In This Issue...

- State-Based Software Development Process
- Information Warfare-Part 2, "The DoD Organizational Structure"
- Solving the Year 2000 Dilemma
- Software Technology Conference (STC'96) Summary
- Analyst's Corner - "Software Crisis or Hardware Success"
- Software Quality Week (QW'96) Summary
- International Conference on software Engineering (ICSE-18) Summary
- DACS Course Announcement - "Cleanroom Management and Technology"

Register Now for Software Tech News
State-Based Software Development Process

by Buford D. Tackett III, ATAMS Project Manager

The original DARPA STARS project had a number of objectives as it sought to infuse leading edge software engineering technology into the Department of Defense (DoD). One of those objectives was to develop well-defined processes that were not only repeatable, but practical and effective in developing software that was better, faster and cheaper. The Air Force STARS/SCAI demonstration project developed a process that combined aspects of the Cleanroom methodology, the Ada Process Model and an object-oriented, state based approach. As the AF STARS demonstration project was winding down, a critical, short-fused requirement surfaced that needed to provide an automated capability to NORAD system controllers in Cheyenne Mountain. This new development project, called ATAMS (Automated Tracking and Monitoring System), was to be fielded in several phases, but the first fully operational phase was to be deployed within one year. Because of the success of the STARS/SCAI demonstration project, the Air Force decided to waive its standard approach and use the SCAI process. ATAMS was deployed early this year, on-time, on budget, and with a user that is delighted with the system. What follows is a very brief overview of some of the major aspects of the State-Based Software Development Process that was used throughout the ATAMS project.

The State-Based Process: The process was enacted within the framework of an evolutionary, incremental approach derived from the Ada Process Model and the Cleanroom methodology. Our overall approach was to develop the software following an overall incremental approach where each increment would be fully executable, testable, and demonstrable to the user for observation and approval. The Application Engineering (AE) team believed that a software development process definition could be developed and described using an Object Oriented (OO) approach. This primarily led to viewing a particular development item as an object undergoing a series of transitions from one state to another with state operations that related to both the OO and Cleanroom methodologies.
Some of the objectives that drove our approach to this process definition were to tie the management process directly to the technical process; fuse OO and Cleanroom methodologies; maximize the use of inspections and state-of-the-art process methodology; and emphasize simplicity in all aspects of the definition and enactment. The resultant process uses a state transition diagram to describe the states through which a particular object must transition to reach its final state. Figure 1. depicts the overall view of the State-Based Development Process.

A Process Guide was developed to detail the process and management activities that must occur within each state and each transition. Team walkthroughs constitute the major activity in which the state of the object is examined, verified, and transition decisions made. Within each state, the guidelines provide for an initial walkthrough, intermediate walkthroughs, and a final walkthrough. Depending upon the progress of the development item, there may be one or many walkthroughs held within a state until the Approval to Proceed directs a state transition.

Each walkthrough has a well defined list of entry and exit requirements. In most cases, the exit requirements consist of Walkthrough Minutes, an Approval to Proceed, and the actual Software Development Item products.

Metrics are collected throughout the process to capture the return on investment in terms of the resources required to conduct the walk- throughs and the number of defects uncovered. At the beginning of each walkthrough, every reviewer is polled for the amount of time they spent in review.

Walkthroughs are the heart of the process. It is important to maximize the effectiveness of the walkthrough and minimize the overhead associated with it. The most important part of the walkthrough is the inspection that is conducted upon the development item. This means that the success of the walkthrough depends mainly upon the professional and conscientious examination of the walkthrough inputs by the reviewers. The objective is to uncover every defect. Reviewers are assigned at the start of the process and they remain the same throughout.

**Walkthrough Metrics:** Gathering metrics is an important part of the process and that includes metrics associated with the walkthrough effort. These metrics include: walkthrough duration; number of participants; external review time; pre and post-walkthrough overhead; major and minor defects/issues uncovered. Major defects are defined as those defects that would have negatively affected the operational system. Minor defects are anything else.

