SEAMAIL

Newsletter from Software Engineers Association

Vol. 13, Number 3 February, 2002

目　次

編集部から		1
SS2001 in 高知		2
基調講演「モデルの図式表現とその意味」	玉井 哲雄	2
玉井先生の講演を聴いて	渡邉 雄一	15
SS2002 in 松江		
プログラム委員会からのお誘い		19
論文/報告/デモ募集		25
CMM とソフトウェアプロセス改善	塩谷 和範	27
ほんとうのプロセス改善を目指して	坂本 啓司	34
加算方式にしただけでは安値落札はなくならない	松原 友夫	47
褒める	端山 毅	50
事件としての CMM	岸田 孝一	52
オブジェクト指向とレトリック	玉井 哲雄	59
音楽とソフトウェア創りの似ているところ	野呂 昌満	63
旅の名残	熊谷 章	74
論文募集 - ISESE2002		76

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増加し，現在，北は北海道から南は沖縄まで，500 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 24社を数えます．支部は，東京以外に，関西，横浜，名古屋，九州，広島，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： 深瀬弘恭

常任幹事： 荒木啓二郎 高橋光裕 田中一夫 玉井哲雄 中野秀男

幹事： 伊藤昌夫 大場充 落水浩一郎 窪田芳夫 熊谷章 小林修 桜井麻里
酒匂寛 塩谷和範 篠崎直二郎 新谷勝利 新森昭宏 杉田義明 武田淳男
中来田秀樹 野中哲 野村行憲 野呂昌満 端山毅 平尾一浩
藤野誠治 松原友夫 山崎利治 和田喜久男

事務局長： 岸田孝一

会計監事： 橋本勝 吉村成弘

分科会世話人 環境分科会(SIGENV)：塩谷和範 田中慎一郎 渡邊雄一
教育分科会(SIGEDU)：君島浩 篠崎直二郎 杉田義明 中園順三
ネットワーク分科会(SIGNET)：人見庸 松本理恵
プロセス分科会 (SEA-SPIN))：伊藤昌夫 塩谷和範 高橋光裕 田中一夫 端山毅 藤野誠治
フォーマルメソッド分科会(SIGFM)：荒木啓二郎 伊藤昌夫 熊谷章 佐原伸 張漢明 山崎利治

支部世話人 関西支部：臼井義美 小林修 中野秀男 横山博司
横浜支部：野中哲 藤野晃延 北條正顕
名古屋支部：筏井美枝子 石川 雅彦 角谷裕司 野呂昌満
九州支部：杉田義明 武田淳男 平尾一浩
広島支部：大場充 佐藤康臣 谷純一郎
東北支部：布川博士 野村行憲 和田勇

賛助会員会社：ジェーエムエーシステムズ 東芝ＩＴソリューション ＳＲＡ ＰＦＵ
テプコシステムズ 構造計画研究所 富士通 オムロンソフトウェア
キヤノン 富士通エフ・アイ・ピー 新日鉄ソリューションズ
ダイキン工業 オムロン 富士電機 ブラザー工業 オリンパス光学工業
リコー アルテミスインターナショナル NTTデータ ヤマハ
福井コンピュータ 日本ネスト オープンテクノロジーズ ＳＲＡ西日本
（以上24社）

ＳＥＡＭＡＩＬ Ｖｏｌ．１３，Ｎｏ．３ ２００２年２月２８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）
〒１６０-０００４ 東京都新宿区四谷３－１２ 丸正ビル５Ｆ
T: 03-3356-1077 F: 03-3356-1072 sea@sea.or.jp
印刷所 有限会社 錦正社 〒１３０-００１３ 東京都墨田区錦糸町４－３－１４
定価 １，０００円 （禁無断転載）

編集部から

☆

やはり船便の SEAMAIL です．この号は，塩谷和範 & 岸田孝一の２人が共同で編集しました．

☆☆

昨年６月の SS-2001 における玉井先生の基調講演について，渡邉雄一さんに Q&A を含めたレポートを書いていただきました．読者の便宜のために SS の Proceedings から玉井先生の論文も再録しておきました．

☆☆☆

今年の SS は，ワールドカップ・サッカーで予想される交通の混乱を避けて，いつもの６月ではなく，７月に開く予定です．開催地は，山陰の城下町・松江．先日開かれた第１回委員会の報告を兼ねて，PC メンバーの方々に企画の PR を書いていただきました．

☆☆☆☆

前号でも特集した CMM をめぐる「空騒ぎ」は，まだおさまっていないようです．そこで，この号でも再び SPI および CMM の問題を，いくつかの論文およびエッセイによって考えることにしました．

☆☆☆☆☆

坂本啓司さんの論文は，昨年の夏に亡くなられる直前に執筆された遺稿です．また，松原さんのエッセイは，「日本版 CMM」との関係で問題になっている政府調達の適正化について，METI が公表した文書に関する辛口のコメントです．アメリカ在住の端山さんからは，坂本さんの追憶を含めた文章を寄稿いただきました．編集担当の２人も，それぞれ，CMM に関するサーベイ(塩谷)，昨年２月の SPIN Meeting での発表のまとめ(岸田)を書かせて貰いました．

☆☆☆☆☆☆

巻末には，玉井先生，野呂先生，そして熊谷さんから寄せられたそれぞれ味わいのあるエッセイを載せました．お楽しみください．

☆☆☆☆☆☆☆

モデルの図式表現とその意味: なぜUMLがはやるのか?

玉井哲雄
東京大学 大学院情報学環

1 はじめに

今年 (2001 年) の国際ソフトウェア工学会議 (ICSE2001) は 5 月にカナダのトロントで開催されるが，そのチュートリアル・プログラムを見て驚いた．全部で 22 あるチュートリアルのうち，1/3 にあたる 7 つのタイトルに UML という語が入っている．さらに，タイトルには含まれていないが，10 行足らずの概要説明の中に 3 個所も UML という文字を躍らせているテーマもあって，それを含めると 8/22 という高比率となる．

なんで，これほど UML がもてはやされるのか，正直なところ筆者にはよく分からない．UML は，オブジェクト指向モデルの記述言語ということだが，これまでソフトウェア開発で使われてきた数多くの図式の集大成，というおもむきもある．そして，それらの図表現の中に，過去に培われたモデル化や設計のノウハウが蓄積されているとも言える．オブジェクトモデルの記述言語というよりも，むしろそのようなより広い範囲で用いられる共通言語として，多くの人に受け入れられている，ということかもしれない．

2 グラフによるモデル化

2.1 さまざまなモデルのグラフ表現

多くのモデル化技法では，なんらかの図式表現を用いることが多い．図式にはたとえば次のようなものがある．ここには，UML に入っているものもいないものも挙げている．また，分析時のモデルだけでなく，概要設計やプログラム設計で使われるモデルの図式も含めている．

1. 制御フローを表すもの (フローチャート，PAD, HCP, SPD, など)
2. データフローを表すもの (データフロー図，SADT，HIPO，など)
3. データ構造を表すもの (Jackson 構造図，など)
4. データ関連構造を表すもの (ER 図，意味ネットワーク，オブジェクトモデル図，など)
5. 状態遷移を表すもの (状態遷移図，Statechart, ペトリネット，など)

これらは皆，丸だか四角だか箱といった形が閉じた図形群と，それらを結ぶ折れ線だか点線だか矢印といった線群から構成されるという，共通の特徴を持つ．数学的に言えば，頂点集合と辺集合からなるグラフ構造をとる点が，共通である．

グラフ構造がこれだけよく用いられる理由として，次のような要因が考えられる．

1. 図として視覚化することで「モデル」らしく見える上，描きやすく直観的に分かりやすいこと

2. 対象となる世界の「もの」を頂点で表し,「もの」と「もの」との関係を辺で表すことにより，世界がグラフによって自然に表現されること

3. 「もの」と「もの」との関係には推移律や何らかの推移的な関係が成り立つことが多いが，それはグラフ上の経路の追跡することに対応させて理解でき，また経路探索のアルゴリズムもよく研究されていること

しかし，直観的に分かりやすいことが，恣意的であいまいな使用を助長している面もある．実際，状態遷移図やデータフロー図の概念を学生に説明すると比較的容易に理解するように見えるが，試しに図を描かせてみると，とても状態遷移やデータフローを表していないしろものができてしまうことがよくある．実は，学生ばかりでなく，かなりのベテランのソフトウェア技術者でもそういうケースを見かける．

図としてグラフ構造が愛用されるのは，実はソフトウェアの世界に限らない．新聞，雑誌，教科書，報告書，企画書，論文，発表スライドなどに登場する図の多くは，グラフ構造を表現したものになっている．そして，それらの図の意味は往々にして曖昧である．たとえば，矢印が何を表しているのか，因果関係か，時間関係か，物の流れか，制御の流れか，といった点が，同じ1つの図のなかで揺れているものをよく目にする．箱の方も同様で，プロセスのような処理主体と，データのような処理の対象が同じ箱だか丸だか表され，混在していることは日常茶飯事である．

グラフはなまじ直観的に理解しやすいがために，かえってその意味（semantics）をあいまいに捉えがちなのであろう．したがって，おなじグラフ構造であるという共通性と，その意味の違いを意識的に考えることが重要だと思われる．

2.2 グラフ表現によるモデル化の特徴

すでに述べたように，グラフ表現では通常，頂点に「もの」（概念，オブジェクト，プロセス，データなど）を結びつけ，辺にものとものとの関係を結びつける．その辺の役割によって，モデルを，構造を表す静的なモデルと振舞いを表す動的なモデルとに分類することもできる．

1. 静的なモデル

   頂点AとBを結ぶ辺： AとBがある関係にあることを示す（AからBへの有向辺の場合は,「AはBとある関係をもつ」ことを示す）．ER(実体関連）モデル，クラス図，意味ネットワークなどが，代表例．

2. 動的なモデル

   頂点AからBへの辺： AからBへ移動する（制御フロー, 状態遷移など），またはAからBへものが流れる (データフロー, ワークフローなど).

静的なモデルと動的なモデルを混同することは少ないが，動的なモデル同士，たとえばデータフローと制御フロー，状態遷移とフローチャートの間の混同はよく目にする．

表1に，代表的なモデル図式の頂点と辺の組み合わせを示す．

表 1: 代表的なモデルのグラフ構造

	頂点	辺
データフロー	プロセス	データの流れ
ER	実体	関連
状態遷移	状態	関連
JSD	プロセス	データストリーム結合 データベクトル結合
フローチャート	実行単位	制御の流れ
ペトリネット	プレース，トランジション	発火とトークンの流れ

2.3 グラフとモデル上の物理量

グラフは頂点と辺の関係というトポロジーを表している．しかし，グラフを用いてモデル化を行う場合は，頂点や辺に何らかの物理量を結びつけて扱うことが通常である．物理量といったが，ここでは「情報」を相手にしているから，多くの場合は物理量とは呼びにくい．しかし，たとえば電気回路のような典型的な物理を対象としたグラフ・モデルは，多くの示唆を与える．

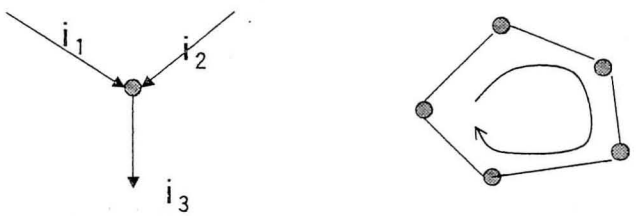
電気回路の場合，頂点に電位，辺に電流が結びつけられる．辺には電位の差としての電圧も対応する．この物理量とトポロジーとを関係づける法則として，キルヒホフの法則 (Kirchhoff's Law) がある．

1. キルヒホフの電流法則

   頂点の周りに流入する電流の総量 (流入を正，流出を負として計る) は，0 である．あるいは，電流は各辺の上を，停滞したりとぎれたりすることなく連続に流れる．

2. キルヒホフの電圧法則

   閉路 (loop) に沿っての電圧の総和は 0 である．



キルヒホフの電流の法則　　キルヒホフの電圧の法則

図 1: キルヒホフの法則

これに関連して，辺の接続に関し直列と並列の 2 種類があるが，それぞれの接続における物理的な性質は，次のように記述することができる（これはキルヒホフの法則から導出される）．

1. 直列：　電流は一定．電圧は加算

2. 並列：　電圧は一定．電流は加算

このような「流れるもの」とそれを駆動する圧力がそれぞれ辺と頂点に結びつけられるモデルは多いが，すべてのモデルがそれにあてはまるわけではない．しかし，キルヒホフの法則のように，

1. 各頂点の周りに出入りする辺全体についての性質を考える．

2. 辺が接続する経路に沿って成り立つ性質を考える．

ことは，基本的である．

2.4 グラフ理論の初歩概念の応用

グラフ理論というほど大げさなものは必要ないが，教科書に出てくるようなその初歩の概念は，ある程度役に立つ．

まず，グラフの種類に，いくつかのものが考えられる．それによって，モデルの性質にも違いを生じる．

- 有向グラフ/無向グラフ
- 木，(有向) 無閉路 (DAG)，一般グラフ

  木構造は階層構造と対応し，無閉路構造は共通的な下部構造を持つ階層構造に対応する．一般グラフは階層構造を持たない，いわゆるネットワーク構造と対応する．
- 頂点集合の分割 (e.g. ペトリネット)，辺集合の分割 (e.g. PAD)

グラフの一部をなし，そこに含まれる辺の両端点を頂点集合として含むものを部分グラフという．部分グラフを１つの頂点に縮約すれば，その部分グラフをカプセル化，抽象化したグラフとなる (図 2)．この操作は抽象の階層を取り扱う自然な枠組みを提供する．実際，データフロー図や Statechart では，この方法が用いられている．

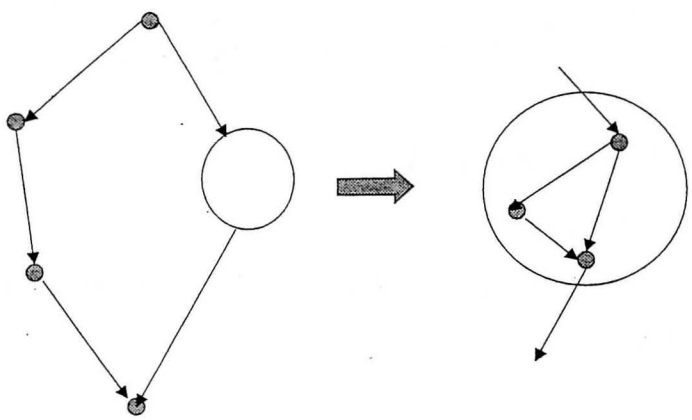


図 2: 部分グラフ

2.5 グラフの弱点と対策

原理的にはどんな複雑なモデルでもグラフとして図示できるとしても，実際に1枚の図に描ける頂点や辺の数は限られる．1つの理由は物理的な制約で，紙やディスプレイの画面は限定された面積しかないから，いくら詰め込んでも高が知れた量しか表せない．さらに認知的な制約がある．よく引用される G. A. Miller の「魔法の数字 7 ± 2」[6] が主張するように，人間が自然に認知できるものの数は，案外少ない．頂点の数をある程度押さえても，辺の数が多くてクモの巣状になった図は，とても理解できない．

これはグラフによるモデル表現の弱点であるが，これに対して以下のようにいくつかの対策はある．

- 配置の工夫： 辺の交差をなるべく小さくするような配置や，その他の基準でグラフを見やすくするアルゴリズムがいろいろ提案されている．
- 画面操作の工夫： ズーミングやスクローリングが活用される．
- ハイパーグラフ： グラフの一般化として辺を2点間ではなく多点間の関係として扱うハイパーグラフを使うと，場合によっては図が簡素化される．
- 階層化：すでに述べたように，上位グラフの頂点を展開すると下位グラフとなるという階層構造を導入することで，1レベルの図は簡素化される．この階層構造をツールで支援することも簡単である．

  関連して，頂点の箱に入れ子構造を描けるようにすることも，よく見られる工夫である．

  Statechart では，この両者の工夫が採用されている．

3 例としての状態遷移モデル

グラフ構造を持つモデルの典型例として，状態遷移モデルを見てみよう．状態遷移モデルの歴史は非常に古いが，並行分散システム，応答型システムがますます普及し，その振舞いの検証に対する要請が高まるにつれて，改めて重要性が見直されている [1].

3.1 状態遷移モデルの基本的な性質

3.1.1 グラフ構造と意味論

状態遷移モデルは，典型的な動的モデルである．そのグラフ表記である状態遷移図は，次のように特徴づけられる．

頂点　状態
辺　　遷移

グラフとしての位相構造 (topology) に対応するモデルの構成要素は，この「状態」と「遷移」のみである．しかし，状態遷移モデルに意味 (semantics) を与えるための重要な概念として「事象 (event)」がある．各辺，すなわち遷移には事象がラベルとして結びつけられており，その事象が発生したときに対応する遷移が起こるというように解釈される．

状態遷移モデルで記述される対象は，内部状態を持つ機械，すなわち状態機械の振舞いである．以下では，状態機械の振舞いをプロセスと呼ぶ．事象は状態機械に外部から与えられる入力と見なしてよい．状態の集合の中に，通常はただ1つの初期状態が存在することを仮定する．また，状態の

空でない部分集合として，終了状態が存在することも通常は仮定する．ただし，反応型(reactive)のシステムを状態遷移モデルで記述する場合には，終了状態は存在しないか，特異な状態（deadlock)を指すことになる．

このような仮定のもとに，状態遷移モデルは次のような動的なプロセスを記述するものと解釈される．

1. プロセスは最初は初期状態にいる．
2. ある状態にいて，事象が発生した場合，その状態から出ている遷移の内で，その事象をラベルとして持つものがあれば，その遷移の先の状態に移る．
3. 終了状態に遷移したらそこでプロセスは終了する．

この説明から分かるように，状態遷移モデルでは「現在どこかの状態にいる」ことが仮定されている．現在いる状態は，必ず１つあり，かつ１つに限る[1]．ある状態にいて，ある事象が生じたとき，その事象をラベルとしてもつ遷移が常にちょうど１つある場合は決定的な状態遷移モデル，複数ある場合は非決定的な状態遷移モデルとなる．また，１つもない場合は，その事象は無視され，何の遷移も起こらないと解釈される場合と，何らかのエラーと解釈される場合とがある．

遷移は瞬時に起こるものと想定され，状態はある経過時間を持つものと想定される．もちろん，瞬時とか一定の経過時間という概念は相対的であり，モデルで考慮されている時間軸において，無視できる時間間隔を瞬時と見なすわけである．

状態機械を外から見た場合，その機械は記憶を備えたもののように振舞う．つまり，機械の動作は現在の入力によってのみ決まるのではなく，過去の入力の経過に依存し，それを「記憶」していると見えるからである．この「記憶」は実は状態に他ならないが，外からこの状態は見えない．外部と機械が共有するものは事象である．すなわち，状態遷移モデルの２つの構成要素である状態と事象のうち，状態は外からは見えず，事象は外から見える（あるいは外から与える)という区別を意識することは，重要である．

3.1.2 具体例

具体例を示そう．ボーリングの点数を計算するプロセスのモデル化を考えてみる．点数計算の規則は，次のようである．

1. ストライクの得点は，10 に次の２回の投球で倒れたピンの本数を加えたものである．
2. スペアの得点は，10 に次の１回の投球で倒れたピンの本数を加えたものである．
3. オープン・フレームの得点は，そのフレームの２回の投球で倒れたピンの本数で，加算はない．

実はさらに，最後の 10 フレーム目に特殊な処理が必要で，そのための規則があるが，ここではそれは考えないことにする．

図３は，この点数計算をするプロセスを状態遷移図でモデル化した例である．この図では，状態集合は { 始，オープン，スペア，ストライク，ダブル，２投前，ストライク後２投前 } である．なぜ，上に述べた規則には現れない「ダブルという概念が現れているのだろうか．逆に，３回連続

[1]ただし，これは逐次プロセスの場合である．並行プロセスの場合は，後で述べるように「現在の状態」が状態の集合を指すことが一般的である．

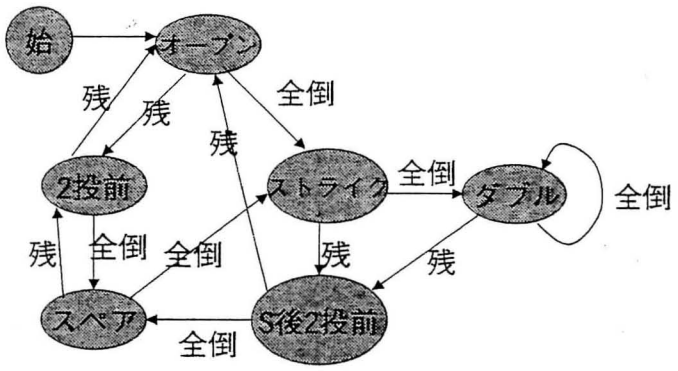


図 3: ボーリングの点数計算の状態遷移モデル

ストライクのターキー（やフォースやそれ以上) という状態を入れていないのだろうか．あるいはフレームという概念に相当するものが直接現れなくてよいのだろうか．

また，事象集合は { 全倒，残 } である．舌足らずな言い方だが,「全倒」は立っているピンが全部倒れたという事象（1 投目ならストライクだし，2 投目ならスペアになる) を表し,「残」は 1 本以上ピンが残ったことを示す．なぜ，倒したピンの本数か残った本数によって事象を区別しないのだろうか．

これらはモデル構築の方針によって選択されたものである．モデルを作るには，目的がある．目的に必要なことがらが選択され，不要なものは捨象されるという「抽象化」がなされなければならない．ここでは，ボーリングの点数を計算するのに必要な場合分けを表すような状態を考えている．だから，事象は投球をしてピンが倒れた時点を取り，1 投目か 2 投目かを直接には区分せず，全部倒れたか否かという 2 つに 1 つの区別に抽象化しているのである．

この状態遷移図では終了状態を明示していない．上に述べたような抽象化をしているために，フレームという概念は表現されておらず，実際のボーリングのゲームが 10 個のフレームで終了するとしても，そのことはここで記述しているプロセスにとっての終了とは直接関係しないといってよい．

3.2 状態遷移モデルの拡張

状態遷移モデルは，さらに概念や記法を付加して拡張されることが多い．ここではとくに，出力事象，遷移条件，並行システム，状態の階層化という 4 項目を取り上げる．これらは D. Harel の Statecharts で採用されており，その Statecharts を採用している UML でも踏襲されている．

3.2.1 出力事象

これまで事象は状態機械にとっての入力であるとしてきた．機械といいながら，入力のみで出力がないのでは，役に立たないのではないだろうか．しかし，状態遷移モデルのもっとも簡素な形である有限状態オートマトンでは，出力がない．そこでは一連の入力事象の列にしたがってオートマトンが状態を遷移させていったとき，終了状態に達したらそれまでの入力列は受容されたとし，そうでなければ受容されないと判定することで，1 つの言語を定めている，と解釈される．

しかし，やはり出力を扱えた方が，モデルの記述力は増す．そこで自然な拡張として，遷移に伴い状態機械から外界に向けて出力事象を発生するようなモデルが考えられる．出力事象を伴う状態遷移モデルは，有限状態オートマトンを拡張した順序機械に対応する．

例として，図3のボーリングの点数計算を取り上げる．このモデルは本来，点数を計算するための情報を出力しないと役に立たない．そこで，出力事象として，次の3つを考えることにする．

1. 加算1（倒したピンの数を加える）

2. 加算2（倒したピンの数の2倍を加える）

3. 加算3（倒したピンの数の3倍を加える）

ここで，括弧書きしたものは，この出力事象を受け取る外部の計算主体が行うはずの計算をコメントとして書き加えたもので，図3の状態遷移モデルにとっては，3つの事象が区分されるということのみが意味を持つ．なお，通常のフレーム単位の点数計算に合わせるなら，

1. 加算1（倒したピンの数を現在のフレームに加える）

2. 加算2（倒したピンの数を現在と1つ前のフレームに加える）

3. 加算3（倒したピンの数を現在と1つ前と2つ前のフレームに加える）

という方がよいかもしれない．とくに10フレーム目の処理を考えると，この表現の方が扱いやすい．

この出力事象を加えて図示したのが，図4である．遷移を表す辺は，「入力事象/出力事象」によってラベルづけされている．

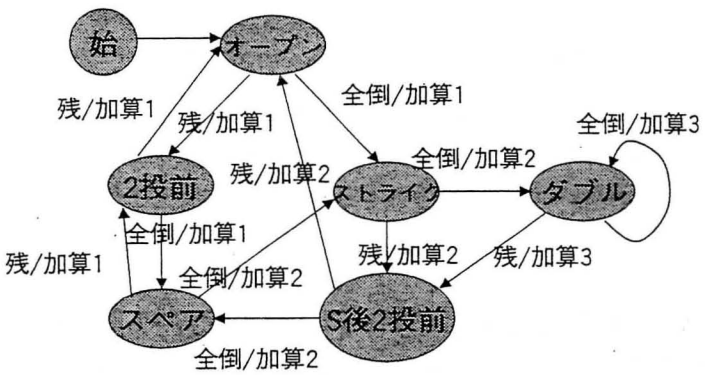


図 4: ボーリングの点数計算モデル (出力事象つき)

3.2.2 遷移条件

ある状態にあって，そこからの遷移を起こす事象が発生した場合に，常に遷移が起こるのでなく，さらにある条件が満たされたときにのみ遷移が起こるという記述をしたいことがある．とくに後で述べる並行システムの場合，遷移条件を書けることで，状態の数の節約になることが多い．

3.2.3 並行システム

並行システムは複数の状態遷移プロセスの共存という形で表現できる．複数の状態遷移プロセスを並べることで,「現在いる状態は1箇所ではなく，プロセスの数だけあることになる．それぞれのプロセスがまったく独立に動作するのでは並行システムとしての面白さはないが，各プロセスは事象を共有することで相互作用を起こす．事象を共有するとは，1つの入力事象の発生でそれをラベルにもつ複数の遷移が同時に起こることや，あるプロセスの出力事象が別のプロセスの入力事象になること，などを意味する．

3.2.4 状態の階層化

状態遷移モデルをシステム開発のためのモデル分析に使用する際のもっとも大きな問題点は，実用規模の問題に対し，ほとんど常に状態数が爆発的に大きくなってしまうことである．この状態数の爆発に対処するにはいくつかの方法が考えられるが，有効なものの1つは，いくつかの状態を包含した親状態 (superstate) を導入することである．包含されている状態を子状態と呼ぶとすると，プロセスが親状態に含まれる任意の子状態にある時，またそれらの子状態の間でのみ遷移が起こっている間は，1つの親状態にあると見なす．親状態の外から中への遷移や，親状態の中からその外への遷移が，その階層レベルでの遷移になる．これは，部分グラフを1つの頂点として階層構造を作るという一般的な方法の例である．

3.3 Statecharts

前節で述べたような状態遷移モデルの拡張を導入し，実用規模のシステムに対して起こる「状態数の爆発」に対処した手法の代表的なものとして，D. Harel による Statechart がある．Statecharts の特徴を挙げると，次のようである．

1. 遷移には事象と，場合により条件や動作が結びつけられている．
2. 複数の状態をまとめたクラスタを導入できる．クラスタは内部を隠して抽象化された一つの状態とみなすことができる．逆に一つの状態を複数の状態からなる部分遷移モデルに詳細化して示すこともできる．
3. クラスタに対してデフォルトの初期状態を指定することができる．
4. 過去の遷移の「歴史 (history)」を使って，たとえばもっとも最近あるクラスタを離れた時にいた状態に入る，といった指定ができる．
5. And 独立な状態群が並行的に動作することを，記述できる．
6. 遷移の分岐 (fork) と合流 (join) を記述できる．

3.4 UML における状態遷移図

UML では，オブジェクトを状態機械と考え，その振舞いを状態遷移図で表す．記法としては，基本的に Harel の Statechart を採用している．たとえば，図5は Fowler の本 [2] からとった．

図の「動作中」と名前を付けているのが親状態 (superstate) である．その中に3つの子状態があるが，それを隠して1つの状態と見なしたものが「動作中」である．遷移の矢印に付けられた

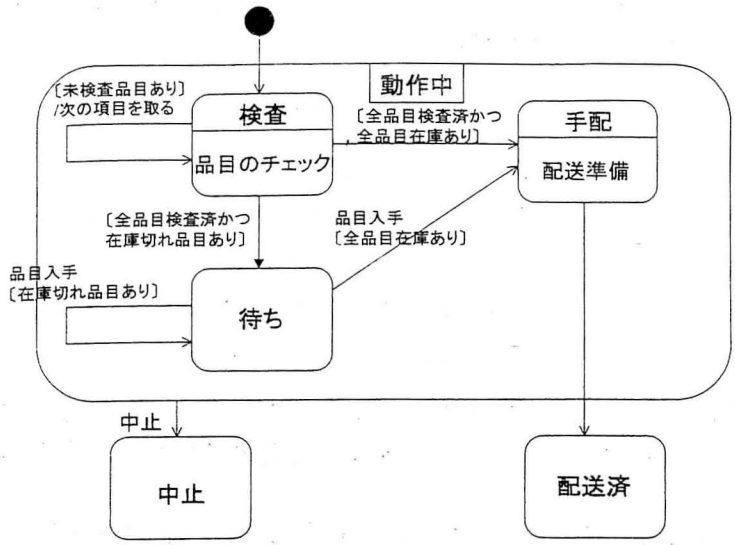


図 5: 親状態のある状態遷移図

ラベルのうち,〔〕で囲われた表現は遷移が成立する条件を記述するものである．単なる文字列は，遷移を引き起こす入力事象を示す．また，/の後に書かれた文字列は，遷移に際して出力として生じる事象を表す．

3.5 状態遷移モデルの系譜 (1)–オートマトン

状態遷移モデルの歴史は古い．ここでその系譜を振り返ってみることは，状態遷移モデルの本質を知るのに役立つだろうとの意図からである．その系譜を大きく２つに分けてみた．１つは有限状態オートマトンに代表されるオートマトンの系譜である．もう１つは AI や OR でこれまた長く取り扱われてきた探索手法の系譜である．本節では，オートマトンの方を取り上げる．

3.5.1 有限状態オートマトン

計算機科学の教科書なら必ず有限状態オートマトンと正規表現の関係について，かなりのページが割かれているはずである．また，コンパイラや言語処理の本でも，字句解析や形態素解析の方法として，有限状態オートマトンは避けて通れない．

有限状態オートマトンの定義は改めて述べるもでもなく，3.1.1 で導入したモデルそのものである．有限状態という以上，状態集合は有限であることが仮定されている．正規表現の定義はどこででも見つけられるだろうが，ここでも煩を厭わず載せておくことにする．

アルファベット Σ 上の正規表現を考える．正規表現は，次のように構成的に定義される．

R1 ϵ(空列を示す)，$\emptyset$(空集合を示す)，および Σ の任意の要素は，それぞれ正規表現である．

R2 R_1 と R_2 が正規表現の時，$R_1 \cdot R_2$ および $R_1 \cup R_2$ は正規表現である．R が正規表現の時，R^* は正規表現である．

正規表現は，Σ を文字とする文字列からなる集合の部分集合（この部分集合を言語と呼ぶ）に写像される．その写像を与える規則は，次のようである．

R1' $\epsilon, \emptyset, a \in \Sigma$ は，それぞれ $\{\epsilon\}, \emptyset, \{a\}$ に写像される．

R2' 正規表現 R_1 と R_2 が写像された言語を L_1, L_2 とすると，$R_1 \cdot R_2$ は $L_1 \cdot L_2$ に，$R_1 \cup R_2$ は $L_1 \cup L_2$ に写像される．ただし，$L_1 \cdot L_2 = \{a \cdot b | a \in L_1 \ \& \ b \in L_2\}$．ここで文字列に対する演算 $\cdot$ は，連接である．

正規表現 R が写像された言語を L とすると，R^* は L^* に写像される．ただし，$L^0 = \{\epsilon\}$, $L^i = L^{i-1} \cdot L$ として，$L^* = \bigcup_{k=0}^{\infty} L^k$．

さて，有限状態オートマトンが受容する言語のクラスと，正規表現で記述できる言語のクラスとが同じであるというのが，Kleene による有名な定理である．証明は省略するが，直観的には明らかに見える．

ある言語を正規表現や有限オートマトンで表すことは，頭の体操としても面白い．以下の言語を正規表現ないし有限オートマトンで表してみよ．

1. 入力が 0 か 1 だとして，1 が偶数個ある文字列のみを表す正規表現，あるいはそれのみを受理するような有限オートマトンを作りなさい．

2. 入力が 0 か 1 だとして，それを順に左から右に並べて得られる 2 進数が 3 の倍数のもののみを表す正規表現，あるいはそれのみを受理するような有限オートマトンを作りなさい．

3.5.2 順序機械

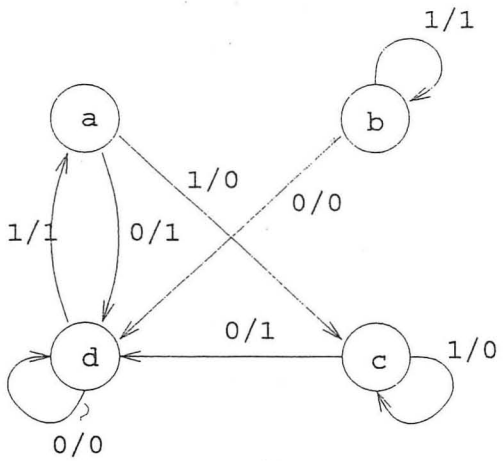
順序機械は有限状態オートマトンに出力が加わったものである．主に，論理回路の設計や解析で用いられてきた．その形式的な定義は次のとおりである．

順序機械 M は，5 組 $(Q, X, Z, \delta, \omega)$ によって定義される．ここで

Q: 状態集合
X: 入力集合
Z: 出力集合
δ: 状態遷移関数，$X \times Q \to Q$
ω: 出力関数，$X \times Q \to Z$

状態および入力の集合が有限であれば，状態遷移関数と出力関数は表として表すことができる．なお，上の定義はいわゆる Mealy 型機械と呼ばれるもので，出力関数 ω を，入力に無関係に状態だけで決まるようにしたもの，すなわち ω: 出力関数，$Q \to Z$ という形のものは，Moore 型機械と呼ばれる．

順序機械をグラフとして表現する場合は，頂点には対応する状態をラベルとして与え，辺には，その遷移を起こす入力とその時に出される出力の対を，ラベルとして与える．たとえば，下の表は $Q = \{a, b, c, d\}, X = Z = \{0, 1\}$ であるような状態遷移と出力の表であり，図はそれに対応する状態遷移図である．



遷移関数表

	0	1
a	d	c
b	d	b
c	d	c
d	d	1

出力関数表

	0	1
a	1	0
b	0	1
c	1	0
d	0	1

3.5.3 LTS とモデル検査

ラベルつき遷移システム (Labeled Transition System) は，並行性を組み入れた状態遷移モデルと同じと考えてよいが，最近,「モデル検査」と呼ばれる手法とともに脚光を浴びている [1]．システムの振舞いを LTS で記述し，それに要求される性質（たとえばデッドロックに陥らないとか，ある望ましい状態にいつかは必ず達するとか）を，多くの場合，時相論理 (temporal logic) 式で与え，それをモデル検査という手法で確かめるのである．この手法の特徴は，自動的に結果が出ることと，検証できない場合に反例が提示されることである．この手法は，1980 年代の前半に提唱され，とくにハードウェアの論理回路や通信プロトコルの正しさの検証に用いられてきたが，この数年，一般のソフトウェア（とくに並行分散システムの振舞いが問題となるもの) を対象とする研究や実践が盛んになってきている．

並行システムの形式的なモデル化と，その性質の数学的な取り扱いについては，C. A. R. Hoare の CSP(Communicating Sequential Processes) や Robert Milner の CCS(Calculus of Communicating Systems) が有名である．これらについては，それぞれの著者の教科書を読むとよい [4, 7].

