SEAMAIL

Newsletter from Software Engineers Association

Volume 5, Number 10-11-12 1990

目　次

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．

SEAMAIL Vol.5, No.10, 11, 12　平成3年1月25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　1500円　（禁転載）

事務局から

☆

あけましておめでとうございます．

☆☆

SEA もようやく満５歳にまりました．まだまだひよわな幼稚園児ですが，会員のみなさんの一層のボランティア努力を仰いで，少しずつ体力をつけて行きたいと考えております．どうぞ，よろしくお願いいたします．
☆☆☆

さて，本来なら昨年末に発行できるはずだった SEAMAIL Vol.5 の最終号をお届けします．

☆☆☆☆

今回は，一昨年秋に京都で行われた国際シンポジウムのポスト・プロシーディングということで，全ページ英語になりました！

☆☆☆☆☆

お知らせ（１）： 昨年の暮れに SEA の Unix Workstation がようやく junet につながりました．とりあえず，会員のみなさんとの連絡用の e-mail アドレスは：

sea@sea.or.jp

となっています．どんどん mail をください．SEAMAIL への原稿もぜひ（！）．いずれ SIGNET の方々の手で電子掲示板なども整備されると思います．

☆☆☆☆☆☆

お知らせ（２）： SEA では，平均すると毎月２回ずつの割合で，会員への郵便を出しています．会員の中で，これを利用してセミナー／イベントの開催案内やプロダクトの宣伝 DM をしたいという方は御遠慮なく事務局へお申し出ください．幹事会の承認を得た上で，お引き受けします．料金は切手代＋手数料（５０円／１通）です．古い会員の方は御存じのことですが，最近新しい会員も増えたので，あらためてお知らせしました．

☆☆☆☆☆☆☆

SEA 1991 年の主要イベント
（いまわかっている分だけ）

1/7	SEA Forum「新春放談会」	東京：事務局会議室（終わりました）
1/31-2/1	第 3 回 ソフトウェア プロセス ワークショップ	静岡：伊東市
2/5	ソフトウェア信頼性フォーラム	宮城：仙台農協会館（参加者募集中）
2/8	SEA Forum「これからの中堅技術者教育を考える」	東京：機械振興会館（参加者募集中）
3/14-15	SEA 春のセミナー ウィーク '91	東京：青年会議所会館（企画中）
3/末	第3回　テクニカル マネジメント ワークショップ	たぶん北海道（企画中）
5/中旬	13th ICSE 研修ツアー	アメリカ：Austin, Texas（企画中）
5/30	1991 年度総会	東京：機械振興会館（企画中）
6/10-12	ソフトウェア シンポジウム '91 および併設チュートリアル	愛知：名古屋国際会議場（論文募集中）
6/13-14	3rd International Symposium on Future Software Environment	滋賀：彦根プリンス・ホテル（企画中）
9/上旬	第 9 回 夏のプログラミング ワークショップ（若手の会）	たぶん盛岡
9/中旬	SEA 秋のセミナー ウィーク '91	東京
10/中旬	1st ICSP 研修ツアー	アメリカ：Los Angels
10/?	International CASE Workshop	中国：北京
11/初	第 12 回 ソフトウェア信頼性シンポジウム	大阪：阪大Σホール
11/20-22	第 7 回 実践的開発環境ワークショップ	宮城：仙台
?	第 5 回 教育ワークショップ	場所未定

[1] 東京でのイベントを中心にリストアップしました.

[2] 3 月以降の SEA 月例 Forum は未定です.

Proceedings of the 1st International Symposium on Future Software Environment

Kyoto Research Park
Kyoto, Japan
November 13 – 14, 1989

Kouichi Kishida, Editor

Sponsored by

Software Engineers Association

In Cooperation with:

SDA Consortium
Kyoto Reseearch Park

Preface

This is the post-conference proceedings of the 1st International Symposium on Software Environment held in Kyoto Research Park (Kyoto, Japan), November 13-14, 1989.

The idea for the symposium has benn emerged through technical discusions at SDA Consortium, which is a unique international research project to develop common framework for future software design support environment. The symposium program was organized as a series of four panels, and at each panel various aspects of futute software environments were discussed from four different viewpoints: namely, user's view, builder's view, manager's view and reaetcher's view.

This post-mortem proceedings was edited by transcribing and summarizing audio-tape record of presentations and discussions of each session. I'd like to thank deeply to Ms. Erin Karp for her excellent work of compiling the report.

Also, special thanks to the secretarial staffs consisted of volunteers of SEA and SDAC for thier contribution to the success of the symposium.

For your information, The 2nd ISFSE was held in Boulder (Colorado, USA), August, 1990. And the 3rd one will be held in Hikone (Japan), June 13-14, 1991, just after SEA's Annual Software Symposium'91 (June 10-12 at Nagoya).

Kouichi Kishida
Symposium Organizer and Proceedings Editor
Software Research Associates, Inc.

1st International Symposium on Future Software Environment

Final Program

(Chairpersons and Panelists)

November 13th (Mon)

Registration (9:00 - 10:00)

Panel 1 (10:00 - 13:00)

User's Perspective: "What will it look like?"

Chair: Akira Kumagai (Fujitsu & PFU)

Panelists: David Barstow (Schlumberger-Doll Research)
Gerhard Fischer (University of Colorado)
Koichiro Ochimizu (Sizuoka University)
Tetsuo Tamai (University of Tsukuba)

Lunch Break (13:00 - 15:00)

Panel 2 (15:00 - 18:00)

Builder's Perspective: "What capabilities will it provide, and how?"

Chair: Masatoshi Matsuo (SRA)

Panelists: Laszlo Belady (MCC)
Isao Miyamoto (University of Hawaii)
William Riddle (software design & analysis)
Yoichi Shinoda (Tokyo Institute of Technology)

Reception (18:00 - 20:00)

November 14th (Tue)

Panel 3 (9:30 - 12:30)

Manager's Perspective: "How will it change the process or project management?"

Chair: John Sayler (University of Michigan)

Panelists: Robert Balzer (USC/ISI)
Mark Dowson (software design and analysis)
Masanori Teramoto (NEC)
Koji Torii (Osaka University)

Lunch Break (12:20 - 14:30)

Panel 4 (14:30 - 17:30)

Researcer's Perspective: "What will it take to achieve it?"

Chair: Nobuo Saito (Keio University)

Panelists: Mark Dowson (software design & analysis)
Takuya Katayama (Tokyo Institute of Technology)
Kouichi Kishida (SRA)
Lloyd Williams (Software Engineering Research)

Roster of Attendees

Name	Affiliation	Inerested in
Satoru Fujii	Matsushita Communication Industry	Management
Terunobu Fujino	Fuji Xerox Information System	Development
Taku Fujioka	Mitsubishi Electric	Research
Takeshi Hamano	SRA	Development
Chifumi Hayashi	SRA	Use
Kaoru Hayashi	SRA	Development
Atsushi Hirata	SRA	Use
Makoto Ikeda	Nihon Network Lab	Management
Katsuro Inoue	Osaka University	Research
Makoto Ishisone	SRA	Development
Yasuhito Kato	PFU	Development
Yusuke Katsuda	Nihon Unysis	Management
Tohru Kikuno	Osaka University	Management
Ayako Kobayashi	SRA	Use
Masato Kondo	Toden Software	Management
Sadatoshi Koshihara	Fujitsu FIP	Management
Eisuke Koshio	Hitachi Software Engineering	Development
Masatoshi Kurihara	SRA	Development
Asako Miura	SRA	Development
Tatsuo Miyagi	NEC Software, Chubu	Use
Naoyuki Motomura	Yasukawa Electric	Development
Osamu Nagamori	INES	Use
Masahiro Nakata	Hitachi	Management
Fumio Nitta	KDD	Research
Toshitsugu Nomura	Japan Information Processing Service	Management
Yasuo Ogasawara	CSK	Development
Toshiharu Ohno	ASCII	Development
Takakazu Okamoto	KCS	Development
Takashi Owaki	Hitachi	Development
Toshihide Sakai	Horiba	Management
Ken-ichi Shima	ATR Communications Systems Research Lab	Research
Hiroshi Sakoh	SRA	Use
Ken-ichi Satoh	SRA	Research
Yoshiaki Sugita	SRA	Management
Norihiko Suhara	Mitsubishi Electric	Research
Kenzo Suzuki	Fujitsu FIP	Management
Toyofumi Takenaka	ATR Communications Systems Research Lab	Research
Takuji Takeuchi	Tateishi Electric	Use
Atsushi Yamaguchi	SRA	Use
Keiichi Yamaguchi	SRA	Management
Takeshi Yamaoka	Keio University	Research

Session 1

User's Perspective
(What will it look like?)

Chair: Akira Kumagai
(PFU & Fujitsu)

Speakers: Tetsuo Tamai
(University of Tsukuba)

David Barstow
(Schlumberger Laboratory for Computer Science)

Koichiro Ochimizu
(Shizuoka University)

Gerhard Fischer
(University of Colorado)

Tetsuo Tamai
(University of Tsukuba)

Professor Tamai started off by mentioning a book, edited by Professors Matsumoto and Ohno, called "Japanese Perspectives in Software Engineering," which is published by Addison-Wesley. This book includes a paper by Professor Tamai entitled "Knowledge Engineering Applications to Software Development." He commented that this book is one of the more accessible books for non-Japanese on software engineering in Japan, as it is not written in Japanese.

Professor Tamai then went on to give his presentation on a Viewpoint of Knowledge Based Systems Engineers. In this presentation, Professor Tamai introduced SOLOMON, a Solution Oriented Systems Design Methodology based on Knowledge, which was developed last year by Mitsubishi Electric and Mitsubishi Research Institute. It is a methodology for problem solving and system design, targeted for systems engineers (SE's) and knowledge engineers (KE's).

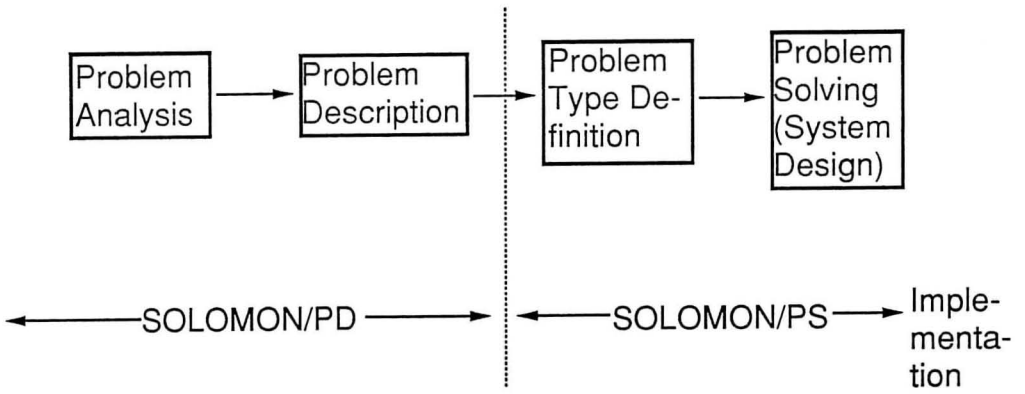
Professor Tamai said that KE training has traditionally emphasized knowledge acquisition and representation, expert system (ES) tools, and artificial intelligence (AI) programming languages. However, the training has not answered the following:

- what is a problem

- how to solve it

- how to design

He suggested that a KE should be a good SE in the first place. ES development has been motivated by a feeling of "Let's try AI," rather than one of "Let's solve this problem." Many KE's have a rather poor knowledge of mathematical approaches, such as operations research (OR) and statistics and system engineering methods. For example, while building an ES for scheduling, they "rediscover" existing scheduling techniques.



Scope of SOLOMON

Professor Tamai proposed that the knowledge based approach is just one of a wide variety of useful solution techniques. A good SE needs to make use of the good techniques available. The scope of SOLOMON ranges from problem analysis and definition to systems design. There are two phases: SOLOMON/PD (problem definition), and SOLOMON/PS (problem solving). Implementation is not within the scope of SOLOMON. He identified a number of problem types, categorized as analysis types, synthesis types, guidance types, and data processing types.

In SOLOMON/PD, one uses a general procedure description and work sheets, as well as related techniques such as interviewing, brain-storming, KJ/KT methods, and ISM/DEMATEL. One also uses general procedure descriptions, which are divided into 4 categories and 13 types as

mentioned above. KE is one of many techniques used.

Professor Tamai said that 10 case studies had been done. Two-week seminars had been held in the spring and fall of 1989, attended by about 8 people each. The participants were eager, and found the seminar helpful. In February 1989, a 500-page SOLOMON manual was completed.

One question remains: Is it beneficial to have SOLOMON computer aided? Professor Tamai is not so sure, and would like some suggestions.

Questions/Answers

Audience:
Could you give some examples of KJ methods?

Tamai:
That is one of the techniques that I mentioned. If the problem fits the technique, we recommend it. KE's are not so familiar with techniques, so we guide them. We provide guidelines for the relation between problems and techniques.

Audience
Who decides on the technique?

Tamai:
We help them decide.

Belady:
Who are the participants?

Tamai:
Mitsubishi Electric and Mitsubishi Research Institute.

Balzer:
How do you merge the techniques?

Tamai:
First, we define the problem, then divide it into smaller ones. The smaller problems may fit 1, 2, or 3 techniques.

Barstow:
You didn't seem to talk about real-time problems.

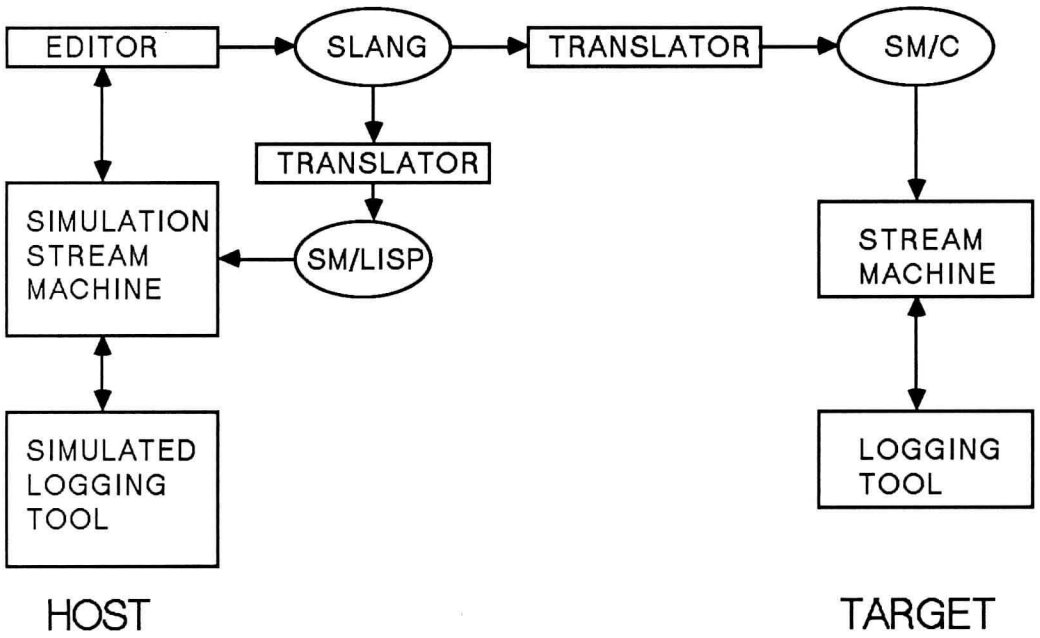
Tamai:
Well, we can find techniques to fit any type of problem.

Kishida:
You said that the question that remains is whether or not it is beneficial to have SOLOMON computer aided. What would be the category of that problem?

Tamai:
That's a key step in SOLOMON.

David Barstow
(Schlumberger Laboratory for Computer Science)

Dr. Barstow's presentation focused on the SPHINX programming environment. Dr. Barstow began with the suggestion that in order to understand the knowledge engineering (KE) technology of the future, one would need to look at the current technology. He said he would give a description of Schlumberger-Doll's environment, where the work involves sending instruments down an oil well. The area of interest there is an example of device control.



SPHINX Programming Environment

The SPHINX programming environment is based on the host/target division, and is highly graphically oriented. There is a simulator on the host side, and a translator that translates towards the concurrent target.

On the target side there is a "Stream Machine," which is a model of distributed computation that involves concurrent processes, and communication via streams in which there is a single writer and multiple readers. A stream is defined as a flow of temporally ordered values. The basic model is equivalent to fine-grained data flow, making it deterministic. The extended model includes time-based constraints, and is not deterministic.

The SPHINX architecture's main user component is an editor, through which the user can access a simulated stream machine to control a simulated logging tool. The tool gives feedback which the user then uses to decide on further editor actions, thereby closing a cycle similar to the edit-compile-test cycle. After testing, the program is translated from the so-called SLANG to SM/C, a kind of C for the target stream machine.

The SPHINX editor contains a topology editor and a process editor, which are both graphically oriented. The topology editor allows graphical display and manipulation. The process editor is a structure editor for C that supports multiple views on the C program.

The SPHINX simulator is a stream machine interpreter with a variable speed clock that is advanced with each machine instruction. Also, there is an event queue for external events, as well as a simulated environment and logging tools.

The SPHINX display is dynamically updated and is integrated with the editor. It displays values on the streams and the states of processes (active, ready to run, waiting for data).

An experiment was carried out for one logging tool, and this viability test showed that SPHINX is quite effective. Many bugs were discovered simply by watching the display. The ability to simulate the tool gave greater knowledge about tool control. In the end, more than half of the code reflected the topology of the system.

Dr. Barstow also talked about other CASE tools for design diagrams and notations, integrated data bases, and code generators. He mentioned domain-specific spread sheets with dynamic displays, that offer a natural interaction. Other trends he sees are increased use of graphics and dynamic displays, reliance on simulation and prototyping, integration through project data bases, and special purpose and domain-specific environments.

Dr. Barstow concluded by giving his predictions for the future, as shown in the following chart:

	Now	Near-Term 0-9 years	Far-Term 10-99 years
Domain expert	Spread sheets	Domain-specific notations	Domain-oriented dialogue
Programmer	SPHINX	Assistance with decisions	Automation
System designer	CASE	Static analysis	Dynamic analysis & simulation

In particular, Dr. Barstow foresees automation for programming "in the small," and simulation on a larger scale.

Questions/Answers

Balzer:

If there is going to be increased automation, do you think that programmers will be around for a while? And if expertise is going to be built into systems, do we still need domain experts?

Barstow:

Programmers will become less and less important, but domain experts will still be around, as they are needed when people have questions and so on.

Dowson:

This seems to focus entirely on development in the small, rather than issues of development in the large. Is that correct?

Barstow:

That was not the intent. I guess that I was hoping that this was beginning to address programming in the large.

Dowson:

By "development in the large" I guess I'm thinking of developments where you are forced by the size of the problem to have large teams, even of system designers. That seemed to be support for the system design process by the individual system designer, which I accept is extremely valuable. But it doesn't seem to be addressing the problems of the co-ordination of the activities of large numbers of people. Unfortunately, we're still facing these.

Barstow:

I agree.

Belady:

Combined activity between the domain expert and system designer is always through the shared database, and that's all?

Barstow:

I would hope they would talk to each other, too. That goes back to what Mark [Dowson]

was asking about. What sort of support might we see for communication and collaboration?

Belady:

What about having the project database external to the machine?

Barstow:

It will be far term before there is anything like that.

Koichiro Ochimizu
(Shizuoka University)

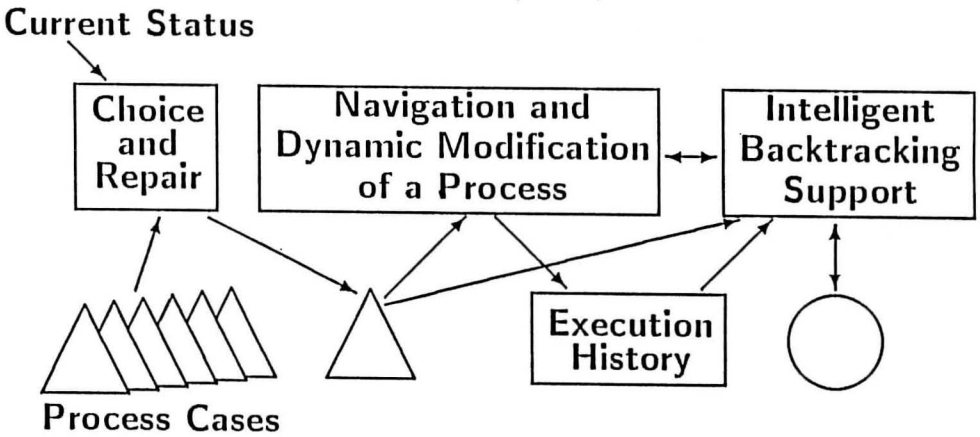
Professor Ochimizu gave a presentation dealing with the question, "How can we incorporate intelligence into future software environments?" His goal was to simply express his expectations for a future software environment, specifically with regards to integrated environments based on the process model, and how intelligence can be incorporated into such environments. He started by discussing a list of examples of users' requirements for a future software environment. He pointed out that the 10 items listed were only examples, not overall requirements.

Professor Ochimizu then examined the question of where intelligence is needed. He discussed the question with several expert designers in the Japanese software industry. Their answer was that they personally felt capable of meeting the requirements previously stated, but that most members of their design teams could not. Therefore, these experts expressed a desire to somehow embed their knowledge into a software environment so as to raise the technical level of their teams. Professor Ochimizu examined several ways in which this might be done.

The first possibility that he examined involves task scheduling for cost-estimation and role assignment. This kind of work, he said, involves the extraction and sequencing of task templates related to the current project features from a project-case database, which would then allow for task scheduling, emphasizing effective role-assignment and cost-estimation.

The second possibility involves project monitoring and deliberation. Here, it is considered desirable for a manager to monitor the natural growth of a cause-effect network over the course of the project, developing an understanding of what is going on. Then the designers can examine constraints that have arisen during this network as a prelude to starting their tasks.

Thirdly, process navigation and dynamic modification were discussed. Professor Ochimizu feels that it is necessary to have some kind of navigation capabilities which can guide the novice designer in answering such questions as "where are we now," "what should be done next," and "how can we deal with this exceptional case?" However, it was noted that since processes from past projects do not always exactly fit the current project, it is necessary to be able to modify the process dynamically.

