110039

N95-16454

SOFTWARE ENGINEERING FROM A LANGLEY PERSPECTIVE

by Susan Voigt

This presentation is intended to provide a brief introduction to software engineering to set the stage for the panel discussion and some of the workshop presentations.

The talk is organized into four sections, beginning with the question "What is Software Engineering?" followed by a brief history of the progression of software engineering at LaRC in the context of an expanding computing environment. Several basic concepts and terms are introduced, including software development life cycles and maturity levels. Finally, some comments are offered on what software engineering means for LaRC and where to find more information.

In an article in the ACM Computing Surveys in 1978 (Vol. 10, No. 2, p. 197), Marvin Zelkowitz defined software engineering as the "process of creating software systems." (Note: ACM is the Association for Computing Machinery.) The IEEE Standard 610.12-1990 (Standard Glossary of Software Engineering Terminology) has a widely accepted definition that effectively is the application of an engineering approach to software.

The term "software engineering" was used at NATO conferences in 1968 and 1969, but became commonplace in 1975 when the first national conference (which became international at the second) was held in Washington, D.C. In that same year, the IEEE began publishing the journal: IEEE Transactions on Software Engineering. NASA started funding software engineering research as part of the Computer Science Research Program in the Office of Aeronautics and Space Technology in 1983. The Department of Defense was also concerned with "the software problem" in this time frame, and in 1984 the Software Engineering Institute was established at the Carnegie Mellon University. NASA's Office of Safety and Mission Assurance (Code Q) established the NASA Software Engineering Program in 1991, with funding for and active participation from LaRC.

Just as software engineering was developing, our computing environment was becoming more dispersed. In the 1960s, computing was done by computing professionals in a "closed shop" environment. However, by the 1970s, FORTRAN was used by researchers across the Center, and they had access to the centrally located computer facility by using the "green tub" service for pick up and delivery of punched cards and printed output (also called computer listings). In the mid-1970s, microprocessors and time sharing came to LaRC, providing remote computing capability. Computing expanded in the 1980s with distributed systems, personal computers, and data acquisition and/or control systems in many facilities. The 1990s has brought even more powerful workstations and networked systems. This changing environment has decentralized the computing and software development at the Center, so that software is now created in many organizations, with little coordination or collaboration.

One of the fundamental concepts in software engineering is that of life cycle. The life cycle is a way to capture the schedule and discipline of key activities, reviews (such as system design, requirements review and design review), and deliverable items at specific points in time. The Department of Defense has identified three "program strategies" in their recent standards, that illustrate classic software life cycles: waterfall, incremental and spiral.

The Grand Design strategy assumes a complete definition of the requirements prior to design. The waterfall life cycle includes the development phases: requirements analysis, design, coding, test and integration and finally operations and maintenance. As each phase is completed, products are delivered that support the next phase.

The Incremental strategy is also called "preplanned product improvement.". The user needs and system requirements are defined followed by a phased development with several releases or system builds. Each phase includes the typical steps in the waterfall process. Experience with early releases in the incremental approach can provide refinements for subsequent releases, along with the new capabilities planned.

The Evolutionary strategy is based upon Barry Boehm's spiral model (described in ACM Software Engineering Notes, Vol. 11, No. 4, Aug. 1986, pp. 14-24; and IEEE Computer, May 1988, pp. 61-72). This approach encourages consideration of risks, constraints and alternatives. The software development occurs in the third quadrant of the spiral, and is similar to the incremental development.

The Software Productivity Consortium (Lockheed, one of our support service contractors, is a member company) has extended the spiral model into the Evolutionary Spiral Process (ESP) Model with extensive training and guidebook materials available to SPC members (and to NASA, as a Lockheed customer).

The Software Engineering Institute (SEI) has defined the Capability Maturity Model (CMM) that can be used to identify how an organization can improve the maturity of its software process. The CMM has five levels, from initial to optimizing. Watts Humphrey, SEI fellow, is considered the author of the CMM. We do have copies of SEI provided documentation on the CMM in the Space Systems and Concepts Division. The lowest level (1) is when software development is informal and each job is only as good as the individual software developer. This is the stage when good software results from heroic effort. Level 2, called "repeatable," is more intuitive, where there are some common practices, but problems invariably arise when something new is introduced into the process. The focus at level 2 is on project management. The "defined level" (3) is qualitative and focused on the engineering process. The process has been written down, and the organization has accepted it as common practice. Training in the process is available, providing continuity with personnel turnover, and the staff meets regularly to discuss improvements. The quantitative or "managed level" (4) has measures in place to track productivity. The focus is on both product and process quality. The process is understood and managed so that bottlenecks can be identified and automated tools can implement parts of the process to reduce human error. When an organization has achieved the "optimizing level" (5), detailed metrics on the process are collected, problems can be anticipated, there is constant process improvement, and new technology can be infused. A level 5 organization is practicing TQM in software development to the full extent. At the present time, most organizations are at level 1 or 2.

The SPA and SCE are two assessment methods defined by the SEI. SPA, the Software Process Assessment, is used by an organization to assess their own actual process maturity and develop a software process improvement strategy. It is only for internal use. SCE, the Software Capability Evaluation, is more like an audit. It is used to gather information on the software process maturity of organizations that might be competing for a software task. Several government agencies are using SCE's in their source selection process. Our panelist from the Naval Surface Warfare Center has been trained in the SCE, and she will share some insights on this later. An analogy to compare the SPA and SCE: An assessment is like having a friend or relative help you prepare your income taxes (it's internal), whereas an evaluation is like having the IRS do an audit of your taxes.

Some other basic concepts of software engineering can be introduced by defining some jargon. CASE (computer aided software engineering) is a generic term to describe tools and environments that provide automated support for software development. The DOD has used CSCI (computer software configuration item) to describe major software modules

(that are kept under configuration control). Submodules are called Computer Software Components (CSC) and often compilation units are called Computer Software Units (CSU). CM stands for configuration management, a process for identifying and for controlling release and change of software items. Object Oriented Design (OOD) and object oriented programming are an alternative approach to procedural-oriented software architecture, treating programs and data as objects. IV&V is Independent Verification and Validation, the testing of software functionality and validation against requirements performed by a team separate from the developers. Software Quality Assurance (SQA) is an activity performed throughout the life cycle to assure that requirements analysis, design, code, and the resulting product satisfy the software requirements.

Additional jargon includes SMAP, which was the Software Management and Assurance Program led by the NASA Office of the Chief Engineer and later the Office of Safety, Reliability, Maintainability, and Quality Assurance (Code Q) in the 1980s. The SMAP team included representatives from all NASA Centers, and they helped define the NASA software documentation standards that have evolved to NASA STD-2100-91. The SMAP has been replaced with the Software Engineering Program in the current Code Q, Office of Safety and Mission Assurance. DID stands for Data Item Description, the Department of Defense (DOD) term used for software documentation format, instructions and outline. For example, the DOD-STD-2167A describing the current Defense System Software Development standard, contains at least 16 DIDs. The DOD program Software Technology for Adaptable, Reliable Systems, called STARS, has been active for over 10 years, and is the focus of considerable effort in areas including Software Engineering Environment (SEE) and Software Reuse. Research into reusing software assets (e.g., design and code segments) has included identification of domains or classes of application areas with common aspects where reuse makes sense.

Since the daily work at LaRC relies on software more and more, and as more emphasis is placed on the transfer of technology (which includes our software products), there is a need to pay more attention to the engineering of our software. There are several resources available to people at the Center, including the Software Engineering and/or Ada Laboratory (SEAL) in the Information Systems Division, an Inter-Group N-Team on Software Productivity, Quality, and Reliability led by Robert Estes, and Internet access to many information resources. The recently formed Hampton Roads Software Process Improvement Network (HRSPIN) offers additional opportunity for professional development and information exchange with individuals from government, industry and academia interested in software improvement. The Technical Library (as well as many individuals) have several of the software journals of particular value to the software engineering specialist.

