SEAMAIL

Newsletter from Software Engineers Association

Volume 9, Number 2-3 June, 1994

目　次

編集部から		1
SIGENV ワークショップの報告		2
SIGENV @ 長野 の概要	渡邉雄一	3
ポジション・ペーパおよび感想		6
共通課題: エレベータの制御		34
SA/SD グループの記録	大塚理恵	37
CRC グループの記録	塩谷和範	42
協力関係図による考察	塩谷和範	47
BON 手法によるもうひとつのエレベータ問題とその解法例	塩谷和範	50
青空の彼方から (2)	伊藤昌夫	54
ラカンとソシュール	伊藤昌夫	59
実世界は机上のようなちっぽけなものではない	M. Weiser (伊藤訳)	61
参考文献	塩谷和範, 伊藤昌夫	63
Call for Papers		
第 12 回 夏のプログラミングワークショップ (若手の会'94)		66
Kunming International CASE Symposium '94		68

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，800 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 40 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，東北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： 中野秀男

常任幹事： 岸田孝一　熊谷章　玉井哲雄　深瀬弘恭　堀江進　山崎利治

幹事： 筏井美枝子　市川寛　伊藤昌夫　臼井義美　大塚理恵　大場充　菊地俊彰　君島浩　窪田芳夫　小林俊明
坂本啓司　杉田義明　武田淳男　田中一夫　鳥居宏次　中来田秀樹　中谷多哉子　西武進　野村敏次
野村行憲　盛田政敏　平尾一浩　藤野晃延　二木厚吉　松原友夫　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)：大塚理恵　小林俊明　人見庸
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　鈴木智　平田淳史
九州支部：平尾一浩
東北支部：菊地俊彰　和田勇

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　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　ダイキン工業　ニコンシステム
ニッセイコンピュータ　ムラタシステム　リコーシステム開発
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　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　オムロン　（以上41社）

ＳＥＡＭＡＩＬ　Ｖｏｌ．９，Ｎｏ．２～３　　１９９４年６月６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　１，０００円（禁無断転載）

編集部から

☆

Vol.9, No.2 ～ 3 合併号をお届けします.

☆☆

この号は, 1 月末に長野で行われた SIGENV (環境分科会) のワークショップ報告の特集号です.

☆☆☆

SIGENV は, SEA の分科会のなかでも, もっとも古い歴史を持つ 1 つです. もともとは, 開発環境一般に関する討論と情報交換の集まりだったのですが, 途中で一時活動レベルが低下しました. そこで, 現在のメンバの方々を中心に, オブジェクト指向その他の新･開発技法とその支援環境に関する地道な勉強会に衣更えして, 今日にいたっています.

☆☆☆☆

一昨年の「若手の会」の企画･運営を担当して, 読みごたえのあるレポートを完成されたことは, みなさんの記憶にも新しいと思います. 今回のレポートは, 内容的に見て, その続編ともいうべきものになっています.

☆☆☆☆☆

次号は, 1994 年度総会報告, SEA のリストラに関するアンケート結果のまとめ etc を予定しています.

☆☆☆☆☆☆

さあ, そろそろ手持ち原稿が底をついてきた. みなさん, よろしく御協力のほどを!

☆☆☆☆☆☆☆

All member - Watanabe @Sashikiri Kyou

SIGENV@長野の概要

渡邉 雄一 *
One of SIGENV 世話人

0. 章前

環境分科会 (SIGENV) が中心になって開催したワークショップに関して報告する。今回は,

(1) テーマ：いま, ソフトウェア開発環境とは?
(2) 場所：長野県東筑摩郡坂北村 坂北荘
(3) 日時：1994 年 1 月 27 日 (木) 〜 29 日 (土)
(4) 参加者：13 名

と規模も小さかったので, 報告書もコンパクトにまとめようとしたのだが, 参加に当って書いて頂いたポジションペーパなども手元にあるので思い切ってそれらも載せることにした。今回の報告書の構成は

- SIGENV@長野の概要
- ポジションペーパ / ワークショップ終了後の感想ほか
- [共通問題] エレベータの制御
- SA/SD グループの記録
- CRC グループの記録[1]
- BOF の話題から[2]
- 参考文献

となっている。

1. ワークショップの背景など

環境分科会 (SIGENV) で, 久々にワークショップを開催した。開催のテーマは, 『いま, ソフトウェア開発環境とは? (− 自然の中で 共通の課題を通して環境を考える −) 』[3]としたが, その狙いは

(1) CASE を中心としたツールの動向を探る
(2) 狭いテーマに束縛されず 自由なディスカッションを中心に行う
(3) 共通の問題を考える etc.

である。特に (1) について, 一時の CASE ブームも バブル経済の崩壊と時を同じくして終ってしまったようである。が, 世界に目を向けると CASE の周辺では様々な動きが活発である。例えば, レポジトリの標準化の周辺では PCTE[4]がある。それらに対して, 国内の大手メーカは際立った動きをしてはいないが, ヨーロッパやアメリカの動きは活発である。

このような動きに追い付くのは無理にしても, 勉強の機会は作って行きたいというのが SIGENV の運営方針の 1 つなので, 企画は以前から練ってはいたものの実際のツールのデモンストレーションなども含めた会合を運営するのは大変で, なかなか実現出来ないでいた。そんな折, PCTE に関しては '91 年から始まった国際会議の第 2 回が '92 年 12 月にフランスで開催され, その様子なども含めて SRA の塩谷さんが情報提供をして下さるとのことで, それに飛びついた[5]。(3) については 岡崎の若手の会で味をしめたので, それを再びというわけである。

---

* (株) アスキー　OA 推進部
[1] 加えて 塩谷さん@SRA が「協力関係図 (Collaboration Graphs) による考察」と「BON 手法によるもう一つのエレベータ問題とその解法例」を書き下ろして下さった
[2] 伊藤さん@MASC が「青空の彼方から」, 「ラカンとソシュール」そして「実世界は, 机上のようなちっぽけなものではない」をまとめて下さった
[3] '課題' は, 個々人のレポートを除いて この報告書では '問題' に統一した。
[4] Portable Common Tool Environment
[5] 残念ながら この部分は今回の報告書には盛り込めていない

2. プログラムの概要

<table>
<tr><th></th><th>1 / 27</th><th>1 / 28</th><th>1 / 29</th></tr>
<tr><td>8:30</td><td rowspan="7"></td><td>朝 食</td><td>朝 食</td></tr>
<tr><td>9:00</td><td rowspan="7">ツールを使っての
共通問題の検討</td><td rowspan="7">全 体 討 論</td></tr>
<tr><td>9:30</td></tr>
<tr><td>10:00</td></tr>
<tr><td>10:30</td></tr>
<tr><td>11:00</td></tr>
<tr><td>11:30</td></tr>
<tr><td>12:00</td><td rowspan="2">受 付</td><td>昼食 休憩</td><td>昼食 休憩</td></tr>
<tr><td>12:30</td></tr>
<tr><td>13:00</td><td>オリエンテーション</td><td rowspan="9">ミニ・パネル討論</td><td rowspan="3">まとめ</td></tr>
<tr><td>13:30</td><td rowspan="4">ポジション発表</td></tr>
<tr><td>14:00</td></tr>
<tr><td>14:30</td></tr>
<tr><td>15:00</td><td rowspan="8">解 散</td></tr>
<tr><td>15:30</td></tr>
<tr><td>16:00</td><td rowspan="4">講 演</td></tr>
<tr><td>16:30</td></tr>
<tr><td>17:00</td></tr>
<tr><td>17:30</td></tr>
<tr><td>18:00</td><td rowspan="4">夕 食
懇 親 会</td><td>夕 食</td></tr>
<tr><td>18:30</td><td rowspan="5">B O F</td></tr>
<tr><td>19:00</td></tr>
<tr><td>19:30</td></tr>
<tr><td>20:00</td><td rowspan="3">自 由 行 動</td></tr>
<tr><td>20:30</td></tr>
<tr><td>21:00</td></tr>
</table>

表 1 . SIGENV-WS@坂北荘 プログラム

2.1 1月27日 (木)

ポジション発表

参加者の自己紹介とポジションの発表を行った。その内容は,「ポジションペーパ / ワークショップ終了後の感想ほか」を参照されたい。

講演

伊藤さん@MASC に, 最近のツールの標準化動向とツールインプリメンタとしてのツールの設計に関する考え方や, その問題点などを話して頂いた。

皆さん, 駅で待ち合わせて旅館のバスで一緒に。長野県といっても、この辺りにはほとんど雪がない。道路の日影に僅かに融けかかった雪が残っているくらいで, 長野＝冬期オリンピック を想像していると, 何か肩透かしをくらったような気分だ。但し, 寒さも東京の郊外のそれとさほど変わり無いように思えるが, 山の中ということもあるのか, ときどきちらちらと雪が舞っている。

宿は村営だが, 風呂場は改装直後できれいだし 我々の泊まる部屋も新館ということでなんら問題ない。夜は '特別料理' ということで季節外れの松茸にありついた。

2.2 1月28日 (金)

ツールを使っての共通問題の検討

あらかじめ課されていた共通問題を SA/SD と CRC の 2 つのグループに分かれて解くための検討の時間とした。

共通問題については「[共通問題] エレベータの制御」を, また作業の結果は「CRC グループの記録」「SA/SD グループの記録」を参照のこと。

ミニ・パネル討論

本来は, “ツールを使っての共通問題の検討” を受けてミニパネルを計画していたが, 実際には分析/設計の作業が続いたので, 行われなかった。

BOF

(BOF とは大袈裟だが) 全員でのフリートーキング。いくつかのテーマが話されたが, 伊藤さんがまとめて下さった「青空の彼方から」以外は, 特に準備をしていたわけでもないので記録はない。

昼は 皆で食事のあとに近くを散策。川は 日本海に向かって流れいるのか自分達の考えていた方向と逆でちょっとビックリ。ここら当たりは秋には山菜採りに人が入るらしく, 所々に注意を促す看板が目につく。

山は, 残った雪と向きだしになった岩肌で 日本画の墨絵でも見ているような感じでなかなかのもの。

夜の BOF の間に誰かが外を見ると一面銀世界。夕方から雪が降りはじめた思っていたが夜の 10 時を過ぎたくらいで, ベランダの手すりには 5cm くらい積もっている。部屋に冷蔵庫が無くて困っていたので, 天然の冷蔵庫に感謝。ビールが良く冷えて議論がはずむ ••• 。

2.3 1 月 29 日 (土)

全体討論

それぞれのグループの分析結果を発表して, ツールの感想などを話し合った。

まとめ

今回のワークショップの運営上の問題点や, 今後の開催にあたってのテーマや意見を話しあった。

昨日までの景色から一変して, まさに辺り一面銀世界。でも, もう雪はほとんど降っていない。9 時過ぎには天気も回復してきた。どうも昨晩はかなりの大雪だったようで, 首都圏などの積雪も凄いらしい。長野市内での大雪で大変だというニュースをテレビでやっているのだから...。
それでも朝のうちには, 宿の前の道路は除雪されているし, 昼までには, 宿の敷地内の道もあらかた除雪がされている。こちらの人達は手慣れたものなのだろう。昼過ぎには マシンを引取に業者が来るが, 高速道路は除雪がきちんとされていて, 名古屋からチェーン無しで走って来たという。

3. この報告書の作成にあたって

この報告書は, 予定ならば 3 月の半ばには完成していたのであるが, 諸般の事情で 4 月末まで ずれこむこととなってしまった。説明が不十分で実際に参加された方でないと意味が通じないと思われるような箇所も無いわけではないが, これ以上 悪戯に発行を遅らせることも出来ないので, 本意では無いがこのような形で皆さんの御目に触れることとなった。何かと至らぬ点が少なくないと思うが, お許しを頂きたい。

[謝辞]
毎度ながら, SEA 幹事の皆さんに謝意を表したい。特に, この企画を提出した昨年 10 月次の幹事会に出席されていた 市川 寛さん (電算) には, 実際にはご多忙で出席頂けなかったにも係わらず, 実質的なローカルアレンジメントを請けて頂いた。今回のワークショップが滞りなく運営出来たのも氏の協力があってのことと, 関係者一同感謝の念に耐えない。また, 今回の目玉の 1 つは, 小規模でも良いから実際に動くものを使いながら議論をしようということであったが YHP 名古屋営業所からは, 最新の HP9000/700 を 2 台無償で[6]貸し出して頂いた。お礼申し上げる。

[6] 輸送費のみ SIGENV で負担

ポジションペーパ / ワークショップ終了後の感想ほか

ここでは, 今回のワークショップの参加者の顔ぶれやポジションペーパなどを掲げる。
まず, 参加者を紹介しよう (表. 1 参照)。

No.	氏名	会社名	グループ
1	山口 登	リクルート	SA/SD
2	南田 邦治	オムロン	SA/SD
3	大塚 理恵	RSK	SA/SD
4	遠藤 浩和	日本電気ソフトウェア	CRC
(4)	(丸山 裕)	(日本電気ソフトウェア)	
5†	渡邊 雄一	アスキー	CRC
6†	田中 慎一郎	SRA	CRC
7†	武田 泰英	アドバンテスト	CRC
8†	後藤 浩	シスプラン	CRC
9†	塩谷 和範	SRA	CRC
10†	伊藤 昌夫	MHI エアロスペースシステムズ	SA/SD CRC
11	浅野 友広	MHI エアロスペースシステムズ	SA/SD
12	直江 魅音子	ネクストファウンデーション	SA/SD
13†	桜井 修	池上通信機	SA/SD

表 1 . SIGENV-WS@坂北荘 参加者一覧

なお, 丸山さんは急遽都合により参加出来なくなったので, 代りに遠藤さんが出席されたが, ポジションペーパは御両人のものを掲載してある。
また No.(整理番号) の右肩に † が付いているのは SIGENV のメンバであることを示す。
グループは, 共通課題 (エレベータ) を解いた際に所属したグループを示す。

次頁以降に,

(1) 今回のワークショップに参加するにあたって事前に提出頂いた各自のポジションペーパ (**プログラマにとっての道具とは?** [1]) とワークショップに関する事前の要望/期待 (**ワークショップに期待すること**),

(2) 当日のポジション発表に対してメンバから出されたコメント (**ポジション発表に対する質問/コメント**) [2], 更に

(3) ワークショップ終了後の感想 (**参加後の感想**) [3]

を, 整理番号 (No.) 順に掲載する。

[1] この課題は, 伊藤さんから頂いたものである。夢想を論じても空虚だし, あまりにも現実ばかりを書き連ねるのも羞恥心を催すし …。なかなか痛い所を突かれたと思っている 8-)

[2] すなわち これは その人以外の参加者からのコメントであることに注意!

[3] 多忙な折に, 参加後の感想などを寄せられた方々に改めて御礼申し上げる。

01 山口 登 (やまぐち のぼる)
性別：男 年齢：30 血液型：A
会社：(株) リクルート
所属：技術統括室

プログラマにとっての道具とは?

1. はじめに

一般にプログラマが「道具」を使用する最終的な目的は,

(1) 開発生産性の向上

(2) 保守容易性の獲得

の2つであろうと思われます。この二大目的は今後もおそらく変わることは無いでしょうから, この2点に照らし合わせることから私なりの, 『今後の「道具」の有り方』について考えを述べさせていただきます。

2. 開発生産性を向上させる「道具」に求められる機能とは?

「道具」が未発達であった時代でも, プログラマは様々な方法で開発生産性を向上させてきました。例えば, "以前に作成した procedure をコピーする。もしくは若干の変更を加える", "決まりきった procedure をサブルーチンやマクロとして局所化する", "プログラムを作成する時に, 自分なりのコーディング手順を確立している。すなわち ad hoc なコーディングすることによるつまらぬミスを防ぐ" 等がこれにあたります。

「道具」においてもこれらの考えの応用が生産性向上に寄与することは, 既存の「道具」を見ても明らかだと思われますが, 今後も「道具」を構成する上でのキーワードたり得ると考えます。すなわち,

(a) ソフトウェアの再利用機能
(b) 部品の作成・管理機能
(c) procedure そのものを記述しなくても良くする機能

の「道具」での実現および促進がかなり重要なのではないかと考えております。

3. 他方, 保守容易性を向上させるための機能とは?

ざっと考えてみてソフトウェアの保守が容易であるという条件として以下のものを思い付くことができます。

(1) 仕様の理解が容易である

(2) 仕様変更による当該変更箇所以外の影響分析が容易である

(3) 物理的な変更箇所が少なくて済む

さて「道具」においてこれらの条件を充足する機能としては,

(1) に対応して,

(d) プログラム仕様書の自動生成機能
(e) 俯瞰性の高い成果物表示機能 etc.

(2) に対応して,

(f) リポジトリとの関連機能
(g) 影響分析ユーティリティ機能 etc.

(3) に対応して,

(h) 部品の一元管理機能
(i) 整合性を自動的に保証する機能 etc.

が挙げることができます。

4. 今後の「道具」に求められるもの

私が今後の「道具」に求めたい機能について (a) ～ (i) の 9 つを挙げさせていただきましたが, 実際には (d) を除いた 8 つの機能にはオーバーラップするポイントがかなり見受けられます。ここでは, それらの整理・統合を試みてみます。

4.1 部品化の促進

(a)(b)(h) については, 「部品化の促進」という言葉でくくることができるのは, 言うまでもないでしょう。ただ, 「使いにくい部品」, すなわち「機能が複雑な部品」「前処理・後処理が多くてかえって生産性を損ねるような部品」を作成しないために, 「宣言的に使える部品」「context に独立に使える部品」の作成・管理機能が望まれます。

4.2 標準化の促進

(c) で述べた『procedure そのものを記述しない』ということは, 『「宣言的」に定義することで全てを済ませてしまう』ということです。実際問題として procedure 記述を皆無にすることは, 不可能に近いかもしれませんが, 「共通部分」を多く見出し「宣言的」定義あるいは「部品」としてくくってしまうことで procedure 記述を減らすことは可能でしょう。そして, もっと言えばその「共通部分」を増やすことが「標準化」だと考えられます。「標準化」を促進する

ことは「procedual」から「decraretive」への転換であると私は考えていますが,「道具」が「標準化」を促進するためのツールであれば良いと思っています。

また, (e) についても「標準化」は寄与するところが大きいと思います。なぜなら「標準化」により増やされた「共通部分」の表示を最小限に押えることができ, プログラムの特徴のより簡明な表現が期待できるからです。

4.3 リポジトリとの連係

(f)(g)(h)(i) および (b) の部品管理については, 今後は「リポジトリ」との関わりなしには語ることができないでしょう。リポジトリは前述した,「部品化の促進」「標準化の促進」にも有効に使用することができましょうし, これからの「道具」にはこの点は是非期待したいと思います。

5. 最後に

というわけで, 私が今後の「道具」に望むキーワードは,

- 部品化の促進
- 標準化の促進
- リポジトリとの連係

という, 今や大流行のキーワードと同じということになってしまいましたが, やはり, この3点に秀でた「道具」が業界を席巻するだろうという感触は持っており, ぜひそういった「道具」を期待したいと思います。

ワークショップに期待すること

初めての参加になりますがよろしくお願いします。

個人的には「ソフトウェアの部品化」「フォン・ノイマンボトルネックの打破」といったことに興味を持っており, そういう観点からの CASE ツールにも大変関心があります。

今回は参加者の皆様の多種多様な御意見に触れ啓発されたいと考えております。

[ポジション発表に対する質問/コメント]

宣言的手法の具体的な利用法。

プロシジャにかわるパネル定義とは?

02 南田 邦治 (みなみだ くにはる)
性別：男　年齢：26　血液型：O
会社：オムロン (株)
所属：EFTS 統轄事業部 開発センタ 第2開発室 第1開発課 (ESA)

プログラマにとっての道具とは?

プログラマにとっての道具, 道具ってそもそも何でしょうね。私自身, 道具というのは,「作業者の作業を目的通りに上手に早く遂行するもの」と思っています。つまり, それが, 漁師さんであれば船, 網, 魚群レーダになるし, 消防士さんであればホース, ハシゴ車になるわけです。

それじゃ, プログラマは? 一般的にいって, ×××ツールといわれるようなツール類がそれに該当するのでしょう。例えば, プログラマにとって一番身近なツールといえば, エディタ, コンパイラ, デバッガといった下流 CASE ツールがそれに該当するであろうし, 主に上流設計を行なう設計者であれば, それが上流 CASE ツールに該当するのでしょう。となると, プログラマといわず世間一般にいわれる「ソフトウェア技術者」にとって必要な道具とは,

上流 CASE ツール + 下流 CASE ツール =
統合 CASE ツール (以後, CASE ツール)

になるわけですね。

それでは, ソフトウェア技術者にとって将来, CASE ツールは, どうあるべきか。最終的には,「ちょこちょこと仕様をある定型フォーマットに従って入力さえすれば, コードを自動的に生成し, テストを行ない, オブジェクトを出力する。又, 保守に関して完璧なまでの支援機能を持つ」といったところでしょうか。あくまでも, ソフトウェア技術者は, 分析と設計を重点に考えていさえすれば, それ以外は, ツール自身が管理, 制御するというイメージです。(私自身, CASE ツールに関しては, 素人ですから, 発想が貧弱かもしれません)

その他, こんなことができればと思うことを列挙してみます。

a) グループウェア作業の支援機能

b) カスタマイズ機能

a) については, 大抵のプロジェクトがグループで作業をしますから, 電子メール, 電子掲示版, スケジュー

リング機能, 文書作成及びコメント機能といったグループで作業を行なう場合にグループ内のコミュニケーションを充実するために必要な機能といえます。プロジェクトリーダ, マネージャの立場からすると, プロジェクト管理機能といったプロジェクトの進捗管理, あるいは予実管理, 開発データの自動収集機能もあれば便利ですね。

b) については, 各プロジェクトにより開発環境が違うため, 自プロジェクトに応じた最適な環境に CASE ツールをカスタマイズできる機能が必要になってくると思います。

例えば, 分析手法の方法論を A → B に変更できたりとか, CASE ツールと人とのインタフェースを変更できるといったことです。また, 必要な機能 (他社ツール類) を追加したり, いらない機能を削除することにより, プロジェクトに最適な CASE ツールを容易に (ここがポイント) 構築できることこそ, ユーザが最も望むことではないでしょうか。そのためには, CASE ツールの標準化を推進し, 現在のようにユーザがどのツールを購入すれば良いのか迷わないようにすることが第一だと思います。

ここでは, CASE ツールの基本機能 (図がうまく書ける, コーディングの自動生成, エラーチェック, テスト機能など) にはあまり触れずにそれ以外に必要と思われる機能について考えてみました。私が使用したことのある CASE ツールというのが, 単なる「お絵書きツール」みたいなもので本格的なものではなかったために, ひょっとしたら上記に挙げた機能が既に実現されている CASE ツールがあるのかもしれません。もし, 仮にそうでしたら, 私の認識不足と解釈してください。

ワークショップに期待すること

- CASE ツール導入事例紹介
  成果, 問題点, 今後の課題等。
- 参加者同士の情報共有化の枠組作り

[ポジション発表に対する質問/コメント]

CASE ツールの標準化についてもう少し聞きたい。

どのようなグループウェアを具体的に考えているか?

参加後の感想

今回, こうした社外の WS に, しかも泊まりの合宿形式で参加するにあたり, 最初は, 非常に不安でしたが, 予想以上に (?) 面白かったです。面白かったというのは, どちらかというと, WS の内容よりもメンバ各人のユニークさに依るところが大きかったです。私自身は, こうした人達の話しを聞き, 行動を見て, はるかに私のような若者 (?) よりもバイタリティにあふれているなぁと感心させられました。その裏には, やはり自分の今までの経験があるわけで, 学ぶべき点は多かったと思います。そうした人達の話しには, 重みがあり, 人を引き付ける何かがあったように思われます。

私は, どちらかといえば今までは, 漠然と仕事をこなしてきた感が強く, メンバの人達のような確固とした「自分の意見」といえるものがなかった気がします。本 WS に参加して改めて, 「自分の意見」をもつべきだと痛感してます。又, 今まで閉じた世界 (社内のマシン, ツール) で開発をしてきた私にとっては, 社外の人達と生で接するという貴重な体験は, 自分に刺激を与えるだけでなく, 前述のように自分を見つめ直すよい機会でした。

最後に, SIGENV の皆さん, どうもお世話になりありがとうございました。今後ともよろしくお願いします。

03 大塚 理恵 (おおつか りえ)
性別：女 血液型：O
会社：RSK
所属：技術開発部

プログラマにとっての道具とは?

道具といっても規模の大きい数千万円もするような大道具から, 手軽に作れる小道具までいろいろあるし, 私が仕事で作っているような設計用紙用のワープロの書式などというのも入るかも知れない[1]。 また AI とか利用すればまた違った道具も見えるのかもしれない。

道具について提供する側と提供される側とあるうち, 私の場合, 市販 CASE を評価・購入するので提供される側に属し, 社内的に管理または利用促進のための小道具を提供する立場にもある。大道具については「今後どのような道具を作ってほしいか」の方が考えやすい。どういう道具が欲しいかと言えば, やっぱり CASE なら統合 CASE のようなもの (いろいろレベルがあると思いますが), ツール間の情報の交換が比較的楽にできるものになる。それと「今後ど

[1] というわけで, 私は純粋なプログラマとは違う

うあるべきか」では, 機能がモジュール化されているというか, そんなに多くの機能を提供しなくてよいが (余計なオプションはいらないから) きちんと目的を実現しているものであるべき。また分析とか設計でCASEが使われるようになってきたことを考えると, 発想を支援するようなことも一層必要になるでしょう[2]。「具体的な姿」となると, 大変現実的なところになります。

さて私は現業であるプロジェクトの分析・設計工程の良く言えば環境支援, その実, 雑用をしており, 低次元のところでPC環境でのツールの適用にしびれているところです。そのプロジェクトに係わっていると分析・設計している開発者にとって道具がこうあるとよい, こういうのはだめというのが見えてくる。しかしそれ以前の話で, そもそも「ツール」を使う・作るという文化がなかったため, 開発上何が必要かとか, 本当に実現できるのかという議論ばかりに時間を費やしている。

今回はワープロでの文書作成だけでなく, 初めて工程の一部にPC環境を使うようになったが, PCのソフトをどのように利用すべきか勝手がわからない。上流CASEを動かすためにPCを使うようになったにすぎず, PCを使うと開発にどんな影響があるとか最初から検討していたわけでない。使い始めるといろいろ欲が出るもので, ユーザー (分析・設計者) はCASEが持つリポジトリの情報を活用したいと言う。いちおうCASEメーカーからユーザーがカスタマイズするための製品は販売されているが高価だし, 使いこなすほどの技術がない。そこにあれもこれもと報告書 (設計情報の一覧表) 作成を希望する。なんとか下流近くまで利用していきたいから, 汎用機でも利用できそうな出力がほしいとか出てくる。

CASE自体にも報告書作成の機能はあるが, 本格的に使うには不十分であるし, それだけでは処理できないので, それが持っているデータを抽出し, テキスト処理することになったが, 誰もPC/DOSでのファイル, テキストの加工方法を知るわけでない。汎用機でならともかく最近までテキスト処理という文化はなかった。もし適当なツールがなければ一度紙に出力して, それをワープロに入力しなおし兼ねないし, 汎用機にファイル転送し, そちらで処理して再度転送仕直すのが常套策でしょう。

結局, 簡易言語によってツールを作ることになったが, プロジェクト外の専任者に一時的に作らせるため, メンテが問題になるに違いないし, ちょっとしたツール開発・管理の問題も組織でどう取り組むか, 考える必要がありそう。

以前UNIXのアプリケーション開発でこの手のツール作りをしたことはあるが, 全然儲からなかったので個人的には好意的ではない。開発に必要なプログラムはメーカーが全て提供するものであって, 我々ユーザーが一部開発するなんてとんでもない, その必要はないという組織の認識だったためか, C言語のプログラムを何本か外注してもシステムがそれだけで動くわけでなく, 結局システムの運用のためのツールが必要となり闇で作らざるをえなくなったことがある。(プログラマがプログラミング上に必要なツールというのと少し違うが) ツールというとまだ市民権がなく, 小さいツール作りも開発や運用上重要なものだと認識させていく必要がある。

またどんどんツールを提供していっても, そのツールというのが例えば設計用紙記述の作業が多少軽減するくらいのものだと, 何もなかった今までよりはいいかもしれないが, 開発工程間での情報の流れを見直して, それで道具は何が必要か考えようというものでないので, 一体どれほどの効果があるのか。たいした機能のツールでもないのに作業マニュアルが完備し, レビューさせると途端に満足, 安心し, なぜ私に作らせたか目的を忘れてしまうことさえある。だから量産する道具の中には屑のようなものもたくさんある。使いたい人が必要なときに簡単に作れるツール, そういうツールを備えた環境を提供できるのが一番いいのではないか。

しかし全社的に帳票の種類の削減キャンペーンをしていた職場で, 使われるかわからない報告書作成用のツールをじゃんじゃん提供していいのか。データベースを提供し, 参照するためのツールの作り方を教えた方がEUC的で賢いだろう。そう考えるとツールの提供の仕方も課題になる。

それから細かいオプション的な機能はユーザーによって趣向が異なるだろうし, そういうところはユーザーがカスタマイズできるようなインターフェースを容易してくれればいい。それより例えば上流CASEで技法が表記法上正しくサポートされてないとか (全然改善されなかったり), リポジトリの情報の維持・管理方法が不明だったり, という最も基本的な機能はメーカーで責任持って提供してくれないと困る。ただメーカーで早く対応してくれるよう催促しつつ, それ実現するのは今の技術じゃ無理とか, できても遅くて使えないのじゃないかというのも多い。

[2] 発想支援というのは立派な方が研究されていると思いますが, まだ興味のレベルです

最後にだめな道具は, 例えば分析・設計で絵をかいているとき, ちょっとした操作に大変時間がかかったり, ユーザーにストレスを感じさせるような, ユーザーの思考を妨げる道具だと思う。

ワークショップに期待すること

共通課題が上げられていますが, このエレベーターの問題にはまだ取組んだことがないので, ワークショップまでにちゃんと問題を理解できればいいですが。(個人的には図書館問題とか酒在庫問題と比べ, 好きでない, 苦手意識がある)
しかし今回のテーマは環境だから分析・設計の CASE STUDY だけをするわけでもないでしょうし, 一体何をするんですか?

