Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ソフトウエア工学は現場に定着しつつある（徐々にしかし確実に）

野村行憲 (岩手電子計算センター)

1. はじめに

私は学会その他の機関には無縁であり，活動の内容も知らない．しかし，「再利用」，「プロトタイピング」，「オブジェクト指向」などという言葉は知っていて興味をもっている．そしてそれらは着実に我々の現場を侵し始めている．この傾向は当社がソフトウエアの開発を受注するようになってから顕著になってきている．

2. いつのまにか構造化が定着

アセンブラでスパゲッティプログラムを書いていた人たちが，COBOL でプログラムを書くようになっても，スパゲッティプログラムを書いていた．それが当たり前のことであって，誰も疑問を持たなかった．ただ，そうした環境の中でも，一部の人はプログラムを小さなサブルーチンに分ける書き方をしていた．しかし，この今でいうモジュール化されたプログラムは，他の大多数の人からは「複雑で見にくいプログラム」と呼ばれていた．それが今ではなぜか大部分の人が，当然のことのように構造化された書き方をするようになっていて（COBOL で！）誰もその書き方が複雑で見にくいなどとはいわない．しかもそれは，ソフトウエア工学の実戦などという大義名分ではなく，それどころか，ソフトウエア工学という言葉も知らないまま，使われている．こうした方法が，いつ誰によって開発され，どんな効果があるかといった議論もなく，問題意識も少ないまま定着している．それでいて，プログラムの生産性や保守性は，かなり向上されている（ようである）．

3. 部品化／再利用も行なわれている

当社ではメインフレーム上で TSS による CRT 上での開発環境が整備され，ソースライブラリの管理が楽になったことと，COBOL の Copy-Library 機能が拡張され，細々とながら部品化／再利用が行なわれるようになってきている．おもな使われ方は，データ構造定義や，ロジックスケルトンの共用／再利用である．これは，そうしなさいと号令をかけたわけではなく，それぞれがそのほうが便利だからそうしているにすぎない．しかし，そのきっかけは文献などからの知識にあったようである．

4. 現場の論理

ソフトウエアの工学的な研究は研究者に任せておけばよいというのが，現場でのおおかたの考え方であろう．その成果のうちから，おいしそうなものを使わせてもらうのが，最も安直な方法である．また，現場には納期最優先の考えが依然としてあり，研究のための調査やデータ収集が行ないにくい環境にある．また，プログラマーの研究意欲も少ない．

5. 管理者の意識

ソフトウエアの工学的な研究が現場に定着しないもう一つの原因に，管理者の意識の問題がある．彼らは依然として，古い殻に閉じ篭り，自分の経験が全てであると信じているように振る舞っている．そして自分の知識の立ち遅れを感じてはいるものの，新しい技術に対してなぜか近づこうとしない．新しい技術を闇雲に取り入れることが得策ではないが，目を閉じているのは単に自身の立ち遅れを助長するだけではなく，その部下をも巻き込み，ひいては会社にとってもマイナスとなろう．ソフトウエア工学の実戦にとって，管理者の再教育こそ，最も重要でしかも最も難しいことであろう．

6. ワークショップに期待するもの

以上で述べたように，私は研究者が苦労して研究した成果を嗅ぎ回って，おいしそうな物を見つけてはちゃっかりいただこうとする，いわばハイエナの立場である．

今回のワークショップでは，鼻の利く人たちが大勢参加しそうなので，彼らとのデスカッションを通じて，最新情報の中から彼らがよさそうだと思っていることを聞き出して，（忘れずに）会社まで持ち帰ろうと思っている．せいぜいそのときまで自分の鼻を鍛えておくことにする．

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

気配りがソフトウェアクライシスを救う !?

濱田 勉 (NTT 九州技術開発センタ)

本題に入る前に私の職場の紹介をします．NTT のソフトウェア開発部門というのは，社内システム，交換機ソフトのサービス・メンテ部門，それに組織の分かれました NTT データなどがメジャーなのですが私の職場はそれらとおもむきを少し異にした，支店・営業所（前電話局）等の色々な開発要望，技術相談の内特にパソコンに絡んだものの技術的サポートをする部門です．NTT 内でははっきりいってマイナーです．

ソフト担当とはっきり位置付けられて１年半，社内向けのソフトを早く仕上げるのを命とした職場です．社員のソフトウェアマインドを醸成することも目的の１つになっています．残念ながらカチッとした組織にできあがっていません．逆にそれが自由な雰囲気につながっているのかもしれません．

さて，私はまともにソフトウェア工学を学んでいません．

ときどき自分みたいなのがソフトウェア開発してていいのだろうかと不安になりますが，この職場は人材育成も兼ねているのだからと自分を慰めています．

そんな私が日頃心掛けていることは，とにかく誰が見てもわかるようなコーディング（たとえ冗長といわれても），技は使わない（技を知らないが正しい），痒いところに手が届くコメント，後で他の人が利用できるようなモジュール作り，及び簡単な利用説明書をつくることなどです．

と思ってはいるものの，その気配りが自分の中でも，プロジェクトグループの中でも標準化というかルール化というかそういうものができていない．結構気分でコメント入れたり入れなかったりしているわけです．

あるとき，社内向けの「プログラム設計技法」なる訓練資料を見る機会があり読んでいると，おきまりの「ソフトウェアの危機」の話にはじまり，「よいプログラムの条件」というのがありました．３つありまして

- 信頼性： 長時間（期間）使用してもエラーがなるべくでないプログラム
- 保守容易性： エラーが出たとき，それをできるだけ容易に修正できるようなプログラム
- 拡張性： ある機能を修正したり追加するときに，プログラムを全面的に書き直すのではなく，できるだけ簡単にこの要求を満たせるようなプログラム

とあり，「エラーがなるべくでない」という表現にはなにか悲しい響きを感じましたが，それはさておいて，まとめが「簡単でわかりやすいプログラム」となっているわけです．その次の章が具体的な設計技法の話になるのですが，ここでふと

「プログラム設計技法の目的は，気配りの標準化なのではないか？」

と感じたのです．

いま，他人の作ったあるシステムの機能追加及び虫取りをしているのですが，すごいというか迷惑というか，詳細設計なしに頭の中のフローだけで数キロステップのプログラムを書いていてコメントもあまりない，変数名が，なんとか１，なんとか２，とかなってたりで，せめてコーディング後でもいいからチャートを残してくれればいいのに，それもなしに転動してしまいました．でどうしたかというと，機能追加に関係している関数が影響を与えていそうな関数や外部変数などを grep, awk, sort, sed なんかでしこしこみつけて因果関係一覧つくったり，ＨＣＰにかいたりといらん手間をくったわけです．

彼はなぜそんな作り方をしたのか？

(1) 納期が決まっていて，とにかく動くものを作らなければならなかった（時間の余裕）．
(2) プログラム設計技法を学ぶ機会がなかったか，知ってもやる気がなかった（スキルとモラル）．
(3) 職場に決まった設計書のフォーマットがなかった（ルール）．

というようなことを想像したわけですが，さていろいろなツールや技法はこれらの不備を補ってくれるのでしょうか？ それとも，このようなことで開発に手間取っているのは私の職場だけなのでしょうか？
この辺のことを聞いてみたいと思います．

ここでハード的な環境を少し紹介します．

開発に携わる人間 9 人に対し，ワークステーションが 4 台，パソコンが 9 台，イーサネットですべてつないで PC-NFS を利用しています．JUNET にもアクセスできます．あと，3 台のプリンターを共同で使えるようにプリンタ専用のネットも張っています．

この環境は徐々に構築していったのですが,「生産性をあげるための環境」というのと「開発者が快適に仕事が出来る環境」というのはかならずしもリンクするとは限らないんだぁと最近思うようになりました．この辺の話も聞けたらいいなと思います．

かなり散文的で拙い文章を読んでいただきまして有難うございます．

私は理論武装できるような知識や経験はほとんど皆無です．今回はソフトウェア技術者の先頭を走るためのワークショップになる可能性があるとか．いわゆる中野先生のいわれる遊離する普通（？）の会員です．それでも応募したのは北海道のラーメンとカニが食べたかったからです:-)

よろしくお願いします．

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ポジション・ステートメント

平尾 一浩 (ヒラタ ソフトウェア テクノロジー)

1. ソフトウェア工学は現場に (今のままでは) 生かせない ??

ソフトウェア工学は本来，開発現場での高信頼性や高生産性を目指して研究が行われてきた．しかし，今まで指摘されて来たように，単体の技法群であり，現場に即した方法なりを明確にして来なかった．また，それを使う側は，ソフトウェア工学を取り巻く環境の受け取り方を間違っている部分がある．(たとえば，CASE ツールは本来，ある方法論または技法に従った開発を行う時，その作業を支援するためのツールであるはず．しかし実際はその方法論までが期待されて，そのツールの評価が方法論の評価に置き換えられている場合がある)．

ソフトウェア工学が現場で生かせるかどうかというのは，現場の開発者がソフトウェア工学に何を期待し，何を望んでいるのかをはっきりさせた上で，それが現在のソフトウェア工学と比べ，どの部分が不足し，どこにギャップがあるのかを検討する必要があると思う．その後，現場で生かせるための方法なりを討論するのがよいと思っている．

ぶつぶついっていてもプロポーザル・ワークショップのポジション・ペーパにはならないので，少ない経験しかないが，開発の現場にいる者として感じた問題の中から，大きな問題と思われるものと，その解決に向けての提案，方向をまとめることにする．

2. 大きな問題だと私は思います

日常の開発現場では色々な問題をその時々に感じる．今回のポジション・ペーパを書く機会を利用して問題を整理してみた．その結果，以下の問題が多くの問題の原因になっているように思える．

- **やってみなけりゃわからない！**

  ソフトウェア工学で提案されてきた方法論なりを使用して分析，設計，開発を行ってみたが，結局やってみなけりゃわからない，という部分が多過ぎる．FA を主業務としているが，ハード(プログラマブル・コントローラ，バーコードリーダ，ロボット・コントローラ，マルチ・コントローラ...etc) によって左右される部分が多い．ソフトウェア工学では，物理的制約などの調査を行うようにいわれているが，実際の "物" の流れから，それぞれが制約を受けることになり，確実な仕様というのを把握しきれない．結局，経験と，感と，現場での開発が主になっている FA 分野で，ソフトウェア工学を使って効果を出すのは困難なように思える．

- **実現へのステップがない**

  複数の分析，設計技法を用いて開発を行っても，それから実際の実現手段というのが明確にされていないように思う．個々の技法，方法論は，分析設計の単体作業を対象にすればよいと思うのだが，それらの方法論を組み合わせた形での全体的方法論がないため，即，現場に持っていくということが困難であるため，実用に則さないというしかないのではと思う．一つに，ソフトウェア開発のライフサイクルモデルがあるが，各作業フェーズの無理な線引きと，かなり抽象化されたモデルれあり，とても定量的な評価や判断まで持っていけない．ましてやそれぞれの方法論をつなぎ会わせることなど無理なように思う．

結果はソフトウェア工学の問題というよりも FA 開発現場での問題みたいになってしまった．しかし，ソフトウェア工学で提唱されている技法を用いても，上に出した問題は解決していないということをいいたい．つまり，今の開発現場ではその効果が出にくい．

3. 現場で必要と感じたもの

上に挙げた問題を解決するために，現場サイドから何があったらよいと思うのかを以下に挙げる．

- 具体的なライフサイクル事例集
- 具体的な技法使用例 (開発事例)

やってみなけりゃわからない，という問題の多くは人に頼り過ぎた開発を行うためであり，過去の実例などを

うまく利用しきっていないために何度も同じ部分で悩み，失敗をしていると思う．しかし，今はその実例，事例を後に伝えるための方法が確立されていない．プロセスを簡単に記述，変更できれば，それを基に経験の少ない者は疑似体験でノウハウの一部を得ることができるだろうし，実際の開発者は見積りを含めてプロジェクトの計画がより正確にできるようになると思う．さらにプロジェクト進行時には今までより数段，信頼の持てる管理が可能となると思う．過去の事例をプロセスという形に残して，ライフサイクル事例(あくまでモデルではない)を蓄積して，何らかの形で分類やそれらの基本となるモデルが出されればよいと思う．

ソフトウェア工学においても，プロセスモデルの研究はされているが，具体的な例として使用するには現場から遠すぎる．このため，現場サイドからその事例を多く提供していく必要がある．そしてそれらの実例を基に，技法，方方論の現場で使い方などを含めたプロジェクトの全体像を描く技術，さらには事前にプロジェクトをシミュレートできるようなものの研究を行っていく必要があると思う．プロジェクト計画も一つのシステム設計でありプログラミングである．となるとこの段階での技法や方法論が必要なのは当然と考える．

まとめると，プロセスと技法，方方論の使用実例を集め，それを基にプロジェクトの設計，開発（プロジェクト計画や進捗）の支援するための方法や，プロジェクト・シミュレーションの技術研究に力をいれていく必要がある，といえるのではないかと思う．勿論，そのためには現場サイドからの実例をより多く出していくということが前提となる．そうなると，その実例を出すためのプロセス記述方法やその支援環境なりを，まず考える必要がある．

ソフトウェア工学は現場に生かせるか？

セッション・サマリ

岸田 孝一
(SRA)

1. プレゼンテーション

チェアマンの気紛れで，話の内容とは無関係に，スピーカの体重の少ない順にプレゼンテーションを行った（詳細は，各人の Pre-WS Position Paper 参照）.

新田： ソフトウェア工学は現場では生かされていない．このままでは生かせない．このへんで，第２ソフトウェア工学を作ることを考えたらどうか？（笑い）

濱田： 現場の最大の悩みは要因の質の悪さ．それをカバーするために CASE の導入を考えたい．ただツールを入れるだけではうまく行きそうにないので，適切なカウンセリングを行いながら....

野村（敏）： 現場はいま惰性で仕事をしている（仕事に追われている）だけで，ソフトウェア工学などを意識していない．せいぜい「こんなツールがあれば！」といった弱い願望を持っている程度．そうした現場に，折角のソフトウェア工学の研究開発成果を導入する唯一の道は，マネージャ達を教育して，「これを使って仕事をせよ」と強制するしかない．

平尾： 現場は「経験とカン」だけで勝負をしている．何とかせねばと思うが，現状，ソフトウェア工学の研究と現場とはあまりに掛け離れている．新しい技法の導入を現場で考えても，結果がどうなるかについての不安のほうが強い．研究者がもっと現場のことを考えて，実証的な研究をしてほしい．

落水： ソフトウェア工学の研究はすでに多大な成果を上げている．にもかかわらず現場に普及しないのはなぜか？ 最大の原因は現場の怠慢（認識）不足である．しかし，研究サイドにも問題がなかったわけではない．「生産性の向上」とか「標準化」とかいった旧来の枠組みはあまりにも夢がなさすぎた．いま必要なのは，現場と研究者とが手を組んで，新しい技術開発の枠組みを考えることである．

佐原： 個人的な経験からすればソフトウェア工学は役に立つ．にもかかわらず現場で使われないのはなぜか？ たぶん，「失敗を恐れる」という保守的なものの考え方が日本の社会では支配的だからではないか？ 失敗を恐れるところに進歩はない．

2. 討論

休憩のあと，自由討論に入った．主な発言は以下の通り（いくらか危険な内容の発言もあるので，記録上はすべて匿名にしてある）:

－ CASE は有効か？ CASE を使いこなすには，それなりの方法論的知識が必要だが，そうした知識を持っている人間は CASE を使わないでもちゃんと仕事ができる．むしろ CASE はわずらわしい．

－ LowerCASE は随分練れてきているから，使えばそれなりの効果がある．UpperCASE がその状態に達するにはもう少し時間がかかるのでは？ つまり，だれでも気軽に使って，簡単に効果が上がることが望ましいが，まだそこまできていない．

－ LowerCASE とかプログラミング言語（コンパイラ，etc）は，いわば個人のためのツールである．それに対して，UpperCASE の場合は，複数の人間の協調という要素を含んでいて，それだけむずかしい．

－ これからのソフトウェア工学研究の目玉は？

－ オブジェクト・ベース，プロセス記述，CSCW あたりか？

－ 教育ということに関していえば，What や How を教えるだけでなく，Why を教える必要があると思う．ソフトウェア工学をほんとうに現場で役立たせるためには，現場のニーズ（多分生産性とか品質とかいうキーワードが重要）と関連させて，Why を十分に説く必要があるだろう．

－ それはそうだが，一面の危険は，生産性・品質重視から安易な「標準化」のパラダイムに走り，現場のやる気をそいでしまうということがある．典型的な悪い例は，メインフレーマのソフトウェア工場．それが制度とし

て固定化すると技術進歩を受け入れる柔軟性が失われる．

－ メインフレーマがそういう方向に走ったのは理解できないわけではないが，アカデミアまでがその影響を受けてしまったというのは？！

－ いや，いまアカデミアはそうしたメインフレーマの動きには批判的になっている．その証拠に，大学の教育現場では，メインフレームのマシンは使っていない（笑い）．

－ 最初に何もなかったところで，標準化を持ち込めば成功する．そのことの悪い影響か？ 旧日本の軍隊が，日清・日露戦争の勝利でいい気になったために，太平洋戦争の大敗北をもたらした．

－ 生産性や標準化を推進している中心人物たちも，それだけでいいとは思っていない．しかし，組織の慣性というか惰性というか，技術の方向を変えられないでいる．

－ メインフレームの世界における標準化のモティベーションは，いま手に入る技術の最高のものを取り入れるというよりは，需要の拡大に伴う開発要員の質の低下を何とかしたい，作業・製品の質の低下を防ぐための歯止めという後ろ向きの標準化ではなかったか？ ソフトウェア工学がそれに手を貸したことがまずかったと思う．

－ 現場は固定的な標準手続きよりももっとフレキシブルなものを求めている．標準化は，アマチュアのために，アマチュアの仕事のレベルを揃えるためのものだ．プロには必要ない．

－ しかし，いまの現場はアマチュアだらけだ（笑い）．

－ 標準化とか，生産性とかいうのは，生産工学とか管理工学のドメインに属している．他の分野の工学では，それぞれの分野での面白い技術トピックスを追求しており，標準化や生産性の話は，生産工学・管理工学の専門家の任せている．どうしてソフトウェア工学の人たちは，それをごちゃごちゃにするのか？

－ ソフトウェアは一品生産だから，標準化とか，生産性とか，品質とかいう問題も，大量生産のハードウェア製造工程とはまったく違う観点で考えられなければならない．

－ 教育現場からの話を１つ．ソフトウェア工学うんぬん以前に，いま企業からの無言の圧力は Cobol や Fortran を一通りは教えてほしいということになっているように感じられる．ソフトウェア工学の観点からすれば，Pascal とか，もっといい言語があるのだが....

