N ASA-CR-189349

SOFTWARE ENGINEERING LABORATORY SERIES SEL-92-004

PROCEEDINGS OF THE SEVENTEENTH ANNUAL SOFTWARE ENGINEERING WORKSHOP

DECEMBER 1992

NASA
National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1992	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE Proceedings of the Seventeenth Annual Software Engineering Workshop	5. FUNDING NUMBERS 552
6. AUTHOR(S) Software Engineering Laboratory	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Branch Code 552 Goddard Space Flight Center Greenbelt, Maryland	8. PERFORMING ORGANIZATION REPORT NUMBER SEL-92-004
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546–0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER CR-189349
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified–Unlimited Subject Category 61	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Seventeenth Annual Software Engineering Workshop, sponsored by the Software Engineering Laboratory (SEL), was held on 2 and 3 December 1992 at the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center in Greenbelt, Maryland.

The workshop provides a forum for software practitioners to exchange information on the measurement, utilization, and evaluation of software methods, models, and tools. This year, more than 600 persons from around the world attended the Workshop, which consisted of five sessions and a panel discussion, followed by a special tutorial session on starting an Experience Factory.

14. SUBJECT TERMS			15. NUMBER OF PAGES 434
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

Proceedings of the Seventeenth Annual Software Engineering Workshop

December 2-3, 1992

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

Proceedings of the Seventeenth Annual Software Engineering Workshop

December 2-3, 1992

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created to investigate the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Software Engineering Branch

The University of Maryland, Department of Computer Science

Computer Sciences Corporation, Software Engineering Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effects of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

Single copies of this document may be obtained by writing to:

Software Engineering Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

CONTENTS

Materials for each session include a summary of the live presentation and subsequent questions and answers, as well as any viewgraphs, abstracts, or papers submitted for inclusion in these *Proceedings*. Materials for the panel session include a summary of the presentations and of the subsequent discussion, as well as any viewgraphs submitted.

SUMMARY OF THE SEVENTEENTH ANNUAL SOFTWARE ENGINEERING WORKSHOP

The Seventeenth Annual Software Engineering Workshop, sponsored by the Software Engineering Laboratory (SEL), was held on 2 and 3 December 1992 at the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC) in Greenbelt, Maryland.

This workshop provides a forum for software practitioners to exchange information on the measurement, utilization, and evaluation of software methods, models, and tools. This year, more than 600 persons from around the world attended the Workshop, which consisted of five sessions and a panel discussion, followed by a special tutorial session on starting an Experience Factory.

In his introductory remarks, Frank McGarry reminded listeners of the unique nature of the SEL: it is a virtual organization combining elements of academia, industry, and government — specifically, the University of Maryland, Computer Sciences Corporation, and NASA/GSFC. The concept behind the SEL is to study various evolving software technologies empirically by applying and assessing them in a production environment; those that prove useful are then packaged in a published set of standards and in guidelines and training.

The Experience Factory exemplifies this process of continuous improvement. Many of the talks during the day, he pointed out, concentrated focus on certain aspects of the SEL as an Experience Factory; he called particular attention to the paradigm under which the Experience Factory operates: achieving continuous improvement by means of change management through an understanding of the strengths and weaknesses of software development technologies.

This strategy has the added advantage of giving an accurate perspective on ways of doing business. Anything packaged in an Experience Factory is driven by the understanding and assessing of the organization, not randomly or arbitrarily put together without reference to the way in which that organization actually produces its product and otherwise conducts its business. Developers' experience and perceptions of the strengths and weaknesses involved, McGarry stressed, must be the primary driver of the evolution of process, of the whole way of doing business.

As an example, McGarry cited the *Manager's Handbook*, "our Bible", as he called it, within the Flight Dynamics Division and within Computer Sciences Corporation. The *Handbook* sets realistic guidelines for estimating costs, developing software, and calibrating other factors of software development. This year, the *Handbook* has been supplemented by the *Recommended Approach to Software Development*, which is also based entirely on the experience gained over many years by a specific organization. These documents — continuously evolving and maturing as the organization's experience grows — embody the packaging activities of the SEL paradigm. These and other documents produced by the SEL, including the *Annotated Bibliography of SEL Literature* and the *Collected Software Engineering Papers*, are available on request from the SEL.

McGarry noted that the format of this year's Workshop would be somewhat different from that of previous years: this year, twelve papers were to be followed by a panel session and, after adjournment, a special tutorial session, *Starting an Experience Factory*. The tutorial would consist of "Experience Factory Fundamentals" presented by Vic Basili, University of Maryland; McGarry's "Laying the Foundation"; and "Establishing an Experience Factory: An Example" by Dana Hall, Science Applications International Corporation. These presentations would be followed in turn by discussion and exchange of information between attendees and presenters.

In closing, McGarry reaffirmed the approach of the SEL: identify technologies of high potential, apply them and extract data in a production environment, and measure the effect of these processes on costs, reliability, and quality. In short, the SEL aims to understand *how* to do business as well as *what* to produce. He cautioned that, in keeping with this approach, the results of the efforts discussed in Session 1 should be tailored to fit the particular conditions of the user's environment.

Session 1

Experimental Software Engineering: Seventeen Years of Lessons in the SEL

presented by Frank E. McGarry, NASA/GSFC

The first paper of the session focussed on experimental software engineering rather than on process improvement or experience bases. Many of the principles that have emerged during the past two decades of experimentation in the SEL, McGarry said, are counter-intuitive, which points up the importance of the experimental process.

At Goddard, many kinds of project have supplied the raw material for the process. Beginning in 1976, analysis of effort and resource allotment dominated the SEL's investigations; subsequently, defect analysis, structured and other design techniques, and testing approaches occupied more and more attention. Still later, object-oriented design (OOD), Cleanroom, computer-aided software engineering (CASE), and other areas of interest followed.

During these years, the first goal was, again, to understand and develop a model of a particular area of interest. This model-building was followed by experiments and comparative studies to determine the potential effects of changing the model by applying certain technologies. Finally, successful changes were packaged as standardized policies and processes.

The packaging phase, McGarry pointed out, did not begin until about 6 years into the history of the SEL, by which time there was a thorough experiential understanding of the processes studied. Written standards, initially very elementary, have been continuously refined and expanded to capture further experience and understanding gained through more than a hundred subsequent experiments.

McGarry crystallized this experience into seven basic principles. First, he said, improvement is characterized by continual, sustained, and methodical change, not by technological breakthroughs. Setting specific goals of productivity and error rates, and searching for the specific tool that will achieve them, is a misdirection of effort. But the Experience Factory model does not restrict an organization from adopting new technologies; on the contrary, it would posture organizations to take advantage of any breakthroughs quickly and rationally when they occur, and, ideally, it would keep organizations from becoming dependent upon them and stagnating without breakthroughs.

Still, many particular methods are preferred by developers, who see in them potential for significant improvement. But, to allow sustained improvement over time, these methods must be grouped and integrated into "methodologies" such as Cleanroom and OOD. These, in turn, are combined with measurement and feedback mechanisms to constitute improvement techniques — such as process improvement, total quality management (TQM), or the Experience Factory itself. A specific goal is the final necessary element; it will determine the focus of the organization and, thereby, it will help select the appropriate methodology.

The second principle is that experimental data analysis must be addressed in a specific context. Therefore, the SEL does not believe that data analyzed outside of their own domain can yield meaningful results; there seems to be little point, McGarry said, in sharing data between domains.

Closely related to this insight, the third principle that McGarry enumerated is that the goal of experimental software engineering must be self-improvement, not external comparisons. Attempts to generate national standards, in McGarry's opinion, are a waste of time. For now, the focus should be on each individual domain. Perhaps, in time, standards generated individually may be synthesized into overall standards, but at present neither the SEL nor any other organization seems mature enough to generate even the national database that would be a necessary first step toward such an overall standard.

McGarry's fourth point was that the key to process improvement is not data collection or measurement, but analysis and application of experience. Lessons learned may be written, but few are read. The goal of reuse is *using* code, not just generating libraries.

Fifth, uncertainty of measurement data, and its fallibility, are facts of life. Experimentation has to be designed to accommodate those facts. Newcomers to the SEL, McGarry noted, are often enthusiastic about the extent and depth of the data available to the SEL, but studies aimed solely at manipulation of that data, without reference to the context in which the data were gathered and to subjective data available from the developers, can produce results directly contrary to actuality. McGarry cited one such study in detail as an example.

Sixth, experience packaging has to be handled by a separate organization, not by the development organization. Too often, McGarry said, when an organization undertakes experimentation, it delivers a series of forms to the developers, requires them to collect data, and mandates the data's interpretation and application. The software development organization does not have a high interest in such procedures; production priorities predominate. Although the insight was a long time in coming, the SEL has learned that the developers have the right to say to the analysts, "If you're going to ask us to fill out these forms and provide you with data, then we want you to show us, in some reasonable period of time, an improved process. We want you to build the tools and have training. Don't lay the whole burden on us." So combining the two functions, the SEL found, is the wrong approach.

Finally, effective packaging must be based on experience. Often, organizations choose outstanding people, those who have observed the process for a number of years, to write standards and policies. This is wrong, McGarry said; it has to be driven by the characteristics of the environment. All of the information, data and lessons learned, must be the principal factor in the policies and standards written for an organization. This is different from the practice of the SEL in the past, but, McGarry found, it is more effective to focus on the development organization, not the analysis organization.

In summary, McGarry said that the SEL's experience-based approach — the Experience Factory — does in fact respond to these lessons, in that it includes recognition of the domain and the context.

The Experience Factory: Can It Make You a 5?

presented by Victor R. Basili, University of Maryland

Basili structured his remarks by outlining five recognized approaches to improvement in the business of software.

Plan-Do-Check-Act is a framework in which the goal is to improve a single process model by means of a feedback loop; this cycle should result in products demonstrating the qualities of greatest importance to the organization.

The *Capability Maturity Model* developed by the Software Engineering Institute (SEI) sets as its goal the achievement of "Level 5", a state of constant process improvement. Assessment in this framework is by comparison against external process areas to determine whether the organization is achieving a certain level in reference to those processes.

Lean Software Development, derived from the manufacturing philosophy called "Lean Enterprise Management", mandates the elimination of all but the most necessary subprocesses in pursuit of process tailored for each particular product. The quality-improvement paradigm here is a matter of building, from the beginning, an organization that continually improves in relation to an evolving set of goals and an assessment of its status relative to those goals.

This, then, is distinct from the Capability Maturity Model in that it is internal: the questions are, "How is this organization doing relative to its own goals and products? Where are the problems, and how must this organization change in response to these problems?" In this sense, it has more in common with "Plan-Do-Check-Act" than with the Capability Maturity Model, in that it is product driven, not externally driven by an idealized view of process.

A quality-improvement paradigm such as the *Experience Factory* also has something in common with lean software development in that its goal is tailoring a process to a particular product, based on what has been learned about each part of the process. Such a paradigm consists of characterizing, setting goals, choosing the process, executing it, analyzing the process execution (with feedback to the execution itself), and finally packaging what has been learned.

