コンピュータソフトウェア Vol.14, No.1 pp.75-82 (1997)

トピックス

ICSE '96 会議報告

井上 克郎　佐伯 元司　鯵坂 恒夫

1 はじめに

今年で 18 回目を迎えるソフトウェア工学国際会議 (International Conference on Software Engineering) は、ドイツのベルリンにある Technische Universität Berlin で開催された。

会期は、前後の併設ワークショップ、シンポジウムを含めて、 1996 年 3 月 25 日から 31 日までの７日間、チュートリアルは 25,26 日、本会議は、 27 〜 29 日の間の３日間である。

この会議は、毎年一度、３月から５月までの間に開かれる。開催地は、米国と米国以外とが交互に選ばれ、さらに、米国内での開催の場合は、東側、西側の場所が交互に選ばれている。 15 回会議は、東海岸の Baltimore、一昨年の 16 回はイタリアの Sorrento で開催され、昨年は、西海岸の Seattle で開催された。来年は、 Boston、そして、 1998 年の第 20 回会議は京都で開催することが決定している（平成１０年４月１９日から２５日まで、国立京都国際会館）。

この会議では、過去、ソフトウェア工学の重要な論文が発表されてきている。例えば、 L. Osterweil が第９回会議の基調講演で発表した “Software Processes are Software Too” という論文は、その後のソフトウェアプロセスの研究や応用に大きな影響を与えた。また、ソフトウェア工学全般に関する最大の会議として、チュートリアル、ツール展示などへの参加を含め、多くの人が集まってきた。

現在は、ソフトウェア工学のメトリックス、プロセス、形式手法、オブジェクト指向、などそれぞれの分野に専門の会議がたくさんあり、参加者が分散してしまっている傾向が見られる。しかしこの会議では、各分野の優れた論文が発表され、また、興味ある基調講演やパネル討議が聞けるので、米国開催の場合、 600 人程度、海外開催の場合は 400-500 人程度の参加者が集まる。しかし、今回は、 852 人という多くの参加者を集めた。参加者の内訳は以下の通りである。

ドイツ	351 人
ドイツ以外のヨーロッパ	268 人
南北アメリカ	173 人
アジア・オーストラリア	60 人

このうち、５３%が研究者（含む政府機関）、４７%が会社関係者である。

チュートリアルや併設ワークショップが２５、２６日の両日行なわれ、続いて２７〜２９日まで本会議が行なわれた。本会議では、３つの基調講演、５２の論文発表、３つのパネルの他、実務経験報告、解説、ミニチュートリアル、ワークショップレポートなどのセッションから構成されていた。本会議と並行して、ツールの展示会場が設けられ、本の展示や企業のシステムの展示が行なわれた。２９日の本会議の終了後、ワークショップが３１日の午前中まで引続き行なわれた。

以下、それらの内容や会議の運営について述べる。

2 基調講演

2.1 DeMarco

“The role of software development methodologies: Past, current and future”, by

Katsuro Inoue, 大阪大学基礎工学研究科, Graduate School of Engineering Science, Osaka University
Motoshi Saeki, 東京工業大学情報理工学研究科, Graduate School of ????, Tokyo Institute of Technology
Tsuneo Ajisaka, 京都大学工学研究科, Graduate School of Engineering, Kyoto University
コンピュータソフトウェア, Vol.14, No.1(1997), pp.1–8.
1996 年 7 月 31 日受付.

Tom DeMarco, The Atlantic System Guild, USA.