－ 開発現場におけるこれまでのソフトウェア工学の実践のプロセスを振り返ってみると，コンパイラ，デバッガ，カバレジ・アナライザとかいったバッチ型の LowerCASE ツールを用いた開発の自動化・標準化を図ってきた（まだそこから脱していない）ように思われる．その分野は，もういくら追求してもあまりおもしろくないので，システム開発の上流工程に焦点をあてて，新しいツール（言語も含めて）の開発を一生懸命やることがいいのでは？

－ 上流工程を考えると，単にプロダクトを記述する「プログラミング言語（仕様記述言語）」だけではなく，開発プロセスを把握しモデル化するための手法／メディアが重要だと思う．でないと，いま紙と鉛筆でやっている仕事をただコンピュータに置き換えるだけに終わってしまう．

－ ソフトウェア工学実践の歴史が LowerCASE 中心だったのは，上流を攻めるのがむずかしかったからか？ それとも，やってみたがうまく行かなかったからか？

－ いや，それはただ，昔のコンピュータ・パワーでは，UpperCASE ツールが動かせなかったからで，いまの CASE のアイデアは，ソフトウェア工学の初期の時点ですでにできあがっていた．いま，ハード（WS）の進歩によって，ようやく実用になっただけの話

－ いま，ソフトウェア開発現場の大多数を占めている Cobol プログラマたちをどうするのか？ 見捨てるのか？ 何かの工夫をして楽にしてあげるのか？

－ 楽にするというのは「殺す」ことか？（笑い）いや，プログラマではなく，Cobol を殺すという意味．

－ 問題は言語ではなくて，Cobol アプリケーションの世界で使われている開発方法論が問題なのだ．いろいろなトラブルの原因をそのままにしておいて，最後につじつまを合わせる．それではうまくいかないということがわかっている．にもかかわらず....

－ その世界はもういくら手をつくしても仕方がないので，新しいアプリケーション分野を切り開き，ソフト

ウェア工学はそこで勝負しようということか？

－　でも，いまのアプリケーション世界の混沌が人間の本性に起因しているとすれば，いくら新しい世界を作っても，結果は同じことになるのでは？

－　これまでのソフトウェア工学の成果は言語（プログラミング言語）の形で残されているといってよい．そうした形のソフトウェア工学の限界が見えて来て，しかし，じゃあどうしたらよいかがまだ分かっていない．それが今の状況だと思う．

－　ソフトウェア工学といっても，「研究」と「実用」という２つをはっきり切り分ける必要があると思う．前者はとにかく新しいアイデアを出すのがポイント．結果がうまく当たるかどうかは「千三つ」の世界（笑い）．一方「実用」の世界では，もっと産学の共同体制を確立して，Programming-in-the-Small ではない，もっと現実的な試作・開発を進めて行かなければならない．

－　企業サイドも将来のビジネスを考えたら，「千三つ」的なアイデアの模索に力をいれるべきだ．横から見ていて大学を批判するだけでは....

－　話を変えて，生産管理や管理工学の分野いについてだが，これらの学問分野はこれまでハードウェア・オリエンテッドな形で発展してきた．その成果をそのままソフトの世界にマッピングすると，ソフトウェア工場とか，ソフトウェア QC とかいう誤った方向に進んでしまう．

－　工学という以上，ソフトウェア工学はもっと現場に密着して，地に足のついた研究をしてほしい．

－　でも，現場ベッタリだけでは駄目．一方では，純粋に研究的なアイデア追求も必要．

－　いま，ソフトウェア工学が現場に生かされていないというのが，この場の共通認識だとすると，それはどちらの責任か？

－　責任がどちらにあるかという話ではなく，常識的な結論だが，双方とも努力しなければ....

－　これからどうしたらいいのか？

－　現場の視点に立ってソフトウェア工学の体系を洗い直す？

－　そへは駄目．いまでさえグチャグチャなのに，もっとひどくなってしまう（笑い）．

－　ソフトウェア工学をわかっている人間が現場に入って実際に指導する．

－　それはまあ地道だが確実．問題は，それができる人間の頭数が少ない．それと受け入れ側も，たとえば，基本的なデータ構造とか，アルゴリズムとか，最低限のコンピュータ・サイエンスの知識ぐらい持っていてくれないと，まったくの野蛮人は教えられない．

－　この前の ICSE でビジネス・データ処理とソフトウェア工学というパネルがあって，そこで，パネリストの１人がいまの世の中のシステムはそれぞれ時々は問題を起こしながらも何とか動いている．だから，ソフトウェア工学なんて面倒なものはいらないのではないか（笑い）という逆説的な発言があったが....

－　経営者の目から見たら，その発言はまったくの誤り．いまのシステムは何とか動いているとはいいながら，毎日多額の損失を播き散らしている．しかも，そのほとんどは，現在のソフトウェア工学手法を適用すれば防げるものばかりだ．

－　もっと大きな目に見える損失の例が，通信衛星の場合だ．せっかく打ち上げた衛星がすぐ故障してしまう．これなんかもソフトウェア工学をきちんと適用すればなんとかなったはずなのに....

－　そうした失敗例を取り揃えて宣伝することもソフトウェア工学の普及に役立つ？

－　CMU/SEI なんて，DoD が過去の失敗にこりて，これからな何とかしようということでできたのでしょう？

－　まだまだ議論はつきないと思いますが，そろそろ時間がきました．

ソフトウェア工学は現場に生かせるか？

セッションの討論を聞いて

酒匂：「ソフトウェア工学」という括り方は雑に過ぎる．よい部品を「よく」組み合わせるための工学が必要であろう．その中には設計工学，意匠工学，合成工学....etc があるだろう．

新田：自分の最近の作業を振り返ってみると，とても「ソフトウェア工学を使っている」とはいえない．もちろん簡単なツールや考えは採用しているが..... やはり文化は言語に先導される．Bottom-Up ではあるが，Programming Language を変えていくことも大事だと思う．

青木：Smalltalk-80（プログラミング言語の１つ）から入って数年，がむしゃらに勉強してきました．そして現在，オブジェクト指向（ソフトウェア工学の一側面）の真髄がわかってきたように思われます．なんと難しいのでしょう！

Smalltalk-80 には，ソフトウェア工学の資産がたくさん内包されていますがそれでも，インテグレーションや高度のデザインなんかには，力が及びません．ソフトウェア工学のその辺の資産が欲しいと願う今日この頃です．

Smalltalk-80 のよい教科書もないけれど，ソフトウェア工学のよい教科書もないのは事実です．言語レベルに固定化されたソフトウェア工学の資産など，たとえば Smalltalk-80 の本は私が書きますから，落水先生，ソフトウェア工学の本を書いて下さい．

林：ウ〜ム，どうもソフトウェア工学というと，「役に立たない vs 役に立ててない」の論争になりがちですね．どうしてなのだろう．ソフトウェア工学に対して持っているイメージが各人によって違い過ぎる（？）．ソフトウェア工学のドメインをはっきりさせたい．

中野：新田さんの話は非常に面白かった．でも，大学の先生（私を除く）は怒るでしょうね．野村さん，大学・研究所に期待しても無理（？）．せいぜい産学共同程度．平尾さん，失敗しても福にすればよいでしょう．落水先生，そんなにカッコよくいってよいのだろうか（？）．

研究としてのソフトウェア工学と，現実に役に立つソフトウェア工学の研究に分けて考えた方がよいのではと思います．前者は，研究としてのレベルを高める（創造性 etc）ことが主眼であり，現場に役立つかどうかは，あまり考えない方がよいと思います．後者は産学共同でガリガリやるべきでしょう．前者は"千三つ"では困りますが，十回に１つ当たればよいのではないでしょうか？　後者はどんどんお金をもらってやれば....

前者は普遍的概念で，後者は Domain Specific になるのでしょうか？

平尾：やっぱり研究するところと使うところが離れすぎている．何とか近づく方法を考えていかなければならないと思った．中野先生のいう研究目的のソフトウェア工学と実用ソフトウェア工学という２面的なとらえ方が１つの解決方法か？

ソフトウェア工学も内容により分類して研究していくことが必要だという意見に賛成．

熊谷：ソフトウェア工学への理解が深まって，私個人にとっては意味があるようにみえた．私の理解では，ソフトウェア作成のあらゆるプロセスに対して，現場に役立つツールや方論を作り出すことがソフトウェア工学の主テーマである，と考える．そういう意味からセッション中に提案した私の案はいかがなものだろうか．ソフトウェア工学専門家の意見を是非伺いたい．

高橋（晃）：わかる人を育てればいいのですが，教育の現場でも大量の COBOL プログラマを生み出す土壌があるのではと感じている．特に専門学校など．

落水：ソフトウェア工学に対する世の中の理解の浅さ（ばらつき）を痛感したセッションであった．

加藤：中野先生のいわれる「研究としてのソフトウェア工学」と「現場で役に立つソフトウェア工学」を区別すべきという意見には賛成です．また，役に立つソフトウェア工学の中でもどこで役に立つかという Domain を明確にし，生かせたかどうか（あるいは失敗したかどうか）を検証して行くことを産学共同でやる必要があると思います．

魚田：ソフトウェア工学は必要である．だがしかし，役に立っていない．今はドラスティックに変わるための充電の時期か？やっぱりよくわからない．個々の技術者（何割かでもよい）が常に新しいパラダイムに向かってアクションを起こして行く必要があるように思う．

野村（敏）：ソフトウェア工学は実践的な学問であり，その成果が現場でいかに活用されるかが重要である．現場とは世の中の 90 数パーセントを占める COBOL/FORTRAN プログラムを作っている人々であり，この人々に福音が与えられない学問であるなら，現場では生かせられないであろうし，不要であるといわれても仕方ない．現実を変えようとする態度は必要であるが，現実がだめだからといって，わずかな数パーセントの人のための学問となっては淋しい．

久保：ソフトウェア工学を現場に役立てるための知恵は得られなかった．落水先生のペーパを引き出したのは，このテーマをかかげたことの成果であろう．このペーパを得たことで満足することにする．

野村（行）：産学間でお互いを批判するだけでは何も解決しない．社会全体の流れを変えるのは難しいが，社会全体（大部分）の流れが何によって流されているのかが問題の一つであろう．また，産学がお互いを理解する努力も必要だ．

たとえば，生産性を２倍にするソフトウェア工学手法を導入

すると，仕事の量（負荷）が倍になる現状では，導入が消極的になるのは白明の理であろう．

濱田： 現場の立場からいえば，入社したてのドクター，マスターに，研究所に行く前に 2 ～ 3 ヶ月，現場で意識はあるが知識のない人たちに対してソフトウェア工学をレクチャーしてもらう－と，何かが変わるかもしれない．

高橋（肇）： 普及の努力が必要であるのは認めるが，やはり現状ではかなりの意識変革をしなければ現場に生かすのは無理であると思う．その体制ができるか？ 現場としては混乱が恐い．

岡本： ソフトウェア工学は今でも現場で役立っていると思います．すべて（ソフトウェア工学の）ではありませんが，現場で生きている事は確かな事実です．ただし，十分に役に立っていると思っている訳ではありません．もし，全然役に立っていないという人がいれば，多分その人はソフトウェア工学の成果を経営学的に，別の見地から評価をしている人ではないでしょうか？ 企業の立場では「生かす」ことよりも「活かす」努力を考える事が重要だと思います．

深瀬： 基本的には，旧来のアプリケーション開発の現場へ直接に生かそうと考えないで，新しい技術革新に伴うアプリケーション分野に主眼を置いて今後のソフトウェア工学を考えることに賛成です．しかしながら，旧来の現場へフィードバックがうまく掛かる "しかけ" も考えておかないと，早い時期での成功は難しいでしょう．

三上： 結局，結論はようわからんかったのですが，双方努力というのには賛成．取り入れるとよくなるというのもわかっていると思うのですが.... 布教には奇跡を起こす必要があるかもしれません．でないと万人には信じられないかもしれません．もしくは強制か？

Seesion 2: ソフトウェア工学は現場に生かせるか？ (Post-Workshop Position Paper)

ソフトウェア工学を生産現場に定着させるための7つの提案

落水 浩一郎 (静岡大学)

ソフトウェアは他の工業製品に比べて，複雑性，非規律性，拡張困難性，不可視性等の厄介な特性を持つ．また，ソフトウェア開発の主な担い手である人間は，上記の問題には特に弱く，そのような人間集団の制御もまた困難である．われわれは，まず，この事実を素直に受けとめ，目標までの距離は遠くとも，日常の継続的な歩みこそが目標到達の原動力であることを認識する必要がある．

ソフトウェア工学の研究者たちは，最初に述べた種々の困難を克服するために，地道な努力を積み重ねてきた．たとえば，プログラム作成過程の可視化と複雑性の制御のために，データ型，データ抽象，抽象データ型，オブジェクト指向の順に，プログラミング原理を改善してきた．また，上流工程におけるソフトウェア動作の可視化をはかるための，種々の図式表現法を工夫してきた．最近，開発過程の可視化と非規律性の制御を目標にして，プロセス·モデルの研究がさかんになりつつある．また，種々のプロトコル解析の結果を基に，集団制御が困難である理由，熟練設計者の行動特性や初心者のつまずきの原因も，次第に明らかにしつつある．

ソフトウェア工学とは，そのような努力の結果生まれる種々の概念や技術の容れ物であり，何を詰めるべきかを考えるための枠組である．

初期の時代の「生産性と信頼性の向上」という枠組は，多くのプログラミング·パラダイム，ツール，プログラミング方法論，設計方法論，ソフトウェア動作の可視化技術，要求定義法，品質管理法，プロジェクト管理法を生み出しつつ，容れ物の容積を拡大してきた．すなわち，役に立ちそうなものは何でも，どの程度の効果があるのかもよく吟味されずに，次々と玉が作られ容れ物に投入された．この時代に生み出されたものがすべて有用なわけではない．しかし，ソフトウェア工学の実践者にとって有用な技術や概念が蓄積されたこともまた事実である．残念ながら，生産現場ではその多くは理解されず，試用されなかった．新しい言語を使いこなすには相応の努力が必要であり，種々の高度で有益なツールを使いこなすには，その背景理論の理解が必須であり，開発方法論を現場に定着させるには試用実験を進んで実施し，カスタム化が必要であることを無視した結果であり魔法の杖のように錯覚され，すぐあきられた結果であろう．

次の時代の枠組は「標準化」である．上記の生産現場の状況に基づいて低水準の標準化のみが強行された結果現在では「新しく光り輝く物」をいれる柔軟性を失ない，しだいに色あせつつある．すなわち，環境の変化に対応できない構造のみが残された．いまこそ，容れ物をつくりなおす必要がある．

枠組が「AI」だけでないことは明らかになってきた．「プロセス·モデル」もまだ役不足である．「???」は一体何であろう．

光り輝く可能性のある玉は以下の7つである：

◆ ソフトウェアの複雑性や拡張性を制御するために

(1) 思考単位と記述単位の対応のよさ，記述量の現象等の効果を持つ抽象データ型やオブジェクト指向の概念をとりいれた言語を使おう．

◆ ソフトウェア開発に規律性を導入するために....

(2) ソフトウェア·データベースとしてオブジェクト·データベースを使おう．

(3) ソフトウェア開発過程を可視化し評価を可能にするために，プロセス·モデルに関する研究をフォロウするとともに，現在の標準開発工程を見直してみよう．

◆ ソフトウェアの不可視性を制御するために....

(4) 上工程での CASE ツールの試用を開始しよう．

◆ 熟練者の経験を再利用し，初心者の抱える問題点を克服してあげるために....

(5) 人間の行動特性についてもっと理解を深めよう．

(6) 人間集団の行動特性をよく把握しよう．

◆ 集団制御の困難さという考え方ではなく，構成員間の風通しをよくするために....

