SEAMAIL

Newsletter from Software Engineers Association

Volume 5, Number 1-2 1990

目　次

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

その主な活動は，機関誌 SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約 200人にすぎなかった会員数もその後飛躍的に増加し，現在，北は北海道から南は沖縄まで，1650 名を越えるメンバーを擁するにいたりました．法人賛助会員も約 60 社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

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

ＳＥＡＭＡＩＬ　Ｖｏｌ．５，Ｎｏ．１－２　　平成２年３月１５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）
〒１０２　東京都千代田区隼町２－１２　藤和半蔵門コープビル５０５
印刷所　サンビルト印刷株式会社　　〒１６２　東京都新宿区築地町８番地
定価　１０００円（禁転載）

編　集　部　か　ら

おひさしぶりです…

というわけで、記念すべき90年代最初のSEAMAILをお届けいたします。まったく一時期はどうなることかと思われた重症の船便シンドロームも、どうやら受容できる程度(ホントかしら?)に回復の兆しがみえてきました。本当はこの1月号を2月中に出して、船便シンドロームを一挙に1ヵ月遅れ(!)にまで短縮するつもりだったのですが、またまた大雪でも降っては迷惑ですので、ここはグッと堪えて2ヵ月遅れとした次第です、ハイッ:-)。

春の便りは?

ところで今冬は暖冬だったということですが、どうやら春も例年に比べると随分早くやって来そうな気配です。気の早い人はもう花見の予定を立てているとのこと、皆さんの周りではどうでしょうか?

一方、この季節になると憂うつになる人も少なくないようです。そう、花粉症の人にとっては何とも辛いのが、丁度この時期なのですね、ヒバリならぬクシャミに春の訪れを感じるようになるとは、まったく予想外でした。もしかするとそのうち俳句の季語として認められるかもしれません、かく言う本人もティッシュとは切れない仲、耐えるのみとは…。

波乱の幕開け

しかしどうも90年代は色々とありそうな感じですね、英国の新聞からコキおろされた選挙も済み、やはり日本は安定指向かなと思っていた矢先、組閣で一悶着の上、日米構造協議で日本の煮えきらない態度にとうとうシナリオにはなかった突然の首脳会談まで飛び出し、他方、話題の人ゴビーはと見れば赤い国に大統領を作ると息まいて、東欧はその急激な革新の歩調を一行に緩めませんし、世界中がlike-a-rolling-stoneになってしまったのではないかとさえ思える今日このごろです。そういえばSTONESは見ましたか?

もっとゲンコウをっ!!!

しかし灯台下暗し、足もとを見ると相変わらずの原稿不足、ここはひとえに会員の皆さんに縋るだけ、どうか皆さん、身の回りのこと、仕事で感じたこと、旅先で見聞きしたこと、メモ程度で結構ですから、ぜひ編集部あてお送りくださいませマセ、otherwise、重症のシンドロームがまた……

ソフトウェア工学と AI

Software Symposium '89 のミニパネルから

熊谷 章
PFU & 富士通

本稿は，1989 年 6月 14日に開催された "Software Symposium '89" におけるミニパネルの内容をまとめたものである．キーノート・スピーカはコロラド大学の Gerhard Fischer 教授であり，パネラーとして電総研の二木厚吉氏と筆者（熊谷）が出席し，司会は筑波大学の玉井哲雄先生であった．

1. Fischer 教授の講演内容

1.1 AI とは何か

AI は，複雑な情報を処理する技術である．対象分野としては，知識ベース・システムやエキスパート・システム，定理の証明(Theorem Proving)などがある．また，違った観点からみれば，AI とは Ill-Structured Design Problem を解決するための技術であるともいえよう．この他に，やはり AI の対象分野として考えられているものに，Machine Learning などがある．

1.2 ソフトウェア工学とは何か

ソフトウェア工学の１つの目的は，複雑で大規模なソフトウェアをうまく開発することであり，その仕事は，依頼人，開発者，エンド・ユーザ，知識ベースのアシスタントらの間のコミュニケーション・プロセスそのものである．また，別の観点からすれば，ソフトウェア工学の対象としての開発作業は，ある種の知識集約活動だともいえる．

1.3 AI と SE

1.3.1 高機能コンピュータ・システムとシステム内のオブジェクト数の関係

いま評判の高いシステムの機能とコマンドの数は，下表の通りである：

システム名	Function の数	コマンドの数
EMACS	170	462
UNIX	?	700
SYMBOLICS	23119	150 (+2681 flavors)

また，評判の高いシステムのドキュメントの量は，下表の通りである：

システム名	ドキュメントの冊数	ページ数
SYMBOLICS	12	4400
SUN WS	15	4600
SUN Beginner's Guide	8	800

このように，最近の高性能マシンを利用するために必要な情報が巨大化したために発生する問題点としては，次のようなことがあげられる：

１）ユーザには，自分が望む結果を得るための，よりよい方法が分からない．

2）ユーザには，システムが提供している装置や機能が分からない．
3）ユーザには，装置や機能をどのようにアクセスすればよいかが分からない．
4）ユーザには，いつ，どのような方法でその装置や機能を使用すればよいかが分からない．
5）ユーザには，装置や機能が作り出す結果がよく理解できない．
6）ユーザには，自分の要求に応じて装置や機能を組合せ／適応／変更することができない．

ソフトウェアエンジニアリングの工程を大きく分けると，直観にもとづく分析，仕様の確定，作成の３段階に分かれる．1970 - 85年の間は，仕様の確定から作成までにいたる具体化の問題が，ソフトウェア工学の主なテーマであった．技術的な話題としては，構造化プログラミング，検証，形式的仕様などが注目された．このとき対象として考えられた問題は，比較的 Well-Structured な問題であり，最適の解答を求めることがターゲットになっていた．

1985年以降は，与えられた問題に対する直観的洞察から仕様の確定にいたるまでの設計作業が，取り上げられるようになった．こうして考案された新しい技術は，ラピッド・プロトタイピング，視覚化，エラーの削減などである．こんどは，あまりよく構造化されていない（どちらかといえば Ill-Structured な）柔らかい問題が対象であり，とりあえずの満足解を作り出すことがターゲットになった．

次に，コンピュータ科学上の重要課題に対して，理論的な見地と AI の見地から考察を加える．

Issue	Theoretical View	AI View
Computer Science	Formal/Mathematical	Experimental
Main Tools	Formal Specification	Experimental Programming
Basic Challenge	Think More Clearly	Better Tools Needed
Programming Methodology	Verification Before Making	Design Is Error Correcting
Problem	Well-Structured	Ill-Structured
Solution	Optimal	Satisfying

AI がソフトウェア工学に寄与できる点としては，まず第１に，ソフトウェア設計者のための知識ベース支援システムを提供し，次の３点を改善することが考えられる．

1）設計者の問題解決能力を増幅させること
2）設計者の作業環境を改善すること
3）設計者の生産能力を増大させること

２番目は,, AI 技術を用いることにより，複雑な問題の設計に対して，より深くかつよりよい理解を可能にすることである．これは，設計対象の性質が以下のように変わることによって可能になる．

Well-Structured Problem	--->	Ill-Structured Problem
Optimizing	--->	Satisfying
Product	--->	Process
Algorithm	--->	Heuristics Knowledge

高名な現代の哲学者ポパーは,「科学の進歩は既存知識の修正と変更を通して達成される」と述べた．この方法は，エラーの削減の途である．また，アレキサンダーは,「われわれは否定的なものの見方からしか，よい解答を見付けられない」と述べている．この言葉は，ほぼポパーの主張と同じ意味を含んでいる．これらのことから，われわれには多くの実験的な仕事が必要であることがわかる．

では，なぜソフトウェア工学用のエキスパート・システムが存在しないのだろうか．それの答えはこうだ．そもそも，エキスパート・システムは，知識の領域が極度に限定された小世界を対象とし，しかもよく理解された知識領域を対象としたときに，初めて存在するものである．一方，ソフトウェア工学は広範な知識を集約した技術分野であり，しかもあまりよく理解されていないのが現状である．したがって，ソフトウェア工学向けのエキスパート・システムは，まだ存在していない．

1.4 AIの新しい解釈 (The New Look of AI)

現在 AI は広く世の中に受入られ，かつその目的も理解されている．この一般に理解されている AI のゴールをひとことでいえば，自動的で知性がある思考機械を作り出すことである．これに対して，IA (Intelligence Augmentation) というもう１つ別のゴールが考えられる．これは，コンピュータを駆使して人間の知性を増幅させようというもので，対話的な知識メディア，会話のためのツール，知的なサポート・システムなどが具体的な内容である．

この考えは，機械に人間の代行をさせるのではなく，人間の能力を増幅させることが目的である．したがって，人間とコンピュータの交信の仕方が最も重要な問題になる．もしわれわれが，人間でもコンピュータでもよいエージェント（作業代行者）を持つとすれば，間抜けなエージェントと知的なエージェントのどちらを選ぶだろうか．答えは自明である．

現在は，技術的に何が可能かという観点からものごとを考える時代ではなくユーザを中心に据えたシステムの設計を心がけなければならない．現代のハードウェアおよびソフトウェアは，それらを用いてわれわれが意味のあることを実現できる能力のレベルを，はるかに越えるところまで発達してしまった．このような状況からして，IA という新しい AI 応用分野が重要になってきたのである．

「私の鉛筆は私より賢い」というアインシュタインの次の言葉は，この考えをうまく表現している．鉛筆というツールが人間の知的な能力を増大させ強くしてくれるのである．示している．「大抵の知識ベースシステムは，人間の努力を支援するものである．それらは決して自動的な代行者にはなろうとしない．だから，人間と機械の交信をするサブシステムが必要であり，かつ重要である」というファイゲンバウムとマコーダックのも，われわれのアプローチの特徴をよく表わしている．

次に，AI と IA の違いに関する２つの主張を紹介する：

主張１： 全面的な自動化アプローチの形跡は，あまりよくない結果に終わっている．たとえば，自動翻訳や自動プログラミングがそうである．

主張２： 部分的な自動化システムは，全面的な自動化システムよりも，偉大な設計課題（チャレンジ）をわれわれに課している．たとえば，パイロット支援システムのほうが，完全に自動的な電子パイロット・システムより難しい．

このように考えると，全面的な自動化システムへのアプローチから，人間と機械が共存協調するシステム・アプローチへの移行が，きわめて重要に思われてくる．つまり，人間の知性の代行ではなく，人間の知性の増幅ということが新しい目的になる．具体的には，チューター・システム，批評システム，コーチ・システム，アドバイザー・システム，コンサルタント・システムなどが考えられる．

このようなシステムでは，人間とコンピュータの間のコミュニケーションのために，知識ベースを用いたアーキテクチャが必要になる：次図にそうしたアーキテクチャの例を示す：

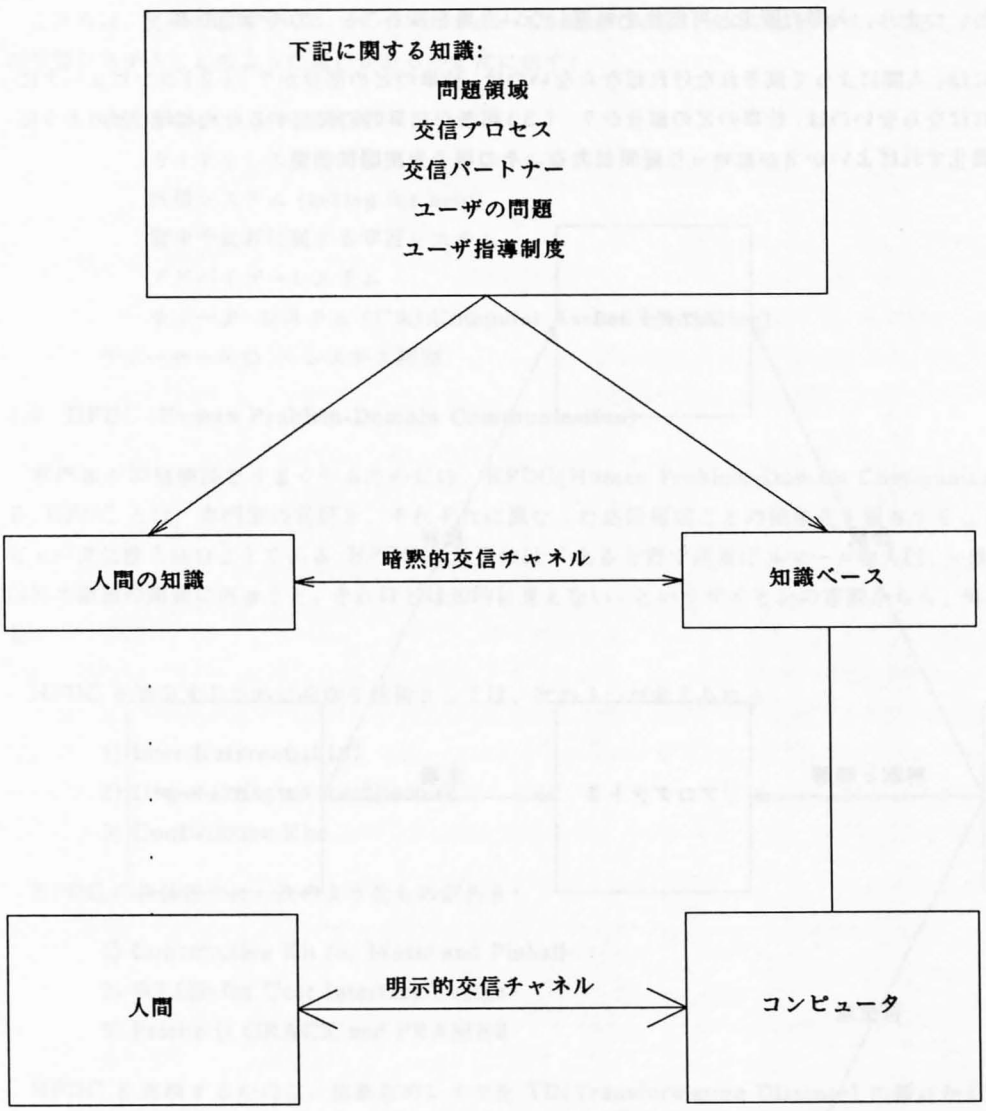


図1　知識ベースの設計環境のアーキテクチャ

知識ベースがなぜ必要であるかを示す会話（Joke）の例を次に示しておく：

User:　"I need more disc space!"
System:　"Try 'rm *'!"

1.5 協調型の問題解決（人間とコンピュータ）

この章では，人間と機械の協調型問題解決システムについて説明する．このシステムの目的は，（１）人間と機械の役割分担を明確にすること，（２）両者が協力的かつより効果的に活動できるような交信プロトコルを設計すること，（３）一発仕事を止揚すること，そして（４）AIの新しい側面の解釈に立って，人間の創造性と生産能力を増進させること，つまり，人間の要求と可能性を無視しない立場を採ること，の４つである．

このアプローチには，人間によって試されなければならないのは，仕事のどの部分か？（２）コンピュータによって試されなければならないのは，仕事のどの部分か？（３）両者が効果的に交信するためには，どのようにこれらの仕事を組織化すればよいか？ といった疑問がある．その様子を次図に示す．

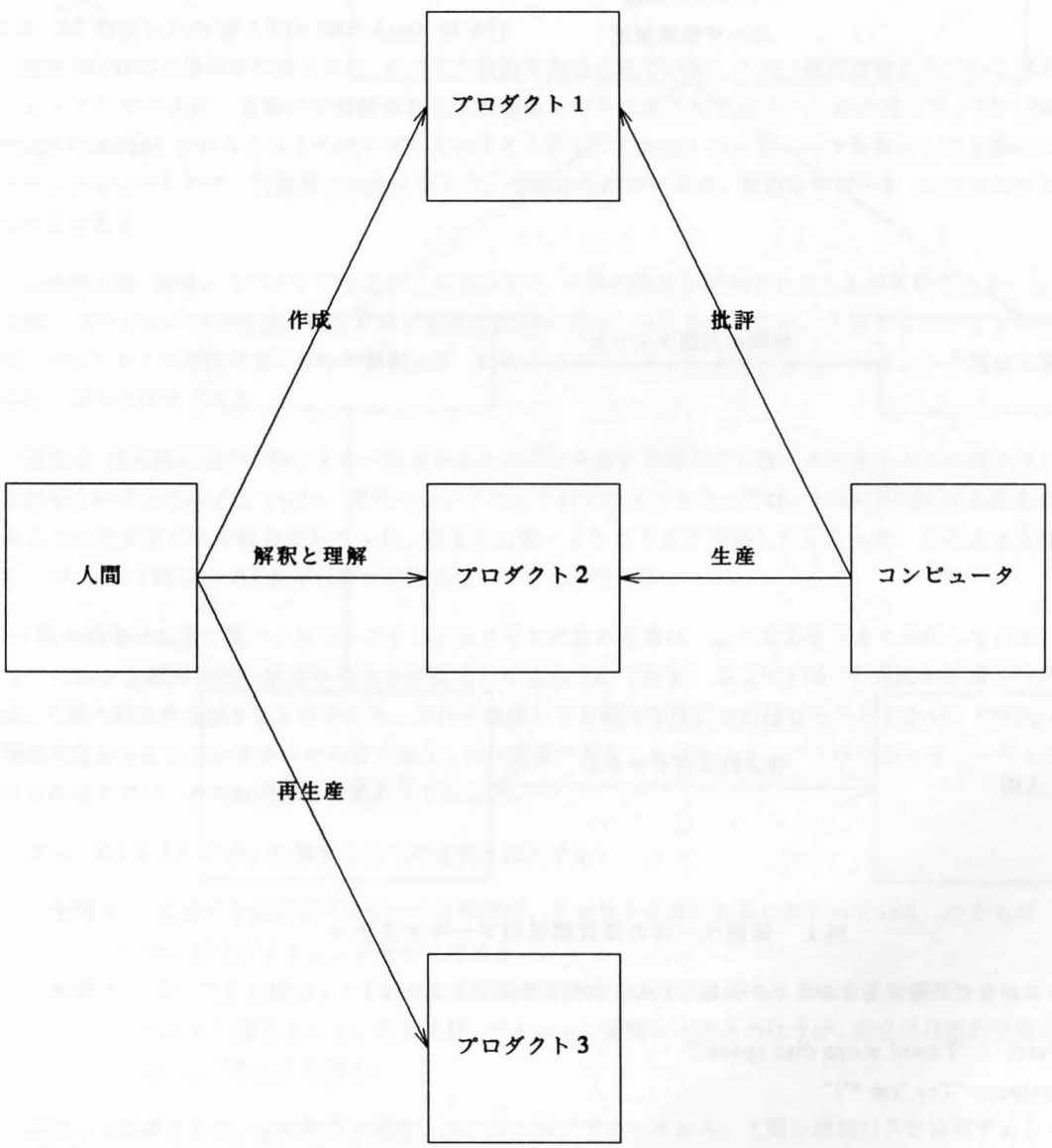


図２ 問題解決のための協調システム

次に，協調問題解決システムの理論的な問題点を指摘する．それは次の２つである．

１）理解の共有
２）お互いが相互にイニシアティブを取れるようなダイアログ

これらは，従来の意味でのユーザインタフェースを超えている．次に，批評をベースとしたアプローチが従来の学習システムとどのような違いがあるかを次に示す：

学習者中心 － ユーザ制御
　ガイドなしの能動的な探究（パワフルなプログラミング環境）
　救援システム (asking for help)
　要求や批評に関する学習システム
　アドバイザーシステム
　チュータ・システム（CAI:Computer Assited Instruction)
ティーチャ中心 － システム制御

1.6 HPDC (Human Problem-Domain Communication)

専門家が問題解決をうまくやるためには，HPDC(Human Problem-Domain Communication) が必要である．HPDC とは，専門家の言語を，それぞれに異なった応用領域ごとの抽象化を賦与することによって，コンピュータに教え込むことである．HPDC の重要性は，「ある分野で非常にスマートな人は，一般的にいって，専門以外の領域の問題に出合うと，それほどは知的に見えない」というサイモンの言葉からも，伺い知ることができる．

HPDC を実現するために必要な技術としては，次の３つが考えられる：

1) Inter-Referential I/O
2) Object-Oriented Architecture
3) Construction Kits

HPDC の具体例には，次のようなものがある：

1) Construction Kit for Music and Pinball
2) WLISP for User Interface Design
3) Palette in CRACK and FRAMER

HPDC を実現するために，抽象化のレイヤを TD(Transformation Distance) の観点から考える．この TD は，始点を問題領域に置く．したがって，問題領域において問題を直接表現できる言語は TD の値が 0 だということだ．いま，TD1 と TD2 を次のように定義する．

TD1 := System Space (LISP, KEE, etc)
TD2 := Application-Oriented Construction Kit (Pinball Construction Kit, etc)

すると，TD1 >> TD2 の式が成り立つ．われわれは TDF の値が小さいような領域向け専門言語を作成しなければならない．かつ，その言語では，設計のやりなおしや再利用を考慮しておく必要がある．この観点から推論すると，HPDC のアーキテクチャ・レイヤには，次の４つの階層が考えられる：

１）独立した設計（Monolithic Design）

汎用言語を用いて言語システムをゼロから開発する．

2）構築キット設計（Construction Kits）

中間レベルの部品を用意しておき，それらを用いて設計する．

3）再設計（Redesign）

既存システムの部品を交換して新しいシステムにする．

4）再利用（Reuse）

既存の部品を用いてまったく新しいシステムを作り出す．

1.7 具体例

Fischer 教授自身がこれまでに開発してきた次のシステム例が簡単に紹介された：

WLISP：ユーザインタフェースの構築キット
NEWTON のユーザインタフェースのウインドウ設計モデル
FRAMER：ウインドウベースユーザインタフェースの設計環境
JANUS-CRACK：構築と議論設計の統合環境
JANUS-VIEWPOINT：構築と議論設計の統合環境

今後の研究課題は，AI, HCC(Human-Computer Communication), SE(Software Engineering) の３者間の関係を樹立することだと述べられた．

2. パネラーとしてのコメント（鹿谷）

2.1 AI と IA について

従来の AI は，自動化つまり人間を代行するシステムの実現を狙っていた．具体例としては，自動翻訳，特定の分野の診断システム，物体認識システム，マシン・ラーニングなどがある．これに対し，IA は，人間とコンピュータが協調して動作するシステムを狙っている．主人公は人間であり，コンピュータは人間の知性を増幅するよう，縁の下の力持ちの役割を演じるという考えである．コンピュータを道具として使うという感覚である．AI システムが自動化できる分野だけを対象にしているのと対照的に，IA システムは，カオスとしての一般的な世界を対象にできる．このアプローチは，コンピュータの応用範囲を大きく変えることができると思う．

	主テーマ	実行者
AI	Artificial Intelligence	Computer
IA	Intelligence Amplifyer	Human Being

2.2 演繹法と帰納法

過去４年間に，われわれ自身（PFU）の研究開発は，次のテーマに的を絞ってきた：

コンピュータの新しい利用分野を見出すこと．
その利用分野に適したシステムのアーキテクチャを設計し，試作し，評価すること．
応用プログラムを試作し，評価すること．
以上の要求を満たすソフトウェアの基本技術を開発すること．

このアプローチによってわれわれが得た成果は，次のようなシステムである：

人間の知的な活動を支援するもの．
アーキテクチャはマルチ・メディアのイメージベース・システム．
HyperCard, HyperText, HyperBook とシステム・モデル作成ツール．

オブジェクト指向開発環境，ビジュアルな MMI 技術，メタ・モデリング技術．

上述の結論に達するまでの思考プロセスを，以下に簡単に解説する．

最初に考えたことは，コンピュータを知的な活動のよき友にするというものだ．AI では，知的活動そのものをコンピュータに代行させることが目的である．多くのエキスパート・システムや，ICOT の言語理解システムなど，これまでに沢山の AI システムが開発されている．これに対して，われわれのアプローチは，Fischer 教授のいう意味での IA と形容できるものであった．すなわち，知的な活動は人間が行い，コンピュータはだだ，人間がその仕事を快適にかつスピーディに行えるような環境を提供するだけである．

そこで，次に，知的な環境および知的な活動とは何か，ということが問題になる．われわれの思考方法は，帰納的実証的なものであった．つまり，理論から演繹される形で知的な活動がなされるという一般の考え方に対して，知的な活動や知識は，既に存在する物（オブジェクト）から帰納的な方法で生成される，という考えである．

ここでいう物(オブジェクト)とは，現実世界に存在しているさまざまな物を指している．コンピュータ・システムの中では，これらの物は，情報とかデータとして表現されている．記号化される前の物は，色々な形で意味を表現している．それを一度記号化してしまうと，物と意味との関係を記号を介して操作することになり，事態が一挙に複雑化してしまう．

明示的にも暗示的にも，共通に話題にしている物の概念を共有していないと，記号を介しての会話は不可能である．したがって，知的活動の支援システムとしては，対象分野に存在するさまざまな物を，コンピュータ・システム内においても，そのまま視覚的に表現できることが必要になる．さらに，それらの物を操作したり，計算したり，処理したり，検索／格納したりできるような機能も，必要である．

きわめて簡単な例を１つ挙げる．囲碁の例である．囲碁のルールや定石を覚えることは，知的な活動である．しかし，碁盤と碁石を使わずにそれらを表現し，知的な活動を支援することは，不可能に近い．次の１手に対する解答も，記号を用いて表現できるとは，到底考えられない．逆に，碁盤と碁石さえあれば，どんなに難しい定石やセンスも，パターンとして表現でき，他人に伝達することができる．この例は，物（オブジェクト）を視覚的に表現することと，それらを操作する MMI が重要なキーであることをよく表わしている．

もう少し別の観点からの考察をしてみよう．従来のコンピュータ・システムにおける演繹的かつ定量的なアプローチと，いま述べた視覚的で帰納的なアプローチの比較である：

比較項目	帰納的方法	演繹的方法
基本要素	認識，パターン	形式，定量化
原理	秩序の生成	秩序の論理
アプローチ	実在への固執	ロゴスの探究
方法	直観的方法	分析的方法
システム構造	動的	静的

従来のコンピュータ・システムでは，ほとんど演繹的方法しかサポートしていなかった．これはコンピュータに自動化，合理化，計算，知的な活動などをやらせようとしたために，理論が最初にあり，それにもとづいたプログラムとデータが作成されたのである．ところが，主役が人間に変わると，事態は一変する．上の表に示したような帰納的なアプローチと，そのための支援システムが必要になる．このようなシステムでは，「百聞は一見に如かず（Seeing Is Believing)」の概念が実現できる．

１つのオブジェクトを，記号（Symbol），意味（Semantics），像（Object）という観点からとらえ，表現技術として考えてみる．

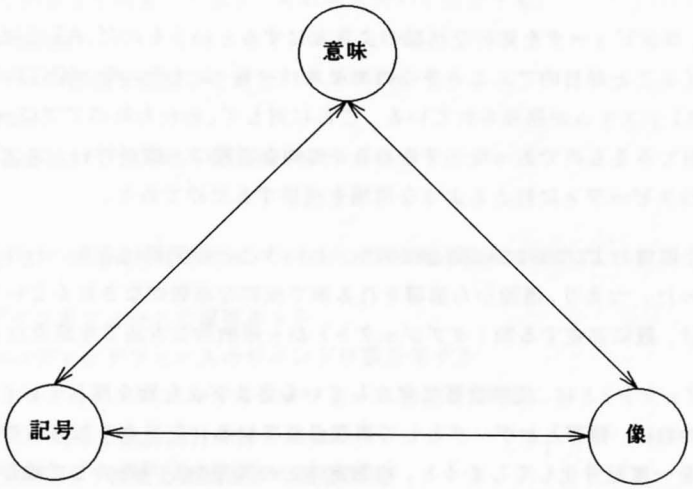


図３　記号と意味と像の３者の関係

従来のコンピュータを使用するときのアプローチは：

記号　－－＞　意味　－－＞　像

であった．このため，Formal Specification がコンピュータ利用において最も重要な技術とされた．そして，Form から Form への変換技術とそれが表わす意味論が，新しい技術の研究開発のテーマになってきた．しかし，意味と像と記号との間には深くて大きな溝があり，これを埋め尽くすのは至難の技である．

これに対する新しいアプローチは，

像　－－＞　意味，記号

という方法である．初めに像をコンピュータで操作できるようにし，それらを駆使して意味や Form や Syntax や記号を考える，というものだ．それは，われわれ人間が日常生活において，自然に行っている思考である．**2.3 ビジュアルな MMI の実験**

