SEAMAIL

Newsletter from Software Engineers Association

Volume 7, Number 4-5-6-7

目　次

編集部から		1
第10回夏のプログラミング･ワークショップ報告書		3
はじめに	渡邊雄一	4
1. 全体プログラム		5
プログラムについて	渡邊雄一	7
プログラムの構成にいたるまで	渡邊雄一	10
若手 '92 のプログラム委員長を努めて	塩谷和範	13
2. 手法の説明		17
SA/SD について	伊藤昌夫	19
SSADM について	岡本隆一	23
JSD に関すること	渡邊雄一	27
CRC カードによるオブジェクト指向設計	福良博史	30
3. グループ活動報告書/担当 PC 感想		41
グループ 1 (SSADM)		43
グループ 2 (SA/SD)		63
グループ 3 (CRC)		85
グループ 4 (CRC)		113
グループ 5 (JSD)		139
4. 講演記録		153
オブジェクト指向分析/設計概観	塩谷和範	155
形式的ソフトウェア開発	山崎利治/記録: 伊藤昌夫	170
付録		177
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
論文和訳　オブジェクト指向設計：責任駆動アプローチ		241
読書案内		247
グループ活動のワークシート例 (G2 模造紙)		249
ワークショップ開催案内		255
岡崎マップ		256
編集後記に代えて　－最近の SIGENV の活動について－	渡邊雄一	257
Call for Papers/Participatrions		
Symposium on the Foundation of Software Engineering		2
Conference on Software Maintenance '93		259
SEA Seminar & Forum (March '93)		260

ソフトウェア技術者協会
Software Engineers Asociation

ソフトウェア技術者協会(SEA)は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985年12月に設立されました．

その主な活動は，機関誌SEAMAILの発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約200人にすぎなかった会員数もその後飛躍的に増加し，現在，北は北海道から南は沖縄まで，1000余名を越えるメンバーを擁するにいたりました．法人賛助会員も約60社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます．SEAは，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

代表幹事： 熊谷章

常任幹事： 筏井美枝子　岸田孝一　玉井哲雄　中野秀男　深瀬弘恭　堀江進　山崎利治

幹事： 飯沢恒　市川寛　今別府芳暢　臼井義美　大塚理恵　落水浩一郎　片山卓也　亀田繁　君島浩　窪田芳夫　小林俊明　杉田義明　坂本啓司　田中一夫　鳥居宏次　中来田秀樹　中谷多哉子　野村敏次　野村行憲　平尾一浩　藤野晃延　松原友夫　盛田政敏　山崎朝昭　渡邊雄一

会計監事： 辻淳二　吉村成弘

分科会世話人 環境分科会(SIGENV)：田中慎一郎　渡邊雄一
管理分科会(SIGMAN)：野々下幸治
教育分科会(SIGEDU)：杉田義明　中園順三
ネットワーク分科会(SIGNET)：小林俊明
調査分科会(SIGSURVEY)：岸田孝一　野村敏次

支部世話人 関西支部：臼井義美　中野秀男　盛田政敏
横浜支部：藤野晃延　北條正顕　野中哲　松下和隆
長野支部：市川寛　佐藤千明
名古屋支部：筏井美枝子　岩田康　鈴木智　平田淳史
九州支部：平尾一浩　藤本良子

賛助会員会社： ＮＴＴソフトウェア研究所　ＮＴＴ九州技術開発センタ　ＰＦＵ　ＳＲＡ　アスキー　エイ・エス・ティ　エスケーディ　オムロンソフトウェア　カシオ計算機　キャノン新川崎事業所　さくらケーシーエス　サン・ビルド印刷　システムラボムラタ　ジェーエムエーシステムズ　ジャストシステム　セントラル・コンピュータ・サービス　ソフトウェアコントロール　ダイキン工業　テクノバ　ニコンシステム　ニッセイコンピュータ　ヒラタ・ソフトウェア・テクノロジー　ムラタシステム　リコーシステム開発　リバティーシステム　安川電機　古河インフォメーション・テクノロジー　構造計画研究所　三菱電機セミコンダクタソフトウェア　三菱電機メカトロニクスソフトウェア　三菱電機関西コンピュータシステム　新日鉄情報通信システム　新日本製鉄エレクトロニクス研究所　池上通信機　中央システム　辻システム計画事務所　東芝アドバンストシステム　東電ソフトウェア　東北コンピュータ・サービス　ＳＲＡ東北　日本ＮＣＤ　日本データスキル　日本ユニシス・ソフトウェア　日本情報システムサービス　日本電気ソフトウェア　日立エンジニアリング　富士ゼロックス情報システム　富士写真フィルム　富士通　富士通エフ・アイ・ピー　富士通ビー・エス・シー　オムロン　（以上53社）

ＳＥＡＭＡＩＬ　Ｖｏｌ．７，Ｎｏ．４－５－６－７　１９９３年２月２５日発行
編集人　岸田孝一
発行人　ソフトウェア技術者協会（ＳＥＡ）
〒１６０　東京都新宿区四谷３－１２　丸正ビル５Ｆ
TEL: 03-3356-1077　FAX: 03-3356-1072
印刷所　サンビルド印刷株式会社　〒１６２　東京都新宿区築地町８番地
定価　２０００円（禁無断転載）

編集部から

☆

異常気象につられて，なぜかまた出た Seamail です (!?).

☆☆

この号は，去年の 9 月に愛知県岡崎市で行なわれた第 10 回夏のプログラミング･ワークショップ (通称： SEA 若手の会) の報告特集です.

☆☆☆

こんどの若手の会は，環境分科会 (SIGENV) のみなさんが企画と運営を担当し，報告書のとりまとめもやってくださいました．当初は年内（つまり 1992 年中）に報告書をだすという意気込みで周囲の野次馬連はなかばマユツバで見守っていたのですが，若干遅れたとはいえ，イベント終了後半年でレポートができあがったというのは，SEA 始まって以来の快挙だと思います．関係者のみなさん，どうもおつかれさまでした.

☆☆☆☆

報告の内容については，また会員各位から読後感を逐次お寄せいただくとして，形式上では，ワークショップ風景の写真を挿入して臨場感をだすという，これもまた初めての試みをトライしています．いかがでしょうか？

☆☆☆☆☆

ちょっと順序が逆になりましたが，次号は，ひきつづいて，一昨年の若手の会のレポートをお送りする予定です.

☆☆☆☆☆☆

そこまでで，編集部としては，手持ちの材料がそろそろ終りになります．会員のみなさん，ぜひとも原稿をお寄せください.

☆☆☆☆☆☆☆

CALL FOR PAPERS

ACM SIGSOFT '93: Symposium on the *Foundations of Software Engineering*

Sponsored by ACM SIGSOFT

Los Angeles, California USA
7-10 December 1993

The ACM SIGSOFT '93 Symposium on the Foundations of Software Engineering will focus on innovative research results that identify and contribute to the foundations of software engineering. The intent is to help establish software engineering as a viable engineering discipline.

We solicit papers in all technical areas of software engineering. A successful paper is expected to report on new principles, methods, or results of experimentation in software engineering (which includes topics related to the specification, design, implementation, and evaluation of software systems). Papers should emphasize how they contribute to a foundation that allows us to effectively engineer classes of complex software systems in disciplined, reasoned ways. Unless a strong tie to software engineering is made, papers more central to other aspects of computer science should be submitted to conferences in those areas.

A paper should clearly state the contribution and its underlying assumptions. It should also assess the results, making appropriate comparisons with and references to the literature. Papers will be judged on clarity, significance, relevance, correctness, and originality. The paper must contain ideas not previously presented in or currently awaiting acceptance to another formal forum.

All papers will be reviewed by program committee members. In some cases, additional external advice may be solicited by the program committee. Papers of particular merit may be recommended to major software engineering journals for expedited review.

Submissions are limited to 6000 words, including reasonable estimates of the size of figures but excluding references. Overly long submissions will be returned without review. Fourteen copies, preferably double-sided, must be sent to the program chair by **April 9, 1993.** Authors will be notified by **July 5, 1993.** Camera-ready versions of accepted papers are due, along with ACM copyright release forms, by **August 27, 1993.** Proceedings will be distributed at the symposium and as a special issue of *ACM Software Engineering Notes.* Tutorials will be held on December 7, 1993.

General Chair
Barry Boehm
Computer Sciences Department
University of Southern California
Los Angeles CA 90089 USA
(213) 740-8163
boehm@cs.usc.edu

Program Chair
David Notkin
Dept. of Computer Science & Engineering, FR-35 (Sieg 114)
University of Washington
Seattle WA 98195 USA
(206) 685-3798
notkin@cs.washington.edu

Tutorial Chair
Dewayne E. Perry
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974 USA
(908) 582-2529
dep@research.att.com

Program Committee

Lori Clarke, *University of Massachusetts*
John Gannon, *University of Maryland*
David Garlan, *Carnegie Mellon University*
Susan Gerhart, *National Science Foundation*
Ross Jeffery, *University of New South Wales*
Nancy Leveson, *UC Irvine/University of Washington*
David Notkin, *University of Washington*
Harold Ossher, *IBM T.J. Watson Research*
Tom Reps, *University of Wisconsin*
Bill Scherlis, *DARPA SISTO*
Tetsuo Tamai, *University of Tsukuba*
David Wile, *USC/Information Sciences Institute*
Michal Young, *Purdue University*

第10回

SEA夏のプログラミングワープショップ

– 現場でのシステム設計を改革するテクノロジを求めて –

報告書

1992年9月2日(水) ～ 9月5日(土)

於：愛知県 岡崎市

ソフトウェア技術者協会

はじめに

実行委員長 渡邉 雄一

SEA のイベントとしては, ソフトウェアシンポジウムと並んで歴史的なこのワークショップも回を重ねること, 今回で 10 回目と相成った。盛岡で開催されることがほぼ常であったこの会合も, 途中 1 度だけ趣向をかえて八ヶ岳山麓の辺境の地で開催された (第 6 回)。今回もその意味では慣例を踏み外して, 尾張名古屋に近い 徳川家康生誕の地, 岡崎での開催となった。『なぜ, この地を選んだのか?』という質問に関しては, さほど深い意味があるわけではない。が, ここでは苦し紛れに, 岡崎も盛岡と同じく 城下町だし きれいな川はある, とだけ回答しておこう。

さて, 今回の開催までの経緯は, あらためて次章で紹介することにして, ここでは本報告書の構成を簡単に紹介することで, 報告書のあいさつに代えさせて頂きたい。

本編は, 4 つの部分からなる。まず 1. **全体プログラム** では, 実施されたプログラムについて述べる。最初にプログラムの全貌を示し, 続いてプログラムが編纂されるまでの経緯も述べる。プログラムの細部まで PC の意図があったことを御理解頂きたい。また, プログラム委員長の塩谷さんが総括をまとめてくれた。

続く 2. **手法の説明** では, 今回取り上げることとなった手法について, 1 日目のプレゼンテーションをベースに新たに書き下ろしてもらうことをお願いした。が, 皆忙しい身であり, 必ずしも それを忠実に再現したものにはなっておらず, また手法の解説としても十分とは言えないのは心残りである。これだけの情報で 各々の手法/方法論がわかるものではないので, 挙げられている文献を手がかりに それぞれの原典に当られて, その極意を究められたい。

3. **グループ報告** が, 今回の参加者の手による本報告書の中心部分である。1~5 グループの順に, その報告書と ともに 各担当 PC の感想を織り混ぜた。

どのグループのアプローチにも多くの興味深い試行/考察がなされており, 実践的な設計を考える際の貴重な資料となるものと考える。

また, 今回の報告書のために無理をお願いして, 読みやすさの形式にも拘っている。しかし, そのことが, 従来の報告書に見られる自由奔放な展開を損なうことになっているとすれば, それは PC 側の責任と言わざるを得ないだろう。読者諸氏の読後の感想を是非とも SEA 事務局まで寄せられたい。

最後の 4. **講演記録** は, 期間中に行なわれた 2 つの講演を収めた。まず 1 つは, OOA, OOD 関連の概要と現状の動向について, プログラム委員長の塩谷さんが, その講演内容を自らがまとめてくれた。現時点での, 極めて有益な指針となるものと考える。

本編の最後には, 最終日に 山崎さんによる '設計におけるフォーマルなアプローチ' についての講演の全容を収録した。この講演は, 最終日に急挙 御無理をお願いして, 氏に受けていただいたもので, 実質 3 日間ではあったが, 個々の手法/方法論に没頭しただけで 少なからぬインパクトを受けた者にとって, さらなる展望を与えてくれた。

この後は **付録** として関連する資料を集めた。参加者から提出されたポジションペーパや最終日に回答してもらったアンケートなどもこちらに収めてある。

最後に, 末筆ながら 今回 特にお世話になった方の名前を上げさせて戴き, 感謝の意としたい。

まず, お名前をあげなければならないのは 山崎利治さん (日本ユニシス) である。氏は, 情報処理学会の『酒倉庫問題による共通設計問題』の発案者であられ, またそのとりまとめをされたことは, この分野に関心のある諸氏にあってはいまさら説明の必要はなかろう。今回は, 特に JSD グループを実質的に全部見て頂くことになり, さらに上にも述べたように最終日には講演まで賜わったことは, 感激の極みである。

九州支部の平尾一浩さん (HST) からは, 運営に関する貴重な意見を頂きプログラムに反映することが出来た。名古屋支部からは 筏井美恵子さん (SRA), 平田淳史さん (SRA) には, PC の 1 人として 伊藤さん (MASC) の推薦を頂くだけでなく, さまざまな形で援助の手をさしのべて頂いた。

代表幹事の熊谷章さん (PFU) には NIFTY へのニュースのポストのトラブルなどにも適切なアドバイスを頂いた。

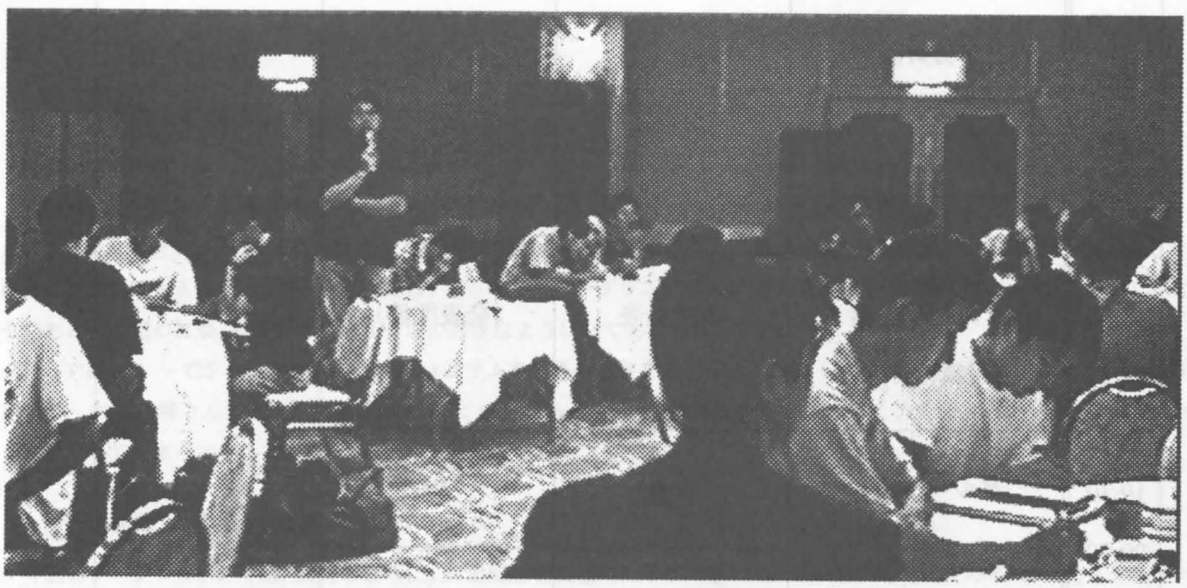
また, ソフトウェア業界に限らず厳しい社会状況を心配されて, 常任幹事の岸田孝一さん (SRA) からは, 極めて早い段階から色々と有益な助言を頂いた。

その他, ここには書ききれないが本当に多くの皆さんからさまざまな援助を頂き無事ワークショップを終了することが出来た。ここに遅ればせながら報告書の完成を報告し感謝の気持ちとさせて頂きたい。

株) アスキー OA 推進部

1. 全体プログラム

- 大まかな予定表 6
- プログラムついて 7
- プログラムの構成にいたるまで (渡邉 雄一) 10
- 若手'92 プログラム委員長を務めて。。。 (塩谷 和範) 13



大まかな予定表

	9/2	9/3	9/4	9/5
9:00		朝食	朝食	朝食
9:30		グループ別 設計方法論 レクチャ	設計見直し	最終発表
10:00				
10:30				
11:00		（共通課題説明）		
11:30				全体討論
12:00				
12:30	受付	グループ別	グループ別	（昼食付き）
13:00		自由行動	自由行動	
13:30	オリエンテーション			
14:00	講演 （ＯＯＡ）			
14:30				
15:00	ＰＣパネル	（設計作業）	（設計作業）	（解散）
15:30	（ＰＣ方針説明）			
16:00				
16:30				
17:00	休憩	発表準備		
17:30				
18:00	グループ討論	中間発表	（発表準備）	
18:30	（顔合せ）			
19:00				
19:30	グループ紹介 ＆ パーティ	夕食	夕食	
20:00				
20:30		自由行動	自由行動	
21:00				
	自由行動			

プログラムについて

1 プログラム委員の構成

今回のプログラム委員は, 東京で活動している 環境分科会 (SIGENV) のメンバが中心になって組織された。プログラム委員の構成は, 表.1の通りである。
SIGENV のメンバのみでは偏りがあるため, 開催地の岡崎に近い 名古屋支部から伊藤さんに, また関西は神戸から 今までにも SIGENV の活動に加わってもらって気心の知れている 岡本さんに, さらに岡本さんの紹介で, 豊田さんに加わって頂いた。また, 今回のプログラムの計画/準備状況を SEA 幹事会に報告した際に, 山崎さんには 是非ともお願いしますと 内々に御協力を打診しておいたところ, 早々に 参加頂ける旨の連絡を頂き, 委員会として 特別ゲスト として参加して頂いた。

表1 プログラム委員の構成

実行委員長	渡邉 雄一	アスキー	G5	JSD
プログラム委員長	塩谷 和範	エスアールエー (SRA)		
プログラム委員	岡本 隆一	さくらケーシーエス	G1	SSADM
	伊藤 昌夫	MHI エアロスペースシステムズ (MASC)	G2	SA/SD
	田中 慎一郎	エスアールエー (SRA)	G2	SA/SD
	桜井 修	池上通信機	G3	CRC
	豊田 英利	セス	G3	CRC
	福良 博史	ジェーエムエーシステムズ (JMAS)	G4	CRC
	後藤 浩	シスプラン	G4	CRC
特別ゲスト	山崎 利治	日本ユニシス	G5	JSD

2 プログラムの概要

今回のテーマは,

現場でのシステム設計を改革するテクノロジーを求めて – 設計における問題点とは何か? –

ということを掲げて, 4 つの設計方法論を グループ毎に実際に使って設計をしたうえで, 討論をするということを目標とした。討論を有意義にするために, 設計する問題は, あらかじめプログラム委員会の方で用意した 共通問題 を最低限 解くこととし, そのあとの問題の拡張/変形などはグループの好きなようにしてもらった。
グループは G1 ～ G5 の 5 つ形成した。プログラム委員長の塩谷さん (SRA) には, G3,G4 の補佐役として, フリーに動いて頂いた。また 山崎さんにも, フリーに動いて頂くはずであったが, 渡邉の不手際で その多くの時間を G5 グループに割いて頂く形となってしまった。

3 共通問題 – 図書館問題 –

今回のワークショップでは, 前述のように, 共通問題を異なった手法で解いて, その違いを考えてみることが, 1 つのテーマであった。
そのための問題としては, 以下の図書館問題を使用した。

図書館問題

以下のトランザクションが発生する小さな図書館データベースを考えよ。

(1) 本の貸出, および返却

(2) 図書館への本の登録, および削除

(3) 特定の作者の本, あるいは特定の分野の本に関するリストの作成

(4) 特定の利用者が借りている本のリストの作成

(5) 特定の本を最後に借りた人の検索

このシステムのユーザは, 図書館の職員と図書館の利用者である。(1),(2),(3),(4) および (5) のトランザクションを発生できるのは 図書館の職員に限られている。ただし, (4) については, 自分自身で現在借りている本の検索だけは, 図書館の利用者でも行なえるようになっている。データベースは次の条件を満たさなければならない。

- 図書館のすべての本は, 貸出可能か, すでに貸し出されている
- ある本が貸出可能であり, 同時に貸し出されている, ということはない
- 図書館の利用者は, 1 回にある決まった数以上の本を借り出すことはできない

4 スケジュール

プログラムのタイムテーブルは, 大まかな予定表 (p.6) に示す通りで今までの若手の会のそれと比べて, プログラムのタイムテーブルの上での変更はさほどない。

なお, 以下は 大まかな予定表 と同時に メンバに 事前に配布する資料のために 委員の 1 人の田中さんが 苦労して作成してくれたものを, そのまま転載する。

プログラム内容説明

1. 講演は, 「オブジェクト指向分析/設計 塩谷和範 (SRA)」を行ないます.
2. PC パネルは,「グループ世話役の方針発表」を行ないます.
   メンバはこれに対して参加を希望するグループを第 3 希望まで紙に書いて提出して下さい. 休憩時間に調整を致します. 基本的に同じ会社の方は同じグループにならないとお考え下さい.
3. 初日のグループ討論は, 2 日目以降の討論をスムースに行なうために不可欠な「グループ内の顔合せ」を中心に行ないます. テーマは「活動方針」です.
   ここでまずはグループ内の親睦を, 次のパーティで参加者全体との親睦を深めることがワークショップ成功の鍵となっています. 以降は基本的にグループ別の活動をすることになりますので, パーティの席で「自分たちのグループをいかに印象づけるか」をサブ・テーマに意見交換して下さい.
4. パーティでは, グループの活動方針と, メンバ紹介を行ないます.
   短い時間で他のグループの人に自分のグループを印象づけるよう, 工夫をして下さい.
5. それ以降, 夜は自由. ただし, 世話役の部屋で自由参加の討論会 (呑み会ともいう) が開催されます. 分からないことや不安なことがあれば, この時に PC や他のメンバに相談するといいでしょう.
6. 2 日目の午前中は, グループ毎に「設計方法論に関するレクチャ」を受けます. グループ別に部屋を用意します.
7. 午後はグループ別の自由行動です. 世話役のもと, あるいは地元出身者のアドバイスをもとに屋外の適当な場所で開放的な討論を行なって下さい. どのような場所を見つけたかも重要な「情報」です. なお, 晴天であれば屋外でミーティングを行なうことが予想されるため, これに適した筆記用具 (個人用のメモその他用) を用意してください. なお, 発表に使用する OHP シートその他は PC 側で用意します.
   またこの時点で, 最終発表/報告の分担を決めておくと, 後々の作業が楽になります. 例えば, 報告書関係で言えば,
   - グループ紹介執筆担当
   - 分析過程執筆担当
   - 設計過程執筆担当

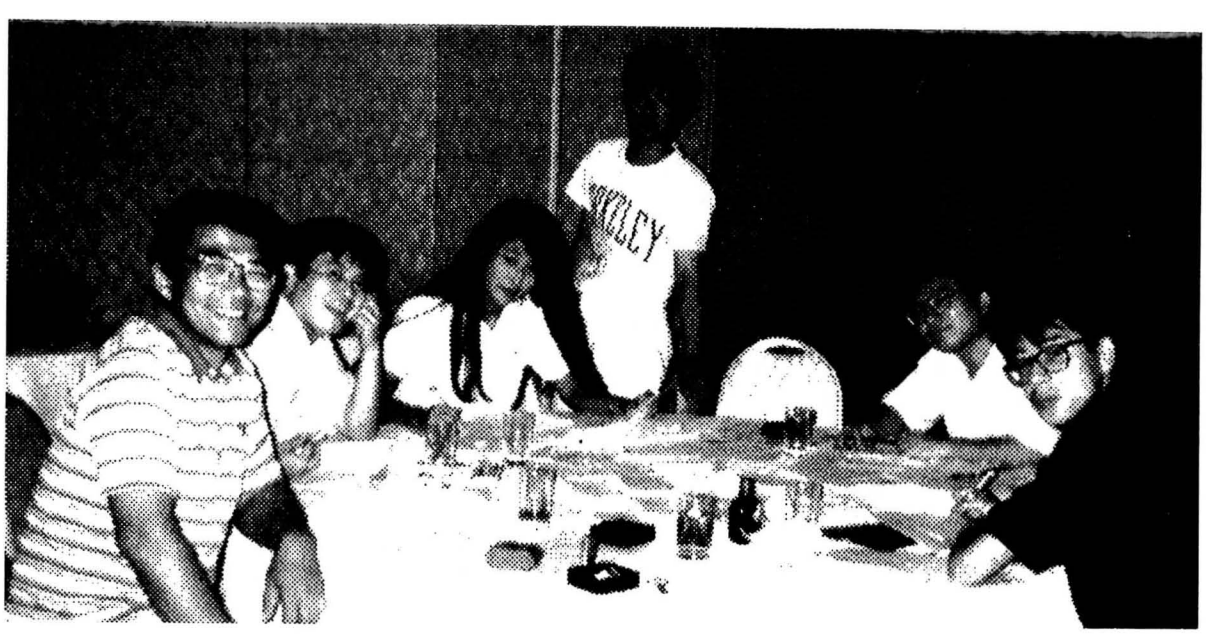
- 設計書作成担当
- 評価結果執筆担当

また, 最終発表で言えば,

- 発表担当
- 発表用 OHP 作成担当
- 最終報告書編集担当

などです.

8. 中間報告では, それまでにどのような活動をそれぞれのグループが行なったかの情報を交換します.
   各グループから 1 名ずつ集まった別グループに別れ, これまでの進捗を報告し合います. またそこでの質疑はグループに帰ってから報告することになります.
   全員が必ず報告することになりますので, 17:00 までにホテルに戻り, OHP がわりの A4 用紙に 1〜2 枚で 10 分程度の発表準備を各自行なって下さい.
   内容的には, 「チーム構成」, 「スケジュール」, 「進捗」, 「発生した問題点」などいろいろあると思いますが, 「チームの活動方針」に沿って「強調しておきたいことを端的に」発表して下さい.
   なお, この発表の結果は PC がまとめる予定です.
9. 夕食時には各グループの行動報告をして, 翌日の行動のための情報交換をしたいと思います. 夜は前日と同じです.
10. 3 日目の午前中は, 昨日の中間報告の結果などを参考に, 設計の見直し, あるいは新たなテーマへの取り組みを行ないます. 午前中のみグループ別に部屋を用意します. 使用は自由です.
    当日は終日自由行動ですが, 遅くとも昼までにに最終発表時の役割分担を明確にしておく必要があるでしょう.
11. 夜は前日と同じ.
    ただし, 昼間作業がすんでいなければ, 最終発表用の作業および報告書用の作業が必要となります. 最終報告は, 事前に決めたフォームに従って最終日の朝までに「発表用 OHP」とは別に全員で分担して草稿を作成／配布します. 概ね, 「活動報告」と「メンバ感想」などからなります. 最終報告担当者はワークショップ終了後, 全体の編集を行なった上, スタッフ宛に電子媒体で 9 月 14 日 (月) 必着で送付してください.
12. 最終発表では, いかに要点を押えた発表をするかが求められます. 報告書にはいろいろなことを盛り込めますが, 時間の制約のある発表ではあまり多くのことを盛り込むと結局何をいっているのか分からなくなりかねません. 発表資料作成の時点から分担を明確にするなど, いろいろ工夫して下さい.
13. 全体討論では, 他のグループの意見などを参考にしながら PC を交えて意見交換をします.
    昼食をとりながら続けますが, 14 時には解散します.



プログラムの構成にいたるまで

渡邉 雄一

以下に, 今回のプログラムが構成されるまでの舞台裏を披露したい。今回の WS を担当した PC それぞれに, ある種の拘り (こだわり) をもって, この開催に望んだ。これは, まぎれも無い事実である。その拘りは当然のことながら, 可能な限りプログラムに織り込まれた。ここでは, それら主だったものを具体的に紹介したい†1。次回以降の開催の参考になれば幸いである。

1 ことの始まり

今年 (1992) のワークショップ (以下 WS と略す) は, 昨年 (1991) の PC の 1 人であった塩谷さん (SRA) に, 開催依頼があったことによる。昨年の WS に初めて参加した彼は, 定期的に開催されている環境分科会 (SIGENV) のメンバでもあり, その開催を一緒にやらないかとの提案があり, 分科会としても積極的に応援してゆこうということになった。

まず, 開催のテーマ/条件であるが, 彼と私が SIGENV の帰りの電車の中で雑談していて, 出されたキーワードは,

1. 参加者自身によるケーススタディとディスカッション
2. PC 側は (SIGENV) の活動の延長としてとらえる
3. 設計方法論の周辺をあつかってみたい

の 3 点だったと記憶している。

1. は, 特に前回の WS で, 参加者が受身になっているような状況が少なからずあったようで, それに対して PC 側にも不満があったことが報告されていたからである。また 2. については, PC として SIGENV のメンバを中心構成することを考えるのなら当然の成行きだからである。3. は, 単に私の趣味を上げたに過ぎなかった。

以上のようなことを叩き台にして, SIGENV なかでプログラムが着々と練られていった。

2 何をテーマに選ぶか

それまで SIGENV では オブジェクト指向 について, いくつかの文献を一緒に読むという活動をしてきていた。最初のとっかかりは, Meyer の本[4] であり, その次は Kim[2] の本だった。今回のワークショップへの開催協力を SIGENV のメンバに取り付けたのが, 昨年の暮れであったと思う。それも考慮したうえで, 『では, 次は何をやろうか?』というときに出てきた本が 2 つあった。

1 つは今回とりあげた CRC(Class, Responsibility, Collaborator) カード を紹介している Budd の本[3] であり, もう 1 つは オブジェクト指向言語を広範かつ的確に述べている Masini らの本[1] だった。最初は, Masini の本をやってから Budd をやろうかと言っていた。が, (結果的には大変に良かったとも言えるが) 塩谷さんの策略にかかってしまって Budd の本を読むことになった。

さて, WS のテーマとして, この本で取り上げられている CRC カードを使った分析/設計を実践するだけでは, 面白くない。具体的に指摘をするなら, オブジェクト指向設計は, まだ確定したものがあるわけではなくて, 具体的な方法がやっと幾つかあげられてきたに過ぎない。その意味で, 従来型の手法を軽視するのは明らかに不適当であるという考えからである。そこで SA/SD などのその他の手法を取り込んで総合的に議論するのがよかろうということになり, 最終的に PC として名を連ねている面々の協力を得ることが出来, SA/SD から 2 種類と JSD を含む, 計 4 種類の手法が並ぶことになった。

3 どのような特徴を出すか

事前の PC ミーティングで合意された主なことを書き留めておこう。

1. 講習会/セミナの色合いとならないようワークショップとして議論を中心に出来るようにする
2. ポジションペーパをきちんと審査する
3. 1 グループ 5 人を原則とする
4. 最大 5 グループとする
5. WS の成功/失敗の分岐点は, どれだけ早くメンバが自然にうちとけられるのかにかかっているので, そのための仕掛けを工夫する
6. 従来の開催で中途半端だったり, 問題だったと思われるプログラム内容については, 幾つかの試行をしてみる

具体的には, 中間発表/最終の形態や報告書作成までのプロセス

このうち 3. については, 後で述べる作業の分担や, 実際に議論に参加できる人数からして適当な人数ということで

株) アスキー OA 推進部

†1 ここに記載されているプログラムに盛り込まれたもの/アイディアは, 委員会の場で出されたものであって, 筆者の固有のものでは決してない。委員各位に深謝する。しかし, この文章に不適当な箇所があるとすれば, それはすべて筆者個人の責任である。

あらかじめ設定した。

4. については, 3. の条件と PC の人数/力量との兼ね合いから, これが適当と考えた。

また, グループ分けに関しては, 最初のオリエンテーションを兼ねた PC の紹介の後で, 個々人の希望を尊重して, 決めることとした。

この他に, PC ミーティングで出された意見として,

- WS そのものの位置付けをどうするのか?
- 方法論に縛られるのは適当とはいえないのでは?