(7) プロジェクト・チーム構成員間のコミュニケーション支援のツールを試用してみよう．

枠組の一つの候補として「作業環境の快適さ」が考えられないだろうか？ 枠組を生成するためのメタ知識は何であろうか？ 私は「現状を革新するためにはよい評価手段を持つ必要があり，革命のためには１０年後を読みきる哲学が必要である．１０年間耐えうる体力が必要であることはいうまでもない」をあげたい．

Seesion 2: ソフトウェア工学は現場に生かせるか？ (Post-Workshop Position Paper)

第 5 回 SEA 環境ワークショップに参加して

新田 稔 (SRA)

1989 年 11 月 29 日から 12 月 1 日まで，釧路で開かれた第 5 回 SEA 環境ワークショップに参加した．私は，ワークショップに「ソフトウェア工学は現場に活かせるか？」についてのポジション・ペーパを提出し，ワーク・ショップでもそのテーマについて発表した．以下は，ワーク・ショップで議論を行なった後，ポジション・ペーパをリライトしたものである．

1. ソフトウェア工学は現場に活かせるか？

ソフトウェア工学を現場に活かすのは，現状ではかなり難しいと思う．それは以下の３つの理由による．

(1) ソフトウェア工学が目指しているものと，ソフトウェア開発の現場が必要としているものがくいちがっていることがある．すなわち，ソフトウェア工学はソフトウェアを正確・精密に作ることを目指しているのに対し，現場が求めているのはソフトウェアを簡単に作る方法である．

あるいはもっと俗な言い方をすれば，ソフトウェア工学の結論は「バカにはプログラムを作らせるな」であるのに，現場は「バカにプログラムを作らせる方法」を探しているとも言える．

(2) ソフトウェア工学は，ソフトウェア開発の「こうであるべき」を追求することを重んじ，現場がいま「どうであるか」観察することを疎かにしていることが多い．現場で有効なのは理想的なソフトウェア開発よりはいまある人やその他のリソースの性質のうまい利用である．

ソフトウェア工学が理路整然と筋道だった方法論を研究しているのに対し，現場では臨機応変なやり方が要求される．システマチックな考え方がそのまま通用するものについては，現場はすでにソフトウェア工学から充分強力な武器を得ている．たとえば，コンパイラ，コンパイラ・コンパイラ，各種のトランスレータ，ジェネレータ，バージョン管理，データベース定義用のツール，エディタ，ワープロ，フォーマッタ,.... などである．しかし，そのような武器だけでは，ソフトウェア開発は戦えないのである．

ソフトウェア工学は，まず，現場で機械的に働く多くの SE 達の仕事ぶりをよく観察し，臨機応変さの少ない作業から機械に置き換えていくことを行なうべきである．いままでのソフトウェア工学には，この観察という作業が不足していたように思える．

人間の臨機応変さの基本は，矛盾・不完全・曖昧さに対する寛容であり，問題にぶつかったときの創意・工夫である．たとえば，はじめから完璧な記述や情報伝達を行なおうとすると，要求定義書や設計書の厚さでご存知の通り，われわれは膨大なデータを扱わなくてはならない．ところが，チョットいい加減でもよいことにすると，扱うべきデータ量はかなり減る．

したがって，次にソフトウェア工学は，どれぐらいのいい加減さを容認するとどれぐらいのデータが節約できるのかや，完全・正確を期すために膨大なデータを記述したとき，それが完成したときにすでにもとになるものが変更されていて修正が必要になるということと，最初はいい加減にしておいて，必要になったとき，必要になった事柄から正確に記述していくことを比較すると，どの程度のいい加減さがもっとも効率的であるのか，などを研究すべきである．

また，ソフトウェア開発現場の悪い性質 (ソフトウェア工学から見た) を利用するという裏ワザ的方法もある．例えば「コボルは将来ずっと使える言語だから機能を拡張しましょうね」と言って機能拡張して処理系を重く，あるいは処理系のインプリメントを難しくしてコボルを葬り去ったり，「設計は開発言語をベッタリ意識する方がパフォーマンスが上がりますよ」と言っておいてから開発言語をいきなりフォーマルな言語に変えて設計作業を改革してしまう，という方法である．

(3) 日本のソフトウェア工学では「in the small」の立場で方法論やツールなどを発案・開発し，それを (企業などに依頼して)「in the large」の立場で実験するというアプローチが多いが，本来やるべきは，「in the large」での発想と「in the small」での実験である．

われわれは，ソフトウェア以外の分野では，いろいろな方法論や道具をきわめて有効に活用しているが，それは，それらの分野では人間の生（なま）の能力がもともと低かったからである．しかしソフトウェア開発では，人間の生の能力がもともとかなり高く，ツールを導入してもその運用のオーバーヘッドだけが目だってしまうことが多くなる．つまり，ツールを使用しても効果が目にみえて上がることは少なく，そのツールを使わないときとの作業効率の差は，ツールを使う人の能力のバラツキに吸収されてしまうのである．

ソフトウェア工学は「ツールはより大きくて複雑なソフトウェア開発に適応するほど効果が大きいい」と言っているが，それは本当だろうか．そのツールを使いこなすためにはある程度以上の能力が必要で，そのような能力を持った人が作業したから，そのソフトウェア開発がうまくいくのではないのだろうか．つまり，「毎朝ジョギングしているから健康である」のではなくて，「健康であるからそんなこと（毎朝走る）ができる」のではないだろうか．

現場に導入すべきツールは，使う人が

(a) 使いはじめる敷居を簡単に越えることができ，
(b) 明確な目的・目標を持つことができ，
(c) 楽しく使えて，そして
(d) 目にみえる成果が得られる

ツールであり，決して「重いコンダーラ」であってはならない．

Seesion 2: ソフトウェア工学は現場に生かせるか？ (Post-Workshop Position Paper)

ソフトウェア工学を現場に生かすために

野村 敏次 (日本電子計算)

1. はじめに

私自身が持つ「ソフトウェア工学」に対するイメージは，ソフトウェア全般に関わる非常に幅広いものであり，かつ実践的なものである．さらにいえば，ソフトウェア工学という学問があって，初めてソフトウェア開発や保守にも進歩が見られると考えている．現実には，ソフトウェアを開発する人間が，裏にあるソフトウェア工学を意識していないに過ぎないのではないか？

先日，ある大学の「ソフトウェア工学概論」という講座の授業内容を拝見した．工学部の情報工学科である．アルゴリズムとデータ構造と記号論理の 3 項目が主要な内容であった．この講義を受けた学生が企業に入り，「ソフトウェア工学は現場に生かせるか？」と聞かれたら，おそらく即座に「No!」と答えるであろう．

ソフトウェア工学の学問としての弱点は，それが体系化されていないことと，実証がきわめて困難であること，および学問として意識されていないことであろう．見方を変えれば，極めて実践的な学問であるが故に自分のしている仕事と「ソフトウェア工学」というものを結び付けて考える人が少ないところに，こうした議論が起こる原因がある．

2. 本質的な理由

ソフトウェア工学が生まれてから 20 年の時間が以上経過しているが，その内容はソフトウェア開発の変化とともに変化し，かつ，その果たすべき役割も変化してきているといってよいであろう．また，現在ソフトウェアに携わる人間は，さまざまである．かれらが最初に教えらるのは言語であり，その言語を使ってプログラムが組めるようになれば，かれらはプログラマと呼ばれ，さらに何年かプログラミングを経験すればシステム・エンジニアと呼ばれ，ソフトウェアのことはすべて理解しているような錯覚に落ちいる．プログラムが組めるということと，ソフトウェア工学を理解し実践するということは異質である．

もちろん，個人的に勉強している方々は沢山おられるのであって，十把ひとからげの議論は慎むべきだとは思うが，ソフトウェア開発や保守に携わる人が，最初に入るのが手段ともいうべき言語であり，以後体系的にソフトウェア工学を学ばないところに，現場で生かされていないというイメージを与えているのではないだろうか？

3. 派生的な理由

それでも，十数年前は，ウォーターフォールモデルが，それなりの威力を発揮し，ソフトウェア開発プロセスの絶対的な決め手とさえ思われた時代もあった．現在，どのような方法論やツールを見ても，多くの人が全面的に賛同するようなものに出会うことはきわめて稀である．その理由は，おそらく以下の派生的な理由によるものと思われる．

(1) 環境の多様化

最近のソフトウェア開発環境は非常にバラェティに富んでおり，開発内容も多種多様である．15 年前までの汎用コンピュータを利用したバッチ形態のシステム開発から，いまやワークステーションや端末を 1 人 1 台で利用する分散ソフトウェア開発が主流となり，データベースやネットワークの利用は当然のこととなったのである．

この開発環境の変化が，それをベースとする開発方法論や技法，さらにはツールに対して大きな影響を与えることは周知の通りである．開発環境の多様化は，また，それらすべてに対応できる方法論・技法・ツールが存在し得ないということを意味する．それは昔の，1 つの開発環境に対応していたソフトウェア工学に較べれば，現場に生かされているイメージは，かなり薄らいだものとならざるを得ないであろう．

見方を変えるなら，ソフトウェア開発環境の多様化は，ソフトウェア工学の多様化をもたらし，共通の学問としての幅を狭めてきているといえるであろう．

(2) 開発者と利用者の背離

開発方法論・技法・ツールの開発者は，往々にして最新技術を駆使した環境の中で仕事をしていることが多い．一方，利用者は，それらの環境から一歩遅れたところで作業しているのが実情ではないだろうか．

ソフトウェア工学の実践者たる技法やツールの開発者と現場でそれらを利用する者との間の環境と発想の相違が，ソフトウェア工学が現場で利用されていないという原因の１つになっているのではないだろうか？

企業内においても，すべてがワークステーションと LAN 環境で構成されている少数の会社を除けば，大型汎用機の TSS 環境とワークステーションの分散環境が併存している．この場合，ソフトウェア工学の実践を真剣に考えているのは，ワークステーションを利用している人が，自分自身の環境上で，如何に実現するかであり，大型汎用機の TSS 環境に，いかにソフトウェア工学を持ち込むかではない．

(3) 概念の先行

概念が先行し実体が伴わないことは多々ある．ソフトウェア工学の世界においても，さまざまな概念やアイディアは出されるが，それだけで終わってしまうことも多い．最近は知識工学との比較や関連において，概念が先行することが多いのではないか．それが，ソフトウェア工学は現実に役に立たないのではないかと考える人に１つの口実を与えている．

(4) 技術移転メカニズムの未確立

大学や研究機関におけるソフトウェア工学の研究は，一般の企業に較べたらかなり進んでいるといってよいであろう．しかしながら，問題はそれらの研究内容が，現場のニーズに合致したとはいわないまでも，比較的容易に応用できる内容であるかどうかということと，それらの研究成果が現場に技術移転されるメカニズムが整っているかである．

おそらく答えは共に「No!」であろう．それは，日本の大学においては，現場のニーズに合わせた研究では，おそらく研究者として評価されないからであろう．また，技術移転に関しては，大学の閉鎖性もさることながら，企業側の態度にも大きな問題があると思われる．

これが，ソフトウェア工学は大学での学問であり，ソフトウェア開発現場とは掛け離れたものであるという印象を多くの人に与えているのである．

4. ソフトウェア工学を現場で生かすために

ソフトウェア工学を現場で生かすために，次の４つの提案をしたい．

(1) ソフトウェア工学教育の必要性

特に企業における新人教育は，コンピュータ概論の後，言語学習に入る．そして，ある１つの言語でプログラミングできるようになった時，かれらは安心し会社も安心する．そして何年かとプログラムを組むことにより，プログラマとしては１人立ちできるようになるものの，「分析は知りません，設計方法は判りません，ソフトウェア開発方法論って何ですか？」という偏った人間が作られて行く．

これは，最初に言語を教えることの弊害ではないかと思う．おそらく，最初にソフトウェア工学の基礎的なことを教育し，ソフトウェア開発プロセスについて一通り理解した後，言語を学習したなら，プログラミング言語は単なる手段であり，自分に（業務に）合った言語を習得すべきであると気付くとともに，ソフトウェア開発に必要な学問としてのソフトウェア工学の重要性も認識されるであろう．

(2) 産学間の技術移転メカニズムの確立

近年，産学共同研究は，ソフトウェアの分野においても行われているが，必ずしも有効に機能していないようである．企業が大学に対して充分な研究費を提供していないから，大学がすばらしい研究成果を挙げ，企業に還元できないのか，大学に余り期待できないから企業が金を出していないのかは不明であるが，余りうまく機能している例を聞かない．産学共同研究を通しての技術移転メカニズムの確立は今後ますます重要となり，推進して行かねばならないが，そのためには企業・大学とも，相互の理解が最初の課題である．

(3) 有能な管理職の必要性

ソフトウェア工学といっても，大きなことばかり考える必要はない．現場にソフトウェア工学（の一部）

を普及させるには，有能な管理職者が１人居れば，環境構築・新しい開発技法やツールの採用といったことは，比較的容易に実行できる．それが体制をも含めた環境の変化をもたらし，真にソフトウェア工学を現場で生かす道に繋がっていく．

(4) 細分化への対応

先に述べたように，ソフトウェア開発環境の多様化・複雑化は，ソフトウェア工学における研究開発分野の細分化を促進している．従って利用者も，分野・目的を絞り，技法・方法論・ツールといったものを考え，要求して行かねばならない．幅広い共通なものを望んではならないのである．

それが，ソフトウェア工学をソフトウェア開発現場に根づかせ，多くの人がソフトウェア工学を身近に感じる鍵となると考える．

5. おわりに

この業界に身を置くことすでに 22 年になる．その間，いろいろな概念，方法論，技法，ツールと出会ってきた．また，さまざまな環境と種々の技術にも出会ってきた．多くの概念は，3 年とか 5 年とかの周期で出てきては消えていった．ソフトウェア工学においても，個々の小さな技術は盛衰があったが，大きな流れは常に悠々と流れているように感じており，ある種の親しみを感じる．

願わくは，「ソフトウェア工学は現場に生かせるか？」といった議論が起こらず，常に現場に生きていて欲しいと願うものである．

Seesion 2: ソフトウェア工学は現場に生かせるか？ (Post-Workshop Position Paper)

あるソフトウェア開発部所の現状

濱田 勉 (NTT 九州)

1. プロローグ

ソフトウェア工学を特段学習したことのないパソコンプログラマーを集めて，とにかくユーザーが望むように動くシステムを短期間につくる部所が，ソフトウェアの内製化というかけ声のもとに作られた．初め，それは，ユーザーに近い人で構成されていたため，ユーザーニーズを的確に反映しうる可能性を期待された．

2. 疑似プロトタイピング？

いくつかのプロジェクトが実用に耐えるシステムをユーザーに提供した．それらの開発プロセスを見ると，結果的にある種のプロトタイピング手法のようであった．結果的にと書いたのは，知識としてプロトタイピングを修得し，それを実践したというより，きちんとした開発仕様書がなく，またユーザー側にもちゃんとしたイメージがなかったので，ユーザーとの会話を通じてプログラマがそれなりにユーザーニーズを感じ取り，とにかくプログラムとして書き上げ，それをユーザーの前で動かし，意見を聞くということを何回か繰り返して，最終システムを作り上げたからである．

これらのプロジェクトは緩い開発計画にも関わらず納期が延びた．なお，緩い開発計画と書いたが，開発見積自体も，信頼に足るものかどうか疑問ではあった....

3. 問題浮上

部所の知名度も低く，少人数で，対ユーザー数も少ない（開発ソフトの提供領域が狭い）うちは，個々のプログラマのパワーに依存して，何とかいくつかのプロジェクトを推進することができた．しかし，組織内である程度の評価を受けるようになり，新人も増え，ユーザー数も増えてくると，開発環境の充実，教育や組織としての標準化，開発技法の活用などが，プロジェクトを成功させる上で無視できない要因となってきた．

4. 開発環境の充実

部所が発足したときのメンバーは，貧弱な開発環境ゆえに，便利な開発ツールや高速なハードウェアに飢えていた．第一歩は，とにかくパソコンを１人１台にふやすことだった．次に，ハードディスクや RAM ディスクを購入した．開発言語を BASIC から C に切り換えた．さらに Unix ワークステーションを導入し，LAN を構成した．マッキントシュも手にいれ，そのマンマシンインタフェースにも触れた．

開発プロジェクトを成功させる１つの要因として，ハードウェアや開発ツールの充実が不可欠であることを初期のメンバーたちは肌で感じていた．だから，ツールの修得にも貪欲であった．ただこの時期に興味をいだいていたツールは，もっぱら，エディタやコンパイラ，デバッガなどの下流工程を対象とするものであり，開発技法そのものは，まだ各個人の無手勝流であった．

5. 人事異動と新人

この部所も人事異動の対象から外れることはなかった．個人のスキルアップのためにも必要な人事ではあったかもしれないが，２年という時間は，開発部所としてはあまりにも短いサイクルである．組織の核となるべく期待された人間に替わって新人が編入されれば，いつまでたっても自立した部所になれない．

最近は新人といっても，常識的に MIFES 等のエディタや C 言語，コードレビューは経験してきているので，早い時期に実務に参加できる．しかし，なまじパソコンの環境だけである程度の開発ができてしまうためか，Unix ワークステーションの敷居の高さか，プログラマーの意識の低さか，なかなかパソコン環境から抜け出そうとしない．

6. 組織としての標準化

少人数で，プロジェクト成功に向けて暗にサインアップし，個人の責任において独自性を出しながら開発を進めていた頃には，特に文書化しなくても，日々の会話の中でなんとなく調整がはかれていた．しかし，人数が増

え，複数のプロジェクトが同時に走るようになると，機器の占有使用に関することやコミュニケーション不足によるモジュール間のインタフェースの解釈違いなどが，クローズアップされてくる．早急にコミュニケーションのミスがないよう，さまざまなポイントで文書化を行う必要があるかもしれない．

7. 開発技法の修得

組織内には，従来から汎用機用のソフト開発のためのカリキュラムはたくさんあったが，相変わらず紙と鉛筆を主なツールにして，トップダウン的手法を教えている．これをあからさまに否定するつもりはないが，世の中で話題になっている手法やツールを敏感にとらえ，実験的にでもよいから大胆にそれらを取り入れたコースを設けてほしいと思う．特にわれわれのような少人数・短期決戦型の開発においては，もっと違う有効な開発手法があると思う．

8. エピローグ

プロジェクトを成功させるための要因を，いろいろと自分なりに考えて見たが，結局，人の要因は無視するにはあまりにも大きすぎると感じる．いいハードウェアや最新のツールがあっても，それを便利だと感じ使おうとしなければ何にもならない．そのツールがある種の開発技法に基づいて使われることを前提としているなら，なおさら，人が開発技法に興味を示さなければ，ツール使用には程遠いものがある．コミュニケーション不足というのも，プロジェクトを成功させるという責任感が希薄なため，納得いくまで議論することがないからではないだろうか？

9. SEA への期待

ソフトウェア開発におけるさまざまな問題に意識を持ち始めたのは，SEA に参加するようになってからである．開発技法についてもいくつか独習した．オブジェクト思考言語にも興味を抱いた．最近，ある種の技法を実践に応用しようと試みている．とにかく同じ問題意識を持つ仲間を増やしたい．SEA あるいは開発ツールを提供する側が触媒となることを期待する．メンバーがそれらに触発され，もっと高次元で自らの開発環境を見つめ直し，改善していこうという気持ちが昂揚することを望む．

Seesion 2: ソフトウェア工学は現場に生かせるか？ (Post-Workshop Position Paper)

ワークショップに参加して

平尾 一浩 (ヒラタ ソフトウェア テクノロジー)

1. 銀の弾丸を求めてはいけないのか？

ソフトウェア工学は本来，開発現場での高信頼性や高生産性を目指して研究が行われてきた．しかし，いままで指摘されて来たように，単体の技法群であり，現場に即した方法なりを明確にして来なかった．また，それを使う側は，ソフトウェア工学を取り巻く環境の受け取り方を間違っている部分がある(たとえば，CASE ツールは本来，ある方法論，技法に従った開発を行う時，その作業を支援するためのツールであるはず．しかし実際はその方法論までが期待されて，そのツールの評価が方法論の評価に置き換えられている場合がある)．

ソフトウェア工学が現場で生かせるかどうかというのは，現場の開発者がソフトウェア工学に何を期待し，何を望んでいるのかをはっきりさせた上で，それが現在のソフトウェア工学と比べ，どの部分が不足し，どこにギャップがあるのかを検討する必要があると思う．その後，現場で生かせるための方法なりを討論するのがよいと思っている．

私は開発の現場でさまざまな問題を抱え，その解決方法をソフトウェア工学に期待している．しかし，ソフトウェア工学の研究者からは，「銀の弾丸を欲しがっている」という少したしなめた表現をされる．

銀の弾丸を欲しがってはいけないのだろうか？　ソフトウェア工学の研究者は何を研究しているのだろうか？　銀の弾丸へ近付くための研究が行なわれていると思っていたが，違うのだろうか．などと考えてしまう．ソフトウェア開発の現場では，品質や生産性などを高めるためにソフト開発の要求分析から保守までを統一的にカバーできる方法論なり技法なりを求めているのはたしかだと思う．その思いとは別の世界で研究が展開されて行くのであれば，もっと別なところでの研究を期待しなくてはならない．もちろん，企業側としても，外に対する期待や理想だけでなく，現実のデータというものを持って，研究側への協力と理解に力をいれて行かなくてはならない．