The looped structure of the paradigm means that, on the one hand, an organization has to attend to software development through project development with reuse in mind — reuse not merely of code but also of models for costs and error rates as well — and on the other hand to packaging information. The problem-solving functions involved in project organization under this paradigm include decomposing the problem into smaller problems, instantiating solutions, and validation and verification. The experience packaging in the other half of the paradigm includes unification of solutions, redesignation of the problem, generalization, and experimentation, with an analysis/synthesis process paralleling the design/implementation process that occurs during the project-organization phase. The paradigm therefore embraces many activities, all integral parts of the software business, that are often not performed at all by anyone anywhere, in Basili's view.

Establishing the Experience Factory, Basili continued, means first putting the organization in place and establishing baselines independent of process and product. Measurement of strengths and weaknesses follows, providing a business focus and goals for improvement.

Selection of methods and techniques, experimentally evaluated, leads then to the definition of better processes and tailoring based on experience. The organization learns about itself in this process, discovering what works and what does not, and slowly begins to understand the relationship between process and product. As processes change, new baselines are established and new goals for improvement can be set.

Finally, Basili reviewed the features of the familiar TQM framework and proposed a set of questions to establish judgement criteria for a comparison of frameworks.

All of these frameworks have an evolutionary side — they all assume that the organization will improve and that no organization will know everything that it needs to know from the beginning. Some have an experimental side, too, assuming that purposeful study of process can initiate or accelerate learning. Most of these paradigms presume statistical quality control, adequate for production but not suitable for development. Most of the frameworks explicitly assume a production environment (because their basic techniques and methods presuppose a great deal of consistent data), and moreover a production line that is single and constant; they also assume that predictive models can be built on that basis rather than on a higher level of abstraction.

Many organizations have tried to take concepts like "Plan-Do-Check-Act" and generalize from them. But Basili maintained that, while improvement paradigms have the same parent — scientific method — they recognize that there are multiple models at many levels that must be understood.

Returning to the original question, "Can the Experience Factory model make you a 5?", Basili defined a Level 5 organization as one that can manipulate process to achieve various product characteristics. That definition mandates a process and an organization with certain characteristics, which Basili enumerated. The organization embodying these characteristics, he added, needs to pull itself up from the top, not push its way up from the bottom. (This strategy, he added, tends to align an organization's processes with the stages outlined in the SEI model.)

The idea is to start with a Level 5 organization, but not necessarily with Level 5 capabilities. The development process, the maturation of the organization, is driven by understanding of business, product, processes, goals and experience specific to the environment. This affords a good grasp of the particular job before one's organization; it does not, Basili cautioned, give one a grasp of truth in general, but then, he said, nobody makes a profit on truth in general.

Basili's final answer to the initial question is, "Maybe not." The reason for the equivocation, he said, is that the Level 5 model is not entirely clear at the moment. Conceptually, though, having a set of processes and technologies that have been selected and tailored will bring an organization to Level 5: a continuously improving organization that manipulates process to achieve various product characteristics. But, he cautioned, it is conceivable that questions will arise that are not relevant to the organization, which may militate against the formal achievement of the Level 5 rating.

In conclusion, Basili advised that an organization interested in securing the Level 5 rating and in pragmatic improvement may want to take both the internal and the external assessments. After all, he said, the question behind the Level 5 organization really is, "Do I do my business the best I can do — for my business?"

Impacts of Object-Oriented Technologies: Seven Years of SEL Studies

presented by Michael Stark, NASA/GSFC

Stark prefaced his remarks by recalling a statement made at last year's workshop: that OOD may be the most significant technology studied by the SEL to date. To explore the activity behind that assessment, Stark reviewed the history of the technology at Goddard.

OOD was first adopted at FDD, he said, in the expectation of appreciable savings and increased reliability. Also, thinking about objects promised to be a more natural, more intuitive way of structuring problems in the Flight Dynamics environment.

At that time, 20 percent was a typical level of reuse; now, as then, each project (whether the production of a ground-support system or an actual experiment) is treated as an experiment, as part of the Experience Factory. Each contributes at least data, he said, and many, like the Gamma Ray Observatory Dynamics Simulator in Ada (GRODY), contribute even more, such as detailed analysis of the process.

What really evolved? Between 1985 and 1992, many different OOD principles were used. The division started assigning OOD to more and more phases of the life cycle, and later concentrated effort on building tools and packaging a training program.

The classes of systems studied included early simulators (GRODY and others of the following generation); ground-support systems built in FORTRAN but using some OOD concepts; second-generation simulators built using the experience gained during production of the early simulators; and, finally, generalized system work now in progress.

The Multimission Three-Axis Support System (MTASS) was somewhat object oriented, in Stark's view; data enter from the network and are fed into the flight dynamics facility to be processed. What really changed, he said, is that the sensor models were looked at in an object-oriented way, combining the sensor data in files and the operations on the data in the interface packages. This allowed the development of subsystems that were reusable as they stood.

However, a specific front end had to be built for each mission; the architecture as a whole was not reusable. The developers were not using formal OOD at first; they simply had a new idea about organizing the sensor models. Learning from concurrent work in Ada, they adopted OOD techniques to clarify their own conception of the design process, most notably in the area of combining functionality and the data on which it operates — a basic concept of OOD.

Early Ada simulators, beginning with GRODY, benefitted from experience gained in academic investigations by Seidewitz, Basili, and others, and from work in the commercial sector. The GRODY project began with formalized training on many kinds of design methods: Grady Booch's OOD, process-abstraction methods, stepwise refinement, function design, and others.

OOD emerged as a clear favorite, Stark said, but the existing processes were somewhat immature, not quite adequate for developing production-level design diagrams or design notation. Therefore, while GRODY was being developed, Stark and co-author Ed Seidewitz developed General Object-Oriented Development (GOOD), a methodology applied in full on the Geostationary Operational Environmental Satellite (GOES) project.

These initial simulators were also shaped by the Division's examination of Ada as a technology. The emphasis at that time was not on producing a real-time system, but rather on the study of emerging technologies of promise. The resulting systems, consequently, were somewhat disappointing in speed, but performance studies conducted by SEL researchers showed that the slowness resulted neither from Ada nor from OOD *per se* but from the developers' inexperience with these technologies, which impaired the technologies' full expression in the designs.

The next generation of systems, a set of telemetry simulators, was designed as a second generation succeeding the Gamma Ray Observatory (GRO) and GOES projects. The main difference in concept was that the design was built around Ada generics, with specific functionality added as needed. Even the specific modules, though, were built around Ada generics; consequently, the code size was cut by about 25 percent (a telemetry simulator for GOES ran to about 90,000 SLOC, but the one for the following project, with similar functionality, included only 69,000 SLOC).

However, the new generation of simulators encountered new limitations. They were also built for three-axis-stabilized spacecraft, which mandated considerable reworking to make the systems suitable to spacecraft stabilized in other ways. This shift to a new set of problems motivated developers to think in terms of generalized system development and generalized domain analysis; these techniques promised new ways of accounting for the variation in problems and enhancing reusability.

Until this time, Stark said, developers had not used certain OOD programming features, such as inheritance or polymorphism; they depended instead on the idea of abstraction, most of which was provided in these systems by Ada packages (in the FORTRAN systems, abstraction had been largely a matter of combining function and data, with no distinct construct like the package).

These second-generation telemetry simulators were also the first designed from the beginning to be a reusable architecture. There were problems with complexity and transfer to new problems, but this generation established the feasibility of reusable architectures in this environment. Again, the limitations encouraged domain analysis; and this time, the analysis really examined the entire way in which the FDD does business.

The major change from the second generation to the generalized systems was that OOD was extended into the specifications phase. The major change in design, Stark said, was that classes were implemented as abstract data types, with generics used only to parameterize dependencies between different classes instead of being seen as templates into which models were fitted. But the key point, he stressed, is that this stage of development was marked by an effort to extend OOD into all phases of the life cycle.

In sum, the impact of the technology is twofold, Stark said: the extension of OOD into all areas of the life cycle extended reuse concepts in the FDD, and specifications became more general. The benefits — increased productivity and reliability, reduced cost per line of code and project duration, and so forth — increased as reuse increased from the initial 20 percent to the current 80 percent.

Now, Stark concluded, the FDD is working to extend this message to a wider range of projects within the division, a process facilitated by the lessons learned to date.

OOD is, in Stark's opinion, indeed the most influential method investigated to date in the SEL. It demands a new way of thinking about problems; it has been applied to all phases of the life cycle, influencing not only coding but the Division's entire way of doing business. It has facilitated reusability and reconfigurability, which in turn have increased productivity and yielded other benefits highly attractive to management.

Discussion Following Session 1

Question: Once a packaging is completed and a standard established, how do they get incorporated into the culture of the organization?

Basili: In a variety of ways. Michael [Stark] raised that issue, as a matter of fact, when he talked about the education. As we learned more about OOD, it just became part of the next training process; that's one of the things that happens. It changes how we teach what it is we're doing. It becomes part of the recommended approach that's used when people are building the software. It just permeates every aspect of what we do.

McGarry: One other point: people always say that we're lucky here in the SEL, but I happen to be the boss of both organizations, so what happens is that when we package something, it does become an adopted set of standards and policies within the organization. But management has to buy off on this, if you do develop training or standards or whatever. Management has to believe in it and incorporate it into the development organization.

Question: When you switch systems, say from state-transition-oriented systems to operating systems that are heavy database and user interface, you tend to customize process. And it seems that there's a little bit of tension with organizational maturity and sameness across an organization.

Basili: I agree with you; I think that there's a problem. But I think that if somebody looks at two different products and says, "Gee, you're doing that differently; how does that affect your organizational maturity?", I'd say, "We're really mature because we recognize that you don't build different products in the same way."

Question: Basically, with the Experience Factory, what type of people actually do the analysis? Do you have people with software backgrounds, or engineering backgrounds, or —

Basili: Some of us are from a project organization; we move across lines. Mike, for instance, is sometimes in a project organization, and sometimes he's in the Experience Factory, depending on what role he's playing in a particular thing. These aren't a separate group of people who are off to the side; people who worked on a particular problem are the ideal packagers of that information, because they'll have a deep insight into what happened with it, and they can package it and then go back and use it again.

Question: What are the specific examples of packaged products? Are they on-line databases?

McGarry: As I mentioned in the opening, they're what are being handed out here: the *Recommended Approach*, for instance, is a packaged product right there. It's a set of standards that we follow.

Basili: What this means is that somebody didn't just write this as a neat idea. Everything in there has been used and evaluated, and it's effective, and people think it works, and it's OK, so we can now say that it's worth using.

Follow-up: So they're primarily papers?

Basili: They're also databases: databases full of information, databases full of models. It's also papers, training materials — it's all of that, put together.

Question: What's a Level 5 organization without Level 5 capability?

Basili: What I meant by that is that, with regard to Level 5 capability, you're talking about your assessment relative to a set of key process areas. What we're saying is, "You're not that." You have to have the physical organizational structure in place to pull you up, to do that kind of thing. You're already asking the right questions; you've already set yourself up to say, "What is the right process to be using? What might I manipulate here?"