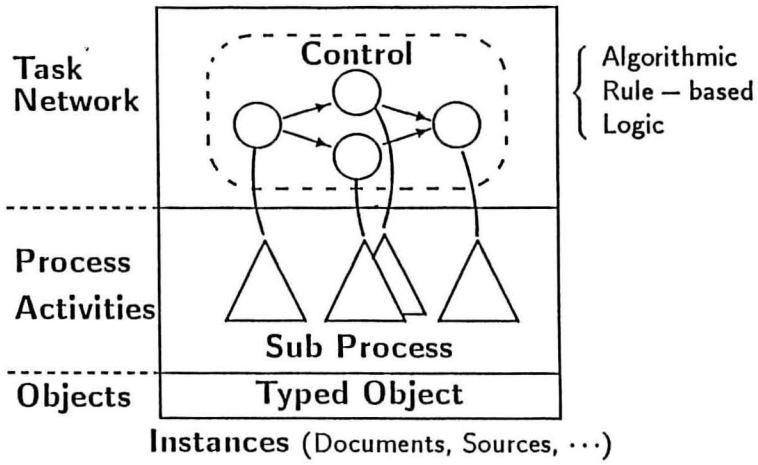


Process Navigation and Dynamic Modification

The final possibility mentioned was that of providing views for analyzing and evaluating products. By creating and analyzing typed objects called "firm information" (such as artifacts/products which are standardized within a project/organization), designers would be able to produce a lot of information based on their own mental processes. This information is called "soft information." Professor Ochimizu feels that "soft information" would be useful in understanding "firm information" if it were possible for designers to record the "soft information" and to dynamically answer questions posed during specific situations.

Professor Ochimizu stated that considering the requirements that were previously mentioned, we need a process model which can deal with these requirements naturally. After a

general description of a software development process, he stated that a large quantity of documents are produced during software development, and that these documents can be classified into several categories. He feels that by using typed objects to redefine or rewrite a developer's way of doing development, it would be possible to formally describe the software process. At present, such processes as can be obtained are fragmental, and there seems to be no way in which to get a process description. This means that a control mechanism to invoke subprocesses is necessary. The presentation ended with a brief look at such a control structure.



Process Model

Questions/Answers

Belady:

Your presentation is about how we can incorporate intelligence into future software environments. What kind of intelligence do you mean? Is it machine-stored, or just general? Also, execution history was mentioned in one of those charts. It's a little ambiguous, because sometimes "execution" means program execution. Do you mean the history of the program being developed or the history of the process?

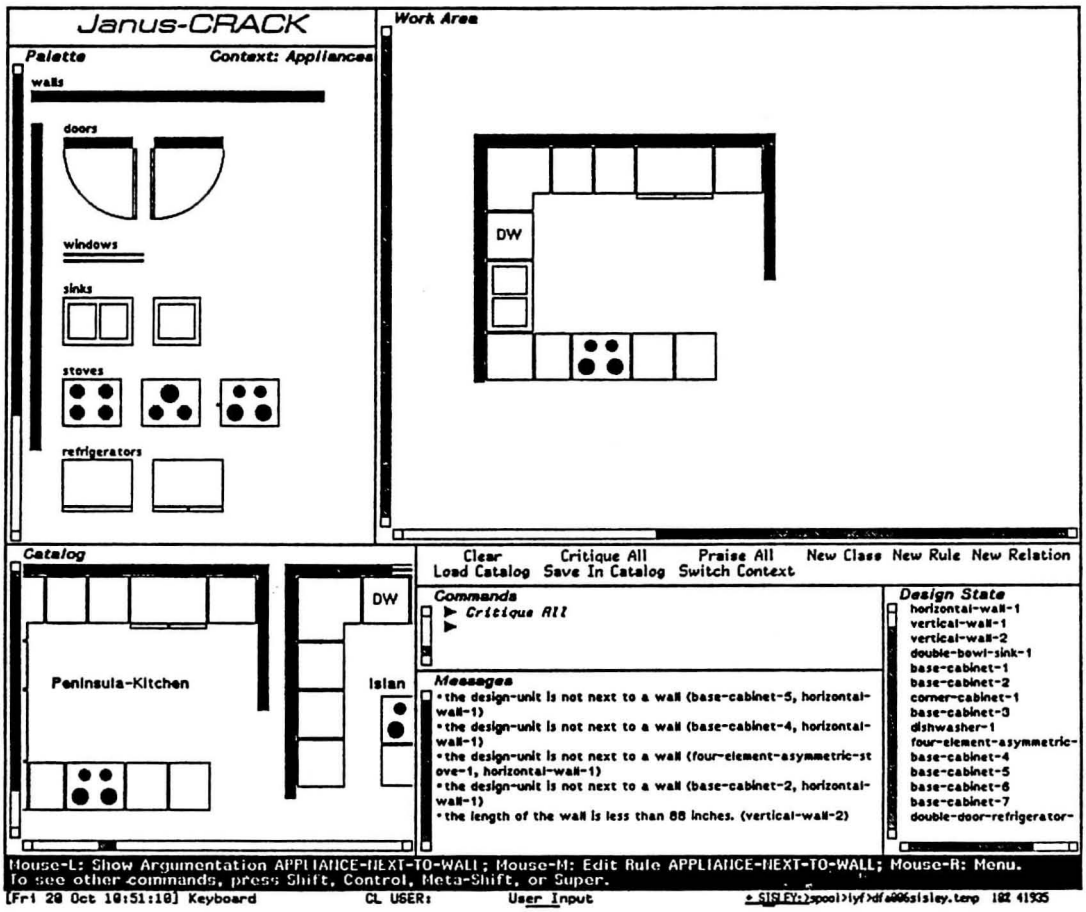
Ochimizu:

Execution is the process execution. As for intelligence, I think we can use some of the results from knowledge engineering or natural language, reasoning by analogy to extract some patterns of tasks and for extracting the process related to the features of current work. This is very doubtful, and I think it can't be achieved easily. I think that such kind of intelligence can be embedded into the development environment, and that intelligence will hepp the manager or designer to grasp or summarize something. So we work using the intelligent software development environment. Execution history is just a log of the processes.

Gerhard Fischer
(University of Colorado)

Professor Fischer gave a presentation entitled "Domain-Oriented Knowledge-Based Design Environments or Making Computers Invisible by Getting out of the Turing Tar Pit." In one of his first slides, Professor Fischer characterized current software environments as either a Turing Tar Pit, in which everything is possible but nothing of interest is easy, or the converse of a Turing Tar Pit, where operations are easy, but little of interest is possible. He said that the goal is to find out how to do something in the middle, between these two extremes.

His approach is to have a system that is somewhere in between manual and automatic, by automating certain parts. He wants to come up with cooperative problem solving systems, where human-computer communication is important.



JANUS' Construction Component

He intentionally wants to move away from general purpose programming environments such as Lisp, Pascal, C and Ada, to domain-oriented construction kits with sets of reusable objects.

In the past, many people mapped problems to programming languages or even to assembly languages. In the 1980's there were attempts to make the distance between problems and languages smaller, by providing construction kits. For the 1990's, he predicts situations where knowledge engineers build design environments using programming languages, and domain experts use these design environments to solve the particular problems at hand, thereby reducing the transformation distance to that between the problems and the design environments.

As an example Professor Fischer introduced JANUS, a knowledge-based design environment for the architectural design of kitchens. An artifact can be designed quickly, but that artifact is not necessarily good. The system responds to first attempts by producing messages from critics. For example, the stove critic might complain that the stove is next to the sink, and is therefore a dangerous design. This feedback then allows the user to improve the design. The user can look up the argumentation for a particular criticism through hypertext mechanisms, allowing for a dialogue about the stove and sink. Fischer has been in contact with architects in Denver, who say that this system provides a good model. In this system, the critic is important because it provides a link between the construction situation and the argumentation.

It is necessary for the end-user to be able to modify the system, because the world is constantly changing and because users have different preferences. For example, a user might want to introduce a microwave into the picture. It is desirable to make changes at a high level, i.e. in the kits. In JANUS, it is possible to introduce a new type of appliance by using existing objects, subclassing a particular superclass. For example, one might use the stove to define a new rule.

JANUS has a layered architecture, with Lisp at the bottom and the kitchen problem domain at the top. The high levels are supposed to make the lower levels invisible. Professor Fischer went on to talk about FRAMER, a knowledge-based design environment for window-based user interfaces. Framer allows the user to design interfaces using panes, scrollbars, and so on that have to do with the knowledge and rules of the system. Framer sits on top of a construction kit, such as the NeXT interface builder.

Professor Fischer concluded by saying that his approach addresses various problems, such as those experienced at MCC relating to the thin spread of application knowledge. It also attempts to tackle the problem of co-evolution of specifications and implementations, which was brought up by Bob Walter. As a final conclusion, Professor Fischer said that he feels that software engineering is the wrong term. There is too much emphasis on the medium. By way of contrast, he said that people do refer to civil, electrical, mechanical and architectural engineering, but not to wood, plastic, concrete and steel engineering.

Questions/Answers

Audience:

How can we incorporate knowledge into computer systems? Can we use natural languages, and would that be easy?

Fischer:

No, we should stay away from natural languages. We can obtain knowledge by talking to domain experts and by reading books. Taking the microwave as an example, we want to include knowledge about it at a domain-specific level, otherwise we have to descend to lower levels, changing the programs and so on.

Balzer:

Can an expert annotate the critics and the kits, or is the system read-only?

Fischer:

No, you cannot annotate. If you delete a critic, you need to reflect that in the argumentation space.

Balzer:

Is it easy to delete a critic? By that I mean, is it procedurally encapsulating?

Fischer:

For example, if you don't want the dish-washer, you need to recompute the argumentation space.

Barstow:

What about the work of Colin Potts at MCC , with arguments and all that?

Fischer:

Well, for every argument you have a counter-argument, and so on, to get a new position.

Discussion

This discussion session focused on several topics in sequence. First, the panelists and audience discussed how to define the word "user." Several people gave their opinions, which led into a discussion on the need to distinguish between experts and novices. The next focus of the discussion was on problem solving. It was pointed out that none of the panelists had dealt with this issue, and a number of theories and concepts related to problem solving were discussed. Finally, the question of domains was brought up. Specifically, how to define the borders of a domain.

Torii: In your slide about near and far term predictions you only mentioned CASE tools for the system designers of today. Aren't there other things?

Barstow:
Well, if there are others, I would like to know about them.

Torii: What about the design phase?

Barstow:
I foresee static analysis in the near future.

Balzer:
Do you mean static analysis of the code?

Barstow:
No.

Audience:
According to the title, this session is supposed to be about the user's perspective, but it doesn't seem like that was talked about. What is the user's perspective?

Tamai:
I talked about the methodologies in knowledge engineering systems, and described the activities and guidelines that we provide for the procedures. I don't see the motivation for automating methodologies.

Barstow:
My table about future predictions shows the requirements for different users. Especially dynamic analysis will be important, I think. And systems will become increasingly domain-specific.

Fischer:
What is a user? Is it a domain expert? Or a systems engineer? These are also questions we have to ask ourselves. I believe we need to make the underlying levels invisible. We need to move away from general purpose environments, to domain specific ones. But we also need to improve the lower levels, which requires knowledge about computers. Domain abstractions are representations of knowledge.

Ochimizu:
I think we can say that users are designers or managers. But we should also realize the distinction between experts and novices. In my presentation, I implicitly showed the user's perspective.

Kishida:
We are really talking about communication between us and the environment, at a variety of levels. But hopefully, the environment will only have to communicate with humans. Sometimes it is frustrating to find that the environment does not learn. Expert users feel the machine is cheating them after about 1 hour of use. We need learning capabilities built into the system.

Fischer:
Yes, I agree we need to accommodate the transition from beginner to expert. We need adaptable systems, systems whose behavior we can change. But we also need adaptive systems, that are smart enough to change by themselves. We shouldn't assume that the user stays at the same level. For example, if you know that I know UNIX, your explanation of

something will take that into account.

Barstow:
What time frame do you see for these things?

Fischer:
Well, that's difficult to predict, but if you take GNU emacs, for example, it behaves differently towards experts. But this is a research issue, definitely. I postulate that the 90's will be the decade of the experts.

Kishida:
Do you foresee that the 21st century will bring adaptive systems?

Fischer:
Well, modifications can come from both sides, really. For example, let's think about init files used in many UNIX systems today. The user can adjust them to suit his/her needs, but the system can also update them. But we cannot just tag on this kind of functionality - we need to reconsider the systems.

Tamai:
You can think of various types of people, for example, a teacher type, and an apprentice or assistant type of person. Also, I sometimes wish to have a clone of myself, so that I can discuss things with the clone. We want to discuss preferences and ways of handling personalities. This is hard to realize, however. But Ochimizu said earlier that he would realize this.

Barstow:
What we really need is a simple init file. And we also want others to have the same init file, so that we aren't confused by different bindings and preferences and so on. We need consistency within a group of people.

Dowson:
One example that I can think of when hearing you talk about moving from one environment to another, is that "rm *" might be safe on other systems, but it isn't in UNIX. Today, we have listened to various talks, ranging from Tamai's global one to Barstow's problem specific one. However, we have not heard a theory of problem solving. Perhaps the panelists have such a theory. Can we proceed without a theory? And what would such a theory look like? Would it be part of computer science, or would it be psychological?

Fischer:
Various people have influenced our way of thinking about problem solving, such as Newman and Simon, and also Polya. We say that my pencil is cleverer than I, but we really mean the combination of the pencil and I. The computer is a powerful pencil.

Tamai:
I don't believe that there is any unique theory. We have several theories, and we need to select an appropriate theory or technique. And we should notice the similar structure of theories in different areas. We need to abstract the level of the theory.

Dowson:
Yes, I agree about the structural similarities. It is advantageous to have more choice, so that we can have a reason behind the theory.

Fischer:
General purpose programming environments don't scale to large problems. Domain specific solutions are more important.

Balzer:
In researching learning techniques, we now go back to first principles. Take Sewer's DEO as an example. In this case, we attack by chunking, in other cases by caching. The power is in the knowledge. We need to have a general learning capability to augment the systems.

Barstow:
I don't think this is inconsistent.

Fischer:
There were various phases of AI: the power phase and the knowledge phase. We encode knowledge into rules. Expert systems are brittle outside their areas. So maybe we should look for something in the middle, with weaker methods.

Ochimizu:

I do not have any expectations for general solving. We need to store rules. We need to make it so that we can reuse some of that, through an intelligent search. But I do not have a dream in this area.

Audience:

My first question is, is it possible to define the borders of domains? And if so, what theory can the user apply to make the definition? My second question is about Professor Schaefer's research on capabilities. I'd like to isolate the domain, so that we lean more towards the personal side of things. What I mean is to isolate individual knowledge as domain knowledge.

Barstow:

Well, there are several domain models. The first is the generally accepted domain, for example, physics in oil wells. The second would be a subset, for example, the electronics part of physics. The third is where we consider the performance analysis of the software itself to be the domain.

Belady:

We're always talking about domain knowledge, but we should really be talking about system knowledge. I would like to defend the panel. I think they have been presenting the user's perspective quite well. AI provides solutions sometimes, other times it does not. We need to realize that we have a particular problem to solve. That is what the user's perspective is all about. The advantage of computer aided systems is that they keep us from drifting away too much to individualized systems. The tools keep it on track. Also, can we distinguish simulation and prototyping? Real engineers use prototyping, but discover it's not necessary because it simulates what we're trying to achieve. I'm a bit confused about this. Is design a subset of problem solving, or is it the other way around? I recommend that all of you see Poya's videotape in which he teaches students how to create a mathematical formula from scratch. The process is very fascinating.

Barstow:

I'd like to make a comment about the role of AI. If you take a typical system and count the lines of code, you'll find that about 15% is AI, and the other 85% is for the user interface, system code, networking, and so on. AI is just one of the tools. And about simulating and prototyping: they're really very similar words. I realized that while I was writing the slide.

Tamai:

I think a good point was made about there being less scatter by having computer aided systems. Methodologies aren't really for general purpose programming environments.

Fischer:

I think the panelists tried to talk about what we should focus on. But we should be careful that we do not take these things to an extreme. For example, the early teletypes were very slow, and so we tried to improve the printing speed, but now the printers are too fast for humans to read, and they produce too much. Problems are usually not understood well enough. And there is no unique best solution. As an example, we can take the wicket problem.

Session 2

Builder's Perspective

(What capabilities will it provide, and how?)

Chair: M. Matsuo
(Software Research Associates)

Speakers: L. Belady
(Microelectronics and Computer Technology Corp.)

I. Miyamoto
(University of Hawaii)

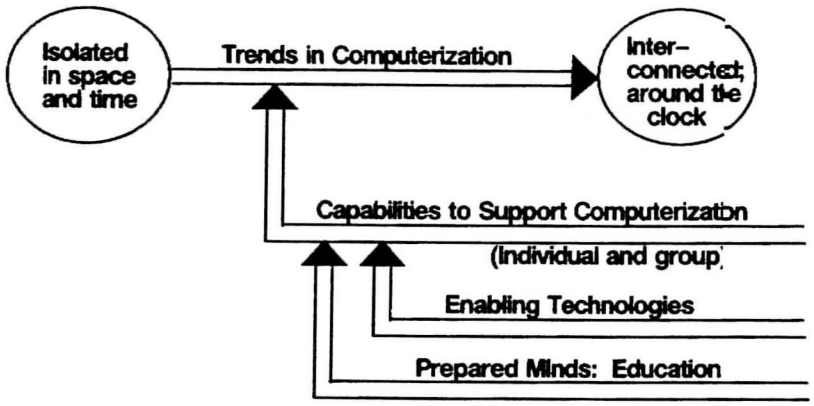
W. Riddle
(Software Design & Analysis)

Y. Shinoda
(Tokyo Institute of Technology)

Mr. Laszlo Belady

(Microelectronics and Computer Technology Corp.)

According to Mr. Belady's presentation, trends in computer applications back in the 1940s were for applications to be isolated in space and time. As time goes by, such trends are changing towards interconnection by hard wiring and around the clock applications. This changing trend, which increases integration of software that becomes more difficult to develop, is supported also by software. Further, to support software, people (both individual and group brains), enabling technology, and education must be prepared. In other words, to push computerization in society, we have to make it possible to build complex software more easily. In order to do that, we have to apply technology to build such an environment, as well as to educate people.



Chainging Trends

Mr. Belady then moved to a more detailed discussion of trends. The predominant trend has been such that applications have been integrated in terms of time and space over 40 years. Nowadays, data can be stored in the machine for longer periods of time, which means the end of isolation in time. Also, connection into networks allows applications to enlarge the available space. This trend has led to distributed/networked systems which require software to control traffic and to glue together such integrated system.

The next topic in this presentation was Type A and Type B software. Type A involves the manufacturing of pure software components, while Type B, which is called "glue software" involves general purpose applications. The purpose of Type B is to computerize the whole enterprise, or to develop software for interconnected networks of different enterprises. To customize Type B software, we have to be careful of the increased variety of domains, which increases complexity. Also engineers who develop Type B software must understand the application and team up with the "customer." Therefore, development can't be done "off-shore."

The necessary capabilities for individual system designers, according to Mr. Belady, are tools, a platform, and information handling/presentation facilities. Tool/Information spaces must be saved not only for programming, but also for system design. The environment must have an HW catalog/taxonomy, as well as a domain DB/KB and system simulation. All of these should be held in a platform. Information handling/presentation facilities should include such things as Hypermedia, in-machine transformation of informal to formal, scaled up, topology oriented graphics, and animation in support of system simulation.

Necessary capabilities for a group are communication, coordination, and training. Communication includes groupware, local/remote electronic meetings, and easy access to remote sources of information from the workstation. Coordination technology to exploit concurrency in the development process is important, but the best processor is human organization. Regarding training, one paradigm of software projects is mutual teaching. Mr Belady feels that this environment may be good for CAI.

Mr. Belady warned that we have to be careful about the enabling technologies which pop up in front of us, and said that we should grab them at once, in order to enhance productivity. In the hardware category there are things like MIPS, multimedia, large area displays for meetings, and bandwidth to transmit animation. As for the software, general/standard topology packages, animation languages, OO-DB, and scaling up of virtual displays are available.

Mr. Belady finished with a look at education. There are, he feels, two kinds of education: typical school education, and retraining to catch up with improvements in the field.

Questions/Answers

Saito:

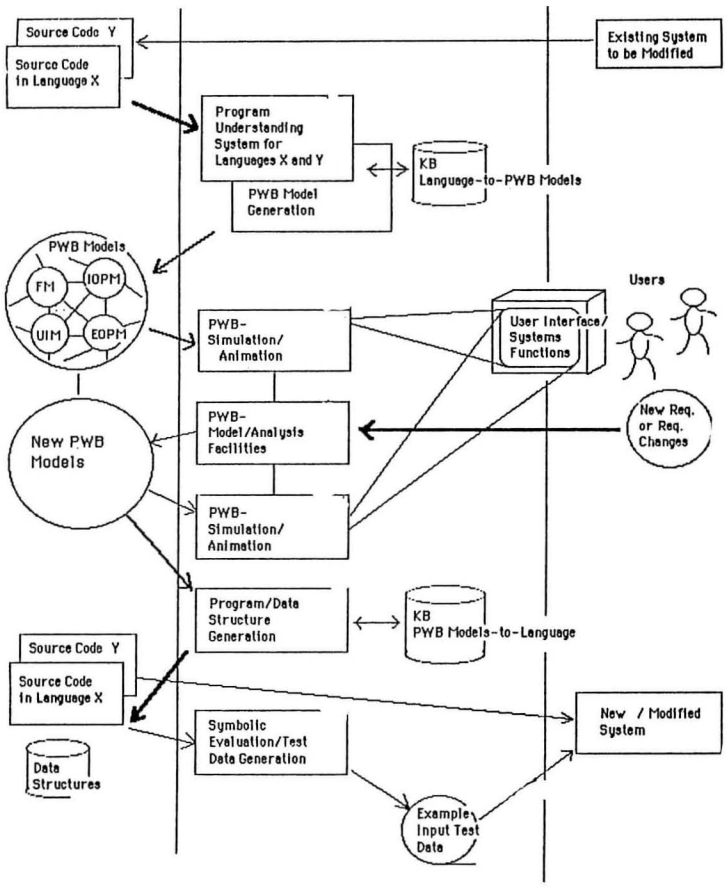
What do you mean by the last sentence of your OHP on education, where you say, "All the above represent Software opportunities themselves?"

Belady:

Teachers are too busy teaching new students to take care of retraining. Retraining should depend on CAI. That means that education itself can be a business.