前章で述べた内容を背景として，次のようなシステムを研究開発するプロジェクトを実施した．

HyperBook のプロトタイプ試作
ビジュアル･プログラミングの作成と評価
イメージ･データのパターン認識
ソフトウェア開発モデルの記述のためのメタ･モデルの作成
動的なシステム記述の作成（アニメーション）

これらの実験を通じて，いくつかのことが判明した．最大の困難は，われわれがターゲットとしたシステムが，演繹的なアプローチでは開発不可能なことであった．しかし，日本におけるソフトウェア開発では，演繹的な開発方法が一般に定着しており，プロトタイピングを多用する開発を実践しようとすると，多くの問題が発生した．われわれの社会は，組織的にも技術的にも閉鎖的なツリー構造に固まっており，プロトタイピングや MMI のような試行錯誤的システム開発が，日本のソフトウェア社会には向いていないのだった．

第２の困難は，われわれシステム･プログラマにとって所期のシステム開発を行うための開発環境を何にする

かという選択の問題であった．最初は，この問題に対して，マルチ言語システムをもって答えようとしたのだが，そうしたシステムを実際に試作し試用してみて，このアプローチを断念した．試作したシステムは LIPS(Lisp, Ingres, Prolog and Smalltalk) というものであった．断念した大きな理由は，LIPS 自身の開発が大変であることと，Smalltalk 以外の言語のシステム記述記述能力が低すぎることであった．結論として，われわれが採用したのは Smalltalk-80 と C であった．４年前の時点では，それ以外は使い物にならなかった．

第３の困難は，われわれがターゲットとしたシステムがエンドユーザ指向であり，応用分野がよく分っていないと，手も足も出ないことであった．従来のシステム･プログラマや研究開発者は，ハードウェアに通暁していることが先端技術開発の必要条件であったが，新しい視覚的なシステムは，彼らとは無縁の世界であった．従来とは異なる資質の技術者が必要なことが判明した．

第４の困難は，Smalltalk-80 がこの実験に最適であることが分ったにもかかわらず，身近に Smalltalk システムが存在しなかったろとである．また，優秀な Smalltalk プログラマは数が少なく，育てにくいのも，問題であった．

結局，われわれは基本システム･アーキテクチャとしては UNIX を，応用プログラミングの環境としては Smalltalk を採用した．具体的には，自社マシンに UNIX と Smalltalk を移植し，その上で応用プログラムを動作させることにした．このとき，不足する機能は，さまざまなプログラムに吸収させることにした．

ビジュアルでかつ動的な MMI を設計し作成するのに，従来のワープロや紙を土台とする設計方法を使ったのでは，話にならない．また，実現の手段として C 言語や図形用ライブラリを使用したのでは，やはり話にならない．前章で述べた帰納的アプローチおよびビジュアル･システムを可能にするためには，次のような要素技術が必要である：

オブジェクト指向プログラミング環境
イメージ，グラフィクス，文字列，数値などのマルチメディア･アーキテクチャ
高速処理アーキテクチャ
HyperText 技術
アイコン･プログラミング技術
メタ･モデリング技術
アニメーションおよびシミュレーション技術
必要情報を収集する道具（情報の大河から自己用の情報をすくい上げる網）
収集した情報を操作する道具
ドメインに依存したモデリング技術

上記の技術を実現したアーキテクチャを次ページの図４に示す：

Visual Man_Machine Interface	
CASE Tool	HyperBooks
Meta Model	ToolBox
Object Management System	
Object Programming Environment	
Network OS	

図4　Object-Oriented Architecture

こうした技術および環境を準備した上で，われわれは，実験的な視覚的 MMI システムとして，次の２種類のシステムを開発した：

HyperBook とその応用システム

- 鳥類図鑑と旅行案内システム

PWB (Prototyper's Work Bench) システム

- UpperCASE をサポートするシステム

このような自分の開発経験に照らしてみると，Fischer 教授の主張は非常に興味深く，重要なポイントをついていると感じた．今回のパネルからは，実に多くの示唆を受けたと思う．われわれは，今後も教授のの考え方を拠り所にして，自分なりの IA アプローチを探求したいと考えている．

SEA 秋のセミナーウィーク '89
Session B3

ソフトウェア開発環境の構築／利用と技術移転

講　師：岸田　孝一（SRA）
報告者：尾形　修一（大日本スクリーン製造）

1. パラダイムの変化

1989 年春の ACM/SIGSOFT のニュースレターに，C.Floyd が「ソフトウェア工学におけるパラダイム変化の概要」という小論を発表した．その中で彼女は，現在のソフトウェア・エンジニアリングの動きを，次のようにとらえている．

- 世の中は，プロダクト指向からプロセス指向の考え方に変わりつつある．これは古典力学の限界に対して相対論や量子力学が登場したような意味での変化（パラダイムの変化）である．それは，天動説から地動説へのようなコペルニクス的転換ではない．プロダクト視点とプロセス視点は，お互いに相補的である．ただ，ウエイトが除々に前者から後者へと移りつつある．

以下，このパラダイムの変化がどのようなものかを見るために，いくつかの概念をそれぞれの視点からとらえて比較してみよう．

1.1 プログラムとは

＊プロダクト指向の（古典的な）視点

- プログラムとは，フォーマルで数学的な対象である（べきである）．
- プログラムは，抽象的な仕様から出発して，一連のフォーマルな手続きにしたがって作成される（べきである）．
- プログラムの正しさは，仕様と照らし合わせることによって，数学的または論理的に検証することができる（はずである）．

＊プロセス指向の（現代的な）視点

- プログラムとは，人間（ユーザ）の仕事を助ける道具または作業環境にすぎない．
- プログラムは，ユーザのニーズにあてはまるように，一連の学習およびコミュニケーションの手続きを通じて，設計され具体化される（始めから固定された仕様などは存在しない）．
- プログラムの適切さ（正しさではない）は，ある期間の試用および改訂のプロセスを踏んで，ようやく確立される．

1.2 システムとは

システムとは，１人あるいは１群の人間が，ある期間何らかの理由によって，一定の関心を持って切り出した世界の１部分である．

システムは，いくつかの構成要素から成り，各構成要素はそれぞれの特性によっておたがいに区別され，またこれらの特性を操作するために動作が定義されている．

システムには，対象システムとソフトウェア・システムの２つがある．対象システムとは，われわれがプログラムを開発するために切り出した現実世界の１部分である．そして，ソフトウェア・システムとは，われわれが開発したプログラムの集合とそのインタフェイスである．

＊プロダクト指向の（古典的な）視点

- 対象システムは，「ソフトウェア・システムの開発がうまく行くように」という観点に立って，選択される．
- 対象システムの記述は，一定のルールにしたがったデータ処理の流れを示すという形で行われる．
- 対象システムは，本質的に静的なものとしてとらえられる．ソフトウェア・システムの導入「以前」および「以後」という２つの状態だけが考えられる．ソフトウェア・システム導入の効果は，あらかじめ予測可能である．
- ソフトウェア開発者は，対象システムの外に立っている．システムの開発と利用はそれぞれ別物だと考えられる．
- ソフトウェア・システムは，閉じた存在であると考えられる．それは，対象システムに関する完全で矛盾の

ない理解にもとづいて作られたプログラムの集合である．

- ソフトウェアとそれを取り巻く周囲の環境との相互作用は，開発に先立ってあらかじめ規定されている．

＊プロセス指向の（現代的な）視点

- 対象システムは，「ソフトウェア・システムによって支援されるユーザの作業プロセスの最適化がうまく行くように」という観点に立って，選択される．
- 対象システムの記述は，作業・学習およびコミュニケーションの各プロセスをミックスした形で行われる（ユーザと開発者が対話しながら開発を進めて行く）．
- 対象システムは，動的であり，時間とともに進化するものとしてとらえられる．ソフトウェア・システムの導入により作業環境は大きく変化し，それによって対象システム自体に予測しがたい影響がもたらされる．
- ソフトウェア開発者は，対象システムの一部になる．システムの開発と利用は，お互いに影響を及ぼし合うものと考えられる．
- ソフトウェア・システムは，開いた存在であると考えられる．任意の時点において現存するプログラムの集合は単なる1バージョンにすぎず，対象システムのさまざまな部分に関する限られた（そしておそらくは相互に矛盾した）理解を実現したものであるがゆえに，いずれは改訂される運命にある．
- ソフトウェアとそれを取り巻く周囲の環境との相互作用は，ユーザの手で，実際の作業プロセスにおいて生ずるニーズに合わせて，適当に調整される．

1.3 品質とは

＊プロダクト指向の（古典的な）視点

- 品質とは，プロダクトの持つ性質である．
- 品質特性とは，プログラムの信頼性や効率性などを指す．それらの値は，プログラムを変えることによって変化する．
- 品質特性の値は，テスティングや検証によって測られる．
- 品質特性の定義は，プログラムからユーザを見ることによって行われる．たとえば，User Friendlinessとは，プログラムを主語とし，その観点から（標準的なしかし仮想的な）ユーザ・プロセスを眺めることによって定義される．

＊プロセス指向の（現代的な）視点

- 品質は，プロダクトを利用するユーザ・プロセスの持つ性質である．
- 品質特性とは，プログラム利用状況の信頼性や効率性などを指す．それらは，作業環境が変わることによって変化する．たとえば，同じワープロ・ソフトが，ある人には使いやすく，別の人には使いにくいということがある．また，初めは使いやすかったソフトが，1ヶ月使ったあとでは使いにくく感じることもある．
- したがって，品質特性の値は，ユーザの（主観的）評価によって決定される．
- 品質特性の定義は，ユーザからプログラムを見ることによって行われる．たとえは，ソフトウェアの適切性は，ユーザ・プロセスを主語とし，その観点からプロダクトを眺めることによって定義される．

1.4 ソフトウェア開発とは

＊プロダクト指向の（古典的な）視点

- 開発の目的は，1つのソフトウェア・システムを作り出すことである．
- 開発に引き続いて，誤りの除去や新しいニーズへの適応を目的とするメインテナンスの工程が発生する．
- ソフトウェア開発は，それ自体独立した活動として考えられる．

＊プロセス指向の（現代的な）視点

- 開発の目的は，1つのソフトウェア・システムの一連のバージョンを作り出すことである．
- メインテナンスの概念は，ソフトウェアに関しては無意味である．俗にいうメインテナンスに関連する諸活動は，すべてバージョンの開発として位置付られる．
- ソフトウェア開発は，システム開発（つまりユーザ・プロセスを改革するためのより大きな開発活動）の一部を構成するものだと考えらる．

1.5 開発方法論とは

＊プロダクト指向の（古典的な）視点

- 開発方法論はそれ自体1つのプロダクトである．方法論を正しく利用すれば，だれがやっても必ず同じ成果が得られる．
- 方法論は，開発者間のコミュニケーションを制約する働きをする．
- 方法論は，ソフトウェア開発から属人的な要素を排除する効果をもたらす．
- 方法論は，一般的に適用可能であることが望ましい．

- 方法論は，静的であり，よく定義され，状況に左右されないものでなければならない．
- 方法論は，プロダクトの特性をきちんと記述するのに役立つ．

＊プロセス指向の（現代的な）視点

- プロダクトとしての開発方法論などは存在しない．ただ，ある特定の方法論が開発され利用されてきたプロセスだけが存在する．これらのプロセスは，方法論を利用した成果に影響を与える（つまり，だれがその方法論をどう使ったかで，結果は異なる）．
- 方法論は，開発者間のコミュニケーションを支援する働きをする．
- 方法論は，それが人々によって使われてはじめて意味がある．
- 方法論は，対象とするアプリケーション・ドメインおよび開発工程と密接に関連している（汎用的なものではない）．
- 方法論は，経験によって進化する．また利用者のニーズや開発状況に応じてカスタマイズされる必要がある．
- 方法論は，プロダクトが次第に明らかになってくる協同作業のプロセスを支援する働きをする．

1.6 ドキュメントとは

＊プロダクト指向の（古典的な）視点

- プログラムは，ドキュメントを読むことによってのみ理解されるべきである．
- プログラムの意味は，その仕様書で与えられる形式的意味論によって定義され，ドキュメントを機械的に解釈することによって導かれる．
- 仕様書の重要な構成要素は，プログラム中に具体化されるべきモデルである．このモデルは，１つの形式的言語を用いて記述されなければならない．自然言語による説明はあまり好ましくない．
- プログラムに対するわれわれの知識は，形式的な規則を理解することによって得られる．ドキュメントは完全で，一貫性を持ち，あいまいさのないものでなければならない．でないと，解釈が一意的に定まらない．
- ドキュメントは，プログラムが何をするかを，はっきりと記述していなければならない．また，それがどうやってなされるか（開発者の視点）を抽象的に説明すべきである．
- ドキュメントは，設計（およびその他の）プロセスから得られた結果を記述するべきものである．

＊プロセス指向の（現代的な）視点

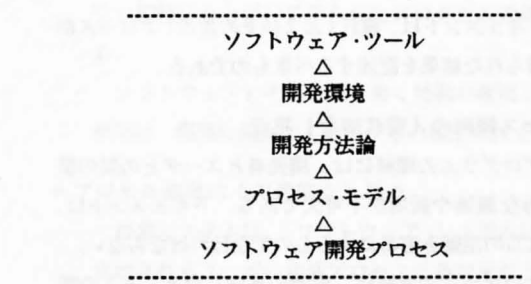
- プログラムの理解には，開発者とユーザとの間の個人的な議論や試用が不可欠である．ドキュメントは，これらの活動を容易にするものでなければならない．
- プログラムの意味は，作者の意図したところとの関連において考えられるべきものである．またそれは，プログラムがどのように使われるかと密接に関連している．仕様書と現実世界との関連は，仕様書に関する議論を通じて明らかになる．
- あらゆるドキュメントの本質的な部分は，自然言語で書かれ，いろいろな概念を紹介し，モデルに関する定義や説明を与える．形式的表現は，もし必要なら使ってもよいが，何ら新しいことを付け加えるものではない．
- プログラムに対するわれわれの知識は，ほとんど，背景となる既存の知識を用いた実例やたとえを通じて得られる．適切な例示やたとえは，ドキュメントのどこがポイントであるかをはっきりさせ，あいまいさに関する人間の側の許容度を拡大する．
- ドキュメントは，プログラムが何をどのようにするか，またそれが「なぜ」であるのかを，読者に理解できる言語を用いて，しっかりした観点から説明するものでなければならない．
- ドキュメントは，また，ソフトウェア開発の最終成果がいかにして得られたか（学習の経験，さまざまな意志決定やその理由，採用されなかった代替案など）を記述すべきものである．つまり，ドキュメントは，結果ではなく，プロセスを記述したものである．

2. プロセス指向の視点から見た環境開発

2.1 開発環境の概念構造

さて，われわれの主題であるソフトウェア開発支援環境というものを考えてみよう．それは，プロダクトとしては，一連のツール群から構成される．対象システムは，われわれ自身の手によって行われるソフトウェア開発のプロセスである．

したがって，きわめて常識的な観点に立てば，次のような概念の階層構造を描くことができる．



上の図は，５年前に行われた第２回 ISPW（International Software Process Workshop）のオープニング・セッションで，会議のコーディネータ側から提示されたものである．

この図に対し，ワークショップ参加者の一部（B.Balzer他）から，強烈な異議申し立てがあり，ワークショップが冒頭から紛糾したことが，記憶に残っている．

その異議申し立ての主旨を整理すれば，およそ次の通りである：

- この図が示す枠組みは，プロダクト指向の視点は快適であるが，プロセス指向の視点からすると，きわめて不快である．
- すなわち，この枠組みは現状の開発プロセスを前提とし，今後のソフトウェア開発もほぼ同じプロセス・モデルにしたがって進められると考えている．しかし，ほんとうの意味でソフトウェア開発の改善を意図し，そのための開発支援を考える場合には，現状の開発プロセスそのものを根本的に変える必要性も含めて考えるべきはないのか．
- そして真に新しいツールあるいは環境というものは，この図が示す概念構造図の土台であるプロセス・モデルそのものを破壊してしまうような力を持っていなければならない．

われわれがいま構築しようとしている開発環境とは，この意見のように，従来の開発プロセスを変えてしまうような環境であろう．

紙とエンピツで書かれていた仕様書を単にワープロ化しただけでは，これまでの開発プロセスは変化せず，それほど大きな改善は期待できない．しかし，何らかの方法論にもとづく要求／設計支援ツールを含んだツールを適当なワークステーションに載せて導入すれば，プロセスそれ自体の変革によって，大幅な改善がもたらされるであろう．

2.2 開発環境の構築

われわれにとっての対象システムは，組織や人間を含むソフトウェア開発作業である．これをどう切り出すかは，それぞれの対象システムが置かれている状況に依存する．

そして，この対象システムを支援するソフトウェア・システムすなわち開発環境は，あるホスト（ハードウェア／OS／Network）上のツールの集合と，さまざまなサーバであろう．

現在，講師自身が参加している SDA コンソーシアムでは，「ソフトウェア設計支援環境の標準的なアーキテクチャ」の研究が行われており，その中でこのサーバの詳細が考えられつつある．たとえば，オブジェクト・サーバ，プロセス・サーバ，アナリシス・サーバ，アダプテイション・サーバなどが考えられている．

2.3 対象システムのとらえ方

伝統的なプロダクト指向の視点に立つ環境構築では，開発プロセスは変えずに，比較的インパクトの弱いツール群を導入して，生産性や品質に関して使用前と使用後の比較をすることが多い．また，ふつうは，環境構築者（ツール提供者）と環境利用者（ソフトウェア開発者）とを分離して考えている．

そして，きわめて非現実的な仮定ではあるが，環境構築の期間における技術進歩を一応は凍結して考えることも一般的である（その典型的な失敗例が SIGMA プロジェクトであろう）．

一方，新しいプロセス指向の視点に立てば，まず，プロセスの変更（破壊）を想定する．したがって，環境構築者と環境利用者の密接なコミュニケーションを可能にするメカニズムを確立することが第１である．環境構築とは，ツール開発者とそのユーザ（すなわちそのツールを使ってアプリケーションを開発するエンジニア）との間の絶えざる学習およびコミュニケーションのプロセスだからである．

もちろん，急速に発展する技術進歩への対応も十分に考慮し，そのための準備も考えられなければならない．

2.4 環境利用の効果

プロダクト指向の視点では，同一プロセスにおける生産性や品質の比較が中心であり，「もの」に換算したコスト／パフォーマンス分析が行われる．

一方，プロセス指向の視点では，新／旧プロセスの相互比較が中心であり，開発チームのパワーに換算したコスト分析が考えられる．

2.5 環境とユーザ

プロダクト指向の視点では，「有能性」とは，ツールまたは環境の働きに関する概念である．ツールがうまく機能しない場合，それを「エラー」と呼び，エラーは，ツールの側におけるその扱い方により分類される(例: プログラムの構文エラー)．

そうしたエラーは，ツールの能力の限界をあらわす．人間の能力は，あらかじめ定められた状況下でツールを正しく使えるか否かによって，判定される．

環境は，ルールと推論に基づく意志決定機能を具体化したものであることが望ましく，データ処理·記号操作·規則に基づく推論などに関して，「知的」であらねばならない．

ユーザは，それぞれの有能さにしたがって何種類かに分類され，環境は，それ自身が内蔵するユーザ·モデルにもとづいて，ユーザの行動をモニターする．,ti +1 一方，プロセス指向の視点では，「有能性」とは，人間がツールの助けを借りて仕事を遂行する活動に関する概念である．人間の期待とツールの働きがうまくマッチしない場合，それを「エラー」と呼び，エラーは，人間の側におけるその原因により分類される（例: ファイル名のスペルミス)．

そうしたエラーは，学習過程（つまり，ツールの開発·導入過程)には必ずつきものである．それは，人間が環境の使い方をマスターする上での前提条件だと考えられる．

知性が要求されるのは，ツールではなく人間の側であり，人間はツールを「知的」に利用しなければならない．すなわち，複雑なニーズ，変化する目標等のオープンな状況に対処できなければならない．環境は，現実のニーズや予想外の状況に対する人間の意志決定を支援する．

ユーザの能力は，ツールの使用という学習プロセスの進展にともなって，つねに変わって行く．ユーザは，ツールの活動に自由に介入し，自らのニーズに合わせてツールのカスタマイズができるようになっていることが望ましい．

3. 技術移転の Case Study

3.1 技術移転のモデル

世の中には新しい技術を生み出す人間と，それを使い現実的な問題を解決しようとする人間がいる．これは，たとえば，大学などの研究機関と企業との関係に，典型的に見ることができる．また，一つの企業の中でも技術の生産者(あるいは輸入者）と消費者が存在する．

現実的な問題として，この２種類の人間，すなわち技術の生産者と消費者の間に橋を架けるのは難しい．というのは，かれらはそれぞれ別別の世界（島）に生きており，両者の間には，「相互蔑視の冷たい海峡」が存在するからである．

しかし，ソフトウェア技術の歴史の中で，１９８０年代にはこの「技術移転プロセス」が最も重要な課題として，多くの人々の関心を集め，その改善のためにさまざまな努力がなされてきた．具体的な１つのあらわれとしては，DOD がスポンサーとなって作られた CMU/SEI がある．

一般的にいって，環境構築チームは，次のような両面的役割（２重人格者的性格）を持つ必要がある：

- 外部に対しては Technology Pull（すなわち，新しい面白そうな技術やツールを見つけだし，それを取り入れてくる）
- 内部に対しては Technology Push（それを自分の組織にあわせてカストマイズし，採用するよう現場を説得する）

この２面的な役割を遂行できるために，環境構築チームは，現場のプロジェクトよりも一層質の高いスタッフを必要とする．そうしたスタッフをそろえることが，また，もう１つのむずかしい問題である．

以下では，技術移転の実際的な例を見ながら環境構築チームのあり方を考えてみる．

3.2 構造化プログラミングの教訓

７０年代初めに，アメリカの IBM／FSD においてチーフ·プログラマ·チームの実験が行われ，その成功が，いわゆる構造化ブームのきっかけとなった．ひとことでいえば，その教訓は，新技術の導入によるプロセスの改善には，質の高いエンジニアを有効に使うことが大切だということである．俗に「２：８の原則」（つまり，２割の人間が仕事の８割を行い，８割の人間が残りの２割の仕事を行っている）ということがいわれる．ポイントは，だれが２でだれが８かを見分け，２の人間を大切に扱うことである．

多くの人々に Gotoless プログラミングだと誤解された構造化ブームが去ったあと，SA／SD や Jackson 法などの新しいソフトウェア設計方法論が登場した．

これらの方法論は，プロセス破壊（改革）を進展させる上で，一定の影響があったと評価できる．つまり，単にプログラミングの工程だけを部分的に改善するのではなく，ソフトウェア·システム設計プロセス全体を新しく考え直す必要性が提起され，さらにシステム分析（対象システムの切り出し方）についての見直しが要求されるようになった．

現在の CASE ブームやオブジェクト指向技法（言語）への一般の関心は，その延長線上にあると考えられる．すなわち，単なるツールや言語の導入ではなく，新しい方法論の

導入にともなうプロセスの変革をどうすべきかが問題にされているのである．

3.3 SRA における Unix 環境

(1) Unix 導入にいたるまで

SRA では，７０年代前半に，独自の構造論的プログラミング技法を開発し，その教育（社内および社外）に努力を傾けた．しかし，自動化支援環境がなかったため，結果としては不徹底に終わった．

そのことの教訓から，次（７０年代後半）には，構造的設計言語にもとづくアプリケーション・ジェネレータおよび構造的テスティング・ツール（カバレージ・アナライザ等）などの開発／普及に力点を移したが，やはり個別ツールの限界に突き当たった．

こうして，総合的支援環境の必要性に目覚め，８０年夏，日本で最初に（いや，おそらく世界でもソフトウェア・ハウスとしては初めての試みとして），アプリケーション開発ツールとしての Unix の導入に踏み切った．しかし，Unix は単なるツールではなく，１つの文化（サブカルチャー）であり，その組織内への移植には，いろいろな問題を解決しなければならなかった．

(2) 最初の Unix

まず，Technology-Pull の側面では，小人数のスタッフの選抜と既存組織からの分離を行い，同時に新しい文化の伝導者として，当時 Unix 開発のメッカ的存在であった UC Berkeley から大学院生のハッカーをコンサルタントとして採用した．

次に，Technology-Push の面では，選抜されたスタッフを中心に，新しい組織単位としての環境開発グループを編成した．

これは，私自身の信念であるが，技術を売り物にするソフトウェア・ハウスには，研究開発コミュニティにおける絶えざる技術進歩のプロセスをウォッチし，その成果を速やかに取り入れるために，「企業内遊園地」ともいうべき組織単位が必要だと思う．

そこには，最新の設備（ハード／ソフト）があり，若い有能なスタッフが，プロセス革新のための自由な実験を続けていなければならない．慢性的な人手不足（特に質の高い技術者の不足）に悩んでいるこの業界にとって，こうした組織を維持することはきわめてむずかしいが，それはマネジメントにとって第１義的な責務であろう．

新技術を組織に導入するさいに，戦術的にもっとも重要なこと（とくに Technology Push の観点から）は，最初に適切なショーケース・プロジェクトを選び，そこで華々しい成功をおさめることである．

それには，ツール・スタッフと現場との密接な連係が必要であり，また，生産性や品質などの伝統的なプロダクト・メトリクスで測った成功データを提示する必要がある．もちろん，（組織の内外における）成果の PR も忘れてはならない．

われわれの場合，たまたま開発の最終段階にあった中国・上海の製鉄所向けアプリケーションがショーケースに選ばれ，ツールとしては比較的無難なモジュール・テストベッドの応用が試みられて，予想通りの成功をおさめた．結果は，ソフトウェア・シンポジウムなどの場で発表され，社内よりも先に，社外で大きな反響をまきおこした（８１年から８２年にかけて，約百社から数百人の見学者がわれわれの Unix 環境を見学に訪れたという事実が，この成功を実証している）．

(3) その後の環境構築

その後の１０年間には，技術移転の観点から見て興味深いイベントがいくつも起きている．

まず第１は，環境の日本語化である．Unix 自身の日本語化は，ASCII の深瀬氏らの努力によって早くからなされていたが，SRA 社内の環境は端末装置の問題などもあって，なかなか日本語化されなかった．

しかし，８２年秋に，ある研究開発プロジェクトにおいて，設計仕様の整理および納品用ドキュメントの作成を目的として，W.Riddle の Joseph システムのアイデアを転用したツールが試作され，PC を入力端末として日本語ドキュメントの作成が成功すると，その噂があっという間に社内に広まり，一般の Unix 環境に自然に浸透していった．

これは Unix それ自身についてもいえることであるが，ツール開発者が自らのプロセスを改革しようとして生みだしたツールは，Technology Push の観点から見て，きわめて強い説得力を持つことの証拠であろう．

環境構築者と利用者との交流を促進し，学習およびコミュニケーションのプロセスを改善するためには，開発現場から「遊園地」への短期または長期の留学制度が考案され，それなりの効果を上げた．

８０年代後半には，それまでのスーパーミニを中心にした集権的な環境から，ワークステーションと LAN による分散環境の構築への試みが開始された．

もともと，国家プロジェクト SIGMA は，そうした目的

でソフトウェア業界が中心になって企画されたものであるが，実際のプロジェクト運営が，Unix 環境の構築に経験の浅いメインフレーマを中心に行われ，当初に期待した成果がほぼ絶望的になった時点（８５年暮）で，自前の努力に切り換えることが意志決定され，たまたま Sony 社からの話もあって，NEWS ワークステーションの開発にユーザとして全面的に協力することになった．

その結果，いまではワークステーションの台数も飛躍的に増大し，真の分散環境を実現するハードウェアおよびソフトウェアのコンポーネントは，ほぼ整備されつつある．しかし，それらの要素を１つの環境として統合するためには，まだいくつかの技術的課題を解決しなければならず，それには，研究開発コミュニティにおいて現在試行されているいくつかの新しいアイデアを導入し実用化する必要がある．

(4) これからの環境構築

上に述べたように，真の意味での分散環境の実現には，先進的なソフトウェア・エンジニアリングの研究成果を取り入れることが急務であり，そのためには，いろいろな形での Technology-Pull のメカニズムを模索して行かなければならない．現在，そうした目的のために，環境の標準アーキテクチャを研究する産学・国際共同プロジェクト（ SDA および PMP）を推進している．

