後の活動予定などについて, 独断と偏見も交えて紹介したいと思います。

最近の SIGENV の形式は依然として勉強会が続いています。世間のお堅い人に言わせると雑談と言うらしいのですが, まあ我々にとってはそんな雑音はどうでも良くて, マイペースで適当な文献を見つけてそれを読んでいます。最近の主たるテーマは**オブジェクト指向分析/設計**です。その 1 つが今回のワークショップで紹介させてもらったようなものです。

今回取り上げた Budd の本は たまたま 1992 年の 10 月の終わりくらいに翻訳が出版されましたが, すでに我々はそれを原文で読むことを 1992 年 1 月から開始していたわけです。今回は, ワークショップのレポートということで実際に参加してくれたメンバのレポートが中心になっていますが, それと同じ様なことは我々もその本の中の練習問題にあった「カードゲーム 21」で体験しています。実際に同じ 1 つの問題†3を複数の人間が顔を突き合わせて議論するのはなかなか面白いものです。

で, 「そんなことをやっているのならその様子を SEAMAIL に載せれば良いではないか」という意見が少なからずあることも分かっているのですが, 世の中には著作権という問題があってあまり勝手なことは出来ない訳です。一応, 私は出版社に身を置く者ですから, 本のゼロックスコピーには顔をしかめたくなります。もちろん ACM の主要な雑誌のように, 用途によってはそれを妨げない旨の但し書きがあるものは別ですが…。それほど極端でなくとも, やはり方法論の紹介なども含めた上で我々なりの議論の結果をレポートしようとすると, その背景の説明をする必要があります。そのときにそれをどう紹介するのかに非常に気を使うことにならざるを得ないのです。

具体的な例をあげれば, 2 年前の 11 月に同じ岡崎での合宿を開催したことがあり, ここでは ACM から刊行されている OOx の本を全員で読んだのですが, このうちの 1 つ

> Lynn Andrea Stein, Henry Lieberman, David Ungar : "A Shared View of Sharing : The Treaty of Orlando." In Won Kim and Frederick H.Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, chapter 3, pages 31-48, Addison-Wesley Publisihng Co. Reading, Mass., 1989.

は結局その後 1 年近くをかけて読み直すはめになって, 我々に非常にインパクトを与えてくれたのですが, 残念ながらこれも著作権の問題から公の刊行物に掲載することは出来ません。†4

今回の報告書のなかでもこの点は気を使いました。具体的には CRC カードの例題などは Budd の本の中に適当なものが幾つも転がっていて, それを転載することの方が楽なのですが, いくら我々が原本から独自に訳したとしても既に訳書も刊行されているので, やはり適当ではなかろうということで, グレーゾーンに足を踏みいれることは避けました。が, そんなことはものともせずに, Budd の本にまさるとも劣らない例を福良さんが書いてくれたのには, 本当に感謝しています。

ところでちなみに, 上記の Stein らの論文は [1] で,

> A good characterization of different kinds of inheritance

と紹介されています。また, この中で 今回翻訳して紹介し

株) アスキー

†1 但し, 今回は最初から最終発表に使われた OHP などを資料として縮小コピーのような形で載せるのは絶対に避ける方針でいました。その理由は, OHP は発表の際に見せる資料であって, 報告書のような読ませる読ませる資料には不適等と考えたからです。

†2 正確には最初の 2 年間 (1986〜1987)

†3 それも普段の個々人の仕事とはちょっと違ったものを!

†4 SEAMAIL のような non profit な刊行物はグレーゾーンなのかもしれませんが, 敢えて面倒な事は避けたいので …

た Wirfs-Brock らのものは

Good introductory articles include the following:

として他に 6 つの論文と一緒に紹介されています。

さて, 何故**環境分科会**と銘打っているのにオブジェクト指向に拘っているのか? と言われると話はいろいろと長くなりますが, その答の 1 つは[†5], それがやはり環境を考える上で少なからず重要な対象のとらえ方であるということです。例えば, ここ数年来 CASE がいろいろと話題にあがりました。特にそれらのうち統合化 CASE といわれるものの中で, 使える (使いものになる) ものの背後になんらかのきちんとした方法論があります。[†6]で, 例えば仮に そのような CASE ツールを使用/評価することになったとしましょう。その場合, その評価をする際のファクタとかそれらの重み付けとか色々と議論出来ると思うのですが, 重み付けといったようなことは, 実際に評価する段になって決めるべきもので共通の問題とは考えにくい訳です。で, 色々とあるファクタのうちで, そのツールが

