SEAMAIL

Newsletter from Software Engineers Association

Volume 5, Number 8-9 1990

目　次

編集部から		1
ソフトウェア・プロセス		
ISPW 初体験の記（6th ISPW に参加して）	玉井　哲雄	2
Software Process - Themes and Issues (OHP)	Mark Dowson	6
全国縦断 CASE Forum から		
CASE の一般認識（CASE Forum NAGANO より）	市川　寛	17
CASE について考える (OHP)	岸田　孝一	21
これからの CASE と Object 指向 (OHP)	熊谷　章	26
Software Symposium '90 基調講演	Gerhard Fischer	44
akanc でのプログラミング入門（1）	新田　稔	54
儲かってまっか SEA 関西日記	中野　秀男	64
気がついたら3年半－長野支部活動報告	市川　寛	71
Call for Papers		
7th International Software Process Workshop		72
1st International Conference on Software Process		73

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，1700 名を越えるメンバーを擁するにいたりました．法人賛助会員も約 60 社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

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

代表幹事： 熊谷章

常任幹事： 落水浩一郎　岸田孝一　中野秀男　野村行憲　平尾一浩　深瀬弘恭　松原友夫

幹事： 青島茂　天池学　新井秀明　市川寛　臼井義美　江木康雄　岡田正志　大木統雄　大林正晴　片山禎昭　片山卓也　加藤重信　加藤重郎　窪田芳夫　杉田義明　田中一夫　玉井哲雄　玉川滋　鳥居宏次　中来田秀樹　中山照章　野中哲　野村敏次　林香　人見庸　藤野晃延　二木厚吉　北條正顕　松本省二　盛田政敏　山崎朝昭　山田淳　渡邊雄一

会計監事： 辻淳二　吉村成弘

分科会世話人 環境分科会(SIGENV)：田中慎一郎　渡邊雄一
管理分科会(SIGMAN)：大久保功　加藤重郎　野々下幸治
教育分科会(SIGEDU)：大浦洋一　杉田義明　中園順三
ネットワーク分科会(SIGNET)：青島茂　野中哲　久保宏志
法的保護分科会(SIGSPL)：能登末之

支部世話人 関西支部：臼井義美　中野秀男　盛田政敏
横浜支部：熊谷章　林香　藤野晃延　松下和隆
長野支部：小林貞幸　佐藤千明　細野広水
名古屋支部：岩田康　鈴木智
九州支部：植村正伸　小田七生　藤本良子　平尾一浩　松本初美　中島泰彦　後藤芳美

SEAMAIL Vol.5, No.8−9　平成2年11月26日発行
編集人　岸田孝一
発行人　ソフトウェア技術者協会（SEA）
〒160　新宿区四谷3−12　丸正ビル5F
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

☆

ごぶさたしました．なかなか編集長から足を洗えないでいる岸田です．とりあえず手元にある材料をかき集めて，夏以来のびのびになっている船便をお送りします．

☆☆

９～１０月はいつものことながらイベントが続きました．そのなかから，函館のプロセスワークショップの速報（本当に速い！，玉井先生ありがとうございました）をお届けします．また，プロセスという yet-another-buzzword に戸惑っておられる方々のために，東京で開かれた国際シンポジウムの資料のなかから，M.Dowson 氏のチュートリアル･プレゼンテーションのコピーを再録しました．

☆☆☆

５月からはじまった全国縦断 CASE Forum は，いちおう今年の分は（え？来年もやるの？）先日の大阪でピリオドを打ちました．７月の長野 Forum のサマリ，および２人も基調講演者の OHP のコピーをまとめてのせておきます．ほんとうは，テープ録音から起こしたレポートのほうが望ましいのでしょうが，先日の幹事会での議論の結果，きれいなレポートが間に合わなければ，OHP のコピーだけでものせたほうがよいということになりました．

☆☆☆☆

６月のソフトウェア･シンポジウムからは，Gerhard Fischer 先生の基調講演の記録ができあがりました．阪大の中野研究室の学生さんたちの縁の下での御努力に深く感謝します．

☆☆☆☆☆

このページで何度も呼び掛けているのですが，なかなか会員からの技術的な原稿が集まりません．ようやく，今回新田さんに貴重な原稿を寄せていただきました．第２回以降もよろしく．他の会員のかたも，ぜひ，御寄稿ください．

☆☆☆☆☆☆

幹事の野中さんから，各支部および分科会の世話人の方々に原稿依頼が出ています．とりあえず事務局にとどいているのは，今回のせた関西（中野先生，御苦労さま！）と長野（市川さんも，ありがとうございました！）の２つだけです．他の支部および SIG のみなさんも，ぜひ．

☆☆☆☆☆☆☆

次号（１２月発行予定）は，昨年秋に京都で行われた国際シンポジウムの全記録を特集する予定（もちろん、英語で！）です．御期待ください．

ISPW 初体験の記

— 6th ISPW 報告 —

玉井哲雄
(筑波大学)

1．はじめに

第６回国際ソフトウェアプロセス･ワークショップ (6th ISPW) に参加した．場所は，函館の北，大沼国定公園のほとりのプリンス･コテージ．期間は 10 月 29-31 日である．

このワークショップは，今回初めて日本で開かれた．といっても，筆者はこれまでのワークショップに参加したことがなく，過去の開催と今回とを比べることはできない．米国メイン州で開かれた前回については，SEAMAIL, Vol.5, No.1-2, に岸田，中川，中島，各氏の大変おもしろい報告がある．その中で，岸田さんはそれ以前の開催についても，自身の参加経験を基に触れられているので，参照されたい．

白状すれば，今回参加したといっても実はポシション･ペーパを出していない．だから，正式な参加メンバーとは言えない．そもそも，ソフトウェア･プロセスという概念に，つい最近までそれほど注目してはいなかった．昨年京都で第１回が，今年コロラド州ボウルダーで第２回が開かれた”未来のソフトウェア環境国際シンポジウム (ISFSE)”に，続けて参加する機会を得た．そのシンポジウムの核となるメンバーが ISPW とほぼ同じであり，内容もかなり共通することから，”プロセス屋社会 (process community)”に新顔として出入りを許されるようになり（もちろん，この”社会”のかなりの人々，とくに日本のメンバーとはずいぶん前から顔なじみではあったが），あまつさえ次の第７回 ISPW の組織委員会に入ってほしいと誘われた．深く考えもせずに引き受けてしまったが，すでにかなりの歴史のあるワークショップに，一度も出ていないで委員をやるというのは余りに無謀だ．そこであわてて日本で開かれる今回に，ポジション･ペーパの締切はとっくに過ぎ，セッション･プログラムまで確定した段階で，何とか主催者にお願いしてオブザーバとして出させていただいたものだ．

2．ワークショップの構成

6ISPW の組織委員長は片山卓也（東工大），委員は G.Kaiser(Columbia Univ.)，岸田孝一 (SRA)，落水浩一郎（静岡大），W.Riddle(rMISE)，J.Wileden(Univ. Mass. ただしワークショップには欠席)．参加者は約 40 名，内海外から 30 名，日本から 10 名という割合である．海外は米国とヨーロッパで，米国 20 名，ヨーロッパ 10 名といったところ．日米欧が，１対２対１ときれいな比になったのは，多分に人為的なものであろう．実際，20 ぐらいのペーパを落とさざるを得なかったということだが，同じプロジェクトからの複数申込は１つに絞るなど配慮があったらしい．

日本で開かれた催しにしては日本の参加者比率が少ないが，このワークショップの運営に継続性があり安定していることの証拠と言える．むしろ，海外から多くの参加者のあったことを喜ぶべきであろう．また，ヨーロッパからの参加が比較的多いことも目につく．Esprit プロジェクトに代表されるように，ヨーロッパでこのところソフトウェアを対象とする大型プロジェクトが盛んで，その成果が出始めているようだ．それに，W.Schafer(Univ. Dortmund) も言っていたが，いまやヨーロッパと日本との間の時間距離は案外に短いのだ．

３日間で６セッションが組まれた．各セッションは，まずまとめ役の人が１５分ほど口火となる話をし，あとは自由討論をするという形態である．この形は，第４回以降定着したそうだ．討論中心と言うのは，人数を絞ったワークショップならではのスタイルで大いに意義があるが，なかなか大変なものでもある．とくに，新入り，

英語を母国語としないもの，控え目な性格のもの，議論をすることの訓練を受けていないもの，生活文化背景を共有することを前提とした人間関係を作ることに慣らされているもの，などにとっては，議論を追うことはできても議論に参加することは難しい．筆者は上の条件がすべて成立する上に，オブザーバという立場で発言権があるのかどうかも怪しい．しかし，まったく黙っているのもつまらないので，少しはしゃべったが．

それだけではない．議論は，往々にして抽象論に走る．空高く舞い上がる．そうなると，発言者も，いわゆるISPW マフィア（岸田さんの文中に出てくる言葉）に限られてしまう．前回のメインでは，中川さん，中島さんの報告にもあるようにこれが甚だしかったらしく，その反省から今回とられた対策もある．一つは，片山先生の提案で，プロセス記述の共通問題を作って各自記述実験をし，その結果を基に具体的な議論をしようというものである．M.Kellner(SEI) が苦心して作った問題に対し，19 の解答例が集まっていた．解答の比較検討会はワークショップ開催前の 27 日に東京で開かれたそうであるが，本ワークショップの中でもその結果が活かされる予定であった．

もう一つは，議論の交通整理役を置いたことで，この"おまわりさん"がかなりの権威で割り込みや勝手な発言を抑え，基本的には挙手順に発言者を指名する建前である．これは，ある程度機能した．

３．セッションの進行

各セッションについて細かく報告していったら長いものになりそうだし，大体それができるほどノートもとっておらず，記憶力も減退している．以下，およその流れを思いつくままに記す．全体のよいまとめは，最後のセッションで I.Thomas(HP) が見事にやってみせた．いずれ，ワークショップ・レポートとしてでる予定なので，関心がありしかも辛抱強い読者は（というのも，出るのが大分先になることは確かなため．前回のものがいまだにできていない），それを読まれるのが一番よい．なお，ここまで日本人だけに敬称をつけてきたが，考えてみるとそれもおかしなものだ．思い切って，すべて敬称抜きの姓だけで済ますことにしよう．関係者には，ご容赦をお願いする．

最初のセッションは，プロセスモデリングの概念というテーマである．まず，熊谷（富士通）が基調報告をした．面白かったのは，プロセスを外部から観察して分析する場合と，内部にいて見る場合とでは様相が異なるという指摘で，これを比喩的に叙事詩と叙情詩との違いになぞらえた．これは十分に刺激的な発言だったが，残念ながらその後の議論の展開はこの方向には進まなかった．Balzer がソフトウェアプロセスの特徴のリストアップという，はるかに散文的で，片山が指摘したようにこのワークショップでもすでに繰り返し議論してきている論点を出してきたら，議論はほとんどそちらの方へ行ってしまった．

２番目のセッションが，先に述べた共通問題の結果報告で，Kellner と Rombach(Univ. Maryland) が担当した．しかし，せっかく豊富で具体的な材料がありながら，議論がそれを活かす方に進まなかったので，不満の声が出た．そこで動議がだされ，以降のセッション予定を，この共通問題の成果を取り込む形に変更することが求められた．動議は通り，それに基づくプログラムの変更の仕方については委員会一任となった．その夜，委員達が協議した結果，セッション構成は変えずに，翌日の午前のアーキテクチャのセッションで，いくつかの記述例をアーキテクチャの観点から紹介してもらうこととなった．

結果的にはこの措置が，またまた波乱を招くもととなった．30 日午前の第３セッションは，ソフトウェア開発環境のアーキテクチャというもので，まず落水が関連するポジションペーパーをとりまとめながら，要領をえた基調報告を行った．それから共通問題の解答記述の紹介という趣旨で，当初３件の発表が予定されていたが，それなら自分も話したいという申し出が２件あったらしく，都合５件の発表が行われることになった．しかし，時間の制約からすると５件は多すぎたのであろう．それぞれの発表が，この際自分達の仕事をよく伝えようと長くなりがちなことに加え，さまざまな質疑，討論がされるから，最後の１人がはみでてしまった．それが，たまた

ま隣に座っていたイタリア人の Ambriora(Univ. Pisa) である．彼はすでに事態がそうなる以前から，それまでの発表があまりアーキテクチャと関係がないこと（これは確かにそうだったが，同時にアーキテクチャとして想定しているものが，人それぞれでずいぶん違っていたことも確か），また会議がどうしても米国人主導で行われること，に不満感を抱いていた様子で，午前中に自分の発表ができないことがわかると，かなり感情をあらわにして文句を言った．筆者は隣同士というよしみからも同情したが，後でもともと彼の発表はいわば飛び入りであったことを知り，その同情の気持ちもやや後退した．

この件には，実は発表中心か議論中心かという運営の基本方針の問題もからんでいる．すでに書いたように，このワークショップでは当初発表中心にやってきて，それへの反省から議論中心を基本方針とするという意思決定が，核メンバー達によってなされた．その連中にとっては，１セッションに５件の発表があること自体が方針に抵触し，まして発表できないと文句をいうことはおかしいということらしい．確かに，テーマからはずれたり，自分の研究の宣伝のような発表が長々と続くことは問題である．しかし，ある程度は準備された発表がプログラムに組まれていた方が，議論もいたづらに哲学的にならず，効果的なのではないかと思う．実際，筆者には，多くの問題があったにも関わらず，このセッションの方が前日のよりも面白かった．

件の Ambriora の発表は，とにかく繰り下げて午後に行われた．その後，午後の本来のセッションテーマである”オブジェクト管理システムとメカニズム”について，Kaiser が例によって早口で基調報告をした．これもなかなかよく練られたまとめだったが，途中で議論が入るので，このセッションはほとんどそれだけで終わってしまった．Kaiser は，発表するときはものすごい勢いでまくしたてるが，自由討論の時はほとんど発言せず，見かけによらず控えめな印象を受けた．このワークショップに限っては，相対的に女性の発言が少なかった．もっとも，Penedo(TRW) は別である．問題は，性別による違いというよりは，年期とか場数とか権威とかの違いによるものかも知れない．しかし，日本はそもそも女性の参加者もいないのだから，それ以前である．

最終日 31 日の午前は，チーム作業をテーマとするセッションだった．まず，D.Perry(AT&T) がなにかしゃべったはずだが，まったく覚えていない．ただし，Perry はこの会議を通じて活発に発言し，独特のユーモアでなごやかな雰囲気作りにも大いに貢献していたことを記しておく．続いて，岸田がいつもながら聞きどころのある発表を行った．組織，プロジェクト，グループ，個人，というレベルでプロセスを考えたとき，上位から降りてくるゴールは，個人レベルのゴールと通常相入れない．逆に，下のゴールを上位へ昇華させる仕組みを考える必要がある，というもの．孔子と老子の政治思想の対比を比喩として出すところなどがにくい．

その後の議論は，不活発だった．チームや共同作業の問題，人間間のコミュニケーションの問題，認知プロセスといったテーマでは，少なくともここに集まった人達の中に，それほど成果をあげているものがいないということだろう．中では，プロセスの成熟度の評価というユニークな視点で実証的な成果を出している Humphrey(SEI) の発言がもっとも迫力があったが，それでも議論全体の流れを作り出すまでにはいたらず，単発的な印象だった．

午後のまとめのセッションは，すでに述べたように Thomas が要領のよいまとめをした．Thomas の話の手際がよかったためというより，最終日の午後で皆の疲れがでたせいで，ここでもあまり議論は盛り上がらなかった（誰かが，”金曜日”の午後の昼食の後はこんなものさといった．実は水曜なのだが，気持ちが出ている）．

４．感想

初参加の自分としては，ソフトウェアプロセスの研究状況を手っとり早くつかめたという意味でもよかったが，全体になかなか楽しめた．セッション以外でも多くの人達と話す機会の多いことが，こういう集中的な会議のいいところだろう．

それに何より場所と季節がすばらしく，しかも期間中見事な晴天続きで，例年より遅いという見頃の紅葉をひ

きたたせた．池を眺望するホテルの露天風呂はよい設計で，とくに海外からの女性参加者に好評だった．Balzer がコウイチ（岸田）は，こういう設営にかけてはお手のものだと言っていたが，運営をされた方々に感謝したい．

会議を通して印象に残ることの一つに，process community という言葉が実に頻繁に出てきたことがある．冒頭でこれをプロセス屋社会と訳してみたが，プロセス派，プロセス（学）界などというより，縄張り意識がさらに働いているような気がする．しかし，これは日本人である自分が，ヤクザの世界や，自民党の派閥，役所や会社のセクショナリズムに，引きつけて解釈しすぎているのかも知れない．たとえばどんな文脈で使われるかと言うと，チーム活動に関する議論が盛り上がりに欠けた際，CSCW の方の動きはどうかという質問に対し，CSCW のコミュニティも最近沈滞気味で，むしろわれわれプロセス・コミュニティの方に期待しているらしい，などという発言が出てくる．Kaiser はデータベース・コミュニティでも活躍しているようで，そことプロセス・コミュニティとの違いなども話題に出た（Kaiser の話は，どちらかと言うと技術的な違いについてではあったが）．人が集まり組織ができれば，何らかの政治が内部でも外部に対しても生まれることは当然である．ソフトウェア開発という場面に限定はしているが，人の営みとしてのプロセスを研究対象としているグループが，そのグループとしての活動プロセスについては，方法論的ではなく，他のあらゆる集団的人間活動と同じように動いていることがおもしろい．

５．第７回の ISPW

次回は，Ian Thomas を委員長として 1991 年 10 月 16-18 日に，サンフランシスコの北，ワイナリーで有名な Napa で行われる予定である．フランス育ちのイギリス人でワイン通の Thomas らしい，場所の選定である．会議上として予定している場所の近くに大変いいフランス料理店があるので，晩餐会をそこでやると言っていた．

最初のプログラム委員会を，このワークショップ期間中に開いた．ポジションペーパの評価の方法，セッションの運営方法，とくに準備された発表をいくつか行わせるかどうか，などについて議論した．

ポジションペーパの締切は，1991 年 3 月 31 日である．日本からもまた，多くの参加があることを期待したい．極上のワインが待っています．

SOFTWARE PROCESS THEMES AND ISSUES

Mark Dowson

SOFTWARE DESIGN & ANALYSIS INC

Reston, Virginia

Process Themes: slide 1 Ref. No. 5-3-2, 14 September, 1990

THE PROBLEM

We are part of an industry that manufactures (and maintains and evolves) large, complex artifacts – software systems. We would like our products:

- to be of high quality and reasonable cost
- to be delivered on time and within budget

The only way to achieve these objectives is to focus on the manufacturing process – the software process.

Process Themes: slide 2 Ref. No. 5-3-2, 14 September, 1990

SOFTWARE PROCESS

A set of steps for creating and evolving software systems encompassing both technical and managerial concerns. In particular it must address:

- technical development
- project management
- data and configuration management
- quality assurance and control

throughout the software life cycle.