There are several standards that also are applicable, and these can prove useful in guiding a software process. Experienced software engineers at NASA Langley are willing to share their knowledge, and the SPQR N-Team provides them an opportunity to network and work together to improve the quality of software at the Center.

Software engineering techniques can improve the software products developed for and by LaRC. The panel represents several perspectives on software development, and these experienced software developers and managers are willing share some of their views on where we are and where we should be going.

SOFTWARE ENGINEERING

by

Susan J. Voigt
Space Systems and Concepts Division, SASPG

Presented to
The Role of Computers in LaRC R&D Workshop

June 15, 1994

Outline

- What is Software Engineering?
- A Brief History from LaRC Perspective
- Introduction to Some Basic Concepts
- What does this mean for LaRC?

What Is Software Engineering?

- "Process of creating software systems"
  (Zelkowitz, 1978 Computing Surveys)

- "The application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"
  (i.e., the application of engineering approach to software)

  IEEE Std 610.12-1990

A Brief History - LaRC Perspective

- 1968/69 "Software Engineering" used at NATO Conferences
- 1973 Structured Programming in vogue
- 1975 First National (Int'l) Conference on Software Engineering
  IEEE Transactions on Software Engineering journal started
- 1983 NASA Computer Science Research Program funded (with Software Engineering component)
- 1984 CMU/Software Engineering Institute Established by DOD
- 1991 NASA Code Q started Software Engineering Program

Progress Of Computing At LaRC

- In 1960s & early 70s, Programming done in ACD (Closed Shop)
- 1970s - FORTRAN used by researchers
  Green Tub service (Open Shop)
- Mid-70s - Time Sharing in Central Facility
  Microprocessors in Labs
- 1980s - Distributed systems, PCs, and automated facilities
- 1990s - Networked workstations, virtual systems

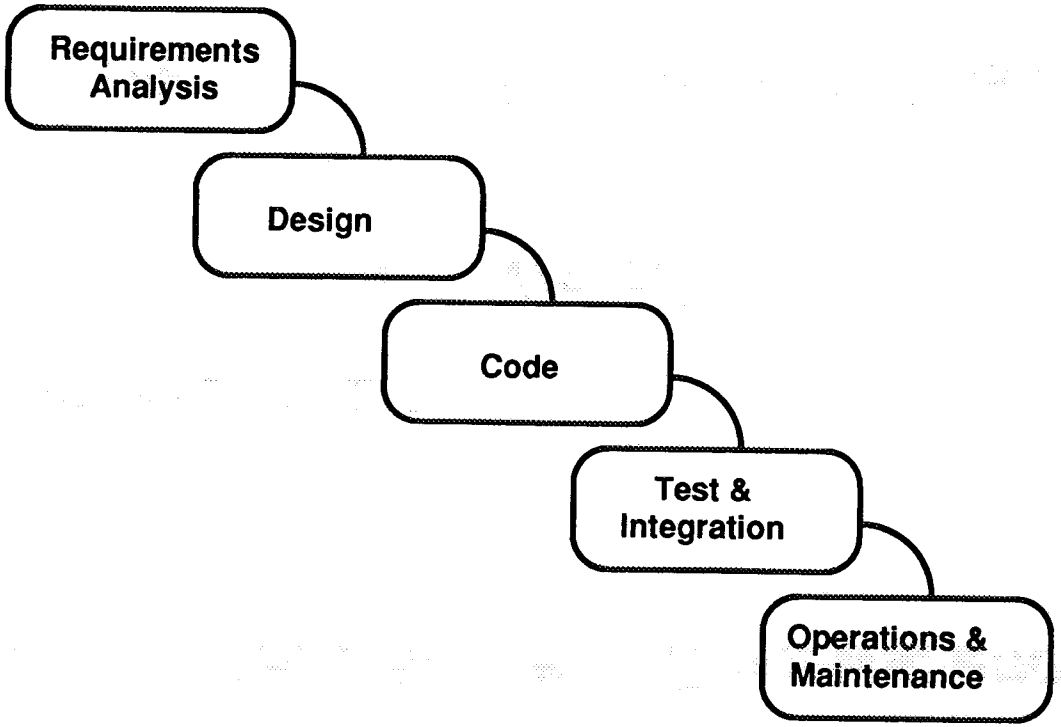
Software Life Cycles

Schedule of Activities, Key Reviews, and Deliverables

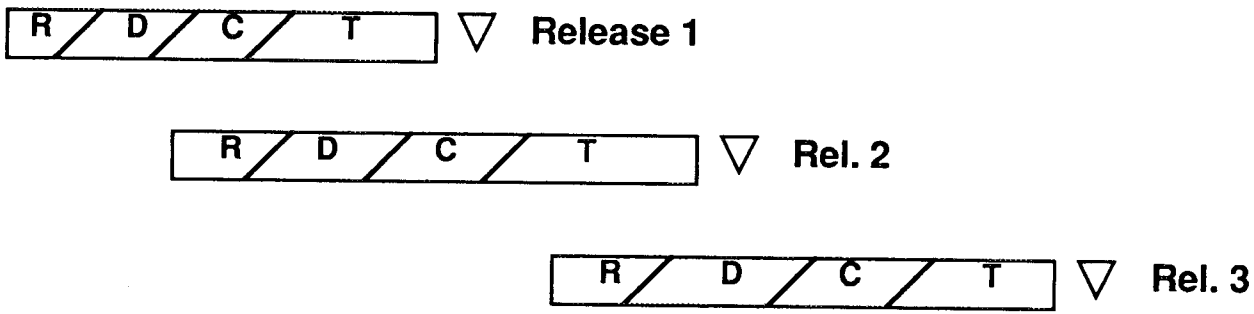
DoD Program Strategies

- **GRAND DESIGN** — **Waterfall**
- **INCREMENTAL** — **Preplanned Phased Development**
- **EVOLUTIONARY** — **Spiral Process**