という問いかけが (事情で後から PC に加わって頂いた) 伊藤さんからなされた。これについては,

1) WS の目的が, 方法論の習得にあるのはない。共通の体験を通して, 議論をすることで, 参加者なりに得られるものがあれば, それで十分といえる
   → 特に伊藤さんは, '小さな発見' を大事にする ということを掲げてグループを引っ張っていってくれた
2) そのための共通の土俵として 個々にグループ毎に課せられた手法で問題を解くことを前半の主眼とし, 後半の活動は議論が中心になることが意図される

のような合意をみた。

4 ポジションペーパの位置付け

ここ数年, このワークショップの応募段階で, ポジションペーパを軽んじている風潮があったように思う。もちろんそれは, それぞれの開催年の PC の意向なのでその善し悪しをここで議論しようとは思わない。が, 今年のテーマを考える限り, ある程度のものを書ける人に集まってもらわないと, 結果どころか過程すらきちんと記録出来ない危惧があったので, 審査基準をあらかじめ設けて, ある一定水準以上の人を採用することとした。ポジションペーパの内容については, 基本的には自由に何を書いてもらっても構わないが, 最低限設計ということについて何を考えるか, そしてそれを自分の言葉できちんと表現出来るかを見ることが出来るようなテーマを考えた。そのための課題として

1. 今までに経験した開発作業で, 設計に感じた「あらがたみ」について
2. 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

のいずれかについてという設定をした。

このなかの「うらめしさ」という言葉について『あれはどういう意味ですか?』と, 何人かの方から (WS 開催中に) 質問された。この趣旨は,

> 今まで経験してきた作業で, たとえ自分が設計という作業に直接関わることがなかったにしても, 例えば 1 人のプログラマとして, 設計されたものをどう評価するのか?。そしてそれが結果的にプログラマの目から見て, 明らかに問題と思われるにも関わらず, それに制約されるなどの「うらめしさ」を感じるようなことがあったのならば, その問題点について自分なりの分析/提案を期待した

のだった。

5 課題の設定に関して

共通課題を前半で解き, 後半はそれに基づいて個々のグループに任せた, 発展させた議論を考えていた。そのための共通問題として,

1) 図書館問題
2) 銀行の現金引落し機 (ATM)
3) トランプゲームの 21

が候補に上がった。2) は, 適当な例題が CRC で出されているために, むしろそれを WS の際に紹介したいとの意見が出たためにはずした。また 3) は, ゲームのルールから始めると結構大変で, またゲームという特殊なもので, 設計をみんなで考えるには不適当ではとの判断により退けられた。

これに対し,

i) 第 8 回の若手の会 (CASE を扱った) の, 問題に上がりながら, 実際には回答例が 1 つもなかった
ii) 誰でもが 1 度は接している図書館というものを問題の中心に据えている
iii) 問題の大きさとして十分に小さく, 逆に変形/拡張も容易だろう

などの理由から 1) が選ばれた。

6 グループの形成とその立ちあげ

グループ分けは, どのようにすべきか? を議論したが, 個々人の意志にまかせるのが自然だろうということになった。そのタイミングは, 1 日めの各 PC のプレゼンテーションの後に, 希望を聞いてグループ分けをすることにした。但し, 今回のテーマの進め方からグループの人数が極端にちがうのは不適等なので, 第 3 希望までを出してもらいそれに沿って PC の方でグループを決定することにした。

また, WS の活動 は, 兎にも角にも メンバが他のメンバの顔を出来るだけ早く覚えて 気さくに話せること から始まると考え, 事前に 東京ないし大阪などで 事前ミーティングを開催することも検討された。しかし, それには地方からの参加者が出席出来ず, 逆にその人たちにハンディを負わせることになるとの意見がありとり止めた。

それを補い グループの実質的な活動を出来るだけ早い段階

で立ちあげるべく以下のように工夫した。

- 1日目に ともかくグループ分けをして (内容はともかく) グループの活動を始める
  そのために, 1日目の夕食前にグループ分けをして, 短時間でもグループ・ミーティングの時間をとる
  → 非常な成果があった
- 1日目の夕食は, 懇親の意味も兼ねて, 通常の着席のではなく 立食のパーティにする
- パーティで 自己紹介をグループ単位で行なう
  → 少なくともグループメンバの顔を覚えるという点で効果的だった
  → 但し, 他グループとの交流時間をとる配慮が必要だったようだ

7 中間発表に関すること

従来の中間発表は, 最終発表とほぼ形態を同じくしたものを, 開催2日目の夕刻に行なうのが常だった。
そのタイミングには問題無いが, その形式は, まだ内容が不十分なのにも関わらず, 各グループの混沌とした議論の過程を発表者以外が黙って聴くという受動的な面を問題と考えた。
これに対して, 田中さんから

『これを実践的な発表の場ととらえて活用するために, 発表の場を, 一時的に編成した各グループ1名からなる場で報告し合う』

形式が提案され, それを採用することとした。これにより (このWSの開催期間中に) 最低1回は, 自ら発表する場を得ることになり, 時間も短く議論に集中できると考えた。
また, それぞれの発表をPCがメモをとり, 遅くとも翌日の朝には, そのメモを各グループに配布して, その内容がメンバに確認できるようにした。

8 最終発表に関すること

従来の最終発表は, 最終日の午前中をフルに使って, 各グループの成果を発表するというものであった。この形式の問題点は, 全グループの発表がかならずしもうまくないのにそれを聞くだけで疲れてしまうほど冗長なことが少なくないことにあると考えた。また, 今回のテーマ内容から, 個々のグループの発表を聴いた上でそれらを相互に比較検討して反芻してこと有意義であろうとの考えもあって, 極力発表の時間を抑えることを試みた。具体的には, 各グループの発表の時間を10分〜15分に制限した。
また, 討論の時間を用意して, 最終日の終了/解散時刻を通常より2時間後ろにずらした。

9 報告書の位置付け

最後にこの報告書であるが, これについてもPCの間で少なからず意見が出された。私などは, 最初は作らなくても良いという立場にあったのだが,

1) 一般に公表される (少なくともSEAの会員には配布される) このような報告書に文章を書くチャンスは, 少ないと言えるのだから, きちんとしたものを形に残してあげるようにPCが努力すべきだ
2) そのためには具体的なガイドラインが示された方が良い
3) メンバもPCも, 多大な負荷を被ること無く極めて短時間にホットな報告書が出されなければならない

という方針で作成されることが決定された。
報告書には, 単に発表に使用したOHPを掲載するようなことは避けて, あくまで報告書としての内容/体裁に見合うようなものを作成してもらうべくあらかじめフォームと呼ぶレポートの形式を規定して, それに準じる形でグループの活動を記載/報告してもらうようにした。また, それを作成するに当たっても, 極端に作業バランスが崩れたりすること, またそのことを貴重な討論の時間に考えてもらうことを出来るだけ避けるために, おおよその配分をあらかじめ規定しておいた。
その意味で, 参加者メンバ各位には「地獄の特訓」のような4日間になってしまったとしたら, 実行委員長としては真に心苦しいが, おかげでこのような立派な報告書を作成することが出来たことは感謝の念に耐えない。

参考文献

[1] 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)

[2] Won Kim(ed.) : Object Oriented Concepts, Database and Applications, Addison Wesley (ISBN 0-201-14410-7)

[3] Timothy Budd : An Introduction to Object-Oriented Programming, Addison-Wesley Publishing Company, Reading, MA,1991.

[4] Bertrand Meyer : Object-Oriented Software Construction, Prentice-Hall, 1989 (酒匂 訳:オブジェクト指向入門, アスキー)

若手'92 プログラム委員長を務めて。。。

プログラム委員長 塩谷 和範

1 はじめに

思い起こせば若手'92は, "若手'91盛岡"に私がプログラム委員として参加した後の帰京の新幹線車中で, 代表幹事の熊谷さんと"若い娘っこ"と私がビールを飲みながらの雑談中に, 来年はどうしようかという話になり, 成り行き上私がプログラム委員長を引き受けて, 「SIGENVでやりましょう。」と言ってしまったのがそもそもの始まりでした。

その後, SIGENVの会合で次年度計画を立てる段になり, 「若手をSIGENVでやったらどうか?」と誘い水を向けたら, 自分たちの勉強を兼ねてやりましょうということになったのでした。

2 CRC手法採用について

当時SIGENVでは, オブジェクト指向の勉強会を続けており, オブジェクト指向のワークショップを通じて, なるべく実践的に分析設計作業をやって見ようということになりました。その為には, ツールが必要であるとか習得に時間が掛る手法は避けて, 手軽にできかつ共同作業がし易い手法として, 当時テキストに使っていた, Buddの本[1]にあるCRCカードを使った手法でいこうと決まりました。実際には, 全てOOだけでは面白くないので, OO以外の手法も取り上げ, ワークショップの最後の全体討論の時間で各手法の評価をすることにしました。そしてSIGENVメンバが興味のある代表的な手法として, ジャクソン法と構造化分析を選び3本立てで望む事になりました。

3 CRC手法と仮想現実

CRC手法の概略については, 本報告書にある福良さんの解説のとおりです。もともとこの手法は, 1989年のOOPSLAでKent Beck†1とWard Cunningham†2が発表した"A Laboratory for Teaching Object-Oriented Thinking" [2]に述べられているように, オブジェクト指向の考え方を身につける為の訓練手法でした。その最大の特徴は, KJ法[3]と同様にカードを使い分析を進めていく点です。具体的には, オブジェクト(クラス)をカード1枚に対応させ, そのクラスの成すべき責任(Responsibility)および, そのクラスが必要とする別のクラスを協力者(Collaborator)として列挙したものを作成します。この作業を繰り返しながら分析を進めていきます。様々な大きさのカードが文房具屋さんで安く買えますし, グループで共同で分析を行なうのに最適です。ある程度分析が進んで, オブジェクトが決まってきたら, 参加者にオブジェクトを割り振ってシミュレーションを行なう事により, 抽出されたオブジェクトの役割を実感し(自分がオブジェクトになる), 確認することができます。これはとても大事なことです。熊谷さんの最近の話の中にも, 長期間に渡る大規模プロジェクトの開発(例えばOS)では, あるプロセスになりきってしまい, そのプロセスとして振る舞う事によるデバグが, しばしば意識せずに行なわれるそうですが, これによりタイミングの微妙な, 複雑な例外処理などの不具合の発見が容易になることがあります†3。不具合がある種の痛みを伴って体験できる†4わけですね。

4 CRC手法からの展開

いずれにせよ, オブジェクト指向を実体験できるわけですから, 入門者には最適というわけです。もちろん, CRCは限られた記法で分析/設計を行なうわけですから, 限界があります。この限界を越える為に, "責任主導設計"[4][5]のように, CRCを初段の分析に使用した後, クライアント-サーバー概念を取り入れ, さらに詳細設計を行なう設計法が提唱されています。

また, 余り馴染みのない手法ですが, "オブジェクト指向概観"でも簡単に紹介しましたように, ヨーロッパ特に北欧においては, 言語(Simula, Betaなど)だけでなく, 分析・設計手法についても独特の手法[6][7][8]が提唱され使用されています。この中でも, Reenskaug等[8]のアイデアOORASS†5では, いきなりオブジェクトを探すのではなく役割(Role:CRCにおける責任に相当)に着目し, これを先ず抽出する事により分析を行なっていこうという立場を取ります。そして, 役割(Role=Why)分析 →オブジェクト型定義(Type=What) → クラス定義(Class=How)と分析を進めます。ただし, ここでは, "オブジェクト"および"クラス"は, 狭い意味に限定されています。

(株) SRA ソフトウェア工学研究所

†1 当時 Apple Computer, Inc.

†2 当時 Wyatt Software Services, Inc. 元 Tektronix, Inc.

†3 私もかつて同様な体験をした事があります.

†4 バーチャルリアリティーの世界ですね.

†5 OORASS : Object-Oriented Role Analysis, Synthesis and Structuring

話が本論からずれてしまいましたが, 参加者からも, CRC 手法の分析力は弱いとの指摘がありました。これは先に述べたように, オブジェクト指向の導入時あるいは初期段階での分析に使用し, その後より強力な分析手法・ツールを使用するべきでしょう。ただし, SIGENV で体験してみた結果では, これでも十分使えると言う意見でした。グループ 3 のコーディネータの桜井さんは, CRC 手法を使って組込み型のシステムの設計を仕事で行なった経験を, ポジションステートメントとして話してくれました。†6一般には CRC 手法だけに頼るのではなく, 状況と開発の各段階に応じた手法・ツールを取り入れていくのが現実的でしょう。

5 OO ブレインストーミング

やはり, OO ブレインストーミングとも呼ばれるように, グループで多面的な視点からブレインストーミング的に, 自由な発想で共同で分析を行なうのがこの手法の醍醐味ではないかと思います。その意味で, CRC 手法を採用した 2 グループの分析状況は, 対照的でした。一方のグループ (G4) には既に OO 手法を実践している人がいて, CRC によるクラス (オブジェクト) 抽出時に, 他の OO 手法によるオブジェクト関連図を作成していました。もう一方のグループ (G3) は, 経験者がいない為, 散々悩み討論した末に, スパッと割り切って思い付くままクラスカードを作成していきました。結果的には G3 の方が素直な構造になったように思いました。一方, 経験者がいるグループ (G4) では, 他の OO 手法との折衷案的な結果が出ていました。どちらのグループでもやはり, オブジェクトの見つけ方が問題となっていました。これについては, 幾つか指針が出ています†7が, 経験を積まないとなかなか体得できないものです。この点でも, 手軽に安上がりにできる CRC 手法を何回か試してみる価値はあると思います。

なお G3 および G4 で使用した CRC カードは, 桜井氏に SEA 若手の会の名前入りの特製カードを作成してきて頂き, これを使用しました。†8

6 ジャクソン法

ジャクソン法を取り上げたのは, 第 5 グループでした。これについては, 手法の解説とコーディネータの渡邊さんの感想に詳しく触れられています。ただ, ジャクソン法については, OO 以上に理解が難しい面がある為, グループメンバは苦労したようですが, 山崎さんの助言と要所要所での解説が大変助けになったようです。またこのグループは, OHP を使い, 記入したものを投影しながら, 全員で分析討論を行なっていました。†9

7 構造化分析

グループ 1 の岡本さんによる SSADM と, グループ 2 の伊藤さんによる SA/SD は, さすがに実績もあり実際に仕事に活用しているため, 地に着いた着実な分析が成されていました。この両グループでユニークだったのは, 分析手法そのものではなく, 分析の態度が重視されていた点です。岡本グループでは白板を使用し, 全員が白板の前に立って図を描きながら分析するというアプローチを取っていました。これは, 岡本さんによると, 座って行なうのでは発言者のみが頑張り他の参加者は思考停止しがちであるので, これを防ぐために全員を当事者として発言を促す方法だそうです。†10

また, 伊藤グループでは, 伊藤さんが実践している模造紙を使った分析を行なう為に, 伊藤さん自身が持ち込んだ模造紙をテーブル上に広げて, 全員が下を向いて作業を進めていました。これも伊藤さんによると, 下を向いた姿勢が思考に適していることを実践する為の方法だそうです。†11伊藤さんから CRC のカードでは小さすぎると指摘されました。

8 山崎さんの "形式的ソフトウェア開発" レクチャ

山崎さんのレクチャは, 私が急に「図書館問題を例にやって下さい。」と挑戦的にお願いしたにもかかわらず, 分かり易く説明して下さり感銘しました。†12フォーマル手法の有効性について, 自分の不勉強を棚に上げて, 疑問を持っていたのですが, なるほどと考えさせられました。これから勉強してみようと思います。

9 運営について

先ず最初に, 実行委員長の渡邊さんならびにプログラム委員の皆さん。言い出しっぺにもかかわらず企画・準備・実行の各段階において, 何も役立てなくて済みませんでした。皆さんの努力により立派に開催出来たことを感謝致します。(報告書をまとめる段になっても足を引っ張っているのが夢であれば良かった。)

9.1 熱気溢れる討論と反省

今回の若手'92@岡崎は異様に熱気溢れるワークショップでした。これは運営側が意図したわけではなく, 考えてみる

†6 桜井氏のこの内容は 近いうちに SEAMAIL に掲載予定

†7 福良氏の CRC 解説 (pp.30) 参照

†8 CRC 法の利点は保存にも検索にも便利で安い易い!

†9 OHP 法の利点は多人数での検討と発表に使えることですね。

†10 そういえば, 何処かの会議室は 4 面の壁全てが白板になっているそうです。また誰でもやるように "消すな!" とか書いてあるそうです。

†11 模造紙法の欠点は保存と検索が大変なことです。

†12 伊藤さんのまとめ (pp.170) 参照

に今回のテーマが "設計" という身近なテーマでありとっつき易かったことと, コーディネータの熱気がメンバーに伝播し, 勢い早朝から深夜 (時には明け方) まで作業をする結果となってしまいました。しかもなお, 良い経験だったと言う声が多いに及んでは, 全く宗教団体の集まりか, はたまた悪名高き XX 研修・YY 一日入隊の類のように思えました。熱気溢れる若手の集まりですから, 少々ハードであってしかるべきですが, それにしても今回はやり過ぎの感がありました。やはり運営側でしっかりガードをかけるべきでした。このような場になると, 全体の熱気・勢いに逆らえなくなりがちです。これをきちんとガードするのが運営側の役目だったのですが, 熱気に侵されてしまいました。反省しています。

その結果, 最終日の明け方迄作業することになってしまい, 最後の全体討論は, 消耗し切った参加者およびスタッフには, 手法の比較検討まで力及ばず, まとまりの無いコメントをする (私自身の総評) ような結果となってしまいました。

9.2 事前準備が効果を発揮

しかしその反面, あらかじめ準備したガイドラインと記入用紙により, 各グループが検討経過を逐次記録しておいた為, 終了時点で既に報告書の素材が出来上がっていました。これがその後の編集作業に大いに役立ったことは, 言及しておくべきでしょう。それにもかかわらず, やはり編集作業は大変で, 私などが足を引っ張ったことで報告書発行が遅れ, スタッフの皆さんにご迷惑をおかけしました。

9.3 持ち込み機材の活用

もう一つ, 今回は準備した機材が大いに役立ちました。その一つは, 直前に購入した携帯型のプリンターで, これを SEA の J3100 および福良さん持参の 98note に繋ぎ, テキストの作成・中間報告のまとめ・印刷・配布と大活躍しました。プリンタと併せて, SEA 中部支部の携帯型コピー機は, 資料のコピーと OHP 作成に役立ちました。申し訳なかったのは, せっかく伊藤さんに用意して頂いた NeXT dimension が, 私の準備不足の為, 伊藤さんのプレゼンテーション以外使われなかったことです。伊藤さんならびに重い機材を膝に抱えて運んで頂いた MASC の皆さん済みませんでした。

9.4 開催地について

最後に挙手してもらった結果によれば, このワークショップ参加については, 場所が何処であろうと参加決定の要因ではないという意見が大勢を占めました。今回開催地を岡崎に決めたのは, 不況の時期であるから関東からも関西からも近い, 中間地点を選ぶことによって参加し易くしようとの配慮だったのですが, 考え過ぎたようです。しかし, 岡崎だから参加できたという地元からの参加者も居て一安心でした。なお, 次回は何処がいいか尋ねたところ九州という声が強かったようです。(異様に九州人が多いワークショップでした。;-))

10 最後に... 謝辞

参加者の皆さん, 大変なイベントに参加して頂きましてありがとうございました。この体験を少しでも実作業に役立てて頂ければ幸いです。

プログラム委員兼スタッフとして参加して下さった皆さん, ごくろうさまでした。SIGENV ならではの夏のプログラミングワークショップが開催できたと思います。特に伊藤さんにつくって頂いた岡崎マップ (pp.256) は, 大いに役立ちました。

特別に参加して下さった山崎さん, 要所要所での助言はとても役立ちました。また, 突然お願いしたレクチャーを心良く引き受けて頂きありがとうございました。

未だ報告書の最終編集作業が残っていますが, 原稿は全て出来上がりました。編集作業に入ると, いろいろ手直しをしたい部分が出てきますが, 山は越えたようです。

最後に, SEA のビデオカメラで撮った画像を伊藤さんが持ち帰り, NeXT dimension を使って, Tiff ファイルに変換して頂きました。報告書はこの画像データを使った, 写真入りの画期的なものになるはずです。伊藤さんどうもありがとうございました。

とりあえず, やっとできた!!

参考文献

[1] Timothy Budd : An Introduction to Object-Oriented Programming, Addison-Wesley Publishing Company, Reading, MA,1991.

[2] Kent Beck and Ward Cunningham : A Laboratory for Teaching Object-Oriented Thinking, OOPSLA'89 ACM, Oct 1989

[3] 川喜田 二郎 : 続・発想法 KJ 法の展開と応用, 中央公論社 中公新書 210, 1970 年 2 月.

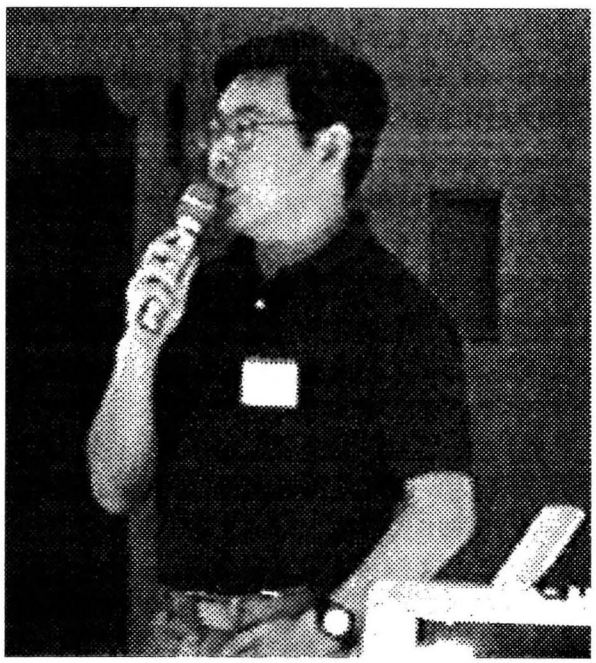
[4] Rebecca Wirfs-Brock and Brian Wilkerson, A Resposibility-Driven Approach, OOPSLA'89 ACM, Oct 1989

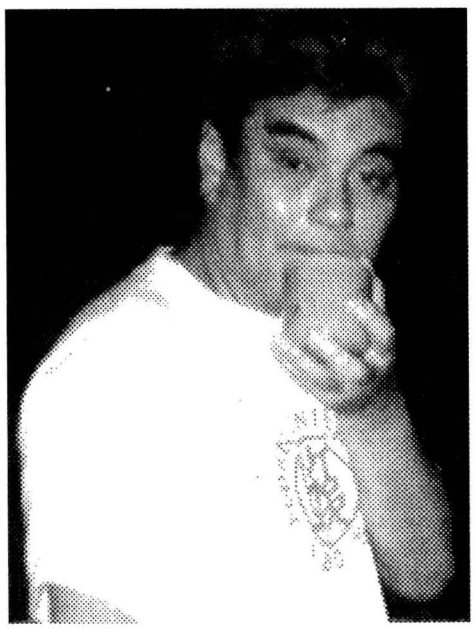
[5] R.Wirfs-Brock, B.Wilkerson, L.Wiener : Designing Object-Oriented Software., Prentice Hall, 1990.

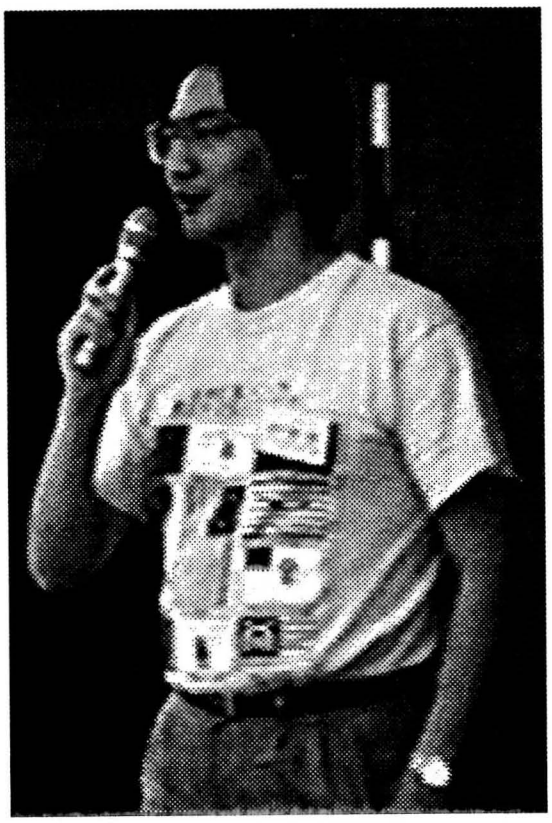
[6] Ivar Jacobsen : Object-Oriented Development in an Industrial Environment., OOPSLA'87 ACM, Oct 1987.

[7] Ivar Jacobsen, Fredrik Lindstrom : Re-engineering of old systems to an Object-Oriented architecture., OOPSLA'91 ACM, Oct 1991.

[8] T.Reenskaug, E.Nordhagen : The Description of Complex Object-Oriented Systems: Version 1. Senter for Industriforskning, Oslo, Norway, 1989.