Process Themes: slide 3 Ref. No. 5–3–2, 14 September, 1990

PROCESS COHERENCE

All software is produced by *some* process; but often the process is incoherent and implicit. This leads to:

- lack of predictability
- lack of repeatability
- lack of a basis for improvement

For predictability, repeatability, and improvement we need processes that are *explicit* and *coherent* so that they can be followed consistently, monitored, reasoned about, and evolved.

Process Themes: slide 4 Ref. No. 5–3–2, 14 September, 1990

PAST APPROACHES

Riddle's software decades:

- 1960s: Code Cowboys – rugged individualism
- 1970s: Program Rustlers – organized gangs
- 1980s: Software System Drives – corporate professionalism.

Process Themes: slide 5 Ref. No. 5–3–2, 14 September, 1990

1960s

Emphasis on technical development and on the coding phase:

- Small projects depending on small groups of good programmers.
- Improvements in programming techniques seen as the key success factor
  - high level languages
  - programming methods.

Process Themes: slide 6 Ref. No. 5–3–2, 14 September, 1990

1970s

- Large projects requiring team–based project management.
- Emphasis on technical standards and procedures:
  - standard languages and technical methods
  - configuration management
  - quality assurance.
- Widening of technical improvement focus to include 'upstream' activities.
- Emergence of software engineering.

Process Themes: slide 7 Ref. No. 5–3–2, 14 September, 1990

1980s

- Corporate standards and procedures.
- Professional management.
- Some integration of project, configuration and quality management.
- Phase–based processes.
- Separate tools supporting project and configuration management, and upstream technical activities (CASE).
- Attempts at 'integrated environments'.
- Emergence of process view.

Process Themes: slide 8 Ref. No. 5–3–2, 14 September, 1990

PROCESS VIEW

- Initially promoted by a small group e.g. Lehman, Stenning in the UK; Boehm, Balzer in the USA, via a series of small, discussion–oriented workshops – the International Software Process Workshops:
  - Runnymede, UK, 1984
  - California, USA, 1985
  - Colorado, USA, 1986
  - Devon, UK, 1988
  - Maine, USA, 1989
  - Hokkaido, Japan, 1990
- Interest in the process view has been growing over the last few years, and it is now the central topic for numerous groups and research projects.

Process Themes: slide 9 Ref. No. 5–3–2, 14 September, 1990

COMMERCIAL BREAK

1st International Conference on the Software Process

21 – 22 October 1991

California

Process Themes: slide 10 Ref. No. 5–3–2, 14 September, 1990

PROCESS THEMES

- process models
- process assessment and improvement
- process description formalisms
- process enaction
- process programming
- process support

Process Themes: slide 11 Ref. No. 5-3-2, 14 September, 1990

PROCESS MODEL

- A descriptive representation of the structure of a software process
  - general enough to represent a range of particular processes
  - specific enough to allow reasoning about them.
- Building process models involves:
  - studying actual project processes, good and bad
  - inventing hypothetical ideal processes.
- Well-known examples include:
  - waterfall model
  - Boehm's spiral model.

Process Themes: slide 12 Ref. No. 5-3-2, 14 September, 1990

PROCESS ASSESSMENT AND IMPROVEMENT

- Assessment and improvement efforts focus on "Software Process Realities"; the process(es) used by real organizations to perform real software projects.
- Process assessment is not a beauty contest; but a necessary pre–requisite for systematic improvement.
- Improvement efforts do not attempt to enforce conformity to a specific "ideal" process, but aim to achieve an explicit, repeatable process that can be progressively improved.

Process Themes: slide 13 Ref. No. 5–3–2, 14 September, 1990

PROCESS DESCRIPTION FORMALISMS

- Existing formalisms range from English text "policy and procedure manual" style documents to formal programming language–like representations.
- As yet, there is no consensus on appropriate formalisms.
- Multi–faceted process descriptions involving multiple formalisms (for various aspects of process) are likely to be needed.

Process Themes: slide 14 Ref. No. 5–3–2, 14 September, 1990

PROCESS ENACTION

Process descriptions are *enactable* if they can be executed or interpreted by an environment mechanism so as to support, guide, check, or enforce the described process.

Enactable process descriptions are *different* from traditional (purely procedural) programs, for example:

- they may need to express constraints and proscriptions as well as prescriptions
- they may be enacted, partly, by human agents
- they need to be dynamically modifiable.

Process Themes: slide 15 Ref. No. 5-3-2, 14 September, 1990

PROCESS PROGRAMMING

Process programming (Osterweil) is an influential paradigm that stresses the *similarities* between enactable process descriptions and programs. Potential advantages include:

- We understand how to design programming language–like formalisms, and enaction mechanisms for them.
- We can adapt existing approaches to program specification, design, implementation etc. to the domain of process programs.

Process Themes: slide 16 Ref. No. 5-3-2, 14 September, 1990

PROCESS SUPPORT

Process–centered software development environments must provide, at least, effective support for guiding, checking, enforcing and monitoring effective processes. To do this they must:

- Provide comprehensive support for all aspects of the process.
- Support the coordinated activity of large project teams in ways acceptable to the team members.
- Include mechanisms that allow (and encourage) within–project and between–project process improvement.

Process Themes: slide 17 Ref. No. 5–3–2, 14 September, 1990

PROCESS ISSUES (1)

- Should process models/descriptions emphasize managerial issues or technical issues?
- Should process models be product–centered e.g., 2167a, or activity–centered?
- Are existing formalisms (or adaptations of them) suitable for describing processes; and if so, which formalisms are appropriate? If not, what are the requirements for process description formalisms?
- Should process descriptions be descriptive, prescriptive, proscriptive, constraint–based, or some mixture?

Process Themes: slide 18 Ref. No. 5–3–2, 14 September, 1990

PROCESS ISSUES (2)

- Should processes be strict or liberal?
- Should process enforcement be strict or liberal, and should it be active or passive?
- Should processes (and process support) emphasize top–down discipline or bottom–up collaboration; is a balance (constrained cooperation) possible?
- What architectures and mechanisms are needed for flexible, process–centered environments?
- What mechanisms are needed to support process improvement and evolution?

Process Themes: slide 19 Ref. No. 5–3–2, 14 September, 1990

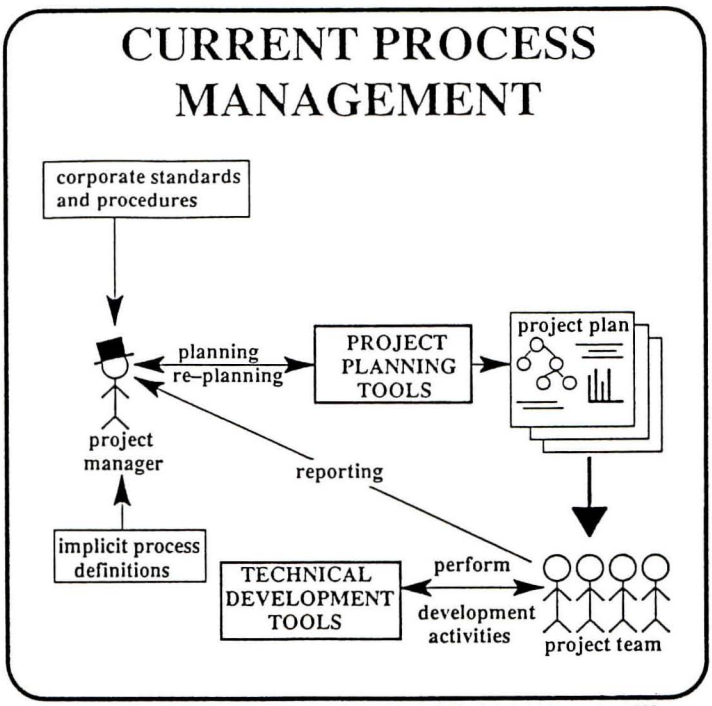
WHAT THE SOLUTION MUST PROVIDE

An improvement in the manufacturing process used by the software industry that enables it to:

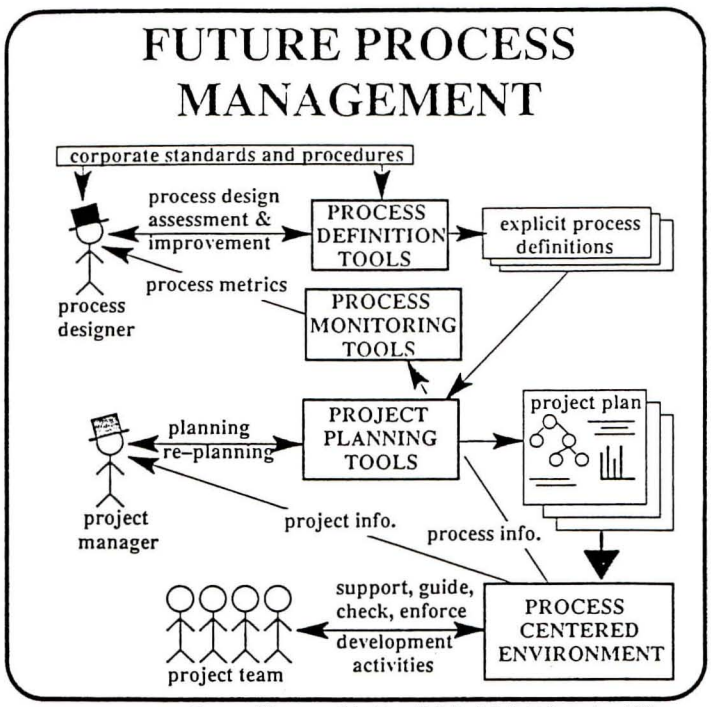
- Produce products of higher quality at lower cost.
- Deliver those products on time and within budget.

To achieve this, we must move from current process management approaches to approaches which support explicit, evolving processes.

Process Themes: slide 20 Ref. No. 5–3–2, 14 September, 1990



Process Themes: slide 21 Ref. No. 5–3–2, 14 September, 1990



Process Themes: slide 22 Ref. No. 5–3–2, 14 September, 1990

CASEの一般認識

～ *CASE Forum in NAGANO* より ～

報告者: 市川　寛(電算)

はじめに

長野にとつては非常に暑く感じられた7月17日、長野駅より歩いて5分のホテル信濃路を会場にして、CASE Forum in Naganoが開催されました。

前回の仙台と同様、地域の情報サービス団体である長野県情報サービス振興協会NISAとの共催とし、参加者33社64名を得ることができました。

プログラムは次のとおりで、パネル討論までにアンケートその1、パネル終了時にアンケートその2を実施しました。

プログラム

09:30～　開会挨拶
09:45～　講演1:「CASEツールの現状」岸田　孝一(SRA)
11:00～　デモンストレーション
　PWB&HyperBook(PFU)
　SAVER(ヴエストソフトウエア)
　Smalltalk 80 &Grapher Gear
　　(富士ゼロックス情報システム)
　StP(SRA)
　Teamwork(東陽テクニカ)
13:45～　講演2:「これからのCASE」熊谷　章(富士通)
15:00～　パネル「CASEの導入と活用」
　司会:佐藤　千明(長野県協同電算)
　パネラー:
　青木　淳(富士ゼロックス情報システム)
　加藤　康人(PFU)
　小林　昇(東陽テクニカ)
　桜井　麻里(SRA)
　近藤　治彦(ヴエストソフトウエア)
　市川　寛(電算)
　コメンテータ:
　海尻　賢二(信州大学)
　岸田　孝一(SRA)
　熊谷　章(富士通)
17:00～　情報交換パーティ

講演内容については、この場では割愛させていただき、アンケート結果およびパネル討論を通じて浮かび上がってきたCASEの現状認識を報告したいと思います。

アンケート

まず、集計結果を提示します。

アンケートその1

Q1. 今回のForumに参加された動機は?
　一般的な知識修得/情報収集のため 35
　CASEツールの導入を検討しているので . 11
　CASEツールを現在使っているので 3
　その他 3

Q2. CASEツールの背後にあるソフトウェア工学技法の普及度について。
(1)次の技法についてあなた自身の知識や経験は?
　構造化分析/設計法
　　よく知っている 8
　　一応知っている 29
　　名前程度は 15
　ジャクソン法(JSD/JSP)
　　よく知っている 2
　　一応知っている 7
　　名前程度は 27
　　知らない 16
　オブジェクト指向設計法
　　一応知っている 11
　　名前程度は 31
　　知らない 11
　プロトタイピング技法
　　よく知っている 1
　　一応知っている 15
　　名前程度は 27
　　知らない 10
(2)これらの技法はあなたの周囲でどの程度普及していますか?
　構造化分析/設計法
　　よく使われている 17
　　少しは使われている 19
　　ほとんど使われていない 15
　ジャクソン法(JSD/JSP)
　　よく使われている 3
　　少しは使われている 2
　　ほとんど使われていない 45
　オブジェクト指向設計法
　　少しは使われている 4
　　ほとんど使われていない 44
　プロトタイピング技法
　　よく使われている 1
　　少しは使われている 10
　　ほとんど使われていない 39

Q3. あなたの周囲では、品質面/コスト面/時間面/技術面でソフトウェア開発のどの工程が最も問題てすか?

品質面
- 要求分析/定義 7
- 概要/基本設計 20
- 詳細設計 6
- プログラミング 3
- テスティング 7
- 保守 5

コスト面
- 要求分析/定義 9
- 概要/基本設計 3
- 詳細設計 7
- プログラミング 11
- テスティング 10
- 保守 10

時間面
- 要求分析/定義 10
- 概要/基本設計 7
- 詳細設計 5
- プログラミング 10
- テスティング 14
- 保守 4

技術面
- 要求分析/定義 18
- 概要/基本設計 15
- 詳細設計 8
- プログラミング 5
- テスティング 4
- 保守 1

Q4. そうした問題を解決する上で何が役に立つと思いますか?

UpperCASE
- 必ず役に立つ 19
- 少しは役に立つ 21
- わからない 9

LowerCASE
- 必ず役に立つ 15
- 少しは役に立つ 22
- 役に立ちそうもない 1
- わからない 10

EDP用4GL
- 必ず役に立つ 11
- 少しは役に立つ 17
- 役に立ちそうもない 2
- わからない 19

エンドユーザー用4GL
- 必ず役に立つ 10
- 少しは役に立つ 17
- 役に立ちそうもない 5
- わからない 17

Q5. CASEツール普及には何が重要だと思いますか?
- より安価なCASEツール/ソフトの提供 .. 29
- 一人1台の端末/WSによる開発環境構築 . 23
- より安価なCASE用端末機/WSの提供 ... 20
- CASEツールの標準化 18
- UpperCASEとLowerCASEの統合 17
- ユーザーの方法論に合わせて カストマイズできるCASEツール 15
- ソフトウェア工学技法の教育、習得 13
- より実用的なソフトウェア工学技法の開発 13
- 国内ユーザーの効果的活用事例 12
- ツール導入に対する技術者の抵抗払拭策 12
- メインフレーム上で使える CASEツールの発達 11
- 大規模プロジェクトでの使用実績 10
- より高度なユーザーインタフェース 10
- ツール導入に対する経営者の説得策 8
- ベンダーの教育・サポート体制 5

Q6. パネル討論で期待する話題は?
- CASEツールはどこまで 実際に使われているのか? 36
- 実際のCASEツール導入時の 問題とその対処策は? 23
- これからのCASEツールへの期待と見通し 23
- 構造化手法は 日本の企業社会、風土に馴染むのか? . 13
- 日本に適したCASEツールはでるか? 13
- CASEツールの選定方法は? 12
- UpperCASE、LowerCASEどちらから 整備してゆくか?、統合化を狙うか 10

技法の普及度をみると、SA/SDについてはかなり浸透しているとみてよいでしょう。しかし、熊谷氏が講演の中で述べられていた『これからはオブジェクト指向だ!』というところのOOA/OODは極一部にしか取り入れられていないのが現状です。

次に、ソフトウェア開発における問題認識とその解決法についての質問には、上流工程における品質面技術面での重要性を認識し、CASEといわれるものへの期待は大きいと答えています。

とはいうもののCASEツールの普及に関しての質問から分かる通り、効果などに対する不安から、ユーザーとして積極的に導入するのではなく、ベンダーにガンバッテほしいということになります。

パネル討論

「CASEの導入と活用」というテーマでスタートしたのですが、少し期待を裏切られる結果となりました。つまり、ベンダーからの事例発表が社内ユースであったり、試行としての導入事例であったり、まだ未発表であったりしたためで、CASEツールを導入するとこういうメリットがあるのですよという明快な答えを得ることができなかったからです。

しかし、CASEツールに対する基本的な姿勢は得ることができたように思います。もっとソフトウェア開発に関する分析/設計技法を習得すること、道具とはないよりあったほうがよいという考えに徹すること、CASEのAはAidedであってAutomaticではないということ、などでした。

ソフトウェアを提供する側が対ユーザーに戦略的情報システムの導入を叫んでいる現在、CASEツールもそのように位置づけるべきだ、もっとドラスティックにソフトウェア作りを変えるべきだ、との意見もありましたが、いままでの膨大なプログラムを保守しなければならない一般SEにとっては、大変なギャップを感じたようです。

もう少しソフトウェア工学や技法を大学や企業内で教育し、CASEを徐々に普及させていく方向が妥当であろうというのが結論でした。

参加者の感想

ここでパネル後のアンケートに書かれた参加者の声をまとめておきます。

アンケートその2

Q1. あなた自身の職場にCASEを導入したいですか?

導入したいが難しい	32
効果	(16)
コスト	(12)
機能	(2)
その他	(6)
当面導入しない	8
効果	(3)
コスト	(3)
ぜひ導入したい(できそうだ)	7
すでに使っている	1

Q2. あなたはCASEの今後についてどう思われますか?