Waterfall


Requirements Analysis
Design
Code
Test & Integration
Operations & Maintenance

Incremental

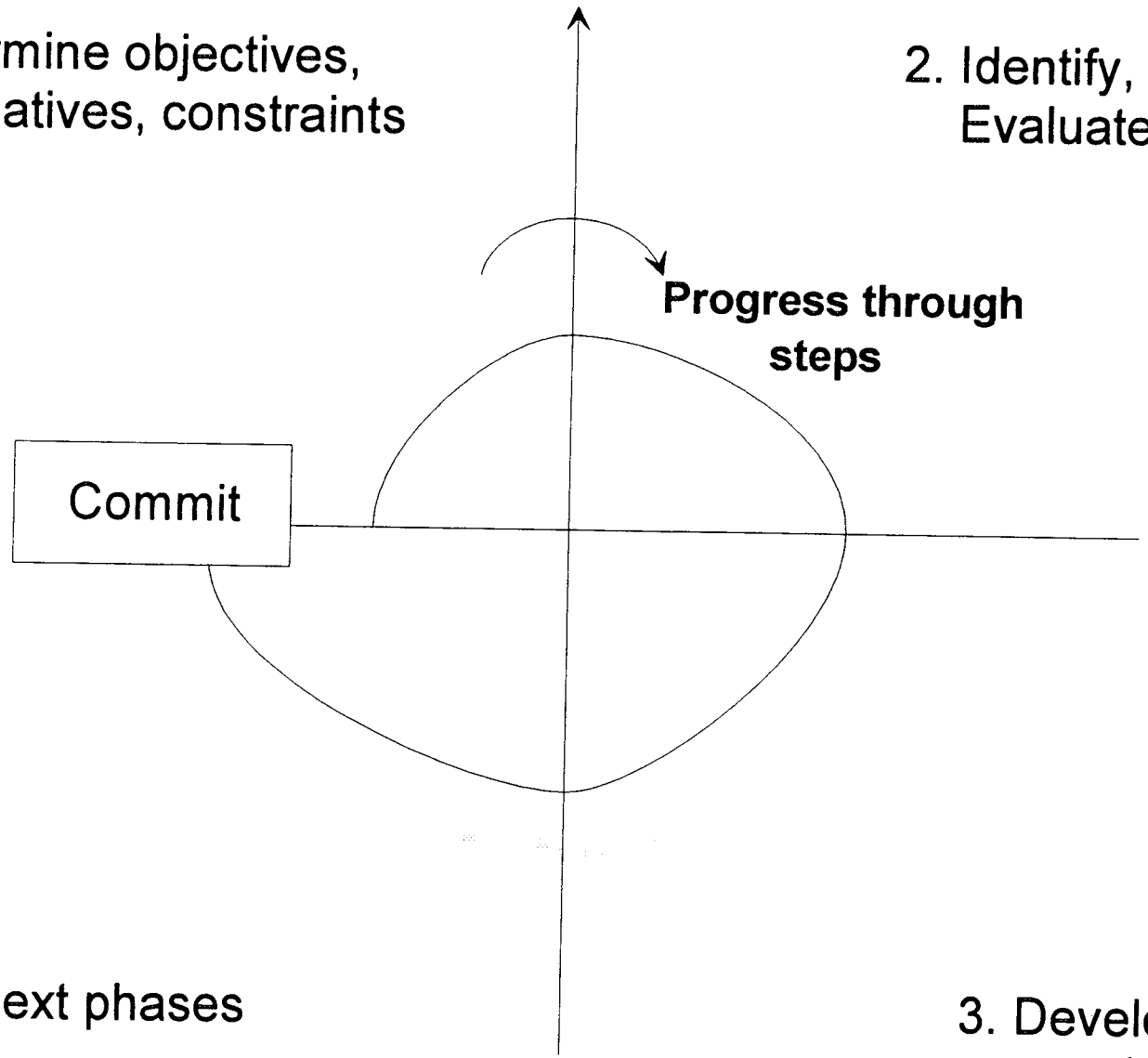


R - Requirements
D - Design
C - Code
T - Test & Integ.

Boehm's Spiral Lifecycle Model

1. Determine objectives, alternatives, constraints

2. Identify, resolve risks
Evaluate alternatives



4. Plan next phases

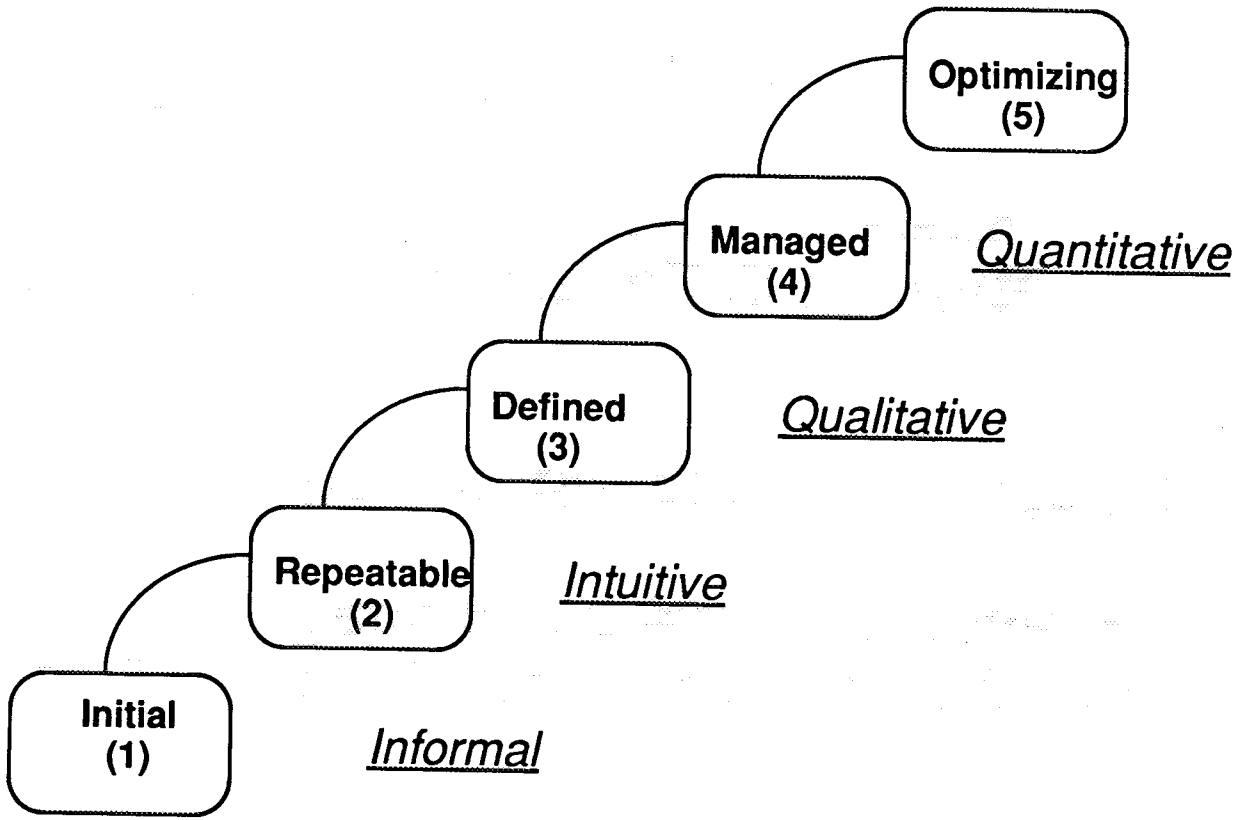
3. Develop, verify next-level product

Software Process Models

- SPC Software Productivity Consortium
  (Lockheed is a member company)
- ESP Evolutionary Spiral Process Model

- SEI Software Engineering Institute
  at Carnegie Mellon University
- CMM Capability Maturity Model
- SPA Software Process Assessment
- SCE Software Capability Evaluations