2. ソフトウェア工学の分類

ソフトウェア工学は，いまのままでは使えないという意見が多い．研究の場と開発の現場があまりにも離れすぎているため，相互の理解不足が始まっているように思う．その意味で，研究としてのソフトウェア工学と使う側のためになるソフトウェア工学とを分けて行く必要がある，という中野先生（阪大）の意見は興味あるものだった．

他にもソフトウェア工学はあまりにも大きくて，何もかもが含まれているため，もっと必要に応じて，その内容を分けて研究していく必要があるという意見も出た．

これらは今後，進んで検討を行わなければならない問題だと思うが，その時には，分類されたソフトウェア工学類間のインターフェースと最終的な結合イメージを明確にして行く必要があるように思う．最終的に開発の現場で必要とされるものは，要求分析からメンテナンスまでをトータル的にサポートする環境であり，管理のための機能も含まれていることが望ましい．そうした環境を実現するためにも，分類にさいしては，分類結果として出てくる要素技術を，最終的にはトータルなシステムに組み入れる仕組みであるとか，お互いの関係を最初から明確にしておくことが必要だと思う．別の見方をすれば，分類されたソフトウェア工学類をまとめ上げる類を作るのがよいのかも知れない．

また，研究のためのソフトウェア工学と，現場で役に立つソフトウェア工学を分けた場合も，同じく，そのインターフェースと導入までを明確にしなければ，現場でのデータと研究の場での方法論とが，また，離れたものになり，現在指摘されているギャップはなくらないだろう．そのためにも双方の関係と理解を早いうちに考えておく必要があると思う．

3. まとめ

いまのソフトウェア工学はその範囲と区切り方が大きく，外部とのインターフェースが明確にされていないために，現場から離れたものになっていたり，認識の差が出たりしている．そのため，それぞれに研究が必要な単位にソフトウェア工学を分類し，その内部インターフェースを確定し，総合的なソフトウェア工学を目指すことと，実際に使われる開発現場とのインターフェースを担当する工学類を存在させ，全体が実際の現場から離れることなく，また実際のデータを使用し，研究に生かせる仕組みを作っておく必要がある．これは，ソフトウェア工学の捉え方を企業も大きく変えなければ実現しない．それにしても，現在の仕組み自体を変えるということはとてつもなく難しいことには違いない．しかし，必要なのだと思う．

Birds of a Feather Session

企業間コンピュータ・ネットワークの構築と活用

(1989 年 11 月 30 日 18:00 − 21:00)

Chair 深瀬 弘恭 (アスキー)

Speaker 参加者全員

討論の狙い

ちかごろ，システム・インテグレーションという言葉をよく聞くようになりました．コンピュータ・メーカをはじめとして，ソフトウェア・ハウスも計算機センタも，みな同じ方向をめざしているようです．ところが，どの会社をとってみても，そのために要求されるすべての技術を自社内にそなえているわけではありません．当然，ゼネコン形式（外注化）あるいは，コンソーシアム形式（企業連合）で仕事を受けることになります．ここで当然必要になってくるのは，企業間ネットワークや広域ネットワーク（いいかえればメトロポリタン・ネットワーク）の確立です．

この BOF では，「そうしたネットワークがもし形成できたら，われわれの生活環境はどのように変化するか？」という話題を酒の肴にしたいと考えています．（ホンネをいえば）そうしたネットワークをうまく使って，人月ベースの発注に対抗する闇のカルテルを組もうということを狙っているのです（!?）．

BOF Session: 企業間ネットワークの構築と活用

目次

Pre-Workshop Position Papers 101
深瀬 弘恭 101
佐原 伸 103

Session Summary 106

Birds of a Feather Seesion : (Pre-Workshop Position Paper)

企業間コンピュータ・ネットワークの構築と活用

深瀬 弘恭 (アスキー)

1. ソフトウェア技術者 (リソース) の不足

このことは皆さんも周知の通り，すべての業種といってもよいほど，各企業で人材の確保には苦慮しているところです．人材の質・量両面での不足を補うため，これまでにもいろいろな努力がなされてきました．効果的な教育システムの構築，分散型開発環境の導入，高信頼性ソフトウェア作成のガイドラインの設定，設計工程でのコンピュータによる支援システムの構築，協調作業支援のメカニズムの議論，等など，多くの分野で　改善がなされようとしています．

しかしながら，これらのアプローチは，個々の企業内での改善努力であり，そこには企業間にまたがる視点を持ったリソース配分の考え方は見られません．

2. SI とパッケージ・ソフトの狭間で

一方，利用者側の企業では，ソフトウェア開発のリソース不足を補う方策として，パッケージ・ソフトの積極的な導入，および，旧来の個別アプリケーションの発注から一括発注方式 (SI) への移行を考えるようになってきました．「何でもやってね，あとはよえろしく」方式と，「これしかできません，利用スタイルを合わせてね」方式の混在です．

開発側では，仮にすべての分野にそれ相当のスキルのあるチームが企業内にあったとしても，あるいは，協力企業の中に適当なチームがあったとしても，程よいタイミングでそのプロジェクトに割り当てられるとは限りません．ほとんどの場合が割り当て不能でしょう．

3. 発注側の論理と受注側の論理

SI 形式による受注側の思惑は，一定枠の予算内で，納期通りのソフトウェア開発の責任を課せられるメリットがあり，従来の工数オーバーによる予算の増額，納期遅れによる損失を，ペナルティーを課すことで明文化することが可能だと考えています．受注側からすれば，1 人月なにがしという人月計算から技術料，コンサルティング料という見積り形式への移行が可能だと考えられます．

しかしながら，実際のところは，まだ，人月工数ベースの積み上げ方式の考え方から，発注側／受注側とも抜け出せていないのが現状です．

4. 企業にまたがる分野，すみ分けを目指して

このようなソフトウェア産業の現状では，ともすれば，それぞれの企業では，内部の開発環境の改善／教育カリキュラムの見通し／新しい開発方法論の導入・改善といった，内へ向かう努力に，かなりのエネルギーを費やすことになります．自社内にはないファンクションを育てるまでには，それだけの時間と費用がかかります．もちろん，その努力は必要ですが，いますぐに欲しい技術が 3 年後には可能になりますといわれても，ビジネスにはなりません．

見方を変えれば，技術力を持ち，その仕事に適当な工数の余裕がある企業との「連合」を考えた方が早道です．自社のアイデンティティー，独自技術に短絡的にこだわるのであれば，企業連合は難しいかもしれません．しかしながら，ある特定の分野においては必ず優位となる部分を各企業持っていると考えられます．そうでなければ，企業として成立しないのですから (たとえそれが Man Power だけの問題だとしても....)．

そこで，この BOF セッションでは，企業の枠を越えて緊密に結合したネットワークを考え，その可能性を議論したいと思います．

現在，junet をはじめ，電子メール・システムの利用がさかんですが，いずれも非営利的な情報伝達を目的にしています．IP 接続が拡がって来たとしても，正面からの営利目的の情報伝達の場は難しいのが現状です．MCI メールのようなサービスも，日本のパソコン通信では見られますが，やはり個人ベースのサービスであり，企業

活動に組み込まれて，利用されるサービスとは異質のものでしょう．近い将来，MHS 規格の電子メール／ファイル転送が可能となる予定はありますが，一般への普及には 5 年位かかるでしょう．われわれはそれまで，やはり FAX と電話だけで仕事を続けるしかないのでしょうか？

5. 企業間ネットワークのメリット／デメリット

秀れたエンジニアは，そうそう簡単には育ちません．しかも，近年は優秀なエンジニアがこの業界へ就職することが少なくなっています．産業としての魅力がないせいもありますが，人材の最適配置を，企業の枠を越えて，そろそろ考えなければならないのではないかと思っています．各企業はそれに応じた対価を支払える体質へ転換すればよいのですから．そのためにも，ネットワークを利用した「価格カルテル」が必要ではないかと考えています．なぜなら，発注側，受注側とも，人月単価の呪縛から逃れられていないのが現状ですから．

通産省は，かつて他の産業分野では，国内産業の育成の視点から，価格カルテルを認める施策を行った経緯があります．人さえ集まれば利益が出る産業構造では，産業基盤が弱いといっても簡単には認められないでしょう．しかしながら，資産も人材も少ないソフトウェア産業は，次の時代での競争力をどのようにして獲得するのでしょうか？

半導体産業は，いま，日本の「産業のコメ」だといわれています．4Mbit DRAM はファウンダリーの間で「疑似価格カルテル」の形で価格が決定されています．ソフトウェア産業はどうでしょう !? いつの日か「産業のコメ」になり得ないのでしょうか？ 肥大した信用経済の枠組みの中で，優秀な人材を実産業へ取り戻すという視点からも，企業間ネットワークを，個人ベースから一歩進めて早く構築する必要があると思われます．

参加者各位のの活発な発言を期待しています．

Birds of a Feather Seesion : (Pre-Workshop Position Paper)

ソフトウェア技術者 = サッカー選手 <> 棋士

佐原 伸 (SRA)

はじめに

ソフトウェア技術者であり、かつサッカー選手でありサッカー=コーチであることから、日頃、日本のソフトウェア技術者とサッカー選手の共通性に興味を持っていました。「なぜ、独創的なソフトウェアが生まれないのか？」、「なぜ、個性を持った選手がいないのか？」という2つの疑問の答えは同じではないのかと。

また、かって囲碁と将棋の選手であったことから、この２つの疑問は囲碁・将棋のプロの場合には全くあてはまらないことに気がつきました。すなわち「囲碁や将棋の独創的な手は次々と生まれている」し「個性を持った棋士は履いて捨てるほどいる」。

そこで、私なりの解答を考えました。皆さんの意見は？

なぜ独創的なソフトウェアは生まれないのか？
なぜ個性を持ったサッカー選手が少ないのか？

日本製の独創的なソフトウェアを思い浮かべることは難しい。米国のそれに比べると格段の差があります。ソフトウェアの基本概念に関する独創的な研究も少ない。欧州のそれに比べると雲泥の差があります。なぜこのような差が起こるのでしょう？

日本で個性を持ったサッカー選手を見つけることは難しい。日本と外国の選手が試合をしていると、日本の選手は誰が誰だか良く分からないことが多い。逆に外国の選手は、遠目に見てもすぐに誰と分かる。　自分のまわりを見まわしてみると、ある程度この原因に思い至ります。一つは教育であり、もう一つは環境です。

なぜ知識は叩き込まれるのか？なぜ階段を登るのか？

日本の教育は、旧日本軍の思想をそのまま受け継いでいます。体育の授業などはほとんど軍事教練です。要約すれば「人と違うことは考えるな」「上の言うことには従え」「生意気なことをするな」「連帯責任を取れ」ということになります。サッカーの例でいえば、「パスはコーナーフラッグに向かって蹴れ、考えなくてもいい(そんな常識はサッカーにない)」「先生やコーチの言うことは黙って聞け(しばしばコーチより子供の方がうまいにもかかわらず)」「トーキックなんて生意気なことをするな、基本をやれ(ゴールに蹴り込むことが基本で蹴り方なんてどーだっていいのに)」「点を取られた責任を皆で取れ、ハーフタイムにグランド１０周(サッカーのルール違反)」ということになります。

野球が元らしいのですが、雨の日に校舎の階段を登り降りして、下半身を鍛えるなどというトレーニングがまかり通っています。あるドイツのコーチ曰く「階段を登り降りしても、階段の登り降りがうまくなるだけである」。

サッカー以外の教育でも似たりよったりでしょう。知識を叩き込まないと大学には受からないそうです。しかし、絶対の真実などありはしないのですから、そうやって叩き込まれた知識は、特定の人々の主観に影響されています。極端にいえば間違った知識を鵜呑みしている可能性が強いわけです。独創的ということは従来の常識を破ることですから、叩き込まれた知識はほとんど役立ちません。

こうやって教育されたら、よほど強烈な個性を持っていないかぎり、大人になるまでに個性が少なくなっていくのも当然かも知れません。

逆に、囲碁や将棋の教育ではこれと反対のことが行なわれています。師匠は弟子に生活面のこと以外は口を出さない。弟子は自分達で上達していくしかない。先輩や自分が打った碁を、皆で検討するのが一番の勉強というわけです。

なぜソフトウェアの開発環境は貧弱なのか？
なぜサッカー場の芝生は禿ているのか？

日本のソフトウェアとサッカーの環境も旧日本軍方式です。一人一人の装備は貧弱で、芝生も貧弱です。そもそもサッカー場には芝生がないのが常識になっています。なぜでしょう？

貧乏だからです。「日本は経済大国である」という説はこの点で間違っています。ソフトウェアの開発環境は世界中と比較したことがありませんが、サッカー場は欧州・南米はもちろんのことアジア・アフリカ・中南米・北米・オセアニアのほとんどの国より貧弱なのです。GNPの比較ではこういう差が表面には出てきません。

もう一つの理由は、年寄が組織のトップにいることです。「わしの若いころは機械語でプログラムした」とか「わしは石ころだらけのグランドでもスライディングした」と言うだけならまだしも、それを今も実行させようとし、それができる地位に頑張っています。隣の国でも、同じような弊害がこの前表面化しましたが。

棋士の場合は、いくらでも碁や将棋を打つ環境が整っています。年寄は後進の指導にあたりますが、協会の理事などは他の分野ほど高齢化していません。

なぜソフトウェア=プロジェクトはうまくいかないのか？
なぜ日本のサッカーは弱いのか？

日本ほどチームプレーが口やかましく強調される国は少ない。しかし、なぜかソフトウェア・プロジェクトはうまくいかないのが普通だし、サッカーはチームプレーが苦手です。なぜでしょう？

まず、日本で言うチームプレーは、一人一人が無責任であることを許しています。例えば、サッカーで相手に追い詰められると盲パスをするという、日本特有のプレーがよくあります。この場合、パスを受けた選手が相手に囲まれて、ボールを取られ怒られるのが常です。パスをしたほうは「チームプレーに忠実である」として褒められることはあれ、滅多に怒られません。外国だったら即座に首でしょう。

もうひとつ「声を出す」ということがチームプレーの象徴になっています。「シュートを外しちゃだめじゃないか」「どうしてボールを取られるんだ」などという構文で良く使われます。失敗した選手は首をうなだれます。そうすると「ドンマイ」と他の選手が声をかけます。この瞬間に、声をかけた選手と失敗した選手はチームプレーを忘れているのです。味方が失敗した瞬間にそのカバーをし、失敗した選手はそれを回復するために、すぐにカバーした選手のスペースを埋めるのがチームプレーなのです。声をかけている暇などはありません。

ソフトウェア=プロジェクトでも同じようなことが起きていませんか？

なぜ根性が要求されるのか？
なぜソフトウェア科学・工学は軽視されるのか？

ソフトウェア開発もサッカーも、体力と根性の勝負と思っている人達がいます。しかし、サッカーでは

- ブレイン
- ボールコントロール
- ボディバランス

がこの順で重要と言われています。ソフトウェア開発で言えば

- 頭脳
- 技術
- 基礎学力

といったところでしょうか。根性などはどこにも入っていません。しかも、実際の試合を見ると、根性があるはずの日本選手の方が根性のないプレーを見せることが多いようです。

MacintoshやNeXTコンピュータの開発者達は、超人的な「根性」で製品を作り出しました。日本のコンピュータはこれらに比べると「根性」が足りないような気がします。どうしてでしょう。

日本では「根性」と言うと、「歯を食いしばって」面白くない練習を耐えることを意味しています。しかし、外国では「笑ってぶっ倒れる」のです。つまり、面白くやっているのでつい度が過ぎるというやつです。日本でも酒場ではよく見かけます。

ところが人間の体は、イヤイヤやる練習より、楽しんでやる練習の方が効果があるという特性を持っています。体育の専門家なら常識になっていることです。そういう訳で、日本選手は実際の試合で「根性がない」ように見えるのです。

にもかかわらず、なぜ指導者・管理者は根性を要求するのでしょう。彼らが専門家でないからです。プロでないからです。自分達のよく分からない技術や科学や工学では、選手や技術者にかなわないので、「根性」を持ち出すわけです。

棋士の場合は、指導者も専門家です。「根性」という言葉もあまり聞かれません。そのそも定石を知らない専門家などいません。

おわりに

実はここでとりあげたサッカーは、日本のスポーツではまだましな方なのです。野球などはもっとひどいものです。野球が世界のレベルで見ればひどいものであることを気づかないのは、「鎖国」のせいです。「情報鎖国」と言ってもよいかも知れません。「日本にいては世界が分からなくなる」と言ってもよいでしょう。ソフトウェアの世界はどうでしょう？

面白いことに、マスコミのサッカー報道とソフトウェアに関する報道が、一番誤りが多いような気がします。

さて最後に問題を一つ。「日本にいては世界が分からなくなる」といったのは誰でしょう？

企業間コンピュータ・ネットワークの構築と活用

セッション・サマリ

深瀬 弘恭
(アスキー)

残念ながら，このセッションはオフレコで行われたので，記録は公開できません．あしからず．

Session 4

ツール・インテグレーションの理想と現実

(1989 年 12 月 1 日 9:30 − 12:30)

Chair 岡本 隆一 (ケーシーエス)

Speaker 青木 淳 (富士ゼロックッス情報システム)
魚田 昌孝 (ケーシーエス)
酒匂 寛 (SRA)

討論の狙い

ソフトウェア開発プロセスを効果的に支援するためには,開発環境の中にあらかじめ多種多様なツールを用意し,時と場合に応じてそれらを有機的に組み合わせて利用する必要があるでしょう. しかし,もともと特定の個別ニーズに対応して別々の場所で開発され,当然のことながら各種のインタフェイスが異なっている複数のツールを,1つの環境としてインテグレーションすることは,技術的にきわめてむずかしい問題を数多く含んでいます. また,開発環境は,それ自体が1つの生きたシステムであり,開発対象ソフトの種類やプロジェクトの規模といった利用条件の変化や,目まぐるしいハード/ソフト技術の進歩に適応して,時間とともに進化して行かなければなりません.

このセッションでは,ツール間インタフェイスの標準化や環境プラットフォームの統一といった技術課題の検討とあわせて,環境の進化や適応に関するマネジメント上のガイドラインの確立を,参加者全員の知恵で考えてみたいと思います.