一番最初の基調講演では、 Tom DeMarco が、プロセスは有害である、という主題で話をし、フロアでかなり議論を巻き起こした。彼は、４つのパラドックスと称してプロセスの考えの問題点を指摘した。

1. プロセスを改善すると、簡単な作業が自動化されて、結局人間が深く介在しなければらない困難な問題（エッセンス）しか残らない。
2. 戦争の歴史を見ると、重武装だが動きにくいものが、軽装備だが動きが軽いものに負けている。プロセスの考えは、この重装備にあたる。
3. 銀行システムなど過去の大規模システムの再利用はうまくいかなかった。再利用のプロセスを考えてもだめで、再利用したくなるようなものを作る必要がある。
4. 人間はリスクを乗り越えて大きな変化をつくり出す力がある。新しい技術で大きな利益を作り出すことを目的とすべきである。しかし IS9000, CMM などは、リスクの小さくして安定した開発を目的としており、それでは、技術の発展は起こらない。

プロセスの問題の例としてあげたのは、ＡＴ＆Ｔなど CMM レベル３やそれ以上を得ている会社が、最近大量にレイオフを出しているという事実である。一方、Bohem の Win-win 理論、 mediation の技術などが将来重要になるという指摘を行なった。

この講演に関して、賛否両方の意見が、会議発行の新聞で述べられている。(http://www.cs.tu-berlin.de/~ icsewow) かなり一方的な講演ではあったが、会議の冒頭で、参加者同士の会話のきっかけになる、面白い内容であった。

2.2 Hoare

“How does software get so reliable without proof?” by Anthony Hoare, Oxford, UK.

彼は、基礎的な研究の重要性を主に述べた。

現在、非常に大きなプログラムも実用的に使われているが、それらのプログラムの作成に関して、形式的な手法は、直接的には大きな貢献はしなかったが、プログラミングの考え方の基礎に大きな影響を与えてきた。

うまくいかなかったプロジェクトには管理上の問題があろう。特に、要求仕様に力を入れなかったものは失敗している。形式的手法は、その仕様作成の分野で成功している。

また、テストの技法は、ソフトウェアの信頼性を上げる上で大きな貢献をした。さらに、安全性を高める上で、 Over-Engineering（橋や建物で、実際かかる荷重の２倍とかに耐えるように設計すること）のための、防御的なプログラミング (defensive programming) やデータの正当性のチェックが役立ってきた。

計算機科学者は、現実から離れて先を行っている必要がある。純粋な科学的な興味が研究の将来を切り開いていく。一方、実務家は、研究成果のなかから役立つものを見つけて、現実に使えるようにするべきである。そのための教育が重要である。

この Hoare の講演に関して、突っ込み方が足りない、表層的な意見である、現実には基礎研究がほとんど役立っていない、などの意見が、新聞で述べられていた。

2.3 Basili

“The role of experimentation: Past, present, and future” by Victor R. Basili, University of Maryland, USA.

最終日の講演は、 Basili の実験的ソフトウェア工学の話であった。

ソフトウェア工学分野以外の、例えば医学研究などでは、研究で得た新たなアイディアは、かならず実験を行なって、それが良いか悪いか、効果があるかないかの評価を行なってきている。しかし、ソフトウェア工学の分野では、多くの新たな提案は、そのような評価なしで論文として掲載されてきている。

新たな提案を評価するためには、管理された実験が必須である。幾つかの被験者のグループのもとで、提案する方法と従来の方法とで実際に作業を行ない、その結果を統計的に解析、検討する必要がある。このような実験を通じて、クリーンルーム手法やシナリオに基づくプログラム読みの効果が確かめられた。

一方、このような実験には、非常にコストがかかる。一つのことを確認するために、多くの人間と時間がかか

る。そのような実験の情報を交換しあい、効率を高めるための交流組織としてISERNを設けている。また、実験的ソフトウェア工学のための雑誌も発行した。

彼の話は、何度か日本でも紹介されており、比較的なじみのあるものであった。

3 論　文

今回、この会議には、34ヶ国から213の論文投稿があって、そのうち52本が採録された。日本からは、2つの論文が採録されている。

今回は、ソフトウェア工学の各分野それぞれバランス良く採録されているように見える。要求仕様、メジャメント、テスト、解析、形式的手法、構成管理、分散システム、再利用、プロセス、開発環境、保守などのセッションが作られた。