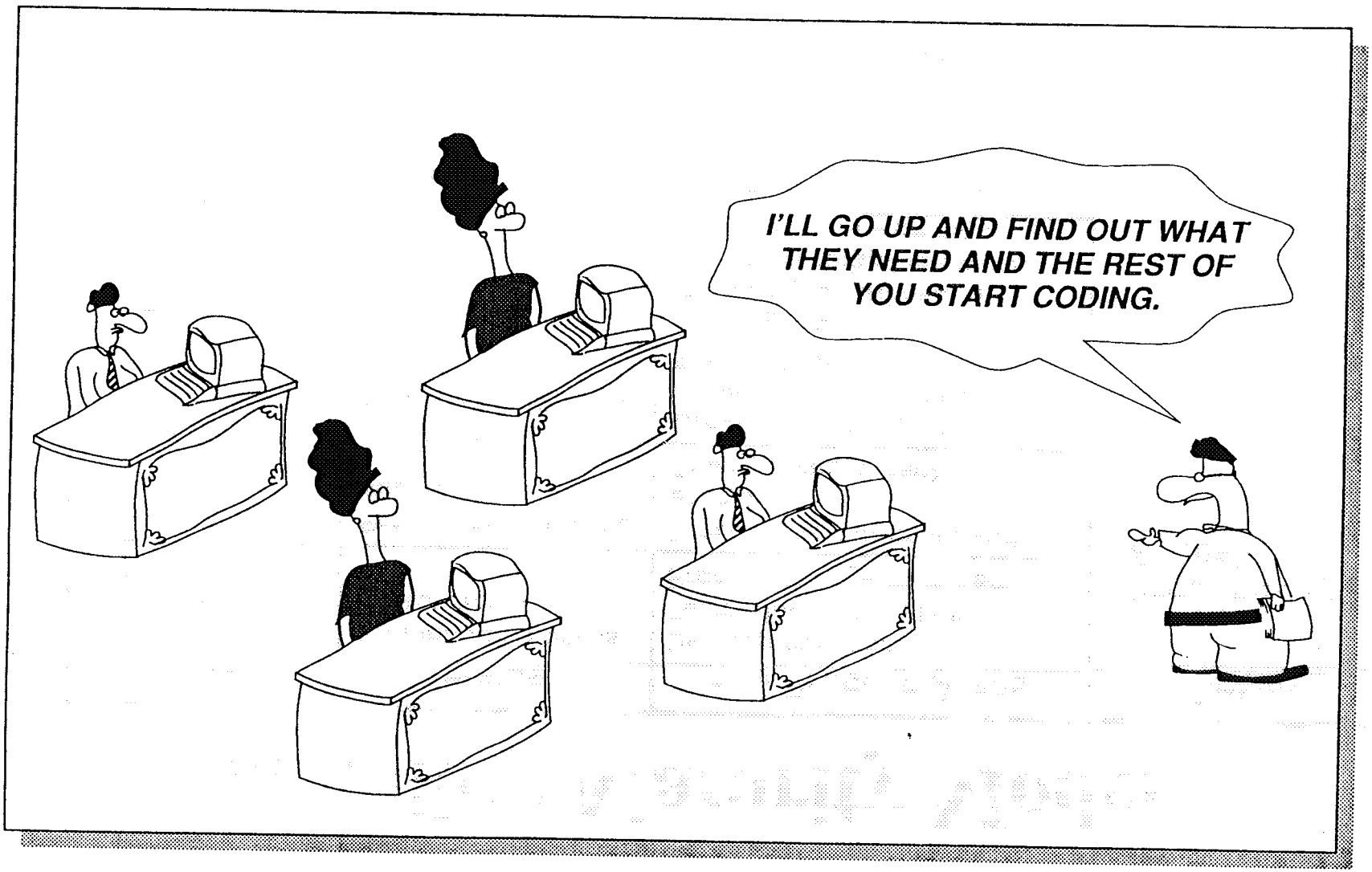
FIVE MATURITY LEVELS



Capability Maturity Model

Level	Focus	Key Process Areas	Result
5 Optimizing	Continuous process improvement	Defect prevention Technology innovation Process change management	Productivity & Quality
4 Managed	Product and process quality	Process measurement and analysis Quality management	
3 Defined	Engineering process	Organization process focus Organization process defn. Peer reviews Training program Intergroup coordination Software product engineering Integrated software mgt.	
2 Repeatable	Project management	Software project planning Software project tracking Software subcontract mgt. Software quality assurance Software configuration mgt. Requirements mgt.	
1 Initial	Heroes		Risk

Characteristics of an Initial Level Organization



An Analogy

- An *Assessment* is like asking your brother-in-law to help you prepare your income taxes

- An *Evaluation* is like having the IRS do an audit of your taxes

Some Software Engineering Jargon

- CASE — Computer Aided Software Engineering
- CSCI/CSC/CSU — Computer Software Configuration Item/Component/Unit
- CM — Configuration Management
- OOD — Object Oriented Design
- IV&V — Independent Verification and Validation
- QA or SQA — Software Quality Assurance

More Software Engineering Jargon

- **SMAP** NASA Software Management and Assurance Program (1980s)
- **DID** Data Item Description (Document Format)
- **STARS** DoD Software Technology for Adaptable, Reliable Systems Program (1984 - present)
- **SEE** Software Engineering Environment
- **Software Reuse** Incorporation of previously developed software products
- **Domain** Category or Application Area

What Does This Mean For LaRC?

- Increasing reliance on Software in Daily Work
- Technology Transfer Focus => Software Releases
- Several Activities and Resources are available
  - ISD/SEAL (Software Engineering & Ada Lab)
  - SPQR N-Team
  - Mosaic/WWW Information
  - HRSPIN (Hampton Roads Software Process Improvement Network)
  - Journals (IEEE Software, IEEE Trans on SE, ACM SIGSOFT, ...)

Standards can be Resources

- NASA-STD-2100 Software Documentation Standard
- ISO 9001 Quality System Standard
  A NASA Standard
- ISO 9000-3 Software Quality Guidelines
- DOD-STD-2167A Defense System Software Dev
- MIL-STD-498 Software Development and Doc
- MIL-STD-499B Systems Engineering
- many IEEE Standards

SUMMARY

- Software Engineering is necessary for better software products
- Experienced Software Engineers do work at LaRC
- We need to improve our software quality in the spirit of "faster, better, cheaper"
- Join the LaRC Software Productivity, Quality, and Reliability N-Team !

Summary of Panel on Perspectives on Software Development

The panel consisted of five NASA Langley employees representing different application domains and a representative from the Naval Surface Warfare Center in Virginia Beach, VA. Each panelist began with a short statement reflecting both experiences and perspectives on software development. The panelists, their application domain area, and organization were:

Chuck Niles	Facilities Software, IOG
Pat Schuler	Flight Software, IOG
Tom Zang	Researcher Software, RTG and LCUC Chair
Pam Rinsland	Embedded Systems Software, IOG
Peg Snyder	Science Software, SASPG (retired)
Brenda Zettervall	Software Quality Improvement, Naval Surface Warfare Center
Susan Voigt	Moderator, SASPG

Chuck Niles is in the Electrical and Electronic Systems Branch in the Facilities Systems Engineering Division of the Internal Operations Group. He has 15 years of experience in software development for wind tunnel control systems, process monitoring, and ground facilities communications on minicomputers and the whole family of Intel microprocessors. He is responsible for software configuration management for many wind tunnels at LaRC and has developed documentation for all phases of software development. His opening remarks, "Perspectives on Software Development," are included following this section.

Pat Schuler is in the Advanced Computer Systems Branch in the Information Systems Division of the Internal Operations Group. She began her Langley career providing support for scientific research computer applications. She was software manager for the first embedded systems (flight software) project in the Software Engineering and Ada Laboratory (SEAL). Since the SEAL was formed, she has been active in developing it as a center of excellence in software engineering at LaRC, with support from the NASA Office of Safety and Mission Assurance (Code Q). In this discussion, Pat represented the flight software for Langley scientific instruments.

Pat cited three characteristics of flight software development: embedded systems, distributed processing, and real-time. She went on to clarify these as follows:

> Embedded systems - A specialized computer with custom-programmed software used to control functions within the device it's controlling.
>
> Distributed processing - A system in which tasks to be performed by the available computing resources are executed by a number of processors, often in parallel.

Real-time - Results are calculated in sufficient time to guide the physical process under control.

She cited four typical examples of space flight projects at LaRC: CERES, JADE, LITE, and MIDAS with flight life-times ranging from 11 days to a few months to 5 years, and flight code size ranging from 2K to 18K (where K represents 1000 source lines of code). In addition to on-board flight software, ground support software, including simulators, test subsystems and mission operations subsystems must be developed, and these range from 2 to 10 times the size of the flight code. The SEAL has standardized on Microsoft Windows and other MS software, Ada, object-oriented design, formal inspections, and Novell as their local area network for internal mail and a shared group calendar. The SEAL tools, based on PC and Intel, are considered a Center resource. The SEAL is also trying to baseline their software development process and document it in guidebooks. They also are collecting metrics on how software is developed in the SEAL. SEAL personnel provide consultation to and arrange training for other groups at the Center in software engineering processes and tools, but they do have a limited staff. A list of the tools and software documents available from SEAL follows this section. Anyone interested in learning about the tools, their use, and related training should contact her.

Tom Zang is head of the Multidisciplinary Design Optimization Branch in the Fluid Mechanics and Acoustics Division of the Research and Technology Group. He also is the chair of the Langley Computer Users Committee (LCUC). He represented researcher software on the panel.

Tom said that the LCUC intends to reorganize itself in the fall to align with the new Center organization. The LCUC was set up about 20 years ago to provide a voice for user concerns and desires to the Analysis and Computation Division for short-term tactical and some long-term strategic planning.