1. どのような方法論に立脚していて,
2. それらをどの程度正確に, あるいはどのような解釈で,
3. 何をどのように支援したり拡張したりして,

具体的な製品に反映させているのか?

ということを見究める必要があります。このようなことを探る上で, 方法論の 1 つとして オブジェクト指向分析/設計に関するリサーチは現段階で極めて流動的で 最も興味深いものの 1 つであると言うことが出来ると思います。それゆえ, 狙いはメンバそれぞれに多少違うかもしれないけれども, 共通の話題として OOx (特にオブジェクト指向分析/設計を) 一緒になって考えているわけです。

我々メンバは, 普段は必ずしも理想の環境にいるわけではありませんから, 日常の仕事の憂 (う) さはらしを兼ねて夢の世界を除き見るようにオブジェクト指向を見ていなくもありません。が, 何も究極の夢だけを追いかけ 机上の空論を繰り返しているだけでもありません。**月例会**と称するミーティング (兼 呑み会) は, 雑談を通して お互いの疑問やヒントなどの計り知れない情報交換の場です。例えば, 福良さんが紹介してくれた PC-98 で Little Smalltalk を動かしてみた例なども, その一例です。IBM-PC があって Smalltalk/V が使える人はそれでも良いでしょう。また Macintosh の恵まれたユーザなら Objectworks\Smalltalk が使えるでしょう。もちろん Unix があってそれを自分の裁量で自由に出来る人なら GNU Smalltalk を install すれば良いでしょうし, お金さえ許せば 本家本元の Objectworks\Smalltalk を存分に使えるのでしょう。でも, それらがままならない人でも出来る範囲でちょっと工夫をすることで勉強のきっかけは作れるわけです。(SIGENV に限らず SEA での) 分科会活動は, そのようなきっかけを見つけられる場だと言っても良いでしょう。で, ある程度集中して一つのことをやらないとモノにならないのは何でも同じだと思うのですが, そのきっかけをもってして**深み**にハマルらないようにするのが 勉強のコツなわけですよね!。で そうゆうことも, SIGENV でみんなと意見を交換していると ごく自然に見えてくるのです。

月例会の開催のペースは, 他の分科会と同様に原則的に毎月 1 回です。が, 年末から翌年の 4 月くらいまでは事実上殆ど活動出来なくなります。みんな何かとドタバタして全員が集まれるように調整が難しいのです。それでもやっと 5 月くらいになると, 何とか調整も容易になって気合いが入ってきます。多い月には 3 回も開催するなどということもあります。このペースが大体 8 月くらいまで続きます。で, 夏休みを契機に暫く活動が途絶え気味になりつつも, その年の終わりまでは, 5〜8 月までの材料を再確認などしたり, 秋に開催される種々のイベントをメンバ相互に情報交換したりといったことが, ここ数年の開催パターンです。

'93 年の活動テーマは, 特に決っていませんが, Budd の本は翻訳も出てしまったことですので 4 月末位をメドに, 適当なところで切り上げられるように努力したいと思っています。5 月以降は適当な本が見つかればそれを読むことになるでしょう。今のところ目を付けているのは,

1. Comunications of the ACM
2. ACM OOPSLA Proceeding
3. ACM OOPS MESSENGER

の中から適当な物を読んできて, 発表するなんてゆうのも悪くないね〜!? と皆で話している所ですが … 。

という訳で, しばらくは皆さんに直接の成果は報告することが出来ないかもしれませんが, 「便りの無いのは元気な印」ということで SIGENV はひっそり (?) と活動を続けていますので, 興味のある人は是非とも参加してみて下さい。

参考文献

[1] Gary T. Leavens : "Introduction to the Literature on Object-Oriented Design, Programming, and Languages" *OOPS MESSENGER*,2(4), Octover, 1991, pp.40-53

†5 ここは渡邉の独断と偏見で SIGENV メンバの総意ではけっしてありませんので注意して下さいネ!