これらの論文のなかからいくつかを紹介する。

3.1 セッション2A：プログラム理解と解析

- The Design of Whole-Program Analysis Tools

  大規模なソフトウェアシステムを理解するための効率的なツールを作るのは困難である．多くのプログラム理解のためのツールはまずプログラムの表現 (control-flow 表現,data-flow 表現) を作るため，大規模システムの解析の場合に非常に多くの空間と時間が必要となる．しかしこれらの表現の多くは解析の間に使用されない．

  そこで本論文で設計するツールでは，プログラムの表現をあらかじめ作っておくのではなく，それらを要求に応じて作る．これにより，時間・空間の節約になる．また，空間の節約のためまれにしか使われない表現は使用後に捨て，必要であれば再計算する．これにより記憶装置の遅い階層を使わずにすみ，時間の節約にもなる．更に，解析の際の時間と正確さの折り合いや終了条件についても，ユーザに制御できるようにする．ここで，時間と正確さの折り合いとは関数呼出の際に再計算を行う深さについての設定で，終了条件とは解析の繰り返し回数である．

  これらの技術は MUMPS 言語で書かれた病院管理システムである Comprehensive Health Care System (CHCS) のスライサの設計に適用され，その結果についての考察を行っている．

  要求に応じてプログラム表現を作る，余分な内部表現を捨てるなどはよくある話である．しかし，実際に百万行規模のシステムの解析を行いツールの方針が正しいことを示している．また、関数呼出の再計算の深さを2,3に制限しても，無制限にしたものとほとんど結果のスライスの大きさは変わらない、解析の際の繰り返しのはじめの20興味深い。

- How to Identify Binary Relations for Domain Models

  ソフトウェアシステムの分析ないし要求定義の初期過程で用いられる実体－関連 (E-R) モデルやオブジェクト構成モデルには，多くのさまざまな関係が登場するが，与えられた要求から何を関係と同定すべきかの指針は，これまでほとんど明確に示されてはいない．一方，多くの実システム例で，要求は自然言語で提示され，またそこで用いている用語定義集が整備されることが多い．本論文はこの用語定義集を解析して，関係の候補を効果的に半自動抽出できることを示している．用語定義集は主に名詞を見出しとして整理されるため，動詞句よりも名詞すなわちオブジェクトクラスの定義のされ方を中心に解析を進めることになるが，これが動詞に注目するよりもむしろ実態に近い関係を抽出する結果となる．

  支援ツールを用いた実験，応用経験でも，二次的に導出可能な余計な関係もあわせて抽出してしまう傾向はあるものの，拾わなければならない関係を落すことは，深い対象知識を必要とするものを除き，あまりなく，良好な結果を示している．

3.2 セッション2B：要求獲得支援

- An Analytic Framework for Specifying and Analyzing Imprecise Requirements

  この論文は、顧客からの不正確な要求をFuzzy Logicを用いて数値化し、不正確な要求に対し、開発者側がどのような具体的な要求案を採用すればよいかを決めるための手法について述べたものである。まず、例えば「開発コストを小さくしたい」といった不正確な要求が、採用する開発プロセスに

よってどれだけ満足されるかの度合いを０～１までの値を返すFuzzy関数でモデル化する。０はそのプロセスでは全くこの要求を満たさない、１は完全に満たすことを表す。２つの不正確な要求があったとき、これらをモデル化したFuzzy関数の挙動の関係でConflictの度合い（２つの関数をグラフとして見たとき、勾配が逆になっている区間が何割あるか）やCooperationの度合い（勾配が同じになっている区間の割合）を定義し、これらの数値をもとに、顧客の不正確な要求に対する、開発者側の具体的な要求案をある種のトレードオフで優先順位をつけて求めようとするものである。

3.3 セッション2C：テストと解析

- An Empirical Study of Static Call Graph Extractors

  call graphとは、プログラム中の関数の呼び出し関係を表現したものであり、主にプログラムを理解するのを助ける目的で使われる。この論文では、call graphを作成するツールを5種類用いて、いくつかのソースに対し実際にcall graphを抽出し、その結果に基づき、ツールごとの特徴を分析している。