**Quality Assurance Activities:** One of the primary objectives of the State-Based Development Process is to engage every member of the team in quality assurance activities and make it a natural part of their development paradigm. We were convinced that the only way to insure quality in the end product was to make sure the process builds it in from the beginning to the end. If the process is followed, it will naturally lead to a higher quality product. However, project management must continually monitor and
evaluate this aspect of the process to insure that good quality assurance efforts are being conducted.

**Results:** The State-Based Development Process was used throughout the ATAMS project and with great success, both from a productivity and a quality standpoint. Some sample metrics that characterize the walkthroughs are shown in Figure 2. One can discern that the return on our investment is 2.5 major defects found and over 6 minor defects uncovered. That equates to a little over one hour of staff time per defect when considering all cost and both major and minor defects.

From a quality/productivity standpoint, our results have been fairly consistent from the initial SCAI project through the ATAMS phases to-date. The statistics in Figure 3 are from the first operational release of ATAMS. Of the four Incident Reports that have been written for ATAMS, only one is applicable to the over 37,000 hand-developed lines of code.

![Figure 2. Sample Metrics](image)

**Conclusions:** The process was enacted with great success and continues to undergo refinement and improvement. Both management and the development team are sold on it and would be reluctant to develop software without it.

Management visibility was greatly improved because the development process was driven down in its granularity and status could be more readily seen.

To the project manager, the process has offered a number of benefits:

- Easily understood by managers and engineers; easily taught and caught, one-page graphic representation
- Provides clear communication of status within the team and to management
- Provides concise/definable states with clear entry/exit criteria, activity steps, procedures and methodologies
- Natural points and clear guidance for valuable capture of metrics
- Promotes "team" concept and ownership
- Facilitates continuous process improvement
- Greatly increased the quality of the product

<table>
<thead>
<tr>
<th>Staff Months (SM)</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>162</td>
</tr>
<tr>
<td>Displays / Windows</td>
<td>35</td>
</tr>
<tr>
<td>-------------------</td>
<td>----</td>
</tr>
<tr>
<td>Developed LOC</td>
<td>37,384</td>
</tr>
<tr>
<td>LOC / SM</td>
<td>612</td>
</tr>
<tr>
<td>Optical Incident Reports (IR)</td>
<td>4</td>
</tr>
<tr>
<td>IRs / KLOC</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Figure 3. ATAMS Statistics**
Information Warfare - Part 2

by Robert Thompson, Kaman Sciences Corp.

The Department of Defense (DoD) has implemented numerous organizational changes in response to the evolving paradigm of Information Warfare. This article will highlight the Service (i.e., Army, Navy/Marine Corps, and Air Force) initiatives in preparation for Information Warfare, as well as provide an overview of some of the organizational changes at the OSD-level (Office of the Secretary of Defense).

Department of the Army

Among the initiatives enacted by the U.S. Army was the creation of the Land Information Warfare Activity (LIWA) in 1994 at Fort Belvoir, Virginia. LIWA’s mission is to provide the Department of Army (DA) level Information Warfare/Command and Control Warfare (IW/C2W) support to the Land Components and Army commands to facilitate planning and execution of Information Operations. In addition, LIWA will coordinate with National, Joint, and Service IW/C2W centers in the exchange and sharing of intelligence and information support. LIWA provides Field Deployment Teams to support Land Component Commanders.

Department of the Navy

The Navy has distributed IW/C2W responsibilities among several organizational elements. The Director of Space and Electronic Warfare (OPNAV) (N6) has the overall responsibility for IW/C2W development and guidance. The Fleet Information Warfare Center (FIWC) is located at Little Creek Amphibious Base in Virginia Beach, VA. FIWC is viewed as the Navy's IW Center of Excellence and acts as the Fleet CINC’s (Commander in Chief) principal agent for IW/C2W tactics, training, and procedures. The Naval Information Warfare Activity (NIWA) functions as the Navy's technical agent and interface to other Service and National IW organizations. NIWA also works closely with FIWC in the development of IW technical capabilities for Navy and Joint Operations. Other Navy organizations implementing IW/C2W measures include the Director of Naval Intelligence (N2); Naval Operations (N3); Deputy Chief of Naval Operations, Resources, Warfare Requirements, and Assessments (N8); Commander, Naval Security Group Command; Commander, Naval Doctrine Command; and the Chief of Naval Education and Training (CNET).

