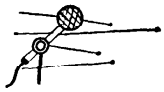
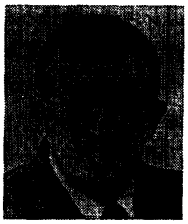
講　演



ソフトウェア工学の展望†



D. Bjorner††　澤田正方†††(翻訳編集)

0. はじめに

ソフトウェア工学並びにプログラミングについて，現在どのようなことが考えられているかについて，皆さまにお話しする機会をいただいたことを大変感謝しております．今日私がお話しいたしますのは，私が永年にわたって仕事をし，考え，人と討論する中から生み出された成果であります．私のアイディアを形成するには多くの人々の協力を得ました．IFIP の WG 2.2 及び WG 2.3 の仲間には大変感謝しています．

私はソフトウェアの形式仕様の重要性を信じておりますので，今日の話の仕様をまず申しあげておきます．

全体は5部にわかれております．第1部は序論というか，概要の部分で，ここで私は情報技術と，より広い概念としてのインフォマティックスとの違いについて述べます．つぎに第2部とし話題をしぼってソフトウェア工学とプログラミングの違いを述べ，第3部としてその違いについて，少し技術的な議論をしたいと考えております．第4部としてこれら三つの基礎の上に，情報技術とインフォマティックスの違い，ソフトウェア工学とプログラミングの違い，及び技術的な例を示し，最後にいくつかの結論を導き出したいと思います．

1. 定義について

我々が携わっている若い，たかだか 40 年の歴史のこの分野の問題の一つは，確立した術語が足りないと言うことです．

そこでいくつかの定義を行いたいと思います．最初に計算機科学と計算科学とソフトウェア工学の区別を行いたいと思います．私にとって，計算機科学とはプログラムの研究であり，計算科学はプログラミングの研究です．もちろん，同一人が両方の才能を持って活躍されることも多いと思いますが，そこにははっきりと境界線があると考えています．プログラムの研究では，我々はなにを計算できるか研究します．あることを計算することがいかに困難か研究します．そこで我々はプログラムの特性を研究します．それに対してプログラミングの研究ではどのようにプログラムを作るかを研究します．

ソフトウェア工学はプログラムとプログラミングの実践です．

それから私は同様にプログラマとソフトウェア技術者の違いを明確にしたいと思います．

プログラマは計算科学の規則に従って仕事をする人であり，ソフトウェア技術者はプログラミングとシステムの間には技術とインタフェースが存在することを認識している人です．この場合も，しばしば同一人があるときはプログラマで，あるときはソフトウェア技術者であることがあります．

ここで過去と現在と未来との間の違いを整理してみたいと思います．

過去では，計算機と計算科学，プログラム科学とプログラミング方法論の間にはなんの区別もされていなかったと思います．また，プログラマとソフトウェア技術者の区別にもしばしば混乱を生じました．今日，計算科学者の中にはソフトウェア工学なるものは存在しないと考えている人もいます．

私に言わせれば，プログラマは理論と実践の架け橋であり，ソフトウェア技術者はプログラミングと技術の架け橋となっています．理論の成長は大変ゆっくりしており，技術の発展は非常に急速です．変化しつつある技術に，同じプログラミング上の原則の多くがあてはまります．

2. インフォマティックス

過去現在未来の違いの話を続ける前に，インフォマティックスがなにを意味するか明確にしておきたいと思います．私は，今日のシステムの最善のものと明日

† 創立25周年記念特別講演（昭和60年9月9日学習院記念会館）
†† デンマーク工科大学
††† 永楽電気(株)