The two products from research are reports and software. However, managers and researchers simply do not recognize the importance of their software as a technical product. He observed that NASA encourages the quality aspects of technical reports, but not of software. Four types of software products are produced by researchers at LaRC: concepts, portable modules, pilot codes, and production codes. The concepts may include new algorithms and these are published in technical reports. Modules are usually available as commented code. Pilots are prototype software for early release with caveats since it is not thoroughly tested, still may be in development, and has little documentation. Production codes are well written and well documented computer programs. A good example of multidisciplinary code at LaRC is FIDO (described by Bob Weston at a later session at the workshop). In closing Tom stated he would like to see management place greater value on good software, researchers write their

software for others as well as themselves, and software engineers act as a resource for others at LaRC. He did note that software engineering is included on the list of necessary skills in the Research and Technology Group (RTG).

In these proceedings, he has included a few charts from the LCUC files of a 1980 briefing by Jarek Sobieski which cite some of the same issues. A copy of Tom's transparencies "Perspectives on Software Development" are included following this section.

Pam Rinsland is the assistant head of the Electronics Systems Branch in the Aerospace Electronics Systems Division of the Internal Operations Group. In her 22 years at LaRC she experienced the transition from the batch-oriented central computing and plotting without preview to the "instant gratification" of time-sharing. She has developed software for a wide range of aerospace applications, including writing code to execute on computers ranging from Intel's first 4-bit processor to the first supercomputers delivered to LaRC. In her current position, she is in a hardware-oriented branch and promotes her firm beliefs in the absolute necessity of close ties between hardware and software specialists, and in maintaining discipline in the software development process.

Her opening remarks, Reflections from a "Jurassic Programmer" on Software Development at LaRC, follow this section.

Peg Snyder, prior to her retirement from NASA in May 1994. was in the Data Management Office in the Atmospheric Sciences Division of the Space and Atmospheric Sciences Program Group. She has 31 years of experience in software development at NASA, starting at Lewis Research Center with FORTRAN code on a mainframe with 30K of 36-bit words (memory) for basic research in nuclear physics scattering analysis and non-steady fluid flow. An early lesson she learned was to number your punched cards (artifacts now found in the museums in Washington, DC). She worked for several years in the Space Station Freedom Program Office prior to coming to LaRC 3 years ago. Her software experience ranges from office automation software and space applications to wind tunnel applications, data reduction, and CERES data processing.

Peg's most important message to the audience was that best results are obtained when an engineering approach is applied in the development of software. Specifically, her approach has six steps: 1) Define the problem (in the 1960s an engineer would bring a notebook to the programmer with "requirements" documented); 2) Figure out how to solve (reformulate the problem in terms of mathematics and select appropriate numerical analysis techniques); 3) Design the solution; 4) Implement the solution; 5) Test; 6) Use and maintain. We actually practiced more software engineering back in the batch days than we do

now. A second important message was that automated tools are only useful if they help you implement a process already in place.

Brenda Zettervall is Quality Improvement Administrator for East Coast Operations of the Port Hueneme Division of the Naval Surface Warfare Center located at Dam Neck in Virginia Beach, Virginia. She has 18 years of experience in software development including land-based integrated combat simulation programs and systems engineering necessary to translate operational requirements into simulation performance requirements. She is a member of the Software Engineering Institute (SEI) Capability Maturity Model (CMM) Advisory Board and the CMM Based Appraisal Review Group. She also is qualified to perform Software Capability Evaluations. She is the first chair of the recently formed Hampton Roads Software Process Improvement Network (HRSPIN).

Three years ago, Brenda became involved in quality improvement as part of a competition between Naval support centers and between the Navy and AF for software post-deployment support. Since the Navy is down-sizing and decommissioning many ships, software process improvement was necessary for survival since most of the systems supported at Dam Neck were on the "hit list." Being able to maintain cost and schedule is highly dependent on the maturity of the process in place. Hence they have embarked on establishing a management discipline for software development and maintenance. This means their process is documented, trained and enforced. The Navy is challenged to survive and to improve their software engineering process, since the Air Force has a vision to do all software engineering for the Department of Defense.

Questions from the Audience and Panelist Answers:

Q) Other engineering disciplines are based on mathematics. What is the basic science on which software engineering is based?

A) Mathematics is the basis for formal methods and algorithms such as rate monotonic scheduling.

Q) Suppose your organization were in charge of developing software for the next generation aircraft. Would you fly on it?

A) Four panelists said "Yes" and two said that flight critical software was outside their domain, and their organizations did not have the appropriate expertise and training.

Q) How will the software development process have changed 10 years from now?

A1) We will be doing it at home.
A2) Researchers will write code from day one using good practices - even if it is just "for themselves".
A3) We hope to raise our organization to higher maturity levels, hopefully close to a CMM level 5 and the Center to level 3 or 4.
A4) Necessity is the mother of invention. In the 60's there were incentives to make programs work smarter (e.g., you could be called in the middle of the night about your wind tunnel software if it didn't work properly). Things are changing, so we will be forced to be more rigorous.
A5) There will be a trend toward graphical programming models, and off-the-shelf packages available for control systems. There will be "6th generation programming languages".
A6) We will be rewarding people for good software engineering practices (activity will not be confused with productivity).
A7) Rapid prototyping and workstation platforms will be common.

Q) Where is a good place for Software Engineering at LaRC? In an N-team, a Branch or a Group?

A1) Software people are throughout the Center and there is no central focus or mechanism for software developers to exchange ideas and information except in the N-team. In a closed shop (as in the 1960's) professionals sat closely together and could share ideas and software. The Software Productivity, Quality and Reliability (SPQR) N-team is a good place for professional sharing.
A2) The Information Systems Division has a business thrust in Software Engineering and it is focused in the SEAL, the LaRC Center of Excellence in software engineering encouraged and supported by the Code Q Software Engineering Program. (The GSFC Software Engineering Laboratory just won the first IEEE Software Process Award; JSC has a Software Technology Branch; JPL has the SORCE).
A3) Perhaps the Center should form a local SPIN (software process improvement network) or SEPG (software engineering process group) in addition to the N-team.

Q) More than half of the software is being developed by people who are not software professionals. Engineers, doctors, and lawyers often write their own code. I can't find good textbooks written by professionals. Do you share that view?

A1) Perhaps we can never get non-software professionals away from programming. Would it help to have more training in software engineering?

A2) The Information Super-Highway may be more of a threat to a disciplined approach than interactive programming!
A3) The SEI apparently is now hiring mathematicians rather than computer science graduates, going back to the basics.
A4) I would defend the engineer who writes his own code. We need better practices in software development so the researcher can do the software work.
A5) Everyone needs to work more closely with the customer. The research engineer and the programmer need to work closely together.
A6) We need to have more fundamental training for "FORTRAN-type" programmers (basic training for research and prototype software). Unfortunately, the training office doesn't like to repeat classes, which makes it difficult to offer basic classes to a wide audience.

Q) I build "Flight Systems" and there is electrical hardware that is not well-documented. Are we confusing software engineering with engineering as a discipline?

A) There is a difference between scientific research (prototyping) and systematic engineering (final product) software. Software engineering professionals should be involved with the final product.

COMMENT) We need to distinguish between scientific research and engineering development. Be careful not to compartmentalize or constrain research. I did the software development on one of my own mathematical models - it helped me to understand the problem.

In closing, Moderator Susan Voigt proposed 5 domains for software classification: flight software, facility software, ground support equipment software, management information systems, and research software (see Attachment). Also the intended use of software may affect its level of disciplined development: My use only, use within my work group at LaRC, Informal release outside LaRC, Beta release outside LaRC, and Formal release (e.g. COSMIC) outside LaRC.

Members of the audience were invited to comment on the domains and intended use categories and to join the LaRC software N-team if they were interested.

Perspective on Software Development

Charles E. Niles

What types of software do you develop? The domain is ground facility automation systems, specifically closed circuit and blowdown wind tunnels and research labs. Applications include control algorithms for test environment conditions (Mach number, Reynolds number, pressure, temperature), model support systems and other test articles (pitch, roll, yaw, Alpha, Beta), and high pressure air systems (pressure, temperature); process monitoring; operator interfaces; and utility functions such as data logging and sequence of events recording. Hardware systems include 80486-based microcontrollers, industrial PCS, PLCs, minicomputers, and combinations of these. Supporting systems include commercial analog and digital controllers, motion controllers, and servocontrollers.

