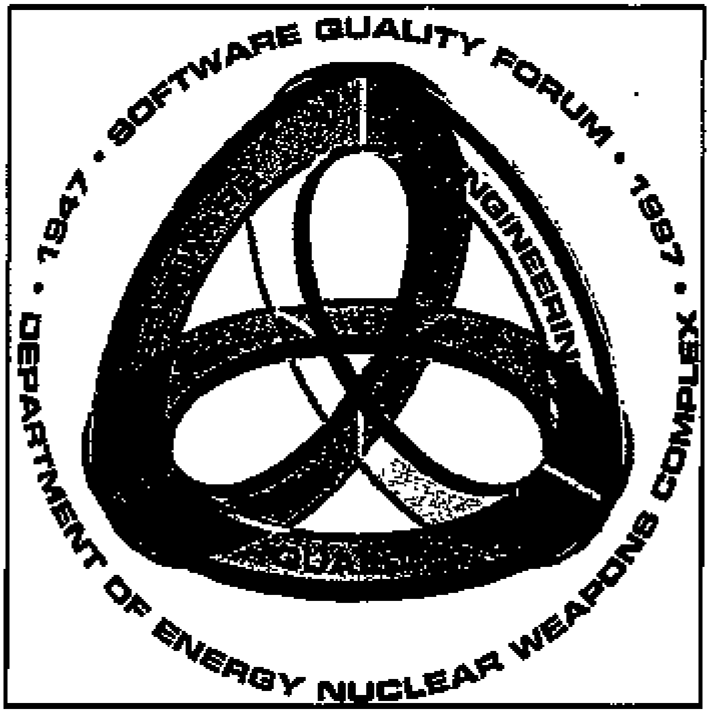
CONF-970417--

SQAS/97-1R

The Fourth Triennial
Software Quality Forum



SOFTWARE QUALITY FORUM
1947 • SOFTWARE QUALITY FORUM • 1997
DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX
ENGINEERING
QA

Software: Our Quest for Excellence

Honoring 50 years of software history, progress, and process

Co-Sponsored by:
DOE/AL/WQD NWC Quality Managers
Software Quality Assurance Subcommittee

Kirtland Air Force Base
Albuquerque, NM
April 1-3, 1997

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Table of Contents

Keynote Tutorial

Tutorials April 1, 1997

Keynote Address

General Sessions April 2, 1997

General Sessions April 3, 1997

Closing Session

SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX • 1947 •

General Information

- Summary Schedule
- Forum Committee & Program Committee
- History of the Software Quality Forum
- Software Quality Assurance Subcommittee
- Forum Awards, Proceedings, and Participating Organizations
- Tours of National Atomic Museum and RMSEL
- No Host Dinner - El Pinto Restaurant
- Bus Schedule for Social, Tours, and No-Host Dinner
- Maps: Local Area and Forum Site

SOFTWARE QUALITY FORUM
April 1-3, 1997
CONFERENCE SUMMARY

Tuesday, April 1, 1997				
08.00 - 09.00 am	REGISTRATION/CONTINENTAL BREAKFAST - TTC LOBBY			
09.00 - 11:00 am	Keynote Tutorial TTC Auditorium	Design Through Documentation: The Path to Software Quality Dr. David Parnas, McMaster University		
11:00 - 01:00 pm	LUNCH - ON YOUR OWN			
	Track Z - Keynote Tutorial SNL Bldg 823, Breezeway	Track W SNL Bldg 822, Room A	Track X SNL Bldg 822, Room B	Track Y TTC Conference Room
01:00 - 03:00 pm	Z1: Inspection of Critical Software Dr. David Parnas, McMaster University	W1: Natural Language Modeling, Dr. John Sharp, SNL	X1: Defining Software Processes Dr. Gerald McDonald, SNL Contractor	Y1: How the NWC Handles Software in Product, David Vignes, Pantex
03:00 - 03:15 pm	BREAK - TTC LOBBY			
03:15 - 05:15 pm	Z2: Structure and Documentation Dr. David Parnas, McMaster University	W2: Writing Testable SW Requirements Dr. Dwayne Knirk, SNL	X2: Using COTS Software in Development Projects Lt Col Nancy Crowley, USAF Phillips Laboratory	Y2: Software Inspection Process Overview Larry Lane and Randy Dobbs, SNL
05:30 - 06.30 pm	Social Hour and Birds of a Feather National Atomic Museum	Meet in TTC Lobby - Round Trip Transportation Provided		
Wednesday, April 2, 1997				
07.30 - 08:30 am	REGISTRATION/CONTINENTAL BREAKFAST - TTC LOBBY			
08:30 - 09.00 am	Welcoming Remarks TTC Auditorium	Mike Blackledge, Forum Chair Earl Whiteman, DOE/AL Director John Crawford, SNL Executive VP		
09.00 - 10.00 am	Keynote Address TTC Auditorium	Software Quality for 1997 - What Works and What Doesn't? Capers Jones, Chairman Software		
10.00 - 10:15 am	BREAK - TTC LOBBY			
	Track A TTC Auditorium	Track B SNL Bldg 822, Rooms A&B	Track C TTC Conference Room C	Track D
10:15 - 11:45 am	A1: Software Management Chair: Don Schilling, ASFMAT	B1: Software Testing Chair: Larry Rodin, Pantex	C1 SW Quality for Scientific Applications Chair: John Cerutti, LANL	Birds of a Feather / Networking
11:45 - 01:30 pm	LUNCH - ON YOUR OWN			
	Track A TTC Auditorium	Track B SNL Bldg 822, Rooms A&B	Track C Tours	Track D Tours
01:30 - 03:00 pm	A2. Software Engineering Processes Chair: Kathleen Carroll, DOE/HQ	B2: Internet WEB Applications Chair: Faye Brown, LMES-ORNL	Robotics Lab Register in TTC Lobby Meet in TTC Lobby before 1:30 pm	National Atomic Museum Register in TTC Lobby Meet in TTC Lobby before 1:30 pm
03,00 - 03:15 pm	BREAK - TTC LOBBY			
03:15 - 04:45 pm	A3 Software Process Improvement I Chair: Mike Lackner, ASFMAT	B3 High Integrity / Formal Methods I Chair: Dave Peercy, SNL	National Atomic Museum Register in TTC Lobby Meet in TTC Lobby before 3:15 pm	Robotics Lab Register in TTC Lobby Meet in TTC Lobby before 3:15 pm
06.00 - 07.00 pm 07.00 pm	El Pinto Restaurant	No-Host Social Hour No-Host Dinner		
Thursday, April 3, 1997				
08,00 - 08,30 am	CONTINENTAL BREAKFAST - TTC LOBBY			
	Track A TTC Auditorium	Track B SNL Bldg 822, Rooms A&B	Track C	Track D
08:30 - 10.00 am	A4. Software Process Improvement II Chair: John Hern, AWE UK	B4: High Integrity / Formal Methods II Chair: Larry Dalton, SNL	Birds of a Feather / Networking	Birds of a Feather / Networking
10.00 - 10.15 am	BREAK - TTC LOBBY			
10:15 - 11:45 am	A5: Software Quality: Experiences & Y2K Chair: Cathy Kuhn, AS/M&T	B5: SW Standards for Quality Engineering Chair: Patty Trellue, SNL	Birds of a Feather / Networking	Birds of a Feather / Networking
11:45 - 12:30 pm	WRAPUP & AWARDS - TTC AUDITORIUM			

Forum Committee

General Chair:

Mike Blackledge, Sandia National Laboratories, mablack@sandia.gov

Tutorials and Workshops:

Dave Peercy, Sandia National Laboratories, depeerc@sandia.gov

Planning Committee:

Lorraine Baca, Sandia National Laboratories
Ray Berg, Sandia National Laboratories
Dwayne Knirk, Sandia National Laboratories
Patty Trellue, Sandia National Laboratories
Gary Echert, DOE - Albuquerque Offfice

Arrangements:

Theresa Griego, Sandia National Laboratories

Program Committee

Mike Blackledge, Sandia National Laboratories
Patty Trellue, Sandia National Laboratories

Faye Brown, Martin Marietta Energy Systems, Y-12 Plant
Kathleen Canal, DOE Headquarters
John Cerutti, Los Alamos National Laboratory
Orval Hart, Los Alamos National Laboratory
Mike Lackner, AlliedSignal Federal Manufacturing and Technologies, Kansas City Plant
Dave Peercy, Sandia National Laboratories
Larry Rodin, Mason & Hanger, Pantex Plant
Don Schilling, AlliedSignal Federal Manufacturing and Technologies, Kansas City Plant
Pat Tempel, Sandia National Laboratories
David Vinson, Mason & Hanger, Pantex Plant

History of the Software Quality Forum

The Software Quality (SQ) Forum was established by the Software Quality Assurance Subcommittee as an opportunity for all those involved in implementing SQA programs to meet and share ideas and concerns. The SQ Forum is open to the public. Participation from managers, quality engineers, and software professionals provides an ideal environment for identifying and discussing the many issues and concerns raised by the Forum attendees and speakers. The interaction provided by the Forum contributes to the realization of a shared goal -- high quality software product.

Topics presented at the SQ Forum generally include: testing, software measurement, software surety, software reliability, SQA practices, assessments, software process improvement, certification and licensing of software professionals, CASE tools, software project management, inspections, and management's role in ensuring SQA.

The Software Quality Forum is held every three years; past Forums are identified below.

Date	Site
Spring 1988	Sandia National Laboratories
Spring 1991	AlliedSignal Aerospace Kansas City Division
Spring 1994	Lawrence Livermore National Laboratory

Software Quality Assurance Subcommittee

The Software Quality Assurance Subcommittee (SQAS) serves as a Technical Advisory Group on software engineering and quality initiatives and issues for the Department of Energy's Quality Managers. The Quality Manager at each DOE site has the opportunity to select one Primary and one Alternate representative to the SQAS.

The Subcommittee grew out of a Software Quality Assurance Information Exchange Forum which was held in March of 1988 at Sandia National Laboratories. The Subcommittee provides a continuing forum for the exchange of information and work issues in the area of software quality engineering.

For additional information about the SQAS, visit our web site at:

http://www.pantex.com/sqas/sqas.htm

Forum Awards

The Forum Program Committee would like to recognize those presenters who, through their tutorial or technical presentation, have made a significant contribution to the success of the Forum. A Best Tutorial and Best Presentation award will be presented at the Forum Wrap-up session on Thursday, April 3. Selection of recipients for the Awards will be determined in two parts:

- technical content, scored by the Forum Committee
- delivery and usefulness, scored by attendees

Forum Proceedings

Forum Proceedings will include abstracts and presentation materials for all technical presentations, presenter biographies, tutorial materials, and final Forum program information. Forum Proceedings will be distributed at the Forum with the registration packets. Additional Forum Proceedings can be purchased at the Registration Desk in the TTC Lobby.

Participating Organizations

AlliedSignal, Federal Manufacturing and Technologies, Kansas City Plant (AS/FM&T)
Atomic Weapons Establishment, United Kingdom (AWE UK)
Department of Energy, Albuquerque Office (DOE/AL)
Department of Energy, Headquarters (DOE/HQ)
Lawrence Livermore National Laboratory (LLNL)
Lockheed Martin Energy Systems, Oak Ridge, Y-12 Plant (LMES/OR)
Los Alamos National Laboratory (LANL)
Mason & Hanger, Pantex Plant (Pantex)
McMaster University, Communications Research Laboratory, Canada (MU/CRL)
New Mexico State University (NMSU)
Sandia National Laboratories (SNL)
Software Productivity Research (SPR)
United States Air Force, Phillips Laboratory (USAF/Phillips)
Westinghouse, Savannah River Site (SRS)
Pioneer Technologies (Pioneer)

National Atomic Museum Tour

Operated by the Department of Energy, The National Atomic Museum contains a large collection of declassified nuclear technology. Since its opening in 1969, the objective of the National Atomic Museum has been to provide a readily assessable repository of educational materials, and information on the Atomic Age.

Prominently featured in the museum's high bay is the story of the Manhattan Engineer District, the unprecedented 2.2 billion dollar scientific-engineering project that was centered in New Mexico during World War II.

A portion of the Museum is devoted to exhibits on the research, development, and use of various forms of nuclear energy. Historical and other traveling exhibits are also displayed in this area. Located outside of the museum are a number of large exhibits. These include the Boeing B52B jet bomber and a Navy TA-7C Corsair II fighter-bomber as well as many other nuclear weapons systems, rockets, and missiles.

Robotic Manufacturing Science & Engineering Laboratory Tour

Intelligent systems bring diverse technologies together: computers, software, sensors, vision systems, and hardware such as robots. At Sandia National Laboratories, combinations of these technologies are merged to create robotic and intelligent systems that range from micro to mega.

To advance the evolution of robotic and intelligent system technologies, Sandia National Laboratories and the DOE created the Robotic Manufacturing Science and Engineering Laboratory (RMSEL). It is the first centralized facility designed specifically for bringing intelligent machine technologies and technologists together.

The RMSEL facility was designed as a special environment to accommodate the unique needs of robotics and intelligent systems research. A second-floor viewing gallery concourse overlooks ground-floor laboratories used for the development of large-scale robotics systems. The State-of-the-art physical resources coupled with outstanding intellectual resources make RMSEL unique in robotic and intelligent systems research and development.

One of the main purposes of RMSEL is encouraging collaborative development with industry and academic partners.

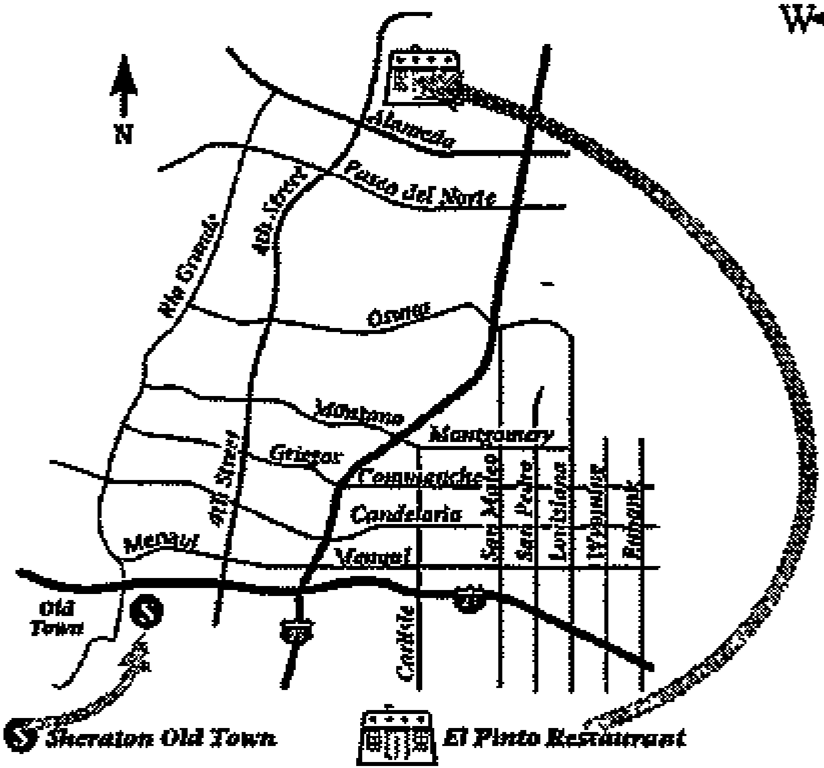
No Host Dinner - El Pinto Restaurant

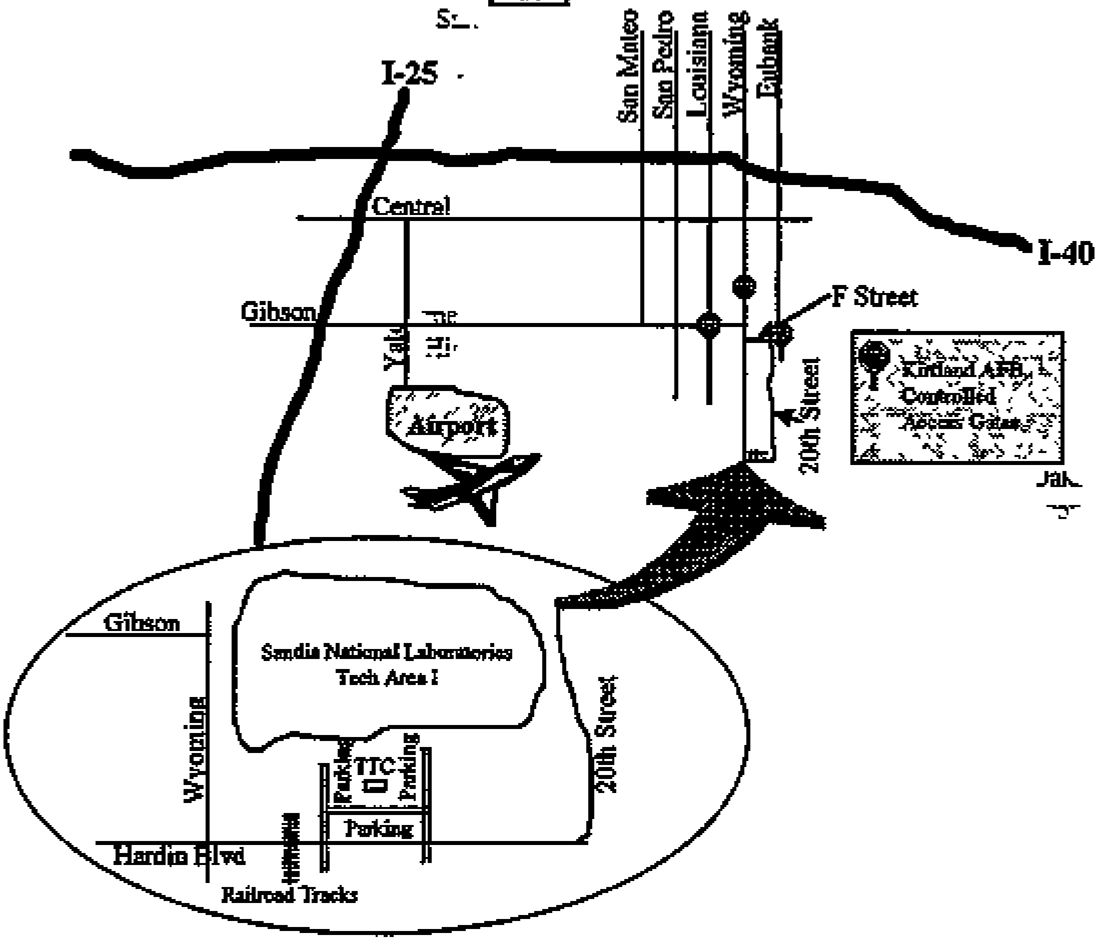
A No-Host dinner has been planned for Wednesday Evening at the El Pinto Authentic New Mexican Restaurant located at 10500 4th NW. There will be a variety of dinner selections offered that should accommodate all tastes. The cost of the dinner is $15. Check at the Registration Desk in the TTC Lobby if you would like to attend or if you are planning to use the bus transportation provided from the Sheraton Hotel to the El Pinto Restaurant. El Pinto is located at 10500 4th NW; the phone number is 898-1771.

Bus Schedule for Social, Tours, No-Host dinner

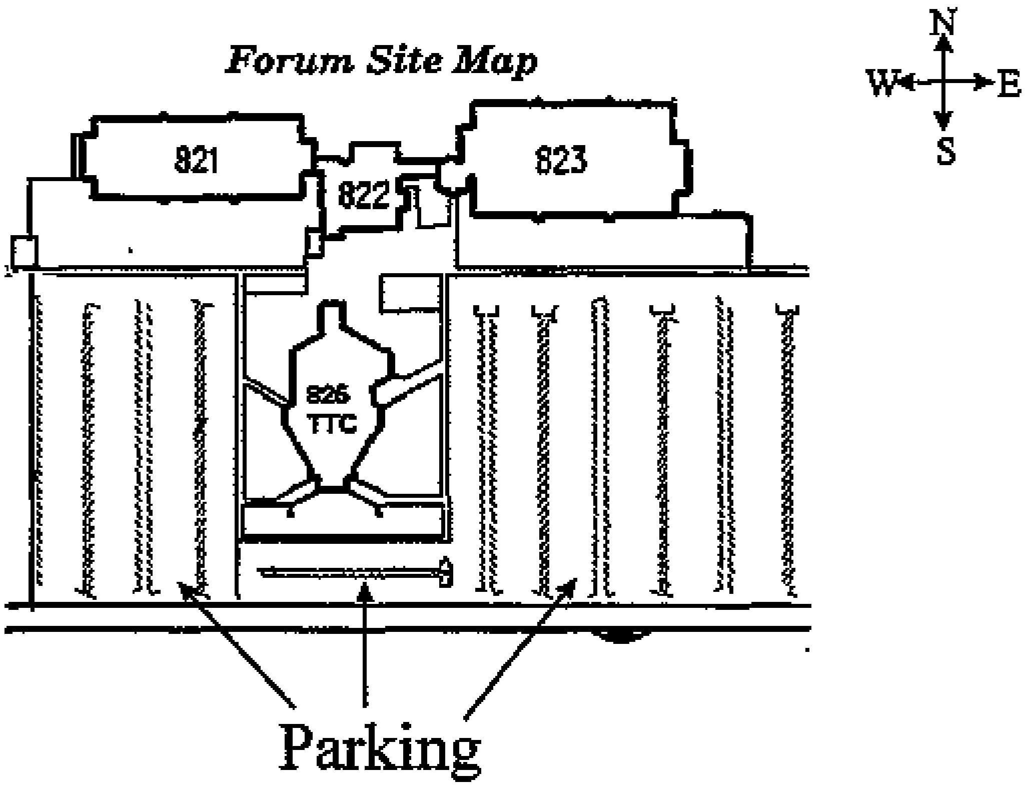
<table>
<tr><th colspan="4">Tuesday, April 1, 1997</th></tr>
<tr><th>Depart</th><th>Time</th><th>Destination</th><th>Return to Sandia National Labs, "Pick-Up" time</th></tr>
<tr><td>Sandia National Labs, TTC</td><td>5:30 p.m.</td><td>National Atomic Museum (Social)</td><td>6:30 p.m.</td></tr>
<tr><th colspan="4">Wednesday, April 2, 1997</th></tr>
<tr><td>Sandia National Labs, TTC</td><td>1:30 p.m.</td><td>Robotics Lab</td><td>2:45 p.m.</td></tr>
<tr><td>Sandia National Labs, TTC</td><td>1:30 p.m.</td><td>National Atomic Museum</td><td>2:45 p.m.</td></tr>
<tr><td>Sandia National Labs, TTC</td><td>3:15 p.m.</td><td>Robotics Lab</td><td>4:45 p.m.</td></tr>
<tr><td>Sandia National Labs, TTC</td><td>3:15 p.m.</td><td>National Atomic Museum</td><td>4:45 p.m.</td></tr>
<tr><td>Sheraton Old Town</td><td>5:45 p.m.</td><td>El Pinto Restaurant</td><td><b>Return to Sheraton Old Town, "Pick-Up" time</b> ~8:30 p.m.</td></tr>
</table>

LOCAL AREA MAPS





Forum Site Map



Location of Conference Rooms

TTC Auditorium, TTC Lobby, TTC Conference Room
Located in Building 825. Enter through the doors on the north side of the building.