のシステムのほとんどは，いくつかの原則に基づいて作られると信じています．ここにその原則のうちの4つをとり上げました．もちろんこれ以外の原則もあります．

我々のシステムは，第1に計算機と計算科学のルールに従って作られています．第2にいわゆる認識的特徴を表しています．第3にそれらは機械をとおして人間との間の通信と対話を可能とします．第4に，それらはダイナミック，かつ変貌する組織のために，そしてその組織により作られます．したがって我々がこれらの分野を理解し，それらがいかに関係しているかを知ることは非常に重要であります．計算科学及び工学ではそれぞれの機械にどのような対象が存在するかに関心があります．認識においては，我々は情報であるなにかをどうやって発見し，このような情報，知識，方法をどのように表現し取り扱うかに関心があります．コミュニケーションといっても私はデータ通信を言っているのではありません．私が言うのは機械を介しての人と人のコミュニケーションです．データ通信は計算，認識，通信及び組織の諸問題を解決する一つの方法かもしれません．ある意味ではこれを「言語学」と呼ぶべきかもしれません．ここで我々は言葉のパターンと対話のルールを考えます．人と人との間で対話をするためのルール，伝えたものが正しく理解されるようにするためのルールがそれです．

我々が組織を扱うとき，組織の構造がどのように作られ，構成要素はなにで，これらをどのように統合し，時間と共に発展させるかに関心があります．このように，将来，計算においても認識的場面のますます多くの集積をみることになるでしょう．ある意味では，計算と認識を一緒にすれば人工知能になります．いま，東京にいるのですから，それについてこれ以上言う必要はないでしょう．私が知っているよりも，もっと皆さんは知識をお持ちであると思います．私は第5世代コンピュータの計画の成否は，計算，認識，通信の3分野の適切な組立にあると信じています．

しかし，おそらく，そのシステムを使う組織の変化に自動的に適応するシステムを設計する必要があります．私はそれが実現できるかどうかわかりませんが，しかし少なくとも努力してみるべきだと思います．

3. 情報工学とインフォマティックス

情報工学とインフォマティックスの違いを整理したいと思います．情報工学はインフォマティックスの一部，つまりインフォマティックスのサブセットです．私は過去のシステムは情報技術を代表すると考えています．そして私はこのようなシステムをつぎのように特徴付けたいと思います．それらは技術に縛られています．我々がそのようなシステムを語るとき，我々はVLSI，バブルメモリ，プラズマスクリーン，光ファイバ等といった次元で話をします．我々がこのようなシステムを構築しようとするのは，我々がプロセスを自動化したいからです．つまらないプロセスを自動化する，人間にとってあまり面白くないプロセス，重工業にあって人間が病気になったり気分が悪くなるようなプロセス，人間が手作業でやればなん千人の人が必要なプロセスを自動化したいからです．

しかし，このようなシステムは使うには非常に厳しい訓練を必要とするということと，これらのシステムによって個人のプライバシ問題が脅かされるようなことがあるということが，少なくとも私の国ではわかってきました．そこで私達は未来に希望をつなぎます．私達はもっとインフォマティックスの概念を統合したいと考えています．

情報工学によるシステムはテクノロジ指向型であったのに対して，インフォマティックスによるシステムはアイディア指向型であってほしいと思います．情報工学のシステムがプロセスを自動化したのに対して，インフォマティックスのシステムは人間の知性を補強するものであって欲しいと思います．情報工学のシステムが大変綿密なセキュリティシステムを必要としたのに対して，インフォマティックスのシステムは，あまりに官僚主義に陥ることがないように我々を助けるものであって欲しいと思います．

このように過去の情報工学はテクノロジにしばられたものでありましたが，将来のインフォマティックスは概念指向型であるべきであると考えます．過去の情報工学は工学の問題，信頼性であるとか，フォールトトレランスであるとか，頑丈であるとか，保全性，正確性，安全性等に焦点をあてていました．また過去と現在の情報工学は，より大型にとかより小型に，より速く，より安く，より多くといったマーケットの問題に重点が置かれてきました．未来のインフォマティックスのシステムは人間の問題，機械を介した人間対人間のインタフェース，自然言語による対話に焦点をしぼって欲しいと思います．要するに未来のインフォマティックスのシステムはより良いものを指向してほしいということです．

過去の情報工学においては物理学者とか電気工学技師といった人たちが中心でした．彼らは開発，いわゆる進歩の主導権を握っていました．過去においては，プログラマやソフトウェア技術者は奴隷の地位にありました．彼らはハードウェアの連中の考えに従わざるを得ませんでした．