Department of the Air Force

The Assistant Chief of Staff for Intelligence (ASCI), along with the Deputy Chief of Staff for Operations (XO) and Assistant Chief of Staff for C4 (SC), has the responsibility for IW doctrine development. One of the initiatives implemented by the Air Force was the creation of the Air Force Information Warfare Center (AFIWC) in September 1993. The mission of AFIWC is to support operations, campaign planning, acquisition and testing through the application of IW/C2W assets. Support includes the organization and training of deployment teams, the development of C2W databases and applications, and
Air Force Computer Emergency Response Team (AFCERT) functions. AFIWC is located at Kelly AFB in San Antonio, TX.

Office of Secretary of Defense
At the OSD-level, two panels have been created to address Information Warfare (IW) issues. The first panel, the Information Warfare Executive Board (IWEB), is chaired by the Deputy Secretary of Defense and has been chartered to examine IW planning, policy, and legal issues. The second panel, the Information Warfare Council, is chaired by the Assistant Secretary of Defense (C3I) and provides support to the IWEB on IW issues. An overview of panel(s) membership is provided in Figure 1. Administrative support for both panels is provided by Office of the Assistant Secretary of Defense (C3I).

Source: Dr. Barry H. Brief, October 18, 1995

<table>
<thead>
<tr>
<th>IW Executive Board</th>
<th>IW Council</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaired by:</td>
<td>Chaired by:</td>
</tr>
<tr>
<td>DEPSECDEF</td>
<td>ASD(C31)</td>
</tr>
<tr>
<td>Membership:</td>
<td>Membership:</td>
</tr>
<tr>
<td>USD(P) USD(A&amp;T)</td>
<td>Representative for Each</td>
</tr>
<tr>
<td>USD(C) USD(P&amp;R)</td>
<td>Principal on the Web</td>
</tr>
<tr>
<td>ASD(C31)</td>
<td>Both J3 &amp; J6 for Joint Staff</td>
</tr>
<tr>
<td>VJCS</td>
<td>Dir SpecProj</td>
</tr>
<tr>
<td>Vice Chief's of</td>
<td>USD(A&amp;T)</td>
</tr>
<tr>
<td>Military Service</td>
<td>Dir Net</td>
</tr>
<tr>
<td>GC DoD D / DCI*</td>
<td>Assessment</td>
</tr>
<tr>
<td>Dir NSA Dir DIA</td>
<td>Dir IW</td>
</tr>
<tr>
<td>Dir PA&amp;E Dir DISA</td>
<td></td>
</tr>
<tr>
<td>Dir IW ExDir</td>
<td></td>
</tr>
<tr>
<td>NSC Exc*</td>
<td></td>
</tr>
</tbody>
</table>

* Invited Non-DoD

Figure 1. DoD IW Panels

Next Issue: IAC Program Support for Information Warfare
You have heard about the computer systems and software problems involving the year 2000 haven't you? Do you know the extent of the problem? Do you know how it will affect your organization? Do you believe it? Does your management believe it? Do you have a plan in place to solve the problem? Are you working the problem now? Do you know where to get help and more information?

What's the Problem?

Back when code was tight and memory was scarce it became a standard practice to abbreviate date code within software as mm/dd/yy. Note the two places allocated for the year designation. That worked just fine till recently. Unfortunately we now understand that as we head toward the end of this century we'll see massive failures of software and systems. They will fail because when the clock rolls over at midnight on the last day of December 1999, the computer won't know the correct date. It may indicate the year is 1900 or it may pick some other year. In software and system assessment tests, the years 1980 and 1984 are popular. If you have any programs that do date-time projections and incorporate some hashing and random number generators that use parts of the system date, you'll have a problem when your program attempts to divide by zero and your program crashes - repeatedly.