また，環境の地方展開のために，ネットワークの拡充および環境スタッフの地方分散も，除々にではあるが，進められつつある．

3.4 TRW の SPS プロジェクト

アメリカの代表的ソフトウェア・ハウスである TRW 社が Unix を開発現場に導入するための実験を行ったのは，１９８１年のことであった．

この実験は，ショーケースとして綿密にプランニングされ，まず，事前調査（社内における開発プロセスの実態調査）という形での PR が行われた．この時点で，現実のソフトウェア開発作業の大半の時間が，いわゆる OA 的な仕事に消費されており，したがって適当な支援ファシリティを用意してやれば容易に生産性の向上が達成できるという予想が成り立っている．

実際の導入実験は，中規模の現実のプロジェクトを対象に行われ，建物設備までを含めた本格的試行がなされた．結果は，当然予想された通りの成功をおさめ，その成果は学界および専門誌で効果的に発表された．

TRW では，その後も，プロジェクト・データベースを中心とする環境の統合を目指す息の長い実験を続けており，また，プロセス・プログラミングのアイデアを実験する Arcadia プロジェクトも強力に支援している．

SPS プロジェクトを指揮した B. Boehm は，これまでの環境構築から得られた教訓を次のように総括している：

- ソフトウェア技術への投資は必ずペイする
- ユーザ層の拡大による進化の促進
- 「メインテナンス」問題を忘れるな！
- ツールはカスタマイズ可能であること
- ツール・ジェネレータの有用性
- 買えるものは作らない
- 環境 DB の重要性

3.5 JSD 社における国家プロジェクト

７０年代半ばに，国の支援による業界協同の技術開発を目的として設立された JSD（協同システム開発）社では，これまでいくつかのプロジェクトが行われた．それらを，技術移転の観点から総括してみよう．

(1) PPDS: 1976-81

JSD における最初の国家プロジェクトであり，プログラミング工程における生産性向上ツールの開発を目指した．しかし，当初の計画があまりにも先見性を欠いていた．私自身も企画委員会のメンバーだったが，バッチ処理から TSS 時代へという時代の変化を読みそこなったし，また IBM の影響力を買いかぶりすぎて PL/I を開発言語に選んだのもよくなかった．

しかも，官制プロジェクトであるがゆえに，

初年度 － 概念設計
２年目 － 基本設計
３年目 － 詳細設計
４年目 － プログラム作成
５年目 － テストと評価

という，杓子定規なウォータフォール型の開発プロセスにしたがわざるをえず，成果の是非を問うプロダクト視点からは，ほとんど失敗に終わった．

しかし，角度を変えて，プロセス視点から見れば，このプロジェクトは，日本のソフトウェア業界の代表的な技術者たちが初めて一緒にツール開発に取組み，その過程を通じてお互いの間のコミュニケーション・チャネルを確立することができたという点で，大きな成功をおさめたと評価できよう．

今日の SEA の母体となった(旧)ソフト協の技術委員会は，JSD プロジェクト関係者を中心に組織され，よい意味での「業界内遊園地」の崩芽となった．

(2) SMEF: 1981-85

このプロジェクトは，建て前上は PPDS の「成功」を受けついで，ソフトウェア保守の支援環境構築を目指した．しかし，本音の目的は Unix のわが国ソフトウェア業界への導入(および，当時はまだ耳新しいコトバであった開発環境という概念の普及)を狙ったものであった．

PPDS における経験から，プロダクトよりもプロセスを重視し，プロトタイピングの考え方を取り入れたり，ワークショップ・スタイルの技術ミーティングを連続的に開催したりといった新しいプロジェクト運営が試みられた．
また，TRW/SPS や Toolpack その他のアメリカにおける環境プロジェクトとの情報交換も積極的に行なわれ(Technology Pull の役割)，結果としては，Unix およびそれを土台とする開発環境の概念を業界内に定着させることに，かなりの成功をおさめた．

(3) FASET: 1985-89

現在その最終年度にあるこのプロジェクトの本来の意図は，さまざまなフォーマル・アプローチの(開発現場での)有用性を，実験的に評価することであった．

実際のプロジェクト運営にさいしては，研究分野からの Technology Pull を行なうことの難しさに直面している．また，先進技術の研究とプロダクト開発とのジレンマをどう解消するかも，大きな問題である．

3.6 SIGMA プロジェクト

このプロジェクトに関しては，ソフトウェア業界のエンジニアたちが中心になって作られたオリジナルな計画と，実際にプロジェクトが開始されてから MITI の意向にしたがって具体化された内容とが，大きく食い違っている．

以下に両者の特徴を併記しておくので，技術移転の側面からの比較を試みていただきたい．

(1) オリジナル・プラン

- まずネットワークを
  - 国立大学／研究機関を利用したノード展開
  - そこに接続できるものはすべて SIGMA-WS
  - ネットワークを用いたプロジェクト運営
- All-Japan の環境スタッフ編成
  - 真の意味での業界内遊園地
  - 個性的なテクニカル・マネジメント
- ユーザ(ソフトウェア業界)主導型の開発
  - 有償の開発プロポーザル募集
  - 公開ワークショップ形式の審査
- 技術開発と移転
  - 新しい分散 OS の試作
  - ツール・インタフェイスの標準化

(2) 具体化されたプラン

- ネットワークはあとまわし
  - 立派なセンタ
  - LAN によるメンフレーム接続だけ
  - いまだに稼働しないまぼろしのネット
- 管理主体のスタッフ編成
  - 旧態以前たる出向体制
  - 没個性的な官僚的マネジメント
- 官僚主導型の開発
  - 密室での開発企画
  - 失敗に終わった OS 標準化
- 技術開発と移転
  - 技術凍結にもとづくツール開発
  - プロセスの固定化

4. 環境構築者へのメッセージ

最後に，環境(ツール)開発と技術移転の問題に関して多くの示唆を含んでいる先人(哲学者および芸術家)のことばを，環境構築者へのメッセージとして紹介して，結びに代える．

(1) ツールの本質(エリック・ホッファ)

- 人間が最初に飼いならした家畜である犬は，まず遊び友達としての習性を買われたのである．狩猟用に使われるようになったのは，だいぶあとのことであった．また，弓は，はじめ武器としてではなく，楽器として発明された．

(2) 分析とは何か？(パウル・クレエ)

- 化学における分析では，未知の物質を要素に分解しその特性を調べ，謎を解く．芸術活動においては，そうした分析のアプローチはとらない(それは，単なる模倣またはにせ作へと導くだけにすぎない)．芸術における分析は，作品と向かい合って，その作品が生まれた道程を辿ってみることである．それによって，初めて，自分の足で歩き出すことができるようになる．

SEA 秋のセミナーウィーク '89
セッション C1

ソフトウェア・プロジェクトはなぜ失敗するか

講　師：松原　友夫（日立ＳＫ）
報告者：椎熊　敏朗（日本データ通信）

1. イントロダクション

プロジェクトの失敗は，多くの人が経験していることでしょう．講師もその例にもれず，今までに数多くの失敗を経験しているので，このテーマでお話しする資格があると思います．

講師の初期の仕事は機械工場での設計計算でしたが，そこでは，機械工学の世界に昔からあるパラダイム，すなわち「事実と自然から学ぶ」という考え方を身につけました．そうした立場から，１０数年前に，ソフトウェア開発における数多くの成功と失敗からの教訓を１冊の小冊子にまとめたことがあります．その内容は現在でも新鮮に生きていると感じています．

最近，アメリカでは失敗の話が増えています．日本では，あまり失敗プロジェクトが人々の話題にならないようですが，それは，失敗しかかっても（力まかせに）立て直して何とかしてしまうか，あるいは，失敗をウヤムヤにしてうまく隠してしまうかの，いずれか場合が多いためでしょう．アメリカは契約社会のせいか，プロジェクトの途中での放棄がよく起こり，裁判沙汰になったり，また事故が公になることも増えているようです．しかし，実態は，どちらの国でもあまり大差がないのではないでしょうか．

2. 米国での事例と論議

アメリカでは，"恐ろしい話"として，次のような失敗例が紹介されています：

- 原子力発電所がたった１つの安全係数の'＋','－'の符号誤りで停止命令を受けてしまったこと
- １９８０年の最後の日（３６６日目）にOSが一斉に停止してしまったこと
- ミサイル発射システムの誤動作によるスクランブル
- F-16 のフライトシュミレータが赤道を越えたところで機体が反転してしまったこと

また，以上のようなシステムの故障の他に，いくつものプロジェクトの失敗事例が報告され，これらによる訴訟も増えています．

たとえば：

- プロジェクトの途中放棄
- システムが完成しても運用に至らず捨てられてしまう
- 完成して運用はしたけれども納期の大幅な遅れ，コストオーバー，ユーザ要求との不一致，間違いだらけのアウトプット，性能の悪さ，頻繁なシステムダウンが発生した

といったぐあいです．

また，E・Yourdon は，平均的な事実を次のように要約しています：

1) プロジェクトの１５％が完成していない，あるいは動いていない
2) そのうち，特に２５人年以上の大規模システムのプロジェクトでは２５％がキャンセルされている
3) ある典型的なプロジェクトでは（予算，スケジュール共）見積の倍かかっている
4) 自社開発のソフトでは１００行中３～５のエラーがあるものが平気で使われている

こうした失敗の背景にある原因としては：

- 質の高い技術者の不足（日本も同様）
- １．４年毎に人が入れ替わる（アメリカ固有）
- 予算の７０％が古いシステムのメンテナンスに費やされている（日本も同様）
- 複雑で大きななシステムが増えている（１千万ステップ規模のシステムがあたりまえ）
- プロジェクトマネジメントが余り進歩していない
- クリティカルなシステムの増加

等が挙げられています.

特に,クリティカルなシステム(一旦トラブルを起こすとはかり知れない国家的・社会的な影響をもたらすようなシステム)の信頼性の問題は,新しいソフトウェア危機の1つとして認識されるようになりました.

このようなことから,"アメリカはこのままでは危ない","ソフトウェアも半導体の2の舞になる","このままではいずれ日本に負けてしまう"といった発言が,あちこちで聞かれるようになりました.

[報告者注]この点は,実際のところかなり疑問で,報告者としては,むしろ,"日本はこのままでは危ない" と叫びたい心境です.

さて,それでは,ソフトウェア工学がプロジェクト失敗を克服できるかということですが,まず,悲観的な要素としては: ,sp .5

- システム規模の拡大(1千万ステップ以上のものがふつうになってくる)
- システムの構造の複雑化
- 楽観的な開発者が危ない契約を結ぶこと
- 失敗の経験が他人に伝わらず,同じ過ちが繰りかえされること

等が挙げられます.

逆に,よいニュースとしては:

- 大規模システムに対処するためのエンジニアリング手法の向上
- マネジメント・テクニックの向上
- CASE ツールの発展や利用環境の整備(これは生産性よりも品質向上に役立つ)

等が考えられます.

しかし,先日の第10回 ICSE のパネル討論では,技術移転が,人と人との間,研究開発コミュニティと産業界の間で,あまりうまくいっていないことが問題にされ,ソフトウェア工学が,まだ,ハードウェアの自動車工学や土木工学,あるいは電気工学のように,技術者たちの日常の意志決定の中にはに生きていないといった共通認識が語られていました.

対策としては,社会学的アプローチの重要性,成果物中心の研究,ソフトウェア工学を特定領域にカテゴリ化することなどが提案されています.社会学的アプローチは,De Marco が「ピープルウェア」で説いており,ソフトウェア工学のカテゴリ化は,講師自身が ISO に提案している標準化テーマの1つです.

3. 何故失敗するか

ソフトウェア・プロジェクトの失敗には,いくつかの,特定の原因があります.

その第1は,ソフトウェアそのものの本質的な性格である無形性です.つまり,ソフトウェアが,直観的に見えないものであるため,それを動かしてみて,はじめて失敗に気付くことが多いということです.このため,欠陥が目に見え,それにすぐに気付く場合が多いハードウェアとちがって,修正のループが大きくなり,打つ手が遅れます.

また,この無形性のため真の原因を誤って認識することによって,混乱を増幅させてしまう場合も多いようです.よくあるケースは,技術的原因を人手不足と誤認することです.この場合には,ただ,やみくもに配員を増やすという安易な解決策が採用され,F. Brooks のいうところの「Man-Manth の神話」の落し穴に管理者を陥れます.結果的に,そうした助っ人の効果は少なく,ただ,はなはだしいコスト超過だけがもたらされるのです.

また,ソフトウェアの無形性は,よい技術やアイデアが伝わりにくく,自己流が横行し,気の利いた工夫などを実物から学ぶ機会が少ないという現象を生みだします.作ったものが目に見えないために,プログラマたちは,えてして"とにかく動きさえすればよい"というメンタリティに陥り勝ちになるです.

次に,客観的評価方法が乏しいということが挙げられます.ソフトウェアの規模,工数,コスト,生産性などの評価基準は,一般にあいまいで,品質評価のポイントも,人によってまちまちです.また,評価の前提としてのソフトウェアの価値観さえ,人により異なっています.これでは,まるで綿アメをゴムの物差しで測るようなものです.

もう1つは,すでに述べたように,ソフトウェア開発が人間的要素を多く含むため,社会学的,心理学的な考察がかなり重要になって来ます.このために,プロジェクトの先行きの予想が難しくなり,また,それは人の能力差によって,かなり大きく影響されます.

すなわち,管理対象としての無形物(=ソフトウェア)は,有形物(ハードウェア)に比べて扱いにくいということです.

以上をまとめてみますと

1) 直観的に見えない

2) 客観的に評価しにくい
3) 全体象がはっきりつかめず，常に変化する
4) 人の知的創作物ゆえ能力差の影響が大きい

ということになります．

逆に，ソフトウェアが持つ唯一の大きな利点としては，ソフトウェアが情報であるために，管理対象としての生産物そのものと管理情報とを，１つの一貫した情報システムの上に統合できることです．このため，最近の開発環境には，管理ツールをそうした意味で統合したものが多くなりました．ソフトウェアの自由度は，ハードウェアの場合と比べて格段に大きく，これを放置すると自己流が横行することになってしまいますが，逆にうまく利用すれば，一貫性のある開発システムを作ることができるでしょう．

4. どうやって克服するか

4.1 リスク・マネジメント

失敗を克服するには，"失敗してあたりまえ"という前提に立って考えることが必要です．

失敗には色々の視点があり，赤字や一寸した納期遅れも失敗には違いありませんが，最も避けなければならないのは"混乱"です．一旦プロジェクトが混乱すると，数倍のコスト，半年～数年の遅れなど，桁違いの悪影響が生じるばかりでなく，個人の生活にも悲劇をもたらします．

こうした混乱のもとは，大きな問題を見逃してしまうこと，および，トラブルの真の原因を誤認することから起こります．そうした大問題をできるだけ早く見つけ出し，速やかに処置すことが先決で，それは，あたかもガンの治療（処置）に似ています．

そこで，転ばぬ先の杖として，プロジェクトを開始する前に，あらかじめ予想される問題をリストアップし，その処置を考えて置こうというのが，リスク・マネジメントのアプローチです．

問題は，たとえ発生しても，それを小さなうちに解決すればする程，少ない労力と安いコストで済みます．リスク・マネジメントは，１０種類のハイリスク項目を選び，その各々について起こり得る確率と，起こった場合の被害度を定量化し，それらを掛け合わせたたものを１つの指標として，その値を事前の対策によって減らそうというものです．

いいかえれば，ソフトウェアが持つ不確定性を定量化する科学的アプローチだといえましょう．

そこでは，また，優先順位の誤りに対してダイナミックな計画修正が行えるようなフィードバックのメカニズム，あるいは，ピープルウェアでいうところの結束したチームや，自発的に行動し品質に妥協しない人間を育てることも，失敗を防ぐ上で重要な要因になります．

4.2 構成管理

構成管理は，品質管理上の必須の管理手法として普及しているもので，アメリカでは，国家規格にもなっています．

この考え方は，もともとハードウェアから来たもので，機械の構成を管理するという考え方を，ソフトウェアにあてはめたものです．

つまり，ソフトウェアを工業製品として位置付け，システムの構成上はハードと同じように，それらの構成品目（コンフィギュレーション・アイテム）を定義し，それらにソフトウェア固有の絶えず設計変更が加えられるという性格を考慮して，変更を管理しようとするものです．

構成管理から見た工業製品としての性格は：

- 構成を持つこと ・ 階層的分割が可能（モジュール化）
- 仕様を持つこと
- 開発のライフサイクルを持つこと

の３つが挙げられます．

管理の対象としてのデータ・プロダクトは

- 要求仕様
- 設計仕様（機能，操作，全体，サブシステム）
- 製品構成
- インターフェース
- 基準線 ・ １つのコンフィギュレーション・アイテムが正しいかどうかの基準で，仕様変更に対して常に最新なもの

などです．

制度としての構成管理は，ライブラリ，ライブラリアン，構成制御委員会などの組織の確立，監査や構成制御を行なうための機能の確立，標準化などから成っており，それらは管理的色彩の強いものとなっています．

欧米では，構成管理はコンピュータ・システム化されていて，いくつかのツールにより支援されています．ツールは，自社開発の場合と購入ソフトの場合がありますが，欧米では一般的には購入ソフトの方が多いようです．日本では，あまりこれを実施しているという話を聞きませんが，なぜでしょうか．

これまでは，どちらかといえば精神論が幅を利かせてい

たようですが，これからは，こうした科学的アプローチが重要になるでしょう．

参考文献: ANSI/IEEE STD 828「ソフトウェア構成管理計画書」sp .5

4.3 フィードバック・メカニズムと可視性の向上

個人用のワークステーションを利用した開発環境とその上で動くツールの利用は，ソフトウェア開発に今まで考えられなかったアプローチを可能にしました．もはや TSS 環境化でホストのレスポンスに悩むこともなく，ダンプリストを追いかける必要もなくなって来ました．

また，今までは，開発用と管理用には，別のコンピュータが用いられることが多かったのですが，これからは，この両者を統合することのメリットが増大するでしょう．

欧米では，すでに開発環境の中に，構成管理やプロジェクト管理などの管理用ツールを統合して，生データによる問題の早期摘出に効果を上げています．また LAN を活用した分散開発環境も，問題の摘出と処理の迅速化に役立つでしょう．

また，適切なフォーマリズムを導入し，論理や構造の表現上を工夫することによって，無形物としてのソフトウェアの難点であった可視性も，著しく改善することができ，こうした道具をうまく使いこなせば，問題の早期発見と処置が今までよりはるかに容易になってきています．

しかも，こうした管理を，強制感をともなわずに自主的に行える環境が望ましく，このために，局所的自立性の付与と自主管理が行えるようなローカル・フィードバック・メカニズムを開発環境に組み込むことが，失敗を未然に防ぐ決め手になるでしょう．

4.4 結論

失敗を防ぐには，科学的アプローチを優先させるべきだと思います．特に日本では，これまで，精神主義の風潮が強かったようですが，問題の所在を科学的に明らかにした上で，それらを誠心誠意処置すれば，効果はもっとはっきりあらわれます．

早期発見・早期対処が特に大切なことですが，それには，開発環境と一体化した開発ツールや管理ツールの利用が有効です．

また失敗の中で最も恐ろしいのは混乱で，これには大きな問題を的確に把握することが重要なこです．これがデザイン・レビューの真の目的です．開発過程でひとりでに見つかるような小さな問題は，放っておけばよいのです．失敗を防ぐにはこうした重点感覚も重要です．

5. おわりに

長年この業界で働いている人々にとって「プロジェクトの失敗」は，重大な関心事であり，かつ日頃から脅威を感じていることではないでしょうか．ソフトウェア技術者の過酷な長時間労働の原因の多くが，ここに由来しています．

このセッションでは，主に欧米での事例を中心に話をしましたが，現在のアメリカで，De Marco の「ピープルウエア」に代表されるように，効率第一の管理主義に疑問が持たれ，ヒューマン・ファクタの重視が求められているのに対し，日本では，逆に，精神主義（つまりよきにつけ悪しきにつけ人間本位の物の考え方）の横行がさまざまな弊害をもたらし，合理的・科学的なマネジメントの必要性が叫ばれているのは，興味深い現象です．

２つのまったく対照的なアプローチが並列的に存在し，なおかつ，どちらでも同じようなソフトウェア・プロジェクトの失敗が起こっているわけです．恐らく欧米流の科学万能主義，日本流の精神主義のどちらでも，真の解決策は得られないでしょう．

アメリカで「ピープルウェア」が叫ばれた背景には，科学的アプローチの壁があります．その逆に，日本で今必要なのは，精神主義の限界を打ち破るための科学的アプローチの確立だと思います．それによって，問題そのものの意味とその位置付けを的確にとらえることができます．西欧風の言葉で言えば Divide and Conquer の原理，科学的実証主義の導入です．

こうした観点に立って，今後，日本と欧米との間の交流を深め，お互いに学び会うことが問題解決の一つの糸口となるのではないでしょうか．

われわれソフトウェア開発に携わる者にとって，考えなければならないことは，失敗が起こったとき，現場サイドで力まかせにつじつまあわせをしてしまうことではなく，その真の原因は何か，どう対処していったらいいかということを，事実を前にして真剣に考え，それを組織的に実行することであり，そのための科学的マネジメントを育てていくことではないでしょうか．

［報告者注］このセッションは，時間不足のため自由討論ができず，そのため，当初期待していた生きた体験談をあまり聞くことができませんでした．また，あらためてこのテーマについて議論する機会を持つことを期待したいと思います．

ＳＥＡ秋のセミナーウィーク　'８９
セッションＢ２

NeXT のソフトウェアアーキテクチャ

講　師：吉平　和浩（キャノン）　Michael j. Smolenski（ＳＲＡ）
報告者：西郷　正宏（日本システムウェア）

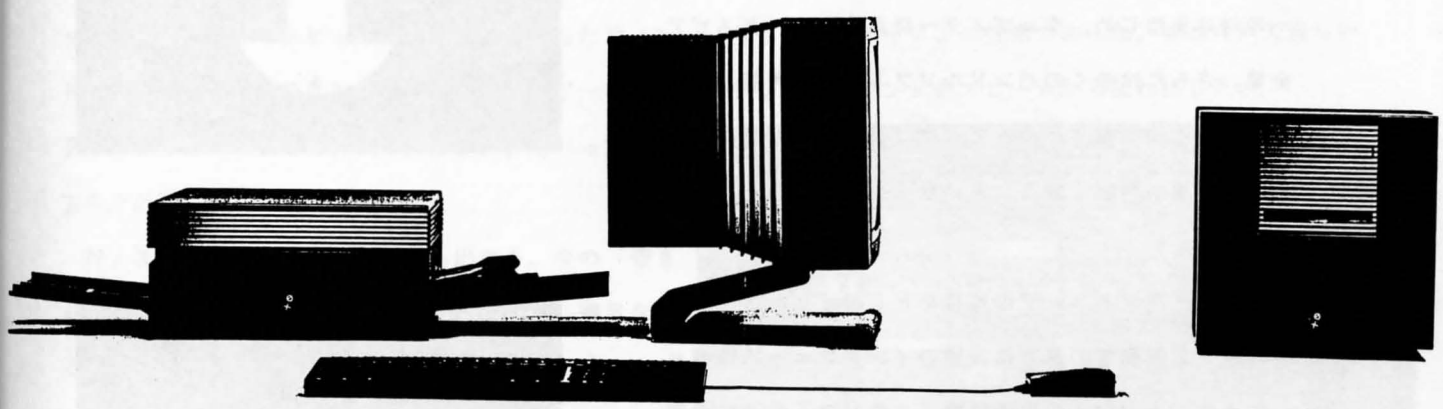
1. はじめに

現在最もＨｏｔなコンピュータであるＮｅＸＴが、遂にこの７月、日本でもデビューを果たした。かねてよりＳＥＡでは話題に上っていたマシンだが、今回は実物が登場してのＨｏｔなセミナーとなった。

前半は、キャノン・マーケティング部門の吉平氏が７月に行われたジョブズ氏の講演で使われたスライドを用いて説明され、後半はＮｅＸＴのトレーニングコースを受講して来られたＳＲＡのM.J.Smolenski氏が、マシンデモを交えてＮｅＸＴのソフトウェアアーキテクチャを『ＮｅｘｔＳｔｅｐ』を中心に説明された。

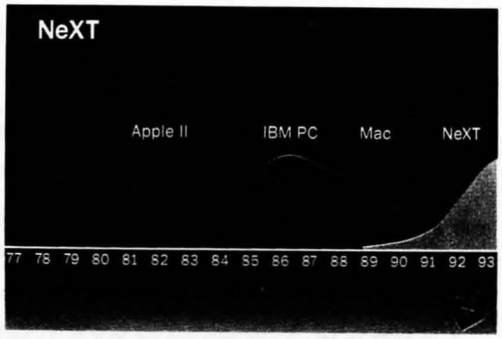
セミナーの主な内容は以下の通りであった。

●ＮｅＸＴの基本的な考え方
　・ＮｅＸＴの開発思想
　・ハードウェアの構成と特徴
●ＮｅＸＴのソフトウェアアーキテクチャ
　・ソフトウェア構成と特徴
　・ＮｅｘｔＳｔｅｐの概要
　・ｲﾝﾀﾌｪｰｽﾋﾞﾙﾀﾞｰによるﾌﾟﾛｸﾞﾗﾐﾝｸﾞの実演



NeXT Computer System

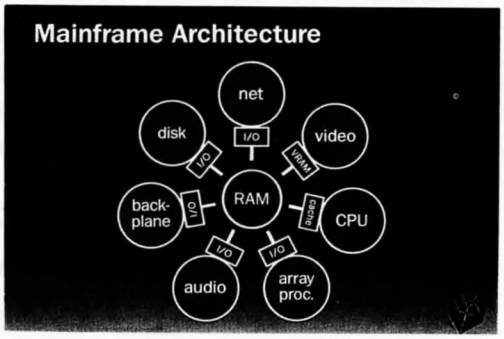
2. ＮｅＸＴの基本的な考え方 -Philosophy-



①．９０年代の１０年間がＮｅＸＴの時代

＝「コンピュータの寿命は１０年間」という認識＝

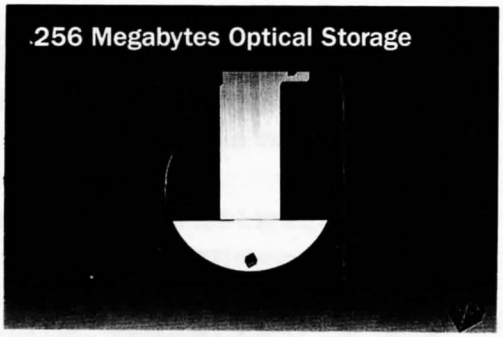
AppleⅡ、IBMPC、Macintoshはそれぞれ77年、82年、84年からの10年間を創ったが、ＮｅＸＴは90年代の最初のコンピュータとして位置付けられ、開発された。



②．パーソナル・メインフレームというアーキテクチャ

＝ＰＣでもワークステーションでもないマシン＝

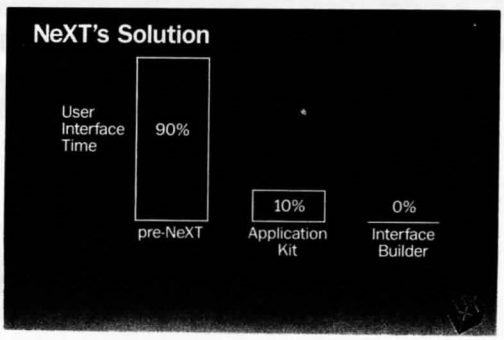
ユーザフレンドリなマンマシンインタフェース、ディスプレイ・プリンタ統一のPostscript、組み込まれたネットワーク、サウンド等これらを実現する為に、全てのＩ／ＯチャネルにＩ／Ｏプロセッサを置いたメインフレーム・アーキテクチャを採用した。



③．世界中どこでもシステム環境ごと持ち運べる

＝ソフトもライブラリもバンドルする光磁気ディスク＝

ＮｅＸＴが光磁気ディスクに執着するのは、イメージやサウンドといった大きなメモリを必要とするメディアへの対応をはじめ、ウェブスター辞典やシェークスピア全集、さらには多くのバンドルソフトを収納可能とすることで「この１枚さえ持って行けば世界中どこでも自分だけの環境が実現出来る」というコンセプトの為である。