CASEはWSの普及により、徐々に普及していく	19
CASEはCASE自身の改良により、徐々に普及していく	18
CASEは方法論のマスターにより、徐々に普及していく	14
CASEは爆発的に広まり、CASEツールを使わない技術者は一人前といえなくなる	0
ブームは一時的なもので、CASEはあまり普及しない	0

Q3. 今回のForumを終えて、CASEについて感想がありましたら以下に記入してください。

- 第2世代のCASEが実現されると有望な使い方が見えてくるのではないでしょうか。
- CASE導入を勘案する際に、展開できる業務があるかどうか検討する必要がある。ただし、余裕のある企業は設計技法の教育用として購入を考えたら面白いと思う。
- "CASEツール"という言葉は聞いていても、それがどんなものかあまり知りませんでした。システムはますます複雑になってきていると思いますので、早期にCASEツールが導入できる状況になるとよいと思います。(コスト面、社会制度的に)
- CASEツールを導入した場合の効果がやはり気になる。ツールを用いるか否かの前に我々の仕事のプロセス、姿勢など、見直す必要があると思われた。
- 青木氏の話の中によいツールを持つと自分も高められる。そのためそのツールを離したくない"ということがありましたが、ソウトウェア技術者にとって必要なものは、そのようなツールおよびそのように自分を高めたいという意欲であろうと思う。
- 販売状況をきくと、製品として成功しているものは一つもない。それは製品として未完成かまたは社会の要求にマッチしていないと思われる。
- 現状のCASEツールの操作はマニアックであると思われます。一般的にもっと人間工学的な使い易いものを望みます。
- CASEの全体像と今後の発展の可能性を知ることができた。EDP部門に向かない理由は本質的にはなく、要は技術系/文科系人間の新システムに対するアンテナ(情報収集力と消化力)の違いであると思いました。
- CASEが全国的に普及するようにベンダーの方々は頑張ってください。
- CASEベンダーはエンドユーザーにCASEを使ってもらうようにしたらよい。そうするとソフトウェア技術者はあわててCASEを導入せざるおう得なくなるであろう。そのためにも、PC上でのCASEを希望する。

▮CASE自身の課題が多い。ツールとしてどこまで活用できるか、どの分野に適用できるか今後の課題。
▮うーむ、日本って国は変な国だ。
▮現状のCASEはdomain specificなワープロであるという認識を固くした。
▮各フレームメーカがCASE特にI-CASEについて、これからの対応が不明もしくはマチマチであると認識している。CASEツールのベンダーとフレームメーカ双方の努力を要するが、実現性に難ありか。同一WS環境もしくはCASEツールで複数のメーカのシステムの開発が可能な環境ができれば一層導入も促進されるのでは...!?
▮CASEについて勉強不足を感じた。しかし、今後のCASEの進化に期待している。
▮定量的な効果がでてくるようにならないと導入、普及はむずかしいと思う。ユーザがカストマイズできる機能を強化して、開発のプラットフォーム的なものになれば望ましい。
▮Upper CASEに対する期待は残念ながら、まだ使えないという認識が一番です。Lower CASEに対しては各企業等でそれなりの教育ができていると思いますが、Upper CASEに対しては難しく経験に頼らざるを得ない状況です。現状のCASEでは結局事後処理の複雑さは解消できても、設計･分析を手助けできるものではないのではないでしょうか。
▮今後、システム開発をするのに、設計者によりさまざまな技法を使用していたのではむだである。CASEにより、ある程度の統一化が行えるので、今後の開発には十分有効であると思う。
▮CASEはまだまだ進歩すると感じた。
▮CASEはコードの自動生成の方向(Automation)へはあまり熱心になってもらいたくない。CASEはソフトウェアをつくる側(ソウトウェアエンジニア)にとって楽になるものとしてPRして欲しい。
▮構造化分析手法についての理解と実践が重要であることが分かった。CASEツールを単なるお絵書き道具に終わらせないためにも、理論面での現場レベルの統一が必要であろう。
▮自分の知識としてCASEはどんなものかが、少しわかりましたので大変有意義でした。
▮自分自身の現状の仕事とのギャップを感じた。
▮CASEに全てを期待するのは決定的な誤り。プロジェクト毎に適したCASEツールを選択できるような組織的バックアップが必要。(生成の形式が基本部分で統一されていれば、すべて標準化する必要はない)
▮もっと詳しく勉強したいと思った。
▮「果たしてCASEは必要か?」あたりから語られたらいいと思います。
▮具体的な成果などの情報がまだ少ない。

Q4. 今日のForumはいかがでしたか?

講演	
かなり参考になった	27
とても参考になった	16
あまり参考にならなかった	3
デモンストレーション	
かなり参考になった	22
とても参考になった	9
あまり参考にならなかった	15
パネル討論	
かなり参考になった	21
とても参考になった	18
あまり参考にならなかった	4
フォーラム全体として	
かなり参考になった	31
とても参考になった	13
あまり参考にならなかった	0

さいごに

いつもながらのノミニュケーションは前夜/情報交換パーディ/二次会と渡って話もはずみ、おいしいお酒とお蕎麦で過ごしました。

これからも、地方における最新技術の情報不足を補うべく、このような企画を大いに期待し、筆を収めます。

全国縦断 Forum 資料

CASE について考える

岸 田 孝 一
SRA, Inc.

[OHP - 1] なぜ いま CASE なのか？

- Software Engineering
  - ・開発現場でのニーズは？
  - ・技術移転のギャップ！
- Computer-Aided
  - ・現場の開発環境は？
  - ・どんな支援が必要か？

[OHP - 2] ソフトウェア工学

- 1968 年の危機意識
  - ・納期の遅れ
  - ・コストの超過
  - ・品質の悪さ
- そして いま
  - ・システム・テスト時の大トラブル
  - ・原因は上流（要求・設計）工程に？

[OHP - 3] 危機の表層

- いくつかのキイワード
  - ・生産性の向上
  - ・スケジュールの厳守
  - ・製品の品質保証
  - ・要員の質
- とりあえずの解決策
  - ・ウォータフォール・マネジメント
  - ・QC 活動
  - ・さまざまな標準化
  - ・自動化（そして CASE !?）

[OHP - 4] プロダクト指向のパラダイム

- ウォータフォール・モデル
  - ・もの（プロダクト）としてのソフトウェア
  - ・仕様 →｛設計 → 製造 → 検査｝→ 稼働
  - ・開発プロセスとユーザ・プロセスの分離
- 開発方法論とは？
  - ・それは一種のプロダクトである
  - ・だれでも同じ成果が達成できる
  - ・開発プロセスから人間的要素を排除する
  - ・プロダクトの標準ドキュメントを提供する

[OHP - 5] ウォータフォール・モデルの崩壊
- プロセスとしてのソフトウェア
  - ・ユーザ要求は「永久に」確定しない
  - ・開発プロセスはユーザ・プロセスから分離できない
  - ・ソフトウェア開発は「製造プロセス」ではない
    「学習とコミュニケーション」のプロセスである
- 開発方法論とは？
  - ・それが提案され発展してきたプロセスが重要だ
  - ・使い方は人によって異なる
  - ・人間相互の理解を促進する
  - ・プロセスの標準ドキュメントを提供する

[OHP - 6] ソフトウェア工学の発展
- ものを作る技術
  - ・構造化（プログラミング）技法
    - SP, JSP, Warnier, SD, etc.
  - ・アプリケーション・ジェネレータ
  - ・ライフサイクル・マネジメント
- 仕様を確定する技術
  - ・要求分析・設計技法
    - SADT, SA, JSD, etc.
  - ・プロトタイピング
  - ・開発プロセスの見直し
    - スパイラル・モデル etc
- コミュニケーション支援技術
  - ・統合的開発支援環境
  - ・オブジェクト指向
  - ・CSCW

[OHP - 7] ソフトウェア工学の社会的側面
- 一般的現象
  - ・技術移転の遅れ
  - ・現場の保守性（現状肯定主義）
- プログラマ固有の受動的姿勢
  - ・「求めよさらば与えられん」
    - 開発課題はユーザから
    - 開発環境はメーカから
    - 技法やツールは研究者から
- 日本的ソフトウェア工学の悲劇
  - ・経済至上主義 → 生産性パラダイムの重視
  - ・管理至上主義 → 技術者／研究者の疎外
  - ・前近代的大学制度 → 学界の現実逃避

[OHP - 8] そして いま われわれの状況
- HW 環境の整備
  十分 =====

一応 ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
まだ ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
絶望 ＝＝＝

－ CASE の背景にある方法論

知識・経験

十分 ＝＝＝＝＝
一応 ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
少し ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
なし ＝＝＝＝＝＝＝＝

利用度合い

よく ＝＝＝＝
少し ＝＝＝＝＝＝＝＝＝＝＝＝
全く ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

[OHP - 9] 物理的開発環境の変化

－ PC の高性能・低価格化
・アメリカにおける CASE ブームのきっかけ
・ユーザに使えて，なぜわれわれに使えないのか？
－ ワークステーションの進化
・机の上のスーパーコンピュータ
・ネットワークと分散アプリケーション
－ Unix 革命
・（メインフレームの）閉じた世界からの解放
・ツール開発の標準プラットホーム

[OHP - 10] ソフトウェア開発に対するコンピュータ支援

－ 第 0 期（バッチ）
・原始的ユティリティ
－ 第 1 期（TSS）
・アセンブラ → コンパイラ → PDL
・原始的 CASE 群
－ 第 1.5 期（Graphics PC）
・Stand-Alone CASE
－ 第 2 期（PC または WS+LAN）
・Integrated CASE Environment

[OHP - 11] ユーザ・アプリケーションからの教訓

－ まずコンピュータ化しやすいものから始める
・給与計算，経理システム
・現行業務をそのまま乗せる
・関係部門だけで OK
－ 次に中心的な仕事をコンピュータ化する
・販売／生産／ etc
・現行業務の形態を革新する（それが狙い！）
・トップの参加が必要

[OHP - 12] ソフトウェア開発プロセスとは？

- 製造プロセス
  - ・コード & コンパイル & テスト
    - ツールによる自動化
    - 人間的誤りの排除
- コミュニケーション & 学習のプロセス
  - ・要求分析・定義・設計の諸問題
  - ・保守と呼ばれる「進化」
    - 自動化ではない形の支援が必要
    - 人間的理解力の増幅

[OHP - 13] 概念の階層構造

CASE ツール群

△△△

開発支援環境

△△△

開発方法論

△△△

プロセス・モデル

△△△

ソフトウェア開発プロセス

この階層構造は快適か，否か？

（第２回国際プロセス・ワークショップ冒頭での激論）

[OHP - 14] CASE ツールの効用は？

- 開発プロセスを固定しての評価
  - ・使用前・使用後の比較（生産性 etc）
  - ・むかしのアプリケーション開発の考え方
    （コンピュータはサブプロセス最適化の道具にすぎない）
- 開発プロセスの改革を含めた評価
  - ・新・旧プロセスの総合的比較
  - ・いま（これから）のアプリケーション開発の考え方
    （コンピュータはプロセス全体を革新する道具である）

[OHP - 15] 開発におけるコミュニケーション支援

- Lower CASE
  - ・開発者とコンピュータとの間
    - コンパイラ
    - テストベッド
- Upper CASE
  - ・開発者と他の人間（ユーザ，etc）との間
    - データフロー図
    - プロトタイピング

[OHP - 16] どのような支援が望ましいか？

- ハイパーテキスト効果
  - ・さまざまなダイアグラムの利用
  - ・コミュニケーションのバンド幅を広げる
- ソフトウェア・データベース
  - ・統合環境におけるツール・インタフェイス
  - ・ケアレス・ミスの総合チェック
- CSCW
  - ・グループによる「もの」作り
  - ・グループによる意志決定
    - gIBIS（MCC）

[OHP - 17] コンピュータ vs 人間 vs 人間

- ソース・コードにかかわる時間の割合
  - ・TRW の調査（1980）では 25% 以下！
  - ・ソフトウェア開発は OA の一種である
- ピープルウェアの重要性
  - ・ソフトウェア開発上の問題の多くは
    技術的というより社会学的なものである
    （トム・デマルコ）

[OHP - 18] 人はいかにしてプログラマになるか？

- われわれの行動の内的必然性
- 狼少年の教訓
- 人間のすべての社会的活動は自己疎外活動である
  （オルテガ・イ・ガセット）
- 概念の階層化の方向づけ

  ［わたし］→［あなた］→［人びと］

  vs

  ［わたし］←［あなた］←［人びと］

[OHP - 19] ３レベルの開発活動

- 社会的活動
  - ・プロジェクト（の成功／失敗）
  - ・自己疎外のキーワードとしての「生産性」
- 個人間的活動
  - ・設計のよしあし（美的判断）
  - ・グループとしての価値観の共有
- 内的活動
  - ・プログラマの自己発見・成長
  - ・自己沈潜・自己実現のキーワードとは何か？

これからのCASEとオブジェクト指向
(CASE and Object Oriented)

平成2年11月13日
CASE Forum in Osaka

PFU&富士通　熊谷　章

伝統的な開発方法とその欠点、問題点

(開発方法)

グループ作業と組織的管理運営(incorporate direction)
定型用紙ベース(form)
標準仕様／標準作業の順守(acceptance standard)

(欠点)

論理的な開発方法の欠如(lack of formality)
保守性の欠如(lack of maintability)

(解決不可能な問題点)

コスト見積りが正確にできない
提供時期が遅れる
所期の性能が出せず不適合になる
保守ができない
修正のために多くの経費が掛かり手が出せない

CASEの中心テーマ

統合化(Integration)
　マルチプルツールの統合　tailorable_framework
データベース(Database)
　統合化された情報の格納庫
　ERA技術とオブジェクト指向DB(変革の起爆剤)
　Hypertext system
　Metamodeling
標準(Standard)
　ツールの結合のキーポイント
進化のための機能(Evolution)
　全ライフサイクルをカバーする自動化への進化
　第一世代CASEから第二世代CASEへ
エキスパートシステム(SE_KB)
　設計支援と自動プログラミング
品質保障(Quality assurance)
　分析、テスト、信頼性への新しい可能性
　操作と性能の品質保証
　シミュレーションとプロトタイピング

第一世代のCASEの機能と制限

(機能)

方法論の推進とトレーニング
　ツールを介して実現
システム分析ダイアグラムのサポート
　要求と設計を対話型のエディタで可能にした
Single-entry specification bookkeeping
　同一情報格納庫に総ての情報を格納しマルチビュウにする
Reminder and consistency check
　データの無矛盾性や他のコンポーネントとの整合性のチェック

(CASEの制限:使用されていない理由)

ツールは新しい概念のもので学習が必要だ
ツールの価格が高い($5000～$30000／WS)
次の制限がある
　分析用の技術がない
　統合化やモニタリングができず管理が困難
　ドキュメンテーション機能が不足
　グラフィックスのアーチストが必要

第二世代CASE

(第二世代CASEの機能要求)

方法論の適用(methodology adaptation)
　アプリケーションに応じたテイラーリング機能

管理運用機能
　管理者とシステム設計者間の情報交換や管理運営の方法

ドキュメントの出力
　レポート作成言語や印刷機能の装備

知的なダイアグラム作成支援機能
　データエントリからの自動的なダイアグラム作成など

CASEと信頼性工学

検査と分析　構造分析の方法を適用する

品質保証とテスト工程の標準

カバレジテストの実現

動的な情報格納庫

実用的な百科辞典(ソフトウェア設計のための)

ドキュメントの修正が容易に

システム変更のための活性源(encourage to modify)

ソフトウェア工学の今後

ソフトウェア開発:3つのフェーズ

フェーズ1:直観(Intuition)
フェーズ2:仕様の確定
フェーズ3:作成

'85年までの ソフトウェア工学の主テーマ

対象フェーズ:フェーズ2、フェーズ3
技術的テーマ:構造的プログラミング、検証、formal specification
ターゲット :この段階の問題は構造化された問題であり
最適の解答を求めること

'86年以降のソフトウェア工学の主テーマ

対象フェーズ:フェーズ1、フェーズ2
技術的テーマ:ラピッドプロトタイピング、視覚化、エラーの削減
ターゲット :この段階の問題はあまり構造化されない問題であり
最適解ではなく、満足する解を見出すこと

理論的とIAアプローチの比較

問題	理論的アプローチ	IAアプローチ
計算機科学	formal, 数学的	実験的
基本姿勢	明晰な思考	よりよい道具
主なツール	形式的仕様定義	実験的プログラム
プログラム方法論	作成前の検証	誤り修正が設計
問題の性質	well-structured	ill-structured
解の性質	最適解	満足解

OODの概要

ソフトウェアシステムの複雑さ

その原因は対象システムの階層的構造に起因

解決策 ： 分割、 抽象化、階層化の技術

具体的には次の設計技法

表記法、プロセス、ツール

トップダウン構造化技法

データ主導型設計

オブジェクト指向設計

オブジェクト指向のモデルの特徴

抽象化、カプセル化、モジュラリティ

型付け、並列性、持続性

BoochによるOODの表記法

クラス図

論理設計中のクラスとそれらの関係

状態遷移図

クラスの状態遷移図とそのイベント

オブジェクト図

論理設計中のオブジェクトとそれらの関係

タイミング図

オブジェクト図中のメッセージパッシングの順序

モジュール図

クラスとオブジェクトの物理設計でのモジュールへの割当

プロセス図

物理設計にお[illegible]るプロセスのプロセッサへの割当

オブジェクト指向設計のプロセス

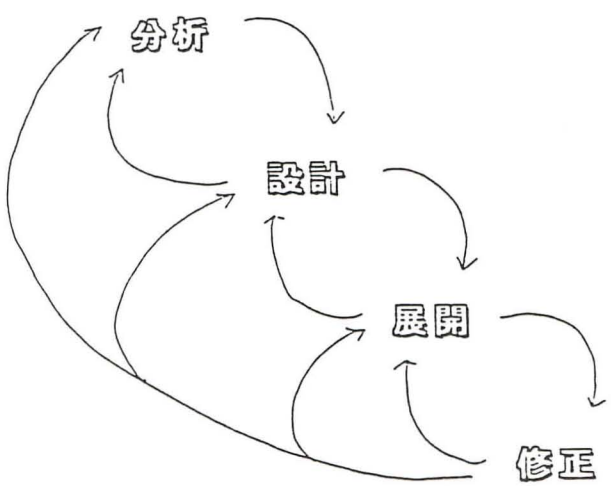
問題記述の中からクラスとオブジェクトを見出す
クラスとオブジェクトのリスト、テンプレート

クラスとオブジェクトの意味の確定
詳細なテンプレート、オブジェクト図、実行可能なプロトタイプ

クラスとオブジェクト間の関係の決定
クラス間の関係、静的と動的な意味の定義
設計の論理モデル、実行可能なプロトタイプ

クラスとオブジェクトのインプリメント
クラスとオブジェクトの実現上の選択、モジュールやプログラムの割当

オブジェクト指向開発のライフサイクル



OODのプロジェクト管理

スケジュールの短縮化
高品質化
資源の節約

OODのためのツール

オブジェクト指向ワークベンチ
ブラウザ
インクリメンタルコンパイラ
ソースレベルデバッカ
構成、バージョン管理
クラスライブラリアン

オブジェクト指向設計の利点

オブジェクト指向プログラミング言語の開拓

ソフトウェアコンポーネントの再利用

変更に対してタフなシステム開発

開発のリスクを減少

人間の認知活動に対して興味を与える

開発期間の短縮

Smaller programs に帰着

オブジェクト指向設計の欠点

性能がよくない

メッセージパッシング、レイヤリング
ダイナミックアロケーション、オブジェクトの再構築

立上がるまでのコストが高い

ツール開発の投資
ドメイン依存コンポーネントの欠如
開発要員のトレーニング

オブジェクト指向設計への移行方法

開発者とマネージャにフォーマルなトレーニング

最初はリスクの小さいプロジェクトに適用

開発者とマネージャをうまく構造化された
オブジェクト指向システムにどっぷり漬ける

設計になれるまでに必な要期間

新しい言語の構文、意味理解	2〜3週間
クラスとオブジェクトの正しい理解と使用	数週間
クラスコンポーネントの設計	数カ月

オブジェクト指向分析技法

対象システムの複雑さを扱うための方法

抽象化(手続、データ)

カプセル化(作成者と使用者の分離)

階層化(共通性の表示)

3種類の組織化(物と属性、全体と部分、クラス分け)

動物の行動の分類(直接原因、進化、生体機能)

種々の分析技法の比較

分析技法	手続きの抽象化	データの抽象化	階層化	組織化	行動の分類
機能分割	x				
データフロー	x				x
情報モデル			x	x	
OOA	x	x	x	x	x