[ポジション発表に対する質問/コメント]

- 「人間の発想を支援する」ツールについては私の興味があります。既存のものも含めて、いろいろなダイアグラム等が出てくれば良いと思います。
- 「知的な仕事をしたくなる道具」をもう少し具体的に聞きたい。
- OMT に基づいた作業の標準化, どんなことをしているのか?
- 知的 CASE の具体例。

参加後の感想

1. ツール観は変わったか?

上流 CASE に対する「お絵かきツール」という言葉は一般化したのか, 今回も「お絵かきツール程度にしか使っていない」と聞いた気がする。機能はある程度使用可能なのに手抜きして活用していないだけだったり, 上流 CASE はまだ未成熟といった感じがしたり, 今後に期待するところが大きいせいもある。

誰にも使えるお絵かきツールも必要だが, 今回 CASE を使って共通課題に取組んだり話を聞くうちに, やはりプロはプロのための道具を持たなくてはいけない, 持つべきだ, 持った方がいい仕事ができる, そんな感じがしてきた。

現在はとりあえず CASE なりツールを使って仕事をこなすのに精一杯で, その CASE が本当に使い易いのか, 役に立っているのか, あるいは使い勝手の改善は可能か, と考えたこともないのではないか。これは怠慢でしょうか? プロとは程遠い私が言うのも気が引けますが, プロなら考えましょう。こういうワークショップに参加すると考える機会が得られます[3]。

ただプロならたいした道具がなくてもよい仕事はできるでしょうが, それでもプロならよい道具の選び方を知っているはずです。

同じ目的の CASE にも素人用のものとプロ用のものがあってよい。素人用のは特別な学習をしなくてもある程度の成果物は作れるが, プロ用のはソフトウェア工学の教養や修練がないと動かせないが成果物には高品質が保証される, という CASE が現れてもよいと思う。そんな CASE を作ってください!

2. 共通課題

エレベータ問題が好きでないのは, 「エレベータが動く」からです。つまり「動くもの」をどのように捉えればよいか, 今までの経験を総動員してもアイディアがない。いわゆるリアルタイムというのが銀行のオンラインシステムなら身近にあるのだが, こうした Toy Case Study となるとパニックになる。仕様を記述し, モデル化すれば何とかなることはわかっているのだが, 成果物が想像できない。

ところが今回 4, 5 人で SA/SD で考えながらモデル化し, C プログラムまで出来てしまったのは, ここ 1, 2 年 OMT に時間を費やしてきた私にとって大変な快挙 (大笑い!) です。なぜでしょう。

3. BOF の話

予定では BOF の時間に, 'What do we think about?' と書かれた伊藤さんの OHP シートの説明を求めることから始まり, 3 時間ほどかけて環境, ツール, プログラマについて熱心に話し合っていた (と思う)。会社でそこで聞いた話, 興味を持ったことなどを話していたら, 同僚が親切にも「イラスト 西洋哲学史」を貸してくれた。東洋哲学史も必要なんですけど。

4. 自然の中で考える

坂北荘周辺ガイドマップによると「◇水と緑に包まれた, 雄大な自然境。奇巌奇勝県立自然公園 差切峡」とあります。近くにある大滝は「差切峡ではこういうのを大滝というのか」と感激するばかりでした。観光地化されていない, 冬景色一色というところが今回のワークショップの副題ー自然の中で共通の課題を通して環境を考えるーにふさわしかったと思い

[3] 今回のスタッフではありませんが, SEA で開催するワークショップへのお誘いです:-)

ます。この近辺を曼陀羅の里というのですが, 曼陀羅の里で触れる nirvana なんて周囲の環境に調和しているというか何というか。。。

隣接のゲートボール場以外娯楽施設は何もありませんが, 山菜料理が嫌いでなければ「四季折々のおいしい料理」と 24 時間入浴できる展望風呂が楽しめると思います[4]。パソコン持参で行かれるときっと仕事がはかどるでしょう。

04 遠藤 浩和 (えんどう ひろかず)
性別：男 年齢：25 血液型：A
会社：日本電気ソフトウェア (株)
所属：共通ソフトウェア事業部 CASE 開発部

プログラマにとっての道具とは?

1. 現状

最低限 エディタさえあれば, 各種仕様書の作成から, プログラミングまで可能

[利点]

- エディタが使えれば OK (sed, awk, perl)
- 軽い (早い)

[欠点]

- フリーフォマットで書けるため, 個人差が大きい

2. そこで

CASE の「わく組み」(方法論) による制約

[利点]

- 個人差が入り込む要素が減る
- 「やらなければいけないこと」がはっきりしていて, もれが少なくなる (テンプレート)
- 部品化の促進

[欠点]

- 自由度が減る
- ベンダの方法論に依存
- 作業量の増加
- 遅い

[4] 婦人用の方は展望も何も間近に迫った裏山が見えるだけ。でも改装されたばかりらしく, きれいでした。

3. そして今後...

- 既存の資産の有効利用 (リエンジニアリング)
- グループ開発の支援 (グループウェア)
- 言語に依存しない表現 (記法)
- 簡易入力機能
- 視力回復
- 運動不足解消

ワークショップに期待すること

幅広いものの見方を発見したいと思っています。どうぞよろしくお願いします。

[ポジション発表に対する質問/コメント]

有効に保守を行うための「CASE」というものについても CASE 開発者の立場からお聞きしたいと思います。

リポジトリはどう使うか? ストレスを除くには?

方法論込みのツールでもそれだけではユーザ (開発する SE) は使えないので, メーカの SE (作った人) が十分サポートしてお金をいただくのがよい道だと思います。大手メーカはメーカなりの売り方があるのではないかと思うし, その期待もする。

参加後の感想

「今, ソフトウェアの開発環境とは?」というテーマで 3 日間過ごした訳ですが, CASE 開発に携わるものとして, 反省すべき点が幾つもあることに気づきました。

その中から一つ挙げるとするならば,「管理するための CASE」になってしまっている, ということです。

確かに, CASE を使う目的の一つに「管理」という側面が入るのは当然としても, 使う (使わされる) のはプログラマーです。プログラマの視点に立った CASE にしなければそれは, 何をアシスト (サポート) しているのか, 何のための CASE なのかが曖昧になり, 結局それは使われなくなってしまいます。

本当に現場ではどういうツールがあれば便利なのかを, じっくり考えて開発にあたる必要があることを痛感しました。今回のワークショップでは, (通常の開発作業では決して聞くことの出来ない) 数々の異なる意見が聞けたため, 大変参考になりました。ワークショップを通して体験したこと, 感じたことを, 今後 CASE の開発を通じて生かしてゆきたいと思います。

(04) 丸山 裕　(まるやま ゆたか)
性別：男　年齢：27　血液型：A
会社：日本電気ソフトウェア (株)
所属：共通ソフトウェア事業部 CASE 開発部

プログラマにとっての道具とは?

“道具” の今後

1. 自動車開発とソフトウェア開発の相違点

まず最初に道具 (CASE ツール) の今後を考える前に CASE ツールの使われる場面を考えてみることにする。私は, 自動車が好きなので自動車の開発とソフトウェアの開発を比較しながら CASE ツールの使われる場面を考えてみる。

自動車の開発の場合, 大きく分けて設計 (もしくは計画), 製造 (いわゆる図面作成から製造ラインまで), テスト (検収) に分けられると思う。それぞれは, 以下のような作業になる。

- 開発段階
  イメージから模型を作り何度も評価をする (デザインの検討)。この後, 模型から図面 (ラフ線図) を作成する。
- 製造段階
  ラフ線図から細部の検討を行い, だんだん内部の仕様が決定される。このときに CAD/CAM を使用する。最終的な製造は, コンピュータによって制御されたロボットがやってくれる。
- テスト段階
  基本的には人間がテストを実施する。場合によっては, 機械などを使って測定したりする。

以上のように簡単に示したが, 自動車開発というのは自動化が進んでおり, かなりの人員削減に成功している。一方ソフトウェア開発はなかなか自動化するまでにはなっていない。この違いは次のような点にあると思う。

- 自動車の開発は, 試作を何度もすることによって早期に問題点を潰している。
  → ソフトウェア開発は, 製造段階で (テスト段階でも) 問題が発生する (実際動かさないと分からない??)。
- 自動車の開発には, 特定の道具 (ツール) が存在する。
  →ソフトウェア開発は, ツールはあるにはあるが特に決まっていない。開発手法も対象によってまちまち (構造化, リアルタイム)。
- 自動車の開発は, 完全に分業されている。それぞれの担当はそれぞれの道具の使い方にプロフェッショナルである。
  →ソフトウェア開発は, たいていの場合は 1 人で全てをこなすような感じである。

以上から考えると, **「早期に問題点を洗い出す」**ということと, **「各工程毎に目的に合った道具」**, **「道具を使うプロフェッショナル」**ということが大切であると言える。

2. 工程と道具

それぞれの工程の目的に合った道具が必要であるのは前述の通りであるが, 現状のソフトウェア開発にはどんな道具が存在するのか整理してみる。

- 分析
  システムフロー図, 実体関連図等, 概念的な図を作成するエディタが多い。
- 設計
  データフロー図, プロセスフロー図, ストラクチャチャート, ロジックフロー図等, 実際のソフト構成を意識して作成するものが多い。
- 製造
  ソースコードジェネレータ, 自動コンパイル等, コマンド的に使えるもの。
- テスト・保守
  テスト網羅度測定, 性能評価, デバッガ等, コマンド的に使えるもの。

以上のようにどちらかといえば, 製造以降のツールはコマンド的な物が多く, 種類も豊富である。しかし, 分析・設計では組み込み型のツールが多く, 種類も少ない。なぜこのように分析/設計では道具が少ないのであろうか?

3. CASE ツールの使い方/使われ方

ソフトウェア開発の道具 (CASE ツール) は, まだ道具としての認識が低い。それは使う側として**「全てコンピュータがやってくれるのでは?」**という考え方を捨てないからである。自動化が進んでいる自動車の開発でさえ分析/設計は人間が考えている (道具は使っているが)。ソフトウェア開発者は “CASE ツールは道具であって万能ではない” という考えを頭に

入れる必要がある。つまり道具というのは, 適した局面でのみ威力を発揮する。また, その道具の効用・機能に精通して欲しい。そうすることによって, ますます分析/設計に集中することができるからである。

CASE ツールを作る側としては, 開発工程全てを解決しようとしない方がよい。分析/設計というのは, それぞれ性質の違うものである。分析は, 抽象化 (イメージ) したものを扱い, 設計は詳細化したものを扱う。分析/設計を同時に扱うツールの欠点として, 分析したものを設計に移行できないということがある。このように考え過ぎるあまり, それぞれの工程に最適なものを提供できない。

4. これからの "道具"(CASE ツール)

今後, 道具としての CASE ツールに要求されることは, "各工程に対応したツールが必要である" や "汎用的に使えるもの" ということである。以下のようなツール体系が望ましい。

- 分析段階
  抽象化したモデルをシュミレーションまたは, 過去の資産を再利用することにより, すぐに実行できる。抽象化には, GUI を使用し対話的かつビジュアルに作業する。このことにより, システムの持つ問題点を早期に抽出する。
- 設計段階
  分析によってできたプログラムを解析またはコンバートすることにより, システムの構成を作成する。構成図から内部を詳細化していく。ここでは, 分析によって作成されたモデルの外観を損なわずに詳細化する。また, 設計した情報は分析にもフィードバックできる。フィードバックすることによって, 分析者の意図を確認する (最初の要求とずれが無いか?)。
- 製造段階
  完全に自動化する。設計情報から自動的に動作システム上のオブジェクトを生成する。
- テスト
  テストは人間が行う必要がある。ただしテスト項目の抽出や性能分析はツールを使う。

上記のことをもう少し補足すると,

分析段階では, 過去のプログラムを部品化して登録できるようなものがよい。登録した部品から必要なものを取り出して使い, ときにはを多少手を加えたりできる (結局オブジェクト指向?)。この段階では分析者が要求に基づいたシステムのアウトラインを作る。

設計段階では, 実際のロジックや内部のインターフェースを決める。ここでは, 複数の画面を見るため (ブラウジング), 1 つ 1 つのツールが手軽 (起動が早い, 編集が楽) である方がいい。

分析ー設計は問題点が無くなるまで繰り返す。設計が詳細になるにつれ, 最初分析によって作られたモデルも詳細になり現実へ近づいて行く。つまり, 抽象的なモデルが設計によって具体化される。この 2 つの工程の目的は, 実際のシステムに忠実な設計書を作成することである。製造以降は, この設計書を元に単にコード化していくだけである。

これらを実現するための技術として, 以下のものが必要である。

(1) マルチウィンドウ
  設計情報をさまざまな角度から参照するために必要。

(2) ビュジアルなインターフェース
  システムの構成や関係を視覚化。シュミレーション的な機能も必要。

(3) モデル化するための理論
  過去の生産物をモデル化するための方法論 (オブジェクト指向?)。また, オブジェクト指向で作成されたもの以外もモデル化できる方法 (理論)。

(4) データベース
  単に仕様情報だけでなくモデルの属性や関係を格納できるデータベース。

外国のおもちゃに "REGO"[1] というものがある。これは, 20〜30 種類のブロック型の積み木を組み合わせ, 色々な物 (自動車や建物) を作りあげる創造的なおもちゃであるが, 分析/設計もこのように簡単にできるといいと思う。積み木のようにシステムのモデルを簡単に作り上げ, そこから詳細な設計を開始する。このような分析/設計工程が理想的である。開発者は製造 (設計情報のコード化) に時間を取られるのではなく, 分析/設計に十分時間をかけることができる。分析/設計は完全な自動化する必要はなく, むしろ人間の考えを導き出す為のアシスタントをする道具が必要である。

また, 現在の状況では, 1 つのベンダーで最適な全ての CASE ツールを揃えることは難しい。使う側で最適な CASE ツールを選択するような使い方ができると良い。そのためには CASE ツール同士でのデータの互換性や操作の統一性が必要である。

[1] たしかそうだったと思う

分析と設計の「分立」と「データの連携」これが今後のCASEツール成功の鍵ではないだろうか?

ワークショップに期待すること

期待

CASEツールの持つ問題点が少しでも明らかになって欲しい。
今後の開発作業に役立つ内容。

取り組み方

できるだけ発言をする。(活発な意見)
他の参加者の意見をよく聞く。
発想の転換(日頃考えないことを考える)

05 渡邉 雄一 (わたなべ ゆういち)
性別:男 年齢:34 血液型:A
会社:(株) アスキー
所属:OA推進部

プログラマにとっての道具とは?

1. はじめに

ソフトウェア開発に関わるようになってもう10年以上になるが,もっぱらエンドユーザの代弁をするようなプログラム作り/システム運用に携わってきた。このポジションペーパでは,簡単に現状を紹介したあと,今までに自分が使用してきたツールの使用感も踏まえ,今後のツールに期待していることなどを書き散らす。

2. 現状

まず自分の現状を簡単に紹介する

2.1 主たる開発環境

X-terminal + X11R5
Sun SPARC Station 2 (Sun OS 4.1.1)
Informix 4GL SQL
AWK
C-shell

2.2 開発の対象

社内向けの事務アプリケーション

- 利用者は1〜5多重位で動作する
  → 排他性(トランザクション制御),機密保護(アクセス制御)の考慮が必要
- Informix 4GLで業務用の画面や夜間に動作するバッチ処理用のプログラム開発
  → 画面系は,キャラクタベース。Informixに用意されている画面制御に依存している。
  → バッチ処理のプログラムは,データベースを跨ぐような大量のオンラインである必要はないものが中心。

2.3 現状のツールへの利用など

残念ながら現在利用しているツールと呼ばれるものは無い。informixにオンラインデバッガがあるようだが使ったことはほとんど無い。4GLで作ったプログラムがcoreを吐いたときにdbxくらいは使ったことはありますが…

2.4 現状の開発/問題解決の方法

現実のプログラム開発の要求は,「xを入力する画面が欲しい」という形で提示されることがほとんどである。
このような要求に対して,通常は簡単なヒアリングをし,システム化の必要性が十分にあると判断されたら,現場から再度 詳細な画面の構成などを提示してもらって,それをたたき台にシステム作りが始まる。主として画面に関しては,Informixに備わっているRDBテーブルを操作する簡易画面 performを利用してテーブルの構造を決定しながら画面の動き(主としてカーソルの動きとメッセージの表示内容,さらにwindowと呼ばれる画面内に必要に応じて作成出きるミニ画面)を決定してゆく。
これらはすべてperformでシミュレート出きるわけではないが,紙やテキストを画面に表示するだけでは,項目間の空き具合やカーソルの操作感覚までは現場の声が聞こえない。
これを2〜3回繰り返して,現場の要求が固まったと思われるころに画面の仕様が固まったとし,4GLで本番用のプログラム作成にかかる。

3. どんなことがしたいか,どんなツールが欲しいか

簡単な(=部内で構築実績がある)アプリケーション開発に関しては,そのノウハウを開発者から聞き出し,そのソースを流用することでかなりのことが出きるで,ある意味で満足している。プロダクトの再利用ほどの生産性は望めないと思っているからである。
問題は,実際に開発経験が無い/乏しいものや,設計そのものに困難を極めるものへの対応に新たなツールや方法論の修得の必要性を感じている。例えば,自分が抱えている課題からでは以下のようなものである。

会議室の予約システム

- 望まれるインタフェースは, 既存の informix のそれに難しい
  → cursus ないしは X-window を利用したインタフェースの提供を検討しなければならない。
- 利用者毎にいくつかのインタフェースの要望が分かれる
  → 利用者毎にカストマイズ出来る仕掛のサポートが難しい
- セキュリティの実現が難しい
  → 組織毎に望むセキュリティのレベルが異なる
  → 組織が流動的
- システムそのものの仕様が難しい
  → 会議室の大きさがパーティションで区切れるなどで変化するなどを考慮する必要がある
  → 予約出来る範囲（期日, その操作を出来る人の限定）といったセキュリティとも関連する
  → モニタタスクが個々のリクエストに答えるといった要求仕様モデルが考えられるが, その実現は TSS でどうすればよかろう?
- 他のシステムとの連携
  → 個人のスケジュールデータやメイルとのデータの双方向の連携が理想だが ···

これらツールへの要望を, 既存の CASE ツールがみたしているであろうことに照らし合わせて, 望まれる真のツールのあり方をみてみよう。

3.1 下流の CASE ツールは ···

いわゆる下流の CASE ツールは, これら問題に対処しうるであろうか?画面の作成支援に微力ながら効果を発揮するかもしれないが期待出来無かろう。それは, 画面上の項目の状態で, 現在のフィールドを規制したり無視したりといった論理の積み重ねであるからだ。既存のツールでそれらをまとめることは非常に難しい。

現状の業務画面で当たり前に実現されている機能の例としては

1) **表示されているものと実際に格納されているものが異なる**
   性別のフィールドに '1' 'M' '男' のいずれかが入力されると '男' と表示されて, データ項目内には '1' と格納されている。
2) **データの組み合わせ条件で処理が分岐する**
   • 売上種別が 'カード決済' の場合には 'カード番号入力画面' が開いて番号の入力を促す
   • 特定のフィールドに有効な値が入っていない場合には, 画面から抜けることは許されない
3) **データの内容でデータの編集操作を行う**
   • 明細行に誤った項目が入力された場合に, その金額欄に '0' を入力するとその行は削除され, 後ろの行は自動的に前に詰められる
   • 備考欄に '1' を入力すると '製品' と変換される

が上げられる。これらの要求仕様が, エラー時のカーソルの動きなども含めて実現する画面の特性を考慮せずに記述することは, 事実上 無意味な気がしている。

では下流の CASE ツールは, どのような問題ならうまく対処しうるのか?
私個人の経験からすると, 帳票出力などにはかなりの効果を発揮することが判っている。
それは, 帳票という限られた枠のなかで, 合計, 平均, グルーピング, ソートといった特定の機能の組み合わせだけが要求されるにすぎないからである。しかしこれとて, 文字中心の情報を扱っている場合には耐えられるが, 人事情報などで画像を帳票の 1 部に出したいとかなったら, どうにもしょうがない。

3.2 上流の CASE ツールは ···

いわゆる上流の CASE は上記の問題にどのような解やヒントを与えてくれるだろうか?
まず期待することは, どうモデルを形成するかということと, その実現の具体的な糸口が提示出来るかによる。

前者に関しては, CASE ツールの出来不出来ということよりも, 多くの設計事例を記述して蓄えていることが重要なのではないかと考える。要は, その CASE ツールで記述出来ることが万能であれば嬉しいわけで, 万能な記述道具があっても書けるだけの知識と才能がなければどうしようもなかろう。だが, 多くのプログラマが他人のプログラムに感心したり失望して書き直したりを繰り返してプログラムを修得するように, 今ある有望な方法論や表記法を場数を踏むことで修得するだろう。
現在の CASE ツールは, 3 つの問題を一緒に考えて無理をしているように思う。3 つの問題とは, 1) ツールの背後にある方法論の修得と, 2) それらのツールでの利用方法と 3) 真の問題解決のためのワークベンチとしてのツールである。3) は, 実際に上記のような問題をエンジニアが考える場面でどんな支援をしてくれるのか?ということである。
3) に対する答は, 設計事例の RANDOM HOUSE で

あって欲しいということでもある。それは，我々が辞書に書き込みをすることで，自分の辞書を育てるように追加や訂正がそれらに出来る必要があろう。

後者については，CASE で支援すべき性格ものでは無いような気がするのでここでは議論しない。

4. CASE ツールへの嫌悪感

以前に勤務していた会社で，CASE ツールのはしりのものを使う機会を得たことがあった。SPACE というそれは，ディシジョンテーブルと DFD で仕様を記述するということで，COBOL のプログラムを生成するというものであったが …

このツールを使うことは苦痛以外の何物でもなかった。それは，生成すべきものの形そのものを規定しなければならないからである。ターゲットの言語にその言語の規定があったにせよ，その前段階においてツールを使うからにはツールを使うメリットがなければならない。いわゆる CASE という言葉に期待するものは，メタな思考のを妨げないように，現実のプログラム支援をしてくれることである。そのためには，複数のビューだとか，ある程度進んだ段階での記述でのセマンティックスのチェックなども有効であろう。
しかし，本当に有効なのは メタのデータ構造の支援ではなかろうか？数学でいえば，集合に族があり，また商集合に準同型写像があるようにデータのまとまりを1つのデータとして定義したり，されにそれらを配列などの構造として宣言し利用出来るだけで，アプリケーションの仕様を記述することは，かなり容易になると思える。
また，このツールのもう 1 つの弱点は，図の入力にあった。今日では，PostScript などもあることだし，限られた図の情報を文字情報として入力することで，絵として完成させるような仕掛は欲しい。
Macintosh などで書いてもそうだが，見栄えのする DFD など，作成しだすと 1 日に数枚しか出来ないことが多い。構造が特定のパターンに帰着出来るのならば，その初期情報の入力は，より簡易な方法でなされるべきであろう。修正して見栄えを追求するのは，その確からしさが確かめられたあとで，ゆっくり WYSIWYG で修正出来れば良い。
勿論，それとは別に，思考を助ける，視点を変えるという意味で，異なる表記法や方法論の支援が受けられれば嬉しいが …

参加後の感想

ツールを使うということは楽しい。勿論，それはツールが便利であればこそである。今回 ツールのデモを見せて頂いて，そのようなものがどうあるべきなのかを理解するきっかけが得られ感謝している。

自分自身の一番の成果は，CRC カードのツールとしての実装例を具体的に見せて頂いたことで，改めてカードというもの あるいはメタファと呼ばれるものの検討の重要性を知らされたことにある。

また，普段はモノクロのディスプレイに向かっている者からすると，情報の分類に‘色’というのは便利かつ重要な要素だと感心してしまった。

1. CRC で分析をしてみて

- 共通課題の言葉の整理を，本格的分析に入る前に行うべきである。それは自然と課題を解く際のクラス名となってくる。
- 分析を行うにはカードは不適当である。
  分析の出発点は，問題の全体像の理解と把握にある。そのためにはカードは小さ過ぎるし，カードであることの利点 (いくつかに分類して記載し整理するといった利用方法) を利用出来ない。
  我々はホワイトボードを使った。あいにくと複写がとれるものではなかった。記録を取る手間を考えると，岡崎で伊藤さんが主張していた模造紙が適当かもしれない。
- カードは A4 の紙を 半分に切って (A5 に裁断して) 急あつらえで作ったが 快適だった。
- 現実のカードの特性とは
  (1) 色がつけられる
  (2) 配置を自由に変えられる
  (3) 複数の人間で同時に見れ，かつその場で議論出来る
  (4) 記載してある内容が即座に確認/修正出来る

  といったような事である

2. CRC ツールを使ってみて

- CRC の持っている分析のためのカードとしての特性をツールが 実現出来ていない。
  しかしこれはカードというものの特性を 現実の計算機でどの程度実現するべきか，またそれは実現出来るのだろうかという深遠な話になりそうだ。
- その意味で カードに記載されて情報を テキストファイルで作成してリポジトリにロードする

プログラムを急遽作成してもらったが, これはとても有益だった。

- カードに記載されていることの 論理チェック (カード中の Collabolation とクラスの相互関係のチェック) は有効だった。
- カードに記載されている情報を 画面上の物理的カードに因われて操作することよりも, その整理として OMT なりシュライヤ・メヤなどの手法へのデータとして活用すべきである。
  それは CRC カードで 目的とするシステムのクラスとその責任は明確になったが, それを整理し的確に表現することが出来そうもないと思うからである。
  特に, クラスの洗い出しとその責任までは討論したメンバのコンセンサスはあったと思うが, 全体の構造に対する認識が一致していたか否かは疑問。

3. その他感じたこと

- 要求分析は難しい
  → 当り前のことだが、要求分析は ユーザ (顧客) からの要求の正しい理解とそれらの中の矛盾の発見との葛藤の作業とも言える。
  それを本当に支援出来るのだろうか?
- 運営について
  (1) 人数は適当だった
  (2) 開催期間は もう 1 日くらい長い方が より有意義な議論が出来るかもしれない
  (3) 次回やるとしたら, 勉強の成果を持ち寄る形が好ましいだろう
- ツールの視点の重要性
  討論の中で出てきたことだが, 誰の為 (作る人, 使う人, 使う人をマネージメントする人 etc) のツールなのか? ということは自分ではあまり気にしたことは無かった。
- ツールへの偏見/誤解 (1)
  これも討論の中で出てきたことだが, CASE ツールの利用の大きな期待の 1 つが, ツールからのなんらかの支援であるが, それは上流 CASE であれば方法論の適用をユーザがした際のチェックにある。
  これを拡大解釈してツールを学習の道具のように捉え 機能の有無というか (CASE ツールの) 性能の良し悪しを議論しがちである。これは間違えではないだろうか?
  → もちろん ツールを使いながら方法論の理解があっても構わない訳だ。方法論の提唱者は, その多くを自分の頭の中で考え, 論文なり成書の形で世に問うたであろうが, その理解を彼らとまったく同じ方法でする必要は無いはずだ。もちろん学習のためのツールなり環境が提唱者のそれを正しく表していないなら話にならないが … 。
  さて このようなことは私の興味の範疇外である。しかし何故, 世間の教育部門に所属されている人達がこのような問題にアプローチしないのだろうか?
- ツールへの偏見/誤解 (2)
  これも討論の中で出てきたことだが, CASE ツールの利用が 'お絵書きツール' に留まっている例がかなりあるようだ。
  しかしそのことからだけでは ツールが悪いのか, それとも機能なり背景にある方法論を利用者が理解していないのか判断出来ない。
- データ辞書が意味があるか?
  → 無いよりはあった方がよかろう
  しかし, それが思考の支援をしてくれるとは限らない

06 田中 慎一郎 (たなか しんいちろう)
性別：男 年齢：34 血液型：A
会社：(株)SRA
所属：PSEL

プログラマにとっての道具とは?