Another problem surfaces because when a number of programs were developed, the programmer didn't always understand that the year 2000 is a leap year. In grade school we all learned that each year actually consisted of 365.25 days and to account for the fraction, every four years we tacked on another day, February 29th. Some of us also learned that there is an exception to that. If the year is the first year of a century, say 1900 as an example, then it is not a leap year. Who remembers that there is an exception to the exception? If we hadn't played hooky that day to play stickball, we would have learned that if the first year of a century is divisible by four (for argument's sake, let's use the year 2000 as an example) it is a leap year. You can appreciate the magnitude of miscalculations when your software and system can't correctly determine time. If we're lucky and the problem is not corrected the program will simply crash. What happens when the program continues to operate and the operators fail to note the errors until after thousands of dollars of interest or payments are lost?

There are other problems too. There are the "year-in-century" problems. What happens when you want to arrange for a 15 year home owners loan. You set it up in 1996 for payoff in 2011. However, if you subtract the two year-in-centuries, using the existing two-place year representation, the calculation is 11-96 = -85. That is truly an early payoff schedule! What if you are working with large stocks and inventories? Your system may tell your people that the shelf life of items has expired and they may dispose of perfectly good, high cost items, just like the computer told them to do. Will you catch the error...
in time? These problems are just the highlights. There are more, like systems interpreting 99 as \textsuperscript{3}end of file\textsuperscript{2} and stopping your processes. Or it might look at 00 as a null value and sit in limbo waiting for you to tell it to do something.

If uncorrected, the costs to business, industry and defense will be astronomical. Already it has been estimated that the cost to correct the millennium date problem will run somewhere between $300 billion to $600 billion, with some estimates placing the cost at over $1 trillion worldwide. Some estimates put the cost of taming the Year 2000 compliance beast for the US Federal Government at somewhere in excess of $25 billion. The costs are so high because the date problems are pervasive. Greater than 90\% of all software programs will be affected. Depending on software programming complexity, integration, and interaction with other modules and systems, cost estimates range from $1.00 per line of code to over $8.00 per line of code for the more dedicated, esoteric sorts of applications - like those in embedded defense systems. In general, from 1 to 5\% of code may be date involved. You'll have to arrange to check it all to make sure you haven't missed any lines. Certainly that percentage can be higher when dealing with business software which deals with costs, schedules, inventories, and the like. By the way, we're not just talking about software here. There are a lot of embedded chip sets with the two-place date schema hard coded. Those legacy systems with affected chips will have to be remanufactured or the systems may have to be scrapped. Bridges and routers anyone? How about all those personal computers in the work place and all of the non-compliant software in use?

Many systems and software packages will have to be updated and replaced. Anyone running a network? Anyone have archives? Do you have legacy data? Anyone doing work on a home system and uploading their data to the office system?

What Should I Do?

First, brief the top echelons of management about the problem and the imperative for immediate action. The drop dead date is fixed. The year 2000 will be here whether or not you are ready. Recognize that you must start work to determine the extent of the problem within your organization and then work to correct it. In large measure, bringing your systems and software into compliance for the year 2000 is a complex program management task. You will have to determine whether you can do the work in-house or whether you will need to out source some or all of the work.

The steps are as follows:

- All software and hardware will have to be inventoried and assessed to determine the extent of the year 2000 problem within your organization.
- Develop an action plan which permits you to be ready to test your systems and software starting no later than 1 January 1999. We strongly recommend you prioritize your compliance initiatives so that mission essential systems are identified, assessed, fixed and tested first. For many of you who have not already started the process, there is no time to waste on pilot studies and demos. You must act now.
- Renovate, modify, replace or make a decision to scrap existing systems and software based on your mission and business priorities and take action.
- Test and validate the changes over as long a time period as the magnitude of your effort permits. Since reaching 1999 may cause a number of failures from unmodified code and systems, your plans should include testing and modification during 1999.
- Implement any corrections based on the results of your tests and validation. Make sure you make education a part of your plan so that well-intentioned organizational members don't load corrupt data into tested, compliant systems from non year 2000 compliant systems.
- Do some contingency planning. In all likelihood, "Murphy's Law" will apply during your work on this task and things will go wrong at the worst possible time. Priorities will change, staff changes may impact your efforts and the funding picture may at best be a moving target.

Can the DACS Help?