Session 4: ツール・インテグレーションの理想と現実

目次

Pre-Workshop Position Papers		109
	岡本 隆一	109
	青木 淳	112
	酒匂 寛	113
Session Summary	岡本 隆一	116
討論を聞いて	参加者アンケート	121
Post-Workshop Position Papers		123
	魚田 昌孝	123
	久保 宏志	124
	酒匂 寛	126

Seesion 4: ツール・インテグレーションの理想と現実 (Pre-Workshop Position Paper)

ポジション・ステートメント

岡本 隆一 (ケーシーエス)

1. はじめに

1980 年代の半ば(1986 年頃)から「CASE」という言葉が使われだした．CASE が私自身の考えるソフトウェア・ツール・インテグレーションの理想かと問えば，答えは否です．CASE の目指しているのは，CASE テクノロジの自動化だと認識しています．

CASE は，CASE テクノロジと，CASE テクノロジに基づき統合化されたツール(CASE ツール)で構成されたシステムで，主として，その目標は以下の 2 点だと理解しています．

(1) ソフトウェア開発支援ツールと方法論の統合
(2) システム開発全工程（分析，設計，導入，保守）自動化による生産性の向上

私の考えているツール・インテグレーションの理想と，CASE のそれとの違いは，ツールの統合機能の柔軟性に対する考え方と，生産性向上に対する意識にあると思います．CASE の試みには，自動化を目標とする考えに起因した，一種の冷たさを感じます．CASE を柔軟に解釈すれば，私は，変なことをいっているにすぎませんが，CASE のツール統合では，ツールをパッケージ化する考えが強過ぎるように思います．

その分，人間性に対する配慮が不足していると感じられます．生産性の向上は，もちろん重要な問題です．が，もう少し，物を創造する仕事のために，方法論や技法からもう一歩踏み込んで，開発環境・開発場所の変化や，開発対象のアプリケーション・ドメイン，プロジェクトの特異性，各ソフトウェア開発者の作業の進め方等に合わせて，ツールの分離・再編成を柔軟に繰り返すことのできるような統合機能の実現を理想としています．

2. ツール統合のアプローチ

開発環境におけるソフトウェア・ツールの統合で，一番重要な問題は，ツール相互間の情報交換インタフェースを確保することだと思います．また，一番に難儀な問題は，統合対象のツールそれぞれが用いるユーザ・インタフェースおよびマン・マシン・インタフェースの標準化だと思います．この辺りをいい加減にやってしまうと，統合ではなく継ぎ接ぎになってしまいます．

これまで，自前の環境の中で，ソフトウェア・ツールを開発し統合する場合，基本的なアプローチは，次のように考えて進めてきました（もちろん，教科書的な考えではありますが....):

(1) ソフトウェア開発者の作業の基本動作を見つけだし，そのような個々の作業を支援するためのツールを切り出す．
(2) 次に，切り出したツールを組み合わせ利用する仕組みを作る．
(3) そして最後に，ツールの存在を検索するための仕組みを作る．

まるで，Unix のアイデアそのものみたいな話なので，Unix を使えば，何も大変な苦労を背負わずにすみそうですが，そんな訳にはいきません．大型汎用機に何でもやらせようとすることに無理があるように，Unix でもすべてが OK という訳にはいきません．Unix は，有用なツールを多数備えていますし，ツール統合の有用なプラットホームに使えることは事実ですが，それだけでは飯が食えません．

収入を約束してくれるソフトウェア・ニーズの大部分は，まだまだ大型汎用機の世界にあります．

これから開発するソフトウェア・ツールは，将来の統合を予想して作っていけばよいのですが，いま，一般的に望まれているツール統合は，過去に開発されたツールを組み合わせ利用する場合のほうが多いと思います．これは，あくまで私の主観です．私自身は，もともと大型汎用機の出身なので，いま使っているワークステーション用のツールというより，大型汎用機の環境で，その昔愛情をタップリ注いで書き貯め残したツールのほうが，数も種類も豊富なのです．どちらがよいという問題ではありません．要は，自前のツール統合のアプローチでは，「ある物は使え」という主義を優先しているということです．

3. ツール統合における問題点

前述でも触れていますが，ソフトウェア･ツールの統合にさいしては，以下の問題を解決しなければなりません．

(1) ツール間相互の情報交換インタフェースの確保
- ツールそれぞれが使用するデータ･コード体系
- ツールそれぞれが使用する入出力インタフェース等

(2) 統合するツールそれぞれが用いるユーザ･インタフェースの標準化

(3) ツールそれぞれの操作に関わるマン･マシン･インタフェースの標準化

これらの問題の解決は，統合されるツールのすべてが，ホスト集中型のようなユニークな環境（OS，ハードウェア等）の上に構成されている場合でも難しいのですが，最近普及が進んでいる分散型の開発環境では，さらに，以下のようなことが必要になってきます．

(A) 異なるファイル･システムの操作インタフェース

(B) ハードウェア，ソフトウェアの通信インタフェース

コンピュータ機器の高性能化･低価格化とネットワーク構築技術の目覚ましい進歩で，ハードウェアはかなり柔軟に繋がるようになったと思います．とくに，ワークステーションの高性能化が開発環境の分散化を促進し，1人1台，あるいは1台以上のワークベンチを用意し，作業用途に応じてハードウェアを効率よく使い分けできる環境すら構築できるようになりました．

しかし，そうした環境の根幹の進化に適応して行われたツール統合はあるのでしょうか？　私は知りません．もし，あったとしても，ネットワーク機能や分散環境等を充分に理解して開発されたツールは，まだまだ，そんなに数多くはないと思いますし，大変な苦労の成果ではないかと思います．

4. ケース･スタディ

分散環境の普及によって，ソフトウェア技術者に提供されるツールは，量･種類とも，かなり豊富になったと思います．私は，普段の仕事の道具としてパーソナル･コンピュータを使っています（本当はEWSが欲しい！）が，ネットワークは大好きな分野（使う方か作る方か定かではない）なので，流行にしたがってLAN接続し，仕事の仕儀に合わせて，何台かの大型汎用機やUnixマシンにアクセスしています．

前述の通り，もともとが大型汎用機の出身なので，汎用機上のツールとUnix上のツール，パソコン上のツールを組み合わせて仕事をしています．しかし，単に，ネットワークで接続しているだけの環境では，この操作がなかなか大変でした．

これも前述の通り，そもそも，「ある物は使え」主義でネットワークを使っているので，初めからツールの移植等は考えませんでした．そこで，3年前になりますが，ネットワークに繋いだ各コンピュータ上のツールを，最寄りのEWSから透過型でアクセスできるようなユーザ･インタフェースの開発を考えました．このプランには，CAMELという名前をつけました．

CAMELは，ソフトウェアの開発方式やプロジェクト毎の仕事の進め方等，時と場合に応じて，利用者自身がメニューを組み換えることで，ツールの結合順序の変更やグルーピング，新ツールの追加ができる仕組みで，異機種間通信のインタフェース･ライブラリ，データ変換（フィルタ）ライブラリ，応用プログラム（ソフトウェア･ツール）結合ライブラリ等で構成し，ツールの呼び出しまでのマン･マシン･インタフェースを標準化しようとする1つの統合プランでした．

その時は，単なるプラン「絵に書いた餅」で終わりました．終わってしまった原因は，人手不足，自分の力不足等，情けない話ですが，現在は，自社開発･自社利用（一応）システムのUTOPIA（分散型情報システム）の中に，CAMELが生きています．

CAMEL（「絵に書いた餅」の方）は，統合対象のツールマン･マシン･インタフェースや，データの入出力インタフェースの標準化を最初から放棄した試みです．したがって，ツール統合のアプローチとしては不完全ですが，分散環境に対応し，メインフレームとUnixが，見かけ上1つの架空の環境の中で仲よく使えるところは，評価できると思っています．

5. まとめ

年々，ソフトウェアに携わる人口は増えています．が，いまでも，オペレーティング･システムを含むソフト

ウェアは，ハードウェアの特徴を最大限に引き出すように作るのが普通....，と考えている人が多いようです．この考え方は，ハードウェアのメーカが主導する現状のソフトウェア市場では，普通だと思います．しかし，この風習が変わらない限り，ハードウェアが前を走り，ソフトウェアは，その後を追いかけ続けなければなりません．ツール統合を困難にする本当にやっかいな問題は，プロダクツの差別化と背中合わせになるこの辺りにあるのかも知れません．

そんな状況の中で，唯一の例外といえば Unix でしょう．Unix は，ハードウェアに対する依存度が低く，移植性のよさに魅力があります．したがって，処理能力に関する対ハードウェア効率は，必然的に低いでしょう．Unix が，ビジネスの世界で使われ始めるのに時間がかかったのは，この辺りに起因していると思います．

ハードウェア依存のソフトウェアの世界では，悲しい話題がたくさんあります．同じメーカのハードウェア製品でも機種や OS が異なれば，先に述べたような問題で悩みます．私の仕事は，(年令的な？) 立場上，システム提案や設計作業が中心になります．図・表・絵を多く交えたドキュメントを作成する場合，最近，手書きで仕事することは稀になりました．大抵の場合は，ワープロや DTP ツールを使用します．

使用するツールは，用途や好みに応じて選択しますが，受け手 (顧客) が，必ずしもその成果物に満足してくれるとは限りません．一生懸命作った名作 (迷作？) を，受け手の要望に応じて，既成のフィルター (データ・コンバータ) で処理した時に，同じメーカの製品同士でありながら，悲しい結果を招いて泣いた人は，数多くいらっしゃると思います．もちろん，異なるメーカ製品の間での悲劇はあらためて語るまでもないでしょう．

毎日がドキュメンテーションで疲れている人も，沢山いらっしゃると思います．ドキュメンテーション・ツールが取り扱うプリミティブは，少なくとも，CG の分野以上に標準化を図る必要があると思いませんか？ この辺りが標準化されると，ソフトウェア開発の上流工程支援ツールと下流工程支援ツールの統合に，かなり，期待が持てそうに思われます．

それから，新たにツールを開発する時，少なくとも，情報交換インタフェース (特にデータ・コード体系) を特殊化して，他にある類似機能ツールと差別化するのをやめるだけでも，ツール統合はかなり容易になると思います．通信の問題等，他にもいろいろありますが．後者の問題については，特にメーカ側の立場の方に考えていただきたいと思います．

Seesion 4: ツール・インテグレーションの理想と現実 (Pre-Workshop Position Paper)

ツール・インテグレーションについて

青木 淳 (富士ゼロックス情報システム)

ツール・インテグレーションを行なうためには，インテグレーションしようと思うすべてのツールが，1つの言語を内包していなければならないと思います．よくできたアプリケーションは，必ずといってよいほど，言語プロセッシング能力を保持しています．Smalltalk-80 や HyperCard などがよい例です．

Smalltalk-80 は，それ自体が言語であり，開発環境であり，そこで作成されたツールのインテグレーションの場になっています．HyperCard も，Programmming-by-Example を指向するスタックウェア構築環境ですが，その内部には，HyperTalk という言語を処理する機能があります．昨年のソフトウェア・シンポジウムで，SRA の酒匂さんが発表されていた「X ウィンドウの統合化ツール」も，たしか内部に言語処理系を内蔵していたと記憶しています．私も最近，Smalltalk-80 の上に Grapher Gear というものを開発しましたが，これもリスト言語を処理する能力の上に構築されています．

また最近，プレゼンテーション・スプレッドシートなどのキッチフレーズで，WINGZ や V-Cell が販売されていますが，その内部にはスクリプトやマクロなどが存在し，それを処理して複雑なプレゼンテーション動作を行います．もともと，スプレッドシートをプログラマブルにするために，必ずといってよいほど，マクロ機能が存在していたのですが，WINGZ や V-Cell などは，この機能を高級にしたとものとして考えることができるでしょう．

そして，巷に多く存在する「インタフェース構築ツール」や「統合化ツール」も同様で，そのよし悪しを見分けるには，その内部記述言語の処理能力を比べればすんでしまいます．もっと身近かな例でいえば，Unix のシェルや Emacs なども同様です．Unix のシェルは，シェル・スクリプトを処理するインタプリタですし，高機能なエディタである Emacs は，私が開発した Grapher Gear と同様に，リスト言語を処理する能力を持っています．

以上のように，ツール・インテグレーションを指向する「ツール」や「環境」そして「アプリケーション」は，必ずといってよいくらい，「言語処理能力」を保持しています．さて，「問題は，その言語である」ことはいうまでもありません．「ツール・インテグレーション向けの言語があるのだろうか」ということが「ツール・インテグレーションは可能なのだろうか」という問いかけと等しいと思います．

1. 言語仕様が均質で一貫性があり，学びやすいこと
2. 言語を定義する諸概念が直感的で，インタフェースも分かりやすいこと
3. プログラミングは既存のソフトウェアを改良するという形で進むこと (Programmming by Refinement)
4. 複雑なものは，全体－部分の関係で表現できること (Programmming by Composition)
5. さらにインテグレーションするために，抽象化が行なえること (Programmming by Abstraction)
6. 再使用および部品化をするために，要素分解が容易なこと (Programmming by Factorization)
7. 開発用のツールやサポート機能が豊富であること (Programmming by Example and Tool)

ここに揚げた 1 から 7 までの項目を満足する言語および環境がツール・インテグレーション向けでしょう．私は，そうした「ツール・インテグレーション環境」が喉から手が出るほど欲しくてたまりません．この環境に最も近いのが Smalltalk-80 なので，現在はそれを使用しています．そして，「いつか，何時か....」と思ってます．

ツール・インテグレーションを志す人は，この世界を「物理記号系」と見なせる能力に長けていなければならないでしょう．そして，少なくとも 1 つぐらいの言語処理系を作成した経験または能力がなければならないと思います．

ツール・インテグレーションの根底には，「データ統合」，「コントロール統合」，「ビジュアル統合」があります．さまざまなツールのインタフェースを統一して結合するためには，この 3 つの統合化を避けて通ることはできません．現在，これを解決するのは，「オブジェクト指向」しかありません．言語もオブジェクト指向の考え方を取り入れ，選択するのに困るほど，色々な言語を生みだしています．また，実体関係図や意味データモデルに基盤を置く，データベース屋さんも，オブジェクト指向の枠組みを取り入れ，オブジェクト指向データベースの実現を目指しています．そして，ソフトウェア工学の方々も，構造化手法から，オブジェクト指向の方法論に移行し，オブジェクト指向の設計法や分析法を提唱しています．今，オブジェクト指向 CASE ツールの開発が望まれています．

ツール・インテグレーションは，「オブジェクト指向」に基盤を置いた「データ統合」と「コントロール統合」そして「ビジュアル統合」によってなされるというのが，私の結論です．

Seesion 4: ツール・インテグレーションの理想と現実 (Pre-Workshop Position Paper)

ポジション・ペーパー

酒匂 寛 (SRA)

1. 何が問題か

少し前の時代を振り返ってみよう．まだ LAN もなく，MS-DOS, UNIX などというある程度の共通基盤もなく，それどころか CPU もメーカーごとにばらばらだった時代があった．それほど古い話しではない．そのころ，ツールの統合といえば，単に別のツールが書き出したデータを読み出せるだけでも十分であった．もちろん処理はバッチなので，並列処理に頭を悩ますことはなかった．ネットワークを介して，アーキテクチャの違うマシン同士をつないで，バイトオーダーが違うなどと騒がなくともよかったし，出力はキャラクタ指向だったので，ユーザーインターフェイスが，天と地ほど違うこともなかった．パンドラの箱が開けられる前には，希望から生じる苦しみが存在しなかったように，ツール統合に関して計算機に過大な期待をして苦しむことは少なかったのである．

ツールの統合の問題が重要になったのは，ウィンドウ・システムが普及し，LAN が日常のものとなり，複数のツールが同時に操作できるようになってからである．最低限のデータ交換手段は確保された，ユーザ・インタフェイスを構築しやすくする道具立てもできた，それらを用いて同時に複数のツールを利用するための仕掛けも存在する．それなのに，こちらのパソコン，あちらのワークステーション，壁の向こうのメインフレーム上に，データは散在し，ツールもそのマシンの上で利用するしかないのである．ご馳走を目の前にして，お預けを食わされた犬のような心境である．

ここで，苦しみの元凶を考えてみるのも一興だろう．大きく分けて，以下の 4 種に分類できよう．なお順番と，重要度は関係していない．

(a) 表現の不一致： 同じような内容を表現しても，形式が違う．細かくは色や書体，図形のプリミティブの違い．やや抽象的なレベルでは，操作対象の選択方法の違い(直接 vs 間接，マウス vs キイボード)，目で見えるものと実際に記録されているもののずれ(更新のタイミング)，同じ名称なのに，違う振舞いをするコマンド(QUIT コマンドは何をするものか？) 等が利用者を惑わせる．行なわれる操作の意味は同じようなものであるのに，作られた場所が違うために，それぞれのポリシーが衝突して，一緒に使うことが困難になっている．

(b) 意味の不一致： 各々のツールは，それら固有のデータ形式と，意味を持っている．他のツールからそれを利用するためには，形式と意味を解釈して，変換を行なわなければならない．ところが，N 個のツールが相互にデータを交換できるようにするためには，単純に考えると N*(N-1) 個の変換ツールを用意しなければならない．これは現実的には不可能な話しで，簡単な形式のファイルに対して，簡易言語を用いてフィルタを作るか，重要な変換だけを選び，本格的なプログラム言語を用いて変換ツールを作るぐらいの対応が精々である．

(c) 環境の不一致： ハードウエアの制約に依存するシステムは，それだけでツール統合の敵である．たとえば，画面の解像度，同時に発色できる色の数，マウスのボタンの数，ファンクションキーの存在，特定のプリンタ等に強く依存するツールは孤立無援に陥りやすい．ソフトウエアの制約でも同じような問題は生じる，つまらない例だが，ファイル名の長さの制限が OS ごとに違ったり，あるプログラム言語の外部変数名は，最初の６文字で区別しなければならなかったりといった事実は，データの移送，プログラムの移植に対して大きな障害となるのである．

ネットワークがあってもファイル転送ができるだけでは個々の機械がバラバラに存在していた時代とあまり変わらない．それぞれのもつリソースをどこからでも，同じように扱えることが必要なのである．

(d) 行動の不一致： ツールのコントロールも問題である．大きなツールを複数同時に使っている場合を考えよう．それぞれのツールが持つ部分的な機能を，交互に呼び出したい場合があったとしても，その同期をとったり，動作を細かく制御するのは容易ではない．ただまとめて呼び出すだけなら，現在でもよく行なわれていることである．違うマシンの上で実行されている，別々のツールを，きめ細かに制御しながら，同じ画面上で見ることは，一般に難しい．そもそも，ツールの中の個々の機能が，普通よく独立しておらず，ユーザーから直接ではなく，別のツールからそれを制御すること自体が難しいのだが．