Even though you don't know all of the answers, you're starting to do that. Rather than saying, "Let me just define a bunch of processes for a while, and after I define them, we'll decide how we use them." This says, "No good: let's use them, see if they're any good, and then we'll start to define them. Then we'll start to package them." So, in fact, a lot of information can be gathered just by observing how an organization does business. People are probably doing reasonable things; can't I just package what they already are doing and analyze it, and then you hone it from where it is, rather than just throwing in something new. Change is not easy for people.

So the issue is to take whatever you're [thinking of] doing and try to fit it in with what exists. One of our prime philosophies, with regard to organization, is that if you don't understand what you're doing now, you couldn't possibly improve it.

Question: First of all, Vic, have you packaged the Experience Factory so that others can use it?

Basili: The answer is yes, it is packaged so that other people can use it, and part of that is the tutorial that we'll be doing tomorrow. Another part of it is that we are trying to move it to several other sites within NASA, which we can do because it is packaged.

Follow-up: Frank, benchmarking is important, and yet it runs counter to several of your themes. Could you comment on that, please?

McGarry: We talked about that, and several of the people who went through our draft materials made that exact point. I'd say that it is an acceptable approach but must be used very carefully. The domains of the process can be different, but the product has to be the same. If you're trying to benchmark something, to look at some error rate on some product, if two different processes are used, then you possibly could compare those — as long as you know that the products are identical.

Basili: Another view is to benchmark as you reach higher levels of abstraction. If you do it as a low-level data issue, it's more complicated.

Follow-up: This is counter to a lot of the things that ASQC [American Society of Quality Control] and ISO [International Standards Organization] are recommending, in the sense that they say to do it within a domain and then do it across other domains; then do it across industries and compare yourselves to world-class organizations to lift yourself up.

McGarry: I would say, ideally, that that's a good thing to do, but we're a *long* way from being there. And, unfortunately, too many of us are attempting to do Level 5 before we do Level 1.

Question: I have a question about the Experience Factory. Doesn't it sound a little bit ironic that we're saying a Level 5 organization is going to institutionalize what makes a Level 2 organization work? I mean, a Level 2 is the key individual and the experience of that key individual. And we're saying now we're going to take that experience, and rather than de-emphasize that we're going to emphasize massive amounts of individual experience plus organizational experience.

Basili: That's a Level 1 that's characterized by the key individual.

Follow-up: Right. Now, the Experience Factory — I'll have to come to your tutorial — helps generate more organizational experience?

Basili: What it would say is, rather than putting a process in place immediately, start out by saying, first of all, what are these key people doing? Where are you making errors? What processes do you have in place that other people can use? Then let's start from there and see what next step we can put together, relative to the problem with our product. Is it defect rate? Is it cycle time? What are the kinds of defect we're making? That will give us an insight into the kinds of technology to bring into place.

Question: What is the program for agreeing to be an experiment? I mean, if I'm a software manager, and I'm thinking, "Well, here's a new technology; I don't know about this. I'm more comfortable doing it the old way."

Basili: I would argue that every project, as it exists today, is an experiment, in a sense. If you knew what was going to happen, then it wouldn't be an experiment. We don't know what's going to happen. The thing is that we're putting some bounds on it, saying that we're at least going to measure it and help you set your goals and determine whether you've achieved them. And try to convince people that there are reasons to make that change.

Follow-up: So would you say, then, that the experiments are driven by the projects? I mean, there are new technologies that the projects need to implement—

Basili: That's a selling point, that they're driven by the need. I know that if I'm a project manager and I've got a problem, I can come to you and you can say, "Based on that problem, here's a way to solve it, and let me explain how it worked before (or it's just off the drawing board, which we do occasionally). Would you be willing to try it?" They're much more willing to try it, not because I can say, "Gee, someone out there says this process is neat," but because I can say, "This process relates to your particular problem. How about going and trying it?"

The interesting thing, about TQM, is that everybody here thinks that way. Early last week we started our fourth Cleanroom project, and we had a group meeting. I wish I'd had a movie of the discussions that went on there, with people saying, "Gee, how about getting us technology to help here, because on the last project we thought that this was a problem?" This is what you want people to start saying, and thinking that way. So it just becomes a way of everybody's thinking about how the problem gets solved.

Follow-up: So there's no selling process that goes on.

Basili: It sells itself, in a sense. Once people buy into it.

Question: One of your stated goals, up front, was to improve maintainability. Do you have any data to show that that's actually happening?

Stark: Unfortunately, the answer to that is no. The organization that has maintained the simulators hadn't been folded in to the data-collection process until very recently. The development organization and the maintenance organization were different, and the people who were maintaining the simulators just didn't fill out their forms. So we have no data.

Basili: But they are part of us now, and we have a project to do exactly that.

Session 2

The MERMAID Project

presented by John O. Jenkins, City University, London

Jenkins began by describing the lessons learned in the Metrication and Resource Modelling Aid (MERMAID) project, which was done under the auspices of European Strategic Programme for Research in Information Technology (ESPRIT). He acknowledged two previous speakers, McGarry and Basili, as having set the stage for his own remarks. Particularly, he emphasized again the importance of understanding, evaluating, and packaging, terms in which the MERMAID project was conceived.

In fact, ESPRIT emphasizes these same factors, Jenkins said; in its early years, the program's emphasis was definitely on understanding, but now, 9 years into the program, the emphasis has clearly moved on to evaluation and packaging.

The account of specific lessons learned in the MERMAID project, he said, was intended only as a basis for his explanation of the project as one that had benefitted from lessons learned by others. The MERMAID project set out to create an improved cost-estimation procedure and to implement it throughout the European software community. In addition to this ostensible objective, the team had no fewer than six other objectives, each held by one of the institutions participating in the project. Each of these objectives, Jenkins said, was in fact met by a specific part of the project's outputs, which consisted of the improved methods themselves, the dissemination of this information, and a number of tools (designed not solely for managers nor for the specialist estimator but for everyone on the software development team).

A particular focus of the project was the intractable problem of the interrelationships of effort, schedule, and manpower levels of projects within a given software environment. While lessons learned elsewhere were considered, Jenkins reported that the team elected to use their immediate experiences as the basis for their investigation of these interrelationships. Therefore, they used the systems dynamics techniques developed at MIT to follow the development process.

In this study, the team considered cost estimation as an integral part of the process of project management, not as an isolated activity. Their premise was that it is only possible to develop a model that describes a particular environment. The idea of a generic cost-estimation model, Jenkins said, does not appeal to the team; they maintain instead that good forecasting is based on a relatively simple, probably linear model that offers a consistent measurement regime. For that reason, the team initially turned to NASA/GSFC for assistance; they believed that NASA/GSFC data would indeed have been produced by a such a regime, as distinct from data available to them from local organizations.

Because of ESPRIT's emphasis on packaging, the team's statistically based estimation functionality is now packaged in a commercially available tool, as well as more freely in the form of R&D prototypes. The toolset integrates data collection and estimation functionality; the risk-assessment tool is not fully integrated, Jenkins explained, but nevertheless it is fully compatible with other functionality. He admitted that the approach has not been validated to his own satisfaction, but he believes that European plans to adopt the technologies are realistic and will open new opportunities for validation.

Jenkins pointed out that the statistically based estimation tool follows only one of the several methods that the MERMAID team developed, one that is only possible in the presence of a fair amount of data. The team has also dealt with other methods more appropriate to environments in which data is in short supply. Here, Jenkins characterized the team's approach as "case-based" reasoning or "single-analogy" reasoning, which requires some skill in identifying projects that are most similar and a useful set of measures for both similarity and difference.

Further work based on the results of MERMAID, and guided by lessons learned during that project, is now in progress.

Software Process Assessments

presented by Anthony Verducci, AT&T Bell Laboratories, USA

This presentation began with a definition of software process assessment programs as a way of determining the effectiveness of processes used in software production. In the course of his presentation, Verducci pointed out differences between the AT&T assessment process and the more familiar SEI process.

At AT&T, process assessment is used as part of an overall quality improvement program. Therefore, assessment has specific objectives, the first of which is the identification of the strengths and weaknesses of the various organizations within AT&T that his group assesses. Assessors also seek to baseline their organization against industry standards, to compare it to other organizations within AT&T, and, most important, to prepare a road map for improvement activities.

Assessments are run every 18 to 24 months, Verducci said, which allows adequate time between assessments for working on improvement; therefore, organizations within AT&T want assessments to articulate objectives very clearly. Two methods of assessment are used: first, the questionnaire associated with the SEI maturity model and second, the questionnaire developed by Software Productivity Resources (SPR), modified for use in the AT&T environment. These methods are integrated into a specific process.

In this process, three of the four full-time members of the corporate assessment team conduct each assessment. They meet with members of the organization to plan the assessment and to discuss schedules for data collection and feedback. During the actual assessment, the team administers the SEI questionnaire and the modified SPR questionnaire to managers, who fill out the forms by consensus. At the managers' option, the assessment team will work with the managers to explain terms and procedures, although most prefer to do this themselves.

The SPR questionnaire is filled out, usually, by the development staff alone. This dual approach gives a double perspective, the SEI information being taken from management, and the SPR information from the developers.

Usually, data gleaned from the SEI questionnaire are used to structure a series of interviews lasting about a week. Verducci's group, by contrast, uses information from the SPR questionnaire to validate responses to the SEI questionnaire. That way, they believe, a much more structured assessment process results, and, because the same questionnaires are used across the company, comparisons of individual organizations are possible. Also, staff hours invested in the assessment are reduced from the 100 estimated by the SEI to only about 25 to 30, in AT&T's practice.

Subjective data are collected by means of eight open-ended questions such as, "What do you feel are the strengths and weaknesses of your organization?" and "What would you recommend to your management (if you couldn't change your job or your salary)?"

The assessors then synthesize the results of these questionnaires and supply feedback to the project. The feedback session takes the form of a detailed presentation lasting perhaps two hours; all members of the organization are invited to attend, even those who did not participate in the questionnaire process. Top management is especially encouraged to participate; this not only facilitates management's endorsement of the process but signals that fact to the organization as a whole. One of the tools used is the kiviat or spider chart, which plots the organization's characteristics within a range of industry norms to identify high-risk areas.

Most important, the assessors offer a proposal that outlines tools, techniques, and other resources that can be used for improvement, to meet the challenges posed by perceived weaknesses. They also offer continuing assistance in the post-assessment improvement process.

The assessors find that their customers have varied expectations of the assessment process. Most are implicit before and during the process, Verducci said; only after the feedback session are they articulated, when the organization finds that its unstated expectations have been met. Many expect all commitments to be filled, so one of the assessors works closely with an organization coordinator to ensure that the organization receives a checklist of what is to be done, the schedule of activities, and so forth.

Organizations also expect the assessors to be independent of any particular organization, and, indeed, they are. Therefore, the assessors can maintain their objectivity, inasmuch as they have no vested interest in the outcome of the assessment. Managers, too, are much more amenable to an assessment by an outside organization, although, Verducci said, an internal assessment would probably arrive at much the same result.