1. はじめに

プログラマにとっての道具を考える場合, どうしても「現在プログラマが行っている作業をコンピュータで支援する方法にはどのようなものがあるか」という方向に思考が向かってしまう傾向がある。

この思考形態は当然ではあるのだが, おそらくそれは「今後」のツールの姿とは違っているような気がする。「新しい道具」というものは常に「新しい使い道」を同時に, いやむしろ後から産み出すようなものだからである。

コンピュータしかり, コピー機しかり。「プログラマにとっての今後の道具」もきっとそういうものになると考える。

私は現状のプログラマにとってどのようなツールがあると役にたつのかについてはよく分かっていない。また,他の多くの方がこの点については言及されることが予想される。このため,ここでは少し違った視点から考えを述べることにする。

2. 設問の解釈について

まず視点の提示を行う。それは「プログラマにとっての道具」とは一体何を意味しているのかについてである。

例えば,ここでは「プログラマ」の定義がなされていない。もし,ここでいう「プログラマ」が「設計者」等と役割を区別してのものであるとするならば,私には「新しい道具」を考えるにはそもそもちょっと違うような気がする。おそらく今後登場するツールは,要求分析/設計/コーディング等を現状のような形で区別するものではなくなると考えているためである。

このため,私はこの「プログラマ」を広義に解釈し,「プログラムを書くために必要な作業を行なうすべての作業者」と考える。

もっとも,これは「設計をしながらプログラムを書く」というような,現状よく行われている「自称プロトタイピング」とも少々異なる。それは単に「設計をしないでプログラムを書く」という行為を美しく言い替えているだけである。決してそういうやり方をイメージしているわけでも,また既存のソフトウェア工学を否定しているわけでもない。単に「新しい」道具であるからには,その用途は既存の作業工程とは異なる形になると予想しているのである。

3. 作業工程の変化について

作業工程が変わるであろうという予想は主に以下の2つの理由による。

a) これからのシステム開発は要求分析,要求定義,設計などいわゆる上流工程がいままで以上に重要なものとなってくる。

b) 従来からのソフトウェア工学は,現状,あるいは近い将来の情報技術の進展を十分に予測したものではない。

前者は,これからのシステム開発は,現行の業務プロセスを細かく分析し,まったく新しい業務プロセスとして再定義した上で開発することを要求されるようになると考えるためである。

また後者はまったく当然のことで,情報技術が現在のようなレベルにまで向上していない段階では,分業はもっとも効率の良い,確実な方法だったはずである。しかし現在では分業は必ずしも最良の方法とはいえない。この点から考えると,それをコンピュータで支援するいわゆる既存の CASE ツールもおそらくは充分なものではないはずなのである。今後はコンピュータで支援するソフトウェア工学はまったく違う形態をとると考える。

この点に関しては,残念ながらこれ以上の言及は私の知識からは行えない。ここは視点の提示のみとして,もう一度視点を変えて,システム開発の形態を大きく変更すると考えるプロセス・リエンジニアリングについて述べることにする。

4. プロセス・リエンジニアリングについて

プロセス・リエンジニアリングはなぜ必要か。それはこれまで「新しい道具」について述べてきたこととまったく同じである。「コンピュータに支援をさせる以上,従来のプロセスをそのままシステム化するのは無駄」だと考えるのである。

現状行なわれている業務は,間違いなくそれなりのプロセスにのっとって行なわれている。しかし,それが最適なプロセスであるかどうかはわからない。既存のプロセスを「なぜそのように行なっているのか」という視点でとらえ直し,新たなプロセスとして再構成することが有効になる場合が多いと考えるのである。

しかし,このリエンジニアリングは決して自明なものではない。従って,人間の思考を支援するために,プロセス・リエンジニアリングのための支援ツールが絶対に必要となる。前章で述べた予想から,実際のリエンジニアリング支援ツールはまったく違う形態となるとも考えているが,それでは何もできないことになってしまう。そこで,プロセス・マップの作成をまず手がかりにしてみたいと考える。

4.1 プロセス・マップの作成について

まず,要求定義に際して,現在対象となっている業務はどのようなプロセスを通じて行なわれているかを把握する必要がある。これは「どのように」ではなく「なぜ」行なわれているかの視点でとらえるのが必要だと考える。

事細かに,どのように行なわれているのかを把握することも決して無駄ではないが,プロセスのリエンジニアリングを目的とするならば,むしろそれぞれの行為が「どういう目的で」行なわれているのかがはっきりわかる形で記述できることの方が望ましいように思われる。

従ってこの際,対象業務内部のプロセスだけでな

く, 関連する外部事象との関係も表現できなければならないだろう。

4.2 プロセス・シミュレーションについて

また, 記述したプロセスの簡単なシミュレーションも行えるようにすることではないだろうか。

プロセスのシミュレートとはどういうことで, どのような支援が可能で, どの程度できることが有効なのかについては, 現在具体的イメージが持てていない。

5. まとめ

どうも, イメージが漠然としてしまっていて, うまくまとまったものにならない。「コンピュータ支援による新しいシステム開発作業のイメージ」あるいは「プロセス・マップの作成を支援する道具」について皆さんのアイデアが聞かせていただければ幸いである。

ワークショップに期待すること

この業界では, なぜがワークショップとシンポジウム, フォーラム, はてはセミナーまでもが同じものであるような誤解が存在するように感じている。特にそれが問題にならない場合も多いとは考えるが, その企画意図が存在する場合は, できる限りその意図を尊重する形で活動がなされるべきである。シンポジウムやフォーラムの聴衆, あるいはセミナーの受講者とは違う, 参加者の自主的活動による具体的な成果を作成できる可能性のあるような有意義な「ワークショップ」となることを希望する。

[ポジション発表に対する質問/コメント]

- プロセス・リエンジニアリングへのアプローチがもっと明確になれば, 確かに新しい道具のネタとなるかもしれませんね。
- 分業について, ソフトウェア工学に基づく支援について
  - プロセス・マップをどう活用するか?
  - なぜをどう記述するか, 活用するか?
- 情報処理学会で「BPR と Software 工学」の研究会みたいなのが一日あります (あった) が, どういう討論になったか関心があります。「ツールによる支援」はどうなるか, 可能か? 堀内一さんが出席されますが, 一昔前の DOA かなという気がする。金融システム業界では xx 銀行では結局失敗したというのが有名なので, 皆用心深くなっている。
- プロセス・リエンジニアリングに関する考えをもう少し詳しく聞きたい。

SIGENV–WS94 を終えて...

我々環境分科会が中心となったイベントも, これで通算何度目となったのであろうか. フォーラム形式でやったもの, 論文輪講の形式でやったもの, 若手の会のスタッフとなってワークショップ形式でやったもの. 結構, いろいろと行なわれ, 企画/参加者として毎回それなりの成果も獲得して来た.

今回も, 少しじっくりと環境について討論をしようという意図でワークショップが計画された. 若手の会はワークショップといってもちょっと趣向が異なるため, そういう意味では今回が始めての本格的なワークショップ開催となるはずであった. しかし, どうもこの前後はみんな急がし過ぎたようである. これは, 若手の会を除けば始めてびのイベントが連動していなかったことを見ても明らかであろう :-)

実際, 個人的にもちょっと準備不足だったかなという気がする. はっきりいってあまり記憶がない.

温泉に都合 6 回入ったこと. 長野なのに雪がないなと思っていたところ, しっかり雪は降ったが, その時東京はこれを上回る積雪だったこと. せっかくだからと差切峡近辺の観光に繰り出したが, 気合いが入り切らずにどうやら肝心なところを見ずに帰ってきたらしいこと. ポジションの発表をし, CASE ツールや CRC カードを使った作業を行なって討論の基盤を作っていったこと. そういうことは印象に残っているのだが, 討論内容はあまり覚えていない. 結構疲労が溜っていたようで, 夜の BOF ではいつもより良く睡眠がとれたこともあるが, 本格的な討論をする時間が十分にはなかったことが一番大きな理由だと思われる.

やはり現地で共通体験を基に討論を組もうとすると, 3 泊くらいの日程が必要なのであろうか. しかし, 3 泊あればあったで企画を盛り込み過ぎて同じような結果になりかねないことも事実であろうし, なによりも準備不足で他の人の発言をしっかり受け止めるだけの心準備が私にできていなかったことも大きい. 受け止める心準備がしっかりしていれば, もっと記憶が鮮明であっただろうに, 少々反省.

もともと今回のワークショップでは, ツールや環境の新しい支援について, 何か斬新な発想が得られれば面白いなと個人的には考えていた. しかし作業による基盤作りの時間が足りずに延長したため, あらかじめ注目点を設定した複数のミニ・フォーラム

(シンポジウム？) を設けるという当初計画は実現できなかった．このような時間的制約から，討論は簡単な感想を報告し合うに留まることとなってしまい，企画者側の立場としてはこの点がちょっと残念である．しかし参加者としての立場からすれば，温泉に浸かって十分な休養をとり，作業を通じてエレベータの動きの解釈を皆で考え，また新しい仲間といろいろな話ができたことは，とても良い時間が持てたと考える．

最後に．今改めてメモを覗いて見ると，「プログラマ中心主義」，「“どうしてか”が欲しい」などのキーワードが見られる．また，ミニ・フォーラムの希望テーマとしては以下のようなものが上がっている．

(1) 標準化とカスタマイズ．
(2) 発想の支援．
(3) なぜ CASE ツールが使われないか
(エディタと比較することで道具に必要な要件を検討する)．
(4) どこまで道具だてすべきか
(技術の空洞化ができる？)．
(5) CASE ツールの限界について．
(6) チーム活動支援．
(7) プロセスはどのように分けるか．何を支援か．
(8) 既存資産活用．
(9) 保守に必要なツール．
(10) 開発プロセスについて．
(11) リポジトリまたは情報の扱い．
(12) プロセス，工程．
(13) ツール，再利用，部品化．
(14) 道具の使いやすさ．
(15) 紙と鉛筆をどうやって越えるか．
(16) 良い道具とは．条件とか．

これらもまたとても興味深いテーマではあるので，できればまた近いうちに，同じような場が持てればと思う．

[追記] – 今回の宿の特徴 –

- 料金が安いこと
何と言っても 1 泊 2 食付きで 7,287 円 (税・サ，入湯税込み) の安さ．懇親会用に 3,000 円増しでお願いしたら，食べ切れないほどものが出た．名物はマツタケらしいので，季節が合えば最高かも知れない (山の中に「取るな」の看板がたくさんあった)．

  その他，会場での飲物等も，売店/自動販売機を使うことが前提なので，比較的安上がり．酒の持ち込みも，部屋で飲む分にはうるさくないようだし，そもそも大した金額ではない．

- 静かで景色が良いこと
大したことはないと言ってしまえばそれまでだが，空気も澄んでいるし，渓谷もなかなか絵になる．が，当然と言えば当然のように，夜出かけようにも行くところはない．むしろ近くの住民がここに出かけてくるようなところなのだ．

- 交通の便が悪いこと
東京から午後一に着くための足は，上野発信越線特急白山，新宿発中央本線特急あずさ 5 号，高速バス 1 本．後は自前の車だけという選択の狭さ．最寄りの駅からは村営のバスがあると言うので安心していたら，12 時の次は 3 時半という当然と言えば当然のスケジュール．タクシーも駅前に 3 台くらいはあるらしいが，やはり皆でまとまって宿の迎えを待つのが一番．

- コーヒーが飲めないこと
メニューにコーヒーはなかったようなので，コーヒー中毒の方は辛いかも知れない．コーヒーメーカー持参で行けば問題は解決するかも．

以上を利点とするか欠点とするかは人によって大きく変わるんでしょうが，温泉は 24 時間入れるし，カンヅメ目的で腰を据えて話をするにはなかなか良いとも言えそう．過剰な期待を持つことは危険ですが，予算の少ない中小規模の会合には向いていそうです．気が向いたら他の分科会の方々もぜひどうぞ．

07 武田 泰英　(たけだ やすひで)
性別：男　年齢：33　血液型：A
会社：(株) アドバンテスト
所属：ATE ソフトウェア統括部

プログラマにとっての道具とは?

私は，開発ではなくむしろ既存ソフトウェアの保守に近い部分を担当しています。そこで“プログラマにとっての道具”を，ソフトウェアの保守という視点から考えてみたいと思います。
保守作業の中で実施される作業として次のような項目があげられます。

1) ソフトウェアの不良改善/機能アップ

2) 保守作業の効率アップ/軽減

ソフトウェアに費やされるコストの大半が保守に割り当てられていることを考えれば, 保守を行なってゆく上で 2) が重要なテーマになることが十分理解できます。

保守作業 (不良改善/機能アップ) が右から左へスムーズに流れれば良いのですが, 大抵の場合, その流れを妨げる障害物が現れます。この障害物が除去できるような環境/道具が 2) を実現するために必要になってきます。実際に現れる障害物の例をあげてみます。

a) 機能が分からない
しばしば, 何を実現しているのか分からないことがあります (大きな機能ブロックが分からない場合, 個々のモジュールの機能が分からないなどサイズはいろいろあります)。主な原因はドキュメントの不足, 各種ドキュメントとプロダクトの不整合などです。ソースコードがいわゆる ball of spaghetti というのも要因の一つになります。ソフトウェアの修正に先だって, これらの不明点を明らかにする必要があります。これが大きく保守効率を低減します。

b) なぜそうなっているのか分からない
あるコードフラグメントが何をしているのかは分かるが, 何故そのようになっているのかが分からない。
機能上の制限があるが, 何故その制限が付いているのか分からない。
この何故を理解しないまま修正を行なうと, 後で思わぬ所に影響が出てしまうことがあります。これがまた, 大きく保守効率を低減させることになります。

c) 多重開発
機能追加を行なう場合, それに必要な部品が既にソフトウェアに存在する場合があります。が, これに気づかず, 同じ機能を果たすモジュールを新たにつくってしまうことがあります。無駄な作業をしたことになり, 保守対象を増やす結果にもなります。

障害物発生の原因を見てみると, ドキュメント, コードの貧弱さがかなりのウェイトを占めているように思います。一般的には, ライフサイクルの各フェーズの出力が適切でないということだと思います。良くない点を大まかにまとめて見ると,

- 不足
必要な情報が欠落している。このため内容理解が難しい。
情報が欠落したままドキュメントが完結している場合, 機能的なディグレードが起こる可能性がある。
- 過多
不要な情報まで記述されているので理解の妨げになる。
要/不要の選択を誤ると望まれないソフトウェアが出来てしまう。
- 曖昧
誤解が発生しやすい
- 乱雑
必要な情報を取り出すに時間がかかる。場合によっては情報が見つからない (情報の喪失)。
- 不整合
ドキュメント間で内容に矛盾がある。保守, 改版の過程でドキュメントがアップデートされないために生じる場合が多い。
- 何故が無い
そうなっている理由が書かれることがほとんど無い。

などが考えられます。 このうち, 特に, "乱雑", "不整合", "何故が無い" をカバーしてくれるものが "プログラマにとっての道具" として重要ではないかと考えています ("不足/過多/曖昧" については, 道具というより, むしろ手法で解決してゆくべき内容だと思います。"何故が無い" も手法の範疇かも知れませんが, これを積極的に支援している手法や道具を私は知りませんので取り上げてみました)。

では, 具体的に, どんな道具が必要かということですが…

まず一番重要なのは, "成果物を種々のフラグメントのネットワークとして扱えること" だと思います。このフラグメントの中には, "何故" を示す情報も含まれます。

成果物を細かなフラグメントとして扱うことの利点についてはいくつかの文献で報告されています。フラグメント溜りに対して適切な view を設定することで "乱雑" が解消されます。同じ内容を示す情報はすべて同一のフラグメントから導き出されます。これは "不整合" を解消し, 成果物の保守時間を短縮します。

次にフラグメントを扱う道具として, どのような機能が必要かを考えてみます。

まず, フラグメントとそれらを関連づけるネットワークがどのように構築されるかという観点から, フラグメント化の方向として“自由フラグメンテーション”が重要な機能になると思います。つまり上流から下流へ流れるようにフラグメントネットワークができるだけでなく, 下流から上流へ, 中流から上下流へとどのような方向へも自由に発展できるような構築メカニズムのサポートです。また, フラグメントの分割/統合, それに伴うネットの再編も重要です。特に保守の面から見ると, 再発見された機能/設計等の情報を保管/利用するために是非とも必要な機能になります。

フラグメント検索の面からは view の設定/変更が容易なことが不可欠です。一度設定したら変更がえらく大変というのでは, 作業の自由度を最終的には阻害してします。View を設定することで, 不足している情報が何かをチェック出来ることも必要な機能だと思います。これで, 保守効率向上のために残しておかなければならない最低限の情報が開発段階から確実に残されてゆくことが保証されます。

あまり具体的にすることは出来ませんでしたが, まとめると

(1) 成果物を種々のフラグメントのネットワークとして扱えること

(2) ‘何故’を示す情報の入力/保管/利用を支援できること

が, 私の考える道具です。

ワークショップに期待すること

現在, 開発環境の見直しなどを行なっています。今回のワークショップを見直し活動の参考と出来るよう, また, それにより見直し方針がより適切なものになるよう取り組みたいと思います。

[ポジション発表に対する質問/コメント]

- 保守を行う場合の「なぜそうなっているのか」という問題に, 私も常々悩まされています。こういうメタな情報の管理も確かに必要ですね。
- 自由フラグメンテーションをどのように作りだし, 管理活用するか?
- View が矛盾することはないか?
- なぜをどう記述するか, 利用するか?
- 保守・開発手順を標準化すべきでは?(管理技術の問題)
- 設計理由を残すとかって Design Rationale とかいう用語でしたっけ? CSCW の特集記事なんかで良くありますよね。ツリー形式で意見を記録していくツールがあった気がしますが。(実用化はどうか知りません)。
- 紙(仕様書, 設計書)に残すものもあるし, コードの中に書き入れたようがよいものもある。ユーザが出すニーズの理由, SE が設計した理由, プログラマが実装した理由などいろいろある。
- CASE をつくるための CASE とは?

参加後の感想

WS から戻ってから, ある現象に悩まされるようになりました。それは何かというと, 頭の中に ‘紙と鉛筆’ が出現してくるというものです。

WS の前はそこにあるもの(プログラマが扱うさまざまな情報)をどう見るかという点を考えていたわけですが, ‘そこにどうやって存在させよう’ という部分は考えていませんでした[1]。ポジション発表では, その部分の指摘をもらいました。そこで「うーむ」と考え込んでしまったわけですが, それが ‘よく使う道具は紙と鉛筆です’ という何人かの発言と反応してしまったようです。(反応の触媒は, 初めて飲んだ ワイルドターキーの 12 年とジョニ黒だったかもしれない[2])。

あらためて考えると ‘紙と鉛筆’ はすごい。それ自身 直接的にプログラムを作成するわけではないのですが, よく言われる ‘手法に依存しない’ とは, 紙と鉛筆のことをいうのかもしれません。

何のインターラプトもなく思った通りに描けます。モードの選択や, 図形選択などはいっさいいりません。どこでも使えます。一人でこっそり使うことも, 大勢でよってたかって使うことも出来ます。

> もちろん, それを後で見るとか, 流用して何かに使うとか考えれば紙と鉛筆だけでは, これまで CPU が解決してきた諸問題をむし返すことになりますが…

私は CRC カードを使ったグループでしたが, そこで活躍したのはやっぱり紙と鉛筆でした。

[1] 意識的に避けていた面もあります

[2] 渡邊さんご馳走さまでした

それを CPU に入力することになるわけですが, この時は, 伊藤さんの機転で次のような方法を用いることが出来ました。

(1) CRC カードに盛り込まれる内容をあらわす簡単な言語処理系をつくる。

(2) カード (紙) に書かれた内容を, エディタを使ってその言語で表す。

(3) 言語処理系で一気に入力する。

このような方法は, 紙と鉛筆を開発環境に結び付ける良い手がかりになるのかも知れません。

というわけで, 頭の中の '紙と鉛筆' が消えてくれるまで, 少し '紙と鉛筆' のことを考えて見ようかと最近はひらきなおっています。

08 後藤 浩　(ごとう ひろし)
性別：男　年齢：28　血液型：O
会社：(株) シスプラン
所属：営業技術部

プログラマにとっての道具とは?

1. プログラマの "仕事"

プログラマにとっての "道具" について考える前に, プログラマの "仕事" について確認する。以下に大別できると思う。

- 要求分析行程 (予備調査および要求分析作業)
- 設計行程 (基本設計および詳細設計)
- 導入行程 (コーディング, テスト, コンバージョン)
- メンテナンス行程 (修正作業および拡張作業)

ソフトウエアの開発行程を掲げたにすぎないが, プログラマ=ソフトウエア開発者と理解しているので, 私にとってのプログラマの "仕事" というと, こうなる。

各方面で, リストラが叫ばれる昨今, 「ソフトウエア開発行程についてもリストラしたい」などと考える今日この頃である。

2. プログラマにとっての "道具"

ソフトウエア開発の各行程を支援あるいは自動化してくれる "道具" について考えるわけであるが, 個人的には「グループウエア」の必要性を第一に感じている。「グループウエア」の定義については, 一応「ひとりひとりの生産性向上と仕事の連携を実現し, グループ全体をパワーアップするソフト」ということにします。「グループウエア」自体は, その枠組みを提供するにすぎないので, その上にどのようなシステムがあれば良いか, 以下に記します。

2.1 過去の経緯が分かる "道具"

予備調査および要求分析を開始する前に, 似たようなシステムの開発経緯あるいは同一顧客についてのシステム開発経緯が電子的に取り出せるシステムが欲しいです。できれば自社内のみならづ, 競合他社が収めたシステムの開発経緯までが取り出せるシステムがあると大変ありがたいと思います。

2.2 現在の開発状況が分かる "道具"

開発チーム内の進捗状況がリアルタイムに取り出せるシステムが欲しいです。

3. プログラマにとっての "道具" について考えてみて

ポジションペーパとして与えられた「プログラマにとっての"道具"」について, 整理を試みたが, 上記の結果である。

参加募集要項を再び見直し, 改めて参加資格を自分に問うてみたがかなり怪しい部分がある。副題の「—自然の中で共通の課題を通して環境を考える—」に重きを置き, あえて参加申込をお願いする次第であります。

ワークショップに期待すること

私自信は, 設計, 製造行程の仕事をやらなくなって数年が過ぎるが, 生産性の向上という観点から, 当社でも下流行程の CASE ツールを導入し現在使用している。今回のワークショップでは, 開発中の CASE ツールを使用できる環境があるとのことなので, 是非使用してみたい。

また, 仕事上で Windows 上の下流 CASE の販促を担当する予定があるので, 漠然とした言い方ではあるが, 「使用者からみた場合の使い易い CASE ツール」について何等かの結論を持って終わりたい。

参加後の感想

ハライド・エレベータ社 御中

(株) シスプラン
後藤　浩

拝啓 時下益々ご清祥のこととお慶び申し上げます。

さて, この度私どもは御社が取り組んでおられます「エレベータ制御」について「SIGENV-WS94」の場で「エレベータ問題」として取り上げ, 参加者全員でこの問題に取り組みました。取り組み内容についてご興味がある場合は別途お問い合わせください。報告書を郵送させていただきます。

「エレベータ問題」を熟読していて疑問に思ったことがあります。御社のエレベータシステムの発展にもお役に立つものと信じ, ここに書き留める次第です。「上下左右に移動できるエレベータはないものか?」というのが私の疑問です。大きなビルで, 複数号機のエレベータがある場合, 1 階で 1 号機に乗って上昇し, 5 階で降りたら 3 号機の場所だったといったことになっても特に問題は無いように思えます。

単純に考えて, 10 階建てのビルでエレベータが 5 台あった場合, 45 エレベータ分の空間が空いていることになります。地価高騰の折り, まことにもったいのないことではありませんか。

上下左右に移動できるシステムを完成させることにより, 数倍の移動効率が得られると考えます。

さらに, 空いた空間を倉庫に利用します。倉庫用のエレベータは常に人間用のエレベータの邪魔をしないように移動しなければなりません。

もっと発展させれば, n 号機のエレベータは基本的に動かず, お店が開くというのはどうでしょう。朝の早い時間は, キオスク号がいます。新聞, サンドイッチ, 飲み物を売ります。10 時から 11 時 30 分までは, オフィス用品のサプライ号がいます。コピー用紙や FD 等を売ります。お昼時は当然, 弁当号です。夕方には花でも売りましょうか。このシステムを具現化する際には, 一言ご相談ください。

というわけで御社のご発展を祈ってやみません。敬具

09 塩谷 和範 (しおや かずのり)
性別 : 男 年齢 : 42 血液型 :
会社 : (株)SRA
所属 : PSEL

プログラマにとっての道具とは?

プログラマにとっての最低限の道具とは, 昔ながらの紙と鉛筆だろう。これが全ての物書きの基本であり, これに何をつけ加えていくかで便利さが増してくる。プログラミング作業だけについて考えてみると, 言語処理系 (コンパイラ (アセンブラ) ・リンカ) ・テキストエディタが最低限の道具であろう。

1. 道具とは?

道具は, 使う人に馴染み使うにつれて使い易く { なる/できる } ものである必要があるし, 馴染んだ道具は長く使い続けることが出来なければいけない。このような観点からあまり複雑な道具は好ましくなく, 単純な組み合わせ可能な道具が良い, というのが成功した unix のアプローチであった。unix 方式の問題点は組み合わせ機構に制約 (パイプと shell インタプリタ) があり, テキストベースのユーザインタフェースには適するが, 図形ベースあるいはウィンドウベースのインタフェースが全然考慮されていないという点だった。また, パイプと shell の環境変数による文字ベース情報の受渡しというあまりにも単純なデータ型 (基本的には文字型) しか定義出来ないという問題点もある。

これを改善する方法の一つが, shell 機能にカスタマイズ可能なウィンドウ機能を取り入れることであり, 最近 Unix 上で Tcl/Tk と呼ぶ新しい shell インタプリタが急速に普及しつつある。Tcl/Tk は, Tcl と呼ぶコマンド言語インタプリタと, Tk と呼ぶ Motif 風なウィンドウ表現をもつウィンドウライブラリから構成する。Tcl は, 型定義が出来るインタプリタ言語で, 内部関数の他に外部プログラムの起動機能と, Tk ライブラリを呼び出し Tk ウィンドウ (Wish) 上に領域を定義し, 入出力を行なうことが出来るようになっている。Tcl スクリプト同士はお互いに send 関数を通じて通信可能である。

また, 別の方法として Smalltalk に代表されるようなインタプリタベースの総合プログラミング環境を使用し, 特にオブジェクト指向の機能などにより差分プログラミングあるいはライブラリ関数を利用したプログラミングを行ない, 最小限の労力で問題領域に特化した道具を作っていくやり方もある。この方法は, オブジェクト指向で無くても, Lisp 系のシステムでも同様で, Emacs の利用者がこのようなカスタマイズ化を好んで行なっている。この方式の問題点は, Smalltalk などの GUI ベースの総合環境以外では, GUI 表現に制約がある。ただし, 多くの場合基本環境が巨大であるため, 製品化するには何らかの形で未使用部分をそぎ落とす工夫が必要である。また, 特定の言語環境に閉じこもってしまいがちになる弊害もある。

2. 何をするための道具か?

さて, 何をするための道具か ? という視点は重要である。プログラマであっても日々の作業の多くは,

各種の文書 (事務連絡 / 手続き・仕様書・技術情報・ソースコード・データ・開発履歴・報告書など) の処理 (検索・参照・更新・合議・承認・転送など) , 上司・同僚・部下・顧客・外部の情報源などとの通信連絡 (会話も含む) など多様な作業も行なう必要がある。当然ながら, これら多様な作業を通じて, 発信・受信する情報の多くがプログラミング作業に関連している。このように作業環境全体を考える時には, これらの作業も含めて, 一貫した流れの中で出来るだけ使い易い道具が提供されていることが理想となる。その意味で, 現在私が使用している NEXTSTEP 環境は, 一つの現在入手可能な最高のプログラミング環境ではある。その理由の一つは, 上記文書処理機能が優れていることにある。また, Objective-C というマイナーな言語しか対象としないが, GUI インタフェースを作成するための GUI 構築ツールとオブジェクト指向のクラスライブラリが揃っていることもある。また人によっては, Smalltalk 環境あるいは Lisp/Emacs 環境がそうであろう。

3. 今後あるべき道具とは ?

しかし, それでもなお理想のプログラミングのための道具とするには問題がある。それは一つには, これらの環境は, unix あるいは DOS 環境ほど広く普及した環境ではないこと。また, 普及には各マシンベンダーの政治的な思惑あるいはライセンスが絡み, なかなか多数のベンダが採用するには至らないこと, さらに先に述べたような問題点があるということである。

このように考えると, プログラマにとっての理想の道具とは, なるべく多くの環境かで動作し, かつ最低限カスタマイズ可能な道具。自分で出来ない場合あるいは時間が無い時は, カスタマイズサービスも受けられるものということになろうか。そして, 出来れば, 文書処理作業・通信作業も取り込めるような環境, あるいは道具の組み合わせが可能な物と言うことになる。

このような道具の候補としては, 先に述べた Tcl/Tk が有力ではないかと思う。その理由として, ソースコードが公開されている PDS であること, 既に広く広まりつつあり各種の機能拡張も続々と公開されていること, これを使った製品・サービスが入手可能であること, オブジェクト指向拡張と日本語化がなされ共に公開されていることである。 Tcl/Tk 製品は基本的にインタプリタ言語で記述されているため, 市販 /PDS 製品を問わず作業領域用にカスタマイズ可能である。将来的には, これと組み合わせた, 総合的な文書・通信機能を含む総合プログラミング環境が構築できると考えられる。

もう一つの候補としては, データ中心の統合環境である PCTE (Portable Common Tool Environment) がある。これは現在, ISO 標準として提案中のリポジトリ標準で, データ型の定義を中心にプログラムすることにより, プログラム間でのデータ共用を促し, データとプログラムの可搬性と再利用性を向上させようと言うものである。現在はデータ中心統合機能に, 制御統合機能として COSE の共通メッセージング機能を取り入れた PCTE が提供されつつある。PCTE の利点は, 特定のベンダに偏らない公開された仕様に基づく環境であるということで, ベンダ各社ともこれを採用・サポートする姿勢を見せている。唯一のソフトウェアリポジトリ仕様として今後普及するものと思われる。

PCTE 上のツール例として, 前述した文書処理, 特に検索処理を支援する HyperWeb という, ハイパーテキストベースの文書処理統合環境が製品化されている。これにより, ソフトウェア開発に関わるプログラム・データを含むあらゆる文書を, ハイパーリンクにより関連付け自在に参照更新が可能となる。この機能の拡張として C プログラミングの編集 – コンパイル – デバグ作業を支援する環境も提供する。

PCTE に期待することは, 国際語化と PCTE が目指すプログラム間のデータ共用の促進である。このためには, 各ツールベンダが, 積極的に各々のツールのデータ構造をスキーマ (データ構造定義) として公開し, このスキーマを共通に使っていくような仕組みが必要となる。

既に, この 2 つの候補 Tcl/Tk と PCTE を合体させた製品 (Tcl/Tk の PCTE 上への移植と開発ツール化) も作成されている。

ワークショップに期待すること

久しぶりのソフトウェア環境に関するワークショップなので, 参加各氏とソフトウェア環境あるいはその上の “道具” について意見を交換したい。できれば, ソフトウェア技術者の必要とするソフトウェア環境, あるいは道具の今後のあるべき姿について意見をまとめ, ソフトウェア業界に提言を行いたい。

(個人的には, OpenStep が広く普及すれば良いと思っているのですが, それでは自己満足に過ぎないので, もう少し広い視野で考えて見たいと思っています。)

[ポジション発表に対する質問/コメント]

- PGMを書くようなこまごまとした作業と 図を書くような大まかな作業の切り分けは 私も大事だと思います。
- 「環境に依存しない」ということは, いままで考えていなかったので参考になった。
- 個人として扱う道具には同意できるが, グループとして扱う道具の理想について どう考えているか?