④．プログラマとユーザの垣根をとるＮｅｘｔＳｔｅｐ

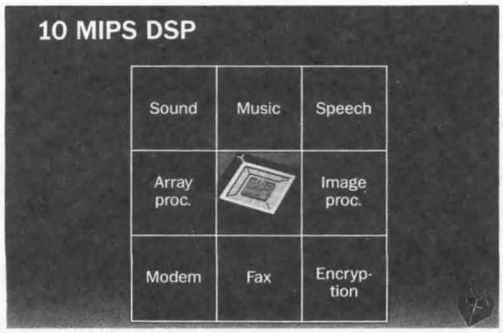
＝ユーザと開発者の双方に共通のインタフェース環境＝

マッキントッシュの出現以降、今やソフトウェアの作成に占める比率は90％ともいわれるユーザーインタフェース部分の構築作業比率を０％にすると同時に、専門家でないと使えなかったＵＮＩＸを全ての人の手に届ける為の統一操作環境の提供を行うというのがＮｅｘｔＳｔｅｐの開発思想である。

⑤．新しいメディアへの対応

＝未来を向いて設計されているマシン＝

テキスト、グラフィック、イメージ、サウンド、音声、それらの組み合わせによるマルチメディアへの対応、そしてアニメ、ビデオといった動画の時代へと対応できるよう設計されているのがＮｅＸＴの特徴。２つのＶＬＳＩチップも、メインフレームアーキテクチャもＤＳＰの搭載も３枚追加できる拡張ボードも、光磁気ディスクの装備も、これらは全てこれからの新しいメディアへ向けて設計されている。



3．“しゃべる”ＮｅＸＴメールの実演

ＮｅＸＴの独創性をいちばん分かり易く説明する為に電子メールの実演が行われた。

ＮｅＸＴの電子メールは文字のフォントやサイズも自由に選べるという特徴もあるのだが、やはり最大の特徴は「リップサービス」であろう。

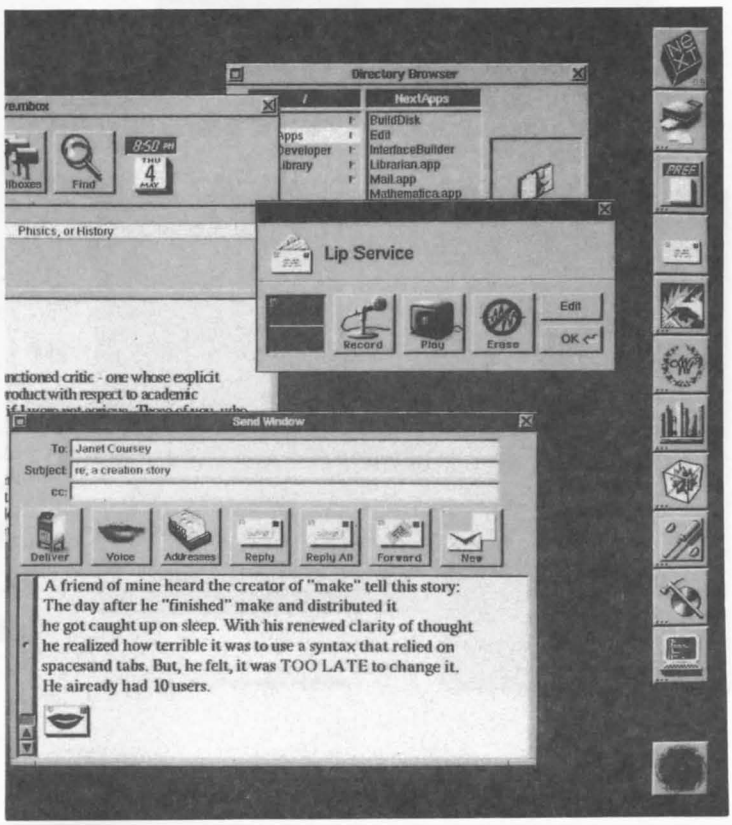
まず、音声付きのメールが送られてきた場合だが、自分宛の電子メール上にリップマーク（唇の絵）のアイコンがシールの様に貼ってある。そのリップマークをクリックするとマイクやスピーカのアイコンが入ったウィンドウが現れる。ここでは声を聞きたいのでスピーカをクリックすると、「皆さん、こんにちわ。」といった送り主の声が聞こえてきた。

ＮｅＸＴではさらに音声の編集も出来る。今の「皆さん、こんにちわ。」の波形を画面上に出して、適当なある一部の波形をコピーしてつなげ、再びスピーカをクリックすると、「皆さん、こんこんにちわ。」としゃべり出した。つまりコピーした部分は「こん」という部分だったという訳である。

送る場合は全くその逆で、マイクのアイコンをクリックしてディスプレィ後部のマイク端子につないだマイクに向かって好きなことをしゃべり、好きなように編集してスピーカで確認した後、Deliverと書かれた郵便ポストのアイコン（日本のとは形が違うのでちょっと分かりにくいが）をクリックすればＯＫである。

今回は、ジョブズ氏が日本でのデモンストレーションショウで使用したマシンをそのまま使用した為、キャノンの田中専務からジョブズ氏宛のメールになっていて、リップマークのシールが二つ付いていた。

一つは田中専務からジョブズ氏に対する挨拶であり、もう一つはキャノンの社歌であった。女性の声で美しいデジタルサウンドがスピーカから流れ出した。



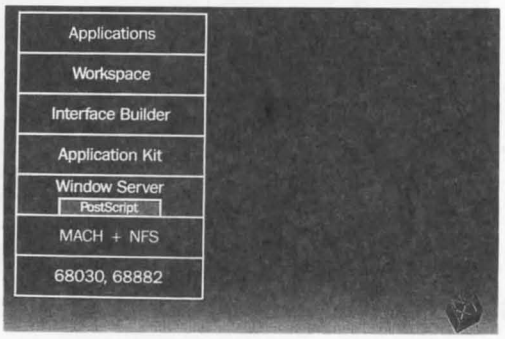
4. NeXTのソフトウェアアーキテクチャ

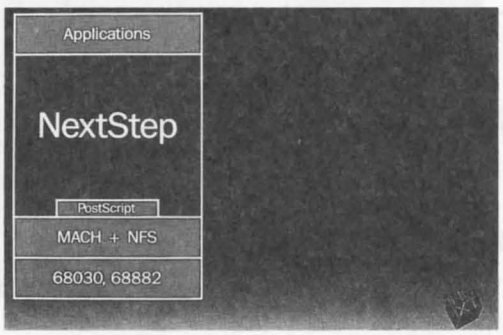
ＮｅＸＴのソフトウエアは右の図の様な階層になっている。

1．ＭＡＣＨ（マーク）
2．Display PostScript
3．ウィンドウサーバー
4．アプリケーションキット
5．インタフェースビルダー
6．ワークスペースマネジャ
7．各種アプリケーション

（3．〜6．：ＮｅｘｔＳｔｅｐ）

このうちの３．から６．までがＮｅｘｔＳｔｅｐと呼ばれるユーザインタフェース開発・操作環境であり、徹底したオブジェクト指向によって作られている。





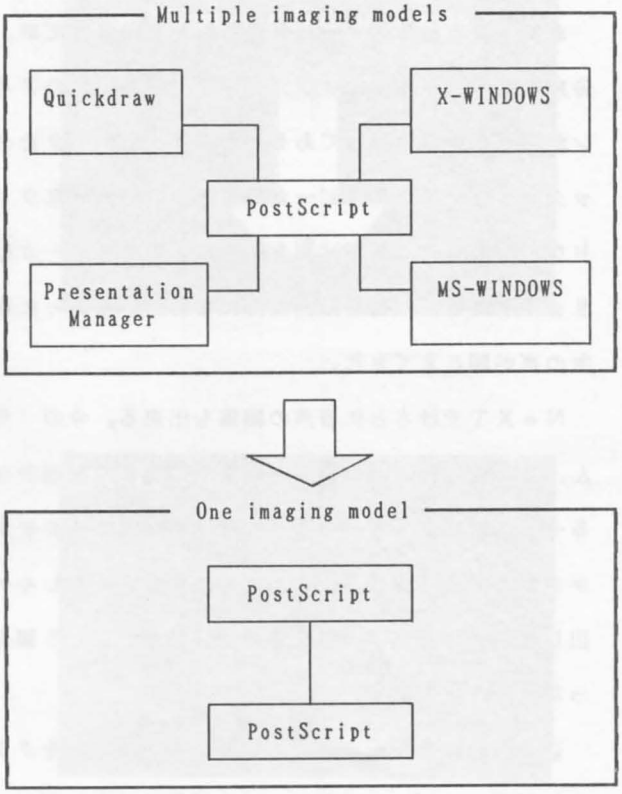
(1).分散型ＯＳ・ＭＡＣＨ

ＭＡＣＨはカーネギー・メロン大学で開発された分散型ＯＳで、ＭＡＣＨが目指すのはネットワーク上に接続された複数のマシンをあたかも１つのシステムであるかのように効果的に協調して動作させる事である。

基本的にはＵＮＩＸの４．３ＢＳＤと互換性があり、“デスクトップＵＮＩＸ”と呼ばる程コンパクトなカーネルを持っている。

(2).スクリーン表示言語・Display PostScript

Adobe Systems社のページ記述用言語PostScriptは、アウトラインフォントを用いたページプリンタの実質標準と言われる程普及しているが、ＮｅＸＴはAdobe Systems社と共同でDisplay PostScriptを開発し、画面表示にもこれを用いて真のＷＹＳＩＷＹＧ（What You See Is What You Get）を実現すると共に、アプリケーションにとっての統一的な表現手段を提供した。（右の図）



(3).ユーザインタフェース環境・ＮｅｘｔＳｔｅｐ

前にも述べたように、ＮｅＸＴのユーザインタフェース環境であるＮｅｘｔＳｔｅｐは、次の２つの目的を持って開発されている。すなわち、

１．プログラマに対するﾕｰｻﾞｲﾝﾀﾌｪｰｽ開発負担の軽減

２．ユーザに対する統一的で自然な操作環境の提供

である。この２つの目的を果たすために、ＮｅＸＴは徹底した階層化とオブジェクト指向を行っている。

ウィンドウサーバー

すべてのイベントを管理するタスクマネジャであり、イベント起動型になっている。マウスクリックやキーボド入力等のイベントを受け取ってアプリケーションオブジェクトにイベントレコードとして渡す。また、画面への描画命令を受けるとDisplay PostScriptを使って実際のビットマップイメージを展開する。

アプリケーション・キット

プログラム開発用のオブジェクトライブラリであり、Macintoshのツールボックスに相当する。ただし、ツールボックスが４００ものサブルーチンで構成されているのに対し、クラス化と属性の継承（インヘリタンス）を持たせることによって数十のオブジェクトに圧縮してある。これらのオブジェクトコールだけで各種のウィンドウやメニュー、ボタン等のグラフィックインタフェースを簡単に作ることが出来るようになっている。

ワークスペースマネジャ

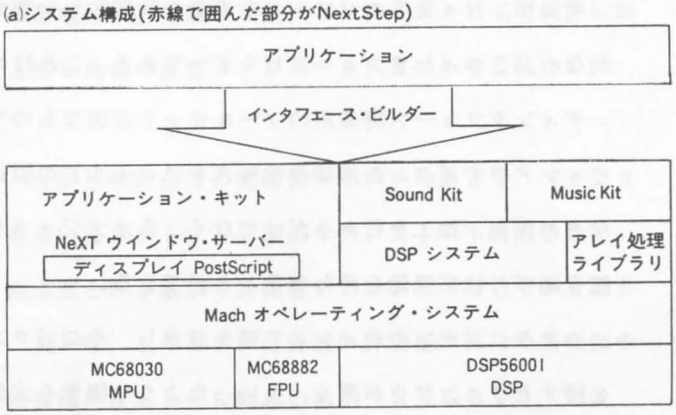
Macintoshではデスクトップに相当するスクリーン全体を、ＮｅＸＴではワークスペースと呼んでいる。

従ってワークスペースマネジャの仕事は、ワークスペースの管理とファイルやディレクトリへのユーザのアクセス管理であり、これは、MacintoshではMultiFinder 、ＵＮＩＸではShellに相当するものである。

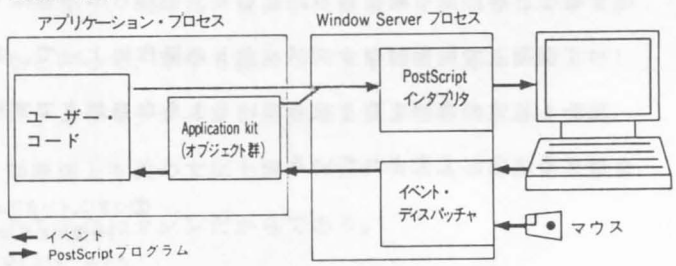
ワークスペースマネジャによって表示される初期画面のディレクトリブラウザは階層構造が把握し易い。

●NextStepの概要。(a)はシステム構成, (b)は動作時の制御の流れ

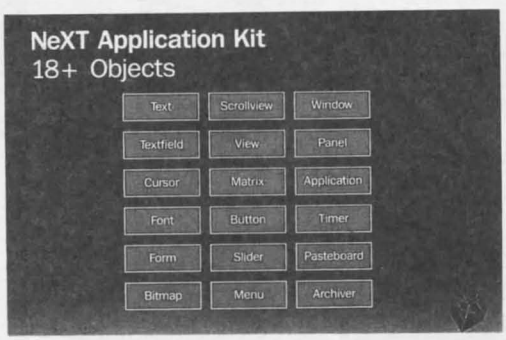
(a)システム構成(赤線で囲んだ部分がNextStep)

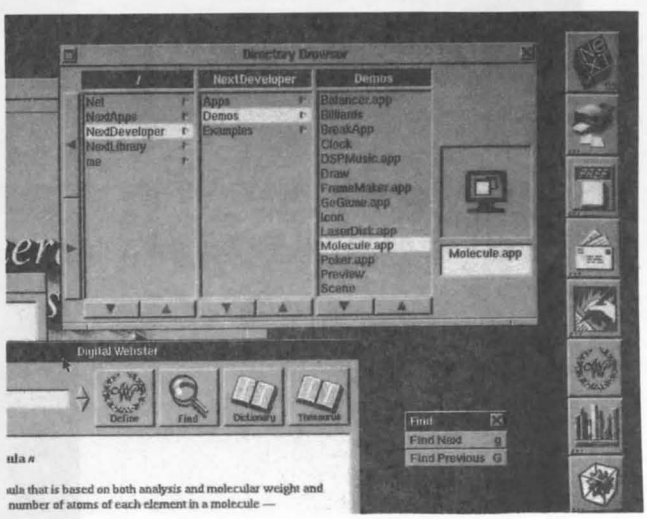


(b制御の流れ



（上図は、日経ＢＰ社・日経ｺﾝﾋﾟｭｰﾀ別冊「ｿﾌﾄｳｪｱ」
「ﾏﾙﾁﾒﾃﾞｨｱ時代のﾕｰｻﾞｰ・ｲﾝﾀﾌｪｰｽ」P.45、図７より）





インタフェースビルダー

そして、ＮｅＸＴのソフトの中で最も画期的かつ独創的なのがこのインタフェースビルダーである。これはユーザインタフェース開発用のツールキットの様なもので、ウィンドウやボタンの形状やテキストフィールドの設定などの編集・加工を行えるだけでなく、各オブジェクト間を結び付けて機能を持たせることができる。

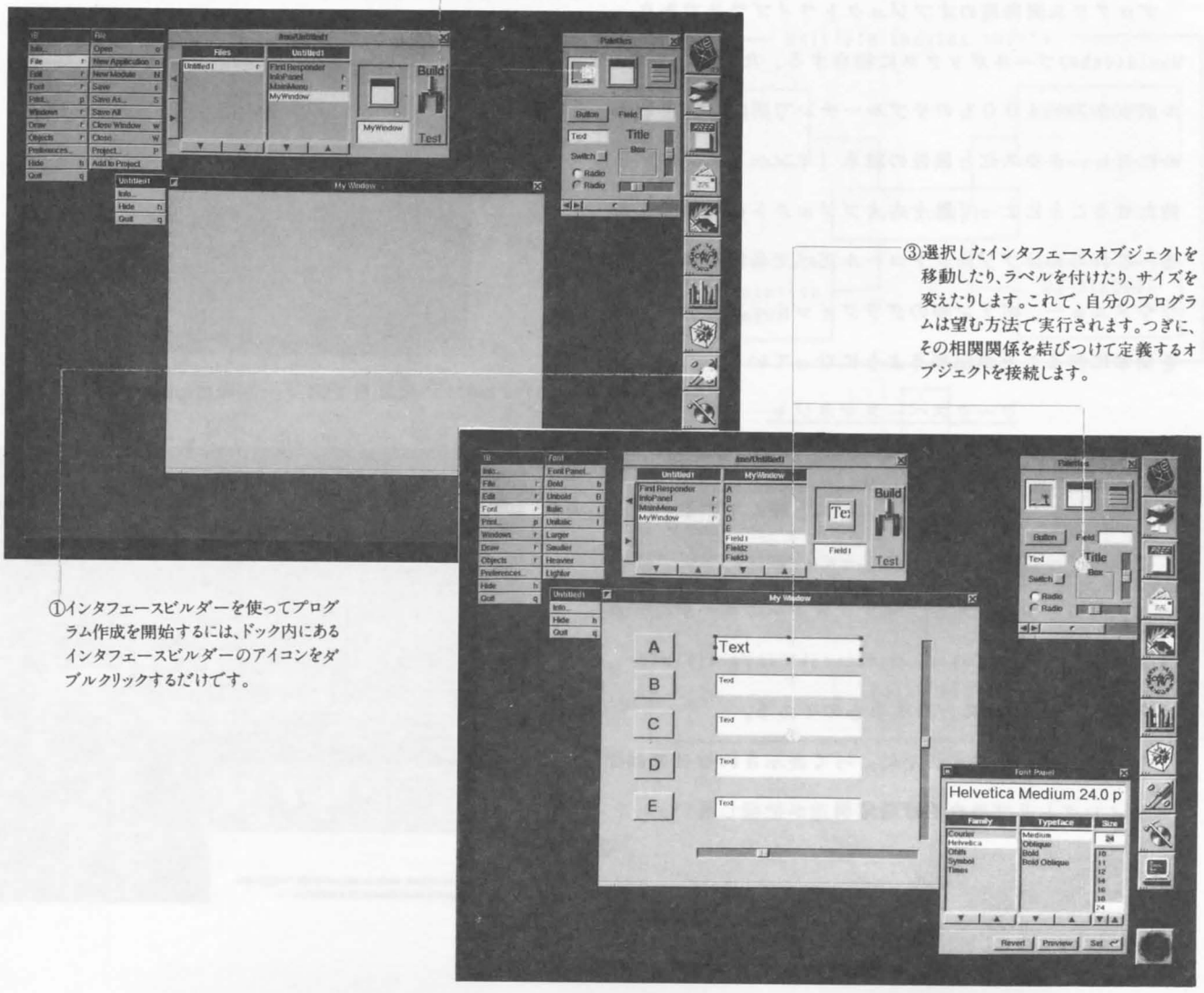
つまり、ボタンやウィンドウ等を設定し、そのボタンを押すとウィンドウが開く、といったような機能を定義することが出来る訳である。そして、それらの作業はすべて画面上で視覚的なオブジェクトの操作によって、あたかも日常の事物を取り扱っているような自然さで実現できるように工夫されている。

インタフェースビルダーを用いた簡単なプログラミングの実演

当日のセミナーでは、簡単なプログラミングの実演が行われた。インタフェースビルダーを起動し、新しいウィンドウを定義し、次にこの中に、右上のパレットと呼ばれるウィンドウからボタンやスクロールバー、テキストフィールド等を取り出して並べた。そして、バーを動かすとフィールドの中の数字が変化するようにバーとフィールドをコネクトしたのだが、これは二つのオブジェクトをCTRLキーを押しながらマウスでドラッグして結ぶという極めて直感的な作業で行えた。

プログラミングが終わると、ディレクトリブラウザの右端にあるスイッチをBuildからTestに切り替えるとプログラムは実行可能な状態となる。ここでセットしたバーやボタンを動かして動作を確認する。修正したければまたスイッチをBuildにして設定し直せばよい。

②つぎに、インタフェースビルダーのパレットを使用して、ボタンフィールドおよびメニューのような各種のインタフェース機能を選択します。それを自分の希望するウインドウに入れます。



③選択したインタフェースオブジェクトを移動したり、ラベルを付けたり、サイズを変えたりします。これで、自分のプログラムは望む方法で実行されます。つぎに、その相関関係を結びつけて定義するオブジェクトを接続します。

①インタフェースビルダーを使ってプログラム作成を開始するには、ドック内にあるインタフェースビルダーのアイコンをダブルクリックするだけです。

5. 感 想

とかくＮｅＸＴは話題に事欠かないマシンである。アップル社の生みの親でありマッキントッシュの育ての親でもあるジョブズ氏がカムバックを賭けて世に問うマシンであり、アドビ社と共同開発したDisplayPostScriptやＩＢＭ社にライセンス供与が決まっているNextStepなど何かと業界の注目を一身に集めていると言ってよい。

特に私はインタフェースビルダーに革新的な予感を覚えた。その予感とは「これは、ユーザとプログラマの垣根をなくすデザインではないか」という予感である。（ユーザをプログラマにしてしまうというよりもプログラマをユーザにしてしまうといった感じ）それは丁度、銀行のＣＤが利用客とオペレータの垣根をなくしてしまった（利用客をオペレータにしてしまった）ように、或は電話のダイヤルが、利用者と電話交換手の垣根をなくしてしまった（利用者を交換手にしてしまった）ように、である。そしてその後、銀行のオペレータや電話交換手という職種がより専門性の高い一部の限られた仕事になったように、コンピュータのプログラマという職種も、ＯＳやプログラミングツールを開発するごく一部の人達だけを指す時代がくるのだろう。（つまりその時こそ「今にプログラマの人口が世界の人口を上回る」という預言が成就する時である）このソフトにはそんな世界を肌で予感させるものがある。

逆にこのＮｅＸＴを今の時代の目で見ると随分ムリをしたように見える点やムダな点、不要な機能等がある様に見える。これは、ＮｅＸＴが現在の世の中に対して最適化を図っているのではなく、少なくとも５年は先の世の中に対して最適化を図って作られているようにも思えるのである。マッキントッシュが世に出た時、アラン・ケイが「批評するに足る最初のパーソナルコンピュータ」と評したように、ＮｅＸＴは来たるべきコンピュータの家電時代を先取りした「インテリアとしての批評に耐え得る最初の家電コンピュータ」なのかもしれない。

だから現時点での光磁気ディスクの遅さや、プリンタあるいはネットワークまわりのソフトの不安定さなど、完成度という基準で評価をしてしまうのはどうかと思う

我々は、このように自分達の評価基準にない異文化の価値観にはもう少し敬意を払うべきであろう。

つまり、現時点の世の中に最適フィットをしているからといって、半年も経たないうちに時代遅れになってしまうようなデザインを絶賛するような完成度至上主義、効率至上主義の文化土壌からは、ＮｅＸＴは決して生まれてこないマシンだからである。

5. おわりに

ＮｅＸＴはおろか、ワークステーションもＵＮＩＸも全く触った事が無いという実に無謀なレポータの私は、このレポートを書く為に当然勉強を始めざるを得なくなった。雑誌の関係記事をあさり、ジョブズ氏の講演ビデオを見、キャノン主催のＮｅＸＴフェアにも出向いてセミナーを聞き体験スクールにも立ち寄った（おかげでＮｅＸＴのＴシャツを貰えた。嬉しい！）そして受講後１ヶ月も経った最近、ようやくセミナーで聞いた言葉が幾つか理解できる程度になった。恐らくＳＥＡ史上最低のレポータに違いないが、力強くご支援下さったキャノンの吉平氏、ＳＲＡのSmolenski氏に支えられてやっとこのレポートが出来た次第である。両氏並びにレポータをさせて下さったＳＥＡ事務局の方々に心から感謝を申し上げます。

（レポータ：西郷正宏・日本システムウエア）

5th ISPW に参加して

3年ぶりの ISPW

岸田　孝一
(SRA)

1. 初めに昔話を少し

この ISPW (国際プロセス・ワークショップ) にはなぜか縁があって，８４年２月の第１回から３回続けて参加している．

第１回目は，ソフトウェア進化論を提唱し，７０年代の半ばごろからプロセス問題に入れこんでいた M.M.Lehman 先生が，「そろそろ開発環境とそれが支援するプロセスとの関係をみんなで討論しようよ」ということで，開催されたものである．

しかし，冬のロンドンでのワークショップと聞くと，何となく寒そうで，もともと私は参加するつもりがなかった．ところが，８３年の秋，別の用事で欧州へでかけたついでに，インペリアル・カレジの研究室を訪ねると，「日本からまだ１人も応募がない．きみ，今夜ホテルでポジション・ペーパを１ページ書きたまえ！」という．とうとう無理やり引っ張り込まれてしまった．

この１回目の会議は，開発パラダイムとプロセス・モデルについての一般的議論がテーマだった．まだ，プロセスはおろか，統合的開発環境の概念に対する一般の関心がそれほど高まっておらず，たとえば，プロトタイピングのアプローチを導入してウォータフォール型プロセスに風穴を開ける試みなども，TRW の Boehm さんあたりが USC での小規模な学生実験を始めたばかりのころであった．

ワークショップ主宰者の Lehman 先生は，ちょうどポーランドの Turski 先生と一緒に，フォーマル・アプローチにもとづくプロセス・モデルの体系化に熱中されていた時期で，(その後日本でも私や日立の野木さんあたりが宣伝して有名になった) 例の２本足の LST モデルが，会議でも大いに話題になった．セッションの進め方は論文発表と質疑を中心とする形で，ワークショップというより，シンポジウムに近かった．

２回目は，８５年の初夏にロサンゼルス郊外のリゾート・ホテルで開催された．今度は，自分が ICSE の委員会に深く巻き込まれていたので，Riddle, Balzer といった仲間たちとの打ち合わせを兼ねての参加になった．

開会冒頭から，新技術のインパクトに関する哲学論争(環境をめぐる概念の階層化とその解釈)が激しく展開され，アメリカ式討論のやり方を目のあたりにして，たいへん興味深かった．個別の技法の優劣ではなく，技術というものの本質にズバリと切り込んだ哲学的な議論がこうした場でもできるのだと，目のさめるような思いがした．

たしか，TRW の Boehm さんがスパイラル・モデルによるリスク・マネジメントの考え方を初めて提案したのは，このワークショップであったと思う．まだかれ自身のアイデアがよく固まっていなくて，プレゼンテーションの意味が理解できない部分があった．そのことを，ロス空港へ帰る車の中で Riddle さんと話題にしたことをおぼえている．

第３回は，私と Bob Balzer が Co-Chair を務める ICSE-9 のプログラム委員会に引っ掛けて，ロッキー山中のスキー・リゾートで，８６年の晩秋に開かれ，日本からは，鳥居 (阪大)，斎藤 (慶応大) の両先生も参加された．

何しろ海抜３千数百メートルの高地での討論だったから，酸素不足と早口の英語で，いささか頭が痛かった．技術的な話題としては，L.Osterweil さんのプロセス・プログラミングの提案が，もっともホットであった．実は Bob と私の策略で，この話題を盛り上げて ICSE のキーノート・スピーチに推薦する手筈になっていたのである．

他には，Lehman-Dowson のコントラクト・モデルとその支援環境 ISTAR も新しいトピックスであった．「日本には弁護士が少ないから，こんなモデルは役に立たないだろうネ」と隣に座っていた AT&T の人が，半分真顔でいっていたのを思い出す．

2. ワークショップの運営

この第３回目まで，セッションの運営形態は，いずれも何人かのスピーカの発表のあと，それをめぐっての自由討論というスタイルであった．

８８年の５月にふたたびイギリスで開かれた第４回 (私自身は出席できなかった) から，それが，各セッションごとにキーノーターをおき，かれ(かの女)が全員のポジション・ペーパを読んだ上でスピーチをし，全員でセッション・テーマについての討論をするという形に変わった．

だいたい，欧米におけるこの種の技術ミーティングは，問

題意識を持った人がボランティアで最初の企画を立て，どこか適当な組織（学会など）の名前を借りて，とりあえずスタートするのがふつうである．そして，内容が面白ければ，何回か続けて行われる．そのうち，集まった人びとの中から，自然発生的にやはりボランティアの推進委員会が形成され，持ち回りで，企画や事務処理が引き継がれて行く．

いまでは，国際的にもっとも権威のあると認められている ICSE も，最初は１５年前に，当時はまだ中堅の研究者・技術者であった Belady, Boehm, Ramamoorthy, Yeh などの人々が，ソフトウェアの高信頼性に関する国際会議をロサンゼルスで開き，それがうまくいったので，２回目から名前を変えて現在にいたっているにすぎない．

８２年に東京で第６回が開かれた時には，まだ推進委員会の構成や企画運営のルールなどがはっきり固まっておらず，共同開催の準備にあたった日本の学会の人たちがかなりいらいらしていたのを思い出す．

そういえば，われわれのソフトウェア・シンポジウム（やその他のイベントも）も，そろそろ推進委員会の組織を明確にして，企画・運営のガイドラインなどを整備しなければならないだろう．

ISPW の場合は，最初のいいだしっぺだった Lehman 先生は，忙しいせいかいまは企画から手を引いて，プロセス・モデリングに関心の強い研究者・技術者のグループが，推進役を引き受けている．今回の運営は，第４回のスタイルを踏襲し．３日間を半日ずつ６テーマのセッションに分けて，討論が行われた．

3. 討論の印象

今回の討論のテーマは，「プロセス記述の経験に学ぶ」であった．このテーマから一部の（特に初めてこのワークショップに出る）参加者が期待したのは，いわゆる地に足のついた具体的な討論であったと思う．

しかし，ISPW はもともとそうした個別的な議論を行なう場ではなく，たとえ背後にどんな具体的な事例を踏まえていたとしても，それを一段昇華させた形での抽象的・哲学的なやりとり（よい意味でもわるい意味でもウィット・コンバット）を行なう場として，これまで運営されてきた．

今回も議論の主導権を握ったのは，そうした ISPW マフィアたちであって，聞き手のメンタリティ如何によって，よしあしの評価がまったく２つに分れるようなスタイルで（私はそれが好きだ），議論が展開されることになった．同行した中川・中島両氏のレポートを読むと，お２人がかなりいらいらした様子が感じられる．次回の日本での 6th ISPW をこのままの路線で進めるべきか否かは，やはり意見の分かれるところであろう．

Control のセッションで Balzer が提起したプロセス・プログラミング対プロセス・プランニングという図式はおもしろかった．とはいえ，私自身はアンチ AI の純粋ソフトウェア工学派なので，プランニングのパラダイムに全面的賛成というわけではない．プロセス・プログラミングの問題点は，それがややプロダクト指向的なアプローチでプロセス記述に挑んでいることに起因しているのではないか，というのが個人的な感想である．

4. 興味をひかれたペーパー

いずれ正式のレポートが刊行されるだろうが，配布された討論資料の中から，私の興味を引いたポジション・ペーパをリスト・アップしておく．いま，プロセスに関して，どんな話題が議論されているかを探る手がかりとして参考になれば幸いである．

Boehm & Belz (TRW): プロセス・モデルの選択にともなうリスクをスパイラル・モデルで回避するというのは，１つのアイデアだと感じた．プロセス・モデル・デシジョン・テーブルとは，まさにアメリカ的発想．

Curtis (MCC): ここにもまた，プロセス対プロダクトというパラダイム変化の１例が見られる．

Finkelstein (Imperial College): 協調プロセスのモデル化に対する１つのアプローチ．

Garlan (Tektronix): 組込み型ソフトの開発プロセスのフォーマル記述（Z による）の実践．たしかに１つの実例ではあるが.....?