**Yes.** We are gathering information concerning the year 2000 problem, tools, methods, testing measures, vendors and compliance program experiences. We will provide information through the DACS Newsletter, our web page and other methods as necessary. We will assist you in defining the problem to top management and explaining it within your organization. Members of the DACS staff will also work with you to conduct an inventory, assess your year 2000 compliance problems, effect solutions, and test corrective measures.

For further information check the DACS Topic Area [New Millennium: The Year 2000 Problem](#) or to discuss arranging for DACS assistance via a Technical Area Task, contact:

Elaine Fedchak - DACS  
Y2K Project Leader  
775 Daedalian Drive  
Rome, NY 13441-4909  
Telephone: (315) 334-4900  
Fax: (315) 334-4965  
E-mail: efedchak@rome.kaman.com
Software Technology Conference (STC'96)

by Edmond VanDoren

The theme of this Eighth STC was integrating people, process, and technology. Within that theme were strong tracks on topics which related strongly to the DoD problems of today doing more with less, and doing it better.

Larger than ever, with more than 3,100 attending, This STC was a well-planned conference, one with something for just about everyone in the software business. There seemed to be a balance of service personnel, DoD civilians, and contractors, with a noticeable commercial presence.

Nine concurrent tutorial tracks covered the first morning. The one I attended, Practical Software Measurement, had a book-sized handout that covered a huge amount of information about an ongoing effort to define and implement a meaningful software measurement process. Judging from break-time reports, the other tracks were also a half day well spent.

The keynote speaker, Retired admiral William Owens, addressed the central issue: The shrinking DoD budget.

The figures are at odds with each other:
- Downsizing goal (in ten years) - 45%
- Projected savings from privatization -- 25%

But the DoD budget's two components are:
1. Fixed - (infrastructure support costs) - 65%
2. Variable - (new purchases and procurement) - 35%

To reach the downsizing goal requires more than shrinking the infrastructure. It means reducing procurements by as much as 70%! Challenging figures, made even more challenging by the 500% increase in contingency operations. Technology, he noted, will be called on to meet much of this challenge.

Standards were a particularly hot topic. Sessions about the Defense Information Infrastructure (DII), the Global Command and Control System (GCCS), and the Global Combat Support System (GCSS) both developed using DII standards were standing room only only.

Interoperability, the elusive butterfly of defense systems, also came in for a good share of the discussion. Dr. Marvin J. Langston, Deputy Assistant Secretary of the Navy of Research, Development, and Acquisition, spoke of the influence of software technology on the way systems are built. He cited a very real accomplishment of the services, an information system built and fielded in two months using
existing communications satellites and an internet link from our Bosnian peacekeeping force to servers and based on Netscape. He also described a pressing reason for interoperability in the engagements of today all the forces fight in the same theatre of battle, making it imperative that information be shared to avoid shooting at our own forces.

The Year 2000, or millennium, problem was heavily discussed at several sessions. There are several defense-oriented web pages available for exchange of information, and each one has at least one working group associated with it.

Process continues to be a popular subject. It was the subject of several tracks, and many talks, but people issues were a strong emerging theme even in process-related reports. Tom DeMarco and his associate Tim Lister delivered powerful messages across several presentations about how the recent emphasis on process has overshadowed how essential your people are. If you could have one just of the three: people, process, or technologies would there be any question?

On the nontechnical side, I hope no one who attended missed Tom Sullivan's remarkable show and speech. Tom has been blind since childhood, but he never let it stop him from becoming a singer, musician and actor (he even tried baseball along the way). He related his powerful, never give up message to our own endeavors in computer technology, and made the small things seem just that. Get information next year's conference, STC '97!
The "software crisis" is characterized by an inability to develop software on time, on budget, and within requirements. A seminal paper in quantitatively assessing software costs predicted that software costs would come to consume an increasingly larger share of system costs (Boehm 73). Figure 1 resembles a graph presented by Dr. Barry Boehm, illustrating this prediction. Figure 1, however, is derived from a mathematical model yielding this curve, not Boehm's paper. This model shows that as long as productivity increases faster in hardware than in software, the percentage of system costs spent on software is bound to increase. This trend is not a manifestation of a software crisis, but the inevitable result of trends in hardware which are unlikely to be duplicated in software or almost any other industry.