2. 手法の説明

- SA/SD について (伊藤 昌夫) 19
- SSADM について (岡本 隆一) 23
- JSD に関すること (渡邉 雄一) 27
- CRC カードによるオブジェクト指向設計 (福良 博史) 30

SA/SDについて －ベルクソン風まやかしのSA/SD論－

伊藤昌夫

標準的なSA/SDに関しては良書が多くでており，私の下手な紹介よりもそちらを読んで頂いた方がよいのは明らか。ここでは，ちょっと違った視点からSA/SDを紹介したい。冗談が嫌いな人は，最後の参考図書のみ参照して欲しい。†1

1 方法としての直感

「繰返」・「忍耐」・「飛躍」は，SA/SDを行う上での三つの大きな段階を示すものである。本論の目的は，この三つの概念がどのようなものであり，どのように関連するかを解き明かすことにする。

これら三つの段階に於て最も基本になるのは直感である。直感は単なる思い付きとは違う。この辺がさみしいから丸(プロセス)を一つ書いておこうとか，コンテキストダイヤグラムは左右均等でないといけないよねと云って，四角(ターミネータ或いは，源泉/吸収)を偶数個にするのとは違う。

ここでの直感は非常にとぎ澄まされた感覚が，僅かな時間だけ認識できる，人間にとって普遍のものである。これは，西田幾太郎氏の「純粋経験」と言い換えてもよい。

それは，徹夜でSAを書きながら窓の外に朝焼けを見た時ふと心をよぎるものである。しかし，「これだ」と思った時にはもう真の「直感」は逃げていく。この時あわてて悟った気になるとたいていの場合良いものはできない。

2 必然としての「繰返」

「直感」を磨くためにはそれなりの訓練が必要となる。

SA/SDの基本は世の中の事象をすべからくデータの流れとして捉えようとしているのであり，データフロー図をして，プログラムのロジックフローを頭の片隅に於てはいけない。人間は中途半端に優秀であるがために，データフロー図を書きながらも，これを余り細分化するとドキュメント数が増える，従って仕事をした気になれると即座に思う，つまり偽のイマージュを無意識下に作りあげてしまうのである。

これは，作りあげようとするシステムをDFD(或いはCFDを含め)にマッピングしようとする純粋活動から大きく離れてしまう。

繰返，繰返我々は，こういった偽のイマージュをそぎ落としていかなくてはならない。

構造化分析は，分割がその基本にある。例えば，プロセスは必要に応じて次々と細分化される。細分化された途端，それはシステムのあるがままの姿から変貌する。連続的なシステムは無限の方法で分割可能であることを知るべきである。

従って，現実を見ない繰返は，無限ループに陥ることを充分注意しなくてはならない。(注意しないと，「朝焼けがきれい」と一人呟くことになる。)

3 共存としての「忍耐」

ここでの「忍耐」は，今時SA/SDをやっているとOOADをやっている人間から笑われるので，それに対して耐えると云うことではない。

ここでもキーワードは，「直感」である。SAにより明確になった(と誤解できる)DFDは，それを共通語として，要求元と議論することになる。また，SDより明らかになった構造図やモジュールスペックは，同じく共通語として，コーダと議論することになる。

ここで，我々は「忍耐」の何たるかを知ることになる。いやそれ以前に複数人でシステム分析を行えば，我々は自分の仲間のたちの悪い性格を嫌と云う程知ることになる。SA/SDの最大の阻害要因はここにあると知るべきである。我々はよいシステムを作ると云う共通の目的に立ちながらも，人間関係を大切にするがために，この「忍耐」を避ける。

この解決策が「直感」である。「直感」は全ての属人性を排除した正当に議論できる知であり，深い「忍耐」を通して，「直感」を得るようにしなくてはならない。

しかし，初心者にとってこの「直感」を得るのは非常に困難である。SA/SDではガイドラインの形で「直感」に近づく幾つかの方法がある。例えば，SAに於ける分割限度の考え方，SDに於けるモジュール結合度，凝結度がそれに当たる。

これらを通して，「直感」を妨げる不要なものをそぎ落としていくことができる。

MHIエアロスペースシステムズ(株)

†1 編集部注: 「敢えてSA/SDの基本的な概念/表記法などは説明しないよ」という，伊藤氏 一流のjokeである。なおSA/SDについては，SEAMAIL Vol.6 No.8–11 pp.10-19に萩原先生(阪大)の丁寧な解説があるので まずそれを読まれることをお薦めする

4 差異化のための「飛躍 (ヴァイタル)」

ここでの「飛躍」は, SA/SD をサポートする CASE ツールの導入によって, 一挙に, 生産性を 3 倍にあげると云うことではない。(そんなことはありえないと云う「直感」は常に正しい。)

云うまでもないが, CASE ツールが大事なわけではなく, SA/SD が重要なわけでもない。システムを分析することや, 設計することが重要である。

ここで云う差異化のための「飛躍」は, 自分なりの突拍子もない表記法を編み出して, その表記法に自分の名前をつけて, これで論文が書けると自己満足すると云うことでもない。

SA/SD に関しては, 幾つかの標準的な表記法がある。例えば Yourdon/Demarco のものと Gane/Sarson によるもの。また, リアルタイム拡張したものとしては, Ward の方法と Hatley の方法がある。

しかし, 大事なのは, 分析・設計を通して流れる一貫した思想であり, それには, どの考え方や表記法が優れていると云う分けではなく, 必要に応じて, (細かい部分では, 自分たちのルールを定めつつ,) 自分が一番使いやすいものを使えばよいのだ。

では, 一体差異化のための「飛躍」とは何か。

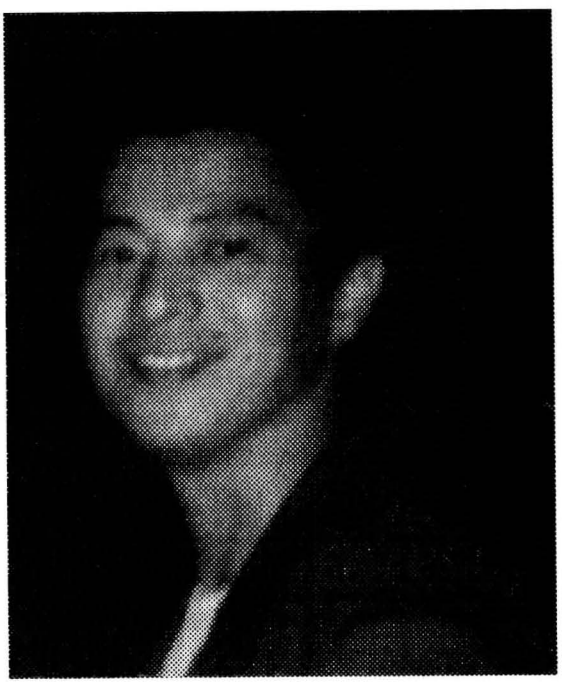
簡単である。ごたごた云う前に SA/SD を使ってみることである。間違っても, OOAD などに惑わされてはいけない。:>

参考文献

[1] T.DeMarco (高梨/黒田 監訳), 構造化分析とシステム仕様, 日経 BP, 1986
1978 年に出た構造化分析に関する古典の訳。随所にある質疑応答は素朴な疑問解決に役立つ。

[2] D.Hatley (I.Pirbhai 訳, 立田監訳), リアルタイム・システムの構造化分析, 日経 BP, 1989
構造化分析手法をリアルタイム化した Hatley 氏の方法の著者自身による解説。Ward 氏との手法の比較としては, P.Ward, 立田「2 つのリアルタイム SA 手法 (ワード VS. ハトレー)」と云う記事が bit 誌の Vol.22, No.11 にある。

[3] M.Page-Jones (久保・新谷 共訳), 構造化システム設計への実践的ガイド



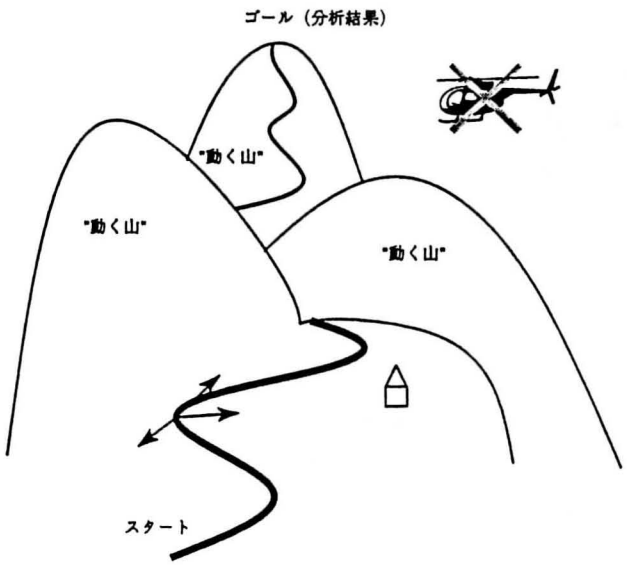
第１０回夏のプログラミングワークショップ
現場でのシステム設計を改革するテクノロジーを求めて

第２グループ　「ＳＡ／ＳＤ改」設計方針

「現場でのシステム設計に、明日から役立つ"小さな発見"を求めて」

ＭＨＩエアロスペースシステムズ株式会社
開発Ｇ　伊藤　昌夫

1. 設計とは



人生は重荷を背負いて、遠き道を行くが如し。（家康）

【方針Ｉ】
・根気強く
・分析／設計は、微小振動を加えながら最適解へ
・分析　⇔　設計　を繰り返しながら。

2. 設計手法とは

2.1 設計手法のありがたさ

(1) 教育的側面

(2) 迷った時のガイドブック

(3) 複数人での意思の疎通（但し、遠隔非同時が主か？、過程は？）

2.2 4つの型（Song&Osterweil)＋1

(4) 行動型－物理的かつ心理的実行ステップ
(例) 実体をを識別するのに先ず名詞を捜す。

(3) 表現型－生成物を記述し、識別する
(例) p-specを記述するのに用いる構造型言語

(2) 生成物型－設計プロセス中に含まれるある種のエンティティ
(例) データフローダイヤグラム

(1) 概念型－設計手法の基となるアイデア
(例) データフローに着目する

(0) 心の持ち型－他者を理解しようとする心（非技術的）
(例) 相互に信頼しあう、理解しあう。"美しさ(永続性、整い）"の理解

【方針Ⅱ】
・模造紙の利用。
・小さな迷いと解決策を記述しておく。
・"美しさ"について理解する。

3. 目安となる進め方

（1）【模造紙上の落書き】－何もかも一枚の紙に書く（但し、最初は調査票で個人ベース）

（2）【落書きの整理】－色ペンを使って整理する。（消しゴムは使わない。）

（2）【シミュレーション】－各人がターミネータ或いはシステムの一部になりきる。

（3）【ＳＡ形式への変換】－ＳＡ上の制約・記法・やり方を反映する。（但し、大事なのは正しい心の持ち方）

（4）【要求定義】－設計対象となるシステムを明確にする。

（5）【ＳＤへの変換】－（実装ベースのＤＦＤ等の変換をやった後)、ＳＤ記述

【その他】

・p-specは設計記述言語inferで記述。（もし、ここまでいったらの話だが)

・悩んだこと、考えたことを調査票に記述する。

SSADMについて

岡本 隆一

構造化システム分析・設計技法 SSADM(Structured Systems Analysis and Design Method) は, 1980 年代に英国において導入されたシステム開発の方法論の中のひとつです。SSADM の原案は LBMS 社 (Learmonth Burchett Management System) のもので, 英国政府の CCTA(Central Computer and Telecommunication Agency) と協同で開発されました。

SSADM は, システム開発の規範として下記 (1)〜(5) の基本的な考えに基づいて開発されています。

(1) システム開発努力をいかにして成すべきか..., 規定。
(2) プロジェクトをどう組織化するか。
プロジェクトの中で行う仕事の順序や仕事と仕事の間のインタフェースを記述したもの。
(3) わかりやすくて論知的で一貫性があるもの。
(4) あくまでガイドラインとして。
(5) だから, いくらでもアレンジできる, カスタマイズできる。

CCTA は, 1981 年 1 月に LBMS 社の方法論を受入れ, それをプロジェクトで実践し, 1983 年 1 月には, SSADM を英国政府の標準技法として位置付けました。OOA・OOD のように目新しい方法論ではありませんが, 以降, 改良が加えられ, 1986 年 6 月には第 3 版が発表されるに至っています。

SSADM は 以下のような (可能性) 構造をしています。

局面 1 可能性調査
　段階 01 問題点の定義
　段階 02 プロジェクトの定義
局面 2 システム分析
　段階 1 システムの働きと現行の問題点の分析
　段階 2 要件の定義
　段階 3 技術的オプションの選択
局面 3 システム設計
　段階 4 データ設計
　段階 5 処理設計
　段階 6 物理設計

このうち**局面 2:システム分析の段階 1:システムの働きと現行の問題点の分析** と **段階 2:要件の定義** の 2 つの段階のステップの関連を表した図を参考に掲げます。

数多く出版されている構造化システム分析・設計を扱った文献の中では, [1] に SSADM が解説されているので参考までに紹介します。

参考文献

[1] Ed DownsPeter, ClareIan Coe (伊藤武夫 訳): 構造化システム分析と設計技法 SSADM -その適用と状況について-, 近代科学社

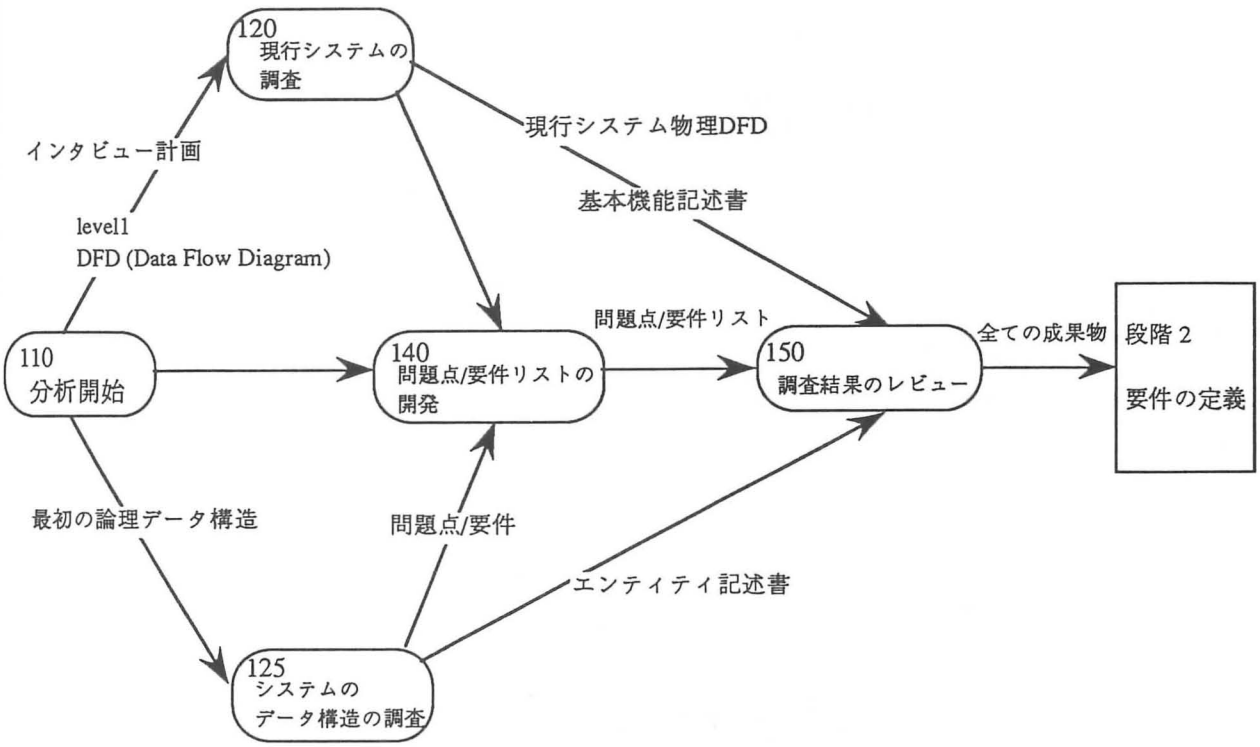
株) さくらケーシーエス

図1 SSADMの骨組み

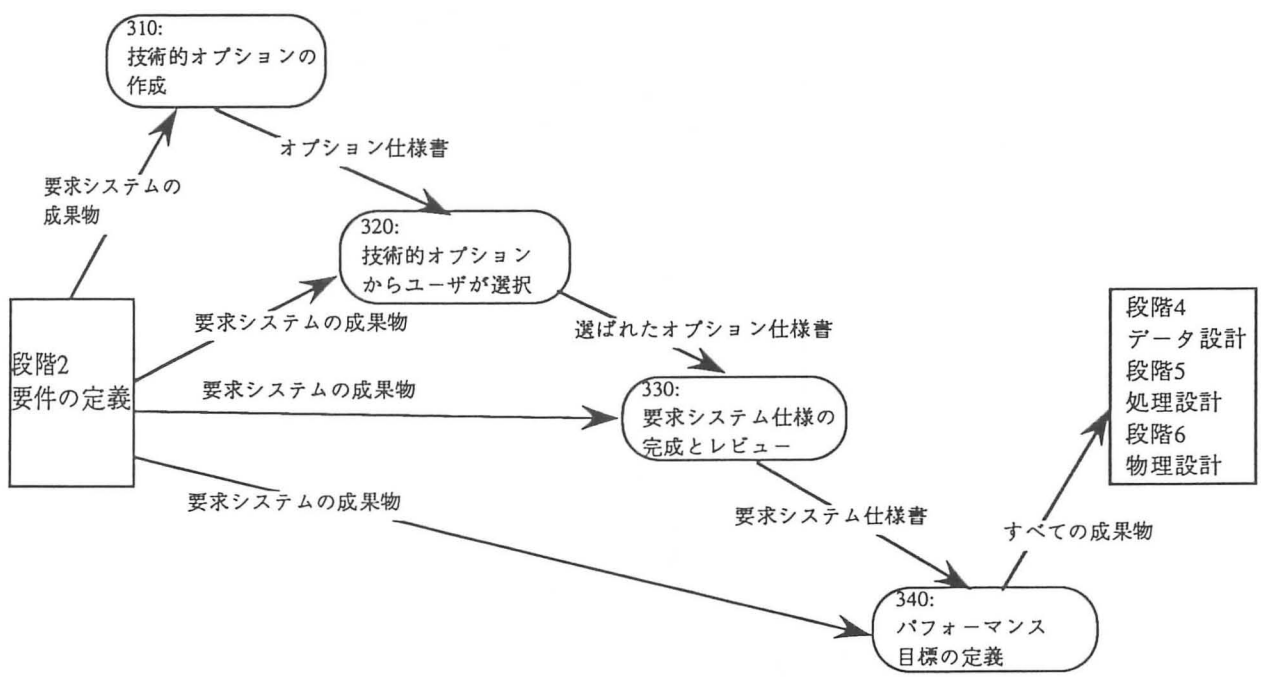
技法 \ ステップ	010	011	012	013	014	019	021	022	023	029	110	100	125	140	150	200	205	210	220	230	240	250	260	310	320	330	340	410	420	510	520	530	610	620	630
DFD (Data Flow Diagram)	*	*		*			*	*		*	*	*				*			*	*															
LDST (論理データ構造技法)	*		*					*		*	*		*							*	*	*													
ユーザの選択 (決定を下すのはユーザ)									*									*				*													
ELH (Entity Life History)																					*										*				
対話設計 (LDO, LDC)																					*									*	*				
RDA (関係データ分析)													*															*							
CLDD (複合的論理データ設計)																													*						
PO (LUPO, LEPO)																														*	*				
FCDD (物理データの設計)																																	*		
FCPD (プログラムの設計)																								*										*	
PDC (物理設計の管理)																								*											*

LDO Logical Dialogue Outline：人とコンピュータの論理的対話の概要
LDC Logical Design Control：LDO を連結してオンライン処理の相互作用 (論理的な設計制御) を実現
CLDD Complex Logical Data Design：複合的論理データ設計
PO Process Outline
LUPO Logical Updating Process Outline：イベントの論理的更新処理の概要を作成
LEPO Logical Equiry Process Outline：論理的な照会処理の概要の定義

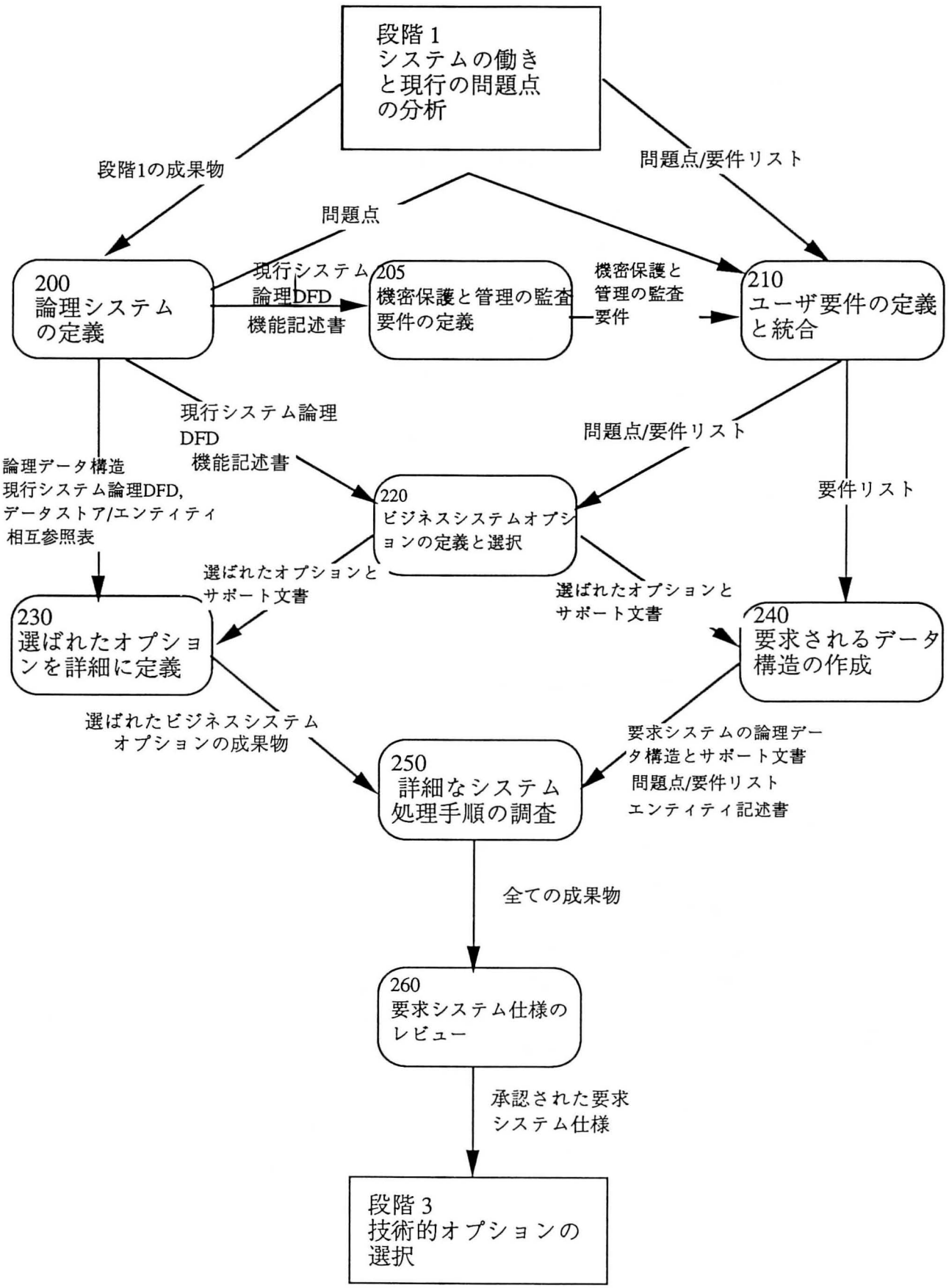
段階１：システムの働きと現行の問題点の分析



段階３：技術的オプションの選択



段階２：要件の定義


段階 1
システムの働き
と現行の問題点
の分析
段階1の成果物
問題点/要件リスト
問題点
200
論理システム
の定義
現行システム
論理DFD
機能記述書
205
機密保護と管理の監査
要件の定義
機密保護と
管理の監査
要件
210
ユーザ要件の定義
と統合
現行システム論理
DFD
機能記述書
問題点/要件リスト
論理データ構造
現行システム論理DFD,
データストア/エンティティ
相互参照表
220
ビジネスシステムオプシ
ョンの定義と選択
要件リスト
選ばれたオプションと
サポート文書
選ばれたオプションと
サポート文書
230
選ばれたオプショ
ンを詳細に定義
240
要求されるデータ
構造の作成
選ばれたビジネスシステム
オプションの成果物
要求システムの論理デー
タ構造とサポート文書
問題点/要件リスト
エンティティ記述書
250
詳細なシステム
処理手順の調査
全ての成果物
260
要求システム仕様の
レビュー
承認された要求
システム仕様
段階 3
技術的オプションの
選択

JSDに関すること

渡邉 雄一

プログラム委員としては, はなはだ準備不足であるが, JSD(Jackson System Development) を担当することを希望した。ただ 困ったことに, 自分自身で普段からこの手法になんらかの関わりがあるわけでもなく, 唯一の関わり合いと言えば, 今は伝説となりつつある 黄色の表紙の SEAMAIL で 伊野さん (当時 日本ユニバック) の JSD についてのセミナのレポート†1 [2] をしたことがあるだけである。ただ, これは Jackson の本[1] が刊行された今となってはSEA の歴史を語ると言う目的以外には, 特に意味のあるものではなくなっている。

JSD をひとことで言い表すのは, 私のような未熟ものに到底出来るようなことではないので, こちらも腹をくくって Jackson の本 [1] の最初の部分を読んで理解したことをプレゼンテーションで示しただけで終った。以下にそのメモを整理した形で掲げる。

1 JSD とはどのような手法か?

モデルを段階を追って完成させることで, システムの設計を行なう。

Step1 : 実体行動段階

実体 (Entity) とその実体が振舞い振舞われる**行動** (Action) によって, システムの背景となる実世界を記述して, その主題を定義する

実体 (Entity) に対する一般的な判断基準

1. 実体は意味のある時間的な順序づけに従って, 行動したりされたりしなければならない
2. 実体はシステムの外部にある実世界に存在するものであって, 単なるシステムそれ自身の一部分, あるいは, システムが作成するものであってはならない
3. 実体は, 個体とみなせるものでなければならず, 同じ型の実体が 2 つ以上あれば, それは 1 つひとつが名前を持ち, 識別出来なければならない

行動 (Action) に対する一般的な判断基準

1. 行動は一定期間に持続するものではなく, ある時点に起こるとみなされるものでなければならない
2. 行動は, システムの外部にある実世界で起こるものであって, システムそれ自身の単なる動作であってはならない
3. 行動は原始的なものであるとみなし, その分割は出来ない

Step2 : 実体構造段階

実体の行動は, 時間的に順序づけられているので, その順序を表現する必要がある。これを, ジャクソン構造図で示す

Step3 : 初期モデル段階

- システムの模擬を記述することから, システムそのものの仕様化を始める
- 結合経路の陳述
- システム仕様図 (SSD:System Specification Diagram) とモデルプロセスの構造

Step4 : 機能段階

- モデルの上にシステム機能を仕様化する
- 必要な機能毎の仕様を次の 3 通りの方法で文書化する
  1. 日常用語による要求を初期モデルの用語を使って書き直す
  2. 初期モデル SSD を洗練して, 機能がどのようにモデルプロセスに組み込まれ結合されるかを示す
  3. 構造文によって, 機能の詳細を示す

Step5 : システムタイミング段階

- 発生する可能性のある 遅れ について考察し, システムの利用者とともに, システムの各部に対してどれだけの遅れが許容されるかを決定する。
- その結論を, 非形式的な形で文書化し, 実現段階への入力とし, モデル内で応答時間や時間の遅れを満足するように実現の選択範囲を限定する

Step6 : 実現段階

- SSD とプロセスの構造の中に盛り込まれたシステム仕様を実現において保存する
- システム実現図は, システム仕様図の変形であるとみなすことも出来る
- 実現段階で作成する文書は, システムの利用者との話し合いに使用するものではなく, 純粋に技術的

株) アスキー OA 推進部

†1 参加者には, これを配布した

なものである

2 JSD のねらうところ

ここでは,『第 1 章 モデルと機能』から 重要と思われる箇所を書き出す。

- JSD ではシステムの機能については開発の後期段階で考え, 実世界のモデル化を最初に行なう
- JSD の基本原理は, まず最初に現実をモデル化し, そうしておいてからはじめてシステムが遂行しなければならない機能の詳細を考える
- JSD の中での モデル とは つねにコンピュータシステムの外部にある実世界のモデル という意味で用いるシステムそれ自体のモデル, システムの属性や特性のモデル, あるいはシステムの開発手順などのモデルなどの意味では決して使わない
- 実世界の JSD モデルを作成するには,
  1) 実世界の抽象記述を作成
  2) この抽象記述をコンピュータ内部で具体化
  という 2 つの相異なった作業が必要
- 具体化されたモデルは, そのモデルが果たす役割を十分達成できる水準において, 抽象記述を満足するものでなければならない
- モデルから始める理由
  1) モデルは, システム機能に対する計算の基礎とともに概念的な基礎をも与えてくれる
  2) モデルがそこに付加できるシステムの機能の範囲を間接的に定義する
- JSD においては, 前もって作られたモデルによって, 機能は曖昧さなしに仕様化出来ることを保証している
- 機能を直接考えることから開発を始めると, 厳密に関連し合った一連の機能群を定義することが常に困難になる
- 機能的に開発された解の欠点は, それが関係している実世界をモデル化していない
- モデルは機能よりも安定している
- モデルと機能は不可分の関係にある。実世界のモデルを前提としないでシステムの機能は語れないし, また念頭にある目的を前提としないで実世界は語れはしない
- JSD の場合には, 開発者と利用者の初期の話し合いはすべて利用者の実世界について行なう
- 結論 : 実世界のモデル化は, すでに存在しているものの記述であり, システム機能の定義仕様化は何か新しいものの創造である

3 プロセス

ここでは,『第 2 章 プロセス・モデル』から 重要と思われる箇所を書き出す。

- JSD でモデル化したい実世界は, 時間軸がつねに最重要である世界である
- 時間軸の重要性が, 事実, JSD の応用分野を定める特性になっている
- 事象の発生順序が固定していようと変化しようと, 順序そのものがいつも関心の的であり重要なのである
- 静的なモデルと動的なモデル
- 動的な実世界は, (通常の意味の) データベースでは適切にモデル化出来ない
- だから 実世界には動的なモデルを必要とする
- 最初に静的なモデルを作り, 後から, モデル化される実世界の動的な側面を適合するように, そのモデルを修正するようなことを JSD では採用しない
- JSD の動的なモデルは, いくつかの 逐次プロセス によって構成する

参考文献

[1] M. Jackson (大野, 山崎 [監訳]) : システム開発 JSD 法, 共立出版, 1989
[2] 伊野 誠 (渡邉 [記録]) : システム設計技法 JSD と支援ツール, SEAMAIL, Vol.1, No.11 (1986) pp.8-11

JSD 記法の一部

Jackson 構造図

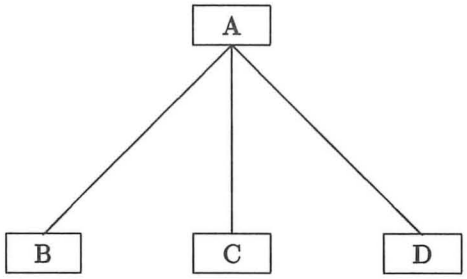


fig.1 A は連接要素

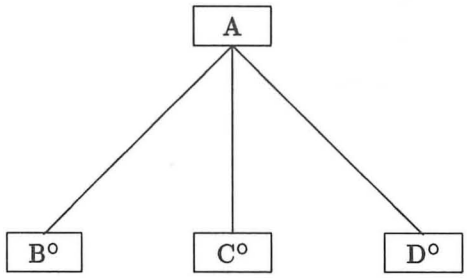


fig.2 A は選択要素

fig.3 A は反復要素

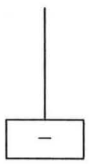


fig.4 (–) は空要素

構造文

```
A seq
    B;
    C;
    D;
A end
```

eq.1 A は連接要素

seq →

```
A sel(cond - B)
    B;
A alt(cond - C)
    C;
A alt(cond - D)
    D;
A end
```

eq.2 A は選択要素

select →

```
A itr while(cond - B)
    B;
A end
```

eq.3 A は反復要素

repeat →

```
A itr
    x;
A quit(cond - 1)
    y;
A quit(cond - 2)
    z;
A end
```

eq.4 A は後戻りを伴う反復要素

```
A posit
    x;
A quit(cond - 1)
    y;
A quit(cond - 2)
    z;
A admit
    w;
A end
```

eq.5 A は後戻りを伴う選択要素

CRCカードによるオブジェクト指向設計

福良 博史

1 はじめに

ソフトウェアの設計/製造/保守を通して常に問題になるのは, 品質の維持である。また, 既存の設計にも問題がある。この品質要因と, 現在の設計の問題点の概略を B. Meyer は [5] で以下のように述べている。

【安全性にかかわるもの】

(1) **正確さ** Correctness
システム要求を完全に形式的に表現

(2) **頑丈さ** Robustness
仕様に明示されていないことが発生した場合, システムが破滅的な状況に 陥ることの回避

【分散型アーキテクチャが必要なもの】

(3) **拡張性** Extendibility
大規模プログラミングの仕様の「変更, 拡張」を向上させること
・簡単なアーキテクチャ
・モジュールが非集中的で, 独立しているほど影響範囲を局所化可能

(4) **再利用性** Reusability
再利用性に富む → 労力の削減

(5) **互換性** Compatibility
互換性に富む → 製品相互の組合わせ容易
・個々のプログラムが同質
・プログラム間の通信規約が一致

【その他】

(6) **効率性** Efficiency
ハードウェア資源の使用効率

(7) **携帯性** Portability
移植の容易性

(8) **実証性** Verifiability
テストデータの準備, エラーの探索の準備が容易

(9) **統合性** Integurity
不当なアクセスから構成部分を守る能力

(10) **使い易さ** Easy of Use
使い方の修得/操作, 入力データの準備, 結果の解釈, 誤った操作によるエラーからの回復

以上のような各種の要因がある。これらを全て満足させることは不可能なので開発するシステム毎に, なにが重要なのかを明確にして, 取捨選択することが必要である。

尚, 現在のソフトウェア設計においては, 以下のような問題がある。

1. データの物理構造などの情報がシステムのあちこちに散在
データが統一的に管理されていないために, 同じデータの情報が複数箇所に散在している。これを解決するためには, 「抽象データ型」の考えを導入する必要がある。… この考え方[†1]は, オブジェクト指向の基本である。

2. ドキュメントがソースと分離
時間の経過とともに, ドキュメントがソース・コードと遊離してしまう。これを回避するようにすることが望ましい。このためには, ドキュメントの内容が, ソース・コードに含まれるような形態が望ましい。このためには, 初めから設計情報の記述が可能なような言語になっていなければならない。

2 設計方法の比較

従来の設計方法と, responsibility–driven による設計方法の違いを論文 [2] から紹介する。

1. **Data-Driven Design**
この設計方法は, 従来行われていた設計基準である。この方法の基準は, 概ね以下の 2 点に要約される。
   1) 何 (どういう構造) をこのオブジェクトは表すのか?
   2) どういう操作をこのオブジェクトは行なえば良いのか?