Huff (GTE): プランニング・パラダイムのプロセス記述への応用．SEA のツアーで参加した前回の SDE シンポジウムでもかの女の発表を聞いたのだが，その時はあまりピンとこなかった．

Humphrey (SEI): スーパー・プログラマの個人的プロセスをどうとらえ，環境はそれをどう支援すべきか，という問題提起．この人は，SEI で産業界における組織的なプロセスの調査などを熱心にやっていたので，こんな問題意識を持っていたとは以外だった．

Kaiser (Columbia Univ): ごぞんじ MARVEL での戦略的なプロセス・モデリング．批判したらすぐやりこめられそうで，コワイ！

Kellner (SEI): Statemate をつかった現実的プロセスのモデル化．なかなか頑張ってますネ．

Lehman (LSTA): 最後のテーブル（モデルの役割）は思

考の整理に便利．しかし，ISPW を始めた張本人が今回欠席だったのはさみしい．

Osterweil (UCI): この人も欠席だった．おかげで Control セッションの議論が盛り上がらなかった．

Penedo (TRW): 有名な SPS プロジェクトの後を受けて，もうかなり長年続いている PMDB プロジェクトの総括．日本でも，ソフトウェア DB の構築・管理に関して，このくらい息の長いプロジェクトをやらなければ！

Roberts (Praxis): Alvey IPSE2.5 におけるプロセス・モデリング．欧州では，この他にも Eureka Software Factry Project（今度 ICSE-12 のツアーで訪問する予定）でも，プロセス・モデルのフォーマルな記述をまじめに試みている．某 SIGMA あたりも少しは見習ったらどうかしら？

Thomas (YHP): 目標指向の活動としてプロセスをとらえないと大局を見失う危険があるのではないかという，鋭い指摘．

Zave (AT&T): AI と SE の境界領域では，アプリケーション・ドメインに関する理解や知識をどう扱うかが大いに問題になる．

5. 最後にコマーシャル

いよいよこの１０月には，6th ISPW が日本（函館）で開かれる．私も PC メンバ（というよりローカル・アレンジメント）として，会場（大沼プリンス）の手配を開始した．実は，１月のテクニカル・マネジメント WS も，私にとっては，そのためのロケハンをかねていたのである．

日本でやるからといっても，コトバはやはり英語だという点が，われわれにとってはかなりのハンデだが（特に概念的な議論は苦しい），みなさん，ぜひポジション・ペーパを応募してください．

ISPW の感想

中川　中
（SRA）

1. はじめに

５回目を数えるプロセス・ワークショップは Maine 州の Kennebunkport のリゾート型ホテルで 37 名の参加を得て開催された．自然と人手とが綾織る美しいパノラマに囲まれては，３日足らずの監獄生活も楽しいものである．

今回のテーマは「プロセス・モデルの使用経験」である．ソフトウェア・プロセス，プロセス・モデルあるいはプロセス記述は数年来ソフトウェア工学の一大分野を形成しており，数多くの研究成果が報告されているものの，具体的な開発現場においてそれらの成果が適用された例は現在に至ってもそう多くを数えない．

そのためか，あるいは参加者の性向によるものか，議論の多くは抽象的であり，さもなければ我田引水であり，討論の価値は乏しかったといわざるをえない．そのものずばりの指摘が第３セッションの終了間際に Marc Kellner よりなされた．

2. セッション各論

討論は６つのセッションより構成された．各セッションのテーマは「現在の技術レベルのレビュー」，「機構（メカニズム）」，「政策」，「データ」，「制御」，「今後の問題，総括」であったが，議論の実態は必ずしもこれらの言葉の意味するところとは一致しない．たとえば，「機構」と銘うたれた第２セッションでは，一般にソフトウェア・プロセスにおける単位と考えられている「活動（アクティビティ）」の定義に関する恐らくは不毛な議論が Mark Dowson によって開始された．

組織委員の Dewayne Perry および議事進行役を務める Colin Tully の開会宣言の後，第１セッションが Bill Curtis の軽妙なお喋りで始められた．何が議題であったかは殆んど記憶に残っていない．「実施可能な（enactable）」ソフトウェア・プロセス・モデルを論じるというテーマが不快であった所為もあるが，既述のとおり，現状では空を翔ける抽象論にならざるをえないことがその主な理由である．Robert Balzer，Curtis 外の掛け合いばかりが目立つセッションであった．

同様の指摘はプロセス・モデルの機構・文法を論じる片山卓也の基調スピーチで始まった第２セッションにも妥当で

ある．討論が失速したのは言葉の定義をめぐり20分もの幕合いが続いた時である．用語の定義に一致がえられれば討論中に不要な誤解を生じないというメリットは認められる．しかし，特定の機構や言語構造を取り上げるわけでもないこのセッションにおいて活動とはなんぞやと問うてみても致し方ない．ことほど多様なモデルや言語が提案されている状況だからである．その後討論はさらに空洞化を進め，全く関心を失くしてしまった．

このセッションの途中，Maria Penedo, David Garlanらによる分野を特定したモデル化ないし実践の意義に関する議論が沸騰した．Balzerが指摘したとおり，分野を限定する試みはその限定が一般性を致命的に損なわない限りで価値があることは(もちろん，特定のプロジェクトで特定の効果があげられたという事実は直接の関係者にとっていずれにせよ重大であるが) 明白である．

分野を特定して成功したという報告例は，ソフトウェア・プロセス論に限らず，この点を当閑に伏す傾向がある．一般化を意識している，と言うのはたやすいが，機構を考える上でそれをいかに意識したか，するか，すべきかを説得力のある論理で展開した例はあまりない．実際に一般化されたモデルその他が実効果を生じた例証を要求するのは苛酷に過ぎるといえようが．

第３セッションはプロセス・モデルにおける当為（すべき，あるべき）をテーマにDowsonのスピーチから始まった．さきの「機構」が文法（syntax），この当為が意味（semantics）であるという用語使いには大変な違和感があるが，それはともかく，「政策」というテーマでソフトウェア・プロセスを正しく導くべき指針を考える，というのが本セッションの狙いであった．

討論は政策という名に何でもぶち込んでしまえふうの粗雑な枠組で進められた．とくに，Naftaly Minskyの類推論法は討論を台無しにしてしまった．比喩はしばしば効果的な議論を醸成するが，ソフトウェア・プロセスにおいて政策という言葉から想像されることどもと統治における政策のあれこれとを対置して得られる利益は，彼の論法のようである限り，なにもない．類推は本質に曖昧な部分を宿しているだけに，それを用いる論理は厳格であらねばならない．

終ってみれば，政策の（again!）定義をめぐる最初の応酬がセッション全体のトーンを決めてしまっていたのである．もともと，基底を成す「機構」の姿が前のセッションで一向に明確にならなかった以上，それを覆うような位置にある当為の存在を論じる意義は希薄である．

第４セッションはソフトウェア・プロセスにおけるデータの扱いに焦点をあてた．基調スピーカはこれまでだんまりをきめこんでいたJack Wiledenであり，オブジェクト管理の重要性が指摘された．Balzerはこれに対し，ソフトウェア・プロセス論に特有のオブジェクト管理上の問題があるのかという疑問を表明した．構成(configuration)管理を自己の研究分野とするPeter Feilerをはじめ，Wileden, Gail Kaiser, Ian Thomasらが解答を試みたが，いずれも納得のいくものではない．

私こと中川は勝手に彼らオブジェクト管理派に互選され，ポカンとしていたものである．伝統的な意味でのオブジェクト管理はソフトウェア・プロセス論にとり既与として扱われるべきである，というBalzerの暗黙の主張には説得力がある．セッションの途中からためにする自己主張が目立ち始めた．とくにFeilerは議論の帰趨を弁えず独善的であった．

第５セッションのBalzerの基調スピーチは，Karen Huff等に代表されるプランニングをソフトウェア・プロセス・モデルあるいはモデル記述言語の基底に据える試みの是非により参加者を二分することから始まった．この種の色分けは争点を明確にする効果があり，これまでに発言の少なかったWilhelm Shaefer等が議論に加わるという効果を生んだ．目標(goal)のアジェンダとそれを逐次変更し最終目標を達成するためのプランからなるパラダイムが非決定性や流動性の支配的な状況に適すること，その由来からして自明である．

問題は，このパラダイムが他の選択肢との比較においてどうであるのか，また「目標」の形態・意味等を明確にするという前提条件とこのパラダイム自身との重要性の比重はどうであるのか，に集約される．後者はWiledenがオブジェクト中心思考との相補性という形で指摘した．後半はDavid Carr, Clive Roberts, Feilerがそれぞれの立場からセッションのテーマである「制御」について簡単な発表を行なった．総体としてみるとこれらは相互に無関係であり，またしても討論は闇に覆われた．

最終セッションは欠席のため知り得ないが，想像をたくましくすると，以下のような総括で終ったのではあるまいか．

3. おわりに

ソフトウェア・プロセスの話題はソフトウェア工学におけるおおよそすべての話題を包含しあるいは交錯するだけに，ワークショップのような小規模の会合にも非常に多様

な関心が集う．プロジェクト管理上の議論に熱心な Carr はひとつの極であり，特定のプログラミング環境を常に想定してしまう David Notkin はまたひとつの極である．人間の主体的意思決定に重点をおく Anthony Finkelstein とプロセスは実行（execute）するものであると決め込んだ口調の Thomas とはいずれが水かいずれが油かであり，幾人かの宙に浮いた抽象論者と具体例に力点を置く Penedo 等とはまともな討論を譲り合わない．しかしながら，異なる背景や指向を持つ者共が議論を戦わせることそのものは健全な行事である．

本ワークショップが（おそらく参加者の大多数が認めざるを得ないように）失敗に終った理由は，すでに指摘したとおり，すべてテーマの設定にある．浅すぎる，あるいは狭すぎる経験が経験談議を水で薄めたスコッチにしてしまった．次回のワークショップに向けては，セッションのそれも含めたテーマの設定に真剣であらねばならない．片山が記述方法の優劣を比較するための共通問題の必要性を指摘したのも（すでに「片山の挑戦」と命名された），議論の空洞化を避けるべき手段としてである．ただし，Pamela Zave の指摘のとおり，特有の問題は特有の解法に向いているという了解が解法を比較する上で枢要なことは強調されねばならない．

最後に，自身の立場が現在どう変わったかを述べる義務がある．私のペーパーで披露した立場は，Wileden のそれと奇妙に似ている．その理由は，意味をもつ実体，たとえば（「ファイル」ではなく）文書，をモデル化しようとすれすれば，実体，その間の関係およびそれへの操作の3要素に帰着せざるをえないことにある．とくにそのモデルが抽象的であれば，他との差異は用語や対象範囲だけである．今回ペーパーは掲載されていないが，Perry の立場もある抽象レベルでは非常に似ている．

進路が分岐するのは，形式性をどこまで強調するか，またどう形式化するか，においてである．この点において，厳格な形式性のみがモデルを意味あるものにするとする私の立場がペーパーでは十分に述べられていない．また，その背後にある形式化可能な部分あるいは形式化すべき部分とそうでない部分との峻別も，十分強調されていない．これらの主張はより具体的なモデルを規定する際に明確にする必要がある．

議論をたどっていて一般的に感じた印象は，ソフトウェア・プロセスは機械的に実行されるものであるという意識が強いことである．とくに，用語にその傾向が反映されている．たとえば，implementation, execute, control, instantiation などなど．これはおそらくプロセス・プログラミングの台頭の影響である．人的要素が「制御」の問題にすり替えて論じられるのは，こうしてプログラミングとのアナロジを濫用した結果である．

5th ISPW 報告

中島 毅
（三菱電機）

1. 会議の概要

ISPW-5 は，1989年 10月 10日から 13日の３日間にわたって，米国メイン州で開かれた．このワークショップの主旨は，ソフトウェア開発プロセスをモデル化し，管理や作業者支援を行なう技術を議論することである．会議は，６つのセッションに分かれており，次のように進行した．

- 各セッションにはキーノータがいて，OHP を使って話の口火を切る．
- 発言はキーノータが喋っている途中でも終ってからでもできる．
- 発言したければ手を上げる．
- 司会の C.Tully がそれを控えておいて次々に指名していく．

発言は，ほとんどの場合１分程度の短いもので，OHP を使う人はほとんどいなかった．

2. 内容

以下に，各セッションのテーマと議論のようすについて記す．

(1) 現状のレビュー

Keynoter: Bill Curtis (MCC)

このセッションの目的は，集められたポジションペーパから技術の現状と問題点を洗い出すことである．彼は，まず「Enactable の意味は？」「どんな効果があるの？」という問いからスタートし，種々のプロセスモデルが提案されているが，それらが何をコントロールしようとしているのか明確でないと指摘した．彼の主張は，次の二点である．

- 個人・チーム・プロジェクトの各レベルでは異なるプロセスモデルが必要である．
- 個人のレベルの問題解決作業をコントロールしようとするのは悪いアプローチであり，チームコーディネーションをコントロールすべきだ．

(2) 機構

Keynoter: 片山卓也 (東工大)

このセッションは，モデルのシンタックスに関する議論が目的であった．片山先生は，種々のモデルを共通問題によって評価しようという提案をした．これは，Katayama's Challenge と呼ばれ後で参照されていた．このセッションで話題になったのは，Finkelstein の会話論理である．これは，協調作業する人間間のやりとりを形式的に記述し記録することを狙っている．

(3) ポリシー

Keynoter: Mark Dowson (sda)

このセッションにおけるポリシーは，種々のプロセスモデルの機構の差を吸収するためのゴミためと化していた．ポリシーとは具体的に何だというところで議論が紛糾していたのが印象に残っている．ポリシーと機構をどこで切り分けるのかということが問題になっていた．議論がいきづまったセッションの最後で，M.Kellner が「ここは経験について話し合う場じゃないの？」と怒り出し，B.Curtis が「社会科学者でも呼んで聞いた方がいいんじゃないの」と議論を一蹴したのが印象的であった．

(4) データ

Keynoter: Jack Wilden (UMass)

J.Wilden は Arcadia で苦労しているだけあって，プロセスモデリングにオブジェクトベース特に Typing System の必要性を強調していた．タイプモデル・永続性モデル・インタオペラビリティ・版管理・名前づけなどの問題を解決すべきだと言っていた．しかし，「プロセスのためのデータ記述は特殊かそうでないか？」という議論も出て一時紛糾した．

(5) コントロール

Keynoter: Bob Balzer (USC/ISI)

B.Balzer の議論は興味深く，参加者の注意を引き付けた．まず次の仮説を立てた．

「プラニング言語は，プロセスモデルの基盤として望ましい」

スクリプトではなく，プランのインスタンシエーションとしてプロセスの定義があるという立場である（詳細は K.Huff のポジションペーパ）．彼はその仮説を基に，参加者をその賛成者と反対者に分けた．結果は，賛成 22，反対 7，その他 4であり，かなり大勢を占めた．彼の予め用意された結論は，「プロセスはプログラムされるのではなく，プランされモニタされ再プランされるものだ，プラ

ンならプロセスの発展を自然に取り扱うことができる」というものである．一部には「結局またAIか」という批判も出ていた．

(6) 出てきた問題

Keynoter: Sam Redwine (SPC)

参加者の半分は，イソイソと引き上げてしまっていて議論は盛り上がらなかった．

3. 技術的な傾向

今回の会議では，程度の差こそあれ各々の研究者が実行可能プロセス記述言語へ取り組んだ結果，以下のような反省点が挙がってきているように思う．

1) 記述の目的

個人の問題解決プロセスの記述は難しいし，これをコントロールしようとする試みはアプローチとして間違っているのではないか．

課題は，大規模ソフトのチームによる開発にあるのではないか．

2) 言語の問題

プロセスは動的である．手続き的なプロセスプログラミングのアプローチでは，例外や発展を自然に取り扱うことが難しい．

3) 記述の対象

プロセスはドメイン依存である．プロセス記述言語の一人歩きは危険である．ドメインの理解を深め，その理解の基にプロセスを一般化すべきである．

これらの問題点を踏まえて，今回のISPWでは次の3つの方向が出てきているように思う．

1) チーム協調のコントロール

・チームの作業支援（CSCW）－ MCC

・協調作業の形式的記述 － Finkelstein

2) より豊かな記述言語の模索

・リフレクション － 片山

・Statechart － Kellner

◎プラニングパラダイム － Huff

3) ドメイン理解の重要性の再認識

◎ドメイン理解 － Zave, Garlan

4. 感想

この会議では技術的側面についての議論より宙に浮いたような概念論に終始し，技術的でないおしゃべりが続いていたように思う．下名もプロセス記述実験をいくつか行っているが，詳細な記述をすればするほど，詳細なプロダクトの記述が必要になることがわかってきている．しかし，参加者の中には，ソフトウェアオブジェクトの特殊性を軽視する発言があるなど，プロセスの記述に真面目に取り組んでいる人が少ないのではないかと思えてくる．

今回の参加者は皆壮々たるプロセスの研究者であり，一つ一つのアイデアは素晴らしいと思うが，問題点のおき方が少しずれているのではないかというのも素直な感想である．つまり，そのアイデアのほとんどは，既存の言語や支援システムをスタートにしてその応用例をプロセスに求めている．そのためプロセスの何を書き何を支援したいのか（記述の目的）を見失っているようにみえる．B.Curtisは最初のセッションで，そのあたりを明確にしようと考えたのだと思う．

本来は，記述言語の優劣を稚拙な例で比較するより，実プロセスの例を基に実プロセスがどんな性質をもっているかを探求し，その後にそれに合う記述言語をさがすべきだと思う．しかし，今回の会議の雰囲気はその方向を許さないものであったし，これからもそうではないかだろうか．もしそうだとすると，ISPWは今後も実りのない議論を続けることになるのではと予想してしまう．

Position Paper for ISPW-5

Kouichi Kishida

Software Research Associates, Inc.
1-1-1 Hirakawa-cho,
Chiyoda-ku, Tokyo 102
JAPAN

1. Travel with/without Map

The role of a process model in software development is similar to the role of a guide-map for a traveller. Suppose you are visiting an unknown city, there are a variety of situations to conside.

CASE 1:

You have a street-map and you also have a detailed list of routing instructions to reach your destination. In this case, your trip will be very effective, but with little fun. Maybe you will miss some interesting points of the city.

CASE 2:

You have a map, but the destination point is specified only by its address. You must find out a route to reach there by examining the map. In this situation, sometimes you may get lost. But, with the help of the map, you will be easily able to find the right direction to proceed in.

CASE 3:

Your map is not a detailed one. Only major streets and buildings are shown on it. The destination address is given, but you have no idea where on the map it is located. In this situation, you are likely to get into a number of problems. How serious are they ? It depends upon your expertise as a traveller. Also, possibly, you could enjoy some exciting adventures on your way to the destination.

CASE 4:

You get into the city without any kind of map. In this case, you can only rely on your imagination, namely a white map of the city in your mind (a conceptual process model).

2. Discussion on Process Models in the SDA Project

In the past years in the SDA (Software Designer's Associates) project, there have been a series of discussions among the participants of the project about modeling, describing or programming software design processes.

University professors have applied their own formal techniques to construct detailed scripts for some systematic design methods such as JSD, and demonstrated sample executions

of their scripts using fancy prototype software tools.

A common feeling among the industry participants is as follows:

"Yes, we agree that these scripts are beautiful and easy to manipulate. But, the processes in real world are very complicated and messy. It seems almost impossible to apply these beautiful formal scripts to actual project."

Returning to the analogy of a travel map, academia claims that it is convenient for travellers to have a well-organized street-map with a list of routing instructions. On the other hand, industry people say that such a map is useless because they are not travelling through modernized cities but straying upon villages in the wilderness.

Academia keeps suggesting the construction of a frame work of a modern city to wipe out uncivilized wilderness as the first step of travel. But industry thinks that such an approach is too radical and unrealistic. Rather they prefer more realistic or bottom-up approach, such as post-scripting actual design process and accumulating them. then we could find out some commonalities among them and extract an evolutional process model, or add some retrieval mechanism to that archive of design process records for future references.

3. Concept of Process Monitoring

I hope to fill this gap between the academic and industrial approaches to process modeling.

The key idea is to have a conceptual white map (instead of a real, detailed map) to guide and monitor the behavior of travellers.

In the case of software design, this conceptual process model is somehow enactable, and embedded in the heart of a design support environment to guide and monitor actual design processes.

The idea is still immature, but I hope I will be able to develop some realistic proposal by the time of the workshop.

Product-Based Process Models

Ataru T. Nakagawa
SRA Inc.
Tokyo, JAPAN

Kokichi Futatsugi
Electrotechnical Laboratory
Ibaraki, JAPAN

1 Guiding Principles

For a process model to be accepted by a variety of practitioners, it has to be flexible, in the sense that it can be easily and naturally adaptable to diverse software development scenes. It may, although not must, also be of help if it is derived evolutionary, in the sense that the model accepts the basic tenets of prevalent software practices with only essential modifications. We have to avoid, in this respect, mistaking an idealised software process for an actual one. Actual development processes are not linear, hierarchical, nor organised in any other neat way, both microscopically and macroscopically. Development phase/stages used in common parlance do not depict actual development scenes; such scenes consist of lots of feedbacks, implicit assumptions of products not yet seen, related yet asynchronous activities, and dynamic revision/modifications. Most of all, maintenance phases are bogus phases. Software maintenance involves all the activities of the initial development process, writ small. Problems that are said to be particular to maintenance phases, moreover, do arise in initial phases also.

For a process model to be a process model, on the other hand, it has to satisfy the raison d'etre of process models. There emphasised several motivations for process models according to occasions, standpoints, and/or models themselves, some of which are quite unrelated if not incompatible. Most of all, technical aspects and managerial necessities are sometimes intermingled in discussing process models. In this paper we shall ignore the latter and focus on how to organise technically advantageous software processes. In particular, on how to make processes transparent, predictable, and, as a corollary, mechanical.

To realise the goals just mentioned, we cannot avoid describing software processes by some means. Hereon arises a seemingly inescapable antinomy; the more detailed a description is, the more limited its relevance and in many cases the more objectionable it looks to practitioners. At times it is claimed that detailed descriptions can help develop software efficiently. We have grave doubts of that claim; even if some experiments were enlisted as its supportive evidences and we accepted *their* efficiency measurements, a glance of descriptions used in such experiments would convince us of their very strict limitations in applicability.

We believe, however, detail descriptions need not have such drawbacks. For a process to be mechanised, in any way, it has to be described up to (down to?) being mechanically comprehensible. For now and for foreseeable future that degree of comprehensibleness requires stating every nitty-gritty of the process. We should not stop attempting to describe software processes. We need, instead, change directions in searching for relevant description methods. We suspect the matter is not whether such methods are procedural or declarative, as some researchers indicate.

One of our current research objectives is to establish a process model and a description method that satisfy, in considering all of the above observations and much else besides, at least the following characteristics.

- the model is describable in detail
- the model is familiar to prevalent software processes
- the model is flexible and adaptable

- the description method can describe software processes in great detail
- the description method is expressive enough for the model
- the model and the description method have rigorous semantics

Note that we do not use the word language here. A description method subsumes a language as its core and includes pragmatics and environments of the language, in addition to its syntax and semantics.

2 Against Activity-Based Process Models

Existing software process models, including such methodologies as JSP/D, invariably decompose a software process into a set of unit activities and relations among them. Reflecting this proclivity of process models, existing description methods also describe a software process as a hierarchy, control flow, dependency, and/or relations of activity units of various granularity. Put another way, those models and description methods try to organise software processes in terms of activities.

But step back a little and watch carefully, and such activities are defined, in turn, in terms of such products as documentations, programme codes and test cases. So an obvious question arises: why not try to organise processes directly in terms of products? Indeed, why not. After all, we have a nice analogy in the sphere of programming languages. Broadly (very, very broadly) speaking, there are two classes of programming languages, here we tentatively christen instruction-based and data type-based. The second class contains languages based mainly on abstract data types and so-called object-oriented ones. The first class contains everything else. We should consider, in the sphere of software processes, product-based models in addition to or in place of activity-based ones.

This idea is not new. Inferential programming and transformational programming are among ideas that have turned on products to guide software processes. Our ambition is to go further and construct product-based process models and description methods that have more rigorous semantics with equally rigorous attention to flexibility and practicality.

3 Product-Based Process Models

In this section we briefly explain one way of achieving the aim proclaimed in the last sentence, by presenting very abstract process models, or rather process paradigms. The notations are informal and the definitions are intuitive.

(1) Gorgonian model

First of all, we need a picture of the products, which we call the target system, of a software process. For a target system S that is not a toy programme, some decomposition and structurisation is de rigeur. The model recognises this necessity in the following definition.