Who are the users of this software? Facility operators, in support of LaRC and commercial aerospace researchers.

What is the life cycle (how long is the software used)? Indefinitely. Generally, the software is replaced during a CoF upgrade to computer hardware and control rooms.

Is there much maintenance or enhancement required? Steady work - correcting bugs and improving performance.

Maturity of Software Development - Software development, as we know it, has been around for 40-50 years. Software engineering, has been around since the early 1980's. Considering that other engineering disciplines have existed for 1900 years or so, the software world has come a long way. Understand though, that software engineering is still in its infancy. The point is that other engineering disciplines are not exact sciences and neither is software engineering. Only the laws of physics and the mathematics upon which they are based are.

Software Development Relative to Operating Systems - In recent years, more popular languages, notably C and C++, have advanced the portability of applications from mainframes, to minicomputers, to PCs, Macs, and workstations. Witness that different operating system platforms are capable of running the same application. However, the class of applications is generally confined to office automation tools. I believe that application software developers should be able to develop an application with no concern for the operating system it will run on. Of course, we have no universal operating system today and probably never will. But, the proliferation of operating systems and programming languages demands a consistent application programming interface. Currently, we have at least as many APIs as there are operating systems and hardware platforms to run them on. Perhaps, a universal API will emerge as the POSIX standards are developed.

Software Reusability - DoD mandated the use of Ada to promote reusability of source code, among other things. C++ was developed to foster the development of reusable class libraries and methods. Neither has accomplished this goal, and never will. **Problem - reusability is more trouble than it is worth**. How many of you have ever written a five-to-ten line routine to do something because there was not a library call to do it or because you could not afford the time to locate one ? ... How many of you have ever obtained source code that seemed to meet your needs but would not compile initially or did not execute as you expected ?... Individuals and small teams will usually reuse source code they have written themselves because they know where to find it and they know how it works - and it does not matter the language in which it is written - they will convert it, if necessary. But seldom will you go to another organization to find code that you need. Can you imagine Microsoft and Borland sharing source code ? Forget it. There are instances where commercial developers license software packages from other developers - it is less expensive than litigation. **Problem - new or unique software does not already exist**. An entirely new application can be based on a existing components (system calls, intrinsic functions, internally reused segments, etc), but they must be blended into a new overall package. Blending it all together to create a new application is still time-consuming, even if 50% of it is built from existing components... In my opinion, widespread software reuse will not happen on a nationwide basis and definitely not on an industry-wide basis. It is unlikely to be harnessed on a domain basis.

What is wrong with software developed at LaRC? What should we do about it?

1. Funding is always inadequate because of the politics involved in selling a facility modification project. The the higher the cost of a project, the less likely that it will be approved by HQ. When it is approved, the budget has been decreased too much to accomplish the overall job, let alone the software part. So, ultimately, we must complete the project in-house. The transition time for a 50,000-line job is not instantaneous. When we do a job in-house from its inception, the product is better, but the project takes longer because only minimal resources can be applied. The solution begins with properly planning a job and estimating the cost, including risk factors, and selling it for what it will cost, not for what management believes HQ will approve it.
2. Documentation is generally poor - it is usually outdated and incomplete. Face it, programmers like to write code, not documents. Programmers are extremely optimistic estimators. When they have used their allocated time getting their code to run, they do not have enough time to document anyway.
3. Management/customers do not understand the true costs of software. Most managers believe that software is something that comes on a set of disks or CD-ROM, costs $500, and has a life of 6-12 months, depending on when the latest revision is released. Management fails to recognize that the developer probably spent $1 million and 20 person-years to develop the initial release and must sell 200,000 copies to break even. Facilities automation personnel and/or contractor personnel, by comparison, have performed miracles with $200,000 and 4 person-years. Unfortunately, our products have been overshadowed by late delivery and hardware reliability problems.

How can software improvements be institutionalized at LaRC?
1. Define standards and the criteria for their applicability.
2. Train and equip developers better.
3. Apply newer, industry-proven techniques.

Aside: Software has improved. Consider that the applications we develop today are significantly larger and more complex than their predecessors. I suspect that most of us could rewrite some piece of software we developed a few years ago in less time and far more robustly. So, what was it that really improved ?

What data should be collected on software developed at LaRC and how should this data be used?
1. Description of the software - function, size, platform, language(s), etc.
2. Resources applied - personnel, cost, tools
3. Why it was developed - benefits
4. Techniques - requirements analysis, design, coding, testing
5. Lessons learned

Information of this type could serve two purposes. First, a project team could use it for guidance. Secondly, after some period of time, a committee could evaluate this database to establish recommended practices, identify common attributes across different domains, identify common problems and how to avoid them, develop cost/resource criteria for future software projects, etc.

How can we encourage problem (defect) reporting & collection at LaRC?
There are two distinct categories: pre-release and post-release. Pre-release is the responsibility of the person(s) testing the software. Since the programmer usually performs intial testing, any data is virtually meaningless. Post-release is the responsibility of the users. I have found that encouragement is generally not an issue in this case. Problems having potential safety impact - a portion of which are software related - are reported when a subsystem fails. Under the facilities Configuration Management program, the Facility Safety Head is responsible for reporting such problems. Problems are generally reported when a certain function of the software becomes important to a test. Generally, there is no mechanism to report problems specifically with software at LaRC.

What suggestions would you make for how we should be developing software at LaRC in the future? I believe an individual representing each software domain (i.e. Blue team) should visit the SEI or a major commercial developer, spend a few days observing, return, draft a software development handbook, obtain feedback from a different set of individuals representing each software domain (i.e. Red team), revise the handbook, publish it, and encourage management to enforce it.

Perspectives on Software Development

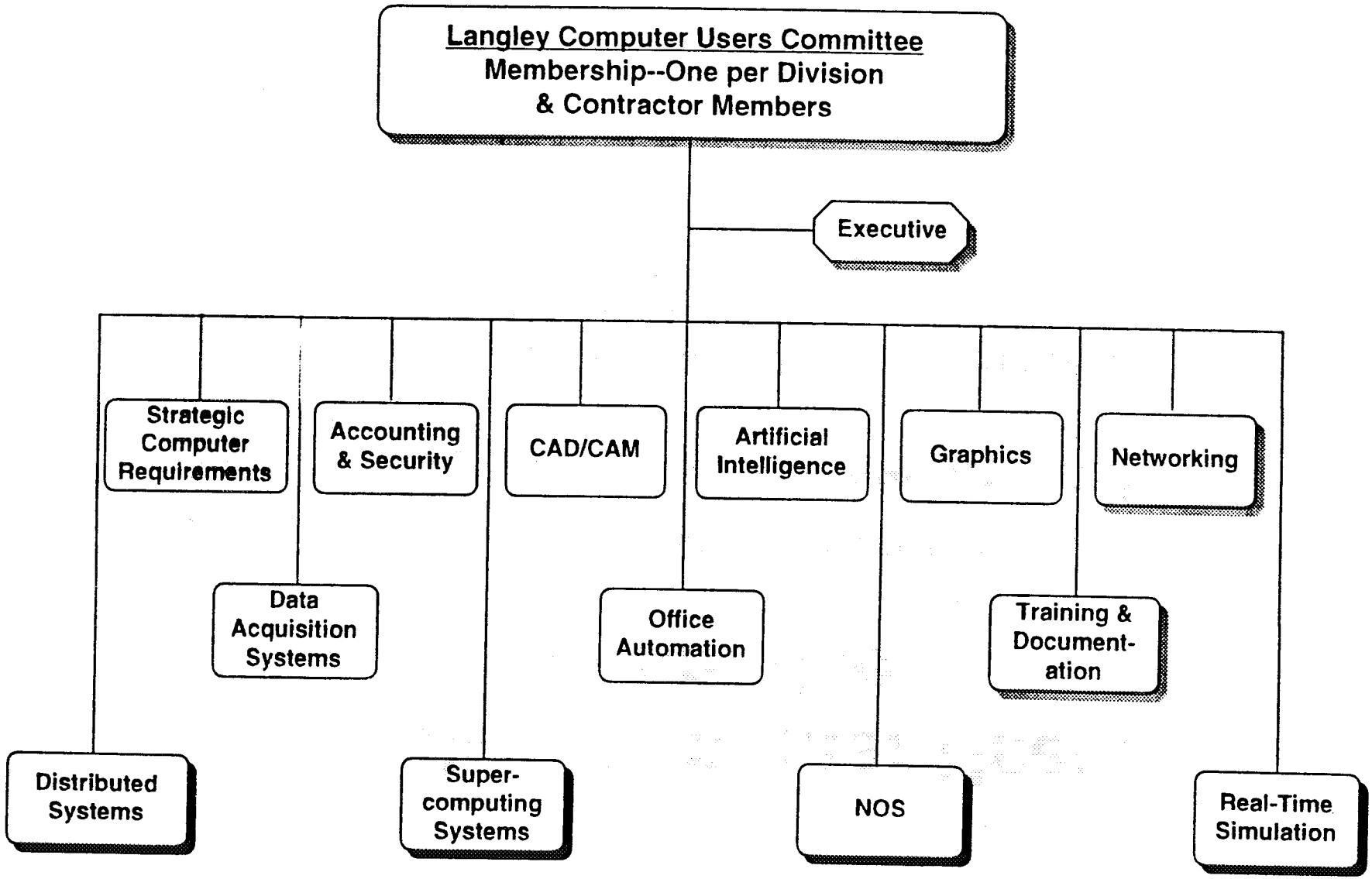
Thomas A. Zang
Chair, LCUC
Head, MDO Branch, RTG