10 伊藤 昌夫 (いとう まさお)
性別：男 年齢：34 血液型：AB
会社：MHI エアロスペースシステムズ (株)
所属：技術部開発グループ

「見よ, 私は君たちに超人を教える! 超人は大地の意味である。君たちの意志は云うべきだ, 超人を大地の意味たらしめよう!と。私は君たちに懇願する, 私の兄弟達よ。あくまで大地に忠実であれ, そして, きみたちにもろもろの超地上的な希望について話す者たちの言葉を信じるな! 彼らがそれを知ろうが知るまいが, 彼らは害毒者なのだ。」

ニーチェ「ツァトゥストゥラ」序説

プログラマにとっての道具とは?

ツールビルダの立場としてどう思うかと云うことは, 何れ別の場所で, お目にかけることができると思います。

このテーマを出した一人として, 解題を少ししておきたいと思います。

私は, プログラマ中心主義者です。何もこう力を入れることはないのですが, 多くのソフトウェア開発にまつわる問題は, プログラマを中心に考えることで, 解決可能であると思っています。

私が描く図式に於ては, 分析/設計者がいて, コードを書くだけのプログラマがいるのではなくて, 全てに於てプログラマしかいないのです。

ソフトウェアの歴史を振り返ると, 必要な人たちが最初プログラミングをしていた。例えば, 流体解析をしている人が, 計算機の有用性を認めてプログラムを始めた。それから, 分化が始まりました。プログラムは専門の職業となり, 分析/設計及び検証の必要性が認められて, それぞれが専門として独立した。そうして, プログラマはコードを書くだけの人となった。

私のプログラマ中心主義とは, 再度プログラマに全ての役割を担わせようと云うことです。それは, 歴史の流れを遡ることではありません。これら歴史から得られた知見を元にして, 再度プログラマがソフトウェア開発全てを担えるように考えようと云うことです。そこでのプログラマには, コーディングはもとより, 分析/設計/検証と云った能力もまた同時に要求されます。それを有している人間が, プログラマ中心主義の私にとっての, (プロの) プログラマです。

従って, そこから導き出される "道具" とは, 最終的にプロとしてプログラマが使うと最大の効果を挙げ得るものであって, それはソフトウェア開発の全ての段階に及びます。また, そのようなプログラマが存在し得るために, 誰が使っても同程度の能力を発揮すると云うのは, 道具として不十分であり, プロのプログラマが使うことによって最大限の効果を発揮しえるものでなくてはなりません。

そうだとすれば, どうすれば良いか。それが**今回のワークショップに期待する**ことです。

ワークショップに期待すること

先ず一番に, 楽しく議論できたらと思います。

計算機環境の議論の多くは, 管理者やツールビルダの立場からの意見が多いようです。そういった人達は, ある意味で環境についての知識が多いからそうなのでしょう。

しかし, 使う側の立場も, 知識はなくとも考えることはできる筈ですし, ユーザと云う意味で, 日頃ドキュメントやプログラムを作っているからこそ, 唯一考えられる立場の人であると思います。通常は, 仕事に追われて余り考えることはできませんが, 今回のまとまった時間の中で, 議論を通じて, 色々考えると云うそのプロセスは, 何等結論が出せなくとも, 我々にとって実りあるものだと信じています。

[ポジション発表に対する質問/コメント]

- すべてを行うプログラマはすべての仕事を一人で行うのか?
- 職人とのちがいはあるか?
- ツールの分析にはどんな手法を利用したのか?
- CASE(IPSE) に期待するプログラマに与える刺激とは?

11 浅野 友広 (あさの ともひろ)
性別：男　年齢：26　血液型：O
会社：MHI エアロスペースシステムズ (株)
所属：技術部開発グループ

プログラマにとっての道具とは?

私がプログラムを書くときに使用するツール, コマンドを頻度の高い順に挙げてみるとエディタ, コンパイラ, デバッガとなる[1]。 これらはみな, コーディング, テストといった下流工程で使用するツールばかりである。分析設計といった上流工程ではツールと呼べるようなものは何も使っていない。

しかし, 分析設計をやっていないというわけではない。標準や手法に基づいた分析設計は行っていないが, プログラム構造やアルゴリズムを考えるときなどコードにする前に紙に書いて頭の中を整理したりする。

会社のワークステーションには有名な分析設計用のツールがいくつか入っているが, それを使おうと思ったことは一度もないし, 誰かが使っているところを見たこともない。

何故だろうと考えてみてると, 使い方をマスターするのに時間がかかるということが先ず思い付くが, それが上流ツールを使用しない決定的な要因ではないと思う。ツールを使うことは少なくとも鉛筆を使って紙に書くことよりは難しいが, 使用法のマスターの難度は程度問題である。例えば, 私は入社当時エディタの使い方に慣れるのにも苦労した。しかし, 何故エディタを使うことを諦めなかったかというと, エディタが使えなければプログラムが書けないからである。分析設計ツールを使わなくてもプログラムは書けるが, エディタを使わなければプログラムは書けない。

何が言いたいのかというと, エディタの使用は最終製品であるプログラムの作成に直結しているが, 分析設計ツールを使ってもプログラムの作成には直接結び付かないということである。DFD やモジュール構造図からスケルトンコードを生成する CASE ツールはあるが, 直接実行イメージを生成するものを私は見たことがない。最終的なコードはスケルトンに手を加えなければならないので, 上位との不整合性が生じる可能性もある。そのためコードの変更をすると上位の分析設計情報も変更しなければならない。この二度手間が面倒くさい。

では何故二段構えになっているのだろうか。何故コードの上流に分析設計手法及びノーテーションがあるのだろうか。それは, 現在のプログラミング言語があまりにも人間の思考パターンに合わないものだからではないだろうか。私は普段, 複数人でプログラミングをすることはあまりないが, それでも人にコードの説明をするのは大変だと感じている。また, 人が作ったプログラムの保守を担当したときはその逆のことも感じた。プログラミング言語で書かれたコードを一目見ただけでは, 頭の中にイメージを作り上げることはできないからだ。しかし, DFD を見れば我々はある程度イメージをつかむことができる。現在のプログラミング言語はオブジェクト指向などを取り入れて人間の思考に近付いているのかも知れないが, まだまだコンピュータに優しく人間に厳しい。

それでは私が開発時に書く手書きのメモは DFD と同じ力を持っているのだろうか。両方とも人間に分かりやすい方法で表現されているが, メモの方は書いた私にしか分からない場合がある。なぜなら相手との間に統一された表記法の約束がないからだ。また紙に書いた非形式のメモを計算機に取り込むことは困難である。

以上の様に考えてきたら, 人間の思考パターンに近く共同作業者と意志の疎通を容易にできる表記法を, 最終製品に直結できるような環境が必要だと思った。つまり, DFD の表現力を持ち分析設計フェーズをもカバーすることができる様な人間に優しいプログラミング言語[2]があったとしたら, それはプログラマにとって強力な道具となり得るのではないかと思った。

ワークショップに期待すること

私は今回のワークショップで使用する CASE ツールの開発に携わっている一人なので, ツールの使われ方を知りたいし, 実際にツールを使った人のいろんな意見を聞かせてもらいたいと思う。

[ポジション発表に対する質問/コメント]

- 現在の PGM 言語が人間の思考パターンにあわないというのは, 私も同感です。プロシージャを書かないことが一つの鍵であると私は考えます。
- いつ手慣れたツールから新しいツールを使おうと決心するのか。

[1] ここで“プログラムを書く”ということは詳細設計に基づく単なるコーディングという意味ではなく, 要求に基づく分析設計も含んでいる。

[2] 言語とは呼べないかも知れない

- 設計分析ツールは不要か?
- 生産性についてのメジャーがないとなると, ツール自体の評価は?
- 人間の思考パターン... はツールとは別に検討しなくてはいけない分野でしょう。
  何で書いてあっても, (プログラム言語, 仕様記述言語, 図) わかりにくいですよね。後で出てきたけど, プロトコルを求めて, ルールなり何なり決めておかないと。

参加後の感想

今回のワークショップに参加することによって, 開発者が日頃から感じている現状の開発環境に対する不満, 要望, 期待をある程度知ることが出来た。以下に印象に残っているものを挙げてみる。

1. CASE ツール使用による開発の自由度の減少

自らが開発している CASE ツールの旧バージョンを用いて, 新しいツールを開発している人の話によると, その CASE は作業手順を厳しく定めていて, それから外れるような行為は一切出来ないとのことだった。

また, 現在自分流のやり方でメシを食えているので, そのやり方を変えることを強制されるようなツールは使いたくないという意見もあった。

自分はどうかと考えてみると, 仕事のやり方を束縛されることは耐えられないことだと思うし, もし, どうしてもある決まったやり方に従わなければならないとなると, 仕事の能率はガタ落ちになるだろうと思う。しかし, 現在の自分のやり方が最良の方法であるという保証は何処にもないし, 自分が全ての手法の経験があるわけでもないので, 自分のやり方に固執するのは良くないということは理解している。

以上のように考えると, 多様な開発方法の中から自分に合ったものを選択でき, それを更に自分が使いやすいようにカスタマイズ出来るツールが必要だと思った。

2. ツール使用によるプログラマの阻害感

ツール使用を好まない理由として次のようなものがあった。それは, ツールによって多くのことが自動的に行われてしまうと, 逆にプログラミングの楽しみが無くなってしまうというものである。この意見には個人的に大いに共感した。何でもかんでも, ツールに任せてしまうということは, プログラマにとっては寂しいことである。

実際の経験はないが, 自動生成されたスケルトンコードを使うことには違和感がある。仮にそれを基にプログラミングを始めても最終的には全て自分の手で書き換えてしまうような気がする。

しかし, 我々は趣味でプログラムを作っているわけではないので, 効率良く開発できる環境があれば, それを使わなければならない。プログラミングによってお金を稼いでいる以上, 自分の感情は押さえて開発効率を優先するべきである。しかし, プログラマも人間である以上, 潜在的な感情は開発に大きく影響を与えると思うので, その辺りも, 今後のツール提供者は考えなければならないと思った。

3. 保守に必要なツール

主に保守を担当している人が, 既存のプログラムを解析する場合に, プログラムの動きは把握できても, 何故そうなっているのかということを理解することが難しいと言っていた。

自分も保守を担当したときに全く同じことを感じた経験がある。この何故を理解していない状態で, プログラムの改修を行うと, 無駄なコードを書いてしまったり, 予想外の所に不具合が発生しまうことがある。

何が原因かと考えるとドキュメントやコードの貧弱さが挙げられるが, この二つがしっかりしていさえすれば, 何故を容易に見つけ出すことが出来るようになるとも思えない。

何故そういう設計やコードを書いたかというのは, 開発を担当したプログラマの経験, その時のプログラマの精神状態に大きく影響されると思う。開発時のプログラマの本当の意図を理解する為には, 以上のことも考慮にいれなければならないと思うが, そこまでをドキュメントやコードという形で残すのは不可能だと思う。

画期的な良い方法は浮かばないが, 思いつくところでは次のことが考えられる。

- ドキュメントの表記法やプログラミング言語の改良によって, それらの表現能力を高める。
- プログラムを解析して多くの視点 (機能, 状態, データ) からプログラムを眺められるようにする。
- 開発者と保守担当者との情報交換を促進する機構を整備する。

4. ツールの速度

ツールが遅いと，かえって思考の妨げになってしまうという意見があった。ツールに限定しなくても，レスポンスの悪い環境は利用者にストレスを与えてしまう。自分自身はそれほど遅い環境で仕事をした経験はないが，負荷の高いときなどに，少しでもコマンドのタイムラグがあると思考が途切れてしまう場合がよくある。

しかしプログラムを書く側としては，ハードウェアの性能の向上がめざましい現在では，プログラムの効率は，それほど重要ではないと思っている。効率を重視してプログラムの可読性を犠牲にするようなことはするべきではないと思う。

ツールが遅いというのも，結局は無駄な入力項目やチェックが多いのではないかと思う。

5. 発想支援

ポジション発表や討論の時に“プログラマの脳を活性化する”とか，“知的な仕事をしたくなる環境”という言葉が何度か聞かれた。

非常に漠然としていて，具体的にイメージしにくいものだと思った。

しかし，上記の言葉を発想支援と解釈すると，もう少し具体的なイメージを浮かべることが出来る。

私が知っている範囲では，発想支援ツールは何もないところからアイデアを出すことを支援してはいない。ある程度出されたアイデアを分類，整理し，いろんな視点から眺めることによって，新たなアイデアを促すものだと思っている。

しかし，最初のアイデアを出すというのは非常に重要かつ難しいことで，それさえ出てしまえば，後は次々に思い付くということがよくある。

何もないところから最初のアイデアを出すという部分は個人差が大きいし，何故思い付いたかなどということをノウハウとして言葉にすることは難しい。本来，発想方法を方法論化することは不可能かも知れない。

しかし，そう言っていては何事も進まないので，まずはプログラマにヒントを与えるようなレベルでの支援を実現していくことから始め，徐々に高度な支援にしていく様なアプローチをとるのが良いのではないかと思った。

12 **直江 魅音子**　(なおえ みねこ)
性別：女　血液型：AO
会社：(有) ネクストファウンデーション
所属：金沢事業所

プログラマにとっての道具とは?

プログラマにとっての道具といえば，まずCASEツールが思い浮かびます。しかし正直なところ，私はCASEツールを使ったことというか，さわったことすらありません。

また，現在出ているCASEツールに対する知識もほとんどありません。それは，CASEツールが非常に高価で，中小企業ではなかなか購入できないからです。(もっとも，後の知識がないに関しては単に勉強不足のせいですが) そんなわけで，望むのはもっと安価なCASEツールを！です。

と前置きが長くなりましたが，では実際にどのようなものが欲しいのかです。本来，CASEはシステム設計のためのものでしょうが，「プログラマの道具」ということで，ここではSEではなくプログラマの立場から上流よりも下流工程を中心に述べます。

1つは，すでにあると思いますが，設計時に論理上の矛盾点を検索する機能です。例えば，図なり日本語なりでシステムに必要なすべての機能と動作，それに伴う条件と関連づけ，状態推移等を入力すると，論理上の矛盾点や条件不足などをチェックしてくれるものです。フローやモデル図を書いても，人間の場合「完璧」は無理で必ずどこかにヌケや矛盾点が出てくるものです。

2つ目は，オートコーディング機能です。

これは，モジュール設計書 (図) を指定した言語に落す機能です。つまりは「力仕事」の軽減です。

プログラマはモジュール設計までで，後は勝手に言語に落してくれるとかなり作業量が減るのではないかと思います。(ただしここで問題なのは，効率良いプログラムにするためには，かなり細かいところまで指定しなくてはならず，結局のところ従来の言語に近いような書き方をしなければいけないであろうということです。) 設計したものを言語に落すということでは，OBJなどの形式記述言語で書かれたものを，矛盾点がないか検証した上で，実際使用する言語にトランスレートする機能や，オブジェクト指向で設計したものを非オブジェクト指向言語にする機能もあるといいですね。(巷ではオブジェクト指向がはやっているとはいえ，実際行なってる仕事は従来の手続き型言語

が主ですから。) まとめていってしまえば, 乱暴ないい方かもしれませんが, プログラマにとって本当に必要なのは, ブレーン (頭脳) とセンスと経験です。
ですから, プログラマにとっての "道具" は, 余分な力仕事を排除し, 本来本当に必要であるそのブレーンに刺激をあたえ, 活性化するための手助けをしてくれるものだと思います。

余談ではありますが, 私個人が「今」欲しい "道具" は, UNIX 上で使える, よいエディタと機能豊富なデバッガーと多彩な数値計算ライブラリと使い物になる X の GUI 構築ツールです。

ワークショップに期待すること

私は, 現在事業所で 1 人で仕事をしております。そんなわけで, 普段会話をする人もなく, 外からの刺激というものがほとんどありません。
ですから, 設計やプログラムを組む時もどうしても自分の知識と経験の中ですますしかなく, できたものに対して本当にこれでいいのかと絶えず疑問に思っております。たまに, 客先に出向し複数人からなるプロジェクトに携わった時など, 他の方の考え方に刺激を受けたり, 人と会話することによって出てくる自分の発想に驚いたりして, 他人との接触がいかに大切か思いしります。1 つの問題に対して取り組み方, 視点等を議論する今回のワークショップは, そんな私にとりましては, 喉から手が出るほど参加させていただきたいものです。
ただ心配なのは, 私は **プログラマにとっての道具とは?**でも書きましたように, 最新の CASE ツールに対する知識はほとんどありませんし, ちゃんとした SE 教育 (?) も受けておりませんので, とんちんかんなことを言って皆様の足をひっぱることになるのではないかということです。
しかし, 許されるものであれば, オブザーバーとして皆様の意見を聞くだけでもよいので, 参加させていただければ幸いに思います。

[ポジション発表に対する質問/コメント]

「力仕事」の軽減にもっとも重要な命題だと私も考えです。もっと楽をしましょう!!

設計時に「ヒント」を与えてくれるような CASE は確かにほしいと思う。

13 桜井 修 (さくらい おさむ)
性別：男　年齢：33　血液型：O
会社：池上通信機 (株)
所属：ソフトウェア推進室

プログラマにとっての道具とは?

私は, 主に機器制御用のマイクロプロセッサのソフトウェアを作っております。会社が放送関連機器メーカーであるため, それらの制御ソフトウェアのプログラマです。放送用パラボラなどのモーターの駆動制御プログラムや人間のインタフェースとなる操作部のプログラム。さらに最近ではビデオ/オーディオのデジタル化にともなって, 映像/音声それ自体もコンピュータデータとしてあつかうプログラムを作っております。

そのため, 大型コンピュータの経験もなく, 事務処理向けのソフトウェアは作ったことがありません。さらに消費者向けのパッケージソフトウェアやツールのためのソフトウェアもないのです。結構, ユーザーには「見えない」部分のソフトウェアで, 狭い分野で「専門的」にメシを食っていたわけです。

それでも最近は, マイクロプロセッサが高速となり従来ハードウェアロジックで組み立てていた回路を CPU で処理させることでより小型化, 低価格化が可能をなってきたしだいです。さらにパソコンも低価格となって, グラフィックな画面となり「いや, これからはソフトウェアだね。」という感覚でいままでハードウェア技術者を養成していたのを新人や若いメンバーにどしどしソフトウェアを勉強させはじめ, 実際, ソフトウェア技術者をまずは数だけ増やしはじめています。そうなってきて, 自分たちが使い慣れた "道具" についてあらためて考える機会が多くなってきたのです。

道具として形あるもの, 商品としてあるものは簡単に上げることができます。「エディタ」「コンパイラ」「デバッガ」「コンピュータ」などがあり, これらの道具はそれぞれ進歩してきたし, 私にとっても貢献してくれています。だいたい改善されてきた要素を思い付くままに上げてみると, よりプログラム完成に近いところに集中しているのが分かります。

1) 使いやすくなってきている。
　→エディタなどは, 先輩が魔法のような暗号を覚えてミニコンピュータへ入力しているのが「素晴らしい」と感じたのは新人のときであっ

ただろうか。いまでは「マウス」なんかで見たままで操作して，キーボードをいじればそのままキーインされる。いや，昔は違っていましたよね。

2) 処理時間や応答速度が速くなってきている。
→この要素への改善はプログラマにとってもっとも貢献できるものと感じている。このため，深夜のコンパイルもなくなり，翌日朝にならないと結果が分からないようなことは最近なくなった。

3) 安くなってきている。
→コンピュータが安くなって個人専用に使えるようになったのも大きな進歩である。
コーディング，パンチ入力，コンパイル作業が手元で出来るのだがら驚きである。しかし，リストとにらめっこして赤ペンで修正，パンチ入力を待つこともなくなったが，これによりプログラマが本来の「プログラミング」作業に集中できることが可能となったのだろうか。

4) 環境に左右されず，どこでも使えるようになってきている。
→コンピュータの小型化がそうさせた。現場でノート型パソコンで作業ができるようになり，効率が良いように感じる。

5) 専門的な知識を必要としなくても使えるようになってきている。
→プログラムが走る CPU の知識がなくても，アプリケーションができるようになってきている。いや，コンピュータそのものの知識も怪しい人間がアプリケーションが提供するスクリプト言語でびっくりするグラフィカルなアプリケーションを作ってしまう。

しかし，昔からプログラマとしてその作業の中で“道具”として進歩していないところもあります。そしてその中で特に「何とかしたい！」という気持ちが現われたところに「道具化」できるチャンスがあるように思えます。それは設計段階のものやテスト段階のところです。これを“道具”と呼ばれるようにするためには，もっと機能を明確化しなければならないのでが，現在のところそれらはたいてい「メモ」で作業をしているところです。今後，上記の道具たち (よりプログラム完成に近いところの道具) と同じような環境で仕事ができる姿にもっていけなければならないと感じています。

- ユーザーの要求を仕様としてまとめ上げること。
- 仕様やその動作環境を検証すること。
- 仕様を分割して，プログラムとしてのまとまりを作ること。
- そのまとまりに「状態」を作ること。
- 各プログラムのフローをつくること。
- プログラムフローからプログラムコードに落とすこと。
- テスト項目を荒い出し，チェックする方法を見つけること。
- テストを行なうこと。

その解答の１つが「CASE」であると考えます。しかし，現在出回っているツールをストレートに賛成できないのは。なぜでしょう。明らかにいまやっている自分の作業のやり方と違うことを要求しているように感じてならないからである。内容的には同じかもしれないが，機能の「状態」の持ち方や面倒な図や変数を定義しなければならないみたいです。なぜ，わざわざ面倒な「道具」に合わせなければならないのかと思うのです。いや設計方法が違うのだから仕方がないと言われればそれまでですが，それでも自分流のやり方でプログラムコードは完成してメシが食えるのだからこの方法を“道具”にできればよいなあと思うのは，私だけではないはずです。

しかし，最近になってつっぱってもいられな状態であると感じはじめた。それは，プログラミング自体の道具が結構そろってきているため，プログラマがますますプログラミングそのものに技術を集中しなくてもよくなってきており，いきおい，その設計の能力を早い段階から付けなければならなくなってきているからです。数年前だとプログラミングそれ自体の道具がそろっていないため，プログラマも 2〜3 年かけないとプログラミングが満足に出来なく，それくらいの経験を有すると自然と「設計する力」が付いてくるものです。そしてその経験がメシの食えるプログラマとなっていくように思うのですが，最近では，いきなりコンピュータはあるは，エディタは簡単に使えるは，コンパイラも早いは，デバッガも万全で，すぐに「設計する」ことが必要とされるようになってしまったのです。生産性のアップです。

それでも，分析や設計の作業はそう簡単に取得できるものではないと思っていますが，「CASE」がその点を支援してくれる“道具”であるなら使っていこうという姿勢であたらないと設計のできるプログラマの数は急速に増えません。

ワークショップに期待すること

以上のように，私は「プログラムを設計するための道具」「テストするための道具」を欲しいと思ってします。しかも，いままでの自分のやり方を大きく変えないものがあればベストです。使い慣れた方法や使い慣れた道具は，なかなか変革できないものです。だってそれでメシが食えるのですから，別な道具に変えようとするとそこには大きなギャップが必ず発生します。しかし，自分のやり方もしっかりしていないのにいきなり「しっかりした設計方法」に準じた「CASE」を会社なりがその生産性に合わせて導入しかねません。さらに，それについていけない人は「おはらい箱よ」と言われて，最近の不況でリストラに脅えるサラリーマンとしては死活問題です。ここは我に立ち返って考えたいと思います。

今回は，「エレベータの問題」のテーマに対してプログラマがどんな“道具”で立ち向かって行かなければならないかを再認識することで，いま“道具”として形づけられていないところについて，議論できればよいと思っています。

[ポジション発表に対する質問/コメント]

- ハードに近い部分をどう支援するかについてもう少し聞きたい。
  ハードウェアの性能向上，価格低下により，ソフトウェア側での効率向上は余り重要ではないのではないか。(効率向上によりプログラムの可読性が下がるなら)。
- 設計法によらない設計ツールは有り得るか?
- メモを越えるツールはどんなツールか?
- 処理速度に共感レスポンスで脳をストップさせる。断続的にさせてしまうようなのは，私も思考の妨げになると感じている。(自動化でなくて) 支援する方の良い CASE に求められる条件ですよね。CASE には，成果物の実稼働中に起きるアクシデントに対しては対応できないが，過去の過ちを蓄積しておき学習に使うのが賢い CASE の使い方。私のいいたいことをかなり代弁してくださいました。

[共通問題] エレベータの制御

SIGENV 有志*

1. 共通問題について

岡崎での若手の会の時には「図書館問題」を取り上げたが, 今回も同一の問題を皆で解くことにした。もちろん前回のそれでは, 設計そのものに主眼があったわけだが 今回はあくまでそれをネタに 環境(特にその中でもツールについて) を全員で考えようという趣向である。

今回の問題は, 伊藤さんの薦めで JSD の教科書 [JSD] の中の『ハイライド・エレベータ』を取り上げることにした。事前に SIGENV の定例会で, この本から抜き出した ハイライド・エレベータ の記述[1]と, 伊藤さんから補足していただいた説明[2]を有志で検討し, その疑問点を整理した。

2. 『ハイライド・エレベータ』のあらまし

ハイライド・エレベータ社は, ある階から別の階へと人間を移動するためのエレベータを設置する会社で, 同社は手動制御式のエレベータに関しては豊富な経験がある。同社のエレベータの特徴/構成は

- 階への到着検出機
- モータ
- 巻上げ機
- 切換え装置, etc.

にある。

コンピュータ制御は, 同社にとっては 新しい問題なので, 顧客の個別の要望に応じて設計/開発すべきと考えている。

今, 設置しようとしているエレベータは以下のようなものである。

- 6 階建/エレベータ 2 基
- すべての階で利用可能
- 乗車ボタン (フロアボタン) :
  1 階は昇りのみ
  2〜5 階は昇りと降り両方
  6 階は降りのみ
- 行き先ボタン (エレベータボタン) : 行き先階を示す 6 つのボタン
- ドア : 各階 と エレベータ
- 昇降 : 屋上のモータでロープを巻き上げ/巻き戻し
- 到着検出器
  各階の昇降路の内側にある
  エレベータの車輪で検出
  定位置から 15 cm 以内に入ると, 車輪が検出器を押し, 電器スイッチを閉じる.
- ドアの開閉検出器
  完全に開いたか, 完全に閉じたかを検出する

同社は, コンピュータ制御には懐疑的であり, 「利用者からの呼び出しに応じて運行計画をたてること」と「モータとランプへの指令」に範囲を限定すべきだと考えている.

- モータへの指令
  始動 : モータを始動させる
  停止 : モータを停止させる
  上向き : モータの極性を上昇方向にする
  下向き : モータの極性を下降方向にする
- ランプへの指令
  点灯 : ランプを点灯する
  消灯 : ランプを消灯する

ドアの制御と定位置への停止の保証は, 従来の技術を用いる.

- 但し, 定位置から 15 cm 以内に達したら停止指令が出される
- ドアの制御の詳細は省く

[問題]

- エレベータがより良く振舞うようシステムを仕様化したい.
- 呼び出し分布などの問題は無視し, 数学的な直観に基づく可能な解決策を考えるだけとする.

* 伊藤 昌夫, 田中 慎一郎, 武田 泰英, 塩谷 和範, 桜井 修, 福良 博史, 後藤 浩, 渡邉 雄一

[1] 「2.『ハイライド・エレベータ』のあらまし」がそれに相当する

[2] 「3. 非形式的なハイライド・エレベータの定義」がそれに相当する. 但し, これはその整理した結果である

[着想]

- ある階の乗車ボタンから呼び出されたら, 1 つのエレベータの制御プロセスが運行を約束する. 例えば, 1 階に止まっている時に, 4 階行きの行き先ボタンを押されたら, 途中階の「昇り乗車ボタンの呼びだし」に運行を約束できる.
- すでに他のエレベータが運行を約束している呼び出しに関しては, 制御プロセスは無視する.
- 約束は, それを約束したエレベータか, もしくはたまたま着いた別のエレベータが履行する.

3. 非形式的なハイライド・エレベータの定義

前章で概略を見た ハイライド・エレベータ社のシステムから, 今回の共通問題として, 以下のように定義をする。

エレベータ

- ドアが閉じている場合のみ, 移動できる。
- エレベータ制御からの方向指示がある場合, 方向指示に従って, 移動する。
- エレベータ制御からの指定階と到着検出器からの階情報が一致したら, 停止する。
- 最上階から上方には, 移動できない (安全のための冗長な条件)。
- 最下階から下方には, 移動できない (安全のための冗長な条件)。

ドア

- エレベータが停止するとドアが開く。
- 一定時間ドアが開いていると, ドアは閉まる。
- 停止中に, 停止階のフロアボタンないしはエレベータボタンが押されるとドアが開く。

フロアランプ

フロアにありフロアボタンが押された時に点灯するランプ

- 上下を示す 2 つのランプからなる (但し, 最下階に於ては上向きのみ, 最上階に於ては下向きのみである)。
- フロアボタンが押された時に [同一フロアの全ての][3] 該当ランプが点灯する。
- エレベータが到着すると [同一フロアの全ての] 該当ランプが消灯する。

フロアボタン

フロアにあり移動したい方向を指示するボタン

- 上下を示す 2 つのボタンからなる。(但し, 最下階に於ては上向きのみ, 最上階に於ては下向きのみである)
- 「押されたボタンの示す方向への移動」と云うサービスを要求する

(フロア) 方向ランプ

エレベータの進行方向を示すランプ

- 上下を示す 2 つのランプからなる。(但し, 最下階に於ては上向きのみ, 最上階に於ては下向きのみである)
- フロアに設置されている
- フロアボタンからのサービス要求があり, かつエレベータがその方向へ近づいている時, 方向ランプは点灯する。
- エレベータが到着すると, (フロア) 方向ランプは消灯する。

エレベータランプ

エレベータ内の階表示ランプ

- 階の数だけある。
- エレベータボタンが押された時に, 点灯する。
- エレベータがエレベータランプの示す階に到着した時, 消灯する。

エレベータボタン

エレベータ内の行きたい階を指示するボタン

- 階の数だけある。
- 「押されたボタンの示す階への移動」と云うサービスを要求する。

(エレベータ) 方向ランプ

エレベータの進行方向を示すランプ

- 上下を示す 2 つのランプからなる。
- エレベータ内に設置されている
- エレベータが移動している場合, 移動方向のランプが点灯している。
- エレベータが停止中で, かつ方向指示がない場合, 方向ランプは消灯している。
- エレベータが停止中で, かつ方向指示がある場合, 方向指示と同じランプが点灯する。

[3] '[ ]' で囲まれた部分は, 複数エレベータの場合を示す.