2. 何がなされているか

現在存在している，ツール統合を意識した，あるいはそれに寄与すると思われるものをいくつか選んでみよう．

(a) 標準ウィンドウシステム： 最近の流行は，たとえば X-Window と Presentation Manager である．ユーザーインターフェイスに対して，統一的な枠組を与えて，表現の不一致を克服しようとするものである．X-Window は，ネットワークを介して，さまざまな表示用ハードウエアにアクセスできるので，環境の不一致もある程度解消される．後者はポリシーを強制し過ぎるきらいもある．

(b) 分散ファイルシステム： 別のマシンの上にあるデータを，自分のワークステーション上にあるデータと同様にアクセスできるようにする仕掛け．これも環境の不一致をある程度軽減する．

(c) 専用言語マシン： 世の中がすべて単一の言語，データ形式でできていることを前提に作られたマシン．UNIX は C で書かれているが，データと C 言語の間には直接的な繋がりはないので除外．Lisp Machine や Smalltalk Machine，Prolog Machine などが代表的なものである．ユーザーインターフェイス，個々のツール(？)，データまでもが単一の形式で記述され，また傾向としてハードウエアは高度に抽象化されているため，前の章であげた問題点は比較的克服し易い．データの変換も容易である場合が多い．

部分的な機能の呼びだしは得意とするところで，それらの同期／管理機構も用意されている．ただし別種のマシンが入ってくると柔軟さは失われる．

(d) 統合環境： 既存の OS の上で，ツールを統合しようとする試みも多数ある．たとえば PCTE，SDA などはその例である．この場合もその環境が提供するプロトコルに従えば統合は容易だが，任意のツールを自由に使えるわけではない．

3. ツール統合のためのチェック・リスト

ここで，独断的にツール統合のためのチェックリストを掲げてみよう．ここでは単に望ましい性質を挙げているだけで，現実のツールがこのリストに当てはまらないことは明らかだが．

(a) 表現の一致：

(1) ユーザーインターフェイスと，ツールの機能部分の分離(分離していないツールが多い現状では，なるべく共通のウィンドウシステムを用いるしかない)ポリシーを自由に決められることが大事である．その意味で多くの標準ウインドウシステム構築の努力は正解だが，ポリシーを定めるためのポリシーが曖昧で，システム構築者は苦労している．

(2) 特にメッセージを自由に書き換えられる機能は重要（たとえば各国語対応のため）

(b) 意味の一致：

(1) 意味を含んだ標準データ形式が存在

(2) そのデータに対する入出力機能

(3) 拡張可能なデータスキーマ

(4) ネットワーク全体にまたがるデータ管理機構（単なるファイルシステム以上のものが要求される．必要ならデータの変換等も自動的に行なう）

(c) 環境の一致：

(1) データの透過性（位置を意識しなくともよい）

(2) ハードウエアの透過性（大きな画面で見たいときはそれを選べる．テープ駆動装置を，ネットワークのどこからでも同じ操作で使える，等）

(3) 精度からの独立（解像度，色など）

(4) 機械のアーキテクチャに依存した，内部データ表現（バイトオーダ等）が，利用者から隠されている（X-Window, Sun RPC の XDR 等）

(5) ここで述べるのは適当でないかも知れないが，統合を考えるさいスケーラビリティも重要な因子である

(d) 行動の一致：

(1) ツールの部分機能の呼びだしが可能

(2) 実行中にリアルタイムで，データ交換が可能

4. 現実的解決は？

理想的なことはいくらでも考えられるが，現実に存在するツールを捨ててしまうわけにはいかない．現実的な当面の解決策は，上で挙げてきた条件を踏まえつつ，既存のツールを別のエージェントで覆い隠してしまうことであろう．

たとえば個々のツールが出力した結果は，エージェントによって変換され，共通のデータ管理機構に格納される．直接的なユーザーインターフェイスは，変更することは非常に難しい．仮定しているウィンドウシステムが違う時はほぼ不可能である．ソースコードがあれば，共通ライブラリを使うことも考えられるが，現実のツールがバイナリで配られる場合が多いことを考えると難しい．

LAN + Window 機能を持つ UNIX ベースのマシンを中心にしたツール統合が現時点では，もっとも行ない易いものの一つであろう．

筆者の好みをいえば，超インテリジェントターミナルの上に，複数のマシン上のエージェントからのウインドウが開かれている状態がよいと思っている．

いずれにしろ，使い易いネットワークデータ管理機構（バージョン管理／コンフィギュレーション管理を含む）と，既存のツールをうまく覆い隠してくれるエージェント管理機構があれば，現在の混沌も少しは収まるのではないかと思っている．

ツール・インテグレーションの理想と現実

セッション・サマリ

岡本 隆一
（ケーシーエス）

1. はじめに

ツール・インテグレーションは，すでに使い古された言葉であり，ソフトウェア開発効率を改善するアプローチとして決して新しいものではない．が，実際の開発環境で，それを実践しようとしたとき，技術的な困難やマネジメントの難しさなど，さまざまな問題が出てきて，なかなかうまく行かないことがわかる．この点に着目して，ツール・インテグレーションを今回のワークショップのセッション・テーマに採り上げた．

「ツール・インテグレーションの理想と現実」と題し，ツール間インタフェースの標準化やツール統合に適応するプラットホームの統一等，技術面における検討と，環境の進化や適応に関するマネジメント上のガイドラインの確立について，参加者全員で考えてみようという試みである．

このセッションでは，全体討論を行う前に，青木淳さん（富士ゼロックス情報システム），魚田昌孝さん（KCS），酒匂寛さん（SRA）の 3 氏に話題の提供をお願いした．それぞれの方々からは，個々に提案・報告が行われると思うが，ここで方々の発表内容をまとめてみる．

2. 話題提供

2.1 基本姿勢，オブジェクト指向，Smalltalk-80

話題提供の１番手は青木さんにお願いした．最初に，御自身の「ツール・インテグレーションに対する基本姿勢」の説明があり，そして，ソフトウェア開発におけるオブジェクト指向の有効性・可能性とプラットホームとしての Smalltalk-80 の真価についてお話いただいた．

(1) ツール・インテグレーションに対する基本姿勢

-1- インテグレーションしようと思うすべてのツールが１つの言語を内包していなければならない．
-2- ツール・インテグレーションのためのＪａｒｇｏｎ．
-3- この世の中を「物理記号系」とみなせる能力が必要．

(2) よくできたソフトウェアは必ず言語処理能力を持っている

システム［言語］
Smalltalk-80 [Smalltalk-80}
HyperCard [HyperTalk]
WingZ [HyperScript]
V-Cell [Macro]
GrapherWorks [S-Expression]
Unix-Shell [ShellScript]
Emacs [S-Expression]

(3) ツール・インテグレーションのための言語

-1- 言語仕様が均質で一貫性があり，学びやすいこと．
Object Oriented Programming.
-2- 言語を定義する諸概念が直観的で，インタフェースも分かりやすいこと．
Object Oriented Programming.
-3- プログラミングは既存のソフトウェアを改良するという形で進むこと．
Programming by Refinement.
-4- 複雑なものは，全体・部分の関係で表現できること．
Programming by Composition.
-5- さらにインテグレーションするために，抽象化が行えること．

Programming by Abstraction.

-6- 再使用および部品化をするために，要素分解が容易なこと．

Programming by Factorization.

-7- 開発のツールやサポート機能が豊富であること．

Programming by Example and Tool.

(3) オブジェクト指向の有効性・可能性

オブジェクト指向は：

- ○ ソフトウェアの再利用性
- ○ モジュール性
- ○ コンパクト性
- ○ メンテナンス性
- ○ チーム開発性
- ○ バージョンと履歴の管理

に対する有効な処方箋である．

(5) Smalltalk-80 の真価

- ○ 数少ないソフトウェア中心の文化
- ○ ソフトウェア技術者の知識増幅器
- ○ Frameworker のためのシステムである
- ○ Customizer のシステムではない
- ○ 試行錯誤を重ねる人の道具である
- ○ 設計を行う人のワーク・ベンチである

(6) ヒューマン・インテグレーションの必要

「ツール・インテグレーションを行う以前にヒューマン・インテグレーションが必要」

［質疑応答］

「クラスの抽象化：フレームワーカーのためのシステム」の見方に対して参加者から次のような意見が加えられた．（青木さんの発表資料に）提示されたクラス階層は，システム・プログラマの目から見たクラス階層であり，もっとアプリケーション寄りのシステム・アナリストやシステム・デザイナから見た場合，もっと違った構造になってくるか，あるいは違った構造に見えるかも知れない．

その他の質疑応答は Smalltalk-80 に集中した．９０年代の開発環境として，機能向上・性能改善の著しい Smalltalk-80 への期待が高まる．

2.2 新しいソフトウェア・オフィスの構築

話題提供の２番手は魚田さんにお願いした．魚田さんの発表では「オフィス環境の理想を求めて」と題し，種々雑多な環境を統合する試みとして，KCS のアプローチが紹介された．

(1) 発想

「ソフトウェア開発環境の理想を求める」発想から，種々雑多な環境（開発環境）をネットワークで統合化する．現場の技術者の仕事は，プログラムや設計書だけを作ることではない．まず，開発過程で作り出される成果物を共有する仕組みを作り，色々なソフトウェア開発上の問題点を見えるようにする．

(2) ツールとツール統合について

-1- すでに色々なツールがはびこっている．新しいもの（ツール）を作ればよいが，そうではなくて，これまで作ったものをうまく使って何とかしたいという要望がある．

-2- 個々のツールの入出力インタフェースを揃えていかなければならない．

-3- ツール自体が統合できるようになっていなければならない．

-4- ツール自体をその様に作っていかなければならない．

(3) KCS のアプローチ

UTOPIA システム： まず，オフィス環境のインフラの整備．

-1- ワークベンチにパソコンを使ってアプローチ．

-2- メインフレーム＋ワークステーション＋パソコンで全体を構成．

［質疑応答］

(1) なぜ，ワークステーションを使わないでパソコンなのか？

答え： ワークステーションを使えば，より一層，容易に解決できることは分かっているが，フィールド個々の事情もある．コストの問題等も含めて，今は第一段階（過渡期）と考えている．

(2) 以前 JStar を使って，ドキュメンテーションの観点から開発環境の改善にアプローチしていたと思うが，その後どうなっているのか？

答え： 現場の技術者の仕事は，プログラムや設計書を作ることだけではないし，ドキュメンテーションだけでは問題の解決が難しい．それ以上に，ネットワークによる統合で成果物の共有を進める際に，JStar の場合，中身（ファイル）ではなく，ゼロックスがリザーブしている非公開のプロトコル（通信）に問題があり，思うように環境が作れない現状になっている．

(3) 会場からの指摘： KCS では，どこにでもあるようなツールを自社開発しているようですが，どこにでもあるようなものは，探して入れる（購入する）べきで，その方がコストも安くつく．

2.3 ツール・インテグレーションをするときにどんなチェックをすればよいか？

話題提供の３番手は酒匂さんにお願いした．酒匂さんの発表では「ツール・インテグレーションをするときにどんなチェックをすればよいか」についてお話しいただいた．

(1) ツール・インテグレーションに関する「諸悪の源」として

-1- 表現の不一致：色，書体，図形プリミティブ，直接 VS 間接，ポリシーの衝突．

-2- 意味の不一致：変換ツールの必要性，含意の違い，プロトコルの不一致（不明）．

-3- 環境の不一致：精度，入力デバイスの差，外部名の長さ．

-4- 行動の不一致 ： 並列，同期，部分機能の呼び出し．

(2) ツール・インテグレーションに関するチェックリスト

-1- 表現の不一致：（またか!) UI と機能の分離，ポリシーを定めるためのポリシー (?)，外国語対応....

-2- 意味の不一致： 意味を含んだ標準データ形式，拡張可能なスキーマ，ネットワーク全体にまたがるデータ管理機構（VC/CC は必須)....

-3- 環境の不一致： データの位置自由性(データ透過性)，ハードウェアの透過性(精度からの独立)，マシン・アーキテクチャからの独立（バイトオーダ等)，スケーラビリティ！....

-4- 行動の不一致： 部分機能の呼び出し，スマートな並列処理....

(3) 既存のツールをどう取り込むか？

エージェントでカプセル化： 一番賢い方法は使わないことですが，現実はそうはいかない．

(4) ツール統合のためのプラットホームについて

積極的に使う気，使いこなす気にさせるには，環境に麻薬的・刺激的な要素が必要である．また，トライ＆エラーが気軽にできること，簡単に変更できることでも目に見える効果があがる（深みにはまる第一歩)．それと，抜けがあってもそれなりに動くこと（やれるところまでやる)．人の環境を盗みやすいこと．

［質疑応答］

次のような意見が交換された．

(1) 環境には使う側の要求を満たしてくれる「作れる余地」を残し，進化していくためのトリックを設けなければならない．

(2) 病気がちの人が長生きするように，何でもインテグレーションしようとするような無理をしないこともツール統合を進めていく上で重要な姿勢．

(3) UNIX は，これから先，サーバとして長生きする．Smalltalk-80 は，先のチェックリストをほとんどカバーしている．

3. 全体討議

以下に，全体討議での意見・提案等を発生順に紹介する．

3.1 それぞれの話題から

(1) Smalltalk-80 は，現在選べる中で最善の選択だと思えるが，完全な選択ではないと思う．

(2) ツールは，１本１本，地道に作ってもじきに全部捨てなきゃいけなくなるので，現実的なアプローチは，なるべく手を抜いてやってよいのではないか？　そして，来るべき日に備える動きをしていくべきだ．

(3) アスキーでは，パソコンはターゲット．テストだけできればよい（プラットホーム）として×は×．

3.2 提案に向けて

☆「結局，どうしようというのか」: キーワード．

ドラスティック派の見方として，MML と同様，メインフレームを捨てる考えがある．穏健派の表現も含めると....，

○ 道具立て

- Well defined なツールを....，
  - 作るように努力する　！
  - 作らせるように努力する　！
  - （標準ツール・インタフェース，OBIF，PCTE＋，UIMS，ハイレベル UIF）
- Ill-defined なツールは....
  - 作らない！
  - 買わない！
  - 使わない！（３ない運動....!?）

○ 組み合わせの知恵....，

(1) 基準を設ける

すべてのツールをインテグレーションする必要はない．そして，無経験・無制限にインテグレートしてはいけない．ある基準を設けてインテグレートする．「このツールは，是非，新しい環境にインテグレートしたい」そのようなツールを新しい現実にパスするように作り直す．

(2) 基準を設ける：　チェックリスト方式

「ある基準」として，チェックリスト方式では邪魔臭いから誰もやらない．

(3) インテグレーションの標準

ヨーロッパでやっている PCTE のようなやり方がある．そうしたスタンダードを作る．インテグレートできるような仕掛けを作る．「ツール・インテグレーションの理想と現実」を議論していて，PCTE の話がでないのはおかしい．ヨーロッパではあれだけ努力している．

(4) PCTE

言語にディペンドした PCTE のやり方はちょっとまずい感じがする．オブジェクトベースを一番下に，物のレベルとツールの間にもう一層設けないといけない．

(5) 差別化の問題

インテグレーションのスタンダードを押し進めるとツール（商品）の差別化ができなくなる．自由競争の考えに反するのではないか．

3.3 ひとつの提案

(1) SEA

理想に向かって（われわれは）何をしなくてはいけないか？　いまあるものを使わない．いまのツールが OODB とインタフェースするように変えればよい．使いたいというツールは，そうしていくように仕向ければ，あるいは，すればよい．SEA がやらなきゃいけない．

(2) SEA

SEA は，やる必要があるという指摘にとどめるべきか．しかし，やばいと思った人がやらなきゃいけない．X-Window コンソーシアムの例もあるし，やるしかない．

(3) どこから始めるか，どのように始めるか？

全部は一度にできないからドメインを決める必要がある．どのようなツールから扱って行くか....！？　分科会のようなスタイルでスタートさせればよい．が，ペーパとしての提案だけではアピールがない．プロトタイピング，インプリメンテーションの両方が必要になってくる．

4. おわりに

ツール・インテグレーションをテーマとした当セッションは，暖かい（？）冬の釧路の朝日とともに最終プログラムとして設けられ，こころもとないチェアマンとしては，心身リラックスの薬も使えず，ますます不安を抱いてのプログラムでした．

議論は，今を（今あるものを）どうするか....，ではなくて「これからにどう備えるか」に流れました．セッションの中身は，知識・経験豊かなスピーカの３氏と参加者の方々のおかげで，ツール統合に纏わる問題に対して，話題性豊かな情報が順次関わるセッションになりました．ＳＥＡがツール・インテグレーションの問題にどのように取り組んでいくべきか....，解決行動の糸口がでてきました．

結果的に，チェアマンが，自身の勉強不足と力足らずから，まとまりのつかないままに終わらせてしまったのが口惜しい限りですが（反省：ごめんなさい），ワークショップ参加者の皆様のおかげで「ＳＥＡがやらなきゃいけない」等，何とか次に繋がる討論となりました．参加者の方々に，深く感謝いたします．

ツール・インテグレーションの理想と現実

セッションの討論を聞いて

熊谷: ツール・インテグレーションは，現実的には必要な技術であるが，実現不可能に近いことも事実である．したがって，提案できることは次の２点である：

(1) 現存するツールで統合化の可能性を持っているもの（ソフト／ハードの両方）をリストアップする．
(2) これから作られるであろうツールに対するインタフェイスを提案する（篠田氏あたりがやればよい）．そのさい，ユーザ・インタフェイスと機能とを区別すべきか？

そうしたことの経験と実現のノウハウを将来のワークショップで集めることあたりからスタートするのかな？

深瀬: 考えうるアクション・アイテムは次の通り：

(1) いまやるべきこと： ダメな環境と見込みのありそうな環境とをハッキリ区別して，ダメなものは早めに捨てる．見込みのありそうなものについては，そこそこ統合化へ向かっての努力をしつつ，近未来の技術的変化に対応できるように，リソースを確保する．
(2) 近いうちにやること： OODB とユーザ・インタフェイスとの間に位置するツール群のためのインタフェイスの標準化作業．あるドメインの具体的なツールについて，他の同様のプロジェクトの状況を眺めつつ，SEA で研究会をスタートさせる．

濱田: 既存のツール間のインタフェイスを無理やり合わせる努力よりも，インタフェイスの標準ガイドラインを設定して，それに合わせたツールを作ることのほうが，たぶん望ましいと思う．