OOA:オブジェクト指向分析

Coadのオブジェクト指向分析技法

オブジェクトの切り出し
データと手続きのカプセル化

構造の見出し
一般化／特殊化のクラス構造、全体／部分の集合構造

サブジェクトの決定
オブジェクトとそれらの間のメッセージコネクション

属性の定義
インスタンスを記述するために用いられるデータ項目

サービスの定義
インスタンスを操作する手続き

OODとビジュアルなシステム設計

次のシステムの研究開発を行った

HyperBookのプロトタイプ作成

ビジュアルプログラミングの作成

イメージデータのパターン認識

ソフトウェア開発技術のモデル記述

動的なシステム記述の作成

PWBの研究開発

PWBとHyperBookの統合

PWBの日本語化と製品化

OODを実践したときの問題点

遭遇した困難さ

1) 日本のソフトウェア技術社会は技術的にも組織的にもツリー構造であり、試行錯誤的な帰納的開発方法に向いていない。

2) システムプログラムの開発システムとして何を採用するか。帰納的な設計開発のためには何か新しいプログラミングパラダイムが必要だった。ーー> Smalltalk

3) システムプログラマの資質が質的に変化
ハードウェア指向 ====> 問題領域指向

4) Smalltalkプログラマの育成と教育

新しいパラダイムOODに必要な技術

帰納的なアプローチとビジュアルなシステム開発のための技術

オブジェクト指向プログラミング環境
マルチメディアアーキテクチャ
データ高速処理アーキテクチャ
Hypertext系技術
アイコンプログラミング
アニメーション／シミュレーション
メタモデリング技術
アプリケーション領域に依存したモデリング技術
収集した情報を操作する道具

新システムの開発とOOD

開発の場の形成

ハードウェア環境 ： ビットマップ付WS1台／人
ソフトウェア環境 ： Smalltalk
開発の理念と方法論 ： 科学的な方法とプロトタイピング技術

プロジェクトの運営

組織形態 ： セミラティス構造
コミュニケーション ： オブジェクト指向のコミュニケーション
分析と評価 ： 計算機のビジュアルな結果による分析と評価

技術力の開発
設計技術, 概念設計力
アルゴリズム ： 考えながら作成
プログラミング技術 ： やりながら学習
技術移転:情報格納庫 ： ワークステーション

OMSの一案(HyperBook)

OMSの機能要求
OMSにおけるデータの特性とその操作

比較項目	OMS	DBMS
データの抽象度	低い	高い
データの意味	利用者に依存	データ構造が表現
データ構造	動的	静的
データの種類	マルチメディア	文字と数値
データのサイズ	～数MB	～1KB
ユーザビュー	可変	固定
スキーマの設計	データ収集後	データ収集前

OMSの機能要求

ビジュアルなマンマシンインタフェース
複数オブジェクトの同時アクセス
ユーザインタフェースのカスタマイズ機能(TOOLBOX)
システム開発のすべてのプロダクトをオブジェクトとして格納
OMSと他コンポーネントとのインタフェース :
　　1) **プログラム呼出し**
　　2) **コマンドレベル呼出し**
オブジェクト間のリレーションの関係付けと操作
オブジェクトのブラウジング機能
リレーションの変換(フィルタリング)
思いどおりに素早く操作できること

OMSの概要(その1)

プロダクト(オブジェクト)の定義
名前、アイコン、属性、他オブジェクトとの関係

プロダクトの産出
クラスから産出し名前、属性などの情報を書き込む

プロダクト間の関係
関係の定義をマニュアルで行う
関係の種類には次のものが考えられる。
参照、等値、部分、分解、組立て、抽象、包含、先者、後者

プロダクト間の関係
関係を対象とした操作
注視の対象を動的に変化させそのリレーションを取り出す

OMSの概要(その2)

プロダクトの種類

常識プロダクト : プロジェクト、キャビネット、本、シート、プログラム、データ

非常識プロダクト : ユーザ定義のプロダクト

プロダクトの操作

各プロダクトの操作はオブジェクト指向型としてプロダクトに付加

操作をするためにはその操作を指定

シートの操作と編集

既存シートへのアクセス、新規シートの作成

OMSの概要(その3)

プロダクトの参照と操作

ビジュアルにかつ高速に探す方法

キーワード、印で検索する方法

属性の内容で検索する方法

インプリメンテーションの見通し

要求される機能	実現するメカニズム
動的構造	オブジェクト指向、lisp
ビジュアルインタフェース	ST-80, ERAエディタ
カスタマイズ	インヘリタンス、メタモデル
カプセル化	オブジェクト指向、Ada
リレーション操作	Hypertext, ERA, MMDB
ネットワーク機能	UNIX, MACH

進化が期待される技術テーマ(その1)

データ構造とファイルシステム
メモリ内と外部装置上でのデータ構造と高速アクセス
Hypertext、拡張ERA、拡張RDB, 意味データモデル、オブジェクト指向

ネットワーク機能
軽い解答 : 既存のネットワーク機能を利用(例、UNIX)
重い解答 : 並列処理型オブジェクト指向システムの開発

各プロダクトの編集作成ツール
各種のダイアグラムエディタ
文章用エディタ
各種のメディアエディタ
描画、イメージ、音声、アニメーション、ビデオ、CGなど

進化が期待される技術テーマ(その2)

管理運用機能(Administration)

1) メンバ間で自由にコミュニケーションできる機能
メール、電子掲示板、ブロードキャスト、電子会議

2) OMSの使用状況のチェック
システムの使用負荷、残余容量、不要プロダクトのチェック

3) 新しい方法論や技法の紹介と学習
教科書、学習書、ノウハウ集、CAL

4) 事例集や事例研究の成果の紹介
過去のプロジェクトを評価分析した結果を集大成

Hypermediaの定義

情報はあるユニットの束である

ユニットの情報はウィンドウに表示される

ユニットはリンクによって関係付けられる

ユニットの創成、編集、関係付けによって様々な情報を表現

Hypermediaは共有されマルチアクセスが可能である

Key Issues : Data model, Direct manipulation

データモデルの問題

1）各ノードの表現にどんなデータモデルが適切か
2）どんなサイズが必要か
3）どんなタイプのノードがあるべきか
4）関係のリンクのソースとしてどんな種類のオブジェクトを用いるべきか
5）リンクのデストネーションはどんな種類のオブジェクトを用いるべきか
6）どんな種類のリンクが必要か
7）リンクは内部構造を持つべきか
8）ノードはどのようにしてより大きな構造にまとめられるか
9）バージョンサポートはどのようにしてやるのか

ユーザインタフェースの問題

1）どんなスタイルのユーザインタフェースを使用すべきか
Direct Manipulation Paradigm
2）ノードはデスプレイ上でどのように表示されるべきか
3）リンクをデスプレイ上でどのように表現するか
4）リンクを辿った時システムはどれ位の速度でレスポンスすべきか
最も重要な技術 ：< 0.25秒／ノード
5）システムはブラウジングをどのようにサポートすべきか
6）データベースのグラフィカルなビューは必要か
7）いかにして "lost space problem" を解決するか
階層スケルトン、ナビゲーションコマンド、高速レスポンス
8）ブラウジングの他の検索の方法として何が必要か
9）ユーザインタフェースをどのようにしてカスタマイズするか

協同作業におけるHypermediaの問題

1）情報はいかに集中され共有化されるか
論理的には一個のデータベース、物理的には分散マルチWS
2）マルチユーザのオーバーヘッドをいかにして少なくするか
ユニットの単位を小さくし衝突を避ける
3）クリティカルナデータへのアクセスの禁止をいかにして実現するか
Read only, Add annotation, Ristricted at all
4）コミュニケーションをどのような形でサポートするか
グローバルなフレーム(ノード)に直接書き込む(メール、BBS)
5）アノテーションのサポートの仕方をどうするか
ユニットに直接コメントを記入、リラックスな規範

Hypermediaの今後の課題

システムの拡張性
Programming Languageが必要(例、Hypertalk)

巨大なデータベース作成が難しい

電子情報から紙情報へのコンバート

Hypermediaの自由度の大きさは利用者には両刃の剣

採用したデータモデルがそのシステムのすべてを決める

人間とコンピュータの交信の問題
ソフトウェアシステムの複雑さは半永久的に増進する
Hypermediaではデータモデルの標準を研究開発することができる
Desktop Metaphorの革新が可能である

Software Symposium '90
基調講演

Cooperative Knowledge-Based Design Enviroments for the Design, Use and Maintenance of Software

Prof. Gerhard Fischer (Univercity of Colorado)

レポータ：中野秀男（大阪大学）

０．はじめに

これから，ソフトウェアの設計･利用･メンテナンスのための，知識ベースに基づいた協調型設計環境」というタイトルでお話しさせていただきます．

はじめに，今日お話することの概要についてですが，まず，現在ソフトウェアが抱えている問題というのが，３０年前，１９６０年代にソフトウェアが抱えていた問題とは違ってきているということについてお話します．次に，ソフトウェアの本質とは何か，何がソフトウェアをハードウェアとは違ったものとして性格づけているかについて，お話します．それから，現在私たちが探求しております研究テーマについて，この１０年から１５年間研究してまいりましたことについて，解説いたします．そして，その間にいくつかのプロトタイプシステムを構築したわけですが，そこから得られた結果について報告します．

ところで，私たちの研究室のアンドレアス･レムケ氏が午後にツール･プレゼンテーションとして，デモを行ないます．これは，Mac Ivory 上で Symbolics で動いているのですが，もし興味がおありになればそちらを御覧下さい．最後に，我々の現在得ている結論，そして今後追求すべき課題についてお話申し上げます．

１．ソフトウェア危機とは？

ソフトウェアの開発というのは，設計もすべて含めて３つの段階に分けて考えることができます．ソフトウェア開発においては，私たちは現実世界における問題というものを持っており，そこから仕様記述を導き出して，それを実際にコンピュータ上で動くプログラムに変換していくわけです．これまで通常行なわれてまいりましたソフトウェア工学というのは，仕様記述からどのようにしてインプリメンテーションに変換すればよいか，というところに注目してまいりました．この仕様記述からインプリメンテーションへの変換は，これまでいろいろな方がいろいろに呼んでおられますが，ある一例としては「down-stream activity」，それに対しまして問題から仕様記述への変換は「up-stream activity」と名付けられています．

１．１　１９６０年代のソフトウェア危機

１９６０年代後半から７０年代初めにかけて，「ソフトウェア危機」と呼ばれたものは，この down-stream activity，仕様記述からインプリメンテーションへの変換の部分に関しての問題でした．この down-stream activity というのは，もし仕様記述がはっきりとわかってさえいれば，この変換の問題というのが問題解決の観点からすれば「well-structured problem」，つまりきちんと整理できる問題であるようなものと考えることができます．それに対しまして，ある現実世界の問題から仕様記述を引き出してくるという変換の部分は，「ill-structured problem」，すなわちきれいに整理することができない問題であるとみることができます．このきち

んと整理できない問題というのは，それを解決すればよいだけではなくて，その前に問題が何であるかを理解して良く整理することが必要であるような，そういう問題なのです．

これまで，down-stream activity の部分の研究対象となっておりましたのは，インプリメンテーションがいかに仕様記述と正しく対応しているかということでした．それに対して，問題と仕様記述間の対応を見る表現については，その表現が正しいかどうかを判断するのは不可能なわけです．なぜなら，問題というものがきちんと定義することができないからです．そこで正しさというよりはむしろ，適切であるとかわかりやすいとか，またそのソフトウェアが使いやすいとか使っていて楽しいとかといった，そういう概念に対応することになります．

ここで私が申し上げたいのは，ソフトウェアが正しいものである必要がないといっているのではなく，ソフトウェアの正しさを見ただけでは不十分である，つまり適切さとか，わかりやすさとかいった概念をも包含したソフトウェアを構築していかなければならない，ということです．

現在では，仕様記述からインプリメンテーションへのフェーズを支援する環境としまして，一般的な目的のプログラム環境，つまり，特別なドメインは意識していない一般的なプログラム環境がサポートされています．私がこの発表の後半で御紹介します「design enviroment」，設計のための環境と申しますのは，この一般的な目的から一歩前進したものです．またこの一般的プログラム環境というのは，「human computer communication」，人間とコンピュータの間のコミュニケーションをサポートするものになります．

これに対して，デザイン環境というものは，もっとドメインに重きを置いた環境，つまり人間と問題領域の間のコミュニケーションを重視するものです．大局的観点からしますと，まず問題を認識してそこから仕様記述を引き出し，それをインプリメンテーションに変換するというのはあまり効果的なアプローチではないのではないか，それよりもむしろ仕様記述とインプリメンテーションを同時に，徐々に発達させていくことがより自然なアプローチであると考えます．

ところで，１９６０年代から７０年代にかけて騒がれました「ソフトウェア危機」というものがもたらした利益，結論というのは，コードを書くこと自体がソフトウェア工学であって，その結果構造化プログラミングという概念が得られたというものです．

１．２　１９９０年代のソフトウェア危機

私たちが現在直面しております「ソフトウェア危機」，１９９０年代のソフトウェアの問題というのはまた違った面を持っております．実験的にいろいろなシステムで研究されている中にも，いくつか「これは本当に大変だ」という問題が，ソフトウェア開発において認識されています．

まず一つめとしては，アプリケーション・ドメインの知識が足りない，ソフトウェア開発者がアプリケーションの知識を持っていない，という点があげられます．この問題は，つまり，ある人が C とか Pascal とか Lisp などのことをとても良く知っていたとしても，書くべきプログラムのアプリケーションの知識を持っていないと，非常に大変だということです．

また，プログラムというものは，仕様記述をインプリメントすればよいというものとは限りません．例えば，ユークリッドのアルゴリズムを書くための仕様というのは，何年もの間変わっていないわけですが，このように仕様さえあればインプリメントできるというものではないのです．ですから，私たちが直視すべき問題は，仕様記述が変わっていったり，以前のものと矛盾を起こしたりといったことに対処できる能力が必要である，ということです．つまり，仕様というものは不変なものではなくて，時間が経つにつれて変わっていくものであるというのが実際の世界で起こっている事実なのです．

最後に問題となりますのが，コミュニケーションとコーディネーションの問題です．まず，コミュニケーションの問題としては，開発者のチーム間，メンバー間のコミュニケーションの問題があります．ソフトウェアは，一人や二人の開発者によって作られるのではなく，多くの場合チームによって作られるためです．また，別のコミュニケーションの問題としては，ソフトウェア開発者と顧客との間のコミュニケーションの問題があります．私たちはこの問題を明確にするために，実際に建築家と話し合ったりしました．建築家も同じ問題を抱えていて，顧客がつくって欲しいものと，建築家が認識している問題との間のコミュニケーションの問題があるわけです．

最後に，このコミュニケーションの問題が，今日私が強調したい最も大きな問題なのですが，ソフトウェア開発者と知識ベースに基づいた設計環境との間の，コミュニケーションの問題があります．この知識ベースに基づいたデザイン環境というのは，そのアプリケーションの知識をも包含したデザイン環境です．

ソフトウェアの設計･開発をするにあたって，建築などの他の分野での設計というものを解析することも必要ですが，と同時に何がソフトウェアに特有の特徴であるのか，何がソフトウェアの本質であるか，ということについても理解しなければなりません．

２．ソフトウェアの本質

最も大きなソフトウェアの特徴としましては,「複雑さ」というものがあります．というのは，コンピュータのソフトウェアというのは，実にさまざまな現実世界のアプリケーションの知識，アプリケーションの世界を反映することができるからです．

二つめに，何がソフトウェアをそんなに複雑にしているのかということに関して，実際にモデル化しなければならない問題と，それをモデル化したソフトウェアのコードとの間の距離が，とても大きいということです．この変換しなければならない距離を縮めるための一つの解決策としては,「階層的構造化」というものがあげられます．すでに申し上げましたとおり，ソフトウェアというものは，外界をモデル化しなければなりません．ですから，ソフトウェアと実際にモデル化すべき外の世界との適合性が，必要となります．,sp .5
私は，あえて，ソフトウェアのメンテナンスというのは，プログラムを進化させることである．外界が変わっていくにしたがってソフトウェア自体も変わっていかなければならない．したがって，ソフトウェアのメンテナンスというのは「保守」というよりも，プログラムを「発展」させていくことであると，このように思うわけです．

そして，もう一つ私がここで主張したいことは，大規模ソフトウェアというのは一から，ソースコードから書かれるものではなくて，今までに構築されたことを発展させてより大きな，大規模なソフトウェアを作っていけるのではないか，ということです．

ですからここで言いたいのは，ソフトウェアというものは「ソフト」でなければならない，柔軟性のあるものでなければならないということです．現在私たちが使っていたり，また生産していたりしているソフトウェアは，実際そんなに「ソフト」ではなくて，他の人が修正したりするのはとても大変なわけです．ですから，後ほど御説明申し上げますが,「end-user modifiability」,エンドユーザがシステムを修正できるということが，結果的にはソフトウェアの再利用･再設計の概念とうまくつながっていくのです．

３．われわれの研究の焦点

ここ何十年間かにわたって，私たちはソフトウェアの本質に迫るべく研究してまいりました．まずはじめにオブジェクト指向型の設計というものから得られた事実として，この設計方法が，ソフトウェアの開発というものが何であるかということを理解するのを，助けるのではないか，と考えたのです．私たちが実験的に得た結論は，

より複雑なシステムは，もしきちんと作られたサブシステムから構築されていれば，より簡単により速くそれを「進化」させることができるということです．そして，interactive なある程度知的なプログラミング環境というものは，優秀なデザイナーがその問題の詳細な部分やつまらない部分を考えずに，大事な部分だけに注目してソフトウェア開発を行なうことを支援してきました．

一方，ソフトウェアの進化というものは，「rapid prototyping」の結果得られると考えることもできるわけですが，それはソフトウェアを組み立てるのではなく，買えばいいのではないか，このことが，ソフトウェアの再利用や再設計という概念につながる，というように考えます．

また，UNIX や Emacs エディタや Lisp システムなどのソフトウェアを見てもわかりますように，それらは一から作られたものではなくて進化した結果得られたものであります．つまり前のバージョンを見て「ああ，ここが足りない」とか「この部分がいけないな」ということで，徐々に改良していって結果的にとても便利なソフトウェアが作られているわけです．

最後に，これが非常に重要な問題なのですが，いくつかの支援システムが現在ある中で，それらに関して，例えば前のバージョンのソフトウェアの説明をしているドキュメントですとかそれらの膨大な情報が，どのようにすれば今ある問題をダイナミックに支援することができるか，そのメカニズムが必要だということです．

４．概念的フレームワーク

私たちは人工知能，それから「human computer communication」，人間とコンピュータがいかに対話しあっていけばよいか，そして「cognitive science」，認知科学というそれら三つの側面から研究を進めてまいりました．そして私たちはいくつかの点に注目したのですが，ここでは，この中の一つ，高機能コンピュータシステムとは何かということについて御説明いたします．