エレベータ制御

サービス要求があると, エレベータに対して, エレベータの停止時ないしはフロア通過時 (即ち, 到着検出器が到着情報を発行した場合) に, 以下の規則に従って, (停止時は) 一定時間ごとに, 方向指示及び指定階指示を出す。

3.1 エレベータ制御規則

条件にヒットするまで, (1) から順に規則を調べていく。

(1) [各] エレベータが停止している場合 (即ち前回方向指示が発行されていない), サービス要求を満足するように, [任意の一台の] エレベータに方向指示が発行される。

(2) [各] エレベータの移動方向に, フロアボタンからサービス要求がある場合, [各] エレベータへの方向指示は変わらない。

(3) [各] エレベータの移動方向に, [当該] エレベータボタンからサービス要求がある場合, [当該] エレベータへの方向指示は変わらない。

(4) [各] エレベータの移動方向と反対に, [当該] エレベータボタンからサービス要求がある場合, [当該] エレベータへの方向指示は逆になる。

(5) [各] エレベータの移動方向と反対に, フロアボタンからサービス要求がある場合, [任意の一台の] エレベータへの方向指示は逆になる。

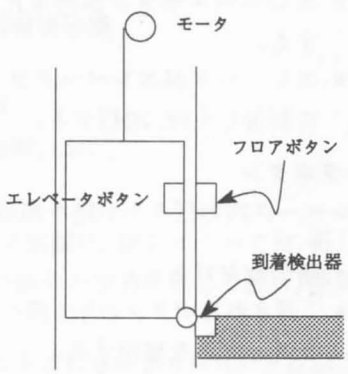
(6) [各] エレベータに対する方向指示はない。

3.2 その他

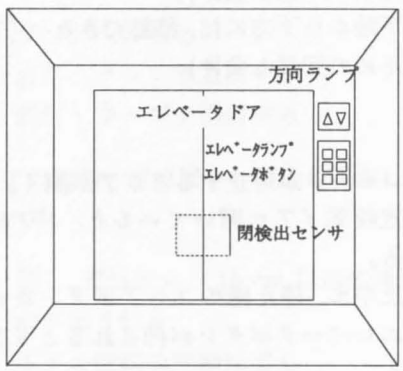
ドアの開閉に関して, 障害物発見のためのセンサは考慮しない。
また [JSD] では, ドアの開閉に関しては, 考慮していないが (「技術陣と検討しなければならないが, それは後で行う事とする」と云う記述参照), 今回はモデルとして完結させるために, 考慮する。
但し, フロアドアとエレベータドアを一体とし, 閉検出センサもドアの一部とし, 別扱いとはしない。

参考文献

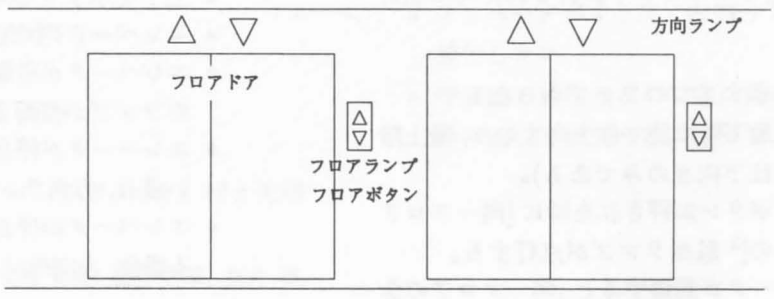
[JSD] M. Jackson (大野, 山崎 [監訳]) : システム開発 JSD 法, 共立出版,1989



断面



エレベータ内部



正面

SA/SD グループの記録

大塚 理恵 *

1. SA/SD グループの方針

SA/SD グループは最初からマシンを囲み, ツールの操作性を交互に確かめながら分析および設計を行い, 最終的には 1 つのプロセスをコード化し, コンパイルまで進めた。仕様について議論しながら DFD を作成するのが作業の中心となった。

作業を始めるとき, 手順や方針, そして演習の最終目標を決めていたわけではない。

メンバーは必ずしも実務で SA/SD を使っているわけではなく, 確認したわけではないが, かつて参加した若手の会でこの手法で CASE STUDY した経験があったり, 一方の CRC をあまり知らなかったりのどちらかだと思う。演習後, 問題提供者からエレベータ問題の実装例を見せていただき, 実際にエレベータの動き, 仕様を確認することができた。

演習中は積極的にツールによる設計支援や操作性を試みようとした。しかし設計の厳密さにあまり注意を払っていなかったり, ほとんど記録を残していなかったため, 後日この報告用の DFD を見て, メンバー間でどのような合意のもとにこうしたシステム分割がなされたのか記憶に自信がない。

2. 分析および設計

次のような流れで作業を進めた。

(1) DFD を作成

(2) DFD から構造図を自動生成

(3) データ辞書にデータ定義

(4) 一つの処理を中間言語で記述

(5) 中間言語から PAD 図を自動生成

(6) 中間言語から C のコードを自動生成

(7) Softbench にリンクしてコンパイル

必ずしも SA/SD では上記のようなステップで行なわないのかもしれないが, SA/SD 用にツールから提供される機能を一つ一つ試していった。

(1) ではコンテキストダイアグラムとレベル 0, 1 の DFD を作成したが, 全ての処理について分析したのではなく, 図中 (DFD0) の「1 特定エレベータを制御する」だけについてレベル 1 まで検討した。仕様について議論しながら進めたので, 時間的に演習の大半をここに費やした。最初はコンテキストダイアグラムをかき, レベル 0, 1 と進んだが, 途中仕様の見直しで前のフェーズに戻ることも何回かあった。後日 DFD を見て「エレベータ状態データ」の位置付けがどうもおかしいことに気付いたが, 図は作業途中のものということでお許しください。

当グループでは複数のエレベータを 1 つの処理「1.4 指示を振り分ける」に管理させようと考えた。あるエレベータの現在位置, ドアの開閉状態, 等を「エレベータ状態データ」に持たせる。そして

1) 1.4 が一定間隔で個々のエレベータの状態を見に行く。
   あるいは
2) 個々のエレベータからの状態変化情報をトリガーとする。

という, 1.4 が 1.1, 1.2, 1.3, 1.5 の各処理に指示する 2 つの方法を検討したが結局結論はでなかった。

(2) の構造図は (1) から自動生成できた。

(3) のデータ辞書には, 「エレベータ状態データ」について定義し, 図中 (DFD1) の停止予定階, 現在位置等についてデータのタイプ, 制約条件等[1]を入力した。このデータ辞書は他のツールとリンクしていたかもしれないが, とりあえず上記の入力だけで終わってしまった。

(4) では DFD レベル 1 の処理のうち, 「1.2 ドアを開閉する」をツール固有の中間言語で記述した。

この中間言語からは (6) のとおり, 数種類の言語でソースコードを生成できる。データ辞書にデータが定義されていれば, コードのデータ定義に関する記述も簡略化できたと思う。この中間言語からは数種類の図式, ドキュメントを生成できたが, 今回は PAD 図を生成した。ここで中間言語の記述画面と PAD の画面を両方出しておけば, 中間言語を記述/修正するとリアルタイムで PAD 図が更新されるという機能があり, PAD 図を見て確認しながらコーディングを進

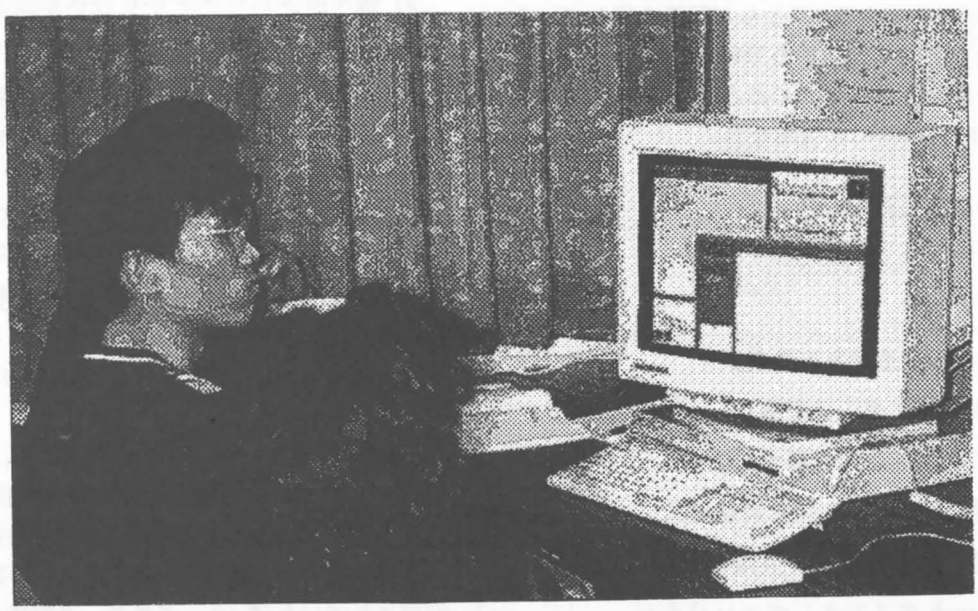
* RSK

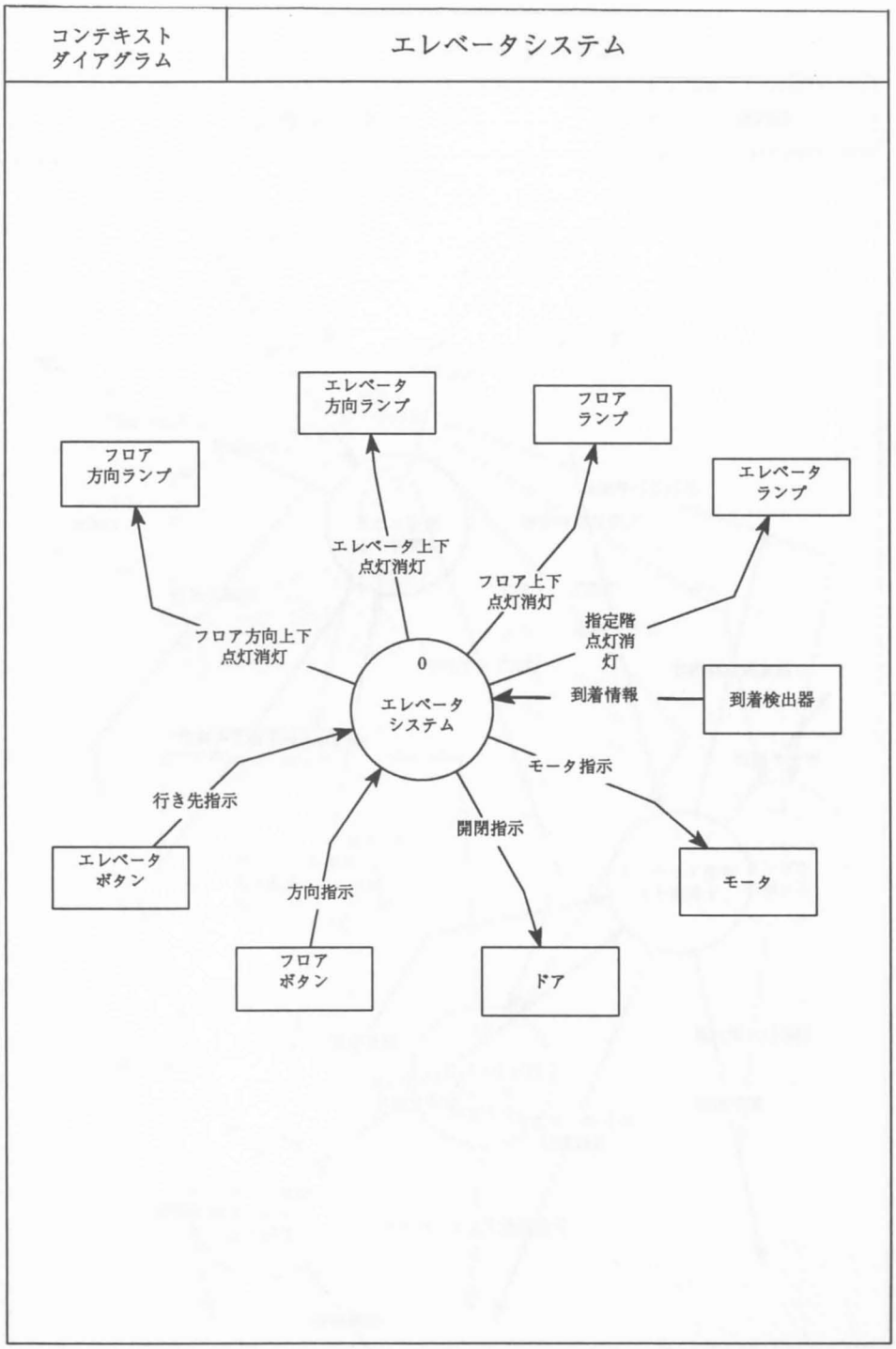
[1] 大塚が入力していたのですが詳細な項目は忘れました。

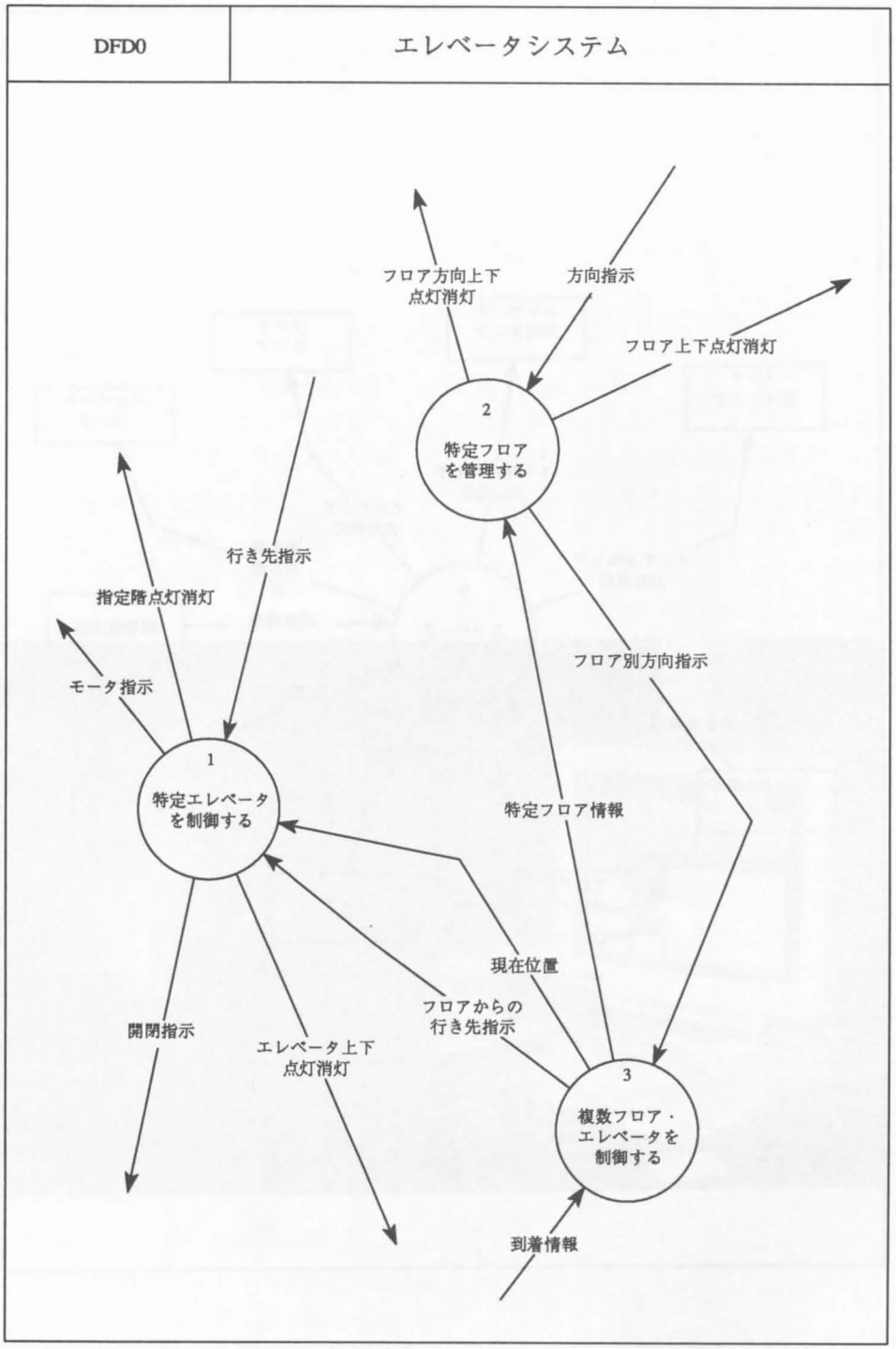
めることができた。

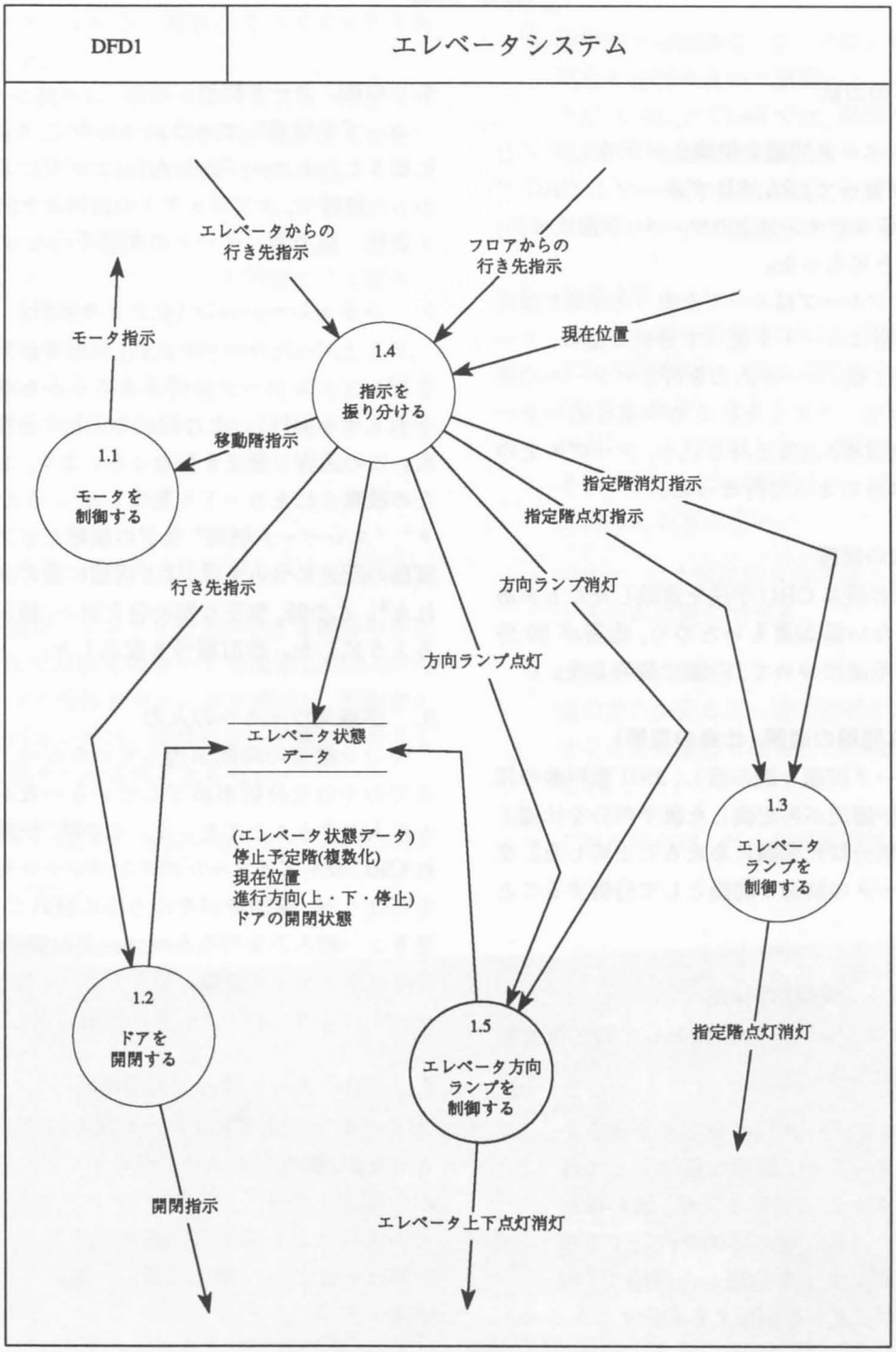
3. おわりに

演習ではツール開発者をインストラクターに，SA/SD 用の様々なコンポーネントを使用させていただきました。メンバーからの我が侭なリクエストが機能改善につながれば幸いです。




コンテキスト
ダイアグラム
エレベータシステム
エレベータ
方向ランプ
フロア
ランプ
フロア
方向ランプ
エレベータ
ランプ
エレベータ上下
点灯消灯
フロア上下
点灯消灯
フロア方向上下
点灯消灯
指定階
点灯消
灯
0
エレベータ
システム
到着情報
到着検出器
モータ指示
行き先指示
開閉指示
エレベータ
ボタン
モータ
方向指示
フロア
ボタン
ドア


DFD0
エレベータシステム
フロア方向上下
点灯消灯
方向指示
フロア上下点灯消灯
2
特定フロア
を管理する
行き先指示
指定階点灯消灯
モータ指示
フロア別方向指示
1
特定エレベータ
を制御する
特定フロア情報
現在位置
フロアからの
行き先指示
開閉指示
エレベータ上下
点灯消灯
3
複数フロア・
エレベータを
制御する
到着情報


DFD1
エレベータシステム
エレベータからの
行き先指示
フロアからの
行き先指示
モータ指示
現在位置
1.4
指示を
振り分ける
移動階指示
1.1
モータを
制御する
指定階消灯指示
指定階点灯指示
行き先指示
方向ランプ消灯
方向ランプ点灯
エレベータ状態
データ
(エレベータ状態データ)
停止予定階(複数化)
現在位置
進行方向(上・下・停止)
ドアの開閉状態
1.3
エレベータ
ランプを
制御する
1.2
ドアを
開閉する
1.5
エレベータ方向
ランプを
制御する
指定階点灯消灯
開閉指示
エレベータ上下点灯消灯

CRC グループの記録

('94 年 1 月坂北荘にて)

塩谷 和範 *

CRC グループの方針

共通問題のエレベータ問題を伊東氏が用意した 2 台の HP マシンを使って, SA/SD グループと CRC グループに分け, 各々のマシン上のツール (伊藤氏提供) により, 解くことになった。

しかし, CRC グループはカードを使った作業が重要であるので, 最初はカードを使って分析を進め, カードによる分析終了後, ツール入力を行ないツールの使い心地を確かめることにした。この作業は広いテーブルが使える別室に, 白版を持ち込み, テーブルを中心に畳の上に車座になって行なった。

1. CRC 手法の理解

一部の参加者は既に CRC 手法を実践したことがあるが, 全く知らない参加者もいたので, 塩谷が 20 分程簡単に CRC 手法について, 白版で説明した。

2. エレベータ問題の理解 (仕様の理解)

次に, エレベータ問題を読み直し, JSD 教科書の元問題ではなく, 伊藤氏が再定義した後半部分を仕様として考え, 不足部分は元問題にあたることにした。また, 複数エレベータの制御を前提として分析することにした。

3. オブジェクト (候補) の抽出

問題中からオブジェクトの候補としての名詞を抽出し, 白版に記述した。

> ここで, エレベータ利用者をどう扱うかの議論があったが, 最初の段階としてはオブジェクトとして取り上げ, 後の検討分析段階で, (利用者の挙動を代行する複数のオブジェクトを認識した段階で) 利用者オブジェクトを削除する方針とした。

同様に, 明示されているデータと状態についても抜き出した。

* (株)SRA PSEL 部長

4. CRC カード記述

カードを用意していなかったので, A4 白紙を半分に切りこれをカードとした[1]。オブジェクト抽出が終わった段階で, オブジェクトの担当者を決め手分けして責務・協力者・データの記述を行なった[2]。

5. シミュレーション (シナリオ実行)

でき上がったカードを元に, 利用者がフロアボタンを押してエレベータを呼ぶところから始め, 一連のシナリオを実行し, その記述の正しさを検証していった。この過程で修正を行なった。また, まとめられたため破棄されたカードも発生した[3]。また, "エレベータ", "エレベータ制御" などの複雑なオブジェクトの責務の記述については, より明確に書き直しが行なわれた[4]。この時, 修正は別の色を使い, 修正部分が分かるようにした。改訂番号も記入した。

6. 伊藤氏ツールへの入力

マシン撤去の時間が迫っていたため, カードベースでの十分な分析が終了していないながらも, ツールに入力することになった。この時, 伊藤氏のツールは CRC カードイメージのテンプレートを使って入力を行なうが, 参加者がその方法に慣れていないので, 急きょ一括入力を行なうコマンドを伊藤氏が作成し, テキストエディタを使って入力データを作成し, 変換ツールにより CRC カードに変換した。

7. CRC カード間の協力関係図

ツール入力後, CRC カード間の協力関係図を表示させたが, 孤立したカードができてしまった。調べてみたら, "ドアー", "ドア", "フロア", "フロアー" などの個人による表記上の違いが原因であった。これを伸ばす音 ("ー") 無しに統一・修正して無事協力関係図が表示できた。

[1] CRC カードは, 単純な形式なので簡単に作成できる

[2] つまりこの段階では, 記入した個人の判断で記述している.