   このような基準で設計された場合, あるクラスのデータ構造に他のクラスが依存しているため, 構造の変更が困難となる。

2. **Responsibility-Driven Design**
この設計基準を用いることにより, あるクラスのデータ構造が, 他のクラスに依存しないで済むようになる。この基準の基本になる考え方は, クライアント/サーバ型の考えである。この考え方を以下に示す。

株) ジェー・エム・エー・システムズ

†1 この考え方のイメージは, 付録 B (pp.37) を参照

3. **Client/Server Model**

クライアントは, サービスを遂行するために, サーバーへの要求を用意する。サーバーは, 要求に対するサービスの一揃いを用意する。クライアントが, サーバーと相互に作用するための方法は, 契約によって記述される。これは, クライアントが, サーバーに依頼できる要求の一覧である。クライアントは, 決められたことの範囲でのみ要求できる。サーバーは, これらの要求に責任を持つ。

以上がクライアント/サーバ・モデルである。これを基にした設計基準は, 以下のようになる。

1) どんな振舞いにこのオブジェクトは責任があるのか?

2) どんな情報をこのオブジェクトは分担するのか?

この基準で注意してほしい点は, オブジェクトによって分担されている情報は, そのオブジェクトの構造の一部であっても, そうでなくてもよいということである。

3 CRC カードによるオブジェクト指向設計

CRC とは,

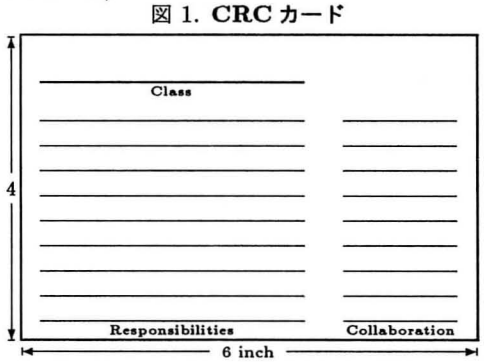
Class	クラス名
Responsibility	(契約した) 責任
Collaborator	協力者

の略である。つまり, 設計の基本的な考え方は, responsibility-driven の設計方法である。

表面には, 上述の CRC の三種類の場所がある。そして裏面は自分が管理しているデータを記述する (information hiding)。

CRC カードの構成を以下に示す。(4 インチ * 6 インチの大きさのカード)

図 1. **CRC カード**



システムの全体的なつながり, 挙動をカード間のやりとりで定義していく。カードはその内容が複雑にならない程度の大きさで机の上に置くことができる。

また, 移動させることで他のカードとの関連も表現できる。

さらに, 費用がかからなく, 書き直すことができる。

複雑になりすぎたら 新しいカードにわければよい。

3.1 設計手順

設計の進め方 CRC カードでの設計は, 従来のソフトウェア開発で行われているトップダウン設計でもボトムアップ設計のどちらでもない。かわりとするなら, 既知から未知へ進めていく過程である。

設計はもっとも明かなクラスからはじめる。そのクラスはアプリケーションのはじめに処理されるために必要なものである。設計者が「こうしたらどうだろう」形式で図などで説明しながらシナリオをシミュレーションしていく。

グループの利用 CRC カードでは, このシミュレーションプロセスにおいて具体的な役割りを演じることが出来る。もし, いくつかの個人が設計上一緒に仕事をしているなら, おのおのが 1 つまたはそれ以上のクラスの役割りを演じる。おのおののクラスが関連したシナリオになるように「実行している」とき手を上げ, 1 つのカードから他方へ (一人の設計者からもう一人の設計者へ) 動くようにはっきりと視覚的に「制御」の渡りがわかるようにする。それにより, お互いの協力者として閉じているならカードは一緒のグループとすることができ, お互いから独立したものは, 物理的に分離することが出来る。シナリオが同一とみなすことができるなら, その部分を新しいクラスとして演じさせればよい。さらに, 違ったシナリオ (エラーや例外処理) も試みることにより, その責任も生み出される。以上のことを設計者は手で直接問題の解決を促進させることができる。

評価 このテクニックは, 本来, 実験的設計のためのアプローチを促進する。カードの修正の容易さから違った設計の評価が早くできる。その作業は, システムのおのおのの構成要素の理解と明確な記述の段階的な発展を図る。シナリオの作業を行うことにより, 不必要な機能を取り除き, 必要な情報によって構成された設計が完了することが保証される。

カードの「もの」としての姿が設計者にとって責任のより構成要素の理解とはっきりした記述の発達途中のものとなる。シナリオの作業により, 不必要な機能をさけ, 作業の完了のための必要な情報を含んだ設計となると保証できる。

カードのものとしての姿が設計者にとって責任のより直感的な理解を促進する。設計者がカードを掲げて「これをするためには, 私はあなたにそれをするように言わなければならない」と言い, 設計チームのもう一人を指

し示す。

CRC による設計の手順の概略を以下に示す。

1. Class を見つける
   - 仕様書から名詞を洗い出す
   - 洗い出した名詞から, その役割を推測し, 必要なクラスを抽出する
2. Responsibility を各 Class に記述する
   - Responsibility の無いクラスは, 廃棄する
3. Collabolator を決める
   - クラス毎の役割分担を決める
4. Inheritance 関係を決める
   - クラス毎の親子関係を決め, 役割の共通化を決める

以上の手順により, クラスとクラス間の関係を決め, 各クラスの役割を明確化する。なお, この手順は 1 回限りで完了するとは限らない。何度もリファインを繰り返す場合がある。†2

3.2 名前の付け方

3.3 にてクラスの見つけ方の説明をする。その前に, クラスに限らず名前をつける場合の注意を以下に示す。

1. 読み易い名前であること
   大きな声で叫んでも, 舌がもつれたりせず, ハッキリと呼べるような名前であること。
2. 誤読しない名前であること
   間違って読まれるような, まぎらわしい名前にしないこと。
   (例) 数字は使わない：ゼロ (0) と オウ (O), 2 と Z
3. 誤解しない名前であること
   他人が読んで誤解するような名前をつけないこと。特に, 省略形の時に注意すること。
   (例 1) 収支計：収入/支出の計算? それとも, 収入/支出の合計?
   (例 2) TermProc : TerminalProcess ? それとも TerminateProcess ?
4. 理解しやすい名前であること
   他人が理解しやすい名前をつけること。特に, 真偽値による状態などを説明するための名前などは, 理解しやすい名前にする。
   (例)「プリンタ状態」よりも「プリンタ印字可能」が望ましい。

3.3 Class の見つけ方

クラス名は, CRC カードの左上に書く。先ず, 最初にクラスを抽出する必要がある。どのようにしてクラスを見つけ出すのか。参考までに, 以下に幾つかのクラスを選択するための考え方を紹介する。

1. 仕様書から名詞を取り出す
2. Data Manager となるクラスを見つける
   データを維持管理しているクラスである。
3. Data Source となるクラスを見つける
   データを保持しないクラスであり, 例えば, 乱数発生, データ入力, データ出力などを扱うクラスである。
4. View となるクラスを見つける
   例えば, 端末の画面表示を行うクラスである。
5. Helper となるクラスを見つける
   複雑な仕事の実行の補助を行うクラスである。

以上のような観点からクラスを決める。

なお, 2 から 5 までのクラスは, 一例であり, そのほかにもクラスの種類は, 考えられるであろう。又, これらが複合して, 一つのクラスを構成している場合もある。

とにかく, 第一のクラス抽出基準は, **名詞**である。

3.4 Responsibility の決め方

クラス名のすぐ下には, クラスの責務を掲げる。

Responsibility とは, そのクラスが責務をもって遂行する契約事項を明記する欄である。

責務欄は, 解決されるべき問題を記述し, おのおの短い動詞句で表現する。なすべきことは何かを記述し, どのように仕事がなされるべきかの詳細は無視する。この欄は, 契約として見ることもでき, 実際の契約のようにその代償物も記述する。→あなたがこのオブジェクトを使用するときそれが得られる結果となるように。

この欄に何を記述すべきか。その基準を以下に示す。

1. Responsibility(責務) は, 仕様書の動詞から抽出する。
2. 責務遂行に必要なデータにアクセスし, それ以外のデータにはアクセスしない。
3. 他のクラスが所有しているデータを直接変更するような責務は間違い。
4. 1 クラスの責務の数は, CRC カードの範囲内にする。あまり沢山の責務を 1 個のクラスに負担させない。この場合, 他のクラスに責務を分担させる。
5. 責務を伴わないものは, クラスにせず削除する。
6. 使われないクラスは, 削除する。
7. データと責務がバラバラにならないようにする

†2 4. CRC カードからコードまで (pp.33) 参照

3.5 Collabolator の決め方

Collabolator は, CRC カードの右側 1/3 ほどのカラムを使用する。

Collabolator とは, 自分だけでは責任を遂行出来ない場合に, その責任を果たすために必要な協力者である。つまり, 他のクラスに依存している場合, そのクラスを Collabolator として明記する。依存されているクラスは依存しているクラスへのサービスの供給者である。

3.6 Inheritance の見つけ方

Inheritance 関係は, 最後に洗い出す。

1. is–a 関係
   α is–a β という関係が成り立つ場合, この関係がインヘリタンス関係である。この時, クラスβがクラスαの親クラス (スーパー・クラス) となる。
   (例) Car is–a Vehicle → インヘリタンス関係
   Vehicle が親, Car が子供の関係
2. 複数のクラスに同一のコードがある場合
   この同一のコードをスーパー・クラスとする。
3. has–a (part–of) 関係
   α has–a β の関係が成り立つ場合, βは, αの属性である。
   (例) Car has–a Door → 属性関係
   Door は, Car の属性

4 CRC からコードまでの例

CRC を用いた 段階的な詳細化 及び それによるコード化までの具体例として簡単な在庫問題を例に示す[†3]。これにより, 自然言語的な記述から, コード的な記述に徐々にリファインする過程を示す。

問題の説明 簡単な在庫管理の問題を考える。

1. 商品を, 品番にて管理するために, 在庫管理用データベースを作成する。
2. 管理のために, 商品番号を, キーとする。
3. レコードとしては, 以下の項目を持つ。
   - 商品名
   - 入庫数
   - 出庫数
4. 初期登録, 入庫, 入庫の訂正, 出庫, 出庫の訂正, 棚卸し時の在庫修正および, レコードの削除を可能とする。
5. 在庫内容の照会は, 以下の事が可能であること。
   - キーを指定して, レコードの内容と在庫数の参照
   - 商品名, 入庫数, または出庫数の大小比較による検索

設計

1. 第 1 段階の設計 – class の洗い出し –
   データベース と **レコード** を クラスとする。

2. 第 2 段階の設計 – responsibility の洗い出し –

データベース
初期登録 入庫/入庫訂正 出庫/出庫訂正 在庫修正 削除 検索

レコード
初期値設定 入庫数加算/減算 出庫数加算/減算 在庫数修正

3. 第 3 段階の設計 – collabolator 関係の決定 –
   データベースがレコードのサービス (レコード内部の項目の更新/参照) に依存している。

4. 第 4 段階の設計 – CRC カードの整理 –

在庫データベース	
初期登録 (キー) 入庫/入庫訂正 (キー) 出庫/出庫訂正 (キー) 在庫修正 (キー) 削除 (キー) 検索 (キー)	在庫レコード

在庫レコード
初期値設定 (商品名, 入庫数, 出庫数) 入庫数加算/減算 (入庫数/-入庫減算数) 出庫数加算/減算 (出庫数/-出庫減算数) 在庫数修正 (商品名, 入庫数, 出庫数)

(在庫レコード の裏)

在庫レコード のデータの値
商品名 入庫数 出庫数

†3 この部分は, W.S. の当日には Budd の本にあるトランプ・カードの例や ATM での例を用いて説明を行ったが, SEAMAIL に掲載するに当たり著作権の問題などもあるので, 新たに書き下ろした

A コード例 と 実行例

以下に 4. で解いた在庫問題を Little Smalltalk によるコーディングした例とその実行例を示す。

```
"
     Little Smalltalk  example                    1992.12.05   JMAS 福良
"


"
     在庫管理データ・ベース
"
Class ZaikoDatabase
| records |
[
    new
       records <- Dictionary new
|
    upDate:idNumber
       (records includesKey: idNumber)
           ifTrue: [(records at: idNumber) upDate]
           ifFalse:['RecordKey is not found in ZaikoDatabase' print]
|
    nyuuko:idNumber
       (records includesKey: idNumber)
           ifTrue: [(records at: idNumber) nyuuko]
           ifFalse:['RecordKey is not found in ZaikoDatabase' print]
|
    syukko:idNumber
       (records includesKey: idNumber)
           ifTrue: [(records at: idNumber) syukko]
           ifFalse:['RecordKey is not found in ZaikoDatabase' print]
|
    delete:idNumber
       (records includesKey: idNumber)
           ifTrue: [records removeKey: idNumber]
           ifFalse:['RecordKey is not found in ZaikoDatabase' print]
|
    list: condition
       records do: [:x | (condition value:x) ifTrue: [x print]]
|
    confirm: idNumber
       (records includesKey: idNumber)
           ifTrue: [(records at: idNumber) print]
           ifFalse:['RecordKey is not found in ZaikoDatabase' print]
|
    newZaiko
       | newNumber |
       'id?' printNoReturn.
       newNumber <- smalltalk getString asInteger.
       (records includesKey: newNumber)
           ifTrue: ['RecordKey is already exist' print]
           ifFalse:[records at: newNumber put: ZaikoRecord new].
]

"
      在庫レコード
"
Class ZaikoRecord
| goodsname nyuukoSuu syukkoSuu |
[
    new
```

```
    self initialize.
|
initialize
    'goodsName?'     printNoReturn.
    goodsname <- smalltalk getString.
    'nyuukoSuu?' printNoReturn.
    nyuukoSuu <- smalltalk getString asInteger.
    'syukkoSuu?' printNoReturn.
    syukkoSuu <- smalltalk getString asInteger
|
upDate
 goodsname
     <-self promptString:'goodsName' currentValue:goodsname.
 nyuukoSuu
     <-(self promptString:'nyuukoSuu' currentValue:nyuukoSuu) asInteger.
 syukkoSuu
     <-(self promptString:'syukkoSuu' currentValue:syukkoSuu) asInteger
|
nyuuko
 self confirmString:'goodsName' currentValue:goodsname.
 nyuukoSuu<-nyuukoSuu +
   (self transactionString:'nyuukoSuu' currentValue:nyuukoSuu) asInteger
|
syukko
 self confirmString:'goodsName' currentValue:goodsname.
 syukkoSuu<-syukkoSuu +
   (self transactionString:'syukkoSuu' currentValue:syukkoSuu) asInteger
|
promptString:aString currentValue:aValue
  | reply |
   (aString, '[', aValue, ']replace?') printNoReturn.
   reply  <-  smalltalk getString.
   (reply size = 0)
       ifTrue: [^ aValue]
       ifFalse:[^ reply ]
|
transactionString:aString currentValue:aValue
  | reply |
   (aString, '[', aValue, ']add?') printNoReturn.
   reply  <-  smalltalk getString.
   (reply size = 0)
       ifTrue: [^ 0]
       ifFalse:[^ reply]
|
confirmString:aString currentValue:aValue
  | reply |
   (aString, '<', aValue, '>') print
|
goodsname:aString
   goodsname <- aString
|
goodsname
    ^ goodsname
|
nyuukoSuu:aString
   nyuukoSuu <- aString
|
nyuukoSuu
   ^ nyuukoSuu
|
syukkoSuu:aString
   syukkoSuu <- aString
|
syukkoSuu
```

```
        ^ syukkoSuu
|
    printString
        ^ 'goodsName:', goodsname,
          ' nyuukoSuu:', nyuukoSuu,
          ' syukkoSuu:', syukkoSuu,
          ' zaikoSuu:',  (nyuukoSuu - syukkoSuu)
]
```

この Little Smalltalk のソースを実際に MS-DOS 環境で 動かした例を示す。

Little Smalltalk の実行例

```
A> st sample.st     .... 在庫管理の関連のクラスが sample.st に記述されている。

zaiko <- ZaikoDatabase new              初期化

zaiko newZaiko                          在庫レコードの各項目にセットし，挿入
id? 111                                 商品コードのキーに 111 をセット
goodsname? abcde                        商品名 abcde を入力
nyuukosuu? 333                          入庫数 333 を設定
syukkosuu? 111                          出庫数 111 を設定

zaiko confirm:1                         以下の内容を検索
          "goodsName:abcde nyuukoSuu:333 syukkoSuu:111 zaikoSuu:222"

zaiko confirm:2                         このレコードは，無い。以下のメッセージ出力。
          "RecordKey is not found in ZaikoDatabase"

zaiko nyuuko:111                        キー (品番)111 に入庫件数を追加
nyuukoSuu[333]add? -1                   入庫数量を 1 個減算

zaiko confirm:1                         入庫数減算後の内容表示
          "goodsName:abcde nyuukoSuu:332 syukkoSuu:111 zaikoSuu:221"

zaiko list:[:x|x nyuukoSuu>100]         入庫数累計が 100 以上のレコードを表示
                                        この場合は，以下の 1 件のみ表示可能。
          "goodsName:abcde nyuukoSuu:332 syukkoSuu:111 zaikoSuu:221"
```

B 抽象データ型による設計例 –在庫管理による例–

B.1 従来の設計方法の問題点

従来の構造化設計によって製造されたシステムは, 製造時には利点のみ強調されている。欠点の指摘はあまり聞かれない。しかし, 保守段階からながめてみると, 機能別にモジュール化されている場合でも, 保守時に制度変更などが発生した場合, 修正の対象となるモジュールが, 数本に閉じている場合は少ない。一般的には, かなり沢山のモジュールが修正対象となる傾向にある。そして, システムが大きくなればなるほど, 影響範囲の捕捉が困難になりつつある。(富士通 平成3年度LS研分科会報告書「リエンジニアリングによる保守支援」参照)

ここでは, システムの簡単なモデル化を行って説明してみる。

通常, 構造化されている場合は, 機能による分割がなされていると言われている。しかし, この機能分割というのが実は問題をかかえている。それは, 「データを更新している箇所が, 複数箇所に散在していて, 一箇所に限定されていない。つまりデータの更新がバラバラになっている」という事に起因している場合が多いと考えられる。

データの更新がバラバラになっているという事の説明を以下に示す。

従来, 抽象データ型の説明には, スタックとかキューの例がよく利用されているが, 事務処理をよりよくイメージできるように, 在庫管理DBの簡単なモデルで説明する。

B.2 在庫管理の例

商品DBがあり, そのキーは「商品コード」である。その中に「商品在庫数」という項目があるとする。この運用形態を以下のように仮定する。

新しい商品が発生したばあい, この項目は, 初期化されゼロとして登録される。この商品が入荷した時に, 入荷数が加算される。出荷された時には, 減算される。当該商品が, 不要になった時に, 削除される。商品の問合せ時に, 在庫数を表示する。

このような, システムの場合, 一般に以下のような二種類の設計がなされる。

案1 システムの機能分割の概略

1. レコードの初期化
2. 入庫処理
3. 出庫処理
4. 削除処理
5. 問合せ処理

と, 5分割され, DBは, それぞれアクセスする。

案2 システムの機能分割の概略

1. レコードの初期化
2. 入庫処理
3. 出庫処理
4. 削除処理
5. 問合せ処理
6. DBアクセス・ルーチン (参照, 追加, 削除, 置換)

と, 6分割され, DBのアクセスは, 共通化する。

但し, レコードの内容は, 1. から 5. の機能が, 別々に内容をアクセスする。どちらの案でも, 概略以下のようになっている。

1. レコードの初期化

   「商品在庫数」にゼロをセット

   当該レコードの追加 (**案2** の場合は, 追加の指示。以下同じ)

2. 入庫処理

   当該レコードの参照

   「商品在庫数」に入庫数を加算

   当該レコードの置換

3. 出庫処理

   当該レコードの参照

   「商品在庫数」から出庫数を減算

   当該レコードの置換

4. 削除処理

   当該レコードの参照

   当該レコードの削除

5. 問合せ処理

   当該レコードの参照

   「商品在庫数」の参照

このとき, 「商品在庫数」という項目では, 情報量が不足である, との判断がなされ, 「入庫数」と「出庫数」という新しい項目を追加し, 「商品在庫数」はDB上からは削除することになった, とする。

この場合にどのような事態が生じるか?。

まず, DBの切り替えが必須である。

これと並行して, プログラムの修正がなされる。**案1** の場合と, **案2** の場合では, それぞれ, 以下のようになる。

案1 の場合は, 大混乱である。全ての「商品在庫数」を扱っている箇所を, その目的に応じて以下のように修正しなければならない。

1. レコードの初期化
   「商品在庫数」にゼロをセットしている処理の箇所を,「入庫数」と「出庫数」にゼロをセットする処理に置き換える。
2. 入庫処理
   「商品在庫数」に加算している処理の箇所を,「入庫数」に加算する処理に置き換える。
3. 出庫処理
   「商品在庫数」から減算している処理の箇所を,「出庫数」に加算する処理に置き換える。
4. 削除処理
   影響なし
5. 問合せ処理
   「商品在庫数」を参照している処理を,「入庫数」ー「出庫数」を求めた結果を使うように変更する。また, この処理は変更の主旨からすると,「入庫数」と「出庫数」も表示が必要かもしれない。

案 2 の場合, 以下のような方法を用いなければ**案 1** と同じ修正が生じる。

6. DB アクセス・ルーチン で以下のような対応をする事が考えられる。
   物理的には,「入庫数」と「出庫数」しか無いが, このルーチンの利用者には,「商品在庫数」←「入庫数」ー「出庫数」を返却する。このようにすれば, 一見リコンパイルだけですみそうである。
   しかし, アクセス・ルーチンは大変である。レコードの置換指示の場合に,「商品在庫数」が元の値より大ならば, その差を「入庫数」に加算し, 反対に元の値より小ならば, その差を「出庫数」に加算しなければならない。このようにしなければ, 案 1 と同様の修正となる。
   ここで, もし上記のような修正をしたとしたら, 何が問題となるか。今後, システムに変更が発生しなければ良いが, 通常は, 継続して利用されているシステムであれば, 変更が発生するのが一般的であるである。世の中の変化によっては, 思わぬ修正を強いられることが出て来る。
   例えば, このシステムでも,「入庫数」と「出庫数」に分けたという事は, 入庫の取消, 出庫の取消などの数値はどうするかが問題になる可能性がある。正しくは, 入庫の取消は, 入庫からの減算, 出庫の取消は, 出庫からの減算になるであろう。しかし, DB アクセス・ルーチンで上記のような修正をしていると,「商品在庫数」しか判断材料がないため, 入庫の取消は, 出庫への加算を行い, 出庫の取消は, 入庫への加算を行うことになる可能性がある。つまり, もしこのような修正を行うと DB の内容が非常にイビツになってしまう。

このような状況を, 少しでも回避するための方法があるのか?。このための方法として考えられている発想が「抽象データ型」の考え方である。この考え方の基本的なイメージを利用して**案 3** を考えてみると以下のようになる。

案 3　システムの機能分割の概略

1. レコードの初期化
2. 入庫処理
3. 出庫処理
4. 削除処理
5. 問合せ処理
6. 在庫 DB 管理 (入力, 追加, 削除, 置換, 入庫, 出庫, 初期化)

と, 6 分割され, DB のアクセスは, DB 中のデータ項目の扱いを一元化し,「在庫 DB 管理」が処理する。

この案での, 概略を以下に示す。

1. レコードの初期化
   初期化指示
   当該レコードの追加指示 (「在庫 DB 管理」に依頼する。以下同じ)
2. 入庫処理
   当該レコードの参照
   入庫数を渡し, 入庫指示
   当該レコードの置換
3. 出庫処理
   当該レコードの参照
   出庫数を渡し, 出庫指示
   当該レコードの置換
4. 削除処理
   当該レコードの参照
   当該レコードの削除
5. 問合せ処理
   当該レコードの参照
   「商品在庫数」の参照
6. 在庫 DB 管理 (入力, 追加, 削除, 置換, 入庫, 出庫, 参照, 初期化)

内部データ :
商品コード
商品在庫数
役割 :
CASE
初期処理 open 処理
終了処理 close 処理
入力:キー パラメータのキー指定で必要なレコー

ドを読む。
読んだレコードをカレント・レコードとする。
もし, 無ければ, notFound を返す
追加 カレント・レコードを insert
削除 カレント・レコードを delete
置換 カレント・レコードを replace
入庫:入庫数 パラメータの「入庫数」をカレントの「商品在庫数」に加算する
出庫：出庫数 パラメータの「出庫数」をカレントの「商品在庫数」から減算する
在庫数参照:商品在庫数 カレントの「商品在庫数」を返却する
初期化:キー キーのみセットし, 他の項目を総て初期化する
ENDCASE

このようになっていれば, 「在庫 DB 管理」(単なる DB アクセス・ルーチンではない) を変更する事によりほぼ問題は解決する。このインターフェースに「商品在庫数」があれば, その他のプログラムはリコンパイルのみとなる。
もし, 「商品在庫数」をインターフェースに持たないならば, 「問合せ処理」のみ修正すればよい。

「在庫 DB 管理」は, 以下のような修正になるであろう。
(注) 以下の (変) の項が追加または修正となる事項を示す。
内部データ :
商品コード
商品在庫数
(変) 入庫数
(変) 出庫数
役割 :
CASE
初期処理 open 処理
終了処理 close 処理
入力:キー パラメータのキー指定で必要なレコードを読む。
読んだレコードをカレント・レコードとする。
もし, 無ければ, notFound を返す
追加 カレント・レコードを insert
削除 カレント・レコードを delete
置換 カレント・レコードを replace
入庫:入庫数 (変) パラメータの「入庫数」をカレントの「入庫数」に加算する
出庫：出庫数 (変) パラメータの「出庫数」をカレントの「出庫数」から減算する
在庫数参照:商品在庫数 (変) カレントの「入庫数」ー「出庫数」を「商品在庫数」に返却する
初期化:キー キーのみセットし, 他の項目を総て初期化する **(変) (「入庫数」と「出庫数」の初期化が加わる)**
ENDCASE

この**案 3** の発想は, **データの扱いを大切にすること**である。言い替えれば, 「自分が占有しているデータは, 他人が直接いじるな」という発想であり, もっと極端な例で言うと「俺の財布から勝手に金を出すな」となる。(これが, 抽象データ型の基本的な発想である。) つまり, 「データ」と「手続き」の「一体化」を図ることになる。

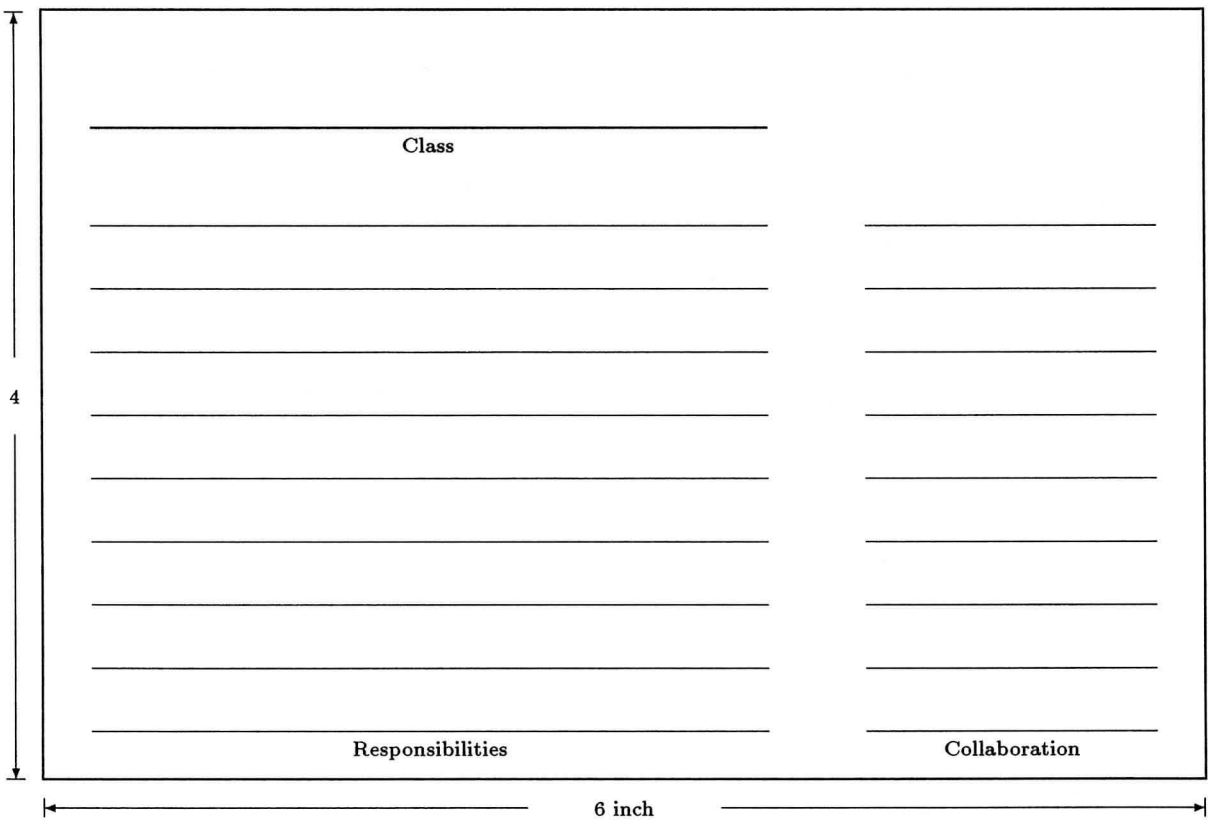
この発想に, 加えて強力な再利用の考え方を導入したものが, オブジェクト指向的な設計方法の考え方の基本である。

尚, この発想を利用して, ホスト環境で設計する場合には, 事務処理プログラムの製造によく利用されている COBOL 言語を用いるのが一般的であろう。
しかし, この言語を用いて製造する場合は, USING でのパラメータの受渡し, 一本のモジュール内での複数 ENTRY の定義, 等々に言語上制約がある。このためかなりの工夫が必要になると思われる。

参考文献

[1] Kent Beck and Ward Cunningham, A Laboratry for Teaching Object-Oriented Thinking, Proceedings of the 1989 OOPSLA(Conference on Object-Oriented Programming Systems, Languages and Applications), reprinted in Sigplan Notices,24(10):1-6,October 1989.
[2] Rebecca Wirfs-Brock and Brian Wilkerson, Object-Oriented Design: A Responsibility-Driven Approach, Proceedings of the 1989 OOPSLA(Conference on Object-Oriented Programming Systems, Languages and Applications), reprinted in Sigplan Notices,24(10):71-75,October 1989.
[3] Daniel Keller, A Guide to Natural Naming, Sigplan Notices,25(5):95-102,May 1990.
[4] Timothy Budd, An Introduction to Object-Oriented Programming, Addison-Wesley Publishing Company,Reading,MA,1991.
[5] Bertrand Meyer, オブジェクト指向入門, アスキー
[6] Timothy Budd, Little Smalltalk 入門, アスキー