  抽出したcall graphがツールごとにかなり異なっており、その原因は、マクロをどう扱うか、関数へのポインタを認識できるか、などである。それによって抽出された情報の量が時には100倍以上も違っていたりしている。また、あるはずのない呼び出し関係を誤って抽出したりする原因も分析している。

  同じようなツールなのに顕著な違いがみられることを示した本論文は、このようなツールを使うものにとっては大変興味深いものである。

3.4 セッション3A：オブジェクト指向の実用

- Industrial Experience with Design Patterns

  ７人の著者がDesign Patternを用いて、ソフトウェア開発を行ったときの経験をまとめた論文。まず、６つの会社でソフトウェア開発（開発対象のソフトウェアの分野も異なる）にPatternがどのように用いられたかの概略が述べられ、彼らの経験から得た知見が述べられている。それによると、1) Patternは開発チームのメンバー間のコミュニケーションを向上させる、2) Patternは実際の設計より抽出される、3) Patternは設計の本質的な部分をコンパクトな形で捉えることができる、4) Patternは最良の実践を記録し、かつそれを再利用することを促進してくれる、5) Patternは必ずしもオブジェクト指向である必要はない、6) 組織の中にPatternに関する助言者や指導者を置くことにより、Patternを用いた開発が開発者に受け入れられやすくなる、7) 良いPatternは書くのが難しく時間もかかる、8) Patternの実践は最も重要である、ということである。

3.5 セッション4B：コンポーネントベースのソフトウェア

- Experience Assessing an Architectural Approach to Large-Scale Systematic Reuse

  製品単位の大きさの部品を再利用して組み立てる，いわゆるコンポーネントウェア開発の経験をまとめている．部品を統合する仕掛けとしてOLE，部品としてVisioとAccess，部品を仲介するプログラミングシステムとしてVisual Basicを用いている．基本的には，OLEのような統合機構によるアーキテクチャ不整合の解決，コンポーネントウェア開発の生産性の良さや汎用性，実用性を認める論調であるが，指摘された問題点の方がむしろ今後の展開の参考になろう．問題点は統合機構の側の問題と部品の作られ方の問題の二つに分かれる．統合機構と部品間仲介系の大きな問題として，部品間のイベントやメッセージの双方向の流れをすべて的確に捉えきれないことがあげられている．一方，部品の方も，その内部状態の重要な変化を伝えるのに必要なイベントをそもそも適時に発生していないことがあるほか，統合機構に十分に対応していない，カスタマイズ可能性が不十分，といった問題がある．

3.6 セッション6A：形式的な設計手法

- Executable Object Modeling with Statecharts

システムをオブジェクト指向的にモデル化するためのチャート図の提案。O-chart と呼ばれるクラスとその間の関係（リンクと呼ぶ）を表す図と、著者らが以前に開発した Statechart を用いて表現する。Statechart は、存在するオブジェクト間の通信だけでなく、オブジェクトやリンクの動的な生成・消滅といった振る舞いを記述するために用いられる。このため、従来の Statechart で用意されている基本オペレーション（動作のトリガなど）に加えて、生成・消滅を行うオペレーションが用意されている。つまり、状態遷移が起こる際に、オブジェクトやリンクの生成・消滅が起こる。O-chart は、いくつオブジェクトが存在するかや、オブジェクトの集約による階層構造（複合オブジェクト）を図形の包含関係で記述できるようになっている。Inheritance は、Statechart に新たな状態遷移を追加することによってサブクラスを作り出す、いわゆる「記述の共有」の機構を持っている。また、テキスト部分の記述を C++ の記述とできるだけ合わせることにより、C++ プログラムへの変換のしやすさもねらっている。

3.7 セッション 8A：プロセスの効果

- A Systematic Survey of CMM Experience and Results

  本論文では，実際に CMM による評価を行なった組織を対象にして，CMM によるソフトウエアプロセスの改善 (Software Process Inprovement, SPI) がどのように行なわれているかを調査した結果を詳細に報告している．この調査では SEI で管理されているデータベースから，過去 1 年から 3 年の間に CMM による評価を行なった組織を対象にしているため，特定の成功例に片寄らない，実際の CMM による効果を類推することに成功している．