Professor Isao Miyamoto
(University of Hawaii)

Professor Miyamoto gave a presentation on looking towards the future of software support environments. To start with, he wanted to give some personal observations about software support environments, and then introduce some ideas that he is, in co-operation with Bill Howden of the University of California, trying to carry out for tomorrow's software support environment. In his personal observations, Professor Miyamoto first listed several points about existing environments, and then suggested that for the future, along with several other requirements, software environments must support human essential activities for software development and maintenance to make use of 100% of computer potential. He then discussed a number of experiences that he has had in his work with software development environments.



Software Maintenance Assistant System

Professor Miyamoto then went on to point out a number of things that he finds to be important for future software environments, focusing on the software lifecycle model and support environment. Next, Professor Miyamoto gave an example of their Software Maintenance Assistant (SMA) system, which he plans to have finished by the end of next March.

He focused more closely on the user interface part of their environment, and then the components of the SMA system. Professor Miyamoto explained the natural language-oriented UI, called MOANA, which will be delivered with the University of Hawaii's specific SDA. Next, he showed a sample of the CPM graphics which MOANA can generate.

To conclude, Professor Miyamoto gave the opinion that more user-oriented support environments should exist. He does not think that process-model-oriented environments are the best possible idea, but rather should be implicitly kept inside the system.

Questions/Answers

Audience:

My question is about the balance of adaptability that the user is really able to decide. Because the more freely adaptable the system or user interface is, the more difficult it is to debug or trace certain adaptations that the user makes. So to define the balance, how much adaptability is produced for the user, and how much constraint is needed on them.

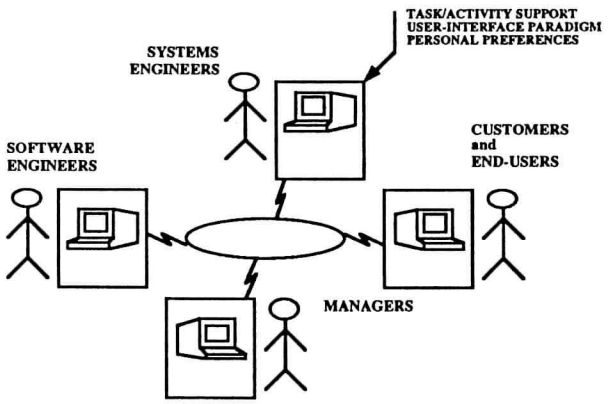
Miyamoto:

The first thing is that the object to be adapted may include many things; the user interface maybe, or objects, or tools, or sometimes the knowledge-base. So depending on the object, you must decide the balance or trade-off. But the important thing is the support environment. Adaptable support environments must be organized in this way: knowledge or model, and actors, programs. So components, or information about the adaptable thing shouldn't be imbedded in the program. So-called model-driven architecture is a must. I'm not answering your question, but if you achieve the architecture of the support system, and a mechanism to keep information about the component which must be adaptable in the environment, as a knowledge-base or some other thing, it is very, very flexible.

Dr. William Riddle
(software design & analysis)

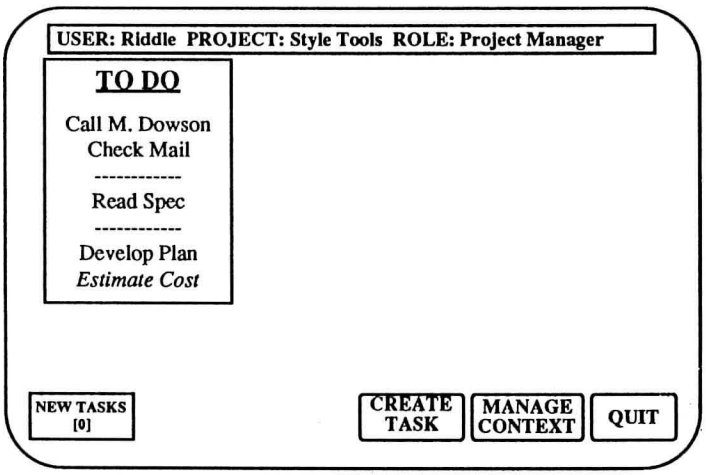
Dr. Riddle gave a presentation on support for flexible, disciplined software processes. The main topic of his presentation revolved around the questions of "what do we need to put into an environment to support processes,

Environments in the future should be based on specialized workstations which are used by domain experts such as systems engineers, software engineers, customers/end-users and managers. All of these are gathered into one generic environment, which would support the user interface paradigms that are natural to each person. One aspect of physical configuration is the networks of workstations.



A Distributed Software Development Environment is a network of workstations

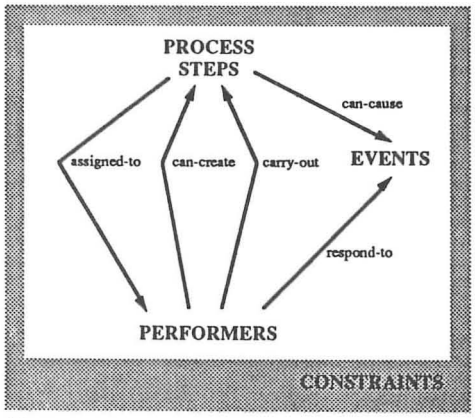
Processes that are supported provide a balance between discipline and chaos. The issue is to take that chaos, discipline it, focus it, channel it, and converge it so that the result can be produced on time within a budget. What Dr. Riddle wants to do is to find the balance between top-down constraints to enforce some discipline, and bottom-up cooperation among people sharing activities. Bottom-up cooperation is a collaborative problem-solving that is carried out in terms of people working in their own contexts, and mutual influences between these contexts. This is highly evolutionary. On the discipline side are found constraints which result from top-down thinking, the ability to confirm progress and make plans and assignments, and the idea of convergence over time.



A Possible User Interface

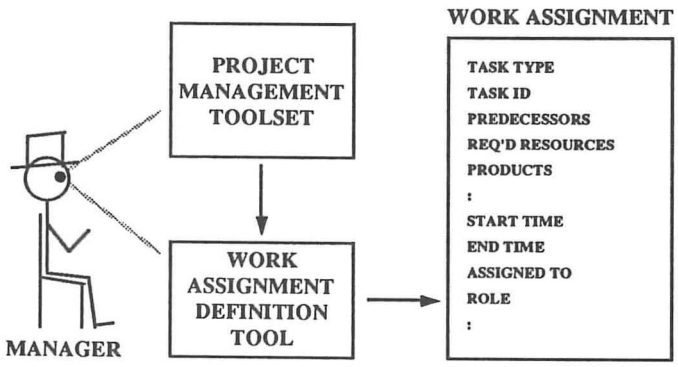
One possible user interface is to have the workstation screen display what context people are working on and indicate the types of activities. Meanwhile, the system provides information

to the environment that would support the current activity. For process management, the following supports are required. First, support for an information base, such as shared, persistent information and distributed databases, are required. The ability to view information in various ways outside of the information base is also required, as well as support for precise process management, such as consensus building and negotiation, constraint definition and enforcement, and task identification, assignment and monitoring.



Fundamental Process Management Conceps

Support can be provided by a process server, which is oriented towards the task of managing processes, and which is responsible for accepting process related information about managed and unmanaged operations as steps in the process of work to be done, information about the agents performing this work, about the constraints upon the process steps and agents, and about significant events. Then, the process server performs the requested actions based on this information, and handles events by notification, default responses or invocation of supplied handlers. The process server also keeps an inspectable history.

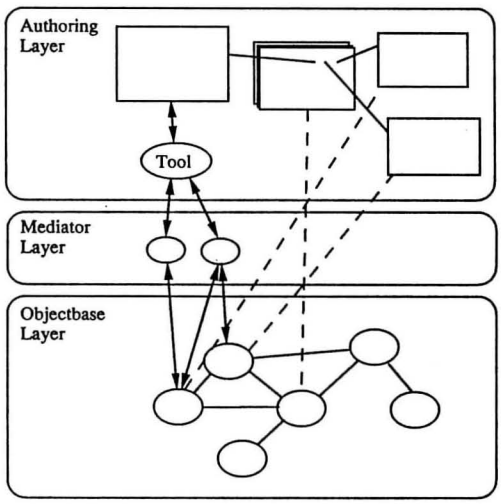


Process Server Interface

Finally, Dr. Riddle talked about the process server interface. One part of the interface is specifically defined data items called work assignments, which indicate the nature of the various tasks. The manager who develops the work assignments utilizes the already existing project management support tools, and uses them to develop work assignments.

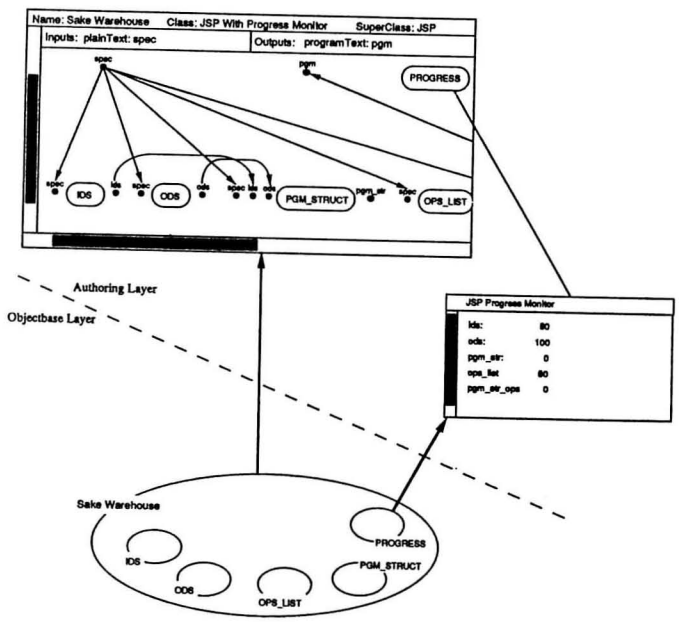
Dr. Yoichi Shinoda
(Tokyo Institute of Technology)

Dr. Shinoda gave a presentation on "Objectbase based Integration of Software Environments." His goal was to examine the more practical side of future software environments, and to look at what can be started today, not sometime in the distant future. The basic premise of this presentation was that the objectbase will be the integration base for future software environments.



What is Objectbase based?

First, Dr. Shinoda examined the question of "why objectbase based?" At this point he expressed the belief that we must start now to put software objects into an objectbase, not to wait any longer. Next, he moved on to a discussion of what is meant by "objectbase based."



Simple Example

In his structure, there are three layers. The bottom layer, the objectbase layer, is a shared database of software objects, a software object class library, tool objects, operators and several additional object capabilities. The central layer, the mediator layer, provides an interface for the objects in the objectbase layer to interact with conventional tools. Finally, the authoring layer is probably going to be a Hypertext-based browsing/authoring tool, similar to the interface builders provided by Next computers and the prototypes for Macintoshes and PWB.

After a look at an example of what this structure will look like, Dr. Shinoda explained what he feels to be the most important capabilities which objectbase based integration of software environments provides. In answer to the question of whether or not we are ready for this, Dr. Shinoda feels that as far as hardware technologies go, the answer is yes, whereas the answer is not yet for software technologies. In conclusion, he asked for the start of collection of software process objects to fill the class library.

Questions/Answers

Audience:

Is a human being a software object?

Shinoda:

I've never considered humans to be software objects. I'm not sure whether the answer would be yes or no.

Audience:

What part of a software object would a human being be?

Shinoda:

In certain cases it would be useful to treat humans as objects. It's a good idea but I personally don't like it.

Discussion

This discussion session was quite long, with Mr. Belady doing a lot of talking. The first major topic in the discussion was that of the integration of heterogenous systems. Sub-topics included how such integration could be done, when it should be done, and whether it is possible. Next, the discussion shifted briefly to the issue of technology transfer, before moving on to a brief discussion on the desired granularity of software objects. Several participants discussed various issues related to the evaluation of technology before it is released. Next, a large amount of time was spent discussing Mr. Belady's concept of Type A and Type B software. Finally, the question of the scalability of environments was examined.

Audience:

I have a question for Mr. Shinoda. I think your environment is very practical, but your presentation is rather based on the builder's view. I'd like to know how you write programs under such an environment.

Shinoda:

Basically, each object has two types of attributes in my model. There is an equation which defines the input/output relation, and you can use external functions or external tools. It's data dependency driven, so whenever there's an input in the input attribute, the external function would be involved, it would be popped up as a window or something like that. The process of creating applications is a chain action of automatic invocation of tools or functions. If you write every dependency correctly, you can obtain the correct output from the final output attribute.

Audience:

In the near future, we would want to pick up good points from various environments here and there. How do you deal with such needs?

Riddle:

I think you are saying that a good objective must be able to collect together the existing technology and get it to work together. And you are saying that there should be individualized workstations for each individual person on the team. I think both of those objectives are good, and it's the only way to succeed. But getting a set of workstations working together doing networking, or even getting a distributed database, is a difficult problem. Getting tools to work together is the difficult problem. We need technical solutions to that problem, and to get people who build tools to observe some relatively simple guidelines about the openness of the tools, whether we can get in and make changes. Since these kinds of guidelines have been developed, technical people would respond well to suggestions and follow the guidelines. But there are some economic and strategic reasons within companies why they do not want to follow guidelines. We have a management problem in terms of getting these guidelines accepted.

Miyamoto:

We have to use formal and informal technology as well as graphical and textual techniques. In regard to hardware, we have to use a variety of hardware capabilities, such as RISC, Prolog, Smalltalk, or UNIX machines. Also, we have to use a group of workstations and supercomputers.

Belady:

From the answer, I don't understand the question. Have you asked the following: Since today different people use quite different hardware and software pieces, how are they going to transit into some kind of a future glorious environment? Is that the question? How do we integrate heterogenous different systems into one?

Audience:

Yes.

Belady:

Okay, then I understand the question, but not the answers. It's a very tough problem, and I think I have two different comments about it. One of them is that it occurred to me that

one panel session is missing, and this is "technology transfer." You see, the problem is that all of us on the technology side have nice dreams, and we have, particularly in the environments area, the tendency of dreaming up future environments, and I appreciate very much that it's fun. But let's talk about today, not necessarily the future. The reason is that how can you architect something in the future when you don't even have the experience about its components? The current thinking is that you have to work hard on the components of the different aspects of it, and gently introduce it into the troops, and keep improving it, and adjusting it to the demands of the population in the current environment, and unfortunately in different environments. It becomes extremely costly, because if I come up with a tool, say gIBIS, I don't have my own future environment yet in the research organization, and therefore I have to port it to four or five different environments to gather a rich enough experience of how it works. And I do it with different components. I don't think the "big bang" works. That we design a glorious new environment and watch the people with their silly stone-age things and wait for a day when I say, "Now stop these silly stone-age things. Here is my beautiful thing, isn't it nice." There is no way to do that. So it has to be evolutionary and piecemeal. Another related effort, which we started because we had to, is some serious study, by trying to do it instead of just theorizing about it, how heterogenous tools can work together, or heterogenous machines. So a part of the project now is actually what I call "integration technology." Finding out good ways of having different platforms, if you like, different tools, different pieces of technology working together on a single platform. The other set of comments is that there is good news with this kind of standardization. We are talking, in some sense, about standardization. That will help, but until there is some reasonable progress there will be lots of misery in this area. I think, although we should dream about the future, occasionally we should descend into the mud and worry about these things. Otherwise, our dreams become implemented as laboratory animals without ever being able to go out into the jungle.

Barstow:

The use of standards as a way of having tools communicate is only part of the problem. It may also be that tools have to share a kind of common framework, or approach, or something like that, and that may be harder to achieve.

Ochimizu:

I'd like to know your opinions about the proper granularity of software objects. In the case of car development, I think we can put the object at "car," and we can define it according to the definition of the object, and the working style might be well-disciplined. But in the case of software objects, we cannot put into the environment the software we are to develop. So, I'd like to know what is the proper level of granularity of software objects to bring about a simpler manner of work design style, or something like that.

Shinoda:

So your question was, "what is the proper granularity for the software objects which will be stored in the software environment?" Well, that depends on how fine you want to control the software objects as an environment. Like data dependencies between two software objects. If they are hidden, maybe encapsulated in another larger object, it won't be visible from the software environment. It would require another tool to un-encapsulate it and seek the dependencies inside. It's more like a hierarchical view of granularities. The granularity of the software object depends on what do you want the things you want in the software environment to do, or take care of. If you want the software environment itself, the framework, to take care of every entity and relation in your model, it's going to suffer from a lack of efficiency.

Ochimizu:

I'd like to know more completely your idea about the level of granularity. Documents, or source code, or a higher level of concept of software to be developed.

Riddle:

I would like to know the answer to that question also, but I don't think we can answer it. I think the problem is that that's an unanswerable question.

Sayler:

I would like to ask the question of the panel in general that as these environments are evolving, what help do you see them providing in helping to assess the value of these

extensions and additions to environments? We seem to be proceeding by building new tools, we have these new ways of thinking about things, but do we encode in that knowledge a way of assessing, or even for establishing criteria for evaluating this new technology?

Belady:

Obviously you are a hard-nosed manager, because these guys keep asking these questions all the time; how do you evaluate it before using it? I think we've been thinking a lot about it, but it turns out that evaluating this kind of technology is a very expensive process. And I envy those people who are developing other kind of technologies which are not manifesting themselves being useful in the hands of people. I really honestly do not have an easy answer to your question. I myself worry about it, because that's part of an obstacle to technology transfer. It's that chicken and egg problem, that someone says, "alright, I'll buy your technology if you prove to me that it'll bring me benefit," and then I say "well, first you have to buy it, and then you'll observe it." I think somehow cultural change should take place such that there is a willingness from those who need that technology to accept that you need money to evaluate it. Otherwise, it will be a very slow process.

Miyamoto:

There is one organization who is doing evaluation of technology, and that is the University of Maryland and NASA software engineering lab. They haven't experimented yet on advanced software support environments, but they are evaluating somehow some traditional types of technologies for programming or testing or basic software maintenance type things.

Sayler:

Wouldn't that perhaps be more specific artifacts that they are looking at, rather than process or environments in the whole?

Riddle:

Or to put it differently, there's the question of how to get somebody to buy some piece of new technology versus how do they decide which instance of that technology to purchase. I think actually that Maryland is looking more at the second question than the first question.

Miyamoto:

The important thing is discussing evaluation issues without having any kind of preconceptions or biased ideas. Then you can measure by making creative use of technology by having an objective set up, a measurement method. I think they are doing a pretty good job.

Riddle:

I want to make a response to John's comment. I used to be at the Software Productivity Consortium, and like Les we had the problem of keeping our customers happy. My feeling was that there are two things that companies can do to foster tech transfer and to get evaluative information back and successfully help people understand the value of the new technology. The first one is kind of a silly one, and that is find somebody who you don't have to convince, preferably that person should have control over the money that you're asking for. It is a critical element, to find somebody who is not looking for the data, but has a basic gut feel that the technology will be valuable for their organization, and they're usually at the strategic level in the organization. The other was to focus on a very small part of the customer base to generate that critically needed positive experience that will then help the rest of the customer base understand the value. So one thing we were doing at SPC was going out and identifying a single project within one of the member companies, and working very closely with them to get them to use the technology and evaluate it themselves, and do a little bit of self-inspection and come out with some information that other parts of that company or other companies could use to understand what it meant to them.

Saito:

I completely agree with Mr. Belady's opinion to divide the software demand into type A and type B. Maybe he insists that we need to shift from type A to type B. But unfortunately, Japanese industry doesn't consider type B to be important. Maybe you know that several big companies, very famous ones, have a contract with only one. Many Japanese companies consider that type B is only one yen, while type A is maybe $100,000. From my personal experience, we are now planning to build a large-scale campus network, and we have contracted with SRA and ASCII as an integrator, because they don't worry if we don't pay much money. So type B is very important, I believe. I would like to know how

important US people consider type B to be.

Belady:

I did not attach importance to A relative to B, like blood types. I'll tell you the context where I first talked about this type A and type B. There was, in the US earlier this year, the National Academy of Sciences organized workshop on the competitiveness of the US computer industry. I prepared this thought for that particular panel that I was on. And without attaching importance to one or the other, I concluded that they are different with respect to where they can be produced. Both are important. It's like saying what is more important, to build airplanes or to build blades for turbines. Both are important. One is component-oriented and one is the assembly of the whole. The truth of the matter is, though, that while type A software sort of has the notion of being able to be a component in different systems, shared like a tire which could be used on a Toyota and a Mercedes interchangeably, and can be developed everywhere else, type B software has to be very much related and localized, very intimately working with the people who are in that particular application, and cannot be done against a specification which sits somewhere and the other people do it. I really very strongly believe that this will happen soon, because in almost every other industry it happens. The type A activity is working toward the componentry, and the type B is assembling and reconfiguring the componentry in a particular, specific, individual application. I think if you think about it, that it's clear that you cannot easily say which one is more important, one or the other. Both are important. My final comment is, type A can be shipped everywhere, and this is in other industries as well. If you do the car assembly, the type B activity, it's close to where the market is. But the type A can be anywhere on earth, and it is shipped. But the assembly process has to be closely related to the ultimate customer and the user. This is why I believe that this script will happen.

Barstow:

Just one more comment on the type A and type B. I'm not sure how closely they correspond, but there's a sense in which the type A sort of corresponds to the components and type B sort of corresponds to the glue. We did a couple of experiments of counting lines of code, and it looked to us like the lines of code corresponded typically to about 60% was glue, and 40% was components. So from that point of view, it says that 60% of it, if there is a good correspondence with the kinds of lines that we were counting, is the glue.

Saito:

I believe the type B market is increasing more and more, so that's why I think that type B is more important. Integration is very interesting for technology. It's a new technology for integration, so we need to study it.

Belady:

I agree totally with that. When I used that slide, the heading was "Trends." There are trends in that direction. Now whether these trends continue or not, only the future can see. But I believe that there is a tremendous market. At least one Japanese company that I know of is very much interested specifically in system integration, and that's Nomura Securities. I predict that more and more companies will grab this business. So whether we do have technology or not, the market is there. What I'm saying is that if we talk about future environments, we should not ignore the need to support system integrators' work.

Teramoto:

It's still a matter of software types A and B. I agree with your perspective, and we think it is very important in the type B area. About 3/4 of our software related people are engaged in the type B work. So we have a very serious shortage of that type of people. My question is, are there any concepts of quality or productivity in this type B area? If so, what is the extent of the quality of the type B system design area? We are very troubled by making this kind of co-ordination and modification, and sometimes we need a multi-ended system. To select such kind of package and hardware sometimes affects our system's quality. It's a very difficult problem, but are there any such concepts which can be defined, or are there any concepts to make a measurement to improve this kind of work?

