SEAMAIL

Monthly Newsletter from
Software Engineers Association

Volume 2, Number 9 September 1987

目　次

ソフトウェア技術者協会（ＳＥＡ）は，ソフトウェア・エンジニアの，ソフトウェア・エンジニアによる，ソフトウェア・エンジニアのための団体であり，これまでに日本になかった新しいタイプのプロフェッショナル・ソサイエティたることを目指して，１９８５年１２月２０日に設立されました．

現在のソフトウェア技術が抱える最大の課題は，ソフトウェア・エンジニアリング研究の最前線（ステイト・オブ・アート）と，その実践状況（ステイト・オブ・プラクティス）との間に横たわる大きなギャップを埋めることだといわれています．ソフトウェア技術の特徴は，他の工学諸分野の技術にくらべて属人性がきわめて強い点にあります．したがって，そうしたテクノロジー・トランスファの成否の鍵は，研究者や技術者が，既存の社会組織の壁を越えて，相互の交流を効果的に行うためのメカニズムが確立できるか否かにかかっています．ＳＥＡは，ソフトウェア・ハウス，計算センタ，システム・ハウス，コンピュータ・メーカ，一般ユーザ，大学，研究所など，さまざまな職場で働く人々が，技術的・人間的交流を行うための自由な＜場＞であることをを目指しています．

ＳＥＡの具体的な活動としては，特定のテーマに関する研究分科会（ＳＩＧ）や地方支部の運営，月刊機関誌（ＳＥＡＭＡＩＬ）の発行，各種のセミナー，ワークショップ，シンポジウムなどのイベントの開催，既存の学会や業界団体の活動への協力，また，さなざまな国際交流の促進等があげられます．

なおＳＥＡは，個人参加を原則とする専門家団体です．その運営は，つねに中立かつ技術オリエンテッドな視点に立って行われ，特定の企業や組織あるいは業界の利益を代表することはありません．

ＳＥＡＭＡＩＬ　Ｖｏｌ．２，　Ｎｏ．９　昭和６２年９月１日発行

編集人　岸田孝一

発行人　ソフトウェア技術者協会（ＳＥＡ）

〒１０２　東京都千代田区隼町２－１２　藤和半蔵門コープビル５０５

印刷所　サンビルト印刷株式会社　〒１６２　東京都新宿区築地町８番地

定価　５００円

編集部から

1. 特集について

すでに予告した通り，この号では，１９８５年の秋にアメリカで開かれた環境ワークショップの報告書全文の日本語訳を，特集として掲載する．

オリジナルは，主催団体である rMISE からテクニカル・レポートとして刊行され，また，ACM/SIGSOFT の機関誌 Software Engineering Notes の１９８６年１月号にも，ほぼそのままの形で転載されている．

日本語への翻訳許可は，rMISE　の主宰者であり，このレポートの執筆者の１人である W.E.Riddle 氏の御好意により，約２年前に，氏が第１回春のセミナー・ウィークに基調講演者として来日された折り，SEAMAIL が取得した．実際の翻訳がここまで遅れたのは，単純に編集部サイドのパワー不足のためであって，別に他意はない．

ただし，このワークショップの討論内容は，ソフトウェア開発支援環境の構築に関する技術動向を広くサーベイし，９０年代半ばへ向けての技術戦略を探るというものであって，現在のわれわれにとって，依然として数多くの有益な示唆を含んでいる．

たとえば，このワークショップが開かれたのは，偶然にも，「あの」シグマ・プロジェクトが正式に発足し，開発本部内で，プロジェクトのポリシーや計画をめぐって，さまざまな論議が闘わされていた時期と一致する．

そうした観点から，いわばこのレポートを１枚の海図として，シグマのこれまでの航跡をふりかえってみると，個々の技術項目についても，また，全体的な開発戦略や組織上の問題についても，考えさせられることがいくつもあるように感じられる．

そうした反省は，シグマだけに限らない．個々の組織内において，ソフトウェア環境の問題に関心を抱き，それぞれの立場での改善努力を心がけておられる方々とって，このレポートは，これまでの技術開発のあり方を再検討し，今後の進路を設定するさいに，大いに参考になるものと信ずる．

2. ちょっと余計なコメント

ここで，蛇足ながら，rMISE　とこのワークショップの周辺について，多少解説を加えておく．

rMISE (Rocky Mountain Institue of Software Engineering) は，ソフトウェア工学の普及を目的とするセミナーやワークショップなどの活動を実施する非営利団体として， B.Boehm, L.Druffel, W.Riddle の３氏によって，数年前に設立された．８４年から８６年にかけて毎年夏にコロラドで開かれた集中セミナーには，日本からも多くの方々が参加されている．

今回ここに報告されたワークショップは，１９８５年の１１月に，NSF の後援を得て，Riddle 氏が中心になって企画・運営したものである．

「環境」をテーマとするワークショップは，このレポートにも述べられているが，１９８０年の初夏にカリフォルニアで開かれたもの（主催は NBS -米連邦標準局）が最初である．このワークショップは，TOOLPACK　などのプロジェクトを発足させるきっかけとなり，また，プロトタイピングやプロセスに関するワークショップをその副産物として生み出すなど，８０年代前半のソフトウェア技術に大きな影響を与えた．

日本でも，たとえば８０年代前半の代表的環境プロジェクトである JSD の SMEF プロジェクトは，このワークショップの影響のもとに，TOOLPACK　などアメリカの同世代プロジェクトと連携を取りつつ進められた．

rMISE　のワークショップは，そうした８０年代前半の開発成果を総括し，今後の１０年間の開発の枠組みを設定することを目的として，計画された．そこには，また，惜しくも（政治的な理由で？）中断された DOD/STARS プロジェクトで描かれた新環境構築のプランを，より一般的また現実的な形にパラフレーズしようという意図も含まれていたと推測される．rMISE　の関係者３氏は，いずれも STARS 立案の中心人物であった．その壮大な（いささか壮大に過ぎる）計画の全容については，IEEE Computer 誌１９８３年１１月号を参照されたい．

現在アメリカで推進されている大規模な環境プロジェクト（たとえば， MCC や SEI あるいは SPC におけるそれ）の狙いが，いずれもここに報告された討論の延長線上にあることは，明かである．また，昨年から日米共同で開始された SDA プロジェクトも同様であると考えられる．

この原稿が印刷所にまわっている頃，長野市で第３回 SEA 環境ワークショップが開かれている．そこでの熱の入った討論と成果に期待したい．

環境構築技術の現状と未来を探る

rMISE ソフトウェア環境ワークショップ

討 論 記 録

William E.Riddle ・ Lloyd G.Williams 編

岸田孝一・村井進 訳

1. はじめに

１９８５年１１月１２～１５日の４日間，米国コロラド州ボールダーにおいて，NSF (National Science Foundation) および rMISE (Rockey Mountain Institute of Software Engineering) をスポンサーとして，ソフトウェア開発環境に関するワークショップが開催された．

ワークショップの中心的な議題は，もちろん，（自動化された）**ソフトウェア環境**であったが，討論の過程において，**ソフトウェア・プロセス**の概念もまた，きわめて重要なテーマとして浮かび上がってきた．ここで，プロセスとは，ソフトウェア・システムの開発・進化の過程で行われる一連の活動を指す．プロセスと環境は，明らかに，お互いに深い関係を持っている．プロセスに関する１つのパラダイムは，それを支援する特定の環境を必要とするし，逆に，環境は１つまたは複数のパラダイムを支援している．今回のワークショップでは，環境に関するさまざまな問題点を討論する過程で，プロセスについても間接的な言及がなされた．しかし，プロセス自体の改善は，直接の話題としては取り上げられなかった．

このワークショップの目的は，ソフトウェア環境に関するいくつかの重要な問題領域における基本的な課題を洗い出し，技術の改善や高度化のために行うべき特定の活動を明らかにすることであった．討論は，これらの改善提案がなるべく広範囲な応用分野をカバーできるように，ソフトウェア環境の一般的な（応用分野と独立した）側面に焦点を当てる形で進められた．

会議の全体的構成は，図１に示す通りであった．すなわち，１９８５年の現状から出発して，９０年代半ばにおける望ましい状況に到達するための改善プロセスが，かなり詳細に討論された．討論の枠組みを統一するために，まず，この改善プロセスに影響を及ぼすであろういくつかの要因の確認が，ある程度の時間をさいて行われた．これは，主として，現在および１０年後の状況を特徴づけるソフトウェア開発のパラダイム，成果物，開発・管理組織，支援環境，基礎技術などの各項目について，それぞれの属性に関する討論上の仮定を立てるという形であった．将来の予測にあたっては，何が望ましいかではなく，何が実現可能かという観点からの検討がなされた．影響要因を決定するためには，当然，この改善プロセス自体が持つ一般的性格に関する議論や，改善の具体化のための周辺条件に関する仮説の設定を行わなければならないが，後者の話題はそれほど真剣には論じられなかった．

改善プロセスの一般的性格については，かなりの時間をさいて討論が行われた．そこでは，とりあえず，プロセス改善のための諸活動が，きわめて革新的なパラダイムへの切り替えや，革命的な新技術の導入によって，現状からの大幅な飛躍を図るのではなく，現在の環境がそなえている能力を徐々に拡張して行く進化論的また

は漸進的な考え方にもとづいて行われるという仮説が採用された．短期的には，そうした漸進的なアプローチのほうが，成功の可能性が高いと思われたからである．

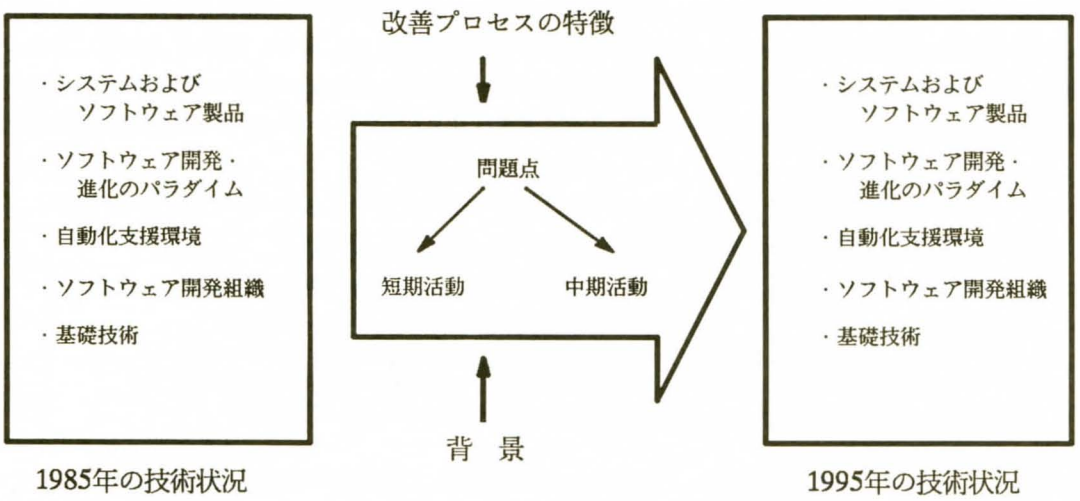


図１　改善プロセス

しかしながら，ソフトウェアの開発・保守に対する革命的なアプローチや新技術がまったく無視されていたわけではない．実際，いくつかの研究開発活動，すなわち，近い将来において，漸進的なアプローチをガイドし，新しいプロセス・パラダイムやソフトウェア技術への転換を容易にしてくれることが期待されるようなものに対しては，特別な関心が寄せられた．漸進的な改善プロセスと革命的なそれとの対比，そしてまた，両者をいかにして統合するかという問題は，以下のレポートを読めばわかるように，ワークショップ全体を通じて，何回となく論議の対象として取り上げられた．

参加者の分布は，大学・産業界・政府機関の３者にほぼ等分にわかれ，かれらの関心事も，環境改善に関する高次元な計画の問題と，具体的な環境の設計・構築上の問題とに，ほぼ同程度にわかれていた．ワークショップの開催に先立って，いくつかの話題についての予備的な議論の材料が，あらかじめ全参加者に配布された．それぞれの参加者は，これらの話題に関するポジション・ペーパを執筆し，討論のグループ分けは，このポジション・ペーパにもとづいて行われた．

ワーキング・グループでは，ツール，分散環境，データベース支援，環境の拡張性，AI技術の利用，という５つの技術的な話題について，討論が行われた．各グループは，まず，それぞれの分野で今後１０年間に起こるであろう変化についての予測を行い，次に，その中で特に注目すべき問題点が何であるかを議論した．そして，最後に，それらの問題点を解決するために必要な短期および中期の活動項目を洗い出した．

これらの個別的な話題に加えて，すべての技術課題に影響を及ぼすようないくつかの「横断的話題」に焦点を絞ったワーキング・グループも作られた．これらの横断的グループは，実際，それぞれの技術グループのメンバーを必ず含むような形で構成された．横断的な話題として取り上げられたものは，以下の通りである：環境の統合，環境構築におけるプロトタイピング・アプローチ，計測と評価，単一言語環境と多重言語環境，人間とのインタフェイス，改善への漸進的アプローチと革命的アプローチ．

ワークショップの冒頭で，まず，これらの個別的および横断的話題が，ここで取り上げるにふさわしいものかどうか，が議論された．図２に示すように，その他の話題も数多く提案されたが，それらはあまり重要なものとは見なされなかった．とはいえ，それぞれの話題のあいだには，強い相互関係があるので，こうした付加的な話題も，全体討論やワーキング・グループの討論の中で，少なくとも部分的には取り上げられた．全体セッションは，各ワーキング・グループからの報告の場として用いられた．また，特に全員で考えるにふさわしい話題は，ここで取り上げられた．たとえば，現在および９０年代半ばの状況の性格づけ，いろいろな改善活動を実行する組織／機構の問題，５年前にカリフォルニア州ランチョ・サンタフェで開かれた類似の環境ワークショップの回顧，などである．

- **アニメーション**： 開発途中のソフトウェア・システムの動作を目に見えるようにする．
- **ソフトウェア・プロセス**： ソフトウェア開発・進化のための基本パラダイム．
- **履歴管理**： バックアップ，再スタート，取消等の動作を含むセッション情報の管理．
- **ソフトウェア・プロセス管理**： ソフトウェア開発／保守の過程に関する情報の管理，方法論上での制約の強制，バックアップなどの操作を含む，チーム活動に対する配慮が重要である．
- **グラフィックス**： グラフィックス技術を用いたソフトウェア・プロセス支援．
- **特定目的の環境**：
  - **タイミング／位置づけ**： 特定目的の環境の開発を，どのようにして，一般的な環境構築技術の研究開発の枠組みの中に組入れるか．
  - **部分集合**： 特定目的の環境と汎用環境はどうか．たとえば，特定目的の環境は，汎用環境の一部であるかどうか．
  - **調整と応用**： 汎用環境から特定目的の環境を作り出す技術．汎用環境の部品のプールの中から，「必要に応じて」特定の環境をどのように構築するか．
- **ソフトウェア・プロセスからの独立性**： 環境とそれが支援するソフトウェア・プロセスとの間の関係についての諸問題．独立性を保つための技術．独立性を維持しながら，特定のプロセスに関する知識をその上に盛り込むことによってもたらされるメリット．
- **移植性**： 異なった環境間で人間・ツール・情報を移動させることを可能にする．異なったホストに環境を移動できるようにする．
- **再利用性**： 環境の部品がどれだけ再利用できるか．
- **ツールの分解**： 大規模でモノリシックなツールを，具体化に即した部分に分解する．
- **部品のカタログ化**： 部品集合の保守．
- **再利用にもとずく開発**： 部品の再利用をして環境を開発する．再利用を支援するアーキテクチャ．
- **環境のための部品**： 再利用にもとずく環境開発，特定のパラダイムを支援する環境の需要に応じた開発，パラダイムの独立性の維持等のさまざまな目的のために，どのような部品をどのように使えばよいか．
- **ツール構築ツール**： ツール・ジェネレータや宣言的表現等を用いて環境構築者を支援する．

図２　直接取り上げられなかった話題

以下の各章では，このワークショップでの討論を体系的に報告する．すなわち，まず第２章では，ソフトウェア環境の現状と今後１０年間になされるであろう改善に関する参加者全体の意見を要約し，環境技術が抱えている現在の問題点とそれらを解決すべく提案された諸活動を理解するための基礎を設定する．各技術ワーキング・グループ，横断的ワーキング・グループ，および全体セッションでの討論は，それぞれ，第３，４，５章にまとめてある．以上のまとめは，ほぼワークショップ自体の構成に対応している．続く第６章では，これらの個別討論の中から浮び上がってきた一般的な問題点について報告する．最後に，第７章では，今回のワークショップの総括的な結論を述べる．