未来において，プログラマはインフォマティックスの分野を通暁することが必要になると考えています．

ここで一般的な話を終わりまして，次にソフトウェア工学とプログラミング，更に具体的な話題に入りたいと思います．

4. ソフトウェア工学とプログラミング

3年前にこの会場でベーム博士がソフトウェアのライフサイクルについて話しました（訳者注．第6回ソフトウェア工学国際会議．Dr. Berry Boehm）．

私は今日は同じ問題を若干違った観点からみてみたいと思います．ソフトウェアシステムの寿命，一般にシステムの寿命を考える場合，システムの要求仕様が確立し，それからそれらの開発に入り，その最後の段階でこれらのシステムを据付け運用します．システムの保守はすべての局面で含まれます．

インフォマティックスの技術をマスタした人たちのグループが最初の段階では重要です．中間段階ではプログラマが，ソフトウェア技術者は更にその後で登場します．ここで再びプログラマとソフトウェア技術者との違いをことばを換えて述べますと，プログラマは意味に関心があって，プログラムやプログラミングを形式対象として扱います．ソフトウェア技術者は実践に関心があるべきで，生産と製品，品質問題を管理，監視しなければなりません．

ごく簡単にソフトウェアの品質問題はなにか述べますと，私はそれらを二つの分野，即ち生産の問題と製品の問題にわけて考えます．我々は，効率よく，予測可能で，経済的かつ管理できる方法を捜してきました．我々は正しい，頑丈な，信頼性の高い，正当な，保守可能な，フォールトトレラントな，安全なソフトウェアシステムをつくる方法を捜しています．日々このような問題を取扱うのがソフトウェア技術者の分野です．

プログラマに対しては私はプログラムとプログラミングを形式対象として扱って欲しいと思います．端的に申しますと，プログラミングは仕様と設計と実施で構成されており，私はプログラマはこれら仕様，設計，実施を形式対象として扱ってほしい，そして仕様の設計への変換，設計の実施への変換についても形式対象として扱って欲しいと思います．

5. プロジェクトグラフ

これから私の話の第3部に入りますが，プログラムとプログラミングを形式対象として扱う提案について詳細にふれたいと思います．

ここで私がいま同僚と共に行っている研究開発について触れますが，いまから話のスタイルをかえて，技術的な内容をお話したいと思います．プロジェクトグラフという概念を皆さんに紹介いたします．これは有向グラフですが，ソフトウェア開発の計画を定義しています．その意味ではこれはソフトウェア開発のモデルを構成しています．これらのグラフのノードはドキュメントに至るアクティビティを指します．これらのドキュメントはある理論に基づいています．ノードを結ぶ弧あるいは辺はドキュメント間の関係を表します．

以下に示したグラフ(図-1)は大変複雑な言語のコンパイラの開発に関連して使われたものです．これはデンマークソフトウェア研究開発センターで，Adaのコンパイラばかりでなく，CHILL のコンパイラの生産に従事したときの基本的なプロジェクトグラフです．

このグラフは基本的に三つの部分，即ち，仕様または形式定義部分，設計部分，それから下に向かってコード部分があります．

ある意味では我々はこのプロジェクトグラフを実行しているのです．3番目の部分のおわりで皆さんにこのようなプロジェクトグラフを実行するためのマシンの概要を示します．しかし覚えておいてほしいのは，これはコンパイラのようなある種のソフトウェアにのみ適用可能であるということです．オペレーティングシステムの開発のためのプロジェクトグラフはまったく違ったものになります．そして又データベースシステムの開発のプロジェクトグラフとも違ったものになります．これらのプロジェクトグラフの実行は入力のないノードから始まります．そして平行してアクティビティが実行されるように，トポロジカルな順序に分岐していきます．プログラミング言語は一般的によく知られているし，私自身コンパイラの開発に慣れておりますので，この例について詳しくお話しようと思います．