Belady:

I don't think that there will be particularly new things here in order to improve the type B software quality and productivity, but they will be tremendously emphasized and tougher, particularly one thing which many people talked about this morning, namely the

importance of domain knowledge, will be much accentuated. So what it means is that a much larger number of people have to be involved in the process of designing type B software, which will be a tremendous load on co-ordination, communication among people, etc. I'm not claiming that technology is the only solution, but this is a much more complex problem than writing type B. I think our friends on the expert systems side could help a lot in order to come up with knowledge acquisition which is rapid enough to do it during the project itself. So we bet on good players and computer aided co-ordination of this process. This is our current emphasis, because individual productivity is not enough, probably, in system integration.

Audience:

As Mr. Shinoda has pointed out, nowadays object-oriented database models or adaptable user interfaces are considered very effective commonly. That means that application programmers also want to put such a capability into their product. So I guess that it is possible to adapt some part of the software environment facility itself into a product. In other words, when can the software environment itself be divided into some sort of objects, or in other words when does the software environment itself become reusable? I'd like to ask your opinion.

Shinoda:

You want to sell the software environment itself, is that the question?

Belady:

I think some of the Japanese companies already use that concept, the EAGLE, I think by Hitachi, environments for COBOL programming. They were automated environments, and these companies, both Fujitsu and Hitachi, developed these systems quite well in the late 70's and early 80's with the intent to sell on the market for those who were interested in automated or computer-aided COBOL programming. Now with respect to configurable object-oriented design of an environment, I think some of them are coming now. Some companies in the US are developing it for their own purposes. UNIX is an environment for sale. Now if there are any of those developed object bases, I think the object experts should take over and list those. But you are absolutely right that future environments will be built and are being built using, hopefully, the same techniques they are preaching about.

Shinoda:

I guess the Smalltalk package is one example of an application which comes with its own software environment.

Fischer

Just a comment on this last thing. In some ways I feel these innovations are quite slow because Smalltalk has been around for at least 15 years, and it now seems that object-oriented techniques are the newest kid on the block, but one could also argue that it's at least 15 years old. But I wanted to come back to this question of type A versus type B software, and I think that at the moment, as I tried to argue, there still are software engineering people too much concerned with type A activities. I think that if one looks into complex systems, there are arguments that complex systems evolve faster if they can build on stable sub-systems. So I think what we have to do, in part, is to provide these stable sub-systems, and as we go up in layers, moving further away from type A activities, we will be increasing taking into account what are the semantics of these things which will be determined by the application domain. I feel that there's some resistance among the software engineering community, because we have to give up this notion that our software products will be applicable to all kinds of problems, and software engineers are, in my mind, too much concerned with this general applicability. So I think if one looks at the engineering of complex artifacts, one sees that we need to have these layers, and what didn't come clear, or what I didn't see from this panel, is what steps would be necessary to provide an infrastructure that we can migrate from sort of the type A activities, generic software logs, to type B activities. And how does that become a cumulative effort, so in five or ten or fifteen years from now, we provide a richer infrastructure for type B activities. So that's my question, do you have any ideas how this process may take place, that we provide this rich infrastructure on which type B activities can be developed?

Belady:

One of them I suggested for type B environments, and that is the question, is a general

broader band of communication with a different kind of background for people. That is, the groupware co-ordination and all these computer aided technologies should be added much more to the environment than just focusing on the individual capabilities, and supporting these individual capabilities. I don't know what will make it possible, but many of the things which your panel discussed, namely the customization of an environment's many windows into it for that person's particular needs and expertise is important, and then internally translate these things somehow in a computer-aided and perhaps automated fashion to make it understandable for the others, because they have to share the concern about this developing design which happens to be an integrated thing. On the one hand there will be the domain experts, and on the other hand experts on the different sub-components, and they have to ultimately totally combine as a community to design the systems. I think it would be very useful to rethink this environment, because of the aforementioned issues, into a much more computer-aided instruction environment. That is, use lots of technologies which we could, perhaps, import from the computer-aided instruction people to convey this insight, and partial insight, and component insight to other parts of the community, the developing community such that the different aspects of the system become understandable for everybody involved. Another thing occurred to me which I think Bill Riddle mentioned, and perhaps many others. I do not have an answer for it, but at least I share the problem which was mentioned. Namely, so much information will have to be stored which is related to this integration process because the domain knowledge, the component knowledge, the system knowledge, programming knowledge and so forth, are all necessary and it will be impossible to absorb and even manipulate it easily by the participating individuals. That is, in this environment I think we have to worry about data compression techniques, or filtering techniques.

Miyamoto:

Maybe I can add one thing. Some of the software in type B, as time goes by, may become type A, or some of the type A software may become B, so I think the synergy between software and software, or hardware and software should be considered in the future.

Williams:

I'm not quite sure why we ended up picking on type A and type B software in a session on the builder's perspective of environments, but since it's popular, let me continue. I'm uncomfortable with the way the conversation has been going, for a couple of reasons. The first is that software components are much different than hardware components. You don't build a copy of Lotus 1-2-3 out of lots of little pieces in the way you assemble a car out of engine blocks and tires and so on. The second is that it seems to me that the leverage, in terms of gaining something from reuse, if you tend to think about components that are usable across a wide variety of applications, is fairly small. David's [Barstow] figures of 60% glue and 40% application support this. I like Gerhard's approach of thinking about domain approaches, and rather than thinking about components, I like to think of supporting domains. For example, tire technology is a domain that supports the construction of automobiles and aircraft. I think that's a more useful analogy for building software systems.

Belady:

I accept totally your argument. That's an interesting view of a very complex issue.

Kishida:

About the type A and type B. I think the important characteristics of type B software is that it's a kind of tool environment to improve or to support some user process. The user process changes over time. So in the case of type B software, we have a continuous process of maintenance, or some enhancement development over time. So in the case of type B software, the software itself is embedded in the user application system, but also the development and maintenance process is embedded in the user process. So I think Mr. Ohno's comment is quite right, that the environment must be part of the application to be embedded in the user process.

Balzer:

I, too, was moved to come up here to try to identify some of the characteristics of type B. I was particularly struck by Les' original characterization that type B was glue, and I asked myself "how good is that metaphor?" Part of it is that it reaches out, that it gets into all the different places, that it's the strength of the system. But there's another part of the

metaphor which says that it doesn't have any structure, that what you think of, especially when you think of it in terms of the components, is meatballs and spaghetti. The meatballs are the components, and the spaghetti is somehow tying all this together, and I think that's wrong. That's really what I want to address, is that there really are two components to this type B, which we haven't distinguished yet. One part of it is the domain specific information which is what specializes the system, and why it's not in a component, and it's the real structure of whatever you're going to build. So in some sense, the architecture of some system is likely to be the type B, and that becomes more so the more domain specific you get. The other part of it is what people commonly refer to as envelopes, which is the stuff you build around a component to make it fit in with other stuff. And this, it seems to me, is more properly called the glue part, and that isn't highly structured at all. It's just a coercion type mechanism. The part I really want to focus on, and I think Kishida-san was quite right in saying that this part which is structured is also likely to change. Not necessarily because it was wrong, but because people are likely to bring in more information, and the addition of that information is going to change the total of this complex, and one goes to different structures in order to do that. So in that sense, going back to the glue analogy, it may be more fluid but yet it is highly structured. I think we need tools to help us with these different aspects of what is taking the off-the-shelf components and somehow composing them to produce something useful. And again, there are two parts to that usefulness. One is an adaptation part, which does coercion, and that's glue-like, and the other part is the real structure that makes it domain specific and allows you to customize what's happening.

Belady:

I agree with you that sometimes I talk about these things, like "the design of type B is reconfiguring." Sometimes I call the software not the glue, but the traffic control. That's another way. Because that is indeed what it is, because it's a dynamic thing; information flows, control flows, and traffic control takes place. Another comment which I would say is that I think during this discussion now, we give too much thought towards original design, namely the matter for the pieces off the shelf out of which put the new system together. But remember how I introduced this whole problem, through trends. One example is that we developed computer-aided design, and then independently computer-aided manufacturing, and now these two applications are well understood and now we combine the two. So this is a kind of a higher and higher level of integration of existing applications. And I don't know whether you can call this a component, but in some sense yes, computer-aided design is a component, and computer-aided manufacturing is a component, but they are quite large.

Balzer:

I agree with the fact that these are trends, that we're learning how to do these things, that in general by abstraction we sort of slowly learn what's going on, but many people I know in the computer-aided design community are saying that one of the big problems there is that they don't have any infrastructure. Each of them has, in some sense built their own monolithic systems, and there's no agreement yet as to what kinds of components would provide the right basis for doing the kind of integration that you're talking about. Now they all like to do that, but the question is "do we as a community know enough to identify those good parts?" And it seems to me that that's one of the slowest things, the community learning of what the right abstractions are. That's why traditional fields like mathematics, which had 300 years to figure out what the right abstractions are, do so well. Everybody knows what the right things to put in a mathematical sub-routine library are.

Audience:

What does the term "infrastructure" mean?

Balzer:

"Infrastructure" is a building base, a platform, which could be a set of components.

Audience:

From the builder's point of view, I'd like to ask especially Dr. Shinoda and Dr. Miyamoto about the scalability of the environment. I mean, maybe the methodology which would be applied to the small project and the large project would be different from each other, but in both cases we'd like to use the same infrastructure, or the same tools, or environment, or user interfaces. What is the scalability issue in, for example Dr. Shinoda's objectbase based

system. Can your system handle from the very small granularity system to the very large scale?

Shinoda:

In my version system, the scalability will be provided through hierarchical composition of software objects.

Audience:

The general strategy to solve the complex system is "divide and conquer" if it can be easily mapped to the hierarchy systems. But sometimes, even usually, we'd like to treat all of the stuff at one time, but usually to solve the complex problems you divide the problem into small parts and you treat each problem as independent, and you have the assumption that there's no interaction between these small pieces. In the real situation, in the human world, there are many interactions between the divided small pieces. In that case, we should treat all of the pieces simultaneously. In that case, the simpleminded hierarchy strategy would be harmful.

Shinoda:

You mean you want to express something like the project party in the software environment?

Audience:

After the project starts, sometimes you'd like to treat all of the project as some division of some company.

Miyamoto:

The basic idea might be to build a small one, the size of a toy. You may not want to use big hammers and nails and that sort of thing. So the idea is that if you want to develop a small or micro-sized system, the tools or facilities should be compact enough, simple enough. But in the case of large systems, a large complex system may have a variety of aspects. So multiple views must be supported. Different characteristics are there, so one single technique, one single view, or one single whatever is not sufficient. So for the complex large system, you should apply a variety of techniques, tools, and methods. So I think in the case of large-system development or maintenance, a reasonably large-scale system or facility may

Audience:

I completely agree that we need to use a variety of tools for the nature of the problem, but at least we'd like to use the same, for example, user interface, but in the existing state, in most tool systems we cannot use the same look-and-feel system because they have different platforms.

Miyamoto:

I'm not sure about platforms or inside support environments, but at least the user interface of the environment itself should be compatible, from small ones to large ones.

Session 3

Manager's Perspective

(How will it change the process or project management?)

Chair: J. Sayler
(Software Analytics)

Speakers: K. Torii
(Osaka University)

R. Balzer
(University of Southern California)

M. Teramoto
(NEC Corp.)

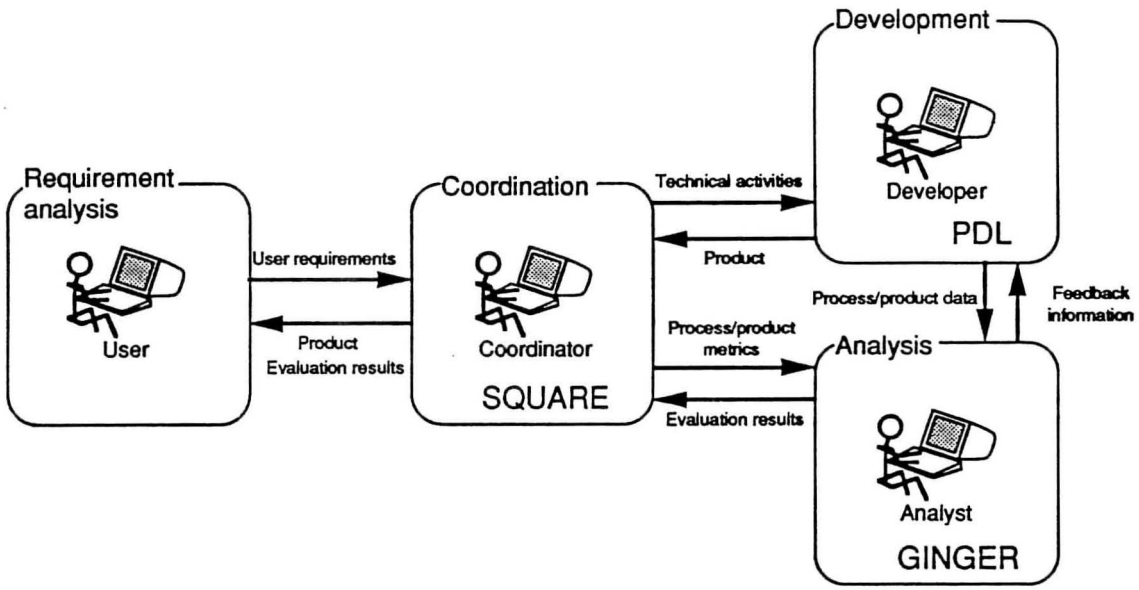
M. Dowson
(Software Design & Analysis)

Professor Koji Torii
(Osaka University)

Professor Torii gave a presentation on how future software environments will change management from an academic point of view. He stated that since he is an academician, not a manager, he could only discuss academic experiences.

Professor Torii started out by discussing definitions of the word "management," both the concrete meaning of the word and what it implies. His conclusion, after integrating several of these definitions together, was that management is something that can be discussed in terms of products and processes. The skills that he feels are required for management include planning (projects), organizing (teams), staffing (teams), directing (processes/products), and controlling (processes/products), as well as tools, knowledge and experience.

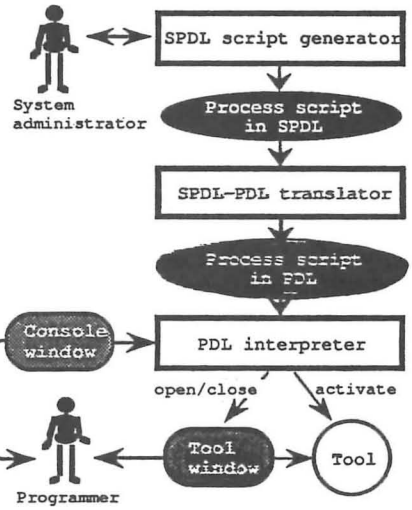
Next, Professor Torii mentioned the three projects that are being conducted in his own laboratory. These projects involve the independent development of three different subsystems. The first of these systems is called SQUARE. This is a system which co-ordinates and transfers information and requirements from the user to developers' terms. In other words, the SQUARE system is concerned with co-ordination between the user and the developer. The system takes the requirements which are stated by the user, and maps them to the developer's activities. This project is being joined by a large number of people from the industry, and it is now at the point of beginning experimentation.



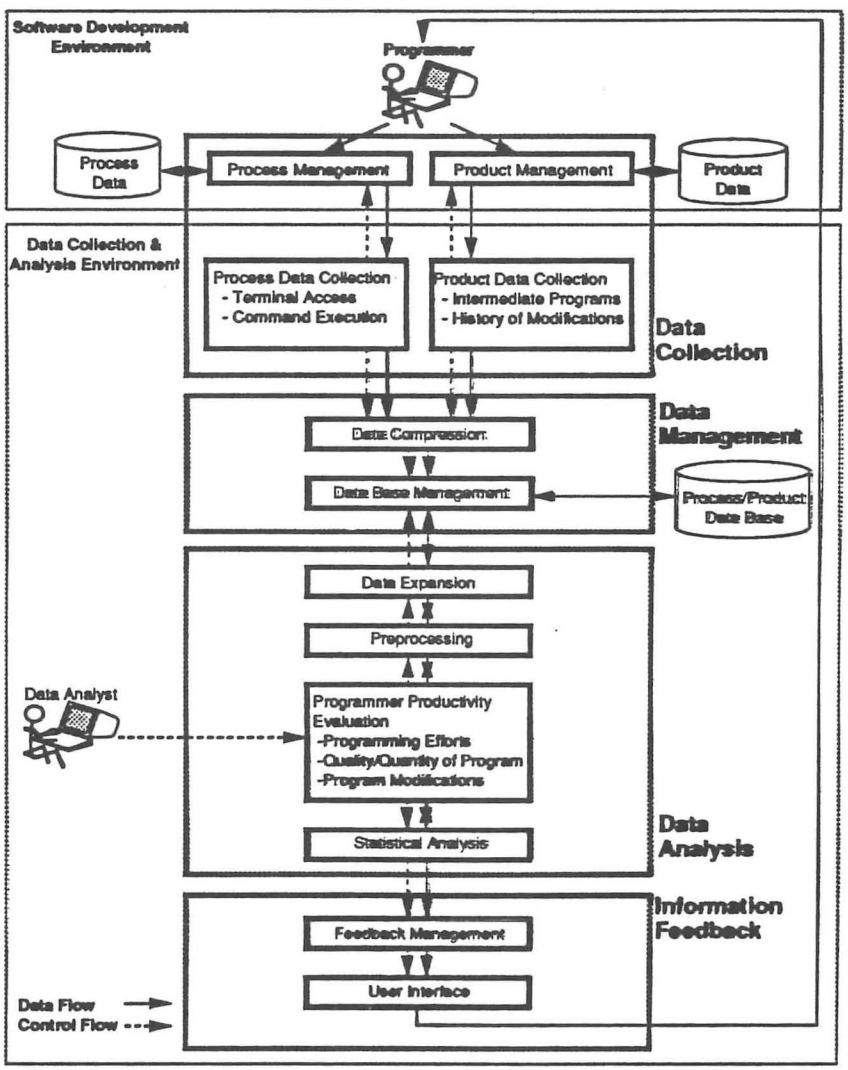
Integrated Software Development System

The second system is a developing system named PDL. In this system, first a process description is written, as in process programming. The next step is a process script, which is written in a process description language. Since PDL is a low-level language, a higher language called structured PDL is used. The PDL description is interpreted by the interpreter, which then automatically activates tools according to the PDL script. The results are shown through a window system, which is monitored by the programmer.

The third system is called the GINGER system. The GINGER system consists of four components: data collection, data management, data analysis and information feedback. The feedback information can be of great use to students. For example, they may be told that the program is insufficient and that more work must be done, or that too many changes have been made. Although Professor Torii began his presentation by emphasizing the division between product and process management, he stated that currently this system combines product data collection and process data collection. For product data collection, mechanisms like SCCS and RCS are used, while only the Unix accounting system is used for process data collection.



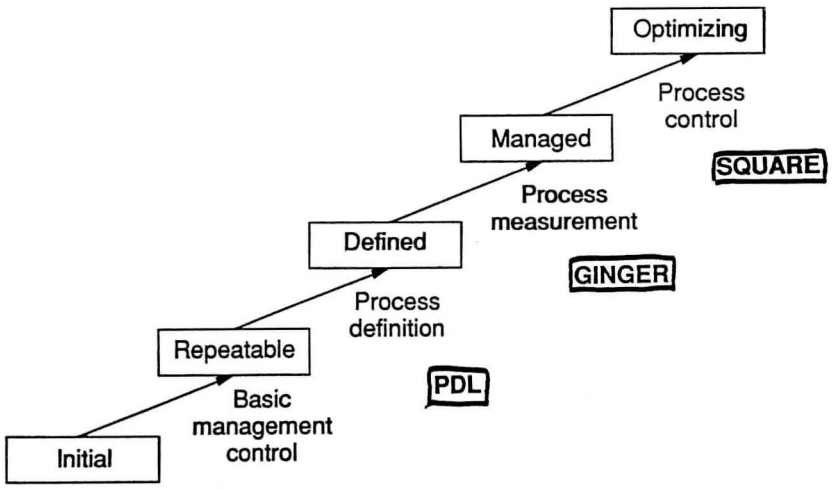
Overview of PDL system



System architecture of GINGER

Professor Torii then returned to the question of what management is. He feels that regarding configuration management, a standard definition can be found. Here, the important aspects are seen to be identifying, defining, controlling, recording, reporting and verifying. He feels that these are activities which can all be found in the experiments that have been going on for the past 5 years at his university.

Professor Torii explained that in a paper by Humphry, five process maturity levels are found. The first level is "initial"; second is "repeatable" (basic management control); third is "defined" (process definition); fourth is "managed" (process management); and fifth is "optimizing" (process which can be controlled). Looking at the three subsystems described above, Professor Torii feels that the PDL system, which tries to define the process, corresponds to the third level. GINGER, which measures and analyzes the process, corresponds to the fourth level. And SQUARE, which accepts user requirements and transforms them into developers' terms may reach the last level, as it hopefully will allow for the assurance of software quality. However, Professor Torii again pointed out that these three systems are being developed independently.

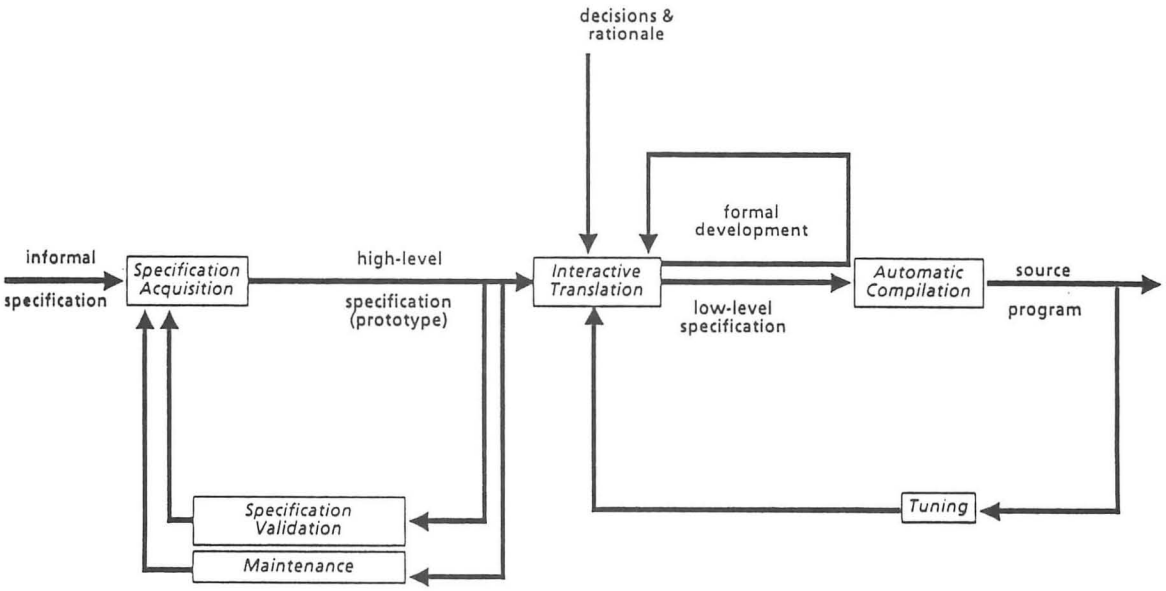


Process maturity levels by Humphrey

Faced with the question of where to go from here, Professor Torii stated that the first step is to make clear the role of each subsystem. Then the subsystems should be integrated. It is desirable for the subsystems to allow the overall system to attain higher and higher levels of controllability and manageability for the environment.