The organizations also expect a report of strengths as well as weaknesses; the assessors are careful to balance their report to accommodate this expectation, and they support their assessment with actual examples, quotes from the SPR session, and other specific information. Confidentiality, of course, is expected and strictly observed. Verducci distinguished two levels of confidentiality: no comment is attributed to any one person by name, which is one level, and, after the assessment, the results are released only to the subject organization. Aggregate results are compiled and published, but these merely show trends across the entire company.

Finally, delivery of an action plan is usually not expected, but, Verducci said, this expectation is often retroactive on delivery of the plan. The feedback session, he said, often triggers the question, "Where do we go from here?" The action plan points the way to an answer, although some organizations prefer to develop their own (often using the assessors' plan as a starting point). In a few cases, the organization chooses to enlist an independent process engineer to work with the organization throughout the improvement process. This last option is especially attractive to organizations that cannot hire additional personnel for the improvement process but can employ an engineer as needed.

In conclusion, Verducci said that the assessors have rarely found an organization that was surprised at the results. Usually, he said, they have a feeling for what's right and what's wrong before the assessment begins.

The Role of Metrics and Measurements in a Software-Intensive Total Quality Management Environment

presented by Charles B. Daniels, Paramax Space Systems

When Paramax won a consolidation contract at the Johnson Space Center (JSC) in Houston some 7 years ago, Daniels recalled, the company was faced with the problem of assuming maintenance responsibilities for twenty million lines of code written in about fifteen different languages, inherited from eleven different suppliers.

The company attacked the problem by defining and documenting the processes and procedures already in place. After evaluating this (highly complicated) baseline and gaining an understanding of the environment's complexity, maintenance personnel examined the strategies that seemed likely to help bring order out of this chaos, developed productivity estimates, and set realistic goals. The next step toward achieving those goals was creating action plans against the existing baseline.

The test bed available consisted of mature software that had been operating in the NASA environment for as long as 25 years: mission-control software, the shuttle mission simulator, and similar systems. But the company also began developing new systems, such as the flight analysis and design system, and experimenting with new information-systems methods, some of which have been exported from JSC to Paramax's Goddard site.

Throughout these processes, Paramax has maintained its focus on primary requirements, principally achieving and maintaining reliability of the software that they maintain or evolve. Productivity is another primary requirement; efficient production of new systems and responsiveness to customer needs are also crucial factors, mandated by NASA's ambitious goals for future missions.

Daniels attributed the company's success to its view of software engineering as an endeavor with a dual focus: on the actual process of software development and on the processes concerned with the product — processes that help the company understand, define, and measure factors like portability, accessibility, testability, and maintainability.

Holding to this viewpoint, the company uses metrics as part of an integrated process, not a stand-alone and not as a silver bullet. The first role of metrics, Daniels said, is to determine the baseline in the state of process measured; then, one can track the progress towards improvement goals as time goes on. Most important, he affirmed, is the idea that the real purpose of collecting and analyzing the data is to make decisions; otherwise, data collection is useless, and the data collected are inert. Paramax therefore always aims to structure its metrics so as to facilitate decision-making on the basis of information received.

A specific set of guidelines directs this structuring. Data must be easily collected; otherwise, they might not be meaningfully aggregated for use in decision-making. Metrics should bear only one interpretation of what the data mean; they should be meaningful, offering a definite benefit in return for the effort of collecting and interpreting the data. Only important metrics should be collected, Daniels said. At first, the tendency is to measure every feature, but, over time, significant factors will emerge to the exclusion of the trivial. Further, only controllable factors should be measured. Those that cannot be changed, he observed, are irrelevant to any improvement effort. Finally, he advised, measurements have to extend sufficiently across time to constitute a meaningful perspective; a single snapshot of a process offers little insight into the actual nature of the process.

The process used is captured in a waterfall model, one in which the starting point is a clear definition of the process to be measured, including a precise view of the points in the process at which measurement is important. Then, on this basis, objectives can be set and action plans developed to reach them. Throughout the process, decisions are reached on the basis of data being collected.

The standard metrics used at Paramax reach from the lowest levels of the organization to the highest. This facilitates organizational communications, he said, and helps form a common culture to which all employees can relate. Further, the standardization of metrics offers a top-level summary of the company's progress, allowing management to understand quickly the state of the organization and of each of its elements.

The standard metrics are arranged into four categories, Daniels said. Work flow measures involve looking at software engineering processes and determining their efficiency. Productivity metrics audit the use of resources per line of code; measures of quality performance examine standard failures but also decompose them into procedural, requirements, and post-release error categories, as well as assessing backlog figures and mean time to repair.

Team building, the fourth measurement area, Daniels pointed out as being of particular importance. The "Team Excellence" infrastructure, he said, underlies every aspect of the company's software engineering, which is organized into twenty-one "centers of excellence". Each team tracks the standard metrics as well as a set of metrics unique to their organization. Each of these is displayed on a number of bulletin boards that give an immediate view of the organization's status and thus serve as a management aid.

One particular tool used in this environment is the objectives matrix developed at the University of Oregon; after a process is described, the matrix assists in the establishment of criteria by which the process can be measured and in tracking progress toward goals. One of the most useful aspects of this matrix, Daniels said, is that it delivers a number at the end of the day that gives a general sense of the current state of the software engineering process.

Daniels concluded by listing the significant improvements that have resulted from Paramax's approach over the past 7 years. Among these are increases in productivity (41 percent) and quality (from an initial 210 errors per million LOC to a current rate of somewhat less than 60), as well as a reduction in discrepancy backlog (down 51 percent). Plans for the future include re-engineering old code, introducing new technologies like application-specific languages and requirements-management tools, and continual improvement of the entire software-engineering process.

Discussion Following Session 2

Question: Is there any published information that offers more details on these MERMAID cost-estimation and risk-assessment procedures?

Jenkins: Well, obviously, there's the companion paper to this presentation, in the *Proceedings*, and I do have with me some additional information. I have a few copies, so if we run out, business cards, please! We'll get them sorted out and get the information to you.

Question: Is there a resource that we could read, rather than burdening you with the task of sending out the copies?

Jenkins: At the moment, the answer is no. While there have been a number of conference presentations of some of the ideas being developed in the project, there's nothing that I would call a fully comprehensive account of the work. However, we are working on it. Indeed, we have as one of the targets the special IEEE [Institute of Electrical and Electronic Engineers] issue on the European software engineering research. Of course, I suspect that every other project has that particular publication in view, and, if we don't make that one — well, we hope to find some other editor somewhere who will allow us to share this with you.

Question: In the AT&T work, can you distinguish between an "organization" and a "project"?

Verducci: At AT&T, an "organization" is typically two to three hundred people. Sometimes, an organization works on one project, or there can be five or six projects within an organization. A project is people who get together to develop a release of, say, some switching system software, etc. The projects we assess are typically about fifty or a hundred people, but we assess projects from five people up to two thousand people. That's quite a range.

Question: At Paramax, you've used what some of us would consider a very standard set of metrics for the process, and you say that you have applied this across the environment. Earlier this morning, we heard speakers who, I think, said that's not the right thing for us to be doing. How would you respond to that?

Daniels: Well, I noticed that, too, and I was hoping you wouldn't ask me that question! I don't think there's a particular conflict there, because even though we're a test-bed environment, we really have a lot of mature software in place, and we're in the process of evolving that software. So we're not talking about new project development in the sense that the SEL is, because they're bringing new projects up in a common baseline.

The other answer to that is that, in the organizations, there's an awful lot of unique metrics that we don't use. So we have a combination of both. We have the standard metrics that give us kind of the overall picture — and I think that one of the messages was that if you abstracted at a certain level, then that's not such a bad idea. But if you try to get down into too many details, then trying to home in on a single set of metrics is probably not good. And, if you noticed, in some of our metrics, they're abstract to some degree. They're not into the great details. In fact, we're doing some function-point analysis in some of our areas, and we're not doing it in others.

Question: Mr. Daniels, the errors per million LOC: were those discovered errors or estimated residual errors left in the code?

Daniels: For the most part, they were residual errors in the code. But we also decomposed that into errors that are inherent, errors that we induced, and errors within the different life cycle phases.

Question: Mr. Verducci, in your experiences, have you noticed any divergence between the results portrayed by the questionnaires and the actual results of the project, for example, schedules and so forth. I ask that question because sometimes [people who respond to] questionnaires tend to be intuitive, and they know the answers that you're trying to get to, and some of the fears that engineers have are about how the results are going to be used. So they might give you false indications.

Verducci: In fact, we never have. We make it clear to people at the beginning that no names are going to be used, and that the data will not be used against them. We can't guarantee it, but we have never seen a case yet where it has been. And, as soon as people get talking, after the first 10 or 15 minutes, they realize that they feel comfortable, and they talk about schedule slips, problems with defects — you name it, they talk about it. Once they get going, once you get engineers talking about problems, they just love to keep talking. We've never run into a case where we felt that people were actually covering up what was actually going on in the organization.

Session 3

Maximizing Reuse: Applying Common Sense and Discipline

presented by Sharon Waligora, Computer Sciences Corporation

Waligora's presentation took a broad overview of software reuse at CSC over the past 20 years. Reuse has been part of the company's business strategy — indeed, of corporate culture — for many years, she noted, but has seen considerably more emphasis in recent years, as demand has increased within the industry. Such programs, she said, do not come into being overnight, and their evolution is basically a management issue, a matter of applying good common sense and discipline.

Originally, reuse attracted CSC's attention on purely business grounds: it seemed that software could be produced more cheaply if portions of it were reused. In this regard, Waligora said, reuse at CSC has proved notably successful, as shown by the steady decrease in costs shown in SEL data. However, early reuse efforts were entirely dependent upon individual developers. Each produced code that could be reused, but few shared reusable code, and attrition could cause major setbacks in reuse efforts. Again, good business sense mandated the capture of the information that would ordinarily be lost to the company when the individual developer left.

The expanding scope of NASA's activities in the early 1970s only heightened the urgency of this kind of systematization. Ten ground systems were required from CSC, on a 6-year schedule, and the company had relatively few experts on staff at that time. It was clearly imperative to focus on incorporating people's knowledge in products.

The creation of a library of utilities was the first step in this early reuse program. Common utilities like routines to calculate Sun vectors or to perform a coordinate transformation were the early candidates for inclusion in the reuse library, but even the easy availability of such reusable components did not capture the level of knowledge that had to be conveyed from one developer to another. Therefore, Waligora noted, the company began building reusable packages, larger functional units like orbit propagators or telemetry processors. These packages were not just code; they involved documentation and design diagrams as well, and the code itself always needed to be modified for each specific mission.

This kind of reuse facility encapsulated the knowledge about the application domain at an appropriate level for transfer from one person to another. In fact, the rapid growth of staff that the company experienced in those years was made much smoother by the availability of package libraries. Owing largely to the viability of these early reuse strategies, the challenges of producing ten major systems in only 6 years were met. Still, the *how* of doing business had yet to be captured in a similar way. The next focus, therefore, was process itself.