The model was developed based on techniques developed by the economist Luigi Pasinetti (1993). Suppose hardware and software can each be measured in a standard unit, and that a computer system consists of a unit of hardware and a unit of software. Suppose the cost of producing both hardware and software is proportional to the amount of labor used in producing the respective outputs. Let $l_h$ denote person-years required to produce a unit of hardware, and let $l_s$ be the number of person-years required to produce a unit of software. Assume productivity grows at the rate $P_h$ in the hardware industry and at the rate $P_s$ in the software industry:

$$\frac{dl_i}{dt} = -P_i l_i, \quad i = h, s.$$  
Equation (1)
Define $P$ as the proportion of system costs spent in software in the production of a computer system. The rate of growth of this proportion is given by Equation 2:

$$\frac{1}{p} \frac{dp}{dt} = (\rho_h - \rho_S)(1-p)$$

Equation (2)

Given a rate of growth of productivity in hardware exceeding the rate of growth of productivity in the software industry, software will consume an ever increasing proportion of system costs, as in Figure 1.

Assuming the above argument is correct, is increasing emphasis on software process improvement therefore misdirected? No, economic reasoning shows it is appropriate. Optimally, management should continue to add additional funds to a software process improvement program as long as the marginal decrease in the expected cost of software exceeds the marginal cost of a software process improvement program. If the ratio of the marginal costs of hardware and software process improvement programs remain unchanged, then management should direct proportionality more to software process improvement as software costs become a greater proportion of system costs. Progress in software process improvement technology may also increase the desirability of funding such programs.

References:
Quality Week is an excellent conference which is held in the Spring of every year in San Francisco. The QW'96 event was organized by SRI's (Software Research Institute) Executive Director Ms. Rita Bral, and cosponsored by the ACM (Association for Computing Machinery) and IEEE Computer Society. Quality Week '96 was the first event of its kind to receive sponsorship status from both the IEEE Computer Society and the ACM.

Quality Week Conference focuses on advances in software test technology, quality control, risk management, software safety, and test automation. Software analysis methodologies, supported by advanced automated software test methods, promise major advances in system quality and reliability, assuming continued competitiveness. The mission of the QW '96 was to increase awareness of the importance of Software Quality and methods of achieving it. It promotes software quality by providing technological education and opportunities for information exchange within the software community.

Over 825 persons attended the four-day QW'96 event with around 10% from the greater San Francisco Bay area, 40% originating from the elsewhere in the Western USA, 40% from the Eastern USA and 10% from outside the USA. QW'96's vendor area had 29 exhibiting companies.

QW'96 attendees came from over 20 countries, from as far as Hong Kong and Finland. Delegates from over 375 different companies, ranging in size from small consulting agencies to giant international corporations, attended the event. QW'96 speakers' backgrounds were about equally divided among industrial researchers, academics, and real-world practitioners. The technical strength of QW'96 is the work of the international Advisory Board, composed of 75% from the USA and 25% from outside the USA. They reviewed a total of 185 proposed presentations. Track Chairs included Dr. Boris Beizer, Dr. Bob Birss, Mr. Bill Bently and Mr. Rob Schultz.

The keynote presentations were extremely interesting. John Marciniak brought the audience up to date on the recent activities of the National Software Council. Tom Gilb talked about his concept of "How to Get to Level 6" referring to the five levels of the popular SEI Capability Maturity Model for software. Lee Osterweil of the University of Massachusetts offered many challenges to the audience, and finally, Watts Humphrey of the SEI talked about his "Personal Maturity Model" and the success that he has had in its use.