Dr. Robert Balzer
(University of Southern California)

Dr. Balzer gave a presentation on what future software environments will, in his opinion, look like. He emphasized that although he is a technologist, not a manager, he asked to be put on this panel because one of the things it addresses is how will it change the process. That is, to him, really the key issue. He began with the question of what future environments will look like, and attacked the question through extrapolation. His opinion, as a technologist, is that we are very close to being able to build systems that will be able to do all of the things that he feels a future environment should be able to do. Examining the main themes of the presentations from the previous day, Dr. Balzer concluded that their themes were essentially in agreement with his conclusion. He feels that through those presentations we got a flavor of the ways that we can extrapolate what we are doing now towards our future systems. However, he feels that automating current practice is not a desirable thing to do. Instead, he would like to use the capabilities that everyone has been talking about, capabilities that we are beginning to have the technology to support, to change the current process, particularly to improve it.



Extended Automatic Programming Paradigm

Dr. Balzer showed that in the traditional waterfall life-cycle model system, one usually goes from requirements to informal specifications, do coding, validating, testing, tuning and maintenance, all in the concrete source program. However, he feels that the main thing that is wrong with software today is that all of the feedback loops are done on the right-hand side of the diagram. He agrees with the many people who have said that we must move more of the intellectual activities into the final processes.

Dr. Balzer pointed out that the kind of system we've been trying to build is one that supports a process in which a formal specification acts as the centerpiece of the system, which can then be used as an operational prototype of the system. This means that you can do your validation on that specification, and what you're doing isn't actually validation, it's invalidation. You find out that the specification is wrong, and it is not the spec of the system that people really want. Then you have to iterate several times before you get the specification right. Then what he'd like to do is to automatically compile the specification, but the trouble is that if you've been at all aggressive in your spec language, the distance between a good high level spec and an optimized program is too great for even the so-called smart AI compilers to bridge. He feels, therefore, that what we need to do is to somehow get people's decisions and rationale into the system through some sort of an interactive system to guide us down to a low level specification that can

be automatically compiled.

Dr. Balzer then discussed what he thinks is the fundamental problem that we have in software systems. There are actually two problems. One is the inherent dilemma between optimization and maintenance. Optimization is the process of spreading information. You take advantage of what you know in one place some place else. This builds up the interconnections between all of those parts, and those interconnections are not explicit. They make it much more difficult to change the system, because maintenance wants this information to be localized. Unfortunately, both maintenance and optimization are required for the same set of programs, the ones that are around. Current practice pessimizes the situation, because we take the most optimized form of the system that we have, which is the output of the human programmer, and we try to do maintenance on that. So what is being suggested is that maintenance ought to be done instead by modifying the high level specifications, and by high level specifications we mean not optimized. Dr. Balzer stated that everybody is aware that virtually any change to a program can be described in a sentence or two. Doing it, however, can be arbitrarily hard, because instead of doing it at the spec level, programmers always operate by trying to take the optimized code, and that is much, much harder. So the suggestion is, let's stop beating our heads against the wall, instead do it the way we describe it to one another, and just re-implement. The other problem that we face is that design, and by design is meant optimization, occurs outside of computers. It's done inside peoples' heads. It's unrecorded, unanalyzed, and maintenance depends on this information. Right now people who are doing maintenance have to play detective. They have to work backward from the source code to figure out what's going on in the system. So what Dr. Balzer would like to do is to get an explicit rendition of this, and there are several people working in this direction. The previous day there was some discussion about the argumentative systems, that the way of recording the arguments constitutes design. This is something Dr. Balzer feels we need to take and put in our systems and base our technology on, because if we are going to do maintenance on the specification, then it means that every time we maintain the program we need to rederive it. And when we rederive it, two things need to happen; the re-derivation process needs to be reliable and it needs to be cheap. And in order to do that we need formal structures, so we have to capture a formal development of how we did it before, and put that in a form where it can be reused.

To Dr. Balzer, the key answer to the question of how should we be changing the process is that we want to maintain the specification and then rederive the implementations. To do that we need tools. We need tools that help us evolve the specification. The changes we make to it are not random, they're coordinated changes We ought to have tools beyond the text editor to make these changes. We need ways of reading specifications, because we all know that formal specifications written in any language are unreadable today. And we need ways of validating specifications, that is, to get feedback from the spec before you go through the implementation process to make sure that that system is what you intend to build. And then to rederive it, we need to capture that initial design, and then ways of replaying it.

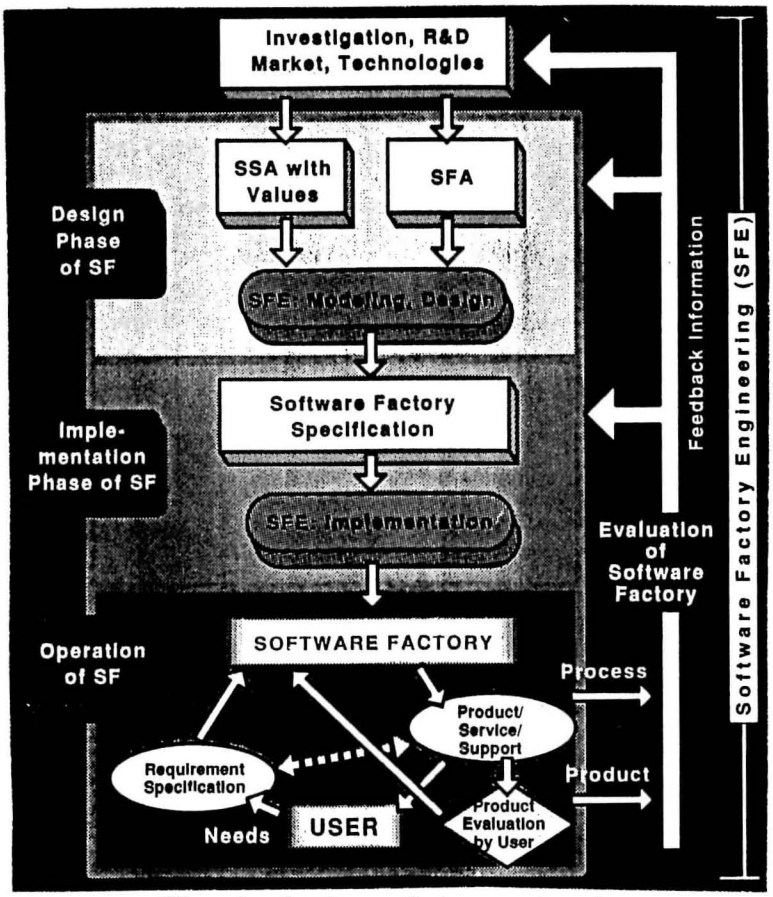
Dr. Balzer then posed the question of what are going to be the effects if we are successful in changing the process. The first one, he said, is obviously increased productivity. He feels that it's hard to look at current programming and believe that this is the best we can do. We can do a lot better in creating systems, and it's only because we haven't stepped back from the optimization problem, that we've made it so hard on ourselves. If we can drastically increase productivity, one of the things we can do is start building software with smaller teams. While he's not a program management fiend, one of the things he knows is that small teams are much more effective than large teams. People know how to work together in small teams. We have to put a lot more energy in to get large groups of people to work together, and a lot more effort goes into management overhead when such a situation arises. What Dr. Balzer sees happening is less project management. He thinks that we should be moving to take advantage of the capabilities of building larger systems with smaller teams by making smaller teams go through the process iteratively more often, so that systems arise by a larger number of smaller changes to the systems, rather than trying to build them in one big shot, or several big releases. This is a much more incremental, evolutionary approach to things, and what the management should really be about is controlling the change process, that is the sequence of new features in capabilities, performance upgrades and things like that that we want to put in the system. So it's managing the effects on the end product, rather than managing the activities of the people that needed to carry it out, because in the small group they can manage that themselves. The last thing that he thinks is an expected

effect perhaps may be the biggest one. That is really getting a handle on reuse. Dr. Balzer thinks reuse is very hard as currently described, because it requires that people have enough foresight ahead of time to have created the thing that you want to reuse. If we can be successful with near misses, then we have greatly increased our chances. He stated that if he can find something that's close to what he's interested in, and if he can do adaptation, because we just take the thing we want to reuse and evolve it a little bit by changing its spec, and then using the recorded development to re-implement it, then he thinks we have a firm foundation for really doing reuse.

Dr. Balzer then closed his presentation with a brief look at how we are going to distinguish future environments. He thinks the answer is by changes in the process. In the current environments, what we do is maintain the optimized code and design outside the computer, whereas in future environments, we are going to maintain the specification and rederive the implementation. To him, that is a very clear distinction between the current generation of environments, and the ones he thinks are worthy of the name of being called future environments.

Mr. Masanori Teramoto
(NEC Corp.)

Mr. Teramoto gave a presentation focusing on his concept of a software factory as a future management area. He feels that this idea is a way of making current technology development more effective. Although he admits that the term "software factory" gives the impression of a greasy and gloomy atmosphere, he feels that it would be the most convenient place to manufacture products using advanced technology, thereby allowing for improvements in management style.



The role of software factory engineering

In Mr. Teramoto's image of an advanced software factory, production is viewed as a cycle, with the usual kind of production line as a process. Requirement analysis, testing and inspection are also included. Given a need for pre-support, meaning activities with the customer before signing the contract, an adaptation area is also needed. Additionally, post-support is necessary after the product has been shipped, along with an information base for products and parts. Other necessary areas of the software factory are documents, quality and productivity data, process cost data, personnel, equipment, resources, and subcontracting capabilities.

One concept of the advanced software factory is called Software System Attributes (SAA). This concept deals with the question of what kind of product should be produced in the factory. There are several aspects to this question. First, there is the question of the domain in which the product will be used (business, public, home, etc.). Next there are system characteristics, such as on-line, real-time, batch, and TSS. Also to be considered is the operation environment (stand-alone, LAN, WAN, distributed processing). The question of functional level includes such possibilities as application programs, operation systems and built-in software. The quality level is also important, looking at such things as levels of reliability, function, and ease of use.

Mr. Teramoto then examined another concept called software factory architecture (SFA). He explained this to be the necessary work and activities for producing software. This includes such things as communication within the factory, research and development, sales and operation support, production lines, and management systems. The factory information base is extremely important for managing the software factory. Also necessary, however, are a methodology paradigm and fundamental technology. This technology includes things like computers, a communication environment, factory automation and even the human factor.

Mr. Teramoto explained that since there is not just one single type of software, different kinds of software factories can be derived depending on the situation and characteristic domain. Once these have been clarified, a model is made and a software factory specification drawn up. Next the software factory itself is constructed. The software factory should be evaluated, and information about how well it works should be fed back into it. Currently, no automated measures for deriving such a software factory exist, but in the future Mr. Teramoto hopes that such a delivery system can be automated.

Mr. Teramoto then concluded his presentation by examining a number of pieces of data collected within his company regarding conventional software metrics and management.

Mr. Mark Dowson
(Software Design & Analysis)

Mr. Dowson gave a presentation on his version of project management and the evolution of process, from a project management perspective.

Unlike Bob Balzer, Mr. Dowson would like to support and evolve the best of current practice. He also believes that better processes, evolved processes will indeed reduce team size. But not sufficiently to eliminate management, which seemed to him to be Dr. Balzer's suggestion. However, Mr. Dowson also believes that fundamentally, Balzer and he mostly agree on what has to be done. But, he thinks that they were addressing slightly different aspects of the overall process.

Mr. Dowson started off by taking a view of what software development consists of. As Bill Riddle suggested, he sees software development as being an activity based on constrained cooperation. It's a cooperative activity subjects to constraints, but not constraints which make all the participants totally submissive to some higher authority.

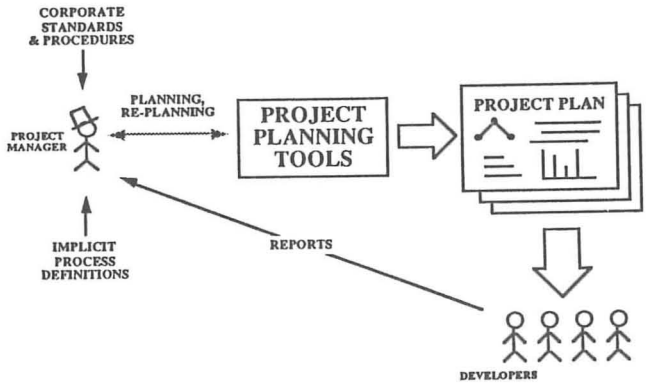
Cooperation, of course, requires planning, and it requires coordination of behaviors. A mutual coordination by agreement, some third-party coordination, and a certain amount of top-down management are necessary. With constraints which prescriptively or proscriptively define the limits of acceptable behaviors, people are told those things that they must do and those things that they must not do, and those define some limits within which they have freedom to be creative.

Given that perspective of software development, Mr. Dowson believes that a whole spectrum of levels of concern can be identified. One can distinguish, to simplify, two basic levels of concerns. One is the macro-level, the large grain, where we are talking about things like project organization, the efficient utilization of resources, the scheduling of project tasks and coordination, prediction and monitoring of the course of the project. These are, as it were, classical management activities. And then at the finer grain level there are issues like, just how do you perform the project tasks, how do you efficiently use tools to accomplish the goals of tasks, how do you sequence activities within tasks, what development standards do you apply for tactical development, and so on. Mr. Dowson stated that the apparent disagreement between Dr. Balzer and himself is that he was mainly addressing the macro levels of concern, and Dr. Balzer mainly addressed the micro levels.

When addressing the macro levels, Mr. Dowson says that we are dealing with what is called project management, the subject of the panel. Here we're concerned with planning, guiding, monitoring and, when necessary, re-planning the performance of project tasks. How is that done currently in many software projects? It is done essentially by managers with very little support. A project manager knows about, may even have access to, written corporate standards or project standards on how to run projects. A good manager from his experience has, implicitly, some process definitions; how do you run projects, how do you run different kinds of projects well. Some of that is written down occasionally, but mostly it's knowledge which is inside the head of a skilled manager. These days, managers use project planning tools, scheduling tools, resource allocation tools, critical path analysis tools, and so on, in order to plan a project and to produce a project plan. A good project plan is a very useful and a well structured document. It includes text on what is to be done on the various tasks. It includes perhaps work break-down structures, schedules, critical path charts, and so on. A good manager produces a good project plan, allocates the tasks in the project plan to his team of developers, receives reports from the developers, and perhaps re-plans when necessary. The point of this is that some of the time this works extremely well. We all know of well-run projects, of medium or even large size, that have kept to time and budget and produced good products. The trouble is, it doesn't always work, and then you have disasters. And it doesn't actually work quite well enough because our productivity, on the average, using these existing techniques is too low. And the software crisis that everybody's always talking about is creeping up faster and faster.

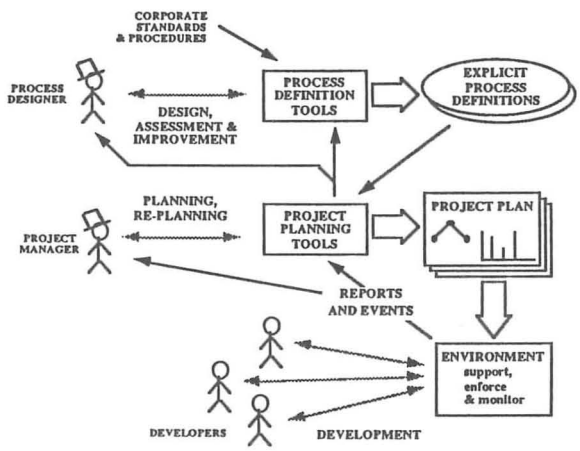
If we look at current project management, these are really the problems, according to Mr. Dowson. The success of a software project depends to a large extent on the skills of the manager who is executing the project, who is running the project. And those skills are hard to acquire. They are not very explicit. It's difficult to train people in these skills, and there aren't many

really good managers around. There's also very little automated support for the performance of a project plan and for monitoring it. A lot of that is just clerical tasks that are perfectly adapted for computer based support, but there is very little computer based support for current ways of executing, managing projects. And, there is no systematic way to improve the process built in to that. Managers, through experience, learn to do things rather better, and that's about all. There isn't really a great deal of system.



Current Project Management

So, what would we like to see? What should future project management be like? In essence, Mr. Dowson thinks that there are very similar activities. A project manager uses project planning tools to produce a project plan that a team executes. But there should be some differences. What he would like is that at the execution end of things, instead of little bits of the plan being handed out to individual team members, there is environmental support for the execution of the project plan. That is, the project plan, something very similar to the kinds of good project plans that we have today, drives an environment that the team of developers use to do their jobs. This makes it very much easier to automate many of the clerical aspects of software development and to generate reports systematically, which can then be used to actually drive the re-planning process, if necessary, so that managers can get information at the appropriate degree of abstraction in order to do their planning and re-planning tasks better.



Future Project management

Mr. Dowson stated that this is really the effort of people who are talking about process-driven environments. At the other end, the top end, how do we arrive at better project plans? He would like to be able to have ways of creating abstract project plans, process definitions, at a higher level of abstraction that can be used to guide the manager in producing a good project plan for a specific project. These are talked about as process definitions. A lot of process modeling effort is directed towards finding abstract representations of plans whose execution can then

be automated. If we can do that, we can talk about improving those process definitions by abstracting information from the actual performance of projects and from the plans that we use to drive the performance of the project, to create better process definitions.

We now have a new role in this enterprise, process designer, who gets information about the history of project performance, who uses process definition tools to produce better process definitions, which can be used by actual project managers in project planning. And now there is not so much information you have to rely on that's entirely in the heads of people.

Here, success depends not so much on the skills on the individual project manager, but on the availability of good process definitions. There is automated support for plan performance and monitoring, and because there are explicit representations of the process, there's an opportunity for systematic process improvement. Mr. Dowson said that he would not like to pretend that doing this is going to be easy. There are still quite a lot of problems that need to be solved. But it seems to him to be a reasonably coherent program for improving process, and improving project management at the macro level, which is not at all in contradiction to Bob Balzer's view of how we should improve the software process at the more detailed and technical level.

Discussion

During session 3, the chair asked that all questions be held until the discussion session, so a large part of this session was devoted to questions on the individual presentations. There was a long discussion on the differences in the viewpoints presented by Dr. Balzer and Mr. Dowson. Next, the discussion shifted to a look at management techniques regarding large groups and small groups. This led into an examination of some problems which are encountered when introducing new technology. The next major focus of the discussion was on the comfort level of developers. The discussion looked at how that level can be measured, as well as what management can do to improve comfort. Moving on, an analogy was brought up comparing different management styles to centralized state planning economies versus free-market economies. This analogy was used by several participants to illustrate their views on management. A brief look was taken at education of developers as a function and responsibility of mangement. Finally, the participants looked at the rates of change that take place in both technology and human thinking.

Balzer:

The issue that Mark (Dowson) raised was the difference between the micro- and macro-levels, and aside from the fact that I don't particularly like to be put in the micro box, I would have drawn it a little bit differently. Nevertheless, it seemed to me that on one level I was arguing for the difference between domain specific and domain independent, that the kind of environment that I was trying to paint was trying to look very deeply at what it is that is our domain, which is the software development domain, and asking what's unique about it, that we could really leverage in the way we constructed our environments. My answer was essentially trying to separate optimization from specification. The thing I find very exciting about explicit process is that I think we can, by making process explicit and by building environmental support for monitoring activities, and providing feedback, and measuring progress, that we really can do a lot better, and in fact achieve the kind of things that Mark had on his last slide. But what I wanted to point out was that in our endeavor to do that, we're switching back to a domain-independent approach. That is, I didn't see anything in what Mark was describing that was unique to software. So again, as a technologist, what I don't know is management. What I do know something about is what's unique about software. So I wanted to base my approach on one of the few things I know something about. So that's the difference in our approach, I think.

Dowson:

If I could just respond a little bit to that, I think that's right. What we're talking about is the co-ordination of a complex human activity. The only thing that is unique about the activity of software development is the opportunity that that provides for doing that co-ordination better, more accurately and more efficiently, because the artifacts that you're producing by that co-ordinated activity are inside the machine, therefore you can monitor them, you can analyze them, faster and more accurately. Also, the developers, unlike building workers or power-station constructors or whatever, do their work through the machine, so it's possible to do this co-ordination on a finer grain, and to make more changes to what they are doing more dynamically and more responsibly than you could if you were writing out a piece of paper and taking it five miles away to where somebody is hammering a nail into a piece of wood. But yes, I agree that this is not particularly the kinds of things that you need to do to define a process and manage a project at the level of concern that I was talking about. They're not inherently different for the software than they would be for anything else.

Balzer:

One final note. Again, from my limited perspective of management, I'm amazed at how well small groups can work together in the kind of very rich fabric that gets linked between the people, and the way that they interplay their various capabilities. On the other hand, my view of large project management is not one of co-ordination, but one of separation. The key dictum is, let's divide up the work so that people can independently do their things. The whole idea of management is, let's separate people rather than having them work together, because that's in some sense all we know how to do once things get up above a

certain size. So again, from the perspective of one of the guys that might be in a small box, I find it much more interesting to be in one of those boxes with a small team that isn't trying to separate me from a whole bunch of things, but allows me to have a rich co-ordination mechanism.