ここでは，Symbolics Lisp マシンを例にとって，高機能コンピュータシステムについて御説明しますが，これはどのシステムについてもいえることで，例えば Sun ワークステーションですとか，他の高機能なコンピュータシステムについても同じことがいえます．現在私たちが使っております Symbolics のバージョンでは，３０，０００余りの関数と３，０００以上のクラスがあり，それらの説明が何と４，４００ページのドキュメントに書かれているわけです．

ですから，高機能コンピュータシステムをうまく使うためには，単にお金があってそれを買えばいいという問題だけではありません．ソフトウェアデザイナーがそのシステムの使い方をまず学ばなければなりません．そしてその後，オブジェクト指向型の設計を使って再利用とか再設計をする場合には，それらの機能をまず見つけて，それを理解して，今度はそれを修正して自分の今解決したい問題に合わせていかなければなりません．

このことについて私たちが実験的にいろいろ研究して得ることのできた事実がいくつかあるわけですが，その中でも大きなものが二つあります．

まず一つめは，これらの高機能システムでは，最も適切に使える機能というものが，しばしば使われていないという事実です．そして二つめの事実としては，もはやこういった高機能システムにおいては，エキスパート，このシステムに関しては彼が専門だという人が存在しなくなるという点です．なぜならシステムを完全にあらゆる側面から，あらゆるレベルから理解することがもはや不可能であるからです．もちろん，

私たちの研究室ですべての問題について考慮しているなんてことは，口が裂けても申しませんから，私たちが現時点では関心を持っていないということをいくつかあげてみましょう．

まず，私たちの研究室では，大規模なプログラムの開発のチームの支援は考えておりません．それからソフト

ウェアプロセス自体の支援にも挑戦しておりませんし，よりよい外部仕様記述をするためにはどうすればよいかということも，考えておりません．時間的制約というものも考慮しておりませんし，ハードウェアの値段ですとか，ソフトウェアの品質やテストについても，現時点では我々はあまり力を注いでおりません．

もちろんこれらの問題点がソフトウェア開発において重要でない，といっているわけではありませんが，私たちがこれらの点に注目していないことにはいくつか理由がございます．

まず，なぜ私たちが大規模なソフトウェア開発チームの支援を考えていないか，ということですが，いくつかの今注目すべきソフトウェアというのは，大規模チームからは開発されていない，という事実です．例えばSpreadsheets ですとか UNIX ですとか Smalltalk などのシステムは，とても小人数で開発されていたり，時には個人によって開発されたものです．

それから，ソフトウェアプロセスの管理についてですが，これは，もし大規模なプログラム開発が行なわれているのでなければ，あまり重要な問題ではないのではないか，もっと管理的オーバーヘッドにつながる問題であって，大規模ソフトウェアに特有の問題であると考えます．

次に，よりよい外部仕様記述を書くためにはどうすればよいかということについて関心を持っていないことの理由ですが，例えばアメリカの防衛省では，とても大規模な外部仕様記述を用いたソフトウェアを作っているわけですが，この問題に関する critical な問題というのは，それがドキュメント本位のソフトウェアであるという仮説に基づいているという点です．まず仕様記述を行なってそれからソフトウェアを作るというアプローチであれば，この点は重要になるのですが，私たちが考えております仕様記述とインプリメンテーションがともに進化していくという側面からは，この問題はあまり重要ではなくなってきます．

時間的制約という点に関しましては，私たちの研究室では「研究」をしているのであって，コマーシャルプロダクトなどを作っているのではございませんので，この研究があとどれだけかかるとか，そういう時間的制約に振り回されることはあまり現実的ではありません．

また，技術的進歩にともないまして，ハードウェアの値段というのもどんどん下がってきているわけで，これも致命的な問題にはならないと思います．

最後に，ソフトウェアの品質とテストに関する問題ですが，この分野は日本のソフトウェアの研究者たちの間でとても進んでいる分野だと思いますが，ここでの問題というのは先ほど申し上げました「down-stream activity」，仕様記述からインプリメンテーションへの変換という分野に関してのソフトウェアの品質とテストに対するものであり，私たちが注目しています「up-stream activity」についてはこの分野は難しいのではないかと思います．

ここまでお話しましたのが，私たちが行なっております研究の概念的フレームワークです．

５．われわれが開発してきたシステム

次に御紹介しますのが，私たちが提案します新たな問題解決へのアプローチとしての「知識ベースに基づいたデザイン環境」ですが，なぜ私たちがこういう環境を提案するに至ったか，そしてこれからどうしていこうと考えているのかを，歴史的にこの１０年から１５年の間行なってきました研究に沿って御説明します．

ここで一つだけ申し上げておきたいのは，オブジェクト指向型の設計というものが，そんなに目新しいものではないということです．Smalltalk というのはもう１０年も前から存在しているものですし，重要でないということではないのですが，Object-Oriented Design 自体はそんなに目新しいものではありません．したがって私たちの研究というのは，

５．１　ドメイン指向システムの研究まず general purpose，一般的目的のためのプログラム環境ではなく，ドメインごとの，ドメイン指向型の「Construction Kit」,「組立キット」ともいえますが，これを考えることから始まりました．

私たちはまず，次のような組立キットを持つシステムを研究することから始めました．それは，一つは作曲をするための組立キット，またもう一つはピンボールマシンを作るための組立キットです．私たちは，これらのシステムがどのように使われているかを観察した結果，人間がやっていることは単純なプログラミングであるということがわかりました．

これらのシステムはもはや，一般的なプログラム環境ではなくて，ドメインに依存したプログラム環境です．そして，これらのドメインごとの環境というのは，一般的なプログラム環境よりもずっと便利になっているわけです．例えば，C や Lisp などの一般的なプログラミング言語を使って，ピンボールマシンを設計しようとすると，本当に大変だろうということは想像に難くないと思います．それに対して，このようなドメインごとの組立キットを使えば，とてもプログラミングが簡単になります．私たちはこれらのシステムを「ドメイン指向型の組立キット」というように呼んでいるわけですが，そこで，ある一つのとても重要な，致命的な問題を見つけました．

それはこれらの組立キットは組み立てるためにはとても便利なわけですが，今行なわれているデザインの品質についてフィードバックが返ってこない，ということです．つまりシステムからの返答がないというのが致命的な問題なわけです．

５．２　建築設計支援システム

そこで私たちは，デザイン環境というものを考案いたしました．これは組立キットに加えて，「critic」，批評という要素を付け加えたものです．この「批評」というものが，今行なわれているデザインの品質について，フィードバックをソフトウェア開発者に与えます．私たちの研究室では，建築設計という側面に関して，いろいろなプロトタイプシステムを作ってきているわけですが，ここで，私たちがデザイン環境，設計環境というものについて行なっております研究の例をいくつか御紹介します．

まずはじめは，台所の建築設計に関するシステムの例です．このシステムの機能については後ほど詳しく御説明いたします．また，私たちの研究室のアンドレアス・レムケが午後お見せしますプロトタイプは，ユーザインターフェースを設計するためのデザイン環境です．ここでは詳しくこのシステムについてお話しませんが，もし興味がおありでしたらぜひ午後のセッションに御参加下さい．他には，Symbolics のプレゼンテーションタイプの定義のためのデザイン環境があります．これも詳細についてはお話しませんが，私たちの研究を charactarize するものです．

今ここでお話していますアイデアというのは，別に研究室の中だけのものではなく，実際の企業の中でも使われ始めています．私たちの研究室に以前おりました学生が NYNEX という（これはアメリカの NTT のようなところですが）企業に行きまして，COBOL プログラミングのためのデザイン環境を今構築中です．そして，ちょうど私たちが今回新しく始めたリサーチなのですが，これはネットワークの設計に関するデザイン環境を構築するものです．

では，台所の建築設計をドメインにとりましたプロトタイプシステムについて，少し詳しく御説明申し上げます．

まず，これは先ほどの二つの組立キットとほぼ同じ機能を持っておりまして，ドメインに応じた台所設計を行ないます．台所のドメインに応じた要素がパレットの中にあるのですが，それをワークエリアの中で組み立てて

いくことができます．そして，組立キットとこのシステムとの決定的な違い，とても重要な発展箇所というのは，このようなメッセージが，今ワークエリアにあるデザインの品質を，システムがドメイン知識を用いて評価した結果が「批評」として表示されるという点です．ですから，システムが，どういうデザインが良いデザインで，どういうデザインがあまり良くないデザインであるといった，そういう知識を持っているわけです．

私たちは最初，このように批評した結果を表示するだけで，設計者が「ああそうか，ここがダメなのか」というふうに理解できると考えておりました．ところが，そのように批評した結果のメッセージを，時には設計者が理解できないということがあることがわかりました．

そこで私たちは次の一歩としまして，デザイン環境から統合化したデザイン環境へと移行していきました．それは，批評のメッセージが，組立キットの中で組み立てている構造的な台所の設計図と「argumentation」，論議的部分とをリンクするというものです．argumentation とは何かといいますと，どうしてこのデザインが良いのかとか，どうしてこのデザインが悪いのかといった論理的根拠の知識を知識ベースとして含んでいる部分です．論議的設計部分をサポートしたシステムの方では，なぜこういうデザインが良いのか悪いのかといった論理的根拠が表示されます．そして，もう一方の「CRACK」という名前の組立設計の部分からこちらの argumentative な設計へのリンクを，私たちは「critic」，批評というメッセージを通して実現することができると考えます．

例えば，CRACK システムの方でガスレンジがドアのそばにあるから良くない，という批評が表示されるようになっています．この批評はいったいどういうことかというのは，論議的設計をサポートするシステムで説明されます．また，この論議的要素の部分はハイパーテキストあるいはハイパーメディアシステム上で構築されています．

さらに，このシステムはある設計方法論に基づいています．論議的部分での方法論というのは，設計にはこれが最適であるという解決法は，一つに決まらないというものです．つまり，長所・短所をそれぞれの設計が持っていて，どうしてここが良くて，どうしてここが良くないのかということの説明をこのシステムが行なうのです．また，組立設計をサポートするためのシステムについてですが，これには二つ方法があります．一つは，それぞれのデザイン要素を集めてきて組み立てていく方法，もう一つは，カタログにある以前に設計されているデザインをとってきて，それを修正して新たにデザインしていく方法です．

さて，先ほどお話しましたように私たちはいくつかのプロトタイプシステムを作っているわけですが，それらの統合化環境の一般的なアーキテクチャについても検討いたしました．そして，そのアーキテクチャの要素のリンクについて私たちは次のように考えました．それは，論議的デザインというのはその性質上とても抽象的なので，たいがい文章で表されたとても概念的なレベルでの設計論議しかできないので，カタログに保持されている具体的な設計例を用いれば，その論議的部分をイラストレートすることができる，ということです．

私たちの研究の最終目的と申しますのは，このアーキテクチャの中のそれぞれのリンクがいったいどういうものであるのか，ということを分析し理解することにあります．例えば，「EXAMPLE FINDER」というのは，組立キットの部分からカタログの間のリンクについて分析するためのプロトタイプシステムです．また，明日私と中小路さんとで共著しました論文を発表するのですが，それは仕様記述の要素の部分とカタログの部分のリンクについて考案したプロトタイプシステムです．

６．ソフトウェア工学と知識工学

ここまで，今私たちが具体的に行なっておりますプロトタイプシステムについて御説明申し上げたわけですが，今度は，最初に申し上げましたもっと概念的なレベルの話に戻りまして，私たちがソフトウェア工学に関し

てどのように対処していけばよいかについて見ていきたいと思います．

１９５０年代のソフトウェア工学というものを考えますと，ソフトウェア開発者というのは，解かなければならないいろいろの問題をアセンブリ言語にまで落とさないといけませんでした．１９５０年代には，このアセンブリ言語から実際の解決すべき問題との変換距離がとても大きかったということになります．

１９６０年代になりますと，ハイレベルなプログラミング言語が出現してまいりました．これは FORTRAN とか Lisp とか ALGOL などの高機能言語ですが，その結果として，プログラム開発者が二種類できてきました．一種類の人たちは，解決しなければならない問題をプログラミング言語で書く人たち，そしてもう一種類はそのプログラミング言語に関するコンパイラを作成する人たちです．

私は，知識ベースに基づいた設計環境というものを提案することで１９８０年代に入って新たな概念が入ってきたと思います．１９８０年代に入ってからのソフトウェア工学の構造というのは，ある一つの特別な問題領域に注目した場合，それに関する組立キットというものがそこに出現し，その結果として今まで三層に分かれていたのが今度は四層という新しい階層構造となっているわけです．

そして，先ほど申しましたようにその組立キットに加えて，設計中の設計の品質に関しての知識を持っているような統合化環境が必要である，ということになるのです．これが１９９０年代のソフトウェア工学になると思います．ですから，近い将来ソフトウェア開発者はまた新たな二種類の人たちに分かれていくと思います．一方は，デザイン環境を構築するためのソフトウェアデザイナー，そして新たに，ある特別な問題をそのデザイン環境にマッピングするドメインの専門家というのが出現してくることになると思います．

このようにして，デザイン環境というものが現れたことによって，変換しなければならない距離というのがとても短くなったわけです．１９６０年代には，問題からアセンブリ言語にまで変換しなければならなかったのが，今やデザイン環境と問題の間の短い距離を変換させる，マッピングさせるだけで良くなりました．ですから，最初に申し上げましたアプリケーションの知識が不足しているというソフトウェア開発の問題点は，このようにドメインの専門家が一部ソフトウェア開発を担うことによって解決されていくのではないかと考えます．

７．エンドユーザの問題

ここまで，私たちが研究してまいりました歴史的経過を見てきたわけですが，最後に，これから私たちが行なっていこうと考えている研究課題である「end-user modifiability」，すなわち，エンドユーザが自分に合わせてカスタマイズできるようなデザイン環境の構築について御説明します．前に申し上げましたように，現実世界というのは常に変わっていくわけです．これによって，すでに解決済みの問題も，現実世界が変わるにつれて変わっていきます．ですから，新たにシステムのいろいろな部分をユーザが拡張・修正できるようなデザイン環境を構築する必要があるのです．

この概念についてもう少しわかりやすく説明するために，先ほどの台所設計の「JANUS」というシステムを用いて御説明します．例えば，あるユーザが，このシステムにあるデザインの設計エレメントを含んだパレットの中に新しく「電子レンジ」という要素を入れたいと考えたとすると，この場合，新たな拡張が必要になります．そこで問題となるのは，誰がその拡張をすればよいのかということです．自然な考え方としては，その拡張はデザイン環境を設計したソフトウェア技術者が行なえばよいのではないか，ということになりますが，私たちが目標とするのはそうではなくて，ドメインの専門家がそれを行なえるようなしくみを提供できないか，というものです．

そこで私たちは，デザイン環境のコードなどの詳細な部分，機械に依存したような部分にわずらわされないで，新しく「電子レンジ」という概念をシステムに取り入れるための仕組みは，どのようにすればよいかについて，研

究を進めています．今から御紹介しますのが，私たちがインプリメントしました拡張されたプロトタイプシステムで，これはエンドユーザが修正できるようなデザイン環境を構築しております．これについては４月にシアトルで開かれました「Human Computer Interaction」という Conference で，論文を発表しましたのでそちらの方を御覧下さればと思います．またこのシンポジウムの予稿集にも，もっと詳しく書いておりますので御覧下さい．

さて，電子レンジを新しいオブジェクトとしてシステムにわからせるための仕組みですが，今「MICROWAVE」(電子レンジ)という新しいクラスを定義したとします．すると，ではスーパークラスは何にしますか，というふうにシステムが聞いてきますので，ユーザはそれでは今いったいどういうクラスがありますかとシステムに尋ねると，システムが，こういうクラスがありますということで，システムが現在すでに持っているクラスのオブジェクトを示します．ところで電子レンジというのはどちらかというと「加熱器具」ですから，ガスレンジの分野に入るのではないか，と考えたとします．またそこで，そのガスレンジというのはどういうものかということで，今度はガスレンジに関しての知識ベースに入っていますルールの一覧表を見ることができます．その中のある一つ，例えばレンジと冷蔵庫の間の関係のルールを選びますと，その内容が具体的に表示されるわけです．

しかしこれでは，新しいルールを定義する場合に，たとえば Lisp の S 式で書いたりといったようにまだとても形式ばった書き方をしなければなりません．そこでここに示したのは，もっと自然にルールを定義することができないか，という点が問題となります．例えば，現時点のシステムでは，冷蔵庫のドアを壁と向かい合わせにしても何も文句を言ってきません．なぜなら，そのようなルールがまだ記述されていないために，システムがこのような設計も良い設計だとみなしてしまうからです．

ですから，より自然に新しいルールをユーザがシステムに対して定義する方法としては，これが良い例で，これが悪い例だということをシステムに示すと，システムが「ドアが壁側にあったらダメなんだな」というふうに推測し，新しいルールとして定義できる，といった感じになるのではないかと思います．

ここで，私たちが推奨しているデザイン環境のすべてに当てはまる一般的な階層的アーキテクチャについて説明します．エンドユーザが修正できるという「修正性」というのは，一番上のレベルであるキッチンの図面から一番下のレベルの Lisp コードにまで変換せずに，キッチンデザインのレベルにまで変換さえすればよい，というようなメカニズムを提供することができないかというものです．ですから，私はこの発表のまとめとしまして，メンテナンス，「保守」という面に特に注目して述べたいと思います．

ここで使っております「ユーザ」という言葉は，単にシステムを使う人という意味ではなくて，アプリケーションの知識を十分に持っている，そして問題が何であるかということをわかっているような人を指しています．これまでのソフトウェアに対する見解というのは，ソフトウェアの設計と保守はソフトウェア開発者によって行なわれるというものでした．それに対して私たちのソフトウェアに対する見方というものは，ユーザは単にソフトウェアを使うだけではなくて，それを進化させることができなければならないというものです．これをエンドユーザの「修正性」というふうに呼んでいるわけですが，今回の発表の最初に申しましたように，モデルにしている世界は変わっていき，それに応じて使用や要求や定義も変わっていくわけで，ユーザがその変化に対処できなければいけないのです．

いろいろな実験や研究者間の分析の結果，ソフトウェア開発にかかる費用の６０％は保守に使われていることが明らかになりました．また，その保守にかかる費用のさらに７５％はプログラムの改善に使われています．つまり，単にデバッグしたいとか小さなミスがあるとかではなく，７５％は外界の変化に伴う改良をソフトウェアに要求しているわけです．

ですから，私が今回の話を通して一番言いたいことは，このプログラムの進化は，ユーザによって最も適切にサポートされるべきであるということです．

８．結論

最後にまとめとして，ソフトウェアの開発者は協力的な問題解決システムによって初めて力を与えられて，より良くソフトウェアを開発していけるのではないか，そしてそのシステムというのは統合化された，そして多面性を持ったドメイン指向型の知識ベースに基づいた設計環境によって実現されるのではないか，ということが今回の私の結論であります．

akancでのプログラミング入門（第1回）

新田稔（株）ＳＲＡ

まえがき

1988年の9月から、(株)ＳＲＡと大阪大学基礎工学部との共同研究プログラムで、同学部情報工学科の鳥居研究室にお世話になっている。現在の共同研究のテーマは、ソフトウェア開発プロセスを記述する関数型の言語とその処理系の開発、その応用、および評価である。私自身、ここで初めて「関数型言語」に触れることになった。プログラミングを始めてから十数年間、ずっと