> Given a set Π of product types and a $\Pi \times \Pi$-sorted set B of relation types, S is a pair $\langle E, R \rangle$, where E is a Π-sorted family of product sets and R is a B-sorted set of binary relations over E.

with requirement that

> Each product is independent of the other products, sharing no common part, and no two products can interact except via relations between them.

In typical cases, Π will contain types of documents, programme codes, test cases and approval signatures. B will contain such relations as is-part-of, is-document-of, sends-message-to and is-required-by. We do not

preclude a hierarchical composition of products; Π may contain a type of, say, subsystems, that are none other than target systems, and so on.

We then proceed to define activities that constitute a software process.

> Given product types Π and relation types B, and given a $\Pi + B$-sorted set A of activity types, an activity α of sort $\pi \in \Pi$ is an affectant on a product in E_π, and an activity of sort $\beta \in B$ is an affectant on a relation in R_β.

An activity on a relation changes the structure. We require that

> An activity directly affects only a single product or a single relation.

With no congenital restriction, the definition below ensues.

> A software process is a collection of activities.

These definitions, although extremely flexible, ensures visibility of a software process by way of the facts that (a) each activity that constitutes the process directly affects only a product/relation, and (b) indirect effects, if any, of an activity can be traced solely via explicit binary relations. This process model is Gorgonian, allowing haywire movements. For example, the model in no way prevents programme codes appearing earlier than documents supposedly describing them.

(2) rational Gorgonian model

It is generally agreed that the earlier it is detected, the less expensive an error. Put positively, it is desirable that a target system is correct, by some definition, at any time of its development. We capture this consensus in the following definition. For product types Π and relation types B, assume there defined correctness for each $\pi \in \Pi$ and $\beta \in B$. Then

> A target system $S = \langle E, R \rangle$ is correct iff all $\epsilon \in E$ and $\rho \in R$ are correct.

The correctness thus defined ignores sufficiency or completeness of the overall system, which has to be treated separately. Nevertheless, it can direct software processes in the following way.

> A software process is a collection of activities that preserve correctness.

A process model that so requires is rational Gorgonian, allowing only rational haywire movements.

(3) directed Gorgonian model

The Gorgonian model, rational or irrational, can be adapted to various kinds/extents of disciplines by means of, inter alia, (a) changing the allowable activity types, (b) changing interpretation of correctness of each product/relation, and (c) collecting activities in some organised way. For example, by restricting activities to decomposition cum refinement cum implementation, thinking of everything as correct ipso facto, and demanding a rigid products sequel, we get a traditionalist process model: vanilla plain waterfall.

We show a few more fruitful disciplines. We first try to synchronise activities to avoid conflicting effects. As the effects of an activity is eventually a function on products, it suffices to require that activities that may affect the same products not occur simultaneously. Define conflict-free collections on this line and we get

> A software process is a conflict-free collection of activities (a safe Gorgonian model).

And another example. A rational Gorgonian model is hard to obtain if products and/or activities are fine-grained, since in such situations an activity, however useful, is liable to destroy correctness. So let us loosen the definition arbitrarily and have

A software process is a collection of activities that preserve correctness at selected junctures.

No doubt this definition is wooly, so let us be more precise. If an activity destroys correctness of product/relations, other activities on them should be withheld until they recover correctness. But such recovery of correctness itself requires activities on those product/relations. This confusion suggests we distinguish activities into two types, active and reactive, the former motivated by external decision making and the latter solely concerning correctness recovery. Let us accept this suggestion and define

A correctness-preserving activity sequence is an — ungainly phrase, methinks — active activity followed by a sequence of reactive activities that ends with a correct system.

Then we have a fairly rational Gorgonian model that states

A software process is a collection of correctness-preserving activity sequences.

4 A Description Method

The models informally described in the previous section are not meant to be of actual use per se; they are not even parametric models, to be instantiated to particular models. Instead, they provide guidances to construct process models when specific methods and environments are chosen. And it is time we considered process description methods.

As they stand, our models are best described as (mutable) algebra, products constituting the carrier of sorts and activities being operations over them. Naturally, then, we regard algebraic specification languages[1] or their extensions[3] as *the* most promising candidates for process description languages, and are thinking[5,6] of creating environments based on one (family of) such language, OBJ[2,4]. We do not recite all its powerful features here, but would like to explain in what way it is suitable for the immediate purpose. Especially,

- Rich semantics. Its model theoretic, proof theoretic, as well as operational semantics provide us with many-faceted and rigorous means of analysing features of, and assessing, process models.
- Abstract construction. OBJ strides virtually unrestricted gamut of abstraction, thus can cover any level of granularity (if sometimes too stretched).
- Texts and semantics are vigorously distinguished but closely and uniquely related. What is written is not automatically what is meant — we need to interpret the texts for that —, so flexible construction is possible *without* compromising semantics.

Just to show the flavour and how abstract construction can be, we sketch the rational Gorgonian model in OBJ3 below.

```
obj APRODUCTTYPE is
sort AProductType .
op correct? :
  AProductType
    -> Bool.
--- stuff
endo
```

```
obj ANOTHERPRODUCTTYPE is
sort AnotherProductType .
op correct? :                    --- and the other
  AnotherProductType             --- product types
    -> Bool .
--- stuff
endo
```

```
obj RELATIONTYPE is pr APRODUCTTYPE . pr BPRODUCTTYPE . --- &c
sort RelationType .
sorts RelationTypeAB RelationTypeAC ... .
subsorts RelationTypeAB RelationTypeAC ... < RelationType . --- &c
```

```
ops rel1 rel2 : AProductType BProductType -> RelationABType .
--- &c
op correct? : RelationType -> Bool .
--- stuff
endo

obj SYSTEM is
pr SET[APRODUCTTYPE] * (sort Set to AProduct) .
--- &c
pr SET[RELATIONTYPE] * (sort Set to Relation) .
--- &c
sort System .
op system : AProduct BProduct ... Relation -> System .
op insert : AProductType System -> System .
op delete : AProductType System -> System .
--- &c. these operators enumerate activities
op affect? : AProductType BProducttype System -> Bool .
--- &c
op correct? : System -> Bool .
--- &c
eq insert(A,system(S,S',...,S")) = system({ A } $ S,S',...,S") .
--- stuff
eq affect?(A,B,system(S,S',...,S")) = rel1(A,B) isin S" or rel2 isin S" .
--- stuff
eq correct?(system(phi,phi,...,phi)) = true .
eq correct?(system(E $ S,S',...,S")) = correct?(E) and correct?(S,S',...,S") .
--- stuff
endo

reduce in SYSTEM :
   correct?(system({ a(1) },{ b(1) },...,{ rel1(a(1),b(1)) })) .
   ---> correct?(a(1)) and correct?(b(1)) and correct?(rel1(a(1),b(1)))
--- &c.
```

References

[1] Futatsugi,K., *Experiences in Algebraic Specification/Programming Language Systems*, internal report, Electrotechnical Laboratory, 1989

[2] Futatsugi,K., Goguen,J., Jouannaud,J.-P., and Meseguer,J., "Principles of OBJ2", *Proc. 12th POPL*, ACM, 1985, pp.52–66

[3] Goguen,J. and Meseguer,J., *Unifying Functional, Object-Oriented and Relational Programming with Logical Semantics*, Tech. Rep. SRI-CSL-87-7, SRI International, Jul. 1986

[4] Goguen,J.A. and Winkler,T. *Introducing OBJ3*, Tech. Rep. SRI-CSL-88-9, SRI International, Aug. 1988

[5] Nakagawa,A.T. and Futatsugi,K., "Stepwise Refinement Process with Modularity: An Algebraic Approach", *Proc. 11th ICSE*, IEEE, 1989, pp.166–177

[6] Nakagawa,A.T. and Futatsugi,K., "Software Process à la Algebra", internal report, submitted for publication, 1989

A Method for Recording and Analyzing Design Processes

Tsuyoshi Nakajima
Information Systems and Electronics Development Laboratory
Mitsubishi Electric Corporation

1 Introduction

We are trying to develop new design processes in which any designers can get level results. We have to not only define products to be created and procedure to be done, but know the creative part of design activities through analyzing many practical design processes.

It is clear that recording design processes is needed before analyzing them. Unfortunately, we now don't have the qualified method for recording[1]. Recording method for analyzing requires:

1. it can record not only products and procedure but design knowledge such as know-hows,
2. it doesn't disturb any designer's activities.
3. it can be traced in time order.
4. it allows us to view a design process from different perspectives and to organize it.

Recently gIBIS[2][3], which is a design support tool based on IBIS model, provides powerful recording facilities. However IBIS model emphasizes just one view of design processes, i.e. design discussion, and is not for analyzing design processes.

In this paper, we propose a method for both recording and analyzing design processes, which satisfies the above four requirements. The method allows us to record on-the-fly design processes, to organize them, and analyze them from multiple views.

2 Recording Method

Our recording method consists of

1. recording procedure, which defines what designers are to do and
2. description model, to which the designers add their knowledge later, and with which analysts can organize and analyze.

2.1 Recording procedure

During design activities, recording proceed following procedure described below.

1. Identify problems(a) to be solved, and write the starting time.
2. Solve the problems in your own methods. In this phase, you have to write down the following things:
   (a) notes and figures(b') used for solving the problems,
   (b) what you think as much as you can(c'),
   (c) what you refer, such as existing notes and figures(b), and

(d) problems(a') risen in 2.

3. Repeat 1 2 until all problems are solved.

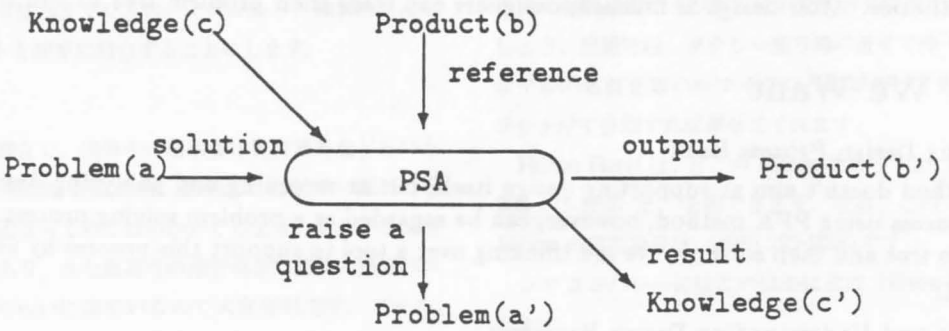
4. Make a fair copy of the record and add some information(c) which was needed for each problem solving activity.

2.2 Description model

By using our recording procedure 1-4, designers produce three kinds of information: problems(a,a'), products(b,b'), and knowledge(c,c'). It includes a lot of useful information, but it is so huge that other designers and analysts cannot understand and analyze it. It is necessary to organize this recorded information so that they can view it from different perspectives and abstract it.

For this purpose we propose a process description model, called Problem-Product-Knowledge Model (PPK model). According to PPK model, shown in figure 1, a set of these information(a,a',b,b',c,c') is connected with a problem solving activity (PSA). This allows us to regard a design process as a network of these activities.

Our method consists of 2.1 recording procedure and 2.2 PPK model. We call it PPK method.




```
where
input    problem(a)      problems to be solved
         product(b)      products referenced
         knowledge(c)    thoughts and knowledge applied
                  e.g. design method, experience, or premonition
output   problem(a')     problems risen newly
         products(b')    products created
         knowledge(c')   thoughts and knowledge acquired
                  e.g. applicability of design method,
                       premonition of next problems,
                       or feeling of achievement.
```

Figure 1: Problem-Product-Knowledge Model

3 Characteristics of PPK method

PPK method has the following three advantages for analyzing.

1. It provides two different views for design processes

Different persons view a design process from different points of view. For instance, managers are interested in designers' activities and the resulting products, and designers and those who maintain the software products want to know how problems of design were solved.

PPK model allows us to view a design process from two different perspectives. First, if we extract only problems(a,a') and PSAs from a PPK description, we can get a problem tree, which represents a problem solving process. If we extract only products(b,b') and PSAs, we can get an Input-Activity-Output tree, which represents a product producing process.

2. It can be abstracted

   PPK description can be written in a hierarchical manner like Data Flow Diagram. PSAs which have same purpose can be gathered into a more abstract PSA. At the same time we can hide detailed information, such as trivial problems, intermediate products, and too specific knowledge.

3. Design knowledge can be added to it later

   In general, it is impossible, during design activities, to write down design knowledge, such as design methods, experience gained in previous design, and premonition of design risks. We rather think these knowledge should be added after the design is finished.

   According to PPK method, designers record their problem solving activities as a problem tree during design activities. After design is finished, designers can trace their problem tree to add their knowledge.

4 What We Want

1. Supporting Design Process Itself

   PPK method doesn't aim at supporting design itself, but at recording and analyzing design processes. A design process using PPK method, however, can be regarded as a problem solving process: designers grow a problem tree and then solve it. We are thinking over a tool to support this process by using hypermedia technology.

2. Authoring and Understanding Design Processes

   If a description of a designer's process is understandable for other designers, it will help them maintain and reuse the design, and learn design know-hows. A description of a design process written in PPK model is a kind of book about the design. It shows not only how products were produced, but also how problems rose and were solved, and what the design knowledge was. If one cannot understand the description, it means knowledge(c,c') of the description is not sufficient. Then the designer may add new knowledge to his description. This feedback helps designers author an understandable book on a practical design.

5 Conclusion

We proposed a method for both recording and analyzing design processes. We have described a practical design process and found the method was good for analyzing. We are planning to describe more practical processes.

Although some software process models and design methods have been proposed, very few analysis of practical design processes have been reported. In order to handle design processes as scientific objects, a design process description is required, which is good for analyzing. PPK method is able to meet these requirements.

References

[1] 4th International Software Process Workshop, *Representing and Enacting the Software Process,* ACM SigSoft, 1988

[2] J.Conklin and M.Begeman, *gIBIS: A Hypertext Tool for Exploratory Policy Discussion,* MCC Technical Report, No. STP-082-88

[3] C. Potts and G. Burns, *Recording the Reasons for Design Decisions,* Proc. of 10th ICSE, pp.418-427

シリコンバレーを楽しむために

— その3 —

大神原　薫
San Jose, California

0. はじめに

その1の書店編に始まり，以後，エレクトロニクスショップ編，ショッピングセンタ編，レストラン編と続き，一応，今回が最終回の宿泊／観光編となりました．

本編では，シリコンバレーの"よいホテル"といくつかの観光スポットを紹介します．ただし，今回に限って，筆者が経験したことがあるところに限るという原則は適用されないことをご承知おきください．そのかわり，筆者の住んでいるアパートを簡単に紹介することにします．

1. 宿泊

短期間の滞在で，出張中の宿泊費は実費精算などという場合には，Hilton, Marriott, Sheraton などの一流どころのホテルに泊まってみてはいかがでしょう．レストランやバーなどもあり，また最寄りの飛行場まで無料の送迎バスが 30 分毎くらいに出ているので大変便利です．プール，フィットネスルーム，サウナなどがついていることも多いのですが，その半面，1泊＄100近くかかるようです．この種のホテルは日本の一流ホテルの様子を想像すれば当たらずといえども遠からずといったところでしょう．

一方，やや長期に渡る滞在の場合には，モーテル形式のホテルがよいと思います．たとえば，Best Western というグループのホテルならば，ある程度のクオリティは保証されていますし，値段もそれほど高くはありません．簡単なセルフサービスの朝食がついていることが多いようです．

かなり長期に渡る滞在ならば（1か月以上），家具付きのアパートを借りるべきです．場合によっては 1か月分くらいの敷金を要求されることもありますが，居心地はホテルなどの比ではありませんし，お値段もぐっと安くなります．大きなアパートならプールやテニスコート，フィットネスルームなど一流ホテル並みの設備は整っていますからずっとお得でしょう．

1-1. Hilton Hotel

有名なホテルチェーンですから知らない人はいないでしょう．シリコンバレーにもいくつかありますが，San Jose ダウンタウンにあるのは Saint Clair Hilton です．San Jose State University の近くで，Convention Center や Art Museum にも近い，大変便利なところです．空港からだと，Guadalupe Pkwy. を南に辿り，Market St. へと出てから約1マイルほどで West San Carlos と交差しますが，その交差点のところにこのホテルがあります．空港までの無料送迎バスがあるはずですので利用するとよいでしょう．空港では，タクシー乗り場の近くで待っていると，ホテルの名前を書いたマイクロバスがやってきますので，手を上げて合図すれば乗せてくれます．

Hilton Hotel は，どこの Hilton であろうと，料金不要の 800-445-8667 で，予約できます．なお，この Saint Clair Hilton の電話番号は，(408) 295-2000 です．

シリコンバレーにはこのほかに次の Hilton Hotel があります．

Newark-Fremont Hilton
39900 Balentine Dr., Newark
(415) 490-8390

Sunnyvale Hilton
1250 Lakeside Dr. , Sunnyvale
(408) 738-4888

また，San Francisco 空港の近くと，San Francisco のダウンタウンにもありますので，参考のために掲げておきます．

San Francisco Airport Hilton
(415) 589-0770

San Francisco Hilton
(415) 771-1400

1-2. Marriott Hotel

これもまた有名なホテルチェーンです．シリコンバレー内では，後で述べる Great America という遊園地の近くに Santa Clara Marriott があります．もちろん空港までの無料送迎バスがありますが，車で行くのなら，フリーウェイ

101を北へ向かい，ほんの数マイル行ったところの Great America Pkwy. を北行きへと出ます．最初の信号（Mission College Blvd.）を右へ曲がるとまもなく右側にホテルへの入り口があります．

ほとんどビジネス一色のシリコンバレーにあって，ここはどちらかといえばリゾートホテルの感覚です．４面のテニスコートは無料で，別に宿泊客でなくても使えると思います．温水プールもあり，また，近くには Great America がありと，シリコンバレーの休日を過ごすにはよいところでしょう．ホテル内のレストランの味には定評があります．

Santa Clara Convention Center にも近いので，会議やショーが開催されるときはかなり混むようです．

全米共通の予約電話が，800-228-9290 Santa Clara Marriott の電話番号は，(408) 988-1500 です．

また，San Francisco 空港の近くには，Airport Marriott Hotel があり，こちらは，(415) 692-9100 です．ここの１階にあるカフェテリアの Marriott Burger は特筆に値しますので，機会があればぜひ一度お試しください．

1-3. Sheraton Hotel

Hilton, Marriott ときて Sheraton を忘れるわけには行かないでしょう．シリコンバレー内には次の２つがあります．

Sheraton Silicon Valley East
1801Barber Ln. , Milpitas
(408) 943-0600

Sheraton Sunnyvale
1100 North Mathilda Av. , Sunnyvale
(408) 745-6000

ただ，残念ながら筆者はどちらもよく知らないので詳しいことは分かりません．

1-4. Hyatt Hotel

これもけっこう有名なホテルですが，シリコンバレーでは，San Jose 空港から 1st St. に出てきたところ，フリーウェイ 101と 1st St. がちょうど交差するところにあります．ここの１階のレストランは日曜日のブランチが有名です．

予約は料金不要の 800-228-9000，このホテルの電話番号は (408) 993-1234 です．

1-5. Howard Johnson

シリコンバレーには，上述の Hyatt Hotel の向かい側と，Stevens Creek Blvd. をフリーウェイ 280 の下をくぐってから西へ少し行ったところとの２ケ所にあります．それぞれ，下記の通りです．Stevens Creek の方は割と新しく，またちょっと中世のお城風の雰囲気があります．

1755 North 1st St., San Jose
(408) 287-7535
5405 Stevens Creek Blvd., Santa Clara
(408) 257-8600

1-6. Economy Inn

これはモーテルに分類されるあまり大きくないホテルです．この種のホテルは当たり外れがありますので注意が必要ですが，一般的に，1泊いくらとかかれた大きな看板を出しているところは良心的といえるでしょう．この Economy Inn は，シングルで＄40弱です．もっとも，シングルとはいえベッドはクイーンサイズだし，部屋は広くて清潔だし，温水プールもあり，極く簡単ながら朝食がついていてこの値段ですから，なかなかだと思います．場所は，フリーウェイ 880を Calaveras Blvd.（州道237号）の東行きに出て，最初の信号（abbott Av.）を右に曲がったところの左側です．

所在地と電話番号は次の通り．

270 South Abbott Av., Milpitas
(408) 946-8889

なお，この種のモーテルを探すのなら，AAA（American Automobile Association）の Tour Book が最適でしょう．州別に観光地やホテルを網羅したもので，AAA の会員なら無料でもらえます．これはかなりすごい本で，人口 100 人くらいの小さな街のホテルでもちゃんと載っているから，自動車旅行などには必携の一冊といえるでしょう．車を持っている人は多分 AAA の会員になっているでしょうから，もしアメリカに知り合いがいたら入手してもらうよう頼んでみてはいかがでしょうか．

1-7. Oakwood Apartment

これが，実は筆者の住んでいるアパートです．San Jose には，North と South の２ケ所がありますが，それほど離れているわけではなく，どちらも，既に紹介した Yaohan というスーパーマーケットの近くです．

アパートとはいっても，３階建てのビルが30近くありますので，住宅団地というべきでしょうか．ただ，かなり広い敷地に，駐車場のスペースをたっぷり取ってあり，また，きちんと手入れされた芝生や木がありますから大変ゆったりとした感じがします．この種のアパートは大抵何処でも温

水プールやテニスコートを備えていますが，この Oakwood Apartment には他に，アスレチックジム，サウナ，ラケットボールのコートなどがあり，また，プールサイドにはバーベキューグリルもあっていつも香ばしい煙が立ち上っています．

筆者が借りているのは，1 Bedroom と呼ばれる構造の部屋で，ダイニングと共用になった居間と小さな台所，バス，トイレのほかに寝室が1つあります．これに，食器，電子レンジ，冷蔵庫，テーブル，ソファ，テレビ，電話，ベッドなど必要な物がついていて，電気代と水道代が含まれて，1ケ月＄1200 です．一番安いホテルでも1日＄40くらいなので，居住性や自由度などを考えれば，長期滞在にはやはりアパートの方が適しているといえるでしょう．しかも，長期契約すればかなりのディスカウントが適用されます．なお，普通のアパートは最低でも1ケ月の契約が必要です．シリコンバレーには，最近の地価の高騰を反映してかこの種のアパートがかなりたくさんできています．けっこう安いところもあるようですから，イエローページなどで探してみるとよいでしょう．

2. 観光スポット

シリコンバレー内のいくつかの観光スポットと，車を使って1泊くらいで行ける範囲の有名な観光地とを紹介します．

シリコンバレーはもともとビジネスエリアなのでそれほど有名な観光地はありませんが，それでもけっこう楽しめるところは多いようです．もちろん本稿で取り上げたところ以外にもたくさんの観光スポットがあることでしょうが，それらの情報は，たとえばイエローページの Places to Go というページで得ることができますので，参考にしてください．

また，車で少し遠出をすれば，大自然の美しさでも，あるいは人工の享楽でも思う存分に味わうことができます．週末を利用してちょっとしたドライブ旅行なんていうのはかなり楽しいものだと思います．

2-1. Winchester Mystery House

525 South Winchester Blvd., San Jose

(408) 247-2201

ライフル銃で有名な Winchester という会社のオーナーだった Sarah Winchester さんに悪霊が取り付いたのでそれを祓うために建てた家なのだそうです．だからといってお化け屋敷みたいな物ではないのですが，建て増しに次ぐ建て増しでやたらと広くなり，160の部屋，2000のドア，13のバスルーム（これは多分トイレと兼用になった物でしょう），10000の窓，40の階段などのほか，物置や秘密の通路もあるそうで，その複雑な構造はこの家の主人や使用人ですら地図がないと迷ってしまうほどだといわれています．

場所は，フリーウェイ 880を Stevens Creek Blvd. の西行きへと出，Valley Fair Shopping Center を通り過ぎたところで交差する Winchester Blvd. を左折したまもなくのところ，右側にあります．

約1時間ほどのガイド付きツアーもあり，また Winchester 社に関する博物館などもあって，1日中でも楽しめるところです．

2-2. Great America

(408) 998-1776

フリーウェイ 101を北に向かうと，San Jose 空港から5マイルほどのところで Great America Pkwy. がありますから，これを北行きへと出ます．そのまま直進すればまもなく右側に駐車場入口が見えてきます．

筆者はジェットコースター恐怖症なので，この Great America にはまだ入ったことはありませんが，何でも，Demon, Wizard, Tidal Wave, Grizzly wooden と4つものローラーコースターがあるそうで，他にも Free Fall だとか，Rip Roaring Rapid とかいう筏乗りなどがあって，なんとも危険なところのようです．もちろん，お子様向けのアトラクションもたくさんあり，また，劇場や野外コンサートホール，イルカのショーなどもあって，家族連れでもたっぷり遊べるのだそうです．

夏の間は毎日，春と秋は土日だけで，冬の間は閉鎖されます．

2-3. Winery

カリフォルニアワインといえば，Napa や Sonoma が余りにも有名ですが，実はシリコンバレーにも Winery があります．Cupertino の南から Saratoga にかけての一帯に10件ほどが点在しているのですが，まだ歴史が浅いためか，あまり知られてはいないようです．

どの Winery でも見学や試飲ができるようで，またピクニックにも最適です．普通は朝 11時頃から夕方 5時頃まで開いています．

2-4. Flea Market

日本語でいえば蚤の市ですが，英語でも蚤という言葉を使うのは，単なる偶然なのか，それとも深い歴史的背景があ

るのか，あるいはまたどちらかの翻訳なのか，筆者には知る由もありません．さらには，時々聞くことのある自由市場というのが，この Flea Market とは別の Free Market のことなのか，あるいは発音が似ているために間違って認識されたものなのかということも分かりません．ただひとつ分かっていることは，この Flea Market は大変に楽しいということです．

San Jose 空港から 1st St. に出て南へ行くとやがてフリーウェイ 880の下をくぐり，Hedding St. と交差します．この Hedding St. を東行きへと左折してしばらく行くと，やがてフリーウェイ 101をまたぎ，Berryessa Rd. と名前が変わり，そして左側にこの Flea Market が見えてきます．ただしやっているのは週末だけのようです．

ここは 28年もの歴史があるとかで，出ている店の数は2000を越え，ありとあらゆる物をかなり安い値段で売っています．ほとんどの店が地面にテーブルを置き，その上に品物をおいて商売をしています．中古品などは極めて安い値段がついていますし，また電気製品や日用品の店も多いので，長期滞在のための生活用品を仕入れるには最適でしょう．またカラフルな Tシャツなどはお土産にもよいと思います．

Nintendo のゲームカセットの交換が面白いという話を聞いたことがありますが，残念ながら筆者はまだ実際に見たことはありません．なお，アメリカで Nintendo といえば，任天堂という会社のことではなく，アメリカ版のファミコンを指します．従って，"Nintendo のゲームカセット"というのは，任天堂社製のゲームカセットという意味ではなく，アメリカ版ファミコン用ゲームカセットという意味になりますので念のため．

2-5. Lake Tahoe

San Francisco からだとツアーのバスが出ているのですが，残念ながらシリコンバレーからのはありませんので，車を使って行くことになります．まず，フリーウェイ 680を北上し，フリーウェイ 80の東行き（Sacramento 方面）に入ります．Sacramento に入ってからフリーウェイ 50に乗り換えひたすら走り続けるとやがて South Lake Tahoe に辿り着きます．ここまで，恐らく５時間くらいでしょう．

Lake Tahoe の水の青さはほとんど神秘的です．特に Emerald Bay の美しさは筆舌に尽くしがたいものがあります．これを満喫するには South Lake Tahoe から出ている観光船を利用するのがよいでしょう．２時間半で＄12くらいです．

また，湖を巡る道路に沿って数多くのビーチやスキー場があり，夏はウィンドサーフィン，水上スキー，パラセール，冬はスキー，スケートと一年を通して楽しむことができます．

ギャンブルの好きな方は，カジノへどうぞ．South Lake Tahoe にある大きなホテルにはいずれもカジノがあって，ブラックジャック，ルーレット，ポーカー，バカラ，クラップス，スロットマシンなどにたくさんの人が熱くなっています．ゲームのコツを解説するのは本稿の主旨から外れてしまいますので別の機会に譲りますが，クレジットカード用の端末装置を使って現金を引き出せるということは知らないほうが身のためでしょう．

なお，どこのホテルでも＄10前後で飲み物付きのショーが見られます．フロントに聞けばその日の出し物と時間を教えてもらえると思います．トップレスでの踊りの合間にマジックやアクロバットを折り込んだものが比較的無難で，コメディショーは英語に自身のある方に向いているようです．