2. １９９０年代ソフトウェア技術の特徴予測

今回のワークショップでは，現在および将来の技術的状況について，さまざまな次元からの特徴づけが試みられた． ここで取り上げられた諸次元は，システムおよびソフトウェア製品，ソフトウェア開発・進化のパラダイム，自動化支援環境，ソフトウェア開発組織，および基礎技術，の５つであった．一般に，そうした特徴づけはきわめて困難であり，参加者の意見の一致は到底期待できないのではないかと予想されたが，結果的には，かなりの合意が得られた．

この章では，技術の現状評価および将来予測に関する参加者全体の意見を要約する．それぞれの技術ワーキング・グループおよびいくつかの横断的ワーキング・グループでは，担当分野での問題点を認識し活動項目を洗い出すために，それぞれの立場からの現状評価と未来予測を行った．ここでは，それらの成果もあわせて報告する．

2.1 現在（１９８５年）の技術状況

(1) システムおよびソフトウェア製品

今日のシステムおよびソフトウェア製品は，その複雑さの点で，大きな幅を持っている．他のシステムに埋め込まれ，その制御機能を提供するような形になっているものも多い．しかし，大多数は，スタンドアロン型である．製品の信頼性にはかなりの差があり，保守性や拡張性は限られている．移植性にも，やはり限界があり，分散処理の機能はほとんど用いられていない．製品の規模はバラエティに富んでいるものの，非常に大規模なものはまれである．多数のバージョンを作成し，保守することは，比較的よく行われている．

(2) ソフトウェア開発・進化のパラダイム

研究開発の最前線で考えられているパラダイムと，現場で実践されているものとの間には，大きなへだたりがある．いま研究コミュニティで検討されている最先端のパラダイムは，旧来のウォータフォール・モデルにプロトタイピングの概念を補強した変形版だとみなすことができよう．一方，実践の現場は，かなり無秩序な状況にあり，はっきりと定義された方法論を採用している組織は，ほとんど存在しない．一応の方法論が用いられている場合には，たいがい，プロジェクト規模がかなり大きく，そこでは，ウォータフォール・モデルにもとづく管理指標が採用されている．新しい応用分野や要求が不確定な場合など，特殊な状況下では，リスクを減少させるために，ラピッド・プロトタイピングも利用されている．

現在のソフトウェア・プロセスは，研究開発段階のものも，また，実践段階のものも，すべて「集中型」という特徴を持っている．集中型のプロセスは，集中型のシステムと同じく，かなり可視性が高い．すべてのものがすべての人間の目に見えるということが基準であり，こうした可視性を除去するためには，特別な手順を踏まなければならない．並行作業の管理，一貫性の維持，可変な情報への過度の依存などの理由から，必要以上に多くの情報が，複数の人間によって共有されている．

(3) 自動化支援環境

現存するほとんどの自動化されたソフトウェア環境は， Unix や Lisp またはこれらのシステムから派生し

た基本概念にもとづいている．そして，全体として，Fortran, Modula, Smalltalk, Ada など，広範囲の言語を支援している．

現在の環境は，データベースに関しては，最低限の支援しか提供していない．原始的な履歴管理の機構は存在し，また，並列性・統一性・一貫性に関する制御が得られる場合もある．しかし，大小さまざまな単位でモジュール化されたオブジェクトを総合的に管理する仕掛けは，ほとんど存在しない．さらにまた，大部分の環境データベースは，格納されたデータに対して，ただ１つの視点しか支援していない．一般的にいえば，現在の環境データベースは「受動的」であり，その制御は，主として自動化ツールや人間など，データベース自身の外部で行われている．

(4) ソフトウェア開発組織

現在のソフトウェア開発組織は，その規模も複雑さもまちまちである．規模については，数人の技術者が集まった小規模なカスタム・ソフトハウスから，複雑な大規模システムの開発を担当する数百人規模の企業内組織に至るまで，さまざまなバラエティがある．組織上の複雑さは，大まかにいって，その規模に比例しているが，会社の構造やプロジェクト管理のやり方などによって，複雑さが増す場合もある．どんな形の組織においても，その構造や管理は，理想的な状態にはなっていないように思われる．一言でいえば，現在の状況は，かなり「チーム・エントロピー」が高いといえよう．

(5) 基礎技術

現在のソフトウェア環境は，主として，中央のメインフレーム・コンピュータとダム端末を通じて対話するような形で具体化されている．いろいろなワークステーションも使えるようにはなってきたが，まだ一般には普及していない．

分散型の環境もいくつか存在するが，分散の程度はきわめて限定されている．複数のホスト（メンフレームおよびワークステーション）にまたがるような環境を支援するためのネットワーク技術も利用できるようにはなったが，まだその応用範囲は限られている．現在のネットワーク構造は，大きくいって，ＬＡＮとＷＡＮの２つのカテゴリに分かれる．前者は，近距離での高速なコミュニケーションとファイル転送を提供し，後者は，地理的に分散した開発者に対して，原始的なコミュニテーションと情報の共有手段を与える．ＬＡＮとＷＡＮの主なちがいは，コミュニケーションの周波数帯域と情報の分割単位とである．

人工知能は，ユーザ・インタフェイスなどいくつかの分野で，現在のソフトウェア環境に間接的に貢献している．しかしながら，ソフトウェア・プロセスへの支援という面では，ＡＩ技術はまだ十分に成熟しておらず，大幅な直接的貢献をもたらすには至っていない．また，現在のＡＩ技術は，大規模プロジェクトに応用するには，まだいろいろ問題があり，その意味で，ソフトウェアの開発・進化パラダイムの面での利用も限定されている．

2.2 将来（１９９０年代半ば）の技術状況

(1) システムおよびソフトウェア製品

１９９０年代半ばのソフトウェア製品が，その複雑さのバラエティや，埋め込み型要素としての利用のされ方，あるいはスタンドアロン型としての特性などの点において，極端に現在と異なっているとは考えにくい．最も大きな変化は，おそらく，複雑さの程度が飛躍的に増大するという形で発生するだろう．過りが許されないような分野でのソフトウェアの利用が拡大することで，信頼性に対する要求がより強くなり，製品の信頼性はそれだけ向上する．しかし，それでもなお，信頼性にはかなりのバラツキが存在するものと予想される．保守性や拡張性も改善されるだろう．１９９０年代半ばにおける応用システムおよびソフトウェアは，高度な分散化の傾向を示すようになり，また，エキスパート・システム技術がかなり利用されるようになるものと考えられる．しかし，ホスト・システムの種類が増加するために，移植性は今日のシステムよりも悪くなるだろう．１９９０年代のシステムでは，規模のバラツキが広がり，きわめて大規模な応用がより一般的になってくる．また，開発・保守の対象となるバージョン数は，現在のシステムよりもさらに多くなるだろう．

(2) ソフトウェア開発・進化のパラダイム

１９９０年代半ばにおいても，ソフトウェアの開発・進化の諸活動が無秩序な形で行われることは，やはり一般的であるだろうが，より多くの組織が，系統的なアプローチを採用するようになるにちがいない．その意味では，研究開発と実践との差は，いくらか縮まるものと期待される．１０年後に一般的になっているであろうパラダイムの有力候補としては，プロトタイピングで補強された現実的なウォータフォール・モデル，プログラム変換を中心とするフォーマル・アプローチ，再利用にもとづくソフトウェア・プロセス・アプローチなどが考えられる．

ソフトウェア・プロセスに対する将来のアプローチは，集中型ではなく，分散型のシステムに近くなるであろう．それらのアプローチでは，プロセスの可視性はいまより低くなる．情報の共有形態は，各個人やチームがそれぞれ固有の情報を管理し，プロセスおよび成果物の特定の局面に関する情報を別々に格納する形の，ゆるやかに結合されたソフトウェア・システムのスタイルに近くなるだろう．情報は，十分に定義されたプロトコルにしたがって共有され，並行性の管理や，一貫性および完全性の維持などの問題が軽減される．また，共有すべき情報を注意深く選択することにより，成果物の変更による影響を，より効率的に局所化することが可能になる．

(3) 自動化支援環境

将来のソフトウェア環境は，ワークステーションを基本とするものに変わって行くだろう．個人用ワークステーションの採用により，それぞれの作業に合わせて環境をカストマイズすることが容易になり，特殊目的の専用環境がどんどん作られるようになる．これらの専用環境で，プロジェクトの管理や，システム設計，保守などの，専門的な活動が支援されるようになる．こうしてワークステーションを専用化するためには，分散型の異機種アーキテクチャが環境のホストとして想定される必要がある．

１９９０年代の初期から半ばにかけてのソフトウェア環境では，より高度なデータベース支援が可能になる．すなわち，より複雑なデータ・モデル，一貫性および並列制御のメカニズム，ユーザに対する複数の視点の提

供，モジュール化のレベルが異なるオブジェクトの一括管理，主要なプロセスおよび成果物の履歴管理支援，などの機能が一般的に利用できるようになる．デーモンやトリガなどのメカニズムを利用して，ツールとデータベースをより密接に統合化することにより，環境データベースは受動的な存在ではなく，能動的にデータベースの内容および構成の管理に「参加」するようになる．

(4) ソフトウェア開発組織

１９９０年代半ばのソフトウェア開発組織の規模や複雑さは，たぶん現在と同様にまちまちであろう．しかし，高度なコミュニケーション手段など，あちこちに散らばった開発者グループを支援する技術が実用化されることによって，プロジェクト・チームは，地理的により分散化するようになるものと考えられる．こうした分散型のプロジェクト編成は，１９９０年代の前半にはまだかなり高いまま残る「チーム・エントロピー」の改善に役立つだろう．

(5) 基礎技術

中央のメインフレーム・コンピュータを使用して環境を具体化する方法は，やはり，基本的なアプローチとして残っているだろう．しかし，ワークステーションをスタンドアロン・コンピュータとして，あるいは中央システムのインテリジェント端末として利用することが，より重要な意味を持ってくる．将来のワークステーションは，高解像度カラー・グラフィックスや音声入力，より大きなスクリーンなど，ユーザ・インタフェイスの面で大幅に改善されるだろう．人間とコンピュータのコミュニケーションに対する理解も進み，これらの機能が効果的に使用できるようになるものと期待される．

将来の環境では，分散システム技術がより広範囲に利用されるようになるだろう．ユーザとワークステーションの作業を支援するための機能は，いちじるしく改善されよう．しかし，ＬＡＮとＷＡＮのちがいは残り，同じオフィス内の人間と対話するのと，遠隔地の人間と対話するのとでは，やはり明らかに事情が異なるだろう．

１９９０年代半ばの環境では，人工知能技術がより直接的に利用されよう．最も有望なのは，知識表現と設計モデルの２つである．知識表現は，プログラミング技術だけでなく，問題分析や仕様化にも有用である．また，完成度の高い設計モデルは，ユーザに対する環境側からの支援機能を改善する．

１９９０年代半ばまでには，各応用分野別の特殊な仕様記述言語が，完成の域に達するだろう．これらの言語によって，高水準言語によるシステム仕様の記述と，その仕様を効率てきなコードへ変換することが可能になる．小規模から中規模プロジェクトで，そうした半自動的な変換を支援するツールが現実のものになるだろう．

3. 技術ワーキング・グループでの討論

技術ワーキング・グループでは，それぞれの話題に関連して，特定の活動項目を洗い出すための討論が行われた．各グループは，短期（８０年代後半）および中期（９０年代前半）の活動項目を区分して確認するとともに，それらの背景にある基本的な問題点を明らかにし，各項目の実施時期や優先順位についても検討を加えた．

3.1 ツール

現在利用できる個々の環境は，比較的範囲が限定されている．一般的に，既存の環境は，ソフトウェア・ライフサイクルの一部をカバーするだけであり，ソフトウェア・プロセス関係者のうち，ほんの一部の要求に応えているにすぎない．また，ソフトウェア・システムの諸特性のうち，限られた側面についての意志決定を支援しているだけである．

しかし，現存する自動化ツールを集大成すれば，ソフトウェア・ライフサイクル，要員，意志決定ニーズなどに関して，非常に広い範囲をカバーしている．したがって，現状における問題は，実用的なツール集合がカバーする範囲の幅を広げることではなく，利用可能なツール群をどんな形で統合すれば，それらが全体として広範囲な一貫した自動化支援を提供しうるかを，考えることである．

3.1.1 問題点

ここでの重要な活動項目は，ツール群を集めて環境を構成するプロセスを容易にすることに関連している．拡張容易性や統合化のレベルなど，環境の持つ個々の特性は，このプロセスおよびそれに対するわれわれのアプローチに，かなりの影響を与える．しかし，これらのことがらから派生する問題点は，他の技術グループおよび横断的グループの報告で扱われる．ここでは，どのような自動化ツールが利用可能であるか，および，別々に開発されたツール群を１つの環境にまとめあげるための能力をどうすれば改善できるかという点に焦点を絞った形で，討論が進められた．

(1) どのようにツールを構造化し汎用的なユーティリティ部品に分解するか

環境ユーザの目に見えるツール群は，一般に，数多くの共通な内部機能を持っている．たとえば，大部分のツールでは，ソフトウェア記述を解析して，何らかの内部表現に変換している．したがって，ツール・フラグメントを環境全体で再利用できる可能性は，比較的高い．UNIX 環境は，多数のツール・フラグメントを用意し，それらを拡張し，相互に組み合わせるための手段を提供するという形で，この点の改善に力を注いでいる．

ここでの問題点は，分解の単位，ツール・フラグメントの汎用性および機能性，ツール・フラグメント集合の構成管理のメカニズム，新規ツールの開発に関連して再利用できるツール・フラグメントの識別を可能にするようなそれらの記述法，などに関連している．このことは，単にツールの機能分解のみならず，環境開発にさいして高度の再利用性をどうやって達成・支援するか，あるいは，さまざまな環境開発アプローチを意味のある形で支援するにはどうしたらよいかという，一般的な問題をも含んでいる．

(2) ツール・フラグメント，ツール，およびツール集合間のインタフェイス

ツール・フラグメント，ツール，およびツール集合をうまくまとめあげるためには，インタフェイスに関する取り決めが必要になる．また，いくつかの環境（たとえば開発用環境と保守用環境）を一緒に使うためには，それらの環境間で成果物のインタフェイスに関する取り決めが必要になる．この取り決めは，やりとりされるデータの構造を決定するものである．また，情報交換の方法（たとえば，パラメータ方式かメッセージ交換か，など）を確立し，ツール・フラグメント，ツール，ツール集合，および環境の間の一般的な時間制御の流れ（たとえば，起動順序の制御か，非同期的なメッセージ・フローか，など）を決定するものである．

(3) どのような自動化ツールを設計工程以前の作業に提供すべきか

現在のツールの大部分は，ソフトウェア・ライフサイクルの下流工程を支援するものである．その中で，いくつかのツールは要求定義や設計用に拡張されており，また，これらの上流工程を支援する新しいツールも開発されつつある．しかし，要求定義や設計工程において，具体化工程で現在行われているのと同程度の支援を提供するためには，より一層の努力が必要である．それには，もちろん，既存のツールをどのようにして変更または拡張すればよいかが，検討される必要があるだろう．また，さまざまなソフトウェア・プロセス・パラダイムのもとで，どのような自動化支援が上流工程において最も効果的であるかも，十分考えられなければならない．さらに，既存ツールの拡張や新規ツールの構築とは無関係に，われわれは，上流工程における問題を解決する上で役立つようなソフトウェア抽象化技法を開発し，そうした技法に対する自動化支援の基礎を確立し，ツール集合の高度な統一性が達成できるように，努力する必要がある．

(4) 他のツールや環境の構築支援にはどんなツールやアプローチが必要か

ツールや環境の構築を支援するために，これまで，数多くの技術（たとえばパーサ・ジェネレータ，宣言型記述の利用，階層型アーキテクチャなど）が開発されてきた．しかし，より一層の支援が必要である．パーサ・ジェネレータなど，現存するいくつかの機能は，拡張して他のツールの構築に利用することができる．しかし，環境の構築・進化に関する新しいパラダイムの出現とともに，より高度なツール構築支援機能が必要になってくるであろう．