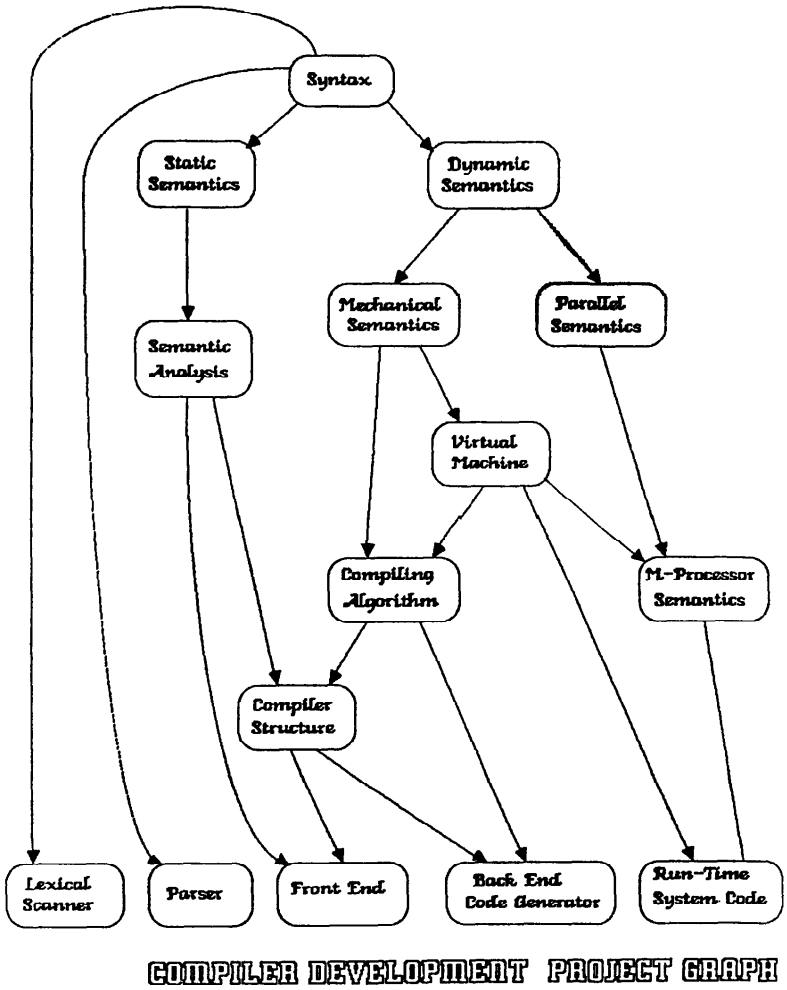


図-1

5.1 プログラミング

最初我々はプログラミング言語の構文を確立し，それから互いに独立して構文から静的，動的意味を作りだします．静的意味の定義はプログラムの静的特性を明確にします．言語によりまして，たとえばアルゴール型，パスカル型，エイダ型の言語の特徴は静的にどの型であるか調べることができます．たとえば使われている名前が定義されているとか，オペランドがそのオペレーションで期待されたものかどうか，またパラメータと手続きのアーギュメントリストの間が整合性がとれているとか，そのほかいろいろあります．動的意味はプログラムの実行時の特性を表します．静的意味はコンパイラのフロントエンドのところで処理されます．もしコンパイラを開発しているとすると，動的意味はそのコンパイラのバックエンドで生成されるコードの形で実現します．

形式定義の段階では，なにが問題なのか，なんの言語のコンパイラを作ろうとしているのか，できるだけ抽象的に理解しようと努めます．抽象化によって複雑さを克服することができます．そして我々がインプリメントすべきものはなんであるか理解することができます．

次にこれをどうインプリメントするか見いださなければなりません．抽象的定義では直接コーディングに使うにはあまりに抽象的過ぎます．そこで抽象的静的意味をもう少し具体的なものに変換します．同様に動