CRC カード



3. グループ報告

- グループ 1 報告書 43
- 「若手の会'92」に参加して思うこと (岡本 隆一) 60

- グループ 2 報告書 63
- 「第 10 回若手の会」感想 (伊藤 昌夫) 79
- 第 2 グループの世話役としての感想 (田中 慎一郎) 82

- グループ 3 報告書 85
- はじめて CRC カード手法でオブジェクト指向設計をやってみて (桜井 修) 109
- グループ活動の感想 (豊田 英利) 112

- グループ 4 報告書 113
- 若手 92 の感想 (福良 博史) 135
- 「若手の会」に参加して (後藤 浩) 137

- グループ 5 報告書 139
- グループ活動雑感 (渡邉 雄一) 148

G1 ワークショップ報告書

R－1・グループの活動概要

1．メンバの簡単な紹介と役割分担

氏名	会社名	グループ内での役割	業務傾向／特徴
山口裕史	㈱高岳製作所	グループリーダ R１，最終発表担当	UNIX下でのﾌﾟﾛｸﾞﾗﾑ開発 自社内の開発環境の管理
八百板司芝	㈱ＳＲＡ	グループサブリーダ R２担当	証券・銀行関連のｼｽﾃﾑの 開発及びﾒﾝﾃﾅﾝｽ
沖さとみ	MHIｴｱﾛｽﾍﾟｰｽ ｼｽﾃﾑｽﾞ㈱	R３担当	戦闘機能力評価ｼｽﾃﾑの 設計・開発
佐藤公	日本電気 ｿﾌﾄｳｪｱ㈱	R４担当	機械系CAD/CAM製品の開発
高橋友重	㈱大和ｿﾌﾄｳｪｱ ﾘｻｰﾁ	R５担当	ﾌﾟﾛｸﾞﾗﾑ作成 ｼｽﾃﾑ移行（ｺﾝﾊﾞｰｼﾞｮﾝ）

2．使用した手法

構造化設計　　（ＳＳＡＤＭ方法論）

3．解いた課題（特にグループ独自で拡張した部分に関する説明）

3．1　共通問題の拡張について

共通問題を解いた上で、さらに現実的な事例に即し、以下のような条件を定義した。

・利用者が使用できる端末と職員が使用できる端末を別々に設定することとした。

・利用者の貸し出し期限を１週間に設定することとした。

・利用者の貸し出し可能数を２冊に設定することとした。

・同じ本は、２冊とないこととした。

４．作業プロセスの概要

４．１　初日（９／３）レクチャを受けた後、以下の作業スケジュールを立てた。

・９／３　ＳＳＡＤＭの局面１（可能性調査）及び局面２（システム分析）の作業完了を目標とする。

・９／４　ＳＳＡＤＭの局面３（システム設計）の作業完了を目標とする。

４．２　実際の作業経緯は以下の通り。

１）図書館のモデル設計

２）図書館を取りまく状況を構造化

３）関係データ分析によるＤＢの分類

４）各処理のデータフロー作成（制約事項付）

５）データフローのブレークダウン

６）ユーザが要求するデータの洗い出し（出力データを例として）

７）最後に借りた人の検索についての出力するデータが不足していることに気づく。

８）追加ＤＢの設定（他のＤＢの構造が簡素化）

９）各ＤＢの項目の洗い出し

１０）データフロー再設計

１１）プロセスアウトラインの設計（ＰＯ）

１２）モジュール設計ができる状態まで設計が終了

上記の作業の経緯を，ＳＳＡＤＭの作業ガイドラインにあわせると、局面１は１）～２）、局面２は３）～１０）、局面３は１１）～１２）となると考えられる。

R－２・分析内容

１．着目点、アプローチの概要

１．１　作業方針及び概略

ＳＳＡＤＭで規定されている手順に従って作業を進めていくことにした。但し、課題がそれほど複雑でないこと及び作業時間の短さを考慮に入れ、問題解決に不要と思われるタスクは割愛した。

実際に分析で行った作業概要は以下の通りである。

a　課題の図書館の特徴の検討
　（モデル設計、取り巻く状況の構造化）
b　図書館ＤＢに必要なデータの検討１（実世界の図書館より考える）
　（関係データ分析によるＤＢ分類、ＤＦＤの作成）
c　図書館ＤＢに必要なデータの検討２（課題システムのアウトプットより考える）
　（出力データの洗い出し）
d　ＤＦＤの再検討

１．２　アプローチについて

１．２．１　概略ａ（課題の図書館の特徴の検討）に対するアプローチ

課題システムの図書館がどういうものであるかを想定した。分析というよりも調査にあたる作業である。

決定したことは以下の通りである。

・ある小観光都市の図書館
・利用者は地域住民が主。特に子供、主婦、老人が多い。
・利用者に合わせた分野の書物や、郷土史に関する書物が多い。
・本のほか、レコード、ビデオの貸し出しも行っている。

これにより、（課題とは直接関係ないが）その図書館特有のサービスを考え出すこともできるし、メンバー間の課題に対する意識の刷り合わせもできた。

所要時間　０．５０ｈ

１．２．２　概略ｂ（図書館ＤＢに必要なデータの検討１）に対するアプローチ

１）実世界の図書館に関連のあること（人、ものなど）の洗い出し

思いつくままに箇条書きにした。ここでの成果物は以下の通り。

・箇条書きにされた図書館を取り巻く情報

所要時間　０．２５ｈ

２）洗い出した項目のカテゴリー分けと論理データの構造の作成

今まで洗い出した項目をカテゴリー分けすると４つに分類できた。

その各々のカテゴリー毎に、項目間の関係をはっきりさせるため、ツリー構造を用いて表すことにした。ＳＳＡＤＭでいうＲＤＡ（関係データ分析）の技法である。ツリーで表すようにしたらこの項目はこういう要素から成り立っているのではないかということがよくわかるようになり、さらに洗い出しが進んだ。

ここでの成果物は以下の通り。

・図書ＤＢ（本に関する情報を持つ）の項目の関連図。
・利用者ＤＢ（利用者に関する情報を持つ）の項目の関連図。
・図書館ＤＢ（図書館に関する情報（休館日とか）を持つ）の項目の関連図。
・状況ＤＢ（本の利用状況に関する情報を持つ）の項目の関連図。

所要時間　２．２５ｈ

３）論理データ構造の妥当性のチェック

今回のシステムを作るにあたって、２）の作業で考えたＤＢで実現可能かをチェックするため、に１レベルのＤＦＤを作成した。この作業で、今回の課題には図書館ＤＢが不要ではないかと思われた。幸い足りないＤＢはなかったが、この段階でチェックをするのは良いことである。

ここでの成果物は以下の通り。

・課題１～５に対する１レベルのＤＦＤ

所要時間　４．００ｈ

４）詳細ＤＦＤの作成

３）の作業で論理データ構造が妥当であることが認められたのでＤＦＤを詳細化し、ＤＢに必要な項目をさらに洗い出した。この作業で、今回の課題には図書館ＤＢは不要と決定。

ここでの成果物は以下の通り。

・課題１～５に対する２レベルのＤＦＤ

所要時間２．５０ｈ

１．２．３　概略ｃ（図書館ＤＢに必要なデータの検討２）に対するアプローチ

１）システムアウトプットの項目検討

問い合わせ（３～５）の詳細ＤＦＤを作成したときに、ＤＢからどのような項目を持ってくればよいかがペンディングとなっていたので、問い合わせで如何なる項目を出力すべきかを検討した。

ここでの成果物は以下の通り。

・問い合わせ（３～５）にの出力項目一覧

所要時間　０．５０ｈ

ここで様々な問題が発生した。（内容は「４．作業の手戻り」を参照）

解決までの所要時間　４．５０ｈ

1．2．4　概略d（ＤＦＤの再検討）に対するアプローチ
今までの作業での成果物よりＤＦＤの再検討を行った。

ここでの成果物は以下の通り。

・確定したＤＦＤ

所要時間　1．50h

2．分析で困ったこと（発生した問題点）

2．1　困ったこと、仮定したこと

1）利用者と職員の機能をどう分けるか
課題の条件により検討する必要があった。作業時間との兼ね合いから、利用者端末と職員用端末を分けるという簡単な手段をとることにした。「自分自身で現在借りている本の検索だけは、図書館の利用者でも行える用になっている」という部分については、家族の本を返す場合などに備え、人の借りている本の検索もできるようにした。

2）貸し出し期間、貸し出し可能冊数をどうするか
1週間、2冊までにした。

3）図書館の方式をどうするか
全ての本は貸し出せるか否かのどちらかの状態でなければならないので、館内貸し出しの場合でも必ず職員の作業が必要な閉架方式を採用することにした。
（普段利用している図書館が開架方式で、館内で閲覧する場合には勝手に持ち出せたのでこう考えたのだが、開架方式でも手続きをすれば実現できたと思う。）

4）本の削除をどうするか
手元にある本の削除しかできないようにした。紛失した場合どうするのかも議論されたが、時間の都合で考えないこととした。

5）更新の排他制御をどうするか
これも時間の都合で、職員用の端末は1台で排他制御の必要がないということにした。

6）同一の本の取り扱いについて
本はＩＤにより管理することにした。同一の本でも違うＩＤをふることにしたが、借りる人が要求した本が貸し出し中でも別のＩＤで同じ本があるかもしれない。その場合は貸してあげる方が親切だということになったが、本ＩＤに同一図書には枝番をふったらどうかという話で時間切れ。ユーザに話を聞いたところ、あまり予算がない図書館なので、同じ本を2冊買うよりは、たくさんの種類をそろえたいということで、同一の図書は1冊しかないということにした。

7）実世界の図書館に関連のあることを箇条書きにしていた時、行き詰まった。
ＲＤＡの技法を採用することにより解決。

2．2　発生した問題点
1）ＤＢ上の項目の不足
システムのアウトプットを洗い出した時に論理データ構造を決定した段階のＤＢでは項目が足りないことに気づいた。

2）ＤＢの項目をどう持つか
日本語の分野名をどこに持つか。

3．解釈・意見の相違の扱い
1）分野名をどこに持つか

案1　図書ＤＢ上に持つ
案2　分野ＤＢを新たに作成する

図書ＤＢで本をＩＤで管理し、本ＩＤを分野ＩＤ＋分野別通番とすることにした。すると、分野名と分野ＩＤは同じことを表すのだから分野名をここに持つべきではないのではという意見が出た。全ての本にせいぜい数十種類の分野名を持つのは無駄ではないかということである。また両方もつと、登録時の間違いも多くなるという意見も出た。結局図書ＤＢに分野名を持つメリットは見つからず、分野ＤＢという分野ＩＤと分野名からなるＤＢを新設した。

4．作業の手戻り

4．1　発生した手戻り
ＤＢ構造を決め、ＤＦＤを作成した後にＤＢの不備に気づいた。1．2．3から1．2．2．2に戻った。

4．2　原因
システムのアウトプットを検討し、「特定の本を最後に借りた人の検索」に返却日を出力することにした。ところが、ここまでで検討したＤＢ（最後に借りた人のＩＤを図書ＤＢに持っていた）では出力することが不可能だった。

4．3　対処
貸し出し履歴ＤＢを新設し、返却されたら貸し出しＤＢからそちらに移すことにした。これにより、図書ＤＢから利用者ＩＤの項目がなくなった。

5．評価・問題点

1）処理の都合で制約を設けたことが多かった。

2）成果物の割に時間がかかった。

3）ＤＢの構造はそれなりにできたと思う。

6．そこでの発見など

1）アウトプットの洗い出しを詳細ＤＦＤ作成以前にやればよかったと思う。

2）ＳＳＡＤＭの手順に従って作業できた部分が少なかった。

7．分析結果やワークシート

分析（調査）によって得られた結果（ＤＢ関連図、ＤＢ項目）を以下に示します。

1）ＤＢ関連図

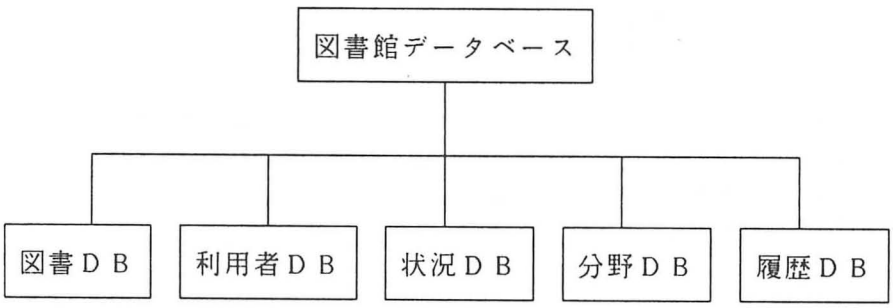


図２－１　ＤＢ関連図

2）ＤＢ項目（属性）

a．図書ＤＢ‥‥本ＩＤ、タイトル、作者、出版日、出版社、登録日

b．利用者ＤＢ‥利用者ＩＤ、氏名、性別、生年月日、住所、ＴＥＬ、登録日

c．状況ＤＢ‥‥本ＩＤ、利用者ＩＤ、貸出日

d．分野ＤＢ‥‥分野ＩＤ、分野名

e．履歴ＤＢ‥‥本ＩＤ、利用者ＩＤ、貸出日、返却日

Ｒ－３・設計

１．そのプロセス

分析段階での成果物としてＤＢの構造（各ＤＢに含まれる論理データの項目を含む）及びほぼモジュールレベルに分割されたＤＦＤが得られた。従って、設計段階ではそれらを基に、本課題で示された５つの要求事項についてそれぞれプロセスアウトラインを作成した。プロセスアウトラインとは時間軸を縦方向にとり、処理及び入出力データの流れを表したものである。

分析段階で確定したＤＦＤが作成されていたこと、また作業時間の短縮化よりプロセスアウトラインの作成をメンバーで分担して行った。その後レビューを行い、一部修正を加えて、最終的なプロセスアウトラインが完成した。

この時点で時間切れとなり、設計作業を打ち切った。従って、

・物理的なデータ設計
・プログラム仕様書の作成

が設計作業の残務である。

所要時間　２．５０ｈ

２．設計で困ったこと（発生した問題点）

ユーザＩ／Ｆについて、操作により入力されるデータ及びリスト（特定の利用者が借りている本のリスト、特定の作者の本に関するリスト等）として出力するデータについては、分析作業でＤＦＤを作成する過程で検討したが、画面の遷移についてはあまり議論を行わなかった。また、分析作業段階でＤＦＤを作成した際に利用者に対する本貸し出し情報をチェックするという処理がもれていたため、利用者が１冊本を借りている状態（貸し出し可能冊数は２冊としていたので、あと１冊しか借りられない状態）で、さらに２冊本を借りたいと申請した場合の貸し出し業務をどの様に行うかということが問題になった。

これまで考えていた貸し出し業務のＤＦＤでは、本１冊１冊に対して利用者ＩＤのチェック及び本ＩＤのチェック、その本の貸し出し状況のチェックを行うように考えていた。この段階で、同時に利用者の貸し出し状況のチェックも行い、その利用者が現在本を２冊借りていたら貸し出し不可能とすることにした。ところが利用者が１冊本を借りた状態で２冊の本を借りたいと申請した場合、この処理方式ではどちらか１冊の本の貸し出し処理を行い、２冊目の本の貸し出し処理を行った時点で貸し出し不可能ということがわかる。ところが利用者は２冊のうちどちらか１冊しか借りられないのならば、２冊目の本の方が読みたいかもしれない。そのような場合、今貸し出し処理を行った本の返却処理を行い、その後で新たに２冊目の本の貸し出し処理を行わなければならない。これではあまりにも使い勝手が悪いということで、貸し出し処理を行う際、利用者ＩＤを入力してその利用者の貸し出し状況をチェックした後、画面にその利用者が借りることができる冊数を表示することにした。そうすればその時点で図書館職員が、
「あと１冊しか貸し出しできないのですが、どうなさいますか？」
と利用者に伝えることにより、無駄な操作がなくなる。

３．解釈／意見の相違の扱い

リストの出力処理を行う際、出力するデータを作成する処理と出力処理の間をどのようなインターフェースにするかということで、メンバー間で解釈の違いがみられた。結局、各処理の単一機能化を図るため、出力処理では加工の終了したデータを受け取り、リスト形式に出力のみ行うということにした。

4．その評価、問題点
（1）分析作業の段階ではかなり大きな手戻りが何度が発生したが、設計作業の段階では殆ど発生せず、分析作業の重要性を確認した。
（2）メンバー間でのシステムに対するイメージを統一するためにも、ユーザＩ／Ｆ（どのようなメニュー画面になるか、どのように画面が遷移するのか等）をどのようにするかということは設計作業に入る前に、できればなるべく早い時期に決定しておく必要があった。

5．そこでの発見など
（1）このフェーズで作成したプロセスアウトラインは、1つの処理に関する入出力データや処理間で受渡しするデータを明示的に示すので、図が複雑にならずわかりやすい。ユーザに提示するには説明しやすく有効であると思われる。
（2）設計作業では、メンバーで作業を分担し、作成したプロセスアウトラインをレビューするという方法をとった。分析作業段階でほぼ処理や入出力データが確定していたこともあり、非常に能率的に作業が進んだ。

Ｒ－４・設計書

トランザクション毎の設計書を以下に示します。

①本の貸出
②本の返却
③本の登録
④本の削除
⑤特定の作者の本、特定の分野の本に関するリストの作成
⑥特定の利用者が借りている本のリストの作成
⑦特定の本を最後に借りた人の検索