(5) ツールおよび環境の費用対効果をどうやって決定するか

現在，ツールや環境が広く使われていない最大の障害は，それらの利用効果が必要な費用を上回ることを，われわれがうまく説明できないでいることである．今後，より高度な新技術が開発されるにつれて，この問題はより深刻になるだろう．経験にもとづく説明は，もちろん重要である．しかし，費用対効果の算定をより明確に行なう方法が開発されなければならない．費用は，単純に経済的なものだけではない（たとえば，極端な習熟曲線を持つツールもある）．したがって，経済的な要因も含めて，ツールや環境の導入にともなう「苦痛」と，それによってもたらされるさまざまな利益とが，正確に把握されなければならない．

効果対苦痛の定量的評価は，いくつかの要因が組み合わさった複雑なものになるだろう．たとえば，目的の応用分野やソフトウェア・プロセス・パラダイム，ユーザや管理者および組織の期待などが，いろいろな影響を与える．これらの要素はお互い複雑に関連しあい，要求や期待の高度化につれて変化する．また，すでに認識されたコミュニティ・ニーズ（いわゆる Technology "Pull"）に対し，どちらかといえば消極的な形でデモ

ンストレーションを行うのに加えて，技術を積極的に売り込む（いわゆる Technology "Push"）ためにも効果対苦痛の評価機能が必要になることで，この複雑さはさらに増加する．したがって，定量的な尺度の開発は，それだけでは十分ではない．その評価機能は，本質的に定性的な要因も説明できるものでなければならない．また，時間の経過とともに評価をより洗練されたものにするプロセスも，考えられなければならない．

3.1.2 活動項目

どのような活動項目が必要かを洗い出すにあたって，このワーキング・グループでは，現在利用可能なツールの機能を段階的に改善することに焦点を絞った．上に述べた基本的な問題点に関する進歩は，中期の目標と見なすしかないと考えられた．集中的な改善活動を開始する以前に，より多くの経験を蓄積する必要があるため，短期の活動項目によってもたらされる成果は少ない．

したがって，５年間の短期目標は，既存の自動化支援ツールの改良に置かれる．１０年間の中期目標は，支援対象プロジェクトの範囲を拡大して，複雑・大規模・高信頼性・多バージョンのシステムを扱えるようにし，現実のプロセスをより適切に反映するようなパラダイムを支援できるようにすることである．

活動項目は，その結果としてもたらされるツールの種類にしたがって，３つのグループに分類できる．最初のグループには以下のものが入る：

- **テスト環境**：動的解析によってソフトウェア・システムの動作を分析するツールの集合．
- **バージョン管理**：複数バージョンのソフトウェア・システムを管理するツールの集合．

- **進化型および調査型のプロトタイピング**：段階的なソフトウェア・システムの開発と要求定義確定のためのプロトタイピング支援ツールの集合．
- **ツール構築ツール**：他のツールや環境の構築を支援するツールの集合．
- **複数言語による開発**：ソフトウェア・システム中の異なる部分を異なる言語で開発するための支援ツールの集合．

これらの作業を支援する自動化ツールで，かなりの機能を持つものが，すでに存在する．しかし，上にあげた短期活動項目は，それらのツール機能を改良し，使用評価の経験を積み，不足している機能に対する要求条件を確定し，現状でどのツールがこれらの要求をどの程度満たしているかを確認するために，どうしても必要である．また，より広範囲なソフトウェア・システムに対する効果的に支援を確立するための基礎でもある．

第２のグループに属する活動項目に関しては，短期の調査研究も可能であり，また有益であると考えられる．しかし，実用的なツールなど，具体的な成果があらわれるには，４ないし６年の期間が必要だ思われる：

- **実験型プロトタイピング**：いろいろな代替案に関する経験的な調査を行うためのプロトタイピング支援ツールの集合．
- **再利用性**：再利用可能なソフトウェア部品の作成・管理・検索・収集を支援するツールの集合．
- **仕様解析**：具体化以前の記述を検査してソフトウェア・システムの特徴や性質の分析を支援するツールの集合．
- **ツールの統合化**：ツール集合や環境の統一性を高めるためのツールや技法．

・ **無秩序への支援**：かなり無秩序な形で進められる自然なソフトウェア・プロセス・パラダイムを支援するツール群.

・ **状態の計測と情報収集**：ソフトウェア開発・進化の過程をモニタし評価する作業を支援するツール群.

最後のグループに属する活動項目は，方法論やコンピュータ・サイエンスの理論など，他の分野の発展を待たなければ，本格的な検討ができないようなものである．したがって，これらの活動項目は，今後約８ないし１０年は中心的なものにはならないと思われる：

・ **抽象化支援**：特にソフトウェア・システムの要求定義と設計における，抽象的記述の作成と解析を支援する技法と関連ツール.

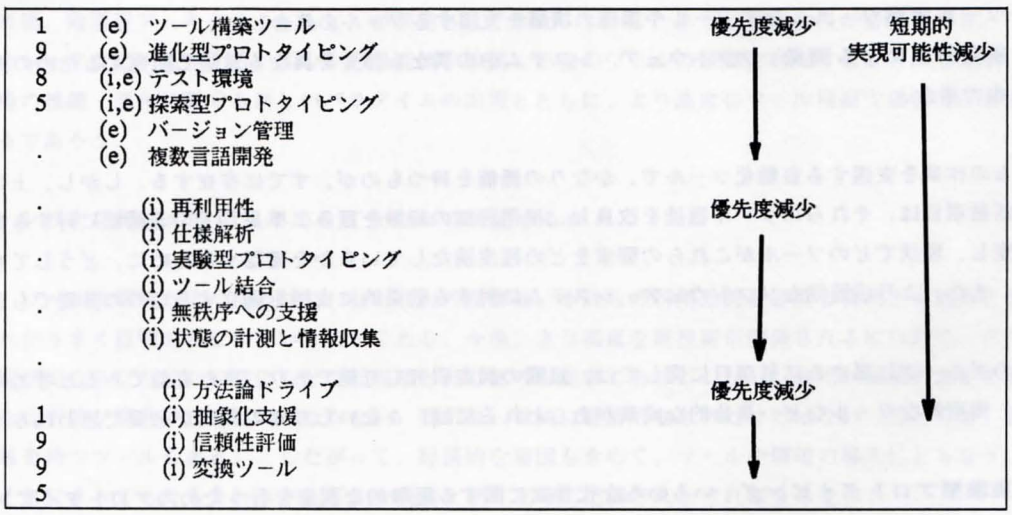
・ **信頼性評価**：特に高い信頼性を要求されるソフトウェア・システムの開発や進化を支援する技法とツール群.

・ **変換ツール**：さまざまな関係者からの相異なる要求に対応して，開発あるいは進化途上のソフトウェア・システムに対する異なる視点からの記述を提供するツール群.

・ **方法論ドライバ**：特定のソフトウェア・プロセス・パラダイムの使用を積極的に奨励あるいは強制する自動化ツール群.

3.1.3 優先順位とタイミング

活動項目のグループ化によってある程度の優先順位は定まっている．ワーキング・グループでは，さらに一歩進んで，各グループ内の活動項目の優先順位を定めた．その結果を図３に示す．ここでは，各活動項目を，それらがどのタイミングで最も効果的に実行できるかを大まかに評価した結果にもとづいて，優先順位および実行可能性の順序に並べてある.



凡例：e －現在の技術を拡張することで実現する
i －基本的問題として研究を行う

図３　ツール分野における活動の優先順位とタイミング

3.2 分散化

このグループでは，ソフトウェア環境の分散化を議論するにあたって，関連項目を次の３つに分類した：

- 分散システムの開発・進化を支援する環境機能の提供
- 環境機能自体の（論理的または物理的）分散化の実現とその支援
- （論理的または物理的に）分散化された環境を用いて分散システムを構築するさいに，ホストと環境機能をどのように組み合わせるべきかについての問題

時間的な制約のため，ここでは，２番目の話題に絞って討論が行われた．したがって，以下に述べる問題点および活動項目は，環境を１つの分散システムとして論理的に認識する方法，および，分散型のホストをソフトウェア環境に利用する方法に関するものである．

3.2.1 問題点

(1) ワークステーションのネットワーク

１９９０年代における環境の多くは，ワークステーションにもとづくものになる．ここでの主な問題点は，卓上型のコンピュータ資源を用いて複数メンバのプロジェクトを支援することである．そのさい配慮すべき重要なことがらは，チームの大きさである．６人，６０人，６００人のチームに対して，それぞれ，ワークステーションにもとづく開発を支援することができるだろうか．また，関連する問題点として，ワークステーションの特殊化と専用環境の派生ということがある．ワークステーション群（または複数のワークステーション集合）によって提供されるハードウェアおよびソフトウェア環境は，ソフトウェア・プロセス内の特定の役割にあわせてカスタマイズされるべきであろうか．

(2) 異機種間ネットワークの支援

個々のワークステーションを特定の目的に利用し，それぞれの専用環境を構築して行こうとすると，どうしても異機種間ネットワークが必要になってくる．ここでの主な問題点は，メインフレームや特殊なプロセッサ，および専用ワークステーションから成る雑多なハードウェアの集合を，どうやってネットワーク内で支援するかである．

(3) 分散環境のアーキテクチャ

論理的または物理的に異機種コンピュータ資源の集合から構成された環境には，基本的にどのような構造が適しているのだろうか．たとえば，階層型アーキテクチャは，分散環境に適切であろうか．いろいろなアプローチを比較するには，アーキテクチャの適切性を測るための評価基準を，分散という特性に則して定める必要がある．また，分散化の支援は，完全性や拡張性など，環境が持つ他の特性にも影響を与える．分散化を支援する活動にさいしては，これらのことがらも考慮しなければならない．

(4) 環境機能の分散資源への配置

これは，環境の持つべき機能を，分散された（おそらくは異機種の）ホストによって提供されるプロセッサに，どうやって割り当てるかという問題である．たとえば，環境は，ツールの起動に関して，分散型のプロセス・スケジューリングを支援すべきだろうか．その場合，スケジューリングは，プロセスやユーザから可視的にすべきだろうか．また論理的なことがらも考慮して設計する必要がある．たとえば，ユーザの役割などの基準にしたがって事前に機能割り当てを行なうべきか，あるいは与えられた機能に対するニーズにしたがって動的な割り当てを行なうべきか，それとも，両者の混合で行くべきか，など．

(5) 柔軟性

ここでの柔軟性とは，ソフトウェア開発プロセスの変化や環境を構成する分散資源の変化に対応して，分散環境を論理的に再構成する能力をいう．もう１つの注目すべき機能は，それが論理的に分散型であるかどうかにかかわらず，１つの環境を集中型のホストから分散型のホストへ移行する機能である．ここで中心となる問題点は，どの程度の柔軟性を支援すべきかということ，および，そこでの分散化アプローチの役割およびメカニズムである．

(6) 分散環境上における共同作業

分散環境の各構成要素間の相互調整を行うには，どのような機能が必要であろうか．環境ユーザ間のコミュニケーションに対しては，どのようなモデルや手法，メカニズムが最適なのだろうか．電子メイル，複数のユーザ間の同期的コミュニケーション（オンライン・ミーティング），共有オブジェクトの変更などが，その候補として考えられる．どのような種類のコミュニケーションが適切かは，待ち時間によっても影響される．したがって，ＷＡＮ上の分散型環境は，ＬＡＮだけを利用したそれとは必然的に異なった機能を持つことになる．

3.2.2 活動項目

分散環境に関する活動項目を検討するにあたって，このグループでは，次のような特徴を持つ分散環境のプロトタイプを開発するプロジェクトを設定するのが，最も有用であると考えた：

- **地理的に分散したノード**：そのプロトタイプは，ＬＡＮだけでなく，（UUCP や ARPA などの）ＷＡＮ上に分散した環境も支援する．
- **高度な再構成機能と適応性**：そのプロトタイプは，下部構造としてのハードウェアのバラエティを支援すると同様に，さまざまなソフトウェア・プロセスやプロジェクト組織を支援メカニズムを持つ．

以下にあげる３系列のプロジェクトが提案された：

(1) 環境アーキテクチャの研究

分散環境アーキテクチャを研究するプロジェクトが明確にすべき項目としては，以下のものが考えられる：

・環境の機能および資源の分散

・環境資源の再構成
・ソフトウェア・プロセスの分散型支援のモデル
・特定プロジェクトの分散型支援のモデル

(2) コミュニケーション・モデルとメカニズムの研究

この分野のプロジェクトでは，ユーザ相互およびユーザと環境間における高度なコミュニケーションを可能にするモデルやメカニズムを研究すべきである．

(3) 柔軟なアーキテクチャと高度なコミュニケーション能力を持つ環境群の構築と評価

ここでのプロジェクトは，上記２つの分散環境構築プロジェクトの流れにしたがった各種のアプローチを統合化するものであり，そうした評価は，１つの現実的なプロジェクトを通じて行われるのがよい．

3.2.3 優先順位とタイミング

最初の２系列のプロジェクトが，第３系列のプロジェクトの前提として，まず実施されなければならない．グループ討論のなかでは，最初の２系列のプロジェクトを並行的に進めることで，５年以内にかなりの成果が得られるものと考えられた．その後で，それらの成果を統合して，柔軟なアーキテクチャを持ち，高度なコミュニケーション能力を持つ環境群を作り出すことが可能になる．

3.3 データベース支援

最近まで，データベース支援は，ソフトウェア環境がいったん構築された後の追加機能程度に扱われてきた．したがって，この分野における改善活動の目標は，データベース支援が，ソフトウェア環境開発の中心的な課題として考えられるようにすることである．この分野におけるプロジェクトでは，対象とする問題，問題領域，解決法，ソフトウェア・プロセス，プロジェクトの資源や履歴，開発組織など，非常に幅広い範囲の情報に関する格納・管理・アクセスを支援する機能が開発されなければならない．

3.3.1 問題点

(1) 適切なデータ・モデル

データベース支援の提供にさいして，まず第１に重要なのは，ソフトウェア環境において使用すべき適切なデータ・モデルが何かをはっきりさせることである．関係型データ・モデルなど，現在の多くのデータ・モデルは，エンティティの種類が比較的少なく，これらのエンティティに対するインスタンスが多い場合に，最も有用である．こうした条件は，MIS や ADP の分野では十分に成立し，したがって，既存のデータ・モデルが活用されている．一方，ソフトウェア環境におけるデータは，エンティティの種類が多く，各エンティティに対するインスタンスの数は，１ないし数個の場合もあれば，多数の場合もあって，さまざまである．したがって，全体的に見て，既存のデータ・モデルは，ソフトウェア環境にとっては十分ではない．

データ・モデルに関する一般的な問題点は，以下の通りである．もちろん，これらは相互に関連している：

- **CAD/CAM とソフトウェア環境の類似点（相違点）**：CAD/CAM 環境は，ソフトウェア開発環境よりもかなり進んでいる．これらの環境は，多くの場合，特殊なデータベース支援機能を持っている．はたして，CAD/CAM 環境のデータベース・モデルは，ソフトウェア環境でも有効に利用できるだろうか．
- **汎用データモデル**：関係データ・モデルは，MIS や ADP に対しては，よい汎用データ・モデルである．ソフトウェア環境に適した汎用データ・モデルは存在するのだろうか．ソフトウェア環境固有の要求に合うように，既存の汎用モデルを拡張・制限・変更することができるだろうか．
- **多様なユーザ「視点」の支援**：ユーザが異なれば，環境に保存されているデータに対する見方も異なってくる．これらの視点は，マネジャ，アナリスト，プログラマ，保守要員など，ユーザの役割に依存している．環境ユーザ全体を支援するためには，どのような概念データ・モデルが必要であろうか．
- **概念モデルと具体化モデル**：概念データ・モデルは，具体化のさいに使用される下部のデータ・モデルとは異なるであろう．概念モデルと具体化モデルの根本的な相違点，類似点，相互関係はどうなっているのだろうか．
- **複雑さ**：ソフトウェア環境の支援に必要なデータは非常に複雑である．大きさが異なる数多くのオブジェクトが存在し，これらの間の関係も複雑である．ほとんど変化しない永続的な性質を持っていると考えてよいものもあれば，たびたび変化し動的に取り扱うべきものもある．データ処理内容の複雑さも大きく異なっていて，しかも，これらの異なった処理を行うツールが，すべて同一の論理データベースをアクセスしなければならない．このような極端な複雑さをうまく処理するには，どんな汎用メカニズムやアプローチが有効だろうか．ツールの統合化とデータベースの統合化の関係はどうなっていて，どうすればこの２つの問題を同時に解決できるのだろうか．
- **制約メカニズム**：多くのユーザが異なる視点からデータにアクセスするため，データの一貫性や完全性が失われる可能性がある．これは，分散型環境では特にむずかしい問題になる．データの一貫性や完全性を保つためには，適切な制約条件と，デーモンやトリガなど，関連するメカニズムを開発する必要がある．