野村（行）: 異なった文化から生まれてきた複数のツールを結合する技術が問題になっているのだが，しかし，本当の問題点は文化が異ることである．それぞれの文化を気に入っている人間たちを改宗させるのはナンセンスだと思う．異文化をむりやり統合するよりも，単純にそれらの間のコミュニケーション手段を確立することに力を注げば，当面はそれだけでよいのではないか？

高橋（晃）: Smalltalk-80 がツール・イングレーションの１つの姿であるというのは，ごく当り前の話であるが，現実的にはやはりインパクトがある．ドラスティックに変わるであろう将来に対して，若い人をあまり手厚く扱わない方がいい，というのは同感です．

三上: ツール・インテグレーションに関しては，当座の問題と新しい環境でという問題が混在している．「現状のトラブルを切り捨てて簡単に新しい環境で」というのも，難しい場合があると思う．現場の立場からすると，「いまできること」というのが一番求められているのではないでしょうか？ ただ，新しいこともどんどん考えないといけないわけで，その辺りの今できることと，新しいことの戦術なんてのも考えないといけないだろう．

野村（敏）: 青木さんの話が非常にわかりやすかった．いままで知らなかった Smalltalk に対する興味が湧いたのは収穫であった．言語によるインテグレーションは難しいかな？ 実際問題としてはインテグレーションすべき有用なツールがたくさん欲しいと思う．

平尾: ツール結合の問題は技術か？ それとも文化か？ ツール・インタフェイス標準化問題が大きくクローズアップされた．Smalltalk-80 はいいシステムだが，そうした特定の言語に偏りすぎるのは危険．むしろ，インタフェイスの標準化あたりから始めるのがよいと思う．篠田さんが提示したレイヤは，既存ツール，これからのツールを含めて考えていく必要があると思った．

高橋（肇）: ツールにプログラミング環境が必要なのでは，という青木さんのお話は，なるほどと感じた．これからは，マシン環境に頼らないでものを作っていくことが必要であろう．とはいえ，ついつい土台に左右されてしまうソフト環境の悲しさ．

加藤: ユーザ・インタフェイス，データ互換など，いろいろな部分での統合化の課題があげられたが，できるところから１つ１つ実現して行く必要があると思う．そうすることにより，既存のツールがこちら（統合化された環境）へ歩みよってくることが可能になるだろう．そのためにも標準化が必要か？

新田: 新しいそして有用なツールを発案し開発する人たちは，ふつう個性が強い場合が多く，ポリシーの衝突は避けられない．水平な切り分け（Presentation と OODB）はできても，垂直方向の切り分け（ツールの守備範囲）はむずかしいだろう．

中野: 青木さんへ，Smalltalk-80 をユーザとして本格的にに使い始めたところです．その意味でたいへん参考になるお話でした．OHP のコピーをください．魚田さんへ，UTOPIA の話しを初めて詳しく聞きました．管理者に対してはよい環境だと思うけど，若手のためのツールは？
酒匂さんへ，いかにも酒匂さんらしい面白いお話で楽しめました．

残念ながら，いま特に提案はありません．PCTE に関して，少し詳しい説明が欲しかったと思いました．そうしたサーベイも必要だったのでしょうか？ もう少し議論が発散するかと思いました（当初このセッションだけは発散さ

せようとした）が，意外に何本かの筋のある展開でした．

林： Smalltalk のようなホモジニアスなインテグレーションは，やればできるが，広く普及しないのではないか？ 他の環境上にも使いたいツールがある．ツール間インターフェイスの標準化については，UNIX に見られるように，標準化されると魅力がなくなるということもあるので，個人的にはあまり興味がない．しかし，世の中が標準化にのっとったツールばかりになったらどうしよう．

岸田： ツール･インテグレーションの標準化に関するプロポーザルをまとめるための地下活動を有志で始めよう．われわれがいまやっている SDA みたいなプロジェクトは，そうしたプロポーザルの叩き台をテストする場所として使えるはず．

久保： メインフレームを含むネットワークが現実の姿になりつつある．その例は魚田さんが示してくれた．ツール･インテグレーションを考える時には，このような環境の上で実現可能であることを，本質的なリクワイヤメントにしなければならないと思う．

魚田： 開発者から見れば，よい（使いやすい）ツールが統合化された環境の中で動けばよい（しかし，実際には，既存のツールの中には，あまりよいものがない！）．だれが，「これがベスト」というインテグレーションの見本を出せば，多くの（意識が高い）技術者は，その標準を使い出すのではないか？ だから，ここらで頑張って標準を作り，それを（強引に）世に出すことにしよう．そして，大学･企業はそれに協力する．

酒匂： 使いやすい協力なオブジェクト･マネジメント･システムと，コンポーネント感覚で使えるユーザ･インタフェイス･マネジメント･システムとがあれば，ツール･インテグレーションの道は自ずから定まるでしょう．

Seesion 4: ツール・インテグレーションの理想と現実 (Post-Workshop Position Paper)

ワークショップ参加レポート

魚田 昌孝 (ケーシーエス)

ターゲット・ソフトウェア・システムが変化(高度化，多様化)することにより，開発現場の環境(マシン，ネットワーク)も変わりつつある．そこで，開発に使うツールの統合化問題が，生産性の問題を含めて，クローズアップされてきている．

1. ツール統合の２つアプローチ

(1) 新しい開発環境の構築

現状のさまざまな問題点(過去の環境の欠点)を解決する新しい環境の中に，必要かつ有効な新しい統合化されたツール作る(１つの期待できる環境として Smalltalk-80 があるような感触を持った)．

(2) 既存のツールを活かした統合化

既存のツールを，いろいろな環境で動作できるよう，新しいアーキテクチャによって統合化する．

当然，(1) のアプローチがうまくいけば理想的である．

2. ツール統合における問題

(1) のアプローチをとる場合，現状の環境を変えるには，その効果がはっきりとデモンストレーションする必要がある．また，異なった文化間の摩擦を克服するために，多くの努力(犠牲)が必要である．無闇にやると，逆効果になりかねない．

(2) のアプローチをとる場合，ツールの中身自体を変えることができない(ソースが提供されているツールは少ない)場合が多く，インタフェイスを合わせるための外付けの仕組みが必要になる．そのことによる機能低下・性能低下をどこまで抑えられるか？

3. ツール統合の現実的な道

現状を考えると，(1) のアプローチをとるには，適当なもの(現状を置き換えられるもの)がない．まだ実験の段階(試行錯誤の段階)である．文化の違いを克服する(他の文化を排斥する)には，新しい環境のもとでツール群を整備して実際の開発に適用し，その効果を実証していく必要がある．環境自体が孤立化しないよう，標準化が必要になる．このようなことは，当然１企業だけの力では不可能であり，たとえば SDA のような機構の共同プロジェクトの仕掛けを工夫しなければならない．

１企業が取りうる現実的な道は，(1) のアプローチがもたらす将来の劇的な変化を受け入れることを想定しながら，いま現実にある有効なツール，欲しいツールの統合化を試みることである．この統合化への試みは，当面のしのぎという以上に，将来の変化を受け入れ，成長させるために必要な準備として役立つと考えられる．

4. 統合化のポイント

- ネットワーク上に接続された色々なマシンの上で動作するツールを統合化できるような仕掛が必要である．
- ツール・環境を取捨選択し，有効と思われる部分を対象にし，やりすぎないこと．
- ツールを使う人がある程度カスタマイズできる柔なん性のある仕組みにする．
- ツールについての情報は簡単に入手できる仕組みにする．

5. さいごに(突然ですが)

とにかく，アッと驚くような新しい開発環境が早くほしい．メーカ(H/W,S/W)のエゴを超えて，ツール統合についての標準化を進め，とにかく実践的なツールを早く作り，効果を実証する方法を考えないと，統合化に向かって際限のない独自の努力を続けなければならない羽目になってしまう．

Seesion 4: ツール・インテグレーションの理想と現実 (Post-Workshop Position Paper)

ワークショップを終えて

久保 宏志 (富士通)

1. ツール統合の観点から環境をみると

追求すべき環境を次の構図でとらえるのは，正しいとらえかたであろうか？ それは，環境構築の諸活動に見取図を与えるのに有効であろうか？

環境は，コンピュータ・ネットワークの上に存在する．そのコンピュータ・ネットワークは，企業の壁，産業の垣根，国の境界を越えて地球的規模の広がりを持っている．環境は，その上にツールを統合化するためのプラットフォームである．環境は，ツールをプラットフォームの上に統合化するときに使う言語をその内部に備えている．統合化すべきツールは，パソコン，ワークステーション，メインフレームなどネットワークにつながっているさまざまなマシンに散在している．環境は，統合化されたツールを使うためのユーザ・インタフェイスを提供する．環境のユーザ・インタフェイスから見る限り，ツールの所在は利用者からは見えない．

2. 上のテンプレートで環境構築の状況をみると

上の構図は，環境をネットワークと，プラットフォームと，ユーザ・インタフェースの３つの要素でとらえようとしている．以下では，この３つの実現がどうなっているかを整理しようと試みる．知識不足のため整理は完全ではない．

［ネットワーク］

コンピュータ・ネットワークは，TCP/IP を実装しているマシンを接続することで実現できる．WIDE は，その実現例の１つである．TISN も実現例である．これらは，いまのところまだメインフレーム OS を含むネットワークにはなってはいないが，メインフレーム OS が TCP/IP を実装する方向にあるので，メインフレームを含めたネットワークが作れる条件は整いつつある．

シグマも，本来的には，このようなネットワークを追求すべきであった．ところが現実のシグマはそうはしなかった．メインフレームに TCP/IP を要求すべきであったのに，それをしなかった．代わりに，ネットワークの側に *NA (*=F|H|DI|S) の一部を取り込む選択をしてしまった．ネットワークの上に作るべき環境を，MML の構図の中で実現する結果になっている．

［プラットフォーム］

UNIX のシェルや Emacs は環境の例に数えることができる．その上のツールを統合するための言語を備えている．shell script, Emacs Lisp がそれである．Smalltalk, HyperCard も統合のための言語を備えているから環境である．

ところが現在の状態のこれらは，ネットワーク・ワイドな環境とはいえない．加えてこれらは，操作対象をソフトウエア・オブジェクトに限定していないから，ソフトウエア開発のための環境とは必ずしもいえない．

ネットワーク・ワイドな環境プラットフォームを作るプロジェクトはあちこちで行われている．その上に作られ，統合して使われるべきツールは，その開発が必要とする技術の質からしても，ツールで支援すべき仕事の種類の範囲の広さからしても，１企業の能力で開発できる限界を越える．したがって，これらのプロジェクトの多くが，企業の共同体のプロジェクトや，国家プロジェクトの形態をとっているのはきわめて自然である．

例外的に，大企業が提案しているプラットフォームもあるが (AD/Cycle という名で知られているものがその例)，その企図が成功を収めるのは，その企業が世界を支配しえたときのみであろう．その可能性は皆無ではないが，相当の難事業であろう．連合体が，意見の対立を克服することに手をやき，コンセンサス形成に手間取っていると，特定の大企業に名をなさしめる可能性がある．心すべきである．

大部分の人々は，自分の力の及ぶ範囲を心得ているから，連合を組んでプラットフォームづくりに取

り組んでいる．その中で時期的に最も早かったのはシグマである．少し遅れて SDA があるが，これが目指しているのも環境のプラットフォーム作りである．より正確には，プラットフォーム作りに向けての研究プロジェクトである．研究プロジェクトであって事業化プロジェクトでないとはいえ，それが取り上げている問題からすると，このプロジェクトは本来的にはシグマのサブプロジェクトとして実施するのにふさわしい，とする見方は成り立つ．

プラットフォームづくりのプロジェクトは，欧米にもある．アメリカの CATIS(Common Application and Tools Interface Standard) はその一つである．ヨーロッパの PCTE (Portable Common Tool Environment) もその１つである．アメリカには ADA を前提の国防省プロジェクトもあるようである．CATIS は，DEC, Apollo などからなるコンソーシアムで実施しているようである．PCTE は，EC のプロジェクトかもしれない．いずれも確かなことは知らない．これから調べてみたい．

［ユーザ･インタフェイス］

環境は，ツール統合のインタフェイスおよび統合のための言語だけでなく，統合されたツール群を使うためのインタフェイスを用意すべきであるが，この問題が OPEN LOOK なり Motif によって解決可能なのかどうかはわからない．たぶん，ソフトウエア開発環境に固有のインタフェイス問題があるであろうと思われるが，問題がまだ未定形なせいか，これといったプロジェクトはなさそうである．このプロジェクトも単一の企業で実施する類のものではない．この点ではプラットフォームのプロジェクトと同じである．

Seesion 4: ツール・インテグレーションの理想と現実 (Post-Workshop Position Paper)

自分のポジション・ペーパへのコメント

酒匂 寛 (SRA)

北の街釧路．学生時代に車で通過したことはあったが，滞在は初めてであった．新鮮でおいしい海産物は，現在山里暮らしの我が身には期待に違わぬすばらしいものであった．さて今回のワークショップでは Smalltalk 救世主神話が生まれかけたり，参加者をテーマにした替え歌が飛びだしたり，ミニ四駆が走り回ったり（？）と，色々楽しいハプニングもあったが，個人的には靴の上から足を掻くようなもどかしさもあった．それはなぜかと漠然と考えていたのだが，まだあまり明確な形をとっていないので，自分自身のポジションペーパを読み直しながら感想を付け加えるという形にしたい．あまり報告とは呼べない代物だがお許しいただきたい．

1. 何が問題か

実際には，LAN を使ったデータ統合さえまだまだという印象を受けた．複数の人間が協調してソフトウエア開発を行えるように，計算機で支援しようという動き（Computer Supported Cooperative Work）が現在大きな注目を集めているが，世に出回っているツールは，せいぜいファイルレベル（＋いくつかの属性付き）のデータ交換を中心に支援するだけで，意味的な構造には配慮していない．

ある意味を持った大規模なデータ構造を，ネットワーク上で矛盾なく管理できるような仕掛けが必要である．このためにはまず強力なデータモデルが必要になるが，単純なリレーショナルモデルや，ERA モデル，ネットワークモデルだけでは不十分である．なぜならこれらのモデルは，自らの意味を主張するにはあまりにも弱い記述能力しか持っていないからである．

その意味では「解釈」をどこかにインプリメントしなければならないが，その仕掛けも口で言うほど簡単ではない．外界に対して，オープンなものにしようとするだけでも大変である．しかし自分の個人環境が，プラットフォームを問わず簡単に持ち歩けたら.... と思う．

2. 何がなされているか

環境のプラットフォームとして，前に挙げたような具体的なものが次々に開発されている．高性能なハードウエア，ソフトウエアも安価に手に入るようになってきたが，利用者の目から見て結局決定的に不足しているのは，「それらを使いこなすためのノウハウ／サポート」である．

あれもできるこれもできる式の環境はその機能の１％でも有効に使われていればまだましなほうである．かつて「眠っている脳の機能を活用しましょう」といった宣伝を目にすることがあったが，今実験的に作られているほとんどの環境は，それを思い出させる．「何でもできる」ということは，裏をかえせば「何にもできない」ということと同義ではないだろうか．

3. ツール統合のためのチェックリスト

好き勝手な項目を並べてみたが，実は前述のリストはよく統合された環境の必要条件であって十分条件ではない．もし十分条件なら我々はもうとっくに専用言語マシンに乗り換えて，「環境に関する心配をしていた日もあったねぇ」と述懐でもしているところである．が現実はそうではない．

自由度の高い環境が統合し易い環境かというと，そうとはいえないのである．むしろ自由度が高過ぎて，基礎工事がなかなか定まらず，高い建物を建てようとしても，ぐらぐらしたり，傾いたりして安定せず，２つの建物の間を行き来しようとすると，高い部分には廊下を渡せずに結局地下室を通って出入りしたりすることになる（さらに電灯がなくて別の建物に行ったりする）．

それぞれの建物は，その必要に応じた機能を満たすべく建てられるのは当然だが，その建物の間に安定した渡り廊下を構築できるような工事法／足場が現在の環境には欠落しているのである．検索可能性／再利用性／拡張性などというキイワードがこの「工事法と」直接かかわってくるだろう．さらにいうならば，「言語」は工事法そのものでは「ない」ということである．むしろセメントや釘などに相当する．

4. 現実的解決は

かなり楽天的なことをポジションペーパには書いたが，これらの「解決策」も実は必要条件であって十分

条件ではない.「安定した」足場を段階的に積み上げて行くことができるような「工事法」(と材料)が必要である,上にも書いたが「言語」それ自身だけでは何の解決にもならない.Smalltalk だろうが Lisp だろうが Prolog だろうが同じである.,sp ,3 ,ti +1 大切なことはよい環境を作るためのガイドラインの作成と,そのガイドラインにそった「経験」をつむことによる技法の熟成である.これを支援するために,「高級言語」を道具として使うなら意味はある.これは環境の開発に限らず一般のソフトウエア開発にも当てはまる原則であるが ... .

[独白+自己反省]

環境構築もソフトウエア開発なのであるから,結局よいガイドラインとそれを実現する人間の高度なスキルが必要とされる.その意味で「どのようにでもカスタマイズできる汎用の環境」という概念は画に描いた餅か,あるいは肝心の勝負を避けているだけではないだろうか?