① 本の貸出

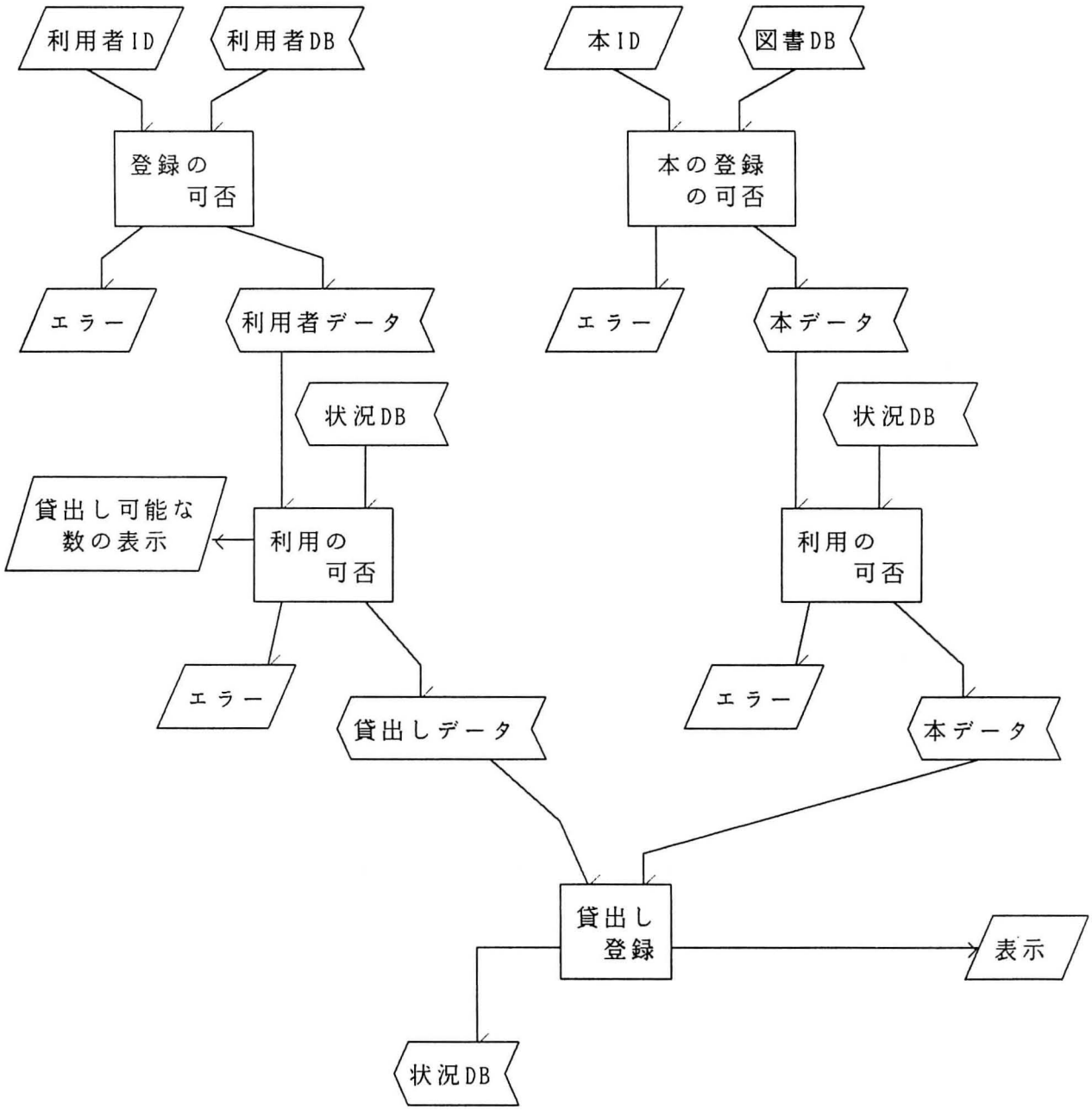


図４－１　本の貸出

※　同一の本は、２冊とない。
※　処理の流れとしては、
　登録可否→利用可否→本の登録の可否→貸出可否→貸出登録
の順に行われるものとする。

２冊借りるとき繰り返し

② 本の返却

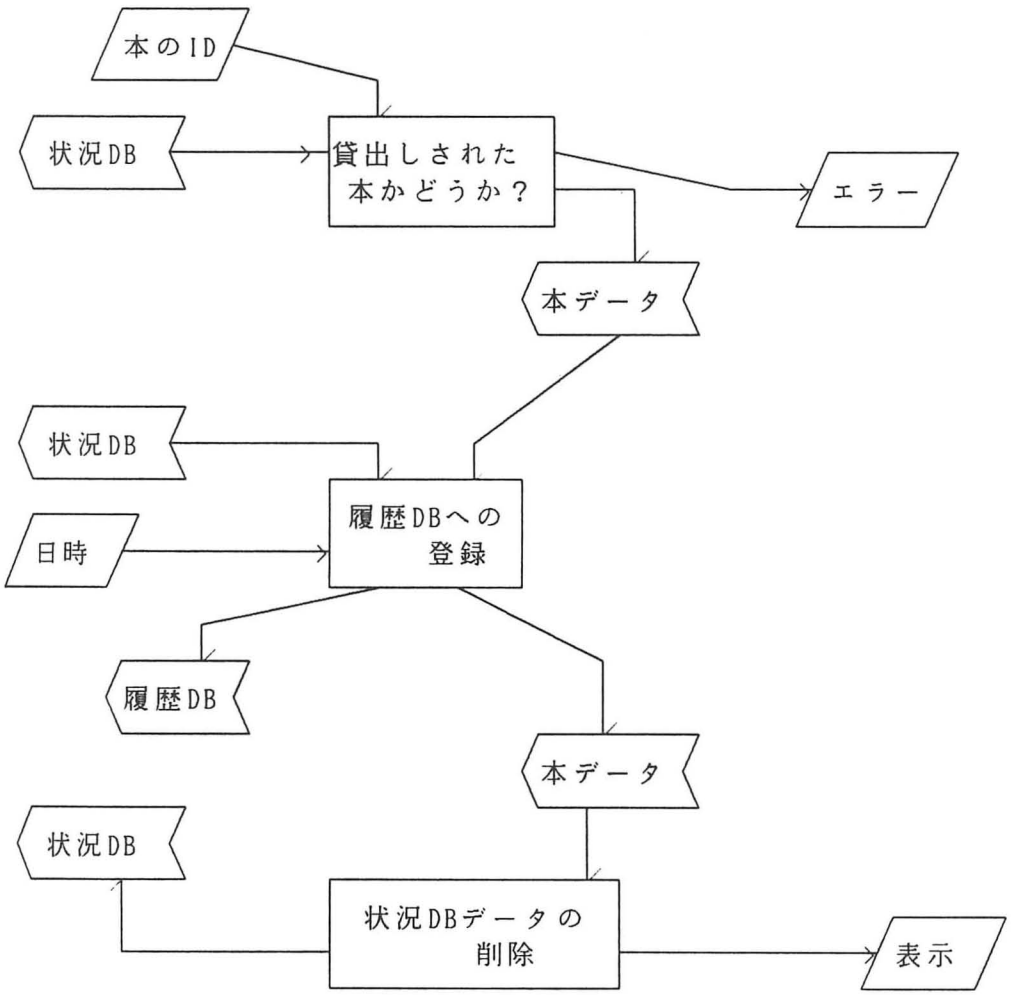


図４－２　本の返却

③ 本の登録

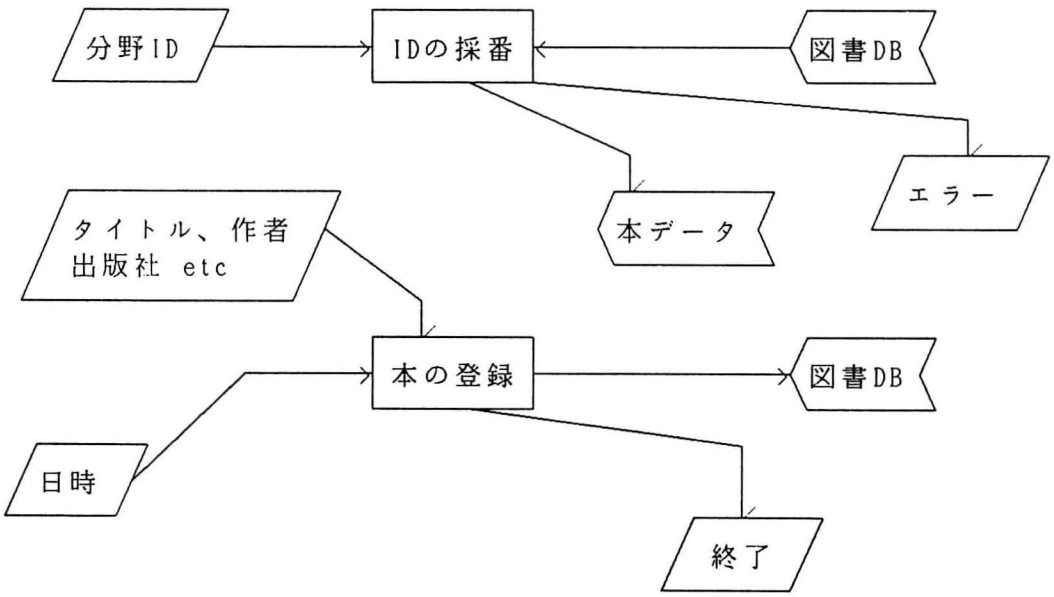


図４－３　本の登録

④ 本の削除

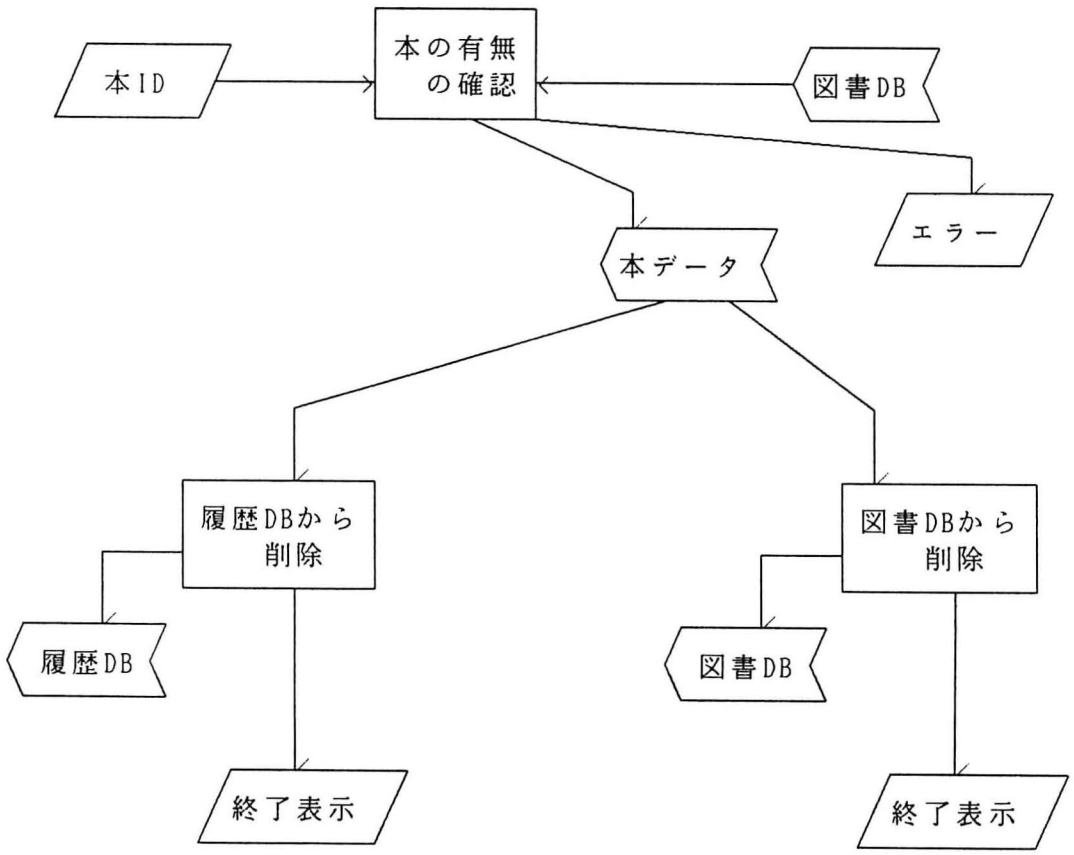


図４－４　本の削除

※　本削除の時のみ履歴DBを削除する。

⑤ 特定の作者の本、特定の分野の本に関するリストの作成

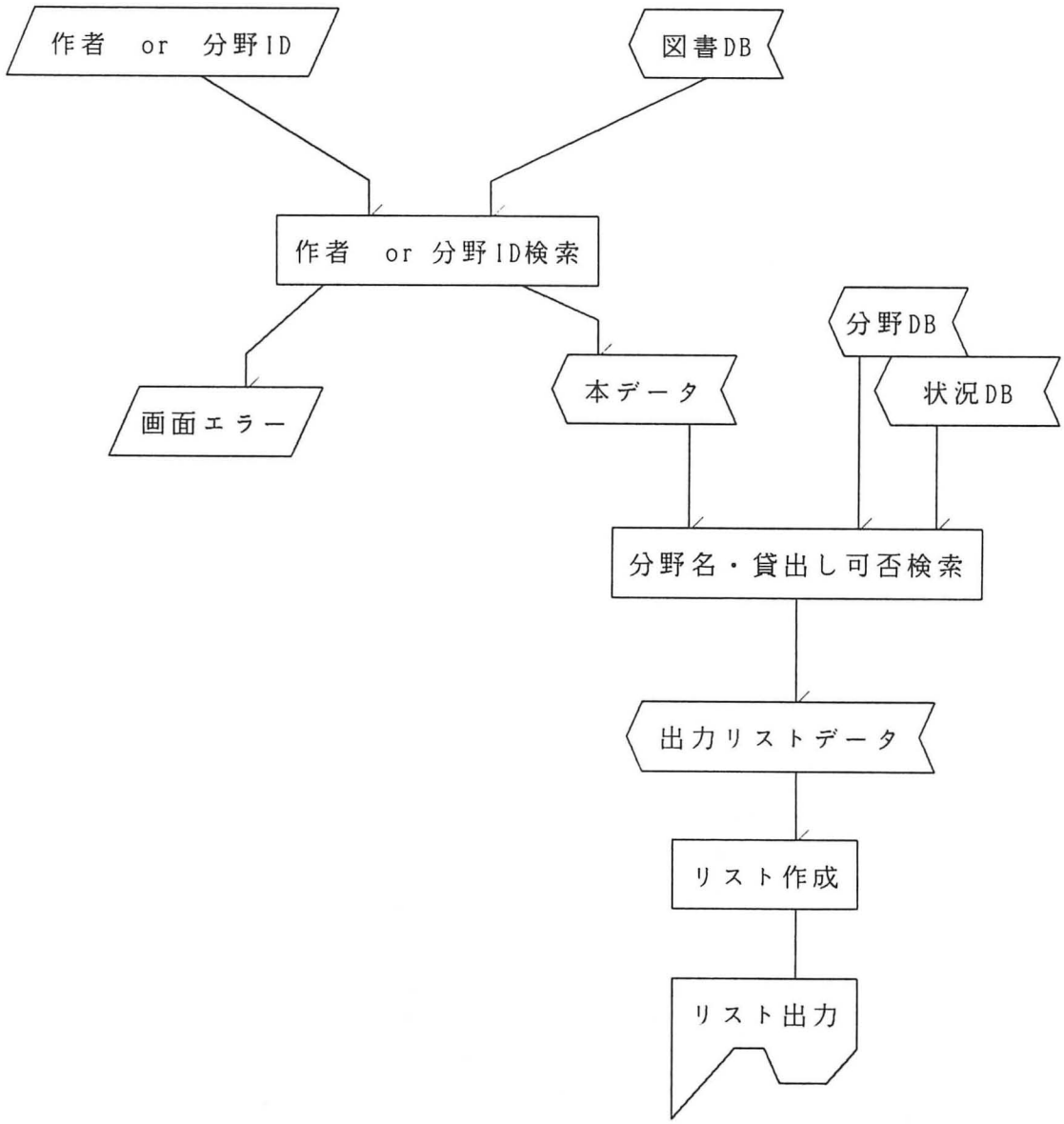


図４－５　特定の作者の本、特定の分野の本に関するリストの作成

⑥ 特定の利用者が借りている本のリスト作成

⑦

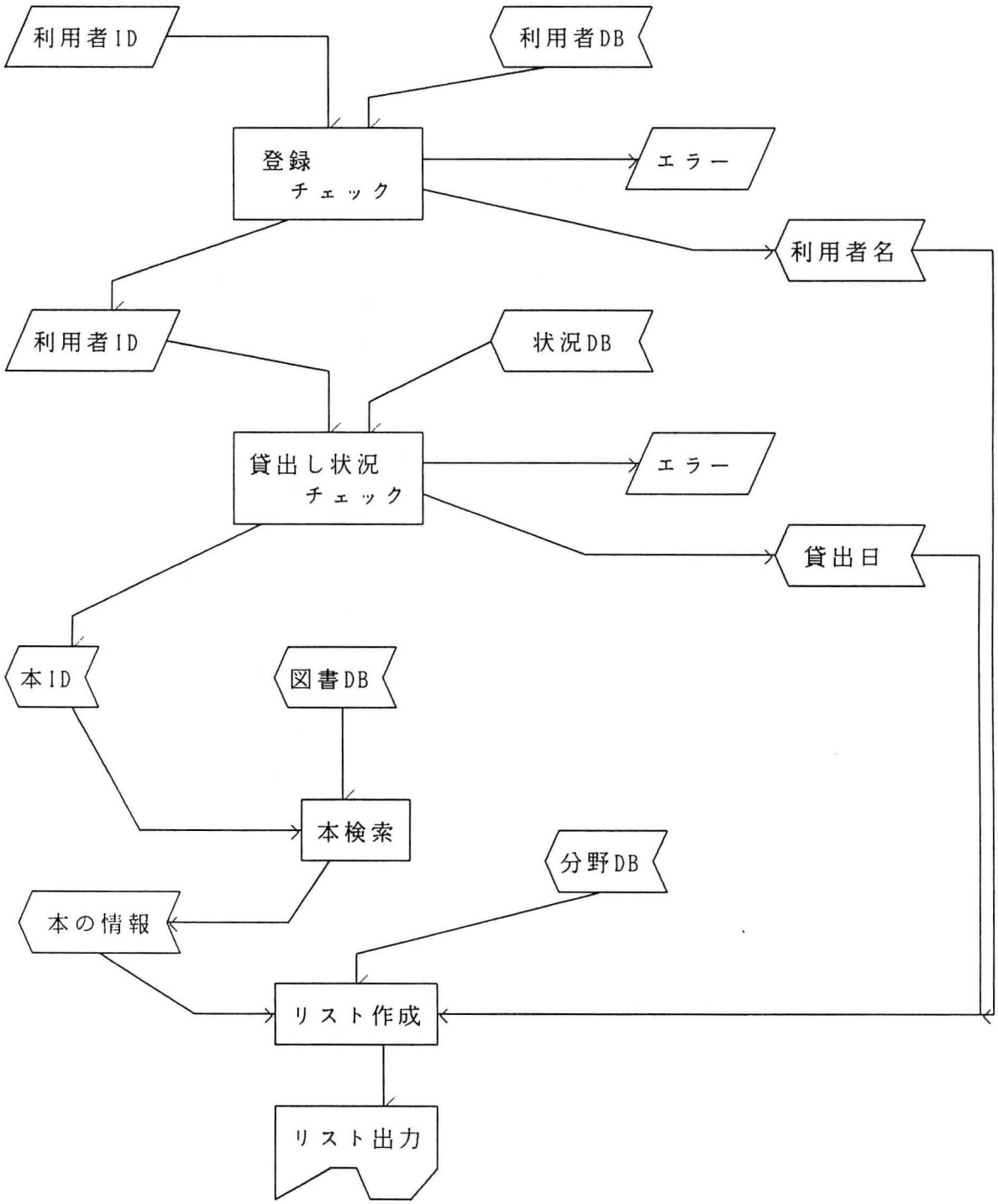


図４－６　特定の利用者が借りている本のリストの作成

⑦ 特定の本を最後に借りた人の検索

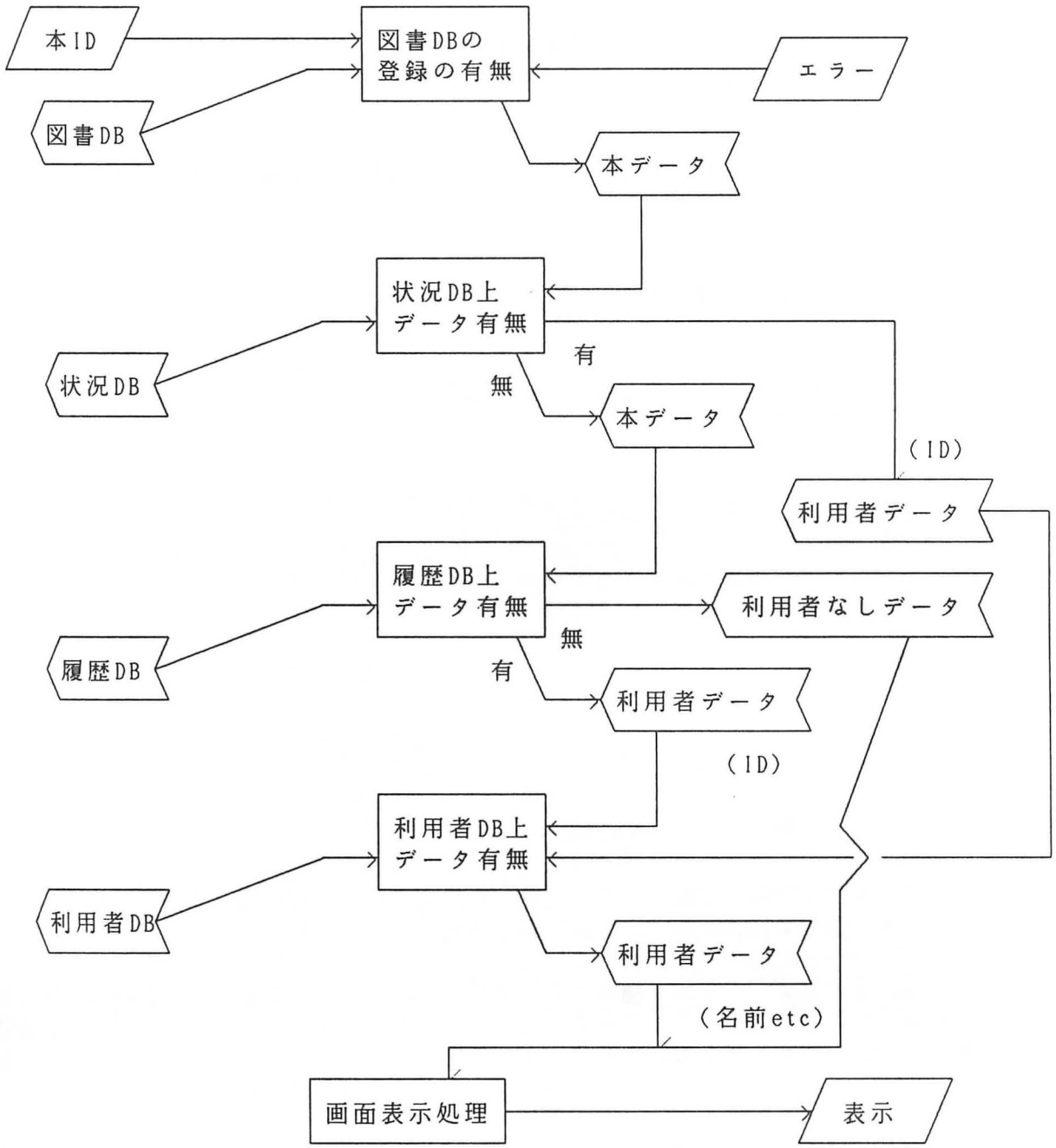


図４－７　特定の本を最後に借りた人の検索

R－5・全体評価

1．使用した手法の全体的評価

1） 利点
・今までの開発環境では、分析の段階で指針になるようなものはなかったが、ＳＳＡＤＭでは、指針が取り決められている。
・アプローチの仕方が決まっているので、分析、設計がやりやすい。

2） 難点
・ユーザインタフェースを上手くデータフローに取り込めなかった。
・ＤＢとＤＢとの相対関係をつかむところがなかった。

2．各自の現場への適用は可能性

・適用は可能だと思うが、調査してみないとわからない。
・実際の業務に、ＳＳＡＤＭを不可能であると思う。しかし、分析、設計等の考え方を取り入れていきたい。

3．その他

・モジュール仕様書の段階までやりたかった。
・「可能性調査」の箇所が、今後役立つと思う。

R－6・グループ活動の感想

今回のグループ活動を行っての感想は以下の通り。

・他の会社の会社の初めて会った人とグループ活動をやるのは初めてなのですが、以外に違和感なく作業ができ、以外に作業は進んだ気がしました。

・会社の内部でプロジェクトを組んで行っている場合と違う点としては、色々な分野のシステムの設計を行ったことのある人の集まりなので、考える発想が色々あり色々な意見がでて面白い気がしました。そして、よい意味で刺激になったような気がしました。しかし、グループとして意見をまとめるのが大変のような気がしました。

・参加する前は、毎日が1日の業務に追われるばかりで設計ということについて、全くというくらい勉強していませんでした。ですから、ワークショップで体験した設計方法論について今回は期間も短かったこともありもう一つ突っ込んだところまで調査出来なかったので、これをよい機会にしてもう一つ突っ込んだところまで調査し、現在私が行っている業務に使えそうなところを見極めどんどん取り入れていきたいと思いました。

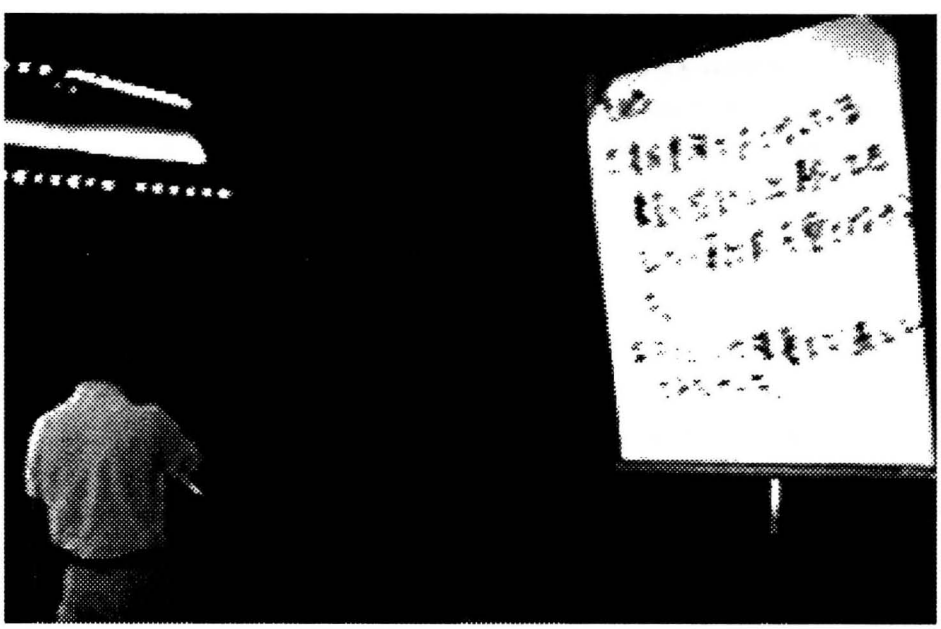
・いろいろな事を考えさせられるワーク・ショップだったと思います。
このワーク・ショップが役に立つかどうかは、今後の自分の勉強にかかっていると思います。

・いろいろな業務に携わる人たちと接することができて、よかった。

・各々の職場、業種により同じ言葉でも微妙に認識が違っていたり、システムに関する常識も違っていて、お互いにわかりあうのに多少苦労した。

・ホワイトボードを用いての作業が中心だったが、途中で気が付いたことをメモすることができたし、作業経過が残って、よかった。

—— 以上 ——



G1 Presentation

「若手の会 '92」に参加して思うこと

岡本 隆一

SEA 主催の若手の会には, 過去, 第 4 回 (1986 年 9 月於, 盛岡) に若手として参加しました。 それから数えて今回の開催が第 10 回だから時のたつのは早いものです。 今度は, おじさん役で参加することになりました。

「設計＋現場＋テクノロジ」のテーマには特に興味がありました。実行委員長を勤めたアスキー社の渡辺氏からプログラム委員としての協力が求められたとき躊躇なく「OK」しました。でも...., 根が不精で, 原稿や資料の準備は一番遅刻するし, 今書いている顛末記の原稿も一番最後になっています。今回のワークショップ関係者には迷惑のかけっぱなしでした。反省はしていますが悪癖は治らないようです。

1 第 1 グループの世話役として

20 代の技術者達が, システム設計・ソフトウェア設計 という仕事にどのような意識で取り組んでいるか...., にはとても興味があります。若手技術者とすれば私も含まれるはずですが, ソフトウェア業界では 30 代も半ばになると 現役作業も行いながら仕事の指導者としての役割を担っている技術者は多いと思います。そのような若手中心の職場の中で, 設計指導に悩む中堅管理職[†1] は少なくないと思います。

岡崎では, 構造化分析・設計の第 1 グループの世話役を担当することになりました。 個人的には「OOA・OOD」に注目していますが, まだまだ勉強段階なので実務経験のある構造化分析・設計でアピールすることにしました。

グループ討論のメンバーは, PC パネルの後のグループ編成まで決まりません。たくさんの 20 代の技術者が岡崎に集まってくれましたが, 「OOA・OOD」に注目している技術者が少ないはずはありません。 第 1 グループの世話役が要らなくなったら神戸に帰ろうか..., と思っているところに 5 人のメンバー[†2]が来てくれました。

第 1 グループには設計のガイドラインの提示として「SSADM」を紹介しました。 構造化分析・設計は現場ではポピュラーな技法なので, あえて技法を紹介することよりも方法論を紹介し, 分析・設計作業を通して行うモデル化の必要性や仕様の分析から設計結果を良しとする意思決定に至るまでのプロセスについてケーススタディして欲しいと考えたからです。

2 なにぶん初対面どうしだから

今回の岡崎では, 分析・設計対象に共通課題が示されました。 ソフトウェア工学の世界では比較的ポピュラーな「図書館問題」ですが, それを各グループのメンバーによる共同作業で消化にあたるので, 果たして, その作業の成り行きはどうなるでしょう。

若手の会の事務局, プログラム委員会から, グループ活動の指針や作業分担の指針等示されていますが, なにぶん初対面どうしだから遠慮があります。グループミーティング初日の 9 月 3 日は円卓を囲んでお見合いのような雰囲気ではじまりました。 一応, 型通り各メンバーの役割分担を済まして中身の検討に進んでもらおうと思いましたが, なかなか遠慮と緊張がゆるみません。

こんなとき, 上手なお手伝いの仕方は...., と考えてもみましたが, 自分も初対面には苦手な方で, とりあえずお茶の時間にしました。

お茶だけ飲んでいてもなんにもならないので, 他のグループの世話役の様子をカンニングしに行きました。 この方法をこの業界では人のテクニックを盗む...., と称するので誰からも責められることはないでしょう。 が, どちらの雰囲気も五十歩百歩の様子。 ウーん, と考え込んで自分の担当グループに戻ってみると先には無かった OA ボードを自主調達して始まっています。 何も悩むことはなかったわけで, フーテンの世話役を決め込むことにしました。

3 昔のこと

何となく初日はこんなだったかな...., と第 4 回の盛岡を思い出しました。世話役がいなくなってもちゃんとやれるし...., 第 4 回に参加した藤本さん@平田機工, 田畑さん@セントラルコンピュータサービス, 正木さん@郵船情報開発, 本田さん@日本情報開発, 大城戸さん@ SRA, 南出さん@中央システム, 糸山さん@八幡コンピュータ...., どうしてるかな...., 覚えてるね。

株) さくらケーシーエス

†1 早く 40 代になって, こんな悩みは 30 代に降りたいなあ...

†2 山口さん, 八百板さん, 沖さん, 佐藤さん, 高橋さん, どうもありがとう。遅くなっていますが資料の約束は忘れていません。必ず送ります。

4 共同で分析する設計するには

フーテンに決めました。 でも用意されたOAボードがグループリーダさんのメモ帳代わりになりかけています。 共同でシステム分析・設計を進めようとしているのは分かりますが, 型から入った様子に写りました。 それぞれが椅子に張り付いて姿勢を正しているので, 各自に普段の仕事の進め方, 様子を聞いてみることにしました。

グループ作業でシステム分析のフェーズを実践する機会には, 各自が, これまで殆ど恵まれなかった...., とのことなので, つい「皆, ペンを持って椅子から立ちなさい!」, 「全員, OAボードの前に立って自分の意見や物の見方をOAボードの上に表明しながら議論しなさい!」とやってしまいました。

一応, 立ってやることの意味と会社に戻ると自分も実践していることはメンバーに伝えましたが, 説教調にやってしまったので, うっとうしいことを言うやつだと思われたかもしれません。

5 SSADMを紹介したひとつの理由

技術屋が要求システムの機能要件や仕様を独善的に決めてしまうことで, 目的システムのユーザとトラブルを起こすことは, この仕事では度々見てきました。 私自身も経験しています。 先のOAボードの利用意識やレビュー等の必要意識を持っていても同じようなトラブルに至ることが度々あります。

私は, このような事態に陥ることを「100%無くすることは困難だ」との意識を持っていますが「かなりしっかりと抑止する」には, 目的システムに必要とされる機能要件や仕様を良しとする最終的な意思決定権は, そのシステムのエンドユーザにあることを忘れないことだ...., との考えを常々とってきました。

SSADMは古いシステム開発の方法論ですが, SSADMでは各システム開発の局面 (工程よりは抽象的な捉え方) において検討された機能仕様, 設計仕様を良しとする最終的意思決定権はユーザにあることを明らかに表明しています。それと, 開発技法は直面するシステム開発の局面や方法論のカスタマイズに合わせて選べば良い。 予てからこの考え方を社内でも説明 (説得?) してきました。 今回の岡崎では, この二つを参加者に伝えて皆からの意見を得たいと考えていました。 グループミーティングが始まるや否や, その自己満足はおよそ達せられました。 後は, 岡崎の最終日まで若手の皆さんに教えられるばかりの立場になり, 本当にフーテンの世話役で通してしまったことを若手の会の皆さんにお詫びしなければなりません。

6 岡崎城に行こう

外の好天が気になります。 薄暗い部屋の中ばかりで議論していても議論が行き詰まります。 会場のレクワールド岡崎で聞きかじって, 岡崎城まで昼飯前の散歩に行こう...., とメンバー全員で繰り出しましたが, 酷暑の下, 行けども行けども岡崎城は遠かった。 とうとうお城にたどり着いたときには (おじさん役は) 顎を出してしまいました。 味噌田楽を食べたあと天守閣で大休止。さすがに帰りはタクシーに乗りましたが, その運転手さんに歩いた距離を聞いてビックリ。(おじさん役は) よれよれで会場に戻って来たので夕方以降もグループミーティングが続くメンバーに 申し訳ない気分になりました。

が, そんな心配は, その日から最終日まで, おじさん自身がしておれば良いことを思い知らされました。

7 若手のパワー

各グループ毎の個性とグループ活動の進み具合に応じて, ディスカッションが始まり終わる時間に多少の違いは出てきます。 折角, 岡崎まで来たんだから第1グループのその日の予定が終わっても他グループの世話役の仕事や他グループの活動も見ておきたい...., と動き始めたつもりなのですが, どのグループも終わらない...., 我が身の分からぬおじさん役は, 本物の若手の絶大なパワーに圧倒されてしまいました。

なんとかついて行こうとして, 初日が3時, 2日目が4時。ひょっとして...., と思ったら3日目はマジで5時まで勉強することになりました。おまけに, 3日目は ルームメイトの塩谷さん@SRAを部屋から締め出したまま寝入ってしまい, 実行委員長が起こしてくれなければグループ活動の最終報告に遅刻していた...., という暴挙におよびかけました。

岡崎は, 我が身が分からぬおじさんには, 我が身を知る良い機会になったわけですが, 穴があったら入っていたでしょう。

8 岡崎の感想

第1グループの活動報告を客観的に見て「まとめに走ったな...., 」と感じたとき, 同時に, まじめに私の話に耳を傾け, 意見を延べてくれた第1グループにもっと密着したアドバイザーとして努めるべきだったかな...., と感じました。ただ, グループ活動に世話役の介入が過ぎると, 結果的にグループの意志に世話役の主観が入り過ぎることも過去の「若手の会」の報告に現れています。 気持ちの迷いを残すことになりましたが, 会社に戻り, 社内の若手と接しながら頃加

減を考えていくことにしました。

「若手の会」岡崎の成果は, 本報告書に記載されたものだけではなく, 3 泊 4 日のワークショップの中で行われた中間報告, 最終報告, SEA 主催のフォーラムにおけるグループ報告, さらに, JSD に関するご指導とフォーマルアプローチに関する取り組みを紹介していただいた超ベテラン技術者の山崎さん@日本ユニシスとの交流等, かけがいのない機会と盛り沢山のプロダクトに実現されました。

実際, 若手と言えども技術・体力の両面でタフな 4 日間は大変だったと思います。 「若手の会」の企画運営側の趣旨を良く理解し, その実施に全面的に協力してくれた若手に心から感謝しています。 また, この会の企画・運営に尽力し, この会に参加できる機会を与えて頂いた実行委員会, プログラム委員会の皆様に心から御礼申し上げます。

Ｇ２ワークショップ報告書

Ｒ－１　グループの活動概要

１．メンバーの簡単な紹介と役割分担

氏名	会社名	グループ内での役割	業務傾向／特徴他
直江　魅音子	ﾈｸｽﾄﾌｧｳﾝﾃﾞｰｼｮﾝ	Ｒ－１担当	ＧＵＩの開発
高巣　英利	ＳＲＡ	Ｒ－２担当	業務アプリケーションの保守
木崎　俊広	ＮＴＴ	Ｒ－３担当	社内ＯＡ化
藤巻　昇	東芝	Ｒ－４担当	ＣＡＳＥの開発
寺井　文康	MHIｴｱﾛｽﾍﾟｰｽｼｽﾃﾑｽﾞ	Ｒ－５担当	ＣＡＳＥの開発

（注）チームリーダは、交替に担当した。Ｇ２でのリーダの役割は単にグループの窓口。全員が対等の立場で、積極的に発言し、考えることを目標とした。

２．使用した手法名

ＳＡ／ＳＤ　＋　α

（注）ＳＡ／ＳＤの表記法等は基本的にペイジ・ジョーンズのものに従った。（「構造化システム設計への実践的ガイド(第二版）」近代科学社、ＩＳＢＮ４－７６４９－０１７９。この本では分析から設計への変換方法、考え方及びシステムの性能予測の方法が、かなり詳細に記述されている。）

３．解いた課題（特にグループ独自で拡張した部分に関する説明）

今回の図書館問題を解くに当たって、問題文中で不足している、仕様分析に必要な情報を見つけ出し、グループ内で議論して仕様を確定した。

本章では、あらためて整理することなく、作業を開始する前に行った質問事項とそれに対して、当初立てた仮定と云う形で記述する。仮定の一部は最終的に変更したものもあるが、基本的には同じである。分析の過程で変更したものについては最終結果も共に示す。

（１）本の貸出および返却

・利用者カード（カードによって本の貸出および返却を行うことにした）に保有させる情報の種類を何にするのか。

　氏名、使用者ＩＤとする。（利用者が手書きで記述するものとする）

・１人何冊まで本の貸出を行えるのか。

　固定として、一人３冊までと仮定する。

・貸出の最大期間は必要ないか。

　固定として、１週間迄とする。（休館日も返却可能とする。）

・貸出期限超過をしたときの処理をどうするか。（例、未返却者リストの出力等。）

　貸出利用状況ＤＢより検索する機能の追加が考えられる。データとして必要な情報はシステムが有しているため、今後の拡張機能とし今回は対応しない。

・本の管理はどこで行うのか。

　本ＤＢで行う。（最終的にはストア「本一覧」で行う。）

・使用者の情報（電話番号、氏名等）の所有はどうするのか。

　データ「使用者」に含む。（仕様には全く触れられていないが、一般的な図書館モデルを考える時に考慮す

べきと考えた。）

・貸出冊数、貸出期限を越えているときは貸出可能とするか。

貸出期限を越えている場合、ペナルティーとして貸出を許さない。

最大貸出数３冊超は貸出不可。管理は、職員が貸出状況ＤＢの照会でマニュアルチェックを行う。

・本の種類によって、貸出期限は異ならないのか。

異ならない。（一般的に、雑誌等が短い場合があるが今回は除いた。）

・本の種類全てが貸出可能か。

全て貸出可能とする。（一般的に、辞書等は貸し出せないが、今回は除いた。）

（２）図書館への本の登録および削除

・パスワード（登録時、職員または利用者）は用いないのか。

（６）システム全般で考慮。使用者ＩＤの帯域使用。（１～１０を職員、それ以外は利用者、パスワードは職員、利用者とも使用する。このコード付番方法については、職員が想定人数を越えた時に、問題が大きいと云う注意がＰＣよりあり、最終的には先頭１文字を専用コードとすることで対応することとした。）

・登録のフォーマット仕様はどうするのか。

本ＤＢの内容とし、具体的なフィールドの構成については、設計時に考慮する。

・パフォーマンスに対する要求は必要ないか。

ペンディングする。余裕があれば、目標を定めトランザクション量の解析を行う。

・「本」のコード体系は不要か。

１０進分類＋ユニーク番号（１０桁）＋サフィックス（２桁）

ユニーク番号はＩＳＢＮを候補とする。サフィックスは、同じ本が何冊あるかを示す。

・貸出の人気ベスト１０等の情報サービスは必要ないか。

採用しない。（但し、ＤＢのジャーナルを用いて、後で実現することは可能）

（３）特定の作者の本、或いは特定の分野の本に関するリストの作成

・分野はどのように捉えれば良いのか。

（２）に示したように図書館１０進分類法を用いる事とする。

・作者の検索に正規表現を認めるか。（例えば、松本清張の時に松＊で検索する）

採用しない。（追加機能として切り出しが容易であり、今回は対象としない。）

（４）特定の利用者が借りている本のリストの作成

・パスワードが必要か。

（２）でも考えたが、ＩＤと含め採用する。

物理的な端末の場所（ある場所の端末しか利用者は使えない）は避ける。

・リスト出力が必要か。

リスト出力を行う。これは、職員は複数利用者の貸出中の本を連続的に確認する必要があるとの判断から付加している。但し、リスト出力は選択とする。

（５）特定の本を最後に借りた人の検索

・この機能は何の為に必要か。

延滞・破損・汚れ等の責任を明確にする為と考える。

（６）システム全般

・ターゲットマシン

この時点では、仮定しない。（汎用機なのか。分散環境なのか。）

ＰＣからのモデリング時には物理条件は考慮しない方が望ましいと云う指摘による。

・システムのバックアップ

ＤＢＭＳ側の機能として、今回のモデリングでは考慮しない。

・画面はどのような作りとするのか。
模造紙上にそのアウトラインを記述した。

4．作業プロセスの概要

まず最初に、「明日から役立つ" 小さな発見" を求めて一美しさの追求」というグループ方針を決めた。その詳細については後ほど触れるが、今回のワークショップを通じて、ソフトウェア開発にすぐにでも役立つことを学ぼう、そして更に美しさも追求しようという意味を持っている。

今回の図書館問題を解くに当たって、実際の作業プロセスのスケジュールを次のように立てた。

（1）仕様の明確化
問題の不明点を洗いだし、その仕様を明確にする。

（2）ＳＡ／ＳＤの理解
ＰＣのレクチャーを受けて、ＳＡ／ＳＤについて理解する。

（3）システム分析
ＤＣＤ(データコンテキストダイヤグラム)、ＤＦＤ（データフローダイヤグラム）、ＤＤ（データディクショナリ）、ＰＳＰＥＣ（プロセス仕様書）を作成する。