3.6 状態遷移モデルの系譜 (2)–探索問題

AI や OR の分野で，古くから探索問題が研究されている．手法は解の空間を探索して目標を見つけるという形をとるが，通常はグラフ上の経路をたどって目標となる状態に達するという定式化がなされる．

1 つの典型は,「山羊と狼とキャベツ男」とか「宣教師と土人」といったクイズである．また，関連して，ゲーム，とくに 2 人の盤ゲームが類似の定式化で分析される．そこでは「盤面」が状態に対応し,「手」が遷移に対応する．

また，ロボットの行動計画 (planning) の問題も，状況とそこで可能な操作の選択を計画するという形で，状態遷移モデルに帰着する．さらにこれらを一般化した Simon & Newell による GPS(General Problem Solver) も同じ形式といってよい．

川合の作った問題を借用して，状態遷移の概念を使うときれいに解が求まるものを挙げる．なお，彼の「コンピューティング科学」[5] は名著である．

1. 3 個のランプ付きボタンが一列に並んだ装置があり，次のように振舞う．

あるボタンを押すと，そのボタンのランプ状態 (点灯・消灯) が反転するとともに，
そのボタンの隣のランプの状態も反転する．

真中のボタンには「隣」が2個あることに注意すること．最初はランプがすべて消えているものとする．真中のランプだけを点灯させるためには，ボタンを最低何回押せばよいか．

2. 容量 a リットルと b リットルの2つのバケツを使って，どちらかのバケツに c リットル残るようにする問題を考える．a と b は正の整数とする．実行できる操作は次のとおり．

   (a) 1つのバケツを蛇口からの水でいっぱいにする．

   (b) 1つのバケツに入っている水を全部捨てる．

   (c) 片方のバケツの水を，他方のバケツへ移せるだけ移す．全部移ることもあれば，移される側がいっぱいになった時点で終わる場合もある．

   初めは，両方のバケツとも空とする．$a = 4, b = 5, c = 2$ の場合に，最低何回の操作で解に達するか．

参考文献

[1] Clarke, E. M. Jr., Grumberg, O., and Peled, D. A.: *Model Checking*, MIT Press, 1999.

[2] Fowler, M. and Scott, K.: *UML Distilled: Applying the standard object modeling language*, Addison-Wesley, 1997.

[3] Harel, David: StateChart: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol. 8(1987), pp. 231–274.

[4] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall, 1985.

[5] 川合慧：「コンピューティング科学」，東京大学出版会，1995.

[6] Miller, G. A.: The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, *The Psycological Review*, Vol. 63(1956), pp. 81-97.

[7] Milner, R.: *Communication and Concurrency*, Prentice-Hall, 1989.

玉井先生の基調講演を聴いて

渡邉 雄一
(アスキー)

1. はじめに

玉井先生のSS2001 (高知)での基調講演「モデルの図式表現とその意味」は，予稿集[1]に，立派な論文が用意され，実際の講演内容も，それに沿って，ソフトウェア世界に関係の深い図式表現の特徴を丁寧にそして深遠に示したものであった．通常，この手のそれに多い，原稿なし，OHPを中心にしたお話だけ，という予想は，見事にはずれてしまい，事前に「SS2001の基調講演の記録を書きましょう」などと安請け合いをしたことを後悔している．

というわけで，ここでは，当日の講演のあらすじのメモと，講演の質疑応答の記録をここに記し，最後に講演の副題にあった「なぜUMLがはやるのか？」に関連して若干の感想などを述べたい．

2. SS2001@高知 講演からのメモ

[1] 図のもつ曖昧さ/意味の相違

矢印：流れを示す vs 制御（コントロール）を表す

箱の意味： 処理（プロセス）を表す vs データを表す．
この差異は，動作の主体 vs 動作の対象 とも云える

[2] （表すものの）多様性への対応

頂点集合，辺集合の要素に変化をつける

逆に，バラエティーを多くすることで，図のもつ特性を損なわれ判り辛さが生じることにもなる．

「モデル化する際の抽象化の主眼は何なのか？」が重要

[3] 静的 vs 動的

これらの混同理解は，経験的に，少ないと思う

静的なモデル
宣言的な関係を図示したもの

動的なモデル
矢印のあるもの ： 今いるコントロールの場所が移る
矢印のないもの ： データが流れる

[4] 図における位相関係

モデルが理論的なものに立脚していることがある．たとえば，物理の法則のようなもの：例としてキルヒホッフの法則

ソフトウェアの世界では，扱う対象や性質も異なるので，一様に書けるわけではないが，「物の流れ」などを表しているときには有効な記述になっていることが多いようだ．

[5] グラフ理論の存在

一般に我々の目的のためには，詳細まで熟知している必要性は低い．理論の基礎概念はとても役に立つだろう．

部分グラフによる階層化は，そのよい例の１つだろう．

DFDの階層構造や状態遷移図での状態の階層化は，これを利用している．

有向，無向

連結性：強連結性，弱連結性

木構造や閉路の有無といった構造上の特性

[6] グラフの表現

行列表現：表現としては，データ量として無駄が多い場合もあるので扱う対象ソフトウェアの特徴・性質によっては，適当でない場合も少なくないが，数学的に扱えるので強力．

[7] グラフの表現の弱点

魔法の数字７±２

意味的＆認知的な制約

物理的な制約に対する工夫

1) 配置の工夫

頂点の配置による，辺が交差しない工夫

3D的な配置

2) ズーミング

GUIの発達の中で進んできた技術

[8] 階層化

非常に強力な手法．具体例としては，部分グラフ．
ただし，オブジェクト指向の「クラス図」には，この手法は有効に機能しない．DFDでは，頂点を展開すると，それはまたDFDになるが，クラス図の頂点を展開しても，それはクラス図にはならない．

状態遷移図では，さらに and の結合で，図が大きくなるのを防ぐ工夫をしている．

[9] 状態遷移モデルので２つの例

オートマトン

探索手法

後者では，特に，記述することにより，具体的な反例を求めることが期待される．

3. 質疑応答

Q: 講演の冒頭で「なぜUMLが流行るのか分からない」と仰られたのでUMLに対して，かなり懐疑的否定的な意見をお持ちなのかと拝聴していましたが，必ずしもそうではないということですよね．で，どうして，そう思われたのかについてお伺いしたい．

A: 正直なところ「どうしてこれほどまでにUMLが持てはやされるのかな？」という気持ちです．無価値なものだというつもりはありません．UMLのどこらへんに価値があるかというと，これは基本的には，記法(notation) で，さらに，その上に何か統一的な方法論を使うことを意識して記法の開発がされていることは間違いないと思います．

しかし，UMLとして見たときには，これは色々な図式の記法の寄せ集めでしかありません．それがなぜ便利かというと，みんなが同じようなクラス図を書いたときに，みんながちょっとづつ個別に改良を加えてしまうと，それを他の人が読んで理解するときに大変だといったことで，みんなで決めたものを使うということの意義は，よく理解できます．

しかし，一方で，単に「図」とはいいますが，個々の図は，長いソフトウェアの世界での技術の蓄積が (UMLで採用されている) 図の中に結集・蓄積されていると思えば，その１つ１つをキチンと理解して使うことによって，その図におけるノウハウ (大げさにいえば人類の英知といったもの) が，さ

らにはその背後にある設計上のノウハウやモデル化をする上でのノウハウが生かされることに，意味・意義があると思うのです．

しかし，言語としてのUMLは，何か明確なポリシーで統一的ですっきりとした言語とは思えない．つまり言語としてのインパクトは乏しい思える．なのに，たとえば，ICSE in トロント(カナダ，2001/05)でのチュートリアルでプログラム全体の 1/3 もUMLを取り上げていたことなど，ちょっと異常な状況と映ります．

Q: 講演中の階層化の説明で，クラスというのは階層化に向かないという指摘がありましたが，実際の分析の中では，パッケージ化とかサブシステム化といった部分グラフといった考え方が適用できるのではないか．階層化が向かないという理由を説明していただきたい．

A: クラスとパッケージの概念レベルは違うものです．単純にいえば，異なる２層になります．逆にいうと，パッケージというものを導入しないといけないところに (概念の構造として) 無理があるわけです．

パッケージをまとめたものは，パッケージなのかも知れませんが..... 状態遷移図やDFDでは，きれいに再帰的な関係が描けますが，クラス図の場合は，パッケージとクラスは別の概念です．

しかも，パッケージの作り方ですが，これもなかなか難しいところで，先にクラス図を作って，どこかにまとめてそれをパッケージにする，あるいはサブシステムにするというような説明がなされることが多いと思いますが，階層というよりは，トップダウンに近いと思います．見方(とらえ方)としては，階層といえないこともないですが，やはり理論的な説明としては苦しいといわざるを得ないでしょう．ですので，もともと再帰的にきれいに書けるものとは，区別して考えるのが適当だろうというのが，私の主張です．

Q: モデルのあいまい性について．仕様記述の中で，グラフ構造を書くということを突き詰めていくと，あいまい性の排除は可能になるのでしょうか．

A: グラフにしたから，あいまい性がなくなるわけではありません．これがまず私の主張したいことです．まず，グラフの各頂点や辺が何を表しているかを明確にして，そこに厳密性ないし統一をとっておかないと，(グラフとしての意味に) あいまいさが生じます．また，そのような意味でのあいまいさが排除できたとしても，グラフ化するという手法が対象とするモデルの性質・特性に適合しているのかどうかは 別の問題です．「グラフによる表現」は，何らかの頂点に対応する物などの「ある単位」と，それらの間にある何らかの「関係」あるいは「動き」などがあるときに，それらをモデル化するには有効です．しかし，それ以外の場合は，何もグラフの関係に乗るわけではありませんし，また仮にグラフで表現できたからといって，分析したい性質が明瞭になってくるというわけではないでしょう．ですから，(グラフ表現以外に) 代わるものが何かと尋ねられても，色々とあるので，一概にはいえません．

4. 図表現の難しさ

環境分科会 (SIGENV) で UML1.3 の仕様書を眺め，UMLのメタモデルや OCLに接することができた．その延長として，OMG から公開されてる UML2.0 Superstructure のドラフトなどを斜め読みしてみて，その壁の厚さにたじろいでいる．要は，難しいのであるが，その難しさを一層そう思わせている理由の１つに，UMLのメタモデルのモデル図がある．UML1.x と比較すると，その記述は，個々の要素に限定的になろうとするために(他との影響度を弱めて独立性を高めようとするために)，逆に全体を正しく把握することを拒む傾向にある．もちろん，このようなモデルを設計・提案したり，厳密にそれらを検証する必要がある人は限られているが，実際に開発するアプリケーションでも同様の場面に遭遇することは容易に想像できるだろう．

もちろん，ドメインにより程度の差はあろうが，アプリケーションの概要をとらえることが容易だろうが，上記のようなメタなものを (ましてドラフト段階の完全性の保証の乏しいものを) 理解することは容易ではない．まして，その完全性を図の上で検証しようとすることは容易ではないように思

う.

ところで，「なぜUMLが流行るのか」という理由の１つに，UMLに関連したツールの存在が，高価なWSではなく，廉価なPCの上でも動作するという時代背景があるかと思う．一昔前の高級CASEツールが，WSを自由に使える一部のハイエンドユーザに限定されていたことからすると，環境の整備は，ずっと容易になっている．しかし「仏作って魂入れず」ではないが，手にした環境を，ツールベンダーの意図どおりに活用されているかというと，そうはいかないようである．先日も著名なＲ社のツールを展示会で見たとき，説明員にそれとなく尋ねたところ，案の定，コードから図を書かせて何かの資料 (説明員の言葉を借りれば，それは仕様書だそうだが) に添付するといった使われ方で終わっていることが多いと嘆いていた．

なぜか．少なくとも，図を書くノウハウが，ないのだろう．お絵かきツールにも若干は図を綺麗に見せる(たとえば，水平や垂直の線をゆがみを補正するなどの) 機能は備わっているが，図の構造をきれいに整理するというのは，なかなかない．まあ，包含関係とかならば，宣言的に書いて，あとから整理するときれいにできるのかもしれないし，半導体の設計のように，対象が，ｎ層の平面となっていればそこへの畳み込みも，ある程度アルゴリズムを駆使すればよいのかもしれない．きれいな図は魅力的である．しかし，実際に，現実の問題を解いていく際に，自分の書いた図を，あるいは描いてみたモデルを適切に評価してくれる人，あるいは評価してもらえる環境にいる人がどれだけいるのだろうか．

記述した図が，論理的に正しくとも，それを美しく見せる工夫も必要かもしれない．いずれにせよ，ソースコードを書く経験に比べて，図式を書く研鑽を積んでいる人は，多いとは思えない．

開発期間や費用の圧縮，さらに高機能で適度な品質が成果物に要求されるソフトウェア開発の多くの現場では，既存の資産やノウハウを最大限が生かす方策が求められ，それに呼応するように，パターンなどの有用なアイディアが具現化されている．しかし，図の意味だけをとっても，玉井先生の講演にあったように，考えるところは少なくない．どんな書き方をするかにもよるが，図を書くのにかなりの労力を伴うことは，みんなが実感しているところであろう．

外見上，似て非なるものは，差分だけ書くということが容易にできるのかなと，つまらない悩みに遭遇することも少なくない．そういった労苦が無にならないアプローチを模索しても，難しい問題は，やはり記述されてみると難しい．やっかいなことである．

しかし，UMLが多くの人に支持されているのは，それによる恩恵が期待されてのことであろう．ここは，ひとつ，経験豊富な先人に，その有益性などをSEAMAIL誌上で披露していただければと思う次第である．

[資料]

[1] SS2001 予稿集(ISBN 4-916227-08-5)
[2] http://www.uml.org/uml

SS2002 プログラム委員会からのお誘い

- 第１回ミーティングの報告を兼ねて -

２月はじめに，ソフトウェア シンポジウム2002 のプログラム委員会が愛知県蒲郡で開催され，今年のシンポジウムの企画や運営をどうするか，また，論文や報告の集め方をどう考えるかなど，熱心な話合いが行われました．

去年の SS2001 in 高知 は，あらかじめ予定されていた企画セッションだけをシングル トラックで行うという形でしたが，今年は，そうした企画セッションと従来の一般公募の論文/報告とを組み合わせ，またツール．デモも含めて，多角的なプログラム編成が考えられています．

そこで，論文公募のPRの意味をこめて，実行委員長・プログラム委員長から一言ずつのご挨拶を，さらに，プログラム委員の方々からは，それぞれの企画セッションについての構想について，書いていただきました．

松江へのお誘い

実行委員長
田中 一夫 (JFITS)

今年の Software Symposium は， FIFA World Cup Korea/Japan の開催が６月なので，参加者の皆さんの不便さを避けるために，７月に開催いたします．開催場所は，島根県松江市で，昨年の高知同様に，歴史ある都市での開催となります．

松江市のHomePageを見ると，以下のように書かれています．

松江市の生いたち

http://www.web-sanin.co.jp/matsue/m-gaiyo.htm

松江市は，古代出雲の中心地として早くから開け，奈良時代には国庁や国分寺が置かれていました．

地名の由来は，慶長１６年（１６１１年）堀尾吉晴が亀田山に城を築き，白潟・末次の二郷をあわせて松江と称したことにはじまります． 江戸時代には堀尾氏3代・京極氏1代・松平氏１０代の城下町として栄えました．そして，この頃，今日に見る都市の基礎が形成されました． 明治４年（１８７１年）廃藩置県によって県庁が置かれ，同２２年4月（１８８９年）全国の３８市とともに市政を施行しております．当時の市域は４．７８平方キロ，人口３５，５１３人でした．その後，昭和９年から３５年にかけて８回にわたり周辺の村を合併，また，公有水面の埋立などを経て現在の市域となっております．

この間，昭和２６年（１９５１年）には松江国際文化観光都市建設法が制定され，奈良市・京都市と並んで国際文化観光都市となりました．

さらに，昭和４１年（１９６６年）には中海地区新産業都市の指定を受け，山陰の中核都市として発展してきております．

今回の会場は，松江駅の側にある「松江テレサ」です．松江駅から，徒歩１分と非常に駅から近い会場を選びました．この会場から江戸時代の歴史を尋ねる路は，決して近くはありませんが，レイクラインという観光周遊バスが用意されていて，１時間強の時間で，ぐるっと市内を一周することでができます．

ここまでは，松江市の紹介でしたが，今年のSSは，論文や経験報告を一昨年までと同様に幅広いカテゴリーで募集し，PCの審査を経て，発表という形をとります．もちろん，PCの方々で検討していただいいていますが，そうした公募論文の発表だけではなく，ミニ チュートリアルやパネルも取り入れた企画セッションも含めて，それぞれのカテゴリーごとに，多彩な構成のイベントにしたいと考えています．

みなさん，ぜひ，松江でソフトウェアの真髄を議論し，美味しい宍道湖の魚介類とお酒で，楽しみませんか？

ソフトウェア シンポジウムの魅力

プログラム委員長
阪井 誠 (SRA-KTL)

ソフトウェア シンポジウムには1995年から毎年参加してきましたが，今年は久しぶりに裏方にまわり，南山大学の青山先生と共にプログラム委員長をさせていただくことになりました．微力ではありますが，より多くの方々に楽しんでいただけるようなシンポジウムにしたいと考えています．

私も，もともとは開発現場での経験を発表するために参加していただけだったのですが，ソフトウェア シンポジウムに参加するたびに，多くの刺激を受けることができました．その時々の最新の研究成果や，これから普及が期待されている技術の実践成果だけでなく，シンポジウムが開催されている地元の文化など，日々のソフトウェア開発では得られない多くの知識やアイデアを得ることができました．

また，ソフトウェア シンポジウムでは，他社の多くの技術者の方々と知り合うこともできました．発表した内容に対する率直な意見の交換や，パネルディス

カッションでの熱の入った議論など，社内では得られない別の視点を持たれている多くの方々と交流できました．技術者は経験の蓄積を重視し，常に身の回りの出来事を効果的に解決するように工夫しているので，ついつい自分の世界の中で問題を見つめてしまいがちですが，別の視点で問題を見つめなおすことで，その問題の本質を改めて考えることができました．

さらに，ソフトウェア シンポジウムは，産学交流の場でもありました．技術者は開発現場での問題(ニーズ)を日々の努力で解決しています．これに対して，大学をはじめとするアカデミアの方々は問題をモデル化して，数学をはじめとする様々な理論(シーズ)を用いた，抜本的な問題の解決法を提案されてきました．また，アカデミアの単純すぎるモデルに対して技術者は，開発現場のニーズとの違いを何度となく議論してきました．このように，産学が議論を通じて協力し合える場であることは，ソフトウェア シンポジウムならではといえる特徴だと思います．

さて，このように魅力がいっぱいのソフトウェア シンポジウムですが，その魅力は毎年変わる開催地とプログラムによるところが大きいでしょう．歴史や文化のある開催地と魅力的な論文，イベント，多くの議論の場がこれまでのソフトウェア シンポジウムを支えてきたと思います．

今年は，開催地として，島根県松江市が選ばれました．古代卑弥呼の時代からの歴史が語られ，出雲大社，松江城，宍道湖など，小泉八雲（ラフカディオ・ハーン）が愛した島根県は，シンポジウムの開催地にまさにうってつけの場所です．島根県ならではといえる歴史的な講演や，Rubyで有名な まつもとゆきひろさん(松江在住)の講演を予定しています．他に，海外からのスピーカをを招いての CHI(コンピュータ ヒューマン インタラクション)の講演も予定しており，最新の技術動向が気になる方も満足していただけると思います．

あとは，論文・事例の投稿と参加者による活発な議論があれば，多くのみなさんに満足していただけるシンポジウムになるでしょう．たくさんの投稿・参加をお待ちしています．

ネットワーク セッションについて

中野 秀男 (大阪市立大学)

ネットワーク セッションでは，「セキュアなシステム/プログラム」をテーマとして，最近ますます増加するウィルスの問題や，アタックに対する防御(IDS，セキュリティ パッチ等)について，いくつかの講演や報告を手がかりにして，パネル風の討論を行ないたいと考えています．想定されるサブテーマとしては，以下のようなものがあります；

1．ウィルスチェック プログラム
サーバの現状と問題点
2．IDSの現状と限界
3．セキュリティパッチはどうしてますか?
4．セキュアなプログラムに配慮していますか?

いま考えているセッション構成は，上記のサブテーマに関する議論のきっかけになるような講演や発表をいくつか冒頭に置き，それをもとにフロアからの討論を行おうというかたちです．SEA のシンポジウムなので，ソフトウェア技術，プログラマの視点，そして技術関連管理者の立場など，いろいろな観点からの議論が盛り上がるものと期待しています．

このテーマに関係のある経験報告や論文の投稿をお待ちします．

プロセス セッションについて

高木 徳生 (オムロン)

プロセス セッションでは，研究論文/経験論文を発表するセッションと事例報告にもとづく討論のセッションという２つを作ろうと，企画を進めています．後者の事例報告セッションは，数年前から SS で行われているもので，主に，いろいろな企業での活動のご紹介および意見交換の場として，大変好評をいただいています．事例報告の応募は，発表用のスライドだけで結構ですから，「論文を書くはちょっと ... 」と思われている方も，お気軽に投稿ください．

このところ，プロセスに関しては CMM を中心とする話題がにぎやかですが，このセッションでは CMM だけにこだわらず，プロセス改善一般について広い議論の場にしたいと考えています．また，恒例になっている夜の BOF も企画します．ぜひ，多数のご投稿，ご参加をお待ちしております．

セッション担当 PC メンバから一言：

■ 私がプロセス改善の仕事に就くきっかけは，このシンポジウムに参加したことでした．プロセス改善の議論は，CMM を中心として，まさに今盛り上がりを見せているところですが，まだまだ「プロセス改善って何？，CMM って何？」と思っている方は，たくさんいることと思います．そういう意味でも，もう一度原点に戻って，多くの方にプロセス改善の実際の活動を知っていただく機会は大切だと思います．どんな些細な事例でも結構ですので，多くの方の投稿をお待ちしています(菅原)．

■ 企業でも政府でも CMM への関心が高まってきていることを力強く思います．企業からの実践体験論や，大学からの新提案をぜひお寄せください (福山)．

■ このシンポジウムは プロセス改善に日々汗を流している企業の方には最先端の研究を，研究者の方には現場の問題をと，どちらも楽しめる貴重な場です．

ぜひ奮ってのご参加をお待ちしています (高木).

保守 セッションについて

田中 創 (アイ・ティ・フロンティア)

最近，まったくの新規開発は少なくなり，現行システム/プログラムを運用・活用しつつ，その一方で部分的なあるいは段階的な改良・改善を実現したり，さらには，刻々と変化する運用環境に適応させるために，維持管理のための保守を行ったりしているのが大方の現状であって，効率化のための保守プロセスの見直しがユーザーや経営から要求されています.

保守費用上昇の原因としては，新たなビジネス上の要請により頻繁に改修を行なわなければならない既存システムが増えていて，その保守が困難であることが考えられます．また，既存システムの保守に対する不満は，システムにバグや欠陥があるからではなく，ドキュメントが貧弱であることや，プログラムを変更するとシステムが壊れやすいことなどが考えられます.

これまで，オブジェクト指向，ダウンサイジング，ERP など新しいシステム開発技術が導入され，既存システムが次第にリプレースされて行けば，年数を経たシステムは次第になくなり，ソフトウェア保守の問題もなくなるだろうと考えられていました．しかし，冒頭に述べたとおり期待どおりに進展していないのが事実です.

ソフトウェア保守の改善方法も考えられてきましたが，これまでのシステム開発技術では，その効果は限られたものでした．特に，数年前に既存システムを保守しやすいように改善することを目的としたソフトウェアのリエンジニアリング (リバース エンジニアリング ツールを活用した既存システムの再利用の方法論) が，一時話題となりましたが，保守現場の環境はあまり従来と変わっていません.

さらに，情報システム部門の専門会社化・子会社化が進み，既存システムの有識者 (たとえば COBOL のプロフェッショナル etc) への依存度が高まる属人化が進んでいます．しかし，今後保守担当者の高齢化に伴なう人事異動等による人材の流出により，保守現場から有識者がいなくなる無人化が問題点として顕在化しています.

このような現状にあって，保守現場で保守の効率化を目指して，日頃努力されているみなさんから，経験にもとづく知恵をシンポジウムの場で，数多く披露していただきたいと考えています.

アスペクト指向 セッションについて

藤野 晃延 (インアルカディア)

このセッションの目的は， ASoC (Advanced Separation fo Concerns) /AoSD (Aspect-oriented Software Development) の考え方についての理解を深めると同時に，課題克服の試みを広く喚起することです.

セッションの主題は，「関心事の分離 (Separation of Concerns)」です．このテーマ [Parnas72] [Dijkstra76] は，ソフトウェア工学における中心的関心事であり，それは，ある特定の論点・用途・ゴールないし概念から見て，きわめて「適当」と判断されるソフトウェアの構成要素を，どのように識別し，カプセル化し，取り扱うかといった一連の行為を指します.

適切な「関心事の分離」がなされることにより，対象を簡潔に把握することが可能となり，それはまた作られるソフトウェアに，柔軟で，可読性が高く，理解しやすいといった特性と，そして，何よりも求められたサービスを過たず提供することを可能にしました.

たとえば，オブジェクト指向がもたらした拡張されたカプセル化の概念は，それまでの手続きプログラミングにおいて可能であった「関心事の分離」の程度をはるかにこえたより高次元の「関心事の分離」を，実践可能としてくれています．OO によってもたらされた効果は，今日，従来より数段複雑なシステムの開発を可能にしました．そのことは OO 方法論や OOP が導入される以前と比較すれば，容易に合点できるでしょう.

しかしながら，ソフトウェア システムはその複雑度をいまだに無く増大させ続けており，今日，われわれは，かつて経験したことのない未曾有の「複雑さとの対峙」余儀なくされています．そこでは，これまでの無邪気 (naive) な「関心事の分離」はもはや十分とはいいがたく，いたるところで「破綻」を露呈しつつあるのが実情だといえます．そのこと当然の帰結として，保守性は不足し，コードは錯綜し，変更は予期せぬ伝播をもたらすなど，貧弱な「関心事の分離」に起因する諸問題に，今日のソフトウェア システムは以前としてあえいでいます.

これまでの「関心事の分離」は，ただ，課題をある一個の側面 (a concern) にすべて集約して捉え，それに対して解決策を提供するという構造をもっており，典型的には，「分割と合成」を通じて，木構造のそれへとモデル化するという単純なものでした．そのような単純さに立脚して提供されてきたこれまでの方法論やツール，そしてプログラミング言語自体が，すでに今日のソフトウェア システムにあらわれているさまざまな破綻の発端となってしまっているのです [Kiczales-et-al-97].

今日のソフトウェア システムは，もはやたった 1 つの側面に集約してそれを把握することは能わず，多重で相互に依存関係を有した関心事 (concerns) にもとづいてモデル化され，そして実現される [Ossher-and-Tarr-2000] のでなければなりません．考慮すべき関心事はさまざまであり，そしてそれらはまたさまざまな

視点や切り口 (aspects or crosscutting concerns) からそれぞれに把握可能であり，そしてさらにそれらは相互に密接に関係しあい，影響を与えあっています．したがってそれは強く「文脈依存 (context sensitive)」しており，それ故に，一度把握し，モデル化すれば完了する質のものでもなく，新たな視点や関心事があらわれるにつれ，変化し続けることになります．

このセッションでは，上記のような問題意識にもとづいて，よりよいソフトウェア システムの開発を可能とするために，どのような側面から，どのようにアプローチするべきか/できるのかについて，たとえば以下に示すような観点 (それに固執するものではないが) から広範な議論を行いたいと考えています．特に現場で，そうした方向を目指して実践している方々の積極的な参加をお待ちします．

A. Technological Support for ASoC/AoSD
B. Design Support for ASoC/AoSD
C. Verification and Consistency Management
D. Formalization of ASoC/AoSD
E. Software Development Process
F. Visions and Directions
G. Experience Reports

参考文献

[Parnas72] D. L. Parnas. On the Criteria to be used in Decomposing Systems into Modules. In Communications of the ACM. Vol. 15. No. 12. December 1972, pp. 1053-1058.

[Dijkstra76] E.W. Dijkstra, A Discipline of Programming. Prentice Hall, Englewood Cliffs, NJ, 1976.