2-6. Yosemite

今更 Yosemite の紹介もないもんだと思いますが，それでもあえて取り上げたのは，即ち Yosemite がそれだけ魅力的なところだということです．その魅力は，もちろん Yosemite Village を歩くだけでも十分に味わえるでしょうけれども，できれば2泊3日くらいの余裕をもって，トレッキングコースを歩いてみることをお勧めします．たとえば，Glacier Point へと登る道は，距離は4マイルほどで大変短いのですが，かなりしんどいコースです．しかしながらこの Glacier Point からの眺めはそれだけの苦労と引替にする価値は十分あると思います．もちろんツアーのバスでここまで連れてきてもらってもよいのですが，歩いて登ったほうが，景色ははるかに美しく見えることでしょう．

もうひとつ，見逃せないところが Tuolumne Meadows です．遥かな山々に抱かれた静かな草原ですが，かなたに立派な角を持った鹿が見え隠れするような素晴らしく雄大な Wilderness の世界です．Yosemite Village からは少々遠いのでこちらは車を使うことを許しましょう．途中の Tenaya Lake もたいへん美しい湖です．

年間300万人もの観光客が訪れる一方で，ここの自然を愛し毎週のように出かけていく人がいるというのも，この Yosemite の魅力の一端にでも触れたことがあれば文句なく納得できることでしょう．

シリコンバレーからなら，いったんフリーウェイ 680で

北へ向かい，途中フリーウェイ580東行きを経てMantecaの街で州道120号に乗り換えれば，Yosemite Villageまで5時間ほどで行けます．San Franciscoからのツアーバスを利用するのもよいかもしれません．

ただし，ポテトチップスとビールは持っていかないほうがよいと思います．2000～3000mくらいの高度なのでポテトチップスの袋は爆発するし，ビールはちょっと油断すると半分以上が泡となって吹き出してしまいます．もっとも残った半分で十分酔っ払えますが．

また，決められたところ以外でのキャンプが禁止されている事は常識になっていますが，川や湖から100フィート以内のところでは，石鹸，シャンプー，歯磨きの使用が禁止されている事はつい忘れてしまうようです．他にも色々な決まりがあるようですが，自然を守るという考え方からすれば不合理な決まりは一切なく，自然と一体になれる人にとっては不自由でも何でもないでしょう．

とにかく，大自然の優しさ，野生のたくましさ，大気の清々しさ，せせらぎの清冽さを心行くまで楽しめるところですので，ぜひ一度訪れてみてください．

3. 番外編

別にたいした事ではないけれど，知らないよりは知っていた方がよいだろうというような事を，番外編としていくつか御紹介しましょう．

誰でも知っていて，実は筆者だけが知らなかった事もあれば，あるいは日本に紹介されるのは初めてという事もあるかも知れません．Californiaだけでしか通用しない事もあれば，世界的な常識になっている事もあるでしょうが，ここでは思い付くままに述べて行くつもりですので，どれに相当する内容であるかは，読みながら適当に判断して下さい．

シリコンバレー

筆者自身，シリコンバレーがどこにあるのか，実際にシリコンバレーに来るまでは全く知りませんでした．

シリコンバレーは，実はCalifornia州にあります．とはいえ，California州自体が日本と同じくらい広いのですが，その中でも，ウエストコースト観光の北の拠点，霧と坂道と夜景の美しいSan Franciscoから，南へ約50マイルほど行ったところに広がる，Palo Alto, Mountain View, Sunnyvale, Cupertino, Santa Clara, San Joseなどの各市をあわせた一帯がシリコンバレーと呼ばれています．

1955年にショックレイが半導体研究所を作ってシリコンバレーの誕生の地となったPalo altoを始め，Apple本社のあるCupertino, Intel本社のあるSanta Claraなど，いずれもどこかで一度は聞いたことのある地名であろうと思います．

バレーと呼ぶには広大過ぎてむしろ平野とでもいうべき土地に，数多くのコンピュータやエレクトロニクスに関係する企業が集中しているわけですが，だからといって日本橋や秋葉原の電気店街をイメージするのは誤りです．二階建て以上のビルは少なく，芝生と駐車場に囲まれた平屋の社屋の企業が点在しているといったほうが，正確でしょう．

より詳しい事は，シリコンバレーについて書かれた本を参照して下さい．少々古い本ですが“シリコンバレーフィーバー”（講談社，E.M.Rogers, J.K.Larsen共著，安田寿明，A.S.Docker共訳）は，シリコンバレーをさまざまな角度からとらえている点でお勧めです．

3-2. San Jose

筆者が住んでいるのがこのサンノゼ市です．人によってはサンホセという人もあり，またテレビのアナウンサなどはその中間くらいの微妙な発音をしているようにも聞こえます．スペイン系の人名に由来する地名だと記憶していますが，そのためこのようなスペイン語風の読み方をするのでしょう．事実，この辺りはメキシコからの移民が多く，スペイン語が極めて頻繁に聞かれます．コモエスタス，ビエン，グラシアス（順に，Hou are you?，Fine.，Thank you. の意味）などは知っておいても損はありません．ほかにも，中国，韓国，ベトナムなどアジア系の人も多いようです．

San Franciscoから車で約1時間，人口70数万人の市で，気候は温暖，治安もよく，たいへん住み心地のよい所です．

3-3. クレジットカード

ほとんど全ての店でクレジットカードが使えますが，MC（マスターカード）かVISAでなければ役に立たないと思っていたほうがよいでしょう．American Expressが使えるのはショッピングモールの中の専門店と大手のホテルくらいのもので，これがJCBとなるともうシリコンバレーでは無力です．ただJCBは，PLUS Systemという金融ネットワークと提携しているので，銀行のキャッシュディスペンサで現金を引き出すことができる点は大変便利です．たとえばBOFA（Bank of America）などは支店の数が多く，ほかにもPLUS Systemと提携している銀行はいくらでもありますから，事実上どこででも現金が引き出せるといっ

ても過言ではないでしょう．ただし，普段あまり使うことのない暗証番号（Personal Identification Number，一般にはPINと呼ばれている）が必要になります．

3-4. 小切手

長期滞在するときは銀行口座を開き，小切手を作ることをお勧めします．口座は，普通預金（Saving Acount）と当座預金（Checking Acount）の２つを作り，まとまった額をSavingの方に預金しておいて必要な分だけCheckingの方に移して使うというのが普通です．両方の口座がリンクされたカードを使って，現金の引き出しや口座間の転送などを年中無休２４時間営業のキャッシュディスペンサで極めて簡単に行なうことができるのは本当に便利です．

小切手は，相手がNoといわなければ全ての金銭の受渡しの場面で用いることができ，しかも支払う相手と金額を明記しているので極めて安全であるという特長があります．電話代や電気代，定期購読の雑誌代，通信販売の代金などは小切手の郵送によるのが普通です．なお，商店で小切手を使うときは２種類のID（身分証明）を要求されることが多く，通常，運転免許証とメジャーなクレジットカードが使われます．

3-5. 現金

紙幣は1, 5, 10, 20ドルが，コインは1, 5, 10, 25セントが用いられます．50ドル以上の紙幣と50セント以上のコインは絶対に使われないといってもよいほどです．コインにはPenny（1セント），Nickel（5セント），Dime（10セント），Quarter（25セント）という名前がついていることはよく知られています．

ところで，４分の１という考え方に馴染みがないせいか，お釣りの勘定で時々戸惑うことがあります．

たとえば，４ドル７６セントの買い物をした時，もちろんお釣りのないように渡すのが一番親切なのでしょうが，そうでなければ，５ドル１セントを渡すのが次善の方法であることにはなかなか気づかないものです．こういう場合，つい，５ドルと６セントを出してしまうことが多いのですが，これではお釣りが３０セントになりQuarterとNickelが返ってきますから，結局無意味な５セントのやり取りをすることになるのです．

なお，このような無意味な５セントの授受をやってしまったことに気づいても，知らん顔をしておくことが大切です．ふと考え込んだり，苦笑したり，あるいは余計な手間をかけたとレジ係に謝ったりするのは，かえってレジ係を混乱させることになります．なぜなら彼らは，多くの場合，渡された金額をそのままキーインし，表示された釣銭をそのまま返してくれるだけで，無意味な５セントの授受には気づいていないからです．それに気づくような人なら，多分初めから６セントを受け取ったりせず，その内の５セントを返してくれながら「これで十分です．」とか何とかいってくれるはずです．なお筆者は，考え込んだことが１回，苦笑してしまったことが２回，思わず謝ってしまったことが１回，十分ですといわれたことが１回あります．もちろん筆者自身も気づかなかったことがあるかも知れませんが，その回数は当然不明です．

3-6. 運転免許

免許証はもっとも一般的なIDとしても用いられています．割と簡単に，しかもわずか＄10で取れますから，長期の滞在の場合には話のタネにチャレンジしてみてはいかがでしょうか．

まず，ドライバーズハンドブックという小冊子（A5くらいのサイズで約30ページ）をDMV（Department of Motor Vehicle）で入手します．最寄りのDMVの場所はイエローページなどで調べればよいでしょう．これをひと通り読んでからDMVへペーパーテストを受けに行きます．申込書に必要事項を記入して窓口に＄10を添えて申し込むと，解答用紙と兼用になった問題用紙をくれます．三者択一の問題が50問ほどあって，約85％以上の正解で合格です．正しいと思う答えの□印の中にXを付けるのですが，時々○を付ける日本人がいるそうです．制限時間はなく，壁際のカウンタで立ったまま，申込書を書くような感じでやるわけで，試験という雰囲気は全くありません．ですから隣の人のをのぞくことは，禁止はされているものの，できないことではないでしょう．ただし，問題用紙は何種類もあって必ずしも隣の人が自分と同じ問題をやっているとは限りません．もちろん英語ですが，ほかにはスペイン語，韓国語，ベトナム語などの問題用紙が用意されているそうです．なぜか日本語はありませんでした．

ベストを尽くしたと納得したら，次は窓口で採点してもらうのですが，これは係員がその場でほとんど瞬間的にやってくれます．解答を完全に暗記していてもこうはいかないと思うほどのスピードです．「不審があったら指摘してください．」という一言を付け加えることによって少々の採点ミスを許容してしまっているわけで，これには，日本の価値観をもっていい加減といい切ってしまうことのできない，合理的な知恵のようなものを感じてしまいます．

ペーパーテストに合格すると Instruction Permit がもらえます．これは，免許を持っている人が同乗していれば，公道で運転してもよいというものです．これで暫く練習して自信が付いたら，実技テストの日時を DMV に予約し，当日は自分で車を用意してテストを受けに行きます．

まずブレーキランプや方向指示器がチェックされ，さらに手信号を知っているかどうかが確認された後，試験官が助手席に乗ってその辺の道路を約 15 分ほど走り回ります．採点はその間に行なわれるわけですが，軽いミスで３点，重いミスで 5～10点減点され，30点以上の減点か１回以上の致命的なエラーで不合格となります．なお，２種目の規定演技がありますが，これは多分誰でもできるでしょう．ひとつはスリーポイントターン（１度切り返してのＵターン）で，もうひとつはまっすぐ５ｍほどバックするというものです．ほかの州ではパラレルパーキング（縦列駐車）や坂道発進などの項目もあるそうですが，いずれにしても高度な技術が要求されているわけではなく，やはりテストというよりは演技といった感じでしょう．

こうして実技テストに合格すると，視力検査があり，写真を撮られ，指紋を取られ，仮の免許証を発行してもらって，それでおしまいです．正式な免許証は２週間ほどで郵送されてきます．従って，郵便の届く住所さえあれば，住民であろうと密入国者であろうと別に関係はないようです．

3-7. 交通違反

パトカーに捕まった時，国際免許だと許してもらえるという話を聞いたことがあります．そんなはずは，と思っていましたが，日本語と英語がやたら細かい文字で混在している国際免許なんて見るのも面倒だからだという合理的な説明を聞くと，ありそうなことだと納得してしまいます．本当かどうかはわかりませんが．．．

なお，パトカーに捕まった時には，おとなしく車を止めてハンドルに手を添えたまま警官が近づいてくるのを待つのがよいそうです．窓を明け，上半身を乗り出して，内ポケットに入っている免許証を取り出そうとするなどというのは，冗談では済まないようです．

軽い違反なら罰金で済みますが，金額，支払方法，レシートがもらえるかどうかなどはまだわかりません．人に尋ねても知らないといわれるばかりです．

3-8. 道路

本シリーズでは，紹介する店にはなるべく所在地を付けるように努力しました．ストリートアドレスというものが，目的地を捜し当てるのに大変便利だからです．

このストリートアドレス，見れば分かると思いますが，番地，道路の名前，必要ならばアパートなどの部屋番号，都市の名前，州の名前（略号で書かれることが多い），Zip Code の順になっていますが，道路の名前にはいろいろな略号を使うことが多いようです．

たとえば，本稿の中に出てきた略号には，Expwy., Pkwy., Blvd., St., Av., Rd. などがあったと思いますが，他にも，Ct., Dr., Cir., Wy., Ln. というようなものもあります．これらは，それぞれ，Expressway, Parkway, Boulevard, Street, Avenue, Road, Court, Drive, Circle, Way, Lane を表わしています．どれがどんな道路かという明確な区別はないようですが，たとえば，Expwy. は信号の少ない広い道路に使われますし，大通りという意味では Blvd. が使われることが多いようです．また，St. や Av. はまあまあ大きな道路ですが，Rd., Dr., Wy., Ln. となるとかなり細い道になるようです．Cir. は大きな建物の周りを取り巻くように環状になった道路を，また Ct. は奥が行き止りになったような道路をいいます．Pkwy. はあまりよく分かりません．Highway というのは，Expwy. と同じような道路をいう場合が多く，日本でいう高速道路は Freeway と呼ばれていますが，でも，Freeway をパトロールしているのは Highway Patrol であって，Freeway Patrol ではありません．

3-9. スーパーマーケット

別にたいした事ではないけれど，知らないとなんの事だかわからないという決まり文句がありますが，スーパーで，"Paper or plastic?"ときかれたときも，初めは何の事だかわかりませんでした．これは紙袋とビニール袋のどちらにするかという質問なのですが，日本でもお馴染みの，スーパーの白い袋の材質をプラスチックと呼ぶのにはまだ多少の違和感があります．もっとも筆者にしたところで，あれをビニール袋といったり，ポリ袋といったりしていますし，正しくは何という材質なのか知らないわけですから，どっちもどっちというところでしょう．

3-10. ガソリンスタンド

セルフサービスとフルサービスに分れていたり，クレジットカードしか使えなかったり，先払いのところがあったりとさまざまですので，知らないと戸惑うことが多いようです．

セルフサービスというのは自分で給油するわけですが，

これは思ったほど難しい事ではありません．フルサービスより10%くらい安いようです．なおフルサービスといってもガソリンを入れてくれるだけのところがほとんどで，窓を拭いてくれるところはたまにしかありません．

先払いのガソリンスタンドでは，まず窓口にいってお金を払います．後はセルフサービスで給油するのですが，支払った分だけ出るとガソリンは自動的に止ります．満タンにしたければ，20ドル札でも渡しておいて後でお釣りをもらえばよいでしょう．

ガソリンは，1ガロンで $1くらいです．1ガロンは3.78リットルなので，1リットル当たり40円くらいでしょうか．フルサービスやクレジットカードでは少し高くなり，また，Unleaded（無鉛）に比べて Super Unleaded とか Extra Unleaded と呼ばれる高性能無鉛ガソリンは10%程高くなります．Regular（有鉛）は普通の車には使われませんが，給油ノズルの太さが違うので誤って入れてしまうことはないようです．

3-11. テレビ

電波による放送は，10数チャンネルを良好に受信できます．3大ネットワーク（ABC, NBC, CBS）はもちろんのこと，ローカル局でもけっこう面白い番組をやっています．変わったところでは，26チャンネルで各国の番組をやっていて，アラビア語，韓国語，中国語，日本語などの放送が楽しめます．主な日本語放送は，ウィークデーの23時頃からの NHK ニュースと，ウィークエンドの夕方からのドラマや歌謡番組ですが，正直なところあまり面白い物はやっていません．

受信状態がよくない場合や，もっと多くの番組を楽しみたいときにはケーブル TV がよいでしょう．ケーブル TV の会社に連絡してケーブルを引いてもらうと，30～40チャンネルを受信できるようになります．工事費が＄50くらい，毎月の料金が＄10くらいです．ESPN というスポーツばかりやっているチャンネルのほかにも，ニュースばかりのチャンネルや映画ばかりのチャンネルがあり，さらに Premium Service という有料のチャンネルでは CM 無しで映画やスポーツ中継を楽しむことができます．これはチャンネル当たり月に＄15くらいで，HBO, showtime, Movie Channel といった家族向けのチャンネルや，子供向けの Disney Channel，大人向けの Playboy Channel などがあります．

テレビ番組で，筆者が欠かさず見ているのは，TV 版 Star Trek の新シリーズ The Next Generation です．登場人物は一新されたものの，それぞれに十分個性的で，Enterprise も曲線を使った流麗なフォルムになり，オリジナルのシリーズに勝るとも劣らない物に仕上がっています．

3-12. TAX

California では，食料品の一部をのぞいて，7%の税金がかかります．日本でも最近，消費税という物が導入されたそうなので特に説明の必要はないでしょう．なお課税される物については，値段のシールの隅に小さく Tax とか Tx などと書かれていることが多いようです．

日本の雑誌を買うと，"定価600円(本体583円)"などという訳のわからない表示がありますが，日本ではこう書かなければならないのでしょうか．税金を含めた額を定価と呼ぶのもさることながら，元々580円のこの雑誌が，実質的に約3円（正確には600/1.03-580=2.54... 円）値上げされていることに強い疑問を感じています．

3-13. 電話

時々アルファベットの電話番号を見かけることがあります．たとえば，The Mac Shop という Macintosh の店の電話が，(408) 736-MACS という具合です．日本でも，日産サニーの番号が2332だったりという語呂合わせがありますが，これはそのアメリカ版とでもいえるでしょうか．

実は，アメリカの電話には，Q と Z をのぞいた24個のアルファベットが，2～9の数字にそれぞれ3つずつ割り当てられているのです．たとえば，2には A, B, C の3つが，6には M, N, O が，7には P, R, S が割り当てられているわけです．従って，前述の MACS という電話番号は，M が6に，A と C が2に，S が7に対応するので6227という番号になります．

電話には他にも面白い機能や習慣がありますのでいくつか御紹介しましょう．

まず，変わった操作として，"Flash" と呼ばれる，フックを短く一度押す操作があります．これは，相手をホールド状態にして自分の回線から切り離すもので，内線電話の転送などに用いられます．たとえば，電話機の説明書きに Transfer : "Flash" + Number とあれば，これは，別の電話へ転送するには一度フックを短く押して，続いて転送先の番号をダイヤルするということを表しています．

また，日本でもキャチホンと呼ばれる，通話中にかかってきた電話を受けることのできるサービスがありますが，同様のサービスはアメリカにもあって，これにも "Flash" が用いられます．このサービスに加入していると，通話中に

他の人から電話がかかってきたときに，着信を知らせる音が聞こえます．その後はこの "Flash" 操作で今まで話していた人と新たにかけてきた人とを切り替えることができるようになります．なおこの場合，一方と話している間は他方は何も聞こえない状態になります．

さらにこの他に，カンファレンスコールとよばれるサービスもあります．これは通話中に "Flash" で相手をホールド状態にしておき，その間に別の相手に電話して，相手が出たらもう一度 "Flash" すると３人で同時に会話することができるようになるというものです．

ところで，そのまま切らずにお待ちくださいというのを英語では "Hold on , please." といいますが(電話を切ることは Hang up といいます)，会社などに電話をかけた場合に，交換手が出た途端にいきなり Hold on といわれて何も聞こえなくなることがあります．これも習慣の違いといえるでしょうか．日本では，交換手はかかってきた電話の取次ぎが終わるまではつぎの電話を取ることはないようですが，アメリカでは，先に述べたキャッチホンの機能を使ってかかってきた電話を次々に取り，次々にホールドしておいてから，順次用件を聞いていくというやり方をするようです．どちらがよいかは個人の好みですが，こういう事情を知っていればいきなりホールドされても戸惑うことはなくなるでしょう．ただ，おおらかな国柄なので，たまにホールドされたまま忘れられてしまうこともあるようです．あまり長く待たされるようならかけ直したほうがよいかも知れません．

3-14. 留守番電話

Answering Machine とか，あるいはそれとはっきりわかるようなときには単に Machine などと呼ばれて，大変ポピュラーに使われています．人によっては居留守番電話として使っているようですが，それはともかく，いつも驚かされるのはアメリカ人のメッセージの要領のよさです．普段から使い慣れているからでしょうか，名前，日時，用件，自分の電話番号などをじつに要領よくしゃべります．日本人だとこうはいかないようで，特に日時を忘れることが多いようです．今何時か，あるいは今日は何日かという情報がないと２時間後とか明日などといわれても困ることがあるのは，経験してみないとなかなかわからないのかも知れません．また，たとえ相手が既に知っているとしても，自分の電話番号も残しておくほうがずっと親切です．メッセージを聞いた相手が一々住所録を探す必要がなくなりますから．また，はっきりしゃべることは必要ですが，特にゆっくりしゃべったり，同じ事を何度も繰り返したりする必要のないことも，使ったことがあれば容易に理解できるでしょう．

ところで，前述のカンファレンスコールを使うと，面白いいたずらができることにお気づきでしょうか．もしかけた相手が不在で，留守番電話のメッセージが聞こえてきたら，すかさず別の相手にカンファレンスコールするのです．別の相手がでたら送話口を手で塞ぐとか Mute ボタンを押すなどの方法で黙っていると，別の相手の「いい加減にしろ！」などというメッセージを最初の相手の留守番電話に残すことができるわけです．もちろん，でたらめな番号にカンファレンスコールして，「おかけになった電話番号は……」というメッセージを残すこともできます．

4. 結び

思い付くままに述べてきましたが，こうしてみると習慣の違いというのは，たとえ知っていても，実際に経験するまでは理解できないという性質があるようです．

"ディズニーランドへ行ったことがありますか？"

「いいえ，ありません．」

"まだ行ってなかったのですか？"

「はい．」

"？？？"というような Yes と No の使い分けも，確かに学校では習ったのですが，慣れるまでは大変です．

思い出すたびに赤面してしまうような失敗もいくつとなくありますが，それは別の機会に譲ることにして，ひとまずは，「シリコンバレーを楽しむために」この辺でお開きにしましょう．御愛読有難うございました．

SEA 会員の中には，国内・海外を問わず，お仕事でいろいろな所へお出かけになる方が多いことでしょう．ぜひその際に得られた情報を御紹介ください．何も，この大神原さんの原稿のように大長編でなくて結構です．「地球を歩く － プログラマ篇」のようなものができればいいなと思っています（編集部）．

ボトルマン始末記

塩谷　和範
SRA

少し旧聞ですが，昨年６月の Usenix（於・バルチモア）に参加した折り，ニューヨークの下町での体験を御紹介します．大神原さんの明るいカリフォルニア案内と好対象のできごとです．

私と jus 幹事の西村氏とは，真昼間１時過ぎ７番街（ここは，ニューヨークではあぶないゾーンに属しています）を，AMTRAK（アメリカの JR？）のペン・ステーション駅からタイムズスクエアに向かって，ぶらぶら歩いていました．

そのとき，突然すれちがった１人の黒人が何か叫んだのです．かれの言い分は，西村君の足と自分が下げていた紙袋がぶつかって，中のワインが割れてしまった，何とかしてくれ，というのです．西村君は「最初から割れていた」といいます．考えて見れば，すれ違いに足にぶつかった位で，ワインのビンが割れるはずがありません．

そこで，西村君に「これはたかりだ．相手は金が欲しいだけだからくれてやるかい？」と聞くと，「われわれはプログラマだ．徹底的に論理で戦いましょう」とやる気満々ですので，２人で応戦することにしました．

すると，相手は困ってしまって，延々と押し問答を繰り返すだけです．ニューヨーカーたちは無視して通りすぎていきます．そこへ，別の黒人が通りかかって相手に助け船を出してきました．「兄弟，何か厄介ごとかい？」って感じです．（この記事を書きながら思ったのですが，えたいの知れぬ黒眼鏡のアジア人が，こちらもあまり上等でない身なりをした同胞にいいがかりをつけているように見えたんでしょうね，きっと．２対１だし）

で，相手は，助け船！って感じで得々と説明しはじめました．すると，後からきた兄弟は，「兄さん方ョ，こいつは客に頼まれた高いワインを，そこの酒屋で買って届けるところだったそうだが，あいにくそこの兄さんとぶつかっちまって品物が台無しになり，困ってんだ．何とかしてやってくんねぇか？」というのです．

そこで我々は論理的に，

1. 足にぶつかった位でワインが割れるはずがないこと，
2. したがって，最初から割れていたはずであること，
3. もしぶつかって割れたのなら，ワインがこぼれているはずであること

を順序よく説きました．

すると，通りがかりの兄弟は，こいつはお手上げだという顔をして，去って行きました．残ったお兄さんは，もう論争するだけの知恵も居直る度胸もないらしく，「何とか弁償しておくれヨ」と小声でいうばかりです．こちらは，基本方針通り相手を論破（？）しおえたようなので，「悪い相手に当たったものと諦めてね」といってその場を立ち去ったのでした．運の悪い敵は，われわれの背中に「オ〜イ，旦那方何とかしとくれよ〜」と声を掛け続けたのですが，無視してほっときました．

週刊誌で読んだ記事によると，やはりニューヨークで１人の日本人が同じような目に会い，そのとき相手は黒人２人組みで弁償しろと喚いたそうです．しかし，「手口はわかっている．警察まで行って黒白をつけよう」といったら，憎まれ口を叩きながら去って行ったそうです．

やはり日本人って，どんな格好をしてても判るものなのでしょうかね？　そんなにお金持ちの顔をしているんでしょうか？　みなさんも気をつけてくださいね．私のケースは，昼間だったし，相棒もいたので，それほど危険な感じもなく，思わぬ英会話のやりとり楽しめましたが，場合によっては危険なこともありますから．

ところで，初めて行ったマンハッタンの印象ですが，目抜き通りに紙屑・空缶・空瓶が散乱しているし，新聞紙・チラシの類は風に舞っているし，ホームレス・ピープルは多いし，汚い街でした．サンフランシスコも似たような印象で，アメリカの景気は回復していると聞いていたのですが，底辺まで浸透するところまで行っていないようでした．何となく病める老大国という感じです．

日本はアメリカの１０年後を追いかけているといわれていますが，その意味でも何とか助けてあげる必要があるのかなと，考えさせられてしまいました．もっとも，日本が一時的に金持ちになったからといっても，社会資本・生活の豊かさ・文化／技術に対する姿勢などまだまだ向こうのほうが豊かですけどネ．

註１：われわれはジーンズ・Tシャツ・デイパックに黒眼鏡という格好でした．お上りさん丸だしのルックスですね．

註２：私はアメリカに偏見を持っていますが，アメリカ人は好きです．黒人に対して偏見はありません．

SEA Forum January '90 参加者アンケート

「CASE ツールの現状と動向」と題して，ベンダー６社のパネル討論を企画した１月３１日のフォーラムは，話題がタイムリーだったせいか，SEA フォーラム始まって以来のすごい人気で，最後は満員につき申込みをお断りするという盛況でした．

当日の参加者の方々に御回答いただいたアンケートの集計結果を報告します．これからの環境整備，CASE の導入，あるいはツール開発の企画などに参考にしていただければ幸いです．

Q1. Forum に参加した動機は？

現在使用中	8
導入を検討中	33
知識習得／情報収集	36
その他	3

Q2-1 現在の職種は？

開発に直接関係している	53
管理者	20
SE	23
プログラマ	11
その他	4
開発に直接関係していない	24
上級管理者	3
技術スタッフ	15
教育スタッフ	2
その他	5

Q2-2 開発対象ソフトウェアは？

事務計算	16
科学技術計算	7
プロセス制御	13
システムソフト	25
その他	17

Q2-3 開発用マシンは？

汎用機	22
ミニコン	24
WS	44
PC	25
その他	4

Q2-4 仕様書や設計書などの機械化

ほぼコンピュータ化	6
大部分はワープロで	37
まだほとんど手書き	28

Q2-5 ネットワーク（LAN）への接続

YES	47
NO	28

Q3. ソフトウェア工学技法の普及度

(3-1) 次の各技法についての知識や経験は？

構造化分析／設計法：	
よく知っている	15
一応知っている	45
名前程度は	16
知らない	0
ジャクソン法：	
よく知っている	7
一応知っている	27
名前程度は	34
知らない	8
オブジェクト指向設計法：	
よく知っている	3
一応知っている	25
名前程度は	44
知らない	3

(3-2) 各技法の周囲での普及度

構造化分析／設計法:

よく使われている	5
少しは使われている	30
ほとんど使われていない	41

ジャクソン法:

よく使われている	2
少しは使われている	14
ほとんど使われていない	60

オブジェクト指向設計法:

よく使われている	1
少しは使われている	15
ほとんど使われていない	60

Q4. 最も問題を含む開発工程は？

要求分析／定義	30
概要／基本設計	31
詳細設計	9
プログラミング	6
テスティング	23
保守	28

Q5. そうした問題の解決に CASE は役立つか？

必ず役に立つ	20
もしかしたら	45
役立ちそうもない	3
わからない	7

Q6. CASE を導入したいか？

既に使っている	10
是非とも	32
したいがむずかしい	27
できそうもない	4

Q7. 世の中一般での CASE の将来

急速に普及する	5
徐々に普及する	55
一時的には流行する	12
普及しそうもない	2

Q8. SEA では２月の Forum でも，ひきつづき CASE に関する討論を（今度は CASE ユーザの視点から）行いたいと考えています．どんな話題を取り上げたらよいと思いますか？

(1) 要求仕様から出発してそれを具体化して行く過程で，いろいろな技法およびツールを有効に活用するにはどうしたらよいか？（具体化の過程では，随所に人間の知的活動が必要であり，単に技法／ツールを機械的に適用するだけではすまないだろう）

(2) 実際の例，それも大規模なプロジェクトにおける適用例の発表を聴きたい．

(3) 実際に使ってみて，定量的／定性的にどのような効果があったか？ 利用者側にどんな問題が出てきたか？ その対応はどうしたか？ IBM の AD/Cycle についての意見を聞きたい．

(4) CASE については，今日の話を聞くのが２度目なので，具体的な話題を挙げることはできません．私個人としては，初歩的／入門的な話題をお願いしたいと思います．

(5) SA/SD はほんとうに使えるものなのだろうか？ 日本独自の CASE はないのだろうか？ メインフレーム上の CASE ツールの評価は？

(6) 実際に CASE ツールを導入し，現実のプロジェクトに使用しているユーザの体験談を聞きたい．

(7) オブジェクト指向型ツールの現状および可能性について（OORA, OOSD, OOD, etc）．近々 StP にオブジェクト指向の支援機能がつくとか？ 他に，考え方として取り入れられているものは？

(8) 構造化手法そのものの導入方法（教育，開発形態の変化，など）．

(9) 導入事例をもとに，ユーザの生の声を聞きたい．現行ドキュメントからの切替え方，導入評価（信頼性改善，工数など），現場の体制造り，etc.