Bldg. 822 Rooms A&B
Located immediately to the right when entering Bldg. 822 from the doors on the south side of the building.

Bldg. 823 Breezeway
Located immediately to the left after the Reception Desk when entering Bldg. 823 from the doors on the south side of the building.

NOTE: To get into the 823 Breezeway, individuals without a valid DOE must be escorted by an individual with a valid DOE badge. They must show a picture ID and sign in at the reception desk. The Breezeway will only be used for the afternoon Keynote tutorials and a Forum committee member will be available to assist you with the entrance details.

Biographies & Abstracts

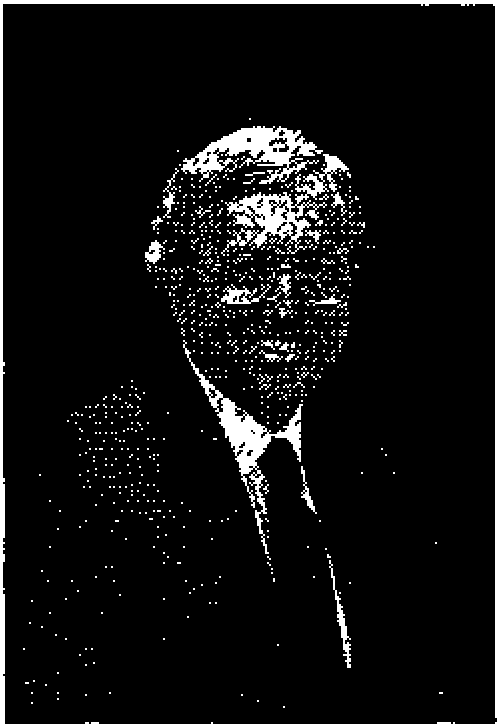
BIOGRAPHIES

1987 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Keynote Biographies

Capers Jones, Chair SPR



Capers Jones is an international consultant on software management topics and Chairman of Software Productivity Research, Inc. (SPR) in Burlington, MA. Following graduation from the University of Florida, Mr. Jones began his software career as a programmer in the office of the Surgeon General, Washington, D.C.. Prior to becoming Chairman at SPR, Mr. Jones also worked at the Crane Company, IBM, and was Assistant Director of Programming Technology at ITT in Stratford CT. Mr. Jones has published nine books dealing with software areas including; programming productivity, software measurement, and software quality. His tenth book, Software Cost Estimating is scheduled for publication in early 1997. Mr. Jones will share his experience and insights in his keynote address "Software Quality for 1997 - What Works and What Doesn't".

Presentation: April 2, (09:00-10:00 am), TTC Auditorium

Dr. David Lorge Parnas, McMaster University

Professor David Lorge Parnas, Ph.D. holds the NSERC/Bell Industrial Research Chair in the Communications Research Laboratory, Department of Electrical and Computer Engineering at McMaster University in Hamilton, Ontario, Canada. His primary area of interest is to promote to Software Engineers the discipline and body of knowledge as practiced by engineers in other fields.

By studying the problems of software engineering since 1965, Dr. Parnas has developed principles and methods that have value to real world problems. In recognition of his accomplishments, he has received numerous honors, including election as a Fellow of the Royal Society of Canada and a Fellow of the Association for Computing Machinery. Dr. Parnas will share his experience and knowledge by leading three workshop/tutorials.

Tutorials: April 1, Z1 (09:00-11:00 am), TTC Auditorium
Z2 (01:00-03:00 pm), Z3 (03:15-05:15 pm), Bldg 823 Breezeway

Tutorial Leader Biographies

(Alphabetical Order)

Nancy L. Crowley, Phillips Laboratory

Lt Col Nancy Crowley is the Acting Chief of the Space System Technologies Division (PL/VTS), Kirtland AFB, New Mexico. The focus of Space System Technologies Division is on the innovative application of software technologies to improve performance and reduce operations and maintenance costs for satellite control systems, including telemetry, tracking and commanding (TT&C), mission data dissemination, data processing, and satellite autonomy. Lt Col Crowley is also the program manager for the Multimission Advanced Ground Intelligent Control (MAGIC) program. MAGIC is developing the architecture for the next generation satellite control system that provides a low cost, flexible software architecture that allows plug and play of COTS products in a vendor independent manner. Lt Col Crowley was born May 13, 1955 in the Bronx, New York. She graduated from Theills High School in Theills NY, in 1973. She received a Bachelor of Science in Electrical Engineering from the University of New Hampshire in 1977 where she was a ROTC distinguished graduate. She later received the Master of Science in Digital Engineering and the Doctor of Philosophy (major of software engineering, minor of artificial intelligence) from the Air Force Institute of Technology in 1982 and 1994 respectively. Her research was in object-oriented methods for software requirements analysis. Lt Col Crowley entered the Air Force in 1972 and was a flight test engineer for Tactical Air Command. There she conducted operational test and evaluation and flew in fighter aircraft in support of projects. After her masters degree, she was assigned to the Flight Dynamics Laboratory, where she was the software engineer for the digital flight control system of the X-29 Advanced Technology Demonstrator and the Ada focal point for the laboratory. There and in subsequent assignments she was a technical consultant to the Swedish government on the development of the digital flight control system for the JAS-39. Her next assignment was at the Systems Acquisition School, Brooks AFB Texas where she was a course developer and instructor of software acquisition courses. There she was also a system administrator for a UNIX and PC-based networked system that serviced the students and staff at the school. After completing her Ph.D., she came to her current assignment in Oct 94. Outside her Air Force duties, Lt Col Crowley teaches software engineering, software management, and computer science courses at local Universities. Her and her husband own a computer consulting business. Both her and her husband enjoy riding horses.

Tutorial X2: April 1, (03:15 - 05:15 pm), SNL Bldg 822, Room B

Randy Dabbs, Sandia National Laboratories

Randy Dabbs is a Senior Member of Technical Staff at Sandia National Laboratories. He has earned a Master of Science in Electrical Engineering from the University of New Mexico. He has held positions at the Sandia Particle Beam Fusion Accelerator in the areas of data acquisition and signal processing; the Kwajalein Missile Range in the areas of range computer systems engineering, range operations, tracking software modeling and development, reentry mission project engineering, digital radar signal processing, radar controller real time software, and software configuration management; and the Sandia Kauai Test Facility in the areas of range computer support and operations, range safety software development, countdown software development, CASE tool selection and modeling of range operational software. In his current position with the Sandia Quality Engineering Department, he has participated in instructing the Software Quality Engineering course and the Software Inspections course. In his role as software quality assurance engineer, he has participated in numerous software inspections for both internal and external customers. In addition, he has helped develop and teach a customized version of the software inspection course to meet the specific needs of Sandia organizations.

Tutorial Y2: April 1, (03:15 - 05:15 pm), TTC Conference Room C

Dwayne L. Knirk, Ph.D., Sandia National Laboratories

Dr. Knirk is a member of the software quality engineering department at Sandia National Laboratories. He provides in-house consulting to line organization projects for software engineering processes, methods, standards, tools, and training. He participates in process assessments and improvement programs, and provides support for configuration management, software inspections, and process automation. Dr. Knirk's primary focus is on the two complementary areas of software specification and testing, in which he works to bring more formal methods into more practical applications. He works actively on IEEE software engineering standards groups. He is a member of the ASQC Software Division Methods Committee. Dr. Knirk previously worked for Programming Environments, Inc., where he was the architect and principal developer of the automated software test design tool, T. That commercial product analyzed a formal software behavior description for testability, designed test cases for demonstrating that behavior, and generated actual test case data.

Tutorial W2: April 1, (03:15 - 05:15 pm), SNL Bldg 822, Room A

Tutorial Leader Biographies

(Alphabetical Order)

G. Lawrence Lane, Sandia National Laboratories

Larry Lane is a Senior Member of the Technical Staff at Sandia National Laboratories. He earned a Master of Arts Degree in mathematics from the University of Kansas. Larry joined Sandia Corporation in 1959 as an assembly language programmer in the field data reduction department. He has also worked as a operating systems programmer and was responsible for the selection and installation of Sandia's first general purpose time sharing computer. Larry also worked as a computer consultant for large scientific computers, as the second computer ombudsman, and was responsible for the development of an electronic tracking system for electrical testing of radiation-hardened microcircuits. Larry moved to his current position in the Quality Engineering Department in 1991, where he is an instructor for the Software Quality Engineering course and the Software Inspection Class. As a software quality engineer, Larry has led numerous qualification efforts for new and upgraded software projects, particularly in the areas of use control and weapon security. He has helped develop and teach a customized version of the software inspection course to meet specific Sandia organizational needs.

Tutorial Y2: April 1, (03:15 - 05:15 pm), TTC Conference Room C

Gerald W. McDonald, Ph.D.

Dr. McDonald has a Bachelor of Science in Engineering Science and a Master of Science in Computer Systems Management from the Naval Postgraduate School. Following his retirement the Navy he received a Master of Engineering in Industrial and Systems Engineering and a Ph.D. in Quantitative Management Science (Operations Research) from the University of Florida. Following receipt of his Ph.D. he worked for BDM International as an executive-level Program and/or Project Manager and technical leader. During his thirteen years with that firm he led both software and non-software projects. During the three years since his retirement from BDM he has acted as consultant to Sandia, SEMATECH, and a number of other organizations. As a consultant he has worked primarily in the field of Software Process Improvement. Besides direct technical assistance he has presented training and workshops in software areas such as: quality engineering, software inspections, process definition and documentation, and metrics.

Tutorial X1: April 1, (01:00 - 03:00 pm), SNL Bldg 822, Room B

John K. Sharp, Ph.D., Sandia National Laboratories

John has performed information analysis in various positions at Sandia for fifteen years. He has worked closely with Prof. Shir Nijssen of the Netherlands for several years to establish the procedure to develop and analyze information problems using structured natural language. They are currently finishing a text on this topic. This procedure was originally based on the NIAM (Natural language Information Analysis Methodology) modeling technique. John and Prof. Nijssen have co-chaired two international conferences on natural language modeling. John is also the editor of the international standard on conceptual schemas.

Tutorial W1: April 1, (01:00 - 03:00 pm), SNL Bldg 822, Room A

Software Quality Assurance Subcommittee, Work Item #16, Nuclear Weapons Complex Sites

The Software Quality Assurance Subcommittee (SQAS) operates under the DOE Nuclear Weapons Complex (NWC) Quality Managers to identify and resolve Software Quality issues and problems common to all DOE sites and facilities. This tutorial is the result of an NWC SQAS work item to define how to manage and control software as product. The work item was established to satisfy a need to define a consistent process for handling product software. The Nuclear Weapons Complex-wide participants and presenters of this tutorial include:

Chair David Vinson, Pantex Plant
Phil Huffman, Pantex Plant
Alvin Cowen, Pantex Plant
Catherine Kuhn, AS/FM&T
Donald Schilling, AS/FM&T
Dave Peercy, SNL
Mike Blackledge, SNL
Orval Hart, LANL
John Cerutti, LANL
Bill Warren, LLNL
Charles Chow, LLNL
Ellis Sykes, DOE/Kansas City Area Office
Gary Echert, DOE/Albuquerque Area Office
Kathleen Canal, DOE/HQ
Ray Cullen, SRS
Faye Brown, LMES, Oak Ridge, Y-12 Plant

Tutorial Y1: April 1, (01:00 - 03:00 pm), TTC Conference Room C

Presenter Biographies

(Alphabetical Order)

John Ambrosiano, Ph.D, Los Alamos National Laboratory

Dr. Ambrosiano received his Ph.D. in Plasma Physics from the College of William and Mary in 1980 and has since pursued a career in Computational Physics. He has written simulation codes for a variety of applications including plasmas and beams, acoustics, fluid dynamics, and electromagnetics. After a postdoctoral appointment at the University of Alaska's Geophysical Institute to study Space Physics, he moved to the Washington, DC area to work with a defense contractor. In 1987 he joined the Lawrence Livermore National Laboratory where he worked on nuclear weapons applications, and later joined the Earth System Modeling project there. The growing complexity of numerical simulations led to a strong interest in Computer Science and in Software Engineering in order to find the leverage to manage the complexity of the new generation of simulation codes. In 1995 he joined the North Carolina Supercomputing Center to lead the effort to build a simulation framework for environmental modeling called the Environmental Decision Support System. This became the prototype for EPA's Models-3 framework. He recently joined Los Alamos National Laboratory to participate in DOE's Accelerated Strategic Computing Initiative. He is currently the leader of a twelve-person visualization and human-computer interaction team in X Division at LANL. He is also the Laboratory's principle investigator for Scientific Data Management within the ASCI program. His current interests are scientific data management, computational frameworks, and software engineering for scientific applications.

Presentation: Wednesday, April 2, Session C1: 10:15-11:45 am, TTC Conference Room C

Rodema Ashby, Sandia National Laboratories

Rodema Ashby has been programming or leading projects at Sandia for the last 13 years. Projects have included configurable software security systems such as the Site Independent Alarm and Display System, and a Logging and Accountability Subsystem. Interactive Collaborative Environments (ICE) which was licensed to SUN Microsystems as their "Show Me" product included a great deal of commercial customer testing and collaboration. A-PRIMED which was a 22 month, 2.5 million dollar cooperative effort involving 10 SNL NM Centers (and minimally KC and SNL CA), demonstrated a 24 day, new product to market cycle. New hardware from new customer requirements was created in a matter of days, after the project realization team had set up a communications network and created and integrated tools for product realization. Rodema is currently writing code to customize solid modeling tools for easier user model modifications.

Presentation: Wednesday, April 2, Session A1: 10:15-11:45 am, TTC Auditorium

Mikhail Auguston, New Mexico State University

Received a Ph.D. degree from the Institute of Cybernetics in Kiev (USSR) in 1983, Diploma of the Senior Research Fellow from the Highest Evaluation Commission of the Council of Ministers of USSR in 1990, and degree of Doctor in Computer Science from University of Latvia in 1992. Research interests are in programming language design and implementation, and program testing and debugging tool design.

Joined Computing Center of Latvia University as Research Scientist in 1971. Since 1983 worked as a Leading Researcher at the Institute of Mathematics and Computer Science of Latvia University. Took part in the design and implementation of the language for file processing, the interpreter for PL/1 program testing, the testbed environment for assembler level language for PDP-11 computers, the implementation of specification language SDL for communication system software rapid prototyping and testing, the tool system GRAPES/4GL for information system design. In the years 1987-88 has designed and implemented programming language RIGAL for compiler writing on PDP, VAX and IBM PC computers. This work was presented at a number of international conferences and is used at several sites for language processor design. In 1990 he has started to work on program formal annotation language FORMAN for sequential and parallel program dynamic analysis, testing and debugging. This work was presented at various international conferences and in several universities in Europe and United States as an invited talk. He is the author of more than 30 scientific articles and co-author of the most popular textbook on PL/1 in Soviet Union (totally more than 100,000 copies printed). Currently he is an Associate Professor at the Computer Science Department of New Mexico State University. He teaches undergraduate and graduate classes on C++, Data Structures, Software Engineering, Compiler Construction, Ada programming language. Member of ACM and IEEE Computer Society.

Presentation: Thursday, April 3, Session B5: 10:15-11:45 am, Bldg 822, Rooms A&B

Presenter Biographies

(Alphabetical Order)

Michael Bell, Lockheed Martin Energy Systems

Michael Bell is a software engineer with Lockheed Martin Energy Systems at the Y-12 Plant. He is the lead analyst on the Electronic Medical Records System project, as well as member of the software metrics team. He has worked in the Oak Ridge area for seventeen years, at both Y-12 and Oak Ridge National Laboratory. His experience includes research- and production-oriented software, in areas such as plasma physics, econometrics, access control, manufacturing, and inspection. In this capacity, he has performed user interface and database design, application migration (cross-platform and mainframe-to-workstation), real-time device control, modeling, statistical and graphical analysis, and all aspects of structured and object-oriented software development. Mike holds a bachelor's degree in mathematics and is currently working toward a master's degree in software engineering.

Presentation: Wednesday, April 2, Session A2: 01:30-03:00 pm, TTC Auditorium

Gail M. Benefield, Lockheed Martin Energy Systems

Ms. Benefield has worked for Lockheed Martin Energy Systems, Inc. (LMES) since 1987. Her assignments include working as an applications developer/analyst at the Y-12 site, an Applications Security Specialist for the Computing and Telecommunications Security Organization, and currently, as a Computing Specialist within the Information Technology Services division at the K-25 site in Oak Ridge. At Y-12, Ms. Benefield was on the team which revised the 80-Series, a document owned by the Y-12 Quality Division, which was the Y-12 implementation of the required software development methodology. She was also a member of the Y-12 Software Configuration Control Board, which reviews all software changes to applications which fall within a certain class of software. In her current assignment, Ms. Benefield is representing her department as an active participant on the team which authored and is supporting the Software WorkPackage Methods (SWM) methodology.

Presentation: Thursday, April 3, Session A4: 08:30-10:00 am, TTC Auditorium

Larry J. Dalton, Sandia National Laboratories

Larry J. Dalton holds a BS in Applied Mathematics and an MS in Electrical Engineering both from the University of New Mexico. Larry has spent the past 19 years at Sandia National Laboratories in Albuquerque, New Mexico engaged in high consequence systems development. Much of that time was dedicated to various aspects of nuclear weapons and associated control systems. He is the manager of the Command and Control Software Department at Sandia National Laboratories which in addition to software engineering research, develops software and systems safety solutions for high consequence operations.

Presentation: Wednesday, April 2, Session B3: 03:15-04:45 pm, Bldg 822, Rooms A&B

Larry Desonier, Sandia National Laboratories

Education: In 1972, Larry graduated from Southwestern Louisiana with a Bachelors of Science in Electrical Engineering. In 1976 graduated from Oklahoma City University with a Masters in Business Administration. In 1979 completed Masters in Electrical Engineering and Computer Science from University of New Mexico. Complete a Masters of Science in Computer Information Systems from the University of Phoenix in 1996. Presently working on a Certificate in Computational Simulation Science from the University of New Mexico under a special Sandia National Laboratories retraining program with completion in May 1998. Work Experience: Officer in the U.S. Air Force from 1972 through 1975 and worked as a Communications-Electronics Engineer. Worked at the U.S. Air Force Weapons Laboratory from 1976 to 1984 as the Director of Communications. Came to Sandia National Laboratories in 1985 and has worked as a Systems Developer, Software Engineer, and Project Leader for over 12 years.