（4）システム設計
モジュール分割の後、構造図及びＭＳＰＥＣ（モジュール仕様）を作成する。

今回のワークショップでは、実際には、（3）システム分析の途中までしかできなかった。スケジュール通りに進まなかったのは、設計段階から後戻りを極力防ぐという意味で、仕様分析、システム分析に時間をかけたためである。
つまり、手法の不完全性のために時間がかかったのではなく、意識的に後戻りを行いながら、全体として確実に分析を進めて云ったためであり、最終的なロードとしては前倒しになっていると考える。

Ｒ－２　分析内容

1．着目点、アプローチの概要

グループで実際に行った手順を、その手段、生産物、成果という形で纏める。（括弧内は要した時間を参考に示す。）

なお、各項目で「美しい」と云うキーワードが出てくるが、これは（ア）整理されていて、（イ）バランスが良く、（ウ）必要な余裕が勘案されていると云うことであり、分析の結果に対する理解を容易にし、システムの保守性を高めるために必要な、各フェーズでの判断基準とした。

初日

（1）不明点の洗い出し（4時間）
手段　：　与えられた問題を読み、思いつくまま不明点を洗い出す。
生産物　：　質問票
成果　：　３６項目の不明点を質問票という形で纏めることができた。（一部をＲ１、３章に示した。）

2日目

（2）ＳＡ／ＳＤのレクチャー（１．５時間）
手段　：　ＰＣからＳＡ／ＳＤのレクチャーを受け、各自ノートに纏める。
生産物　：　自分専用のＳＡ／ＳＤノート
成果　：　ＳＡ／ＳＤについて、その記述法を中心にある程度理解することができた。また、各要素毎に細かな注意点を纏めることで、オリジナルのリファレンスが出来上がった。

（3）不明点の明確化（４時間）
手段　：　人間シミューレーション（後述）の結果を模造紙に書き上げる。
生産物　：　新しい質問票、模造紙
成果　：　新たな不明点を見つけることができた。また、システムの概観を想定することができた。

（4）システム機能の明確化（１．５時間）
手段　：　人間シミューレーションを再度行い、模造紙にその結果を書き上げる。
生産物　：　模造紙
成果　：　ユーザの要求を決定した。また、システムの境界を明確にすることができた。

（5）データ及びデータ構造の決定（１時間）
手段　：　３回目の人間シミューレーションを行い、模造紙にその結果を書き上げる。
生産物　：　模造紙
成果　：　システム内で使用されるデータ及びその構造を決定することができた。

（6）システムの最終確認（１時間）
手段　：　質問票と模造紙を照らし合わせる。
生産物　：　なし
成果　：　システムの仕様を確定することができた。また、メンバー内のシステムに対する共通理解を確認することができた。

（7）ＳＡ／ＳＤの理解度を高める（２時間）
手段　：　各人が実際にＤＣＤ及びＤＦＤ０を作成する。
生産物　：　各人の作成したＤＣＤ及びＤＦＤ０
成果　：　ＳＡ／ＳＤに対する理解不足を認識した。

（8）ＤＦＤ（データフローダイヤグラム）０の作成（１時間）
手段　：　メンバー全員で、各人が（７）で作成したＤＦＤ０を検討する。
生産物　：　ＤＦＤ０、九州メジャールール（後述）
成果　：　ＳＡ／ＳＤに対する理解が深まった。また、九州メジャールールによって、美しくシンプルな記述を行うことができた。

（9）ＤＣＤ（データコンテキストダイヤグラム）の作成（０．５時間）
手段　：　（８）で作成したＤＦＤ０をチータ法（後述）を用いて検討する。
生産物　：　ＤＣＤ
成果　：　ＤＣＤを美しく書くことができた。

（１０）ＤＤ（データディクショナリ）の作成（０．５時間）
手段　：　ＳＡ／ＳＤの手法、九州メジャールールに従い、データ及びデータ構造を検討する。
生産物　：　ＤＤ
成果　：　データ（構造）の定義が明確になった。

（１１）ＤＣＤ、ＤＦＤ０の確定（１．５時間）

手段　：　（１０）で作成したＤＤを、（８）、（９）でそれぞれ作成されたＤＦＤ０及びＤＣＤに反映させる。

生産物　：　ＤＣＤ及びＤＦＤ０の改訂版

成果　：　ＤＣＤ及びＤＦＤ０を美しく書くことができた。また、ＳＡ／ＳＤの手法に対する理解が更に深まった。

総活動時間　１８．５時間　（但し、中間発表（準備含む）２時間、最終発表（準備含む）１３時間は含まない。）

2．分析で困ったこと（発生した問題）

仕様の明確化という以外に、ＳＡに関して次の点が困難であった。

（1）名前付け

前章のアプローチの概要中の（７）に於て、各人がＤＣＤ及びＤＦＤ０を作成したが、その時点で、名前の付け方が問題になった。プロセスは名詞で記述してはいけなく、「～を管理する」、「～を処理する」と云う名前を付けてはいけない。データには、動詞や「～データ」等の名前は付けてはいけないと注意を受け、言葉を選ぶのに苦労した。(最初の段階では、「貸出処理」や「利用者管理」と云った名前をつけてしまうことが多かった。)

（2）データのまとめ方

ＤＦＤ０のレベルで、渡すデータの詳細を、不要に見せないでおこうとすれば、フローに示されるデータが、多くの細かなデータの固まりとなる。従って、不要なデータがプロセスに多く渡ることになり、データフロー図の理解を妨げることになる。

従って、適切なデータ構造を選択するには、何度も繰り返して議論する必要があった。

（3）強い時系列的処理の記述

システムの使用者のアクセス権は、処理としては最初の処理となるのは明らかである。ところがＤＦＤは、処理手順を記述するのではなくデータの流れを記述する表記法のため、上手に表現することが出来ない。今回、時系列の流れは別の形で表現するとして、アクセス権のチェックを独立させた（設計書ＤＦＤ０参照）。もう一つの解決策としては、トランザクションセンタとして処理の流れを意識した書き方も出来るが、０レベルで、ターミネータからの入力が、全てアクセス権のチェックを行うプロセスに流入することになり、データの流れを見る時には望ましくない。従って、今回前者の形で記述を行った。

3．解釈／意見の相違の扱い

今回の図書館問題の内容には、いくつかの不明点があり、それらについては、グループ内での議論によって明確にした。以下、それらのうちで主なものを挙げる。

（1）図書館モデルの設定

三角形をイメージしても、イメージする三角形は人それぞれ異なる。ある人は正三角形をイメージするし、ある人は直角三角形をイメージする。では、問題にある「小さな図書館」に対して、メンバーは共通のイメージを持つことができるのか．．．。

図書館名	：	岡崎市立図書館
図書館の構造	：	３Ｆ建て
		１Ｆ　一般図書
		２Ｆ　辞書、専門書
		３Ｆ　倉庫
利用者数	：	１００人／日
在庫数	：	5万冊
利用者の登録	：	２０人／月
職員数	：	１０人
既存の利用者数	：	３０００人
月当たりの本の移動	：	登録　１００冊
		削除　　５０冊

上記に示す、具体的なモデルを設定することにより、メンバー内に共通の図書館イメージを持つことができた。仕様から規定される以上の指定があるのは、ある程度の拡張性を考えたからである。学校の図書館のような余りにも「小さな図書館」を想定するのではなく、最初のモデリング段階では、汎用的なシステムを想定した方が保守フェーズで改変が容易になるというＰＣからの指示でそのようにした。この汎用性に対する立場は、システム分析・定義中、一貫して保持した。

（注）モデル上は上記の数値に深い意味はない。例えば、利用者数が２００人／日になっても、今回の分析結果に対してはＤＤの定義（範囲）が変更になるだけである。

（2）職員と利用者、使用者の定義

「使用者」という言葉を採用し、その意味を次のように決めた。

使用者　＝　職員　＋　利用者

当初、職員と利用者についてははっきりと区分していなかった。一つには、システムの立場に立つ以前にイメージとして職員と利用者は異なると想定してしまうことにあったと思う。ところが、ＤＦＤ０もほぼ完成したと思われた時点で、区分しているはずの職員と、利用者が混乱してきた。

図書館データベースは、端末にＩＤとパスワードを入力することにより、データベースにアクセスできる仮定であった。また、本の管理もＩＤを用いて行う仮定であった。

混乱の例を一つ挙げると、職員が本を借りるといったような行為を行う場合、利用者用のＩＤを職員用のＩＤと別に設けるのかということである。職員が2つのＩＤを使い分けなければならない煩わしさを考慮して、上記の定義を行い、職員には1つの使用者ＩＤを与えることとした。

（3）端末の配置、使用権限、使用法

（2）で端末上で図書館データベースにアクセスすると述べたが、その端末を図書館内にどのように配置するのか、また、端末の使用権限をどのように割り当てるのかという問題が起きた。議論の結果、次の方式とした。

・カウンタに1台（各フロアにカウンタがあるのかについては決めなかった）、それ以外の場所に複数台配置する。
・全ての端末は職員、利用者とも使用できる。
・カウンタの端末は職員専用という意味であるが、利用者も使用できる。

データベースにアクセスするために、まず、使用者ＩＤとパスワードを入力し、メニューを表示させた後、必要な処理を選択する方式にした。利用者には、選択不可能なメニューを反転表示させるなどして、すぐにわかる形にした。職員専用、利用者専用それぞれの端末を用意するという意見もあったが、その場合、端末の使用方法を別に考える必要があるため、上記の形にした。

（4）使用者管理の位置付け

使用者ＩＤやパスワード等を用いてデータベースを使用するということは、それらを管理する機能をシステムに持たせる必要がある。最初、この機能をシステムの内部に持たせるという意見もあったが、システムがいたずらに複雑になることを考慮して、システムの外部に、この機能をサブシステムとして持たせるという形にした。

（5）削除の取り扱い

削除要求がシステムにある場合ＤＢ側では、そのデータを完全に削除するのか、或いはinvalidフラグを設けて、常にデータとして保持し続けるかで、議論になった。後者は、現ＤＢに過去の履歴が残ると云う利点があるが、反面ＤＢのサイズは大きくなる。能力に関する検討のところで考えると云うことでペンディングにしたが、結局時間切れで結論には至らなかった。

4．作業の手戻り
5．その評価、問題点

基本的には、手戻りを前提とした開発法をとった。そういう意味では、予定外の出戻り作業は生じていない。
質問票や模造紙の見直し、或いはつき合わせは全体で4回程度行った。また、ＤＣＤとＤＦＤについては交互に3回程度見直しを行った。

6．そこでの発見など

今回、図書館問題を解くに当たって、最初に問題中の不明点を質問票の形で書き上げる作業から始めた。この作業を行ったのは、「仕事を依頼されたら、質問項目を必ず挙げる習慣をつけよう」という指摘があったからである。要求元のｗｈａｔと開発元のｈｏｗの間には、明確な境界がなく、お互いの共通理解がなかなか得られない。また、要求元の表現、理解不足によって、曖昧、不正確な要求が出てくる可能性もある。そこで、こういった質問票の形で、要求元に提出し、打ち合わせをすることによって、要求元・開発元両者の理解が深まり、また、より明確かつ正確な新しい要求を得ることができる。（但し今回は、質問票を要求分析・定義に用いている。）

美しさの追求、メンバー内の共通理解、作業の手戻りの防止のために、今回のワークショップを通じて我々グループ独自のやり方､ルールを考えた。それらについて、その定義、発端、利点、適用の順に纏めると、次のようになる。

（1）九州メジャールール
定義 ： ドキュメント表記法のメンバー共通の認識を得るための取り決め。
発端 ： ＤＣＤ、ＤＦＤ０を美しく表現したい。
利点 ： 表記が美しくなる。
適用 ： ターミネータへのデータフローは１本の矢印とする。
ストアの名前とデータフローの名前が同一のとき、データフローには名前を表記しない。
ＤＤ内で、データとして既に定義されているデータは、"＜データ＞"という形で表記する。

（2）人間シミューレーション
定義 ： メンバーを関係者等に置き換え、システムのシミューレーションを行うこと。

発端 ： システムのイメージに対するメンバーの共通理解を持ちたい。
利点 ： 実際にシミューレーションを行うことで、イメージしているシステムの全容をメンバーが認識することができる。
適用 ： 利用者、職員、システムそれぞれをメンバーに割り当て、実際に本の貸出、返却等のトランザクションが発生した場合、どのような作業が発生するのかを調べた。

（3）チータ法（熊本水前寺清子法）
定義 ： 作業のプロセスを
「3歩進んで2歩下がる」
のごとく、1歩ずつ確実に進めること。
発端 ： 先の作業プロセスからの大きな後戻りを避けたい。
利点 ： 1歩ずつ確実に進むことにより、大きな後戻りを避けることができる。また、先の作業の見通しがよくなる。
適用 ： 人間シミューレーションを何度か行うことにより、仕様分析を繰り返した。
ＤＣＤ、ＤＦＤ０、ＤＤ１つ１つを何度か検討・修正し、反映させ合った。

以上のやり方、ルールは、ある意味ではシステム開発を行う上で当り前のこと（特に、九州メジャールール、チータ法）だと思う。しかし、実際の作業で果して行われているのだろうか？ 今回のワークショップでは、実際にやってみたということに大きな意味があったと思う。

我々のグループでは、作業を進めるに当たって「模造紙」が、仕様定義・システム分析の段階において、中心的役割を果たした。
模造紙の利用の仕方としては、テーブルの上に書き込む、その後、周りに書き込み済みの模造紙を貼ることによって、以前の考えを眺めながら次の作業を進める、ということを行った。それによって、メモ用紙やホワイトボード単独では得ることの出来ない以下の効果があった。

（ア）数人程度の人数であれば、書きながら議論できる。
（イ）思考プロセスを確実に残すことができる。
（ウ）一度にいくつかの過程を眺めることができる。
（エ）図や表が自由に書ける。色を用いて濃淡を付けることが出来る。

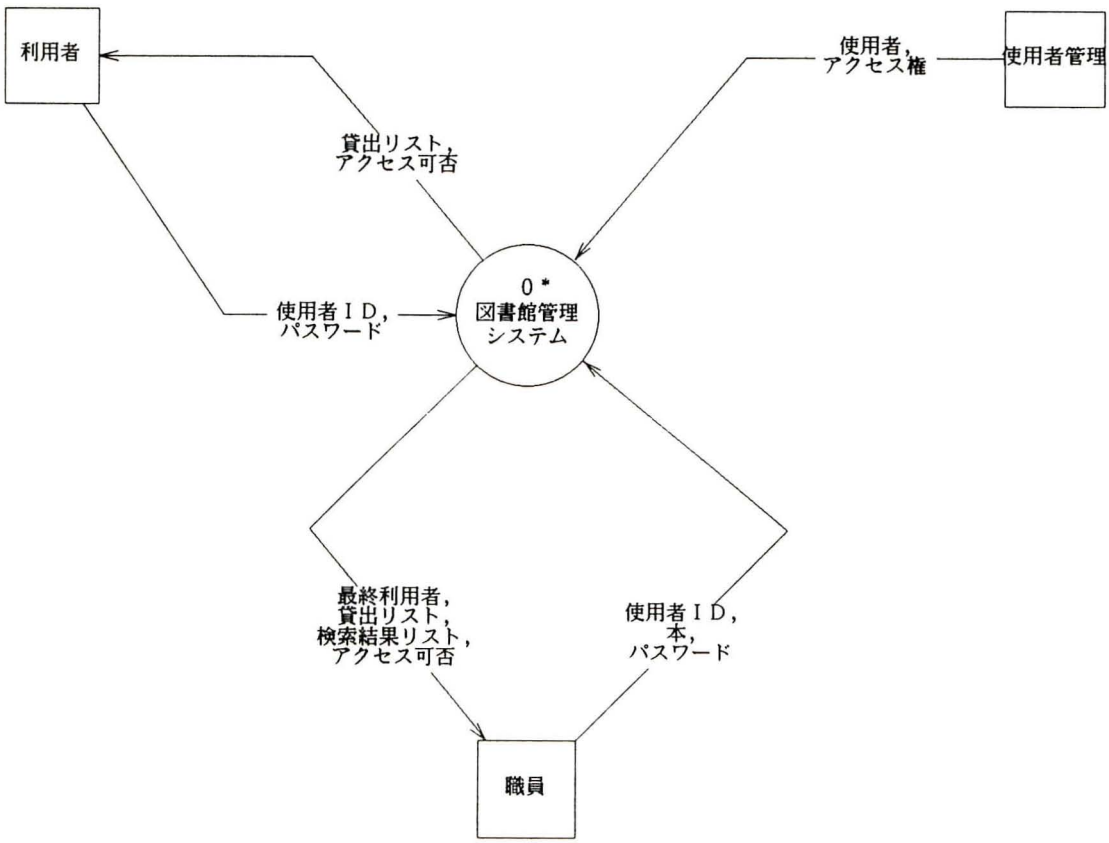
逆に、次の欠点も指摘しておきたい。

（ア）複写がとりにくい。
（イ）書き直しの頻度が高い時には、不適当。（ＤＣＤやＤＦＤ０では、結局ホワイトボードを用いた。）

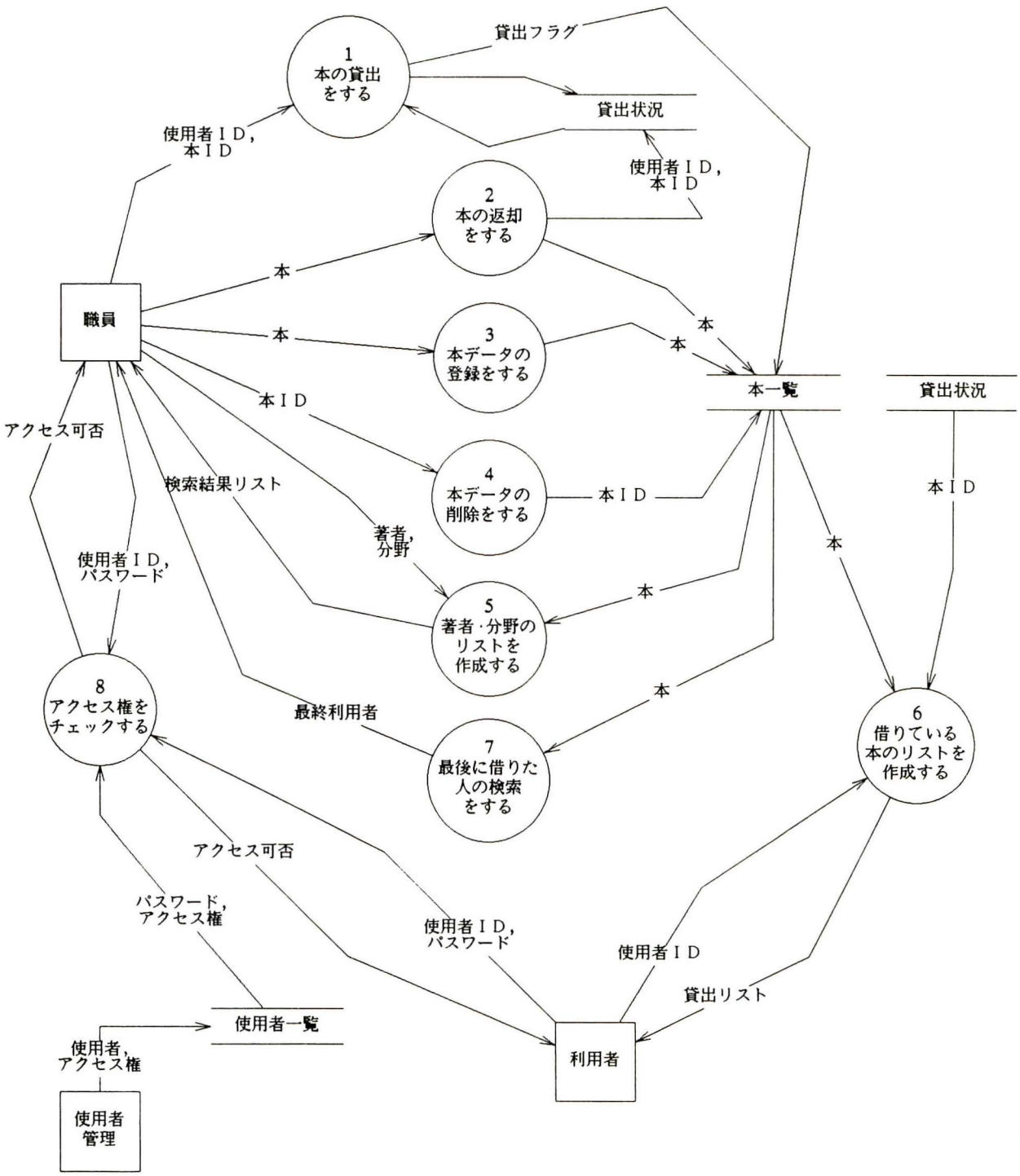
なお、（ア）についてはビデオ入力で可能との指摘があった。

7．分析結果やワークシート

分析結果として得られたＤＣＤ、ＤＦＤ０、及びＤＤを添付する。
これらの図は、若手の会終了後ＣＡＳＥツール、ＳｔＰで書き直している。従って表記法は一部ペイジ・ジョーンズのものとは異なる。下位プロセスの記述がない点、及びストア「使用者一覧」のデータが全て、８の「アクセス権をチェックする」に使用されていない点を除けば、ＳｔＰの持つチェック機能を通した結果、データフロー上及びデータ定義上問題がなかった。
なお、「使用者一覧」で多くのデータを保持しているのは、拡張性を考慮したためで、意識的に追加している。

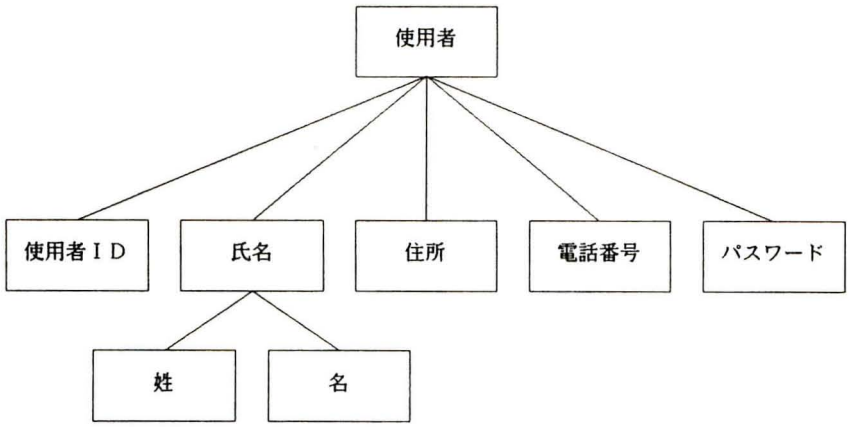

利用者
使用者，
アクセス権
使用者管理
貸出リスト，
アクセス可否
使用者ＩＤ，
パスワード
0 *
図書館管理
システム
最終利用者，
貸出リスト，
検索結果リスト，
アクセス可否
使用者ＩＤ，
本，
パスワード
職員

レベル 0

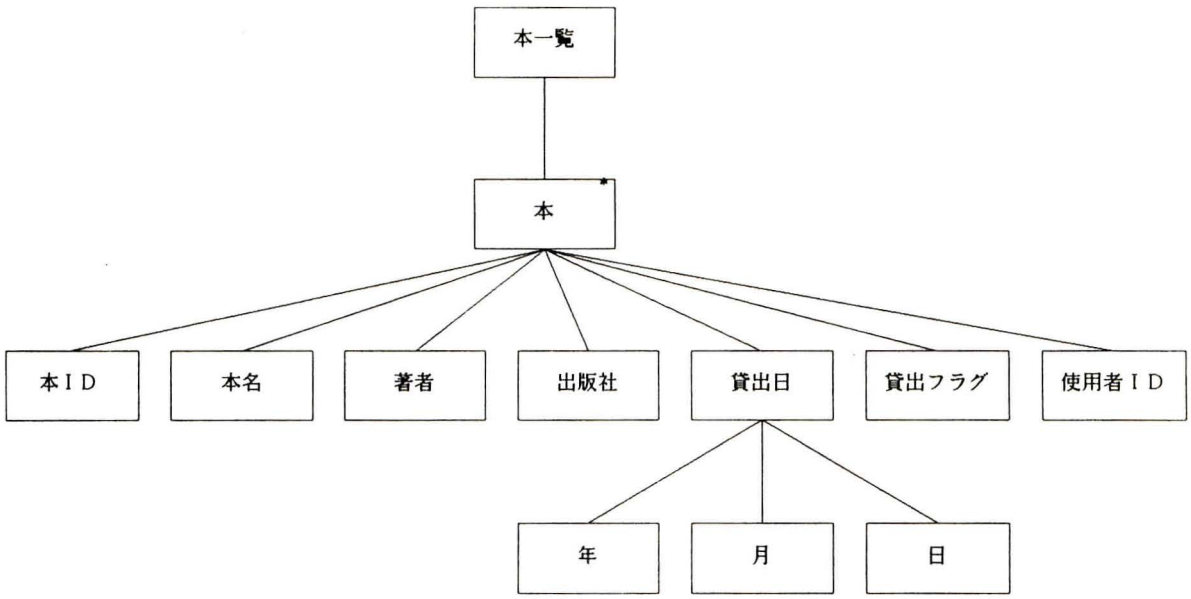


図書館管理システム

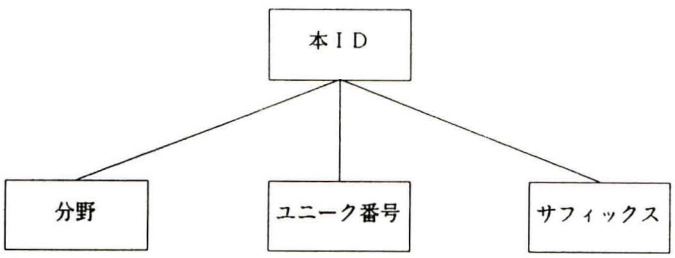
使用者


使用者
使用者ＩＤ
氏名
住所
電話番号
パスワード
姓
名

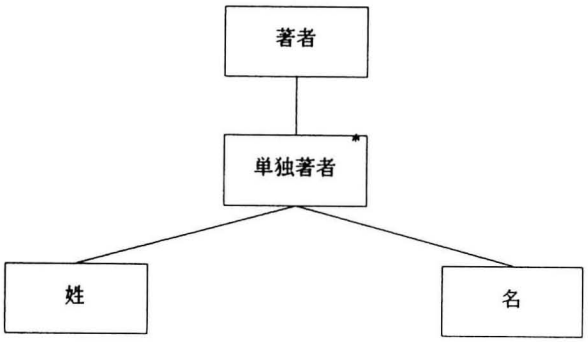
本一覧


本一覧
本
本ＩＤ
本名
著者
出版社
貸出日
貸出フラグ
使用者ＩＤ
年
月
日

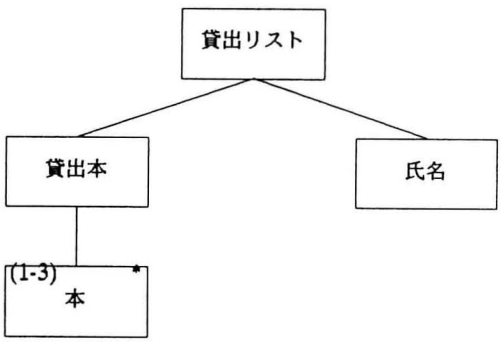
本ＩＤ


本ＩＤ
分野
ユニーク番号
サフィックス

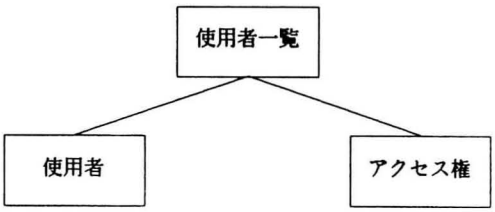
著者



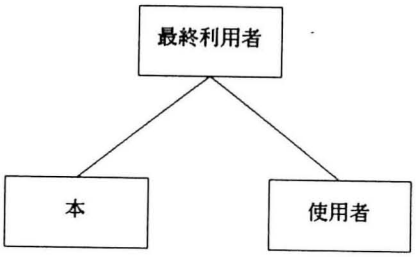
貸出リスト



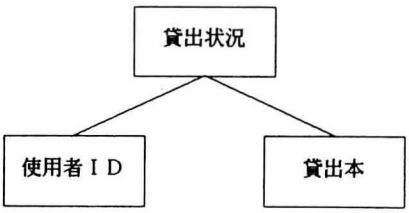
使用者一覧



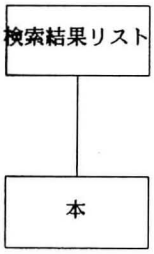
最終利用者



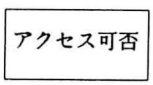
貸出状況



検索結果リスト



アクセス可否



R－3　設計

1．そのプロセス

「R－1　4．作業プロセスの概要」のところで述べたように、今回のワークショップでは、分析段階の途中までしか行うことができなかった。以下、残りの作業のプロセスについて、以下示す。

ワークショップ期間中は、分析段階において、ＤＣＤ、ＤＦＤ０、及びＤＤまで作成した。残された作業は、以下の通りである。

（1）実装よりのＤＦＤに変換する。アクセス許可等に関するプロセスの位置付け、処理の性質の違いによるサブシステムへの分割を考慮しながら、実装寄りのＤＦＤ０に変換する。
（2）（1）で作成されたＤＦＤ０をもとに、プロセスの分割により下位のＤＦＤを作成する。
（3）これ以上分割できないプロセス１つ１つに対して、プロセススペック（ＰＳＰＥＣ）を作成する。ＰＳＰＥＣは、疑似コード、ＩＰＯ／ＰＡＤ／ＨＣＰ等の図式を用いて表現する。

ここまでが、分析段階の作業である。設計作業としては、以下の作業を行うことになる。

（4）ＤＦＤ、ＤＤより構造図を作成する。
ＰＳＰＥＣよりＭＳＰＥＣ（モジュール仕様）を作成する。

R－5　全体評価

1．採用した手法の全体的評価（利点／難点は何か？）

今回採用した手法（ＳＡ／ＳＤ＋α）について、その評価を纏めてみると、次の通りになる。

（1）優れている点

・ＳＡ／ＳＤは、プロセスやデータフローという道具を用いて、上記の成果をある視点から眺めるのに役立ち、整理する過程で、それまで気づかなかった誤りを見つけることが出来る。（システムが外部とやり取りをするデータの多寡、プロセスが処理を行うのに不足するデータ等）
・机上で多人数の議論に向き、また、作業プロセスが残せる模造紙を用いることにより、メンバー間の問題に対する共通理解を持つことができる。
・人間シミューレーションの繰り返し、質問票での仕様の確認、チータ法の採用により、工程の大きな後戻りを防止することができる。

実際に、ＳＡに費やした時間は全体の３０％（５．５時間）であった。この時間でＤＣＤ・ＤＦＤ０・ＤＤがかなり纏まった形で、かつ全員の合意を得て出来上がったのは、前半の模造紙上での突っ込んだ解析が有効であったからと考える。