  本論文では「評価実施後，各組織においてどのようなプロセスの改善が行なわれていたか」「プロセスの改善が成功もしくは失敗した原因はどこにあるのか」「プロセスがより成熟した組織の方がそうではない組織より効率が良い」といった点について，具体的な調査結果の数字を元に議論を行なっている．指摘された点については，ある程度予想できるものが多いが，実体をより反映している調査データを元に，具体的な数値を用いているために，説得力のあるものとなっている．

- DYNAMATE: Dynamic Task Nets for Software Process Management

  タスクネットと呼ばれるソフトウェアプロセスのモデル化用のネットを提案。このネットは、制御フローおよびデータフローの２つの見方でソフトウエアプロセスを記述できる。ソフトウェア開発プロセスをこのネットで記述しておくと、記述に応じて支援系が開発プロセスの実施を管理してくれる。タスク（開発作業）は、ネット上のノードで表現され、あらかじめ用意された状態遷移図 (待ち状態、終了状態、休止状態、実施中などの状態を持つ）にしたがって実施されていく。また、生成されたプロダクトもネットの枝にしたがって、伝播されていく。このネットの最大の特徴は、プロセスが実施されるにしたがって、状況に応じてプロセス記述も変化するというダイナミズムを、グラフ上の書き換え規則を用いてプロセス記述を書き換える枠組みを用意している。つまり、メタレベルの記述として、あらかじめどのようにプロセスが変化していくかをグラフ上の書き換え規則でプログラムしておくことができる。

3.8 セッション 8C：環境

- Simplifying Data Integration: The Design of the Desert Software Development Environment

  ソフトウェア開発環境（プログラミングレベルを中心とする）におけるデータ統合を簡単かつ効果的に実現する方法に主眼をおいた論文．PCTE の導入などによる本格的なデータ統合は，既存環境の継承が容易でないためリスクが大きい．本論文でいう簡単なデータ統合は，ツールの扱うデータ（ファイル）全体を統合するのではなく，関数，型，変数といった論理的要素（フラグメント）を集めてツール統合のための共有データとする．論理的要素は，その定義 – 参照関係の連鎖追跡を主として集められ，

実プロダクトファイルを横断的に捉える仮想ファイル（フラグメントファイル）が作られる．このようなフラグメント統合のほか，本論文の提唱する環境は，ToolTalk をベースとする制御統合，および FrameMaker をベースとする共通エディタからなる．この共通エディタの使用が環境の統合機能を最大限に引き出す要となっているが，それが環境全体の開放性を阻害することにならないか心配である．

3.9 セッション 10A：保守と改良

- A Scaleable, Automated Process for Year 2000 System Correction

2000 年問題とは：プログラムやデータにおいて、日付の年が 2 桁で管理されているために 2000 年の日付において生じる問題である。前年が 99 その次の年が 00 で表現されれば、不具合が起きる。筆者らは、このような問題を抱える COBOL で記述されたシステムとそのデータについて、年の表現を 4 桁になるよう修正するツールを開発した。
そのアルゴリズムは、まず、プログラムのソースリストから、年に関係ある変数を探し出す。探し出すのは、変数名のパターンマッチングではなく、意味解析による。解析には、変数の型の概念を拡張した属性が用いられる。すなわち、ある変数が整数型であれば、その変数には、年に関係ない／時刻として年を表す／期間として年を表す等の属性がつけられる。その変数が式に現れれば、その式に現れる他の変数や定数にも属性が伝播していく。これを繰り返して、すべての年に関係ある変数を探し出す。最後に年に関係ある変数を含む式や文を、修正ルールに従って書き換える。