CSTC Workshop
June 15, 1994

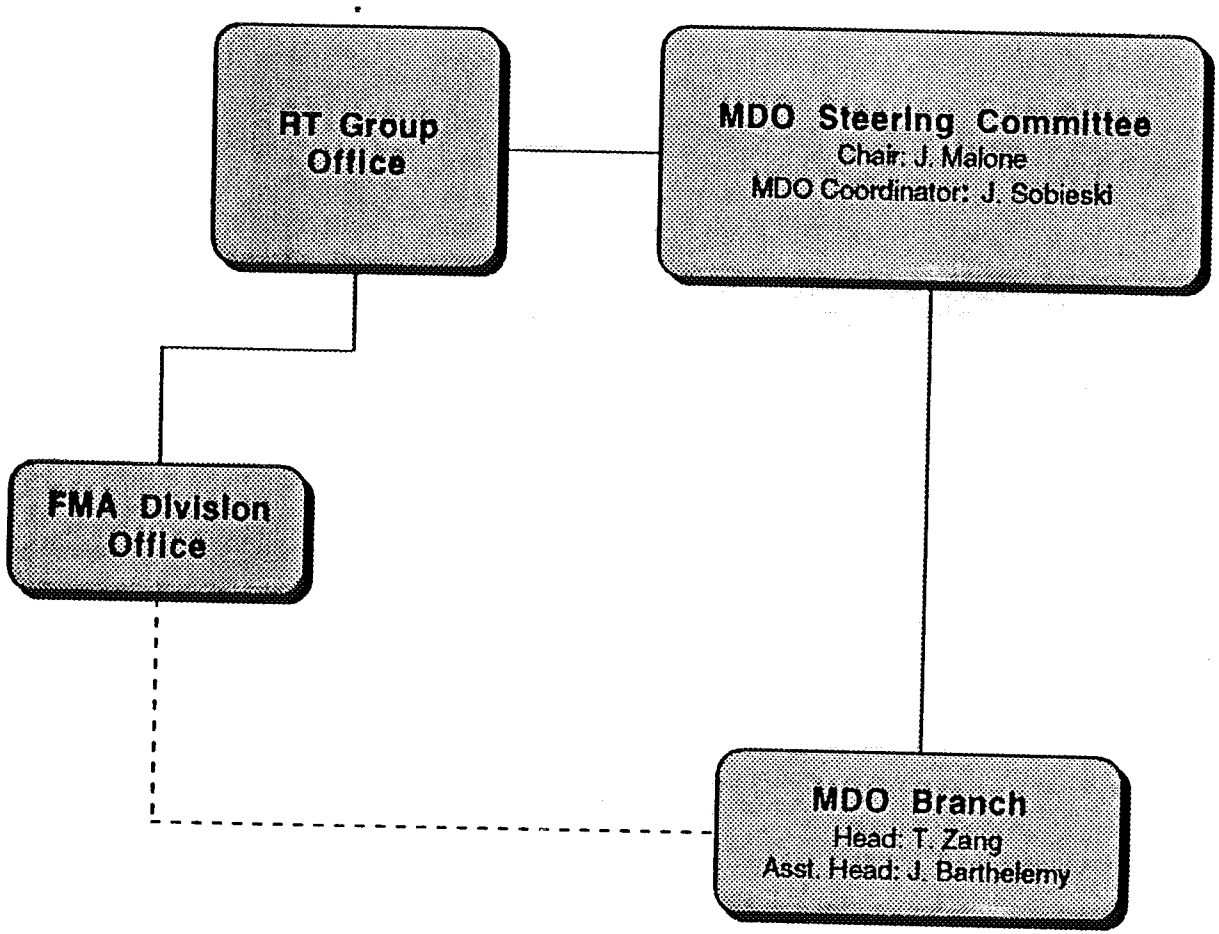
Langley Computer Users Committee

- Membership: research & support staff
- Role: a grass roots users organization that provides
  - tactical recommendations (solicited & unsolicited) to providers on computing issues
  - provides forums for information exchange between users on a variety of computing topics
- Audience:
  - providers (ISD)
  - computer users
- Will be reorganized in late Summer

Langley Computer Users Committee
Existing Structure



RTG MDO Organization



MDO Branch Functions

- Perform research into the methodology of MDO
- Participate in MDO application studies
- Foster the growth of multidisciplinary activities throughout the RTG
- Transfer new MDO methodologies to RTG, APG, SASPG, industry

NASA Products

- **Concepts**
  - NASA disseminates the *concepts* underlying new *algorithms*, *physical models* and/or *computer implementations*
  - these *concepts* are incorporated into customer's own computer codes
- **Portable Modules**
  - these modules are incorporated into customer's existing codes to add or improve capabilities
- **Pilot Codes**
  - early release of a research code that is typically special purpose, requires an expert user and is not fully debugged
  - customer uses full code or else extracts modules of interest
- **Production Codes**
  - robust, efficient, user-friendly, validated, supported
  - customer uses full code or else extracts modules of interest

A Proposal: How to Improve NASA-Developed Computer Programs

by

Jaroslaw Sobieski

Proposed for NASA Software Technology Workshop, March, 1980

(Presented to LCUC, 1980)

Problem

- A large number of computer programs are generated in NASA research work as prototype, proof-of-the-concept programs, typically as companion items to NASA formal reports
- These programs are, typically, researcher-generated and frequently embody new important methodologies and solutions
- Problem is that too often these programs are unreliable and/or inefficient, especially in hands of users other than developers

Sobieski, 1980

Evidence

- A particular group of NASA and contractor engineers (total of 14) have been working for the past 6 years at NASA LRC with an integrated system of programs applied in aircraft aerodynamics, aeroelasticity and structures problems
- The system involves now 43 programs ranging in size from 500 to 30,000 Fortran lines, a statistically significant sample
- Majority of these programs came from NASA research work, in-house and grant
- Nearly all of these programs did not work as claimed when they were first obtained, significant time and effort were needed to make them work and to make them operate with a minimum of acceptable efficiency
- In contrast, most of the contractor generated programs were free of the serious reliability and efficiency problems

Sobieski, 1980

Diagnosis of the Problem

- In a typical twin product of a research work, formal report plus computer program, report will receive all the attention of the organization, program will receive none
- The research organization is geared to support quality report production (editorial committees, illustrators, reproduction , etc.) but not quality program production
- Impression is created that in the report-program pair, the program's status is secondary. Consequence: no incentive to strive for program quality
- Vast difference in the type of skills and degree of effort between a prototype, proof-of-the-concept program, and a professionally coded and tested and documented program
- A researcher is a natural creator o: the former but is ill-equipped to develop the latter

Sobieski, 1980

Reflections from a "Jurassic Programmer" on Software Development at LaRC

In 1972 the software development environment at NASA was very different. It was a batch environment where the programmer's life revolved around the deliveries of the "green tub" and the survival of data and programs on assorted paper media. Some advanced programmers took advantage of 7-track tapes and data cells for storage.