(2) データベース支援の役割

もう一つの重要なことがらは，ソフトウェア環境内におけるデータベース支援の役割である．データベースと他の環境機能とを論理的に統合化することが，重要な課題である．強い統合化を行うには，ソフトウェア・プロセスを制約し，同時にまたプロセスによって制約されるような，何らかの新しい技術が必要になる．弱い統合化の場合には，多少とも標準的なデータベース技術が利用できるが，支援できる機能はその分だけ小さくなる．その他の問題点としては，次のようなものがある：どのように情報を分解すべきか，どんな情報を獲得し管理すべきか，分解の大きさはどの程度が妥当か，ツールや環境間の情報交換を容易にするにはどんな情報構造やインタフェイスが最適か．

(3) その他の問題点

その他，特にソフトウェア環境に固有とはいえないような問題点も，いくつか取り上げられた．データ管理の作業を，ソフトウェア環境から分離することはむずかしい．論理的に集中化されたデータと個々のユーザデータの間には，一般的な組織のデータとプロジェクト特有のデータの間と同様に，ある種の競合関係が存在する．また，性能や，ハードウェア支援，機密保護，安全性などの問題も，考慮しなければならない．

これらの問題点を明確にする上での１つの大きな障害は，われわれ自身に，ソフトウェア環境上で本格的なデータベース支援を試みた経験がないことである．データ・モデルのプロトタイピングや，核となるデータ・

操作・制約条件の識別に関しては，ほとんどガイドラインらしきものが存在しない．

3.3.2 活動項目

ソフトウェア環境におけるデータベース支援機能を強化するための活動は，分散化および集中化という２つの方法で，同時に試行する必要がある．ここでいう分散化と集中化は，概念データ・モデルに関するそれである．実際に具体化された形態は，どちらの場合も，もとのモデルとはかなり異なってくる．

分散化および集中化の双方のアプローチによって試行しうるプロジェクトの案をいくつか示しておく：

- 一貫性の維持や並列制御などのデータベース機能を支援するツールの開発．
- データベース機能のソフトウェア環境への統合化．
- ソフトウェア環境に適したデータ・モデルと，それを支援するデータベース機能の開発．
- 複数の視点をユーザに提供する機能の開発．
- データ・モデルの統合化．
- ソフトウェア環境で利用するための汎用データベース・ツールのカストマイズと，結果の環境への統合．

3.3.3 優先順位とタイミング

(1) 短期目標

今後５年間に有用な研究開発成果をもたらしそうな活動項目を以下に示す：

- ソフトウェア環境向けのデータベース・ツールの開発．
- データベース・ツールのソフトウェア環境への統合化．
- ソフトウェア環境のための単純なデータ・モデルの開発．

(2) 長期目標

かなり大規模な研究が必要であり，したがって１９９５年以前には有用な成果が得られそうにないとおもわれるものを以下に示す：

- ユーザに対する複数視点の支援
- より複雑なデータ・モデルの開発
- データ・モデルの統合化
- データベース・ツールの特殊化または統合化

3.4 拡張性

要求や技術の変化に対して，環境やその構成要素を適応させることは，必要不可欠である．また，そうした適応性によって，いくつかの潜在的なメリットがもたらされる．その第１は，新しいツールを一から作るのではなく，既存のツールや環境の大部分が再利用できることである．環境の適応性が高ければ，将来要求や技術

の変化があっても，投資効果が無駄にならないことが保証され，それによって，多額の初期投資が可能になる．

さまざまな変化の源泉が，相異なる適応性要求をもたらす．図４（[Keen[80]から抜粋）に示すように，変化には３つの大きな領域がある．

- ツールや環境のユーザの側における要求の変化．たとえば，リアルタイム・システム分野への移行など．
- ホスト・システムの能力の変化．たとえば，新しいハードウェアの出現など．
- 環境およびツールの構築者が利用できる技術の変化．たとえば，新しいバージョンのツールの出現など．

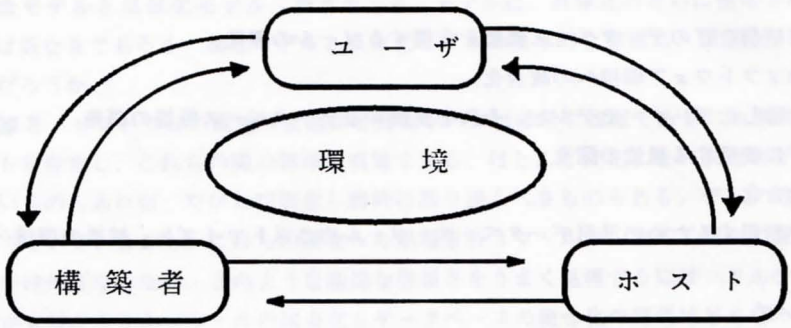


図４ 変化の領域

この図は，ある領域における変化が，他の領域で２次的な変化を引き起こす可能性があることを示している．たとえば，ユーザの要求の変化によって，新しいより高性能のホスト・システムが必要になる．さらに，その次には，環境を構築するために，より高度な技術を開発し利用することが必要になってくる．

ホスト・システムの変更に適応する能力は，ふつう，**移植性**と呼ばれており，１つの環境を何種類かのホスト・システム上で利用する必要がある場合，基本的に重要な特性である．すべての問題が解決されてはいないが，移植性を向上させるためのいろいろなメカニズムやシステム・アーキテクチャは実用化されている．

ユーザや環境構築者の側における変化に適応するための能力の開発は，ずっと遅れている．これらの変化によって必要となる修正が，それほど大きなものでなく，比較的ちいさな改良である場合は，適応性は比較的容易に達成できる．**カスタム化**（環境やツールの応答をユーザ特有のスタイルに修正する）は，その１つの例である．別な例としては，**チューニング**（環境やツールの信頼性・性能・堅牢性などを改良する）がある．カスタム化やチューニングの能力を処理し強化するためのメカニズムやアーキテクチャは，これまでいくつか開発されてきたが，機能的に十分成熟しているとはいえない．

環境やツール機能に大きな変更をもたらすような，ユーザや環境構築者側の変化に対する適応性，すなわち**拡張性**は，適応性の諸側面の中で，もっともわれわれの理解が遅れている部分である．この種の変更は，開発または進化の過程にあるソフトウェアに対して，何らかの新しい，これまでとはちがった視点が必要になることによって生じる（たとえば，プロジェクト要員の交代，要員の役割の変更，組織変更，業務内容の変化などが，その原因である）．また，新しいユーザの活動を支援するために，あるいは，開発上のまったく新しいパラダイムを支援するために，新しい機能を提供する必要が生じ，それによって変更が行われることもある．最後に，新しい高度な技術を用いて環境やツールの機能を作り出すために，変更が必要になることもある．

3.4.1 問題点

上に述べたすべての種類の適応性を支援できるようになるまでには，かなりの技術進歩を待たなければならない．現在，最も要求が大きいのは，環境やツールの拡張性である．拡張性に関する一般の関心やニーズの高まりは，それを支援するための技術が，移植性，カスタム化，チューニングなどと比較して，未成熟なためであろう．また，さらに重要なことは，この種の適応性を改善することの見返りがきわめて大きいことである．したがって，以下に定義する問題点は，環境およびツールの拡張性強化を中心としたものになっている．

(1) 拡張性を強化するようなアーキテクチャとその特徴は何か

移植性を高める伝統的な方法は，階層型のアーキテクチャを用いて，機械依存部分を最下層のカーネル内部に局所化することである．他の階層でも，それぞれ密接に関連しあう要素やフラグメントのカプセル化が行われている．これにより，変更の範囲や影響を局所化できるため，構成要素の交換がやりやすくなり，その結果，拡張性が改善される．たとえば，いくつかの環境は情報管理のための階層を持っており，その基礎を構成しているデータベース管理システムを変更しても，システムの他の部分へ混乱をもたらすことはない．ここで最も重要な問題点の第１は，どんな階層構造が最も適切かということであり，第２は，階層型アーキテクチャに付随するトラブルを解消し，しかも，システムの拡張性を同程度に保てるやり方が他にあるかということである．

(2) 基本的な環境やツール要素はどうすれば再利用できるか

環境やツールの強化に必要な労力は，主な部品が再利用できれば，いちじるしく軽減される．また，環境アーキテクチャを変更する必要がなければ，やはり，かなりの労力削減になる．基本的に必要なのは，新しい状況下でもそのまま利用できるような，十分に汎用的な部品（またはツール全体，あるいは環境）を持つことである．たとえば，基本設計や詳細設計の評価にさいして，もしスタティク・プログラム・アナライザが再利用できれば，それは明らかに望ましいことだが，そのためには，スタティク・プログラム・アナライザを定義し具体化する時点で，かなりの先見の明が必要になる．その他に必要な項目としては，再利用可能な部品を識別しカプセル化する技術，部品の利用可能性を判定し部品格納庫から検索できるようにそれを記述するスキーマ，既存の部品や新しく作った部品を用いてツールや環境を構築する作業を支援するためのツール，などがある．

(3) ツール集合を相対的に独立に保つにはどうするか

階層型のアーキテクチャは，仮想機械と具体化のための境界線を独立に保っている．しかし，これらの境界線は，ツールの具体化にさいしては，必ずしも適切なものではない．個々のツールやツール集合が，この境界をクロスするといったことも，ときには起こりうる．ツール機能のカプセル化が，環境機能のカプセル化と矛盾する場合もありうる．

ツール機能のカプセル化によって得られる交換可能性は，個々のツールやツール集合を相対的に分離することによって，より強化される．単純なアプローチは，ツール間またはツール集合間のデータ交換インタフェイスを定義することである．しかし，かなり洗練された定義機能があったとしても，単に同一機能のツールを交換できるだけであり，したがって十分とはいえない．その他に必要な項目としては，ツールやツール集合を適切に抽象化した集合，動的交換を支援するツール，環境の使い勝手や環境の支援するソフトウェア・プロセスをほとんど（あるいはまったく）損なうことなしに，ツールの交換を行う方法論，などがある．

(4) ユーザを混乱から遮蔽するにはどうしたらよいか

現在の環境は，ほとんどがコマンド形式のインタフェイスを用いており，環境の詳細が，ツールの集合として外部から見えてしまう．個々のツールはユーザの目に見え，その結果，ツール部品の変更も目に見えるようになっている．ある意味で破れかぶれである．こうした事態を避ける方法に，オブジェクト指向インタフェイスがある．この型のインタフェイスでは，ユーザは，環境との対話を，ツールの名前ではなく，オブジェクトの名前で考える．この方法によれば，環境内部の詳細なツール構成を外部の目から隠すことができ，ユーザ・レベルの操作は自動的にツールの起動へ変換できる．こうした高度なカプセル化は，ツール部品の変更などによって生じる不必要な混乱から，ユーザを守る働きをする．特に高い優先度を持つ活動ではないが，オブジェクト指向インタフェイスをはじめとするユーザ遮蔽のためのアプローチを研究することは，拡張性の問題に，完全かつ効果的な回答をもたらすためには，どうしても必要である．

(5) どうやってユーザ・インタフェイスを拡張可能にするか

ここまでに述べてきた項目は，ユーザ・インタフェイスの拡張という，より一般的な問題の一部である．拡張可能な記述法や仕掛けを提供することにより，ユーザ自身が環境を自分の要求やスタイルに合うように，カスタム化できるようになっているのが望ましい．また，インタフェイス自身，対話のメディアが変わっても（たとえば，メニューにもとづくのインタフェイスからアイコンにもとづくのインタフェイスへ），その余波が環境自体へ及ばないような形になっていなければならない．

(6) ツールやツール集合の変更を迅速にするための支援機能とはなにか

コンパイラ技術は，コンパイルされる言語やターゲット・マシンが変わっても，コンパイラの実際の具体化を大幅に変更しないでもすむような支援機能を提供している．この支援機能の中心的要素は，命令型記述とは対照的な宣言型記述（テーブルなど）である（命令型の記述では，記述情報はコードに埋め込まれ，その中に分散されている）．もう１つの鍵は，宣言型の記述を読み込んでコンパイラの一部または全部を自動的に生成するジェネレータである．これらの機能は，他の種類のツール用にも，開発される必要がある．さらに，拡張性のもたらす利益を広い範囲で引き出すためには，プロセス中の個々の活動に対してではなく，ソフトウェア・プロセス自身に対する宣言的な記述を目指す必要がある．

3.4.2 活動項目

一般的な適応性，特にそのうちでも拡張性を強化するための活動項目を洗い出すにあたって，このグループでは，１９８５年の状況から１９９５年への発展は，重要な新機能の追加によるものではなく，現在利用できる機能の拡大によってもたらされると考えた．それには，ソフトウェア・プロセス・パラダイムに対する自動化支援の拡大，異機種分散型ホスト・システムのより日常的な活用，環境ユーザ間でさまざまなオブジェクト（コードやドキュメント）を共有する機能，個々の活動を支援するより汎用的なツール，などが含まれている．

したがって，グループ討論では，今後１０年間にわたって，環境の段階的な手直しを支援する機能に焦点を当て，特定の活動項目ではなく，いくつかの関心のレベルを明確にすることに力を注いだ．また，これらの各レベルにおける改善のための一般的な「ゲームプラン」を明確にした．

関心のレベルは，ユーザや環境構築者，ホスト・システムなどの領域でもたらされる変化の「範囲」と，密接な関連を持っている．こうした変化の範囲は，環境の基本構造（アーキテクチャ）をどの程度まで考えなおす必要があるかということと，大まかに対応している．関心のレベルが高いことは，環境の基本的な構造や設計が変更される可能性が大きいことを意味している．

最も高いレベルは，環境が支援すべきソフトウェア製品やソフトウェア・プロセスに対する視点に関連している．このレベルにおける変化は，ユーザ領域に由来するものであり，プロジェクト・チームの変更，組織上の変更，最終あるいは中間成果物の変更，対象応用分野の一般的なまたは特定の性質の変更，対象応用システムの「・・性」に対する要求の変更，などにもとづいて発生する．これらの変化は，環境機能の大幅な手直しにつながり，したがって，環境アーキテクチャの再構成をもたらす可能性が，非常に大きい．

次のレベルは，環境によって支援されるソフトウェア・プロセス・パラダイムに関連している．このレベルにおける関心は，何人かの人間のグループ（プロジェクト・チーム）が実行する活動や，それらの優先度や実行順序に置かれている．このレベルでの変化は，やはりユーザ領域に由来しており，環境全体に対する手直しに結びつきやすい．しかし，そうした変更は，１つあるいは少数のツールの集合に局所化することができ，環境アーキテクチャ全体に混乱をもたらすことは，まずありえない．

次のレベルにおける関心の焦点は，個々のプロジェクト・メンバが行なうべき業務を反映する個々の活動である．変化を吸収するために必要な環境の手直しは，ふつう，１つのツール（たかだか１組のツール集合）に局所化できる．

その次のレベルは，環境構築者の領域での技術の変化に関連している．その変化が，（エキスパート・システム技術の利用などのように）まったく新しい基本技術に関するものであるならば，それにともなう手直しは環境全体に及ぶものになり，新しい環境アーキテクチャが必要となる．しかし，一般的には，その種の変化が起こる可能性は少なく，特に今後１０年間のソフトウェア環境技術に関しては，そうした大きな衝撃的変化は起こりそうにない．またその変化が，環境の構造化や構築に関するまったく新しいアプローチをもたらすものであるならば，手直しの範囲も全体的なものになる．しかし，この種の変化も，一般的にも特定の場合でも，今後１０年の間には，あまり起こりそうもない．最も可能性の高い技術的変化は，ソフトウェア・プロセス内の特定の作業で使われる技法や，ツールの具体化技術に関するものであり，したがって，それにもとづく環境の手直しは，特定のツールに局所化することが容易である．

最後に，最下位のレベルは，ホスト・システムの領域における変化に由来するものである．そうした変化の例としては，環境の具体化言語の変更，ホスト・システムのＯＳの変更，ハードウェア・アーキテクテャの変更などがある．この種の変化に対応する環境の手直しは，変化を吸収するために，さまざまな既存の（または新）技術を利用しさえすれば，環境の基本的な構造に影響することはありえない．

以上のすべてのレベルにおいて，環境改善のためのいろいろな活動が必要である．あるレベルの変化は，それより下位のレベルに影響を及ぼす．たとえば，新しい応用分野への移行は，開発パラダイムに変更をもたらし，実行すべき作業を変化させる．したがって，活動項目は，いくつかのレベルにまたがるのがふつうである．