的意味に対しても，もっと具体的なものに変換します．言語のシーケンシャルな側面に対しても，またパラレルな側面に対しても同様です．

メカニカルセマンテックスから Ada プログラムを実行する仮想マシンを作り上げることができます．

あるいはメカニカルセマンテックスノードから仮想マシンノードへの辺をとおらなくても，我々は現存のターゲットマシンを受けることもできます．プログラムの実行時の構造を示すメカニカルセマンテックスとターゲットマシンから，いわゆるコンパイリングアルゴリズムを作ることができます．意味解析とコンパイリングアルゴリズムのノードで，コンパイラがどのようにではなくて，なにをするべきかを明細に定めます．そこでそれをどのようにすればいいか見いだすことができるのです．こうしてこの段階，即ちコンパイラストラクチュアのノードで，コンパイラそのものの構造を決めるのです．シンタックスからはコンパイラのレキシカルスキャナとパーサを自動的に生成することができます．なにをどのようにチェックするのかということからコンパイラのフロントエンドをコード化します．どのようなコードを生成するのか，またどのように生成するのかということから，我々はバックエンドをコード化します．これらの4つの部分でコンパイラを構成します．同様にしてターゲットマシンの実行時システムの設計とコード化をすることができます．したがってこれは数多くのドキュメントをある順序で作成する詳細な計画です．

5.2 ソフトウェア工学

この計画をソフトウェア工学の観点から論じてみようと思います．1980年11月にこのプロジェクトグラフを作り，コンパイラを作成するのに，4年間で33人年の人工がかかると予測しました．これらのアクティビティの各々にどれだけの人工が必要か予想しましたが，実際より33%も低く見積りました．我々はあまりに楽観的でした．33人年でできると考えたのに，実際は44人年かかりました．6人が1年間を形式的定義にのみ携わりました．つまり4年の中の最初の1年はコンパイラについてはまったく考えないで過ごしたわけです．それから12人が1年かけて，コンパイラの設計の「なにを」を，また2人が実行時システムに1年を費やしました．2年コンパイラの仕事をやっていて，まだコンパイラの構造がどうなるかわからないのです．それから10人がおおよそ半年間，コンパイラの詳細設計についてただ考えました．

2年半経過した後も，コンパイラの1行もコードは書いていないわけです．皆さん，コンパイラプロジェクトでありながら，2年半かけてなにもコードを書かなかったのです．それから12人がコーディングに1年半かけました．

静的セマンテックスに従事した人はセマンテックアナリシスには従事しませんでした．人を回したわけです．動的セマンテックスをやった人がセマンテックアナリシスをやりました．ここでは新人を入れましたこのプロジェクトの遅れは新人のためではなく，Adaに変更があったからです．この段階で大勢の若い学生を使いました．彼らは Ada のこともターゲット言語のことも知らなかったわけですが，コンパイラのコード化ができました．というのは基本設計の仕様にただ従ってやったからです．

形式的定義をするのに 14,000 行，設計を明確にするのに 59,000 行，コンパイラをコード化するのに 220,000 行を費やしました．

5.3 理論的コンピュータサイエンス

このプロジェクトグラフについてプログラミングの観点並びに工学的観点からお話ししましたが，つぎに，より理論的な観点からプロジェクトグラフを話したいと思います．作成されたドキュメントはある種の理論の対象を表しています．例えば文脈自由文法と言語の理論の対象が具象構文です．抽象構文はスコットドメインでの目的の一つです．ほかのノードはほかの理論の対象物となっています．たとえば，コンパイリングアルゴリズムは拡張された属性文法理論の対象物です．レキシカルスキャナは有限状態マシンの理論の対象物の一つです．構文解析はプッシュダウンマシン理論の対象物であり，フロント及びバックエンドに書かれたプログラムはある意味ではプログラムを訂正するホアロジックの対象物であります．

5.4 ソフトウェア開発システム

ソフトウェアを開発するための未来のソフトウェア技術を開発したいと考えています．プログラマ，ソフトウェア技術者，科学者及び経営者のいずれのためにもプロジェクトグラフを実行する支援システムを開発したいと思っています．我々はこれらのグラフを実行しようとしているということを念頭においていただきたい．つまり，プログラマやソフトウェア技術者や経営者とともにこのグラフを対話的に実行するソフトウェア・ハードウェアのシステムを必要としているのです．ソフトウェアを開発するシステムを必要として