- *The Revolution of 1975*

Two major developments occured at the Center in 1975 that changed the scope and way of doing software development forever.

The advent of micro-processors ended the monopoly held by discrete hardware components or "random logic" in the implementation of control functions. This also exposed engineers to "programmers" who were unfamiliar with hardware and its associated engineering discipline. Critical real-time applications were now in the hands of software developers and opened up the embedded domain. Good programmers saw the value in adopting practices very analogous to those of the hardware designers. As the electronics revolution continued, hardware engineers were forced to become somewhat familiar with software.

The introduction of the interactive development environment was brought about by the installation of a new operating system on the NOS mainframes and the populating of selected offices with dumb terminals. Key-to-disk storage did away with all of those card files. The terminal opened access to any programmer, regardless of background. Requirements to pass proficiency tests on the use of the system and FORTRAN in order to get a user number were deleted.

- *Q&A:* *Was batch all bad?*
  Was interactive all good?

The answers to the two questions are No and No. In retrospect, I believe the batch environment had several good attributes, and the interactive environment has been a major factor in the lack of discipline we see today.

Pre-revolutionary programmers realized the value of desk checking and flowcharting because it could take weeks to get a successful compilation if they weren't careful. Plotting in a batch mode was often an extremely frustrating task! Programmers were freed from the tedium of keypunching because folks at ACD punched and verified from green and white coding sheets. This gave me an opportunity to insert lots of good documentation and scan the code one more time before committing to the initial submittal. The slower pace of life gave programmers more time to sit and stare at their code. In fact, managers expected them to behave this way.

In the interactive world, the lure of instant gratification at the terminal led to a rush to the CRT. People routinely sat down and began typing wildly without even a coding sheet. The most unfortunate result was that people could more easily confuse activity with productivity. Often, the lowest level task - pounding the keys - was the key measure of productivity.

• *The Revolution of 1994 - better, cheaper, faster?*

Here at Langley, times have changed. In fact, times are tough. Software is now a real product, not just a by-product generated along the way to some higher goal like a report. Software is a technology that needs to be transferred outside the gate - and it needs to be good because of its added visibility. Quality issues are brought up everywhere. The dilemma is how to get quality while operating under a constrained budget.

• *No more heroes - we have to work smarter*

There is no more of the "green medicine" to throw at our software problems. There are no additional people to hire. We must realize that faster CPUs and graphics workstations and glitzy tools simply speed up the most visible portion of the development process. Automating a poor process will get us nowhere.

We need to create a recipe for successful software development for the various domains at LaRC. That is, learn from the mistakes that are often the best teachers, share the tips and tricks, and reward the people who do the right things throughout the entire lifecycle that result in quality software. We need to catch our collective breath and treat software like an engineering discipline in order to design, manage, document, maintain, and transfer knowledge.

In short, there is no license to meander anymore. The choice is ours: will we remember the past or are we, as Santayana says, "doomed to repeat it"?

Pamela L. Rinsland

LaRC Software Domains

- Flight software
  Software that performs command, control, onboard data processing, data storage and communication for space (e.g. LITE, HALOE) or aircraft (e.g. LASE, CLASS, Windshear) instruments.

- Facility software
  Software that performs the command, control, data acquisition for key LaRC facilities such as tunnels, flight simulators, hangar data systems, experimental aircraft infrastructure (737, 757), and other test facilities.

- Ground Support Equipment
  Software used to perform test and integration, check out flight instruments, and monitor system performance during mission operations.

- Management Information Systems
  Personnel and financial resource management software used to support LaRC resource management.

- Research Software
  Engineering analysis tools, simulations and models developed to support the research mission of the Center such as CFD codes, data presentation and visualization, models & algorithms, and post-mission data analyses.

Software Engineering & Ada Lab (SEAL) Tools

1) CADRE Teamwork CASE Tools:
   a) Teamwork/SA (Structured Analysis)
   b) Teamwork/RT (Real-Time Analysis)
   c) Teamwork/IM (Information Modeling)
   d) Teamwork/SD (Structure Design)
   e) Teamwork/OOD (Object-Oriented Design)
   f) Teamwork/Ada (Editor, Code Generator, Design Sensitive Editor)
   g) Teamwork/SIM (Simulation Tool)
   h) Teamwork/FORTRAN REV (rev. eng.)
   i) Ensemble "C" Tools
      - System Understanding (High-level rev. eng.)
      - Function Understanding (Low-level rev. eng.)
      - Documentation

2) Paradigm Plus (Object-Oriented Meta-CASE Tool) (4)

3) McCabe Tools:
   a) Analysis of Complexity Tool (ACT)
   b) Battlemap Analysis Tool (BAT)
   c) Ada language parser

4) Ada Measurement and Analysis Tool/Diana (AdaMAT/D)
5) VAX Software Engineering Tools (VAXset)
6) NASA Intelligent Documentation Management System (IDMS)
7) InQuisiX - Reuse Repository

8) In-Circuit Emulators
   a) Microtek MICE-V 386 Emulator
   b) Microtek MICE-V 486 Emulator
   c) HyperSource-386/486 Source/Assembly-Level Debugger
   d) AMC ES-1800 80186 Emulator (2)
   e) Emulation Support Driver (ESD) Software

9) CADRE Software Analysis Workstation (SAW) (2)
   a) Interactive State Analyzer
   b) SoftAnalyst
   c) Probes for 80186/286/386, 1750A, Generic

10) Logic Analyzers/Oscilloscopes:
   a) HP 16500A Logic Analyzer
   b) HP 16530A Digitizing Oscilloscope Module
   c) HP Probes/Preprocessor Interfaces for: 1553B, TMS320C30/31, 80486, HPIB-RS232-RS449, SCSI Bus, user definable
   d) HP Performance Analyzer
   e) HP Inverse Assemblers
   f) Fluke Scopemeters (2)

11) TITAN SESCO Flight Equivalent Computer
    a) SECS 386/30 Single Board Computer
    b) SECS 186/30 Single Board Computer
    c) SECS 80/1553B Single Board Computer
    d) Memory board (386 - 4M, 186 - 512K)
    e) Parallel and Analog I/O Modules

12) PROM Tools
    a) TITAN/Data IO Flight Board Programmer
    b) EPROM Erasers (3)
    c) PROM ICE

13) PC Data Acquisition Hardware and Software
    a) GPIB Boards and Software
    b) AT-DIO-32F (10) AND DIO-96 Boards and software
    c) SF-1 (2) Shuttle SFMDM Cards
    d) LabVIEW For Windows Dev. System (2)
    e) LabWindows
    f) NI-DAQ DOS/Windows

14) Systems
    a) VAXstation 4000 model 60
    b) SUN SPARCstation 10
    c) SUNserver 690MP
    d) Novell 486 Server/UPS
    e) Castelle FAXpress
    f) SMTP Gateway PC
    g) Various 386/486 PCs
    h) Laser Printers (3)

15) Miscellaneous
    a) Soldering/Desoldering station
    b) Wire-wrap tools
    c) Insertion/Deinsertion tools
    d) Proto-Boards/Breadboards
    e) Military & D-shell connectors and cabling tools
    f) HP Power Supplies (4)
    g) Optical Drives

For more information, contact Jerry Garcia at (804)-864-5888.

SESSION 3 Software Engineering Standards, Methods, and CASE Tools

Chaired by

Susan Voigt