- ●プログラムとは計算機への命令を実行して欲しい順に並べたもの
- ●変数とは値を入れる箱で、入れた値は後で何回でも見ることができる、ただし新しい値を入れると前の値は壊されてしまう

という手続き型プログラム・モデルだけを使っていた私の、関数型言語に対する第一印象は、「それで実用的なプログラムが作れるの?」であった。現に、関数型言語でソフトウェア開発をやった、という話は聞いたことがない。

しかし、まわりには関数型言語がいいといっている人が何人かいる。そこで、(1)関数型言語の勉強のためと、(2)本当に関数型言語が優れた言語であり、にもかかわらず実際のソフトウェア開発で採用されていないのは、言語の構文や処理系に何か欠陥があるのではないか － だとしたら実世界で使える関数型言語を作ってみよう、との考えから、自分で言語と処理系を設計・開発してみることにした。それがakanc(言語)とakx(処理系)である。

akanc(「アカンシー」と呼んで下さい)というな前からわかるとおり、最初は冗談半分であったが、そのうちちょっとハマッテしまい、研究会やシンポジウムで発表してしまった。関数型のプログラムは、あれ(ある関数)はこう(その定義式)である、という合同関係なるものの羅列であり、ある処理をこの順序で行なえ、という命令の記述は一切ない。プログラムの実行は、プログラムのどこかにあれがあればそれをこう書換える、という項の書換えで進む(「実行」という言葉も似つかわしくないかもしれない)。値の保存場所としての「変数」もなく破壊的な「代入」もない。この世界では、「変数」や「代入」は代数的な意味で使われている。

そういう関数型言語を、初めてコンピュータ・プログラミングを習う人に教えたら、彼/彼女はどんなプログラマになってしまうのか、というのが、いま私の関心事のひとつである。ところが、関数型言語について書かれたものをいくつかの見たが、どれも手続き型言語との比較や(私にとっては)難しい代数の話しが出てきたりして、プログラミングの初心者に向いているものはなかった。そこで自分で「akancでのプログラミング入門」を書いてしまうことにした。どこにも出すところがないので、SEA-Mailへ投稿する。

なお構文の詳細などの資料や処理系は、請求いただければお送りします。請求方法は

電子メイル nitta@sra.co.jp
または
大阪府豊中市待兼山町1-1 大阪大学基礎工学部情報工学科 鳥居研究室 新田稔

までお問合わせください。

1.やさしい入門

1.1.てはじめに

コンピュータに何か仕事をさせるには、まずプログラムを作らなければならない。プログラムとはコンピュータに依頼する仕事の内容を書いたもので、これを書くときに使われる言語がプログラミング言語である。現在、非常に多く種類のプログラミング言語があるが、本書では、akancという言語でのプログラムの書き方を解説していく。なおテキスト・エディタやその他のUNIXのコマンドについてはここでは解説しないが、まわりの専門家たちに尋ねればそれらの使い方を親切に教えてくれるだろう。

さて、プログラミング言語でプログラムを書くだけでは、まだコンピュータに仕事をやらせることはできない。それはコンピュータがプログラミング言語を直接には理解できないからで、プログラムをコンピュータに理解できるよう翻訳するインタープリタが必要になる。プログラミング言語によってそれぞれ専用のインタープリタが使われるが、akancにはakxというインタープリタがある。

では、次のプログラム1を「hello」という名前のファイルの中に記述してみよう。

```
main(state s) == printf(s,"こんにちは¥n"):external ;    /* プログラム 1 */
```

このプログラムは、「こんにちは」という単語を出力するプログラムである。プログラムが書けたら、次にUNIXのシェルから

```
akx hello
```

とタイプすると、akxがhelloの中に書かれたプログラムの翻訳を開始する。文字を入れ忘れたとか綴を間違えたとかいったミスがなければ、コンピュータが仕事をして、

```
こんにちは
```

が出力される。プログラムがどのようなものであろうとも、この、プログラムをファイルの中に記述して、それをインタープリタを通してコンピュータに仕事をさせる、という手順は変らない。プログラムを記述するファイルの名前には、好きなものを使ってよい。

次にakancプログラムがどういうもので構成されるかを説明しよう。それにはいくつかの特殊な言葉を用いなければならない。以降そのような言葉は「」で囲んで示すが、それらが何を指すのかは読み進むうちに明らかになってくるだろう。最初にその言葉が出てきたときには、そういう名前で呼ばれているものがある、ということだけわかっていればよい。

さて、ひとつのakancプログラムは、その大きさはどうであれ、行なうべき仕事を指定するひとつ以上の「関数の定義」で構成される。関数の定義は、

```
定義する関数の型名　定義する関数名 （ 仮引数の宣言 ） ==　定義式 ;
```

の形をしている。「定義する関数の型名」は、必要のないときには省略できる。さらに「仮引数の宣言」は、

```
仮引数の型名　仮引数名
```

の形をしている。仮引数の宣言が二つ以上あるときには、それらを「,」で区切って並べる。

「定義式」は、「定数名」や「仮引数名」が単独か、あるいはそれらを「演算子」で組合わせた「演算」やそれらを「実引数」として用いた「関数の参照」である。関数の参照は、

```
参照する関数の型名　参照する関数名 （ 実引数 ）　ブロック指定
```

の形をしている。「参照する関数の型名」は、必要のないときには省略できる。実引数が二つ以上あるときには、それらを「,」で区切って並べる。参照する関数は、それが「組込関数」か

「外部関数」である場合を除いて、その定義がプログラムのどこかになければならない。「ブロック指定」は、

: ブロック名

の形をしているが、必要のないときには書かない。

これらの知識を仕入れてプログラム1を眺めなおすと、それはひとつの関数の定義で構成され、その内容は次のようになっている。

```
定義する関数の型名    省略されている
定義する関数名        main
仮引数の宣言          state（仮引数の型名） s（仮引数名）
定義式                関数の参照
    参照する関数の型名    省略されている
    参照する関数名        printf
    実引数                s（仮引数名）          "こんにちは¥n"（定数名）
    ブロック指定          : external（ブロック名）
```

ここでいくつか注意すべき点がある。

- 「/*」と「*/」で囲んで、プログラムに関するちょっとした説明や覚書などを書くことができる。これは「コメント」と呼ばれ、何を書いてもプログラムには影響がない。
- 関数名「main」は特殊な関数名で、その名前を持つ関数は「メイン関数」と呼ばれる。
- 「"」とそれで囲まれた任意個の文字の列は「文字列型」の定数を示す定数名であり、「¥n」は改行を示す特殊な記号である。
- ブロック名「external」は、参照する関数（この場合は「printf」）が外部関数であることを示す特別な「ブロック名」である。

1.2.関数の定義の意味

関数の定義は、(1)定義式の書換えの指定と、(2)関数の計算の指定、という二つの意味を持っている。インタープリタakxは、プログラムを与えられると、まずプログラムが文法どおりに書かれているか、必要な関数はすべて定義されているか、などを調べる。

もしプログラムが完全なら、次にメイン関数の定義式の「書換え」を行なう。組込関数や外部関数の参照、演算子による演算は、言語規則の定めに従って書換えられる。プログラマが定義した関数の参照は、その関数の定義式中の仮引数名を対応する実引数で置き換えた式に書換えられる。この書換えの指定が、関数の定義のひとつの意味である。メイン関数の定義式の書換えが終った後その中にプログラマが定義した他の関数の参照があったなら、その関数の定義式も書換えられる。これが繰返されて、必要なすべての関数の定義式の書換えが行なわれるのである。

すべての定義式の書換えが終ると、インタープリタakxはメイン関数の値を計算し始める。メイン関数の値を求めることがakancプログラムの最終目的である。メイン関数の値を求めるためには、その定義式が計算される。この計算の指定が、関数定義のもうひとつの意味である。組込関数や外部関数の参照、演算子による演算は、どのような値になるかが言語規則で定められている。プログラマが定義した関数の参照の値は、実引数を仮引数に「代入」した定義式を計算することにより求まる。

ここで、定義式の書換えと関数の計算を説明するために、ある数の階乗を求めるプログラムを例に取り上げる。ある数の階乗とは、1からその数までを掛け合わせた値で、たとえば3の階乗は、1掛ける2掛ける3、すなわち6である。ただし、0か0より小さい数の階乗は1であるとする。

```
fact(int n) == if n > 0 then fact(n - 1) * n else 1 ;    /* プログラム 2 */
```

プログラム2は、与えられた数が0より大きければ、その数より1小さい数の階乗を求めてそれにもとの数を掛けてその数の階乗とする、そうでなければ1をその数の階乗とする、となっている。

ここで、「if A then B else C」は「Aが真ならB、偽ならC」を示す選択の演算、「A > B」は「AがBより大きいことの真偽」を示す比較の演算、「A - B」は「AからBを引いた値」を示す減算の演算、「A * B」は「AにBを掛けた値」を示す乗算の演算、である。このような「if then else」、「>」、「-」、「*」を演算子、A、B、Cを演算子の「項」と呼ぶ。また「fact(n - 1)」は関数の参照であるが、このように定義式の中で定義する関数自体を参照することもできる。このような関数の定義を「再帰的な関数の定義」と呼ぶ。

1.3.定義式の書換え

さて、プログラム2は、階乗とはどのようなものかを記述しただけで、完全なプログラムではない。ある数、たとえば3の階乗を求めて出力しようとすると次のようなプログラムを作らなければならない。

```
main(state s) == printf(s,"%d¥n",fact(3)):external ;     /* プログラム 3 */
fact(int n) == if n > 0 then fact(n - 1) * n else 1 ;
```

これはプログラム2に、メイン関数の定義を加えたものである。

定義式の書換えは、次の規則に従い、メイン関数の定義式から始められる。

(書換え規則-1)組込関数、外部関数の参照
　すべての実引数について書換えを行なう。
　実引数が複数あるとき、それらは同時に書換えられる。

(書換え規則-2)プログラマが定義した関数の参照
　(a)まずすべての実引数について書換えを行う。
　(b)次にその関数の定義式中の仮引数名を対応する実引数に置き換えた定義式で
　　関数の参照全体をを書換える。
　(c)さらに書換えた式について書換えを行なう。

(書換え規則-3)項の評価順序がない演算
　(a)まずすべての項について書換えを行なう。
　　項が複数あるとき、それらは同時に書換えられる。
　(b)それらがすべて定数名に書換わったなら、演算子ごとに定められた規則に
　　よって、演算全体をひとつの定数名に書換える。

(書換え規則-4)項の評価順序がある演算

　(a)まず、最初に評価される項について書換えを行なう。
　(b)その項が定数名に書換わったなら、演算子ごとに定められた規則によって、
　　演算全体をひとつの定数名か、あるいは、次に評価される他の項に書換える。
　(c)さらにそれについて書換えを行なう。

(書換え規則-5)仮引数名、定数名
　それ以上の書換えは行なわない。

プログラム-3のメイン関数の定義式は「printf(s,"%d¥n",fact(3)):external」という外部関数の参照である。書換え規則-1と-5から、まず「fact(3)」の書換えが行なわれ、書換え規則-2の(a)(b)から、メイン関数の定義は、

```
main(state s) == printf(s,"%d¥n",if 3 > 0 then fact(3 - 1) * 3 else 1):external ;
```

と書換わる。次に書換え規則-2(c)により「if 3 > 0 then fact(3 - 1) * 3 else 1」を書換えることになるが、「if A then B else C」には「まずA、次にBかC」という項の評価順序がある。したがって書換え規則-4(a)により「3 > 0」を書換えることになる。

「A > B」には項の評価順序がないので、「3 > 0」に対して、書換え規則-3(a)、-5、-3(b)を適

用する。演算子ごとに定められた規則は、

（書換え規則-「A > B」）
Aが Bより大きければ真を表す定数名(「true」)に、
そうでないなら偽を表す定数名(「false」)に書換える。

であるので、「if 3 > 0 then fact(3 - 1) * 3 else 1」は「if true then fact(3 - 1) * 3 else 1」となる。次に書換え規則-4(b)を適用するが、演算子ごとに定められた規則は、

（書換え規則-「if A then B else C」）
Aが真を表す定数名ならBに、偽を表す定数名ならCに書換える。

であるので、「if true then fact(3 - 1) * 3 else 1」は「fact(3 - 1) * 3」となる。書換え規則-4(c)からまたこれを書換えるが、「A - B」、「A * B」ともに項の評価順序がない。「A - B」の演算子ごとに定められた規則は

（書換え規則-「A - B」）
AとBの差を表す定数名に書換える。

であるので、これは「fact(2) * 3」となる。ここまでで、プログラム-3のメイン関数の定義は

```
main(state s) == printf(s,"%d¥n",fact(2) * 3):external ;
```

と書換わっている。さらに「fact(2)」対して書換えを進めていけば、「fact(1)」、「fact(0)」についても書換えを行なうことになり、メイン関数の定義は

```
main(state s) == printf(s,"%d¥n",((1 * 1) * 2) * 3):external ;
```

となる。「A * B」の演算子ごとに定められた規則は

（書換え規則-「A * B」）
AとBの積を表す定数名に書換える。

であるので、「((1 * 1) * 2) * 3」が「(1 * 2) * 3」、「2 * 3」、「6」と書換えられ、最終的にプログラム-3のメイン関数の定義は

```
main(state s) == printf(s,"%d¥n",6):external ;
```

となって、書換えが終了する。

関数の定義式の書換えは、その関数の答えが存在するなら必ず終了する。答えが存在しない関数(たとえばすべての整数の和を求める)では、書換えが無限に繰返されることがある。

1.4.関数の計算

さてプログラム3は定義式の書換えが終ると、その最終目的であるメイン関数の値を求めるために「printf(s,"%d¥n",6):external」が計算される。これは外部関数の参照であり、その値は

（計算規則-「printf(A,B,C):external」）
システム状態A(ここでは引数「s」に代入されている)に対して、
書式Bで(ここで定数名「"%d¥n"」で表されている)、
C(ここでは定数名6で表されている)を出力した後の状態

である。

プログラムが実行されるとき、メイン関数には引数として、

- ●第1引数:実行開始時のシステムの状態(状態型)
- ●第2引数:コマンド・ラインのアーギュメントの数(整数型)
- ●第3引数:各アーギュメントと環境変数の値の配列(文字列型の配列)

が渡される(プログラム3では、第2、第3引数が省略されている)。第1引数で渡される実行開始時のシステム状態とは、あなたがあるファイルにかかれたプログラムを実行しようとして「akx program3」とタイプ(これをコマンド・ラインと呼ぶ)した後の、あなたが使っているコンピュータ・システムの状態のことである。したがってプログラム4のメイン関数の値は、あなたが「akx prog

3」とタイプした後の状態に、書式「"%d¥n"」にしたがって整数6を出力した状態である。実際には、「akx program3」とタイプした後に「6」と出力されて、プログラムの実行が終了する。

答えが存在する関数の計算は停止するが、答えが存在しない関数の計算は無限に繰返される。

さて今度は、次のようなプログラムを作ってみる。

```
main(state s,int c,string*[2] v) ==                    /* プログラム 4 */
      if c = 2 then
           printf(s,"%d¥n",fact(Integer(v[1]))):external
      else
           s ;
fact(int n) == if n > 0 then fact(n - 1) * n else 1 ;
```

プログラム4のメイン関数の構成は少し複雑である。まずそれは三つの仮引数を持っている。「state」、「int」、「string*[2]」は、それぞれ、「状態型」、「整数型」、「大きさ2の文字列型の配列の型」を示す「型名」である。

配列とは、同じ型の要素がいくつか並んだものであり、要素の総数を配列の大きさという。配列の型名は

要素の型名 * [配列の大きさ]

である。それぞれの要素は、0(先頭の要素を指定)から配列の大きさ-1(末尾の要素を指定)までのインデックスで指定される。たとえば「string*[2]」は文字列型の要素が二つ並んだ配列の型名であり、この型の配列の先頭の要素はインデックス0で、末尾の要素は1で指定される。

次にメイン関数の定義式を見ると、これ以上書換えが行なわれないことがわかる。ただし「A = B」は「AとBが等しいことの真偽」を示す比較の演算、「A [B]」は「配列AのB番目の要素の値」を示す要素指定の演算である。また「Integer(v[1])」は組込関数の参照である。

この定義式中に現れる演算や組込関数の参照の計算規則は、次のように定められている。

(計算規則-「if A then B else C」)
　まずAの値を求め、それが真ならBを、偽ならCを計算する。

(計算規則-「A = B」)
　AとBが等しいなら真、等しくないなら偽。

(計算規則-「A [B]」)
　配列Aのインデックスで指定される要素の値。

(計算規則-「Integer(A)」)
　Aを整数型に変換した値。

これらの計算規則からプログラム4のメイン関数の値は、コマンド・ラインから与えられたアーギュメントの数が2なら、インデックス1で指定される文字列型のアーギュメントの配列の要素を整数型に変換し、その値の階乗を実行開始時の状態に出力した後の状態であり、アーギュメントの数が2以外なら、実行開始時の状態そのままである。

コマンド・ラインと、メイン関数に渡されるアーギュメントの数、アーギュメントの配列の関係は、次の表から想像できるだろう。

コマンド・ライン	アーギュメントの数	アーギュメントの配列	
		先頭要素	次の要素
akx program4	1	"program4"	""
akx program4 5	2	"program4"	"5"
akx program4 5 extra	3	"program4"	"5"

「akx program4 5」とタイプしてみよう。プログラムに誤りがなければ、「120」と出力されるはずである。アーギュメントの数が多くても少なくても、出力はない。

1.5.外部関数の参照とその副作用

さて、ここでプログラム1をもう一度見てもらいたい。このプログラムの形式上の目的は、メイン関数の値を得ることである。その値は

(計算規則-「printf(A,B):external」)
システム状態Aに対して、書式Bを出力した後の状態

から、プログラム実行開始時の状態に「こんにちは」と出力した状態、になる。しかしこのプログラムの作者の本当の目的は、この値を得ることではなく、プログラムに「こんにちは」と出力させることであっただろう。

有益なakancプログラムは、必ず外部関数の参照を含んでいる。外部関数の参照がなければ、どのような複雑な計算も閉じた世界の中だけで完結し、外からその過程や結果を知ることはできない。そのようなプログラムは我々に何の利益ももたらさないのである。外部関数の参照は計算の世界から我々がいる世界への情報のトンネルであり、その情報はプログラムの「副作用」という形で我々の手に入る。

プログラム1の「printf(s,"こんにちは¥n"):external」は、引数sで与えられた状態に「こんにちは」と出力した状態を計算するときに、副作用として「こんにちは」を出力する。このようにすべての外部関数の参照は(そして外部関数の参照だけが)、その計算のために何らかの副作用を起こす。副作用を起こすことが目的のakancプログラム、すなわちすべての役に立つakancプログラムでは、副作用を起こす前の状態を実引数として該当の外部関数の参照を行ない、副作用が起きた後のシステムの状態を得る、というようにメイン関数が定義される。