Presentation: Thursday, April 3, Session A5: 10:15-11:45 am, TTC Auditorium

John Hare, Ph.D., AWE, Ministry of Defence, United Kingdom

Dr John T Hare is the Software Quality Manager of AWE Aldermaston, an MOD (UK) facility managed by Hunting-BRAE Ltd. He is a Chartered Engineer and a Member of both the British Computer Society and the Institute of Quality Assurance. John graduated from the Universities of Nottingham (BSc) and York (DPhil). He started his career in 1973 as a scientist at what was then the Royal Aircraft Establishment (of International Airshow fame). He was responsible for analysis of sonobuoy trials data, using computers in the days when 16KByte was a generous amount of core memory! In 1980 John joined AEA Technology, which as UKAEA had been

responsible for the UK Atomic Energy Programme. John was responsible for the design of a number of computer-based data acquisition systems. As the PC took the skill out of this activity, John's team specialised in Management Information Systems, and the provision of Software Engineering support to scientific projects. This was the start of a growing interest in Quality Assurance, as customers and regulatory authorities demanded accreditation to ISO9001. In 1993 John joined AWE, with a brief to improve software quality assurance and raise standards across the company. This is moving into a new phase, with emphasis on Software Engineering. John and his wife Heather have two daughters; Katherine (22) who is a biochemist doing research at Birmingham University, and Louisa (19) who is a student of Modern Languages at Nottingham University. Outside interests include local government and local history. Until recently John was Chairman of Governors at a school with 1000 students.
Presentation: Thursday, April 3, Session B5: 10:15-11:45 am, Bldg 822, Rooms A&B

David L. Harris, Sandia National Laboratories

Dave has a M.S. in Computer Science and A.B in Mathematics from all from the University of Missouri. He was a graduate fellow at the Health Services Research Center in Columbia Missouri and his graduate education focused on multi-processor hardware architectures and multi-processing operating systems. Dave is currently a Senior Member of the Technical Staff at Sandia National Laboratories and is assigned to the Information Systems Engineering Center. Dave has been doing research in using World Wide Web technology in support of collaborative environments for distributed Decision Support Systems. Dave was the software process engineer for the ICADS (Integration Correlation and Display System) program. ICADS is a ground based satellite data analysis system and the project leader for TCAMS (Tech Control Automation, Maintenance, and Support), a five year, $6 - 8M project consisting of over one million lines of software source code. TCAMS has been accepted by the Department of Defense customer and is in operation today. (A fielded and functional system). As the TCAMS Team Leader, Dave was responsible for the device control software subsystem of the TCAMS software project. Earlier in Dave's career he was a software engineer responsible for various systems analysis and design of a large command and control software system. Dave has software engineering experience in real-time, embedded, guidance and control computers for ballistic missiles and systems administration of large, multi-user, time-sharing systems.
Presentation: Wednesday, April 2, Session A1: 10:15-11:45 am, TTC Auditorium

Orval Hart, Los Alamos National Laboratory

Orval Hart has worked at the Los Alamos National Laboratory for 20 years, mainly involved in real-time control systems for nuclear facilities. He has a Bachelor's Degree in Mathematics from California State Polytechnic College (Cal Poly) at Pomona and a Master's Degree in Computer Engineering from the University of New Mexico. Prior to coming to Los Alamos, he worked in real-time data acquisition systems, later moving to the Jet Propulsion Laboratory in Pasadena where he worked on real-time telemetry and communication systems. In 1975, he moved to Los Alamos where he was responsible for the original building control system software for the Plutonium Research and Development facility (known as TA-55). Since then, he has worked on a control system for an unmanned nuclear power supply (later canceled), the original procurement of the Laboratory intrusion and detection system, an environmental monitoring computer network system for the Nevada Test Site and surrounding states, the facility control system for the Special Nuclear Materials Laboratory (a sister facility to TA-55 that was later canceled also), and for the last ten years has been responsible for the control software for the Weapons Engineering Tritium Facility. This system is not only a facility environment control system, but also assists in performing the everyday work in the Facility. Almost all work in the Facility is done from the control console as opposed to hands-on in glove boxes. As many of the procedural interlocks as could be foreseen were implemented in software to avoid human error, taking special care to test and prove them prior to going 'on-line'. Computer controlled automatic sub-systems are monitoring the Facility constantly to mitigate any operational abnormalities. This system was implemented during the early days of Admiral Watkin's tenure and as such, was a test case for increased compliance and formality-of-operations.
Presentation: Wednesday, April 2, Session C1: 10:15-11:45 am, TTC Auditorium

Presenter Biographies

(Alphabetical Order)

Kevin Hill, Mason and Hanger Corporation, Pantex Plant

Kevin Hill is a tester design engineer at the Mason and Hanger Corporation. He holds a BS in electrical engineering from Kansas State University and is currently enrolled in the Interdisciplinary Master of Engineering curriculum at Texas Tech University. Co-author Dr. Mario G. Beruvides is an assistant professor in Industrial Engineering at Texas Tech University. Dr. Beruvides has 10 years of industrial work experience in design, production, and manufacturing. His interests include white-collar/knowledge work performance improvement, productivity engineering, work measurement, technology management, and engineering education. Dr. Beruvides is a member of ASEM, a senior member of IIE, and a member of ASQC and the Academy of Management. He holds a BS in mechanical engineering and an MSIE degree from the University of Miami, and a Ph. D. from Virginia Polytechnic Institute and State University in industrial and systems engineering.

Presentation: Wednesday, April 2, Session B2: 01:30-03:00 pm, Bldg 822, Rooms A&B

Curtis G. Holmes, Jr., Lockheed Martin Energy Systems

Curt came to Lockheed Martin Energy Systems (LMES) at Oak Ridge, Tennessee from Texas Instruments and is currently the Department Manager of the Environmental, Waste Management, and Analytical Laboratories Systems in the Data Research and Development Organization. The purpose of the department is to be a focal point for providing computing support for the Environmental, Waste, and Analytical Laboratory business areas at LMES. Prior to his current assignment, Curt was the Department Manager for the Computer Application's Department in the Engineering Division. The main focus of this department is the design, development, implementation, and deployment of digital systems to support real time process control and data acquisition systems. Curt Holmes holds a B.S. and M.S. Degree in Electrical Engineering from the University of Tennessee with a Minor in Computer Science. He is a licensed Professional Engineer in the State of Tennessee.

Presentation: Thursday, April 3, Session A5: 10:15-11:45 am, TTC Auditorium

Karen Jefferson, Sandia National Laboratories in California

Karen L. Jefferson has worked at the Sandia National Laboratories for 12 years and is currently in the Systems Research Department at Sandia California. Her work experience at Sandia has included high performance computing, realtime control, software engineering, and systems analysis. She is currently the software project lead on the Advanced Atmospheric Research Equipment project. She has a Masters degree in Computer Science from the University of Arizona.

Presentation: Wednesday, April 2, Session A2: 01:30-03:00 pm, TTC Auditorium

Bruce L. Johnston, Mason & Hanger Corporation, Pantex Plant

Bruce L. Johnston is a Project Programmer/Analyst for Mason & Hanger Corporation at the DOE Pantex Plant. In April 1996, he accepted the challenge to be the Project Manager for the year 2000 Project. Before accepting this new assignment he was the Computer Security Site Manager for the Pantex Plant and has worked in a computer security capacity for the last ten years. Prior to joining Mason & Hanger, he worked for Battelle Memorial Institute in Richland, Washington, and with EG&G in Idaho Falls, Idaho. In his personal life he has served as a Scoutmaster for his community and is currently serving as a Bishop for the Church of Jesus Christ of Latter-Day Saints. He keeps a healthy perspective and stays in balance by being a father of four children.

Presentation: Thursday, April 3, Session A5: 10:15-11:45 am, TTC Auditorium

Marie-Elena C. Kidd, Sandia National Laboratories

Marie-Elena C. Kidd is a computer scientist and Senior Member of the Technical Staff at Sandia National Laboratories. During her ten years at Sandia, she has worked as a software engineer on embedded, real-time software systems for such applications as robotics, nuclear weapon components, and control systems. She has also worked on lab-wide information sharing software systems and software engineering initiatives. She has a B.S. in Computing and Information Sciences, Trinity University, San Antonio, TX and an M.S. in Computer Science, Purdue University, West Lafayette, IN.

Presentation: Thursday, April 3, Session B4: 08:30-10:00 am, Bldg 822, Rooms A&B

Presenter Biographies

(Alphabetical Order)

Dr. Dwayne L. Knirk, Ph.D., Sandia National Laboratories

Dr. Knirk is a member of the software quality engineering department at Sandia National Laboratories. He provides in-house consulting to line organization projects for software engineering processes, methods, standards, tools, and training. He participates in process assessments and improvement programs, and provides support for configuration management, software inspections, and process automation. Dr. Knirk's primary focus is on the two complementary areas of software specification and testing, in which he works to bring more formal methods into more practical applications. He works actively on IEEE software engineering standards groups. He is a member of the ASQC Software Division Methods Committee. Dr. Knirk previously worked for Programming Environments, Inc., where he was the architect and principal developer of the automated software test design tool, T. That commercial product analyzed a formal software behavior description for testability, designed test cases for demonstrating that behavior, and generated actual test case data.

Presentation: Wednesday, April 2, Session B1: 10:15-11:45 am, Bldg 822, Rooms A&B

Catherine M. Kuhn, AS/FM&T Kansas City Site

Cathy Kuhn is a Staff Technical Programmer/Analyst from AlliedSignal Federal Manufacturing and Technologies / Kansas City Site. For the past eight years she has been a member of the Kansas City's Software Quality Assurance Group. During that time she has been involved in many Kansas City site and corporate software development and software quality improvement efforts. Currently, she is an active member of the Information Systems' Software Process Group and the Information Systems Software Quality Assurance Group. This presentation is based upon her work with the Information Systems' organization.

Presentation: Thursday, April 3, Session A4: 08:30-10:00 am, TTC Auditorium

Michael F. Lackner, AS/FM&T Kansas City Site

Michael holds a Masters of Science degree in Mechanical Engineering from the University of Missouri-Rolla, and a Bachelor of Science degree in Aerospace Engineering from the same institution. Michael is a Registered Professional Engineer in the State of Missouri. He is currently enrolled in the Doctor of Engineering program at the University of Kansas, specializing in the area of computer-aided and computer-integrated manufacturing. Prior to the SQA assignment eight years ago, he spent 4 years in process and product engineering in plastics products at AlliedSignal. He most recently completed the Blackbelt training in Six Sigma.

Presentation: Thursday, April 3, Session B5: 10:15-11:45 am, Bldg 822, Rooms A&B

David J. Leong, Sandia National Laboratories

David has been a Senior Member of Technical Staff at Sandia National Laboratories for seven years. He is currently the project leader of Sandia's Internal Web Technology Team, the EVE (Enterprise-information Viewing Environment) Team. He has been involved with Sandia's Intranet from its inception in the summer of 1994. David has performed many related activities along the way, including: HTML authoring, browser training, systems integration, application development, browser/server installations, etc.. Sandia's Intranet, which has been featured in WebMaster Magazine and Netscape's Customer Profiles, currently houses approximately 40,000 administrative and technical documents and is accessed on the order of 250,000 times per day.

Presentation: Wednesday, April 2, Session B2: 01:30-03:00 pm, Bldg 822, Rooms A&B

Stewart Meyer, Savannah River Site

Stewart Meyer is currently the software Quality Assurance/Configuration Management Coordinator for the NWPS (Nuclear Waste Processing Support) section for all systems supporting the DWPF (Defense Waste Processing Facility) at SRS (Savannah River Site.) This position involves developing/updating QA/CM plans for process control, process support, and manufacturing support systems. He also performs a hands on role as the configuration manager for the SCMS (Software Configuration Management System) in developing the layered applications, reviewing and approving the software changes, and performing library maintenance. He is the lead for all external (DOE/Site) audits regarding software at DWPF and also participates in committees and task teams at the division and Site level regarding software management procedures. A graduate of McMurry College (Abilene, Texas), with a Bachelor of Science in Computer Science and a background in management, his software engineering career includes; OS/Application development for the DOD MLRS (Multiple Launch Rocket System)

project, process automation design/development for DWPF, group supervisor for the process automation group at DWPF, and his current position (since 1993.)
Presentation: Wednesday, April 2, Session A2: 01:30-03:00 pm, TTC Auditorium

Jennie L. Negin, Sandia National Laboratories

Jennie Negin is manager of Web Services and IS Training at Sandia National Laboratories in Albuquerque, New Mexico. Sandia is a Department of Energy multiprogram national laboratory managed by Sandia Corporation, a Lockheed Martin company. Ms. Negin has been involved in development of many Information Systems at Sandia -- travel, library, procurement, property, security, personnel, nuclear materials management and radiation exposure. Ms. Negin was a consultant to the University of New Mexico (UNM) Law School and the UNM Maxwell Museum of Anthropology before coming to Sandia. Prior to that she was an internal consultant and systems developer at Los Alamos National Laboratories and the University of Florida Computing Center. Ms. Negin is a long time member of the Association of Computing Machinery and the New Mexico Network for Women in Science and Engineering. Jennie is a graduate of the University of Florida with a BSE and MA in Mathematics.
Presentation: Wednesday, April 2, Session B2: 01:30-03:00 pm, Bldg 822, Rooms A&B

Don Rathbun, AS/FM&T Kansas City Site

Don Rathbun holds a BSEE from Kansas State University, Manhattan, Kansas, and a MSEE from the University of Missouri, Columbia, Missouri. Business Systems Reengineering has been the focus of Don's recent assignments including project responsibilities on the Focused Factory initiative and the ISO9001 certification process from its outset. Current assignments include involvement with the NWIG (Nuclear Weapons Information Group), IMOG (Interagency Manufacturing Operations Group), and CAM-I (Consortium for Advanced Manufacturing International) Organizations. Don has made presentations at the last two IMOG meetings and at the September 1995 LLNL Software Engineering Seminar. Prior assignments included project responsibilities on major radar fuzing systems.
Presentation: Wednesday, April 2, Session A3: 03:15-04:45 pm, TTC Auditorium

Larry Rodin, Mason & Hanger Corporation, Pantex Plant

Larry has been 30 Years with Mason & Hanger Corporation working in Quality. He is a Project Manager at the Pantex Plant, Amarillo, Texas, Senior Member of the American Society for Quality Control, Member Software Quality Division. Larry has been an ASQC Certified Quality Engineer since 1970. In deference to the Year 2000 phenomena, his recertification date is December 31, 1999. Larry became Mason & Hanger's SQAS Primary Representative in the fall of 1990. He is currently serving as SQAS Vice-Chair, and previously has served as Secretary . Larry has also worked on many Work Item Groups and developed this presentation as research for one of these groups.
Presentation: Thursday, April 3, Session B5: 10:15-11:45 am, Bldg 822, Rooms A&B

Edward W. Russell, Lawrence Livermore National Laboratory

For the last 15 years Ed Russell has been involved in formal QA implementation on several projects at LLNL. He is currently working toward the ASME NQA-1 lead auditor qualification. Ed has also worked as an FEM code analyst at LLNL in the early 1980's. Ed's academic achievements include an M.S. degree from the University of California Davis in Mechanical Engineering and Materials Science.
Presentation: Wednesday, April 2, Session C1: 10:15-11:45 am, TTC Auditorium

Don Schilling, AS/FM&T Kansas City Site

Don Schilling is a Manager, Engineering Projects, for AlliedSignal Federal Manufacturing and Technologies at Kansas City. He has over 30 years of manufacturing experience in various assignments and responsibilities. He was responsible for the formation of the Kansas City Plant's Software Quality Assurance Group, which has reported to him since 1988. Don has championed numerous Software Engineering and SQA initiatives within AlliedSignal, the DOE Nuclear Weapons Complex, and in national and international forums.
Presentation: Wednesday, April 2, Session A3: 03:15-04:45 pm, TTC Auditorium

Presenter Biographies

(Alphabetical Order)

Joseph R. Schofield Jr., CQA, Sandia National Laboratories

Joe has been applying emerging technology for business and engineering solutions for the past 17 years. Joe guided the evaluation and implementation of Sandia's first large-scale CASE project using Texas Instrument's IEF. Current efforts include a client-served based object-oriented project with tens of millions of object instances. Joe has been a keynote speaker at the Structured Development Forum in San Francisco in 1988 and spoke on CASE at the National Conference on Information Systems Quality Assurance in Orlando, CASEWorld in LA, and the Piedmont CASE User's Group in Charlotte. Several articles on CASE were published by the Journal of Quality Data Processing, System Builder, and Managing System Development. A four-page interview was printed in the CASE Strategies Newsletter and another in Government Computer News. Joe has presented at USE, SHARE, GUIDE, and DOE-sponsored conferences. *The Next Silver Bullet* was published in 1995. His most recent article *The Year 2000 - Finally a Reality Check* is under publication review.

Presentation: Wednesday, April 2, Session A1: 10:15-11:45 am, TTC Auditorium

John K. Sharp, Ph.D., Sandia National Laboratories

John has been working in information systems during a 16 year career at Sandia National Laboratories. He has held technical and management positions covering information system design, application development and data administration functions. John has been working closely with Professor Shir Nijssen in the Netherlands who is creator of the NIAM (Nijssen's Information Analysis Methodology), which is the basis for our approach to Natural Language Modeling. Shir and John have co-chaired two international conferences on Natural Language Modeling and are writing a book on Natural Language Modeling that will be published this winter.

Presentation: Thursday, April 3, Session B5: 10:15-11:45 am, Bldg 822, Rooms A&B

Debra Sparkman, Los Alamos National Laboratory

Debra Sparkman is the Software Quality Assurance Manager for LLNL Safeguards and Security Engineering and Computations Division. She has been the SSEC quality assurance manager since January 1993 and test coordinator for the Argus Security System since October 1994. Prior positions at LLNL have included Quality Assurance/Test Coordinator for the Controlled Material Tracking System and staff member for the Fission Energy and Systems Safety Computer Safety and Reliability group. Other publications include: *SSEC SEI Experiences*, 1994 DOE NWC Software Quality Forum and *Standards and Practices for Reliable Safety-Related Software Systems*, 3rd International Symposium on Software Reliability Engineering. Ms. Sparkman received a Bachelor of Science, Computer Science in 1984 from the University of the Pacific. She is a member of the American Society for Quality Control, IEEE, and IEEE Computer Society.

Presentation: Wednesday, April 2, Session B1: 10:15-11:45 am, Bldg 822 Rooms A&B

Ann Stewart, Lockheed Martin Energy Systems

Ms. Stewart is the Quality Manager of the Data Systems Research and Development Program (DSRD) a division of Lockheed Martin Energy Systems (LMES) in Oak Ridge, Tennessee. She has more than 20 years experience as a software engineer and project manager with extensive experience in areas of quality assurance, performance measurements, and process improvement. She established and managed the Software Quality Assurance Program for the Oak Ridge National Laboratory (ORNL) in compliance with the Department of Energy (DOE) requirements and was responsible for their Performance Indicator and Metrics Program. Ann is a graduate of the University of Tennessee with a B.S. in Computer Science. She currently leads and manages DSRD's Process Improvement Initiative using the Software Engineering Institute's Capability Maturity Model (SEI/CMM).

Presentation: Thursday, April 3, Session A4: 08:30-10:00 am, TTC Auditorium

Nancy A. Storch, Lawrence Livermore National Laboratory

Nancy has over 30 years experience in design and development of scientific software, with emphasis in user interface design, computer graphics and software engineering. Her special interest is usability engineering. Recently Nancy has also become involved in software quality assurance and serves as SQA Engineer to two projects. Nancy is the LLNL SE/SQA Group Leader. Prior to coming to LLNL, Nancy developed software for submarine fire control systems. Throughout her career, Nancy has striven to be at the forefront of the application of computer science and software engineering. She has done graduate work in human factors, user interface design, computer science and physics. Her degree is in mathematics.

Presentation: Wednesday, April 2, Session B1: 10:15-11:45 am, Bldg 822 Rooms A&B

Presenter Biographies

(Alphabetical Order)

Michael Tiemann, Headquarters Department of Energy

Mike Tiemann has served in government service for 25 years. His career started in 1972 at Army Material Command Headquarters, as an Army Lieutenant working in Environmental Program Management. After this he spent almost 13 years at the Federal Energy Regulatory Commission as an Environmental Protection Specialist and a Computer Systems Analyst. In 1987 he joined Headquarters DOE as the Project Management Officer coordinating all information technology services and support for the Offices of the General Council, Inspector General, Hearings and Appeals and the Economic Regulatory Administration and the Board of Contract Appeals. Two years later, he was assigned the primary responsibilities for Information Management Planning at Headquarters. He is currently the Action Officer in the CIO's Information Architecture Team responsible for development of the Departmental Information Architecture. He is also the leader of the Information Management Planning and Architecture Coordinating Team or IMPACT, a diverse and professionally robust group of technology professionals from across the Department which supports the Architecture efforts. In addition to IMPACT, Mike has been a member of several Department-wide teams, and recently sat on an interagency panel on business modernization. Mike holds degrees in Architecture (BED, Texas A&M, 1972) and Systems Management (MSSM, U.S.C., 1977). He is a current member of the Energy Federal Credit Union's Information Technology Advisory Committee. He is married and has two children.

Presentation: Wednesday, April 2, Session A3: 03:15-04:45 pm, TTC Auditorium

Victor L. Winter, Ph.D., Sandia National Laboratories

Victor L. Winter received his Ph.D. from the University of New Mexico in 1994. His dissertation research focused on proving the correctness of program transformations. Currently, Dr. Winter is a member of the High Integrity Software (HIS) Project at Sandia National Laboratories. His research interests include trusted software, formal semantic models (graphical-based and symbol-based), theory of computation, automated reasoning and robotics. Dr. Winter can be reached by phone in the United States at (505) 284-2696, by fax at (505) 844 - 9478, or by email at *vlwinte@sandia.gov*.