3.4.3 優先順位とタイミング

一般的には，最下位の２レベルの作業は優先順位が低いと考えられる．これらの２つのレベルにおける変化に適応するための技術は，すでにかなりの程度整備されており，今後起こりうるであろう避けがたい変化に対応するには，既存の技術を発展・拡張するだけで十分である．

しかし，最上位の３レベルにおける拡張性に関しては，基本的な問題点に対する調査研究と新しいアプローチや技術の開発が必要である．既存の技術の進化だけでは不十分であり，きわめて重要な意味をもつこれらのレベルでの拡張性の問題に対して，直接的かつ集中的に注意を向けて行く必要がある．

最上位の３レベルにおける拡張性は，今後５年間に，まず，小規模な短期的実験を行って，問題についての理解を深め，いくつかの解決アプローチに関する経験を積むことにより，かなり改善することができよう．実験対象として取り上げるべき一般的アプローチには，いくつかの拡張性支援メカニズムの設定，それらのメカニズムのために有用なツールの部分集合の識別（ＰＤＬツールなど），関連するいくつかのレベルでの変化に対応して既存のツールをどのように変更すればよいかの実験，拡張性支援メカニズムを組み込んだツールの構築，それらのメカニズムによる環境の機能拡張，などがある．

こうした短期間の実験に続いて，そこで得られた経験を整理し一般化するための活動を行うべきである．これらの中期的活動項目としては，いくつかの短期的実験結果の総合評価，拡張性支援メカニズムの一般化，５つの関心レベルにおける適応性・拡張性に対するニーズの完全な理解，複数レベルにまたがる変化に対応できる新しい汎用メカニズムの適用実験，などがある．これらの活動にあたっては，特定のレベルにおける変更ニーズを適切に処理するためには，複数のメカニズムを並列に用いる必要があるという事実を，十分に考慮しなければならない．

3.5 人工知能

ＡＩは，半自動的な問題解決のために有用な技術を，数多く生み出してきた．これらの技術は，たとえば，知識表現，推論，自然言語コミュニケーション，計画立案，設計，認知モデルなど，いろいろなことを支援してくれる．これらの技術によってもたらされる支援に特徴的な傾向は，ソフトウェア環境の内部に，プロセス特有の問題解決知識を内蔵すること，また，ユーザを細部の戦術的な仕事から解放して，より高度な戦略的問題解決に専念させること，などである．この分野での活動項目は，ＡＩ技術およびその特徴的傾向を，ソフトウェア環境構築に取り込み，その研究開発の最前線をより発展させることを目指すものである．

3.5.1 問題点

すでに述べたように，現状における最先端のソフトウェア環境技術は，直接的には，人工知能研究の成果をほとんどあるいはまったく反映していない．しかし，数多くの間接的な寄与は受けている．近い将来においても，引続き間接的な成果利用が主体になるであろう．いくらかの直接的な貢献が期待できないわけではないが，しかし，今後１０年間にＡＩ技術がどう発展し，その影響力がどの程度拡大するかを，正確に予測することはむずかしい．冒頭にあげた技術のうちで，今後，中期的にソフトウェア環境構築に対して，もっとも大きな影響を与える潜在的な可能性を持っているのは，知識表現および設計支援の２つである．

「知識」を，ソフトウェア・プロセスにおいて生成される成果物に関する情報と考えれば，知識表現の問題は比較的単純であり，本質的には，（伝統的な）データ構造化や体系化の問題に帰着させることができる．事

態が複雑になるのは，「自然な形」では提供されないような情報を獲得したいと考えた場合である．たとえば，要求分析時には重要ではないように見えて，設計段階になって初めてその重要さがわかるようなデータを獲得するにはどうすべきか，といった問題がある．

「知識」がソフトウェア・プロセス自身に関係する場合には，その情報を記号化し自動的に解釈するための問題がある．これまで，プロセスに関係する知識は，ソフトウェア・システムのコードの中に埋め込まれ，分散されていた．ここでのむずかしい問題は，この情報を手続き的にではなく宣言的に獲得し，ソフトウェア・システムの基本的なコードを変更することなしに，それらを変更し追加したいということである．

ソフトウェア環境に関連する知識表現において最も重要な問題点は，成果物についての情報をコンピュータの支援によって獲得する方法，および，ソフトウェア工学のいろいろな領域に対して，プロセスに関する情報をコンピュータが解釈できる形で記号化することである．ここでいう領域には，各種応用分野や，特定の問題，設計戦略，具体化技術などが含まれる．

設計支援とは，ＡＩ分野で開発された技術やアプローチを利用し，プログラミング言語による具体化に先立って，ソフトウェア・システムの大まかな論理および構造を開発することを意味する．ここでは，２つの大きな問題点がある．１つは，どうやってソフトウェア設計プロセスをモデル化するかということであり，もう１つは，どうやって設計履歴を格納し整理するかである．これらの項目は，ソフトウェア設計プロセスの支援にとっても，また，設計関連情報を以降の工程で役に立つ形で獲得するという意味でも，きわめて重要である．

3.5.2 活動項目

グループ討論では，知識表現および設計支援のためのＡＩ技術導入に関して，３つの活動系列が提案され，ソフトウェアの開発および進化を支援するためのＡＩ技術の効果的な活用法が論じられた．

(1) エキスパート・システムによるソフトウェア工学実践の支援

エキスパート・システム技術が成功しそうな分野としては，構成管理およびバージョン管理，プロジェクト計画立案，データ構造の選択，小規模なプログラム変換などがある．いずれにせよ，いくつかのプロジェクトで実際に適用して，どの点が不十分であるかを明確にし，どのような改善が必要かを指摘する必要がある．

(2) 設計モデル

ＡＩ技術を利用して，ソフトウェア設計モデルの研究開発を発展させて行くためには，次のような活動が必要である：

- **開発過程における設計上の意志決定をインタラクティブに記録し，その記録を保守支援に使用する．**
- **設計プロセスの形式化**：ここでの研究課題は，明らかにソフトウェア・プロセス・モデルの研究一般に影響を与え，また逆にそこから影響を受ける．
- **設計の方法論および戦略の分析と表現**：この課題は，明らかに前の２つと関連しているが，しかし，高度な能力を持つソフトウェア設計者の行動研究という形で，それらとは独立に進めることもできる．そ

うした研究では，設計指標を明示的に設定しておくことが重要である．

(3) 知識表現

ソフトウェア環境構築に利用できるように，最先端の知識表現技術を発展させるためには，以下の活動が必要である：

- **要求情報の獲得**：これは重要な分野であり，かなりの研究を必要とする．十分なケース・スタディを行わなければならない．また推論技術を開発して，要求に対する質問を可能にし，要求仕様作成や設計，コーディング中に答えが得られるようにする必要がある．最後に，解析技術（要求の一貫性検査など）も研究する必要がある．
- **形式的仕様言語**：この分野で最も重要なプロジェクトは，現実のシステムの形式仕様のケース・スタディである．
- **プログラミング知識の表現**：特定のプログラミング技法を表現する技術を開発すべきである．これらの表現に基づいて，小規模なプロジェクトを対象とした助言システムや自動合成システムが開発できる．

3.5.3 優先順位とタイミング

(1) 短期

以下に，今後５年間で有用な成果が得られそうな研究開発の活動項目を列挙する：

- 現在実用化されているエキスパート・システム技術を用いて，現実のソフトウェア工学実践活動を支援する．
- 開発過程における人間の設計意志決定をインタラクティブに記録し，それを保守支援に使用する．
- 形式的仕様言語．
- プログラミング知識の表現．

(2) 中期

設計方法論や設計戦略の分析と記述に関する研究は，５年から１０年後にかけて有益な成果をもたらすであろう．

(3) 長期

まだかなりの調査研究を必要とし，１９９５年までには，それほど衝撃的な成果が得られそうにないものとしては，以下のものがある：

- 設計プロセスの形式化
- 要求情報の獲得

4. 横断的ワーキング・グループでの討論

横断的ワーキング・グループでは，すべての技術項目にまたがる話題について，討論を行った．そのため，各グループは，全技術グループから代表者を集める形で構成された．いくつかのグループでは，問題点，活動項目，優先順位およびタイミングの明確化を行ったが，なかには，問題設定があいまいだったため，時間の大部分を問題自体の理解に消費してしまったところもあった．

4.1 統合化

統合化の本質は一貫性である．ソフトウェア環境に関していえば，統合化とは，環境の機能，ユーザの目から見える構成要素，内部構成要素，その環境を用いて作られた製品などの間での一貫性に関する問題として，とらえることができる．このグループでは，これらの問題の中から，次の３つに着目した．

まず第１は，**外部的な統合化**である．これは，ユーザの目から見て，環境が持つさまざまな機能の間に，外見上の高い均一性が保たれていることを意味する．外部的な統合化の単純な側面としては，ユーザ・インタフェイスを介して各種機能を使う場合の一貫性や，ユーザに情報を表示する場合の一貫性などがある．情報表示の一貫性には，表示に使用するメディア（グラフィックやテキストなど）が含まれる．また，単に同種の情報を同じメディアで表示する（たとえばツリー構造の情報はすべてグラフィックで表示する）だけでなく，同じ形態で表示する（たとえばツリーの表示ではルートを必ず画面の上端におく）必要がある．情報表示の一貫性は，また，情報交換に用いられる言語や開発対象のソフトウェア・システムを記述し理解するために用いられる抽象化のやり方にも関係しているたとえば，デバッグの結果は，プログラミング言語の文法に合わせた形でユーザに表示されるべきだし，設計やプログラミングにさいして用いられる抽象化は，逐次的なプロセス間の情報交換という意味で，ソフトウェアについての情報に関連している．

外部的な統合化は明らかに重要である．「驚き最小化の原理」を適用すれば，当面の開発（保守）作業で何が起こっているかについてのユーザの思考を中断させることなく，環境とユーザとの間の情報交換が円滑に進行する．

第２は，**周辺関係の統合化**である．これは，ソフトウェアの開発や進化を管轄する組織やマネジメントなど，より広い範囲の周辺組織と，開発環境との間の一貫性を意味する．これもまた重要である．この一貫性が存在しなければ，環境のユーザは，周辺の（人間）組織で用いられていることばで表現された要求や行動を，環境を用いて実行できる行動に翻訳するという作業を，つねに強いられることになる．

３番目は，環境の内部的な構成要素の一貫性を意味する**内部的な統合化**である．この型の統合化を正確に定義するためには，環境の内部構成要素の種類やその相互作用の形態を記述したモデルが必要になる．そうしたモデルを用いることによって，内部的な統合化を，さまざまな種類の構成要素が相互作用における一貫性として，定義することができる．また，内部的な統合化は，（標準的な構造を与えることを通じて）環境の設計を容易にし，（相互作用のモードを少数に限定することによって）具体化を容易にし，（環境の構成要素の交換を単純化にすることによって）拡張を容易にする．

4.1.1 問題点

統合化を行うためには，環境に対する認識を統一する必要がある．したがって，短期および中期にわたって第一義的に考えられるべきことは，高度の統合化を達成するための認識を開発することである．この一般的な目標から，いくつかの特殊な問題点が浮かび上がってくる．ただし，以下に上げる問題点は，主に外部的統合化と内部的統合化を目指したものである．それらは周辺関係の統合化にも関係しているが，直接にはそのことに触れていない．

(1) ソフトウェア・システムの抽象化をどのように行えば，環境が提供する機能を統一できるか

ソフトウェア・システムの抽象化は，環境ユーザが，自分の考えを表現し，ソフトウェア・システムの性質に対する疑問を定式化し，解析結果を解釈してそれらの疑問を解決するための手段である．プログラミング言語は，このような抽象化を実現したものであり，設計言語もまた同様である．問題は，ソフトウェア・プロセス全体をカバーできるような抽象化技法（群）を選択することである．単一の普遍的な抽象化が理想的であるが，まず不可能である．したがって，われわれとしては，一貫した特性を持つ抽象化技法の集合を見つけ出し，各技法がそれぞれソフトウェア・プロセスのある部分に対して適切であり，すべてを合わせれば，ソフトウェア・プロセス全体がカバーできるようにしなければならない．相異なるプロセス・パラダイムは，相異なる抽象化技法の集合を必要とするであろう．したがって，いろいろなパラダイムの相互関連を考慮して，抽象化技法の開発を進めなければならない．また，これら抽象化技法どうしの一貫性も確保しなければならない．ソフトウェア・プロセス中の異なる部分では抽象化のニーズが異なり，均質性を損ないやすい．構文上あるいは意味論上は無害なように見えるちがいが，全体の均質性を達成する上ではほとんど致命的な場合もある．

(2) 内部的な一貫性の達成を容易にする環境モデルとはどんなものか

一貫性を持った抽象化技法の適切な集合は，外部的な統合化を達成する助けになる．一方，内部的な統合化を達成するためには，適切な環境モデル（群）が開発されなければならない．一般に，そうしたモデルは，一連のオブジェクトのタイプと，それらのオブジェクト間の相互作用を定義する．各モデルにおけるオブジェクト集合は，環境内の操作型オブジェクト（ツール）と情報型オブジェクト（データ項目）の両方を含んでいなければならない．したがって，特定のオブジェクトやその相互作用に関して，あらかじめ予備的なモデルが定義されると，いくつかの副次的な問題が生じてくる．すなわち，操作型オブジェクト間の制御インタフェイスの定義や，操作型オブジェクトと情報型オブジェクト間のアクセス・インタフェイスの定義，一般的なオブジェクト間の構造的関係の定義，操作型オブジェクト間の情報交換を支援する一時的な情報型オブジェクトの扱い，あるオブジェクトに複数の外見を与えて（何らかの時間的制約のもとで）他のオブジェクトと簡単に相互作用できるようにすること，などの問題である．こうした環境モデルは，たとえば環境構築費用を予測可能にするなど，いくつかの理由から重要な意味を持っているが，また，なんらかの判定基準を持ったモデルの概念を関係者に植え付けるという観点からも，きわめて重要である．少なくとも，ちがった目的のために開発されたモデル間の相互関係を十分理解することが必要である．

(3) 高度の統合化と高度の適応性はどうすれば同時に達成できるか

統合化と適応性はお互い矛盾する関係にある．統合化は環境構成要素間に強い単一性を要求するし，適応性は構成要素間に分離性を要求する．しかし，両者は，同じ技術（環境モデル，一貫した抽象化，十分に定義さ

れたインタフェイス，汎用ツール，ツール部品など）を使って達成される．ここでの問題点は，一貫した抽象化技法（群）および環境モデル（群）の適切な集合を開発することによって，環境の統合化を達成すると同時に，一般的な適応性や個々の特殊ケースに対する拡張性を支援することが，はたして可能かどうかである．

(4) 統合化の尺度

統合化は，0か1かで測れる性質のものではない．少なくとも，相異なるアプローチを，その統合化の度合に関して比較できるような尺度が必要である．

一貫した抽象化技法の集合や環境モデルなどの考え方は，1つのオブジェクトに対して複数の視点を必要とする．したがって，統合化の程度を計量化するアプローチとしては，そうした異なる視点を提供する変換機能の個数を数えること，一連の変換機能間の相互関係の「整合性」を評価すること，ツールのちがいに対する変換機能の透明度を評価すること，などが考えられる．しかし，ふつうは，これらの単純な尺度では不十分である．たとえば，UNIX のように，雑多なツールを寄せ集めて作られた環境では，変換機能の個数は，統合化の度合とは逆比例の関係にある．しかし，InterLisp のように，きわめて均質的な単一言語環境では，変換機能の個数は，統合化の度合に比例する．さらに事態を複雑にしているのは，統合化の種類が異なると，その尺度も異なってくることである．たとえば，外部的な統合化を評価する尺度は，内部的な統合化を評価する尺度とは，まったく異なる．

高度の統合化はつねに望ましいが，必ずしもそれは，つねに必要だというわけではなく，また，費用対効果がよいわけでもない．したがって，どの程度の統合化が必要なのかを，定量的に評価できなければならない．そして，汎用の環境でも，特殊な環境でも，統合化を達成するさまざまなアプローチ（たとえば，既存のツール集合にある変換機能を付加すべきか，それとも，一貫した抽象化集合を与えるツールないしは何らかの環境モデルに適合したツールを構築するか）について，それらの費用対効果のトレードオフが計算できなければならない．

4.2 プロトタイピング

プロトタイピングとは，システムの未塾なバージョンを用意することである．ふつう，プロトタイプは，システムの機能の一部分を正確に表現し実現するが，性能や人間工学的側面など，システムの品質属性を正確に反映するものではない．プロトタイピングは，目的や解決法を明確にすること（調査型プロトタイピング），いくつかの解決アプローチを研究すること（実験型プロトタイピング），システムの開発や進化を繰り返し行うこと（進化型プロトタイピング），などを目的として使用される．