[Kiczales-et-al-97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-Oriented Programming. In proceedings of ECOOP '97, Springer-Verlag LNCS 1241. June 1997. http://aosd.net/

[Ossher-and-Tarr-2000] H. Ossher and P. Tarr, Multi-Dimensional Separation of Concerns and The Hyperspace Approach. In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2000. http://www.research.ibm.com/hyperspace/

オープンソース セッションについて

林 香 (SRA)

ソフトウェアシンポジウムでオープンソースをテーマにするのは今回で4回目となります．

第1回目は1999年盛岡でした．Linuxに代表されるオープンソース ソフトウェアは今後普及していくのだろうか，ビジネスになるのだろうかという点が議論になりました．

SambaやPostgreSQLなどのオープンソース ソフトウェアの開発，普及に実際に携わっている方からの状況報告や開発管理の方法などの発表があり，オープンソースが社会に定着してきた感がありました．

昨年の高知では，「オープンソース ソフトウェアの普及・発展戦略」と題したパネル討論を行い，PostgreSQL,HORB,Apache,GNU,Rubyユーザ会などのコミュニティに関わっている方々からの発表，討論を行いました．今やオープンソースもホットな話題の1つになったようで，オープンソース ソフトウェアの社会的意義などで討論が盛り上がり，セッション時間が足りなかったとのコメントをいただくほどでした．

さて，4回目となる今回は，オープンソース ソフトウェアの内面に入りこんだ議論をしたいと思います．オープンソース ソフトウェアの利点として，高生産性，高品質がしばしば強調されていますが，実際のデータ分析に基づいた議論はあまり無い様におもわれます．ソース コードが公開され，バージョン アップ情報，バグ情報もオープンですので，データ分析の専門家に解析結果論文の投稿を期待します．また，高生産性をもたらしている開発方法としてのオープンソースに関する研究論文にも期待しています．

論文セッションだけではなく，オープンソースをテーマにしたパネル，チュートリアルも企画していきたいと考えています．今のところ，担当委員のあいだではノー アイデアです．チュートリアル，パネルのアイデア，ご希望などあればお知らせください．

これからのシステム開発 セッション および オブジェクト指向 セッションについて

熊谷 章 (PFU)

第1回 PC ミーティングにおいて，「これからのシステム開発」と「オブジェクト指向」の両セッションについて，担当委員の間で行われた議論の内容を報告します．

まず，SS2001 in 高知における「セッション7 これからのシステム開発」の発表論文がどんなものだったかを調べました．そして，SSは，技術者のシンポジウムなので，もの作りは常に重要なテーマかつ身近な問題であり，このセッションをさらに続けようという認識を確認しました．

次に，このテーマに関するそれぞれの考えを発表しました：

熊谷：最近，ソフトウェア エンジニアリングとシステム エンジニアリングの融合や連携の重要性が叫ばれ，具体的なアプローチが見受けられる．たとえば，CMMI，Life Cycle Engineering，システムLSIやSystem On Chips の Co-Design，Feature Oriented Produkt-Line Engineering などがあげられる．

また，WebベースのEngineeringもこの範疇であると考えられる．システムLSIの分野でUMLをソフトウェア開発のキー技術の一つとして現在抱えている問題解決の一つにしようとしているのも正にこの動きだろう．ここでの特徴は，開発するシステム全体を対象範囲とし，ソフトウェアはシステム全体の部分になり，他の部分であるハードウェアとファームウェアとの関係が重要になるということである．このようなシステム開発がこれからますます必要になるだろう．

筏井：Martin Fowlerの“The New Methodology”の視点を興味深く思っている．従来の重厚で堅固な開発方法は，要求や様々な環境の変化に追随できずうまくいかなかった．これに対して，Light Weightな開発方法が注目を集め効果を出している．具体的には，XP，Crystal Family，Peter Cordの Feature Driven Developmentなどの新しい開発方法論が生まれている．この類は，装備と考え方が軽く，スパイラル開発やIterative (繰り返し) 開発を取り入れ，常に変化する要求や環境にうまく対応できる仕組みになっている．つまり，顧客からみればこのようなアプローチは，リスク回避になっている．たとえば，XPは，繰り返しの頻度や規模が問題になるが，２～３日から週間の間で頻繁に繰り返されているのが実態のようだ．開発グループは２名から１２名くらいで，対象システムの規模も決して大きくないのが特徴である．また，これらは，管理的な色彩の強い従来process-oroented開発ではなく，関係者の協調や想像性を開発の主体にしたpoeple-oriented 開発である．

もちろん，こういった方法論は，開発対象や開発環境に強く依存するはずで，safety-criticalなものや，工学的な品質を追求するもの，大規模システムといったものは，これには向いていないだろう．しかし，時代とともに，捕らえようとする世界の性質，複雑さや変化の速度も当然変わってくるはずであり，それに合わせた方法論が求められていることも確かな気がする．

SS2002では，こういった時代の変化にともなう，方法論の多様化について議論してみても面白いのではないかと思う．

蔵川：設計支援をどのようにしてやるかをテーマにしたい．人間の思考プロセスに沿ったCHI（Computer Human Interaction）を対象とするのだ．人間の思考，視点，機能，実体などを考えそこから人間の活動を効率的に支援したい．従って，認知科学や脳コンピュータなどがHCIの人々の研究テーマになっている．例えば，外部からあるものを作成する依頼が来る．設計者は，その要求からある意図をもって設計するが，その意図は必ずしも形式化されていない．それを，形（機械），役割，概念，詳細化，実体などを用いて形にし設計のガイドラインとしたい．こんなことを話題にしたい．

中谷：オブジェクト指向というテーマでも，地域により興味ある内容が変化する．たとえば，東京近辺では，下記のような政治的テーマに人気がある．

大企業の考え方と動向
アメリカの技術動向

一方，東京以外の地域では，東京近辺の人たちが何をどのように考え，どのようなことをしているかを知りたい．また，次のテーマに興味があるように思える．

オブジェクト指向入門（チュートリアル）
UML入門

オブジェクト指向の議論セッションでは，オブジェクト指向と要求工学を取り上げて見たい．OO-Based UMLなんかよいテーマだろう．ここでは，従来なかった要求の定量化など考えてみたい．例えば，UMLで記述されたものの中から未定義なUse Caseの数，未定義な辞書の数などを数え上げ，要求の品質向上に結び付けたい．

SS2002の他のセッションは，「モノ作り」の話であるが，このセッションは「モノ作りの前」を話題にしたい．要約すれば，要求工学は，「ビジネスモデル」であり，これは世界をモデル化した「オブジェクトの世界」の入り口である．このビジネスモデルとオブジェクトの世界を要求定義という行為で変換可能にし，かつ表現するものがUMLである．ビジネスモデルは，下記のものを包含していることが条件である．

リソース（人，金，モノ，時間）
組織
仕事のプロセス
ゴール

東田：オブジェクト指向はあらゆるところに浸透し実用化されており，話題としてはあまり面白くない．しかし，オブジェクト指向プログラミング言語としてであれば，いろいろな議論がある．オブジェクト指向方法論になると，UMLにとんでしまい，面白くない．今回，Aspect Orientedというセッションが新設されたが，これとオブジェクト指向との関係を理解する必要がある．Aspect Orientedはオブジェクト指向から派生した技術の一つである．また，オブジェクト指向技術とフォーマルメソッドの関係を議論することもやってみたい．

コーディネータの現時点でのまとめ：

上述の議論をベースとして，今後議論を重ね，公募で集まってきた論文や報告の内容を見て，最終的なシナリオを作成する予定です．

これからのシステム開発

討論セッション：

(大きな物語 vs 小さな物語)
実践的システム工学とソフトウェア工学

コンピュータを用いた設計支援の実際

論文/報告セッション：

システムLSI開発とSOC開発
UML関係
XPなどのLight Weight 開発関係
Web-Based工学 とWeb-Based情報システム

オブジェクト指向

オブジェクト指向プログラミング
要求工学
ビジネスモデルとUML関係

皆さんのご協力をお願いいたします.

フォーマル メソッド セッションについて

張 漢明 (南山大学)

テーマは「開発現場における形式手法の適用に向けて」です.

来るべきソフトウェア開発における要素技術として，形式手法（フォーマル メソッド）に対する期待は大きいと考えられます.

SEAでも，２０００年３月からフォーマル メソッド分科会（sigfm）が発足し，月例会において，形式手法に関する活発な議論が繰り広げられています.

昨年のソフトウェア シンポジウムの形式手法のセッションや，組み込みのセッションでは，「実際のソフトウェア開発」において形式手法が本当に有効なのか，また，大規模なソフトウェア開発に形式手法が適用可能なのか，といった問いかけが投げかけられました．この時の議論を振り返ってみると，「実際のソフトウェア開発における問題」と「形式手法の適用における問題」が混ざりあって論点が曖昧になり，議論が噛みあっていなかったように思われます.

そこで，今年の形式手法のセッションでは，実際のソフトウェア開発において形式手法が果たす役割を明確にしたいと考えています.

このセッションでは，一般投稿で採択された論文とともに，実際に日本で適用された具体的な事例に焦点をあて，開発現場に形式手法を適用するためのノウハウを議論することにより，「実際のソフトウェア開発の問題」を浮き彫りにして，「形式手法がどのような解を与え得るか」を明示したいと思います.

形式手法は，理論から応用までをカバーする幅広い分野です．現在の形式手法の技術で，実際のソフトウェア開発の現場で何ができるのか，逆に何ができないのか，また，将来の形式手法の可能性と方向性ついても議論したいと考えています.

組み込み セッションについて

石川 雅彦 (SRA)

昨年と同様に，組込みソフト関係者以外の方から多くのご意見を頂戴し，活気のあるセッションとするために，

1) 組込みソフト開発を俯瞰できるチュートリアル
2) 一般からの論文発表
3) ディスカッション

を組合わせた形でのセッションを構成する予定です.

昨年のセッションは組込みソフトウェアを理解（認知といった方が正しいかもった方が正しいかも知れません）していただくことを主眼としましたが，今年は，さらに一歩踏み込んで，組込み開発とそれ以外の開発の違いがはっきりわかることを主眼に置いたセッションを展開しようと考えています．ミニチュートリアルおよびディスカッションのパネラーなどについてはこれから検討しますが，テーマの候補の１つとして，「危機的状況の組込みソフトの品質」を考えています.

昨年の組込み業界では，ソフトに起因する不具合で商品を回収する事態が頻発し，その結果，ブランド イメージの下落も含めて，大きな損失を出す企業がありました．昨年のシンポジウムではこの事態を受け，組込みソフトの品質は危機的な状況にあると訴えました.

さらに堀下げて考えると，組込み開発における品質確保は，設計工程ではなく，より下流のテスト工程において行なわれているのが現状と認識しています（しかも，そもそもテストという工程すらキチンと理解・実践・管理されていないのではないかとも思われます).

そこで，組込み開発のテスト工程と他の開発のテスト工程との違いを明確にし，品質の確保を進めるには何が有効なのか考えて見たいと思っています．組込み開発にとっては，今こそ，この問題に立ち向かわなければならない時なのかも知れません.

ただし，論文については，特にテーマを設けずに広く募集します.

多くの方に組込みソフトウェアの開発に関心を持っていただく場を構成すべく，プログラムの検討を進めています.

論文/報告/デモ 募集

ソフトウェア・シンポジウム 2002

(http://zone.iq.nanzan-u.ac.jp/ss2002/)

日程: 2002 年 7 月 17 日(水)～19 日(金)
於: 松江テルサ (松江市・ JR 松江駅前)
(http://www.web-sanin.co.jp/or/terrsa/)

主 催: ソフトウェア技術者協会 (SEA, http://www.iijnet.or.jp/sea/)
協 賛 (予定): 日本ソフトウェア科学会 情報処理学会 情報サービス産業協会
電子情報通信学会 ソフトウェアサイエンス研究専門委員会

今度で 22 回目を迎えるソフトウェア・シンポジウムは，さまざまな場で活躍している技術者や研究者，管理者などが一堂に会し，ソフトウェア技術に関する多面的な経験や成果を交流するユニークで貴重な場を，これまで提供し続けてきました．今回は，松江市を会場として，「変革期のソフトウェア技術」をテーマに開催の予定です．

ソフトウェアは，現代社会の基盤として不可欠になっています．その役割は，今後一層，重要になると考えられています．また，インターネットや組み込みシステムなど，新たなソフトウェアとその開発技術が研究・開発され，技術とビジネスの両面から急速に発展しています．しかし，一方で，ソフトウェアの信頼性，安全性，有用性，社会性が問われています．このようなソフトウェア技術の直面する多面的な課題を討議するために，本年のソフトウェア・シンポジウムでは，テーマトラックを設け，多方面からの議論を喚起するとともに，広く，技術者，研究者，利用者の研究・経験論文，事例報告を募集します．

また，ソフトウェアやツールなどの成果を目に見える形で発表して頂くために，ソフトウェアデモも募集致します．

応募論文/報告のテーマは，ソフトウェア関連技術全般にわたります．たとえば下記のテーマなど，ソフトウェアに関連する領域のテーマであれば何でも構いません．先進的な研究テーマはもちろん，開発現場での苦労話や努力をまとめた事例報告・経験論文も歓迎します．奮ってご応募ください．

- 開発技術：形式手法，オブジェクト指向，アスペクト指向，ツール・開発環境，保守・進化
- プロセス技術：プロセスと改善，品質管理，メトリクス，プロジェクト管理，人間的要因，教育
- 新しいソフトウェアとその開発技術・応用・社会的影響：
  * インターネット：アーキテクチャ，アプリケーションョン(e-ビジネスなど)，セキュリティなど
  * 組込みソフトウェア：リアルタイムソフトウェア技術，信頼性，プラットフォームなど
  * オープンソース：オープンソース開発，オープンソースシステムの応用
- これからのソフトウェアとソフトウェア技術

投稿論文はプログラム委員会で査読を行い，採否を決定致します．採録論文の中から最優秀論文賞を選定し，表彰します(賞金 20 万円)．

【シンポジウムスタッフ】

実行委員長: 田中一夫 (日本フィッツ)
プログラム委員長: 青山幹雄 (南山大)，阪井 誠 (SRA-KTL)
プログラム委員: 相田範夫(テクノプロジェクト)，荒木啓二郎(九大)，飯田周作(専修大)，
筏井美枝子(Free)，石川雅彦(SRA)，伊藤昌夫(ニルソフトウェア)，大西荘一(岡山理科大)，
岡村耕二(九大)，荻野隆彦(鉄道総研)，菊池 豊(高知工科大)，熊谷 章(PFU)，
蔵川 圭(奈良先端大)，佐原 伸(日本フィッツ)，菅原耕一(富士写真フィルム)，
鈴木郁子(シャープ)，鈴木裕信(Free)，高木徳生(オムロン)，高田広章(豊橋技科大)，
高橋光裕(電力中研)，田中 創(アイ・ティ・フロンティア)，張 漢明(南山大)，
中谷多哉子(エス・ラグーン)，中野秀男(大阪市立大)，奈良隆正(日立システムアンドサービス)，
野呂昌満(南山大)，林 香(SRA)，東田雅宏(PFU)，福山峻一(鳥取環境大)，藤井 諭 (松江高専)，
藤枝和宏 (北陸先端大)，藤野晃延(インアルカディア)，松下 誠(阪大)，松本健一(奈良先端大)，
まつもとゆきひろ(ネットワーク応用通信研究所)，村山優子(岩手県立大)，
谷津弘一(日本ユニシス)，和田喜久男(NEC アクセステクニカ)，和田典子(ソニー)
ローカルアレンジメント委員長：安達 進(テクノプロジェクト)
事務局：岸田孝一(SRA-KTL)

【ソフトウェア・シンポジウム 2002　応募要領】

応募論文/報告は未発表のものに限ります．他への重複投稿もご遠慮ください．
論文/報告は，下記要領で PDF あるいは MS－Word 形式，PowerPoint 形式で作成し，カバー・シートと合わせて，電子メールで ss2002-inq@iq.nanzan-u.ac.jp 宛てに，2002 年 3 月 15 日(金)(必着) までにお送りください．採否審査結果は, 5 月 10 日(金)までに応募者全員に電子メールで通知します．

<<研究論文，経験論文>>
論文の書式は自由ですが，A4 版 10 ページ以内としてください．
なお，論文の最初のページに題名，著者氏名，所属，連絡先 e-mail アドレス，要旨を明記して下さい．研究論文は，研究の新規性，有用性，信頼性を基準に審査します．経験論文は実践の観点から，有用性，信頼性，客観性を基準に評価します．

<<事例報告>>
論文執筆の必要はありません．発表スライド原稿で審査します.発表スライド原稿は A4 サイズ白黒で作成して下さい．OHP スライド 10～20 枚程度．発表要旨（A4 用紙で 1 ページ以内）を添付し題名，著者氏名，所属，連絡先 e-mail アドレス，要旨を明記して下さい．スライド原稿にも要点などを注記していただけば，査読者の理解が深まると思われます．
なお，実践の観点から，有用性，信頼性，客観性を基準に評価します．

<< ソフトウェアデモ>>
デモの概要を，論文と同様，書式は自由ですが，A4 版 4 ページ以内にまとめて下さい．
採録したデモは，概要を論文集に掲載します．デモは，ノート PC を持ち込みとし，
発表時間は 40 分を予定しています．

なお，英文の論文も受け付けますが，論文はできるだけ日本語で執筆して下さい．
その他不明な点は，下記の問合せ先まで電子メールでお問い合わせください．

主要スケジュール:　2002 年 3 月 15 日(金)　論文/報告/デモ応募締切
2002 年 5 月 10 日(金)　論文/報告/デモ採否通知
2002 年 6 月 10 日(月)　カメラレディ原稿締切

問合せ・論文等送付先：　E-mail: ss2002-inq@iq.nanzan-u.ac.jp

****

ソフトウェア・シンポジウム 2002
論文/報告/デモ 応募カバー・シート

氏名（筆頭著者）:　　（ふりがな）
所属（会社・大学・組織）:
部門（学部）:
役職：
住所：（〒　　）
TEL：　　（内線　　）
FAX:
E-mail：

論文タイトル（またはデモソフトウェア名）:

共同執筆者：（もしいれば）
氏名:　　所属:
氏名:　　所属:
氏名:　　所属:

応募カテゴリ：（いずれかにチェックしてください）
□：研究論文　□：経験論文　□: 事例報告　□：ソフトウェアデモ

内容分類キーワード：（最大 5 つのキーワード挙げてください。）
1：
2：
3：
4：
5：

CMM とソフトウェアプロセス改善

塩谷和範
(SRA 先端技術研究所)

1. はじめに

CMM[R] (能力成熟度モデル) は、米国 SEI (ソフトウェアエンジニアリング研究所) の Watts Humphrey が、IBM 社内でのソフトウェア開発実績 (OS360 など) を分析した結果、きちんとした開発プロセスが確立していたプロジェクトに成功例が多いことに着目し、いわゆる“ベストプラクティス”としてまとめた報告[3]を元に、SEI の Mark Paulk らがモデルとして完成させた。

この技術報告(CMM-SW: Capability Maturity Model for Software version 1.1)は、SEI の Web サイトに PDF 形式で公開されている。SEA-SPIN CMM 研究会(プロジェクト終了済)は、SEI との契約の下に、ほぼ 2 年をかけて、この翻訳をボランティアで行い、SEI 公式日本語版「ソフトウェア能力成熟度モデル 1.1 版」[4,5]として PDF 形式(2)で無償公開している。

ここでは、CMM を中心としたソフトウェアプロセス改善 (SPI: Software Process Improvement) の最新動向について解説する。(文献[1,2]参照)

2. CMM によるソフトウェアプロセス改善

CMM-SW(CMM for Software)は、Humprey および Paulk らの様々なソフトウェア開発プロジェクトの観察と分析に基づき、組織のソフトウェア開発プロセス改善のための実践的ノウハウを集大成した 1.0 版(1991 年)の公開後、主に米国のソフトウェアコミュニティーによるレビューと実践からのフィードバックを取り入れ、1.1 版として 1993 年 2 月に改訂公開された。(これまでは、CMM とは CMM-SW を指した。) CMM-SW の成功後、システムエンジニアリング分野をはじめ各分野の CMM が開発された。(文献[1]参照)

統合 CMM あるいは CMM 統合(3)(CMMI[SM]: CMM Integration)は、システムエンジニアリング成熟度モデル(EIA/IS 731)を含む、複数分野の CMM をプロセスという共通項でまとめ、用語と概念を統一し、かつ ISO-15504[6]に準拠するよう再構成する統合モデルである。2000 年 8 月に第 1 版が公開され、'02 年 1 月 12 日に、CMMI システムエンジニアリング/ソフトウェア/統合プロセス及び統合プロジェクト開発 (CMMI-SE/SW/IPPD) 1.1 版が公開された。

CMMI には、 段階(“staged”) と 連続(“continuous”) の 2 つの表現があるが、このうちの段階

[R] CMM, Capability Maturity Model, Capability Maturity Modeling are registered in the U.S. Patent and Trademark Office.

[SM] CMM Integration, CMMI, IDEAL, Personal Software Process, PSP, SCE, Team Software Process, and TSP are service marks of Carnegie Mellon University.

表現は、CMM-SW 1.1 版での 5 段階表現に対応している。これは、組織全体のソフトウェアプロセスを改善し、これを成熟させ、その組織のプロセスとして定着させるために、下位のレベルのプロセス改善が上位のレベルで着手すべき改善の基盤となるよう、5 段階に対象プロセス領域(CMM-SW では KPA と呼ぶ)を分け、段階的にプロセス改善を積み上げていくよう構成されている。(図 1 参照) そのため CMM によるアセスメント時には、下位レベルのプロセスが完全に実践されていなければ上位レベルの評価を行なわない。

具体的には、図 1 のように**成熟度レベル(ML:Maturity Level)**毎に、プロセス領域を ML1 から ML5 までの 5 段階に分け、

(ML1) きちんとしたソフトウェア開発プロセスが存在しない「**初期**」レベルから、

(ML2) プロセスが「**管理された**」(CMM-SW では「反復できる」) レベル、

(ML3) プロセスがきちんと「**定義された**」レベル、

(ML4) プロセスが「**定量的に管理された**」(CMM-SW「管理された」) レベルを経て、最終的に

(ML5) 常にプロセスを「**最適化する**」

成熟度レベルまで、段階を追って到達するように構成されている。

図 1 中の ML 毎の箱の中に、そのレベルで実践し組織のプロセスとして定着すべきプロセス領域略称を示す。また、表 1 に、プロセス略称に加えてプロセス名称を示す。

表 1 は、CMMI-SE/SW/IPPD 第 1.1 版に加えて、第 1.02 版草稿で示された SS:Supplier Sourcing と呼ばれる、外部調達についてのプロセス領域についても記している。略語に"*"をつけた斜体で示されたプロセス領域が SS (調達) で加わると想定されるプロセス領域である。このうち、レベル 2 の SAM (供給者契約管理) が調達においては SSM (供給者選定と監視) に吸収されると想定している。

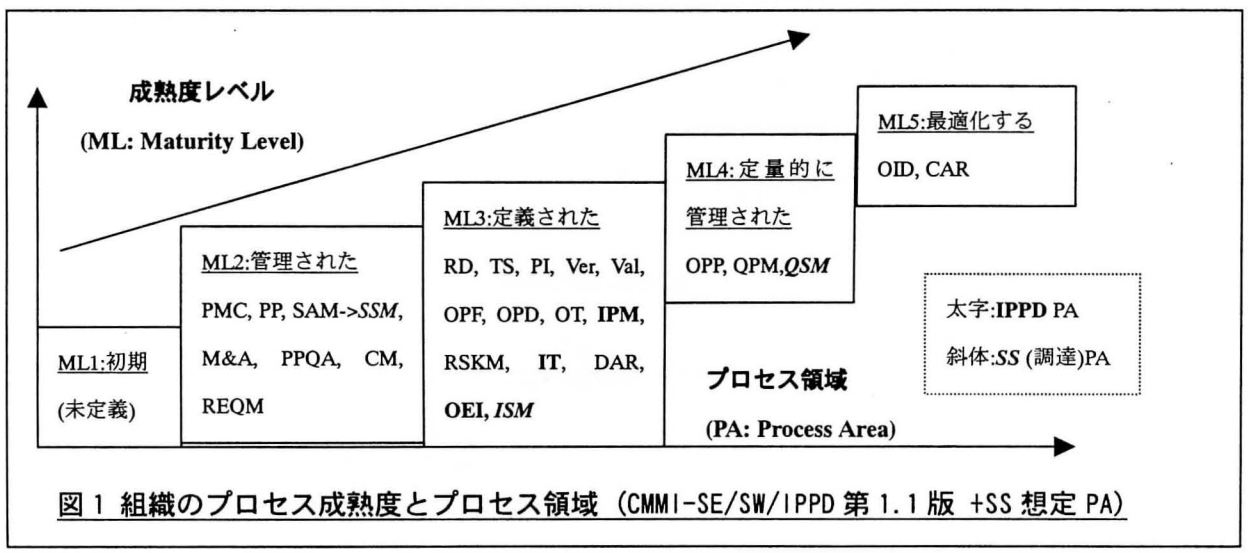


図 1 組織のプロセス成熟度とプロセス領域 (CMMI-SE/SW/IPPD 第 1.1 版 +SS 想定 PA)

表 1 成熟度とプロセス領域および能力レベルの関係 (CMMI-SE/SW/IPPD v1.1 + *SS*の PA*)

成熟度 (PA 数) : 状態		プロセス領域	目標プロファイル
ML-5 (2): Optimizing (最適化する)			**TP5**
	CAR	Causal Analysis and Resolution (原因分析と解決)	
	OID	Organizational Innovation and Deployment (組織改革と展開)	
ML-4 (2+1): Quantitatively Managed (定量的に管理された)			**TP4**
	*QSM**	*Quantitative Supplier Management* (定量的供給者管理)	
	OPP	Organizational Process Performance (組織プロセス実績)	
	QPM	Quantitative Project Management (定量的プロジェクト管理)	
ML-3 (13+1): Defined (定義された)			**TP3**
	RD	Requirements Development (要件開発)	
	TS	Technical Solution (技術的解決)	
	PI	Product Integration (製品統合)	
	Ver	Verification (検証)	
	Val	Validation (妥当性確認)	
	OPF	Organizational Process Focus (組織プロセス重視)	
	OPD	Organizational Process Definition (組織プロセス定義)	
	OT	Organizational Training (組織トレーニング)	
	IPM	Integrated Project Management for **IPPD** (統合プロジェクト管理)	
	*ISM**	*Integrated Supplier Management* (統合供給者管理)	
	RSKM	Risk Management (リスク管理)	
	IT	**Integrated Teaming** (統合チーム)	
	DAR	Decision Analysis and Resolution (決定分析と解決)	
	OEI	**Organizational Environment for Integration** (組織環境統合)	
ML-2 (7): Managed (管理された)			**TP2**
	REQM	Requirements Management (要件管理)	
	PP	Project Planning (プロジェクト計画)	
	PMC	Project Monitoring and Control (プロジェクトの監視と制御)	
	SAM	Supplier Agreement Management (供給者契約管理)	
	→ *SSM**	*Supplier Selection and Monitoring* (供給者選定と監視)	
	M&A	Measurement and Analysis (測定と分析)	
	PPQA	Process and Product Quality Assurance (プロセスと製品の品質保証)	
	CM	Configuration Management (構成管理)	
Level-1 (0): Initial (初期)		**"Staged"では未定義**	**TP:対応なし**

備考: **TP**: Target Profile, ***SS:Supplier Sourcing** (調達)の予定 PA **注: SEI 公式日本語版までの暫定訳**

一方、 CMMI 連続表現では、**ISO-TR-15504** にならい、各プロセス領域毎に 0(不完全)から 5(最適化する)の 6 段階で**能力レベル(CL: Capability Level)**の評価を行う。これにより、プロセス領域毎の細かいプロセス能力の把握ができ、また、組織のビジネス目標に応じた重点的かつ柔軟なプロセス領域をしぼった改善が実施できる。 加えて、段階表現での成熟度レベルに合わせてプロセス領域を集めた**目標プロフィール(TP:Target Profile)**によって、他の 15504 準拠のソフトウェアプロセスアセスメント結果との比較、及び CMMI の連続表現と段階表現の間の対応ができるように考慮されている。(表 1 参照 TP1-5)

目標プロフィール（**TP**）は、そのプロフィールに含まれるすべてのプロセス領域において、TP レベルと同じかそれ以上の能力レベル判定を得ることにより、対応する成熟度レベルに達したとみなす方式である。したがって、連続表現を採用している組織のアセスメントでも、目標 TP レベルに属するすべてのプロセス領域の能力レベルが、その TP のレベル以上であれば、段階表現における対応成熟度レベルと同等とみなすことになっている。

4. CMMI の適用

CMMI は、CMM-SW での主要プロセス領域(KPA) に、システムエンジニアリング能力モデル(EIA/IS-731)からのプロセス領域を加え、さらに、IPPD(統合製品プロセス開発)分野を加えて CMMI-SE/SW/IPPD 1.1 版を構成している。

2002 年春には、米国国防総省筋からの強い要請により、調達に関する 2 つのプロセス領域の追加が行なわれ、併せてレベル 2 の SAM (供給者契約管理) が SSM (供給者選定と監視) に改訂(図 1，表 1 中に斜体で示す。)された CMMI-SE/SW/IPPD/SS 1.1 が公開予定である。（この草稿は、2001 年 1 月の CMMI-SE/SW/IPPD/A 1.02draft で公開済み。）

このように、ソフトウェア開発プロセスの改善を目指して開発した CMM-SW から、他の CMM を統合し複雑になった CMMI ではあるが、開発元の米国 SEI では、これを CMM-SW に代えて広く普及させるために、CMM-SW の維持管理及び関連サービスを 2 年後の 2003 年末には打ちきることを表明している。また、CMM-SW の利用者はそれまでに CMMI に移行することを勧めている。

しかしながら、CMMI に関しては、お膝元の米国でさえも反対を表明している CMM-SW アセッサー及び利用者も多く、移行のコストも考えるとプロセス改善それ自体に関しては、CMMI を使う必要は必ずしもない。2001 年 10 月 1 日に開催した、JASPIC と SEA-SPIN との共催シンポジウムの招待講演者である Bill Curtis 氏（CMM-SW の著者の一人）も、ソフトウェア専業の場合は特に、CMM-SW 草稿 2.0C 適用を勧めていた(注：SEI の Web サイトから入手できる。)。

一方、組織のビジネスの観点からは、後述する外部評価（アセスメント）によるお墨付きを得るために、CMM レベル獲得を目指す動きが既に出てきている。

5. その他の SPA 手法と ISO-15504/9000-3 規格

CMM と対比される ISO-15504-TR2 (SPA: Software Process Assessment) は、CMMI, BootStrap、SPICE など各国、および各分野のソフトウェアプロセス(改善)アセスメント手法の標準的な枠組みとして提案され、発効に向けて ISO 規格化作業中である。したがって、CMMI 及び欧州などを中心に普及を図っている SPICE などは、ISO-15504 の実践モデルとして位置付けられる。ISO-15504 に基づく各種のプロセスアセスメント手法は、欧州を中心とする SPICE と同根であり、日本では、JISA（社団法人情報サービス産業協会）が、普及を図っている。TR-15504 は、2002 年ないし 2003 年には、大幅に改訂し ISO/IEC 15504 として発効の見込みである。

ISO-9000 規格の 2000 年版（JIS Q 9001:2000）の大幅改訂により、ソフトウェア業界ではその実施に対し異論があった、ISO-9000-3:1997 も、ソフトウェア及びシステムエンジニアリングの観点からの大幅改訂を実施し、新 ISO-9000-3 として発効すべく審議が開始されている。新 ISO-9000-3 においては、旧来の形式を重視した品質査定ではなく、システム/ソフトウェア開発の現状を踏まえた柔軟な適用が可能となっている。

6. 日本版 CMM について

2000 年夏頃から動きのあった政府の電子政府構想への布石として、経済省が「日本版 CMM」を政府調達に使用することにより、業界の競争を健全化(?)したいという構想を新聞等に発表した。この波紋は大きく、今までほとんど関心をもたれなかった人々からの CMM に関する問い合わせが頻繁にある。ほとんどがプロセス改善への関心から発したものではなく、政府の構想している調達への参加方法を知りたいがためのようである。この事態は、著者を含むソフトウェア技術者協会(SEA)SPIN 分科会などをはじめとする、地道にプロセス改善に取り組んでいこうとする人々を、認定取得に振り回された ISO9000 禍の再来か？(この表現は穏当ではないかもしれないが)と危惧させている。

本来であれば標準化を司る経済省は、国防省関係とは言え米国の一民間研究所の手法を取り上げるのではなく、ISO-15504(SPA)の日本版実現に本腰を入れなければいけない立場であるはずだが、政府自体が電子政府構想をぶち上げてしまった以上、早急に対応しなければならないのだろう。また、米国からの圧力をかわすためにも重宝であると考えるのだろう。これらの動きについては、ソフトウェア技術者協会の SPIN 分科会の Web サイト(2)にその経緯を含む詳細な報告が出ているが、JISA のプロセス分科会を含め多くの人と組織がその導入による弊害を憂

いている。

とはいえ、日本語版 CMM による調達先選定(足きり)が行われるようになると、やむをえないこととはいえ、企業としてはビジネスの観点からは対応せざるを得ない。この際、真のプロセス改善を目指すための方便として、しっかり便乗するのも手段の一つではある。

7. まとめ

ソフトウェア抜きの製品・システムは考えられない現在、製品品質向上の鍵となるソフトウェアプロセス改善実施の一助として、CMMI の完成と公開は歓迎できる。CMM-SW がソフトウェアコミュニティーと共に成長・発展してきたのに比べ、米国国防省および国防関連大企業主導の CMM 統合は、米国 CMM コミュニティーの大反発にあい、一時は完成が危ぶまれたが、両者の歩み寄りによりなんとか完成した。また、ISO-15504-TR2 の 2003 年の国際規格化(見通し)を控え、米国は欧州陣営(SPICE など)に対抗し、CMMI を ISO-15504 準拠の国家規格(ANSI)とするようである。日本では、「日本版 CMM」を経済省が採用表明しているが、その実体は CMMI そのもののようである。

CMMI の公開によって、従来から CMM への関心が高かったシステム分野、特に組込み製品分野への CMMI の適用が進むと思われる。また、ISO 準拠(誰が判断するかの問題はあるが)により、ISO か実質的なデファクト規格かで迷っていた組織も採用しやすくなった。

さらに、連続モデルの採用は、組織のビジネス目標に沿った重要プロセス領域の選定と、優先付けによる重点的な改善を可能にし、より柔軟なプロセス改善計画を立てることが可能となった。(とはいえ、従来の段階モデルのわかりやすさと長年の実績から、段階モデルあるいは、目標プロフィールに従うのが“お勧め”であることは変わらない。連続モデルの採用にあたっては、十分な検討と適切な実施計画を立てることが必須である。)

最後に、昨年 8 月 10 日にご逝去された坂本啓司氏の遺作となった「CMM」で陥りがちな“罠”を理解せよ」[7]を熟読されることをお勧めする。これまで、日本での知名度の低かった CMM を取り上げ、SEA-SPIN-CMM 研究会にて公式翻訳を出そうという話になったのは、坂本さんの意思であった。この公式翻訳を成し遂げ、更に CMMI の翻訳作業途中に故人となってしまった。これにより故坂本氏が担当していた、印刷版頒布が終了するとともに CMM 研究会は終了した。その後の CMMI 翻訳作業は、政府の日本版 CMM に協力しようとするコアチームが SEA の枠組みを離れ、JASPIC の直接の協力を得る形で翻訳活動を続けている。この成果もいずれ公開されるはずである。

8. おわりに

(1) ソフトウェア CMM (CMU/SEI-93-TR-24&25: CMM-SW: Capability Maturity Model for Software v1.1) は、SEI の Web サイトに PDF 形式で公開されている。SEA-SPIN/CMM グループは、SEI との契約の下に、この翻訳をボランティアで行い、SEI 公式日本語版「ソフトウェア能力成熟度モデル 1.1 版」[4,5]として PDF 形式で無償公開している。

(2) CMMI(関連 URL(3))も同様に、SEI にて PDF が公開されている。SW-CMM の印刷版原本は、SEI からのみ入手できる。CMMI についても SEI に問い合わせていただきたい。

(3) ソフトウェア技術者協会は、特定の手法を応援するものではない、CMM の翻訳活動は、SPIN 分科会 CMM 研究会がボランティアで実施した。(CMM 翻訳プロジェクトは終了)

(4) したがって、ソフトウェア技術者協会事務局は、CMM に関する問い合わせには答えられない。SEA-SPIN の Web ページ(関連 URL(1))を参照されたい。

参考文献

[1] 「CMM とは」, 塩谷和範, SRA 技術報 GSletter vol.28&30, 1999
http://www.sra.co.jp/public/doc/GSletter/vol.30/CMM/CMM.pdf

[2] 「日本における CMM 導入を考える」, 松原友夫・乗松聡, 共立出版, bit pp 2-8, 1998.3

[3] "Managing the Software Process", Watts S.Humphrey, Addison-Welsley, 1989
邦訳「 ソフトウェアプロセス成熟度の改善」藤野喜一監訳, 日科技連, 1991 年発行

[4] ソフトウェア能力成熟度モデル 1.1 版公式日本語版, SEA-SPIN/CMM グループ訳, SEA, 1999

[5] 能力成熟度モデルのキープラクティス 1.1 版 公式日本語版, SEA-SPIN/CMM グループ訳, SEA, 1999

[6] ISO/IEC 15504-TR2 Software Process Assessment (SPA) Part 1-9, 1998 (注: IS-15504 として改訂中)

[7] 「CMM」で陥りがちな“罠”を理解せよ」- 坂本啓司, 日経コンピュータ 7-30 号 pp 226-233

[8] ソフトウェアプロセス改善(SPI)の動向について - 塩谷和範, JPMF ジャーナル第 7 号(2000 年秋)、第 9 号(2001 年春) (この原稿の元になった連載記事)

関連 URL

(1) SEA-SPIN (SEA ソフトウェアプロセス分科会) http://www.iijnet.or.jp/sea/SPIN/index-j.html
SW-CMM1.1 版公式日本語版 PDF と 正誤表が公開されている.