First, the company sought to describe and understand the process for design, coding, and testing as it existed. At that time, the process was communicated only by word of mouth, and it was followed to the degree chosen by the project leader. Over the next several years, the SEL worked to formalize the process through documentation like the

Recommended Approach to Software Development and the *Manager's Handbook*. This documentation specifically addressed development processes in the Flight Dynamics Division; at the same time, CSC was producing its own procedural document, the *DSDM (Digital System Development Methodology)*. Both of these took their cue from an examination of what was actually being done in the organizations, taking careful note of lessons learned and sorting out what worked and what did not work; practicable techniques were systematized and recorded in the documents.

Both of these documentation efforts also addressed the management side of software development, Waligora noted. With these methods in place, the technical processes could be reinforced by the management practices that had been developed concurrently with them in the production environment. While these procedures were being developed, NASA was changing its business philosophy for the future. Multiple small missions were to be consolidated into fewer, more complex missions that would demand fewer, bigger, more complex ground support systems. But at this point the "budget crunch" shrank the resources available to build these growing systems, a combination of factors that spurred the creative drive for more efficient methods of production.

SEL studies during these years showed that reuse was most profitable when code could be reused verbatim. OOD, Ada, and other approaches were investigated in this light, but the functional packages already in use were also examined to determine why so many modifications were needed before they could be reused. The problem was rooted in the way that the specifications were written: because they were so specific to a particular mission, the code could not be generalized to the point of verbatim reuse.

To obviate this problem, developers expanded the scope of reuse in the life cycle. Even during requirements development, reuse became a priority; requirements were written not just for one system, but for an entire family of systems. These "generalized systems" could be built largely of packages reused without modification, with additional specific packages appended as needed to meet mission-specific requirements. This way, reuse within families of systems jumped to the 70-percent range. At the same time, reusable source libraries were expanded to allow the level of configuration control needed for these levels of reuse and to facilitate maintenance and certification of reusable code. All of this activity naturally dictated further refinement of the established procedures to accommodate the expanded role of reuse in the life cycle. This refinement, in turn, led to the next logical step: the construction of reusable architectures.

Over time, each of these steps has positioned the company to take advantage of emerging technologies, and each has led to steady increases in efficiency and production, with concomitant decreases in error rates. Interestingly enough, these gains were made consistently without reference to the languages in which the systems were built, their platforms, or even the organizations that built them.

The reuse strategies that have been developed in this environment, then, show a kind of autonomous validity that makes generalized recommendations possible. These, Waligora concluded, would be that every organization has to pass through certain distinct levels of reuse, with each level staged upon the previous one.

The whole schedule for the development of these reuse strategies was nearly 20 years, during which time the industry as a whole was maturing. Certainly, Waligora said, the processes existing today could not have been developed at the beginning; attempts at building generalized systems in the 1970s failed. An organization wanting to establish viable reuse strategies now, she advised, can profit from this accumulated experience but still has to start with the basics.

Business considerations and customer needs must also be considered, she said, as a regular part of procedures; build systems that meet present needs, but build them with an eye to reuse. Measure progress; learn from successes and failures; and feed that information back into the organization. Finally, she said, make a conscious effort to expand to the next level; be open to new technologies and to business strategies that make the growth possible.

Reuse: A Knowledge-Based Approach

presented by Neil Iscoe, EDS Research

Iscoe began his presentation by reviewing the classic waterfall model, in which specialized knowledge is extracted from general knowledge, mapped into a specification through text or another means, and finally mapped into an implementation level. This familiar process, he said, presents several inherent problems. Information is lost in the mapping, for one thing. He defined application knowledge as knowledge about the application and about mapping; this application knowledge is critical to the success of the project. But this knowledge is thinly spread throughout organizations; few persons in the organization really understand the entire system.

Usually, this knowledge is not written down; it exists in data models, it is embedded in the source code, and it resides in many other kinds of repository. Of course, there are domain experts and source specifications, but anyone wanting to build a new system usually has to synthesize an immense amount of scattered information in order to do so. Even with good application knowledge, changes in specifications cause other problems in capturing and holding application knowledge. Also, organization and structure are themselves forms of knowledge, and technologies like OOD can produce highly complicated models. Multiple subsystems, Iscoe said, require multiple views.

Iscoe cautioned that, at EDS, the systems are usually business-transaction processors that rely on centralized data banks, not real-time systems as in aerospace applications. Still, in his view, this knowledge-based model could be used in virtually any domain.

In the knowledge-based approach, knowledge is captured, stored in a model, and reused as needed. Knowledge about the industry or business domain is encoded into the model in terms of a meta-model or modeling language. A key point, he said, is that the domain models are not text; they are formal representations of knowledge designed to achieve specific operational goals. Because representations bias solutions, one goal of the models is to capture answers to operational questions.

The knowledge-based model, Iscoe pointed out, works in two steps. After the incorporation of the knowledge into the application domain model, developers can extract the knowledge to perform reverse engineering or to structure any number of specification models, formal structures that are computationally tractable and that allow for reasoning and inference to support specific operational goals. These spec models are language independent, and consistency throughout these processes is maintained by means of an automated theorem prover.

Iscoe concluded his general overview of the knowledge-based approach with a detailed example of its application to a specific problem in the domain of maternity benefits, one that creates large hierarchies with numerous leaf nodes.

Large Project Experiences With Object-Oriented Methods and Reuse

presented by William Wessale, CAE-Link

Wessale's presentation centered around work done in connection with the Space Station Verification and Training Facility (SSVTF), a project at preliminary design review (PDR) at the time of the Workshop. The SSVTF is intended to give full task training for the crew and ground flight controllers of the space station *Freedom*. The project comprises four training stations, two of which are to be added by the European Space Agency (ESA) and Japan.

The project embraced several fundamental issues: it was a large project, scheduled in five deliveries; it had to be created under constant budget pressure. Further, it had to be delivered concurrently with *Freedom*, so that the training packages had to be designed even as the shuttle equipment itself was being developed. Finally, and most important in Wessale's view, the system was expected to have a thirty-year life span; this added an extra set of design parameters to the project, particularly in the area of architectures.

Ada was the obvious choice, Wessale said, for this large-scale embedded system; similarly, OOD, incremental development, and other state-of-the-art technologies were also planned in from the initial stages of analysis through testing. An incremental life cycle was adopted for each of the five deliveries, with multiple internal releases for each computer software configuration item (CSCI). Prototyping was employed for the purposes of risk reduction, and inspections replaced formal reviews.

During this project, Wessale said, several lessons were learned that seem to have a wider applicability in the industry. In the development life cycle, he said, prototyping can obstruct the changes in culture that are imperative if new methods are to be implemented. Because incremental development brings with it some unusual management requirements, he recommended ensuring that the customer endorses that method in advance. Use inspections, not reviews; in Wessale's view, this approach makes more sense because it forces a focus on real issues and gives developers early feedback on their progress. Finally, he said, delivery managers are essential in development because their handling of the capability build releases (CBRs) keeps deliveries focused to customer needs and frees the developers to concentrate on future deliveries.

As to processes, Wessale said, the most critically important are those workaday processes that have little or no "sex appeal" — inspections, configuration management (CM), and deliveries, for instance. "Just-in-time" process engineering is feasible, he said, provided that sufficient resources are allotted to it. The critical component of process packaging, he found, is written procedures and checklists; and a cross-disciplinary software review board (SRB) proved necessary for uniform guidance on the application of selected methods.

Methods themselves, Wessale continued, tend to be relatively immature: one may offer strong statics while another may be weaker in statics but stronger in messaging. Fuse them, he advised, to achieve a hybrid method of greater functionality than either of its component parts. Crucial factors in the successful application of a new method are scalability (the avoidance of too-rigorous analysis or design) and the "big picture", keeping the focus broad so that all personnel understand their parts in the context of a larger whole. He warned that the switch to a new method is always accompanied by large-scale changes in the infrastructure; a chief methodologist, therefore, is indispensable if sufficient guidance is to be available through the maturation process.

The project's experiences in the areas of tools, reuse, and training also offered widely applicable lessons. Among these, he said, were that there are no tools that will perform all possible functions; like methods, each has strengths and weaknesses to consider. Rational Ada, for instance, was found to perform well in a great many areas, but formal training was necessary before its desirable functions could be exploited fully. Wessale found CASE tools largely overrated, but organizational tools proved extremely useful during the organization's culture change.

In the same vein, cultural factors played a large part in obstructing the organization's efforts at reuse; organizational tools such as strategic planning and persistence, and a willingness to adapt processes, therefore proved crucial to the success of reuse programs. Wessale pointed to object-oriented methods in particular as a way of identifying reuse opportunities and as an important part of the training essential to the maturation of the process. Training should be obtained from a single vendor, so that it forms a unified whole; and training, not education, is needed so that people will return to the workplace armed with a domain-specific vocabulary and work-related examples rather than having to map abstract information to the task.

The key factors in his organization's success, Wessale concluded, are their defined architecture, methods, and processes coupled with early and frequent feedback and guidance.

Discussion Following Session 3

Question: My question is for Ms. Waligora. In the 1970's, you said, attempts were made to build generalized systems, but they didn't work. What went wrong?

Waligora: I think that there were a couple of things. The computer technology at the time wasn't mature enough to handle the complexity. I think we also didn't have a strong process in place, and we tried to change too much too soon. I actually didn't work on those systems, but that's what I've been told.

Question: Your chart shows FORTRAN, Ada, and C systems having the same trend, but the C chart did not seem to have the same benefits that were associated with Ada.

Waligora: This is a little bit of what Frank [McGarry] was talking about this morning. These are two different application domains, the payload operations control centers and then the attitude systems. So all of the data show the same trend, which is the important thing, but it isn't really fair to compare the two domains; it's more important to look at the trend within domains, but my point was that all of the domains do show the same trend. We have seen major improvements in the Ada systems, and that may be partly due to taking advantage of the generic capabilities there. But I really think that it's more important to compare within the domains themselves.

Question: What is the size of the systems that you have implemented with this knowledge-based system?

Iscoe: Ten thousand classes, fifty thousand attributes.

Follow-up: In lines of code?

Iscoe: In that part of the system, we're not generating code. The derivation of it is from about three-quarters of a million lines of code.

Question: How does this blend with situations where you have people building systems who are experienced in the application domain?

Iscoe: We operate in an environment of, essentially, never creating entirely new systems. We always have to dovetail into old systems. So one of the things that we have to do is to interact at the specification level as well as at the code level. The informal knowledge that's relevant to the task at hand we capture; information about why something is the way it is we don't capture.

Question: Mr. Wessale, can you talk a little bit more about the training program?

Wessale: We had three primary courses. One was on OORA, one was on OOD with Ada, and one was just on straight Ada programming. The course on OORA was four days, the OOD with Ada was five, and the Ada programming was twelve. As of this last week, we've trained 420 bodies — sometimes it's the same person who appeared in more than one class. But we've basically had 160 people in OORA, 60 in OOD, and approximately 200 in Ada. And that's all been done starting in January of this year.

Session 4

Development and Application of an Acceptance Testing Model

presented by Rex Pendley, Computer Sciences Corporation

Pendley prefaced his remarks by pointing out that he approached his subject from the viewpoint of a user rather than from that of a developer. The focus of his talk was concentrated on a single phase of the life cycle, acceptance testing: the phase in which the users decide whether to accept a system.