いるのです．それは基本的には二つの主要部分で構成されています．一つはプロジェクトグラフを実行する部分，もう一つはいわゆる理論マシンです．

第1のマシンであるプロジェクトグラフマシンは基本的に二つの要素から構成されます．まずプロジェクトグラフを定義し，変換し，出力するエンジンで，開発を開始する前にプロジェクトグラフを作ります．プロジェクトグラフはほかのプログラムを構築するためのプログラム，つまり，メタプログラムのようなものです．第2はプロジェクトグラフのノードと辺の実行を扱う部分です．ノードを実行するということはどういう意味か．これは構文的にも，意味論的にもなにかを意味します．構文的には編集し，書式化し，バージョンと構成を維持し，日報をドキュメントにしておく必要があります．それからこれらのドキュメントを意味論的に扱うのです．我々が書いたものの意味を我々のマシンにわからせたい．我々が作っている各種のドキュメントの静的意味の形式のチェックができるようにしたい．仕様なり設計を記号的に実行するとか，プロトタイプを動かし，これによってドキュメントの検査，分析，確認をしたいのです．

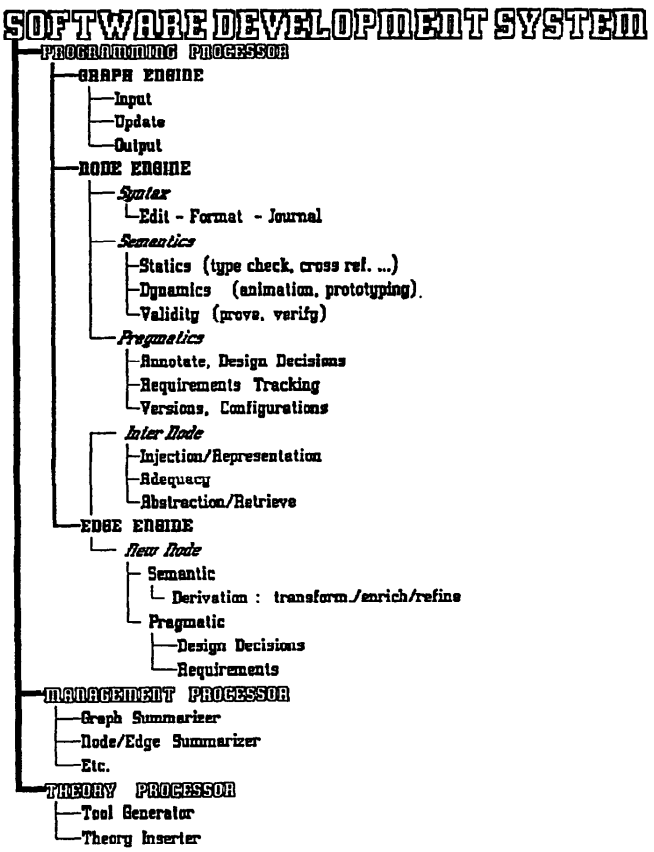


図-2

このような方法で，まずドキュメントをつくり，そのドキュメントを検査分析します．そうすればこのドキュメントについて当該ノードの作業は終わったと考えられます．そこでつぎの新しいノードの作成に進みます．新しいノードを最初のノードと同じやり方で作り，一つの辺で結ばれた隣接するノードを作ってから，それらの間の関係の特性，属性について検証する方法と，あるいは，古いノードから洗練するとか内容を豊かにしていくといったような変換を行って新しいノードをつくる方法があります．ここでは意味を持っているドキュメントに加える操作がわかるマシンなりシステムが欲しいわけです．どのような変換をするにしろ，設計上の決定の結果であり，我々はその設計上の決定を記録したいわけです．これは構文の問題です．そして新しい設計に具現化された初期の要求を見失わないようにしたいわけです．したがって，プログラミング部分，すなわちプロジェクトグラフマシンのプログラミングプロセッサはプロジェクトグラフの作成と実行を行う部分です．プロジェクトグラフマシンはもう一つの側面があります．それは管理面です．管理はグラフの中でどこまで進んだか，ノードや辺でどれだけ作業が行われたか開発を監督，統制できなければなりません．このシステムは特定の仕様や設計言語のために作っているのではなく，多様なものに対応できるように考えています．決定論的属性，非決定論的属性，同時処理属性，時間依存属性といったようなシステムの異なった属性を扱えるような多様性が必要です．今日現在，仕様言語なり，設計言語の一つとしてこのような属性のすべてを平等にうまく取扱うことのできるものはありません．

今日，多数の理論がありますが，明日はまたほかの理論がでてくるかもしれません．ということは我々の

マシンは新しい仕様言語や新しい設計言語の静的動的意味が理解できるように新しい理論を組み込める必要があります.