一般に，調査型および進化型のプロトタイピングは，ソフトウェア開発・進化の上流工程を支援するには，かなり有効である．ソフトウェア環境の場合，そうしたプロトタイピングは，環境が必要とする機能や，新しい環境の構築に既存のツールが活用できる度合い，環境の基本的なアーキテクチャ構造，などを決定するさいの助けになる．

現在ポピュラーなソフトウェア開発アプローチのうち，全体の「ゲームプラン」としてプロトタイピングを用いているものは，ほとんどない（しかし，このことは，現実にはしばしば行なわれている）．進化型プロトタイピングは，以下にあげる理由から，ソフトウェア環境開発にさいして特に重要である．

- 環境開発は，多くの場合，まだだれも試していない技術の利用を含んだ開拓型の作業である．
- 環境の設計や構築に対する新しいアプローチの可能性を実証するには，しばしば，「概念を証明する」ための仕掛けが必要である．
- 環境が実稼働される以前に，その具体化を完成させるのに十分な資源や時間が得られることは，現実には稀である．

これと全く同じ理由で，進化型プロトタイピングは，段階的なアプローチによってソフトウェア環境構築のための最先端技術を発展させる上でも，きわめて重要である．

4.2.1 問題点

プロトタイピングと環境とを考えるにあたっては，２つの観点がある．第１は，ソフトウェア環境の中に，どのようなプロトタイピング支援機能を盛り込む（べき）か，ということであり，そして第２は，ソフトウェア環境構築技術の研究開発を発展させるには，プロトタイピングをどのように使えば有効か，ということである．

このグループで洗い出した問題点や活動項目は第２の観点に焦点を当てているが，討論は主として第１の観点からなされた．プロトタイピングを通じて最先端技術を発展させるには，特定の環境を進化させるより以上の能力を必要とし，また，環境開発に対するプロトタイピング支援には，一般のソフトウェア開発の場合と同じ機能を必要とするというのが，グループ参加者の基本的な感覚であった．プロトタイピングによる最先端技術の高度化を特に重点的に考えれば，あるいは異なった問題点，異なった活動項目，異なったタイミングや優先順位が導き出されたかもしれない．しかし，ワーキング・グループには，そのための十分な時間がなかった．

つまり，このグループは，環境開発に主要な関心を抱きつつ，プロトタイピング技術一般について検討を行ったわけである．したがって，以下に列挙する問題点と活動項目は一般的な性格のものではあるが，これにもとづいて何か特定のプロジェクトを定義しようとする場合には，ここで想定されている成果物が環境そのものであるという点を考慮して，もう一度見直しをする必要があるだろう．

(1) ソフトウェアの開発や進化を支援するのに，どうすればプロトタイプを最も有効に使えるか

プロトタイプはシステムのモデルであり，その基本的な価値は，システムの性質に対する疑問に回答を与える手助けをするところにある．モデルに要求される正確性や，対象とするシステムの性質，使用される解析手法などのちがいに応じて，プロトタイピングには数多くの異なるアプローチが存在する．たとえば，特別に開発したモデルを用いたシミュレーションは１つのアプローチであり，手近な部品を再利用した「早くて雑な」具体化もまた１つのアプローチである．プロトタイピングにさいしては，まず最初に，明確にすべき問題点が何かをはっきりさせなければならない．その上で，各種の疑問に対する回答を得るのに最も適切なアプローチが定まるのである．

(2) 一連のプロトタイプで明確にすべき項目をどうやって識別するか

プロトタイピングを用いてシステムを開発して行く場合，中間的なプロトタイプ（群）を考えておく必要がある．それ（ら）は，最終的な成果物を生成するための踏み台でもあり，また，プロトタイプ（群）を用いて

解決すべき問題をどうすれば最も論理的に整理できるか，ということにも関係している．一方で，開発者の側には，要求の変化や新技術の登場に応じて最終目標を変更できるように，柔軟性を維持しておきたいという願望もある．プロトタイピングにもとづく段階的開発にさいしては，あらかじめ，何らかの方法によって，一連の意志決定事項を確認しておき，開発途中の任意の時点で，いま何が可能かの状勢判断にもとづいてそれらを整理しなおし，その結果としての意志決定の流れを支援する中間的プロトタイプの仕様を考えることができなければならない．また，最終製品に対する要求の変化や新技術の出現に対応して，意志決定の流れやプロトタイピングの順序を変更するための技術も必要である．

(3) 進化型のプロトタイピングに有効な方法論とは何か

上にあげた問題点は，実は，プロトタイピングにもとづくソフトウェア・プロセス・パラダイムにおける，技術上および管理上の総合的問題点の一部分である．今日まで，ソフトウェア・システムの構築にプロトタイピングを利用した経験や，開発や保守に関係する疑問に回答を見いだすためにプロトタイプを評価した経験は，あまり蓄積されていない（少なくとも記録されてはいない）．また，進化型プロトタイピングのマネジメントについて，たとえば全体的な管理や資源の割り当て，進捗管理，終了時期の判定などの経験も，ほとんど得られていない．そしてまた，プロトタイピングを他のソフトウェア・プロセス・パラダイムとどう組み合わせるかについても，それほど経験を積んではいない．たとえば，もし全面的にプロトタイピングを採用しようと思ったとしても，個々のプロトタイプは，旧来の「ウォーターフォール」アプローチで開発したりする場合がある．したがって，今後いっそう経験を積み，それにもとづいて，ソフトウェア・プロセスの指針となりうるような（技術上および管理上の）方法論を確立しなければならない．

(4) プロトタイピングの支援にはどんなツールが必要か

プロトタイピングの方法論がよく理解できて，初めて，その方法論を支援するツールを開発することが可能になる．しかし，現実のソフトウェア・プロジェクトで，プロトタイピングがさかんに（おそらくアドホック的に）行われるようになるまで待つ必要はない．実際問題として，プロトタイピング支援の出現を不必要に遅らさないためには，方法論の探求と支援ツールの開発を並行して行なうべきである．れには，まず，いままでに作られたプロトタイピング支援ツールを確認し，収集し，整理する必要がある．そうしたツールの数は少ないので，さらに追加のツールを開発し，それらの使用経験を蓄積し，３種類のプロトタイピングをすべて有効に支援するツール集合を作り上げて行くことが必要である．

(5) どんなソフトウェア・アーキテクチャがプロトタイピングを支援するか

ソフトウェア・システムの基本構造あるいはアーキテクチャは，システムを高度化する際の難易度に大きな影響を及ぼし，したがって，プロトタイピング・アプローチを実行するさいにも大きな影響を与える．ここでの問題点は，プロトタイピング支援のむずかしさがアーキテクチャによってどう変わるか，そして，どのアーキテクチャが最も支援性にすぐれているか，ということである．

(6) プロトタイプの評価にはどの技法が最も役立つか

すでに述べたように，プロトタイプは，開発途上のソフトウェア・システムに関する疑問に回答を与えるための土台を提供するものである．しかし，また，プロトタイピングの機能自体の進化を支援するためにも，プ

ロトタイプを評価しなければならない．その両方の理由から，プロトタイプを使って解明すべき項目の理解や，モデル化の支援，解析技法の開発などを進めなければならない．さらに，プロトタイプは，しばしば，技術者以外の人々によって，それぞれの立場からの疑問解明に利用されるので（たとえば要求確定時など），そうした人々によるプロトタイプ評価を（人間によって，あるいは半自動的に）支援するアプローチも開発しなければならない．

(7) 拡張性とプロトタイピングはどんな相互作用を持つか

この２つの話題は，多くの問題点を共有している．適応化技術のどの部分がプロトタイピング支援に役立つかを解明する必要がある．そして，適応性一般（その中でも特に拡張性）に関する最先端の技術開発に，どうすればプロトタイピングを効果的に利用できるかを解明しなければならない．特に重要な検討項目は，複数のソフトウェア・プロセス・パラダイムを交互に，または並列に支援するような環境を開発する場合，プロトタイピングと適応性とが，どのように相互作用するかということである．

(8) 人工知能技術を使用してプロトタイピングを支援するにはどうするか

エキスパート・システム技術を利用したシステムの半自動生成は，プロトタイピングの支援にはきわめて有効であるように見える．疑問点は，いつそしてどんな形で，この技術を使うかである．さらに，プロトタイピングは，今日の人工知能システムの開発にさいして，きわめて広範囲に利用されてきた．われわれは，この経験をふまえ，そこでこれまでに作られてきた各種の支援ツールを，ソフトウェア・システムの開発や進化に応用できるよう変更または拡張して行かなければならない．

4.2.2 活動項目

今後１０年間に開発される可能性のあるプロトタイピング支援機能についての討論が行われたが，十分な時間がなかったので，活動項目，タイミング，優先順位の決定には至らなかった．パーサ・ジェネレータ，属性文法，第４世代言語，オブジェクト指向環境など，現在よく利用されているプロトタイピング支援機能からさらに一歩進んで，宣言的記述をもっと広範囲に利用できるようにし，いくつかの重要な応用分野で現在活用されているような再利用支援機能を整備し，未完成バージョンのソフトウェア・システムを用いたいろいろな実験ができるようにしなければならないことが指摘された．

プロトタイピング・プロジェクトに関する指摘事項は，次の通りである： すなわち，調査型および実験型のプロトタイピングにさいしては，まず第１に，ソフトウェア・システムの正確なモデル化を心がけ，次に，そのモデルを使ってシステムの特性解析や設計上のトレードオフ分析を行なうようにしなければならない．一方，進化型プロトタイピングでは，まず，プロトタイピング・パラダイムの厳密な定義を明らかにし，次に，必要とされる機能を必要な時点で一貫性を保ちながら作りだして，開発費用を時間的に分散させるというこの新しいパラダイムの価値を明示しなければならない．

また，プロトタイピングの３種類の用途は，それぞれソフトウェア・ライフサイクル中の長い過程をカバーできるが，それらの間には，調査型プロトタイピングは実験型プロトタイピングの一部であり，実験型プロトタイピングは進化型プロトタイピングの一部であるという，自然な包含関係が存在することが指摘された．この包含関係は，実験型プロトタイピングは進化型プロトタイピングより単純であるという事実を反映したもの

であり，プロトタイピング機能を開発する順序と成果の達成が期待される程度を暗示している．

最後に，どんなアプローチが適切であるかは，対象ソフトウェア・システムの種類によって異なること，また，特定の狭い範囲で定義された応用分野のプロトタイピング支援機能を開発することは，一般的な問題を解決するより容易であることが指摘された．したがって，プロトタイピング機能の改善にさいしては，まず最初にいくつかの特定の分野を支援する機能を開発してみるのがよいと思われる．そこで得られた経験を収集・分析し，結果を整理して一般化することにより，一層の改善を図ることが可能になるのである．

4.3 計測

ソフトウェア環境における計測の問題点を考察するにあたっては，次の２つの点が重要である．まず第１は，ソフトウェア環境自体が持つ特性（たとえば統合化のレベル）を計測できることである．そして第２は，環境の特性がソフトウェア・プロセスや作成される成果物に与える効果（たとえば，統合化のレベルによって，あるプロセスを実行するのに要する時間がどの程度異なるか，ある程度の保守性を持つシステムを開発するには，ライフサイクル全体のどの部分をどの程度を統合化すればよいか，等など）を計測できることである．もちろん，この２番目の機能のためには，ソフトウェア成果物とソフトウェア・プロセスを計測することが必要であるが，その要求事項はここでは直接には議論されていない．

定量的計測は，いくつかの選択枝を比較して進化の方向をガイドする上で，明らかに重要である．しかし，いつも「数値」を引き出すのが可能だとは限らないし，また必要でもない．たとえどのような評価でも，それが単に直感的あるいは経験的なものであろうとも，発展の方向をガイドし制御するのに必要な判断を行う上で，価値を持つであろう．事実，無理に定量的な評価に固執することは，ときに（計測が簡単だからという理由で）誤った性質を重視したり，（定量的な結果を待つことによる）評価の不必要な遅れをもたらしたりすることもある．まったく新しいソフトウェア開発アプローチの採用など，革命的に新しいものへの移行の可能性を探るときには，特に，定性的評価や「勘」による評価に頼ることが重要になる．

4.3.1 問題点

ここでの関心は，進化型の段階的な改善によって環境構築技術の発展の方向をガイドするための計測であった．他の活動（たとえば新技術を広く普及するといったこと）を支援するための計測を考えた場合の問題点や活動項目は，もしかしたら，以下にあげるものと異なるかもしれない．ここにあげた問題点や活動項目は，さまざまな研究開発活動の支援に貢献しうるように感じられる．しかし，グループ討論では，その可能性については検討されなかった．

しかしながら，このグループは，計測の結果が技術進歩のガイダンスに用いられる場合には，費用対利益に対する期待は，通常とは異なるべきだということを指摘している．計測にかかる費用は，どの場合にも当然のことながら大きい．しかし，十分に開発された技術と比較して，開発途上にある技術の評価には，ずっと多くの費用を要する．計測によって得られる利益もまた，どんな場合にも大きいが，ただし，開発現場での狭い範囲の計測と比較すれば，リスクはずっと大きい．したがって，開発途上の技術を評価する場合には，期待される利益に対する費用の比率がかなり高くなることをあらかじめ覚悟しておかなければならない．

(1) 環境の属性，ツール，プロトタイプ（ツールまたは環境），環境全体などを評価する最善の方法とは何か

これは自明な問題点である．環境の全体あるいは一部を評価するよいアプローチとはどんなものだろうか．直接経験することは，１つのアプローチである（たとえば，だれかがあるツールまたは環境を使って，これまでより早くプログラムやシステムが作れることを実際に体験する）．十分に制御された状況で科学的な実験を行うのも，１つの方法である．しかし，この方法は，意味のあるシステムを対象にした場合には，途方もなく経費がかかり，制御が非常にむずかしく，また，規模の問題にきわめて大きく依存する（結果の規模もあるし，大規模システムの実験を取り扱う能力の規模もある）．３番目は，観察と推論にもとづくアプローチである．これは，何をどのように計測すべきかについての思考を鋭くするが，有用な成果を得るためには，かなり目的指向でなければならない．これらの相異なるアプローチを，いつどのような場合に使うか，また他のアプローチがないのか，を研究する必要がある．

(2) どうすれば撹乱を避けることができるか

計測によって対象を変えてしまうのは望ましくない．計測はエラーを隠してしまうこともあるし，対象が実際よりもよく見えるようにしてしまうこともある．計測用の命令を挿入することによって，まちがいなくシステムサイズは増加し，その意味では悪い結果をもたらす．また，計測対象の作業をゆがめたくはない．生産性の向上は，単にそれを計測するだけで達成される（ホーソン効果）というのは，悪名高い事実である．そして，システムのリアルタイム性を損なわずに，その動作を計測するのは非常にむずかしい．計測の影響を最少化し，結果を解釈するときに，その影響を分離する方法を研究する必要がある．

(3) どうすれば計測手段を目立たないようにできるか

上記の例は，計測手段そのものが往々にしてトラブルの原因であることを示している．しかし，計測手段の組み込みは，それがプロセスの進捗状況を追跡し，観測データや履歴レコードを収集し，システムの性能や人間的な要素の特性を調整する唯一の効果的な方法であり，たいていの評価アプローチにおいて不可欠な要素である．計測手段を組み込んだことによって，計測値が変動するような悪影響は，なるべく減少させなければならないし，ツールや環境のユーザに強い悪影響を与えないような計測方法が開発されなければならない．

(4) 実験を行うための研究室をどうやって確保するか

経験，実験あるいは観測と推論という３種類のアプローチのどれを用いる場合にも，評価や比較のための「研究室」を持つメリットは大きい．そうした研究室は，他の科学や工学の分野では，成果物の計測に用いるツールを整理し統合化するのに有効であることが分かっている．究室の物理的な属性（規模など）は重要ではない．また，ソフトウェア技術自体の進歩（コミュニケーション技術など）によって，分散型の機能が実現可能になったことから，従来の慣習（たとえば，テストを行う部屋，オシロスコープなど）にとらわれて考える必要はない．それよりも，最新の技術を活用した研究室的な機能の構築に重点をおいて考えることのほうが，重要なのである．このような機能を実現するには，以下にあげるきわめて困難な問題を解決する必要がある：