Presentation: Wednesday, April 2, Session B3: 03:15-04:45 pm, Bldg 822, Rooms A&B

Alexander R. Yakhnis, Ph.D., Pioneer Technologies

Dr. Alexander R. Yakhnis is a consultant in design of dependable software/hardware systems. He received a Diploma in Mathematics from Moscow State University, Moscow, Russia. He worked as a computer programmer in Moscow, Russia and Houston, Texas. Alexander received an M.S. in Computer Science and a Ph.D. in Mathematics/Computer Science from Cornell University, Ithaca, New York. He then worked as a Research Scientist at Mathematical Sciences Institute, Cornell University. He worked at Command and Control Software Department at Sandia National Laboratories on High Integrity Software project from July 1995 to August 1996. His interests include correctness proofs for concurrent and sequential programs, theory of computations, winning strategies for two person games, control theory, hybrid systems, object-oriented methods, design of hardware/software systems. He can be reached by phone at (505) 298-5854 or by e-mail at AYakhnis@aol.com. Co-author Dr. Vladimir R. Yakhnis is a research scientist at Rockwell Science Center, One Thousand Oaks, CA. He received a Diploma in Mathematics from Moscow State University, Moscow, Russia. He worked as a computer programmer in Moscow, Russia and Houston, Texas. Dr. Yakhnis received an M.S. in Computer Science and a Ph.D. in Mathematics/Computer Science from Cornell University, Ithaca, New York. His research was in program correctness for concurrent and sequential programs, winning strategies for two person games, state transition systems and object-oriented methods. Dr. Yakhnis worked at the IBM Endicott Programming Laboratory as an Advisory Programmer until 1994. There he developed "Generic Algorithms" methodology that allowed the construction of mathematically proved software while "hiding" the actual proofs from the developers. The methodology was designed to take advantage of object class templates in C++ or Eiffel. He worked as a Visiting Scientist at Mathematical Sciences Institute, Cornell University until June 1995. There he developed the groundwork for the semantics of object-oriented stepwise refinements. He worked at Sandia National Laboratories at Albuquerque during 1995-1996. He can be reached by phone at (805) 373-4856 or by e-mail at vryakhni@scimail.risc.rockwell.com.

Presentation: Wednesday, April 2, Session B3: 03:15-04:45 pm, Bldg 822, Rooms A&B

ABSTRACTS

• 1997 • SOFTWARE QUALITY FORUM • 1997 •
DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE
ENGINEERING
QUALITY

Tutorial Abstracts: Tuesday, April 1 1997

Keynote Tutorial 09:00 - 11:00 am

Dr. David Lorge Parnas, MU/CRL
Z0: Design Through Documentation: The Path to Software Quality
TTC Auditorium

Although it is appealing, practitioners are not able or willing to write precise documents. Instead, they write vague blurbs that are useless to those charged with the next steps and cannot be subject to rigorous analysis. This tutorial describes how precise, complete, and testable documents can be produced for software and the ways that these documents can contribute to an improved software process.

Tutorials 01:00 - 03:00 pm

Dr. David Lorge Parnas, MU/CRL
Z1: Inspection of Critical Software
Bldg 823 Breezeway

This tutorial describes a procedure for inspecting software that consistently finds subtle errors in "mature" software, software that is believed to be correct. The procedure is based on three key ideas: the software reviewers are active not passive; reviewers focus on small sections of code; reviewers proceed systematically so that no case and no section of the program gets overlooked. During the procedure, the inspectors produce and review mathematical documentation. The mathematics and its notation allows them to check for complete coverage and to proceed systematically and in small steps.

Dr. John Sharp, Sandia National Laboratories
W1: Natural Language Modeling
Bldg 822 Room A

This tutorial describes a process and methodology that uses structured natural language to enable the construction of precise information requirements directly from users, experts, and managers. The main focus of this natural language approach is to create the precise information requirements and to do it in such a way that the business and technical exerts are fully accountable for the results.

Dr. Gerald McDonald, Sandia National Laboratories Consultant
X1: Definition and Documentation of Engineering Processes
Bldg 822 Room B

This tutorial is an extract of a two-day workshop developed under the auspices of the Quality Engineering Department at Sandia National Laboratories. The presentation starts with basic definitions and addresses why processes should be defined and documented. It covers three primary topics: (1) process considerations and rationale, (2) approach to defining and documenting engineering processes, and (3) an IDEF0 model of the process for defining engineering processes. Process considerations and rationale introduce models for documenting processes; describe the general architecture for product development; and define implications of immature processes versus those for mature processes. The approach describes the top-level subprocesses that make up the methodology for definition and documentation of engineering processes; namely: planning, gaining management approval for a process definition project, collecting data on the as-is process to capture current best practices within the organization, constructing a model of the as-is process, and verifying and validating that model. The final portion presents a four-level, hierarchical model that describes HOW to define and document an engineering process.

Faye Brown, Oak Ridge; Ray Cullen, Savannah River; Gary Echert, DOE/AL; Phil Huffman, Pantex. Cathy Kuan, AS/FM&T; Dave Peercy, SNL; Ellis Sykes, DOE/KCP; David Vinson, Pantex
Y1: How the NWC Handles Software as a Product
TTC Conference Room C

This tutorial provides a hands-on view of how the Nuclear Weapons Complex projects should be handling software as a product in response to Engineering Procedure 401099. The primary scope of the tutorial is on software products that result from weapons and weapons-related projects, although the information presented is applicable to other software projects. Processes for Identification, Qualification, Acceptance, and Delivery are described in terms of an extended case study.

Participant Restrictions: Must be a NWC or government employee; identification will be required. If you have questions, contact Dave Peercy, 505-844-7965, depeerc@sandia.gov.

Tutorials 03:15 - 05:15 pm

Dr. David Lorge Parnas, MU/CRL
Z2: Exercise and Discussion
Bldg 823 Breezeway

In this workshop, participants will be given a small program and will apply the documentation and inspections methods from the previous Design Through Documentation and Inspection of Critical Software tutorials. This will be followed by a discussion of previous experiences in a question and answer format.

Participant Restrictions: Must have attended both the Design Through Documentation and Inspection of Critical Software tutorials.

Dr. Dwayne Knirk, SNL
W2: Writing Testable Software Requirements
Bldg 822 Room A

This tutorial identifies common problems in analyzing requirements in the problem and constructing a written specification of what the software is to do. It deals with two main problem areas: separating the documentation of what is given from the documentation of what is to be created; and determining what facts about the subject software are to be documented, how they should be expressed, and how they are related.

Lt. Col. Nancy Crowley, USAF Phillips Laboratory
X2: Using COTS Software in Development Projects
Bldg 822 Room B

Commercial software and standards must be carefully evaluated prior to selection, carefully integrated, and used where appropriate to reap their benefits. This tutorial will discuss the experiences of the Space System Technologies Division of the USAF Phillips Laboratory in developing a COTS-based satellite control system.

Larry Iane and Randy Dabbs, Sandia National Laboratories
Y2: Software Inspection Process Overview
TTC Conference Room C

This tutorial provides an overview of the Software Inspection (In-Process Formal Review) Process and a mini-inspection workshop. The inspection roles and process steps are introduced. Participants are then divided into inspection groups for conduct of a mini-inspection to gain some practical experience with the inspection process. Discussion of the mini-inspection results concludes the workshop.

Presentation Abstracts: Wednesday, April 2 1997

Keynote Address, 09:00 - 10:00 am, TTC Auditorium

Capers Jones, McMaster University

Software Quality for 1997 - What Works and What Doesn't?

This presentation provides a view of software quality for 1997 – what works and what doesn't. For many years, software quality assurance lagged behind hardware quality assurance in terms of methods, metrics, and successful results. New approaches such as Quality Function Deployment (QFD) the ISO 9000-9004 standards, the SEI maturity levels, and Total Quality Management (TQM) are starting to attract wide attention, and in some cases to bring software quality levels up to a parity with manufacturing quality levels. Since software is on the critical path for many engineered products, and for internal business systems as well, the new approaches are starting to affect global competition and attract widespread international interest. It can be hypothesized that success in mastering software quality will be a key strategy for dominating global software markets in the 21st century.

Session A1: Software Management, 10:15-11:45 am, TTC Auditorium

Rodemy Ashby, Sandia National Laboratories

The Right Rock: Finding and Refining Customer Expectations

Figuring out what the customer wants, making sure the team understands the customer priorities, and negotiating what the customer can have for what they want to pay sets the scene for project success or failure. Getting a clear understanding of the political landscape (can't tell the players without a scorecard), and what is most important to them is essential. The people who will be using the system you produce, and those paying for it are rarely the same, and both must be satisfied for your project to be considered successful for the long term. Ways to bring internal differences of opinion to the fore, and flush out misunderstandings while educating the customers and project team about the cost of different decisions involves creating a vivid, shared understanding of how the target, completed system looks and operates. Approaches to these problems that I've found useful include 1)Erika Jones Organization Charting, 2)Customer Interviews, 3) Quality Functional Deployment and modifications with other "matrix-type" decision-making tools, 4)Creating an initial system acceptance test document, keyed to the requirements as requirements are negotiated, 5) Rapidly-Prototyping an example to show the customer, and modifying it per request if you have a configurable system and/or 5)Create the User Manual first. I'll illustrate the methodology and tool use with project examples.

David Harris, Sandia National Laboratories

TCAMS Lessons Learned

The overall objective of the Technical Control, Automation, Maintenance, and Support (TCAMS) system software is to facilitate the operation of the communication center within the Commander in Chief (CINC) Mobile Alternate Headquarters (CMAH). The software consists of about one million lines of source code and draws heavily upon industry standards such as Ada, SQL, Unix, and X-Windows. Several technical decisions that were made during the design and implementation of TCAMS went awry. This presentation attempts to provide insight into the root causes for these wrong decisions with the hope that these insights can lead to a better understanding of the software development process. An overview of the TCAMS project including some measures of the software complexity is included as introductory information.

Joseph R. Schofield, Jr., CQA, Sandia National Laboratories

The Next Silver Bullet - Or Just Another Shot in the Foot?

Repeated promises of productivity and quality improvements have seldom materialized with the introduction of new technologies. Marginal incremental improvements in productivity have become accepted as the norm. Joe shares a model that explains the unintended outcomes of technology hopping as well as how to extend the investment in a technology. Further implications exist for maintaining and improving the ability to manage the software development process as measured with instruments such as the Capability Maturity Model. The notion of the "in-flight magazine syndrome" only exacerbates efforts to stabilize and maximize our use of technology. This work was recently published as the lead article in Managing System Development.

Presentation Abstracts: Wednesday, April 2 1997

Session B1: Software Testing, 10:15-11:45 am, Bldg 822 Rooms A&B

Debra Sparkman, Los Alamos National Laboratory
A Working Testing Process

Argus is an automated security system deployed at 4 DOE and DoD facilities across the United States. Argus is composed of 3 major subsystems including over 20 software and firmware products. This paper describes the processes performed for testing the Argus Security System. The primary focus is on the independent testing activities. A brief description of unit, integration, and system testing performed by the development staff will be presented. Independent system testing is conducted by the Quality Assurance team using a separate test system. The independent testing process is a practical approach to implementing independent testing for an existing software-based system undergoing major enhancement development. The primary focus of testing is based upon system level regression testing, major feature enhancements and new product testing. Test planning is conducted prior to each testing activity. This planning is based upon risks associated with the degree of modifications and their impact on the customer operational systems. The testing process tracks anomalies detected during testing. From these anomalies, metrics are collected. The testing process is completed by the generation of a test report summarizing the testing activities. This work was performed under the US Department of Energy by Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

Nancy A. Storch, Lawrence Livermore National Laboratory
Testing the Design and Operations of a New Badging System

In response to a DOE mandated order to rebadge the Laboratory, efforts got underway to modify, replace, or adapt three major hardware and software systems. On a prior project, it had been helpful to conceptualize a complex system by gathering all interested parties together and systematically walking thorough a paper process description. However for the rebadging project we needed to do more than conceptualize the end system. We needed to test operational aspects and integration of the systems with users in an environment similar to the actual deployment environment. This became a full-scale mock exercise of rebadging. Each system was in a different state of development. One was somewhat operational and in testing, one had a working prototype, another was in the low-fi paper prototype stage. Also, they were being developed by different teams which rarely interacted with each other. These teams were focused on designing, implementing and unit testing within their system. Therefore, traditional integration and system testing of the combined systems was still a long way off. We wanted to save development time through early identification of issues, integration and operational problems, as well as usability problems. In the mock exercise we had 22 participants, who came from the development teams, operations and maintenance, user groups, managers and customers. Observers were selected both from within and outside the project. Observation posts were identified to include coverage of both individual system operation and overall operations. Operational scenarios based on prior rebadging experiences were developed with hypothetical person's to be rebadged. Realistic artifacts were acquired or created. Message and data communication between systems was modeled using paper messages and records. Logistics were handled to turn a mothballed badge office into the futuristic badge office of the exercise. The exercise took place over three half days. By the third day, we had created a variation on the operational scenarios which held promise for a more streamlined operation. We also gained insights on the interactions and communications between the systems and a list of important issues, problems and action items was produced. This talk will focus on our approach to testing and discuss its costs and benefits within the software development life cycle.

Dwayne Knirk, Sandia National Laboratories
Establishing a Three-Way Agreement: Specification, Code, Test

After we complete software testing, what do we know and what don't we know about the subject computing system? What kinds of system tests will further reduce our ignorance about the suitability and correctness of the computing system for its application? Software-intensive systems are expected to work in a particular environment to bring about desired effects in that environment. To accomplish these effects, the computing system must have a variety of interactions with that environment. Its capabilities and features are directed to establishing a variety of relationships between those interactions, including stimulus-response, constraint, and historical reference. To establish such relationships are the services provided by the computing system. The given environment and required effects in the problem are collectively documented as Problem Requirements. The computing system interactions and services are documented Behavior Specification. The relationship between these two sets of information is an explicit and verifiable *behavior design* task. The Behavior Specification characterizes a computing system independently of its application context. It provides a single reference point for all decisions of software architecture and implementation as well as for test case and testware architecture and implementation. Had we error-free development and testing processes, we should expect specific behavioral equivalencies between the pairs (specification, code) and (specification, test). To the extent these processes are not perfect, we may have defects in our code, our tests, or both.

This presentation explains the logical implications of the behavioral equivalencies, and interprets them in operational terms. It described how testing provides a means of comparing software and testware behaviors and evaluating their behavioral equivalence to the source specification. An integrated testing approach is devised for identifying deviations from the desired equivalence. The approach provides specific guidance for test design, test execution, code design, instrumentation and data collection, and evaluation of test results. The presentation concludes with a summary of what can be known through this logic-based testing approach and what remains to be examined in final system testing. The ultimate goal is validating the behavior of the resulting system through measuring its effects in the application environment.

Session C1: Software Quality for Scientific Applications, 10:15-11:45 am, Bldg 822 Room C

John Ambrosiano and Robert Webster, Los Alamos National Laboratory

Software Quality and Process Improvement in Scientific Simulation Codes

Today the reliance on high quality software is so important that standards for quality assurance are an integral part of software development in both the public and private sectors. Yet as a community, research scientists have not entirely embraced these methodologies and indeed are often leery of them. Is the problem with scientists, or with the standards? As the quest for excellence in software is extended to government research activities, we must understand this phenomenon and either modify how SQA standards are introduced to the scientific community, or understand why they are inappropriate, and if inappropriate, how to modify them. A salient aspect of research software development is that it usually involves a high degree of novelty and risk in the beginning. Only later, after evolving through a series of prototypes, are concepts considered sound enough to be turned into production software. This sometimes leaves scientists at a loss in deciding when to introduce their products into the SQA process. Too early and progress toward developing useful new concepts is impeded. Too late and high quality may be impossible to assure. In this paper we apply process analysis and knowledge acquisition methods to study the evolution of simulation models for nuclear technology applications from seminal prototypes to production design codes. Using use-case scenarios and interviews, we will build a model of the simulation software production process. We will also try to understand how the expert judgments of the scientists involved contribute to their ranking of a software product's quality and readiness for production. We will compare the results of this analysis to the practices recommended to attain SEI's CMM level 2 certification. In doing so we will try to answer the following questions: Which of these software development activities best fit a SQA model such as the SEI CMM and which do not? Is there a modification of the CMM that allows research scientists to more easily introduce their software at some appropriate stage into a standard SQA methodology?

Edward W. Russell, Lawrence Livermore National Laboratory

The SQA of Finite Element Method Codes used for Analyses of Pit Storage/Transport Packages

This presentation will describe the implementation of the SQA requirements of DOE/AL, *Quality Criteria (QC-1)*, Revision 8, July 1995, for Finite Element Method (FEM) codes used at the Lawrence Livermore National Laboratory (LLNL) for conducting design and confirmatory analyses on pit storage/transport package designs. This work satisfies the requirements of the Defense Technologies Engineering Division (DTED) Quality Assurance Policy and Plan for software management of activities associated with high risk, commensurate with the LLNL risk-based graded approach of SQA implementation. Element 14.0, "Software Quality Assurance," of QC-1 dictates the following requirements: (1) organization, tasks, and responsibilities; (2) verification and validation; (3) configuration management; (4) software documentation; and (5) reviews and audits. The FEM codes controlled by this program are utilized for structural and thermal analyses. As an example, DYNA3D which was originally developed at LLNL in the late 1970's, is a nonlinear, explicit, three-dimensional FEM solid and structural mechanics code for analyzing transient dynamic responses. Element formulations include one-dimensional truss and beam elements, two-dimensional quadrilateral and triangular shell elements, and three-dimensional continuum elements. Many material models are available to represent a wide range of material behavior. Sophisticated contact interface capabilities are available, such as frictional sliding and single surface contact. The size of DYNA3D is roughly 100,000 lines of code with 700 subroutines.

The SQA implementation for FEM codes is guided by the commercial standard, *ISO 9000-3: Guideline for Application of ISO 9001 to the Development, Supply and Maintenance of Software*, with increased SQA formality as necessary to satisfy the requirements of the nuclear standard, QC-1. The IEEE SQA standards and guides were consulted for guidance on format of the SQA Plan and associated specifications. The IEEE recommendations were tailored for this application to meet the requirements of the governing document, QC-1. The requirements within the DTED QA system to maintain and control high-quality software include the following documentation for FEM codes: SQA Plan, Requirements Specification, Design Description, Configuration Management System (CMS), and Verification and Validation Report. The CMS uniquely identifies and controls code versions and changes, as well as all pertinent baselines, procedures and documentation. Validation is accomplished by using a suite of analytically and experimentally validated benchmark problems.

Orval hart, Los Alamos National Laboratory

Software Quality Assurance at the Weapons Engineering Tritium Facility

The Weapons Engineering Tritium Facility (WETF) at the Los Alamos National Laboratory began construction in 1982 and finally received authorization to go on-line in 1991. It was the first nuclear facility to receive authorization under Admiral Watkin's increased formality-of-operations. Due to the many changes in DOE orders for nuclear facilities, the facility took longer than would be expected to get on-line. First it was "yes, we'll grandfather you in under the old regulations", then it was "no, you will have to meet the new regulations". The WETF went through several Readiness Assessments (then called Safety Appraisals) and the Operation Readiness Review before finally receiving approval to start operation. The WETF is unique, in that it was the first nuclear facility to place what was previously administrative procedures (interlocks, etc.) into software that was monitoring and controlling major operational aspects of the facility. The Instrumentation and Control System is designed to be inherently safe, i.e., if any of the computers controlling the facility fails, the systems will fail safe. That is, all valves are closed, all pumps stopped, etc. The facility cannot be operated in this mode, but is left in a safe state. Backup procedures allow for the safe restarting of the facility. Many of the operational systems are automatic in their nature, i.e., the ICS takes immediate action when an 'operational' abnormality occurs. Operation of the facility, in general, is performed from Operator Consoles in the Control Area, as opposed to through switches or hands-on in glove boxes. Due to this new method of operation, where software is involved in almost all operation and surveillance of the facility, the DOE was 'extremely' apprehensive about how all this was to work. This presentation will discuss the Quality Assurance program that was adopted to assure that the WETF could be operated in a safe and reliable manner.

Session A2: Software Engineering Processes, 01:30-03:00 pm, TTC Auditorium

Michael Bell, Lockheed Martin Energy Systems

Function Point Count Adjustment by Means of Scaling Touched Function Points

The talk presents an adjustment method to function point analysis that will quantify the work effort involved in a software enhancement project in terms of function points. The technique allows direct comparison of the magnitude of work with the magnitude of functionality change, which is also measured and expressed in terms of function points. The adjustment method is based on effort data that are ordinarily readily available, avoiding complex and costly data collection requirements or subjective judgments. The technique accounts for software development activities that are not directly measured by function point analysis. The adjustment may be used with attribute analysis to predict and baseline a wide range of software development efforts.

Stewart Meyer, Westinghouse Savannah River Co.

Using An Automated Code Management System To Improve Configuration Control Practices

Using a configuration management tool (software library) is not something new, several organizations and Sites use them. There are numerous tools commercially available, some claiming to be extensible and easy to customize. We took a very simple tool and added a front end to it. This front end is the interface to the software libraries and

shields the users from knowing the command language of the tool. In addition, the front end enforces the configuration control policies as set forth in the QA plans and procedures. The methods are then consistent across organizations and software products that are managed using this system as a tool. The front end is a developed product that may be used in other areas at the Savannah River Site, or other Sites, assuming the base system components are available. Although this system is used by one section at SRS, it could be available for use by others, without further investment in hardware. The key processes to improve were:
1. Identification of baselines; 2. Methods for verification of patches in a process control; environment; 3. Performing concurrent development in a controlled environment; 4. Methods for implementing periodic verification; 5. Configuration audits.