(2) CMM 統合 URL: http://www.sei.cmu.edu/cmmi/

(3) 硬派!! ソフトウェアエンジニアとマネージャのためのホームページ
http://village.infoweb.ne.jp/~fwgf2942/process/Proc.Index.html

ほんとうのプロセス改善を目指して

- CMM の落し穴に注意-

坂本 啓司

これは，昨年の８月に急逝された坂本さんが，最後の入院の直前に，日経コンピュータ誌からの依頼を受けて執筆された論文のオリジナル原稿です．その意味で，坂本さんの絶筆になります．雑誌に掲載されたものには，若干編集の手が入っていますので，ここにあらためて原文をそのままの形で収録することにしました(編集部).

1. はじめに

ソフトウェアの開発はいたるところで問題を起こしている．それは，従来からあった問題が解決されていないにもかかわらず，システムのオープン化，ネットワーク化，さらには大規模化といった新しい問題が発生し続け，なかなかよい開発プロセスが確立できないところに，原因がある．そのような状況下では，ますます個人の頑張りや能力に依存する傾向が強いが，それでも，組織的なソフトウェアプロセス改善(SPI)の必要性を感じはじめた会社がふえてきてはいる．そのためか，ここ数年，日本でもCMMに関心をもつ企業が多くなってきた．さらに，昨年来，済産業省がいいだした「日本版CMM」も，こうした動きを加速させている．

そもそも，ソフトウェア開発プロセスは，人間系のプロセスを扱っているので，それを改善をするといっても，なかなか一筋縄ではいかない．何かのモデルにしたがってその通りに実行すればうまく行くといった単純なものではない．にもかかわらず，最近のCMMに対する関心を見ていると，CMMの高レベル評価を受けることにばかり意識が集中しており，これは本来のSPIをゆがめたものにしてしまう危険性があって，問題だと思う．

CMMは，SPIのモデルとしてはよくできており，参考にするべきところは多い．ただ，その活用には，よほどしっかりと基本的な考え方を理解した上で，自分たち独自の工夫がないと，期待した成果はあげられない．

以下に，いくつかの事例を挙げながら，CMMベースのSPIを進める際に注意すべきポイントを示すことにしよう．

2. SPIに取り組む上での注意点

CMMの公式文書でも，「CMMは銀の弾丸[*]ではないし，成功するプロジェクトにとって重要なすべての課題を取り上げているものでもない」と明記されているにもかかわらず，CMMに過度の期待を寄せておかしなことになっている事例をよく見かける．

[*] 銀の弾丸：ブルックスの「No Silver Bullet」という論文で有名になった言葉．銀の弾丸があれば狼男を一発でしとめることができる，というヨーロッパのいい伝えから来ている．ソフトウェアの問題をたちまちのうちに解決する万能の対策はない，という時によく使われる．

2.1. 事例A [技術課題の解決策は提示しない]

ある大手電機メーカーでの話である．どう見てもレベル1と思われる組織がSPIの開始を決心し，内部の対応工数を含め，2,000万円をかけてアセスメントを実施した．アセスメント報告書には，CMM各レベル達成のための重点項目KPA(Key Process Area)ごとに，その組織の強み，弱みが記述されていた．たとえば，「開発要件はドキュメント化され制御されている」とか，「見積りが文書化されていない」，「開発計画が文書化されていない」といったことである．しかし，これを見た部長は，「プロジェクト管理上のあれができていない，これができていないということは，すでに十分承知している．だから，どうしたらよいプロジェクト管理ができるかを知りたかったの

に，何の情報もない」といって，SPIはそれっきりになってしまった．

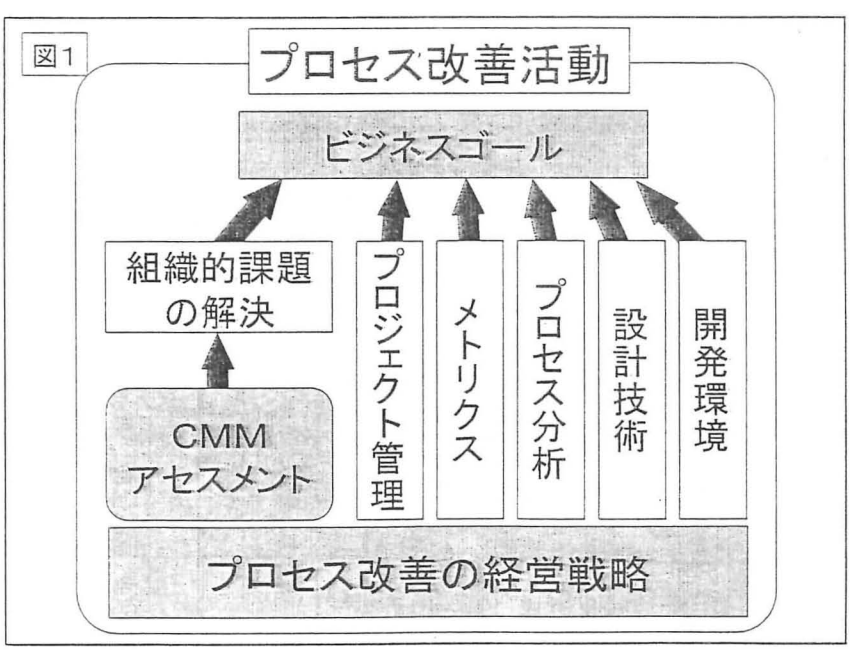
これは，CMMに対して過度の期待を寄せている典型的な事例である．この部長さんは２つの誤解をしている．１つは，SPIはCMMアセスメントからのみスタートすべきであると思っていること，もう１つは，アセスメント結果が課題解決のHowまで提供してくれると思っていることである．

CMMの考え方を最初に提唱したワッツ・ハンフリーは「地図があっても自分がどこにいるか分からなければ役に立たない」と述べている．地図がCMMで，自分がどこにいるのかを調べるのがアセスメントだ，というわけである．このハンフリーの言葉は，一般に強烈なインパクトを与えており，SPIはCMMアセスメントからしかスタートできないと信じている人が多い．このような人は，CMMが要求している事項を実行する能力が不足していることが明らかな場合でも，まずアセスメントを受けなければならないと思っている．

本来アセスメントとは，目に付くところのいろいろな問題を解決した後で，気がついていない問題の見落としがないか，解決した項目が組織全体に行き渡って歯止めのかかる状態になっているか，などをチェックするためのものである．よって，明確に実行能力が不足している自覚があるにもかかわらず，何の改善も行わずにアセスメントを受けても，自覚している通りの指摘があるだけで，何の進歩もなく，SPIを効率の悪いものにしてしまう．

２つ目の誤解も，多くの人が同じように受け取っていることである．CMMは，SPIを進めるにあたって，組織として歯止めをかけるべき項目の優先順位付けをしているだけであって，個別の技術課題に対してどのように取り組めばよいかについては，何もいっていない．そのことは，それぞれの組織が自分たちで考えて解決して行くべきものなのである．

図１に示すように，SPIを成功させるためには，SPIに対するきちんとした経営戦略のもとで，プロジェクト管理とか設計技術といった個々の実施能力を習得した上で，それらが組織として隈なく実施される仕組みができているか(つまり成熟度が高いかどうか)をチェックするために，CMMアセスメントを活用することが必要なのである．



2.2. 事例B [CMMレベルの盲信は危険]

「日本の会社はCMMレベルが低い．だから日本のソフトウェア開発力は低い」という短絡的な意見をよく聞く．しかし，これはほんとうだろうか？ 日本では，CMMアセスメントを受けた会社が少ないこともあり，アセスメント結果の最高はレベル3である．アセスメントを受けていない会社も，もし受ければ大半がレベル1であろうと思われる．しかし，だからといって日本のソフトウェア開発力は低いといってよいのだろうか．そのような考えに反証を与える事実がいくつかある．

「いつまでバグを買わされるのか」(マーク・ミナシ，ダイヤモンド社)に，ワッツ・ハンフリーの言葉として次のようなことが紹介されている：「インド・バンガロールのモトローラの研究所はレベル5の実力に達している．(中略) 最近出荷された2メガバイトのプログラム中，欠陥は総計24個しかなかった」．

あるとき，日本を代表する某メーカーのソフトウェア担当者の集まりでこの話を紹介したところ，思わず会場から嘲笑の声が上がった．かれらの感覚でいうと，このような欠陥の数は，とても商品として成り立たないレベルのものなのである．

インドの事例は，機器組込みソフトで，C言語100～200Kstepのプログラムであろうと思われるので，出荷後の欠陥率は0.12～0.24件/Kstepとなり，日本のメーカー系ソフトの感覚でいえば，かなり悪い品質だといえる．私の知っている限り，特殊な事情のある一品生産物の商品を除き，(特に量産品においては)，日本の大半のメーカーは，上述の例の2倍以上の高品質を実現している．

メーカー系だけではなく，日本の代表的なSIンダーの品質実績データを見せていただいたことがあるが，こちらでもインドの事例に比べてはるかによい品質を実現していた．

インドの事例は，数年前のものだと思われるが，CMMコミュニティで現在議論されているものも，それと大きく外れたものではないようで，1996年作成のカーネギーメロン大学・ソフトウェア工学研究所(CMU/SEI)のホームページには，図２に示すようなCMMレベルとフィールド品質の関係を示すグラフが載っていた．レベル5になってはじめて，フィールド不具合は全体の作り込み不具合数の 5%以下になるということであって，日本の感覚でいえば，CMMレベルが上がってもこの程度のものでしかないのか，とふしぎに感じられるような数字である．この状況は，現在もそれほど大きく変わっているとは思えない．

図2

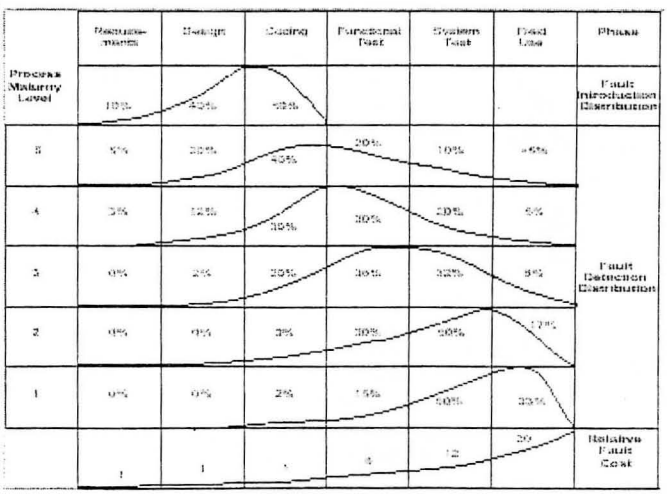


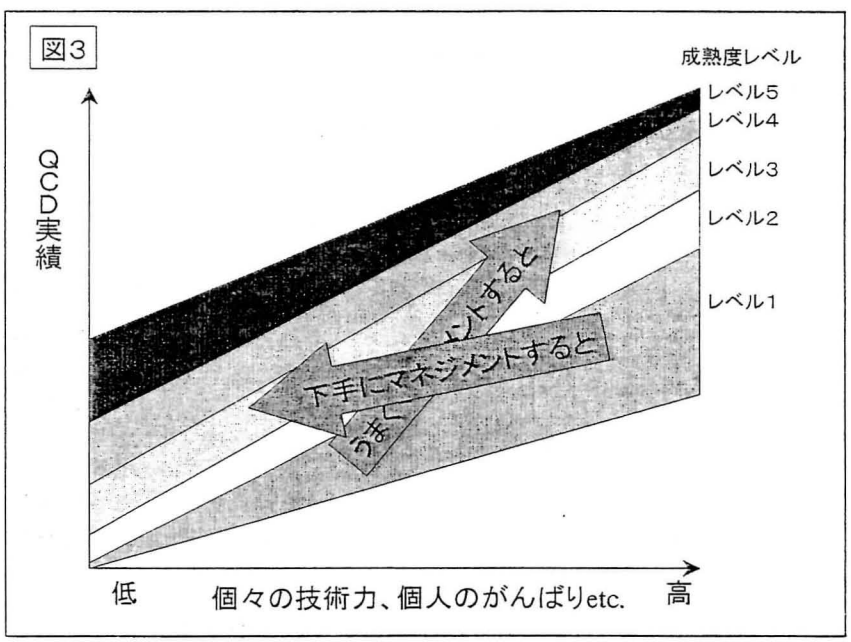
Figure 4-43: Shift in Fault Distributions as Process Maturity Increases

出典: **Handbook** CMU/SEI-96-HB-002 August 1996
Goal-Driven Software Measurement —A Guidebook

また，ファンクションポイントの研究者として有名なケイパース・ジョーンズが「ソフトウェア品質のガイドライン」(共立出版)という著書の中で，国別の品質状況を1996 年に調査した結果を発表している．これによれば，日本はダントツのトップで，第2位のカナダに2倍の差をつけている．ちなみにアメリカは第5位である．

これらのことが何を示しているかといえば，「CMMレベルの高い組織はレベルの低い組織より必ずよいQCD実績を出す」という誤解があることである．これは，あまりにもCMM を絶対視しすぎた一方的な見方だといえる．日本の多くの企業が高品質を実現しているのは，不具合分析の徹底等，独自の取り組みがなされていること，また，設計技術等の固有技術は決して他国に劣るものではないし，さらに設計者個人のがん張りによるところが大きいからだ，と思われる．

これらの要素とCMMレベルが，プロダクトのQCD実績にどのように影響を与えているかを，図3に示す．CMM以外の要素に変化がなければ，レベルが上がることでQCDの向上が期待できる．また，同じCMMレベルであってもその他の要素の状況によってはQCD 実績が逆転してしまうことを，この図は示している．



CMMにしたがってSPIを進める場合，現場の課題解決を中心に設計者の意欲を高めるように取り組むと，QCD実績を高めることができるが，しかし，レベル達成ばかりを目標にし，これまでの改善努力を無視した現場と遊離した取り組みを行うと，CMMレベルは上がったがQCD実績は下がってしまうという現象がおきてしまうのである．

2.3. 事例C [CMMが規定する順番に惑わされるな]

CMMアセスメントを軸にしてプロセス改善を進めている某メーカーで聞いた話である．何年間もの努力の結果，CMMレベルは着実に向上していった．ある時，このことを役員に報告に行くと，「努力をしてレベルが上がったことは分かった．それで，経営から見て何がよくなったのか？」と問われて，答えられるようなデータがなく，困ったということである．

CMMでは，データの収集・活用はレベル2から述べられているが，明確にその活用を謳っているのはレベル4である．しかし，これらのデータ活用は，プロジェクトをいかにコントロールするかに絞られたものであって，役員から質問のあったようなプロセス改善全体としての成果をデータで示す仕組みは，レベル5で初めて述べられる．

忠実にCMMにしたがってプロセス改善を進めてきたこの会社では，そのようなデータはまだまだ先のことと思い，取り組んでいなかったために，役員からの質問に答えられなかったのである．プロセス改善にはかなりの投資を必要とするため，その成果を何らかの指標で示さないと，経営としても継続した投資は難しくなる．レベル5で要求している粒度や精度に至らなくても，経営と共有化できる改善指標は初期の段階から取り組むべきものである．

CMM各レベル達成のための重点項目であるKPA(Key Process Area)は，各レベルにおいて，そのKPAだけを取り組めばよいものとして記述されているわけではなく，組織全体で次のレベルに向けたプロセス改善を効果的に行うためのベースとして，歯止めをかけておくべきものが記述されているのである．

たとえば，レベル3では，「組織の標準プロセスをプロジェクト毎にテーラリングし，プロジェクトのプロセスを定義して，それにもとづいてプロジェクト管理を行いなさい」と書いてある．これは，レベル2でいっている基本的なプロジェクト管理が実施できた上で，初めてできることであり，いくら標準プロセスが存在し，それをプロジェクト毎にテーラリングするガイドがそろっていても，プロジェクト管理をやる気のない，またやっていなくてもチェック機能の働かない組織に対しては，何の役にも立たないことになる．

このように，KPAは改善の歯止めをかける順番を示しているのであって，それは，必ずしも改善活動として取り組むべき順番を規定しているわけではない．どこから改善に取り組むかは，それぞれの組織の解決すべき課題に応じて，CMMのKPAの順番にとらわれることなく，たとえ部分的であろうが，取り組めるところは取り組むべきである．

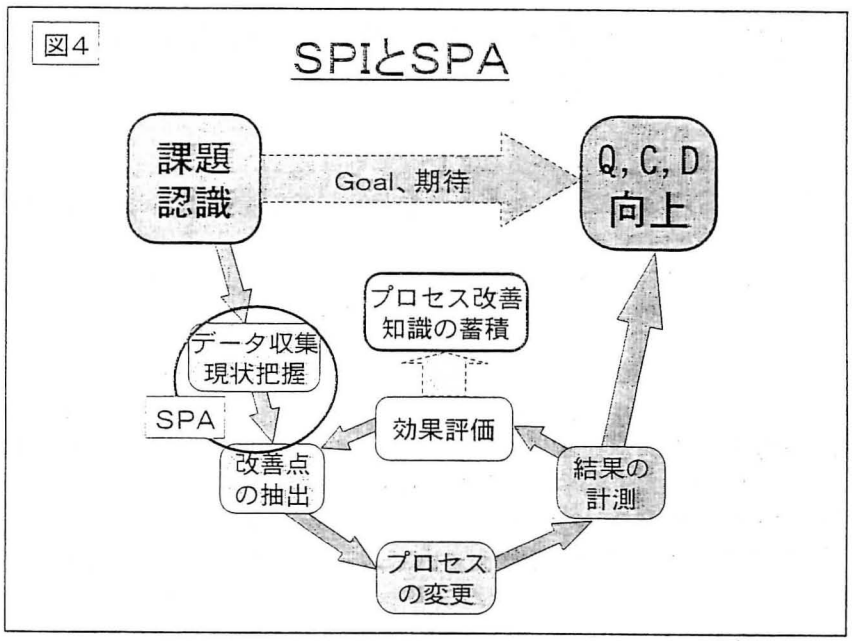
レベル2になっていなくても，ピアーレビューは，できるところからやって行くべきであるし，メトリクスは，どのレベルにあっても，それぞれのレベルに応じた精度と粒度で収集分析し活用して行くべきである．また，CMMではレベル5のKPAで定義されている欠陥予防は，品質向上のために必須の活動であり，レベル5の厳密さがなくとも，できるだけ早い段階から取り組むべきである．つまり，SPIを進める上で，それぞれの組織の課題認識に応じて，KPAの順番とは関係なく，改善を実行して行けばよいのである．ただし，その改善を組織全体の中で定着させようと思えば，CMMレベルのKPA順に歯止めが必要だということなのである．

CMMのドキュメントをよく読めば，そのようなことが書かれているのであるが，これを理解していない人が，レベルの順番にしか改善を取り組んではいけないと誤解し，個別の改善活動で有効なものも後回しにしてしまうことが多い．

日本の企業においては，組織全体で制度化するといった改善取り組みは少ないが，プロジェクト単位やグループ単位で，効果的な改善活動をいろいろと取り組んでおり，これが日本のソフトウェアのQCD実績を高める源になっていると思われる．このような組織が，従来の活動を尊重しつつ，CMMで組織的な歯止めをかけて行けば，さらによいQCD実績をあげることができる．しかし，レベルごとのKPAの順番にしか取り組んではいけないとCMMを誤解して，これまでの活動を排除してしまうと，CMMレベルは上がったがQCD実績は落ちたということになりかねないのである．

3. SPA(アセスメント)の偏重

SPAは，SPIを進める上のある局面では大変重要な取り組みであるが，その重要さを過大評価して，「SPAイコールSPI」と誤解してしまっている人が多い．SPIとは，図4に示す全体の活動のことであり，SPAはその中の現状把握と改善点抽出の一部の活動でしかない．また，アセスメント結果が発注者側の調達基準に使われたりすると，評価基準達成のためにだけ，また，会社のPRのためだけのレベル達成競争になってしまい，本来のSPIの目的が達成されたのか否かが，議論されなくなってしまう．



3.1. 事例D [レベル競争に終始する]

一部上場の某SIベンダーの役員と話していたときの話である．その役員は，長く開発現場で苦労をされた方で，プロセス改善の本質をよく理解されており，開発現場の課題解決指向でなければならないとか，表面的なCMMレベルの達成では意味がないとか，アセスメント偏重の世間の風潮への批判とか，議論が盛り上がり大いに意気投合した．しかし，最後にその方は，「非常に悩ましいところではあるが，やはりCMMのレベル達成はしておかなければならないでしょうね」といわれた．

レベル達成がQCD実績を保証する絶対的なものではないことを理解し，さらにレベル達成にかかるオーバーヘッドや期間を十分理解した上での言葉である．理由は，競合他社がCMMレベル達成を発表していったときに，自社だけが「プロセス改善の本質的な取り組みをしていますからCMMレベルには関心がありません」といい続けて，ビジネスとしてやって行けるかが心配だということであった．

日本でも，CMMにもとづくSPIを進める会社がふえてきて，レベル達成のニュースをマスコミが取り上げることがある．しかし，その取り上げられ方は，「日本で初めてレベルXを達成」とか，「国際的に通用するレベルXを達成」といったように，レベルの達成にばかり注目したものになっている．レベル達成によってQCD実績がどのようによくなったかを示す情報をセットで報道している事例には，ほとんどお目にかからない．

このようなマスコミの扱い方も問題であるが，SPIの目的がレベル達成ではなくQCDの向上であるにもかかわらず，目的の達成度合いを抜きにしてプロセスの評価結果だけが話題にのぼり，また，ほとんどの人がそこにしか関心を寄せないというのは，きわめて異常なことである．1995年前後，アメリカで先行してSPIに取り組んでいたいくつかの会社が，レベル3達成を公表したときには，レベル達成のニュースと同時に，次のような成果を発表していた．

ある会社は，4年間の活動でレベル3を達成し，市場品質20倍以上，生産性2倍，開発スピード2倍，顧客満足度55%向上を達成した．また，別の会社は，同じくレベル3を達成し，生産性35%アップ，開発期間25%削減，開発費15%ダウン，市場不具合35%削減，リリース前バグ20%削

減，テストカバレッジ45%アップを達成した．

このような話を聞けば，レベル達成によってどのようにQCD実績を向上させたかのノウハウを訊きたいと興味を持つが，単なるレベル達成の話だけだと，「それで，だれがハッピーになったの？」と，思わず質問したくなってしまう．

しかし，最近はアメリカでも，いかにしてレベルを達成するか，各社のレベル達成状況はどうかといったことに関心が集中する傾向がある．日本でもレベル達成ばかりが注目され，アセスメントクリアのための情報を求める人が多い．「アセスメントにはいくらかかるのか？ だれがやってくれるのか？ どのくらいの期間がかかるのか？」とか，「アセッサにOKをいってもらうためのテクニックを教えて欲しい」といったことばかりが取りざたされている．

企業の宣伝のためにレベル達成を目指すのは，それぞれの企業の事情から仕方がないことであるかもしれない．しかし，せっかくそのような活動をするのなら，SPI活動の本質をよく理解した上で，レベル達成がQCD実績に貢献するようにしなければならない．そうでなければ，レベル達成はユーザーにとって何のメリットもないものとなり，世間のCMMに対する評価も下がり，長い目で見れば，宣伝のための努力が水泡に帰してしまう．最近話題になった食品安全規格のHCCAPは，認証取得の努力が実績に繋がらなかったため，まさにこのような状況になってしまっている．

3.2. 事例E [入札条件を満たした途端に改善中止]

SPIに関する情報交換の場として，アメリカでは毎年 SEPGコンファレンスが開かれている．年々盛況になり今年ニューオリンズであった会議には，2,000人以上の参加者があったそうである．この会議で，珍しくSPIの失敗事例について発表があった． MITRE corp.という会社の "Success at SW CMM Level 3 but not at Level 4" という発表で，レベル3の達成はしたが，レベル4は経営層のコミットメントが得られず失敗したという話である．

経営層がほんとうにSPIに対してコミットメントをしているのであれば，レベル3達成でよしとする理由が，私にはどうにも解せなかった．そこで，少しその会社のことを調べて見ると，案の定，アメリカ国防省関係の仕事をしている会社であった．国防省は，調達基準としてCMMレベル3以上を要求しているので，とりあえずレベル3を達成しておけばビジネス上の問題はなくなるので「よし」としたのだろうと想像される．この会社にとって，CMMにもとづくSPIは，まったくのお荷物であり，「要求される最低限のことだけをしておけばよい」というのが経営陣の考えだったのであろう． SPIは，QCD実績の向上をもたらし，それが会社の競争力強化につながるということを，まったく理解できなかったと思われる． SPIを中途半端なものにしているという意味で，こうした調達基準には大きな弊害があるといえよう．

また，このような調達基準達成のSPIの場合には，個々の改善活動もレベル達成のための最低限のもので終わってしまうことが多い．開発現場の課題解決にどれだけ有効かは関係なく， CMMアセスメントクリアのためだけの標準やルールが作られて行く．それらは，場合によっては開発効率を落とす「余分な標準」となってしまう危険性がある．

このように，プロセスの評価だけをQCD実績の代用特性として調達基準にすると，多くの問題が発生する．ソフトウェア開発においては，よいQCD実績はよいプロセスによってのみ実現されるので，プロセスを重視することは絶対に不可欠である．しかし，だからといって，プロセス評価によってQCD実績の保証をすることは， CMMのようなよいモデルをもってしても，絶対に不可能である．その理由は，QCD実績に影響を与えるプロセス要素は多種多様であって，それらをすべて評価項目とすることが不可能だからである．

だとすれば，調達側は何にもとづいて調達基準とすればよいのか？ それは，あくまでもQCD 実績そのものである．これから発注するのだからQCD実績はまだ存在せず，評価はできない，というのは誤りである．その論でいうと，プロセスもまだ実行されていないので評価できないことにな

る．実際には，過去の類似プロジェクトのプロセスを評価するのであって，それならばQCD実績についても同様に評価できるはずである．直接的な目的であるQCD実績を評価せずに，間接的なプロセスのみを評価するのはおかしな話である．あくまでも実績で評価し，次も同様の実績があげられる可能性がどの程度あるかを予測するという補助的な目的のために，プロセス評価を用いるべきである．

4. ブームに乗った形でのプロセス改善

CMMが話題になっているからといって，十分な資源も投入せず，また十分な知識もないままに，形ばかりのプロセス改善を進めているところが多い．

4.1. 事例F [無意味なアンケートだけで済ませる]

SPI活動のことをCMM活動といっている会社がいくつかある．メーカー，ソフトハウスを問わずいくつかの会社でこの言葉を聞いた．CMMは，SPIのために使うツールの１つにすぎないというのに，CMMにのっとって活動するという表現は，世の中で話題になっているCMMという言葉を使うことで，いかにも熱心にプロセス改善に取り組んでいるかのように見せかけるポーズでしかない．このような会社は，ほとんどが十分な改善投資を行っておらず，少ない改善推進スタッフで可能なこととして，アンケートによるCMMレベル診断をしていることが多い．つまり，CMMのKP(Key Practice)を一覧表にして，開発部門の自己評価で「できている・できていない」をチェックして，レベル診断をしようというものである．

社員数300人くらいの中堅ソフトウェアハウスA社で次のような話を聞いた．半年に一回，各部門にアンケートを実施して結果の推移を見ているが，毎回着実によい結果が出て，いまではほとんどの部門がレベル3の状態であるという．それでも，改善推進担当は浮かぬ顔をして，「レベルは上がっているがなかなか開発現場の問題は解決しない」と，ぼやいていた．考えて見れば，これは当然の話である．アンケートに回答するだけで，改善自体は何も行われず，さらにアンケート結果の信憑性はだれもチェックをしないので，回答者は前回の回答よりも少しはよい回答をしなければと思い，「できている」という回答を着実にふやして行くのである．これだと表面的にレベルは上がっても何も改善がされないのは当然であろう．

実は，この会社でさらに話を訊いて行くと，驚くべき事実がわかってきた．

「解決されない現場での問題とは何ですか？」と訊いたところ，「開発計画の費用に対して実績コストのばらつきが大きいことです」という回答があった．「だとしたらアンケートをとるより，開発費の計画対実績の状況をつかんで，大きく外れたものに対して問題分析をすべきではないか？」とアドバイスをしたところ，「実はすでにデータは取っているのです」という．そこで見せてもらったグラフでは，ほとんどのプロジェクトが予算の95%くらいで完了していた．これで何が問題かと首をかしげていると，「実はこのデータはプロジェクト・リーダが操作して報告してきたもので，実態はとてもこのようなものではないのです」とのこと！！

だとしたら，真っ先に取り組むべきことは，いかにして信憑性のあるデータを集めるかであるのに，CMMという名前にとらわれすぎて，実質的に改善とは関係ない活動に数少ないリソースを浪費しているのである．

中堅の電機メーカーB社でも，早くから同様のアンケートを実施しているが，具体的な改善成果は聞いたことがない．大手精密機械メーカーのC社では，改善推進役のスタッフが開発部門に出向いて，CMMのKPをチェックリストとしてヒアリングを行っている．ヒアリング結果の報告書は，取り組みの不十分さばかりを指摘するもので，「そんなにあれもこれも改善しろといわれてもできっこない」と，開発現場から反発が出て，その後，改善がまったく進んでいないとのことである．また，中堅の電機メーカーD社でも同様にKPをチェックリストとしたアンケートでプロセス改善を進めているが，なかなか成果が出ないと，改善推進担当がぼやいていた．

「CMMは世界のベストプラクティスを集めたものだといわれているので，KPで記述されていることを実行するのがいいはずだ」といって，アンケートで実施状況をチェックしているが，KPの１つひとつが，それらを実行するには重たくて，現在の改善リソースではほとんど前進が見られないという．

KPAをチェックリストとしたアンケートは，だれでも思いつく安易な方法のようで，アメリカでも多く行なわれたようである．このような方法は，CMU/SEIでもCMMの最悪の使い方だといっている．その理由はいくつかあると思うが，１つはアンケート結果の信憑性の問題である． A社のように，意図的に実際と異なることを記入することもあるが，多くはKPの内容が理解できないまま適当に解答を書くことになるからだと思われる．また，別の大きな問題はC社やD社のように，KPの趣旨を誤解して使おうとすることである． CMM は，KPAの各ゴールが達成されてレベル達成となるが，ゴールで記述されていることだけでは，実際にどのような状態かが分かりにくいだろうということで，例示としてKPが記述されている．

KPで記述されていることが必須のベストプラクティスだというわけではないし，これが実施されていないからといっても，レベルが未達成だというわけでもない．それをすべて必須項目と思って取り組むと，改善努力の割に成果の少ない活動になってしまう． D社のように，権威主義に陥って，自分たちの抱えている課題解決との関係を見出せないまま，「CMMに書いてあるから取り組むべきだ」と思っている人が多い．このように，自分たちの課題解決に実感のわかないまま，世間で話題になっている取り組みを実行して行くと，きりがなくなり，ただでさえ少ない改善リソースを浪費するだけで，何の成果も挙げられないことになってしまう．

これらの事例は，中途半端な取り組みが何の成果ももたらさないだけではなく，開発現場の負荷をふやすだけで，やらなかったほうがましだったという改善活動になってしまうということを表している．

4.2. 事例G [かけ声だけのSPI]

ある大手電機メーカーのSPI担当者が，溜め息をつきながらこんな話をしていた．「経営層は理解があり，SPIが大切だとことあるごとにいって， SPIを支援してくれているが，なかなか思うように進まない」というのである．詳しく訊いてみると，SPI推進の専任者は全社で数名，その他は開発現場の人が，委員会形式で改善の話し合いをしているだけだというのである．

このような少ない改善投資では，成果がほとんど得られないのは当然で，一般的にSPI のROI(Return on Investment)は5～8倍だといわれている．しかるべき改善投資をしなければ，成果は得られないのは当然であるのに，十分な改善投資をしないこの会社の経営層は， SPIに対してほんとうに理解があるとは思えない．ソフトウェアの問題は，開発現場の問題であり，問題を起こしている当人が自分の努力で解決すべきだと思っていて，「SPIが大切」という言葉も，単に開発現場に対してもっと頑張れとハッパをかけているだけなのであろう．

また，十分な定義のないまま，「生産性2倍，開発期間半分，フィールド欠陥ゼロ」といった目標をトップダウンで設定するところが多い．前述のアメリカでの改善事例では，これらの指標は最初から明確に定義され，これらの指標を常に追いかけることで改善の進捗をチェックしている．顧客満足度といった一見抽象的と思えるものでも，厳密な評価方法が定義されている．しかし，日本で聞かれるこうした目標の大半は，単なるスローガンでしかなく，計測の仕組みも目標達成状況の確認方法も不明確なまま， SPIがスタートしている場合が多い．

このような，経営から発せられる「掛け声だけのSPI」は，だれも本気でそれが実現できるとは思っていないので，改善成果が最後まで追求されることはまずない．そのため，SPI 推進担当は，具体的な改善成果を求めるよりも「XXプロセス改善発表大会」といった派手なイベントに注力することが多く，少ない改善リソースをさらに少なくしてしまうことが多い．

4.3. 事例H [何でもSPIに期待]

ある中堅ソフトウェアハウスのSPI担当者が，社長にSPIの進め方について説明に行ったところ，社長から「収益率が上がるようにやってくれ」といわれ，何もいえずに引き下がってきたということである．

また，ある大手電機メーカーでは，商品の中に組み込まれる客先ごとのカスタマイズソフトの開発費が年々増大し，それを気にした経営トップから開発部門に，年間何百本も開発されるカスタマイズソフトの総開発費を「10%カットするように」という指示が出た．開発部門がSPIで頑張って10%のコストダウンを行うようにとの指示である．