†6 Upper CASE とは, 上流工程の分析/設計といったメタな部分の支援をするのですから当然ですね

call for papers, tutorials, panel sessions, exhibitors

Conference on Software Maintenance and Tools Fair '93

Montréal, Québec, Canada • September 27-30, 1993

Improving Software Through Maintenance

Great software systems are maintained, not developed. Software must be built h maintenance in mind. Software systems y become truly well adapted to user needs ough maintenance. Maintainers must take a active view of their role, not just fixing blems but also improving the quality of the tware over time. CSM-93 seeks papers and er contributions that advance the state-of- art or state-of-the-practice of software main- ance, especially those related to improving tware through maintenance.

ggested Topics

ers, experience reports, panel session pro- als, and tutorial proposals related, but not ited, to the following topics:

- *signing for maintainability and adaptability;*
- *ing with legacy systems;*
- *ftware evolution, migration, and conversion;*
- *hieving and maintaining fitness for use;*
- *engineering, reverse engineering, and tructuring;*
- *andards, methodologies, processes, lifecycles;*
- *rmal methods applied to maintenance;*
- *pirical maintenance studies;*
- *ftware maintenance and maintainability trics;*
- *pact analysis;*
- *ftware maintenance management;*
- *ftware maintenance education;*
- *ogram understanding;*
- *ftware maintenance tools;*
- *siness reengineering;*
- *cumenting for software maintenance.*

SCOPE.

Software maintenance includes enhancing, adapting, and correcting software. CSM-93 will provide a forum for discussing software maintenance through refereed papers, experience reports, panel sessions, tutorials, exhibits, and informal meetings. CSM-93 will present the most important current practical, experimental, and theoretical work being conducted to support software maintenance. The conference is open to everyone involved with or interested in software maintenance: practitioners and researchers from industry, government, and academia.

SUBMISSIONS.

Submit 5 copies of papers, experience reports, or panel proposals in English to one of the program chairs at the addresses shown. Accepted authors are expected to present their work at CSM-93.

Papers.

Submit 1000-5000 words, which have not been previously published and are not under consideration for publication elsewhere. Include on the first page the title, all authors' names, lead author's address and phone number, and an abstract (250-word maximum). Authors desiring to propose a tool demonstration as part of their presentation should provide a 1-page explanation of the demonstration and how it relates to the paper.

Experience Reports.

Summarize experiences with practical application of software maintenance technology or process (e.g., using a tool or method, applying a metric, following a project management discipline) and highlight outcomes and lessons learned. Identify submission with the keyword "Experience" at the top of the first page. Follow the instruction for papers thereafter. Submissions will be evaluated for their applicability and relevance to the software maintenance practitioner community.

Panel Proposals.

Include title, proposed chair, tentative panelists (with short vitae), a brief description of the panel session subject, and supporting rationale. Panelists must have agreed to participate before submission of the proposal.

Tutorials.

Proposals for full-day and minitutorials (2-4 hours) should describe tutorial experience and material maturity. Direct 5 copies of proposals to the tutorials chair.

Tools exhibitors.

Send to the tools chair 2 copies of a 1-page tools description that explains its applicability to software maintenance. Identify submission with keywords "Research Prototype" or "Vendor Tool" at the top of the page. Define the amount of exhibit space required.