5.5 RAISE 計画

したがってこれは大変大がかりなプロジェクトで,今日の私の話しの主題であるソフトウェア工学の将来の技術の典型的なものであることを願っています.我々はこれを RAISE 計画, RIGOROUSE APPROACH TO INDUSTRIAL SOFTWARE ENGINEERING 即ち産業用ソフトウェア工学への厳しいアプローチとよんでいます.これは欧州共同体のヨーロッパ情報技術戦略計画 ESPRIT の一部となっています.この計画は6分野にわかれており,我々の計画は第2のソフトウェア技術に属しています.この分野では現在この計画が最大規模のものです.ほかの大型プロジェクトは100人年程度です.この計画は4社が共同で推進しています.即ち,非営利団体でソフトウェアの開発を行っているデンマーク・データマチック・センタ,スイスの会社のデンマーク支店で電力工学関係のコンピュータ化を進めているノルディック・ブラウン・ボバリ社,それと英国の STL と ICL の2社です.この計画は願わくば三つの成果を上げるはずです.それらは手法とテクニックの形態における知的成果,エクスパート援助システムを含むソフトウェア開発環境または支援システムの形態での実質的成果,及び企業に技術移転する教育,訓練材料の形態での知的成果です.これは大変リスクをはらんだプロジェクトで,そのためコンサルタントを雇わざるを得ません.ご存じの方もいらっしゃると思いますが,ドイツのマンフレッド・ブロイ教授,カリフォルニアの SRI インタナショナルからジョゼフ・A・ゴーゲン博士,イタリアのモンタナリ教授,マンチェスターからクリフ・B・ジョーンズ教授,パリからJ・R・アブリエル博士,スコットランドからゴードン・プロトキン博士,このような人たちの協力を得て,あまり多くの問題に遭遇せずにすむのではないかと期待しています.

6. 結　論

それでは4番目の部分,私の話の最後の部分に入りたいと思います.さきに述べました三つの部分をふまえまして結論めいたことを申しあげたいと思います.

抽象的レベルでは情報技術とインフォマティックスについてお話しました.より具体的なレベルでは,ソフトウェア工学とプログラミングの違いをお話しました.極めて技術的なレベルではソフトウェア開発支援システムの可能性のある方法のうちの一つを説明しました.そのため,この影響が(1)ソフトウェア工学,(2)プログラミング,(3)管理,(4)コンピュータ製造業者及びソフトウェアハウス,(5)大学にどのような影響をもたらすか疑問がでてくるでしょう.

ソフトウェア技術者とプログラマがしばしば同一人物であるということがありますが,私は違った時間に違った仕事をしているのだと考えているということを覚えておいていただきたい.

6.1 ソフトウェア工学

それではソフトウェア技術者はなにに挑戦しているのか.私にとって,それはプログラミングとシステムの間におけるソフトウェア工学の役割を理解することであると思います.我々は多くのことばを話します.今日は皆さんに若干の術語の定義をお話しました.その多くは耳新しいものであったと思います.それより悪いことは,同じことばで現在の定義と矛盾を来すかもしれないということです.技術者として互いに話ができるような専門語を確立し,従い,更に開発することを職業とやっていかなければなりません.私の個人的な信念ではソフトウェア技術者として振舞うときもなおプログラムとプログラミングを正式の対象物であると受け入れなければならないと考えています.ソフトウェア技術者としてそれは受け入れ難いことかも知れないが,彼らの役割は技術開発ハンドブックを開発または作ることであると考えます.私にとってソフトウェア技術者は基本的に道具を作る人であります.