これらの例は，CMMを中心としたSPIが世間で話題になっているため，何でもSPIに期待をして，経営数値そのものの改善もSPIだけで解決しようとする経営者がいることを示している．もちろん，SPIによって生産性が向上し，コストダウンによって収益率が上がることは期待できる．しかし，コストダウン分だけ営業が安く売ってしまったら，SPIによる努力は消えてしまう．収益率は，営業の注文の取り方からも含めたビジネス・プロセス全体の結果として出てくる．ソフトウェア・プロセスはそのビジネス・プロセスのサブセットでしかないのに，サブセットだけで全体の改善成果を上げることは不可能である．

１つ目の事例のソフトウェアハウスがその後どのような展開になったかは分からない．２つ目の事例のメーカーでは次のような展開となった．

開発部門でいろいろと問題分析をすると，ソフトウェア設計の効率以前の問題として，開発すべきカスタマイズソフトの数が年々増加し，さらにそれぞれの機能も増大していることが，年間総開発費を押し上げている原因であると分かった．さらに調べて行くと，営業部門の第一目標が売上高となっているために，収益性とか年間総開発費は二の次にして，いくらでも開発に要求を出してくる仕組みになっていることが分かった．そこで開発部門は，営業部門にトップの方針を達成するように協力を求め，収益性重視で受注案件をフィルタリングし，無条件での開発依頼が来ないようなプロセスを確立した．その結果，採算性の悪い案件は受注段階で振るい落とされ，カスタマイズソフトの数も減り，開発部門のSPIによるコストダウンとあわせて，無事10%カットが実現できたとのことである．

これは，10%カットの責任を負わされた開発部門がソフトウェア開発プロセスだけでなく，上位のビジネス・プロセスにまで立ち入って問題分析をしたことによって成功をおさめた事例であるが，本来であれば，トップダウンでビジネス・プロセスをソフトウェア開発プロセスとそれ以外のプロセスに分け，それぞれに対して改善の責任者と改善目標を明確にして取り組むのが筋であっただろうと思われる．

5. 真のプロセス改善に向けて

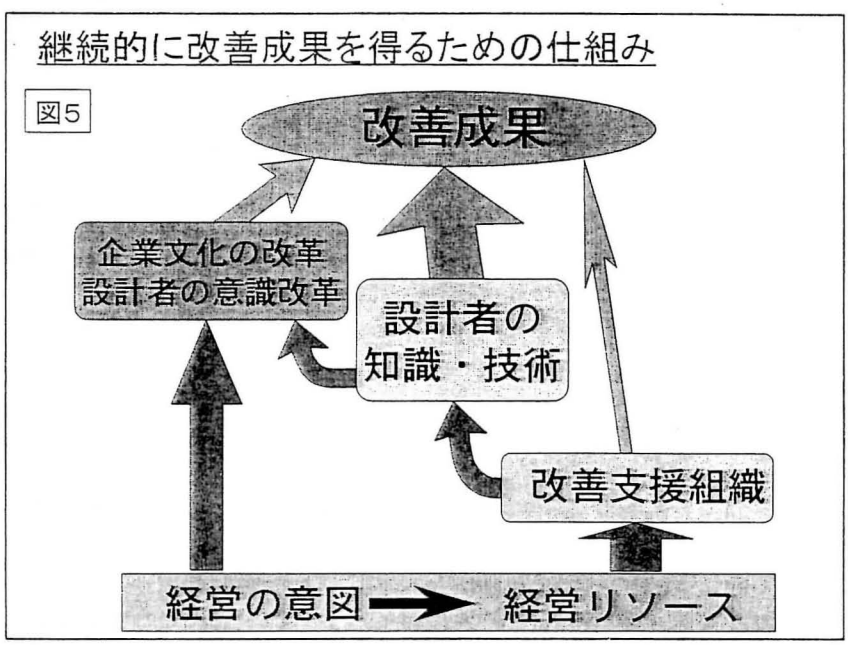
SPIを進めるためのプロセスは図４に示したが，継続的に改善成果を得るためには図５に示すような取り組みが必要である．

まず，経営課題としてのQCD向上を求めるのであるから，明確な経営としての意図を示さなければならない．改善目標をどのように設定するか，CMMのようなモデルをどのように使って改善を進めるかを明確にして改善のリソースを割り振るといった，組織としての改善戦略をしっかりと立てることが必要である．その戦略にしたがって設置された改善推進組織は，プロセスに関する専門知識を持つ人材，いわゆるSEPG(Software Engineering Process Group)として構成される．SEPGは，開発現場へのOJTの中で直接的に改善成果をあげることもあるが，それは付随的なもので，主たる役割は設計者に知識・技術の移転を行うことである．

この時に，定性的な説明だけではなく定量的な説明を行うことにより，移転の効率はきわめて向上する．また，改善の成果をアピールするためにも，SEPGはメトリクスをしっかりと活用しなければならない．これらに支えられて，設計者が効率的な設計を行うことで改善の成果が得られることにな

る．組織的な仕組み云々といっても，それは設計者が必要な技術力を持っていてはじめて役に立つことであって，個別技術力の向上は，なくてはならないものである．そして，最後に設計者へのモチベーションによって，高いモラルを維持することが改善成果の拡大に大いに関係してくる．

以下では，真のプロセス改善に向けた活動として，改善戦略，メトリクス，個別技術力の３点について少し詳しく述べる．



5.1. しっかりした改善戦略

まず，プロセス改善のスポンサーとなる経営層は，改善すべきQCDのゴールをできるだけ数値で明確にし，プロセス改善推進者およびソフトウェア開発担当者に対して，十分に説明することが必要である．目標に対して現状がどうなっているかを十分理解した上で，実現可能な目標を設定することが不可欠である．実現不可能な目標や計測方法未定義の数値だけの目標の設定は，絶対にやってはいけない．

次に，期待される改善成果に見合った人数の改善推進担当を任命する．これらのスタッフには，改善に対して意欲を持つ人間をあてるべきで，その時に手の空いている要員をあてるといった間に合わせの配員は，決して行ってはならない．

その次に行うべきことは，どのようなモデルをどう使ってSPIを進めるかの，具体的な方策を決定することである． SPIのモデルは，ISO9000をはじめとして，さまざまなものがあり，改善すべきゴールがどのようなものかにもよって適切なモデルを選ぶべきだが，一般的には，CMMを参考にするのがよいと思われる．

CMMの最大の功績は，それまではあってはならないものとしてソフトウェア工学の世界でも議論が避けられてきた「混沌として予測がつかない」レベル1の状態を定義し，そこから抜け出すためのガイドを示したところにあると考えられる．ソフトウェア開発を「混沌として予測がつかない」状態から「管理され予測可能な」状態にすることがSPIのゴールであるなら，CMMを参考にすることをおすすめする．

「管理され予測可能な」状態になって，さらに品質や生産性の向上を求めるのであれば，つまりCMMでいうレベル3以上の活動を行うのであれば， CMMだけではなく，プロセス・キャラクタライゼーションとか，GQM，さらに6σといった多くの改善モデルが提唱されている． CMMだけで進め

るのか，他のモデルで進めるのか，またはこれらの組み合わせでSPIを進めるのかは，それぞれの組織の置かれている状況と，ゴールの内容によって，個々に判断すべきものである．

いずれにせよ，SPI推進のモデルとしてCMMを参考にするのはよいことだと思うが，聞きかじりの知識ではなく，CMMの基本的な考え方はしっかりと理解して進めなければならない．CMMベースのプロセス改善を進めるのであれば，少なくとも「ソフトウェア能力成熟度1.1版」(いわゆるTR24といわれるドキュメント) は，経営層も含めてよく読んで理解をした上で進めていただきたい．TR24は正味50ページ程度の資料であって，Web からPDFファイルが無料で入手ができる．この資料をよく読めば，前章で紹介したような誤解にもとづく失敗は防げるはずである．

また，CMMの公式アセスメントを，いつどのように実施するかについても，それぞれの組織の実情を考えて，しっかりと作戦を練って欲しい．公式アセスメントは，次の改善のステップに進むための基盤が確立されたことを確認するために行うものである．それ以上の期待は失望を生み，多大な経費のロスとなる可能性が高い．プロセス改善をいろいろと進めた上で，基盤の確立を確認し歯止めをかけるここぞというタイミングで，公式アセスメントを実施すべきである．

いずれにせよ，これらの戦略が整合性をもって策定され，関係者間で十分な話し合いが行われた上で，あくまでもゴール指向で，決して手段が目的化することのないようにSPIを進めて行くべきである．

5.2. メトリクスの活用

「測定できないものはコントロールできない」というのが，トム・デマルコの有名な言葉だが，SPIをきちんとコントロールして成果をあげるためには，データの収集・分析というメトリクスの活用はきわめて重要なことである．

一般的にメトリクスは次のような目的に使われる．

- プロジェクトの現状把握とコントロール
- プロセスに関する知識の蓄積
- 改善ポイントの抽出と改善成果の確認
- 組織としての改善成果の確認

プロセス・プロダクトの状況を数値で表すことにより，だれでもソフトウェアの状況を把握しやすくなる．また，プロセスとプロダクトの因果関係を数値で説明することにより，プロセスに関する知識の移転が非常に効率的に行える．

CMMの中で扱われているメトリクスのほとんどは，「プロジェクトの現状把握とコントロール」を目的としたものであって，その他の目的はレベル4とか5で出てくるだけである．しかし，SPIを進めるには，これらの目的を持ったメトリクスの活用は欠かせないものであり，どのレベルにあろうが，SPIの初期の段階から取り組むべきである．また，CMMのようなモデルを使ってSPIを推進すると，アセスメント結果に異常な関心が寄せられ，プロセスばかりの議論に陥りやすい．SPIは，プロダクトの品質が向上して初めて成功といえるのである．プロダクトのできばえを何らかのメトリクスで表現し，評価し，アセスメント結果の妥当性をチェックすることによって，プロセスに偏った議論に陥るのを防ぐ必要がある．

経営層から与えられるSPIのゴールは，メトリクスを使った形で表現されるべきであり，改善成果の確認もメトリクスによるものでなければならない．また，経営から与えられるSPIメトリクスは，たとえば「市場品質2倍」といったものになるが，これは，そのままでは，個々の設計者は何に注力すればよいかが分からないので，「市場品質2倍」を達成するためのプロセス要素を，各設計者の作業レベルまで分解し，たとえば，「レビュー工数比率は20％を確保する」というように，具体的な改善目標をメトリクスで表現する必要がある．

SEPGは，このように，SPIの状況を経営層に説明する方法として，また，設計者にプロセスの知識を提供する方法として，メトリクスを十分に活用する必要がある．

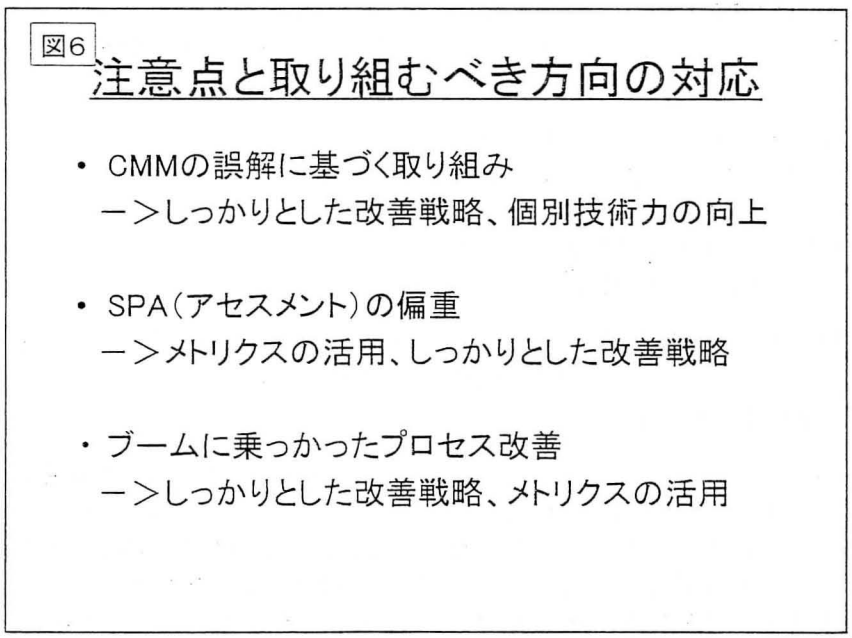
5.3. 個別技術力の向上

個別の技術力がいくらあっても，組織的な管理力がなければビッグプロジェクトはうまくコントロールできないという反省から，CMMのようなモデルが出てきた．もともとは技術力が主で管理力は従であったはずだが，CMMのブームの中で両者の位置づけが逆転し，ときには管理力一辺倒の議論が横行している．しかし，管理力というのはその組織の持っている技術力を100%出し切るための仕組みでしかなく，技術のないところに管理だけをもってきても，よい開発ができるわけがない．

個別技術の内容は，いわゆるソフトウェア工学の領域で議論されていることであり，要求分析，ソフトウェア設計技術，実装技術，検証技術といったものがある．これらの他に，管理自体を効果的かつ効率的に行うための技術もある．その代表がプロジェクト管理で，アメリカのPMI(Project Management Institute)が作成したPMBOK(Project Management Body Of Knowledge)は，世界中でプロジェクト管理の技術を共有するためのバイブルとなっている．

こうした技術力は，個々の技術者が習得するものであり，これらの技術者が高いモラルをもって，無駄な作業に手を取られることなく，効率的に作業を進めるためには，プロセス改善によってプロセスを整理することが必要になってくる．プロセス改善によってそのような環境を整えることは，まさに経営者の責任といえる．このようなことを経営者がしっかりと理解した上で，プロセス改善へのスポンサーシップを十分に発揮し，企業競争力を向上させていただきたいと願う．

図6



加算方式にしただけでは安値落札はなくならない

- 未成熟な見積もりプロセスこそ問題 -

松原 友夫
(松原コンサルティング)

1. 安値より価格の妥当性

「情報システムに係る政府調達の見直しについて」には，現行の総合評価落札方式が安値落札を招き，納入される情報システムの質の低下を招いている，との指摘がある．しかし，一般に，購入者の立場からは，安い方がいいに違いない．もし，安値を防止する手段が高値を誘導すれば，本末転倒で，その方が弊害が大きい．問題は，手抜きをもたらし新規参入を妨げる「極端な」安値である．それならば，問題解決の方向は，価格の「妥当な範囲」について，購買，供給の双方を含むコミュニティーが，知識を共有することでなければならない．

有形の製品の見積もりでは，昨今の事例のような，1億円のシステムを10万円という極端な安値で提出する，ということはほとんどあり得ない．有形物の見積もりは，基本的にコンポーネントの材料費と加工費の積み上げである．製品分野によって，例えば，家屋の場合に平米当たり何万円，またはポンプや発電システムのような大規模なものでも，トン当たりいくら，というように，物理的な規模が与えられれば，単位当たり単価の常識的コンセンサスが購買供給の両者にあって，およその価格が推定できる．したがって，見積もり金額に幅はあっても，せいぜい ±50%くらいである．しかし，重量も面積もないソフトウェアのような無形物の見積もりは，物理的な実体が存在しないので，共有すべき単価の常識が存在しないに等しい．従って，政策的な価格設定を抜きにしても，各社の見積もり価格の開きが一般に大きく，数倍のオーダーになることもしばしばである．

従来の製造指向の企業では，ソフトウェアの購買は，ハード部品を含めて購買部が扱うことが多いが，ハード購買で育った担当者が同じ流儀で安値に応じた名もない小さなソフトウェアハウスに，プラント制御ソフトウェアを発注し，稼働後にバグで爆発事故を起こした，という話を聞いたことがあるが，この種の話が他にもしばしば聞かれるのは，上述の違いを理解していないからだろう．

安値落札の問題は，評価方式を割算方式から加算方式へ変更するという方向で議論が進んでいる．これによって，多少は目立つケースが減るかもしれないが，新規参入を阻み手抜き開発を促す安値落札は，ソフトウェア固有の見積もり問題を解決しない限り，存続するであろう．ソフトウェアの安値落札を是正するための重要な前提は，応札者が科学的な蓄積データに基づいた見積もりプロセスを行うことと，購買側がその妥当性を評価する能力を持つことであり，さらに，こうした見積もり慣行によって，両者が共通の単位開発量に対して妥当な単価に対するコンセンサスを共有することである．報告書にはこの観点が欠落している．

2. ソフトウェアの価値は機能にある

ソフトウェアには重さも大きさもない．その価値を示すものは，提供される機能であるから，機能が測定でき，機能をベースとして見積もった金額でソフトウェアが売買できればそれに越したことはない．しかし，無形の機能を直接測るのが困難なことから，この産業は，長い間，それを開発する人の消費時間を売買の対象にしてきた．その慣行は未だに根強いが，ソフトウェアエンジニアリングの提唱を機に，科学的な開発が叫ばれ，開発量を示す尺度としてlines of code つまりコード行数を機能の代用として用いるようになった．その後次第にコード行数の欠点が数多く指摘されるようになり，80年代になって，function point つまり機能ポイントが，より直接的に機能量に比例するため，それが規模尺度として用いられるようになった．欧米ではこれが見積もりの主流になりつつあるが，日本では，製造業が埋め込みソフトウェアの分野で自らの計画に取り入れるところが増えているにしても，開発量の大部分を占めるといわれるビジネス分野や政府関連システムでは，機能ポイントはまだほとんど使われていないのが実情である．

3. ソフトウェアの見積もりプロセス

ソフトウェアの見積もりは，いくつかのステップに分けられる．最初のステップでは，顧客から提示された概略仕様に基づき，提供すべきソフトウェアの規模を見積る．規模尺度は，コード行数または機能ポイントが使われるが，日本ではほとんどの組織がステップを用いている．第２のステップで，規模から工数を算出する．ソフトウェア開発組織は，過去の開発経験からデータを集め，数多くの開発条件をパラメータにして分析し，単位当たりの工数基準を持っている．規模から工数への換算は，パラメータには，工数にリニヤーに効くもの，冪乗で効くもの，固定値などがある．第３のステップで，工数に企業別の工数単価を掛けて金額を見積もる．最後のステップで，これに管理費や利益を上乗せし，政策を加味して，御見積書として顧客に提出する．

一口に見積もり問題といっても，いろいろな視点がある．第１のステップ，つまり規模の見積もりは，長い間K2つまり経験と勘に頼っていたが，機能ポイントの採用によって，やや科学的な見積へと前進した．後述のCOCOMOのような見積もりモデルは第２のステップに焦点を当てている．第３のステップは，従来の原価計算での人件費の見積もりと同じで，工数単価は当該組織の直接費と間接費負担に依存し，通常企業規模の関数であ．政府調達では，小規模企業の安い単価の恩恵は，政府でなく元請けの大企業が享受してきた．情報サービス産業などの業界が，「大企業はけしからん」と言う場合の見積もり問題の焦点は，利益マージンの搾取に関しては第３ステップと，政策的安値受注に関しては第４ステップにある．

このステップはあらゆるソフトウェア見積もりで共通なのだが，国や地域によって，見積もりの科学的アプローチ，見積もり基準などの知識の共有，それに規模尺度としての機能ポイントの普及などについての状況が大きく異なる．

例えば，Barry Boehmは，彼がTRWに在籍していた頃，蓄積したデータを解析してCOCOMO: Constructive Cost Modelを開発したが，これを作った動機の一つは，米国国防総省 (DoD)との価格折衝に科学的アプローチを導入することにあった言う．つまり，Boehm は，科学的見積もりモデルという正攻法で，公開の場で見積もり問題を解決しようとしたのである．COCOMOは欧米で大方の支持を得てCOCOMOユーザー会が設立され，見積もりモデルの主流になった．データ集積と解析を主な業務とする組織（非営利組織が多い）が，いくつか設立され，データを公表し，あるいは見積もり用のツールを販売している．ユーザーとベンダーが，市販されている同じ見積もり用ソフトウェアを使うことによって，見積もり折衝はパラメータ選択の範囲で行うことになる．なお，業界からの強い要望で，Boehmは，クライアントサーバーシステムなど，最近の技術による開発にも適用できるCOCOMO IIを開発した．

4. 未成熟な我が国の見積もりプロセス

我が国の状況はどうであろうか．私が観察してきたところでは，ほとんどの企業は，公式の見積もりモデルを持っていない．多くの場合，見積もり用の基準データは，整理されずに個人の引き出しの中に入っているか，せいぜい小グループで共有しているに過ぎない．その結果，同じ引き合いに対する複数の組織からの見積もり値の幅は著しく，しばしばオーダーの違いがある．大企業の多くは独自の見積もりモデルを持っているが，その構造は，会社間，事業所間，親会社子会社間で異なる．

ユーザの状況はさらに悪く，ほとんどは見積もりモデルを持っていないから，見積もり値の妥当性についての合理的な判断ができない．このため，意に沿わない見積に対して，まったく合理性のない強引な値下げ要求（予算がこれしかない，など）をする．しばしばユーザーの強引な値引きへの怒りの声が聞かれるが，ベンダーも見積値のしっかりした根拠を示せないから，受注実績を積みたい一心で，無理な数値を提示する．それで経営が破綻しないのが不思議だが，その後のメンテナンスなどの作業で何とか取り返しているのだろう．一方で，甘いユーザーは，見積もりモデルを持たないので，ベンダーの政略的な安値見積もり，または高値見積もりを安易に受け入れてしまう．いずれの場合も健全な取り引き慣行とは言い難い．

CMMの開発者の一人，Bill Curtisは，昨年10月の日本での講演で，「ソフトウェアプロセス改善

は，不可能な見積もりに対してNo! と言うことから始まる」と述べたが，納得が行くデーターを伴ってNO!と言うことをしないで，無理を承知で受注している日本の現状では，プロセス改善は時期尚早かもしれない．

5. 機能ポイントの効用

ソフトウェア規模尺度として，コード行数が適切でないことは，長く問題とされてきた．それに代わる尺度がいくつか提案された．その中で生き残り，改良が加えられつつ次第に普及したのが機能ポイントである．この尺度は，コード行数の欠陥をある程度補い，また外部仕様がほぼ固まった段階で見積もれる利点があるし，ユーザーも理解しやすい．要求段階で見積もるために，当然誤差は大きいが，冒頭で引用した有形物の見積もりでの重量や面積のように，概算には十分役に立つ．前述のように，我が国でのこの尺度の普及は，最近始まったばかりであり，特に政府調達を含むビジネス系の受託開発では，ユーザー側が概して保守的で，従来から用いてきた尺度を使い続けるために，甚だしく普及が遅れている．

外部仕様から概略の規模を測れるという機能ポイントの特徴は，妥当な見積もりに至る一つの可能性を提供する．企業が個別に機能ポイントを適用する段階では十分な推進力になり得ないが，これを産業全体が利用するようになり，共通データベースにパラメトリックなデーターが蓄積されると，欧米ですでに実現しているように，見積もりツールの開発とビジネスが生まれる．つまり，公開される見積もり用データーや見積もりツールによって，見積もり値の妥当性を産業が共有できるようになる．

6. 我が国として推進すべきこと

ソフトウェアの政府調達の見直しで，今後の検討に加えて欲しいのは，第１に機能ポイントの普及である．前述したが，日本での普及の最大の障害は，ユーザーの購買者の保守性であるから，普及の重点はユーザーコミュニティーに置かねばならないだろう．これには，Acquisition-CMM のユーザーコミュニティーへの普及という方向も考えられる．第２は，開発実績データーの収集，蓄積，分析，見積に使える形に整理するパッケージングを目的とするデーターセンターの設置である．外国で開発された見積もりデータや見積もり用ツールを普及させるという案もあるだろう．実際に，COCOMO II の開発コンソーシアムに，日本の企業も参加している．だが，私が欧米と日本の開発現場を比較観察して，大きな違いに気づくのは，文書の作成プロセスである．コードと親和性のよい英語の文書作成プロセスは，日本語のそれとはかなり異なる．その部分は，日本でのデーターの蓄積と分析が必要である．

かつて，我が国でローカルコミュニティーでの実績データー蓄積が何度か試みられたが，すべて立ち消えになった．政府調達では，顧客としての政府の要求の一つとして，データーの提供をベンダーに求めることは容易であろう．データーセンターがうまく機能した暁には，我が国のソフトウェア開発の強みと弱味が浮き彫りになり，さらに具体的な対策に展開できるだろう．

かつて，日本の共通開発環境を作るためのシグマプロジェクトが実施されたが，その重要性を理解できない，またIT技術の将来の展望が読めない人たちが推進したために，ものの見事に失敗したが，もしこれができていたら，共通データー整備はずっと進んでいたに違いない．今さらながら，シグマの失敗で千載一遇のチャンスを失ったことを悔やみたい．　　(2002年 1月 17日)

追記：　この提言は，筆者がかつて従事した，機械工場での改善（9年間），コンピュータ工場でのシステムまとめ（3年間），ソフトウェア会社での見積もり査定権限を持つ部署（5年間），コンサルティングビジネス（10年間），などでの経験をもとにしてまとめた．

褒める

- プロセス改善効果を引き出すために -

端山 毅
(NTTデータ)

議論の最中，静かに聞いていた坂本さんが相槌を打ち始めるのは，何か言いたいことがあるときだった．「うん，うん，うん」と相槌の間隔が次第に短くなり，次に出てくる言葉は，多くの場合，「それで何がうれしいの？ 誰がうれしいの？」だった．常に現場の開発者達の立場から見て意味のあることかどうか，それが坂本さんの判断基準であり，開発者ひとりひとりに対する敬意が坂本さんの改善活動の基盤であったと思う．以下の小論は，1年以上前にSEA-Mail用にと思って書き始めたのだが，まとめられずに時間が過ぎ，とうとう坂本さんには読んでもらえなくなってしまった．内容的には故人にも賛同してもらえるだろうと思い掲載をお願いすることにした．

昔同僚が酒を飲みながらこんな話を持ち出したことがある．

> 人に感謝の意を表すのに，「すみません」と言うのはよくない．やはり「ありがとう」と言うべきだ．「すまない」というのは自分の非を認めているだけであって，相手を賞賛していない．

という趣旨だった．日頃あまり行儀のよい奴ではなかったが，妙に納得してしまった．人を褒めるというのは，結構難しい．管理者向けの書籍や研修などでも部下を褒めることの重要性は必ず取り上げられているが，褒める場合に比べると，注意する場合の方がはるかに多いのが日本の平均的管理者ではなかろうか．問題点を注意して改善チャンスを見つけてあげることが相手のためになるはずで，何も問題点を指摘できないと無能をさらけ出しているようにも思えて，ついつい何か注意点をコメントしてしまう．褒める量と注意する量の割合は，日米で大きく異なるように思う．どうも日本人は自虐的というか，けなし過ぎる．ソフトウェアプロセス改善(SPI)について，CMM(Capability Maturity Model: 能力成熟度モデル)を始め，多くの情報をアメリカから入手しているのが現状であるが，それらアメリカの情報を解釈する上では，褒め方とか，お世辞の具合を考慮する必要があると思う．

> 日本では，子供には9割叱って残りの1割を誉めるという感じではないでしょうか．＜中略＞ アメリカでは，私の目には，9割誉めて1割注意を与えるというように見えます．
>
> 松枝 太「ベンチャーマインド」より

松枝さんは，お子さんが参加したリトルリーグの試合の様子を例に褒め上手なアメリカ人を描写し，それが米国の旺盛なベンチャー精神の背景にある一要因と指摘している．

アメリカ社会が日本社会に比べて楽観的で前向き，悪く言えばいい加減であることには，論を待たないだろう．この文化的背景を抜きにして，CMMという文書だけを日本企業のコンテキストで解釈すると，誤解が生じかねない．

CMM/CMMIは，しばしばアセスメント法であると誤解されている．しかもアセスメント法とは，会社の格付け法だと思われたりもする．子供の頃から鍛えられてきた試験の記憶が甦り，嫌だ嫌だと言いながら，受験勉強のように短気集中，あるいは一夜漬けで突破してやると内心闘志を燃やしたりする．しかし，CMMは受験勉強が得意だった人達が書いたわけではないのだ．毎日"Very Good"，"Great"，"Excellent"を連発する人達が書いたのだ．

CMMは，ソフトウェアを開発する組織が，QCD(品質，価格，納期)の各面でその能力を向上させ，市場環境や技術の急速な変化に適応して，自らを効果的に変化させていける体制を具備するまでの道筋を示したロードマップである．このロードマップを使って，自組織がどのあたりにいるかを理解し，改善の方向，特に当面取り組むべき領域を示唆するという点で，診断法の基盤となりうる．モデルは，現実世界をある一面から抽象化したものであり，他の多くの側面を無視する．状況によって

はその無視された側面の方が重要になり，モデルが役に立たないことがある．したがって，モデルを利用するには，十分にそのモデルについて理解し，またモデル適用先の現実についても十分な知識を持つ専門家の判断(Professional Judgment)が必須である．素人判断はけがのもとだ．工学とはそういうものだ．電気工学の教科書を読んだだけで発電所を作ろうとはしない．建築基準法のセミナーを2,3日聞いたくらいで，建設会社の格付けなどできようはずがない．

アセスメントに目がいってしまうと，どうも減点法で物事を見てしまう．しかし，よいプラクティスを組織に定着させ，継続，改良していくには，減点法は適さない．むしろ，よいプラクティスを積極的に見つけ出し，賞賛して組織内に普及させる気運を高めることが重要である．一般に，短所を見つけるほうが長所を見つけるよりも易しい．成熟度レベルが上がったとか，プロセス記述やテーラリングマニュアルが完成したといったことを賞賛すべきではない．また，そういうことを改善活動のマイルストーンにしてはいけない．これらは目的と手段の混乱であり，第一線の開発者にとってご利益が実感できないからだ．開発者達が，これはうまくいったと思っている事象を，それは偉いと褒めてあげるべきだ．

売上や利益が増えたとか，システムが完成したとか，さらにお客様から高い評価を得たとか，最終的な結果が出ればどんな組織でも褒めてもらえるだろう．しかし，このような最終的な結果につながった良いプラクティスは埋もれてしまう場合が多い．えてして記憶に残るのは，プロジェクトが窮地に陥った際に，それを救った英雄的な活動や，披露困憊の後に味わった充実感であって，粛々と積み重ねられた合理的な活動や判断ではない．しかし，目立たない後者のようなプラクティスこそ，組織として褒め称え継承されるべきものだ．直接関わった開発者でさえ見過してしまいそうな合理的な活動，そもそもプロジェクトの混乱を未然に防いだ言わば当たり前のプラクティスこそ，強調され組織的に共有されるべき知識である．

褒める方が叱るよりも学習効果が高いことは，広く認められている．４分の３世紀も前のElizabeth Hurlockの実験では，賞賛されると学習効果が長期に渡って持続する一方，叱責は一時的に学習効果を高めはしても，その効果を持続できないことが示されている．賞賛されてこそ良い活動は強化され，定着していく．フィードバックのない活動は定着しない．直感的に考えても，叱責や罰則によって強制された行動は，その重しが取れた途端に消滅してしまうのが当然である．企業経営やソフトウェアプロセス改善においても，この点に大きな関心が払われるべきだ．

賞賛と叱責のバランスは，圧倒的に前者に傾斜すべきなのだ．しかし，それは甘やかしと解釈され易く，組織的には「アメとムチ」，「信賞必罰」というように，必ずセットで均衡させるべきもののように思われている．罰することなしに褒めてはいけないかのような雰囲気があり，むしろムチや罰則の方が先行的に強化されていないだろうか．賞賛を例外的事項にすると，強化されるべきベストプラクティスが看過され，強化しても将来の組織にとっては無益な個人の英雄的行為やカリスマ性がクローズアップされてしまう．これが成熟度の低い組織に往々にして観られる「失敗を褒め称える」という状態である．窮地に陥ったプロェクトの救済者を褒め称え，救済者の活躍の前に，窮地に陥った原因追及が軽視され，組織学習が進まない．「喉元すぎれば…」という状況を繰り返す．ここから抜け出すひとつの方策は，思い切って褒めまくることではなかろうか．

同様に，ソフトウェアプロセスアセスメントの誤用による格付けは，ベストプラクティスの強化法として効果的ではない．レベル獲得を目標とした活動では，レベル達成に不足する部分が注目され，どうしても減点法になってしまう．そして，罰則を恐れて強制されたプロセスは，短期間に消滅する運命にある．一方で，すでに実施されているベストプラクティスは看過されやすい．組織の現状把握という本来の目的でソフトウェアプロセスアセスメントを利用し，現状のプラクティスを賞賛して強化することこそ，ソフトウェアプロセス改善にとって必要な働きかけだ．改善活動を持続させるには，さり気ない行動を賞して"Good Job!"と声に出すことが大切だろう．だとすると，ソフトウェアプロセス改善を日本で促進するには，日本の職場で日常的に使える褒め言葉を用意することがカギになるかもしれない．試しにSEA-SPINの月例会では，参加者同士で褒めまくってみてはどうだろうか．

「事件」としてのCMM

岸田 孝一
(SRA-KTL)

1．はじめに

この小論のタイトルは，思想史学者・子安宣邦氏の著書「事件としての徂徠学」[1] から借りてきたものだが，ここ数年の CMM をめぐる世の中の動きは，まさしくひとつの「事件」としてとらえるのにふさわしい状況になってきたように感じられる．

アメリカでは，90年代の初めに小規模なワークショップ形式で始められた SEPG 会議が，数年前から参加者の規模 1,000 人を越す大コンファレンスに発展し，そのアメリカ市場相手のオフショア開発受託を狙ったインドでは，CMM レベル認証がマーケティング・ツールとしてみごとな成功を収め，同じようにローカルな SEPG 会議が盛況を呈している．

そうした動きを対岸の火事として眺めていた日本でも，一昨年の秋から経済産業省(METI) が，例の e-Japan 計画にともなう政府調達とのからみで「日本版 CMM」なる中身のないアドバルーンをぶち上げたせいか，あちこちで「事件性」に満ちあふれた空騒ぎが起こっている．