Defining his terms carefully, Pendley distinguished mission requirements from software requirements, outlined the meaning of acceptance testing as a matter of determining whether a system meets mission requirements, and characterized test plans and test items as they are understood in his environment. He also sketched the assumptions and system parameters under which acceptance testing is performed and scheduled.

First among these assumptions, he said, is that effort is divided between the analysis team and the development team. The development team builds the test load module, which is "accepted" by the analysis team: not accepted in the final sense, but in the sense that it is declared ready for acceptance testing. The development team then runs the tests (a practice adopted to cut down on user errors); the analysis team then reviews the results. Finally, the two teams meet to decide whether to continue testing (correcting the errors that have emerged) or to declare the system acceptable. In either case, this decision ends the round of testing.

The testing model currently in place at CSC was developed in response to certain perceived needs. There was originally no firm basis for estimating the resources needed for testing or for scheduling, Pendley said; there were no standards for comparison, so it was difficult to gauge progress. In fact, because acceptance testing was not popular, he said, every tester had numerous suggestions for improvements, but there was no reliable method for assessing the effects of improvements that were implemented.

The first step in creating the model was collecting data. As this progressed, it became apparent that data analysis, a learning curve, and a pass-rate model had to be combined into a single planning paradigm. The planning paradigm as evolved is extremely straightforward, he said, but it has five particularly interesting features. First, it uses a linear fit of the project size to the staffing data to determine the number of staff-months that will be needed for testing.

Second, there seems to be a definite linear relationship between the duration of testing and the number of testers involved. There must be, Pendley cautioned, a maximum number of testers above which this efficiency is lost, but, he said, to date the pattern has held true, and that number has not been determined.

Third, testing schedules are established using anticipated availability of software for testing. The resulting schedule is subject to almost immediate revision, Pendley admitted, but such changes are easily handled because resources are managed by fairly accurate estimates. Fourth, the schedule is used to plot the number of test items (TIs) to be reported each week; finally, the pass-rate model is used to estimate and plot the number of TIs expected to pass during the week.

When applied to the International Solar and Terrestrial Physics (ISTP) attitude ground-support system (AGSS), the model proved its worth. In overall gross resource predictions, for example, staff hours, estimates met actuality to within 5 percent.

In conclusion, Pendley noted that the prediction model yields reliable estimates; it serves as a credible guide to judging reporting rate and pass rate at any given moment. At present, the model is being adapted for use on massive reuse system. And, he said, it was derived from a functioning Experience Factory although at that time the term itself was not in general use in the organization.

A Recent Cleanroom Success Story: The AOEXPERT/MVS Project

presented by P. Hausler, IBM Corporation

Hausler's presentation centered around the AOEXPERT/MVS ("Redwing") Project, the largest Cleanroom project ever completed at IBM. He began by recalling the considerable amount of work done at the SEL in the area of Cleanroom methodology, and he took issue with the idea that Cleanroom is a theoretical construct. In his organization, he said, there is a great deal of activity in transferring this approach to the production environment.

When Hausler, as a manager, first decided to adopt Cleanroom for a product-development effort, he was faced with many unique management problems. The project's decision-support product was to be marketed commercially; tradeoffs were required and, in the end, the project did not implement the method completely.

Fifty people in four groups (a traditional systems engineering group, two development groups, and a test group) staffed the project; none of the team members had worked together before. The project used a variety of languages, including Assembler, PLX, C, and Presentation Manager, in two environments. The developers' considerable expertise in artificial intelligence (AI) was offset by a relative lack of MVS expertise, as the majority of the work involved MVS. And, while Cleanroom had proved successful for small projects and academic exercises, Hausler said, it had here to be adapted for a multiple-department project in which only six percent of the staff (three of the fifty) had any experience with Cleanroom at all.

To overcome these and other negative factors, Hausler and his group adopted several strategic approaches that proved highly successful. They recognized at the outset that the method should not be mandated, given the experience and composition of the groups involved. All of the technical leaders were asked to evaluate several well-known procedural models, Cleanroom among them; they agreed that Cleanroom should be adopted in this case.

Management support, Hausler emphasized, was absolutely crucial for the success of the method; the business offices had to be trained to understand that the checkpoints of the Cleanroom process were substantially different from those that had been familiar previously. To accomplish this, Hausler broke down the standard Cleanroom process into formal procedural documents that presented his group's approach within the framework of IBM's mandatory "phase processes". This corporate standard process — used to structure the activities of legal, financial, and management offices as well as development — is built around events like PDR and CDR; the incremental approach adopted by Hausler's group had to accommodate these milestones and satisfy the expectations that they raise.

The documents that were generated, Hausler said, embodied Cleanroom features within an acceptable corporate context, but few team members had actually read them when the project began. Managers enforced the procedures in these documents by requiring all team members to certify that they had read them and would adhere to their provisions; this conformity to written procedures became part of each employee's performance appraisal.

Specific changes to the standard Cleanroom process, Hausler said, included the extension of the method's verification ideas; the mathematical models, designed for sequential processing, were unsuitable to the concurrent processing adopted by his group. The group had to invent common processing models and make some simplifying assumptions to tailor Cleanroom for the languages used. They concluded that they should use an "introductory implementation" to allow for preliminary informal specification, subsequent incremental specification, design, verification, coding, and testing, and they agreed to use familiar testing methods until statistical testing methods could be learned and applied by the testing group. (In the event, statistical testing was never applied in the course of the project.)

The tests of the first of the project's three increments showed several major components to be error free. This, Hausler recalled, caused considerable consternation among the management of the testing group, who assumed that the testers, or the testing process, was faulty. Interestingly enough, even the developers tended to insist on the discovery of errors, assuming as they did that the wrong things were being tested. Test results for the second and third increments did not meet with this kind of skepticism. On the contrary, Hausler said, reports of even a single error were at first disbelieved. Subsequent tests at installation uncovered nine errors, and tests during operation uncovered none. Productivity also increased to meet or exceed expectations, but this gain was offset by difficulties with requirements that caused the workstation code, for example, to grow from the projected 10 KLOC to nearly 40 KLOC.

In retrospect, Hausler said, he would have selected the method earlier in the process, to allow time for training and education, performing less of these activities during the first increment. Ideally, he would have initiated a small pilot project with a subset of the entire team, to develop a base of expertise on which other team members could draw. He would have preferred more, smaller, increments; and a Cleanroom consultant — a resident methodologist — would have been most helpful.

Applying Reliability Models to the Maintenance of Space Shuttle Software

presented by N. Schneidewind, Naval Postgraduate School

In his presentation, Schneidewind described the experimental application of a software reliability model to the space shuttle's onboard software at the IBM Federal Services Company in Houston. Tools that Schneidewind derived from this work seem generally useful for predicting failure rate and, related to that activity, deciding how much test time, execution time, or personnel time should be allocated to modules of the total software system. Schneidewind further offered a quantitatively derived rule for deciding when to stop testing.

With regard to IBM Houston's procedure for onboard reliability predictions, Schneidewind specified the objective as using failure history to estimate parameters. Three separate functions are comprised by an integrated reliability program, he said: prediction, control, and assessment.

Prediction consists of estimating future rates of failure, as well as mean time to failure and between failures, the number of failures, and the like. He defined control as the process of marking software that fails to meet predefined goals when those goals are compared with the predictions; assessment is deciding on a course of action when software fails to meet the goals. Each version of software is, after all, a combination of code subsets; in a sequentially upgraded system like those used on the shuttle, the first release is all new code, so the set and subset are coterminous. But the second release consists of two subsets, the first release and any new code that has been added. Subsequent releases each add a subset to the total composition of the system, one new subset being added to the carryover code subsets of the previous release.

Each of these subsets, Schneidewind pointed out, has a distinct and known failure history; this information forms the basis for initial predictions. Naturally, model parameters and predictions are updated according to observed failure rates; the feedback process is continuous. To support this approach, an accurate code change history is indispensable, he said. Every failure must be traced to the version in which it first occurred; this way, a distinct failure history can be built for each release's new code, and failure rates are not predicted against an undifferentiated mix of old and new code.

Basically, said Schneidewind, test time, execution time, and personnel time should be allocated to modules on the basis of their known (predicted) failure rates. Predictions in this approach are based on observing failures during a given time period and deriving two parameters: α is the initial failure rate, and β is a time constant related to the rate of decay of the failure rate over execution time. These parameters, in Schneidewind's model, are basic to the prediction of the failure rate. This, in turn, determines the allocation of time and effort to the testing of each module or release. Because frequent reallocation of resources can be disruptive, Schneidewind suggested predicting and reallocating at major upgrades or other significant milestones so that the changes themselves occur on a predictable schedule.

Just as reliability predictions can guide the allocation of test resources, they can be used to estimate the minimum test execution time necessary to reduce the maximum number of remaining failures to the goal level. As testing uncovers failures, predictions can indicate that more testing is needed or that the number of failures that can be expected is so near acceptable levels that further testing would be pointless.

In conclusion, Schneidewind recommended his software reliability model as a precise mathematical strategy that can help rationalize the assignment of resources to individual modules for testing and maintenance.

Discussion Following Session 4

Question: Is the thousand lines of source code you mentioned sufficient to project the time that will be involved?

Pendley: Yes.

Question: Cleanroom advocates a quality of zero defects; the reliability model you present would indicated that achieving zero defects would require infinite testing time. Could you comment on where you find the breaking point for testing? In your mathematical model, you're obviously not taking zero as your stopping point. I mean, .01 is not zero defects.

Schneidewind: I'd just say that usually it has this asymptotic effect. Obviously, we're all interested in zero failures or defects, but as a practical matter it seems to me that if you want very high quality, if you use a model such as the one I was describing, you want to set that defect level very low — perhaps it's .0001, or something on that order — and that would be essentially zero in terms of the prediction. And of course we're dealing with expected values here.

Session 5

Insights into Software Development in Japan

presented by L. Duvall, Syracuse University

The first presentation of the session was structured around Duvall's recent study, conducted in Japan, that investigated the context and practices of Japanese software development, as well as the problems confronted and the solutions created there.

The research was motivated, Duvall said, by the desire to study software management from the manager's perspective, to determine how the manager views problems as distinct from the view of technologists or researchers. Taking a sociological perspective, Duvall hypothesized that by studying problems one can gain insight into the whole software management process.

Duvall conducted interviews with managers from 14 different companies, 5 in this country, 7 in Japan, and 2 domestic affiliates of Japanese companies. The 32 managers interviewed were in charge of a variety of projects, information system development, embedded systems, and PC tools among them; 20 of the managers were from Japan, and 12 were from this country.

In Japan, the principal players are computer manufacturers, who, Duvall said, are the real powers in the industry there. As commercial users, they do not have large in-house staffs to develop software, preferring to use subcontractors for this function; this parallels the situation prevalent in this country in the 1960s. Subsidiaries set up by the users and by the computer manufacturers are also active, typically concentrating on systems development. Independent development companies also exist, but most developmental subcontracts awarded by the computer manufacturers are won by subsidiaries.