From the many sessions and tutorials, it is clear that this conference is playing a key role in the software industry.
The 18th International Conference on Software Engineering (ICSE -18) was held March 27 through March 29, 1996, at the Technical University of Berlin in Berlin, Germany. A number of tutorials, workshops and technical committee meetings also took place in or near Berlin in conjunction with the main conference, thus the week of March 25-31 was dubbed "International Software Engineering Week." ICSE is considered the premier international conference in software engineering. The primary conference sponsors were the IEEE Computer Society's Technical Committee on Software Engineering (TCSE), ACM Special Interest Group Software (Sigsoft), and the German Informatics Society, "Gesellschaft fur Informatik". The selection of Berlin as the conference site was made to encourage attendance from Eastern Europe. The attendee listing from the main conference contains nearly 900 names, more than half from European countries. 350 people attended the fourteen tutorials and a similar number participated in the nine associated workshops.

The conference featured three keynote speakers: Tom DeMarco on "The Role of Software Development Methodologies: Past, Current and Future"; Dr. Anthony Hoare on "The Role of Formal Techniques: Past, Current and Future"; and Dr. Victor Basili on "The Role of Experimentation: Past, Current and Future." (It appears that the keynote titles were assigned, or at least the result of some collaboration.)

The first keynoter on Wednesday morning was Tom DeMarco, of The Atlantic Systems Guild. He described the 1980s as the methodology decade, and proposed that the 90s is the process decade. He identified four paradoxes of software improvement. 1) Every advance made makes the remaining work harder. We've solved the easy problems -- the accidents; what's left are the hard ones -- the essence. 2) Focusing on process increases risk aversion. 3) Reuse is happening, but not where software engineers expected it. The tools and utilities, such as word processors, spreadsheets, and GUI interface builders, that everyone relies on, are what are being reused, and what account for the greatest increase in productivity among software developers. 4) People don't notice or respond to slow changes as well as to rapid ones, therefore software value has incrementally gone down. DeMarco believes the next great discipline for software will be Risk Management, because dealing with the remaining essence involves nontechnical issues -- conflict management and resolution, mediation and negotiations. He identified the two great current debates among software professionals as: Are there any more silver bullets? Is the Capability Maturity Model (CMM) useful?

Thursday's keynote, on the topic of formal techniques, was given by Dr. Anthony Hoare, from the
University of Oxford. His alternate title was "How did Software get so Reliable without Proof?"

According to Hoare, projects begin to fail in requirements definition. The hard question that does not get asked, or get answered adequately, is how the new system will fit into the existing environment. Efforts at formalizing proofs of correctness, therefore, should be aimed at requirements, because it is easier to find errors in a line of reasoning than in a line of code. As to why software works without formal proof, his answer was that software engineers employ techniques similar to other engineering disciplines, specifically, over-engineering and building in redundancy, which is also known as defensive programming.

Dr. Victor Basili, of the University of Maryland, gave the final keynote on the topic of experimentation. He also made comparisons between software and other scientific disciplines, but noted that software differs because it is about development rather than production. Software does not yet have a good observational knowledge base, yet an understanding of the variables is what will enable us to incorporate risk management into software development efforts. In response to a question about how to encourage Ph.D. students to replicate work, if degrees are awarded on the basis of originality, Basili responded that the act of replication is also a research problem, and we can learn from replication, too, in the form of new insights.

The conference concluded with a final panel session that provided a lively discussion from the audience as well as among the panelists. The provocative title, "If software is so bad, why does it sell so well?" was also restated as "Software Crisis: where is it?" The panelists included Wladyslaw Turski, chair, Manfred Broy, Lee Osterweil, David Parnas, Tony Hoare, and Tom DeMarco. The implication of the topic is that software as an industry is doing well, despite the shortcomings recognized by software engineers (and others). One explanation for the success of software in spite of its quality, from Parnas, is that there is no alternative. Others included: low expectations of software consumers, built-in redundancy, work-arounds, and that we're good at maintenance (bug-fixing). This was echoed by DeMarco, who claimed we've trained our customers to expect a maintenance phase, so that V2.0 and V3.0 of any product are remarkably better than V1. Hoare contended that software sells because the hardware is so good.

The rest of the conference included sixteen paper sessions, containing 52 technical papers in many subdisciplines of software engineering. Sessions on topics related to testing were the most numerous, followed by design issues and methods. Also represented were requirements, measurement, quality, maintenance, and configuration management. Other sessions could be classified as process-related, and several focused on specific techniques. Components showed up in several contexts, from reuse to object-oriented applications. In addition to the paper presentations, there was an industrial experiences track, in which software developers from industry were invited to present examples and case studies of software engineering techniques in practice, and a series of mini-tutorials, which presented particular software technologies, such as methods or tools.