この「事件」の本質を考える場合，Watts Humphrey 氏がもともとどのような動機で CMM のコンセプトを提唱したのかといった分析はほとんど関係がない．子安さんがその著書の冒頭で紹介している哲学者 Ricoeur がいうように，「言説は，それがひとたびテキストとして書かれると，筆者の意図とテキストの意味とは合致することをやめる」からである．

「孔子の道は先王の道なり」という徂徠学の基本テーゼが，思想史的にみてきわめて重要な意義を持っていることはいうまでもない．野口武彦氏がていねいに解説されているように，それは日本思想史上における初めての「メタ言語」の発見であった[2]．見方によれば，現代哲学の主要な話題である「言語学的転回」(Linguistic Turn) をすでに18世紀に先取りしたものだといってもよいかも知れない．

この徂徠テーゼが，ソフトウェア・プロセスの問題を議論するさいにも有用であることは，わたしも，すでに数年前の SPIN ワークショップ in 静岡で指摘したし，その後の SPIN-ML 上での議論においても，何回かそのことに言及した．

しかし，子安さんが指摘する事件性とは，そうした哲学者・徂徠の本来の意図とは別次元の問題である．「孔子の道は先王の道なり」という徂徠の発言が，18世紀初めの日本の儒学者たちの世界で，その言説の意味することとは無関係に，きわめてスキャンダラスなかたちで受け取られ，その影響がそのまま今日まで尾をひいていることを指す．たとえば，明治政府によって始められた過去の偉人たちへの「死後叙勲制度」だが，いまだに，徂徠はその栄に浴していない．いいかえれば，現在の日本国は，この偉大な哲学者の業績を「国としては」まだ公けには認めていないのである．

2．事件の背景

さて，哲学的脱線はそのくらいにとどめておいて，ソフトウェア・プロセスに関する議論のこれまでの歴史を振り返り，「事件としての CMM」の背景を探ってみることにしょう．

ソフトウェア技術者および研究者の世界におけるプロセス論議の出発点は，1970年ロサンゼルスの Wescon 会議で Winston Royce が提唱したウォータフォール・モデルであった[3]．この提案は，そのわずか2年前の NATO 科学委員会の会議をきっかけに誕生したばかりのソフトウェア工学にとって，まさにきわめて重要な「事件」であって，以降の研究開発および実践の方向に大きな影響を与える結果をもたらした．

ウォータフォール・モデルの中身は，ソフトウェア開発プロセスの理想的なイメージを，ハード

ウェア製品の開発および製造プロセスをほぼそのまま引き写したものにすぎなかったのだが，歴史の浅いソフトウェア工学の視点からすれば，先駆者としてのハードウェア工学からの示唆は，無視することができなかったのである．

本質的にプロダクト指向で，形式的なドキュメントの流れを中心とするウォータフォール・モデルが，現実のソフトウェア開発あるいは保守のプロセスと合致しないことは，やがてすぐに明らかになった．だが，多くの人びとは，そのことを理想と現実の乖離としてとらえ，現実を少しでも理想に近づけるべく，結果的には空しい努力を積み重ねたのであった．

もちろん状況に対する反省もなかったわけではない．たとえば Barry Boehm が提唱したスパイラル・モデルなどは，ウォータフォールの非現実性を認識し，より現実に近い形でプロセスを捉えようとした試みの典型的な一例 (ただしまだウォータフォールの影響を抜けきれてはいなかったが)であろう．

そうした動きを，プロダクト指向からプロセス指向への「パラダイム・シフト」としてとらえたのは，Christiane Floyd が1987年に発表した論文であった[4]．かの女はスパイラル・モデルの考え方をさらに推し進めて，再設計－再実装－再評価というステップから成るサイクルの重要性を指摘し，そのサイクルを何回も回るごとに，開発者とユーザとの間の相互理解が次第に深まり，システムは新しいバージョンに進化してゆくという考えを示した．

Floyd 女史が指摘したこのパラダイム・シフトの萌芽は， 1975年に，当時 IBM Watson 研究所に在籍していた Les Belady および Manny Lehman の２人が提唱した Software Evolution Dynamics に見ることができる[5]．２人は，そのころすでに当初の開発から10年経っていた OS360 やその他いくつかの大規模ソフトウェア・システムのデータを分析して，それらに共通するソフトウェア進化の現象を法則の形で提示した：

(1) システムの規模は時間とともに増大する．
(2) それとともにシステムの構造は劣化する．
(3) システム保守の生産性は担当チームの規模 etc とは無関係にほぼ一定である．
etc

ここには，あきらかに，従来の科学工学的なものの見方から社会工学的な視点へという変化がみられる．一般の予想をまったく覆した第３法則について，２人は，ソフトウェア・システムという存在が，それを利用するユーザの業務プロセスやさらに周囲の社会の動的な変化の中に複雑な入れ子構造の形で組み込まれており，したがってその進化のプロセスはわれわれ人間のコントロールの範囲を越えたものであるという解釈を提示した．

その後，母国イギリスに帰って Imperial College の教授になった Lehman は，そうした社会学的分析をさらに進めて，Ｓ型およびＥ型というソフトウェアのカテゴリ分けを提案した．

Ｓ型とは，Specifiable すなわち，あらかじめ仕様が明確に規定できるようなソフトウェアを指す．たとえば，連立方程式の解法などの数学的計算やプログラミング言語のコンパイラといったものである．この種のソフトウェアは，ほぼウォータフォール・モデルにしたがって開発を進めることが可能である．

しかし，残念ながら，Ｓ型に属するソフトウェアの数はきわめて少なく，われわれが扱うソフトウェアの大半はもう１つのカテゴリであるＥ型に分類される．すなわち，現実社会のさまざまな仕事のプロセスの一部をコンピュータ化したものであり，できあがったシステムは現実のアプリケーションの中に組み込まれて（Embedd されて）動き出す．

そうしたＥ型ソフトウェアの開発プロセスは，ユーザへのインタビューによる要求分析から始まって，その結果にもとづく仕様の確定，さらに設計・実装・テストへというステップを踏んで進む．そして，できあがったシステムは現実のアプリケーションの中に組み込まれて動き出す．

しかし，もともとの要求仕様が，そのアプリケーションに関係するさまざまなユーザのリクエストを訊いて，それらの最大公約数的なモデルをまとめたものでしかないので，それぞれのユーザの要求とは少しずつ食い違っている．

また，アプリケーションは現実社会の一部であり，現実はつねに変化しているので，開発期間中にもともとの要求そのものが変わってしまうということも，しばしば起こりうる．

したがって，E型のソフトウェアは，それが完成して動きだした瞬間に，あらためてユーザの声を訊き，周囲の環境条件の変化も考慮した手直しが必要になる．すなわち，要求分析から仕様確定，設計・実装・テストというサイクルを，そのシステムが生きて動いているかぎり，何度もくりかえさなければならない．その１回目だけを開発と呼び，２回目以降をメンテナンスと呼ぶのは不自然であって，全体を「ソフトウェア進化」のプロセスとしてとらえるべきだというのが，Lehman の基本的な主張であった．

３．事件の舞台裏

1984年の早春，その Lehman の呼びかけにもとづいて，ソフトウェア・プロセスに関する問題を討論する初めての国際ワークショップ (International Software Process Workshop，略称 ISPW) がロンドン郊外で開催された．日本からの参加は，その前年に別件で Lehman 先生を訪問し特別に参加を許されたわたし１人であった．日本では当時まだ，学界・産業界を問わず，プロセスに対する関心はきわめて低かったのである．

当時はいわゆる CASE ツールの開発やそれらを統合した開発環境の構築が盛んに行われていたが，そのためにはまず，ソフトウェア開発という仕事のプロセスをきちんと分析しモデル化することが必要だというのが，開催の趣旨であった．たしかに，われわれがアプリケーション開発を行う場合には，ユーザの業務プロセスの分析から始めているわけで，自分たちの仕事のコンピュータ化を考える場合に，プロセスの問題を無視するわけには行かない．

第１回の ISPW は，提案者の Lehman およびそのグループを初めとして，Michael Jackson や Barry Boehm その他，多くのソフトウェア工学研究者が集まり，それぞれの意見を発表するという形で，きわめて示唆に富む内容であったが，ある意味では，議論が多方向に発散しまとまりに欠ける嫌いがあった．

そこで，翌1985年にロサンゼルスで行われた第２回ワークショップでは，プログラム委員会を構成した若手研究者たちが，冒頭で次のような概念のピラミッド構造を示し，それぞれのスピーカがこの概念構造のどの位置から発言しているのかを明らかにすることによって，議論を整理したいという枠組みを提示した：

* ツール *
*** 開発環境 ***
***** 開発方法論 *****
******* プロセス・モデル *******
********* 現実の開発プロセス *********

この枠組みに対して，会場の一番後ろの列（わたしの隣の席）にいた Bob Balzer (当時 USC/ISI) がオブジェクションを唱えた．ツール開発者としてのかれのいい分は次の通りであった：

－ ツールや環境はただプロセスを支援し奉仕するだけの存在ではない．
－ ほんとうに革新的なツールはその登場によってプロセスの形を根本的に変える働きをする．

この発言は，ソフトウェア・プロセスの革新におけるツール (ほんとうの意味での開発支援ツール) が果たすべき役割を指摘したという意味で，きわめて的を衝いた意見であったが，ワークショップの

討論はその衝撃によってかなり混乱してしまった :-).

次の第3回 ISPW は，わたしと Balzer とが Program Co-Chair を勤めた第9回 ICSE のプログラム委員会と併設の形で，1986年の秋に，ロッキー山脈のスキー・リゾートで開催された (海抜3千数百メートルの高地だったので，呼吸はかなり苦しかった記憶がある)．このワークショップでは，その後のソフトウェア工学の研究および実践に大きな影響を与えた2つのアイデアが提案された．

1つは，Lee Osterweil による「プロセス・プログラミング」の提案であり，もう1つは Watts Humphrey による CMM の提案であった．

プロセス・プログラミングの基本的なアイデアは「ソフトウェア・プロセスがそれ自身1つのプログラムだ」というスローガンで表現される（詳しくはソフトウェア・シンポジウム'87における Osterweil 自身の基調講演記録 SEAMAIL Vol.2, No.8 を参照されたい）．

このアイデアは，翌年の第9回 ICSE の Opening Keynote として一般に発表され，M.M.Lehman による Counter Talk をペアで，大きな話題となり，1990年代におけるソフトウェア工学研究のメイン・テーマとなった．そして，その後も数年間続けられた ISPW を舞台に，さまざまなプロセス・モデル化技法の開発や，プロセス中心のソフトウェア開発環境の構築が行われた[6].

90年代ソフトウェア工学のひとつの問題点は，そうした研究開発コミュニティにおける動きが，産業界の関心をあまり強く惹きつけることができなかったということであった．産業界は，依然として，ウォータフォール・モデルに代表されるソフトウェア工学のルサンチマン（怨念）にとらわれており，慢性的な開発スケジュール遅れの解消，コストの削減，そして品質の向上といった課題の解決を求め，その目的達成のために，あるがまま (As-Is) のソフトウェアプロセスを分析しツールや環境を利用してその改善を図るということよりは，ウォータフォールにしたがったあるべき (Should-Be) プロセスを実現するためのマネジメント手法のほうに，より強い関心を示した．

Watts Humphrey の提案した CMM は，かれ自身が長年経験した IBM 社における大規模ソフトウェア開発におけるプロセス・マネジメントのノウハウを集大成したものであって，その意味で産業界とくにマネジメント層にとってかなり魅力的な内容を含んでいた．それが，アメリカ最大というか地上最大のコンピュータ・ユーザである DOD をスポンサーとする組織 CMU/SEI の手で，ある種のマーケティング的な意味を持つ言説（CMM Level 3 でないと DOD からの仕事を受託できない etc) として統合・整理されたことが，今日の「事件」のきっかけであったと考えられる．

そしてまた，一方では，ヨーロッパを中心として90年代に発生した「事件としての ISO9000」騒動があった．それは「認証ビジネス」という新しいマーケットをコンサルタントと呼ばれる人びとに提供し，CMM は SWEBOK や PMBOK と並んで，新しい認証ビジネスの材料として受け取られているのである．

4．事件性の本質

1988年の第10回 ICSE (シンガポール) で，D.Perry と G. Kaiser の2人は，ソフトウェア開発環境を，それが支援すべきプロジェクトの規模に対応して比較・分類して，社会学的なメタファのフレームワークを提示した[7].

2人は，開発環境を，3つの相互に関連しあう要素の集合としてとらえ：

{方針，機構，構造}

としてモデル化した．そして，この枠組みを用いて，開発環境をその支援対象の規模に応じて，次の4種類に分類した：

(1) 個人 (Individual) モデル
(2) 家庭 (Family) モデル

(3) 都市 (City) モデル
(4) 国家 (State) モデル

論文の末尾において，２人は次のように述べている：

> 況を表わしている．しかし，これらの２つのモデルは大規模な開発プロジェクトには適切ではない．そこでは，都市モデルのメタファによって導入されるような方針・機構・構造の３つの側面における質的な変化が必要とされるのである．さらに，いくつものプロジェクトを統括する会社組織の支援環境を考えるならば，複数の都市から構成される国家のメタファが，必然的にもたらされる．

このパラグラフを，より単純な形にいいかえれば，

－ 修身・斉家・治国・平天下

すなわち「まず自分自身の振舞いを正すことから始めよう．それができたら，家庭内の秩序を整える．そして，次に，自分の住む国 (それは城壁に囲まれた都市の形をしている) の統治を心掛ける．そうすれば，天下は自然にきみの手に入るだろう」という儒教的マネジメントの理想を掲げた有名なスローガンに帰着する．

ソフトウェア開発組織のプロセス管理面での成熟度を測るモデルとして CMM を提案した Watts Hunphrey は，その後，組織を構成するチームおよび個人のレベルにおけるプロセス意識を取り上げ，

TSP (Team Software Process)
　および
PSP (Personal Software Process)

の概念を提唱し，現在その普及に力を注いでいる．この枠組みが，１６世紀の中国において朱子によって体系化された儒教マネジメントのそれとほぼ１対１に対応する．すなわち：

修身	PSP
斉家	TSP
治国・平天下	CMM

これは，洋の東西を問わず，農耕文明およびその発展形としての工業文明における「社会」の概念が，個人→家庭→都市→国家という規模の尺度の上での「管理」手法の進化と密接に結びついていることを暗示するものである．

「人間は本来集団を作らなければ生きて行けない動物である．たとえ集団を作っても，そこに一定の秩序がなければ乱れ，乱れればお互いに争い，争えば分裂し，分裂すれば弱くなり，弱くなれば敵に勝つことができない」と，儒教思想のある一面の後継者であった荀子は述べている．人間集団としてのソフトウェア開発組織に安定をもたらすためには，なんらか原理 (荀子のいう「分」) によって，そこに 整然とした階層的秩序を生みだし，組織として成熟なければならない．ソフトウェア工学に期待されたのは，そうした原理としての役割をはたすことであった．

> 名正しからざれば，即ち，言したがわず．
> 言したがわざれば，則ち，事成らず．
> 事成らざれば，則ち，礼楽興らず．
> 礼楽興らざれば，則ち，刑罰あたらず．
> 刑罰あたらざれば，すなわち，民手足を措く所なし．

論語・子路篇に記された孔子のこのことばは，開発環境をめぐる諸概念の階層関係を示したものだとして解釈することが可能である．すなわち，階層の最上部には，適切なプロセス・モデル (名) が

あり，それにもとづいた開発方法論 (言) が下層のレベルで 確立されている．でなければ，プロジェクト (事) はうまく行かないし，そのための開発環境 (礼楽) も構築できない．そうなると，マネジメント (刑罰!?) もいい加減になり，民 (プログラマ) はどうしたらいいかわからなくなってしまう．

こうした概念の階層構造は，プロジェクトの組織構造にほぼそのままマッピングできる．したがって，われわれがもし，適切な礼楽 (開発環境) の導入によって何らかの現状変革を成そうと思うならば，まず，最上階の概念レベル (つまり，対応する組織ピラミッドの最上層) において，名を正さなければ (正しいプロセス・モデルとそれにもとづく方針を確立しなければ) ならないというわけである．

1985年の第2回 ISPW における畏友 Bob Balzer のオブジェクションは，まさにそのあたりに向けられていたと解釈することができる．ここで注意すべきことは，ふつう，開発組織自身には，そうしたプロセス認識を改めるべき自浄力が備わっていないという事実である．それは，組織の構成原理が，すべからく人間は，単一の理想を胸に，単一の法則にしたがい，単一の規範にもとづいて行動すべきだという，普遍的統一化の哲学にもとづいているがゆえの，アキレスのかかとだと考えられる．

しかしながら，マネジメント階層に属する世の管理者たちの大半は，この儒教パラダイムがお好みである．Humphrey 氏が「論語 (Analects)」や「大学 (Great Learning)」などの Confucian Classics を読んでいたかどうかはともかく，CMM から TSP さらには PSP というソフトウェア工学的規律 (Discipline) の体系が，本来ソフトウェアの技術的側面にはあまり関心がないが，プロジェクトの予算やスケジュール，あるいは製品の品質にはかなりナーバスであるマネージャたちの心情に強く訴えるものであることは否定できない．

世界最大のユーザである国防省からの強制がアメリカにおける CMM の普及を促進する働きをしたことはたしかである．また，最近の日本における「空騒ぎ」の原因が経済産業省の不用意な(あるいは意図的な)アナウンスメントにあることもたしかである．しかし，それだけではなく，その裏には，上に述べたようなマネジメント哲学の本質がひそんでいることを忘れてはならない．

そして，そのさいわれわれは，儒教的マネジメントのパラダイムが，その理論的美しさとは裏腹に，伝説の聖王たちを除けば，現実にはいまだに成功した例がみられないという点に注意を払うべきであろう．浅野裕一氏が鋭く分析されているように，儒教は，現実には一度も王権を握ることができなかった「怨念の宗教」(怨念の政治哲学) なのである[7].

短絡的アナロジーをもっていえば，ウォータフォールへの願望にとりつかれてこれまで発展してきたソフトウェア工学も，また，現実には力を発揮しえない「怨念のエンジニアリング」なのかも知れない．

5．事件の行方(？)

昨年の暮れ，経済産業省は「ソフトウェア プロセス改善に向けて」なるドキュメント(案)を Web Page に公開し，パブリック・コメントを求めた．以下は，その案内を貰ってわたしがある ML に投げたコメントである：ほかにどのようなコメントが集まっていて，それらがどのように扱われるのかはわからない．

> 「1．現状認識」にかかれていることは大筋で間違ってはいないと思うが，そこからなぜ「2．ソフトウェアプロセスの改善」が出てくるのか，論理がまったくつながらない．
>
> 推測される理由は，今回のプロジェクトの構想が，「はじめに CMM ありき」(第1回専門委員会における萩原氏の発言) から出発し，「政府調達における日本版 CMM による SPA の採用」という「単なる思いつき」を強行しようとして，多くの批判を浴びたために，なんとか「SPI & SPA」という組み合わせを導入する形で，無理やりつじつまをあわせようとしたからではないだろうか．
>
> 私見では，「1．現状認識」の分析から出てくる当然な論理的帰結は，わが国における実

践的ソフトウェア工学の技術開発の振興であって，SPI & SPA はそのうちの１項目にすぎず，しかもプライオリティは，各種開発方法論や支援ツールの整備・実用化といった急務の事項と比較してかなり低いと考えられる．

もし，一歩ゆずって，「プロセス」問題を第一に取り上げるとしても，CMM や ISO15504 のようなマネジメント技術にかかわることがらの重要性は，プロセスモデル化技術の開発や，インターネット時代のアプリケーション開発に対応する新しいプロセス モデルとそれをベースとする開発支援環境の実用化といった課題と比較して，一段と低いと考えられる．

現在の文書~が，このように偏った内容になった原因は，上に述べたように，そもそもの誤った構想にもとづいて，そうした内容の審議だけを考えた委員会メンバーの選定が行われたからだと考えられる．

国際的なソフトウェア プロセス コミュニティにおける議論は，1984年 M.M.Lehman 教授（英国 Imperial College) の提唱にもとづく一連のワークショップ (ISPW) を中心に行われてきており，プロセス モデル化技法やプロセス中心の開発支援環境などについて，すでに多くの研究成果が得られている（詳細については，岸田監訳の「ソフトウェアプロセスのトレンド」海文堂刊を参照されたい）．日本のソフトウェア工学関係者もこの国際的研究開発には多数参加しているが，その人びとは今回の協議会や委員会のメンバーにはなぜか含まれていない）．CMM はこの ISPW が生み出した多くの成果のうちの１つにすぎず，たまたまそれが世間の注目を集めているのは米国国防省がそれを採用したからだというだけのことにすぎない．

もし，単に CMM の日本への導入という思いつきにもとづく狭い視点からの議論ではなく，日本のソフトウェア技術・ソフトウェア産業の将来を考えてプロセス問題を論じようというのであれば，そのための協議会および専門委員会の人選をあらためてやりなおしてスタートから再出発すべきだと考える．

前回のパブリックコメントをめぐって，故・坂本啓司氏が METI と打ち合わせを行った際のメモには，そうした根本的な見直しを行うことが明記されていたはずだが，それがなされないまま，こうした姑息なつぎはぎドキュメントがまたしてもパブリックコメントに付されるというのは，どうにも納得が行かない．あらためて，新メンバーによる議論を行って，根本的な書き直しをすべきだと考える．

参考文献

[1] 子安宣邦，「事件としての徂徠学」，青土社, 1990. その後，ちくま学芸文庫でも再刊.
[2] 野口武彦，「荻生徂徠 − 江戸のドン・キホーテ」，中公新書，1993.
[3] W.Royce, "Managing the Development of Large Software Systems", ICSE-9 Proceedings, 1987 (pp.328-329) に再録.
[4] C.Floyd, "A Paradigm Change in Software Engineering", in Software Engineering Note, Apr. 1988.
[5] M.M.Lehman & L.A.Belady, "Program Evolution: Processes of Software Change)", Academic Press, 1985.
[6] A.フュジェッタ & A.ウルフ編，「ソフトウェア プロセスのトレンド」，海文堂，1997.
[7] 浅野裕一，「儒教 - ルサンチマンの宗教」，平凡社新書，1999.

オブジェクト指向とレトリック

玉井 哲雄
(東京大学)

佐藤信夫という人を，大変遅れて「発見」した．講談社学術文庫から出ている5冊の本はすべて読んだが，どれも素晴らしい．その中で「レトリック感覚」と「レトリック認識」の2冊は，ヨーロッパの伝統的な分類に従って，さまざまのレトリックを順に解説している．解説といっても堅苦しい講義調のものではなく，自分で採集した文例を多く引き，また佐藤の文章自身がレトリックに満ち溢れた名文となっていて，楽しませる．

さて，そこに登場するいくつかのレトリックを，オブジェクト指向で現れる概念と結びつけて紹介しようというのが，この小文の意図である．オブジェクト指向と関連づけようなどという魂胆は，さもしいものかもしれない．ただただ，佐藤の技を堪能すればいいようなものだが，思わず自分の専門分野に引きつけてしまうのは，一種の職業病かもしれない．

1．直喩 (simile，シミル)

森鴎外の『雁』から，「轡虫の鳴くような調子でこういうのは」というのを引いて，「その虫の実物を知らず，いや，たとえその鳴き声を聞いたことがあてもそれがその虫の声だと名ざすことのできない現代の都会の若者にとってさえ，じつはこの虫の鳴き声の≪直喩≫は決して無効となっていないのである．」というのがすごい．こういうところに感心した．しかし，こちらは「現代の都会の若者」ではないが（ただ佐藤よりは16歳若い），「それがその虫の声だと名ざすことのできない」口である．

*X*を「*Y*のよう」と表現したとして，*Y*をその言葉が指す実世界の対象物（つまりオブジェクトだ）そのものとしてとらえて比喩を理解するよりも，むしろ言葉としての*Y*が歴史や文化の中で育んできたイメージを通して理解するのだという．たとえば「狐のように」とか「獅子のように」という場合のことをいっている．

さらに驚くべき，また首肯せざるをえない指摘は，*X*を「*Y*のような」と直喩で表現されて，*Y*を知らない場合，逆に*X*から*Y*のイメージを作る，という言語行為があるという．フームなるほど，という感じである．これは上の轡虫の例を引き合いに出して，指摘される．

そして，次のような素晴らしい一節がくる．

「≪直喩」や，次章の主題である≪隠喩」は，≪ふたつのものごとの類似性にもとづく」表現である，というのが古典レトリックの定説であった．しかも，現代的なレトリック理論でも，この考えかたはつねに認められている．けれども，ここで私はそれを逆転させ，類似性にもとづいて直喩が成立するのではなく，逆に，≪直喩によって類似性が成立する≫のだと，言いかえてみたい．」

実は，ここに至るまでに，『雁』だけではなく，太宰治や安岡章太郎や川端康成の直喩の誠に適切な引用がある．さらにこの後，漱石や黒井千次や丸谷才一の例文もあって，それだけでも大変楽しめる．文庫の解説で佐々木健一が書いているように「著者の発見を導いた論考が，自前の例文，自分で調達してきた例文に支えられている」のが素晴らしい．

きりがないが，はっとさせられる記述をさらに少し挙げれば，「類似性という概念を≪どれほど似ているか．．．」と考えるくせがあり」，「だからこそ，直喩を類似性にもとづく比喩だと言う」が「類似性とは≪どれほど似ていないか≫ということでもある」というあたり．

しかし，これらの議論は面白いが，オブジェクト指向と牽強付会するには捉えどころがなさ過ぎてしまうかもしれない．あえて言えば，次あたりが関係をつけられそうか．

「*X*をそれに似ている*Y*によって表現する記述法は，つねに一面的であり，不じゅうぶんである．

*X*をなるべく妥当なかたちで認識しようという方策のひとつとして，*X*を*Y1*，*Y2*，*Y3*，*Y4*，．．．*Yn*}．．．と，多角的にたとえる工夫に思いいたる人がいても不思議ではない．」

2．隠喩 (metaphor，メタファー)

さて，隠喩，メタファーである．

まず，ヤブヘビという佐藤としては通俗的な例から入り，定義らしいものとして，「あるものごとの名称を，それと似ている別のものごとをあらわすために流用する表現法が隠喩—メタフォール(メタファー)—である．」という．

続いて，『ロメオとジュリエット』から「君の白鳥がただの烏だったと」というせりふを挙げ，さらにしりとりのように，太宰の『渡り鳥』から「日比谷公会堂から，おびただしい数の烏が」と人の群れを烏と呼ぶ例を引く．この辺が佐藤のうまさであり，またさりげなく博覧強記振りを示すものである．

次にまた，レトリックの歴史を尋ねて古典を引き（といっても*19*世紀と比較的新しい），それを敷衍してより形式的な定義として「形式を見れば，*X*と *Y*というふたつのものごと（あるいは観念）が互いに類似しているとき，*Y*の名称（あるいは記号）すなわち*Y*を借用して*X*を表現すること」が古典時代以来の，まあ平均的な考え方だという．次の言い方のほうがもっと計算機屋に訴えるだろう．「つまり，平常表現ならたぶん*X*という語句が使われるべきところに*Y*という語句が≪代入≫されるのだ・・・」しかしこの≪代入理論≫は近ごろの言語理論家のあいだでは評判がよくないという．その理由もいろいろ書いてあるが，それをいちいち追っていたのでは，全文紹介になってしまう．

さて，歴史的には隠喩は直喩を簡潔に言ったもので，それだけ高級な表現とされてきたらしい．しかし佐藤は，「直喩は隠喩の幼稚な変形」という見方に異を唱える．そもそも同様の表現を直喩と隠喩で書き分けるという例文による説明が，いけないのだという．同じ例題が直喩でも隠喩でも了解できるなら，隠喩の方が表現としてすっきりしているだろう．しかし，隠喩が成立するためには，*X*と*Y*の類似性が語り手と聞き手の間で共通化されていなければならない．ところが直喩は*X*と*Y*の類似性を提案し，類似性を設定するものだから，*X*と*Y*}がまるで似ていなくてもいいと佐藤はいう．すでに直喩の章でもこの考え方が提出されているが，実に卓見だ．

隠喩はあらかじめ了解できる類似性に基づくから，潜在的にステレオタイプな表現になる危険があるともいえる．しかし，それは「危険」などというものでなくて，ステレオタイプ化した隠喩が普通の語や表現として使われるようになり，ほとんど隠喩であることも意識されなくなることにより，言語が豊かになるという考え方がある．極端に言うと，ほとんどあらゆる表現が隠喩と見なされる．それがおそらく*G. Lakoff*等の立場であろう．佐藤はレイコフの仕事をもちろん知っているだろうが，どの著作でもほとんど無視している．多分，評価していないのだろう．

3．換喩（metonymy，メトニミー）

レトリックとオブジェクト指向との関係などとあえて持ち出した大きな理由は，この換喩/メトニミーと次の提喩/シネクドックの存在による．

換喩とは，佐藤が引くモリエの定義によれば「あるひとつの現実*X*をあらわす語のかわりに，別の現実*Y*をあらわす語で代用することばのあやであり，その代用法は，事実上または思考内で*Y*と*X*を結びつけている近隣性，共存性，相互依存性のきずなにもとづくものである．」という．これは分かりにくいが，典型的な例として挙げられるのが，「赤頭巾ちゃん」である．赤頭巾と女の子は似ているわけではないが，女の子が頭にかぶっている赤い頭巾という「近接性」で代用しているのである．それに対し，「白雪姫」は類似性に基づく隠喩表現となる．

余談だが，「赤頭巾」や「白雪姫」は今や，違和感のない日本語となっている．「頭巾」という語は，日常的にはほとんど死語になっているにもかかわらず．あまりに日本語として馴染んでいるの

で，原語で何というか，浅学の筆者などは調べてみないと分からない．ディズニー映画のお陰で，英語では白雪姫を *Snow White*ということだけは，わずかに知っていた．赤頭巾ちゃんはディズニーにも出てこないので，英語すら分からない．調べてみると，ペローの赤頭巾は *Le Petit Chaperon Rouge*である．赤頭巾は女の子だが，換喩では当然ながら代替されている語に性は従うようだと，妙なところで納得する．グリムの方は，*Rotkappchen*というらしい．それぞれの英訳は*Little Red Ridinghood* と *Little Red Cap*である．これが，現在日本語に翻訳されたとしたら，カタカナ名前になる危険が大いにある．

閑話休題．換喩は「近接性」とか「隣接性」に基づくというのだが，なにをもって隣接していると見なすかは，曖昧である．そこで，古典的なレトリックの学者は換喩をさらに分類整理して，「原因」で「結果」を表現するとか（その逆に「結果」で「原因」を表現するのもある），「容器」で「内容」を表現するとか，「抽象名」で具体物を表現する，などと列挙した．しかし，佐藤も言うように，これらは思いつきのような列挙に過ぎず，煩雑な一方で，換喩らしい表現でこのリストのどれにも収まりが悪いものがあるという意味で，不足でもある．

「赤頭巾」以外に典型的な例として佐藤が挙げるものは，たとえば蕪村の次の句である．

春雨やものがたり行く蓑と傘

あるいは太宰治の『道化の華』で口髭を生やした刑事を「髭」と呼ぶ例．永井荷風の『墨東綺譚』で古着屋を「禿頭」と呼ぶ例．鴎外の『雁』を再び取り上げて，末造が無縁坂に住まわせたお玉という女の家を「無縁坂」と呼ぶ例．これらから佐藤は，頭巾や帽子と人間とか建物と場所は確かに隣接しているが，原因と結果のような縁故関係もまとめて広い意味での隣接関係と呼び，さらにひげやはげ頭と人間のように隣接というより含有と呼ぶべきものも，隣接性に入れる，という．そして「言いかえれば，物の全体と一部分との関係もまた，ここでは広く隣接という名称で呼んでおくことにする」と言うのである．

ここでオブジェクト指向モデルに携わる人は，ただちに「全体と部分の関係」あるいは集約関係*(aggregation)*を連想するだろう．換喩とは「集約されるオブジェクトの*1*つで集約するオブジェクトを表現することである」と言ってしまいたい衝動にかられる．

さらにすこし違う話も出てくる．井伏鱒二の『多甚古村』を引いて，「駐在」が巡査を指している例を挙げ，「駐在」という抽象的な語を「具体的な物に流用するという，抽象⇔具体の代入作用による換喩がここに働いている」という．そうなると，このレトリックは「クラスでそのインスタンスを表現したもの」と言えそうである．つまり，換喩には集約関係とクラス/インスタンス関係の両者が使われるということになるだろうか．しかし，普通名詞を個体(インスタンス)化するには，英語などでは定冠詞が使われるし，日本語では「あの」「その」のような連体詞を使うのが一つの方法だ．この「駐在」の例はあまり説得力がなく，他の例は挙げられていない．この後の話を進める都合上は，ここでは集約関係だけに注目しておいた方がよい．

4．提喩 (synecdoche，シネクドック)

まず，例を挙げよう．佐藤が引くのは井上靖の『比良のシャクナゲ』から「その日一日時折思い出したように舞っていた白いもの」である．この「白いもの」は「雪」の提喩表現である．佐藤の提喩の章は，この例文を冒頭に挙げているにもかかわらず，それに即した説明をする前に*23*ページも費やしている．その理由は後にしよう．これと似た，さらに通俗的な例は「花」で「桜」を指すものである．

あるいは，芥川の『羅生門』から「白い鋼の色を，その眼の前につきつけた」を引き，「白い鋼の色」が「刀身」の提喩であるという．これらはあるものを，必要以上に一般的な表現で指すというレトリックである．逆に，必要以上に特殊な表現で指すものもある．佐藤は例として使っていないようだが，「人はパンのみにて生きるにあらず」という時の，「パン」で「食物」を指すような場合である．この類の別の典型例は，好色な男を「ドン・ファン」と呼び，日和見主義者を「筒井順慶」と呼

んだりするものである．また，「ナイロン」とか「セロテープ」のように商品名が普通名詞化したのもこれに入る．

これと換喩との違いについて，レトリックの長い歴史の中で混乱と論争とがあったという．その原因は，このような提喩の性質を「部分」と「全体」の関係で説明しようとしたことにあったらしい．換喩の中にも「部分」と「全体」の関係があることは，すでに見た通りである．というより，われわれはそれを換喩の定義の中心と見た．伝統的なレトリックの理論では，*X*と*Y*が隣接関係にあって，*X*を*Y*で指すレトリック表現に対し，*X*と*Y*が外部的に独立した存在のときにこれを換喩と呼び，*X*が*Y*に含まれる（あるいはその逆）のような内部的な包含関係があるときにこれを提喩と呼ぶ，というものが主流だったという．しかし，想像できるようにこの区分はそれほど明確でない．そこで，換喩と提喩の区別をなくそうという流派もあるらしい．

佐藤はこの問題に*20*ページ余りを費やして，全体と部分の関係についてのΠ様式とΣ様式という古くからある概念を持ち出し，解決を図る．木が「枝および葉および幹および根」からなる，というような「および」，論理的な連言，で結合されるのがΠ様式であり，木は「ポプラまたは柏または柳または樺」からなるというような「または」，論理的な選言，で結合されるのがΣ様式である．もちろん，Π様式に対応するのが換喩であり，Σ様式に対応するのが提喩というわけである．

オブジェクト指向を少しでも齧った人にとっては，このΣ様式がオブジェクト・クラスの一般化/特殊化あるいは継承関係に対応することは，一目瞭然である．つまり，換喩＝集約，提喩＝一般化ときれいに整理できる．ついでに言えば，隠喩＝関連*(association)*となるだろう．ただし，「類似性」という尺度でつけられた関連である．そして直喩は同じ類似性に基づく関連ではあるが，オブジェクト間に生じる自然な関連というよりは，オブジェクト間に作為的に関連の線を引くことによって生まれた類似性ということになるだろうか．

もちろん，オブジェクト指向の技術者は，過去のレトリック学者の頑迷さを嗤うべきではない．何しろ歴史が違う．ヨーロッパのレトリックは古代ギリシャに始まり，中世を経て連綿と続いてきた．そしてこれも佐藤によれば，*19*世紀後半から*20*世紀前半にかけて，科学主義，合理主義，実用主義に押され，無意味な装飾として見捨てられた．新たな視点から復活し始めたのは，*1960*年代に入ってからのことだという．その最近の歴史だけでも，オブジェクト指向をはるかに凌駕する．事実は，このような言語論，認識論の伝統から，オブジェクト指向は直接にせよ，間接にせよ，恩恵を受けたということだろう．

実際，オブジェクト指向に先立つ意味ネットワークや*ER*モデルの時代に，計算機科学の世界でも全体と部分の関係について混乱のあった時がある．意味関連の内の特殊なものとして*is-a*という言い方があったが，これがクラス/インスタンスの関係を指す場合と，クラス/サブクラスの関係をさす場合とがあった．後者を*a-kind-of*と区別して呼ぶ流儀もあったが，必ずしも統一されていない．さらに，*is-a*で集約関係を指してしまう人もいた．これには*a-part-of*という言い方もあるにはあったが．

5．おわりに

さて，レトリックとオブジェクト指向をくっつけた話は，これで大体終りである．またまた佐藤によると，ここまで扱った直喩，隠喩，換喩，提喩をまとめて「転義*(trope)*」というのが普通らしい．これらは「*X*を指すのに別のものを指す*Y*という語を用いる」という形式が共通である．これを代入操作と考えると，提喩のある種のもの以外は型の整合性を破ってしまうが，厳密な型検査ができるようではレトリックの妙味はないということであろう．

佐藤の本はさらに続いて，誇張法，列叙法，緩叙法から黙説，ためらい，転喩，対比，逆説，諷喩，反語，暗示引用と豊富に繰り広げられていくが，そこまでオブジェクト指向に牽強付会を続けるのは，野暮というものであろう．

音楽とソフトウェア創りの似ているところ

野呂 昌満
(南山大学)

本稿は慶應義塾大学理工学部管理工学科大駒研究室で毎年発行されて来た文集Buechelchenに寄稿した文章を加筆修正したものである.