6.2 プログラミング

プログラマまたはプログラミングの挑戦はなにか.私からみると彼らは多くの異なった技術を身に付けています.土曜日に私は渋谷へ行き,東急ハンズで日本の道具類,鋸,かんな,包丁,金槌等を2万円なにがしで買いました.どれもみな形も美しければ,機能も実用的です.私はこれらの道具を使おうというのではありません.それらを壁に飾ってそれらの美を鑑賞し,すばらしい家具を作るには,たくさんの種類の道具がいるのだということを思い出そうと思っています.

プログラマはなにをどのように設計するかを,理論的に明細に記す方法を学ぶ必要があります.仕様を設計にどのように変換し,どのように洗練させ,内容を豊富にしていくかを学ぶ必要があります.彼らはその仕事の成果を一つのことばだけでなく,違った種類の

ことばで，実証し，検証し，確認することを学ばなければなりません．

プログラマ達はいままであまりにも成功してきたので，いまだに石器時代の道具を使って仕事をしています．彼らと対であるハードウェアに従事している人たちが，VLSI 回路の設計に非常に精巧な道具を要求することができるというのにです．皆さん，いまこそ，プログラマ達がハードウェア技術者よりももっと精巧な道具を要求すべきときがまさに来たれりということです．

なぜ要求すべきなのか．それはプログラマが設計する目的物の方が，ハードウェアの対象物よりもはるかに複雑だからです．プログラマは理論を作る人であります．

6.3 マネージメント

マネージメントの挑戦はなにか．プログラミングは科学であると理解すること．プロジェクトグラフがよく理解されないかぎり決してそのプロジェクトに着手しないということ．それを自分でもできるような第一級の管理者や監督者のみを使うべきだということであります．もし我々がこれらの二つの原則に従うとすれば，今後 10 年間はソフトウェアを作れないでしょう．管理にはほかにもいくつかの努力目標があります．

6.4 産業及びビジネス

私は大手のあるコンピュータメーカで 13 年間仕事をしてきました．そのメーカは西欧社会では 50% 以上のシェアを持っています．したがって，私がなにを言おうとしているか皆さんはご承知のことと思います．

ウィーンのゼマネク教授のすばらしい研究室で，私達は我々自身の方法を作ろうとしました．私の苦い経験から，学究的な労働の成果，国際的な場に曝され洗練された成果を同時に取り入れることができないかぎり，産業がそれをしないように勧めます．

プログラミングは知的プロセスであり，ソフトウェアは知的製品であって，ハードウェアで理解されている物差しは適用できないということを大手企業は知るべきであると思います．

6.5 アカデミックな世界

大学の挑戦はなんでしょうか．第 1 に，私は大学で教えていることはすべて間違っていると思います．これを語るにはもう一つ別の講演をしなければならないほどです．わたしの周辺で，大学のグループがプロトタイプの開発にあまりにも多くの時間を浪費したことを，私は見てきました．彼らはもっと理論に力を集中するべきです．

6.6 要 約

さて最後の段階にきまして，結論として私は，情報技術は素晴らしい世界ではありますが，それはいまは物質的量の物であります．情報技術というものに皆さんが日本でやっておられるように，認識科学の側面，通信または言語学的側面，組織的側面，あるいは生物学的側面を増強することにより知的な質の世界，インフォマティックスの世界に入ります．

私にとってプログラミングはアイディアの構築物であり，ソフトウェア工学はこれらの夢を実現する実践の場であります．

私は将来の計算科学者は過去を知らなければならないと考えます．過去のコンピュータとか計算科学を理解しろといっているのではありません．そこから得られることはわずかです．そうではなくて過去と現在の理論と科学のフィロソフィを理解すべきです．

私が今回講演をするようにという親切なご招待をいただきましたときに，将来について語ってほしいという要請を受けました．そこで私が感じたのは科学者に未来について語るように頼むべきでないということです．

むしろ科学者に今，なにを研究しているか話すように頼むべきです．なぜならそれが正に未来だからです．

ご静聴ありがとうございました．