Important dates

Submission deadline: April 16, 1993
Acceptance notification: June 25, 1993
Final version due: August 6, 1993

Conference Committee

General Chair
Marc Kellner
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890 USA
412.268.7721; FAX 412.268.5758
email: mik@sei.cmu.edu

Program Co-Chairs
Dave Card
Computer Sciences Corporation
Systems Sciences Division
4061 Powder Mill Road
Calverton, MD 20705 USA
301.572.3815; FAX 301.572.7502
email: dc@sei.cmu.edu

Mari Georges
CAP GEMINI Innovation
118, rue de Tocqueville
75017 Paris, France
33.14054.6675; FAX 33.14267.4139
email: mari@crp.capsogeti.fr

Tutorials Chair
Shawn Bohner
The MITRE Corporation
611 West Popular Road
Sterling, VA 20164-4733 USA
703.883.7354; FAX 703.883.6991
email: bohner@mitre.org

Tools Chair
Nazim Madhavji
McGill University
School of Computer Science
3480 University Street
Montreal, Quebec H3A 2A2, Canada
514.398.3740; FAX 514.398.3883
email: csm@process.cs.mcgill.ca

onsored by: IEEE Computer Society The Institute of Electrical and Electronics Engineers, Inc.

SEA Seminar & Forum (March 1993)
参加者募集

バブル崩壊の煽りを受けて，いま，ソフトウェア業界のあちこちで「リストラ論議」が盛んですが，本来，確固たる技術を基礎としてビジネスを展開するのがあるべき姿だと思われます．その意味からすれば，営業形態や組織の見直し以前に，ソフトウェア開発技法や支援環境のリストラクチャリングこそが，まず第一に行われるべきでしょう．ダウンサイジングの進展やCASEブームなどの影響を受けて，最近のソフトウェア開発パラダイムは目に見えて大きく変わりつつあります．

3月のSEA Seminar & Forumでは，そうした技術動向変化のキャッチアップを心がけておられる中堅技術者および管理者の方々を対象に，実践的オブジェクト指向設計技法の解説と，フォーマル·アプローチの実用化に関するパネル討論を組み合わせた形のプログラムを組んでみました．多数の方々の参加をお待ちします．

開催および申込要領

1. 日時：　1993年3月15日(月)　14:00 ～ 20:30

2. 場所：　機械振興会館 6F 67号室　(東京·港区芝公園·東京タワー前)

3. プログラム：　Seminarは14:00 ～ 17:30，Forumは18:30 ～ 20:30.

Seminar:「中堅技術者のためのオブジェクト指向設計技法入門ー OMTを中心に」

講師：羽生田栄一(富士ゼロックス情報システム)

90年代におけるソフトウェア開発方法論の主流がオブジェクト指向であることには，誰も異論をさしはさむ人はいないでしょう．しかし，現場への導入と活用は，他のすべての新技術の場合と同じく，まだしばらく時間がかかりそうです．このセミナーでは，話題の名著「オブジェクト指向方法論 OMT: モデル化と設計」の監訳者を講師にお迎えし，この新しい設計方法論の概要と実践にあたっての留意点を解説していただきます．

Forum:「フォーマル·アプローチ：　ドライか，ウェットか？」

コーディネータ：山崎利治(日本ユニシス)

パネリスト(予定)：　佐原伸(SRA)　玉井哲雄(筑波大学)　中野秀男(大阪大学)
羽生田栄一(FXIS)　二木厚吉(電子技術総合研究所)　他

製品としてのソフトウェアが，最後はコンピュータ言語を用いて形式的に記述されるものである以上，その仕様化や設計に始まる開発プロセスの全局面において，フォーマルなやり方を採用すべきだというのは，ある意味で正論だといえましょう．しかし，理論やツールの整備と比較すると，その実践は遅々として進んでいません．このフォーラムでは，産学さまざまな立場のパネリストに，フォーマル·アプローチの現状と今後について討論していただきます．

4. 参加費：

Session	SEA 正会員	SEA 賛助会員	一般
Seminar (午後)	10,000円	15,000円	20,000円
Forum (夜)	2,000円	4,000円	6,000円

5. 申込み方法：　下の申込用紙に必要事項を御記入の上，SEA事務局までFax，郵便またはE-Mailでお申込みください．折り返し受講票をお送りします．SeminarまたはForumのみの受講も可能ですし，もちろん両方続けての参加も歓迎します．なお，参加費は当日会場受付にてお支払いください(領収書を差し上げます)．申込受付後のキャンセルは原則としてお断りします．