- **習熟曲線への対応**：十分な配慮と学習支援教材の開発なしには，習熟曲線はかなり急になり，実験結果を疑問なものにしてしまう．

- **技術とその使い方との相互作用の取扱い**：自動化支援のための新しい技術は，必ず作業の実行方法に影響を及ぼす．これらの相互作用は，十分に制御された実験を行うことを非常にむずかしくし，実験結果自身やその解釈，または結果の価値に重大な悪影響を及ぼす．
- **構成要素の組み合わせ方**：実験に使われるツールや調査対象は，多くの異なる源泉から集められており，それを互いに組み合わせる有効なアプローチやメカニズムが必要になる．一般に，これらのツールや対象を，それらが埋め込まれているシステム全体から分離することがむずかしいという点が，事態を悪化させている．

これらの問題はいずれも困難ではあるが，克服できないわけではない．既存の技術を利用すれば，以下の機能はとりあえず提供することができる：

- 環境の構成要素のうち，さしあたり実験に関係ないものを分離する．
- 習熟曲線を平らにする．
- 進化型プロトタイピング．
- 実験計画の作成．
- 各種の完成した環境あるいはプロトタイプ環境を実験に取り込む，あるいはアクセスする．
- ツール，ツール集合，環境全体，またはソフトウェア・パラダイムを評価するためのテストベッドを構築する．

以上の機能は明らかに原始的でなものではあるが，しかし，かなり有用である．それは，近い将来における進化型の改善のための適切な基盤となりうるものであり，とりあえず必要な機能，および，より進歩したバージョンを開発するための経験や知識を獲得する方法を，われわれに提供してくれる．（短期的ならびに長期的な観点において）研究室がどんな機能を持つべきかをはっきりさせる必要がある．また，どうやって基礎的な初期機能を確保し，それをどうやって時間とともに進化させてゆくかも，きちんと計画しておかなければならない．端的にいえば，研究所は実際には１つの特殊なソフトウェア環境であるから，この報告のあちこちで述べられている問題点について，すべて検討を行えばよい．とはいえ，研究室という特殊性のおかげで，検討の内容は，一般的な解答を求める場合と比べれば，かなり簡単であろう．

4.4 単一言語環境と多重言語環境

単一言語環境は，すべての作業を支援するただ１つの表記法を提供する．この種の環境が作られた動機は，構築の容易性および，複数の開発工程で使用する場合の一貫性である． InterLisp や Smalltalk は，単一言語環境の典型的な例である．

多重言語環境は，互換性のある一連の表記法を提供し，それぞれ軒法は，異なった作業集合を対象としている．これらの表記法は，いろいろな形の抽象化手段を提供し，それぞれ特定の作業向けにカスタム化されている． DCDS [DCDS84] は，多重言語環境の１つの例であり，そこではソフトウェア・ライフサイクルの各局面を，異なる記述法で支援している．Gandalf [Gand85] は，やはり，異なる表記法で異なる作業（プログラミング，構成管理，プロジェクト管理など）を支援している環境の例である．表記法が特定の作業向けに調整してあるので，ほとんどの場合，多重言語環境は単一言語環境よりもユーザの（時には環境の）の作業効率が高い．

現在の単一言語環境は，１つの表記法に焦点を絞ることによって，環境の持つ諸機能を強く統合化し，使いやすさの点ですぐれていることを実証している．しかし，このアプローチの欠点もまた，明らかになっている．この種の環境は，ソフトウェア・ライフサイクルの下流工程に対する支援に偏る傾向があり，ライフサイクル全体をうまく支援できない．そして，個人の作業だけを支援する傾向が強い．現在の多重言語環境は，これらの問題のいくつかを解決しているが，また別の問題を持っている．それは，複数の表記法の間で互換性をとるのが困難なことであり，また，ある表記法による記述から他の表記法への（半自動的な）変換を支援するのがむずかしいことである．その結果，弱く統合化されたいくつかの機能を提供するだけに終わってしまうきらいがある．

いずれの型の環境に対しても，改善活動項目を設定する必要がある．単一言語環境における主要な問題点は，広範囲のソフトウェア開発の作業をどうやって支援するかということである．したがって，現在の単一言語環境を拡張して，(1) 複数の異なる種類の人間を開発保守作業に取り込む，(2) ライフサイクルの上流工程を支援する，という２つのことをカバーする必要がある．

多重言語環境における主要な問題点は，環境の中に存在する各種の表記法や支援機能について，高度の統合化を達成することである．

4.5 ユーザ・インタフェイス

これまでの章で述べてきたことから，１９９０年代の環境はワークステーションを基本とするものになると想定してよい．ソフトウェア環境へのワークステーションの普及は，その上で各種の有用なツールが出現し，高解像度カラー・グラフィクス，音声入力（編集機能つき），広い作業領域を提供する大規模スクリーンなど，ハードウェアの機能が発展することによって，ますます加速されるだろう．これらの高性能ワークステーションの機能を効果的に活用するには，ユーザ・インタフェイスのメディアや，ソフトウェア開発と進化の過程における人間機械間の情報交換モードに対する理解を，十分に深める必要がある．

4.5.1 問題点

ユーザ・インタフェイスに関する問題点は，大きく３つに分けられる．

(1) システムのユーザ・モデル

ユーザと環境との間のやりとりに関して，適切なモデルを見つけ出す必要がある．モデル開発のアプローチには，認知科学的なモデル化の方法にもとづく「トップダウン型」と，人間的要素に関する実験にもとづく「ボトムアップ型」の２通りがある．

関連する問題点としては，インタフェイスをカスタム化すべきか，標準化すべきかということがある．ある面では，カスタム化は個人レベル・組織レベルのいずれか（またはその両方）で実施されるものであるから，これは管理上の問題だといえる．ある程度のカスタム化は望ましいとしても，２つの大きな疑問点が残る．それは，どうやってツールや環境向けにカスタム化可能なインタフェイスを構築するかということと，システムがユーザを「観察」しカスタム化に関して助言を行えるか（行うべきか）ということである．

(2) システムのプロセス・モデル

環境の内部にソフトウェア・プロセス・モデルを構築する主な動機は，システム自身がユーザの支援を行うことにある．システムは，特定の作業に対して（前後関係に依存した）チェック・リストをユーザに提供し，また実際に（ある方法論にもとづく）特定の作業をユーザに強制するなどの支援を行う．この種の支援は，ユーザ・インタフェイスにとって，明らかな意味を持っている．可視性や強制度の面で，どのような支援を行うべきかに関する考察が，特に重要である．

このようなモデルが実現可能かどうかについては，なお研究の余地がある．ソフトウェア・プロセス一般についてのモデルも，また，プロセス中の個々の工程に対する特定のモデルも研究されねばならない．さらに，特定のツールが起動された前後関係をシステムが「理解」できるためには，より「かしこい」，より統合化されたツール集合の開発が必要である．

(3) ユーザのプロダクト・モデル

ここでの主要な関心は，ソフトウェアを「可視化」するための支援を行うことにある．この点については，ソフトウェア環境は CAD/CAM 環境に比べてかなり遅れている．可能なアプローチとしては，システムの動作をアニメーション表示することや，システムを複数の特殊な視点から眺められるようにすること，などが考えられる．

4.6 漸進的改善と革新的改善

今後１０年間にわたって，ソフトウェア環境技術を改善して行くためには２つのアプローチがありうる．**革新的アプローチ**では，先進的な技術を利用して，強制的に新しい開発パラダイム（たとえば革命的に新しいソフトウェア・プロセス・モデル）に移行し，また高度な自動化支援（たとえば，高度に自動化されたプログラム変換）を行なう（[Balz83] 参照）．このアプローチの鍵となる特徴は，革命的に新しい技術を導入することによって，開発環境に不連続的な変化がもたらされることである．一方，**漸進的アプローチ**は，環境構築技術の着実な（かつ迅速な）成熟を基礎として，段階的な改善をもたらす新技術をたえず（頻繁に）取り入れ，それらを既存の環境に統合化することによって達成される．（[Boeh83]参照）．このアプローチの鍵となる特徴は，変化の単位が小さいことである．段階的な変化は危険も小さいが，しかしまた効果も少ない．

過去５年間におけるソフトウェア環境改善の多くの部分は，漸進的アプローチによってもたらされた．一般的にいえば，その理由は，危険度が小さく，ユーザの作業パターンの変更度が小さかったからである．このグループでは，１９９０年代に突入するにあたって，実践状況の改善にはやはり漸進的アプローチが適切だが，最先端技術の進歩に最も貢献するのは革新的アプローチだと感じている．

革新的アプローチの採用が何らかの理由（資金の不足，危険の大きさ）で不可能だとすると，効果的にこの２つのアプローチを統合する道筋が考えられる．この道筋は，次のごく単純な観察にもとづいている．それは，変化が革新的か漸進的かの評価は，評価の時点によるのではなく，変化が起こった時点によるということである．今日の状況からすれば革新的なアイデアでも，１９９０年の状況に投影すれば，単なる漸進的なアイデアにすぎないこともありうる．これは，たとえ漸進的アプローチをとっていても，革新的アイデアを避ける必要はない（避けてはならない！）ことを意味している．実際に，現時点ではまだ革命的すぎると思われる技術に

ついては，その技術自身やまわりの技術が成熟し，その導入が単に漸進的な１ステップになった時点で，速やかに導入できるよう計画しておく必要がある．

現在は革命的だと思われている技術も，漸進的アプローチによって環境構築技術を改善する上で，多くの役割を果たすことができる：

- 長期的に見て実現されるべき環境機能を定義すること．
- 十分に成熟しており，すぐに環境に統合化できるような先進技術に注目を促すこと．
- 中期および長期にわたって漸進的アプローチをガイドし，将来革命的な新技術を導入しやすくすること．

本質的には，革命的技術は，その時点における「技術的な水平線」を定義するものである．漸進的発展はその方向に向かって行われ，「水平線」に向かってしだいに速度をあげ，そこに到達したときにその技術を吸収できる素地を作って行く．

5. 全体セッションでの討論

オープニング・セッションでは，ワークショツプで取り上げるべき話題の決定と，１９９０年代半ばの技術状況の予測が行われた．その結論は，第１～２章で述べた通りである．最後の全体会議では，グループの関心は，(1) 提案された活動項目を成功裡に遂行できるような組織の特徴を洗い出すこと，そして (2) ５年前にカリフォルニア州ランチョ・サンタフェで開催された同趣旨のワークショツプの成果をあらためて見直すこと，の２点に絞られた．そこでの討論の概要を本章で報告する．

5.1 実行組織の特徴

さまざまな分野での活動項目の提言を作成している過程で，現在の組織構造は，必ずしもこれらの活動を実施する上で適切とはいえないということが指摘された．特に，ここで提案されたようなソフトウェア工学的な活動においては，従来からあった研究機関と開発組織との間の区別が，ほとんど不鮮明になってきている．概念の証明を行なうシステムを構築しなければならないことで，多くの研究プロジェクトは明らかに開発的な色彩を帯びてくるし，一方，各種のツールを実用的な環境へと統合して行くさいに発生する根本的な疑問を調査しているうちに，開発プロジェクトには自然に研究的要素が入り込んでくる．

最終全体討論の一部として，ここで提案された活動項目を実行する組織が持つべき特徴についての自由討論が行われた．討論の焦点は，環境開発プロジェクトの一般的構造，プロジェクト・チームの構成，プロジェクト・メンバの能力や経験などのことがらであった．その結果，このワークショツプで取り上げられた各分野でのプロジェクト提案の評価に使えるような，非技術的な判定基準のリスト（の一部分）ができあがった．

このワークショッップで洗い出された問題点を眺めて，すぐに感じられるのは，これらの課題に挑戦するには多くの資源が必要だということである．問題は，一見きわめて困難であり，それらを完全な形で解決するには，「お金のかかる」大規模プロジェクトをいくつも打ちださなければならない．また，プロジェクト，プロジェクト・チーム，およびプロジェクト・メンバに要求される特性（以下に述べる）を満足するには，やはり，非常に大きな財政的資源を必要とするように思われる．

そうしたプロジェクトは，単に成果物（ツール，環境の一部または全体）を開発するだけでなく，その利用も促進する必要がある．つまり，それらのプロジェクトは，ただアイデアを作るだけではすまないということである．アイデアは，ちゃんと使える形の成果物になっていなければならない（ただし，必ずしも製品としての完成度は要求されない）．概念の正しさを実証し，実現可能性やそれによってもたらされるメリットなどを例示するために，何らかの形あるものが必要になる．また，これらの成果物は，長期的な技術進歩の方向を明らかにするとともに，探求すべきいくつかの新しい可能性を示唆する．成果物の実際的活用をプロジェクトの不可欠な要素として取り込むことにより，いくつかのよい結果が得られる．そうしたデモンストレーションにもとづく評価は，その成果物の価値を最もよく認識できるような人間が評価作業に積極的に参加した場合にのみ，初めて最大の効果を上げるものからである．また，それは，プロジェクトの成果物が，頑丈さや性能などいくつかの面で高品質になるという，波及効果をもたらす．

プロジェクト・チームは，研究者，現場の実践技術者，ツール開発者の３者から成る混成チームであることが望ましい．研究者は，必然的に発生する基本的な問題点について，高度で知的な検討を保証するために必要である．ツール開発の専門家は，成果物が工学的見地から見て十分な品質をそなえていることを保証するため

に必要である．また，現場技術者は，成果物が十分実用的であり，評価のためのデモンストレーションが，実世界の要求や関心を反映していることを保証するために必要である．

混成チームがもたらすメリットは数多いが，そのうちのいくつかを以下に示す：

- 現実の典型的なソフトウェア開発・進化の状況を反映した形で成果の開発が行われる可能性が高まる．
- 開発から普及までの時間差を短縮する．
- 成果物が現実世界の関心（会社の方針，組織構造，組織の変更など）を反映する可能性が高まる．
- 現場で実際に用いられているソフトウェア・プロセスの影響下で開発を進める可能性が高まる．
- 開発の成果を大規模システムで利用するさいの規模の問題について，正しい認識や理解が深まる．
- 成果物を実際にする人々に高い信頼感を与える．
- 成果物が本当の要求に合致する可能性が高まる．

現状の組織構造では，このようなプロジェクト・チームの編成はなかなかむずかしいことが，多くの参加者から指摘された．研究者，実践技術者，ツール・スペシャリストが同一組織内に存在することは滅多にない．もし，いたとしても，組織内部の壁によって，効果的な協力関係は阻害されている．少なくとも近い将来においては，ここで提案された問題点や活動項目を消化して行くためのプロジェクトは，学界・産業界・政府機関の共同体によって実行されるしかないであろう．

プロジェクトメンバは，個人的にもまた全体としても，重要であるにもかかわらずしばしば忘れられているソフトウェア・プロセスの２つの特徴を，正しく認識していなければならない．まず第１に，ソフトウェアの開発や進化は，ソフトウェア・ハードウェア・人間という３つの要素から成るシステム全体を開発・進化させるもっと大きなプロセスの一部だということである．システム全体に対する直接の関心をプロジェクトの一部として取り込む必要は必ずしもないが，ソフトウェア・プロセスの外側にあってそれを包含するシステムの概念は，つねに心にとめて置かなければならない．第２に，プロジェクト・メンバは，ハードウェア・ソフトウェア・人間の各要素間の相互作用やトレードオフ関係を，正しく認識していなければならない．再び，すべてのプロジェクトがこの問題（たとえば，システムを実現するためのいろいろな代替案の選択を支援するプロダクトを開発するといったこと）に直接取り組む必要はないが，つねにより広範囲な問題を念頭にとどめておくことによって，開発の成果がより広い応用範囲をもつという（望ましい）波及効果が得られるだろう．

最後に，プロジェクト・メンバは，個人的にもまた全体としても，最先端のソフトウェア技術についての正しい認識を持っていなければならない．実際に，最先端技術の研究開発に携わった経験や知識を持っているかどうかは，あまり重要なではない．重要なことは，「モダン」な進取の精神を持ち，技術の水平線がどこに位置しているかを理解し，その価値や限界，現実との隔たりをわかっていることである．このワークショップで提示された活動項目は，それぞれ独自の有効性を持つ中間的成果物を作り出すことを意図してはいるが，本質的には，より長期的に見て，技術の進歩に大きく貢献することを目的としたものである．したがって，それぞれのプロジェクトは，この目標に沿った成果物をもたらさなければならない．特に，最終成果物が，プロジェクト開始時の最先端技術（最悪の場合はその当時の実用技術）をなんら改善することなく実現したようなものであってはならない．

5.2 ランチョ・サンタフェ再訪

今回とほぼ同じ目的で，５年前にランチョ・サンタフェで行われたワークショップの１つの成果として，ソフトウェア環境が持つ次の５つの重要な特性を指摘した論文が発表されている [Oste81].