筆者らのツールは、COBOL プログラムをの変数の定義を修正して再コンパイルし、ファイルのデータを書き換えるための修正プログラムを生成し、修正が正しく行われたことを確認するためのテストデータまで生成する。さらに、筆者らの対象は商用システムであるから、ツールはインクリメンタルな修正を実現する。すなわち、修正対象のシステムとデータを、運用しながら、徐々に新しいシステムとデータに変えていくことができる。実際に 300K 行の COBOL プログラムを対象に実験を行っていて、良好な結果を得たそうである。米国特許も申請中だそうである。
大規模システムでの使用を想定した包括的なツールであり、しかも実用的な解決を提供している点は興味深い。ただし、他の言語や他のクラスのアプリケーションへの一般化には、問題が残ろう。

4 経験報告とミニチュートリアル

“An operating system development: Windows 3”, Chip Anderson (Microsoft, USA).

と、いう題名で、Windows 3.0 の開発にかかわる話を中心に、マイクロソフトのソフトウェア開発について報告があった。

開発に用いている環境は、MS-C が８０%、残りがインテルのアセンブリ言語で、ソース管理、テスト管理、問題事項管理などのツールを使っている。また、バグ数のメトリクスを用いている。

総員５０名ぐらいのグループで、テストする人間と開発者がそれぞれ１５名、８名が文書作成、４名がプログラム管理者、３名がプロダクト管理者、２名が事務で、特にプロジェクト管理者はいない。こうすることによって、皆が製品に責任を持つようになる。

多くの内容は、最近出版された「マイクロソフトシークレット」に記述されているようなことが多かった。用いたスライドは、
http://www.mindspring.com/~ chipa/icse96.html
に置かれている。

ミニチュートリアルでは、ソフトウェア開発支援、statechart の歴史、大規模マルチメディアソフトウェア開発、ドメイン依存開発環境などについて解説が行なわれた。

5 その他のセッション、チュートリアル、ワークショップ

ソフトウェアの複雑度、ソフトウェア開発の工学的アプローチ、なぜひどいソフトウェアが良く売れるのか、の３つのテーマについてパネル討論が行なわれ、聴衆を数多く集めていた。

また、本会議の事前の２日間、１４種類のチュート

リアルがそれぞれの分野の専門家によって行なわれた。例えば、W. Humphrey による個人プロセス、B. Meyer と J. Nerson による Eiffel によるオブジェクト指向ソフトウェア構成などがあった。

さらに本会議の前後に、9つのワークショップやシンポジウムが開催された。

6 プログラム委員会

採択論文を選択し、プログラムを編成するためのプログラム委員会は、1年以上前から委員の選定が行われ、世界各国から３３名（委員長は除く）の委員が選ばれた。

プログラム委員長は Imperial College（英国）の Tom Maibaum と University of Maryland（米国）の Marvin Zelkowitz である。委員会の構成は、１４人が米国、１２人がヨーロッパ、４人がアジア（日本２人、香港、インド各１人）であった。

第１７回ソフトウェア工学国際会議（１９９５年米国シアトルで 1995 年４月２６日に開催）のときに、第１回のプログラム委員会が開催され、投稿論文の選択を行うための第２回のプログラム委員会の日程が検討され、それまでの作業手順が審議された。その結果、委員会は１０月の初旬にワシントンで開催される Foundation of Software Engineering の第３回会議（以下、FSE-3）に合わせて近郊で開催されることになった。

論文の査読に関する手順としては、投稿論文締め切り後、２週間ほどで全論文のアブストラクトを委員に送付し、委員がその中から、査読をぜひやりたいもの、査読できるもの、査読できないものを選択して、委員長に送ることになった。委員長はこのデータをもとに、担当者の割り当てを行った。これらの作業は、アブストラクトの送付を除いてすべて電子メールで行われた。なお、査読報告の送付や後で述べる委員会の事前の議論もすべて電子メールを使って行われた。

論文投稿数は全部で２１３編あり、１編につき３人の委員が査読につくため、１人あたりの担当は２０編程度となった。原則として委員のみが査読を行い、査読期間は1か月半弱であった。査読結果が直ちに電子メールで集められ、集計が行われ、委員会開催までの１週間の間で、評価が分かれた論文に関して、担当者同士が事前に電子メールにて協議（調整といったほうがよい？）を行った。