（2）劣っている点

ここでは、ＳＡ／ＳＤの問題点を記述する。

・システムについて、ある程度の検討がなされていない状態で、ＤＣＤやＤＦＤを記述するのは困難である。
・機能として明確に区分でき、かつデータ的にも分離している要素を同時に表記するのは困難である。

・データが全体に関与し、かつ強い時系列的処理を記述するのが困難である。
・全プロセスがアクセスするストアがある場合、矢印が増え、図としてかなり煩雑になる。
・一段下がった抽象度（例えば、バッチ、リアルタイムという処理の固まりやサブシステムの固まり）を表す標準的な表記法がない。

設計手法は何等かの視点からシステムを見て、表現する以上、それぞれ長所・短所を持ち合わせていると思う。ＳＡ手法を通じて感じたのは、まさに、ある手法を学ぶことによって今までとは異なった視点からシステムを見、そうしなかった場合に気づかなかったことを気づくことが出来たと云うことである。

２．各自の現場への適用は可能か？、問題点と代替手段があるのか？

グループの方針で示したように、我々のグループでは、「明日から役立つ" 小さな発見" を求めて」作業を進めていった。そうすることで、九州メジャールール、人間シミューレーション、チータ法、模造紙の利用と云ったいくつかの明日から確実に役立つ" 小さな発見" をすることができた。

また、分析に用いたＳＡ手法は、それらを通じて得られたシステムへの理解を整理し、誤りを避けるために、非常に有効であることが分かった。また、その短所についてもある程度理解することが出来た。

今回のワークショップで得られたことは、職場にすぐに適用可能なことばかりであった。そういった意味で、今回のワークショップは非常に有意義な時間を過ごすことができたと思う。

Ｒ－６　グループ活動の感想

直江　魅音子

この手のワークショップに参加したのは初めてで、人見知りするという性格上、グループ活動に対してずいぶん不安感があったのですが、蓋をあけてみれば、メンバーがよかったせいか、とても活発な議論が行えたと思います。
私の場合、普段は人とは言葉を交わさない特殊な環境で仕事を行っていますので、複数の人達と一つの問題に対して議論し、いろいろな視点からの意見を聞き、コミュニケーションを深めていくということは、それだけで大変大きな収穫であり、意味のあることでした。
特に私達のグループは、「明日から役立つ" 小さな発見" を求めて」を方針として、いかにすれば、お客様によろこんでもらえるか、美しい設計ができるかを目標に、技法にはあまりこだわらず、自分達でいいと思ったことはどんどん採り入れて、目標に向かって夜遅くまで議論し、みんなで作り上げていったのを感じました。
そこまでグループが一団となれたのも、PCの伊藤さん、田中さんにずいぶんひっぱっていただいたおかげだとおもいます。
ただ、残念なのは、仕様を煮つめるのに時間がとられ、結局設計までは、いけなかったことです。
また、希望と致しましては、SA/SDも含め今回行ったすべての設計法についてのさらに詳しいレクチャーが欲しかったです。
それらを考え合わせますと、現実問題としては無理かもしれませんが、４泊５日くらいあるとよかったと思います。
今回のワークショップはいろんな意味で勉強になりました。機会があれば、また参加させていただきたいと思います。

高巣　英利
・初めて顔を合わせたメンバーにしてはなかなかまとまりがあってよかったと思う。
・仕様の理解に時間をかけて徹底して行ったことは、後の分析・設計にメンバーが同じ理解の上で行えるのでよかったと思う。
設計をする上でも手法の理解をまず徹底して行った方が良いのではないかという基本の基本を感じさせられた。

・我々のグループは慎重過ぎてか、真面目過ぎてか、出来が悪かったのか、とりあえずトータルの時間は他のグループよりかなりかかったと思う。
全てにおいて「他のグループに負けない！」という『気合い』に満ちあふれていたからかも知れない。
・メンバーと自分達の仕事の上での設計以外での苦労や考え方等の意見の交換もたくさん行う時間も欲しかったと思う。

木崎　俊広
・ＳＡ／ＳＤ法の記述方法が理解できた。今後現場でも構造化分析設計を取り入れていきたいと考えています。
・" 明日から役立つ小さな発見" として模造紙を使用し、ユーザニーズをより早くつかみ、設計における後戻り工程をなくしていきたい。
・データの流れに着目し、モジュールに落としていくコンテキストダイヤグラムやダイヤグラム０といった手法を学び取ることができてよかった。今後も大いに参加しスキルを向上させたいと思う。

藤巻　昇
・設計手法にとらわれない、明日から役立つ" 小さな発見" というテーマで、方法論に入る前段階の要求仕様を明確にすることの大切さを実感できた。時間はかかったが、実際に分析フェーズで認識の食い違いが殆ど無く、結局は一番確実だ。
・ＳＡ／ＳＤは分析フェーズの途中までしか適用できなかったが、「図書館問題」のような、事務処理系に向いていると感じた。
・模造紙を使った討論、チータ法、九州メジャールール、人間シミューレーション、どれも実戦的で印象深かった。

寺井　文康
・今回のワークショップに参加して、設計手法に関していくつかのことがわかった。
　１つは、どんなにすばらしい設計手法を知っていても、要求分析がきちんとなされていなければ、宝の持ち腐れになるということ。与えられた問題がわかっていないと、どんなにすばらしい設計手法であっても、その有用性を活かすことができない。
　２つ目は、多くの設計手法を知っておく必要があるということ。多くの設計手法が存在するということは、絶対的なものがなく､対象となる問題に対して、向き・不向きがあるということだから、それを見極める知識が必要である。
　最後に、頭の中で形を変える設計手法のイメージを固定させなければならないということ。あるときは、使いさえすれば、すばらしいものができるという気になるし、また、自分には自分のやり方あるんだ、そんなもの使えるかという気持ちになる。あくまで、与えられた問題を検討、表現する道具であるという、しっかりとした認識を持ちたいと思う。
・ＳＡ／ＳＤに関する多少の知識はあったが、実際使うとなるとどうすればよいのか分からなかった。ＤＣＤから始めて、その後ＤＦＤ０を作り、分割をしていくというトップダウン的なアプローチをするものだと思っていたが、今回、ＤＦＤ０の作成の後、ＤＣＤを作るというやり方をすることによって、ＳＡ／ＳＤに対して、違ったイメージを持つことができた。
・設計に至る仮定で、実際どのようにモジュール分割を行うのかということが知りたかった。そういった意味で、設計段階までやれなかったことが残念に思う。

「第10回若手の会」感想

伊藤 昌夫

1 全般

社内で新人に数値解析の教育をする時, 計算機を使わずに電卓を使うようにしている。そうすると, アルゴリズムによる差異 (計算時間や精度) が体感できる。例えば一階微分方程式に対する解法としてオイラー法使った後に, ルンゲクッタ法を知ると, アルゴリズムに対する "ありがたさ"を強く (時には指の痛みとして) 感じることができる。

それ以上に教えていて嬉しいのは, アルゴリズムの良さを生徒達が自分で発見しその喜びから, 特に抵抗もなく電卓をたたく姿を見る事である。

今回の若手の会では, それと同じような楽しさを感じることが出来た。今回のような方法をとらなければ, より先の所迄進めたのだろうが, 自分ではこれで良かったと思っている。

2 繰返しのアナロジー

自分自身が何等かのプロジェクトを行う場合, 必ず繰返しを重視するスパイラル方式にする。それも, Boehm のスパイラル開発法よりももっと細かい粒度で行う。

一番の理由は, 設計をする必要のない位対象について理解し, 経験を積んでいるのでもない限り, 人間先へ行かないと分からない事が多いと思うからである。そのアナロジとして SOM(Self-Organization Map) のデモをしたのだが, 却って分かりにくかったかもしれない。少し詳しく説明したい。

2.1 SOM の重み変化

SOM 自体はニューラルネットワークのモデルとしては比較的単純な部類に入る。基本的に入力層と競合層の 2 層から構成されるフィードフォワード型のネットワークである。

入力ベクトルに対して, 入力層と競合層間の重みベクトルのユークリッド距離を計算する (図 1)。その距離が最も近い競合層の一つのユニットが勝者となる。

勝者はその近傍 (図 2) と共に, 次式に基づいて重みベクトルを変化させる。

近傍 : $d = d0 * (1 - t/T)$

重みベクトル変化 : $\Delta w = \alpha 0 * (1 - t/T)$

$d0$ は初期値, t は学習回数,

T は総学習回数, $\alpha 0$ は初期値

最初に, ネットワークに対して非常に小さい重みをランダムに与える。ここでは一回ごとに, ランダムな入力ベクトルを与えた時の, 重みベクトルの変化の様子を表している (図 3–1, 2, 3)。

上記の非常に簡単なルールから, 入力ベクトルは, 重みベクトルにマッピングされる。

2.2 考察

重みベクトルの変化の式に注目してほしい。一回あたりの重みの変化は, 回数に従って非線形に減少する。つまり, 正しいマッピングに至るために, 最初は大胆に変化させ, 終了直前では微小な変移しかさせない。

次に, 同時に重みを変化させる近傍の式にも注目してほしい。回数が進むに従って, 共に変化させる近傍の範囲は非線形に減少していく。ある条件 (入力) に対して, その影響を受ける要素 (ユニット) が少なくなっていく事を表す。繰返しの初めのころは, 物理的な位置と関係のないユニットが勝者となり (初期値はランダムであるから), 誤った方向に進むことが多い。しかし, 近傍が同時に移動すると云うことを通じて次第に正しく, 重みベクトルは入力ベクトルを反映するようになる。

もし, 問題領域に対してこのような「誤りを許す繰返しを何度も行う」ことで, 最終的に設計結果を作ることができるとするならば, 我々の設計はより容易なものとなるに違いない。

別にこのことから, 現実の設計過程もそうあるべきと云うつもりはない。ただ, 一般に我々がものを理解しようとする時或いは, 何かを考えようとする時, 考えの振り方と云うのは, 最初は (誤りを許しつつ) 大胆に行い, 後半になるに従って, その振れ幅は小さくするのではないか。そのことは見方を変えれば, 考えを進めるに従って, 自分の考えが誤った方向に行った時に前に戻る率は次第に小さくなると云うことではないか。つまり, 人間が設計をすると云う行為の本質は, ここ (誤りを許す繰返しを何度も行う) にあり, それを理解することで本当に我々に適した設計手法が見つかるのではないかと考えていると云うことである。

そうやって, 全体を作り上げれば或いは全体が分かれば, 後は分割統治のルールを利用して整理することができる。

MHI エアロスペースシステムズ (株)

従って, 今回進め方としては繰返し (メンバは**チータ法**[†1]と名づけた) を重視した。また, そのための道具として模造紙を用いた。別に模造紙が重要なわけではない。可能ならば, 直接電子的なメディアに書き込め, グループで議論する事に越した事はない。しかし現時点では, 模造紙は幾つかの点でワープロやドローイングツールよりも, 全体を考えると云う点に於て良いと云うだけである。

大事なのは, 我々が物事を考え整理する時にどうしているかを考え, そのことに対して効率を上げるにはどうしたら良いかを考える事であり, そのために良いと云う手段は色々と試してみたい。

3 手法について

3.1 SA/SD

分析・設計手法はドメイン (問題領域) に強く依存する。或いは依存すべきであると思う。例えば, 自分が日頃どうしているかを考えると, NeXT 上の Objective-C を用いたアプリケーションを書く時は, オブジェクト指向的なアプローチをとるし, 一般のアプリケーション, 例えば Fortran で技術計算するとなると, 処理の流れと使える技術計算用のライブラリをきちんと押さえるところから始める。それ以外のものについては, SA/SD のデータフロー的な視点から眺める事にしている。

技術を考える時, 何か単一の手法で現実の問題が何もかも解決すると云うのは, その技術が単に未成熟なだけではないかと思う。本来はドメインに合わせて, とりあえず行けそうなやり方があって, それと同時に別の視点からも眺められるような代替手段もあると云うのが, 正常な姿ではないかと思う。

現在, 設計手法に関して整理された状態にない (整理する程ないのかも知れないが)。しかし, SA/SD に関しては実際の多くの適用例と, 多くの細かなノウハウがあり, とりあえずのデフォルト的な設計手法である, と云うのは云い過ぎか。

3.2 オブジェクト指向分析・設計

塩谷さんの最後の纏めの途中で, オブジェクト指向に構造はあるが, 階層はないと発言した。でも良く考えてみると自信がない。

多分, 階層と云う言葉をどう捉えるかと云う認識の違いになるのだと思う。実際, オブジェクト指向にもクラス階層と云う言葉はあるのだから。

オブジェクトレベルでは明らかに各オブジェクトは対等である。

クラスレベルではどうか。クラス階層という時の階層は, 構造化設計の階層 `part_of` の関係とは違い `is_a` の関係つまり, 継承が軸になる。

従って, クラス設計に於て重要なのは, 継承を如何に有効に使うかと云う点にある。例えば, 自分がクラス設計する時の基準の一つに, コントローラ的な要素をクラスから取り除きたいと云うことがある (ここでの議論はいわゆる MVC で表されるコントローラの概念とは違う)。コントローラ的な要素はいわゆる, 継承で結びついたクラス階層を無効にする。つまり, 継承と云うものによって得られる嬉しさを小さくしてしまうと云うことである。

逆に, クラス階層を SA 的な階層で表せるかを考える。無理やり出来ないことはないのだが, それでは, オブジェクト指向の表現力を大いに減じてしまう (軸となる継承が表現できないため)。つまり, オブジェクト指向とは, 抽象概念としてのクラスの関係を, 単一の上下ではなく, 現実世界にあるもう一つの関係 (`is_a`) の意味を損なわずに表現出来ると云うのが, 第一義的な特徴である (別の云い方をすれば, 世の中には多くの他の関係があり, それらを表現する別のパラダイムもあり得る)。

そういった理由から, CRC は平面で議論できるが, SA だと同時に何枚もの DFD を見比べる必要が出てくるのだと思う。

しかし, 巨大なクラス階層を表現するとなると, やはりオブジェクト指向でも問題が生じるはずである。そのために, 設計方法論としての Coad の方法では Subject layer が用意されているし, 環境としての Smalltalk のブラウザにはカテゴリがある。

つまり, 私の発言と云うのは, オブジェクト指向では構造化設計で云う「階層」にとらわれるべきではないと云う意味だったのだが。

3.3 最後に

他のグループの方には色々と迷惑 (例えば大きな声) をおかけしたのではと申し訳なく思っています。また, 実力不足から G2 のメンバに適切な指導ができなかったのではとも思います。

ただ今回メンバとともに 4 日間が過ごせたことは, 私に

---

†1 G2 の報告書を読む上で, この点に付いての解説が必要になる。G2 チームはメンバーの半数が九州出身者で, このネーミングも「繰返し」→「365 歩のマーチ」→「水前寺清子」→「チータ」ということで採用されたのだが, もちろん水前寺が九州・熊本にあるということが, この連想が成立する上での必要条件であった。
「九州メジャルール」と云うのもそれに類したネーミングである。

とってとても素晴らしい経験でした。

ポジションペーパやプレゼンで，設計手法の役割は，色々な実設計から共通部分を抽出したもの，「乾いた」干物の，或いはレシピのようなものと書いた。メンバ達が，会社へ戻って自分の仕事の中で，この設計手法と云う干物に水を与えて，生きた魚を作って欲しいと思う。或いはもう一歩先に進んで自分たちに合った設計手法を編み出してもらいたい。少なくとも，多少は練習をしたのだから。

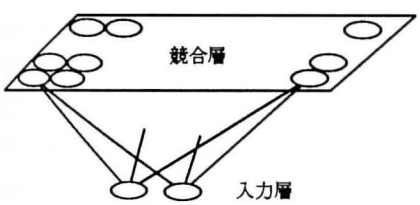


図1. ＳＯＭの構造

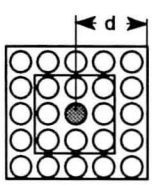


図2. 近傍の選択

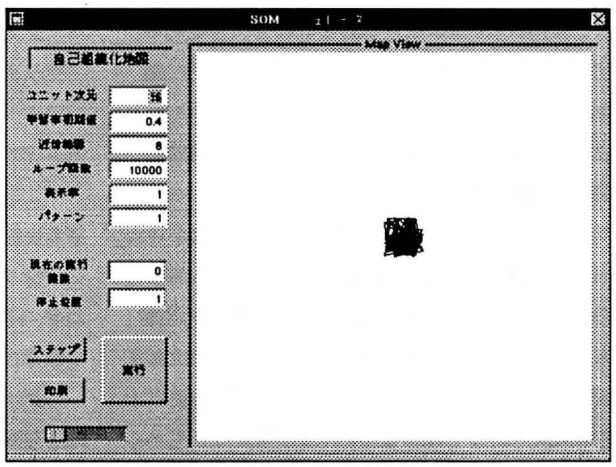


図３－１ 学習開始

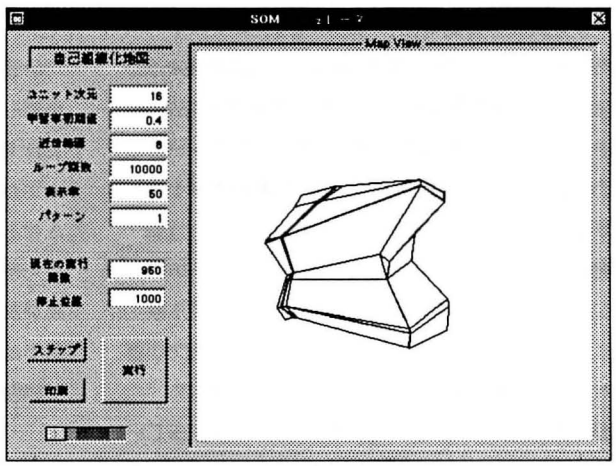


図３－２ 学習途中 （１０００回目）

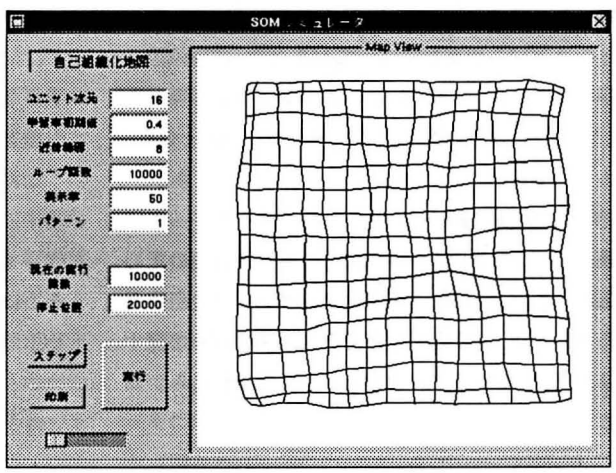


図３－３ 学習終了（１００００回目）

第 2 グループの世話役としての感想

田中 慎一郎

1 はじめに

若手の会にはかつて 1 度だけメンバとして参加したことがある。知る人ぞ知る「八ヶ岳」。 そして今回は「岡崎」。よくよく盛岡には縁がないようである。

といいつつ「岡崎開催」を積極的に進めたのは他ならぬ私でもある。時には西方でも開催したいという思いから合議の上で岡崎の地を選んだのであるが, 「若手の会の主旨」の一つである「横の交流を広げる」を果たすためにはあまりにも辺鄙な場所であった。参加されたメンバは, 必要以上に課題に集中出来たようではあるが, わんこ蕎麦はともかくも, 冷麺とウニ丼には心が残る。小さな反省である。

2 起きてしまったこと

今回の参加メンバのパワーにはすざましいものがあった。「若手」とはかくも偉大なるパワーを持つものであるかと, 我身の老体化を思い知りもした。中でも第 2 のグループの異様なまでの集中力は, かの「セミナー」の洗脳はかく行われるのであろうと思い至るに充分であった。

初日, 講演とグループ分けの後, パーティでグループ毎の紹介を行ったのがよかったのか, どのグループもかなり早くからまとまりを見せていた。パーティの後は適当に散り, 第 2 のグループはホテル内のカラオケ・バーに行った。

店内は他の客がもう 1 グループいるだけというさびしい状況であった。しばらくの間はなんとなく歌いもせずに談笑していたのだが, 話は次第に明日からの作業へと向かって行った。なんと「カラオケ・バー」で不明点を洗い出して質問票にまとめるという作業が始まってしまったのである。

この作業は, 2 ～ 3 曲の歌を交えながらも, 24 時過ぎまで続いた。うち一人はその整理に 3 時までかかったと言う。初日からこうである。

次の日は予定通り伊藤さんに手法のレクチャをやっていただいた後, 今後のスケジュールをたて, 昼飯と親睦のために屋外に出た。昼飯は中華という話になったが, 夕食が中華の予定であったので, 取り合えず行くところまで行って探すことになり, なんと蒲郡まで足を運んだ。既に意気込みが異常である。†1

夕方, 帰宿は予定通りであったが, 作業の方はなかなかそううまくは進まない。悪いことに 何時までも使ってよい大ホールに会場が移ったせいもあり, 異常なノリの第 2 グループは朝食までのみごとな完徹であった。

ここで眠るのが普通である。しかし既に普通ではない。朝食後にスケジュール修正を行い, 岡崎城見学に繰り出した。かなり危ない状況である。そして修正予定通り 13 時に帰宿, 17 時まで仮眠。さらに途中夕食をはさんで夜中の 3 時までまた作業である。†2

グループ作業というものは実に恐ろしいものである。「集団催眠状態だな」と思いつつ, それをきちんと止めない私自身, 充分におなじ状態に陥っていたのだと今は感じている。

3 伊藤さんのレクチャ

伊藤さんのレクチャはもとより, 作業の進め方にはなかなか感心させられた。何も特別なことはしていないのだが, まず自主的にやらせて, 適当にコメントや修正を入れる。はじめから教えてしまっては本当の意味が解らない事柄を納得させるとてもいい進め方である。こういう方法はそれこそ頭で解っていてもなかなか実際に行えることではない。時間がかかる方法なので, 時間が足りなかったことがとても残念ではあるが, メンバだけでなく, 私もとても勉強になった。よい機会を得たと思っている。

4 メンバに対する感想

4.1 直江 魅音子さん

グループ内自己紹介のときから「人見知りがはげしいので,...」と言っていたが, グループ・メンバがよかったのか, すくなくとも今回に限ってはそういう心配はなかった。とても積極的に行動する魅力的な女性である。

なにしろ初日の整理をやっていたメンバが, その作業の途中に「テレビの画面を 9 分割して全ての放送が映る操作(?)」を見つけたそうで, それによるとくだんの「ビデオ」もそこに映るという。が, 最終発表前夜, 発表練習をしようと PC の部屋に集まった時には, なぜか再現ができなかった。

株) SRA　ソフトウェア工学研究所

†1 結局食事はラーメン屋となり, 「ハズレ」であった。成果は水族館でアシカ・ショウを見たことであろうか。

†2 昼食はイタリア料理を予定していたが, 店への到着時刻が早過ぎてやっていない。30 分ほど時間を潰せば良かったのであろうが, 既に全員「何もしないでいる」ことが不可能な状態になっており, 食事ができるところを探し回ってしまった。昼食はハズレまくった。

彼は「ビデオが映るんだ」と力説し，証明のために自分の部屋に見に来いとメンバを誘った。彼女は当然のように皆についていって，残っていた我々 PC に確かに映っていたことを報告してくれたのである。そのくらいひっこみ思案の彼女であった。

4.2 高巣 英利さん

とても元気なノリのいい青年である。熱心で，目立たない(?)形ではあるが，ずいぶん旨くグループを引っ張ってくれていたように思う。

グループの記録係になっていて，熱心にノートを取り，遅くまで整理をしていたようである。感謝している。最後の OHP 書きなど酒でものみながらやればいいのに，呑んではいけないと思っていたのか，2 時ごろぽつりと「ああ酒が呑みたい」と言ったのが印象に残っている。

私は，呑むと作業が出来ないので呑まないでいるだけと勘違いしていて，勧めてもあげなかったのである。考えてみれば「仕事」と思ってやっていたのだから，当然といえば当然の発想。悪いことをした。†3

4.3 木崎 俊広さん

彼もとても元気な青年である。もっとも元気でないとうちのグループは勤まらなかったという話もある。黙っていると，ちょっと恐いかも知れない。でも，笑うといかにも人が良さそうな印象を受ける。

整理の模造紙などは彼がペンをとっているのをよく見かけた。「ちがう」とかまわりから言われながらも，とても積極的に動いていてくれた。「九州流」の発表なども彼が一番板についていたように思う。

しかしあまり直接話をした記憶がない。やはりもっと呑む時間とかをつくって，みんなと話したかったと今にして思う。

4.4 藤巻 昇さん

上記 3 人と比べると若干おとなしい印象を受ける青年である。むちゃくちゃなスケジュールに体力的にまいっていただけという話もある。

おとなしいと言っても，あくまで「上の 3 人に比べれば」という話で，お客さんだったわけではない。九州人 3 人組として，第 2 のグループの特徴付けに一役買っていたことは間違いがない。今回の第 2 グループは，役割分担がうまくできてしまってあまり大きな意見の対立というものがなかった点が逆に残念と思われるくらいであったが，それでもいろいろなタイプの人間とのグループ活動という点で彼にも得るものがあったのではないかと推察している。

4.5 寺井 文康さん

彼も比較的おとなしいタイプに人間に見受けられた。PC の伊藤さんと同じ会社の人だったせいもあり，結構こまめに動いてくれた。体もきつかったのではないかと心配している。また，手法については一番知識があったと思われる。

全員がこれまでの知識から DFD を書き，解っているはずなのに書けない大変さを理解した後，その後の作業を 1 本化するために取り上げられたサンプルは彼のものであった。彼自身は感想として「これまで表記法を学んだ程度で手法を知ったつもりになっていた」と述べているが，実際に書き落とす段階になってからの彼の与えた影響は大きかったのではないかと思っている。

5 全体の感想

さて，PC としての反省点はいろいろある。

やはりまず，参加メンバ間の交流の時間をきちんと確保しなかったことである。これは安易に徹夜をしたのがいけなかった。作業も大事だか，それ以外の交流もとても大事なことだったのだ。すくなくとも公式には 25 時くらいで作業は中断させるように PC 間で事前に徹底させておかなければいけなかったと思う。グループ内でもあまり交流が出来なかったのだから，ほかのグループとはろくに話もしていない。判断ミスである。状況に呑まれるということは恐いものである。

次がやはり徹夜の影響で，最終討論が盛り上がらなかったことである。発表時に質疑を受け付けなかったせいもあり，後から「さあ討論」と言ったときにはもう疑問点を改めて取りあげる気力も記憶も残っていなかったように感じられた。すべては第 2 グループが徹夜をしたことに起因するような気がする。大いなる失敗である。

最後にこれも関連するのだが，まとめ作業が圧迫を受けてしまったことが挙げられる。常々作業も大事であるが，その経過や結果をきちんとまとめることもそれ以上に大事であると考えている私としては，それを充分に指導することが出来なかったことは残念でならない。第 2 グループは伊藤さんが PC であったこともあり，同じ会社の寺井さんが報告書のまとめを一手に引き受ける形になったが，やはり最後まで「共同作業」で行って欲しかったと思う。時間的圧迫からその準備，徹底を促すことが出来なかったことは最大の反省点である。

一口にまとめ，成果といっても，いろいろな側面がある。

†3 PC の方はと言えば，既に「時間外」ですから，当然のように呑みながらあちこちのグループを歩き回っていました。御面なさい。

個人的には若手の会の報告書は「現地でのありのままの成果をとどめるもの」であるべきだと思う。帰社後の清書作業に払われた多大な労力には敬意を表するし, 大いに評価もする。しかし作業の主流を占めた「模造紙」の姿が報告書中にないことや, DFD や DD, アプローチの整理表などの形態が最終発表時の OHP と変わってしまっていることなどは, 本当にグループ成果と言っていいのかと少々疑問にも感じている。これは報告書の「設計結果としての記録」とは別の「イベントに参加したことの記録」を私が求めてしまっているからかも知れない。若手の会に参加したあの「苦労」をもっと「鮮明にとどめるもの」であって欲しいのである。

何もできなかった PC の勝手な思い入れであることは解っている。けちをつけようとしているのではなく, 「共に苦労をした一人として」言わずにはおれなかったということだけ理解して貰いたい。

6 次回のために

今回参加した PC は (メンバもか?), あまりのハードさに次回は遠慮したいといっているので, 是非どなたか後を引き継いで頂きたい。そこで上の反省も交えて, 次回同じような企画をやる方のために気づいたことを整理してみる。参考にして下さい。

6.1 運営について

- 初日の直前打ち合せ/受付/ワークショップ開始にかけては結構忙しいので, きちんと作業分担を決めておかないと, やり忘れなどが出る。最初の講演やレクチャなども, 念のためにテープやビデオを回しておいた方が良い。
- 報告書のフォームや注意書きを作っておいたのはとても良かった。しかし資料説明を充分にしなかったのと, 強制的に読ませる時間を作らなかったので, 効果が半減したような気もする。
- 気づいた点や翌日の修正予定を確認するために, PC ミーティングを毎晩行なうようにするべき。
- 作業外のメンバ交流も重要な課題なので, タイム・キープをしっかりやり, あまり遅くまで作業を行なわないようにするべき。そのためには「遅くまでやるのは無能力の証拠」くらいいってしまった方が良いかも知れない。
- グループ内の交流は元より, グループ間の交流もとれるように努力すべき。例えば, 食事の時間は「別グループとの交流」をはかる方が良い。
- 中間報告の小グループ制もそれなりのメリットがある。但し, 「グループ 1 人」ではなく, 「2 人ずつ」位の方が良いかも知れない。
- 最終討論を盛り上げるためには発表直後の質疑応答が必須。長い発表も良くないが, 質疑の省略や打ちきりは盛り上がりを妨げるようだ。

6.2 テーマについて

- 活動が短期間なので, 同じ課題を扱うと他のグループの作業が理解しやすい。
- 「課題」はもう少し仕様を詰めた状態で提供すべき。曖昧な部分を残しておくことも必要だが, それが多過ぎると本質的な部分にまで進みにくい。
- 「手法」は 3 つくらいにして, 初日に 1 時間位ずつ全員にレクチャができる方が良い。そうでないと, 手法の比較の討論がしにくい。
- グループにおいては「設計において大事なことは何か」を探るのが主題であり, 全体としては「手法間の比較」をするのが主題である。ともすると前者に重きが置かれ過ぎるようなので, 注意した方が良い。

6.3 報告書について

- ほぼ万全の体制で望んだつもりだったが, かなり時間がかかった。事前の計画が重要 (報告書のフォーム, 注意書き, グループ毎の役割分担, 責任者の確認など)。
- 「カメラ・レディ」という言葉の意味を良く説明する必要がある。
- 上下左右のマージンや大まかなフォントの指定もしたほうが良い。
- 写真やビデオを活用する計画を最初から立てておいた方が良い。

7 メンバの皆に

最後に。皆, 今度別のワークショップであった時には, ゆっくり呑みましょう。