**Issues and Trends**

An interesting and valuable feature of the conference was the ICSE newspaper "Window on the World (WOW)." The paper not only reported highlights of the presentations, but also provided a vehicle for rebuttals, and thus captured more of the spirit of the discussions than is preserved in the official proceedings. The three issues of WOW that were published during the conference are available on the World Wide Web at [http://www.cs.tu-berlin.de/~icsewow/](http://www.cs.tu-berlin.de/~icsewow/).
Although ICSE is aimed at both researchers and practitioners, participation from academia, in both attendance and papers submitted, far outweighs the industrial representation. This generated much discussion, and is being addressed by the ICSE'97 organizing committee. The differences are partly due to the emphasis that is put on publishing and professional society activities in university settings compared to within organizations that are concerned with product development and profitability.

There was surprisingly little discussion among the participants about the World Wide Web as a phenomenon of interest. It was, however, apparent that this community uses the Web extensively, and is beginning to rely on it as a basic tool. Many included WWW addresses in their presentations, which implies that information about current research activities is being made available on the Web. Access to this information is no longer limited to conference attendees, nor is the level of detail limited to what can be distilled into published papers.

A discussion of the implications of using the Web as a distribution medium for software came up in the last panel session. For some, this illustrates how the quality of initial releases of new software products is decreasing -- everyone becomes a beta tester, using Version 0.X. For others, this is a sly way to not only get feedback from potential customers, but also get them hooked. Others debated the impact of distributing free software -- how can users complain about the quality of a product, when they haven't paid for it, and how much responsibility can developers be expected to take for supporting a product they're giving away?

Information on the next ICSE, to be held in Boston in May 1997, can be found at http://www.ics.uci.edu/pub/icse97/. Abstracts from the proceedings of ICSE are also on the Web, at www.computer.org/conferen/proceed/icse96/abstract.htm.
Cleanroom Software Engineering: Management and Technology

Course Date: - To Be Decided -

Presented By: Data & Analysis Center for Software

Instructor: Richard C. Linger

COURSE OBJECTIVES

● To understand Cleanroom software engineering as a practical process for developing high reliability software with high productivity
● To understand the process of Cleanroom management through the incremental development life cycle
● To understand the technologies of Cleanroom specification, design, correctness verification, statistical testing, and reliability certification.
● To understand the integration of Cleanroom with the SEI Capability Maturity Model for Software.

COURSE OUTLINE

● Cleanroom Overview and Results
● Cleanroom Management through Incremental Development
● Cleanroom Technologies
● Function Specification and Design with Box Structures
COURSE THEMES

- Risks can be Managed or Avoided through Incremental Development and Disciplined Engineering Processes
- Software Fitness for use can be Scientifically Certified
- It is Possible to Routinely Develop Software that Approaches Zero Failures in Use

INTENDED AUDIENCE

- Software Project Managers
- Software Development and Maintenance Managers
- Software Engineers in both New Development and Evolution of Legacy Systems
- Software Quality Control and Acquisition Personnel

ABOUT THE INSTRUCTOR

Richard C. Linger is a Visiting Scientist at the Software Engineering Institute (SEI) from Lockheed-Martin Federal Systems Division. He is responsible for defining the integration of Cleanroom software engineering with the Capability Maturity Model (CMM) for Software, and assisting the US Air Force in implementing Cleanroom technology. Mr. Linger was a co-developer of the Cleanroom process at IBM, and founded and managed the IBM Cleanroom Software Technology Center. He has published two software engineering textbooks and numerous book chapters and technical papers. Mr. Linger is a member of the ACM and IEEE, and serves as vice president of the National Software Council.
LOCATION

**NTU Satellite Broadcast**

**On-Site Option Available** Contact the DACS Customer Liaison (listed below) for details.

Video tapes of this course may be purchased through NTU.

**For further information please contact:**
NTU Coordinator: Tom McCall
(970) 495-6444