さてakancでは、「外部関数の参照の計算規則は環境によって規定される」と定められている。外部関数の参照は外の世界への働きかけであるから、それがどのように計算され、どのような副作用が起きるのかは、言語規則ではなくプログラムが実行される環境によって決まる。外部関数の参照の計算は、実際は「C」というプログラミング言語で書かれたプログラムで行なわれる。

UNIXにはCプログラムを集めた「ライブラリ」が数多く存在する。そのようなプログラムを外部関数の参照に利用するには、次の規則を守らなければならない。

(Cプログラム利用規則)

(a)関数名にはCプログラムのエントリー名を使う
(b)「external」というブロックに所属する関数として扱う。
(c)第1引数には現在のシステム状態を渡し、第2引数以降にCプログラムに渡す引数を置く。
(d)外部関数の参照の値は
(d-1)Cプログラムから値が返されないとき
副作用後のシステム状態。
(d-2)Cプログラムから値が返されるとき
第1要素を副作用後のシステム状態、第2要素をCプログラム関数が返す値とする構造体。
(d-3)確保したメモリ・エリアのアドレスを渡すと、その中に値が返されるときアドレスの代わりに該当の大きさの構造体か配列の値を引数として渡し、それと同じ型の要素を第3要素以降に持つ構造体で受取る。

これらの規則を正しく理解するにはCの知識が必要であるが、ここではその解説はしない。Cについては多くの教科書が出版されているし、またまわりには大勢の専門家がいるだろう。彼等は必要に応じて、Cプログラムが簡便に利用できるような関数の定義を用意してくれるはずである。また、本書のプログラム例に出てくる外部関数の参照に利用されるCプログラムは、どれも標準的なUNIXのライブラリに含まれるものなので、その内容の詳細を知らなくとも、字面どおりに記述すればプログラム例は実行できるだろう。

1.6.マクロ宣言

では次に、ひとつの整数を入力して、それが0より大きいならその階乗を出力し、0か0より小さいなら「0より大きい数が必要」と出力するプログラムを作ってみよう。

```
main(state s) ==                                          /* プログラム 5 */
with  INPUT # struct {state s,int cc,int*[1] data} scanf(s,"%d",(int*[1])):external,
      STATE # INPUT.s,
      DATA # INPUT.data
==    if DATA > 0 then
            printf(STATE,"%d¥n",fact(DATA)):external
      else
            printf(STATE,"0より大きい数が必要¥n"):external ;
fact(int n) == if n > 0 then fact(n - 1) * n else n ;
```

プログラム5では新しい構文をいくつか用いた。そのひとつは「with句」である。with句は関数定義の「==」の前に位置し、その形は

with　マクロ宣言

となっている。さらにマクロ宣言は、

マクロ名 # 式

の形をしている。「マクロ宣言」が二つ以上あるときには、それらを「,」で区切って並べる。マクロ宣言の意味は、マクロ名で式の値を示すことである。

プログラム5では、マクロ名「INPUT」が値を示す式は外部関数の参照「scanf(s,"%d",(int*[1])):external」であるが、その型名は「struct {state s,int cc,int*[1] data}」という構造体の型名になっている。構造体とはその型のいかんによらず要素がいくつか並んだもので、その型名は

struct { 要素の宣言 }

である。さらに要素の宣言は

要素の型名 要素の名前

の形をしている。要素が二つ以上あるときには、それらを「,」で区切って並べる。それぞれの要素は要素名で指定される。たとえば「struct {state s,int cc,int*[1] data}」は、第1要素:状態型で要素名「s」、第2要素:整数型で要素名「cc」、第3要素:大きさ1の整数型の配列で要素名「data」を持つ構造体の型名である。

マクロ名「STATE」と「DATA」が値を示す式は、それぞれ「INPUT.s」と「INPUT.data」である。ただし「A . B」は、「構造体Aの要素名Bの要素の値」を示す要素指定の演算である。また「(int*[1])」は、大きさ1の整数型の配列の代表的な値を示す定数名である。一般に

(型名)

は、その型の代表的な値を示す定数名である。

プログラム5のメイン関数の定義式はこれ以上書換えられない。計算は、「if then else」演算の計算のためには「DATA > 0」、そのためには「INPUT.data」、そのためには「INPUT」が値を示す式となり、まず「scanf(s,"%d",(int*[1])):external」が計算される。その値は

(計算規則-「scanf(A,B,C):external」)
システム状態A(ここでは引数「s」に代入されている)に対して、
書式B(ここでは定数名「"%d"」で表されている)で、
C(ここでは定数名「(int*[1])」で表されている)と同じ型の値を
入力した後の状態、入力した個数、入力した値、の構造体

である。

もし入力された値が0か0より小さいなら、「printf(STATE,"0より大きい数が必要¥n"):external」

の計算、そのためには「STATE」、そのためには「INPUT」が値を示す式となるが、その式はすでに計算され値が求まっている。そこでプログラム5のメイン関数の値は、プログラム開始時の状態に対して整数をひとつ入力した後の状態、さらにその状態に対して「0より大きい数が必要」と出力した後の状態と求まり、プログラムの実行が終了する。

このようにマクロ名が値を示す式は、高々一度計算されるだけで、2度目以降は前に計算された値がそのまま使われる。

(計算規則-マクロ名)
(a)そのマクロ名が値を示す式がまだ計算されていないなら、その式を計算。
(b)すでに計算されているなら、そのときの値。

マクロ宣言を用いると、同じ式が何度も現れる関数の定義式を短く記述することができ、また同じ式を繰返し計算することも避けられる。とくにその式が外部関数の参照であるときには、必ずマクロ宣言を用いなければならない。なぜなら、外部関数の参照の第1引数は現在のシステムの状態でなければならないが、一度その外部関数の参照を計算すると副作用のためにシステムの状態が変わってしまうので、2度目以降の計算に現在のシステム状態を渡すことができなくなるからである。

1.7.繰返し

プログラム5を、ひとつの整数を入力し、それが0より大きいならその階乗を出力した後またひとつ整数を入力して同じことを繰返す、というように変更してみよう。階乗の計算にも見られるように、akancプログラムではこのような繰返しの処理は、定義式の中で定義する関数自体を参照するという、再帰的な関数の定義で表現する。プログラム5では、関数「loop」の定義が再帰的になっている。

```
main(state s) == state loop(s) ;                                    /* プログラム 6 */
state loop(s) ==
with  INPUT # struct {state s,int cc,int*[1] data} scanf(s,"%d",(int*[1])):external,
      STATE # INPUT.s,
      DATA # INPUT.data,
      OUTPUT # printf(STATE,"%d¥n",fact(DATA)):external
==    if DATA > 0 then loop(OUTPUT) else STATE ;
fact(int n) == if n > 0 then fact(n - 1) * n else n ;
```

プログラム5のメイン関数の定義式は関数「loop」の参照であるが、これには参照する関数の型名「state」が付いている。これは、その関数の参照が状態型の値になることを示すものである。参照する関数の型名は省略してもよいことになっているが、そのときには関数の参照の値の型は、

●以前にその関数の定義か参照があったなら、そのときの型と同じ
●以前にその関数の定義も参照もないなら
　組込関数では定められた型
　外部関数では状態型
　プログラマが定義する関数では整数型

となる。なお、以前に定義か参照があったときの型と異なった型を付けるという矛盾は、インタープリタによって検出される。

関数「loop」の定義にも定義する関数の型名「state」が付いている。関数の値は定義式の値でありその型も定義式の型になるので、通常、定義する関数の型名は省略してよい。ただし、定義式の中にその関数自体の参照があり、それに参照する関数の型名が付いていないと整数型の値になってしまう。また、以前に参照があったときの型と異なった型を付けたり、定義式と異なった型を付けるという矛盾も、インタープリタによって検出される。

さてプログラム6は、「for-do-next句」を用いて次のように書くこともできる。

```
main(state s) ==                                        /* プログラム 7 */
state for S=>s
with  INPUT # struct {state s,int cc,int*[1] data} scanf(S,"%d",(int*[1])):external,
      STATE # INPUT.s,
      DATA # INPUT.data,
      OUTPUT # printf(STATE,"%d¥n",fact(DATA)):external
do    if DATA > 0 then next(OUTPUT) else STATE ;
fact(int n) == if n > 0 then fact(n - 1) * n else n ;
```

「for-do-next句」は、

繰返される式の型名　for　名前付き実引数　with句　do　繰返される式

の形をしていて、式が書けるところにはどこにでも書ける。「繰返される式の型名」はその型が整数型のときには省略できる。「名前付き実引数」は、

名前 => 式

の形をしていて、その名前は繰返される式の中で仮引数の名前と同様に使用できる。名前付き実引数が二つ以上あるときには、それらを「,」で区切って並べる。

プログラム6と7を見比べると、「for-do-next句」が、メイン関数の定義式にあった関数「loop」の参照と関数「loop」の定義に対応しているのがわかる。「for-do-next句」は、繰返しのためだけに再帰的に定義される関数とその参照をひとまとめに記述するために用いられるが、インタープリタで関数の定義とその参照に分解されてから、定義式の書換えや計算が行なわれる。

もうひとつ「for-do-next句」を用いた例を挙げてみよう。プログラム9は、コマンド・ラインのアーギュメントを順に最大100個まで出力するものである。

```
main(state s,int c,string*[100] v)                      /* プログラム 9 */
with  SIZE # if c > 100 then 100 else c
==    state for S=>s,i=>0
      do    if i < SIZE then
                next(printf(S,"%s¥n",v[i]):external,i + 1)
      else
                S ;
```

この「for-do-next句」はインタープリタによって、次のように分解されるだろう。

```
main(state s,int c,string*[100] v)                      /* プログラム 10 */
with  SIZE # if c > 100 then 100 else c
==    state main_1(s,0,c,v,SIZE) ;
state main_1(state S,int i,int c,string*[100] v,int SIZE)
==    if i < SIZE then
                main_1(printf(S,"%s¥n",v[i]):external,i + 1,c,v,SIZE)
      else
                S ;
```

「main_1」はインタープリタが付けた再帰的に定義される関数の名前である。もとの「for-do-next句」の名前付き実引数にはないが繰返される式の中で使われていた、メイン関数の仮引数「c」や「v」、マクロ名「SIZE」(それらの値は繰返しの各回で不変である)が、関数「main_1」の引数として現れている。

このように「for-do-next句」を用いると繰返しの記述を簡便に行なうことができる。

----以上で第1回は終わりです。

儲かってまっか SEA 関西日記

中野秀男
（大阪大学）

周囲の煽てに乗ったのか，二次会以降に喋ったのか，この SEAmail に「SEA 関西日記」を書くことになりました．特に事務局からの勧めは最後のとどめとなったようです．一つのボランテア活動の記録としての気持ちもあったので以前から書きたかったが，なかなか時間がないと言いながら踏ん切りがつかなかっただけです．

今，私の手元に一枚の紙があります．SEA 関西・分科会記録で，SEA 関西の分科会が 100 会を迎えたのを期に SEA 関西のおもての仕掛人の日本電子計算の臼井さんが作ったものです．これは平成元年の３月までのものであり，この原稿を書きだしたのが平成２年のお盆であるから，すでに分科会は 130 回を超えてます．今はあまりやっていませんが，SEA 関西が始まった頃は月に１回に研究会もやっていましたし，SEA 関西の世話人会やその改名した SEAsideSalon もやっていますから，他の全国版の分科会（あえて東京支部とは言わない）や支部の研究会からすると不思議がられているようです．理由はいくつかあるようですが，そのあたりも含めてこの日記に書いていきたいと思ってま．

SEAmail の編集人にも立候補しましたから，細く短く，いや長く続けたいと思いますので宜しくお願いします．話の展開としては，昔からの流れと現在進行形の話を交互にしながらはじめたいと思っています．ただ，この SEAmail は会員以外に会社の中で回覧されたりして，突然，あの話は読みましたと言われたこともあるので，心して遠慮なく書きます．

１．ＳＥＡ関西　平成２年７～８月

SEA 関西の８月は，殆ど夏休みでした．予定しているのは JIP の臼井さんの話だけだったので，SEA 関西の案内も８月の上旬の発送でよく比較的に楽ででした．いつも宴会をやっている SEA 関西だと言われていますが，これは JUS ふうに言えば情報交換パーテイで，これだけに来る猛者もいます．この二次会も一次会（分科会のこと）が決まればほぼ自動的に決まっていて，同じ大学の江木さんに言われて以来，junet の fj.meetings に二次会情報も流しています．最近，この fj.meetings には真面目な案内が殆どになってきましたので少し心配しています．まあ，大ざっぱの O 型だから，まあいいか．

先月の７月は３つの分科会が開かれました．SEA 関西が５つもの分科会を持ちながら長続きしている理由の一つは各分科会の世話人が違うことでしょう．

11.

AI 分科会　臼井 (日本電子計算)
ネットワーク分科会　中野 (大阪大学)
システム技術 上記２人
CAI 分科会　松本 (読売コンピュータスクール)

と一応なっています．それぞれの分科会の今までの話には涙無しには (笑い?) 語れない話があるが，これはこれからのお楽しみである．

７月の SEA 関西の案内は以下の通りです．

ＳＥＡ関西　平成２年度７月度　分科会のご案内

６月だと言うのに暑い日が続きます．でも，これを読まれた時には梅雨に入っているでしょうか．さてさて，ＳＥＡ関西の７月の分科会をご案内します．仕事や研究に疲れたり飽きた方，また出張で大阪にきて楽しいお酒を飲みたい方，是非おいで下さい．参加費は各分科会とも，ＳＥＡ会員５００円，一般１０００円学生３００円です．

□　ネットワーク分科会

日　時：　平成２年７月１０日（火曜日）　１８：３０－２０：３０
テーマ：　シスオペが語るパソコン通信四方山話
ゲスト：　川原　究（アクアス，帝塚山マイコンクラブ代表）
　　Email: 日経 MIX/blue
場　所：　千里国際情報事業財団　セミナー室
　　大阪市北区角田町８－４７　阪急グランドビル１５Ｆ
二次会：　卵２０個入りのオムレツのある居酒屋
連絡先：　06-877-5111 ext.4773（阪大: 中野）nakano@oucom.osaka-u.ac.jp

□　システム技術分科会・ＧＵＩ研究会　合同

日　時：　平成２年７月２５日（水曜日）　１５：００－１７：００
テーマ：　GrapherGear(Smalltalk によるアプリケーション開発支援ツール)
ゲスト：　青木　淳（富士ゼロックス情報システム）
　　Email: r2d2@fujixerox.co.jp
場　所：　富士ゼロックス
　　大阪市中央区本町 3-5-7　ゼロックスショールーム
　　地下鉄御堂筋線本町３番出口上
メ　モ：　SmallTalk を知りたい人必見・格安・駅より１分
二次会：　時間が夕方ですので，ビアガーデンがいいですね．
連絡先：　06-448-6024（ＪＩＰ: 臼井）usui@aster.jip.co.jp

□　ＣＡＩ分科会

日　時：　平成２年７月２７日（金曜日）　１９：００～２０：３０
テーマ：　マルチメディアによる音楽教育
　　－マルチメディアでみるモーツァルトの生涯－
ゲスト：　前野博（兵庫女子短期大学　非常勤講師）
場　所：　ＫＣＳ会議室
　　神戸市中央区浪速町６４　モーシェルビル６Ｆ
　　神戸オリエンタルホテル北むかい
二次会：　神戸風ビアホール
連絡先：　078-391-7733（ｋｃｓ: 平山）hirayam@kcs.co.jp

７月１０日に行われたネットワーク分科会は，アクアスの川原さんを迎えて，梅田にある千里国際情報事業財

団で例によって午後６時半から開かれました．話はパソコン通信の話です．川原さんはパソコン通信の初期のころから帝塚山マイコンクラブでパソコン通信のシスオペをやっておられた方で，たまたま SEA 関西で５月か６月かに NeXT の紹介をした後の例のオムレツ２０個の居酒屋で川原さんの名刺と自己紹介を聞いて，お話を頼んだ訳です．話は画像ファイルの受信の話など一部危ない話がありましたが，草の根のパソコン通信の大変さを物語って面白い話でした．梅田で終わった後は恒例のオムレツ２０個の居酒屋です．今回は私が NEC ACOS ユーザ会で知り合って案内を出した所，それを見て来られた人もいました．

GUI 分科会は，オブジェクト指向では引っ張りだこの，FXIS の青木淳さんの話です．青木淳なら，うん万円のセミナーがやれるのにと思いながら，会費１０００円の分科会です．SEA 関西では，ゼロックスでやるときはショウルームが使えるので，お昼に本町の富士ゼロックスで分科会を開きます．出し物は，SmallTalk-80 + Grapher Gear でした．参加者は２５名ぐらいでした．いずれ参加者もここに載せるつもりです．私は当日，名古屋で来年のソフトウェアシンポジウムの打ち合わせがあったので，丁度，分科会が終わるころに行きました．本当はもっと前に行けたのですが，名古屋で打ち合わせが終わった後に，是非，味噌煮込みうどんを食べたいという人がいたので，つきあったためぎりぎりになりました．分科会の後はビルの地下で，軽く懇談会をしましたが，そのとき阪大工学部電気の学生がいたのにはびっくりしました．そのあとは，青木淳さんを梅田のホテルにチェックインさせて，JIP の井川さんも入って SEA の若者達の meeting をやりました．

８月の出し物は，お盆明けに開いた，CASE の要素技術だけででした．以下が案内です．

ＳＥＡ関西　平成２年度８月度　分科会のご案内

連日の猛暑が続いています．もう何回目の夏ばてですか．ＳＥＡ関西の８月の分科会は，下旬１回だけデフォールトがお話をするだけで，大体が夏休みです．夏休みに疲れたり飽きた方，また出張で大阪にきて楽しいお酒を飲みたい方，是非おいで下さい．参加費は各分科会とも，ＳＥＡ会員５００円，一般１０００円学生３００円です．

□　システム技術分科会

日　時：　平成２年８月２４日（金曜日）　１８：３０－２０：３０
テーマ：「ＣＡＳＥの要素技術と一例」
ゲスト：　臼井義美（日本電子計算）
場　所：　日本電子計算（株）大阪支店　６階会議室
　　大阪市西区江戸堀　１－９－１　肥後橋センター６Ｆ
　　（地下鉄四つ橋線肥後橋下車，ビル入口は１８：００に
　　閉まりますので地下街よりエレベータを利用して下さい）
二次会：　肥後橋センタービル地下「天狗」の予定
連絡先：　06-448-6024（ＪＩＰ：臼井）usui@aster.jip.co.jp

８月２４日のシステム技術は，JIP 大阪の６階会議室で開かれました．内容は遅れていったために最後の討論だけでパス．もう一つ，RISC Luna を８月の持ち札にしていましたが，相手が CMU に滞在する方が多くなったので，しばらく見合わせる事にしました．彼は某社のポストマスターをするうちに doda したわけで，対向のポストマスターとして複雑な気持ちである．A 社の F 氏に言わせれば，当たり前でしょ，だろうが．

２．昔の話（その１）

SEA 関西の分科会の始まりの頃から，私も SEA に関係しだしたので，そのあたりから話を始めます．いろいろな話が織りなす縦糸を横糸が交差するのでややこしいが，年を追って話をしてみます．