(10) どの CASE ツールがデファクト・スタンダードとなるのだろうか？ そのための条件は？ もちろん OA, EA, FA, LA, PA など分野ごとに異なるだろうが...

(11) 導入の「効果」という観点からの評価報告をお願いしたい．(12) CASE の背景にある方法論自体が普及していな

いのに，CASE ツールが真価を発揮できるのだろうか？ 構造化手法は，日本の企業社会（特にソフトウェア業界）の風土になじむのだろうか？ まず，周辺環境の整備から始める必要があるのではないか？ そういった疑問があるので，実際有効に活用されている事例があれば，ぜひお聞きしたい．

(13) 何を期待して，CASE を導入したか？ 実際に，どのように使っているか？ 使っていない分野はどんなところか？

(14) CASE ツールによって作成された仕様書などの新たな資産の管理方法．

(14) 多人数のプロジェクトでうまく使えるか？

(15) 本当に SA/SD を使いこんだユーザの経験談を聞きたい．

(16) LowerCASE と UpperCASE とを分けて，取り上げては？

(17) いろいろな CASE ツールに対する共通 DB の構想は？ 手法を理解していない人に対する支援は？

(18) 各工程間の連係のアーチテクチャをどう考えるか？ かけ声の割には CASE が普及していない原因は？ 現在の CASE に求められている機能を再度考えなおす必要性はないか？ それぞれの手法は，本当にこれでよいのか？ チーム開発の場合，１人１台の端末がないと効率が悪いのだが，どうする？（わが社では端末が１０人に１台ぐらいしかなく，YPS を使用しているが，YAC チャート作りで，みんなが端末をあくのを待っている状態である！）．

(19) CASE ツールのユーザインタフェースは現状でよいのでしょうか？ たとえば，ハワイ大学で作られた PWB のそれとくらべると大きな開きがあり，改善の余地があると思われます．Mac Power 創刊号での Alan Kay の UI についてのコメントは説得力があると思います．遊び心が欲しいですね．ツール自体のチュートリアルも必要でしょう（Mac のソフトでは Tutor がついているものが多い）．また，グループワークへの支援は，どうしたらよいのでしょうか？ Smalltalk の Change 風な変更管理などの話題も取り上げていただければと思います．

(20) CASE を導入すると，ライフサイクルの各工程はどう変わるのだろうか？ 各工程間のつなぎはスムーズに行くのか？ どうすればスムーズに各工程が流れるのか？ CASE ツールは，各工程をどんな手法でサポートしているのか？ それらの有効性は？ まだ，サポートしていない（またはサポートが足りない）工程はどこか？

(21) ソフトウェア開発および保守におけるルーチンワーク（こんなことぐらい，自動化できる）．CASE の修得のしやすさ．CASE に対する組織（いわゆる文化的慣性）の抵抗．

(22) CASE ユーザとして導入する前にやるべきことは何か？ ツールの徹底比較．

(23) 人工衛星，宇宙ステーション，戦闘機用搭載ソフトウェアなどの開発における CASE の有用性．

(24) CASE 普及のためには，ユーザ／業界／ベンダがどう変わらないといけないか？

(25) 実際のプロジェクトでの活用例とその効果．導入に必要な基礎知識の学習方法．導入を誤った失敗例．

(26) 実際の導入や利用にさいして出てきた問題とその対策について．

(27) ツールを導入した組織での実際の印象が聞きたい．導入のためのスケジューリングは？ できればデモも見たいが....

(28) CASE を使って，Write-Only のドキュメントをどうやって保守に役立つようにするか？ ハトレー／ワードの方法論の教育サポートはどうするか？（納入先でトラブルないか？） 下流工程との連動はどう支援されてゆくのか？（単なるオープンアーチテクチャではダメ，ポリシーが必要だと思う）． CASE で設計したドキュメント資産をどうやって同種のアプリケーション開発に再利用できるようにするか？（インスタンス生成はできないか？） CASE で設計したドキュメントから設計エラーをどのように見つけ出すか？（たとえば，状態遷移図をシュミレーションして，デッドロック状態を設計時に発見できないか？）

SEA 会員アンケート中間集計

（２月末日現在，回答 146通）

「1990 年代を迎えて，われわれソフトウェア技術者を取り巻く状況は大きく変わろうとしています．SEA 会員のみなさんは，そうした時代の流れをどんな風に読み取っておられるのでしょうか？」というアンケートを，春のセミナー案内と一緒に実施しました．

締切りは，一応過ぎているのですが，まだボチボチ回答が戻って来ているので，最終の数字はあらためて報告します．それと，最後の質問（未来予測）の自由回答は，まだ入力が終わっていませんので，次号までお待ちください．

Q1 90年代を象徴するキーワードを３つ挙げてください．

オブジェクト指向	62
CASE	56
ネットワーク	43
分散開発環境	19
ワークステーション	19
ユーザ インタフェイス	16
協調支援	14
AI/KB	13
分散処理	13
ハイパーテキスト	12
マルチメディア	12
UNIX	11

Q2-1 日本で，１人１台の WS という開発環境が一般的になるのは，何年後でしょう？

1 - 2 年後	19
3 - 4 年後	60
5 - 7 年後	47
10 年後	19

Q2-2 開発環境のみならず，一般のコンピュータ・アプリケーションも，WS と LAN を活用した分散型に移行すると思いますか？

思う	115
思わない	6
わからない	21

Q2-3 UNIX の今後について？

すべてで UNIX が標準に	29
影響範囲は WS だけ	75
新しい標準 OS が登場	35
わからない	14

Q3-1 CASE の今後について？

ブームは一時的	13
普及にはしばらく時間が	126
爆発的に広まる	7

Q3-2 あなたにとってはどんな CASE が有用ですか？

UpperCASE	84
LowerCASE	19
どちらともいえない	43

Q4-1 Σ計画をどう評価しますか？

成功	0
やや成功	7
どちらともいえない	21
やや失敗	29
失敗	90

Q4-2 ΣOS やΣツールの普及は？

かなり普及する	0
ある程度は普及する	15
あまり普及しない	126
わからない	6

オブジェクト指向，CASE，ネットワークの３つが人気とは，やはり時代の流れなのか，それとも SEA のメンバーには，以外にミーハーが多いのでしょうか？

１人１台の時代が来るには，まだ少し時間がかかりそうですね．CASE の普及が手間どりそうなのも，１つには，そのあたりに理由があるのかも．

それにしても，Σ計画の評価に関しては，みなさんかなり手きびしいですね．

Calll for Papers その他

1. Call for Papers

次ページ以降に，SEA が（というより SEA 会員の何人かが）企画・運営に関係している３つの国際的イベントの Call for Paper を掲載しました．最初の２つは，もう Deadline が目前まであせまっています．急いで応募してください．ますます激しさを加えつつある国際化のプレッシャーにあえいでいる日本ですが，何とかソフトウェア技術の分野だけでも，諸外国とくに欧米と肩をならべて行きたいというのが，「一寸の虫にも五分の魂」的な SEA の意地です．

(1) 6th International Software Process Workshop

１０月末に北海道（函館）で開かれます．今回のテーマは，「プロセスに対する支援」で，ツールや環境だけでなく，組織や方法論上のメカニズムも対象になります．特に大規模開発の問題がサブテーマの１つになっているのは，日本的なプロジェクト・マネジメントも話題にしたいということなのでしょうか？　日本での開催ということで，東工大の片山先生が運営委員長役をつとめられ，他に，落水（静岡大），岸田（SRA）の両氏が委員会に入っています．応募の締切りは４月１日ですが，まあ１週間ぐらいの遅れは許してもらえるでしょう．だれですか，エイプリル・フールだからいい加減なペーパでいいなんていうのは！？

(2) 4th Software Development Environment Symposium

１２月初めに，有名な Arcadia プロジェクトの本拠地であるカリフォルニア大 Irvine 校で開かれます．SEA では，第２回（Palo Alto），第３回（Boston）に引き続いて，今度も研修ツアーを出す予定にしています．応募の締切りは４月中旬．ふつうの論文だけでなく，ツール・デモンストレーションのプロポーザルでもかまいません．日本からは，阪大の鳥居先生が１人だけ PC に入っています．応募される方は，一言声をかけておかれるとよいでしょう．

(3) 13th International Conference on Software Engineering

第１０回以来 Annual Conference になった ICSE ですが，来年は５月に Texas 州の州都（というより有名な MCC 社の所在地といったほうがよいでしょう）である Austin で開かれます．実行委員長は，その MCC の Belady 氏，プログラム委員長は，AI で知られた Barstow さんと鳥居先生の２人です．こちらのほうの締切は，８月末ですから，まだ大分ゆとりがあります．しかし，論文の採用確率が 10 - 20 % と審査のきびしい会議ですから，応募される方はしっかり腰をすえてペーパを準備してください．

2. 日本ソフトウェア科学会への入会案内

以前からいくつかのワークショップを共催して友好関係にあった日本ソフトウェア科学会との間で，これからはお互いの機関誌に相手側の入会案内をのせようという約束ができました．すでに，あちらの機関誌には SEA の案内が掲載され，何人かの方が新入会されています．今後も，こうした友好関係を発展させて行こうと考えていますので，よろしく．

6th INTERNATIONAL SOFTWARE PROCESS WORKSHOP

Support for the Software Process

Hokkaido, Japan 29 - 31 October 1990

(Sponsored by the Rocky Mountain Institute of Software Engineering - cooperation requested from Japanese and US professional societies)

ORGANIZING COMMITTEE

Gail Kaiser Takuya Katayama Kouichi Kishida
Koichiro Ochimizu Bill Riddle Jack Wileden

The 6th International Software Process Workshop will focus on the provision of support for the software process by software development environments. Of particular concern are environment architectures and mechanisms that directly support the recording, monitoring, guiding and enaction (that is, interpretation or execution) of process models. Among the more important issues are the following:

Process Oriented Architectures

Environments that support process model enaction will need specialized architectural features.

- What overall environment structures are needed for software process support?
- What environment components, e.g. process model interpreters, are required?
- How can process support be built on existing bases such as PCTE?

Process-related Mechanisms

Process oriented environments will need appropriate mechanisms to create, customize and instantiate process models, as well as to define, record, monitor, guide and enact model instances.

- What mechanisms are needed to define, record, monitor, guide and enact process model instances?
- What mechanisms are needed to create, customize and instantiate process models?
- What mechanisms are needed for dynamic modification and refinement of process model instances?

Support for Development-in-the-large

Industrial scale software development requires process models and supporting mechanisms which address group cooperation and coordination, as well as support and/or enforce project-wide management policies.

- How is group coordination and cooperation supported?
- How are policies supported or enforced?
- How is support for management activities integrated with support for other project activities?

The three day workshop, which will be held in Hokkaido, Japan will consist of intensive discussion of these issues by at most 35 Participants. Prospective participants should submit a maximum three page position paper in English by 1 April 1990, explicitly addressing one of the workshop issues, and suitable for publication in the proceedings. A small number of participants will be requested to prepare short keynote presentations to initiate discussion. Papers (6 copies or electronic mail) should be sent to:

William E. Riddle
Rocky Mountain Institute of Software Engineering
PO Box 3521, Boulder, Colorado, USA
email: ispw6@sunset.sda.com

electronic mail submissions should include author's full address and telephone number

Call for Papers

ACM SIGSOFT '90: Fourth Symposium on Software Development Environments*

Irvine, California
December 3–5, 1990

Symposium Scope and Theme: Coping with Change

The overall scope of the symposium is indicated by the list of topics below, each of which is a relevant topic for contributed papers. Papers may range from consideration of select principles to presentation of key lessons drawn from experience.

- Conceptual Models of Environments
- Environment Infrastructures
- Support for Different Software Processes
- Environment Integration Strategies
- Security Concerns in Environments
- Database Support for Environments
- Empirical Evaluation Using Environments
- Comparative Analysis of Environments
- Environment Generators
- Environments and Issues of Scale
- User Interfaces
- Human Factors Studies
- Graphics and Editing Models
- Monolingual/Multilingual Environments
- Production Quality Environments
- Distributed/Network Environments
- Workstation-Based Environments
- Environment Technology Insertion

Software Development Environments (SDE's) must cope with the changing needs of their users and changes in available tools and hardware platforms. Yet if environments accommodate change without simultaneously remaining integrated, major benefits of environments are lost. The primary feature distinguishing an SDE from an ad-hoc collection of tools is a set of integration mechanisms which enable tools to cooperate efficiently, and which provide users with consistent views of the environment and its services. Retaining the beneficial features of these integration mechanisms in the face of changes is a major challenge. All the major aspects of environments are involved, such as object management, process models, user interfaces, methodologies, languages, and operating system substrates. Authors are encouraged to emphasize issues relating to support for change in their contributed papers.

Authors should explain what is new and significant about the work presented and must adequately compare the work with similar work or similar systems. Authors must list key ways in which their contribution is unique or important. Papers will be reviewed by the Program Committee.

Information and Instructions for Authors

Please send **six copies** of your full paper, including an abstract, to the Program Chairman. Papers are limited to 6000 words.

Richard N. Taylor, Chair 4SDE
Department of Information and Computer Science
Computer Science Bldg. Room 444
Parking Lot 18
University of California
Irvine, CA 92717 USA

Submission Deadline: April 16, 1990
Acceptance Notification: July 1, 1990
Camera-ready paper due: September 1, 1990

Authors of accepted papers will be requested to sign an ACM copyright release form. The best papers will be candidates for planned special issues of appropriate refereed journals. Proceedings will be distributed at the Symposium, as a special issue of Software Engineering Notes, or may be purchased from ACM.

Demonstrations

Proposals for demonstrations of prototype or production quality environments and tools having novel capabilities related to the scope of the symposium will be accepted by the program committee. **Six copies** of a short abstract (3 to 5 pages) describing the environment or tool, what is new and significant about the environment or tool, experience with the environment or tool, its current status and availability, and its hardware/software requirements should be submitted by April 15, 1990 to the program chairman.

* Sponsored by ACM SIGSOFT

13th International Conference on SOFTWARE ENGINEERING

CALL FOR PAPERS

SYSTEM DESIGN: RESEARCH & PRACTICE

Austin, Texas May 13-16, 1991

CHAIR:
Laszlo A. Belady
ICSE13 Chair
MCC
P. O. Box 200195
Austin, Texas 78759 USA
belady@mcc.com
512/338-3504
FAX: 512/338-3899

PROGRAM CO-CHAIRS:
David Barstow
ICSE13 Program
Schlumberger Laboratory for Computer Science
P. O. Box 200015
Austin, Texas 78720-0015 USA
barstow@slcs.slb.com
512/331-3728
FAX: 512/331-3760

Koji Torii
ICSE13 Program
Dept. of Information & Computer Sciences
Osaka University
Machikaneyama 1-1
Toyonaka City
Osaka, JAPAN
torii%tori2.ics.osakau.ac.jp@relay.cs.net

TOOLS FAIR CHAIR:
Laurie or John Werth
Department of Computer Science
University of Texas at Austin
Austin, Texas 78712 USA
lwerth@cs.utexas.edu
jwerth@cs.utexas.edu
512/471-7316
FAX: 512/471-8885

TUTORIAL CHAIR:
Herb Krasner
Lockheed
Software Technology Center
O9610/B30E
2100 East St. Elmo
Austin, Texas 78744 USA
krasner@stc.lockheed.com
512/448-5738
FAX: 512/448-5728

LOCAL ARRANGEMENTS CHAIR:
Bryan Fugate
ICSE13 Local Arrangements
MCC
P.O. Box 200195
Austin, Texas 78759 USA
fugate@mcc.com
512/338-3330
FAX: 512/338-3899

ell into its third decade, the engineering of software is becoming the engineering of computerized applications. Increasingly, new systems arise through the adaptation and integration of existing applications, with software as the glue that holds the pieces together (in much the same way that CAD and CAM have been integrated as Computer Integrated Engineering). Creating these systems requires the expertise and cooperation of a wide array of specialists — specialists in the application domain, in real time performance, and in reliability. The software engineer becomes a system designer who must understand the application domain in order to select not only the system's software but also its hardware. Perhaps most importantly, the system designer must be able to work well in teams with others.

These complexities are giving rise to new, often interdisciplinary software engineering research subjects. Already, a wealth of experience and insight has emerged from research in such disciplines as computer supported cooperative work, distributed systems design, reverse engineering, integration technology, and computer aided education and training. And the more established research subjects — quantitative methods, languages and representations, project management, and others — are expanding in breadth and excitement. From this work, a new generation of tools and techniques for improving the precision, quality, and predictability of system building projects is arising. Without superb technology and continued research, companies and nations will lack the productivity to be competitive.

ICSE invites researchers and practitioners to share their recent ideas and experiences, particularly those which aim to improve the design of complex computer applications. The program committee will review all submissions for quality and for relevance to future work. Papers, panel proposals, and reports must be received in writing by September 14, 1990.

SUBMISSIONS: Eight (8) copies in English of papers, panel proposals, or experience reports should be submitted to the address below by September 14, 1990:

David Barstow
Schlumberger Laboratory for Computer Science
P. O. Box 200015
Austin, Texas 78720-0015

Papers should be limited to 6000 words, full-page figures being counted as 300 words. Each paper must include a short abstract, a list of keywords, and the lead author's address.

Panel proposals should include title, proposed chair, tentative panelists (including a short vita), a two or three paragraph description of the subject, format of the presentation, and rationale for the panel.

Experience reports should describe practical experience in some aspect of software engineering (using a tool or method, applying a metric, following a project management discipline, reusing work products, etc.). The contributor(s) should submit a 5 to 10-page written description of the experience and a one-page outline for a five-minute presentation.

TOOLS FAIR: A software tools fair will be held during the conference, so that attendees can see and learn about current experimental and commercial software tools. Those who want to exhibit or demonstrate a tool, and especially those interested in presenting a descriptive paper, should contact:

Laurie or John Werth
Department of Computer Science
University of Texas at Austin
Austin, Texas 78712

FURTHER INFORMATION: For further information and/or copy of the advance program when available, write either to ICSE13, MCC, P. O. Box 200195, Austin, Texas 78720 USA, or to ICSE13, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington, DC, 20036 USA.

IMPORTANT DATES: Submission deadline: 14 September 1990; Acceptance notification: 14 December 1990; Final versions due: 8 February 1991.

Sponsored by ACM Special Interest Group on Software Engineering
IEEE Computer Society Technical Committee on Software Engineering

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE

●入会のご案内

日本ソフトウェア科学会

Japan Society for Software Science and Technology

日本ソフトウェア科学会事務局
〒105 東京都港区浜松町2-4-1
世界貿易センタービル私書箱104
電話 03-436-4536

●ソフトウェアとハードウェアが計算機システムにおける車の両輪であることは言うまでもありません．そのソフトウェアの生産性や信頼性を向上させ，計算機システムを飛躍的に高度なものとするためには，しっかりした基礎づけの上にソフトウェアの新しい概念や方法が是非とも開発されねばなりません．

●このためには，ソフトウェアの基礎理論を構築し，ソフトウェア工学を充実させ，人工知能や知識工学を豊かなものにする——つまり，ソフトウェア科学の確かな発展こそが必要だと考えます．

●〈日本ソフトウェア科学会〉は，こうした切実かつ重要な課題に真正面から取り組む学会として発足しました．ソフトウェアの学問的進歩に寄与しようと志す研究者，さらにその他の分野でソフトウェアに関心を持っている方々にも，会員としてご参加いただきたく，ここにご案内申し上げます．

●会員制度

[正会員]——大会，研究会での発表・学会誌への論文投稿をはじめ，学会誌，英文学会誌の受領など，学会活動全般にわたる権利を有します．入会には，正会員1名の紹介が必要です．(年会費8000円，入会金2000円)

[学生会員]——正会員と同等の権利を有しますが，大学学部・大学院在学中の人に限ります．入会には正会員1名の紹介が必要です．(年会費5000円，入会金1000円)

[準会員]——和文学会誌の購読を主とする会員で，資格は問いません．(年会費6000円，入会金なし)

この他に，学会活動を賛助する法人・個人の[賛助会員](年会費一口五万円)の制度もあります．

●学会活動

季刊の学会誌『コンピュータソフトウェア』，年刊の英文学会誌『Advances in Software Science and Technology』の発行．大会(年1回)，研究会，ワークショップの開催，講演会・講習会の随時開催．

●入会申込み方法

[1] 本学会に入会を希望される方は，学会所定の入会申込書（別紙、複写でも可）に必要事項を記入し，お送り下さい．

[2] 正会員または学生会員として入会を希望される場合は，本学会の正会員1名の紹介を得て下さい．お近くに正会員がいない場合は，紹介者の欄は空欄のまま入会申込書をお送り下さい．事務局で検討のうえ，紹介者のお世話をいたします．

[3] 入会申込書の到着後，当学会からの通知に従い，必要な金額を振り込んで下さい．

[4] 毎月末までに会費の納入が確認された分については，翌月から本学会員として登録いたします．

●問合せ

入退会，住所変更等の連絡や会費納入等の会員業務については，次へご連絡下さい．

学会事務センター
〒113 東京都文京区弥生2-4-16
学会センタービル
電話03-817-5801

その他，学会活動に関する事項については，学会事務局へお問合せ下さい．

入会希望者は申し込み用紙に必要事項を記入し下記住所宛お送り下さい。

〒113 東京都文教区弥生2-4-16
学会センタービル内
（財）日本学会事務センター気付

日本ソフトウェア科学会

会員番号　　**日本ソフトウェア科学会 入会申込書**　　受付日

	氏名	氏	名	19　年　月　日申込
正会員・準会員・学生会員	ローマ字			19　年　月　日生
勤務先（在学校）	名称　　部課			職名(学年)
	〒 電話　　－　　－　　内線			
現住所	〒 電話　　－　　－			
学　歴	大学 学校　　学部　　学科　19　年卒 大学大学院　　研究科(修士・博士)　19　年修了			博士号
通信先	□ 勤務先 □ 現住所	名簿記載	□ 勤務先 □ 現住所	紹介者(準会員申込みの場合は不要) 会員番号 (　　)　署名　　印

ＳＥＡ会員状況（平成2年3月5日現在）

正会員	1773名
賛助会員	68社

正会員の勤務地および居住地域分布

	勤務地域	居住地域
北海道	10	10
青森	0	0
山形	1	1
宮城	6	5
岩手	11	11
福島	1	1
秋田	1	1
新潟	7	7
栃木	6	6
群馬	6	6
茨城	13	14
埼玉	24	130
千葉	24	104
東京	930	551
神奈川	147	329
山梨	1	1
長野	38	39
富山	3	3
福井	4	3
石川	3	3
静岡	19	19
岐阜	2	7
愛知	62	51
和歌山	2	3
三重	1	5
滋賀	11	12
京都	25	34
大阪	244	175
奈良	4	16
兵庫	39	77
愛媛	3	3
徳島	7	7
高知	0	0
香川	1	0
鳥取	0	1
島根	1	1
岡山	2	2
広島	5	5
山口	0	0
福岡	19	21
大分	6	6
熊本	34	33
長崎	0	0
宮崎	1	1
佐賀	1	1
鹿児島	5	5
沖縄	1	1
韓国	1	1
米国	4	4
豪州	1	1

＜男女分布＞

男	＝	1681
女	＝	92

＜年齢分布＞

20以下	＝	0
20_24	＝	42
25_29	＝	354
30_34	＝	486
35_39	＝	404
40_44	＝	296
45_49	＝	123
50_54	＝	32
55_59	＝	19
60以上	＝	15

＜血液型分布＞

A型	＝	686
O型	＝	508
B型	＝	390
AB型	＝	189

賛助会員会社名

ジェーエムエーシステムズ，日本システム，リコーシステム開発，ＳＲＡ，辻システム計画事務所，ＰＦＵ，近畿日本ツーリスト，東電ソフトウェア，セントラル･コンピュータ･サービス，構造計画研究所，ケーシーエス，サン･ビルド印刷，富士ゼロックス情報システム，アイエヌ情報センター，富士通，エムテイシー，クレオ，情報システムサービス，日立ビジネス機器，アトラス情報サービス，三菱電機セミコンダクトソフトウェア，シーアイシステムデザイン，三菱電機，富士通ビーエスシー，千代田製作所，村田技研，ニコンシステム，昭和電工コンピュータサービス，エヌ･ティ･ティ･システム，日本情報システムサービス，ヒラタ･ソフトウェア･テクノロジー，日本エム･アイ･シー，日本電気ソフトウェア，ソニー，コマス，三菱電機コントロールソフトウェア，インターフィールド･システムズ，ジャストシステム，マイクロリサーチアソシエイツ，松下ソフトリサーチ，シャープ，東洋エンジニアリング，日本ユニシス･ソフトウェア，セイホーソフトウェア，新日鉄情報通信システム，テックシステムズ，熊本電子計装工業，ＮＴＴ九州技術開発センタ，オムロンソフトウェア，三菱電機関西コンピュータシステム，日立エンジニアリング，スインク，テスク，リコー，三菱電機東部コンピュータシステム，カシオ計算機，キャノン，ソフテック，富士写真フィルム，中央システム，安川電機製作所，富士通エフ･アイ･ビー，日本データスキル，明電舎，リバテイーシステム，東北ＳＲＡ，池上通信機，エイ･エス･テイ

sea
ソフトウェア技術者協会
〒102 東京都千代田区隼町2-12 藤和半蔵門コーフビル505
TEL.03-234-9455　FAX.03-234-9454