Duvall compared the environments in which software is developed here and in Japan. Looking at technology attributes, Duvall found that software tools and processes are similar; most managers cited the waterfall model as a standard, although some admitted that it does not fill their needs adequately. The tools themselves were products of the United States, and, contrary to popular belief, the software factory was not used very extensively; it was reserved to well-known application domains, and most of the managers interviewed were not part of such an arrangement.

Differences between the two countries were also significant, Duvall found. Japan has more distributed development, a natural concomitant of subcontracting. Personnel were much less experienced in Japan than in the United States, with fewer advanced degrees. The ways in which managers talked about their people were also different. In this country, the managers seldom talked about how they interact with their people and never complained; they spoke of employees as colleagues. Japanese managers, by contrast, spoke of their employees as a parent might speak of a child; they constantly returned to the themes of lack of experience and unavailability (cited as a reason for the subcontracting), and they complained that the people working for them — particularly, the young — lacked generalized skills, maturity, and so forth.

Examining problems involves not only investigating their nature but their causes and effects as well, Duvall said. Japan's lack of skilled people required the more qualified employees to work harder, to compensate for their coworkers' insufficiencies, managers complained, but there seems no way to distribute the workload more equitably. Still, Japanese managers had a clear set of objectives: they knew the importance of customer contact and feedback, for instance, in assessing customer needs.

Attitudes toward customers were also widely different between the two countries. Japanese managers tended to work more closely with the customers in partnership, stressing collaboration and negotiation; here, managers seem rather to think of their companies as providing a service to the customer, a service in which they themselves are expert. These differences were paralleled, Duvall found, in the relations between principal organizations and their subcontractors. Decision-making, often cited by Japanese managers as a problem, was not mentioned at all by their American counterparts.

In sum, Duvall said, people, not technology, constitute the main differences between the Japanese software industry and that of the United States.

Myths and Realities: How to Define Re-Engineering for a Large Organization

presented by S. Yin, Internal Revenue Service

Yin's talk centered on her department's experiences in trying to define the meaning of re-engineering and the opportunities that arose during the Internal Revenue Service's (IRS's) modernization of their tax systems during the past two and a half years. This was part of a long-term effort to move existing systems built on old technology to newer technologies based on newer languages. The effort included reverse engineering, re-engineering, and reuse, referred to as "R^3".

The first of the group's three sequential projects was to complete a survey of broad needs at the IRS and to establish a taxonomy and outline appropriate methods. A three-stage redevelopment framework was adopted. The first step in the re-engineering process was inventory and analysis, consisting of technical, functional and feasibility assessments. The second step was positioning, establishing goals and choosing appropriate technologies. The third was the transformation itself, accomplishing architecture reconciliation, mapping of logical and physical data and processes, migrating data and database management systems (DBMSs) to the restructured system, and so on.

Throughout the process, business considerations took a priority: factors like increased productivity, customer service, and sufficient flexibility to meet new needs were important here as in any other business organization. Yin's group found that the IRS's way of doing business was wedded to the available technologies, so their goal was to avoid using new technologies to support outmoded procedures and to take advantage of the re-engineering as a way of improving business processes.

Software re-engineering, then, was closely related to business re-engineering. This dictated a coherent approach to information management, as well as a careful transition to the new state of affairs. Sudden, wholesale replacements of systems and procedures would be expensive in terms of time, money, and staff disruptions. Instead, Yin said, the transition was carefully engineered, with an eye on business functions, to synchronize the legacy systems and the new systems and to use tools as enabling technologies during the transition. Throughout the changeover, she noted, existing production could not be interrupted or delayed: the bottom line, she said, was the operation's baseline. Naturally, the situation called for careful risk management. A spiral model, based on work by Barry Boehm, was adopted for this.

The second of the three projects was to develop an R^3 prototype meeting the specified objectives. The third was an automated tool market survey, again with the objective of meeting the IRS's corporate needs. The agency's stated objectives included producing an inventory of all existing systems, improving maintenance, extracting information from current systems to assist in the generation of new business-area analyses, and successful transitioning to the new target environment. Meeting these objectives, Yin's group found, required the application of specific evaluation criteria to determine which tools were the best in their classes, visiting the top fifteen vendors, and writing a report on the results.

Transitioning presented a particular challenge, Yin said, because it required the management of many overlapping tools: current systems, field operations, configuration and database management, and the rest. This was accomplished through the application of the R^3 model, which simultaneously managed application layers, software life-cycle phases, and the re-engineering tasks. The model gave Yin's team the means to rationalize and standardize the agency's defined infrastructure and accommodate its business needs through carefully selected tools.

In conclusion, Yin offered a series of recommendations for organizations facing re-engineering. These included familiar tenets of wide application such as seeking processes driven by objectives and requirements, careful configuration management, data standardization and reuse of software components. Tools that are "good enough" may safely be selected in preference to "best-in-class" tools that may offer unnecessary, if attractive, functionality. In addition, Yin recommended careful attention to the concomitant processes of business re-engineering and preparing the organization for the resulting "technology push" that must accompany re-engineering in a large organization.

STGT Program: Ada Coding and Architecture Lessons Learned

presented by Paul Usavage, GE

Usavage presented his group's experiences and lessons learned on the Second Tracking and Data Relay Satellite System (TDRSS) Ground Terminal (STGT) project that replaced an entire satellite ground system, including hardware and software, in White Sands, New Mexico.

The TDRSS supplies a communication link from GSFC through White Sands to a geostationary relay satellite and ultimately to the space shuttle or a user satellite. The hardware and software of the groundstation includes more than fifty computers, two hundred workstations, and all of the necessary software. The reliability is specified as 99.99 percent, allowing a total of only about an hour of downtime per year.

Usavage outlined the project management lessons learned during the development of this massive system as follows. He recommended allocating the right computer software configuration items (CSCIs) initially; his group allocated the items for this project based on the idea of a monolithic architecture, by technology area, but the contract specified allocation according to hardware. The subsequent adjustment was costly in time and effort.

Also, he said, the parallel distribution of their development effort meant that a change to a single task's CSCI required adjustment in the concurrent work of every other task. Switching a requirement from one task to another also caused management problems in addition to the technical difficulties involved. Usavage counselled resisting the pressure to accelerate the production schedule, citing the fact that missed deadlines cannot be made up. If the calendar states that the time for requirements analysis is done, he said, the next phase begins, in effect, before the previous phase is really completed. Understand your tools fully, and plan their use throughout the life cycle.

Commercial off-the-shelf (COTS) software and hardware is usable, he said, with reservations; it has to be tested carefully, and the team must be certain that the tool is fully compatible and up-to-date. Prototyping is recommended as a risk-reduction strategy, and code walkthroughs, he said, are a useful training opportunity. In general, projects should adhere strictly to a well-defined method.

Performance and sizing emerged as the source of the most surprises, Usavage said. Ultimately, the team was able to achieve Ada code that ran 10 percent faster than equivalent FORTRAN code — albeit at a greater investment of development time and effort — but Ada code, he cautioned, is slow if written for maximum understandability, deep nesting, and the like. And, although Usavage believes that his group had useful methods for predicting performance requirements, they failed to use those methods on this project. As to sizing, CPU power increased by a factor of twenty during the development life cycle.

The group reused some of their own components, but COTS components, they found, were not optimized for performance, and they probably should not be selected on the basis of what he called the "romance" of reuse.

Discussion Following Session 5

Question: The Japanese society seems to be male dominated, and I'd imagine that a lot of the managers were males — is that correct?

Duvall: Yes, there was just one female manager that I interviewed.

Follow-up: Then my question would be, "Aren't they biased, with them being male and you being female?"

Duvall: I felt that, while I was in Japan, any biases were more because I was a foreigner than because I was a woman. But it wasn't a negative bias. When I was interviewing, I felt as if I were given definitely special attention, and that people were very cooperative. I had done a lot of interviews before I went there, so I was pretty proficient at generating good dialogue with them. If you ask the question about the English versus the Japanese language, then, yes, there was some difficulty there. But I didn't feel that there was any bias there because I was a female.

Question: What level were the managers that you interviewed? Were they at all levels, were they directly supervising programmers, or what?

Duvall: I defined them as second-level managers; that is, two levels removed from the programmers. In the United States, that's what happened. I'd make arrangements, and I'd go interview the second-level managers, usually in their office. In Japan, a different phenomenon occurred. I'd make arrangements to interview the second-level managers, and then I'd walk into this conference room, and there'd be five or six people there. It was always understood that the object of the interview was the second-level manager, but sometimes the manager's manager was there, sometimes a colleague, sometimes a translator. Sometimes there were marketing people there.

Near the end of my visit, I thought that this was certainly affecting my interviewing, if there were more than one person there. So I tried very hard to set up an interview with one manager. I had some friends who worked for a company at Osaka, and they were to set up the meeting for me. I said I'd bring my translator from Osaka University, so there's no need for anybody else. So I took the train, walked into the conference room, and there were *eight* people there! So much for the single interview.

Question: Are they actively practicing total quality management?

Duvall: Never once did any of the Japanese managers specifically use the term, although they did sometimes talk about it. In the interviews, I didn't ask that question, so it wouldn't have come up.

Question: Was their organizational structure more flat?

Duvall: I didn't find that, although I did ask them how many people worked for them, and so forth. In a lot of the American companies, they were in a downsizing mode or reorganization, and the organization had already been flattened, but I did not see that happening in Japan.

SUMMARIES OF PANEL PRESENTATIONS AND OF PANEL DISCUSSION

Marv Zelkowitz, NIST/University of Maryland, Facilitator
Stu Feldman, Executive Director of Computer Systems Research, Bellcore
John Foreman, Director of STARS Program, DoD
Susan Murphy, AAS Software Manager, IBM
Tom Velez, President and CEO, CTA

Marv Zelkowitz of the University of Maryland opened the panel by reviewing the rationale of the SEL's interest in Ada, the history of the SEL's studies of the language, and the results of using it in this environment. He cited cost savings realized through reuse of Ada components, but noted that similar savings resulted from reuse of components in other languages. Problems with Ada, he continued, include the fact that compiler support for IBM mainframes is poor.

The language's syntax, as an outgrowth of the structures of FORTRAN, ALGOL, and other "classic" languages, Zelkowitz finds acceptable, but there are persistent problems, he admitted, such as the difficulty of learning Ada well, the lack of production-quality compilers, and some incompatibility with OOD. A new version of the language, Ada 9X, promises to solve many of these but, on the other hand, Zelkowitz pointed out that Ada 9X may raise new incompatibilities with existing programs. After a decade of Ada development in the SEL, Ada's many features do in fact facilitate large-system design, reuse, and maintenance, he finds, and its use seems to be expanding (albeit slowly) worldwide into many commercial areas. The number of courses and textbooks in Ada seem also to be growing slowly, he added, in relation to the large-scale unsupported growth of other languages like C and C++.

Still, on the whole, Zelkowitz concurs with the opinion of Erhard Ploederer, whom he quoted at length. The issues that remain for discussion, he said, are the questions of whether Ada is an economically viable language for building systems, and, if it is not, then what criteria are needed for determining its economic viability, and for what class of projects is it applicable?