申込み宛先：　〒160 東京都新宿区四谷3-12 丸正ビル5F　ソフトウェア技術者協会(SEA)
Tel: 03-3356-1077　Fax: 03-3356-1072　E-Mail: sea@sea.or.jp

SEA Seminar & Forum (Mar. 1993) 参加申込票

氏名(ふりがな)：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿(　　)

会社名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

部門·役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住所：(〒　　)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

Tel:＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿　Fax:＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

種別(該当欄にチェック)：☐ 正会員(No.　　)　☐ 賛助会員(No.　　)　☐ 一般

参加Session(該当欄にチェック)：☐ Seminar　☐ Forum　**参加費：**＿＿＿＿＿＿円

*******　裏面にCSM93のCall for Paperがあります　*******

SEA これからのイベント開催予定　1993年2～6月

2月12日(金): SEA Special Forum in Shanghai @花園飯店(中国·上海)

今年の国際CASE Symposium (in 山東省)の企画打合せを兼ねた日中技術交流Forum.

2月16日(火): SEA Seminar & Forum (February) @機械振興会館(東京)

2月18日(木): TISA Seminar @宮城第一ホテル(仙台)

主催：東北地域情報サービス産業懇談会(TISA)　協賛：SEA

プログラム(予定)

ダウンサイジング時代のソフトウェア技術者に求められるもの：　堀江進（NES）

エンドユーザにおける分散型アプリケーション開発：　南仁秀(住商)

Unixを用いた社内情報システムの分散化：　岸田孝一（SRA）

3月15日(月): SEA Seminar & Forum (March) @機械振興会館(東京)

Seminar: "オブジェクト指向設計技法入門－OMTを中心に"　羽生田栄一(FXIS)

Forum: "フォーマルアプローチ：ドライか，ウェットか?"　司会：山崎利治(日本ユニシス)

パネリスト：佐原伸(SRA), 玉井哲雄(筑波大学), 中野秀男(大阪大学), 羽生田栄一(FXIS), 二木厚吉(電総研), 他.

4月9日(月): SEA Seminar & Forum (April) @機械振興会館(東京)

インターネットとこれからのビジネス（現在企画中）

4月9日(月): SEA Seminar & Forum (April) @機械振興会館(東京)企画中

4月21日(水): SEA 全国縦断 Forum in 仙台 @(会場未定)企画中

4月22日(木)～24日(土): 第5回テクニカルマネジメントワークショップ @山形(会場未定)

テーマは"業界リストラクチャリングとテクニカルマネジメント".

実行委員長は野村敏次(JIP)& 山崎朝昭(ENICOM). 詳細は現在企画中

5月28日(金): SEA Seminar & Forum (May) および年次総会 @機械振興会館(東京)

6月9日(水)～11日(金): Software Symposium '93 @戦災復興記念館(仙台)論文募集中

.....

申し込み先

〒160　東京都新宿区四谷3-12丸正ビル5F　ソフトウェア技術者協会(TEL 03-3356-1077,　FAX 03-3356-1072)

SEA入会申込書（正会員：入会金3,000円，年会費7,000円）　93-01

氏　名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　(ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿)

生年月日：19＿＿＿年＿＿＿月＿＿＿日　　性別（男　女）血液型（A　O　B　AB）

勤務先名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

所属·役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

勤務先住所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

勤務先TEL：＿＿＿＿-＿＿＿＿-＿＿＿＿（内線＿＿＿＿）

勤務先FAX：＿＿＿＿-＿＿＿＿-＿＿＿＿

自宅住所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

自宅TEL：＿＿＿＿-＿＿＿＿-＿＿＿＿

連絡先（どちらかにチェック）□勤務先　□自宅

.....

SEA入会申込書（賛助会員：年会費1口100,000円，何口でも可）　93-01

会社·団体名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

代表者氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　(ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿)

連絡担当者：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　(ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿)

所属·役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住　所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

TEL：＿＿＿＿-＿＿＿＿-＿＿＿＿（内線＿＿＿＿）　FAX：＿＿＿＿-＿＿＿＿-＿＿＿＿

申込口数：＿＿＿＿＿口

******* SEA Seminar & Forum February 1993の案内があります *******