[3] フロア上昇ランプと下降ランプは, "フロアランプ" にまとめた

[4] エレベータは 2 回, エレベータ制御は 3 回

8. 他手法への変換

伊藤氏のツールは, 一定の意味上の制限の元に手法間の変換を実現している。この機能を用いて, CRC カード群から Coad/Yourdon の図に変換を行なった。これは基本的には, 協力関係図の協力関係をオブジェクト図のメッセージ結合と解釈してオブジェクトを関係付けている。
また, データは属性に, 責務はサービスに, 取りあえず置いてみることができる。時間の関係で, Coad に変換後のこれ以降の分析は行なわなかった。

9. 伊藤氏ツールの評価

本ワークショップは, エレベータ問題を共通課題に具体的にツールを使ってみることによって, ツールと環境について考えてみることが目的だった。しかし, せっかく伊藤氏がツールを用意してくれたので, このツールの CRC 部分について, 使い心地をまとめてみる。

9.1 難点

- カードイメージの入力作業がしにくい[5]。
- カード溜がウィンドウ右上の小さな部分のみだが, これでは多くのカードを表示し切れない。ウインドウ全体にカード群を表示し, 配置変えなどを行ないたい。現状のカード溜でもできるが小さ過ぎてやる気にならない。

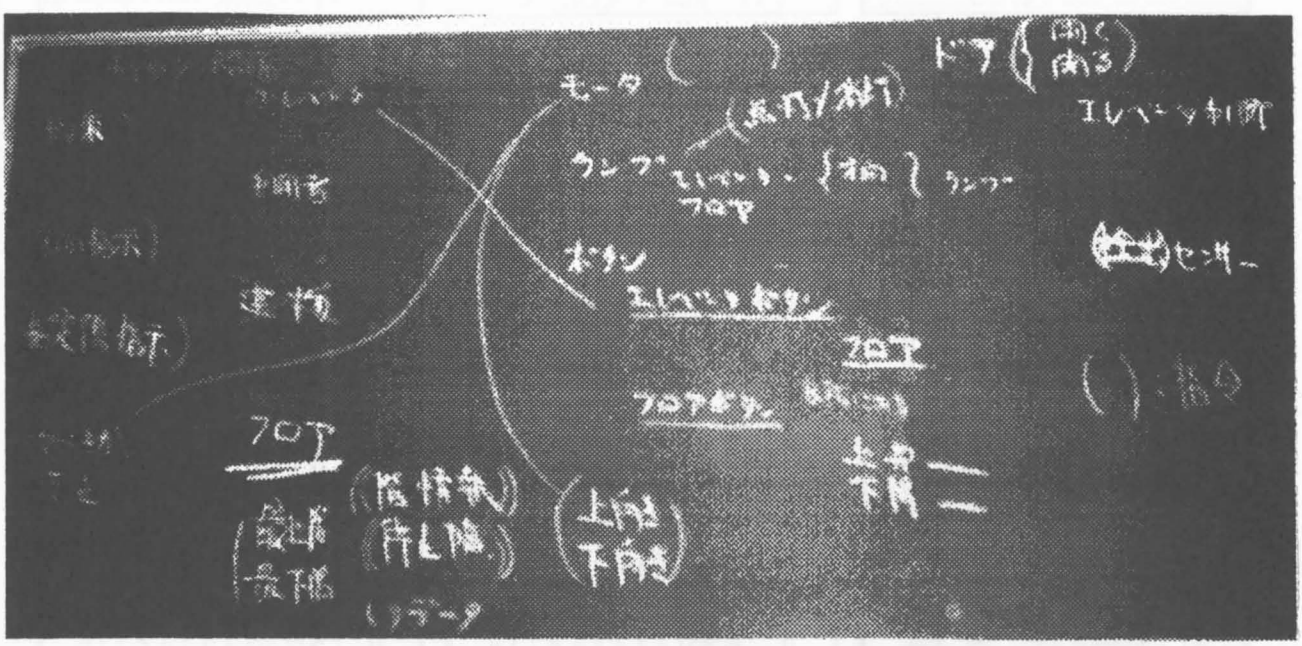
[5] 最初からツールに入力しながら分析していくのであればさほど難点とはならないだろう。それでも, 入力し易くするに越したことはない

9.2 利点

- 紙のカードによる分析終了後, ツール入力を行なう時には, 伊藤氏が用意したような変換ツールが役立った。カード数が多くなればなる程有用。
- 他手法への変換は, カードによる分析終了後の更なる分析のために重要。
  当然, CRC と Coad では, 発想も分析の仕方も違うがこの様な近似を, 取りあえず行なうことによって, 他手法に変換し分析を続行することができることは重要だと考える。

9.3 改善提案

- カード内容の表示は, カード溜から指定のカードを作業域に引っ張って来ることにより行なうが, 先に指摘したように, 作業域全域にカードが表示されているなら, 指定カードをダブルクリックするなどの操作でカードイメージの表示が行なえる方が良い。
- 同様に, 協力関係図も作業域上に関係線を付加するのみで表示できる。
- 仕様から, オブジェクトの抽出・責務の決定・協力者の決定など一連の過程をたどれるようにすることは重要である。この機能を実現して欲しい。
- このためには, データ辞書機能が欲しい。



エレベータ

Class	Responsibilities	Collaboration	Data
	・ドアの開閉状態を知る ・方向にしたがってモータに巻き上げ/巻き戻しの指示をする ・方向にしたがってエレベータ方向ランプを点灯する ・階とセンサからの階が一致したらモータを止める ・ドアに開けるを指示する ・フロアランプと停止階のエレベータランプを消灯する ・最上階、あるいは最下階の場合には、方向を逆転する ・次の移動方向の指示の際は、エレベータ方向ランプを消灯する	ドア エレベータ方向ランプ センサ フロアランプ エレベータランプ モータ	最上階 最下階 方向 階情報

フロアボタン

Class	Responsibilities	Collaboration	Data
	・フロアに階を聞く ・同一フロアのすべてのフロアランプに点灯を指示する ・方向と階をエレベータ制御に指示する	フロアランプ フロア エレベータ制御	

エレベータボタン

Class	Responsibilities	Collaboration	Data
	・指定階への移動サービスを要求する ・押されたことを検出する ・エレベータランプに点灯を要求する	エレベータ制御 エレベータランプ	

センサ

Class	Responsibilities	Collaboration	Data
	・エレベータが到着したことを知らせる	エレベータ制御 エレベータ	

エレベータ制御

Class	Responsibilities	Collaboration	Data
	・ルールによってエレベータに方向と階を指示する ・階のフロアのフロア方向ランプに点灯を指示する	エレベータ フロア方向ランプ フロア	

利用者

Class	Responsibilities	Collaboration	Data
	・フロアボタンを押す ・ドアが開いたらエレベータに乗る ・エレベータボタンで階を指示する	フロアボタン エレベータボタン	

ドア

Class

・開くの指示を受けドアを開き、一定時間後閉じる

Responsibilities

Collaboration

ドアの状態

Data

フロア

Class

・自分のフロアを答える

Responsibilities

Collaboration

Data

モータ

Class

・ロープを巻き上げる
・ロープを巻き戻す

Responsibilities

Collaboration

Data

エレベータ方向ランプ

Class

・点灯する
・消灯する

Responsibilities

Collaboration

Data

フロアランプ

Class

・点灯する
・消灯する

Responsibilities

Collaboration

Data

エレベータランプ

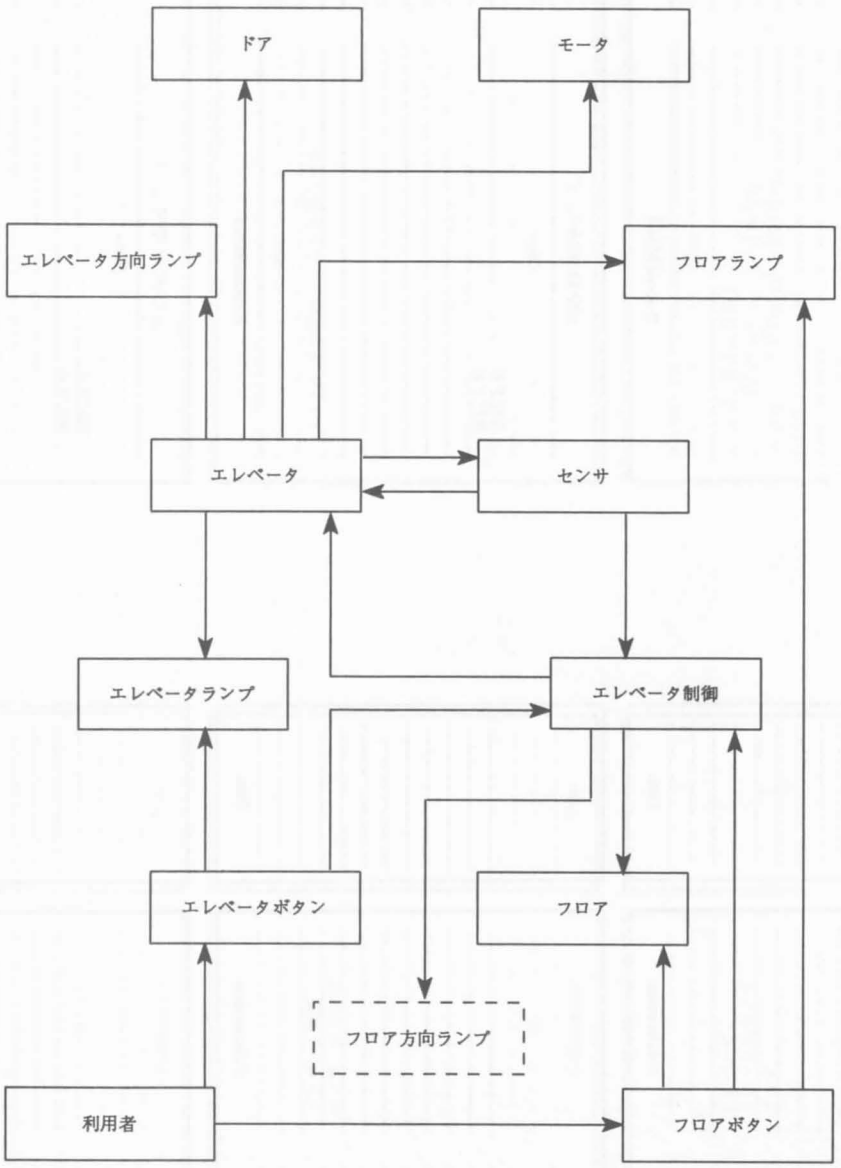
Class

・点灯する
・消灯する

Responsibilities

Collaboration

Data


ドア
モータ
エレベータ方向ランプ
フロアランプ
エレベータ
センサ
エレベータランプ
エレベータ制御
エレベータボタン
フロア
フロア方向ランプ
利用者
フロアボタン

協力関係図 (Collaboration Graphs) による考察

塩谷 和範 *

ワークショップの場では, 最初のカードを作成する時にどれをカードオブジェクトにし, どのような責務を持たせ, どのカードを協力者にするかの議論で時間が取られ, 一旦作成したカードについての吟味をする時間が取れなかった。そのため, 本報告書を作成するにあたって塩谷が個人的に検討を進めてみた。その概要を以下に示す。

1. 協力関係図の作成

まずはでき上がったカードを広い場所に広げて, 関係があるもの同士を近くに配置し, 全体の関係が分かり易くなるよう並べ換えてみる。これを元に, カード間の協力関係矢線を入れた協力関係図を作成する[1]。

良く整理・配置された協力関係図からは, 様々な事柄が読み取れる。

2. 利用者の扱い

通常, 利用者はシステムが制御できない外部要件として定義するため, システムの定義中には現れない。ここでは, 最初の段階は, 利用者オブジェクトも他のオブジェクトと同様に定義し, その関わりが明確になった時点で削除する方針とした。

CRC カード作成, シミュレーションを行なった後, 協力関係図を書いてみることにより, 以下の考察を行ない利用者オブジェクトを削除することにした。

仕様書中に明示的には現れないが, フロアランプ, フロア方向ランプ, エレベータランプ, エレベータ方向ランプなどの表示器は, 利用者が認識し, 情報を得, 確認するための器具である。また, フロア, ドア, エレベータは, 利用者がエレベータを利用する時に乗っている・通り抜ける場と見なすことができ, これら全てがエレベータシステムと利用者との境界面を構成する。(協力関係図参照)

この認識ができた時点で, 利用者オブジェクトを削除することができる。

3. オブジェクトは, 自律したものとして考える

エレベータ制御に多くの機能を取り込み, 全てを行なわせることは可能である。しかし, この様な巨大オブジェクトは好ましくない。この様に一つのオブジェクトに役割を集中した設計を行なうと, エレベータ制御オブジェクトの些細な変更が, 多くの従属的なオブジェクトに影響する。また, エレベータ制御オブジェクト自体も巨大化し, 保守性を低下させる。

したがって, 全てのオブジェクトはなるべく均等に役割を分担し, 特定のオブジェクトに過度に依存することは避ける。これによって, 変更時の影響範囲を局所化できる。

3.1 エレベータをどこまで自律させるか?

- エレベータ方向ランプを点灯するのは誰?
  フロア方向ランプを点灯するのがエレベータ制御であるように, エレベータ方向ランプもエレベータ制御が行なうことが考えられる。しかし, 先に述べた原則によりここではエレベータが点滅制御を行なうことにした。
- 最上階 / 最下階 を知るのは誰か?
  エレベータ方向ランプと同様に, ここではエレベータ自身が知っていることにした。最大の理由は, その方が安全だからであった。しかし, センサ・モータ・フロアなども最上階・最下階を知りうる。例えば, モータは物理的に反転せざるを得ないし, フロア (フロアボタン) も方向ボタンが一方向しかないことから, これを知っているはずである。したがって, 協力関係図の検討時点で, モータあるいはフロアにこの役割分担を変更することができる。

  もしモータにこの役割を負わせることにすれば (可能ならば), エレベータの責務 9 番は, エレベータから削除し, モータの責務に「最上階・最下階に到達したらモータの回転方向 (エレベータの移動方向を) を反転する」を追加できる。モータはこのように設計されているはずなので, この方が現実的かも知れない。

4. グループ化・サブシステム化

協力関係図上で点線で囲んだカード群は, 関連の深いグループと考えられる。グループ内の協力関係は, グループ外には見せる必要はないし, 実際見せない方

* (株)SRA ソフトウェア工学研究所

[1] 白版にカードを貼り, 協力関係矢線を記入しても良い。

が良い。例えば，フロア関係のグループについて検討すると，

- フロアの役割は?
  カード作成時点では，「階を知っていて問い合わせに答える」ことのみが責務であるが，先の最上階・最下階の知識を持たせることも可能である。また，協力関係図に点線でフロアに関係するカード群を囲んだが，このグループをまとめて，フロアで代表させることもできる。この場合は，フロアカード群とのやり取りは，フロアカードが代表して行なうことになる。

このように考えると，フロア関係カードをまとめてフロアサブシステムを構成することができる。これにセンサが直接ドアに「ドアを開く」ことを依頼できるとすれば，ドアとセンサも含めることができる。同様に，エレベータランプサブシステム，エレベータサブシステムを定義することができる。

こうすると，フロア，エレベータ，エレベータ制御，エレベータランプの4グループ(サブシステム)に大きく集約できる。

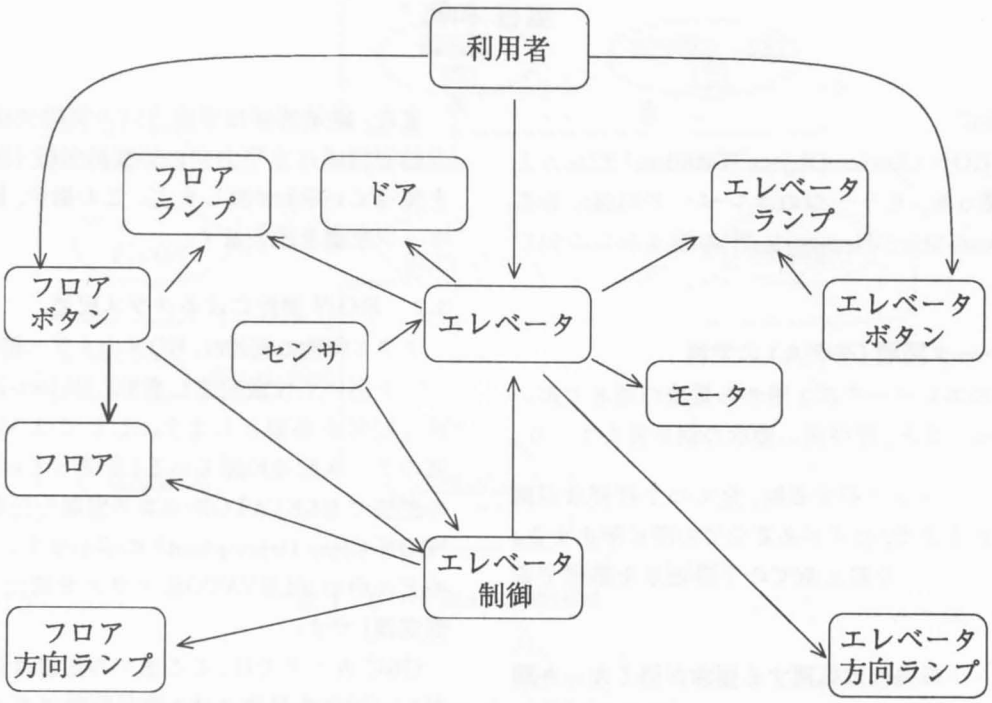
ドアに関する前提条件：問題を単純化させるために，ドアは1つとしたが，安全のために，フロアドアで代表させることにした。
(フロアにドアがないととても危険)

名詞の抽出によって得られたオブジェクトの候補と，その説明(代表的責務あるいは目的)です。

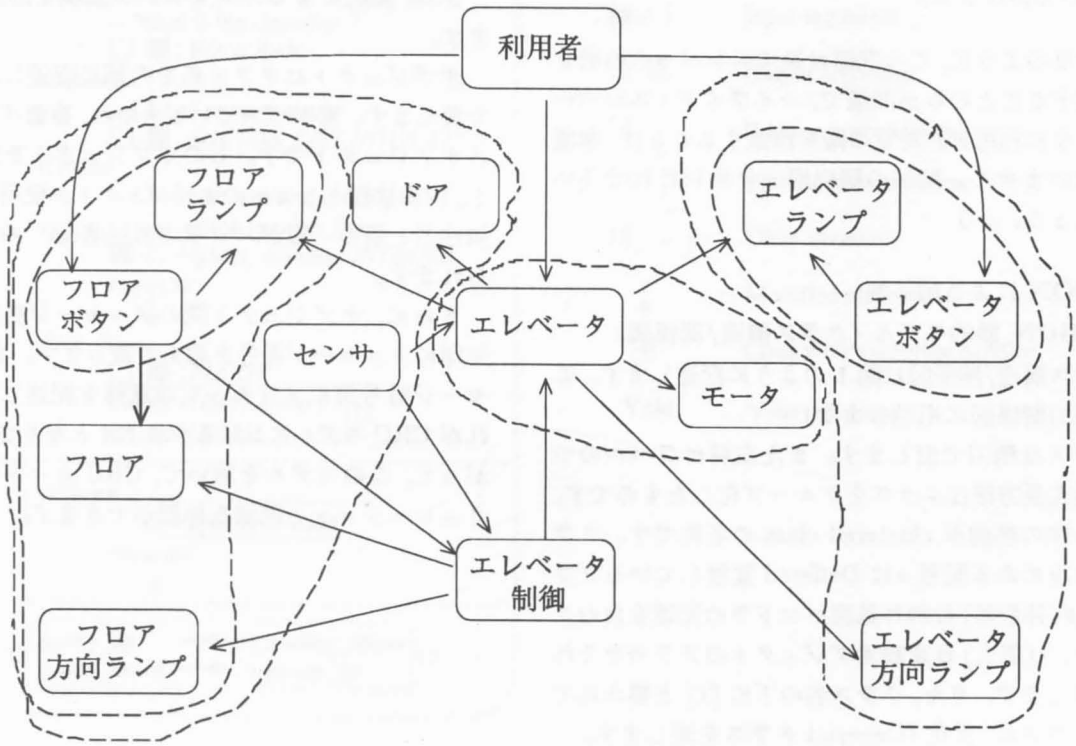
オブジェクトの候補	代表的責務あるいは目的
エレベータ	ドアが閉じている時のみ移動できる。 エレベータ制御からの方向指示にしたがって移動する。 指定階に到着したら停止する。
エレベータボタン	階の数ある。「ボタンの示す階への移動を要求する」
エレベータランプ	階の数ある。対応するボタンが押された時に点灯する。 指定階に到着した時に消灯する。
エレベータ方向ランプ	エレベータ内の移動方向を示すランプ
エレベータ制御	規則に従ってエレベータを制御する。
センサ(到着検出器)	エレベータの到着を検出し、階情報を与える。
ドア	フロアドアとエレベータドアを一体とし，閉検出センサもドアの一部とし、別扱いとはしない。 エレベータが停止するとドアが開く。
フロア	建物の階そのもの
フロアボタン	(行き先方向を指示する)上下を示す2つのボタンからなる
フロアランプ	(行き先方向を表示する)上下を示す2つのランプからなる
フロア方向ランプ	(行き先方向を表示する)上下を示す2つのランプからなる
モータ	モータ・巻き上げ装置・切り替え装置はモータで代表させる。
利用者	エレベータの利用者

表 1．名詞の抽出によって得られたオブジェクトの候補とその説明

Collaboration Graphs
(Object間の協力依存関係)



Collaboration Graphs
Objectのグループ化



BON 手法によるもう一つのエレベータ問題とその解法例

塩谷 和範 *

1. はじめに

これは, BON (Better Object Notation) 記法および手法を使った, もう一つのエレベータ問題の解答例です。Jean-Marc Nerson[1, 2] の論文からの引用です。

2. エレベータ問題 (モデル) の定義

- 一台のエレベータが 1 階から最後の階まで動いている。また, 呼び出し要求の順を覚えている。
- エレベータが上昇する時, 全ての上昇要求が満たされるまで, 要求がある全ての階に停止する。そして, 方向を変え全ての下降要求を処理するため下降する。
- 上昇・下降過程は処理する要求が無くなった階で停止する。
- エレベータがある階に到着した時, 利用者が乗り込み上昇・下降のいずれかのボタンを押してよい。これは, どの方向にエレベータが動いているかによる。

お分かりのように, この問題は単にエレベータの動きを記述することのみが問題で, ハイライド・エレベータのように利用者が特定の階を指定することは, 考慮されていません。記法の説明用ですからこれで良いのでしょう。;-)

3. BON による解答例

3.1 BON 静的モデル (クラス構造/関係図)

クラス構造/関係図は図.1 のように記述します。これは協力関係図に相当します。

クラスは楕円で表します。また左肩にラベルのついた角丸長方形はクラスをグループ化したものです。ラベル中の名前が clustered class の名称です。クラス名の上にある記号 *は Deffered 宣言しているクラスを, #(井桁印) は例外処理ハンドラの記述を含むクラスを, •(黒丸) は永続オブジェクトのクラスをそれぞれ表します。また, クラス名の下に [T] と書かれているクラスは, 汎化 (Generic) クラスを表します。

また, 継承関係は垂直方向の実線矢線で, 利用者-供給者関係は水平方向の二重線矢線 (図中では太線を使っている) で表します。この場合, 線の傍にメッセージを書き添えます。

3.2 BON 表記によるクラス記述

クラス特性の記述は, BON のクラス箱中に Object-Z[3] を用いて仕様記述します。Object-Z の記法は特殊な記号を必要とします。ここでは Void 記号 (電気のアース記号に似ている) を使っていませんが, この記法で ELEVATOR クラス定義を行なったものを "BON Class Description" に示します。これは, 静的モデル中の ELEVATOR クラスを拡大したもの (内部定義) です。

CRC カードでは, ここまでの定義はしません。ただし, CRC を発展させた責任駆動デザイン [4](RDD : Responsibility Driven Design) では, 拡大カード形式を用いて同様な詳細仕様記述を行ないます。

3.3 BON 動的モデル (オブジェクト関係図)

BON 表記によるオブジェクト表現を図.3 に示します。

オブジェクトはクラス名を内部に記述した長方形で表します。影がついているものは, 複数インスタンスオブジェクトです。このオブジェクトを拡大すると, 内部状態としてそのオブジェクトが使用している属性名と値の一覧が, "クラス属性名:値" の形式で得られます。

さらに, オブジェクト間のメッセージ交換は点線矢印にメッセージ番号を添えて表します。また, メッセージ番号毎にメッセージの意味を記述します。これが CRC モデルにおけるシナリオとなります。したがって, このモデルを用いて, CRC カードによるシミュレーションと同様な検証ができます。

* (株)SRA ソフトウェア工学研究所

BON Static Model

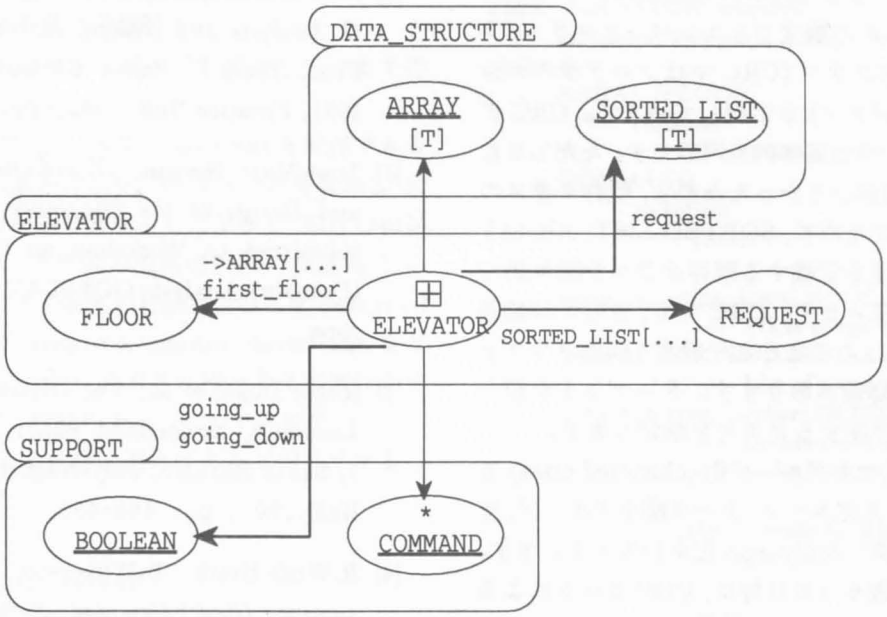


図 1 . BON Static Model

BON Class Description (w/ Object-Z)

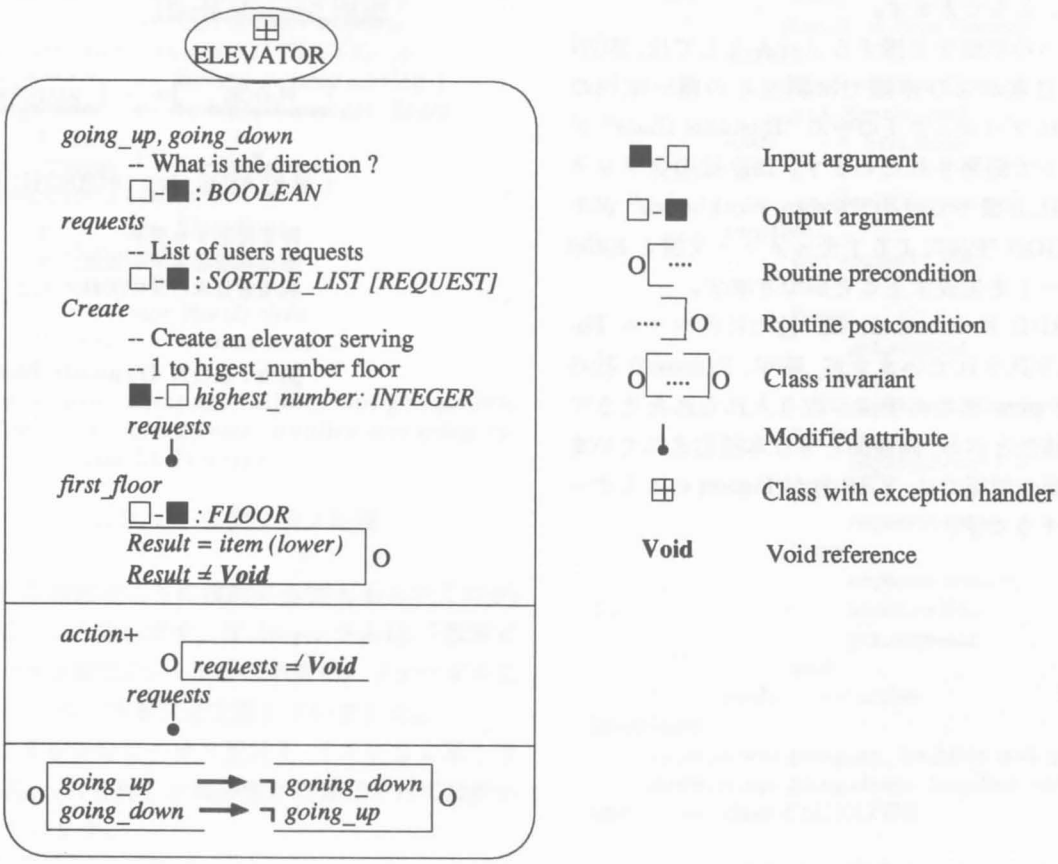


図 2 . BON Class Description

4. まとめ

長々と説明してきましたが, CRC グループでのモデル化のように, この論文でもエレベータクラス, フロアクラス, 要求クラス (CRC ではフロアボタンおよびエレベータボタン) を定義しています。CRC グループのアプローチと基本的に同じです。ただしここでは, さらに問題が小さかったためと, 既存クラスの再利用方法を示すために, SORTED_LIST, ARRAY というデータ構造を定義する既存クラスを取り込んでいるところが違います。CRC でも, さらに詳細設計に入った時には, このような一般的な既存クラスを取り込んで行く必要があります。ターゲットが絞られれば具体的に存在するクラスを指定します。

また, この論文でもグループ化 (clustered class) を定義し, エレベータグループ, データ構造グループ, 支援グループを定義しモジュール化を行なっています。

大きな問題を扱う時には特に, CRC カードによる分析が終了した時点で, BON のようなさらに詳細な分析と仕様記述ができる手法に移行することが現実的でしょう。もちろん CRC の発展系である責任駆動デザイン (RDD) に移行するのがもう一つの選択です。伊藤さんのツールのように, さらに別の手法に切り替えることもできます。

この二つの手法を支援するツールとしては, BON の場合は日本および米国では馴染みの薄い欧州の ESPRIT-II プロジェクトの中の "Business Class" プロジェクトで開発されています。ISE 社のフランス法人の SOL が扱っている "Design-Workbench" がそれです。BON 手法によるドキュメント支援と Eiffel テンプレートを生成することができます。

CRC/RDD については, HP 社の社内ツール Fusion に取り込まれていますが, 昨年, Protosoft 社の Paradigm plus にこの手法が取り入れられたそうです。ただ残念ながら, 両製品とも日本語化されていません。最新の情報では, YHP 社が Fusion のセミナーを始めるそうです。

参考文献

[1] Jean-Marc Nerson, *Extending Eiffel toward O-O Analysis and Design*, asscociated paper in Proc. Tools 5, Santa Barbara, July-August 1991, Prentice Hall, 1991, pp. 69-87.

[2] Jean-Marc Nerson, *Object-Oriented Analysis and Design in the "Business Class" Project*, submitted to Workshop on Object-Oriented (Domain) Analysis OOPSLA'91 - Phenix, Arizona.

[3] Roger Duke, et al., *The Object-Z Specification Language*, asscociated paper in Proc. Tools 5, Santa Barbara, July-August 1991, Prentice Hall, 1991, pp. 465-483.

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

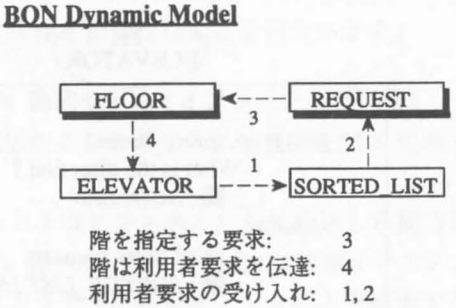


図 3 . BON Dynamic Model

付録

A Eiffel によるクラス設計

クラス特性の記述を Eiffel でも記述していますのでこれも紹介します。

Eiffel 言語の特徴は, 表明 (assertion) を記述できること, **deferred** 宣言により, コーディングを後回し[1]にできることなどがあります。したがって, 設計の初期段階, あるいは, クラスの公開特性部の記述時には, first_floor メソッドの記述の詳細部分などを **deferred** 宣言しておいて, **require**, **ensure**, **invariant** などの表明部のみを記述し, これを仕様と考えて実行させることによって, 設計のチェックができます。

Object-Z による仕様記述とほとんど同じであることが分かります。

```
class ELEVATOR export
    first_floor, going_up, going_down, requests
inherit
    ARRAY [FLOOR];
    COMMAND
feature
    going_up, going_down: BOOLEAN;
                -- What is the direction ?
    requests: SORTED_LIST [REQUEST];
                -- List of users requests
    Create (higest_number: INTEGER) is
                -- Create elevator serving 1
                -- to highest_number floors.
        do      -- ...
        end;    -- Create
    first_floor: FLOOR is
                -- First floor
        deferred
        ensure
            not Result.Void
        end;    -- first_floor
invariant
    up_or_down: going_up implies not going_down;
    down_or_up: going_down implies not going_up
end    -- class ELEVATOR
```

リスト 1：公開クラス記述

Eiffel をこの例のように設計にも使えるというのが, Eiffel の一つの強みです。昔 Meyer さんは「設計にグラフィック記法/ツールなど不要だ。フォーマルに書いていくのに限る」と主張していました。

Eiffel による完全なクラス記述を リスト 2 に示します。**deferred** 宣言した部分他が具体化されて設計が完成しています。

[1] 詳細設計確定時に記述する

```
class ELEVATOR export
    first_floor, going_up, going_down, requests
inherit
    ARRAY [FLOOR]
        rename
            Create as story_create;
    COMMAND
        redefine
            action
feature
    going_up, going_down: BOOLEAN;
                -- What is the direction ?
    requests: SORTED_LIST [REQUEST];
                -- List of users requests
    Create (higest_number: INTEGER) is
                -- Create elevator serving 1
                -- to highest_number floors.
        do
            story_create (1, highest_floor);
            requests.Create
        rescure
            retry
        end;    -- Create
    first_floor: FLOOR is
                -- First floor
        do
            check
                in_bounds.lower <= upper
            end;
            Result := item (lower)
        ensure
            Result = item (lower);
            not Result.Void
        end;    -- first_floor
    action is
                -- Serve requests.
        require
            not requests.Void
        do
            from
                get_request
            invariant
                not requests.empty
            variant
                requests.count + 1
            until
                requests.empty
            loop
                requests.remove;
                move_cabin;
                get_requests
            end
        end;    -- action
invariant
    up_or_down: going_up implies not going_down;
    down_or_up: going_down implies not going_up
end    -- class ELEVATOR
```

リスト 2：完全なクラス記述

青空の彼方から (2) *

— 酔眼に映るもの —

伊藤 昌夫 †

私は, 酔っている時は, 正直で, 理性的だと云われます。

これは, 2 日目の深夜に, 理性的な私が話したお話のまとめです。これを書いている今は, 酔ってませんので, 自分で書いていても可笑しいのですが, 仕方がないですね。

話のきっかけは, 渡邊さんから頂きました。

何故, 人間の思考は振動するのか?