Outline of this presentation:
1. Description of deficiencies in previous software CM methods; 2. Description of methods and practices changed to foster improvements; 3. Description of SCMS system architecture and software tools; 4. Functional description of the SCMS from a user perspective relative to CM practices.; 5. Discussion on how key processes were improved.

Karen Jefferson, Terry Porter & Todd West, Sandia National Laboratories California
Software Engineering and Graphical Programming Languages
In a Work for Others project for the Air Force, The Advanced Atmospheric Research Equipment (AARE) software team used National Instruments' LabVIEW (a data flow graphical programming language) to control hardware used to collect samples of airborne particulate and gaseous species. Along with developing control and data collection software, the customer required MIL-STD-498 processes and documentation. This talk will discuss the processes and tools developed to support this project from the requirements to testing phase. In addition, unique aspects of the processes specifically tailored to graphical programming languages (such as coding standards, coding documentation, and configuration management) will be presented.

Session B2: Internet WEB Applications, 01:30-03:00 pm, Bldg 822 Rooms A&B

Kevin Hill, Pantex Plant
Internet Strategies for Engineers
The tools available on the Internet have the potential to help engineers reduce costs and increase productivity. As the amount of information available increases, so does congestion. Thus the Internet may be a victim of its own popularity. Strategies for effective use become necessary. How can an increase rather than a decrease in productivity be achieved? A survey of engineers' Internet usage is the first step in the search for ways to optimize time on the Internet. Two methods are used to advance this search. The first is the interpretation of survey results and follow-up questions. The second is via literature review. Standard search methods in conjunction with human networking can make the Internet a more productive tool. Concerns which have restricted Internet usage, such as reliability of sources, and unwanted leaking of information are addressed. Survey results and analysis provide a forum to initiate a discussion of this powerful tool's (the Internet's) impact on engineering efficiency and software quality.

David Leong & Fran Current, Sandia National Laboratories
Exploiting the Intranet: A New Architecture for Enterprise Information
The Intranet is an architecture for viewing information within the enterprise. This architecture is based upon the World Wide Web standards. With the global Internet as a proving ground, this architecture is proving to be a very formidable information system for corporate uses. One of the strongest features of an Intranet is its inherent cross platform support. Applications are functional on PCs, Macintosh, and UNIX platforms. The basic purpose of most Intranets today is the electronic delivery of corporate documents. These documents are typically of a static nature; corporate policy, manuals, newsletters. With the presentation capabilities of a web browser, compelling documents with integrated text, graphics, sound, and even video can be delivered via the Intranet. Hypertext links allow documents to be integrated in a way that makes knowledge even more accessible when compared to print media. Database access through a web interface is also a very powerful tool to the corporation. Query access to MIS systems typically living on the mainframe can now be made available to everyone on the Intranet. By adopting a three tiered client-server strategy, the web can become a graphical interface to legacy systems. Now the corporation's electronic phone book, human resource information, and financial reports can be delivered via a web

browser. Creating interactive web interfaces involves additional technologies. Security, workflow, and the 'Javas' (JavaScript from Netscape, and Java from Sun). In the area of security, authentication and authorization are very integral to client-server applications that allow the user to update information. Transactional based workflow is also necessary to route task requests among workgroups in the enterprise. Standard HTML forms offer a stateless user interface. By using Java and JavaScript, one can create applications that establish connections and provide field level event handling on the presentation tier of the application.

This new paradigm for delivering information is not without its share of challenges. Cultural and political barriers exist that must be addressed with the same vigor as the technical challenges. An enterprise solution must have input from users within that enterprise. It is necessary to show the users how the enterprise Intranet can make their daily job easier. The enterprise web (Intranet) is a scalable productivity tool for the corporation that will enhance the way employees do their job.

Jennie Negin, Sandia National Laboratories
"Rightsizing" Software Quality for a Web Services Organization
This presentation describes variations of software engineering and project management as applies to an organization that is supplying services for Sandia National Laboratories' Intranet on a cost recovery basis.

Session A3: Software Process Improvement I, 03:15-04:45 pm, TTC Auditorium

Don Schilling, AS/FM&T
Quest for Excellence 1996: Reaching for the Stars
In the Spring of 1995, a need for software process improvement arose when DOE requested that certain software be handled as product. A solution was needed quickly to meet critical production schedules. This presentation summaries the actions and the processes that were followed in developing and implementing a solution for FM&T to handle product software. It discusses the Total Quality improvement process used and the outputs which were developed. The presentation is based upon the presentation given at AlliedSignal in the Quest for Excellence competition. The Quest for Excellence is a corporate-wide competition designed to show case process improvement. The team won the Teamwork Award for their efforts in defining a system which worked successfully and minimally impacted critical production schedules. This presentation also ties in with the tutorial of how the Nuclear Weapons Complex projects should be handling software as a product in response to Engineering Procedure EP401099. It shows one sites struggle in defining a workable process to meet customer expectations.

Don Rathbun, AS/FM&T
Command Media System at the Kansas City Plant (KCP)
The Kansas City Plant was certified to the ISO9001 Standard in April 1995, following a successful audit by Third Party Auditor, Det Norske Veritas (DNV). The KCP has also successfully passed three six-month periodic audits by DNV subsequent to receiving certification in 1995. A new on-line Command Media System was developed and implemented to help ensure control of the documents associated with the KCP business processes. This control is demanded by the International Organization for Standardization to receive ISO9001 certification. The new on-line system is based upon the **KCP Business Model**. New Process Descriptions (PDs) and Work Instructions (WIs) were created by the KCP Process Owners for each process and released in the **Command Media System**. The development of the **KCP Business Model** and the new **Command Media System** will be discussed during the presentation, including how to access the system and structure of documents within the system. Also to be discussed are the operational structure in place to manage Command Media and proposed improvements to the system in 1997.

Michael Tiemann, Headquarters Department of Energy
Departmental Information Architecture
The Information Technology Management Reform Act of 1996 requires agency Chief Information Officers (CIO) to develop, maintain and facilitate the implementation of sound and integrated information technology architectures. Notwithstanding this act's formalization of this recent requirement, the Department of Energy's Designated IRM official, the Assistant Secretary for Information Management, decided well over a year ago to

establish a Departmental or enterprise-wide Information Architecture. As described in the published document the Department of Energy Information Architecture, Volume One, The Foundations, dated March 1995, the Departmental Information Architecture is a high level, principles and standards based framework within which the majority of programmatic, organizational and field site architectures should be developed and implemented. It is intended to be a template that can guide all information management acquisitions, activities, projects, developments, solutions and implementations. In order to help achieve this goal additional documents have been written to further explain and define the Architecture. Two additional volumes, Baseline Analysis and Guidance, (Information Architecture Volumes Two and Three, respectively) have been published to describe the current or defacto Departmental Information Architecture and to provide specific guidance on the establishment of Information Architectures within other organizational components of DOE. The intent is that they will be treated as nested organizational subarchitectures within the overarching Departmental Architecture. The Baseline Analysis document identifies many of the challenges facing the Department in regard to the divergent, often incompatable, obsolete, or non interoperable technologies and systems currently deployed as well as the duplication and redundancies, inheritant in the applications and data structures. The Guidance document provides useful guidelines for architectural activities in all life cycle phases for DOE and its partners and stakeholders. In addition, there are several architectural standards related documents being published and widely distributed. Presently there are numerous architectural efforts underway at various sites and within several of the major programs. It is the intent of the Office of the CIO to support these activities and to grow this approach further throughout the entire DOE community.

This presentation will summarize the above documents and related actions and activities to date regarding the Departmental Information Architecture Program and explain the future directions as the Departmental Information Architecture becomes the Chief Information Officer's central component in the comprehensive Departmental Information Management Strategy.

Session B3: High Integrity / Formal Methods I, 03:15-04:45 pm, Bldg 822 Rooms A&B

Larry J. Dalton & Marie-Elena Kidd, Sandia National Laboratories

Meeting the High Integrity Software Needs of Today and Tomorrow

Quantifiable measures of the reliability, safety and security for software-based systems remains an elusive goal even after decades of research. Such systems continue to be a major source of safety and security catastrophes. These catastrophes include the of loss of life, environmental or economic damage, and loss of public confidence. In spite of these catastrophes, the usage and complexity of software-based systems in high-consequence applications is continuing to increase. This growth, with the associated safety and security risks, presents a national challenge to the R&D community. Sandia National Laboratories established a High Integrity Software research project in 1995 to begin to address the challenge. The first of two research areas, the Correctness Track, is focused on creating the ability to create software that is "correct by construction." Research projects include advanced concepts for the capture of software specification/requirements, validation through intuitive and visual reasoning and mathematics for correctness preserving transformations covering all steps from specifications to executable code. The second research area, Systems Immunology, is directed towards in-situ techniques and technologies to enable real-time fault detection and safing control (fault response). Systems Immunology research projects include Software Event Execution Reliability (SEER), Digital Isolation and Incompatibility, and Top-Down Fault Analysis of Microprocessor Systems.

Victor Winter, Sandia National Laboratories

An Overview of the AST Software Construction Methodology

AST is a formal method that is being developed within the High-Integrity Software (HIS) project at Sandia National Laboratories. AST stands for Abstraction, Synthesis, and Transformation. Within AST, abstraction, deductive synthesis, and transformation techniques are used to enable the automation of a significant portion of the software construction and verification process. Furthermore, within AST the impact of human involvement is limited to such an extent that it can be formally verified. In AST, the role of synthesis is to construct abstract algorithmic solutions to problems from nonalgorithmic specifications (e.g., precondition and postcondition pairs). This is accomplished by using a sophisticated search engine such as an automated reasoning system to resolve (or remove) the nondeterministic choices that are present in the initial nonalgorithmic specification. Complementing synthesis within our methodology, the role of refinement transformation is (1) to optimize solutions that are obtained in the synthesis step, and (2) to introduce low-level (e.g., machine oriented) algorithmic details for the purpose of (ultimately) producing a machine executable implementation satisfying the original nonalgorithmic specification. Currently, AST is restricted to a somewhat well-behaved subset of reactive systems that we refer to as single-agent reactive systems. Because the burden placed on the synthesis portion of our methodology can be enormous we have found it useful to distribute the synthesis process over an abstraction hierarchy. In order for this approach to succeed, the abstraction hierarchy must have the property that a solution at one level of abstraction "benefits" or "can be used to guide the construction of" a solution at the next lower level of abstraction within the hierarchy. In essence, what is going on is that an algorithmic skeleton is being synthesized at one level of abstraction and is then in some sense "passed down" to the next level in the abstraction hierarchy. This process continues until a machine executable algorithm has been obtained. An undesirable consequence of this approach is that the synthesized algorithms tend to be sequential in nature (i.e., completely parallel or concurrent solutions cannot be readily synthesized in this framework). Fortunately, it is well within the capability of refinement transformations to take a sequential specification of a problem and to then transform it into an efficient parallel solution. This talk gives an introductory overview of AST as well as a brief example of how transformation techniques can be used to compliment synthesis.

Alex Yakhnis & Vladimir Yakhnis, Pioneer Technologies

Towards Automated Construction of Dependable Software/Hardware Systems

Many observers have recognized that software/hardware systems built by Government and by Industry can be very complex. It may be difficult to establish dependability and functionality of such systems. Here are some of the questions that existence of such systems raises. (1) How a software/hardware system should be documented in order to be understood by users and customers of various backgrounds? (2) What should be established in order to conclude that the system is acceptable? (3) Finally, since the system intent is often evolving in the course of system design and use, how should we modify the system to reflect this evolution while preserving the system dependability? Here are some of the approaches which are presently used in Industry in order to resolve the above questions: (1) Presenting a system as a hierarchy of models where the levels of the hierarchy would represent various levels of abstraction. Then an observer could look only at the levels of hierarchy that do not have details that are of no interest for the observer. Another approach to document a system is the object oriented approach. Here, systems are understood through understanding of individual objects from which the system is composed and of interactions among objects. Usually, the approaches are not combined. Also, thus far applications of object-oriented approach were mostly limited to the software-only system components. (2) Exhaustive testing that system behaviors satisfy the requirements. The problem here is that exhaustive testing is not possible even for moderately complex systems. An approach to overcome this is to formalize system requirements, to accurately model the system that is being constructed, and to produce a mathematical proof that the system model satisfies the requirements. However, so far, this was done with respect to system components only. Moreover, correctness proofs are usually not applied to several software constructs, e.g. communication among objects. (3) Maintaining system requirements, models, design, and simulation information in a single data base capable of containing many system versions. However, such a data base alone would not insure that the next version would be as dependable as the previous one. In this talk we will describe a direction of work on how to get better answers to the above questions on the basis of mathematical modeling, formal methods, and multi-agent strategic approach. These methods are aimed to achieve industrial strength automation of system specification, design, correctness proofs, and maintenance without exhaustive testing. Mathematical modeling and formal methods are beginning to be recognized in Industry as promising approaches to deal with high complexity of systems. The formal methods groups have been formed at Intel, Motorola, and HP.

Session A4: Software Process Improvement II, 08:30-10:00 am, TTC Auditorium

Cathy Kuhn, AS/FM&T

AlliedSignal Capability Maturity Model Assessment & Improvement Processes

This presentation provides a summary of the processes used by AlliedSignal to assess progress against the Software Engineering Institutes Capability Maturity Model and the use of this assessment data to plan and implement organizational process improvements. AlliedSignal corporate has committed to achieve CMM Level 3 at sixteen of its key business units within the next three years. This strategy is a key component in an effort to develop a competitive advantage in the aerospace business. What's unique about this initiative is that it is being applied to Information Systems. Staff at the AlliedSignal Aerospace Center for Process Improvement and the AlliedSignal Corporate Information Systems group have developed the methods and materials to assist business units in this strategy. Six certified SEI examiners have been trained to conduct progress assessments and supporting material have been developed. Included in this material is a process guide for using assessment results to plan and drive organizational improvement. Each business unit is scheduled for a formal assessment every 6 - 8 months. Quarterly self-assessment metrics are provided by each business unit and are used to track progress. The presentation focuses on the continuous improvement cycle implemented at the Kansas City site as a result of repeated assessments and planning.

Ann Stewart, Lockheed Martin Energy Systems

Lessons Learned on Utilizing the SEI/CMM in the Federal Government Work for Others Environment

Data Systems Research and Development (DSRD), a division of Lockheed Martin Energy Systems, Inc., has developed a specific approach in applying the Software Engineering Institute's Capability Maturity Model (SEI/CMM) that has been successful in our customer focused environment of research and development within the federal government. This approach is based on establishing an orderly and understood infrastructure consisting of three major building blocks, controls, processes, and information. This infrastructure is sustained through a strong quality program emphasizing technical, peer, and management reviews and quality audits and surveillances. This paper describes the tactical application of this approach and DSRD's experiences and lessons learned in three years of implementation.

Gail Benefield, Lockheed Martin Energy Systems

"SWiM" Your Way to Software Quality

A company quality improvement effort has many aspects. At Lockheed Martin Energy Systems at Oak Ridge, a software development methodology called Software WorkPackage Methods (SWM) has been created and can be considered part of the company's quality improvement efforts. SWM is a methodology for managing, developing, and supporting information system projects and applications. It is composed of methodology guidelines, role definitions and assignments, and work packages. The work packages are in the form of work breakdown structures suitable for project estimating, planning, and management. SWM provides development and support processes which are customizable, yet repeatable. It keeps pace with new software development methods and techniques and provides automation support for the project estimating, planning, and management.

Session B4: High Integrity / Formal Methods II, 08:30-10:00 am, Bldg 822 Rooms A&B

Mikhail Auguston, New Mexico State University

Debugging Automation Tools Based on Event Grammars and Computations over Traces.

Dynamic program analysis is one of the least understood activities in software development. A major problem is still the inability to express the mismatch between the expected and the observed behavior of the program on the level of abstraction maintained by the user. We propose to design software testing and debugging automation tools based on assertion language concepts as well as on precise program execution models. We are developing a PARFORMAN language for the description of computations over execution histories of target programs that provides a basis for tool development for assertion checking, debugging queries, execution profiles, and performance measurements. We use assertion language mechanisms, including event patterns and aggregate operations over event traces, to describe typical bugs and debugging rules, and to evaluate debugging queries. An event grammar provides a sound basis for assertion language implementation via target program automatic instrumentation. These tools and methods may be useful for software testing, debugging, documentation, and maintenance of software systems. Our approach is nondestructive, since assertion texts are separated from the target program source code and can be maintained independently. Assertions can capture the essential dynamic properties of a particular target program and can formalize the general knowledge of typical bugs and debugging strategies. Event grammars may be designed for sequential as well as for parallel programs. Examples of assertions and debugging rules for run-time detection of bugs and bug localization are presented. We have developed a prototype implementation of the assertion checker and debugging rule evaluator.

Marie-Elena Kidd, Sandia National Laboratories
A Method for Critical Software Event Execution Reliability in High Integrity Software
When high consequence systems rely on software for critical control functions, they require high integrity software. A major concern of high integrity software is ensuring the faithful execution of critical software driven event execution sequences. To meet system performance criteria, high integrity software must execute correctly and reliably. In addition, in the presence of transient hardware or software faults in both normal and abnormal environments, safety and security objectives must be maintained. A reliable, repeatable method and application techniques are needed to address these issues. Our technical approach involves an in-situ (embedded in the software) dynamic (run-time) fault detection and mitigation method for ensuring critical event execution sequences in high integrity software. Our method is based on deriving a mathematical description of the critical software controlled event execution sequence from a software model or the software requirements, embedding check points and update points based on that mathematical description into the target code, and adding a software module that implements the functionality of the underlying mathematical model. This extra software is added to the target code to verify that the correct software event execution sequence is maintained.

John Sharp, Sandia National Laboratories
Business Rule Enforcement Via Natural Language Modeling
The topic of my presentation will be business rule enforcement using Natural Language Modeling. A well defined procedure will be explained that allows subject matter experts to specify requirements and then be held accountable for them. I will convey a fundamental truth: 'That requirements can always come in the form of precisely analyzed, elementary natural language sentences.' Requirements include both facts that result in tables for populating data and business rules that do not change the table structure, but they do restrict the population of otherwise good facts in existing tables. A brief review of analysis results will now be discussed to allow you to understand a portion of the capabilities of this procedure. The following sentences all require external data to populate the instances of knowledge that is desired to be maintained.

Professor has degree in subject.
Course requires minimum degree level in subject
Professor teaches course.

Referential integrity applies, in that populations of the third sentence must be from known populations of professor and course in the first two. These sentences cannot enforce the business rule that a professor must be allowed to teach a course before he can be assigned to teach the course. I define this requirement as a "business rule" because no other fields are needed to store the data than appears in the previous three, but the rule can be enforced by starting with the derived sentence:

Professor is allowed to teach course.

This sentence is a derived fact (an SQL query can be established with appropriate triggers) and a set theory rule can be applied to restrict the population of the third sentence. This rule is:

The professor teaching a course must be a subset of the professors who are allowed to teach that course.

All "business rules" can be written as either direct set theory constraints against facts that are externally populated or as derived fact(s) and set theory constraints against other facts or derived facts. The benefit of Natural Language Modeling is that all of the experts and users can understand and be held accountable for the specification of the design because it always exists as a set of understandable sentences. Transformations of this knowledge set can be made into any graphical technique (including relational and object-oriented methods) but I do not know of any graphical presentation that can handle all of the knowledge captured.

Presentation Abstracts: Thursday, April 3 1997

Session A5: Software Process Improvement II, 10:15-11:45 am, TTC Auditorium

Larry Desonier, Sandia National Laboratories

Guns for Hire - Experiences of Quality Software Development Under the Gun

In today's software development environment, a major concern is the quality of the software. Sometimes getting the quality boxes checked seems to take precedence over implementation and delivery. There exists a way to both perform rapid development and have a quality product. There is a saying that 80% of the work gets done in 20% of the time, and the rest may never get finished. The question here is simply can quality software be developed when (1) 80% of the dollars are spent, (2) only 20% of the work is complete, (3) there is 6 weeks to delivery, and (4) no code has yet been written (and the team estimate is many months to code completion). This is just the situation for a "Guns for Hire" team. In some organizations this would be known as a type of "Skunk Works" or software "Swat Team." Our experience has shown that with the right size team, the right skills mix of individuals, and some disciplined development practices, quality software can be developed and projects can be saved. This discussion will reflect on projects accomplished in just this manner: projects developing user interface or command console software, a PC-based graphics display for alarm annunciation, material and personnel tracking systems, a taxi-way monitoring system, and others. This would not be possible without an experienced team, standard development practices, actually reusing code (yes, it is possible), and strictly disciplined development practices. The successes of this process paradigm is why the "Guns for Hire" team is continuously in demand.