- **支援範囲**：ソフトウェア要員あるいはチームに対して，ソフトウェア・プロセス全体にわたる強力な支援を提供する．
- **ユーザ親和性**：ユーザ側の作業手続きやスタイルが変わっても有用さを保てるような適応性を持つとともに，強力な，しかし強制的はでない快適な支援を提供する．
- **柔軟性**：（モノリシックな）ツールではなくツール・フラグメントを利用して，柔軟性や再利用性を高める．
- **統合化**：ツール間の相互作用や協力関係をより密接にし，スムース，かつ連続的，かつ非強制的な支援を提供する．
- **情報の集中化**：中央の情報貯蔵庫へのアクセスによって，ツールやプロジェクト活動の相互調整と集中管理を図る．

また，このワークショップでは，次に述べるような２段階のアプローチが提案された．第１段階は，既存システムの調査研究とあわせて，ツール・フラグメントと中央データベース支援を実験的に構築する．そして第２段階は，第１段階で開発された技術を利用して，与えられた仕様に対する汎用的な環境を開発することである．

このアプローチは，TOOLPACK [Oste83] や Software Productivity System [Boeh84] などの例に見られるように，かなりの成果をおさめた．これらの環境が持つ諸特性やその意義について，われわれの考え方がより洗練されてきたことも，その成果の一部である．最終全体討論では，５年前になされた技術評価の内容が再検討され，新しい考察がいくつか追加された．これらの議論の要約を以下で報告する．

5.2.1 支援範囲

ソフトウェア環境が広範囲のソフトウェア開発活動を支援しなければならないことについては，依然として疑問の余地はない．しかしながら，これまでの５年間に，要求される「範囲」に関するわれわれの考え方はかなり進化し，「ソフトウェア開発要員」に対する考え方も大きく変わってきた．ソフトウェア開発要員や開発チームは，単にソフトウェアだけに関心を持つだけでなく，ハードウェア・ソフトウェア間のトレードオフなど，システム全体の問題に取り組んで行く必要がある．

5.2.2 ユーザ親和性

５年前には，環境は強制的なものであってはならず，また，ユーザ・インタフェイスは快適なものでなければならないということが，「ユーザ親和性」に関する２つの重要な必要条件だと考えられていた．これらの要求条件は依然として重要であるが，他の目標に比較して，こうした意味での「ユーザ親和性」が２次的なものでしかという状況が存在することも，確認されている．ときには，ユーザに環境の持つ機能をはっきり意識させるようにした方がよい場合もある．たとえば，環境にある種の強制力を持たせた典型的な例として，Interlisp の DWIM や構文指向エディタにおけるアドバイスの提供と方法論強制の２つがある．

同様に，快適さの概念についても，環境やその構成要素の利用に関して，特に学習段階においては，利益対苦痛のトレードオフ関係が存在することがわかってきた．環境は，これまでより広い範囲の支援機能を提供しなければならず，そうなれば，統合的な親しみやすいインタフェイスを提供する上で，新しい問題が発生する．有用な付加価値が得られるならば，ユーザは多少の不便は喜んで我慢するに違いない．

5.2.3 柔軟性

環境の柔軟性に関する問題点は，もはや，単に新しいツールを取り込む能力だけに限らない．同様に，新しいパラダイムも取り込めなくてはならない．さらに，柔軟性の範囲は個々のユーザの要求を越えて拡張されなければならない．チームの規模や，組織の方針，組織構造，その他多くの「外的な」要因の変化についても，支援しなければならない．

5.2.4 統合化

統合化の問題は，環境の持つ機能をお互いどのように適合させるかという意味では，ユーザ親和性と非常に強く関係している．統合化が，０か１かで判定できる性質でないことは明かである．実際に，異なるユーザに対しては，異なるレベルの統合化が必要である．また，一枚岩の環境から，部分環境の集合であるような環境への移行も行われている．この移行によって，4.1 節で議論した３種類の統合化（外部的統合化，周辺関係の統合化，内部的統合化）を踏まえた統合化戦略が必要になり，また，方法論的な統合化（たとえば，環境が支援する機能と，特定のソフトウェア・プロセス・パラダイムに関連する自動化されていないその他の機能との一貫性）も重要になってくる．

5.2.5 情報の集中化

すべてのデータを格納する中央データベースの概念（論理的なものであり，物理的である必要はない）は，長い間ソフトウェア環境の設計において基本的なことだと考えられてきた．しかし，この概念がもはや環境を設計する上で真の「駆動力」にならないことは明らかである．本当に検討すべき問題点は，ソフトウェア・プロセスおよびそのプロダクト集合である．この観点に立てば，次にあげる各次元に沿って，論理的に異なる何種類かの情報格納庫を持つのがよいと思われる：

- 範囲
  - 情報の利用者
  - プロセスの各段階からの情報
  - プロジェクト情報
- 時間に依存する属性（たとえばシステム・バージョン）

6. 全体的な関心事項

ワークショップでは，数多くの「全体的な関心事項」が現れた．これらは，ほとんどの技術グループや共通技術グループにわたっているという意味で全体的である．これらは，ワークショップで考察されたすべての分野での進歩を達成する上で重要であり，したがって，直接・迅速・集中的な注目に値することがらである．

(1) 利益対苦痛

ソフトウェア・ツールや環境を利用する上での主な障害は，それを学習し活用するための労力の大きさである．この労力は，ときに，ツールの利用によって得られる利益をはるかに越えてしまう．実験または開発期間中なら，ユーザは，高い苦痛の割に得られる利益が小さくても耐えられるだろう．しかし，実際の使用段階に入った後は，環境の構成要素がユーザに受け入れられるためには，なるべく少ない労力で高い機能を提供しなければならない．ツールおよび環境の開発や利用に関する意志決定を適切にガイドするためには，利益対苦痛の比率を測定し最適化する方法を開発する必要がある．

(2) 抽象化

ソフトウェア・プロセスの各工程を支援するための一貫した適切な抽象化技法の集合を見つけ出す必要がある．こうした抽象化技法群の必要性は，統合化グループによって指摘されたが，それらは，他のグループが提案した活動を実施するさいにもきわめて重要である（たとえば，拡張性およびユーザ・インタフェイス・グループが提案した活動項目）．

(3) 視点

システムに対するさまざまな視点は，ユーザが果たす一連の役割の抽象化に関係しており，したがって，ソフトウェア環境に関する幅広い活動項目の全体にまたがっている．それぞれの活動分野において，適切なユーザ視点を洗い出す必要があり，またそれらの視点を提供する支援機能を開発する必要がある．

(4) モデル

ソフトウェア環境の世界で重要な一般的改善が達成できるかどうかは，ソフトウェア・プロセス，環境のユーザ，プロセスとユーザの関係などを反映した，有用なモデルを作れるかどうかにかかっている．

(5) 柔軟性／拡張性／カスタム化容易性

これは，基本的にはあるグループが指摘した関心事項であるが，他のグループが検討した分野での改善を成功させる上でも，きわめて重要である．

(6) 統合化のレベルとスタイル

統合化はモノリシックに考えることはできない．それは，統合化グループが明らかにしたように，いくつかのレベルやスタイルに分解されるべきものである．この分解によって，環境の高度な統合化が達成できるよう

になる．また，広い意味での統合化の達成と矛盾する可能性のある他の目標を，同時に達成するための枠組みが用意される．

(7) コミュニケーションの質と量

将来の技術では，ユーザ間のコミュニケーションや，環境の構成要素間のコミュニケーション，またユーザと環境要素間のコミュニケーションの幅を広げることが可能となる．しかし，この幅広いコミュニケーション手段を利用して，ユーザ・インタフェイスの改善や，分散機能の利用促進，拡張性の支援などの波及効果を得るまでには，かなりの調査研究が必要である．

(8) 変化の大きさ

１９９０年代に向かって進んで行くさいの個々の変化の大きさは，ソフトウェア環境改善のすべての面において，重要な関心事である．すでに指摘したように，小さな変化量は漸進的改善を意味し，大きな変化は革新的改善を意味している．どのような大きさの変化が，段階的な改善にとって理想的であり，また実行可能であるかを見分ける感覚を養う必要がある．

7. まとめ

今回のソフトウェア環境ワークショップで，各技術ワーキング・グループおよび横断的ワーキング・グループは，今後１０年間にわたってソフトウェア環境構築技術を改善するための基本的な問題点を，広い範囲にわたって明らかにした．また，とりあえず今後の５年間に（場合によっては，その次の５年間も含めた中期にわたって），これらの問題点を解決するためのさまざまな活動項目を洗い出し，それらの活動を実施すべきタイミングと優先順位を検討した．１９９０年代半ばまでに何が達成できるかについての全員の評価を総合すると，環境の機能は現在に比べて大きく改善されていることが予想される．また，ここで合意された提案は，短期的および中期的の改善は，主として，漸進的かつ段階的に行われるべきであり，それにより妥当なコストでかなりの成功が得られる可能性が高い．

ここで作成された問題点および活動項目のリストは，完全なものではない．いわば，それは，相異なる知識を持つ研究者および実践技術者のグループが，２日半にわたる集中討議で作成した可能性のスナップショットである．それは，環境構築技術の発展には何が必要かということについての，ソフトウェア・コミュニティ全体の考え方の種子となるものであり，今後いろいろな開発プロジェクトを計画し，そうしたプロジェクト間の調整を行い，長期目標を設定するさいに，大いに役立つであろう．

ワークショップの全体討論は，ここで確認された問題点の解決を目指して提案された諸活動を実施する組織，プロジェクト，およびプロジェクト・チーム・メンバがそなえているべき，いくつかの重要な特性を明らかにした．成功するために最も重要な要素は，プロジェクト・チームのメンバやその背後にある組織が，たとえ能力的には無理であるとしても，考え方の上では技術の水平線にできるだけ接近することである．

５年前の類似のワークショップでは，単に環境の技術的側面だけが論じられたが，今回の全体討論では，状況が明らかに変わったことが指摘された．すなわち，ソフトウェア環境によって支援されるプロセスにもっと力点が置かれるべきことが提案された．この点に関していえば，現実のソフトウェア・プロセスが実際にどうなっているか，高い生産性や品質をもたらす管理しやすいソフトウェア・プロセスのモデルとはどんなものか，ソフトウェア・プロセスを有効に支援するためには技術をどう統合化すればよいか，などのことがらを理解することが重要である．

このワークショップの最も重要な結論は，ソフトウェア環境構築技術を大きく発展させるためには，基本的に関心の焦点を移動することが必要だという点にある．過去５年間に試みられたいくつかのプロジェクトは，一定の範囲や機能を意図した環境を作り上げるために，基本的な問題点を理解し，新しい技術を開発し，既存の技術をまとめあげることに力を注いできた（これは，たしかに正しかった）．しかし，現在われわれが直面している問題は，環境が満足すべき目的やニーズに焦点を当てることを必要としている．したがって，今後５年間に実施されるプロジェクトは，既存の環境が持つ機能を段階的に拡張し，対費用効果を実例で証明し，われわれの経験や既存技術の拡張に役立つ実験を行なう，といった形のものでなければならない．こうした努力の結果，１９９０年代の初頭には，ふたたび新たに基本的問題（統合化，拡張性，プロトタイピングなど）を検討し，正しい先見的知識をもって新技術を開発する次のサイクルに入るための，基礎が確立できるだろう．

このワークショップは，過去５年間における大幅な進歩にもかかわらず，まだなすべきことがたくさん残っていることを明らかにした．たとえば，ここでは取り上げられなかった重要項目のリスト（図２）を見ても，なすべきことの規模は想定できるだろう．このレポートは，来る５年間に何をなすべきかについて考えるさい

に，1つのガイドラインとして役立つであろう．

8．謝辞

このワークショップに参加したのは，Rick Adrion，Mark Ardis，Besty Bailey，Dave Barstow，Frank Belz，Geoff Clemm，Ed Comer，Ward Cunningham，Dennis Heimbigner，Jack Kramer，Roger Miller，Gil Myers，Lolo Penedo，Charlie Richter，Dave While，Jack Wileden，Jim Winchester の各氏であった．

編者２人は，上記の参加者全員の貢献に対して，深い敬意と感謝を捧げるものである．討論の過程で提示された数多くの先進的なアイデアや材料は，このレポートの作成をきわめてやりがいのある楽しい作業にしてくれた．また，レポートの原稿を査読し，有益な意見や助言を与えてくれた Bob Ellison，Peter Henderson，Dick Tayler，Michael Young の各氏にも，ここで感謝の意を表する．

9．参考文献

[Balz83] R.Balzer, T.E.Cheatham, and C.Green, "Software Technology in the 1990's: Using a New Paradigm," Computer, vol.16, no.11, pp.39-45, 1983.

[Boeh83] B.W.Boehm and T.A.Standish, "Software Technology in the 1990's: Using an Evolutionary Paradigm," Computer, vol.16, no.11, pp30-38, 1983.

[Boeh84] B.W.Boehm, M.H.Penedo, E.D.Stuckle, R.D. Williams, and A.B. Pyster, "A Software Development Environment for Improving Productivity," Computer, vol.17, no.6, pp.30-44, 1984.

[DCDS84] "DCDS Final Report," TRW Report 38392-G950-046, TRW, Huntsville, Alabama, July 1984.

[Gand85] Special Issue on Gandslf, The Journal of Systems and Software, vol.5, no.2, May 1985.

[Keen80] P.G.W.Keen, "Adaptive Design for Decision Support System," SIGOA Newsletter. vol.1, nos.4/5, September/November 1980.

[Oste81] L.J.Osterweil, "Software Environment Research Direction for the Next Five Years," Computer, vol.14, no.4, pp.35-43, 1981.

[Oste83] L.J.Osterweil, "TOOlPACK - An Experimental Software Development Environment," IEEE Transactions on Software Engineering, Vol.SE-9, no.6, pp.673-685, 1983.

ＳＥＡ会員状況（昭和 63 年 1 月 20 日現在）

正会員	1213名（11月17日から15名増）
賛助会員	35社（11月17日から2社増）

＜正会員の勤務地および居住地域分布＞

	勤務地域	居住地域
北海道＝	9	9
山形　＝	1	1
宮城　＝	3	2
岩手　＝	6	6
福島　＝	1	1
秋田　＝	1	1
新潟　＝	6	6
栃木　＝	6	5
群馬　＝	1	1
茨城　＝	7	9
埼玉　＝	17	77
千葉　＝	15	79
東京　＝	700	430
神奈川＝	91	223
長野　＝	32	32
富山　＝	3	3
福井　＝	1	1
石川　＝	2	2
静岡　＝	17	14
岐阜　＝	2	6
愛知　＝	39	33
和歌山＝	2	1
三重　＝	1	2
滋賀　＝	3	4
京都　＝	14	21
大阪　＝	137	108
奈良　＝	3	8
兵庫　＝	27	43
愛媛　＝	2	2
徳島　＝	3	3
岡山　＝	1	1
広島　＝	4	4
鳥取　＝	0	1
福岡　＝	12	12
熊本　＝	12	13
大分　＝	4	4
佐賀　＝	1	1
宮崎　＝	1	1
鹿児島＝	3	3
沖縄　＝	1	1

＜男女分布＞

男　＝	1134
女　＝	64

＜年齢分布＞

20以下　＝	0
20_24　＝	60
25_29　＝	285
30_34　＝	328
35_39　＝	295
40_44　＝	142
45_49　＝	57
50_54　＝	21
55_59　＝	10
60以上　＝	7

＜血液型分布＞

A型　＝	483
O型　＝	349
B型　＝	261
AB型＝	121

＜賛助会員会社名＞

ＩＮ情報センター，旭化成工業，インターナショナル・データ・リサーチ，伊藤忠エレクトロニクス，ＳＢＣソフトウェア，エムテイシー，サン・ビルト印刷，情報システムサービス，ジェーエムエーシステムズ，セントラル・コンピュータ・サービス，ソフトウェア・リサーチ・アソシエイツ，ニッポンダイナミックシステムズ，マイクロキャビン，三菱電機セミコンダクトソフトウェア，ＰＦＵ，リクルート，リコーシステム開発，近畿日本ツーリスト，構造計画研究所，神戸コンピューターサービス，経調，辻システム計画事務所，東海クリエイト，東電ソフトウェア，日進ソフトウェア，日本システム，日本システムサイエンス，日本能率コンサルタント，日立製作所，日立ビジネス機器，富士ゼロックス情報システム，富士通，富士通ビジネスシステム

（アイウエオ順）

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