この質問に答えるために延々4 時間に渡って, 話し始めたわけです[1]。

1. ものの見方

先ず, 思考の振動に関して理解するためには, 我々のものの見方から考え直す必要があります。近代の科学的なものの見方は, ヘーゲルの弁証法を例に挙げるのが一番分り易いと思います。つまり, ここにある 2 つのモノがあるとします。モノでも事象でも良いのですが, そうすると我々は「違いの分る」人間になろうと思うわけです。もちろん, そのモノに対して興味がある時だけですが。

違いが分かると云うのは, 難しいのですが, 理解の仕方としては楽だし, 客観的なものです。何故ならば, 違いが分かる時には, 共通項も同時に分っているわけですから, その共通項は, みんなの合意と云う形をとることで, 客観性を持たすことができるわけです[2]。

いったん共通項が分れば, それに対する差分だけを見つければ良いわけです。例えば, 赤ワインと白ワインがある時に, くだくだ云わずに, 「白取って下さい」と云えるわけです。

ところが, 3 つ目のモノが出てきたとしましょう。これは, 1 つ目のモノとも, 2 つ目のモノとも違う。そうすると不安になるわけです。不安にならない人も不安になって下さい。そこで, 更に, 上位の共通項 (概念) が欲しくなるわけです。それを一生懸命考える事で, より抽象的な, 普遍な共通項 (概念) に到達し, 次々色々なモノに対してこういったことを繰り返す事で, 人間は賢くなっていく。

つまり, 弁証法の基本は, 何等かの矛盾に対して, それを説明するメタな見方をしていくことで, 究極には, 絶対的な (神のみぞ知る) 真理に到達できると云うわけです。例えば, 一般の論文などの書き方がそうですね。こういう解釈があったけれども, これは, ある事象に対しては説明できない。それに対して, このように考えると説明できるとかやるわけです。そういうことを繰り返していくことで, 普遍的な, 所謂真理に到達しようとするわけです。

ただ, 私達が知っている過去の日本や中国の思索する人達にとっては, それは理解した事にならなかったようです。例えば, 仏教で最も知られている般若心経を例に取りましょう。私は, このお経を毎日唱えていますので, ここで読経する事もできるのですが.....。一番有名なところを取り上げましょう。

「色不異空空不異色色即是空空即是色」

色と云うのは, 色気があるとか云う時の色ではなくて, 形あるものということです。従って, 単純に意味を取ると次のような命題になります。

(1) 形あるものは空とは違わない。

(2) 空は形あるものとは違わない。

(3) 形あるものは空と同じである。

(4) 空は形あるものと同じである。

ここでは, 空が難しいと云う事もあるのですが, それ以前に, この命題式は, アリストテレスの形而上学から連綿と続く矛盾律を犯しているわけですから, 分かりにくいのです。悟りの開いた人の言葉である経典には, こういった云い方が非常に良く出てきます。

物事を頭で理解して分かると云うのを, 仏教では, 特に禅宗では「理知」とか「分別」とか云って, それは分かったこととはみなされません。それは, 分かり方としては, 非常に初歩的な分かり方で, 本来何かが分かると云うのは, 主観や客観を突き抜けたところで分からなければならないとするのです。私の好きな哲学者の西田幾多郎, ー彼は若い頃禅宗を通して深い

---

* 実は, (1) は 以前'プログラマについて考える' と云う原稿を SeaMail に載せて頂いたものが (1) です。(2) と書くと'何かな?' と誰か思ってくれるかなと考えただけです。私は, 大人気ないこういう悪戯が好きなものですから:-)

† MHI エアロスペースシステムズ (株)

[1] 酒を一滴も飲めない, そして何時も冷静な同僚, 浅野氏には, 結局, 渡辺さんの質問に答えていなかったと怒られたのですが

[2] このあたりの議論は, cogito ergo sum(我思う故に我あり) から始まる近代哲学の根本的な問題なのですが, 省略しましょう

宗教的体験をしたのですがーは, そのことを発展させて, 「一即多, 多即一」とか「絶対矛盾的自己同一」と云ったりします。乱暴な云い方をしますと, 本当に突き抜けて分かったところには, 言葉がなくて, そこでは一が多であったり, 多が一であったりする。禅では, 不立文字(ふりゅうもんじ)とか云う風に云ったりするのですが, 言葉がない世界ですので, 言葉を使えない。では, どうするかと云うと, 私がそうなんですが, 曹洞宗では, 只管打坐(しかんたざ) つまり, 唯ひたすら座禅をしないさいと云いますし, 臨済宗では, 考案を使っての問答で, 分からせようとする。例えば, 臨済では, 「仏家の本道は何か」と聞くと, 師匠からいきなり殴られたりする。或は, 喝があったりする。それで, あっと分かったりするのです。いわゆる理屈はないのです。

2. 頭で分かることと, 実際に分かること

何か話が少しずつずれているようですね。ついでですから, もう少し脱線したいと思います。

私は, 20 代の前半の頃, 憑かれたようにアイスダンスをやっていました。それで, 今でもスケートに関しては, 多少人より滑れるのではと思いますが, ひとつ例として, 滑ると云うことを考えます。スケートをされない方は, スケートで前へ進のは, 氷を蹴っているからだと思われるかもしれません。しかし, 蹴ったりすると進まないどころか, 下手をすると転んでしまいます。前へ進むのは, 単純な体重移動です。スケーティングレッグに, 上手に体重を掛けることで前へ進んでいきます。さて, ではそのことが分かったとしましょう。いざリンクに立つと, 初心者の場合, ちっとも前に進みません。

私は, アイスダンスと一緒に, (アイスダンスのためだったのですが) 競技ダンスを習っていました。先生は, 桑原佐代子先生と云う方で, 数年前まで, ラテンの全日本チャンピオンだった方なのですが, よっぽど私がひどかったのでしょう, 一年間, 週 2 回レッスンの日は, 歩く練習だけしかやらせてもらえませんでした。歩くなんて何と云うことはないようなのですが, いつまでたっても OK がもらえませんでした。この間, 教えて頂いたのは, 「頭は空に届くようにして, 足の裏は床を突き抜けるように」と云うことだけです。

多分, 皆さんには私の云いたいことはお分かり頂けたと思います。頭で分かることと, 実際に分かることとは別だと云うことが。ただ, それでも, 「おまえの話は体を動かす話で, ものを考えることとは別だ」と思われている人もいるかもしれません。そうすると, ここで私は, ベルクソンの最初の著作「意識に直接与えられたものについての試論」や W. ジェームズを持ち出したくなるのですが, 今日はやめておきましょう。唯, 覚えて於て頂きたいのは, かつての日本や中国の思索家の中には, 論理を超越したところに本当の「分かる」があると考えた人が多かったこと, それからそのことは, 大なり小なり運動を通して (例えば, 自転車に乗ると云ったこと), 我々も経験しているのだと云うことです。このことは, いい加減なことを云って煙に巻く[3]とか, 論理が不要と云っているのではないことに注意して頂いて, 本題に戻ることにします。

3. 超越論的意識

さて, 我々が普通, 「考えている」と云うのは, 何か論理的に推論していることと思っている。少なくとも, 何か論理 (それは曖昧でも良いのですが) を用いて考えていると単純に思いがちです。何故, そう信じることができるのかと云うと, 論理は客観的だからで, それは, いわばメタな公理のようなものです。先程, 脱線したのは, 東洋に於てつきつめるとそうではないと云った人が沢山いると云うことを示したかったのです。但し, これを理解するには, ひたすらここで座禅しなさいと云うのも, 皆さんには随分でしょうから, もう少し別の例を出して, 続きをやりたいと思います。

西田の少し後に活躍した人で, エドムント・フッサールと云う人がいます。有名なハイデッガの先生に当たる人ですから御存知の人もいるかもしれません[4]。フッサールのモノに対する見方は, 現象学と云われます。今世紀非常に力を持った方法で, 先のハイデッガ, サルトルなどに大きな影響を与えています。

さて, この現象学的方法なのですが, 還元と云うのが基本になっています。還元と云うのは, それまでのモノに対する見方を強烈に否定する事です。つまり, 我々は, 先に見たように, 世の中のモノや現象は矛盾があって, それを解決するために, 次々と (普遍的と思える) 論理を編み出して行く。しかし, こうやって編み出された論理や高次の概念と云うのは本当なのだろうか, と問いかけて見る。そうすると, それが本当だと, 決して云えないわけです。つまり, cogito ergo sum ではないのですが, この世の中は, 敦盛の吟にあるような幻でないとは誰にも云えないわけです。ではどうするかと云うと, こういう客観性を, 一旦, 否定しなければならないというわけです。注意しなければならないのは方法として否定すると云う事です。

[3] 丁度私の話?

[4] フッサールも有名な哲学者なので, こういう云い方は彼に失礼ですね。私自身は, ハイデッガが余り好きではありませんし

これを，フッサールは，判断中止とかエポケーとか云い，それによって否定される思いこみをドクサ(憶断)と云います。つまり，還元とは我々が何となく組みあげて来た存在とか概念とかにもう一度疑問符を付けようということで，これは，今まで我々がやってきた事を(時間を順に遡る様に)見直そうと云うわけです。

そうして，例えば，モノがあるという事に対して，次々と還元を行って行く。どんどんとプリミティブな所まで下りて行く。そこに見つかるのは，何かと云うと超越論的意識です。デカルトの懐疑的方法論と良く似ている様ですが[5]，デカルトの懐疑は，基本的に神の存在証明が目標であり，彼はそれができたと思った途端に，所謂還元を止めてしまったのです。フッサールの場合はもっと徹底しています。絶対者をも遮断します。少なくともその徹底度は違うのです。さて，ここで，超越論的意識と云うのが，還元の結果残りました。この超越論的意識と云うのが実はまた分りにくい。また，残ると云う云い方が良くない。超越論的意識は，存在するものではなく，働きなのです。超越的と云うのは，それが主観/客観の世界を超越しているということです。この働きによって，我々の世界が，世界たりえる包括的な働きです。このあたりは，正確にお話しなければならないのですが，取り敢えず超越論的意識と云う働きが残った，この意識を根源の働きとして我々は認識したり考えたりすると云う事を覚えておいて頂いて，詳しくは，フッサールのイデーンを読んで頂きたいと思います。

さて，この超越論的意識なのですが，それはある対象への，志向性を持つと云う云い方をします。つまり，意識は常に何物かへの意識だと云うわけです。例えば，白ワインがある時に，この瓶の形状は一般にワインを入れる物に使われて，ワインと書いてある。そういえば，先程コップに注いで飲んだ時ワインの味がした。従って，あれはワインだとは余り考えません。そう考える必要がある場合も，無論あるのですが，普通は瓶を見て直に分る。これを本質直感とか観取とかいう云い方をします。即ち，働きとしての超越論的意識の向こうから，この本質直感がやってくる。それは，何か推論するのではなくて，向こうからやってくる。もちろん，それには経験が前提となっているのですが，普通我々が分ると云うのは，このようにいきなりやってくる。

[5] 実際 フッサールも「デカルト的省察」と云う論文を書いています

4. さとりの状態とは

さて，再度，日本の思索家に戻りましょう。13 世紀に活躍した人に道元と云う人がいます。私が生まれ育ったのは，福井なのですが，この福井に曹洞宗の大本山永平寺を開いた人です。彼の代表的な著作に「正法眼蔵」と云うのがあります。この中には有名な現成公案と云うのがあって，その中の言葉を少し見てみましょう。

> 「自己をはこびて万法を修証するを迷いとす，万法進みて自己を修証するはさとりなり。」

禅では「花は紅，柳は緑」と言う方もします。これはどういうことかと云いますと，例えば，我々は視界に入る物を見ている様で見ていない。例えば，緑の草原を，唯ふらふらと歩いていたとします。すると，そこに小さなピンク色の花が咲いていた。すると，その刹那，一瞬の事ではありますが，その花で頭がいっぱいになる。視界には，多くのものが入っているにも関わらず，その花で頭がいっぱいになる。多分，このようなことを誰でも経験している事と思います。春先に，何か春を現すものを見たくて，歩き回る。しかし，見つからない。しかし，諦めて，家に帰って庭を見ると小さな梅の花が一輪咲いている。理屈ではない春に全身が包まれる。

ものを，それがあるがままにみる。これは，簡単な様で難しいことです。それをもとめようとすると，かなわなくて，思いもかけずそれはやってくる。つまり，仏法を修めようとする時，そうしようとするとかなわない。それが「自己をはこびて万法を修証するを迷いとす」です。しかし，只管打坐を通じて「万法進みて自己を修証する」ことができれば，それがさとりだと云うわけです。

本当に何かを分ろうとするならば，それを対象として，客観視するのではなく，かと云って主観を働かせるのでもない。それを超越した所に，本当の分るがある。それは，何かと云うと，自分自身が分ろうとするものになることです。将に「仏道をならうと云うは自己をならうなり，自己をならうと云うは，自己を忘れるなり」です。自分を滅した所，西田は「述語となって主語とならないもの」といったりしますが，そういう境地に立たなければならない。つまり，さとりの状態とは，無だとか云われますが，それは，所謂何もないということではなくて，全てがありのまま，自分に現成してくる。無になるのは，自分の意識ではなくて，自分がいなくなる。このことを，心身脱落といったりします。

5. 本当に分かるということは …

色々とお話ししましたが, 云いたかったのは, 物事を見る時の見方として, 客観的な見方, 所謂, 絶対者の見方と云うのが, 正しいと我々は普通思っている。それはある対象に対して, 外から, 或いは上の方から見る見方です。しかし, もう一つの見方と云うのがあって, それは, 外から見るのではなくて, その対象そのものになるという方法だと云う事です。もっと詳しくお話できたら良いのですが, 時間もないようです。最後に, 渡邊さんの質問に答えましょう。

このように我々が分ると云うには, 色々あって, 本当に分ると云うには, そのものになる必要がある。そうした時に, 例えば, フッサールの云う本質観取が可能となるのだけれども, それには (純粋) 経験が必要になる。そうして, 初めて分る。アッと何かが分るというには, アッと分るだけの経験がいる。それは, どうして分るのかと云う筋道があるわけではないので, アッと分るために, あちこち我々の思考は飛び回る, 即ち振動すると云う事です。

長い時間, お話を聞いて下さってありがとうございます。

6. おわりにかえて

最初に云い忘れたのですが, 酔っている私は, 理性的なのですが, 更に, 酔いが醒めると, その時の記憶はないのです。従って, ここに書いたように私が話したかどうかは定かではありません。多分, 違うと思いますが.....

やはり, 渡邊さんの質問には, ちゃんと答えてなかったような気がします。今書いたら数行で終わりましたから。多分, もう一度, 4 時間の時間と, そしてお酒があれば, 続きがあるか。

あと注意が必要なのは, かなりステレオタイプ的な説明しかできなかったという事です。何せ, このような話をする予定は元々なくて, いきなり思い付くまま話しているので, 不正確な表現がかなり多いと思います。例えば, 弁証法を一面的に概念製造器のように描いていますが, ヘーゲル自身は 「一般に世界を動かしているのは矛盾である。矛盾が考えられないと云うのは笑うべきことである」といってますので, 必ずしも, 彼が絶対論理主義者ではありません。こういったことをきちんと説明するためには, 今少し時間を頂きたいと思います。

ところで, 何故私がこういったことに関心を持っているかを少し説明しておきたいと思います。

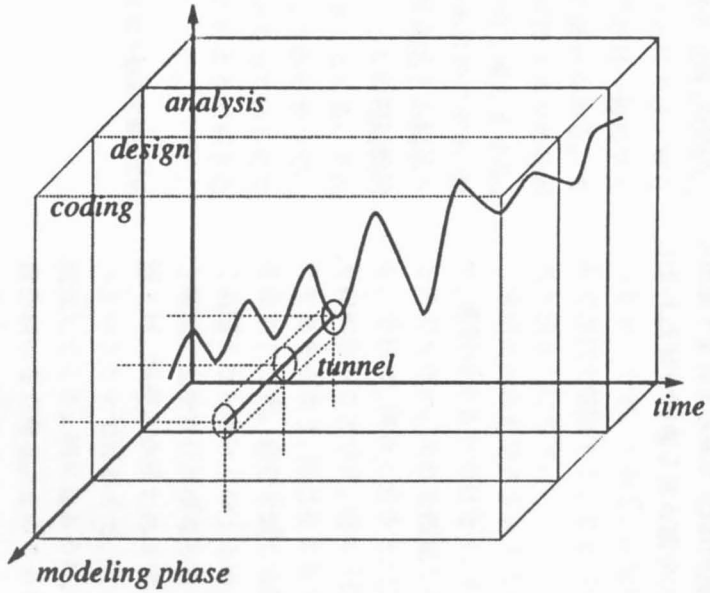
ソフトウェア開発支援環境を考えることの面白さは, 非常にメタなものとして, つまり, 別にドメインとしてソフトウェア開発を選ばなくても適用できる事だと思います。つまり, 人間が何かを分るとか, 考えるとか云う問題は普遍的なわけです。

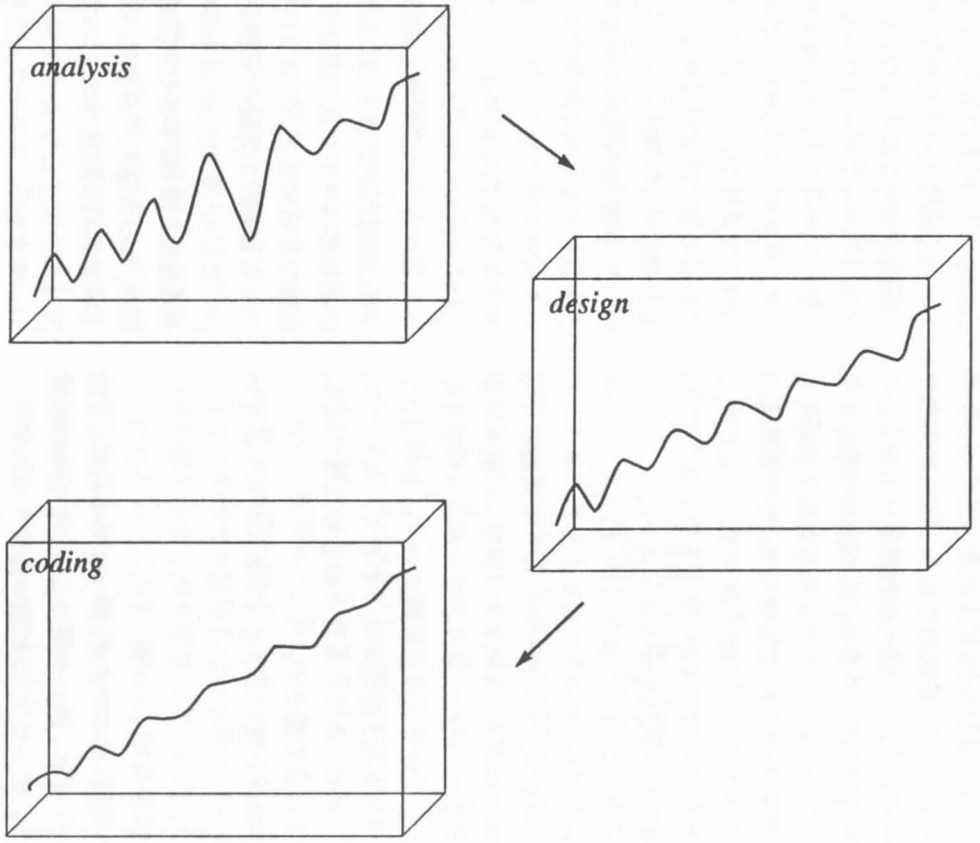
そして, そのことは, おおよそ 2500 年前から色々な優秀な人達 (哲学者や宗教家) が, 考えて来た事でもあり, その人達に学ぶ事は, とても重要な事だと思っています。そして, 未だ, この問題は, 解決していない!

ですから, ソフトウェア開発支援と云う時に, 気軽に自動化とか支援とか云われるとさみしい気がしますし, 分ってないなと思います。何時から, こんなに人間は, 軽いものになったのでしょうか。

1. "Vibration" of our thinking

1.1 What do we think about ?





ラカンとソシュール

伊藤 昌夫 *

補 足

前の原稿 (青空の彼方から) を書いた後, 1 カ月振りに SEA の事務所に行ったところ, 塩谷さんに纏まり過ぎて面白くないと云われました。OHP を読むと自分でも, 良く分からないのです。しかたがないので, OHP にあって, 前の原稿に抜けている部分, ラカン及びソシュールについて補足します。

ラカンやソシュールについて知ることが, 我々ソフトウェア技術者にとって何の役に立つのかと思われるかも知れません。しかし, 彼ら構造主義の思想家たちの一番の問題提起を思い起こしましょう。エピステモロジー, 即ち科学と哲学の諸関係を知ることは, それが未だ解決されてないが故に, 重要であると, 私は考えています。少なくとも, 我々の態度として, 安易な人間に対する理解を, 受入るのではなくて, そこに何等かの見直しをしなければならないと云うきっかけを, 構造主義の思想家たちは, 強く与え続けてくれています。

ラカン (Jacques Lacan) は心理学者で, ソシュール (Ferdinand de Saussure) は言語学者だが, この 2 人は, 言語/記号を通じて結びつく。ソシュールは, 言語学の基本的な道具立てを作った。ラカンは, 精神分析を行う上で, 患者とは言葉を通じてのみ, 理解可能であるが故に, この道具立てを用い, そこに, フロイトが示した無意識, エス, を論理付けた。

1. ソシュール

最初に, ソシュールの示した記号について。記号 (シーニュ) は, 記号表現であるシニファンと, 記号意味であるシニフィエから構成される。例えば, cat の記号表現は, この文字列であったり, \kát\ 云う音である。また, その記号意味は, 概念<ネコ>(ねずみを捕まえ, ペットとして飼われる肉食の ほ乳類) である。

ソシュールによれば, 記号は, 不可分, 一体のものとして次の分数のような形で表現される。

$$\frac{\text{se (シニフィエ)}}{\text{sa (シニフィアン)}}$$

一体と云うのは, シニフィエと切り離して, シニフィアンを考える。或は, シニフィアンを切り離して, シニフィエを考えることはできないと云うことである。それは, 紙の表と裏と形容される。裏を切らずに, 表を分離することはできない。

我々が, 何かについて考える時, 何か云うべきものがあって, それを言葉にしているのではない。言葉を構成する記号の意味と表現が同時に生まれ, それを繋いで行くことによって, 何かを表現する。例えば, 直感的に何かが分ってそれを説明する場合に於ても, その説明のための理屈は, このようにして考えられなければならない。このように, シニファンとシニフィエからなる記号は, 本来的に動的なものである。

* MHI エアロスペースシステムズ (株)

2. ラカン

ソシュールを受け継ぎ, 精神分析の面で発展させたラカンは, 前述のシニファンとシニフィエの関係を次のように表現た。

$$\frac{\text{S (シニフィアン)}}{\text{s (シニフィエ)}}$$

シニフィアンとシニフィエが丁度, ソシュールのものと逆になっている。これは, シニフィエに対するシニフィアンの優位性を表現している。ラカンのシニフィアンの扱いは, ソシュールとは異なっていると云う議論もあるが, ここでは触れないでおく。

2.1 シニファンとシニフィエの関係

さて, ここまでの議論は, 使われている用語は別として, 我々にとって何等新しいことではないようにも思える。例えば, 何らかのプログラム言語に於て, if は, 予約語として, シニフィアンを持ち, 条件分岐の開始と云う意味, 即ちシニフィエを持っている。この時, 次のように思うであろう。シニフィアンとシニフィエの関係は, 単なる一対一の対応づけではないか。

そうではないことを次に示す。先に, シニフィアンとシニフィエは, 別々に考えることはできない[1]と述べた。そこで最初に, 「空虚なシニフィアン」と呼ばれる固有名詞について考える。例えば, 飼っているネコに “たま” と云う名をつけたとする。その “たま” は, 猫と云う言葉が現すものとは違う。猫は, 色々いて, 何らかの意味的かたまりとしての, カテゴリに対する記号である。従って先に見たように, シニフィアンもシニフィエもある。しかし, “たま” は違う。“たま” と名付けた時, 通常, それはシニフィエを持たない。飼っているネコを, 単に, 参照するだけである。

[1] もちろん抽象的な議論では別だが, 少なくとも我々の日常会話上

しかし, 時の経過と共に, この空虚なシニファンは, 豊富なシニフィエを持ち始める。つまり, その猫と1年も一緒に住めば, 「うちの“たま”は悪戯好きで...」と他人に云う時, その“たま”は単なる参照以上の意味, 「そういえば, 最近, 手を引っかかれた」とか, 「この頃, 太っている」とか, 色んな意味が含まれる。つまり, シニフィアンとシニフィエの関係は, 非常に私的で動的であることに注意が必要である。

もう一つ, シニファンとシニフィエは単純な対応関係ではないと云う例を挙げる。我々が日頃, 考える時, 或いは他人に説明する時に用いる図ないしは像について。これらシニフィアンは, 全体として, シニフィエの特性, ないしは関係を表わしている。例えば, 猫の絵は, 猫の姿ないしは, 何等かの特性を示し, ホワイトボードに書かれた言葉の配置は, それら言葉間の関係を表す。一つの図ないし像は, 厳密ではないが, 豊富な意味を持つ。

2.2 シニフィアンの分類

整理しよう。言語表現であるこれらシニフィアンは, パースの分類に従えば, 次の3つに分けることができる。

- 象徴
- 指標
- 図像

プログラム言語に於けるシニフィアンは, 「象徴」であり, 空虚なシニフィアンは, 「指標」であり, 「図像」は, 我々が広くイマージュと呼ぶものである。我々は, これらシニファンを上手に使い分けている。

更に, 「象徴」は, 先に述べたような, “if”ばかりではない。これは, 「象徴」の一部に過ぎない。例えば, 隠喩 (文は人なり) や換喩 (一杯飲む) と云った我々が日常的に良く使用するレトリックもまた, 象徴である。そして, そのシニフィエは, 必ずしも固定的でなく, 社会的, 集団的合意に基づいて定まる。

2.3 3つの世界

この3つのシニファンは, 我々が, 我々の世界で使用するものである。世界は, 我々が経験することのできる世界と, 象徴の世界に大きく分かれる。更に, 象徴の世界は, 理性を通して存在することに疑いをいだかない世界 (例えば, 地球は太陽の周りを回っているということ) と, 元々は空想であるもの (例えば龍) から成っている世界に分かれる。これらを, ラカンは, 現実界, 象徴界, 想像界と呼んでいる。この区分は, フッサールとも一致しているのだが, ラカンの場合これらは結びつけられている「ボロメアンの結び目」としている。

我々は, ソフトウェアで何かを処理させると云う時, 象徴界の象徴のそれも一部 (シニファンとシニフィエが単純な関係にあるもの) を対象にする。それは, そうすることで殆どのことができるからではなく, そうすることしかソフトウェアにはできないからである。その良い例が次の言語取得に関わる問題である。

2.4 言語の取得

主体としての我々は, 如何にしてこの言葉を取得するのだろうか。ラカンは次の様に説明する。生まれたばかりの時点では, 全てが自分の世界である。しかし, 鏡に映る自分を見ることで, 自己を知る (鏡像段階)。この鏡は, もちろん, 文字通り鏡の場合もあるし, 自分が笑った時に, 母親も笑うと云った他人との関係に於ける<鏡>でもある。

次に, 世界は自分のものではないと知る。母親と遊びたい時に相手にしてもらえない, おなかが空いた時に, 何もない。それで身もだえる。今まで自分がいて, 自分はこの世界の主だと思っていたのだが, 何もしないと思い通りにはならない。それは, 耐えられないことで, それを解消するために何かを口にする。その叫びこそが, 言葉である。

このように, 言葉取得の裏には, 常に欲望が隠れている。更に, その欲望は意識されない。何故ならば, 意識されているならば, それは, 言葉なしでは思考できない, つまり意識できない以上, 既にその言葉は取得されている筈だからである。このように, 原初的な, 言語のシニフィエに於て, そこには無意識としての欲望が隠れている。

3. 結び

さて, 非常に乱暴に, ソシュールそしてラカンの主張の一部を眺めて見たが, これだけでも, 言葉が如何に多様なものであるかが分ると思う。そして, そのことは, 人間の住む世界と人間そのものの複雑さを反映している。彼らの主張を全面的に正しいと考えるか否かは別にして, コンピュータによる人間の活動支援を気軽に云える筈のない, 重要な問題を, この簡単な考察の中にも, 幾つか見つけることができる。