論 文 募 集

ソフトウェア・シンポジウム '92

1992 年 6 月 9 日 (火) - 11 日 (木)

於： 長野県農協ビル・アクティーホール (長野市)

主 催

ソフトウェア技術者協会 (SEA)

協 賛（予定）

日本ソフトウェア科学会 (JSST)　情報処理学会 (IPSJ)

情報サービス産業協会 (JISA)　長野県情報サービス振興協会 (NISA)

基調テーマ： ネットワーク時代における分散開発環境

メーカ・ソフトハウス・計算センタ・エンドユーザ・大学・研究所などさまざまな仕事場で活躍している技術者および研究者が一堂に会し，ソフトウェア技術の実践的側面に関する経験や知識を交流するユニークな場として運営されてきたこのソフトウェア・シンポジウムも，早いものでもう 12 回目を数えるにいたりました．ワークステーションやネットワークの活用も本格化し，開発技術／環境も，またアプリケーションの形態も，大きく様変わりしつつあります．

そこで，今回の論文募集にあたっては，そうしたダイナミックな技術変化の方向性を探るべく，以下に示すような問題点および技術分野をセッション・テーマの候補として選び，討論の焦点を絞ろうと考えています．バラエティに富む具体的な経験にもとづく実践レポートや，革新的なアイデアの提案など，みなさんからの積極的な応募をお待ちします．

募集論文テーマ

プログラム委員会としては，次のような問題点および技術分野をセッション・テーマの候補として考えています：

問題点：	分散システムの構築と運用	ヘテロ・ネットワーク	SW の資産化と再利用	CASE の導入と活用
	新技術の評価・導入・改良	AP ノウハウの蓄積と活用	ユーザとの折衝	チームリーダの育成
技術分野：	開発環境の構築と管理	オブジェクト指向	CSCW	ネットワーク管理
	環境の基盤技術	AI と SW 工学	技術者教育	システムの進化
	SW 開発プロセス	SW データベース	品質の定量化	SW 技術の社会的側面

応募要領

応募論文（Full Paper または Extended Abstract, ワープロまたはコンピュータ出力で A4 判数ページ）を **5 部コピー**して，**1992 年 1 月 31 日までに**，2 人のプログラム委員長のいずれか宛に郵便でお送りください．あるいは，junet にアクセス可能な方は，E-Mail で応募されても結構です（この場合はもちろん 5 部コピーの必要はありません．フォーマットは roff. TeX, Plain Text 何でもかまいません）．

応募論文は，プログラム委員会で内容の審査を行い，結果を 1992 年 2 月下旬までに，応募者全員にお知らせします．審査を通過した方々には，最終論文（予稿集にのせるためのカメラ・レディ原稿）を1992 年 4 月中旬までに仕上げていただくことになります．その他，パネル討論のテーマや招待講演，チュートリアル，ツール展示などに関してのアイデアも，どしどし提案してください．

シンポジウム・スタッフ

＜実行委員長＞	市川寛 (電算)	西条経治 (長野県中小企業連合会)	
＜プログラム委員長＞	岸田孝一 (SRA)	玉井哲雄 (筑波大)	堀江進 (NES)
	中野秀男 (大阪大)	手塚正義 (金沢工大	松原友夫 (日立 SK)
＜チュートリアル委員長＞	野村敏次 (JIP)	中谷多哉子 (FXIS)	松村好高 (SRA)
＜ツール展示委員長＞	丸山徳 (電算)	布川博士 (東北大)	松本健一 (大阪大)
＜プログラム委員＞	荒木啓二郎 (九州大)	野呂昌満 (南山大)	山口高平 (静岡大)
	飯沢恒 (三菱東部 CS)	平岡信之 (長野大)	山崎利治 (日本ユニシス)
	筏井美枝子 (SRA)	平野広美 (ENICOM)	山田茂 (広島大)
	海尻賢二 (信州大)	平山伸一 (KCS)	好川哲人 (ASTEM)
	亀田繁 (FHL)	深瀬弘恭 (アスキー)	我妻克彦 (東北 CS)
	熊谷章 (PFU & 富士通)	二木厚吉 (電総研)	

ソフトウェア･シンポジウム '92

論文応募カバー･シート

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

会社（大学）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

部門（学部）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿役　職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

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

TEL: [　　－　　－　　内線　　] FAX: [　　－　　－　　]

E-Mail: ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

論文タイトル（予定):＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

共同執筆者（もしいれば):＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

Abstract：（論文の要旨を10行以内でまとめてください）

＿＿

＿＿

＿＿

＿＿

＿＿

＿＿

＿＿

＿＿

＿＿

＿＿

問題点と分野：　あなたの論文は以下の問題点や技術分野のどれに関連していますか？　あてはまるものにチェックしてください。

問題点：

□分散システムの構築	□ヘテロ･ネットワーク	□SWの資産化と再利用	□CASEの導入と活用
□新技術の評価･導入･改良	□APノウハウの蓄積と活用	□ユーザとの折衝	□チームリーダの育成
□その他（		）	

技術分野：

□開発環境の構築と管理	□オブジェクト指向	□CSCW	□ネットワーク管理
□環境の基盤技術	□AIとSW工学	□技術者教育	□システムの進化
□SW開発プロセス	□SWデータベース	□品質の定量化	□SW技術の社会的側面
□その他（		）	

カテゴリ：　あなたの論文は，次の2つのカテゴリのうち，いずれに属しますか？（いずれかにチェック）

□ Idea Paper:　新しい概念／技法／ツールなどの提案.

□ Experience Paper:　実際の開発／試行の経験にもとづく報告.

---------- 送 付 先 ----------

〒565 吹田市山田丘２－１
大阪大学　工学部　通信工学科
中野 秀男
TEL 06-877-5111 ext.4773　FAX 06-875-0506
E-MAIL: nakano@oucom.osaka-u.ac.jp

または　〒160 新宿区四谷３－１２　丸正ビル５F
(株)SRA ソフトウェア工学研究所
岸田　孝一
TEL 03-3357-1507　FAX 03-3351-0880
E-MAIL: k2@sra.co.jp

********** **応募締め切り：　1992年1月31日（金）** **********

Call for Papers and Participation

第4回ソフトウェアプロセスワークショップ

― 論文および参加者募集 ―

日本ソフトウェア科学会「ソフトウェアプロセス研究会」では，情報処理学会「ソフトウェア工学研究会」およびソフトウェア技術者協会（SEA）との共催で，下記要領により，第4回ソフトウェアプロセスワークショップを開催します．皆さま，奮ってご応募，ご参加ください．

日　時: 1992年2月12日(水)12：00より2月14日(金)13：00まで2泊3日

場　所: 日本タイムシェアシステム総合研究所研修センター(浜名湖湖畔)

目　的: ソフトウェアプロセスに関する研究はしだいにさかんになり，米国では国防省と契約を結ぶ会社の評価項目の一つとして,W.Hamphreyの提唱したプロセス成熟度が加えられる等，社会的にもその重要性が認知されつつあります．また1991年10月には，過去に7回開催されたISPW (International Software Process Workshop)の実績を基礎にして，初の国際会議もカリフォルニアで開催されました．

ソフトウェアプロセスの研究分野では，日本の研究者は，研究が開始された時期から世界的な研究の流れと一致して，研究コミュニティの形成に寄与するとともに，日本独自の問題点も検討しつつ，研究･技術開発の努力を継続してきました．一方，企業においても，各社のソフトウェアプロセスの調査や形式化がさかんに試行されはじめております．

このような活動をより組織的にすることにより，日本におけるソフトウェア開発活動の規範性と柔軟性を高め，自動化を推進していくための努力を結集する必要があるかと思われます．基礎研究にたずさわる研究者の方々，実践の場で問題の解決に努力されている技術者の方々，どうぞ奮ってご参加ください．

トピック: 応募論文の話題は主に以下の範囲を考えています．これら以外でもソフトウェアプロセスに関連する話題であれば何でも歓迎します．

- 大規模ソフトウェア開発におけるソフトウェアプロセスの調査と形式化
- 分散開発環境時代のソフトウェアプロセス
- (CASEツールと)プロセスモデルに基づく統合環境
- ソフトウェアプロセスの表現法(プロセス記述言語)とその実行環境
- ソフトウェアプロセスモデリング手法
- グループウェアとソフトウェアプロセス
- オブジェクト指向開発法への技術移転プロセス

なお「分散開発環境時代のソフトウェアプロセス」についてはパネル討論も計画しています．

論文応募方法:

(1) フルペーパー(A4で8頁以内)またはアブストラクト(A4で3～4枚程度)を10部，1月8日(水)までにプログラム委員長宛にご送付ください．(やむをえず，期限にまにあわない場合はプログラム委員長までご相談ください)
(2) 採用通知は1月24日(金)の予定です．
(3) 論文発表者は，論文コピー50部を当日ご持参ください．

参加申込み方法:

(1) ポジションペーパー(A4で1枚以内)を10部，1月20日(月)までにプログラム委員長宛送付．
(2) 申込者が50名を越える場合は調整させていただきます(通知は1月24日)．
(3) 参加者は，ポジションペーパー50部を当日ご持参ください．

参加費:

- 学生:　会員15,000円 非会員25,000円
- 大学/国立研究機関:　会員25,000円 非会員40,000円
- 一般:　会員30,000円 非会員45,000円

(注)会員はJSSST,IPSJ,SEAの会員です．

報告集: ワークショップの結果をふまえた報告集を後日発行いたします．参加者の方には報告集が完成しだい送付させていただきます．

スタッフ:

プログラム委員会，実行委員会

委員長：　落水浩一郎　　〒432浜松市城北3-5-1　静岡大学工学部情報知識工学科
Tel: 053-(471)-1171内線507，Fax: 053-(475)-4595，e-mail ochimizu@cs.shizuoka.ac.jp

プログラム委員：　鯵坂恒夫(京大)，井上克郎(阪大)，片山卓也(北陸先端科学技術大学院)，佐伯元司(東工大)，
園部正幸(富士通研究所)，玉井哲雄(筑波大)，中島毅(三菱電機)，二木厚吉(電子技術総合研究所)

実行委員：　鎌田孝(日本タイムシェア)

***** (問合わせは委員長までお願いします) *****

1991年9月

第1回 インターネット国際会議 (INET'92)
論文募集のご案内

最近、世界各地においてインターネットワーキングの活動および研究開発が盛んになってきました。そこで1991年末を目指して、インターネット学会という新しい国際学会を設立する運びになっていますが、この学会主催の第1回会議を1992年6月にINET'92として神戸で開くことになりました。この会議の目的は、国内および国際的な研究アカデミック・ネットワークの研究開発、計画、構築、管理などにたずさわっている大学、企業、政府などの関係者を集めて、いろいろな角度から、インターネットワーキングの問題について研究発表を行い、論議を行うことにあります。INET'92ではとくに技術および応用の分野で論文募集を行いますので、ふるって、論文を投稿されるようお願いいたします。

開催期日　1992年6月16日（火）～18日（木）〔15日（月）はチュートリアル〕

開催場所　神戸国際会議場（参加予定者は約500人）

トピックス

〈技術とサービス〉	〈インターネットの応用〉
オープン・ネットワーク・プロトコル	国際協力へのネットワーク・サポート
ネットワークのインターオペラビリティ	国際データベース・アクセス
グループウェア技術	スーパーコンピュータ応用
ネットワーク・セキュリティ	高エネルギー物理
９０年代の技術	ワークステーションによるテレビ会議
メールやディレクトリのサービス	コンピュータによる協同作業
超高速ネットワーク	教育
テラバイト・メモリ装置	ライブラリ
２１世紀のネットワーク・サービス	

主催　Internet Society

協賛　情報処理学会，電子情報通信学会（予定），ACM SIGCOM, IEEE Computer Society

論文投稿要領

論文発表ご希望の方は、1992年1月6日までに、英文（A4版，ダブル・スペースで20ページ以下）の論文のコピー6部を下記まで送付して下さい。（日本からは約10件を予定しています。）採否は1992年3月1日頃に通知いたします。採録された論文については、1992年4月15日までに、カメラ・レディの最終原稿をご提出下さい。

論文提出先　〒113
文京区弥生2の11の16
東京大学大型計算機センター　石田晴久
（電話）03−3818−0287　（FAX）03−3814−7279
（e−mail）ishida@u−tokyo.ac.jp

問合わせ先

本会議についての詳しい内容は下記までお問合わせ下さい。

〒252
神奈川県藤沢市遠藤5322　慶応大学環境情報学部
INET'92事務局（村井　純研究室）
（電話）0466−47−5111 内線3330　（FAX）0466−49−1101
（e−mail）inet92@wide.ad.jp

CALL FOR PAPERS, TUTORIALS, AND PANEL SESSION PROPOSALS

CONFERENCE ON SOFTWARE MAINTENANCE – 1992

ORLANDO, FLORIDA
November 9 – 12, 1992

The Conference on Software Maintenance – 1992 (CSM-92) will gather software managers, developers, maintainers, and researchers to discuss new solutions to the continuing challenges of software maintenance and software maintainability.

Theme: Improving the Processes of Software Maintenance

CSM-92 will focus on the processes employed in performing software maintenance – as a means of understanding, defining, improving, managing, and supporting the practices of software maintenance. CSM-92 seeks papers and other contributions that will advance the state-of-the-art or state-of-the-practice of software maintenance, and help foster increased interaction between the research and practitioner communities.

SUGGESTED TOPICS

Papers, Experience Reports, Panel Session Proposals, and Tutorial Proposals related to, but not limited to, the following topics are invited:

• Software Maintenance Process Modeling or Assessment • Successful Maintenance Processes • Software Maintenance Methodologies • Software Evolution, Migration, and Conversion • Reengineering, Reverse Engineering, and Restructuring • Standards for Software Maintenance • Empirical Maintenance Studies • Software Maintenance and Maintainability Metrics • Software Maintenance Management • Software Maintenance Education • Program Understanding • Achieving Feedback from Maintenance to Development • Software Maintenance Tools • Impact Analysis • Developing Maintainable Systems • Documenting for Software Maintenance • Recertification of Modified Software

CONFERENCE COMMITTEE

GENERAL CHAIR
Vaclav Rajlich
Department of Computer Science
Wayne State University
Detroit, MI 48202
TEL: (313) 577-5423; FAX: (313) 577-6868
email: vtr@cs.wayne.edu

PROGRAM CHAIR
Marc Kellner
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
TEL: (412) 268-7721; FAX: (412) 268-5758
email: mik@sei.cmu.edu

PUBLICITY CHAIR
Dave Card
Computer Sciences Corporation
Systems Sciences Division
4061 Powder Mill Road
Calverton, MD 20705
TEL: (301) 572-3815; FAX: (301) 572-7502
email: dc@sei.cmu.edu

TUTORIALS CHAIR
Frank Calliss
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
TEL: (602) 965-3190; FAX: (602) 965-2751
email: calliss@asuvax.eas.asu.edu

SCOPE AND PURPOSE

Software Maintenance includes the enhancement, restructuring, and correction of software. CSM-92 will provide a forum for the discussion of software maintenance through refereed papers, experience reports, panel sessions, tutorials, and informal meetings. CSM-92 will present the most important current practical, experimental, and theoretical work being conducted in support of software maintenance. CSM-92 will entertain papers and other contributions in the areas of Basic Research, Experiences with Technology Application, and Empirical Research. The conference is open to all who are involved with or have an interest in software maintenance: practitioners and researchers, from industry, government, and academia.

SUBMISSION INFORMATION

Five (5) copies of papers, experience reports, or panel proposals should be submitted in English to the program chair at the address listed. The author of each accepted contribution will be expected to present it at CSM-92.

Papers should be 1000-5000 words in length. Papers must not have been previously published or be under current consideration for publication elsewhere. The first page must include title, all authors' names, lead author's address and telephone number, and abstract (250 word maximum).

Experience Reports are intended to summarize experiences with the practical application of software maintenance technology or processes (e.g. using a tool or method, applying a metric, following a project management discipline, etc.). The contributor(s) should submit a 4-6 page written description of the experience, highlighting the outcomes and lessons learned. The submission should be identified by the keyword "Experience" on the top of the first page, and otherwise follow the first page instructions given for papers. The evaluation criteria for these submissions will be based on the applicability and relevance of the results to the software maintenance community.

Panel Proposals should include title, proposed chair, tentative panelists (including short vitae), a brief description of the subject of the panel session, and supporting rationale. Panelists must have agreed to participate prior to submission of the proposal.

Tutorial Sessions have been a significant part of past CSM conferences. Proposals are solicited for both full-length, all-day tutorials and also for mini-tutorials of two to four hours in length. Direct five (5) copies of proposals to the tutorials chair.

IMPORTANT DATES: Submission deadline: April 1, 1992;
Acceptance notification: July 15, 1992; Final version due: September 1, 1992

SEA Forum / January '92

先端技術の導入と教育

― テクノロジーギャップをどうやって克服するか？ ―

主催：ソフトウェア技術者協会

早や12月．今年もあと少しというところです．クリスマス、お正月の話もちらほらと...

さて，年明け最初のSEA Forumは，9月に北陸で開かれた第5回教育ワークショップでの討論成果を踏まえて，SIGEDU（教育分科会）メンバを中心に「先端テクノロジーの教育と普及」をテーマとするパネル討論を企画しました．

ソフトウェアの世界における技術進歩は依然として急ピッチであり，次から次へと新規なTechnical Buzzwordsが新聞や雑誌の紙面に登場しては消えて行きます．しかし，CASEにせよHyperMediaにせよオブジェクト指向にせよ，新しい技術上のアイデアを具体的な形にし，現場に広めて行くためには，研究試作段階から実用化へ，一部の好き者（教祖様）たちから現場全体へと，テクノロジー・トランスファを促進するための地道な努力が，**特に教育や訓練の場において**費やされなければならないのです．

本来きわめて技術オリエンテッドであるはずのソフトウェア・ビジネスが，ともすれば単なるマンパワー提供業に堕落しかねないのは，われわれの側において，先端技術の動向を追跡し研究開発コミュニティとの間のギャップを克服するための努力が不足しがちであるからだと考えられます．たしかに，通常の平均的技術者教育と比較すれば，先端技術の導入や普及のための教育は数多くのむずかしい問題点を含んでいます．しかし，業界全体を見渡してみると，中には実際にかなりの成果を上げているケースもいくつか見受けられるのです．

今回のForumでは，このような経験をお持ちの方々をパネリストに迎えて，その苦労話やノウハウを交えて討論を展開したいと考えています．多数のご参加をお待ちしております．

開 催 要 領

（1）日　時：　1992年1月27日（月）　13:30 ～ 16:00
（2）会　場：　機械振興会館（東京・港区芝公園）地下3F研修1号室
（3）コーディネータ：　川辺　正明(スタット)
　　パネリスト（予定）：　篠崎直二郎(日本電気ソフトウェア)　　君島浩(富士通)　　中山照章(富士通SSL)
　　藤野晃延(富士ゼロックス情報システム)　　渡辺雄一(アスキー)
（4）参加費：　2,000円（SEA正会員）　5,000円（SEA賛助会員）　10,000円（一般）
（5）申し込みおよび参加方法：　下の申込書に必要事項をご記入の上，郵便，FAXまたはE-Mailでお送り下さい．折り返し参加案内をお送りします．受講料は，**当日現金で受付にてお支払下さい**．ただし，定員（70名）になり次第，締め切らせていただきますので，あしからずご了承下さい．なお，申込後のキャンセルはお断わりします．

申込先：

〒160　東京都新宿区四谷3 - 12丸正ビル5F　　ソフトウェア技術者協会・フォーラム係
TEL: 03 - 3356 - 1077　　FAX: 03 - 3356 - 1072　　E-Mail: sea@sea.or.jp (junet)

.....

SEA Forum / January '92 **参加申込書**（申込日：　　月　　日）

氏名（ふりがな）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿（　　）

種別（いずれかにチェック）：□正会員（*NO.*＿＿＿＿＿＿）　□賛助会員（*NO.*＿＿＿＿＿＿）　□一般

参加費：　＿＿＿＿＿円

会社（学校）名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

部門：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

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

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