Dowson:

The reason for that is that there's such a high complexity of interaction in a large team that without doing a certain amount of separation it's very difficult to manage. And again, that's the opportunity. The fact that we can have computer-based support for managing that complexity in a way that is not one that just isolates people is a great opportunity if we can find out how to do it.

Barstow:

Bob, might this mean that your model, your slide, might only apply up to a certain size of system development, software development?

Balzer

I'd be more than happy if it applied for some size. If you're willing to grant me that I'm willing to take it.

Barstow:

But in terms of your goals or expectations, it may be that there is a limit beyond which it doesn't really make sense to think in terms of your model.

Balzer:

Well, it's certainly the case that the difference between the freedom that you can have within a small team and what you can do with a big team is, I think, a fundamental one that has to do with the number of people. It doesn't matter very much about the technology that you've got to support it. Imagine a very large system, something on the scale of the electronic switching systems that the telephone companies are building, or something on the nature of an operating system. You ask yourself how large a team, with the right technology, would be necessary to maintain (I'm not talking about building right now, just maintain) a thing like that. I can imagine that being quite a small number. If we really could capture all of the design decisions that went into the creation of this large system, and all we had to worry about were the changes to it that were the result of new functionality or new performance requirements being added or something like that, and large amounts of the replay were done by automation, then you might even be able to handle systems of that order. So there may not be the demand for really having large teams if this technology comes into being. Now there will always be some outlyers which you can only do by very large teams, and I don't think I'm addressing that. The question is, how much of the total software activity that we're engaged in falls into that camp? And I don't know the answer.

Barstow:

Now you were, just then, talking about maintenance, not the original development.

Balzer:

Well, in the original development, it seems to me, you need more people to make those decisions, and they require more kinds of different skills. It's not just the amount of people, but the skill set that's involved in doing that. But again, we know that for many systems, even large systems, the vast majority of the design work is done by a small team. A small set of people get together and do the architectural design of that system, and then they farm it out to large numbers of people to actually do the implementation. So even there I think there's some evidence that we as a community have really only learned how to control relatively small teams.

Williams:

Just a brief observation on the difference between science, magic and religion. If something happens and everyone in the room understands it, you're dealing with science. If something happens and one person in the room understands it, you're dealing with magic. And if nobody understands it, you're dealing with religion.

Balzer:

Well, maybe the answer is that we're trying to put it into the science category, and that's what the research is all about. That is, trying to build enough technology to support something like that. One of the things that I didn't say is that the reason that this is future is

that it is an approach which requires a fair amount of technology to make it real. We don't have that technology yet. So the only course that's open to us today is to maintain the source code. What I'm hoping is that at least we'll recognize that that's not the way we should be doing it, and that more people ought to engage in the creation of the technology to make it science, rather than either religion or magic.

Ochimizu:

I'd like to ask Dr. Balzer, do you think your paradigm can change the current development style of NEC revolutionarily? Can you do it partially or totally, supposing you were a manager of NEC? I think it is a very important problem, to change the working style from some level, to move to another new paradigm.

Balzer:

For me, NEC is a domain independent concept. There are two really hard problems. One is, what do you do with all of the old code? We've got huge amounts of systems out there. How do we handle that? I don't have a good answer for that. What I took you to ask was the second hard problem, which is how do we use, let me say old people? Old in the sense that they've become accustomed to one way of doing things. How do we get them to change both the tools and the ideas which they're using? And I think the introduction of any technology is a very tough thing. We've got some real experts here in the audience. I think by and large the successes we've had have not been in transmitting technology, but rather in moving people. You get somebody who understands it, and you get them to go into an organization, and they become the means by which that organization can incorporate it. The short answer, I think, is that we need some successes. What we need is that instead of people like myself talking about what might be possible once we have the technology, first of all we've got to get the technology, and then we've got to do some case studies which show that it really works. I'm not talking about small differences in effect. I'm talking orders of magnitude better. If it works, it'll be very easy to notice it. Once you have that, then I think there's a chance of a few people in different places adopting it, and then they're the ones who can really help make it work within an organization. But it isn't going to happen real quick.

Ochimizu:

I'd like to hear a discussion between Dr. Balzer and Dr. Teramoto to recognize the gap between the ideal and reality.

Teramoto:

Basically, I completely agree with Dr. Balzer. The way of thinking is something different. In Japan, as I showed before, there is concrete data, which is most persuasive for many managers. Many successful examples will work for changing the management thinking and decisions. As another situation, if IBM had a success in that way, all of the Japanese manufacturers would follow. This is the kind of current situation. But in the future, we should select by our own decision, according to real data and real examples, I think.

Katayama:

I would like to ask a purely scientific question of Dr. Balzer. I think that you are based on the existence of specification language, from which you can have good optimizers, and you can observe good correlation between the pictorial information of the specification and the behaviour of the generated codes. It seems to me that this kind of restriction makes you restrict the selection of a specific language. Suppose you want to adopt the purely development languages, say ???? logic. It is not easy to find a correlation between the behaviour of the program, the codes, and the specification. What do you think about that kind of thing? To my knowledge, a very high-level operational language might be usable.

Balzer:

I was not trying to advocate any particular kind of language. I think we need a fair amount of experimentation at the specification level. As much work as we have done as a community on programming languages, we have done only an infitesimal amount on specification languages. So I think it ought to be clear to everybody that we, as a community, don't know how to create specification languages. We don't know what constitutes a good specification. But the main answer to your question was, it is not a property of a specification language that it should make optimization questions easy. In some sense, the language ought to be quite neutral with respect to those things. The more that you can

abstract and build a good specification that doesn't deal with optimization issues, the more you have pushed the hard questions into implementation, and I think that's appropriate. But then you have to deal with them. What I don't believe is that we can build automatic systems that are going to be smart enough to resolve those optimization questions. Instead, I think we have to get the insight of people, and probably many different people, to use their best judgement as they do today when they actually go out and build those things, to decide what kinds of optimization to employ. What I want to do, though, is to have that be the only input we get from programmers, is decisions about optimization. That the actual carrying out of those optimizations is something that should be done by machine, which is because it's clerical in nature and machines can do a much better job of that than people. What we'd like to do is to have people have the opportunity to build dozens of large systems, or dozens of alternative implementations of the same large system, so that they can, through feedback, see what kinds of optimizations really work, and in particular how different optimizations interplay with one another, because they're not independent. There's a high degree of interaction between many of the decisions that you make in a large system. What we need to do is to give people better tools.

Katayama:

I have a big concern about the correlation between the behaviour of the program and specification. If the nature of the two languages is very different, suppose you have a bug which can only be detected by running the program, and suppose you find that kind of behaviour fails to be detected at run time. It must be very difficult to locate the errors in the specification if the two languages are very different.

Balzer:

This is the problem of mapping concepts in the implementation with concepts in the specification. You're right that the larger the distance between those two, the harder it is to maintain those mappings. On the other hand, the more that those mappings were actually carried out by transformations performed by the machine, even if selected by people, the more we can instrument or augment those transitions with mapping information, kind of the way that right now we require compilers to provide some mapping information between the source that they take as input, and the object code that they produce as output. We could do the same sorts of things with this. The other thing to say is that we should be designing specification languages that can act as prototypes themselves, so that at least many, if not all of the bugs that exist at the specification level, can be found via interaction there, rather than waiting for implementations. But of course we won't get them all, so this problem will exist.

Dowson:

I'm somewhat worried by the idea that there are just two artifacts in the world, "the specification" and "the implementation." Of course if that is the case, then the distance between them is very large, and it's very difficult to determine if there is a bug in the implementation which a bit of the specification causes. I think a much better viewpoint is that there is a sequence of representations of the system, from the abstract towards the concrete, almost irrespective of the language which they are expressed in, successive levels of refinement in implementation. So the distance between any two representations is relatively small. Then, the most difficult step, where the distance is largest, is where there is a change of paradigm, perhaps from a declarative to an imperative way of defining the system. There doesn't have to be this enormous distance between the single abstract specification at one end and the single concrete implementation at the other.

Barstow:

Either you or I can say this, but there's a third piece. There are three artifacts, at least, in the slide. One is the spec, one is the code, and the third is that formal development that Bob talked of. The description of the steps that he took to get from one to the other. And it's a very crucial piece.

Balzer:

The other part of the answer is that many of the implementation decisions we take are what I would call "weak implementations." That is, they don't fully meet the requirements of the idealized spec, and because of that, they actually result in an augmentation of that specification which occurs down at the implementation level in implementation terms. So for

instance, the decision to use computer arithmetic for something that is expressed in the real world, provides the opportunity for an overflow. You don't have overflow in the real idealized world. This is only something that can occur because of the implementation technology we've employed, and because of that you may have to introduce how the system responds to an error there. So you say, "well, if we get an overflow we do the following thing." But that is an augmentation of the specification that's caused by the particular implementation choice you've made. Another more realistic example is what happens when you decide to use real computer networks as the way of supporting a distributed system, and you know that the links in that distributed system may go down. So you have to augment your specification of the ideal distributed behaviour by what you're going to do when you can't communicate with different parts of the system.

Dowson:

That makes me worry about one aspect of your slide. I think it's quite correct that one should be paying more attention to artifacts of the specification end of the process than at the implementation end of the process. But you still seem to have this kind of cutoff between the two stages, where you do a lot of work to prepare the specification, and then that's done, and now you start implementing. What you have just been saying has been recognizing that it is not as clear-cut as that. I've been looking at some projects that have been done in the UK, where they've been using formal specification techniques as a way of approaching implementation, and specifying "in z" and then building systems in objective C. One of the comments of the company who are doing this is that a critical aspect of that approach to development is to refine and maintain the formal specification throughout the implementation phase.

Balzer:

I certainly believe in that. I thought I said several times that what you really want to do is to maintain the spec, and part of maintaining the spec is changing it in response to insights that have been gained while you have been thinking deeply about the system, which occurs during implementation. We get lots of good ideas during implementation, not only of limitations of things that we can't do, and therefore have to employ these weak implementations that reflect back in changes on the spec, but also other opportunities, places where we find that the spec didn't recognize the overlap between two different concepts and that we can do abstraction and build a more general system. We see this all the time, that people in the midst of doing something understand the possibility of doing it better, and go back and modify the spec. Unfortunately, today a lot of times they don't go back and modify the spec because there is, in fact, no spec to go back and modify.

Dowson:

Or because there's no management constraint which insists that they do so. This seems to be one of the key contributions of good project management, is to make sure that some exceptionally dangerous shortcuts, like patching the code, do not happen.

Balzer:

I'm glad you said that, because this is the difference between what I would call a technological approach and a management approach. That today we have to rely on management to do that, but with the approach I'm talking about, one has to do it because the technology won't support you unless you do. Unless you modify the spec, there's no way of using the spec to rederive the implementation. So in fact, people are motivated not by management, not by the stick, but by the carrot.

Dowson:

We're in total agreement. We want to support people to do things the right way, and there is more than one approach to doing that.

Sayler:

There are other aspects than motivating people to do things the right way. A question from Mr. Sugita for all of the panel is "how would you measure and analyze the comfortableness of project members in these new future development environments?" If we don't motivate people to use these technologies, we won't have success.

Torii: I hope every student will feel comfortable. We don't have any experience in thinking about comfortableness in our situation.

Teramoto:

We started our research activities nine years ago. At that time, comfortableness, or the work situation, was the largest subject of study. At that time, the usual office situation was very bad. Very narrow places, no workstations or personal computers, and a lot of paperwork. After that we made arrangements to do some study on what is an appropriate space to work in. After such studies and some practice, we got some standards about the working area, and many people answered that it got more comfortable. But most impressive is that after they each got a workstation or personal computer, they became very motivated. There was a great improvement. I think comfortableness is not only the living conditions, but how smoothly the work progresses. I think it should be measured by such aspects. We usually use a questionnaire style to measure whether it is comfortable or how much productivity has raised. Usually the management side and the engineer level are very different in these numbers, but sometimes it is well correlated.

Dowson:

I'd like to comment by describing two different companies, which I will not name. In one of the companies they use very systematic development methods, formal specifications, a disciplined process which has a strong emphasis on reviews and inspections and on quality control. In that company, the developers know what they are supposed to be doing, they have clearly defined objectives, these are realistic objectives, and management takes good corrective action if something is difficult, if a problem arises. From talking to them, these are very happy and comfortable developers. They like working there. I know another organization where no-one knows what they are supposed to be doing, and management is by exception, or a better way to describe it is "management by panic." When the deadline is exceeded, all the management runs around in very small circles shouting. Nobody there is happy; they are not comfortable, because there is no disciplined process. No-one understands what they are supposed to be doing, and they are not well supported in doing it in a co-ordinated way. Now, on the surface, the people in the first company have less freedom, they are subjected to more constraint. But this is constraint to do things well, and when that is the constraint that is applied to the developers, people like it and they're comfortable.

Sayler:

It was suggested in a small discussion yesterday that perhaps we want environments that are like a gentleman, that one is comfortable with them. What you're suggesting, Mark, is that the answer is less in technology than in the principles of management.

Dowson:

Yes, I think so. I'm also saying that a gentlemanly environment doesn't necessarily mean one that is undifficult.

Barstow:

I had a caution for Bob when you spoke about orders of magnitude in improvement. I used to talk about orders of magnitude too, but then I did some calculations, and the best that I could guess for an approach like you were talking about was a factor of two or three. So I fear that you're setting yourself up for failure by saying that if this all works we get orders of magnitude. I think that that's unlikely, because there are other things involved in the process of producing software that it doesn't address, like communication among people. I would just caution you not to set yourself up. If you get one order of magnitude you've done better than my calculations. I hope you do, but I just caution you against setting up an unrealistic type of expectation.

Balzer:

That's a fair comment, and I agree with it. The thing is that when one introduces a fundamental change, it's very hard to identify exactly what the measures are. One of the things that I notice about a lot of systems is that the ones that are being used tend to have large backlogs of things that people would like to have done to them, which there just isn't enough energy to go out and accomplish. The problems of building the first big instance of a system have all of these communication problems, the learning and things like that. On the other hand, if the way that you really operate in the world is by incremental modification, and most of the system knowledge has, first of all, already been incorporated and second of all it's in a reusable form in the system, then a lot of those calculations which get in the way right now, because they talk about big people communication problems, big design

problems that we have to overcome today, may not be part of the process at all. We may be in a very different kind of environment. There is room, it seems to me, both for optimism and for caution.

Barstow:

Mr. Teramoto, in your reuse I guess you have libraries or a set of components or something. What is the granularity of those components?

Teramoto:

The reuse area, the granularity of this kind of thing is rather large compared to the usual kinds of packages. That kind of area is domain specific, as you say. I think currently we have such reusability at large granularity.

Saitoh:

My question is non-scientific. In the last slide of Teramoto-san's presentation, he showed bottom-up and innovation, although NEC's software factory is very rigidly managed. I doubt if innovation can be generated through rigid management. In the presentation given by Bill [Riddle] yesterday, he pointed out that it is important to balance between discipline and chaos. Discipline is rather a top-down constraint, and chaos is rather a bottom-up cooperation. In Mark [Dowson]'s environment, he added the process definer for the process manager. It is rather dangerous to give too many constraints to the developers through the process definer and process manager. In real society, it has turned out that communism is bad because it gives too many constraints to the human beings. I doubt that the environment or the process management give too many constraints to the software engineers. I'd like to ask all of the speakers about this problem.

Teramoto:

I'm sorry, but I cannot explain the details of my idea. What we are doing now is to make some arrangements with the management that is like the solution of Eastern Europe. Until now we've had a more concentrated policy towards managing all the software production. What we are doing now is to distribute to each site. Each management should have their own objectives, and they need some reasonable measures to make their objectives. That's why we make such kinds of productivity curves and quality curves. It depends on their situation. After that we ask the management "how did you get these objectives," and let them explain their measures for co-operating with that improvement. They should think about how they can improve their own situation. At that time we have the chance to put in some new technologies. If they need some new environment we introduce new technology and let them raise their own indexes. As you know, our business area is very wide, so it depends on each domain. The essential point is to let them have their own objectives and reasonable measures.

Dowson:

Personally I think that it's clear from the events in Eastern Europe that centralized state planning does not work very well if you want to manage an economy. It's also clear, I think, from the American and British experience in recent years, that the complete free market is not very good either. Let me offer a suggestion about the proper function of government and equally about the proper function of management. That is that it should create a framework where decisions which are locally optimal are also globally optimal. That is, we should provide a framework which encourages people to make decisions which to them seem like good decisions that contribute to the common good. An example of this might be to make it very easy to test work products in a disciplined way, to make it cheap to do that, rather than expensive, by organizing project structures so that it's easy to test things well, and then modules will be tested, and that will be to everyone's benefit. So it's establishing the right level of constraints that are not telling people exactly what to do, but are providing boundaries, a framework, within which they make good decisions.

Torii: Each management must be adaptable. The problem in management is what is to be manged. Maybe it can be divided into three parts; process, product and human beings. Those three parts must first be measured or evaluated fairly. So we have to think about how to measure mechanisms, and based on that result we have to think about mechanisms for how to manage. It depends on each project or each organization, each company or laboratory. They have their own mechanisms to control or mechanisms to manage, but in either case they have to have something based on that precise value or result. That's why I

would like to emphasize the importance of measurement.

Audience:

I have two questions. One question is, in Japan we always include education in mangement. Maybe the culture is different, I don't know. Do you think we have to think about education, or not? My other question is that of course an environment will change day by day. For example, we have a local area network, so we have to manage others areas of people. I'd like to know if you think management will change or not in the future.

Dowson:

Education certainly is very important. How one manages the education, acquiring new skills as an integral part of doing development is something that I don't think we understand very well, but I agree that it is very important to do that. The dynamic aspect of management, are you asking whether what we have to do will change?

Audience:

I don't know, but maybe some actors or some arguments will change.

Dowson:

I think that there are two ways in which change must happen. I don't think that you can statically plan how a project will proceed, because the world changes from unexpected events, or if it's a long-running project because new techniques, methods, tools, equipment arise. If management is not capable of accomodating that, then it won't work. That's within a project. Between projects, the kind of world that we do projects in is changing, it's becoming more distributed, there's more computer power available, technology is going to let individuals do more. That's great. And the techniques we use to manage work as our environment and technology evolve, are going to have to change. And again, we don't understand that very well, but it is clearly important.

Torii: Your comment reminded me that the education of programmers has become a good business opportunity. In Japan, preparatory schools are very good business. The reason is that there are very big, severe entrance examinations for the universities. But when we think about software engineering, we don't have any exams at all. How can we certify the levels of engineers? Maybe we have to think about evaluation systems. In other words, we have to show them the goals or sub-goals. In my experience, if we can give programmers information about their goals, we can improve many things. Even if we think about the education systems, I think that there are lots of ways we can improve their skills, technologies or motivations.

Balzer:

From the kind of things that I'm talking about, it's clear that education is crucial. Whenever you're trying to introduce some new technology, you've got to get people trained in that, people who've got some experience, so they can start using it. But that's just one aspect. The other is, to me, the closely related question of capital accumulation, and I mean intellectual capital. Mark [Dowson] had a very nice slide that he summed up with, that was showing what can happen if you make process explicit. It gives you the opportunity of making systematic improvements in what's going on. I think the same thing is possible with respect to the domain-specific systems that people are building. People learn by doing, and if they're building some system they're going to learn something about the domain and about the set of techniques that have worked, and that is the computer science aspect of this. If we give them a way of representing that, which it seems to me is what the design record is all about, it's a way of making that explicit in a form that can be reused, then there's a way of building capital within an organization that captures the insights of the developers of the system, and that becomes the basis of doing other things, or evolving this tool for other applications that are coming along. It's another form of capital accumulation, and I think a very important one.

Tamai:

I have two questions for Teramoto-san. One is that you showed impressive data on productivity, quality, and reusability. What I find is that since all of those measurements depend on lines of code, it can be easily cheated if the programmers include meaningless code from the library. If they enlarge the program, the productivity, quality and reuse rates all become higher. Do you have any specific mechanisms or measures to prevent this? The second

question is that recently in the United States the subject of co-operative work and co-ordination has come into focus. I'd like to know if that kind of approach to activities is in the Japanese traditional way of doing things. So I'd like to hear your opinion or comments comparing any similarities or differences in the approaches of Japan and the US in terms of this subject.

Teramoto:

For the first question, we have the same argument in our company, exactly as you said. We have people who are against our approach. If we show only the productivity in lines of code, there may be such a problem. If you see the year by year curve, it's very easy to understand the productivity raise. In the future we will try to compare with more accurate indexes or measurements, but currently it is nice to compare year by year, only to see. We don't evaluate the difference of each productivity and quality. We only look at the trends. We find that many divisions have the same tendencies and the same levels, and it is very useful for the management. Recently many division managers have agreed to this kind of data collection, but they still have some questions about the accuracy of that data. In the future we will have more accurate measurements. For the second question, I think that the objectives are the same as our co-operative work and quality control activities. In Japan, small group activities are very suited to our people's characteristics, who wish to have the same objectives and the same group, and to communicate with others, and also to improve the work environment. In the United States and Europe, people pay more attention to individualism, their first intention is to improve their own abilities. But in Japan and the Orient, people's intention is to raise their group's abilities. We've had some discussion about automation and motivation, so in Japan the motivation is first, and after that they take such automation as environments and tools. In the west, they first try to introduce automation, because human ability depends on such kinds of tools. So first they intend to raise their abilities using such automated tools. That is a difference between the two sides, but the final objective is the same, and the co-operative action should work well for large projects.

Fischer:

I had sort of an interesting experience with respect to trying to separate technological changes from how humans change, or how cognitive behaviours change. One is the technological change. I was at MIT about 10 years ago, and they had their first laser printer there, and could print one page a minute. There was a long line because it was a great thing. Ten years later people feel that they have to have their laser printer in their office, because it's too much work to walk down the hall. So this is sort of the speed of technological change. Now let me say something about human change. I was at the IFIP conference in 1974 in Stockholm, and there was a panel, and the question which the panel addressed was "Will there be a Fortran in 1980?" At the time, it would have been like these people would have said something totally insane if they agreed that there would be a Fortran in 1980. So in 1974 these experts all agreed that there would be no Fortran in 1980. Now, having hindsight, we can make a better judgement about this. The question that I have derived from that is, do your comments about future software environments relate to these two observations about the rapid technological change, and the slow change in how humans are willing to give up cognitive tools which they acquire. How will that impact some of the statements which you have made?