1980 年に入って UNIX という OS があると知って，1983 年に Berkeley に行ったときに BSD UNIX にさわってみて，その面白さ（ネットワーク，プロセス等）を知りました．6800 を触ったり，Lkit-16 や，TP80 等，珍しいものがあると飛びつくのは今だにかわらない性格です．

それと前後して，たまたま１年下の卒業生が PFU(当時のパナファコム）にいたので，会社見学に呼ばれた事がありました．見学の後で恒例のお食事会があったときに，ついてきたのが，現 SEA 代表幹事の熊谷章さんでした．見学の時にも顔を合わさなかったので，後から聞いてみると，社内でお食事会のメンバーを募ったところ手を挙げたのが熊谷さんであったらしい．これが SEA との出合の伏線になるとは思わなかった．当時の私の熊谷さんに対する印象は，お食事の後，寿司屋にいったときに，「中野先生，ここのネギトロはおいしいですよ」と，本当においしそうに食べたので，面白い人だと思いました．

その頃，研究室で VAX を買う金などないし，工学部や計算センターにもないと言う状況で，TANDY の 200 万円弱の 15MB(メモリーではありません．ハードディスクの量です）の UNIX マシン (XENIX) を買いました．UNIX を触っているという事が喜びでした．

1986 年の春に，構造化エディターの勉強会が東京で１泊であったので参加しました．ここで，当時，野村コンピュータの佐原伸さんに会い，SUN workstation や東工大の話などを聞いて，なんとかせねばと思いました．その秋に，所属する学科にまとまったお金が入り，「ワークステーションとネットワーク」に使って良いという話が起こりました．それと前後して（どうもこれが多い），この秋に，伝説の横浜中華街の JUS シンポジウムがあり，この会の始まる朝に佐原さんに会い，会った早々，「今度，ソニーからでる NEWS というワークステーションは，ハッカー達が良いと評価していますよ」と教えて貰いました．

話はまた前後しますが(どうもこの頃が学会以外の人と良く会いだした始まりであったようです)，この年の夏頃に，SEA 関西（やっと出た）の研究会（毎月，土曜の昼から大阪の能率協会でやっていたらしい）の案内が，JUS 関西に顔を出していたために DM で送られてきました。講師の一人が熊谷さんであることと（演題は「近類愛」だったらしい），MAC のデモがあったため行きましたが，SEA という団体である事を後から知って（そのときは知らなかった），研究会の後，熊谷さんに付いて臼井さんたちと食事をしたのが(そのころから２次会には出ていたなあ），SEA 関西との遭遇でした．

本当の遭遇は，横浜のシンポジウムのあとの大阪での JUS のシンポジウムで，臼井さんから，「あら，お兄さん（実は彼の方が上)」と呼び止められたのが，SEA 関西とのくされ縁な始まりのようでした．このシンポジウムでは，１５００万円でワークステーション５台とネットを張るために策を練っていた時でした．このシンポジウムの昼飯時に，近所の喫茶店で同席したのが SRA の林香さんであったのも不思議な因縁を感じます．

このシンポジウムの後，臼井さんから SEA 関西分科会第１号の

AI 分科会

の案内がありました．たまたま（常にたまたまなのは運命としか言いようがない）梅田で別の研究会があったので，その足で分科会に参加したところ，第２回を阪大の溝口先生に頼むというので斡旋したりして，いよいよ泥沼に入って行ったきたようです．

今，思うに，SEA や JUS への接近は，研究環境や計算機環境を良くしたいという一心であったようです．この点については，今でも最新の，正しい話は（すべてではありませんが）ここからくると信じており，私の何でも手を出す「すけべ心」とあいまって，SEA 関西や SEA 全国版に絡んできたのでしょう．

３．ＳＥＡ関西日記平成２年９月

９月の SEA 関西は，やはり月の後半でネットワークと GUI を用意したが，わけのわからないものも必要と思い，SIS 放談もいれた．９月の３日の週に SEA 関西の案内を作成し発送した．今までの臼井さんから引き継いだ封筒もついに切れ，１部は SEA 本部の封筒を使った．この封筒も新たに２０００部を作ったのでしばらくは大丈夫だろう．案内の紙も新たな台紙を印刷屋に頼もうと思うが，Mac のレーザプリンターがあるので，これもやはり自分でやるべきだと思いながら時間がない．以下が９月の案内である．

ＳＥＡ関西　平成２年度９月度　分科会のご案内

やっと，朝晩涼しくなってきました．そろそろまた SEA 関西も活動を開始です．ＳＥＡ関西の９月の分科会は，やはり月の後半になりました．システム技術の SIS のパネルは気楽にやりたいので，日頃うっぷんのある方は是非参加して下さい．涼しくなっても関係なく睡眠の少ない方，出張で大阪にきて楽しいお酒を飲みたい方，是非おいで下さい．参加費は各分科会とも，ＳＥＡ会員 500 円，一般 1000 円，学生 300 円です．

□　システム技術分科会

日　時：　平成２年９月１８日（火曜日）　１８：３０-２０：３０
テーマ：「SIS（戦略的情報システム）に関する討論会」
ゲスト：　SEA 関西の幹事達とそのあたりの面々
場　所：　日本電子計算（株）大阪支店　６階会議室
　　大阪市西区江戸堀　１－９－１　肥後橋センター６Ｆ
　　（地下鉄四つ橋線肥後橋下車，ビル入口は１８：００に
　　閉まりますので地下街よりエレベータを利用して下さい）
二次会：　肥後橋センタービル地下「天狗」の予定
連絡先：06-448-6024（ＪＩＰ：臼井）usui@aster.osaka.jip.co.jp

□　ＧＵＩ研究会

日　時：　平成２年９月２５日（火曜日）　１８：３０-２０：３０
テーマ：　X ウィンドウの話
ゲスト：　石曽根 信 (SRA)
　　Email: ishisone@sra.co.jp
場　所：　千里国際情報事業財団　セミナー室
　　大阪市北区角田町８－４７　阪急グランドビル１５Ｆ
二次会：　卵２０個入りのオムレツのある居酒屋
連絡先：06-877-5111 ext.4773（阪大：中野）nakano@oucom.osaka-u.ac.jp

□　ネットワーク分科会

日　時：　平成２年９月２７日（木曜日）　１８：３０－２０：３０
テーマ：　SONY NEWS の ISDN ボード
ゲスト：　萩原 崇 (SONY)
場　所：　千里国際情報事業財団　セミナー室
　　大阪市北区角田町８－４７　阪急グランドビル１５Ｆ
二次会：　卵２０個入りのオムレツのある居酒屋.そろそろ場所を変えないと
連絡先：06-877-5111 ext.4773（阪大：中野）nakano@oucom.osaka-u.ac.jp

まず９月１８日（火）は JIP ６階の会議室で「ＳＩＳの話」である．あまり盛り下がってもいけないので，前日，ＳＩＳの本を２冊買うが，いずれも技術の本ではなく参考にならず．結局，私は「ソフトウェア業界のためのＳＩＳ」の話をする．話の内容は昨年１２月の釧路の環境ワークショップのＢＯＦの話なのでここでは書けない．この分科会では常連さんが，このあとの懇親会を期待して集まる．２次会はおなじみの「天狗」．昼休みに江木先生に声を掛けたので，ここから参加．分科会の参加者名簿を居酒屋で足元において飲んでいたらやっぱり忘れる．翌日，ＳＥＡ関西メイリングリストで頼んでみたがやはり見つからず．幸い，領収書の必要な方は名刺を貰っていたし，参加費も封筒にいれていたので実害はなかったがここで紹介できず残念．

９月２５日（火）は，ＳＲＡの石曾根さんによるＸｗｉｎｄｏｗの話．電話やＥメイルでの問い合わせが３件ほどあり，いづれも参加してもよいかとの問い合わせ．ＳＥＡは個人ベースの団体なので，自由にどうぞ，但し，参加費は会員と非会員では違いますよと，いつも言っている．１次会でも２次会でも私自身から勧誘はしたことがないが，自然に流れるのが良いと思っている．良いものは良いし，悪いものは悪い．「ワイワイガヤガヤ」も良いし，技術指向も良いし，大切な事は良い技術や環境や考え方を盗む事だと思う．自分の環境を恥ずかしがる事かもしれない．

石曾根さんの話はＸｗｉｎｄｏｗの基礎とｗｉｄｇｅｔの話であり，私はレイヤーの考え方やプロトコル，又，ＸとＭｏｔｉｆ等の関係等大変面白かったが，ＷＳやＸに関係のない人のためには，Ｘのデモをするべきだったかもしれない．これは私がいつも思うことで，私が分科会等をアレンジすると，ＷＳやその関連が多く，Main frame 関連がＳＥＡ関西では多いので，先端をやってしまうと，反対の先端の方は入って行けず難しい．当日の参加者は以下の通りである．会社名は省略する．

大野修一，阪井 誠，中村宰典，佐々木佳一，勝賀瀬豊隆，宮城憲博，川本俊行，山浦康成，花原良宏，三浦あさ子，馬場始三，西川秀樹，鈴木秀俊，川上 優，杉山英俊，青井周一，岡田雅浩，大倉武志，中野秀男

この内，１２名ほどが有名になった卵が２０個のオムレツのある居酒屋へ．次回のＧＵＩはＳＲＡのあさ子さんに決って一安心．

９月の２７日はＳＯＮＹ･ＮＥＷＳのＩＳＤＮボードの話を，実際にソフトウェアを担当された，萩原にお願いした．さすがＳＯＮＹほどの大会社になるとＳＥＡだと言ってもうさんくさく見えるのか，参加予定者リストを送った．大きな会社に頼むと良くあることである．ＳＥＡ関西では，ｊｕｎｅｔを使ってメイリングリストを作り，ここで話を展開しながらテーマを作りたいと思っているが，今回の話はＪＩＰの井川さんとＳＲＡの阪井さんに新しいテーマを，システム技術やネットワーク分科会のために頼んで実現したものである．

９月からは千里国際情報事業財団からＳＥＡ関西の分科会のための会場の便宜をはかって貰っているので，２５日と２７日とも梅田のグランドビルにある財団のセミナー室で開いた．ちなみに，この財団は何年後かに千里に移るが，移る阪急山田駅が私やＪＵＳの幹事の松浦さんのいる所なので，便利につかえそうである．

ＩＳＤＮボードの話も当日に２件ほどの問い合わせがあった．ＳＯＮＹの萩原さんはとってもフランクに話をされ，私のように通信所属でＩＳＤＮにある程度の知識があるものにとっては，とても参考になった．少し，話は脱線するが，良く（ポイントにならない）ＳＥＡ関西の分科会をサポートしますねと直接，または間接（顔を見れば分かる）に聞かれるが，私が担当する分科会では，私が聞きたい，又は興味のあるテーマしかやっておらず，私の勉強だと思っているし，思っているからやれるのである．本や雑誌からだけでは本音や裏話は聞けない．

さて，話をもとにもどして，萩原さんはＩＳＤＮの一般的な話から，ボードのハードウェアやソフトの話をさ

れ，ボードの構成やＵＮＩＸおよびＴＣＰ／ＩＰとの組合せなど，大変参考になった．ボード間では 60Kbps 程度が可能であり，残りの１チャネルでは音声も使えることや，ＩＰ接続での接続のさせかた等の説明があり，ユーザからみると余り手を加えずにやれそうである．ただ，ネットワークとしては，１．５Ｍｂｐｓの高速回線が安くなるかもしれず，良い意味で競合してくれればありがたい．当日の参加者は以下の方々である．

大野修一，田窪英智，瓶子則夫，高橋伸尚，伊藤和生，安徳剛志，横山博司，武居正憲，阪井 誠
西川秀樹，中村宰典，杉山秀俊，坂本利夫，伊藤孝男，竹内常雄，岡田雅浩，臼井義美，勝賀瀬豊隆
三里英雄，松本省二，森 秀和，中野秀男

分科会にはＬｕｎａのＩＳＤＮボードの担当者も２次会まで参加して，私としては週に２回もオムレツを食べてしまった．ＳＥＡ関西の常連の１人である松下の田窪さんが１１月３日に結婚する予定とか，nice middle が中核のＳＥＡも，nice young をたくさん育てないと駄目ですね．

（この項は，中国・上海の華東計算技術研究所で下書きしたものです
次からはやっと，SEA 関西の今までの分科会の歴史になります．）

気がついたら3年半
～長野支部活動報告～

報告者: 市川　寛(電算)

87年5月に長野支部が設立され、早いものでもう3年6ヵ月が過ぎようとしています。支部の月例会も今年の10月開催分で38回目を迎えますから、ほぼ毎月の計算になるでしょうか。月例会の場所は長野市中心ですが、SEA会員が県内に広く分布していることもあり、3回に1回は松本市に出向いて開催しています。また、開催日時も現在のところ、毎月中旬か下旬の土曜午後3時からが一般的です。

テーマについては、この38回を振り返ってみますと、非常に広範囲に渡っています。というよりは、的を絞り込めないので、手当たりしだいということでしょうか。ちょっと、分類してみました。

汎用機における開発ツール

MAPPER	日本ユニバック
MAPPER KID	日本ユニバック
CANO-AID	Canon
CASET,YPS	富士通
SOFPIA,IDLII	日本電気
LINCII	日本ユニシス

実現されているネットワーク環境

信州大学キャンパスネットワーク
長野大学インテリジェントビル
アルプスネット(パソコン通信)
OS/2におけるネットワーク
NTTのISDN

PC･EWS上のソフトウェア

J Star	富士ゼロックス
創玄	エー･アイ･ソフト
Empress	エム･ケー･シー
創考	エー･アイ･ソフト
大創玄	エー･アイ･ソフト
dbMAJIC	ワコム
AS-Documents	東芝

フォーラム･情報処理学会中部支部講演会等

信州コンピュータフォーラム
ソフトウェア開発環境に関するセミナー
ソフトウェア開発プロセス
知識ベースシステムの考え方と方法
全国縦断CASEフォーラム

その他

Σワークステーション
データ交換システム
大規模オンライン開発
データ中心設計
技術者教育
フロッピーディスクの互換性

これらから分かる通り、地元の技術者およびアポイントのとれるメーカを動員したテーマ作りの結果ということです。そのためもあって機器の持ち込み実演が数多くありました。しかし、そろそろ息切れの感じがしています。そこで、今後は長野支部以外からの応援をいただき、より密度の濃い月例会に育てていきたいと思うこの頃です。

虫のいい話ですが、どなたかテーマを持って進んで長野支部に参加していただける方はいませんか。よろしくお願いします。

連絡先　株式会社　電算
教育研修室　市川
Tel 0262-34-0151
Fax0262-33-1139

7th INTERNATIONAL SOFTWARE PROCESS WORKSHOP

Communication and Coordination in the Software Process

16-18 October 1991, San Francisco Bay area

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

Organizing Committee

Dennis Heimbigner Karen Huff Maria Penedo
Wilhelm Schaefer Tetsuo Tamai Ian Thomas

The 7th International Software Process Workshop will focus on communication and coordination aspects of the software process and related issues. A software development process typically involves many individuals, often fulfilling many roles, many sites, often geographically distributed, and many sub-processes. Sub-processes may be described in differing process description techniques adapted to their specific domain, and implemented by differing and distributed tools and automated support mechanisms. The workshop will focus on communication, coordination and interaction of process technology, process descriptions and their supporting automated mechanisms. Some of the issues are as follows:

Multiple process description techniques.
Different process description techniques may be appropriate for distinct parts of the process thus requiring well-defined interfaces and support for inter-operability.
- which process description paradigms are appropriate for what aspects of the process?
- can we characterize commonality across different paradigms?
- how are process descriptions at the levels of the enterprise, project, team, and individual related?
- how appropriate are process descriptions paradigms for interface definition and interoperability within and across paradigms?

Communication, composition, and information flow.
Process descriptions (possibly described/implemented in different paradigms) need to communicate, to manage information flow in the process and to be combined into larger process descriptions.
- how is information flow modeled in process description techniques?
- how do enactable processes (possibly using different description/implementation) techniques communicate with each other?
- how are sub-processes composed into larger process fragments ?
- how do communication techniques support the "small–scale" versus the "large-scale" aspects of processes?
- what impact do those capabilities have on the underlying information management systems of process–centered environments?

Design and implementation of process-centered environments.
Environments are increasingly providing automated support for the life–cycle process.
- what are the consequences of geographical distribution of projects on the design of supporting process enactment platforms?
- what empirical evidence exists on the adaptability of users to a process–centered environment?
- what are the likely effects of process-centered environments on SEE and tool architectures?

The three day workshop, which will be held in the San Francisco Bay area, 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 March 1991, 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) shoud be sent to:

Ian Thomas
Hewlett-Packard, Software Engineering Systems Division
1266 Kifer Road, Sunnyvale, California 94086, USA.
email: ispw7@hp–ses.sde.hp.com

Electronic mail submissions should be in Ascii; Latex form is acceptable. They should include author's full address and telephone number.

Preliminary Announcement and Call for Papers

1st International Conference on the Software Process

21 – 22 October 1991, California, USA

Sponsored by the Rocky Mountain Institute of Software Engineering (rMise)
Cooperation being sought from Asian, European and US Professional Societies

Program Committee Will Include

Bob Balzer	Bill Curtis	Mark Dowson (chair)
Watts Humphrey	Gail Kaiser	Kouichi Kishida
Manny Lehman	Lee Osterweil	Maria Penedo
Dewayne Perry	Bill Riddle	Win Royce

The software industry manufactures (and maintains and evolves) large, complex artifacts – software systems. We would like its products to be of high quality and reasonable cost, and to be delivered on time and within budget. One important way to achieve these objectives is to improve the manufacturing process – the software process. A software process is a set of steps for creating and evolving software systems that encompasses both technical and managerial concerns. In particular, it must address technical development, project management, data and configuration management, and quality assurance and control throughout the software life cycle.

Historically, the software engineering community has focused on the products of software processes. Recent advances in the understanding of these processes, focusing on the activities involved in creating software products, present an opportunity to solve many of the problems underlying software creation and evolution.

This conference will address the description, enaction, and automation of software process activities, covering a wide range of topics including:

- software process models and modeling methods
- software process measurement, assessment, and improvement
- process description formalisms and languages
- process–centered software development environments and tools
- empirical studies of the software process
- descriptions of proposed or actual software processes.

The conference, which will be the first of a continuing series, will present ongoing work on the above and other software process–related topics through a single track of sessions which will include invited presentations, refereed papers, experience reports, and panel sessions.

High quality original papers, short experience reports, and panel proposals addressing the above topics should be submitted by 1 April 1991. Papers should be limited to 6000 words. Experience reports should describe practical experience with software process–related approaches, and be limited to 2000 words. Panel proposals should describe the topic to be addressed and identify the prospective panelists. Five copies of submissions should be sent to:

ICSP1
rMise
PO Box 3521, Boulder, CO 80303, USA
Tel: +1 (303) 499 4782

Electronic mail submissions (plain text or postscript only) may be sent to:

icsp1@sda.com

All submissions should indicate the full postal address, telephone number and e–mail address (if available) of the author(s).

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.03-356-1077　FAX.03-356-1072