Zelkowitz concluded his introduction by asking the panelists to state their positions on these and related questions and to support their opinions with objective or subjective evidence. He asked them to then outline the actions that, in the panelists' opinions, should be taken, and to sketch their predictions for the language in the future.

Stu Feldman, Executive Director of Computer Systems Research at Bellcore, began the discussion. Ada is a perfectly acceptable language, he said, but not one that he has ever particularly wanted to use. His criteria were altogether informal and subjective, he said, based on a "careful-observer" model rather than on careful metrics. Of course, anything and everything can be written in virtually any language; all programming languages of any interest are equivalent in terms of what one can write in them, he said.

The significant properties of Ada, in his view, are that it is complete, it is well supported by compilers and tools, and it is sponsored (which he finds to be both a strength and a weakness). It is designed for real time, he added, but configuration support is questionable, and the language has some definite drawbacks.

Comparing Ada's features with those of other languages, Feldman found its maintainability to be somewhat greater than that of C and C++; its complexity and compiler difficulty he ranked greater than those of C and C++.

For the future, Feldman projected that Ada would, by the year 2000, find a niche in the development of embedded, real-time systems and in software engineering, but he saw it as giving place to other languages in commercial, engineering, and scientific systems, in prototyping, and in research.

John Foreman, Director of STARS Program, Department of Defense, next presented views that differed somewhat from Feldman's. Ada is not dying, he said, but the language is subject to "over expectation" about its possibilities. He cited its potential for growth, but cautioned that the insertion and transfer of new technologies always takes time. He also raised questions of maturity: Ada itself is still maturing, but is the receptor community mature enough to receive it, even in its present state? Speaking from his own experience, he asserted that, just as the Department of Defense still has its own unique requirements to satisfy, so too must the commercial sector.

Foreman listed criteria for judging Ada. Tool quality has improved markedly, he said, and continues to get better. Recent years have seen major improvement in the hardware base for Ada, such as the appearance of 32-bit processors. The language has been used successfully for large projects in many countries, real projects with real results. And he cited the importance of Ada stability and verification.

For the future, Foreman urged further case studies to evaluate the language, as well as a greater concentration on education — approaching the situation from another standpoint.

Susan Murphy, Advanced Automation System (AAS) Software Manager at IBM, maintained that Ada is alive and well on the AAS. This system, consisting of more than 2.5 million LOC, will be used in more than 400 towers at nearly 200 terminals by 2000; hundreds of Ada programmers have already worked on the project, and it has been used as the basis for other programs everywhere from Taiwan to Belgium, by way of Mexico.

For Ada to grow, she said, Ada 9X has to be fully compatible with Ada 83, or all of these and other major systems will not transition to Ada 9X, and hundreds of Ada programmers will concentrate on transitioning problems and will not evolve to use its features.

Tom Velez, President and CEO of CTA, began with an observation that derived from a survey conducted by the Air Force: 40 percent of development in 1991 was in COBOL, 30 percent in FORTRAN/JOVIAL, 17 percent in any of 450 other languages, and only 10 percent in Ada. Projections for 1995, though, put Ada at 40 percent, with only 20 percent of development in COBOL and 25 percent in FORTRAN/JOVIAL.

This growth, he said, is paralleled in projects outside his company's primary client, the Department of Defense. He finds similar trends in commercial and academic projects, international projects, and in other non-defense government projects, and the breakdown among specific armed services shows the same pattern. The basic competition that Velez sees is between Ada and C++ and, to date, he finds that Ada seems to be winning.

Zelkowitz then opened the discussion for further comments from the panelists. Feldman pinpointed what he called the real problem with languages, that it is very difficult to get a programmer to learn a second one. Therefore, he said, "unbounded bigotry" about languages dominates the field. He noted that the problems in software engineering are well above the level of languages or language support; when designing a product, one asks what the pieces will be, how they are going to fit together, how they will be manipulated, and the like. Some engineering problems, he said, are at the moment running against Ada, simply because of the general configuration of the programmers' world.

The comment that Ada 9X should be fully compatible with Ada 83, Feldman continued, reminded him that the only reason, in his view, that C++ has been so successful is that it really was C plus added features. People who loved C, he said, were willing to accept the software engineering features of C++. But Ada is clearly not dying; nothing in the field of software engineering dies, in his view.

Foreman added that he agreed fully with Murphy's contention that Ada 9X should be fully compatible with its predecessor. He feared that perhaps the issue is not being adequately considered in the Ada 9X development process. He recalled a personal experience of some years ago, when he received a telephone call from the program office at an Air Force base asking about costs to build an Ada compiler. When Foreman asked for more information about the desired functionality, the officer replied that an aircraft having a 14-bit processor was to be upgraded, and the Air Force wanted a compiler built for it so that they could reconfigure the software in Ada. Foreman recalled that he asked why they wanted to do that, only to be told "because we have to". Such a processor would cost around 10 million dollars, he told them, because not many people would be interested in a 14-bit processor. Of course, he concluded, producing such a processor would also take a great deal of time, and it would not solve their problem. Instead, Foreman recommended that they begin with system engineering to find a new, 32-bit processor and put it into their system. This kind of "over-focus" is what goes wrong; system engineering and common sense, he said, are of primary importance.

Murphy offered no additional comments but called for remarks from the audience and panelists on two points. One was her perception, echoed by Feldman, that C++ has been so successful because one can move into it easily from its predecessor. How can the industry approach this factor in the case of Ada, she asked, and is it appropriate to pressure the Ada 9X community to recognize that production systems must be accommodated responsibly? Second, she wondered about the fact that open systems may not necessarily accommodate Ada; again, she called for ideas about appropriate ways to address this.

Velez observed that there is a move toward commercial software in the Department of Defense, and his company is increasingly packaging such products in their preferred language.

One participant pointed out that the industry is in the midst of a paradigm change in which traditional hardware is being replaced by nontraditional architectures. In this light, design principles and packaging principles will change. Did this, he asked, promise to influence the languages to be used in the future? Yes, said Foreman, with Feldman's express agreement.

Another participant raised the question of young people now being trained on computers who are not taught Ada. If Ada is to survive, he maintained, its proponents should support efforts to put students in contact with Ada before they embrace unsuitable languages. The marketplace is essential to the survival of a language, he asserted, and the age for entry into the marketplace for learning languages is below six years.

Feldman countered that teaching Ada to six-year-olds was not an entirely unclouded goal, but certainly, he said, many of his colleagues in academic positions note that their first job is deprogramming new students from the BASIC they have learned previously. He remarked that some schools are now using languages like ML and LISP as first languages, partly as "shock treatment", partly because they don't look anything like BASIC, and partly to try to get students thinking in viable ways. Also, he said, the languages are small enough to be taught in the first few weeks of a course, which is very important in an academic schedule.

Because Murphy was speaking about the future of the AAS, another questioner wondered whether there was any information about prototyping efforts in which Ada-developed systems would actually perform within real-time constraints. One of these systems, she answered, is currently in formal factory test at the Rockville site, and a great deal of attention has been paid to its performance. The questioner noted that the MITRE Corporation was called in to troubleshoot some of the problems that arose during development; some of the Ada applications, running on UNIX platforms, had apparently not been performing in real time. This was some time ago, he said.

Murphy replied that the information was accurate, but added that this system was bid in 1987 or 1988, and one of the target processors that was bid, for example, was understood as a 16-million-instructions-per-second (MIPS) machine, in the best engineering understanding of the project at that time. However, when the new processor was installed, the actual speed was measured at less than 10 MIPS. This caused the performance problems, none of which was associated with Ada. The questioner added that, because Ada has to run on an operating system, he wondered whether the Ada programs actually could be made to run acceptably; Murphy answered that the primary target operating system is Advanced Interactive Executive (AIX™), a UNIX operating system that is successfully running at acceptable performance levels.

One participant, from the management information systems side of DoD, said that his department was having a great deal of trouble getting adequate structured query language (SQL) bindings. Although there are many products available for this, he said, they are not standardized. Foreman responded that he understood there to be efforts in progress to achieve some standardization, but that he was not in a position to give a full answer.

Another participant added that one should not buy into SQL at the language-statement level; the program suffers from this, in his experience. He advised writing a few SQL statements called in as a procedure instead.

Another questioner estimated that, in the competition between C++ and Ada, C++ is the language of choice in free-market situations, while Ada is mandated by some clients. This difference, he said, may be simply between tactical marketing and strategic marketing, but it may be a true reflection of the market. Velez offered the opinion that the principles behind Ada preceded the availability of tools by enough time that people looked for other ways of achieving the same objectives using COTS software.

Murphy added that, although it is not widely known, the FAA required a language trade study by both of the principal contractors (Hughes Aircraft and IBM) during the design competition phase for the AAS. Both recommended Ada, but, at that point, the FAA was under significant criticism from Congress and from the Government Accounting Office (GAO) for considering Ada, which was then considered too risky. The FAA commissioned an independent study, organized by MITRE and headed by Vic Basili, that, like the language trade studies, unanimously recommended the use of Ada.

In response to the question of possible attribution of problems in cost and schedule management to the use of Ada, Murphy said that inefficiencies in the configuration-management tools and immaturities in the compiler were not at all significant. There have been problems with schedule and cost, she added, but, as the presentations showed, these were attributable to other factors. The single biggest factor has been what she calls "requirements convergence", which results from a conflict between the requirements phase and the acquisitions phase. In this case, she recalled, there was first a long and complicated competition phase during which the FAA was required to communicate rigid requirements to both competing contractors. After the award, multiple customer sets within the FAA put forth different versions of what they wanted. Resolution of these discrepancies, she said, was the single largest problem.

In response to another question relating language choice to the issue of tactical versus strategic marketing, Feldman stated that world of C and C++ is highly tactical; the language is preferred because of the convenience of binding to COTS software, like most of the operating systems available today. Trying to work into an extant world, he said, is made easier; similar products in Ada are not easily found.

One participant found it interesting that a panel discussing the viability of Ada did not include a representative from an Ada company. Offering some industry information as a "benchmark" for how well the language is faring in the marketplace, he noted that the industry, including applications contracts, exceeded four billion dollars in revenue. With the recent merger of Telesoft and Alsys, there are now two Ada vendors in the top register of software vendors in the United States. Most major Ada vendors, he noted, experienced a 40-percent rise in revenues. This evidence, he said, shows the language to be both vital and growing. As a former fighter pilot, he expressed his own unwillingness to trust any aircraft systems that were not written in a language like Ada — a language specifically designed, he noted, to enforce good software engineering practices.

Feldman agreed with the participant's views, but added that the Ada vendors seem to lack the technique of marketing the language outside of defense applications. The language will not find easy acceptance in the commercial world, he warned, because of its technical nuances and because of the many statistically valid studies of Ada versus C or C++ that pose a real challenge to sales and marketing segments of the industry. The vendor community as a whole seems to be recognizing that; at least 50 percent of their income is from work done outside DoD.

Session 1: The SEL

Frank E. McGarry, NASA/Goddard

Vic Basili, University of Maryland

Michael Stark, NASA/Goddard