Bruce Johnston, Pantex Plant

The Year 2000 Challenge: A Project Management Perspective

Today we are faced with the biggest threat to computing ever discovered. As the year 1999 makes its final tick into the year 2000, many time-sensitive business applications like accounting, payroll, project management and many, many more will either completely fail or make disastrous mistakes. Why will this happen? In the 1970's and early 1980's when data processing shops were buying mainframe computers by the truckloads, the high cost of memory persuaded programmers to drop the century digits from a date field to save two bytes of memory. Although shortsighted, this practice was universally accepted because these early computer applications were not expected to be in operation today. Using only two digits for the year 1996, for example, is represented simply "96." This means when the year 2000 arrives, tens of thousands of old software programs still in use will think the year is 1900. If the doomsday predictions hold true only half of the worlds computer applications will be completely fixed or replaced before December 31, 1999. This will be a real challenge: finding, changing, and testing date parameter software changes and the challenge will be an even greater Software Quality Assurance problem for legacy programs. This paper will address the year 2000 challenge from a project management perspective and give insight into managing the project of the century.

Curt Holmes, Lockheed Martin Energy Systems

Year 2000 Awareness

The Date 2000 challenge has been referred to as both a technical problem and a business risk. It has also been called the single largest information technology project which corporations and government agencies will undertake in the next several years. Current estimates for the cost of remediating Date 2000 software problems in the U.S. range between $600 billion to $1000 billion, and are increasing. The problem will affect all hardware platforms and all software systems in various ways and with unpredictable results. On average, organizations are finding that over 80% of their existing applications portfolio is impacted by two-digit year date processing (i.e. 19xx). Some systems will shut down, while others will corrupt data and generate spurious output. In all cases, the business operational risks, resulting from the failure of internal operating systems, far out weigh the potential cost of remediation. The purpose of this presentation is to create an awareness of Year 2000 issues, promote collaboration among DOE sites, and propose electronic sharing of resources to save money in infrastructure and software resources costs.

Session B5: High Integrity / Formal Methods II, 10:15-11:45 am, Bldg 822 Rooms A&B

John Hare, AWE UK

ISO and Software Quality Assurance

Emerging International Standards now promise a global approach to Software Quality Assurance; ISO/IEC 12207 provides a framework for Software life cycle processes that has already attracted the attention of both US and UK customers. The ISO 'SPICE' standards give international weight to the concept of self-assessment, and a model that could take the SEI CMM world-wide. Previously our customers have independently developed their own standards, which include QC-1, AQAP 150 and DefStan 05-95. Whilst ISO9000-3 can be adopted for assessment, this is non-mandatory and has not been well received in the US although widely used in Europe. TickIT, the scheme for third party assessment, could refocus on ISO/IEC 12207. This presentation reviews customer requirements and the new International Software Standards, with particular emphasis on ISO/IEC 12207 and SPICE. It is concluded that ISO Standards will become a dominate driver for Software Engineering, and could now succeed in promoting a world-wide approach.

Larry Rodin, Pantex Plant

Licensing and Certification of Software Professionals

This report presents information on software engineering certification programs, licensing of software engineers, reasons to become certified, certification as a condition of employment, the body of knowledge and examination structures for the certification programs, and an overview of the Institute of Electronic and Electrical Engineers recommendations for software engineering as a profession.

The Software Quality Assurance Subcommittee of the Nuclear Weapons Complex Quality Managers completed a Work Item to research software-related certification and licensing efforts and provided status reports to the Quality Managers. A white paper was a significant result of that work item and this presentation has been updated to reflect changes in the licensing and certification processes.

Certification is a voluntary process administered by a professional society. Licensing is a mandatory process administered by government. Two professional organization have been identified as having or developing certification programs, and one state has developed legislation for a licensing program:

- The Institute for Certification of Computer Professionals (ICCP) has two levels of certification — Associate Computing Professional, and the Certified Computing Professional;
- The American Society for Quality Control has implemented its program for Certified Software Quality Engineer;
- New Jersey is the only state identified as actually enacting software development legislation, their licensing program covers "software designers".

Included in the presentation are considerations and implications for licensing and certification. What problems are we solving by having licensing and certification. Equal Employment Opportunity (EEO) laws will be discussed to address issues such as: can certification testing being considered discriminatory; or can certification as a condition of employment be considered discriminatory.

Michael Lackner, AS/FM&T

Operational Excellence (Six Sigma) Philosophy: Application to Software Quality Assurance

The Kansas City Plant, as part of AlliedSignal Aerospace, has committed fifteen individuals to each receive four months of training in Six Sigma and at least a year in the position established as a Blackbelt. Six Sigma is a philosophy of doing business encompassing the methodologies of defect prevention (versus defect detection) through the use of statistical tools, i.e., process mapping, design of experiments, and process controls. Business includes providing any product or service. Continuous improvement to the way business is performed is achieved through the identification of optimal target values in products and processes, and the reduction of variation around those targets. An overview of the tools and training will be discussed, along with the application to the processes included in Software Quality Assurance.

Session Z: Keynote Tutorial

Dr. Dave Parnas
NSERC/Bell Industrial Research Chair in Software Engineering
McMaster University
Ontario Canada

Session	*Title*
Z0	"Design Through Documentation: The Path to Software Quality"
Z1	"Inspection of Critical Software"
Z2	"Exercise & Discussion"

Dr. David Lorge Parnas

Keynote Tutorial

Professor David Lorge Parnas, PhD holds the NSERC/Bell Industrial Research Chair in the Communications Research Laboratory, Department of Electrical and Computer Engineering at McMaster University in Hamilton, Ontario, Canada. His primary area of interest is to promote the discipline and body of knowledge to Software Engineers as practiced by engineers in other fields. By studying the problems of software engineering since 1965, Dr. Parnas has developed principles and methods that have value to real world problems. In recognition of his accomplishments, he has received numerous honors, including election as a Fellow of the Royal Society of Canada and a Fellow of the Association for Computing Machinery.

Dr. Parnas will share his experience and knowledge by leading the three workshop/tutorials described on the next page.

David Lorge Parnas
NSERC/Bell Industrial Research Chair in Software Engineering
Communications Research Laboratory
Department of Electrical and Computer Engineering
McMaster University Hamilton, Ontario
Canada L8S 4K1
Phone 905-648-5772
FAX 905-648-5943
Email parnas@qusunt.crl.McMaster.CA

Z0: April 1 1997, 09:00 - 11:00 am, TTC Auditorium
"Design Through Documentation: The Path to Software Quality"

In traditional engineering design, a series of documents precedes the actual construction of the product. These documents permit review and analysis, then after revision, serve as input to the next phase. When the (inevitable) errors are discovered and changes are required, the design documents already on file are updated and reviewed again. Each new refinement is reviewed against the previous documents.

In software design this "waterfall" method is almost never applied. Although it is appealing, practitioners are not able or willing to write precise documents. Instead, they write vague blurbs that are useless to those charged with the next steps and cannot be subject to rigorous analysis.

We will describe how precise, complete, and testable documents can be produced for software and the ways that these documents can contribute to an improved software process.

Z1: April 1 1997, 01:00 - 03:00 pm, Bldg 823, Breezeway
"Inspection of Critical Software"

Software is devilishly hard to inspect. Serious errors can hide in a software product for years. People are hesitant to employ software in safety-critical applications. Many companies are finding correcting and improving software to be an increasingly burdensome cost.

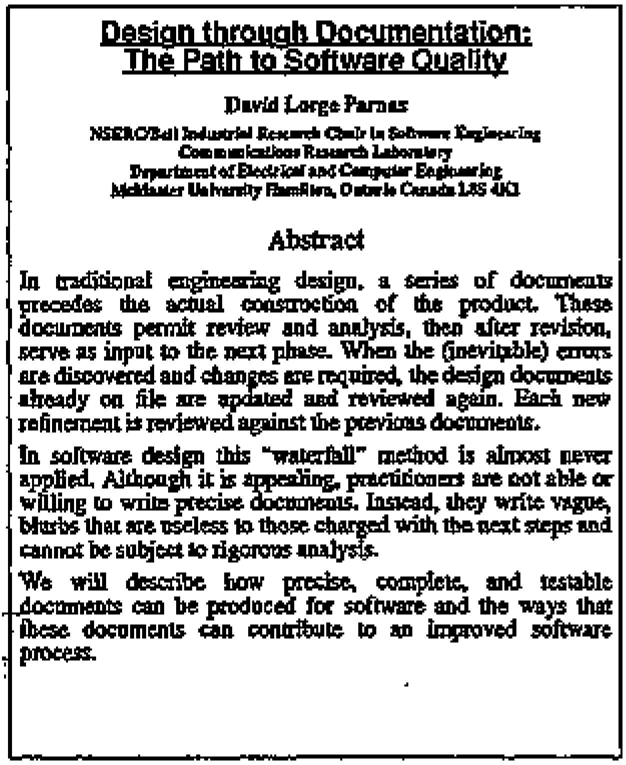
This talk describes a procedure for inspecting software that consistently finds subtle errors in "mature" software, software that is believed to be correct. The procedure is based on three key ideas:

- The software reviewers are active not passive
- Reviewers focus on small sections of code.
- Reviewers proceed systematically so that no case and no section of the program gets overlooked.

During the procedure, the inspectors produce and review mathematical documentation. The mathematics allows them to check for complete coverage; the notation allows them to proceed systematically and in small steps.

Z2: April 1 1997, 03:00 - 035:00 pm, Bldg 823, Breezeway
"Exercise & Discussion"

Participants will be given a small program and will apply the documentation and inspection methods to them. This will be followed by a discussion of previous experiences in question and answer format.



Design through Documentation: The Path to Software Quality

David Lorge Parnas

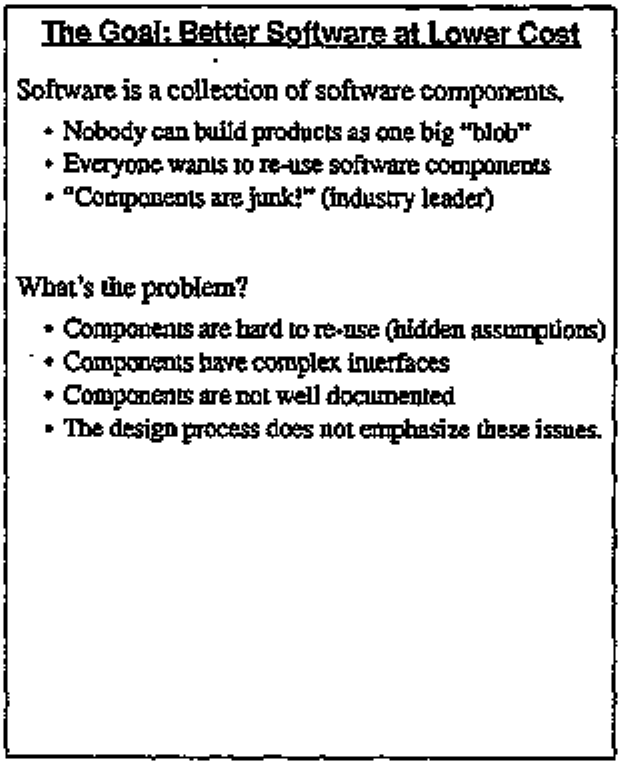
NSERC/Bell Industrial Research Chair in Software Engineering
Communications Research Laboratory
Department of Electrical and Computer Engineering
McMaster University Hamilton, Ontario Canada L8S 4K1

Abstract

In traditional engineering design, a series of documents precedes the actual construction of the product. These documents permit review and analysis, then after revision, serve as input to the next phase. When the (inevitable) errors are discovered and changes are required, the design documents already on file are updated and reviewed again. Each new refinement is reviewed against the previous documents.

In software design this "waterfall" method is almost never applied. Although it is appealing, practitioners are not able or willing to write precise documents. Instead, they write vague, blurbs that are useless to those charged with the next steps and cannot be subject to rigorous analysis.

We will describe how precise, complete, and testable documents can be produced for software and the ways that these documents can contribute to an improved software process.



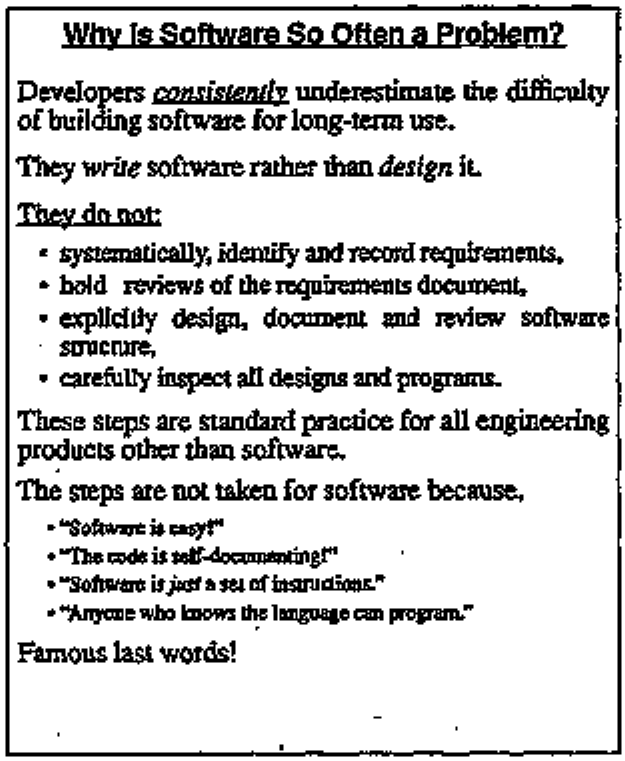
The Goal: Better Software at Lower Cost

Software is a collection of software components.

- Nobody can build products as one big "blob"
- Everyone wants to re-use software components
- "Components are junk!" (industry leader)

What's the problem?

- Components are hard to re-use (hidden assumptions)
- Components have complex interfaces
- Components are not well documented
- The design process does not emphasize these issues.



Why Is Software So Often a Problem?

Developers *consistently* underestimate the difficulty of building software for long-term use.

They *write* software rather than *design* it.

They do not:

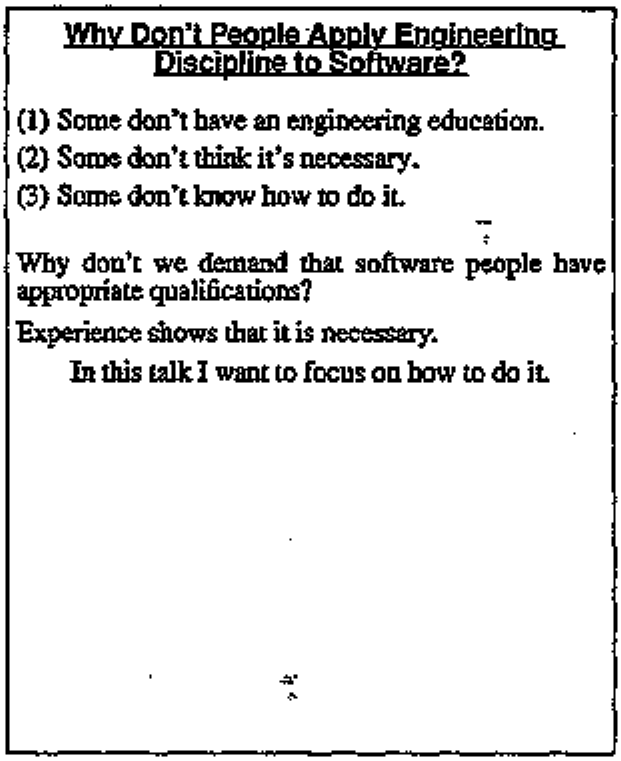
- systematically, identify and record requirements,
- hold reviews of the requirements document,
- explicitly design, document and review software structure,
- carefully inspect all designs and programs.

These steps are standard practice for all engineering products other than software.

The steps are not taken for software because.

- "Software is easy!"
- "The code is self-documenting!"
- "Software is *just* a set of instructions."
- "Anyone who knows the language can program."

Famous last words!



Why Don't People Apply Engineering Discipline to Software?

(1) Some don't have an engineering education.

(2) Some don't think it's necessary.

(3) Some don't know how to do it.

Why don't we demand that software people have appropriate qualifications?

Experience shows that it is necessary.

In this talk I want to focus on how to do it.

The Relevance of Documentation

"We have better things to do than document"

"We sell code, not design documents."

But,

- We cannot collect, review, or check requirements for completeness unless we document them.
- We can't make, review, or live up to structural decisions unless they are documented.
- We can't inspect designs, without design documentation.
- We can best inspect programs with the help of program documentation.

Design through documentation is the key to better software.

Two Aspects of Better Software:

(1) Better design

(2) Better documentation

Two Aspects of Better Documentation

(1) Better design (easier to document)

(2) Using mathematics, which is

- more compact,
- less ambiguous,
- more useful (mechanically interpretable)

than natural language.

Two Aspects of Better Design

(1) Following Software Design Principles

(2) Raising consciousness: documenting design.

In other words, design and documentation are irrevocably linked. They help (or hurt) each other.

Writing Down Requirements

The most costly errors are those made early in the process - they are the hardest to change.

Misunderstandings about requirements lead to early mistakes.

Programmers need to be told what is needed.

They must also be told what is subject to change.

Requirements must be subject to review.

Safety reviews of software must be based on a previously agreed statement of requirements.

Maintenance actions must be based on requirements.

None of these things is possible unless we have a *written* statement to work with.

That *written* statement must be precise and complete.

What's Wrong with Requirements Methods?

We think of requirements as a set of elements, each element being one requirement.

Consider three such requirements.

- The output must be an integer.
- The output must be positive.
- The output must not be zero.

Consider an alternative formulation:

- The output must be a natural number

These are equivalent - one requirement or three?

We cannot count requirements or list them?

If we try, we have no hope of checking for completeness, consistency, correctness.

There is a better way, based on the basic model used in control theory.

How to document system requirements?

The first step is to:

Identify monitored variables ($m_1, m_2, \bullet\bullet\bullet, m_n$).

Identify controlled variables ($c_1, c_2, \bullet\bullet\bullet, c_p$).

The primary monitored variables are things outside the system whose values should influence the output of the system. Examples:

- customer meter reading
- steam temperature
- time of day

The primary controlled variables are things outside the system whose values should be determined by the system. Examples:

- what the operator sees
- what appears on a bill
- control positions

This is only the beginning, but for many projects you cannot even find a complete list of these variables and there is no agreement on what they are.

Monitored and Controlled Variables Will Be Added During The Design Process.

It is inevitable that the need for additional variables will be discovered as we get into detailed work.

Further, new monitored and control variables are created during the design process.

The primary monitored and controlled variables are outside the system.

Sometimes we want to monitor the system itself, i.e. measure things that did not exist before the system was built.

Sometimes we may even want to control (adjust) parts of the system.

As the design is developed, we may add these monitored and controlled variables to the requirements document.

It is essential that the document be updated. Otherwise reviewers and maintainers are lost.

Bringing Time into the Picture

All of these variables can vary with time.

For each scalar variable, x, denote the time-function describing its value by "x^t".

The value of x at time t is denoted "$x^t(t)$".

The vector of time-functions ($v^t_1, v^t_2, ..., v^t_n$) will be denoted by "$\underline{v}^t$".

Contrary to the statements of some computer scientists, there is no problem dealing with "real" time.

Bringing Math Into our Tool Kit

The implementors need to know the following relations:

Relation NAT:

- domain contains values of $\underline{m}^t$, range contains values of $\underline{c}^t$.
- ($\underline{m}^t$, $\underline{c}^t$) is in NAT if and only if nature permits that behaviour.

This tell us what we need to know about the environment.

Relation REQ:

- domain contains values of $\underline{m}^t$, range contains values of $\underline{c}^t$.
- ($\underline{m}^t$, $\underline{c}^t$) is in REQ if and only if system should permit that behaviour.

This tells us how the new system is intended to further *restrict* what NAT(ure) allows to happen.

If we can describe these relations, we have our system requirements written down.

We can get the "scary" math out of the documents by using the right notation.

Why Use This Approach?

(1) For all the "motherhood" reasons that we try to find the requirements first.

(2) Because we can check for completeness.

(3) Because we can check for consistency.

(4) Because we have a precise description.

(5) Because we have a reviewable document.

(6) Because we can often simulate the system.

(7) Because the design can be based on the document.

(8) Because the programming goes much faster.

(9) Because the programmers work consistently and do not duplicate each other's work.

(10)Because we will discover ways to simplify the system.

(11)Because we can build monitors for testing or supervising the system.

Why not?

(12)Because it requires some training.

(13)Because it is a risky *front-end* investment that slows down the *initial* part of development.

How can we document system design?

$\underline{i}^t$ denotes the vector valued time function $(i^t_1, i^t_2, \bullet\bullet\bullet, i^t_r)$ with one element for each of the input registers

$\underline{o}^t$ denotes the vector valued time function $(o^t_1, o^t_2, \bullet\bullet\bullet, o^t_q)$ with one element for each of the output registers

<u>Document the following relations</u>

<u>Relation IN:</u>

- domain contains values of $\underline{m}^t$, range contains values of $\underline{i}^t$
- $(\underline{m}^t, \underline{i}^t)$ is in IN <u>if and only if</u> input device permits that behaviour

It must be the case that
domain(IN) $\supseteq$ domain(NAT)

<u>Relation OUT</u>

- domain contains the possible values of $\underline{o}^t$
- range contains the possible values of $\underline{c}^t$
- $(\underline{o}^t, \underline{c}^t)$ is in OUT if and only if output device permits that behaviour

When Can We Skip System Design?

Sometimes the I/O devices are simple and we can have simple relationships between the controlled and output variables as well as between the monitored and controlled variables.