Balzer:

The observations you make, I think, are well taken. People resist a lot of change, and you have to provide a lot of motivation to overcome that. I think in the case of programming languages the case hasn't really been made. The differences among different programming languages are small compared to the cost of learning, but more importantly, the cost of integration. Most of the people who are still using Fortran have huge inventories of Fortran systems that they and their colleagues have built, and they want to augment and improve those. And a new language, which doesn't interface very well with all of that is really a problem. A lot of things that I've been talking about are going to face the same sorts of barriers to being adopted. Again, I think it will be a slow process, that it necessarily is very difficult. People have to be willing to, essentially, step outside their job in order to adopt something new. They have to go learn how to do it, they have to begin to incorporate it, and that whole process and how you manage the technology integration is very tough. It

seems to me that people who have found incremental ways of allowing technology to be introduced have a much easier time of it, and I haven't figured out that path yet. It takes a lot of people experimenting and finding out how to remold things so that you can do this. We need to lower our aspirations with respect to how fast any new idea is going to sweep. On the other hand, let me point to the fact that the ideas of structured programming have really been quite widely adopted, as have the ideas of object-oriented systems. Of course in the adoption of those things, there's been a certain amount of what I'll call corruption, in the sense that people mean very different things when they use these terms. But that's in some sense a natural part of the adoption process, that people pick up some aspects of the overall idea, and incorporate them and specialize them in ways that fit into the environment.

Barstow:

There's another example of very rapid change, and that was the introduction of spreadsheets. This is an idea that kind of just came to somebody, and it's really taken hold. So that's one cause for optimism.

Balzer:

Yes and no. Yes, they were adopted very quickly, but there are actually two nos. One is that it wasn't on the first try, that there were in fact several spreadsheet systems before Visicalc caught on, and a large part of why it caught on was the availability of the individual machines, the pcs, to make use of it. The other thing is that they weren't really used in the way that was originally envisioned. They wound up being used for experimental purposes. People were asking "what if" type of questions, and using them in a kind of CAD-type of mode to get some feedback and understand the sensitivities of the models that they were building. So it did happen quick, but it didn't happen in the way that the creators had envisioned.

Fischer:

I think that there's another part to the story, and I would agree with Dave that spreadsheets have caught on much quicker, whatever measurement you take. I think it's related to what was discussed a little bit yesterday about domain orientations, the immediate recognition of the non-trivial side of the user communities that this is something useful. There has been a growing characterization of computer science research in general that we have kind of operated, if you compare it at a supply versus demand dimension, in sort of a supply dimension. We have constructed tools, and then thrown them over the fence to the rest of the world and said, "look, we have this beautiful tool, why don't you use it". And again, I would claim that why spreadsheets may have caught on so quickly is because they are considered more in the demand dimension. It wasn't that we, as computer scientists, invented a new tool, threw it over the fence, and said "we thought this was a nice tool, why don't you out there use it." That also sort of explains the Fortran thing. The user community had used Fortran, and had huge investments in it, and they couldn't care less whether there was a new formula and tricks which some computer science community, because it's also their professional esteem, realized in a new programming language. So I think this dimension which we have to keep in mind, and which may explain the differences, is the supply versus demand orientation.

Audience:

I have two questions directed to Professor Torii, and one directed to Professor Balzer. My first question is very specific: please explain about the SQUARE environment. What kind of co-ordination does it have support for, and what kind of protocols are used in the SQUARE environment? Those are my two questions. The final one to Professor Balzer is that according to your paper which I read some time ago ("Living in the Next Generation Operating System"), some kind of DBMS is an essential facility of the new generation operating system. In the OHPs you presented today and your presented process model, where was such kind of new generation operating system addressed?

Torii: I'm sorry, but I cannot answer in a minute, so if you don't mind I'd like to explain later.

Balzer:

I'll try to answer in a minute. I categorized the goals of our group into essentially two camps. One is to make real the paradigm of software development that I've been describing today. The other is to develop a style of system development, a style of software building,

that makes use of database concepts as the way of envisioning, conceptualizing, and implementing whatever kinds of software we're building. In that "Living in the Next Generation Operating System," we described how the operating system as an application, that is if you think of it as something that you're building, how it would use for its own purposes this kind of advanced database that I was describing. We are pursuing how you develop a development methodology that helps you use those database notions in whatever application you're building. To my mind, it's essentially the movement away from pointer-based data structures to data structures that allow content addressing, associative retrieval. So that in our specifications we do not have to create the kinds of access structures that occupy large portions of our application programs. So we're trying to create specification languages and implementation technology that'll allow us to use those kinds of notions. It's good that David brought up the example of the spreadsheet, because there are two things that made the spreadsheet so powerful. One was that everybody knew how to program them, because they knew how to write those formulas. The other was that the system took over the maintenance of consistency, and that's a very powerful notion, but one which we have not really utilized in the systems that we build. That's again a large part of what system design programmers do, is operational mechanisms for maintaining consistency. It's very nice to be able to at least create specifications that by merely stating the constraint, cause effects to propogate out, the way that spreadsheets propogate things, as a way of describing what we want to have happen. Those are the key notions, the ability to use constraints, the ability to have descriptive access to the data that's being managed, and one more, which is to hide the difference between derived data and ground data. We all know that those store-recompute issues are what dominate the implementation decisions that we make in systems. We're trying to get that out of the specification. So the answer is that we think there are some very powerful notions of how you should conceptualize a system, that make use of database ideas as a way of defining the semantics of some environment which you want to build a performance program, and the performance part is something that we want to attend to after we've got the functionality laid out, and we think that these database notions help a lot in making that kind of separation.

Session 4

Researcher's Perspective
(What will it take to achieve it?)

Chair: N. Saito
(Keio University)

Speakers: M. Dowson
(software design & analysis)

T. Katayama
(Tokyo Institute of Technology)

L. Williams
(Software Engineering Research)

K. Kishida
(Software Research Associates)

Professor Nobuo Saito
(Keio University)

In his opening remarks for this session, Professor Saito set forth the thesis that in the field of research there are two categories: the theoretical approach, and the realistic, or practical approach. Examining first the theoretical approach, it was shown that a researcher must have a good conceptual framework. Also, if there is a good framework it is necessary to have a good description and construction paradigm. Professor Saito expressed the opinion that this is not a particularly useful approach.

The realistic approach, in his view, is more useful to researchers. In this approach researchers investigate the real world, making use of a good architecture and integration scheme. In such an approach, the researcher must construct a real environment.

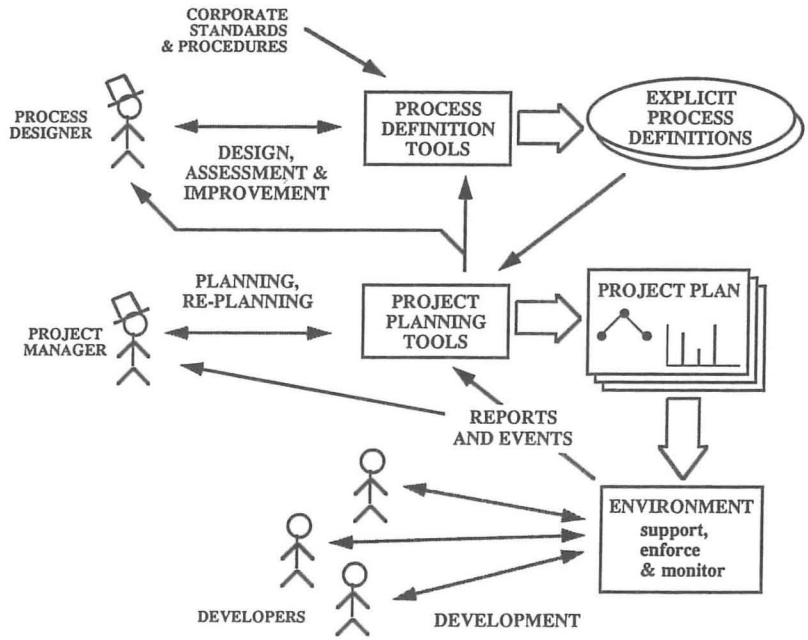
Based on these premises, Professor Saito then opened the session to the presentations from the four panelists.

Mr. Mark Dowson
(software design & analysis)

Mr. Dowson gave a presentation on "Research Issues in Process Evolution." He began by asking the question, "given the view that has been presented of project management, what research problems must we solve?" He then proposed that explicit process definitions can be created by a combination of the skills of the process designer and by abstracting from other projects.

Mr. Dowson then presented a number of issues that must be dealt with if we are to guide the improvement and refinement of project plans in the future. The first of these issues is that of representation schemes for organizational aspects of project plans as part of process definitions. He listed several examples of the kind of things one wants to be able to represent. The next issue is that supposing a means of representing the above information exists, we then want a way to derive abstract representations to infer actual process definitions which are characteristic of actual process performance. The next issue, the opposite of abstraction, is that of reification, or making concrete. In this sense, reification techniques are necessary to support the use of process models as constraints on the planning process. It is also necessary to have planning approaches and tools that support distributed, co-operative, incremental planning. The final issue raised was that of reactive planning, where it is necessary to solve the problem of how to modify existing plans when faced with unexpected project events.

In summary, Mr. Dowson explained that the above are problems without good solutions which are critical to process management. We cannot, in his opinion, proceed without solving these problems.



Future Project Management

Questions/Answers

Balzer:

The issue I wanted to raise was the set of representation problems that you mentioned. To me, it doesn't seem to be a representation issue. To me, all of those things are constraints of one kind or another. The problem is not representation, but what do you do with the information once you've represented it. That is, you want some effect of having stated that constraint, and so the question is how you, in some sense, propagate the statement of that constraint on the processing being done in the rest of the system.

Dowson:

Well, that was what I called reification. Given some constraint stated as a type almost, as a general prescription, how do you create project plans that observe that constraint? But in order to do that, you have to know how to represent the constraint in the first place. And I don't know how to represent that kind of constraint. How do I represent a constraint like "organize implementers into small teams"? I don't know how to represent that in a way that can drive a tool-supported plan generation process.

Balzer:

The problem is not representing the constraint, because there are lots of languages in which you can say that if you define a type that is "implementer team," you put a constraint on it which says that you can't have more than three people in that set. The problem is, having made that constraint, what other part of the system is willing to listen to that constraint that you've imposed. The difficulty is that we have fairly narrowly focused tools that aren't willing to take in a very large set of possible constraints we might describe. The other thing is that you've raised the issue of trying to use existing tool sets, and that's important. But these existing toolsets weren't built to accept the kind of constraints that you're talking about. It may well be that we have to build new tools, or at least have to convince the builders of those tools that they're going to be used in a different environment, and they've got to come up with new versions of these things, so that we can make more effective use of these tools.

Dowson:

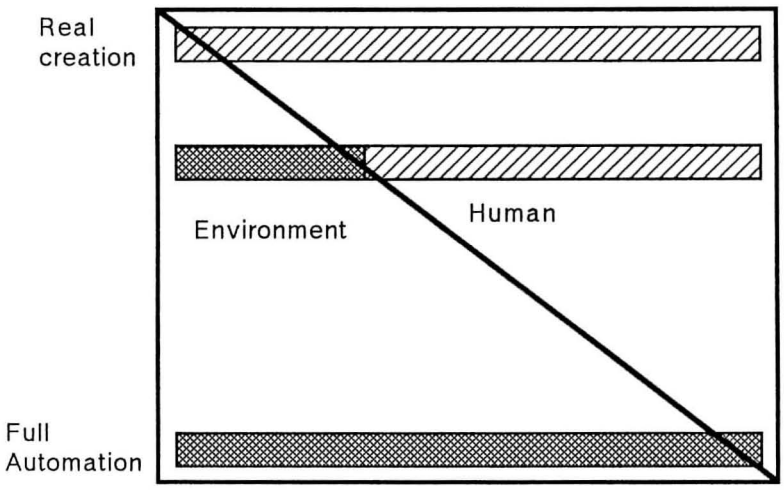
Yes, I think that's right, and I think that in the end indeed we will have to build new project planning tools as part of this exercise. I would certainly like to be able to start without doing that, to find incremental ways of doing that, to exploit existing tools for as long as possible, and get the maximum advantage out of them, before we are reluctantly forced into building them ourselves. Perhaps I should have said that more explicitly, but it seems to me to be such an obvious principle, that you don't rebuild when you can reuse, and if you can't reuse directly, you start by adapting.

Professor Takuya Katayama
(Tokyo Institute of Technology)

Professor Katayama gave a presentation on "Research Topics for Future Software Environments" with respect to software process research. He began by stating that environments can be considered to consist of processes, an object base, tools, and a user interface. In turn, processes consist of functions, behavior, and enaction. The two questions that he raised are those of how to formalize/model processes, and how to prepare process scripts. He explained that at one time he felt that attribute grammars could be used for describing processes and object bases, and he found that he was, to some extent, correct. However, it seems that this premise is, for the most part, incorrect.

Professor Katayama proposes three views of software processes. The first, the functional view, looks at the influences of activities on the environment. The second, the behavioral view, examines how activities will occur. Finally, the enactional view studies what must be done in order to make activities occur. He then went on to examine some desirable properties for describing software process formalisms, before taking a closer look at each of the three views of software processes. In the course of this examination, Professor Katayama presented his theory of ill behavior. In this theory, we ourselves are found to be the source of ill behavior, through our mistakes and unreliability. He expressed the desire to build a theory that can treat computations for formalizing ill-behaved processes.

After a look at the issue of process script preparation, which Professor Katayama feels needs more study, he raised the question of what we should do. Briefly, he explained that we need to learn from real processes, try to formalize difficult problems and ill-behaved processes, and remedy the lack of description of real processes for scientific study. In conclusion, he showed a diagram explaining how detailed processes should/could be described, examining the balance between environmental and human description.



Knowledge Distribution

Questions/Answers

Barstow:

Are there any Japanese companies doing studies of processes for real?

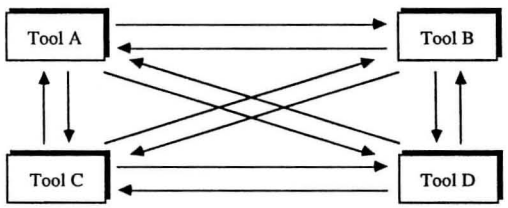
Katayama

I don't know, but it seems that some companies are trying to do that. I'm not watching companies in Japan, but my impression is that doing process studies is time consuming, and even if a person is interested in describing processes it is not enough. In my case, I think a great deal of enthusiasm is required. If a process is important and repeatedly used, I think that companies will pay for this study.

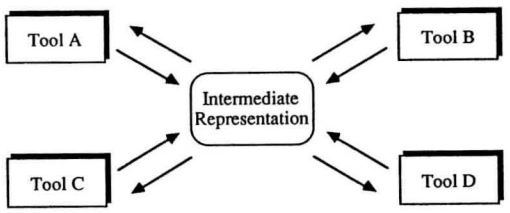
Dr. Lloyd Williams
(Software Engineering Research)

Dr. Williams gave a presentation on Object Management and Tool Integration as research issues for future software environments. His goal was to give an outline of a few of the currently interesting research issues, and to give some idea of how they are being approached by some projects. Focusing first on object management as a research issue, Dr. Williams broke this down into the two main research topics of type systems/information structures, and cooperation among users and tools. In a closer look at type systems and information structures, he pointed out that the current thinking is that OMSs will handle information within an environment as a collection of typed objects. He then looked at some requirements for type systems for software environments. He also examined several current research projects on type systems, noting that all of them deal with some variation on the entity-relationship-attribute (ERA) model. Moving on to a look at the topic of cooperation, Dr. Williams pointed out that in a software environment, information is generally shared among a number of users and/or tools. He discussed the necessity of preventing corruption by simultaneous writing to the same data, and looked at the traditional way of handling this problem through transactions. Again, several current research projects in this field were examined.

The other research issue that Dr. Williams dealt with was that of tool integration. He pointed out that this is a complex area, which is only recently beginning to be explored. Although there are three major aspects to tool integration (external, internal and orchestration), Dr. Williams chose to focus on internal integration, where the tools in an environment share information in a consistent manner. This kind of integration requires support in the three areas of information formats, semantics, and information transfer. After a brief look at intermediate representations as a possible solution to the problems of tool integration, Dr. Williams concluded with a look at some of the current research projects on internal representations.



Tool integration without Intermediate Representation

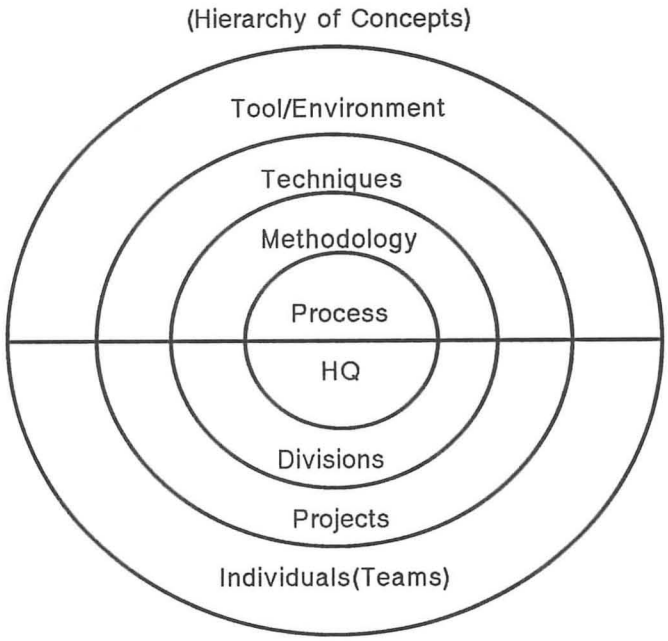


Tool integration with Intermediate Representation

Mr. Kouichi Kishida
(Software Research Associates)

Mr. Kishida gave a presentation on "Technology Transfer in Research and Construction of Software Environments." He began with a discussion of type A and type B software as presented by Mr. Laszlo Belady on the previous day. The main characteristic of type A software, as Mr. Kishida pointed out, is the clear separation of user processes and development processes. In type B software, the software, development process and the environment are all embedded in the user process. Therefore, Mr. Kishida drew the parallel that in B software systems, the environment is regarded as part of the organization, or corporate culture. He showed how a hierarchy of concepts can be mapped to a corporate culture. However, the outside world is important in order to make this organization work, as new events occur only in the outside world, while nothing new ever happens within the corporate culture.

Mr. Kishida drew an analogy with the city-states of ancient China. He showed how it is necessary to import new ideas from outside in order to make improvements, which corresponds to the penetration of a new environment based upon a new process model. In conclusion, Mr. Kishida presented a number of mechanisms that can be used to encourage "strangers" (people with new ideas) to enter the corporate culture.



Environment as a Part of Corporate culture

Discussion

Again, the discussion session for this panel was rather long, involving a number of foci. The discussion started by focusing on various concepts of tool integration, as it had been brought up in several presentations. Another discussion topic that came up several times was that of application domain knowledge. Next, the participants examined issues related to communication and co-ordination. The question was brought up of whether there is such a thing as a research process, as opposed to a development process, and a number of discussants talked about their own research process. This shifted to a look at requirements for research. The final main topic to be discussed was that of researchers' priorities and motivations, looking mainly at fun/interest versus funding.

Balzer:

In the discussion of the integration of tools, there was mention of several kinds of integration, and I was disappointed to not find my favorite form listed among them, so I want to find out why not. That form is what I would call "indirect." Instead of having explicit calls between different parts of the system, and doing integration that way, it's having the new part, whichever part is being added on, get there by integration through the activity which is already going on within the system, so that one identifies the activity of the existing part that is relevant to the new part, and for me it's "through the database." That you do integration by describing those kinds of mechanisms, and because the activity is already happening in the database, there is necessarily a language for describing that activity in a formal way. I just want to know how come you didn't list that as one of the integration possibilities.

Williams:

I don't think I meant to exclude that, and there's that whole category of things I called "orchestration," and there's buried in that set of issues things that I just didn't have time to talk about. Co-operation among tools in the sense that you suggested, where there may be some existing interactions and you bring in another tool to make part of that activity or set of interactions, would fall into that category.

Balzer:

Part of this is the question of what has been systematized, what kind of support is being provided for integration. And I think that that is a very valid question to ask of your environment, "how is it helping you to do integration?" One way of doing that integration is to actually change the thing with which you are being integrated. That's sort of the standard way. You'd like the orchestration to not be a co-operation between the two, but that the new kid on the block takes the responsibility for describing things in terms of the existing mechanism, and the orchestration happens, in some sense, without the knowledge of one of the parties that's being orchestrated. I think the systemization of that, so that you can systematically say "If I've got some set of capabilities that's now working together, I can add a new piece without having to have predefined the interface that's going to exist between those two things, because in general I can't define that, because I haven't decided what that's going to be until I know the new mechanism that's being brought in." It seems to me that this gives you a much richer basis for doing integration and evolution than exists where one has to define the new interface or one has to insert explicit calls in the old system. It's very different than just saying that there's a bi-lateral agreement between the two, because in this case all of the agreement is on the side of the part that's being added in.

Williams:

I guess I'm still not sure that I understand the question, since the way that I tend to look at this is that one looks for an intermediate representation that might describe existing tools and sort of is a standard for new tools, and so when you bring in a new tool, then it conforms to the standard. If it is an existing tool that wasn't developed for this environment, then you provide a mapping between it and what the environment uses.

Dowson:

There are many issues related to tool integration, and Lloyd is aware, of course, that he was not trying to cover the whole issue of tool integration. At least we could talk about data

integration, which it seemed to me that he was focusing on, activity integration, which links up a bit with what Bob [Balzer] was talking about when he was talking about invocation of tools and interaction of tools that may be implemented through the database but is not logical to do with data, and user interface integration. We could have had a whole workshop on tool integration, and there would be plenty to say, so I think it's not a fair criticism to say that Lloyd didn't cover everything. There are clearly many other issues to do with tool integration that need to be addressed.

Saito:

In the Sigma Project they first tried to find a common data structure, but they had no ideas about environments, so it failed.

Sayler:

I've been waiting all day for somebody to respond to Bill Curtis' article of a year ago, studying 27 large projects. He said there were 3 major problems that he uncovered. The first is the thin spread of application domain knowledge. The second is the volatility and contradiction of requirements, and the third is communication and co-ordination breakdowns. What do you see as the research objectives to answer his problems?

Dowson:

I think that that is a good analysis of many of the problems. We often underestimate the importance of application domain knowledge in software projects, and believe that simply by being good software engineers or good programmers that we will succeed in building good systems. That is obviously not the case. If there is not sufficient knowledge of the application domain that the system is going to operate in, I think it is inevitable that the system is of no use. The other problem that I'd like to address is communication and co-ordination breakdown. Some years ago at the first software process workshop, Ari Friedman observed that he had been studying software projects for many years, and that the single common reason why software projects went wrong was that people did not know what they were supposed to be doing. And if that is not a communication and co-ordination breakdown, I don't know what is. What do we do about it? Well, we have been talking a little bit about at least some of these issues throughout this workshop.