- 「象徴」, 「指標」, 「図像 (イマージュ)」といった様々な形態を持つシニファンをきちんと扱えるか。
- 動的に変化し, 機械的に対応付けられないシニファンを知り得るのか。
- 無意識を持てるのか。

実世界は, 机上のようなちっぽけなものではない*

Marc Weiser [著] 伊藤 昌夫 [訳] †

将来のコンピュータに於けるメタファとはなんであろうか。知的エージェントだろうか。テレビ (マルチメディア) であろうか。3 次元グラフィック (バーチャルリアリティ) であろうか。宇宙大作戦 (Star Trek) に出てくるような何処にでもある (ubiquitous) 音声コンピュータであろうか。洗練された GUI デスクトップ環境であろうか。それとも, 魔法の様に我々の望みをかなえてくれる機械か。

このどれでもない というのが正しい答えである。何故ならば, これら全てのコンセプトは, 基本的な 1 つの欠点を持っている。即ち, コンピュータを “見えるもの ”(visible) としているのだ。

良いツールは, “見えない” ものである。“見えない” と云う言葉で, 私が意味しようとするのは, ツールが, 決して我々の意識を侵すことがないと云うことである。眼鏡は良いツールである。眼鏡をかける時, 世界を見るのであって, 眼鏡を見ているのではない。目の見えない人は, 地面を杖でたたくことで, 通りの様子を知る。杖がどうかを知るのではない。もちろん, ツールそれ自身は “見えない” ものではない。しかし, 利用と云うコンテキストに於ては, “見えない” ものである。充分に訓練を積むことによって, 我々は, 多くの明かに困難な事が, そうでないようにすることができる。良いツールは, “見えない” と云う点で, 優れている。

“見えない” ことの価値は, 一般的に理解されていると思う。残念なことに, コンピュータとの相互作用に関する我々の共通のメタファは, “見えない” ツールから離れる方向にある。そして, ツールは, まさに意識されるものとなっている。

マルチメディアを考えよう。そのアイデアは, おおむね, 次の様なものである。人々は, 家庭でテレビを見るのに, 週に何時間も費やしている。それは, テレビが魅力的だからである。それならば, 我々はコンピュータインターフェイスを魅力的にしたいのだから, テレビをコンピュータの中に入れてしまおう。しかし, この推論式には, 幾つかの間違っている部分がある。我々は, 魅力的なことをするのに, 多くの時間を費やしているのだろうか (例えば, 寝たり, お風呂に入ったり, 心配事をしたりはどうか)。何億ドルもかけて撮影所で作られている TV(番組) の魅力を, 普通のコンピュータの TV に移し替えるのであろうか。そして, このエッセーでもっとも重要なことだが, コンピュータインタフェースは, 結局のところ, 魅力的でなければならないのだろうか。魅力は, “見えない” ことの対局にあることに注意が必要である。

知的エージェントを考える。このアイデアは, おおむね, 理想的なコンピュータは人間のようであるべきだ, 異なるのは柔順さだけということである。潜在的に心を引き付ける全てのものに対して, 一寸, 考えて見る必要がある。どうして, コンピュータは人間のようなものでなければならないのであろう。飛行機は, 鳥のようでなければならないのか。タイプライタは, ペンのようでなければならないのか。アルファベットは, 口のようでなければならないのか。車は馬のようでなければならないのか。人間のやり取りには, トラブル, 誤解, 曖昧さがなくて, それはコンピュータが目標とすべきものなのだろうか。更に, 人間のやりとりは, 複数人からなるチームを作り, 維持するのに, 非常に多くの時間と注意が必要である。それは, 2 人の場合ですらそうである。私が話しかけたり, 指令を与えたり, 関係を持つ (親しいと云うには程遠いが) 必要があるコンピュータは, 非常に多くの注意を払う必要のあるコンピュータである。

魔法について考える。このアイデアは, おおむね, 望みをかなえてくれることを目的としている。望みとは, 次のようなものである。私は今のままで, もっとお金持ちでありたい。私のボーイフレンドはもっと賢くて, 魅了的であって欲しい。コンピュータは, 私が興味のある事だけを見せて欲しい。しかし, 魔法は, 心理学や販売学のようなものである。そして, そ

---

* [原題] “The World Is Not A Desktop”, interactions, ACM, pp.7-8, Jan. 1994.
©1994 ACM (Association for Computing Machinery)
この論文のすべてもしくは一部のコピーは, そのコピーが直接の商用利用のために作成もしくは配布されるものではなく, ACM の著作権告示と出版物の名称および発行日付が表示され, コピーは Association for Computing Machinery の許可によるという提示がなされるという条件においては無料で許可される。著作権を明示した, 概要は許可する。そうでないコピー, もしくは再配布は, 費用および/もしくは特別な許可を必要とする。
ACM の許可に基づいて翻訳掲載する。ACM の許可に反しない範囲での再配布は自由とする。
† MHI エアロスペースシステムズ (株)

れは, 良い設計や生産技術に対する危険なモデルであると考えている。その証拠は, 細かな部分にある。即ち, 魔法はそれらを無視するのである。更に, 魔法は, 自分自身を賞賛し続ける。丁度, アラジンの中で Robin Williams の, 魅力的な魔神が, 大げさな説明するのと同じである。

ヴァーチャルリアリティを考える。そのアイデアは, おおむね, 次のようなものである。全身の感覚情報を利用したり, コンピュータとの間で相互作用を行うように移行することで, 我々の身体の入出力チャネルの全てを最大限に利用することができるようになり, ユーザインターフェイスの問題を解決することが出来る, と云うものである。人間の "世界の中での人間 (being-in-the-world)" に対する「入力」メタファの適切さについては, 取り敢えず置いておくとして, VR は, コンピュータを我々の視界から隠して"見えない"ものとするゴールを持っているようである。しかし, 我々の現在のユーザインターフェイスに関する問題は, 我々が充分なユーザインターフェイスを持っていないと云うことである, と云うのは本当に真実なのだろうか。それは, 量の問題, 即ち, 少ないユーザインターフェイスは良くて, より多くのユーザインターフェイスは, 更に良いと云うことなのだろうか。ユーザインターフェイス設計への, この貪欲なアプローチをとることによって, VR は, 関心の核にあるインターフェイスを, 実世界の背後に追いやり続ける。

音声入力を考える。そのアイデアは, おおむね次のようなものである。もし, コンピュータに対して, ただ話しかけるだけで済むならば, コンピュータは, 私を理解しているということであろう。問題は, もし, 今日, 私が自分のコンピュータに話しかけるとするならば, 私は, C や Fortran や C++ で話さなければならないということにある。私は自分のコンピュータに対して, 電子メールを送ることができ, 私の云う事を理解した上での (DWIM-Do What I Mean) 返事が返ってくるならば, 限定されたアプリケーションに於て, 私は, 音声コンピュータの存在を認めるようになるだろう。限定されたというのは, 私の生活時間の多くは, 他の人々とともに過ごしており, 彼等に話しかけたい, 或いは彼らの話を聞きたいと思っているのであって, コンピュータとそうしたいとは, 思っていないからである。もし, 私が, ノートを取ったり, 情報をざっと眺めたい時, 私は, 静かにやりたい。音声コマンドは, SF の世界で良く知られている。それは, そうすることが目立ち, 注意を引くと云う理由によってである。小説は, 我々の注意を惹きつけるものである。しかし, 良いツールは, そうではない。

私は, エージェントや会話理解等に関する研究を重要であると, 考えている。しかし, 問題は, それら全て, "意識することが必要な" 相互作用の領域にあることだ。その結果として, 研究上の議論は, 必要以上に狭いものに限定されている。良い, "見えない" ツールに関する幅広い問題は, 扱われない。"見えない" ことの技術に対して, 我々は, より多くの関心を払うべきだと, 私は考えている。その中には, 我々が, 今日知っているコンピュータを捨てることも含まれる。

"見えない" ことを理解するためには, 人文科学や社会科学は, 特に有益である。何故ならば, それらは, 別の種類の見えないものを明かにすることを専門としているからである。例えば, 民俗学は, 文脈, 環境そして文化的な背景の詳細の重要性を, 示すことができる。また, 女性解放主義者の解体 (deconstructionism) 主義は, 真の理解に対する異なった, 深い所に根づいている視点の必要性を少し教えてくれる。

時計や時計仕掛けの機械は, 過去数百年の間, 技術の隠喩であった。"見えない" 技術は, "見えない" ことの価値を, 我々に知らしめるような, 隠喩を必要とする。私は, その隠喩として幼年時代を提案する。幼年時代は, 遊びで一杯で, 基礎を形作る時で, 常に学び続け, 少し不思議で, 大人になると急速に忘れてしまうような時代である。コンピュータは, 我々の幼年時代のようであるべきだ。即ち, それは, 急速に忘れはするが, 常に我々と共にあり, 我々の生涯を通して努力することなく用いることができる "見えない" ことの基礎である。

参考文献

塩谷 和範 * 伊藤 昌夫 †

1. 「CRC グループの記録」では ...

[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.

CRC 手法に関する初めての論文, オブジェクト指向教育に効果をあげたことが報告されています。

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

SIGENV で輪講を行なっている本。分析・設計の説明に CRC 手法を用いている。オブジェクト指向の重要な概念について例題を示し, これを Objective-C, Smalltalk, C++, Object Pascal の 4 言語のコーディングを示すことにより, オブジェクト指向の考え方と 4 つの言語が一遍に分かるようになることを目指す意欲的な本。訳本は, 誤訳ならびに大胆な意訳が目立つ。できれば原本を参照しながら読んだ方が良い。

[3] ティモシー A. バッド, 羽部正義 [訳], オブジェクト指向プログラミング入門, (株) トッパン 1992, (ISBN 4-8101-8048-4)

[2] の翻訳。誤訳, 大胆な意訳に注意

[4] SIGENV 有志, 「第 10 回夏のプログラミング・ワークショップ報告書」, ソフトウェア技術者協会, SEAMAIL Vol 7. Number 4-5-6-7, 1992

この報告書は, 図書館問題を例にとって CRC 手法・SA/SD 手法・JSD 手法の 3 手法により設計演習を行なったワークショップの記録です。この報告書の pp. 30 - 39 に福良博史氏による「CRC カードによるオブジェクト指向設計」についての解説があります。この中では, 在庫問題の CRC 手法による設計と Little Smalltalk によるコーディング例が示されています。また, pp. 241-246 には, SIGENV 有志による Rebecca Wirfs-Brock の「責任駆動アプローチ (Responsibility-Driven Approach)」論文の翻訳がある。

[5] Bertrand Meyer, 酒匂 [訳], オブジェクト指向入門, アスキー

オブジェクト指向が分かるための定版の本。

2. 「協力関係図による考察」では ...

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

CRC 手法を発展させた責任駆動設計 (RDD: Responsibility-Driven Design) 手法の実践について, 段階的に一歩一歩解説した分かり易い本。「考察」の中で触れたグループ化サブシステム化についても, その方法を具体的に解説している。

* (株)SRA ソフトウェア工学研究所, 1. 「CRC グループの記録」では ... と 2. 「協力関係図による考察」では ... を担当
† MHI エアロスペースシステムズ (株), 3. SA/SD 関連文献 と 4. 補足 を担当

[7] Rebecca Wirfs-Brock and Brian Wilkerson, *Object-Oriented Design: A Responsibility-Driven Approach*, Proceedings of the 1989 OOPSLA, reprinted in Sigplan Notices,24(10):71-75,October 1989.

CRC 手法を発展させた責任駆動アプローチについての論文。この論文では, データ駆動設計と比較することによって, 責任駆動設計の優れた点を論じている。

3. SA/SD 関連文献

リアルタイム系の分析/設計手法は余り知られていませんから, 比較的入手容易なものを中心に, 余り SA/SD にとらわれずに, ご紹介したいと思います。

私が良く利用するのは, Gomaa 氏の DARTS (Design Approach for Real-Time Systems) と云う手法です。DARTS ではワード方式で SA を行った後, その変換図を用いて, タスク設計等を行う方法です。この DARTS に関しては, 幾つかの論文と DARTS を支援する CASE が存在していますが, 日本では, 余り知られていない様です。最近, 金色に輝く良い本が出ました。

[8] Hassan Gomaa, *Software Design Methods for Concurrent and Real-Time Systems*, Addison-Wesley Publishing Company, 1993 (ISBN 0-201-52577-1)

タイトルにもあるように, この本では並列処理も射程に入れています。非常に読みやすく, また, SA/SD からの移行が, 非常に丁寧に対応づけられている他, 他の手法との比較も含まれています。

古典的なものも挙げておきます。

[9] Stephen J. Mellor, Paul T.Ward *Structured Development for Real-Time Systems*

Volume 1: Introduction & Tools (ISBN 0-13-854787-4)
Volume 2: Essential Modeling Techniques (ISBN 0-13-854795-5)
Volume 3: Implementation Modeling Techniques (ISBN 0-13-854803-x)
上記は 3 冊構成で, 出版社は全て, Prentice Hall, Inc です。1986 年の出版。

いわゆる, ワード–メラー法の教科書です。

[10] D.Hatley, I.Pirbhai (立田 [監訳]), 「リアルタイム・システムの構造化分析」, 日経 BP 社, 1989 (ISBN 4-8222-7075-0)

もう一つの有名なリアルタイム構造化分析設計手法である, ハトレー–ピルバイ法の教科書。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 にあります。

[11] R.J.A. Buhr, *Practical Visual Techniques in System Design with Applications to Ada*, Prentice Hall, Inc., 1990 (ISBN 0-13-880808-2)

これは, Ada のコミュニティでは, 有名な本です。タイトルの通りタスクレベルの設計から細部設計に至るまでを, この本で示された方法論及び図式によって, かなり細かくビジュアルに記述することが出来ます。

さて, ここまでは, 特定の設計法に関する教科書ですが, 最近幾つかのリアルタイム設計法を解説した本が出ています[1]。

[12] Shem-Tov Levi, Ashok K. Agrawale, *Real Time System Design* McGraw-Hill, Inc. 1990 (ISBN 0-07-100797-0)

[1] 邦訳されていないのが, 非常に残念なのですが

リアルタイム系の設計に於ける公理的アプローチや V&V の説明が含まれています。後半は，オペレーティングシステムを扱っているのですが，前半だけでも，リアルタイム系の基礎技術を修得するのに良い教科書となっています。

また，文献を集めたものとしては，次のものがあります。

[13] Krishna M. Kavi, *Real-Time Systems-Abstractions,Languages, and Design Methodologies*, IEEE Computer Society Press (ISBN 0-8186-3152-X)

4 章に，分析設計法に関する論文が含まれます。

この論文集を見ていると，リアルタイム系では，やはり SA/SD をベースとするシステムの見方が主流であるように思えます。理由としては，一つには，リアルタイム系は機能重視であること，それから，Time-Critical であることがあります。従って，一般に動的バインディングを特徴とするオブジェクト指向の考え方は，実時間で動作する必要のないシミュレーションは別として，適切ではないように思えます。

例外的に次の本では，オブジェクト指向を利用したリアルタイム系の分析です。

[14] Sally Shlaer, Stephen J. Mellor, *Object Lifecycles Modeling the World in States*, Yourdon Press, 1992 (ISBN 0-13-629940-7) [2]

この本では，大規模，リアルタイムシステムの分析をオブジェクト指向で解くための教科書です。タスクの概念は持っていないものの時間記述はできるようになっています。ただ欠点は，余りにも図表の種類が多く，図表間の関連や，手法そのものを理解するのが困難なことです。従って，これら問題を解決するための CASE ツールが幾つか存在しています。

4. 補足

今回は，道具に関するワークショップでしたので，CASE 関連の文献も紹介しなければならないと思いますが，余りに多いので，今回は，少し古いのですが，次の文献紹介をポイントしておきます。

[15] Veli-Pekka Tahvanainen, Kari Smolander, *An Annotated CASE Bibliography*, ACM SIGSOFT, Vol.15 No.1, pp. 79-92, 1989

[16] Leonard L. Tripp, *CASE Evaluation and Selection* ACM SIGSOFT, Vol.15 No.2, pp. 95 ,1990

現在の CASE に於ける要素技術をサーベしたものとしては，次のものがあります。

[17] Alfonso Fuggetta, Politecnico di Milano and CEFRIFL, *A Classification of CASE Technology*, IEEE Computer, Vol. 26 No. 12, pp. 25-38, 1993

最後に，将来の CASE について考える時，次のエッセーは役に立つと思います。直接，CASE に言及したものではないのですが，道具を考える時に，人間とコンピュータの関わりについては無関心では済まされず，このエッセーの視点 “見えないもの” は，その点で考慮すべき事がらの一つであると思います。

[18] Marc Weiser *The World Is Not A Desktop*, interactions, ACM, pp.7-8, Jan., 1994.

簡単な，そして私の良く利用しているものだけの紹介になりましたが，それ以外については，別の機会に纏めて見たいと思います。

[2] なお，この本には日本語訳が出ていますが，誤訳，訳語の不一致，漏れが散見され，商品としてのレベルに達しているとは思えません。それで，敢えて紹介しないでおきます。改訂されることを望みます。

参 加 者 募 集

第 12 回　SEA 夏のプログラミング･ワークショップ (若手の会'94)

新時代における協調活動：人･社会･技術

－　CSCW の進むべき方向を探る　－

主催：ソフトウェア技術者協会

社会生活をいとなむ上で，人と人の協調活動は有史以来の基本ですが，マルチメディアやネットワークをはじめとする新技術の到来によって，時間や空間の制約を越えたさまざまな活動が可能となり，協調というコトバの意味や形態に，大きな変化が生じつつあります．したがって，CSCW やグループウェアのこれからの展開を考えるさいに，われわれは，ただ単に従来から行われてきた協調活動をそのままの形で支援するだけではなく，そうした新技術が現実の仕事のスタイルにどんな変化を与えるかという「相互進化」の視点から，ものごとを考えて行く必要があります．

今回のワークショップでは，技術の最新動向に関するチュトリアルと，ワーキング･グループによる討論とを組み合わせた形で，《新時代における協調活動とその支援技術》を探ることを企画しました．

チュトリアルでは，現在，国際舞台で活躍されている第一線の研究者を招待講演者にお迎えして，CSCW の背景･概要･未来への展望などについて解説していただきます．続くグループ討論では，

(1) これまでの協調活動にはどのような問題があったか?
(2) 新しい技術はどのような改善をもたらしうるか?
(3) 技術と社会の相互進化の視点から見た将来の方向性は?

といった話題を中心に，ソフトウェア開発をめぐる身近な問題から，人と社会そして技術の相関関係についての本質論にいたるまで，盛岡のおいしい食べ物とお酒を肴に，じっくり語り合うことにしたいと考えています．

なお，このワークショップの目的は，上記の諸問題に対する解答を見出すことではなく，そうした問題点の存在を認識し，ともにその解決を語り合うことのできる人びととの出会いの場を提供することです．日頃，プロジェクト･チームのあり方や運営に疑問を感じておられる新進気鋭の若手技術者の参加をお待ちしています．

開 催 要 領

期　　日： 1994 年 9 月 21 日 (水) 午後から 9 月 24 日 (土) 午前まで
場　　所： 岩手教育会館 (〒020 盛岡市大通 1-1-16　TEL: 0196-23-3301)
定　　員： 25 名
参加資格： 年齢 30 歳以下，経験 2 年以上のソフトウェア技術者
費　　用： 55,000 円 (SEA 正会員)　70,000 円 (SEA 賛助会員)　80,000 円 (一般)
現地集合･現地解散とし，期間中の宿泊費･食費および資料代を含みます．
実行委員長： 筏井 美枝子 (SRA)，野村 行憲 (岩手電子計算センター)
プログラム委員長： 熊谷 章 (PFU)
招待講演者： 石井 裕 (NTT ヒューマンインタフェイス研究所)，中小路 久美代 (コロラド大学)
プログラム委員： 江谷 典子 (富士ゼロックス情報システム)，佐藤 琢美 (岩手電子計算センター)
中小路 久美代 (コロラド大学)，人見 庸 (JMA システムズ)，久島 広幸 (精密形状処理研究所)
アドバイザ： 岸田 孝一 (SRA)
申込方法： 裏面の参加申込用紙の必要事項を記入の上，1994 年 7 月 15 日 (金) までに，郵便，Fax または E-mail で，SEA 事務局宛お送り下さい (E-mail 等の電子媒体を歓迎します).
なお，申込者多数の場合には，プログラム委員会でポジション･ステートメントの内容を審査の上，8 月中旬までに参加者を選定させていただきますので，あらかじめ御了承ください．

申込み先： 〒160　東京都新宿区四谷 3 - 12 丸正ビル 5F　ソフトウェア技術者協会
Tel: 03-3356-1077　Fax: 03-3356-1072　E-mail: wakate94pc@sea.or.jp

***** 裏面に申込用紙があります *****

第12回SEA夏のプログラミング･ワークショップ(若手の会'94)

参加申込カバー･シート

氏　名：＿＿＿＿＿＿＿＿＿＿＿＿(ふりがな)＿＿＿＿＿＿＿＿＿＿＿＿

種　別：□ 正会員(No.　　)　□ 賛助会員(No.　　)　□ 一般

年　齢：＿＿＿＿(歳)　　性　別：□ 男 □ 女　　血液型：＿＿＿＿

会社名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

部　門：＿＿＿＿＿＿＿＿＿＿＿＿　役　職：＿＿＿＿＿＿＿＿

住　所：(〒　　)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

TEL:(　　)-(　　)-(　　)内線(　　)FAX:(　　)-(　　)-(　　)

【ポジション･ステートメント】：　以下の質問1および2については，必ずお答え下さい

Q1. 過去にあなたが体験した"グループ作業(=協調作業)"で，仕事と，仕事以外のものと，できれば，それぞれ一例づつ以下の項目を入れて説明してください．

1.1 その作業の目的および背景
1.2 人数，
1.3 人員構成(全員が同じ程度の専門度，ひとりのエキスパートと後は初歩，etc.)
1.4 物理的／地理的ちらばり
1.5 時間的ちらばり
1.6 みんなで作業して良かったこと
1.7 みんなで作業して困ったこと

Q2. 今回のワークショップに参加するに際してのあなたのゴール(何を討論したいか)を説明して下さい．

☆ 以下の質問はオプションです。答えられるものだけ回答して下さい。

Q3. 1の作業をするときに協調を高めるためにあなたが用いたモノやくふう(戦略的なことを含む)を，以下の項目に分けて，いくつかあげて見て下さい．また，それらを用いたことで，成功だったこと，失敗だったことについて，それぞれ説明して下さい．

(a) 場の設定(対面会議，飲み会，昼食会，etc)
(b) やりかた(KJ法，ブレーンストーミング，etc)
(c) コンピュータ･ツール(電子メール，意志決定ツール，etc)
(d) コンピュータ以外のモノ(FAX, 白板，etc)
(e) 用いた情報，あるいは情報の取り扱いかたのくふう
(f) その他

Q4. 協調作業を進める上で，コンピュータによるどのような支援があると良いと思いますか？　以下の二つに分けてお答え下さい．

4.1 現在の技術で考えられる支援は？また，その長所と短所は？
4.2 将来の技術で可能となるであろう(なると良い)支援は？

Q5. 理想の協調作業のありかたについて，アイデアがありましたら簡単に説明して下さい．

--------　送 付 先　--------

〒160　東京都新宿区四谷3-12丸正ビル5F　ソフトウェア技術者協会
Tel: 03-3356-1077　　Fax: 03-3356-1072　　E-mail: wakate94@sea.or.jp (junet)

*****　**応募締切り：　1994年7月15日(金)**　*****

Call for Papers

Kunming International CASE Symposium '94

November 16-19, 1994
Kunming City, Yunnan Province, China

Sponsored by
Software Engineers Association(Japan)
and
United Nations University/International Institute for Software Technology(Macau)

KICS'94 is the 4th International CASE Symposium in China organized by Software Engineers Association of Japan in cooperation with several academic institutions; the 1st was in Beijing '91, the 2nd was in Urmuqi '92, and the 3rd was in Taian'93.

The symposiums brought together software practitioners, researchers, managers, and educators, from a variety of organizations across the countries to examine the state of software engineering technology and practice in light of CASE in a broader sense.

All attendees will have the opportunity to actively participate: exchange ideas, obtain information, and gain insight about software engineering activities across the countries. We are anticipating an exciting experience. Participation is open to industry, government, and academia. We welcome and encourage your participation.

In this year's KICS'94, we are planning to focus our main attention on the idea of "Domain-Orientation" in a boarder sense. All participants are expected to explain how he/she apply his/her research/technique/ experience to domains. Another typical question expected to be answered might be "whether his/her research/technique/experience is domain dependent or independent, and why it is so?".

All participants should submit full paper or extended abstract to the either of program cochairs by July 15th. Authors will be notified by the end of August whether their paper is accepted or not. Someone may send position statement for panel sessions instead of paper. All submissions are expected to clarify topics which you are interested in. The following list gives the typical candidates of topics.

Topics:
- Domains Analysis
- Domain Oriented Architecture
- CASE and Domains
- Object Orientation and Domains
- Domain Oriented Software Process
- Domain Orientation and Independence

CALENDAR:

Paper deadline:	July 15, 1994
Acceptance notified:	August 31, 1994

ORGANIZING COMMITTEE:

General Chair:	Kouichi Kishida (SRA, Japan)	
Program Co-Chairs:	Huowang Chen (CIT, China)	Kokichi Futatsugi (JAIST, Japan)
PC Members:	Keijiro Araki (NAIST, Japan)	Dines Bjorner (UNU/IIST, Macau)
	Kegang Hao (North-West Univ, China)	Keqin He (Wuhan Univ, China)
	Dehua Ju (ECUST, China)	Kyochul Kang (POSTECH, Korea)
	Akira Kumagai (PFU, Japan)	Wei Lee (BUAA, China)
	Tomoo Matsubara (Peopleware, Japan)	Baile Xi (Fudan Univ, China)
	Toshiharu Yamasaki (Free, Japan)	
Treasurers:	Yoshiaki Sugita (Nihon NCD, Japan)	Ran Zhang (Fudan Univ, China)
Local Arrangement:	Jingshan LO (Kunming Univ, China)	

PAPERS SUBMITTED TO:

Huowang Chen
Department of Computer Science
Changsha Institute of Technology
Changsha, Hunan 410073, China
Tel: +86-731-443-4601 ext 3021
Fax: +86-731-442-9355

Kokichi Futatsugi
Japan Advanced Institute of Science and Technology
Tatsunokuchi-machi, Ishikawa 923-12, Japan
Tel: +81-761-51-1255
FAX: +81-761-51-1116
E-mail: kokichi@jaist.ac.jp

SEA 1993 年～ 1994 年のイベント（実績と予定）

1993	5/28	Seminar (技術の国際化･標準化)	東京
	5/28	Forum (業界リストラクチャリング)	東京
	6/9 ～ 11	Software Symposium '93	仙台
	7/16	Seminar & Forum (ダウンサイジング時代の技術者教育)	東京
	9/1 ～ 4	第 11 回 夏のプログラミング･ワークショップ	金沢
	9/14	Seminar & Forum (インターネット)	東京
	10/13 ～ 16	泰安国際 CASE シンポジウム (TICS'93)	泰安
	10/26	Seminar & Forum (ISO 9000)	東京
	10/28 ～ 30	第 7 回 教育ワークショップ	福山
	11/12	Seminar & Forum (CASE, OO, Formal Approach)	東京
	11/25 ～ 26	第 14 回ソフトウェア信頼性シンポジウム	奈良
1994	1/27	Seminar & Forum (技術者教育)	東京
	1/27 ～ 29	SIGENV ワークショップ	長野
	3/24	Seminar & Forum (SPICE)	東京
	4/21	Seminar & Forum(最新インターネット事情)	東京
	5/26	Seminar & Forum(New Metaphor)	東京
		**********	
	6/2 ～ 4	第 6 回 テクニカル･マネジメント･ワークショップ	秋田
	6/10	SEA 特別 Forum in 名古屋 (マインドネットワーク)	名古屋
	6/15-17	Software Symposium'94	函館
	6/30	Forum(Software Lemmingineering)	東京
	9/21-24	第 12 回 夏のプログラミング･ワークショップ	盛岡
	10/27-29	第 8 回 教育ワークショップ	四国
	11/16-19	Kunming International CASE Symposium	中国･昆明

入会申し込み先
〒 160　東京都新宿区四谷 3 - 12 丸正ビル 5F
ソフトウェア技術者協会（TEL 03 - 3356 - 1077,　FAX 03 - 3356 - 1072）

SEA 入会申込書（正会員：入会金 3,000 円，年会費 8,000 円）　94-06

氏　名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿　(ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿)

生年月日：19＿＿＿年＿＿＿月＿＿＿日　　性別（男　女）血液型（A　O　B　AB）

勤務先名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

所属･役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

勤務先住所（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

勤務先 TEL：＿＿＿＿＿-＿＿＿＿＿-＿＿＿＿＿　（内線＿＿＿＿＿）

勤務先 FAX：＿＿＿＿＿-＿＿＿＿＿-＿＿＿＿＿

自宅住所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

自宅 TEL：＿＿＿＿＿-＿＿＿＿＿-＿＿＿＿＿

資料送付先 & 連絡先（どちらかにチェック）　□勤務先　□自宅

.....

SEA 入会申込書（賛助会員：年会費 1 口 100,000 円，何口でも可）　94-06

会社･団体名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

代表者氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　(ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿)

連絡担当者：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　(ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿)

所属･役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住　所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

TEL：＿＿＿＿＿-＿＿＿＿＿-＿＿＿＿＿　（内線＿＿＿＿＿）　FAX：＿＿＿＿＿-＿＿＿＿＿-＿＿＿＿＿

申込口数：＿＿＿＿＿口

ソフトウェア技術者協会
〒160 東京都新宿区四谷3-12 丸正ビル5F
TEL.03-3356-1077　FAX.03-3356-1072