In that case, we can use the systems requirements document as a software requirements document.

Many applications have this property.

In some, we can cheat and mix the two.

Dividing the Software to Conquer Complexity

Small modules are easier to understand, if <u>the interfaces to other modules are simple.</u>

To keep interfaces simple, "hide" the details inside the module.

Use the requirements documents to help structure the software:

- Some modules hide the requirements (REQ)
- some modules hide software decisions (which are not in the requirements document).
- Some modules hide the hardware (IN, OUT) These modules are support software.

These modules "create" virtual:

- data structures,
- devices,
- "actors",

"objects" that do part of the job.

It is at this stage that we have the best chances for re-use - but we must document the interfaces.

Documenting Module/Object Interfaces (1)

It is wise to design software by designing a set of objects.

- Each object is implemented by a module (a set of programs) using a data structure that is "hidden from" (never used directly by) programs outside the module.
- Changing the state of the object, or getting information about the object's state, is only done by invocations of programs from the module.
- An object is a finite state machine.
- The input alphabet of an object is the set of operations one can perform upon an object.
- The output alphabet of the object is the set of values that can be returned by such operations.

The state of an object can be hidden.

Describing or specifying objects is very different from describing or specifying programs.

Hiding the state means that we must discuss event sequences, but it makes future changes easier.

Documenting Module/Object Interfaces (2)

Black-box interface descriptions must be written in terms of (input, output) sequences (traces).

- A *trace* of a finite state machine is a finite sequence of pairs, each containing a member of the input alphabet and a member of the output alphabet.
- A trace, T, is considered *possible* for machine M, if M could react to the sequence of inputs in T by emitting the sequence of outputs in T.

Descriptions and specifications of objects can both be written as predicates on classes of traces.

These predicates are the characteristic predicate of an extension function/relation.

We organise our descriptions in terms of:

- A canonical *abstract* state representation, and
- single event extensions of those traces.

Result: a systematic, reviewable reference document

12 Element Queue

(1) SYNTAX

ACCESS PROGRAMS

Program Name	Value	Arg#1
ADD		<integer>
REMOVE		
FRONT	<integer>	

(2) CANONICAL Representation

$(rep = \{(a_i)\}_{i=1}^{n}) \wedge (0 \le n \le 12)$

(3) EQUIVALENCES

rep.ADD(a) ≡

condition	equivalent
length(rep) = 12	rep, %full%
length(rep) < 12	rep (a)

rep.REMOVE ≡

condition	equivalent
rep = _	rep, %empty%
rep ≠ _	$\{(a_i)\}_{i=2}^{n}$

rep.FRONT ≡

condition	equivalent
rep = _	rep, %empty%
rep ≠ _	rep

$val(Front) = a_1$

Design Reviews for Module Interfaces

Lots can be wrong with an "innocent" looking interface:

- The implicit assumptions can be wrong.
- The implicit assumption can be inconsistent.
- Interfaces can force inefficiencies on the system
- Interface assumptions can be likely to change
- Interface descriptions can be ambiguous

Interface decisions are early decisions.

Interface decisions affect more than one module.

Interface documents deserve serious thought!

They tend to be casually reviewed.

Effective Reviews are Active Reviews

A dilemma:

- Errors in interface documents should be found before the documents are used.
- Errors in interface documents are often found only when the documents are used.

Another dilemma:

- Everyone's work requires review
- It's easiest to say "OK"
- Reviewer's work is not reviewed.

One more dilemma:

- No individual knows enough to review all aspects of a design.
- When working in a group, people tend to relax in the knowledge that others are also working the problem.

Solutions:

- Make the reviewers use the documents.
- Make the reviewers answer questions.
- Have specialised review questionnaires. Ask the reviewer about things that they know.
- Make the reviewers provide specifics - not one bit.

Documenting Internal Design

We need to document:

(1) The complete data structure.

(2) The interpretation of that data structure (known as an abstraction function).

(3) The effect of each program.

Design Document for Queue12: Implementation 1 - Pascal

(1) DATA STRUCTURE

CONSTANTS

Constant Name	Definition
QSIZE	32

TYPES

Type Name	Definition
<qde>	array [0..QSIZE-1] of integer

VARIABLES

Type Definition/Name	Variables	Initial Values
<qde>	DATA	"Don't Care"
0..QSIZE-1	F, R	"Don't Care"
boolean	FULL	"Don't Care"

Abbreviations:

adjq ≝ (R = F + 1) ∨ (F = QSIZE-1) ∧ (R = 0)
'adjq ≝ ('R = 'F + 1) ∨ ('F = QSIZE-1) ∧ ('R = 0)
adjq' ≝ (R' = F' + 1) ∨ (F' = QSIZE-1) ∧ (R' = 0)
<qde> ≝ qde ∈ [0..QSIZE-1] × [0..QSIZE-1] × boolean

(2) ABSTRACTION FUNCTION

AF: <qde> → <queue12>

AF(DATA,F,R,FULL) ≝

(¬adjq ∨ FULL) ∧ (F ≥ R)	(DATA[F], DATA[F-1], ..., DATA[R])
(¬adjq ∨ FULL) ∧ (F < R)	(DATA[F], ..., DATA[0], DATA[QSIZE-1], ..., DATA[R])
adjq ∧ ¬FULL	EMPTY

Describing Programs

A program is a part of a module.

We wish to describe its effect on the module's private data structure.

We distinguish 3 types of descriptions:

- *constructive descriptions*, which show how a product is constructed from other products,
- *behavioural descriptions*, which describe the visible behaviour of a product without discussing how it was constructed, and
- *specifications*, which describe the requirements that a product must meet.

In my view this is a very important distinction that is ignored by the "formal methods" community.

Relational Program Descriptions and Specifications

Users need to know the relation between the starting values of variables and the final values of variables.

Users need to know the starting states for which the program is guaranteed to terminate.

We base our work on Harlan Mills' ("Cleanroom") program function, but

- Represent the function in a more readable tabular format.
- Deal properly with non-determinism.
- Carefully distinguish between relations as specifications and relations as descriptions.

It is possible to produce short, readable specifications of programs and review them before writing the actual code.

This forces designers to think about issues that they tend to overlook (such as error response).

(3) PROGRAM FUNCTIONS

p_Name	Arg#1	Value	
p_COUNT		[illegible]	→ [illegible]
p_ADD	[illegible]	[illegible]	→ [illegible]
p_REMOVE		[illegible]	→ [illegible]
p_FRONT		[illegible]	→ [illegible]

p_ADD(a) ≝ [illegible]

	[illegible]			[illegible]		
	[illegible]		¬[illegible]	[illegible]		¬[illegible]
	FULL	¬FULL		FULL	¬FULL	
DATA'[X'] =	DATA[X]	a	[illegible]	DATA[X]	a	a
X' =	X	[illegible]	[illegible]	X	X − 1	X − 1
FULL' =	FULL	false	[illegible]	FULL	false	[illegible]

p_REMOVE ≝ NC(DATA) ∧

	[illegible]		[illegible]
	[illegible]	[illegible]	
F' =	[illegible]	[illegible]	[illegible]
FULL' =	false	false	FULL

p_FRONT ≝ NC(FULL, DATA, F) ∧

	¬[illegible] ∨ FULL	[illegible]
[illegible] =	DATA[F]	[illegible]

The "Laws" of Programs

Do Software Engineers have laws for programs that correspond to Kirchoff's laws for circuits?

Yes!

The basic laws of programs are essentially the axioms of the algebra of (LD-)relations.

If you accept the fact that LD-relations provide adequate descriptions of program behaviour, sequential execution is composition.

The laws are the classic results about relations.

These laws allow you to find behavioural descriptions of constructed programs if given:

- the constructive description of those programs and,
- the behavioural descriptions of the primitive programs.

With these laws, all reasonable specifications and descriptions are compositional. Composition is not Conjunction.

Imperfection of Documents?

When engineers work with physical products they must use imperfect implementations of abstract specifications.

With software, imperfection is not always necessary but it may be convenient and acceptable.

The imperfections must be "bounded" and explicitly limited in their applicability.

For example, we may ignore the limits on representations of numbers because we only work with a limited range of numbers.

It is important to include this in the specification.

No new mathematics is needed for this. Implication does the job.

The use of mathematics in engineering does not imply a belief in perfection of programs or maths.

What New Notation *do* we Need?

Although the mathematics is old, and the abstract notation for defining things is old, the applications are new.

We have to describe relations and functions that have non-heterogeneous ranges and domains and can have a discontinuity at arbitrary points.

We have found a variety of tabular notations to be useful.

Ryszard Janicki, has found new ways to unite these tabular notations.

Jeff Zucker and our students are implementing tools for transformations.

We are trying to:

- Make the documentation easier to produce
- Make the documentation more useful

A Simple Conventional Expression

$$(((\exists\ i,\ B[i] = x) \wedge (B[j'] = x) \wedge (present' = true)) \vee ((\forall\ i,\ ((1 \leq i \leq N) \Rightarrow B[i] \neq x)) \wedge (present' = false))) \wedge ('x = x' \wedge 'B = B')$$

A tabular expression:

Specification for a search program

	$(\exists\ i,\ B[i] = x)$	$(\forall\ i,\ ((1 \leq i \leq N) \Rightarrow B[i] \neq x))$
j' |	$B[j'] = x$	*true*
present' =	true	false

$\wedge$ NC(x, B)

The above is one of many kinds of tables!

Simple tables like this *understate* the advantage.

These have "practitioner appeal".

Inspecting Programs

Its the code that "hits the road."

Getting the requirements right, the structure right, the interfaces right, etc. are all important but we have to check the code.

The same review principles apply.

- Make the reviewers use the documents.
- Make the reviewers answer questions.
- Have specialised review questionnaires. Ask the reviewer about things that they know.
- Make the reviewers provide specifics - not one bit.

We want to compare the completed programs with previously reviewed specifications.

We ask the reviewers to produce descriptions.

We then show that the descriptions match the specifications.

It's hard work but it produces results.

- We get good documentation for future use
- We find errors in the best industrial code - programs that were considered correct.

Is it Teachable/Learnable/Practical?

Its the way to start - first year engineering students have learned to read and implement from specs.

Tabular notation - no theoretical advantage, but a great practical advantage.

Short courses introduced these ideas to the nuclear industry in Canada. They now teach their own.

People can apply the inspection technique after a 3 - 4 day course.

Critical Mass in a company is essential. Writers without readers are useless.

There is lots of room for improvement. We will identify these faster if you work with us.

Sets for Describing Programs

Everything about digital computers can be explained in terms of finite sets; the set concept is viewed by many as the most basic concept in mathematics.

A set is a collection of elements from a previously defined set (sometimes called the universe).

The elements in the universe must be known before other sets are defined. Every application of set theory must begin with a careful description of the Universe from which it's elements are drawn.

Sets drawn from different universes cannot be compared.

Set elements are assumed to have previously defined attributes.

The famous anomalies can be avoided.

Call it dull set theory.

Notation for sets

{x,y,z}	enumeration -a set containing x,y, z
|	such that

{x | <condition>}The set of elements such that x satisfies the condition.

$A \subseteq B$	A is a subset of B (could be identical)
$A \subset B$	A is a subset of B and smaller than B
$A \cup B$	set of elements in either A or B
$A \cap B$	set of elements in both A and B
$A - B$	set of elements of A that are not in B
$-(B)$	set of elements in Universe not in B (the complement of B)
$X \in A$	X is an element of A
{ }	an empty set

Only combine sets from the same Universe.

Even empty sets must have an associated Universe.

Relations

<u>What is a relation (e.g. ≥, ≤, =)?</u>

A set of ordered pairs.

<u>What is the domain of a relation?</u>

The set of elements that appear as the first element of a pair in the relation.

<u>What is the range of a relation?</u>

The set of elements that appear as the second element of a pair in the relation.

One need not enumerate all the pairs to describe a relation!

If R is a relation and $(x,y) \in R$, we can write
$x \, R \, y$.

Examples of relations

Both elements taken from the set of real numbers.

(1) $A = \{(x,y) \mid x > y\}$

(2) $B = \{(x,y) \mid x = y\}$

(3) $C = A \cup B$

(4) $D = \{(x,y) \mid x \times x = 4\}$

(5) $E = \{(x,y) \mid x \div y = 4\}$

What is a function?

A *function* is a relation, F, such that if (x,y) is in F, and (u,v) is in F, and x = u, then y = v

If F is a function, and (x,y) ∈ F, we can write
y = F(x).

F(x) would not generally denote a single value if F were a relation that was not a function.

Since all functions are relations we can also write x F y.

In many applications it is important to make sure that a relation is a function. It assures us that a description is unambiguous.

A *partial function* is a function whose domain is smaller than the stated universe.

Examples of functions

Both elements taken from the universe of real numbers.

A = {(x,y) | y = x + 1} - written A(x) = x + 1

B = {(x,y) | x = y} - written B(x) = x

C = { (x,y) | y × y = x and y ≥ 0}
- written C(x) = + $\sqrt{X}$

What is a Predicate?

A function whose range is a subset of
{*true*, *false*}

Predicates are often described by predicate expressions.

Examples:

x > 0 characterises

{. . . (-1,*false*), (0,*false*), (1,*true*), (2,*true*) . . .}

X = [illegible]^{2} describes
{. . . ([illegible],*false*),(1,*true*), (0,*true*), (-1,*false*)...}

$(X = X^2) \wedge (X > 0)$ describes
{. . . (2,*false*),(1,*true*), (0,*false*), (-1,*false*)...}

Characteristic Predicates

Every set has a *characteristic predicate.*

The domain of that predicate is the universe from which the set is drawn.

(x, *true*) is in the predicate if x is in the set being characterised.

Predicate expressions can describe, sets, functions, relations in this way provided that the universe is clearly specified.

true characterises the universe, U

false characterises the empty set, {}

Predicate expressions are described more completely later.

Characteristic Predicates Describing Relations

$\{(x,y) \mid x < y\}$ described by $x< y$

$\{(\text{‘}x,x') \mid x'=\text{‘}x + 1\}$ described by $x' = \text{‘}x + 1$

The use of predicate expressions in this way requires clearly stated conventions about the universe and the naming of the elements of an ordered pair.

‘x can be read “x before” or “x left”.

x’ can be read “x after” or “x right”.

A predicate expression is not a predicate.

A predicate expression is not a set.

A predicate expression is not a function or relation. Predicate expressions can describe:

- predicates
- sets
- functions
- relations

Summary

- A *relation* is a set of pairs (2-tuples).
- The set of values that appear as the first element of a pair is called the *domain* of that relation.
- The set of values that appear as the second element of a pair is called the *range* of that relation.
- A *function* is a relation such that for any given element, x, in its domain, there is only one pair (x,y) in the function.
- If (a,b) is in the function F, “F(a)” means b, often called “the *value of F at a*”. may include tuples.
- It may make sense to write “F((a,b))”, “F((a,b,c))”, and “F(F((a,b,c)))”.
- Functions whose domain is smaller than the universe are called *partial functions*
- Most of the functions that arise in software development will be partial functions.
- A *predicate* is a function whose range contains no members other than *true* and *false*.
- For any set, X, the *characteristic predicate* of X is a predicate whose domain is the universe from which X is drawn, and whose value, for b, is *true* if and only if b is a member of X.

Definition of Predicate Expressions

Built-in functions and predicates are named:

To simplify the presentation we shall assume that all functions and relations have simple names.

$f_1, \ldots, f_k$ are the names of functions (sets)

$R_1, \ldots, R_m$ are the names of the characteristic predicates of relations.

Definition of Predicate Expressions

Terms are constructed from:

A finite set of mathematical variables, $x_1, \ldots, x_n$

A finite set of constants, C

The constants are strings. Each constant represents one member of the universe, U.

“V” stands for a comma separated list of terms (see below).

A *function application* is a string of the form $f_i(V)$.

A *term* is either a constant, a variable, or a function application.

Definition of Predicate Expressions

A *primitive expression* is a string of the form $R_j(V)$.

Nothing else is a primitive expression.

All of our expressions will be built of primitive expressions.

Note that primitive expressions, since they denote predicates, will always evaluate to either *true* or *false*.

Predicate Expressions

All primitive expressions are *predicate expressions.*

If P and Q are predicate expressions and x_k is a variable, then

$(\forall x_k, P)$,

$(\exists x_k, P)$,

$(P) \wedge (Q)$,

$(P) \vee (Q)$,

$(P) \Rightarrow (Q)$,

$\neg(P)$

are also *predicate expressions.*

The previous definitions tell us what we can write, i.e. which expressions are predicate expressions; they do not tell us what these expressions mean.

The Meaning of Predicate Expressions

Evaluating terms:

An *assignment*, a, is a list specifying values for all the variables. We evaluate expressions for a specific assignment.

(1) If t is a constant representing t' (a member of D), the value of the term t for assignment a, (written "val(t,a)"), is t',

(2) t is a variable, x_k, the val(t,a) is the value specified for that variable in a.

(3) if t is a function application, $f_k(V)$, we must evaluate each of the terms in V until we have obtained the values that they represent.

(4) V" denotes the result of this evaluation

We distinguish the following three cases:

(3a)if V' is in the domain of f_k, val(t,a) is $f_k(V')$.

(3b)if V' is not in the domain of f_k, val(t,a) is *not defined.*

(3c)if any of the elements of V' is *not defined*, the value of the function application is *not defined.*

The Meaning of Predicate Expressions

Evaluating primitive expressions:

For a primitive expression, R(V), we first evaluate all the terms in V to get V', and distinguish the following three cases:

(a) If V' is in R_* the value is *true.*

(b) If V' is not in R_* the value is *false.*

(c) If any element V' is *not defined*, the value is *false.*

Evaluating Predicate Expressions

If P and Q are predicate expressions.

(a) $(\forall x_k, P)$ is *true* if P is true for all values of x_k in our Universe. Otherwise, it is *false*.

(b) $(\exists x_k, P)$ is *true* if P is true if there is a value of x_k in our Universe for which P is *true*. Otherwise, it is *false*.

(c) $(P) \wedge (Q)$ is *true* if both P and Q are *true*. Otherwise, it is *false*.

(d) $(P) \vee (Q)$ is *true* if either P or Q are *true*. Otherwise, it is *false*.

(e) $\neg(P)$ is *true* if P is *false*. Otherwise, it is *false*.

(f) $(P) \Rightarrow (Q)$ is *true* if either P is *false* or Q is *true*. Otherwise, it is *false*.

The symbols are read, "for all", "there exists", "and", "or", "not", and "implies".

Identities for Predicate Expressions

If P and Q are predicate expressions.

(a) $\neg(\forall x_k, P) = (\exists x_k, \neg(P))$

(b) $\neg(\exists x_k, P) = (\forall x_k, \neg(P))$

(c) $\neg((P) \wedge (Q)) = (\neg(P)) \vee (\neg(Q))$

(d) $\neg((P) \vee (Q)) = (\neg(P)) \wedge (\neg(Q))$

(e) $(\neg(P)) \vee (Q) = (P) \Rightarrow (Q)$

Parentheses can sometimes be omitted if you remember that "¬" is stronger than "∧" is stronger than "∨"which is stronger than "⇒".

For example, we can write
"a∧¬ b" instead of "(a)∧(¬ (b)),
and "¬ b∧a" instead of "(¬ (b)∧(a))"

Examples of Predicate Expressions

$((x>0) \wedge (y = \sqrt{x})) \vee ((x \le 0) \wedge (y = \sqrt{-x}))$ (1)

$((x>0) \Rightarrow (y = \sqrt{x})) \wedge ((x \le 0) \Rightarrow (y = \sqrt{-x}))$ (2)

$((y = \sqrt{x}) \vee (y = \sqrt{-x}))$ (3)

$(\exists i, ((1 \le i \le n) \wedge (\text{'}A[i] = \text{'}x)))$ (4)

$(\exists i, ((1 \le i \le n) \Rightarrow (\text{'}A[i] = \text{'}x)))$ (5)

$((1 \le n) \wedge (\forall i, ((1 \le i < n) \Rightarrow (\text{'}A[i] \le \text{'}A[i+1]))))$ (6)

Exercise

Try to write English statements corresponding to the above.

Software Inspections We Can Trust

David Lorge Parnas

NSERC/Bell Industrial Research Chair in Software Engineering
Communications Research Laboratory
Department of Electrical and Computer Engineering
McMaster University Hamilton, Ontario Canada L8S 4K1

Software is devilishly hard to inspect. Serious errors can hide for years. Consequently, many are hesitant to employ software in safety-critical applications and all companies are finding correcting and improving software to be an increasingly burdensome cost.

This talk describes a procedure for inspecting software that consistently finds subtle errors in software, software that is believed to be correct. The procedure is based on four key ideas:

- All software reviewers are actively using the code.
- Reviewers exploit the hierarchical structure of the code rather than proceeding sequentially through the code.
- Reviewers focus on small sections of code, producing precise summaries that are used when inspecting other such sections.
- Reviewers proceed systematically so that no case, and no section of the program, gets overlooked.

During the procedure, the inspectors produce and review mathematical documentation. The mathematics allows them to check for complete coverage; the notation allows the work to proceed in small systematic steps.

Responsibilities of (Software) Engineers

- To thoroughly understand the properties of their products.
- To follow established rules of good practice when designing and building products.
- To apply accepted theory where it has been shown to lead to better, safer products.