Rockに憧れて, Folkに惚れ込んでギターを弾きはじめて30年が経った. 最初にOriginal Cast のMr.Mondayを聞いてから軽音楽(なぜ「軽」なのか. 軽蔑の「」軽なのかね)にはまってしまった. そのときの衝撃がもとで歌を創るようになり, いくつかのバンドで音楽をやって来た. まだ音楽では成功していない.

UNIVAC1101のカードリーダにFORTRANでsinカーブを書くプログラムをくべて, ラインプリンタからグラフが出て来たことを滅茶苦茶に喜んでから23年経った. 大学の講義科目で強制させられてやってみたものだが, そのときの衝撃は強烈で, その後プログラムを創ることにはまり, 働くのが嫌で目的意識もなくずるずると大学にいる間に, いまの仕事になんとなくついてしまった. ここでも, 成功はしていない.

私にとっては, 楽曲を創ることと, ソフトウェアを創ることは趣味であり仕事であるわけだが, 多くの共通点を見出して来た. インプロビゼーション(アドリブといった方がわかるかな)は, フレーズの組合せをいかに斬新にタイムリーに行うかが勝負である. フレーズはパターンであり, ソフトウェア創りにおいて, いかに適切にパターンを使うかが勝負であることと似ている.

誰でも創れるものは面白くないのは, 音楽もソフトウェアも同じ. フレーズはあってもそれを組み立てる技は試行錯誤(練習)の末得られるもの. 本読んでデザイン・パターンいくつ知ってもそれをうまく組み合わせることはできなくて, 幾多の試行錯誤の末, 個人のノウハウとして組み合わせ方が熟成される. 個人のソフトウェア・プロセスとして一般化なんかできない.

楽曲を創るときはリフと呼ばれるテーマを思いついたら, コード進行をはめてうまく行けばできあがり. コード進行を支配するのは曲のアーキテクチャ, ロックだと大抵, 「イントロ, 1番(Aメロ, Bメロ, Cメロ(サビ)), 2番, 間奏, 3番, Cメロ繰り返し, エンディング」等とパターン化できる. ブルースなんてほぼワンパターン.

斬新なアーキテクチャやコード進行を発見することが何よりも重要. だから, ソフトウェアアーキテクチャのことを研究の対象にしている. フレーズや曲のアーキテクチャはたくさん音楽を聞いているうちに, 自分の中で熟成される. よいソフトウェアをたくさん読めば, ソフトウェアを創れるようになるのと同じ.

バンドで演奏することはチームコーディネーション. うまい奴が集まればよい音楽ができるかというと, そうでもない. Paulの居るときのMr.Bigがよい例. そんなにうまくない奴が集まっても, 呼吸が合えばよいバンドになる. ウルフルズやローリング・ストーンズがよい例. チームでソフトウェアを創るのも同じ.

これらのことを書いているのは, 以下のエッセイを披露するための前振り.

しばらく休んでから, 再びソフトウェアを書くときは敷居は高いがやりだすと止められない. いっそのこと, いくつかのチームに分けて, 2ヶ月働いたら1ヶ月休むなんてローテーションが組めれば, もっと生産性は向上するかも.

以下は, 学位論文や留学でしばらく音楽を休んでいた後のエッセイ. 煮詰まって来ると義務感だけが先立つもの, 何か眠るのがもったいないとワクワクしている自分が居る.

僕を忘れたころに

(1992. 10)

「僕を忘れたころに, きみを忘れられない, そんな僕の手紙がつく...」. 吉田拓郎の「春だったね」の出だしです. どうして, いまさらまたこんな懐かしい歌がでてくるのかというと. ついに私たちはやることにしたのです. バンドをつくって学園祭でコンサートをすることに. 年甲斐もなくというほど年ではないのですが, 決めてしまったのです.

こと始まりはこうでした. 5年前に今の大学(南山大学@名古屋)に来ました. 同僚に後藤さんという人がいて, 彼も音楽が好きだということでした. それで, よく聞いてみると, 彼のバンドは, あの, 今はなきヤマハのポピュラーソングコンテストのつま恋本選会に出場したというではありませんか. 彼は京都大学出身なのですが, 「酔麻痴草」という彼のバンドは京都ではかなり有名で, いくつかのライブハウスでギャラを貰って演奏していたとのことです.

一方私は, たくろうフリークで, 数々のレコード会社, コンテストにオーディション・テープを送り, いつも第一次審査は通るのですがその次でだめ, という程度の実力でした. 高校では, 友達と2人でグループをつくり, バックバンドを時々頼んで, コンサートをよくやっていたのですが, 大学に入ってからは, そういうことも止めていました. このごろでは, 唄をつくることもなくなって, 「趣味はゴルフです. 」というようになっていたのですが. それが今になってなぜ?

じつは，私にはもう５歳になる子供がいます．５歳といえば当然幼稚園です．その幼稚園の文化祭で，パパさんバンドなるものをつくって子供のために歌いませんか，という，わけのわからない企画が持ち上がったのです．嫁さんが，幼稚園の父兄会の役員などをやらされていたことから，こちらにポインタが向いてきたのですが，最初は当然のごとく，「そんなものやってられるか」という気分だったのです．「まあ，そういわずに」と１回目の練習だけでも参加して下さいといわれ，やってみたのですが，これが結構おもしろいのです．幼稚園の講堂で練習している時に，だれも聞いていないとわかっていても，子供を迎えに来るお母さんが何人か覗いたりすると，もううれしくなってはりきってしまう自分を発見してしまったのです．血が騒ぐというやつです．

それならばということで，子供相手にウルトラマンタローなんか歌っているよりは，本格的にやろうということになった訳です．まずはメンバー探しから始めました．後藤さんもあまり乗り気ではなかったのですが，強烈にプッシュして納得してもらいました．それでも，２人ではやりたい曲を考えた場合パートが足りません．最低限，ベースとドラムを捜さないとだめです．できれば，キーボードも必要です．こうなったら，学生さんを捜そうということで，バンドをやっている奴らを捜してきて，ベースとドラムは確保しました．キーボードはなかなか見つからなかったのですが，何とかメンバーを確保できました．

それで曲を決めて，練習をとなったわけですが，なかなか曲が決まりません．吉田拓郎の曲なんてという奴がいれば，わたしはギンギンのロックンロールはできないし．と，いろいろ紆余曲節はあったのですが，一応10数曲決まりました．そして，いよいよ練習です．久しぶりにスタジオを予約して，楽器を持って集まりました．最初は，全員それほど期待してはいなかったのですが，合わせてみると，結構いいのです．「おっ，これは」という感じで曲をこなしていく内に，全員，「思ったより，いい．これはいけそうだ．」となり，満足して練習を終えました．

出番は学園祭，１１月３～４日．僕を忘れたころに，君を忘れられない．

ソフトウェアツールは私の場合ギター，その構造やデザインが気になってしかたない．SUN Users Conference でサンフランシスコへ行ったついでに，リッケンバッカーを買って来たのですが...

あこがれのリッケンバッカーのはずが....

(1993.10)

ビートルズがよく使っていたエレキギター，ベースにリッケンバッカーというメーカーのがあるのを知っていますか．そのリッケンバッカーのギターを買ってしまった話しをします．去年の学園祭にバンドを作ってライブをやった話は，ここで書きましたが，実はその原稿を書いていたころに僕が持っていたギターはアリアのD-80とグレコのテレキャスターでした．

両方とも高校生のときに買ったもので，D-80はマーチンの D-35のコピー・モデルのアコースティック・ギターでグレコのテレキャスターはフェンダーのテレキャスターのコピー・モデルのエレキギターです．両方とも20年も使って来たもので，かなりガタが来ていました．それでも，アコースティクのほうは大切にしていたので，この世でこんなよい音のするギターないというぐらい鳴りがよくて，あちらこちらにある傷を除けば秀逸なものと呼べました．エレキのほうは，女に振られたやつあたりの対象になったりしたせいもあって，また，ネックの調整を自分でやったり，ピックガードに落書したりしてかなりボロボロになっていました．

ライブをやるのにまともに音の出ないギターでは駄目だということで，エレキギターを買うことにしました．いろいろ悩む間もなく，テレキャスターを買うことにしました．本家本元のフェンダーUSAのを買うと20万円ぐらいするので，コピー・モデルを捜していると，何とフェンダーJapan という系列会社がロゴも寸分違わないテレキャスターを発売していることを知って，それを手に入れることにしました．なんたって，テレキャスターといえば吉田拓郎やブルース・スプリングスティーンが使っている憧れのギターだったわけで，それが安く手に入れば文句ないのは当然です．

ライブでも我テレキャスターは思いっきり活躍し満足していたのですが，いくつか問題が出てきました．その1つは，音造りが難しいということ．アンプによってまったく別のギターのように音が変わってしまい，なかなか安定しない．とくに，自分に聞こえてくる音がめちゃくちゃ気にいる日もあれば，捨ててしまおうかと思うぐらい駄目な音しか出ない日もあることがわかってきました．

不思議と他人にはそうではないみたいなのですが，音楽はプレーヤーの気分的なもので演奏がガラッと変わったりするので(いわゆるノリというやつですね)，自分に聞こえる音が気に入らないというのは非常に大きな問題なわけです．まあ，アンプのセッティングが下手といえばそれまでなのですが．しかし聞くところによると，プロもそうらしくって，テレキャスターは音を造るのが難しいと評判のようです．

そこで，もう1本と思って，いろいろ捜してみたのですが，ついにというかやっとというか，ルックスがよくてめちゃくちゃ気にいったギターをカタログで発見したのです．それが，リッケンバッカーで，何と吉田拓郎が一番新しいCDで弾いているということなので，絶対に手に入れてやろうと思ったのでした．しか

知れませんが，リッケンバッカー独特の，抜けがあり芯のある音がするということです．じつは，リアのピックアップでクランチ設定してやるとテレキャスターをリッケンバッカー風にした音になり，フロントとリアを同時に鳴らしてオーバードライブをかけてやると，ギブソンにも負けないウーマントーンが得られます．

コントロールもギブソンのレスポールと同じ2ボリューム2トーンでピックアップセレクタはリア，リアとフロント，フロントの3段切替えなので，操作も比較的簡単で，トーンとボリュームの組合せで，いろいろな音が作れるところが，楽しいところであり，腕の見せどころでもあります．アンプはフェンダーのツインリバーブが一番相性がよさそうなので，中古を物色中です．

あと，このギターの気に入っている点は，持っている奴が少いということ．レスポールやストラトキャスター，テレキャスターなんかと違って持っている奴を見たことがない．目立ちたがりの私にはぴったりのギターです．さて，このギターを持って今年も学園祭，クリスマス，卒業お祝いコンサートといろいろ出番があるので，さー頑張るぞ．

Jumping Jack Flash，it's a gas!

エレキギターは楽器ではあるのだが，工作物として見ても面白い．ソースコードを見るとやたら張り切るのと同じく分解したり組み立てたり果ては作ってみたり...

またギターの話
(1995.10)

エレキギターは本当に面白い楽器だ．電磁石で弦振動を拾って，増幅機に伝える，それだけの楽器なのに，音に影響を与える要素がいくつもある．思いつくまま挙げてみると，

- ボディーの材質
- 指板の材質
- ネックの長さ
- フレットの形状
- ピックアップの種類
- ピックアップの位置
- 弦の太さ

などが音に影響を与えるものだ．ギター各部の名称や基本的なことはわかっているものとして話を進める．

弦振動をピックアップと呼ばれる電磁石で拾って，その振動で起こった微弱電流をアンプと呼ばれる増幅器に伝えるというのが，エレキギターの動作原理なのだ．そう，電磁誘導の応用である．電気的なものなのに，ボディーの材質がなぜ影響してくるのかよくわからない．

多分，ボディーの材質が弦振動に影響を与えるので，音が変わると思うのだが，果たして，いくつかの楽器屋の兄ちゃんや，はては，ギター製作専門学校の卒業生にまでいろいろ尋ねてみたが，どうも要領を得ない．彼らもわかっていない様子である．いずれにしても，以下わかっている範囲でそれぞれの説明と私の好き嫌いを話してみよう．

ボディーの材質

ボディーの材料として一般的なものは，メイプル(楓)，アルダー，アッシュ，マホガニーなどの木材だ．アルダーやアッシュは硬い音のでる材質で，マホガニーは軟らかい音，メイプルはその中間の材質だと思う(どちらかというと硬めの音かな)．前号でも紹介した，Rickenbackerの650はメイプルボディー，FenderのStratocasterはアルダーボディー，Telecasterはアッシュボディー，GibsonのSGはマホガニーボディー，LesPaulはマホガニーにメイプルを貼ったボディーといろいろである．何本ものギターを集めてみた結果，どうも私は，硬い音のギターが好きなようである．

指板の材質

指板の材料はローズウッドやエボニーなどの黒たん系の木材かメイプルが使われる．いずれも硬い木材だ．指板はネックと一体整形にする場合とネックの上に貼り込む場合がある．ネックにはメイプルやマホガニーなどの材を使う場合が多い．すなわち，メイプルネックのギターのみ指板と一体整形できることになる．

一体整形のものをメイプルワンピースと呼び，マホガニーネックにエボニー指板を貼ったものをマホガニー・オン・エボニーと呼ぶ(何か逆のような気もするのだが，とにかくこう呼ぶので仕方がない)．メイプル・オン・メイプルなんていうのもある．メイプル・オン・メイプルとメイプルワンピースがどう違うのかはよくわからない．一体整形のネックの方が強度が高い，またはその逆など，いろいろ議論はあるが，正しいところはわからない．

さて，メイプルの指板はローズ等に較べて軟らかい音が出るといわれている．ローズやエボニーは音の立ち上がりがシャープで輪郭のはっきりした音が出るといわれている．私にいわせれば，どうも疑問である．どちらも同じようなもので，フレットの形状の方が音に与える影響は大きいと思う．指板の好き嫌いは，触った感触と，見た目だと思っている．

メイプルはボディーはいうに及ばず，指板として使うときも塗装する．ローズやエボニーは塗装しない．これによって受ける感触の差は大きい．メイプルの場合ツルツルすべる感じがするのに対して，ローズはしっとりとフィットする感じがする．私はメイプルの感触が好きだ．また，メイプルは明るい色なのでギター全体の印象が明るいものになるのに対して，ローズ

し，日本で買うと定価40万円弱，中古でも 20万円ぐらいするので，ちょっと手の届かない値段です．あきらめようかと思った時に天の助け．Sun Users Conference でサンフランシスコに招待されたのです．

そのとき，ベイエリア中の楽器屋をさがしてやっと見つけたリッケンバッカーの350JG，何と1100ドルでした．楽器屋の妙にゲイっぽいにいちゃんには金で払うのか，お×××んで払うのかなどとからかわれても，とにかく手に入れたのです．帰りの飛行機ではうきうきして，機内持ち込みまでさせてもらって，大はしゃぎの僕でした．

ところが，ところが，楽器屋であれだけ弾いて，確かめたはずなのに我 350JGはなにか弾きづらいと感じるようになってしまい，ついには，「やはりテレキャスターなのだ」とかいって，リッケンバッカーをストック状態にしてしまったのです．しかし，サンフランシスコまで行って， 1100ドルも出して買ったのだから，と出してみては弾くのですが，どうもフィーリングが合わない．原因を考えてみると，いくつか考えられました．

まず，ネックが細過ぎる，結果として弦間隔が狭くなりミスタッチの原因になる．つぎに，フレット間隔がどうも狭い，とくにハイポジションになると余計に狭くなる．そして，最大欠点は指板に塗装がしてある．知識のひけらかしになるのですが，ギターの指板にはメープルの木を使ってあるものとローズウッドと呼ばれる密度の高い木を使ってあるものがあります(これ以外にもありますがまあ多くはこのどちらかです)．メープルはラッカー塗装と相性がよく，塗装が指の滑べりを助けてくれて弾きやすくなります．一方，ローズのほうは，ギブソンのギターに多く見られるように，塗装しない方が指の滑べりがよくなります．

これは好きずきですが，僕としては，ローズの塗装なしが一番使いやすいと思っています．ちなみに我テレキャスターはメープルにラッカー塗装です．リッケンバッカーはローズによく似たパドゥークという材質に塗装がしてあって，これが引っかかるフィーリングの元凶のようなのです．このせいで，指が滑べらないので引っかかり気味になり，下手に聞こえる．など，自分のテクニック不足を棚にあげてあれこれ考えたわけです．

では塗装をはがせばいいと考えて，リペアショップに持っていって尋ねたところ，パドゥークに塗装してあるのは，音を立たせて音の伸びをよくする工夫で，それがリッケンバッカーのよい点だとの指摘を受けまいりました．また，塗装をはがすとフレットも打ち替えになり，工賃が10万円位になるとのことです．これでは，ギター本体とおなじくらい投資しないといけないわけで，あきらめました．

そして，我リッケンバッカー350JG は時おり思い出したころにちょっと弾かれてはまたケースの中へ逆戻りという状態になっているわけです．やっぱり，ギブソンかななどといって，ローズ指板で2ハムバッカー(パワーがあってハードロックに持ってこいの音を出すエレキギターのピックアップ)のSGやエクスプローラが欲しいなどと浮気心チラチラの僕です．ああ，あこがれのリッケンバッカーのはずだったのに....

いくつもソフトウェアを買って来てはマニュアルやインストール CDはあちらこちらに散乱．私のギターも結構そんな物かも知れない...

リッケンバッカーはどこへ行った

(1994.10)

あのリッケンバッカーはどこへいったのか．そしてどうなったのか．私はジョン・レノンの使っていたのとよく似ているリッケンバッカー350JG を昨年アメリカへ行ったついでに買ってきました．日本で買うと非常に高価なギターを弾きづらいなどといって，改造を計画しながらも果たせず，粗大ゴミ化していたのですが，ついに売ってしまったのです．

で，新しく買った(というよりは新品と交換した)のはというと，これがまたリッケンバッカーで650Atlantisというギターです．これは，色，デザイン，弾きやすさどれをとっても秀逸でわたしのメインギターとして活躍しています．価格は350の方が少し高いのですが(だから新品と交換できた)，650の方を非常に気に入ってます．

では、650Aのスペックを御紹介致しましょう．

ソリッドボディ
24フレット
スケール長: 24.75インチ
ナット位置でのネックの幅: 1.75インチ
12フレット位置でのネックの幅: 2.0625インチ
ボディ材: メープル(楓)
ネック: メープル2ピース
　サイドバインディングなし
　スルーネック
コンビネーションテールピース
6サドルブリッジ
2ハムバッキングピックアップ
ロトマッチクペグ(シャーラー)
ボディ: ターコイズブルー

弾いてみた印象は，全体的に造りがしっかりしていて高級品という感じです．

肝腎の音はというと，アンプとの相性もありますが，普通に設定すれば，ギブソンほど太くはないが，フェンダーほど硬くもないという感じです．このようにいうと，どっちつかずとの印象を与えてしまうかも

系は黒に近い色なので，ギター全体の印象が落ちついた感じになる．当然，ハデ好きの私はメイプル指板が好きである．私のギターのほとんどはRickenbacker 650，Fender Telecaster，Fender Stratocasterいずれもメイプルワンピースネックである．

ネックの長さと弦の太さ

ネックが長いということは，弦長が長くなるということなので，同じ音を出すためには，張力を大きくしてやる必要がある．そうすると，同じ太さの弦を張っても，押えるときに硬く感じる(力がいる)ということになる．これもギタリストの好みで，パンパンに張った硬いのが好きなのと，やわやわが好きなのがいる．

音質からすれば，ネックの長いギターは，弦張力が大きい分硬い音になる．ネックの短いギターは，その逆で，暖かい(軟らかい)音になる． long scale，medium scale，short scaleと標準的な3種類の長さがあるが，私はlong scale のギターがその音質からして好きだ．

同じ長さのネックに太い弦を張り，同じ高さの音を出そうとすれば，当然張力を大きくしなければならない．現在，もっとも一般的な弦の太さは， 1弦で直径0.009インチ6弦で0.042インチだ．このセットを0942(ゼロキュウヨンニイ)と呼ぶ．この太さの弦を技術的に製造可能になったのは， 10年ほど前で，その前は1弦で直径0.010インチ6弦で0.046インチの弦が標準的に手にはいるもっとも細いセットだった．

昔ながらのギタリストは，過去の弦を作る技術が未熟な時代をひきずっていて，「太い弦(1046のセット)の方が弦振動でボディーを共鳴させやすい.」などと仰るが，本当かどうか疑わしいものである．だって，高々0.001インチの太さの違いで，何か変わるのだろう．それはさておき，音質は太い弦の方が硬い音になる傾向がある．

さてこの項のまとめ．私は，long scaleのギターに0942の弦という組合せで，音的にも感触的にも満足している．

フレットの形状

フレットの形状は音に大きな影響を与えると同時に，弾き易さにも大きな影響を与える．尖ったフレットの方がシャープな音質が得られる．しかし，尖ったフレットの場合，弾くときにどうしても引っかかる感じがする．尖ったフレットはやはりコード弾き用のもので，リードを弾くためには平たいフレットがより向いていると思う．こんなことから，Fender のTelecasterには尖ったフレットが打ってある．

私は，平たくもなく尖ってもいない，中間的なJim Dunlopのフレットが好きだ． Rickenbackerの650に打ってあるのがそれで，音の立ちもそんなに悪くない．

ピックアップの種類

ピックアップには，シングルコイルのものとダブルコイルのものがある．電磁石が1本で構成されたピックアップか2本で構成されたピックアップかの違いだ．ダブルコイルのピックアップはハム信号(雑音の元)を打ち消すために2本のコイルの位相を逆にしてある．このことから通常ダブルコイルのピックアップは，ハムバッカーとかハムバッキングピックアップとか呼ばれる．

シングルコイルのピックアップはシャープさが際だった(エッジが立ったという)音を出すが，ハム信号を拾い易く雑音が交じり易い．一方ハムバッカーは雑音は拾ってこない代わりに，音のエッジの立ち方はシングルほどではない．私はシングルの雑音の多さに悩まされ，一時期ハムバッカー狂になっていたが，今はそれほどではない．

バッキングのときはシングルで，ソロでリードを弾くときはハムバッキングを使う．フロント(ネック側)にハムバッカー，リア(ブリッジ側)にシングルをマウントしたギターを気にいっている．またタップといってハムバッカーの2本あるコイルの内， 1本だけをシングルとして使うことが可能なピックアップもある． Rickenbacker 650もFender Telecasterも，私所有のものに限ってはこの種の改造が施してある．

ピックアップの位置

同じピックアップを使っても，それをマウントする位置によって音質が変わる．フロントは軟らかい音リアは硬い音がでる．多くのギターはスイッチで切替え可能でフロントだけリアだけ，または両方の選択ができる．私は，フロントのハムバッカー，リアのシングルの音が好きだ．

以上，ギターについていろいろと書いてきた．最後に私のギターコレクションを紹介して終りにする．

エレキギター

Rickenbacker	650A
Fender U.S.A.	Stratocaster
Fender U.S.A.	Lead I
Fender Japan	Telecaster
Gibson U.S.A.	Les Paul Studion Lite
Gibson U.S.A.	Les Paul Special
Kramer	
Amboy	Stratocaster
COBRAN	FG2
Jackson	Dinky
Order Made	Telecaster
Order Made	Telecaster
Order Made	Stratocaster

Greco	Telecaster
エレアコ	
Takamine	NTP-110
Fender Japan	Telecaster
アコースティックギター	
Martin	D-1
Aria	D-80

いつからちゃんとプログラムが書けるようになったのか．自分にできることは誰にでもできると思ってしまいがちではあるが，確かにソフトウェア創りは誰にでもできることではない．ギターを弾いたり歌を歌ったりすることも，人よりはできるのだが，どのようにしてできるようになったのかはよくわからない．それを敢えて考えてみた．

ギターを弾いたり歌を歌ったりすることは

(1996.10)

僕はスーパーロックギタリストではない，目茶目茶歌がうまいわけでもない．でも，両方とも死ぬほど下手なわけでもない．僕ぐらいの腕のギタリストはここにも，そこにも，あそこにもいる．でも，自慢になるけれど，１０人ロックギタリストをつれて来たら，その中の７，８人より，僕はきっとうまいだろう．

僕は，昔からギターがうまかったわけではない．とりわけ，エレキ弾きに変身したのは，３５歳である．ギターを弾きはじめたのは１２歳の時だから，もう２６年もギターを弾いてきたわけだが，最初の２３年はフォークギターを弾いて来た．ジャカジャカやるコード弾きやフラットピッキング，スリーフィンガー，アルペジオ，これらのことには，その道２０数年の技を持っていた．しかし，リードギターなんていうのは，ほんの２，３年前にはじめたばかりである．でも，うまくなったのである．たった，ここ２，３年にだ．

歌をうまくなりたいと思ったのは，もっとひどく，ここ１，２年のことである．とくに誰を師事したわけでもないのに，その短期間に，ある程度聞ける歌が歌えるようになった．これらのことを可能にしたのは，自分の才能のなせる業と自惚れるのは楽なので，少し謙虚になることにして，私に何が起こったのかを考えてみた．

天啓というか，その類の事実をあなたは信じますか．実は，そんなことをここ数年で数度といわず僕は経験している．いや，決して，わけのわからない宗教を持ったからではない．自分では説明できない体験があり，それは幻ではなく確かに自分の身に起こったことなのである．

たとえば，リードギター．リードギターというのは楽曲の前奏，間奏，最終奏でメロディーを弾く役割が与えられているのだが，これがなかなか難しい．ビートを外すと聞いてられなくなるし，また，即興でメロディーを創り出すことが多いので，音外しが曲を駄目にすることもある．

だからといって，考え過ぎたり慎重になり過ぎと，面白くもないフレーズを連発することにもなる．いくつもうまくいかない理由を考えて，それを数え挙げることは簡単なのだが，それらを矯正する方法は練習しかない．しかし，いくら練習しても，それでライブやレコーディングでうまく演れるかというと答えは否である．では，何が問題なのだろう．

「自分の音を恐がる」という現象がある．これは，エレキをアンプを通して鳴らすとその音の大きさに驚いて，ギターをそろそろと撫でるように弾いてしまい，それが，ビート外しや音外しのもとになり，さらにそれが，超そろそろ弾きを誘発するという負のフィードバック現象のことである．

実は，初心者からの脱却はこの自分の音を恐がっている自分を発見することから始まる．そして，自分が成長していく節目節目で自分の音を恐がっている自分を発見するのである．それを誰に教えられたわけでもないのに，突然振切れるのである．そして，それ以降のしばらくの期間は，それまでのことが嘘のようにカッコいいギターソロを弾ける自分を見ることになる．

実は，このように振切れた瞬間から正のフィードバックがかかり，自信を持ってギターを弾く自分を発見することになる．しかし，これは，練習量とは線形関係にないようである．自分の経験からして，不連続な進歩であることはどうも疑えない事実であるようだ．

僕は，数年前にEaglesのTake it easyの間奏を弾いているとき，２年前にJimi HendrixのStone Freeを演っているとき，そしてつい最近，自分の創った曲の間奏を弾いているときの，合計３回は，少なくともその天啓を受けたものと思っている．

歌心を理解するとはどういうことかおわかりだろうか．その歌の心を理解した瞬間から人の心の琴線に触れる歌が歌えるようになる．

最近，Ben E. Kingの名曲Stand by Meを練習しているときにわかったような気になった．それまでは，その曲は，原曲とはアレンジを変えて，カントリーの曲として演奏してきた．その間は，それなりのできにしかならなかったのだが，僕のヴォーカルがある日突然説得力のあるものになった．つい最近の練習のときにである．

そのとき僕はこう思った，「このアレンジは彼に対して失礼である」と．それまで，幾度となくバンドのメンバーがそういっても耳を貸さなかった僕が，自分からそう思ったのである．結果として，その歌は以前の何倍ものできを誇るようになった．

この予感は，この夏Eric Claptonの Wondeful Tonightを歌っているとき，そして春，Rolling StonesのAngieを Jimi HendrixのLittle Wingを歌っているときに感じたものでもある．でも，そのときにはそれが何なのかはわからなかった．

そして，ごく最近Stand by Meを練習しているとき突然に，誰からいわれたわけでもなく，歌心がわかるとはどういうことかわかってしまったのだ．

かように，楽器演奏，楽曲唱和は無駄な努力が必要な種目である．そして，一見無駄に見える練習は実は，あなたに訪れる天啓のための貯金なのである．僕は，一見無駄に見えるような練習を重ねないとうまくなれない楽器演奏のような，ストイックな行動が好きであり，そんなことを趣味にしている自分をなぜか誇りに思っている．

なかなかバグがとれない，どうしようもない．そんなことと同じことがギターを弾いているときにも起こるのです．

リズム感養成のための練習

(1997.10)

また音楽の話．

どうもリズム感が悪い．そう思い始めたのは，タイトな曲をやり出してからだ． Stevie SalasのThe Harder They Comeという曲があり，それをバンドでやりだしてからのことである．

最初は楽しいだけで始めたバンドが今やかなり高い水準を維持しようとしだして，結構大変な毎日を送っている．その曲は分類するとダンス音楽(ダンスといっても社交ダンスじゃなくて，ディスコ関連だからね)とハードロックの融合形のようなもので，一般にはfunk rockと呼ばれるものだ．音楽的には，１６ビートのリズムに３度音省略のパワーコードその上に主音の単調音階上の音から構成されるメロディーを載せたものと，大雑把にいえばそうなる．

その曲のイントロや中イントロでドラムのハイハットとギターのリフが絡むところがありそこがどうしても弾けない,というか,もたったり走ったりする．バンドのメンバーから,「野呂,こんなことできないんならもうバンドやめたら.」などと厳しい批判を受けながら，負けてなるものかとリズム感養成に乗り出した．

リズム感養成のための練習のもっとも古典的なものはメトロノームを使ったものだ．メトロノームにあわせてメロディーなり伴奏なりを弾く．この方法は最初のうちはかなり効果があるのだが，ある段階まで来ると壁につき当たってしまう．

理由はいろいろ考えられるがもっとも大きな要因はメトロノームは拍の頭を押さえる機械音しか出してくれないことにあると思う．もう１つの理由はバンドの持っているグルーブは機械の出す正確無比なビートとは違うものだということ．ドラムシンセサイザなんかを使った打ち込みといわれる伴奏のCDはタイトで正確なリズムの上で演奏されたものだけれども何か無味乾燥な印象を受けるのはこれによるものだと思う．

人間が叩くドラムはテンポが揺れながら(早くなったり遅くなったりしながら，もちろん少しずつだけど)ビートを出すので，特有のグルーブ感をつくり出せる．そのグルーブをバンドの中で共有できるかどうかがロックバンドにとっては非常に重要なことになる．自分だけがon timeに正しいビートを刻んでいることだけでは決して褒められたものではない．

ものすごくうまいstudio musicianやsuper musicianが集まってCDを作ったりすると,例外なくどこか物足りないものになるのはバンドとしてのグルーブを共有するだけの時間を持てないからだろう．しかし,いずれにしてもメトロノーム相手にタイトにビートをつくり出す練習はリズム感養成の第一歩であり，それ抜きではつぎの段階へは進めない．

よいドラマが人並外れたリズム感を持っているのは当然だがどのようにそれを作ってきたかを知ることは自分のリズム感養成に重要なことである．ドンカマと呼ばれるメトロノームのお化けような機械を使って練習に励むドラマの姿は通常は目の当たりにすることはないけれども，うまいドラマはこれを欠かさない．

メトロノームを使った先の練習と異なるのは両手両足違うことをやりながら連動させる訓練を怠らないことだ．これをギターの練習にとり入れることが，ギタリストのリズム感養成の次の段階である．左足で２拍子，右肘で４拍子,右手の刻みを８拍子でやるという訓練をメトロノーム相手にやる．これが基本で,それができれば刻みとして１６分音符を入れたり３連符を入れたりと変化形を練習する．これを継続的に練習することが重要である．

たとえ話をするなら，自転車に乗る,自動車を普通に運転するなどの一旦覚えたら日常的な行動として繰り返せばそれで忘れない程度の水準では一流のギタリスト(musician)の持つリズム感は維持できないばかりか届かなくなる．また,続けてないとできなくなってしまうというのが頭痛の種である．

話は横道にそれるが,何をやっても継続的な投資が必要であるとわかった．以前，本気でゴルフに取り組んだことがあり，１ラウンド８０そこそこ，たまに７０台で回れるまでになった(これはかなり自慢です)．もうやめてしまったけれど,その頃もいつもどうやって球を打つかということが頭にあり継続的に素振りや練習場での打撃練習を続けないとそのスコアが維持できなかった．１００を切ったり９０台で回ることは，どこのオッサンにもできることだし,練習などしなくても誰にでもできることだ．

同じことが音楽にもいえて,ギター１本で下手な弾き語りをすることは誰にでもできるが，そんな奴が何人集まったってろくなバンドはできやしない．せいぜいできたって長渕剛程度の音楽だろう．そんなレベルの低いことをやりたくって僕は音楽をやっているんじゃない．こういうことをいうのは尾崎豊やX-Japanを批判するのと同じで非常に勇気のいることだが，駄目なものは駄目なのでいってやらないとね．

さて,ここまで来てもまだ例のイントロが弾けない．これまでに紹介した練習で他の曲ではかなりの進歩が見られたのにである．メトロノーム相手には完璧にこなせることが生きたドラマー相手だとできない．「何なんだこれは？」という状態がしばらく続いた．

いろいろ悩んだ末わかったのはハネ系のビートを自分の中に持っていないということである．バンドのもう一人のギタリストがこれを教えてくれた．コーチの重要性をこれほど感じたことはなかった．ハネ系のビートとは,３連符の最初の２つと最後の１つを分けて弾くことを基本にしたリズムで，跳ねた感じがするのでその名がついている．さて,それからハネ系のビートを獲得するためにメトロノームとリズムマシン相手の練習を始めた．しばらくするとあら不思議,あれほどまでに困難を究めたものがすんなりとできてしまった．

ここでわかったのは,「感じ」すなわち体でビートを体感することの重要性である．要するにリズムとは体で感じるものだということを体現しただけではあるが,その道のりは私にとって半年強にも及ぶものであった．

この分じゃブルースの「ため」や「外し」が体に染み着くまでには当分かかるのかも知れない．生きているうちにそこまで到達できるのだろうかと不安にもなるが，人間あきらめなければ何でもできると信じているのでその道を行くのみである．だって,通常は起こり得ないような進歩をまたこの半年で体験してしまったのだから．そして私はさらなる飛躍を目指してドラムの練習を始めた．ちゃんと仕事やってんのかね？

CDをリリースしてないバンドなので詩を披露することは少ない．この場が適切だとは微塵も思ってはいないが...