Katayama:

I agree about application domain knowledge being important. We need more detailed knowledge about domains so that we can successfully describe processes.

Williams:

I think, like Mark [Dowson], that a lot of these issues have been addressed already, in maybe not quite as direct a manner during the last two days. For example, Gerhard [Fischer] has addressed domain knowledge and its use in software environments. Bob [Balzer] mentioned the workshop on domain modeling, and I think there are some encouraging things going on in that area. Clearly, understanding and representing knowledge about an application domain is an important aspect or facet of an environment, but I think there probably needs to be some research done in the "methods" area to provide a little more background before we as environment people can begin to understand how to incorporate that. So in terms of the spread of domain knowledge, I think there needs to be more domain knowledge incorporated in the environment, but I think there's some research to be done there. In terms of requirements' consistency and volatility, Gerhard's critics are, I think, an interesting way to work on requirements' consistency. If you have some way of representing meta-knowledge about the domain, maybe that will help with catching inconsistencies or problems earlier. In terms of volatility, Bob's Genesis/Exodus/Reincarnation process model is a way to approach that problem. If requirements are volatile, it's very expensive when they change and you have to change the code. It's less expensive if they change and you can work on the requirements level. In terms of breakdowns in communication, people have been talking about process models and ways of handling process that provide some automated assistance for helping with communication. But I think we have to continue to recognize that a substantial fraction of the interactions will take place outside of the machine, and the question is deciding how much of that needs to be represented in the machine and how much can be left outside.

Kishida:

My observation is that if the software development process and supporting environment is

completely embedded in the user process, then the problem of domain knowledge, the problem of requirements, and the problem of the breakdown of communication processes are related to some very important aspects of the software development process. I think that a large part of the software development process is the communication and learning process, both on the side of the users and the side of the developers. We must communicate with each other, and the software developer must learn the application knowledge and the user must learn about the computers and software development. So, if the future software environment is successfully embedded in the user process, a very important aspect of the environment is the tool support for this two-way communication and two-way learning process.

Dowson:

Just talking about communication, I think that it would be a very great mistake to believe that we can turn the responsibility of communication between developers and users over to some environment, to do it all by electronic means. The environment can only supplement and help record human communication and natural interaction, not replace it. People must still talk to each other.

Belady:

We are analyzing and modeling and studying the development process to death. Okay, but if there is a development process, there must also be a research process. So, could you please talk a little bit about the research process, the process by which you perform your research in order to have a better software development process?

Williams:

My background is, in fact, in physics and chemistry, and so I'm very much instilled with the experimental paradigm. I like empirical research. But empirical research in software engineering is incredibly difficult to do. So what I think we end up trying to do more often is what the biologists do, which is observe and classify, and then try to generalize from that. My own approach is to do observation and classification and occasionally prove a theorum when I have to, and to do empirical research and come up with hard data to assess what I've done when I'm able to.

Katayama:

I don't really like this kind of question. Thinking about my research process, I think it can not be described. In my studies, I observe several examples and check them very carefully. Then I think about it when I sleep, and sometimes when I'm sleeping some idea might develop. I don't know why this works, but this kind of thing can't be formalized.

Belady:

I'd like to be fair and make the question simpler. My simple question is, "does research, as development obviously does, need early rough requirements? Does it have to start with requirements? If no, please explain why not, and if yes, please explain how that phase works."

Dowson:

I think that the point that Les [Belady] is raising is very important, that research, like any activity that we do professionally, is not immune to having to have objectives, plans, schedules, decision points about whether the funding should continue, evaluation of results, and all those other things. I think we are, in this community, very bad at doing that. Can there be requirements for research? Certainly. Those requirements will not be as precise as the requirements for an accounting system or something that will control, say, a train. But nonetheless, research should have clear statements of goals, and it should have statements of when we expect to achieve those goals. If we don't achieve those goals, then we should sit down and evaluate whether we've been doing the right thing and whether we should continue. It's very hard to apply that to oneself, of course.

Belady:

Where do the requirements come from?

Dowson:

The funding agency is one answer, but I think that there is a sense in which the world poses certain kinds of problems. We can choose to attack these problems or not. If we choose to attack these problems there are some implicit requirements that the problems create, and

our approach to solving them generates rather more specific requirements that we use to drive our research. I think perhaps that that's the same for all of us. But really, of course, the dominant influence is the funding agency.

Fischer:

I think I believe somewhat what people like Popper have figured out in their philosophy. Mainly, that there are problems in this world, and these problems should be addressed by our research. Now whether funding agencies sometimes share these problems or not is, I think, already a different issue. For instance, the military, which has funded a lot of research in the US does not always share my value systems, and they sometimes see very different problems than I see. During this conference a number of problems were articulated which I feel generate a lot of interesting requirements for research. One of them is the retraining problem. There is somewhat this old model of education that you go to school and university until you are twenty-five, and then you are done with your learning. This just does not hold in our world anymore, and so how to integrate working with learning I would define as a research problem. How we build software systems which facilitates this process, where you can integrate working with learning, in my mind generates a number of requirements which are very high-level and unspecific, but from there I would go on and derive more detailed questions. But in my mind, what really drives research is the problems which we want to solve, the problems which we face in this world.

Williams:

I agree with Gerhard, and the things that tends to drive it is the problems. Where the requirements come from is that because of our own interests, abilities, backgrounds, and so on, we see aspects of those problems which we think we can address, and those lead us to, in the sense of the classic scientific method, pose a hypothesis. We then proceed to test that hypothesis in various ways, possibly by doing a prototype where we try to integrate a couple of tools. I think that's maybe an overformalization of the way that people actually do this, but the requirements come from identifying a problem, seeing a particular aspect of it that we can address, and setting ourselves a goal for addressing it. I think where we, as researchers tend to fall down more, is in making part of our requirements a way of evaluating the outcome of the research. We tend to see an interesting problem, build a prototype system that addresses it, and then leave that prototype system aside to go on to the next problem, and not really do a solid evaluation, either empirically or even qualitatively, of what we do.

Dowson:

I think that one word that Lloyd used is very interesting, and that was the word "interesting." He talked about interesting problems, and very often as researchers we attack problems because we believe that the process of trying to solve them will be fun. And I think that's a very important motivation, and a perfectly respectable motivation. Acquiring knowledge is an activity that is important and justifiable in its own right. We shouldn't do it only, perhaps, for the sake of acquiring knowledge. There are real problems in the world that we'd like to solve, but it's a good thing to acquire knowledge, and to have fun doing it.

Kishida:

My observation is that there are two types of researchers, type A and type B. For the type A researcher, the requirements come from the environment, such as the funding agency or some kind of customers. For type B researchers the requirements come from inside himself.

Belady:

I like that it came through in the answers that it has to be problem-directed. What I did not hear, though, and yet I believe in, is that I think that researchers should be driven by the problems. My point, though, is that we should go a little bit beyond that. We should go after the requirements and the research process itself, and should start with a conscious and perhaps even planned effort to gather the requirements. Requirements will not come to you automatically. There are so many problems out there, and if you do it in a random fashion you may work on a problem, but it may be a very insignificant problem. So my plea to the research community is just that, that while I agree that you have to have fun and interest in the thing, plus you must have the ability to work on the problem, there are too many problems, so make an effort and make a value judgement not only yourself, but with the development community. You and the development community together can find a better

priority list and go for the high priority problems, not only the low-priority problems.

Williams:

I agree wholeheartedly, and I think that as a group that is a serious problem. We just don't have a good sense of priorities in our research agendas.

Teramoto:

My question is about the extent of the subject of research, and my question is, for example for Mark, about the incremental planning. You said, "how do we dynamically refine the project plan without violating the global, that is the resource or schedule, constraints?" From the standpoint of the management view, we have observed some projects, even though they are successfully using resources and schedule, make some trouble in the quality. This kind of problem should be linked to satisfied quality, or satisfied the motivation of the person who participated. But I think it makes the subject more complicated. But the software itself is satisfying to the user. But I think such kind of approach, to take into quality, or such kind of values is needed to select such kind of subject for even the research area. How about that?

Dowson:

I'm not quite sure how to respond to that, because I think you were really agreeing with the importance of the area of study. One of the things that I would really stress is that there are successful, well-managed software projects. Perhaps not as many of them as there should be, but that using some tools, perhaps quite good individual tools it is possible to run quite a large software project quite effectively, to adapt to changing circumstances, to do incremental and reactive planning. All the things I said were, to some extent, research issues. Human beings do it quite well. But what I think we're trying to do is two key things. One is to reduce the almost complete dependence on very highly skilled managers, because there are not enough of them, and also to, by representing some of this information, provide a path for systematic improvement of the process. And those process improvements will include the kinds of improvements that Bob Balzer's talking about in the development process, as well as the managerial process. Another thing that I would say is that issues like resource allocation are actually very important and quite difficult.

Torii:

The word "environment" has been discussed in this symposium, and I would like to know in a little more detail the images of future environments. First I would like to ask you what is the "environment" by your definition, and then afterwards I would like to know more about the research topics in environments. And by the way, not only the discussants. This meeting is very informal, and we have a very important chance to hear from some people who are very famous in the AI world: Gerhard [Fischer], David Barstow, Bob Balzer. So, maybe they would be kind enough to answer from their view, the AI views, what the environment should be, especially what kind of topics should be.

Dowson:

I think certainly the view that an environment is where you live, where you work, is entirely correct. Part of that environment for computer professionals is the computer system that they use as part of their work. We live in it, in part. It's not everything, but an important part of where we work is what we get out through the keyboard and through the screen, and I think our objective is to make that a more pleasant and efficient place to live and work.

Fischer:

I think that there was one characterization of artificial intelligence, and that was that is was a losing discipline, in the sense that the problem was not understood. It was usually regarded as AI. Some progress has been made towards understanding this problem, it sort of moved out of the AI area and got picked up by other areas. So in some way, leaving exaggerated claims aside, I think AI has tackled very poorly understood problems, and therefore has encountered many failures, which I wouldn't consider necessarily to be a negative attribute. But because these problems were so poorly understood, I think AI people very early understood that you wanted to have a powerful environment in which you could test your thoughts. So I think that the notion of a powerful programming environment supporting you in your thinking, amplifying your thinking power, tracing out some of your assumptions, got a lot of early attention within AI. So I think that some contribution of AI

has been the notion that an environment is very important, and that has carried over into other domains, and now I think all people would agree that environments which share some of the cognitive burden of tackling complicated problems are of crucial importance. So I would say that AI tackling poorly understood, ill-structured problems has shown us some way of tackling those, and what the role of environments can be to them.

Barstow:

I'm not quite sure how to say this, but one of the things that I've found about what I've been doing is that I've become less and less worried about AI, and more and more worried about software engineering. To me, AI is a set of techniques and a research program that may produce more techniques, and the question is, how do we use those techniques as some of the tools that we can use to build environments. And the particular technique that to me seems most useful is the set of techniques related to knowledge representation. Primarily for representing knowledge about the domain. If there is a thing that we should try to get from AI, it's what are the right techniques to use for representing knowledge about the domain, either as a kind of specification language or as a requirements language or something like that. And that's where we are likely to find some good tools. There may be a secondary area that might be in representing knowledge about design decisions. How do you represent the fact that a decision has been made, or what the reasoning was behind that decision? But those are the two areas that to me seem like they could benefit the most from AI techniques.

Audience:

I need advice. I think software environments, presently and before, are only for development engineers and software engineers. But in the near future, everybody will have to make programs, I think. For example, reception ladies and drivers. Everybody will have to make programs, I think. So I think that software environments will have to be not only just at the engineers level, but at many kinds of levels. I think we will have to supply environments, like cutting baumkuchen [a kind of cookie cut with concentric rings, like the trunk of a tree]. What do you think about such thoughts?

Williams:

I'm not sure that I agree that everyone will eventually become a programmer. On the other hand, I think it was David [Barstow] that first mentioned spreadsheets as a vehicle for people who are not necessarily professional programmers to create programs within a fairly narrow application domain. And I think that's a very useful way to proceed. Gerhard [Fischer] is working on construction tool kits, which I think will maybe show the direction of the next generation of spreadsheet-like software. My own feeling is that if Gerhard's work is successful, that software engineers will become more concerned with constructing the tool kits themselves, and less concerned with constructing the end applications.

Fischer:

I would add to this that software engineers, for the reasons we've discussed, namely that there's a thin spread of application knowledge, are not well equipped to do the second part, to build the end application. At least, definitely not by themselves.

Kishida

I think that the environment is a kind of device for some organizations or companies to do their business. So in other words, I think the environment is the implementation or representation of their view of their own business or process. So the environment is the representation of their own process model.

Katayama:

I am a little surprised to hear that thought, because I think our professionalism has to be established at that time, and I think that the environments should be for professionals. Professional environments should be for professionals, and I don't like to imagine that every person is a professional.

Belady:

Earlier we talked about the research process, not the research content of this panel, and the panel members from earlier today. I don't know whether it is only my perception, but it seems to me that most of this research content was related to the environment, and its modeling and management, and the process itself. Almost all of the speakers talked about

research problems related to the process which takes place in this environment. Is this a correct assumption?

Williams:

In terms of the things that I was discussing, I would categorize them less as process, although they certainly support the process. But in fact the way we've structured the SDA project there is a process aspect, an object management aspect, analysis aspect and so on. But the way that I would categorize my own work is in parts of the environment that deal with managing the objects, the structural aspects of the objects, not necessarily the software process aspects, although there is support for that.

Belady:

My impression was, and that's nothing negative, that it was mostly that which falls into the category of process programming, this new buzzword. You're not mentioning it, but express it in different ways. Let's assume that this is the case. What we call environments, software development environments, I remember we used to call differently, and that was about 15-16 year ago. We called them "software development support systems." There were early efforts in this area, and it was a set of tools, and methods and so on. But then it was too early, so it did not succeed too much. Then I went to the sunken laboratory of Schlumberger around 1980. This was the very first time I heard this word in this context. It took me a while to absorb it. "Environment." I said, "What is this environment?" I immediately disliked this word, and I hoped, honestly, that it would disappear. Instead of disappearing, it became a household word. But to me it still means the "software development support system," which is giving software engineers individually, and now in groups, support for their activities by some machine.

Kishida:

I think the words "software development support system" represents some very square view of the world. I heard the word "environment" for the first time when I talked with some UNIX related people in the United States in the late 1970s. For me the impression is that the word "environment" is a loose thing, shows some loose view of the world.

Belady:

Maybe. I gave up on hating that word. But talking about the research content, at least my perception of it, the following occurred to me. Where technology transfer was mentioned, the best way to have technology transfer is that people move and transfer the technology. Another thing was that small teams, and that was Bob [Balzer] who mentioned it, work better than larger organizations. It occurred to me that there is some commonality in it, and that this is called "externalization." And I think teamwork is more suited to it. People have some sort of easy ways to work with each other, because they do not have to externalize their inner thoughts and broadcast them in communication with movements. I think the trouble started with computers, because more and more we have to externalize, and what we in fact do is nothing else but externalize "stuff," which we developed over hundreds and thousands of years, and try to quantify that "stuff" and put it on the computer. We've been doing this for the first few decades, but now we do it at another level. We try to quantify our organizational behavior beyond the team. So now we attack problems where the informal team is not enough, and right now we are discovering that it's not enough to quantify some already well-quantified process like mathematical things and accounting procedures, but also how people work with each other. Namely, we have involved the computer in coordinating the co-operation and communication of organizations, and make it more efficient. But in fact we have to externalize all these developments and other processes in order to be able to pass it on to the computer, and this is why it's an interdisciplinary effort. This is why behavioral scientists and organizational scientists have to be involved. One final comment, not related to the previous one, is that you were proud to identify a word, "reification," as the inverse of abstraction. So since we talk so much about integration, we should somehow introduce also the notion of differentiation.

Dowson:

I'm very happy to accept that characterization, and I think it's one really that Bill Riddle was referring to earlier when he talked about a balance between discipline and chaos as being the requirement for what we should be trying to achieve. Too much chaos and we don't achieve the objectives that we want to achieve; too much discipline and we are kind of

strangled by bureaucracy, and it's achieving some appropriate balance by some kind of constrained co-operation that is clearly important.

Balzer:

I wanted to go back to the comment about where we really should be heading, and why we're doing all this stuff. It seems to me that the question that came up by John about the relevance of the Bill Curtis triage of questions, the one that seems to me is the most compelling is this question of the thin spread of application knowledge. That's not one that is about to go away. It's going to be with us always, and what we need is some way of capturing what people do know, so that we get more power out of it. The comment came up before about, essentially, end-user programming, and can we build environments that support that. And I think the answer is "no." But what we can do, perhaps, that solves both those needs, is end-user modification, where we use the scarce resource of the people who do have the application knowledge in a deep way, to build an architecture that's domain specific, that captures a lot of that knowledge, and then have more people do adaptations of that, because it is much easier to modify once you've got a base template that you can see and make comments in terms of. So the idea of using the scarce resource to create the architecture and then let end-users do adaptation of that is, it seems to me, a very powerful notion, and one which captures a lot of the kinds of research issues that have been addressed here. That in order to make that successful, there's an awful lot of technology that has to be put in there that has to do with how you architect those systems, how you leave them open for later modification, in what form you capture the information about both the domain and the computer optimization of those systems, and how the people who are going to be doing that modification, that is the end-users, interact with such a system to cause those modifications to occur. So it seems to me that this is one of the grandest challenges that we as a community really have to face.

Williams:

Something in what you said, I'm not sure whether we agree or disagree, so let me try and rephrase it. You said "create architectures within a domain and then allow users to modify them." And the way that I would phrase that is that "architectures within an application domain are a mapping of some end-application specific knowledge onto some implementation specific knowledge." It seems to me that there is a missing domain there, which is the transformation. There's knowledge in mapping the user's application onto a user's environment, and that's another domain of knowledge that needs to be accounted for.

Balzer

The reason I used the word "architecture" was that it is not just a conceptualization of the knowledge in some domain, like let's say aeronautical design, rather also a set of techniques which lead to the efficient computer processing of the kinds of things which are important in that domain. That's something that I don't expect domain experts in that domain who are not software engineers to be able to do. I mean that is, in some sense, the value added that this community has, is to know how once some knowledge about a domain has been stated in a formal way and is understood, how to apply computer processing efficiently to that. I think that the combination of those two is pretty hard work. For instance, we've learned a tremendous amount about business transaction systems, so that we can get very high data rates through those systems. Now that isn't anything particular about the world of banking, but it is about how you employ data processing towards those kinds of systems. The work has to be done first to build those kinds of architectures, so that as a new banking law goes into effect, a banker could add that law to the system, and with the research agenda that we're talking about, have the system adapt to incorporate that without having to employ new programmers to make that adaptation. But the initial architecture, which includes both the domain knowledge and the computer science knowledge of how to efficiently process that, is in my mind, going to have to be hand constructed.

Barstow:

I just want to make a quick observation, and that's that if one looks at software economics models, most of the cost of a piece of software in its lifetime is spent during maintenance and evolution, and yet most of the presentations here have implicitly been looking at development, rather than maintenance and evolution. There were a couple that mentioned the word "maintenance" and a couple that talked about the relationship, but I wonder why

that is, and one reason that might be why so few explicitly talked about maintenance is a perception that the focus should be on development. And another reason might be that there's a theory that says that if we can capture development and understand that, that whatever is done there, whatever knowledge we end up representing or whatever representations we have, will then carry over to support maintenance. It's just an observation, and I was curious about why things turned out that way.

Saito:

Before ending the session, I'd like to add another research topic as the chairman's right. I attended the second international workshop for System Configuration Management held last October near Princeton University, and version control is not exciting topic, but it's a very fundamental background technology. Version control is much related to product and other discussion is much more oriented towards process, but we need to consider more about the product itself. Maybe version control is included in the object management research, but it's a very important technology to support the software development of software environment.

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.

Call for Papers

1st International Conference on the Software Process

21 – 22 October 1991, Los Angeles, California, USA

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

Program Committee

Bob Balzer	Bill Curtis	Gerhard Chroust
Jean Claude Derniame	Mark Dowson (chair)	Watts Humphrey
Gail Kaiser	Takuya Katayama	Kouichi Kishida
Manny Lehman	Lee Osterweil	Maria Penedo
Dewayne Perry	Bill Riddle	Win Royce
Wilhelm Schaefer	Koji Torii	Colin Tully

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 not submitted elsewhere, 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（SEA）入会のおすすめ

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，1900 名近くのメンバーを擁するにいたりました．法人賛助会員も約 100 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．

個人正会員の入会金は 3,000 円，年会費は 7,000 円（入会時の払込は合計 10,000 円）です．法人賛助会員は，年会費 1 口 50,000 円です（入会金はなし）．会員には，毎月機関誌（A4 判約 30 ページ）が配布されます．また各種のイベントには，すべて会員価格で参加できます（法人賛助会員会社の社員は，会員扱いとなります）．

入会ご希望の方は，下記の申込書に必要事項をご記入の上，郵便または FAX で，事務局までお送り下さい．折り返し，会則，機関誌最新号，会費振込用紙などをお送りします．

申し込み先

〒 160　東京都新宿区四谷 3 - 12 丸正ビル 5F　ソフトウェア技術者協会

TEL 03 - 3356 - 1077,　FAX 03 - 3356 - 1072

.....

SEA 入会申込書（正会員）91-01

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

年令 ＿＿才　性別（男　女）　血液型（ A　O　B　AB ）

勤務先名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

所属・役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

勤務先住所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

勤務先 TEL：＿＿＿＿＿ - ＿＿＿＿＿ - ＿＿＿＿＿（内線 ＿＿＿＿＿）

勤務先 FAX：＿＿＿＿＿ - ＿＿＿＿＿ - ＿＿＿＿＿

自宅住所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

自宅 TEL：＿＿＿＿＿ - ＿＿＿＿＿ - ＿＿＿＿＿

連絡先（どちらかにチェック）　□勤務先　□自宅

.....

SEA 入会申込書（賛助会員）91-01

会社・団体名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

代表者氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿（ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿）

連絡担当者：＿＿＿＿＿＿＿＿＿＿＿＿＿＿（ふりがな：＿＿＿＿＿＿＿＿＿＿＿＿＿＿）

所属・役職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

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

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

申込口数：＿＿＿＿＿口

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