このような事前協議は、本会議の時間短縮には有用であるが、今回は委員会の前にある FSE-3 の会議に出席する委員にとっては議論できる時間かわずか２日たらずしかなく、今後余裕のあるスケジュール策定が望まれる。

FSE-3 の後、１０月１３、１４日の２日間、University of Maryland（ワシントンから電車とタクシーで３０分ぐらい）でプログラム委員会が開催され、どの論文を採択するかが審議されていった。

評価がある点以上の論文について担当の代表者１人がその論文のよい点、弱い点を説明し、他の２人が補足する、あるいは反対意見を述べるという形式で審議が進められていった。採否は全員の合議制で決められていったが、意見がまとまらなかったものについては、新たにその場で査読者をボランティアベースで割り当て、最終的な判断はその査読者の報告を加味して行った。

従来の２０編程度しか採択しないＩＣＳＥと異なり、最初からかなりの数の論文を採択しようというプログラム委員長の意識が感じられた。その結果、５２編もの論文が採択されるという結果にいたった。採択された論文の質は、実際に会議での発表を聞くと、採択論文数を増やしたため、やはり玉石混合となってしまったという印象であった。

しかし、採択論文を従来のＩＣＳＥの２倍もとったことにより、参加者が増え、いっそう関心が高まってきたことは大きく評価できる。個人的には、参加者を増やし、活発な交流の場とするためにも、この方向を維持していくことがよいと思われる。

7 会議の運営について

会場は、ベルリン工科大学のキャンパスの一角で、８００名収容のメインホール、２００人以上は入れそうな傾斜のきつい階段教室を５－６つ使った。

大学なので、通常の学生が、会場内をうろうろするので、あまり落ち着かなかったが、しかし、食堂が幾つもあり、インターネットアクセス用の計算機も大学の演習室のものが使え、便利であった。

レジストレーションが、８５２人と、近年になく大き

な数字なったのは、以下のような理由であろう。

- いろいろなチュートリアル、ワークショップがあった。
- 採録論文数が５０ほど（今までは３０程度）に増えた。
- 宣伝が行き届いてた（特にヨーロッパに対しては）。
- 地元の会社にお願いして、寄付と同時に動員をかけてもらった。
- 東欧の人に旅費を出して来てもらっている（旅費は企業からの寄付金）。

最初の会式の挨拶は、一部ＡＴＭのネットを使って、ボンにあるGMDから行なわれた。

また、毎日朝、昨日の内容を反映するような６－８ページの新聞を発行していた（昨年の会議から）。今回は、かなり過激な内容になっていて、昨日の基調講演に対し、否定的なコメントを載せるなどしていた。また、WWW で発行した内容をそのまま掲示し、会議に参加していない人も見れるようになっていた。

8 むすび

本稿では、第 18 回ソフトウェア工学国際会議の内容を概説した。

次回は、 Boston で 1997 年 5 月 18–23 日に開かれる。詳細は、

http://www.ics.uci.edu/icse97/cfp.html

から得ることができる。

1998 年の第 20 回は、４月１９日から２６日まで、京都の国立京都国際会館で開催されいることが決まっている。今回、その会議に向けて、いろいろな意見をもらった。例えば、基調講演には、日本に特有なもの（ゲームソフト開発とか、ソフトウェアの日本語化問題など）について聞きたい、新しいソフトウェア作成動向に対応した内容にしてもらいたい、今までの ICSE の威信や名声を保って欲しい、安い宿を提供して欲しい、など、非常に多岐に渡っている。今後、それらの要求にこたえるべく努力していく必要がある。

なお、本会議のプロシーディングスは、 IEEE Computer Society Press より出版されている。

IEEE Computer Society Press Order Number PR07246
ISBN 0-8186-7246-3
ISSN 0270-5257

謝　辞

論文の要約に御協力頂いた大阪大学基礎工学部大学院の松下、高田、神谷、大下各氏に感謝致します。