おやじの詩
(1998.10)

今年，６月親父が死にました．それで，この歌を作りました．

そんなものは潰してしまえばいいと
その一言でずっとすべてを切り捨ててきた
強いだけの記憶を残して
あなたは死んでしまった

何かが変わってしまったと気づいてから
早過ぎる程死に急ぎましたね
自分の身には不幸なんて起きないものと
人はなぜかしら思い込むものだから
強いばかりの人だったから
悲しさは余計に僕たちを追い込んで行く
今になってもなぜこうなったのかと
顔を見合わせるだけです

動けなくなっていく自分を
どうやって認めてしまったのですか
あのときあなたが諦めてしまうまで
僕たちは少しだけ辛い思いもしました
わがままばかりの人間だったから
さみしさはいつまでも僕たちと一緒にいる
あなたがいなくなった家で
経ち尽くすだけです

いやという程殴られた夜に
どう仕返すかばかりを考えました
自分の場所があなたのいたところだと知って
あなたが何を教えたかったのかがわかりました
こうやって歩いて来た結果
あのとき一番嫌いだったのになぜか
あなたのように生きています
なぜかはわからないまま

少しは感じていたけれど
本当に喜んでくれてたんですね
あなたの自慢になるようなことも
僕はいくつかやってきたんだろうか
いつも不愉快そうだったけれど
子供たちを見て嬉しそうに笑っていましたね
あなたが祈ってくれたことに
感謝するだけです

もう会えないということを
まだ素直に受け入れることはできません
でも僕たちは少しづつ元気になってきました
忙しさも少しづつ戻ってきました
今僕にできることとして
こんな詩を一つ作ってみました
そしてあなたの残してくれたものを
大きく育てていきます

さてもう一つ．なぜ２・２６かって，２月２６日に創ったから．

２・２６
(1999.10)

俺たちのLiveに来てごらん，こんな歌が思いっきりのRockで聞けるんだぜ．へこんだ時は元気に，とんでる時はそのまま...

楽しい話は　ありませんか？
毎日が　退屈なんです，あなたもでしょう

変わったことが　できませんか？
誰かの後　ついてゆくだけで簡単にね
力がないから酒の力を借りる
それは昔のこといや今もそうですよ
からっぽの　心の隙間に
遠い昔の　古いメロディー
それとも君の好きな　流り唄

悲しい話は　ありませんか？
涙が出て　止まらないほど悲しい話
変わったことは　ないけれど
不幸な噂に　群がる奴はたくさんいますね
つめたい雨を降らせてやると
昔のセリフをまだ覚えているよ
やるせない　怒りのやり場は
いつもの　安い居酒屋
それともヘラヘラと　笑っていましょうか

あぶない話は　ありませんか？
あいつの首　とばせるような，でかいやつがね
思い通りに　やってみたい
あいつが邪魔　そんなことよくあるでしょう
心にもない笑顔は作らず
天にかわって刃をひとふり
許せぬものは　切って捨てると
勇ましく　叫んでみましょうか
あなたは狂気の沙汰と　笑いますか

「インターネットする」とか「電子メールする」とか，わけのわからない日本語を使って，ついでにファッションでコンピュータを使う奴らよく聞け！

CDデビューは何時のことやら

(2000.10)

吉田拓郎はやはりかっこういい．いつになっても「今」である．「今時の若い奴は」なんて死んでもいわない．「がきになんてわかってたまるか」という言葉は「大人になんてわかってたまるか」にくらべれば，はるかに軽くてどうでもいいことなのである．拓郎の言葉で今でも私が信じて疑わない．音楽上の拓郎の成長も見逃せない．なかなかできないことを誰にでもできそうにやってしまうのが拓郎のいいところであり，すごいところだろう．

B'zはいまや飛ぶ鳥を落す勢いである．発売したシングルは２０数曲連続１位獲得である．あのサザンもなし得なかったこと，もちろん長淵のバカタレなんかは遠くおよばない．日本のロックが本当の形で花を咲かせたいい例である．確かに彼らの曲は何が元になっているか，TAKのギターは何をイメージして弾いているのか，すぐにわかってしまうけれど，歌もギターも自分のものとして恰好いいんだ．

山崎まさよしには強烈な独創性を感じる．歌もうまいし，ギターもなかなか，彼の音楽には尊敬の念を禁じ得ない．ブルースをわかり易い形で日本のものにしたいい例である．

「ゆず」は，この意味で現代版２人拓郎だろう．秀逸なメロディーにピッタリの詩を載せる才能は敬服に値する．「さよならバス」という歌には，私たちが経験してきた恋愛の悲しい美しさが本当にさりげなく表現されている．「ともだちのうた」を聞いて，友情をどうしようもないサムい形ではなくて，素直に受け入れられる形で表現していることに本当に感心した．ゆずは長淵のファンであり長淵は拓郎のファンであった．長淵のバカはおおいなる勘違いのはてに軽蔑に値する存在に落ちてしまって，死ぬほどサムい阿呆になってしまったが，ゆずはいい形でフォークソングのよさを今にしている．

拓郎とおなじくらい今っぽくてかっこういいのが清志郎である．RCはあのとき，確実にかっこうよかった．日本のロックがみんなのものになる少し前のあの時代，私は清志郎をずっと追いかけていた．そして彼もずっと今である．

さだまさし，谷村しんじ，かぐや姫，甲斐よしひろなど，昔になってしまったのに，わかっていない軽蔑すべき勘違い野郎が多い中で拓郎と清志郎はやっぱり今なのだ．

ここ１０年位日本の音楽は聞かずに過ごしてきたけれど，B'zや山崎のように確実に日本にもいいアーチストがいることを最近知った．エレファントやHigh-Lows，相川やイエローモンキーなど本当にちゃんとした奴らがいることを知った．

でもでも，何か違うのだ．今時の若いものはなんて僕は死んでもいわないけれど，今時のロックをファッションでしか聞いていない奴は仕方ないと僕は思うのだ．この前，B'zのコンサートに行ってわかったことがある．山崎のコンサートビデオを見て同じことを感じた．ロックに敬意を表していない馬鹿な奴しか居ないのである．私だけがB'zをわかっているとはいわないけれど，B'zがなぜあれだけかっこういいか，何がGrayと違うのか，といったことをきっとこの人たちはわからないんだな，というパーが多過ぎるのだ．

稲葉が歌ってるときにはキャーキャー騒ぎ，TAKがギターを弾くと押し黙るという楽曲の聞き方はやめて欲しい．めちゃくちゃうまいギターや歌が聞けたときに思わず立ち上がるのがロックを聞いていることであり，最初から立ち上がってどんなステップを踏めるかを競うことが，"私は知ってるんだぞ,"と知識を誇示することが音楽を聞くことではないことをわかって欲しい．郷ひろみや田原俊彦，ラルクのおっかけとは違うんだから．五十歩百歩との指摘は受けることを覚悟でいってしまったけれど，本当にわかってないのである．

挙げ句のはてに「ロックだねー，チョーカッコいい

じゃん」とか，「ブルースって，いいね」なんていわないで欲しい．何もわかっていないくせに．若い奴は駄目っていうんじゃなくて，そんな外観だけしか見ない奴が駄目なんだ．演る側になるとそういうのがたくさん来てくれて，きゃーきゃーいってくれるのはやる気の元だからいいけれど，その程度の人間相手にしか金にはならないそんな音楽市場に日本はなってしまったのだろうか．

しかし，この問題意識は我々のバンドが，その演奏の巧拙をさしおいても，なぜ客集めが下手かを如実に物語っているものかも知れない．そして，私の最終兵器かつ最大のヒーロー Stevie Salasが駄目になってしまったのも同じだろう．私たちのバンドは男っぽいロックとか骨っぽい音楽を目指しているし，そう評価されることが多い．そしてバンドのメンバーもみんなそれをよしとしている．しかし，こんなことでは，録音はできても，インディーズでさえもデビューはできないかも知れない．ルックスにも問題ありだが，考え直す必要がありそうだ...なんて．バーカ！ 絶対考え直してなんかやるもんか．

何人か集まって1つのことをやるのは難しい．バンドを続けて行くことも...

もう少しのところまで行ったのに

(2001.10)

「大人になんてわかってたまるか」という言葉は「ガキになんてわかってたまるか」なんて言葉よりは，はるかに重い言葉なのだ．このことを身をもって体験した．そして，私たちのバンドは解散してしまった．

何がもう少しだったのかって，それはインディーズだけどデビューすること．随分真剣に音楽をやって来た甲斐あって，今年の夏からようやくハードロックカフェ名古屋でのレギュラーライブの約束ができ，実際に2回ライブをやった．1度目は少し失敗したんだけれど，2度目は大成功．ライブ後にお客さんから，「よいバンドですね」とか「カッコよかったですよ」，「CD出してないんですか」など励ましの言葉を貰って結構嬉しかった．ちょうど，日本テレビの24時間テレビの企画のひとつとして，"Hard Rock Cafe Nagoya All Night Live" が催され，その出演バンドのひとつだったんだけれど，これまでの多くのライブのどれよりもお客さんを，それもフリの客を湧かせられたというのに．やはりというか，結局というか，よいことは続かないもので，私たちのバンドT-Topsはもうない．

思えば，何か嫌な感じは，夏を迎える前の練習の頃から，メンバーお互いの中にあったような気もする．なんとなくうまく行かないという感じは，ハードロックカフェでのライブの打ち合せをお店の人とし始めた頃から，はっきりとした形になって来た気がする．私たちは皆，ハードロックカフェでのレギュラーライブをひとつの到達点と考えていた．そして，10年来の仲間として新しい出発のきっかけにし，これまで少なからずあった諍いや，わだかまりを総決算しようと考えていた．少なくとも私はそう考えていた．

10年前私は33歳，他のメンバーは19歳，23歳，24歳．私は先生，彼らは学生だった．「俺のこと先生と呼ぶのは止めにしようぜ」で始まったつき合いだった．うまくやって行けるはずだった．そのときは，お互いの絆や創って行きたい音楽について本当に素直に話せた．話すことといえば，ほぼ音楽のことと，どうやってバンドで自分たちの夢，すなわち，ミュージシャンになること，をどう実現するかを話し合っていた．

日常の生活，大学の先生であることや家族のことは，何時もどこかに置いてしまって，練習やライブをやっていた．喧嘩になることもたびたび，でも4人がバラバラになると何もできなくなることをわかっているかのように，しこりを残さず問題を解消して来た．しかし，メンバーの1人が結婚し，勤め先で偉くなるにつれて，お互いが違和感を抱くようになって来た．そして，その違和感は解散という最悪の形でしか解消できなかった．それは何だったのだろうか．

バンドをやったことがある人ならよくわかるだろうけど，最初は誰でも，「面白い，楽しい，それでいい」から始める．この時期はメンバーが誰でないと駄目とかなんてことはなくて，ドラムとベースとギターとヴォーカルが揃っていればよい．何回かライブをやることも機材の搬入も撤収も自分たちでやることに何のためらいもない．ライブでもミスだらけ，毎回反省することしきり，落ち込んではまた練習次のライブと繰り返す．たまにうまく行くと大騒ぎオールナイトパーティーで朝まで酒飲んでギター弾いて，遊び疲れて寝る．こんなバンドとしての経験を重ねて行くうちに，お互いの嫌なところが見え始める，

でもそれは実は，お互いの演奏がバラバラからバンドとしてひとつになって行く過程で，人間関係が近くなりすぎて起こること．演奏がうまく行けばすべて解決，なんていうことになる．この頃になると，ライブで極端なミスは しなくなる．そしてついに一体感ができてしまい，ベースだけメンバーが入れ替わるとか，ギターが代打になるとかすると，演奏の一体感を守ることができなくなって来る．でも，一体感を感じ始めた頃でも，曲に対する理解や思い，表現したいことはバラバラ．

それが，そのうち自分の役割がわかって来る．そして，曲の表情のつけ方がよくわかって来るようになる．私たちはここまで来るのに7，8年かかった．この頃は，もうライブでも結構余裕．どうやってお客さんを乗せるか等を考え出す．一旦，バンドのカラーと

か匂いとかができあがってしまうと，今度はそれをお互いが壊しにかかる．ロックはある部分インプロビゼーションの勝負だから，お互いが，「あいつ次はこう来るな」と思うことを裏切ることに楽しみを感じ始める．

もうあんまり喧嘩もしない．お互いのわだかまりが演奏を駄目にすることを，メンバー全部が承知しているから．お互いが背負っているもの(仕事や家族)をバンドの仲間と集まるときは，何時も置き忘れて来るうちはどんどんバンドがよくなって行く．これは，互いのお約束であって，これを止めるととたんにバンド内の人間関係がギクシャクし始める．私たちは，これまでずっと背負っているものを降ろしてから，バンドの仲間と会うようにして来た．少なくとも昨年ぐらいまでは．

私たちは，練習のたびに「ミーティング」と称して，午前零時の遅い食事をとりながら，ライブのことやこれからのことを話していた．ミーティングのときも，家に置き忘れて来たものを思い出すことはない．しかし，何時からか，お互いが，仕事の辛さや愚痴，自らの地位，家族のことをあるときは相談っぽく，あるときは自慢たらしく話し出すようになった．そして，ついには，「何時までもガキみたいにロックばかりやってられないな」と一人がいい出した．

これは話が違うんじゃないかと少なくとも私と一番若いもう一人は思った．そんなボタンの掛け違いも，しかし，ライブでうまくやってしまえば何時でも吹き飛ばせたし，今度もそうできると思った．そしてレギュラーライブの約束を取りつけた夜のミーティングでいった私の一言が引金になった．

「色々考えも違うけれど，お前らと会えたことを俺は本当によかったと思う．お前らは，普通なら偉そうにしてればそれですんでる，この俺に，人間には自分が自分に時間を投資してやらないと絶対できないことがあることを教えてくれたから．十年前の関係はもう取り戻せないかも知れないけれど，これからも子供でいよう．バンド続けよう．CDも創ろう」．これに，ガキみたいにロックばかりやってられない，といったメンバーが反応した．「何度も同じことをもういいよ．何時までも子供じゃいられないぜ．嫁さんもできたし，今度は家も建てるんだし．仕事も俺が切り回さないと駄目になって来たし．何時までそんなこといってんの，野呂」

これで，おわり．

そして，冒頭の言葉を思い出した．私は一生ガキでいることに決めた．だから，もう，こんな話をする奴とは，彼がどんなに優れたドラマーでも一緒にはやって行けない．でもそんな中で，"Hard Rock Cafe Nagoya All Night Live" は成功させることができた．もう一度，かの彼と話をしなおうそうかとも考えた．しかし，私は，壊すことを選んだ．一番大事なものを守るために，大事なものを捨てた．

夢はまだ夢のまま，さあ，またメンバー探しだ．新生T-Topsを作るぞ！ 私は絶対 諦めない．「大人になんてわかってたまるか」という言葉は「ガキになんてわかってたまるか」なんて言葉よりは，はるかに重い言葉なのだ．

ロックを芸術にしてしまったらいかに面白くないかは Deep Purpleを聞けばよくわかる． Jeff BeckやJimi HendrixそしてEddie Van HalenやStevie Salasは様式美を追求しない． Jimmy Pageは無機質な構築物として音楽を捉えているけれど，様式美を追求してはいない． Bluesの持つ泥臭さを消し去ってはいない．ソフトウェアを高尚な芸術作品と考えるのではなく，ソフトウェア創りを高尚な学問にするのではなく，軽音楽と呼ばれるロックを愛するように，「動いてなんぼ」の構築物と捉えてつき合って行きたいと思う．

旅の名残

熊谷 章
(PFU)

１年振りで革ジャンを着たらポケットからすべすべした白い石が出てきた．左のポケットにそれはあった．

エンジュの枯れ葉が舞い落ちている並木道を歩きながら，どこで拾ったものかを懸命に思い出そうとする．ゴビ砂漠，張家界の渓谷，敦厚など中国の景色と共に種々の場所が頭に浮かぶ．この革ジャンは１年ぶりだとすれば，海外に行っているはずがないと思い当たる．と，ここまで推論したときに，はっと思い付いた．土佐の高知の桂浜に違いない．２月に，岸田，中野，中島の４人で訪れたのだった．２７０度全部海の桂浜で波打ち際に寄せられた石の中から気に入ったのを拾ったのだった．

この白い石をシンボルとして様々な記憶が蘇る．不思議なものだ．記憶はシンボルがないと取り出すことができないようだ．いまの場合，手にした白い石がシンボルとなり，契機となり，過ぎ去った旅の名残を懐かしく思い出し，かつ触れることを可能にしてくれる．

先ずは，バーのハチキンママ．中野さんの知り合いが行きつけの店なのだが，このママが名物である．晩飯で食べ少々酩酊気味で行ったので，いきなりスピリタス９６を頼んだ．そしたら，生憎今日はないという．仕方がないので，ウイスキーをストレートで飲んだ．その内，ママがニクラシカで乾杯しようと言い出した．当然，OKと言ったらママが何処かへ消えデカンタに透明な液体を入れて戻ってきた．なんとスピリタス９６を他店から入手してきたのだ．レモン，砂糖とブランデーの代わりに塩，レモン，スピリタス９６でニクラシカの乾杯競争をやろうというわけだ．後に引き下がる訳にはいかない．

輪切りのレモンに上に塩を載せ，それをダブリと噛み，すかさずカップ一杯のスピリタスを呑み込むのだ．レモン，塩，スピリタスの刺激で口の中は燃えるようだ．流石にハチキンママは強い．２～３回乾杯してもビクともしない．もっとも，ハチキンを自慢するだけあって身体は丈夫そのもので，背丈も１７５センチ位はある．楊貴妃も位負けするほどの威勢の良さだ．この狂気の乾杯に参加したのは，ハチキンママ，岸田さんと俺である．岸田さんは最初の２杯で抜けた．残るは２人．５杯まで空けたことは覚えているが，その後から翌日の正午近くまで何も覚えていない．一切の記憶が飛んでしまった記憶しかない．というのは，正に言葉の彩である．論理的には，矛盾した表現であるが意味が通じるから面白い．

気が付いて見ればベットの上に横たわっている．セーターは脱いでポロシャツ姿でズボンをはき靴は脱いでいた．枕元の電話がしきりに鳴っている．一瞬，ここはどこ，俺は誰だ，なぜここにいるという状況に陥る．とにかく電話に出る．向こうで聴き慣れた声が喋り出す．「実は，今朝８時半にロビー集合ということで昨夜別れたのだが，いまはすでに１１時だ．ずっと待っているのだ．早く降りて来い」という岸田さんの呆れた声だ．しかし，俺は即座になにがなんだが判らない．「はい，分かりました」といい，また，眠ってしまった．また，電話が鳴る．「まだ，起きてないの．もう，１２時だよ」

おぼろげながら覚悟を決め，下着とポロシャツを着替え顔を洗ってからロビーに行く．岸田さんと中島さんがニコニコしながら，諦めきった表情で迎えてくれた．即座にチェックアウトをフロントで済ませた．何時かと時計をみたが腕にはない．失くしたかと思ったが，顔を洗うときに洗面所に置いたような気がしたので，フロントで部屋のキーを受け取り部屋に戻る．時計は洗面台にあった．安心してロビーに戻ったら，中島さんが笑みで顔がほころびそうになっていた．「どうしたの」と訊いたら，「くまさん，財布をカウンターに忘れたでしょう」という．ウーーン，危ない．俺は何をやっているんだろう，と自責の念で一杯になりながら，カウンターで無事財布を手にした．このように一応チェックアウトができたが，心身の調子がどうもいかれている．一言でいえば，「宙に浮いている感じ」である．

新阪急ホテルを出て市場に行く．馴染みの果物屋「松田青果店」で文旦を買う．水晶文旦と露地も

のの両方を求め，鎌倉と秋田に送った．中島さんにも薦める．たがが緩んでいるので歯止めが利かなく，いつもより高価なものを沢山買ってしまう．しかし，この選択が正解だった．後で秋田からも家からもとても美味しいとの大好評を得たのだった．ここまで書いてきて，この場にもう１人誰かがいたような気がした．それは多分幻の怪人だったに違いない．記憶とは一体どんな構造とメカニズムをしているのか，摩訶不思議なものだ．

市場の中で昼飯を食うことにした．名の通った「○○屋」だ．美味しそうな肴を選び，酒も注文した．しかし，何も食えないし，何も飲めない．どうしたことか．こんな事は人生始まって以来２度目だ．最初のはハワイでの「PWBワークショップ」だった．「ホテルイリカイ」の８０５号室に滞在したとき，１週間毎日ウイスキーを２～３本ずつ数名で空けて暮らしたときのことだった．皆途中で脱落して，最後の夜は俺と小笠原さんの２人だけになってしまった．それでも，飲み続け俺ら２人はコップを手にしながら眠っていたのだった．翌日の午後は皆でハノウマベイに繰り出すことになった．遅いブランチを摂るために皆でレストランに入った．このときがそうだった．一切の固形物を受け付けず，スープを少し飲めるだけだった．しかも，スープを飲むスプーンがアル中者のように震えていたことを思い出す．こんなはずがあるか，と努力するが心身を制御することができない．仕方がないので，酒を舐めるだけにした．昼食をみなさんが美味しく食べるのを眺めた後市場で買物をした．「中村鰹節店」で，一枚５千円位のだるまするめ，しらす，アジの干物などを買った．これらはすべて殊のほか美味しかった．

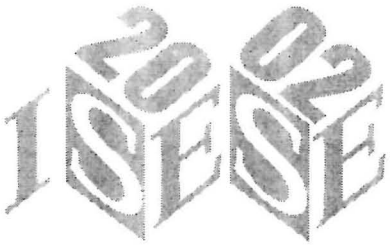
その後高知市内の美術館に行った．水をテーマに設計された建物の美しさと喫茶店で飲んだ松脂の味がするお茶が記憶に残っている．やがて，空港に着いた．予約していたのでカウンターで名前を告げた．しかし，予約がないという返事に困惑した．「そんなはずはない．インターネットで予約したので，ちゃんと調べてくれ」と言った．そしたら，予約がありましたが，昨日の日付でしたという．こんなことが起きるはずがない，とブツブツいいながら正規料金の切符を買い目出度く搭乗できた．インターネットの予約システムのインターフェースが悪いのだ，という結論にし自分を慰めた．飛行機が高知空港を飛び立ち，ようやく正常に戻った感がして安堵した．

しかし，世の中そんなに甘くはなかった．羽田に着いたら東京地方は大雪に見舞われていたのだ．俺は芯から心身疲れきっていたので，電車を選ばず，羽田発大船行きのバスを帰途の手段として選んだ．午後７時に乗れば，車内でグッスリ眠っていれば，午後８時頃には大船に着くという読みだった．しかし，これが大誤算．雪で道路が大渋滞．全く動かない．大船に着いたのが午後１１時頃だった．弱り目にたたり目でバスを降りてタクシーを拾おうとしたが，タクシーを待つ人々の長蛇の列はあるがタクシーは全然来ない．仕方がないので歩くことにしたが，道には雪が１０センチ以上積もっており，路面は少し雪が融けてグジャグジャの状態である．靴はずぶ濡れで水が染み込んでくるし，吹く風は冷たくとても寒い．泣きっ面に蜂とは正にこのことだと思いつつ歩いた．このとき，ようやくスピリタス・ニクラシカの酔いが全身から醒めてゆくのが分かった．

後で知ったことだが，ハチキンママと乾杯競争をやったあとで皆とラーメンと食べに行ったらしい．俺はほとんど食べなかったらしいが，注文はしたという．一応は自分の足で歩きつつも，中野さんに支えられてホテルに辿りついたらしい．ホテルの部屋の鍵を革ジャンから見つけ出し，部屋を開け，セーターを脱がせ，ベッドに寝かしてくれたのもすべて中野さんがやってくれたという．俺は一切何も覚えていない．これは本当のことだ．一体，記憶とはなんなのかと凄く興味が湧く．しかし，持つべきものは友である．中野さん，その節は大変お世話になりありがとうございました．沈着冷静な中野さんが酩酊したときにはこの俺が面倒みるといいたいが，そんな時は来ないかも知れないと確信している．

ところで，みなさんはハチキンママのいわれを知っているかい．４人斬りをハチキンというらしい．５人斬りは，当然ジュウキンとなる訳だ．俺と岸田さんの半分を勘定しても合わせてサンキンにしかならない．これでハチキンママに勝てる道理がないはずだ．こんどは５人で闘おうと考えている．

というようなことで偶然手にした革ジャンの石ころがいろいろなことを思い出させてくれた．懐かしい旅の名残である． (2001/12/26)



2002 International Symposium on Empirical Software Engineering (ISESE 2002)

http://se.aist-nara.ac.jp/isese2002/

October 3-4, 2002,
Nara-Ken New Public Hall, Nara, Japan (http://www.pref.nara.jp/koukaido-e/)

Co-sponsored by (subject to confirmation)

- IEEE Computer Society
- ACM SIGSOFT
- Information Processing Society of Japan (IPSJ)
- Japan Society of Software Science and Technology (JSSST)
- National Space Development Agency of Japan (NASDA)
- Nara Institute of Science and Technology (NAIST)

In cooperation with

- Institute of Electronics, Information and Communication Engineers (IEICE)
- Software Engineer's Association (SEA)

Call for Paper

The objective of the International Symposium on Empirical Software Engineering (ISESE) is to foster the development of the area by providing a forum where researchers and practitioners can report on and discuss new research results in the area of empirical software engineering. The symposium encourages the exchange of ideas within the international community so as to be able to understand, from an empirical viewpoint, the strengths and weaknesses of technology in use and new technologies, with the expectation of furthering the more generic field of software engineering.
The symposium focuses on the processes, design and structure of empirical studies as well as the results of specific studies. These studies may be on original or replicated studies, varying from controlled experiments to field studies, from quantitative to qualitative.

(Selected papers will be published in "Empirical Software Engineering: An International Journal". http://www.kluweronline.com/issn/1382-3256)

Topics of interest include, but are not limited to:

- Empirical studies of software process and product
- Comparison of cost estimation techniques
- Analysis of the effects of design methods and characteristics
- Evaluation of the readability of coding styles
- Development, derivation and/or comparison of organizational models of software development
- Evaluation of testing methodologies
- Reports on the benefits derived from using software development environments
- Development of predictive models of defect rates and reliability from real data
- Infrastructure issues, such as measurement theory, experimental design, qualitative modeling and analysis approaches.
- Experiences with research methods such as grounded theory, protocol studies or families of experiments
- Industrial experience in process improvement
- Quality measurement
- Process modeling and applications in industry
- Experience management

Paper Submission

Papers describing previously unpublished, original work are solicited on topics concerning empirical software engineering. Papers should be written in English, double-spaced, with the length to be no longer than 20 pages, using at least 11 point font. Authors are requested to submit both an electronic version of their paper (postscript or PDF format only) and a hardcopy to the Program Chair. The paper should have a cover page, which includes a 200-word abstract, a list of keywords, and the author's telephone number and e-mail address.

Papers must be submitted to **isese@cse.unsw.edu.au**

Program Chair: Ross Jeffery (rossj@cse.unsw.edu.au)
Centre for Advanced Software Engineering Research (CAESER)
School of Computer Science and Engineering
University of New South Wales
Sydney 2052, Australia

Important dates:

◆**Papers due**: April 1, 2002 ◆**Paper acceptance notification**: June 15, 2002 ◆**Camera-ready paper due**: July 15, 2002

Organization:

General Chair: Koji Torii (Nara Institute of Science and Technology)

Program Chair: Ross Jeffery (University of New South Wales)

Program Committee:

Anneliese Amschler Andrews (Colorado State Univ.)
Reidar Conradi (NTNU)
Takeshi Hayama (NTT Data)
Bev Littlewood (City Univ.)
Audris Mockus (Avaya Labs Research)
Janice Singer (NRC Canada)
Walter Tichy, (Univ. of Karlsruhe)
Bernard Wong (UTS)
Stefan Biffl (TU Wien)
Christiane Differding (Taskforce eLuxembourg)
Philip Johnson (Univ. of Hawaii)
Christopher Lott (Telcordia Technologies)
Markku Oivo, (Univ. of Oulu)
Forrest Shull (Univ. of Maryland)
Isabella Wieczorek, (Fraunhofer IESE)
Murray Wood (Univ. of Strathclyde)
Lionel Briand (Carleton Univ.)
Warren Harrison (Portland State Univ.)
Natalia Juristo (Universidad Politecnica de Madrid)
Kenichi Matsumoto (NAIST)
Carolyn Seaman (Univ. of Maryland)
Shingo Takada (Keio Univ.)
Claes Wohlin (Blekinge Institute of Technology)
Marv Zelkowitz (UMD/Fraunhofer Center- Maryland)

Financial Chair: Katsuro Inoue(Osaka University)

Publicity Co-chairs Shingo Takada (Keio University)
Masafumi Katahira (NASDA)

Registration Chair: Kenichi Matsumoto (NAIST)

Local Arrang. Chair: Hajimu Iida (NAIST)

Corp. Prog. Chair: Makoto Sakai (SRA-KTL)

Publications Chair: Takeshi Hayama (NTT Data)

ソフトウェア技術者協会
〒160-0004　東京都 新宿区 四谷 3 - 12 丸正ビル 5F
Tel: 03 - 3356 - 1077　Fax: 03 - 3356 - 1072
E-mail: sea@sea.or.jp
URL: http://www.iijnet.or.jp/sea