Engineering is Not Management

The art of management is the ability to get things built without knowing exactly what they are.

The engineer is expected to thoroughly understand the properties of the product.

Software projects are hard to manage - especially if they are badly designed, but ...

Unless we have good Engineers, the best managers will not be able to successfully manage these products.

Why Is Software so often a Problem?

Developers *consistently* underestimate the difficulty of building software for long-term use.

They *write* software rather than *design* it.

They do not:

- systematically, identify and record requirements,
- hold reviews of the requirements document,
- explicitly design, document and review software structure,
- carefully inspect all designs and programs.

These steps are standard practice for all engineering products other than software.

The steps are not taken for software because,

- "Software is easy!"
- "The code is self-documenting!"
- "Software is *just* a set of instructions."
- "Anyone who knows the language can program."

Famous last words!

Why Don't People Apply Engineering Discipline to Software?

(1) Some don't have an engineering education.

(2) Some don't think it's necessary.

(3) Some don't know how to do it.

Why don't we demand that software people have appropriate qualifications?

Experience shows that it is necessary.

Why aren't software designers required to be Engineers?

Why do we continue to think of them as scientists and to educate them accordingly?

Why Don't Engineers Apply Mathematics, and "Theory" to Software Products?

The last 30 years have seen great advances in our understanding of software science.

Programs written by most engineers have not taken advantage of this theory.

Programs written by most other programmers do not reflect this theory.

- Many don't know the theory.
- Those who know it don't know how to apply it
- Much of it is difficult to apply, perhaps even not applicable.
  - Deals with impractical languages
  - Deals with unbounded memory size
  - Uses unnecessarily difficult notations
  - Designed for the wrong purpose

There is a need to connect theory to practice.

Let's start with software inspections.

When is Software Critical?

Critical is not necessarily "safety critical"

Other types of critical programs:

- Mass distributed programs in warranty situations
- Critical kernels in many systems
- Financial Systems
- Security (Privacy, Data Protection) programs

The common property of all of these examples is that the cost of a failure is high.

If you value your reputation, your work may be critical.

The Critical-Software Tripod

(1) Precise, well organised, mathematical documentation with systematic review

(2) Extensive Testing

- Systematic Testing-quick discovery of gross errors
- Random Testing -discovery of shared oversights and reliability assessment

(3) Qualified People and Approved Processes

The Three Legs are complementary

The three legs are all needed.

The stool falls over if any leg is forgotten.

The third leg is the shortest.

It's the shortest leg that we should worry about.

Today we discuss only leg (1).

Why Conventional Reviews are Ineffective

(1) The reviewers are swamped with information.

(2) Most reviewers are not familiar with the product design goals.

(3) There are no clear individual responsibilities.

(4) Reviewers can avoid potential embarrassment by saying nothing.

(5) The review is conducted as a large meeting where detailed discussions are difficult.

(6) Presence of managers silences criticism.

(7) Presence of uninformed reviewers may turn the review into a tutorial.

(8) Specialists are asked general questions.

(9) Generalists are expected to know specifics.

(10) The review procedure reviews code without respect to structure. (n lines per hour)

(11) Unstated assumptions are not questioned.

Effective Reviews are Active Reviews

A dilemma:

- Errors in programs and design documents should be found *before* the documents/systems are used.
- Errors in programs and documents are usually found *when* the documents are used.

Another dilemma:

- Everyone's work requires review!
- It's easier to say "OK" than to find subtle errors!
- Reviewer's approval is not reviewed.

One more dilemma:

- No individual can review all aspects of a design.
- When working in a group, people tend to relax in the knowledge that others are also working the problem.

Solutions:

- Make the reviewers use the documents.
- Make the reviewers document their analysis.
- Have specialised reviews. Ask the reviewer about things that they know.
- Make the reviewers provide specifics - not just a bit.

Previous Work on Inspections

Best known approach Fagan - 1976.

Many followers - new book by Gilb.

Explicitly focus on the management aspects.

- Who should be there?
- What are the roles of the participants?
- How long is a meeting?
- How fast do you work?
- Forms for reporting errors?

Read the code in sequence and paraphrase.

Paraphrases are informal.

Most observers find these more effective than conventional reviews or walkthroughs, but ...

... can we do better?

Parnas/NRL/AECB/AECL/Ontario Hydro

Focus on the engineering side.

Depend on hierarchical decomposition rather than sequential reading.

Use mathematical notations to provide precise descriptions rather than informal paraphrases.

Produce useful documentation as a side effect.

Proceed much more quickly if the documentation was already produced by the developers.

Insures that cases and variables are not overlooked.

Applies simple mathematics to check for completeness aspects.

Reviewing Design Documents

Base the review process on the nature of the document.

Begin by identifying desired properties.

Prepare questionnaires for the reviewers. Ask them questions that:

- make them use the document.
- make them demonstrate that the desired properties are present.
- ask for sources of information to support the answers to other questions.

For example:

- Ask reviewers to identify the domain of the program
- Ask reviewers to identify "error" cases.
- Ask reviewers to explain why the behaviour required for each case is the desired behaviour.

For more information read [1].

Inspecting Programs

It is the code that "hits the road".

Getting the requirements right, the structure right, the interfaces right, the documentation right, etc. are all important but *we have to check the code.*

The same review principles apply, viz:

- Make the reviewers use the material they review.
- Make the reviewers answer questions.
- Ask the reviewer about things that they know.
- Make the reviewers provide specifics.

We compare completed programs with previously reviewed specifications.

We ask the reviewers to produce precise descriptions.

We then show that the descriptions match the specifications.

It is hard work but it produces results.

- We get good documentation for future use.
- We find errors in the best industrial code - programs that were considered correct.

Our Code Inspection Process

(1) Prepare a precise specification of what the cod should do - a program function table.

(2) Decompose the program into small parts appropriate for the "display approach" [2].

(3) Produce specifications as required for the display approach.

(4) Compare the "top level" display description with the requirement specification.

Observations:

- You can't inspect without precise requirements.
- Step 2 would already have been done if you use the display method for documentation.
- Step 3 is truly an active design review
- All reviewer work is itself reviewable.
- If you did not already have it, the by-product is thorough documentation.
- It's a bunch of small steps and very systematic.

Descriptions vs. Specifications

An actual description is a statement of some actual attributes of a product, or set of products. □

A specification is a statement of all properties required of a product, or a set of products. □

In the sequel, "description", without modifier, means "actual description".

The following are implications of these definitions:

- A description may include attributes that are not required.
- A specification may include attributes that a (faulty) product does not possess.
- The statement that a product satisfies a given specification may constitute a description.

The third fact results in much confusion. A useful distinction has been lost.

Descriptions vs. Specifications

Any list of attributes may be interpreted as either a description or a specification.

Example:

"A volume of more than 1 cubic meter"

This could be either an observation about a specific box or, a statement of the requirements for a box that is about to be purchased.

A specification may offer a choice of attributes; a description describes the actual attributes, but need not describe the product completely.

Sometimes one may use one's knowledge of the world to guess whether a statement is a description or a specification.

Example:

"Milk, badly spoiled"

Guessing is not reliable. We need to label specifications and descriptions.

Do We Need New Semantics Theories For Programming?

Not for the practical software engineering problems that I see.

I can find 30 year old theory that works for the problems that I will describe today.

Semantic theory has failed to describe real languages, but (in my opinion) the fault lies with the languages.

We do need improvements in:

- the notation used to describe actual programs
- the ability to describe behaviour in terms of the values of observable variables - nothing else.
- convenient ways to deal with all aspects of termination including non-deterministic non-termination.

What follows is mathematically equivalent to some very old ideas, but has some small practical advantages.

A Mathematical Interlude - LD-relations.

A *binary relation* R on a given set U is a set of ordered pairs with both elements from U, i.e. $R \subseteq U \times U$.

The set U is called the *Universe of R*.

The set of pairs R can be described by its *characteristic predicate*, R(p,q). i.e. $R = \{(p,q): U \times U \mid R(p,q)\}$.

The *domain* of R is denoted Dom(R) and is $\{p \mid \exists q [R(p,q)]\}$.

The *range* of R is denoted Range(R) and is $\{q \mid \exists p [R(p,q)]\}$.

Below, "relation" means "binary relation".

A *limited-domain relation* (LD-relation) on a set, U, is a pair, $L = (R_L, C_L)$ where:
R_L, the *relational component* of L, is a relation on U, i.e. $R_L \subseteq U \times U$, and
C_L, the *competence set* of L, is a subset of the domain of R_L, i.e. $C_L \subseteq \mathrm{Dom}(R_L)$.

Using LD-Relations as Before/After Behavioural Descriptions (1)

Let P be a program, let S be a set of states, and let $L_P = (R_P, C_P)$ be an LD-relation on S such that
$(x,y) \in R_P$ if and only if $<x,\ldots,y>$ is a possible terminating execution of P, and
$x \in C_P$ if and only if P is guaranteed to terminate if it is started in state s. [1]
L_P is called the *LD-relation of* P

By convention, if C_P is not given, it is, (by default), $\mathrm{Dom}(R_P)$.

With this convention, our approach is upwards compatible with the "cleanroom" approach for dealing with deterministic programs.

[1] Please note that C_P is not the same as the precondition used in VDM [4]. S_P is the set of states in which the termination of P is certain.

Using LD-Relations as Before/After Behavioural Descriptions (2)

The following follow from the definitions:

- If P starts in x and $x \in C_P$, P always terminates; if $(x, y) \in R_P$, P may terminate in y.
- If P starts in x, and $x \in (\mathrm{Dom}(R_P) - C_P)$, the termination of P is non-deterministic; in this case, if $(x, y) \in R_P$, when P is started in x, it may terminate in y or may not terminate.
- If P starts in x, and $x \notin \mathrm{Dom}(R_P)$, then P will never terminate.

By these conventions we are able to provide complete before/after descriptions of any program but retain a simpler representation to use for those cases that arise most often.

Specifying Programs (1)

Specifications may allow behaviour not actually exhibited by a satisfactory program.

We can also use LD-relations as before/after specifications:

Let $L_p = (R_P, C_P)$ be the description of program P.
Let S, called a *specification*, be a set of
LD-relations on the same universe and
$L_S = (R_S, C_S)$ be an element of S.
We say that

(1) P *satisfies an LD-relation* L_S, if and only if
$C_S \subseteq C_P$ and $R_P \subseteq R_S$, and

(2) P *satisfies a specification*, S, if and only if
L_p satisfies at least one element of S.

Often, S has only one element. If $S = \{L_S\}$ is a specification, then we can also call L_S a specification.

Specifying Programs (2)

The following follow from the definitions:

- A program will satisfy it's own description as well as infinitely many other LD-relations.
- An acceptable program must NOT terminate when started in states outside $Dom(R_S)$.
- An acceptable program must terminate when started in states in C_S ($C_S \subseteq Dom(R_P)$).
- An acceptable program may only terminate in states that are in $Range(R_S)$.
- A deterministic program can satisfy a specification that would also be satisfied by a non-deterministic program.

Note the following differences between the description and the specification of a program.

- There is only one LD-relation describing a program, but that program will satisfy many distinct specifications described by different LD-relations.
- An acceptable program need not exhibit all of the behaviours allowed by R_S ($R_P \subseteq R_S$).
- An acceptable program may be certain to terminate in states outside C_S. ($C_S \subseteq C_P$).

The intended use of each LD-relation (specification or description) must be stated explicitly!

Tabular Descriptions and Specifications

Specification for a search program

	$(\exists i, B[i] = x)$	$(\forall i, ((1 \le i \le N) \Rightarrow B[i] \ne x))$
j' |	$B[j'] = x$	*true*
present' =	true	false

$\wedge$ NC(x, B)

Description of a search program

	$(\exists i, B[i] = x)$	$(\forall i, ((1 \le i \le N) \Rightarrow B[i] \ne x))$
j' |	$(B[j'] = x) \wedge (\forall i, ((j' < i \le N) \Rightarrow B[i] \ne x))$	*true*
present' =	true	false

$\wedge$ NC(x, B)

The above is one of many kinds of tables!

Simple tables like this *understate* the advantage.

These have proven "practitioner appeal"

A Simple Example

```
(integer array H[1:N];

 (integer c; integer n; n ⇐ 1;
it (n ≤ N →
   (
(integer u; integer l; boolean p; l ⇐ 1; c ⇐ 0;
it (u ⇐ l + n -1;
(u ≤ N → (


(integer i; i ⇐ 0; p ⇐ true;
it (i < ⌊(u - l + 1)÷2⌋ →
      (A[l+i] = A[u-i] → (i ⇐ i + 1; ↵)
      | A[l+i] ≠ A[u-i] → (p ⇐ false; ● ))
 | ⌊(u - l + 1)÷2⌋ ≤ i → ● )
ti)
;
    (¬p → skip | p → c ⇐ c+1); l ⇐ l+1; ↵ )
| u > N → ● ))
ti)
;
     H[n] ⇐ c; n ⇐ n+1; ↵)
  | n > N → ● )
ti)
)
```

Decomposition

```
( integer array H[1:N];
 (integer c; integer n; n ⇐ 1;
 it (n ≤ N →
    (
 (integer u; integer l; boolean p; l ⇐ 1; c ⇐ 0;
 it (u ⇐ l + n -1;
 (u ≤ N → (

 (integer i; i ⇐ 0; p ⇐ true;
 it (i < ⌊(n - l + 1)÷2⌋ →
       (A[l+i] = A[n-i] → (i ⇐ i + 1; ☛)
       | A[l+i] ≠ A[n-i] → (p ⇐ false; ●))
 | ⌊(n - l + 1)÷2⌋ ≤ i → ● )
 ti)
 ;
     (¬p → skip | p → c ⇐ c+1); l ⇐ l+1; ☛ )
 | u > N → ● ))
 ti)
 ;
       H[n] ⇐ c; n ⇐ n+1; ☛)
   | n > N → ● )
 ti)
)
```

Display: An Example

Problem: ctpal ≡

	true	H₁
c' = H₂	card^a({l | pal(A,l,n + l - 1) })	G

∧ NC(n,A)

a. card(x), where x is a set, is the number of elements in x.

Solution: ctpal ≡

```
( integer u, l; boolean p; l ⇐ 1; c ⇐ 0;
it ( u ⇐ l + n -1;
(u ≤ N → (palul; (¬p → skip | p →  c ⇐ c+1);
          l ⇐ l+1; ☛ )
| u > N → ● ))
ti )
```

palul ≡ NC(l,u,A) ∧ (p' = pal(A,l,u)

where

pal(A,b,c) ≡ ((1 ≤ b ≤ c ≤ N) ∧
(∀ i, 0 ≤ i < ⌊(c- b +1)÷2⌋ ⇒ A[b+i]=A[c-i]))

Displays: An Explanation

The top part of each display is the specification for the program in the middle.

The program in the middle is kept small by removing sections, creating a display for them, and including their specification in the bottom part.

The bottom part contains a specification of these invoked programs.

To check a display determine the description of the program in the middle, and see if it satisfies the specification at the top. In doing this, use the specifications of the invoked programs, not their text.

To check a set of displays, make sure that every specification at the bottom of one display is at the top of another. The exceptions:

- standard programs
- primitive programs

Completeness can be checked mechanically.

Structure and Inspection

Well-structured programs are easier to decompose. They can be decomposed by purely syntactic means.

Well-structured programs are much easier to inspect.

Inspection encourages good structuring.

Inspection suggests structural improvements.

Inspected programs are easier to maintain.

Modified programs need not be completely re-inspected.

The cost of future maintenance is greatly reduced.

The definition of "well-structured" should not be based on the absence or presence of certain control structures. It has to do with the ease of decomposition. [2]

Our Initial Experience: Darlington Nuclear Power Generating Station[1]

Three control systems in Canadian reactors:

- one normal control system
- two independent shutdown systems

Safety analysis *assumes* control system will fail. Only shutdown systems are considered safety-critical.

Previous shutdown systems were analogue and relay systems.

At Darlington they are software controlled.

Each Software System has a simple task.

Their designs are "diverse".

The systems are more complex than their predecessors with the result that AECB[2] could not be confident of their trustworthiness.

How can we increase that level of confidence?

[1] Discussed in more detail in [4] and [3].
[2] Atomic Energy Control Board of Canada

Why We Could Not Use English

The following type of sentence was found in the requirements document.

> *"Shut off the pumps if the water level is above 100 meters for 4 seconds"*

What does this simple sentence mean?

Three Reasonable Interpretations:

"Shut off the pumps if the mean water level over the past 4 seconds was above 100 meters".

$$[(\int_{T-4}^{T} WL(t)dt) \div 4 > 100]$$

"Shut off the pumps if the median water level over the past 4 seconds was above 100 meters".

$$(MAX_{[t-4,t]} (WL(t)) + MIN_{[t-4,t]} (WL(t))) \div 2 > 100$$

"Shut off the pumps if the "rms" water level over the past 4 seconds was above 100 meters".

$$\sqrt{(\int_{T-4}^{T} WL^2(t)dt) \div 4)} > 100$$

A Fourth (Unreasonable) Interpretation:

"Shut off pumps if the minimum water level over the past 4 seconds was above 100 meters".

$$MIN_{[T-4,T]} [WL(t)] > 100$$

This is the most literal interpretation!

It is a disaster waiting to happen!

If you use natural languages, there are thousands of such phrases waiting to "bug" you.

The Inspection Process at Darlington

Four teams:

(1) Application Experts

(2) Programming Experts

(3) Verifiers

(4) Auditors

Roles of the teams:

(1) Produces requirements tables.

(2) Produce Program Function Tables (Displays).

(3) Show (1) = (2) and that (2) are correct.

(4) Audit the "proofs".

Subsequent Experience

In classes on this method, we have applied this to numerous small industrial programs that were believed to be correct.

In most cases, we found unexpected errors.

In some cases, the participants could not state the requirements.

In other cases, the program could not be decomposed (machine code w/o documentation).

I believe that one program was correct.

In all cases, we could improve the program.

We have found errors in textbook programs, library programs, and well-used and tested programs.

No process is perfect, but this one engenders confidence. It produces code that people trust.

Essential Point: Divide and Conquer

The initial decomposition is essential. Attempts to simply scrutinise the program fail.

Trying to read the program the way a computer would is much less effective. Logically connected parts may be far apart.

The use of tables is essential. It breaks things down into simple cases so that

- We can be sure that all cases are covered
- Each case is straightforward

We consider all variables, but one at a time.

We consider all cases, one at a time.

We can take "breaks", go home and sleep, even take holidays, without losing our place.

Using displays and tabular summaries is far more work than Fagan's English paraphrasing, but it imposes a discipline that helps.

The Other Essential Point: Precise, Abstract Descriptions

Having lots of little parts is not enough.

We have to be sure that the parts fit together.

We have to be able to do that without page-flipping.

Each part's behaviour must be precisely summarised without giving intermediate states.

We must be sure that the description at the bottom of one display will be identical with that at the top of another display.

These global checks can, and have been, mechanised.

Precise descriptions are painstaking work, but if quality is important, they are essential.

It's not always easy!

The most critical step, besides decomposition, is finding a good representation for the state space.

A 1:1 relation between names and elements of the data structure cannot be assumed.

When preparing the displays, the creative step is data state representation.

Some Suggested Reading

(1) Parnas, D. L., Weiss, D. M., "Active Desig Reviews: Principles and Practices", *Proceedings of the 8th International Conference on Software Engineering*, London, August 1985.
Also in *Journal of Systems and Software*, December 1987.

(2) Parnas, D. L., Madey, J., Iglewski, M., "Precise Documentation of Well-Structured Programs", *IEEE Transactions on Software Engineering*, Vol. 20, No. 12, December 1994, pp. 948 - 976.

(3) Parnas, D. L. "Inspection of Safety Critical Software using Function Tables", Proceedings of IFIP World Congress 1994, Volume III, August 1994, pp. 270 - 277.

(4) Parnas, D. L., Asmis, G.J.K., Madey, J., "Assessment of Safety-Critical Software in Nuclear Power Plants", *Nuclear Safety*, vol. 32, no. 2, April-June 1991, pp. 189-198.

The Problem of the Dutch national flag[1]

There is a data type color $\overset{\mathrm{df}}{=}$ {blue,red,white}
There is an abstract data type “buckets”.

Variables of this type may be used as a vector of N “pebbles” of “color” type, where $N \geq 0$ is an integer.

The only operations on v are: PUT(i,c), LOOK(i), SWAP(i,j)

Design a procedure to rearrange (if necessary) the pebbles in the order of the Dutch national flag using no Arrays, and calling LOOK(i) once for each value of i.

[1] Introduced and (perhaps) solved by E. W. Dijkstra in 1976

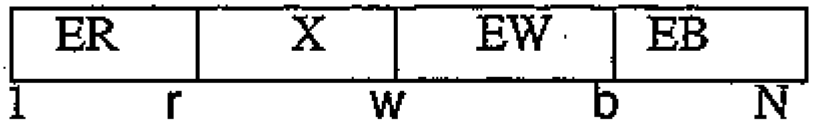
$1 \leq k < r$: the k^{th} bucket is in zone ER (number of buckets $r-1 \geq 0$)

$r \leq k \leq w$: the k^{th} bucket is in zone X (number of buckets $w-r+1 \geq 0$)

$w < k \leq b$: the k^{th} bucket is in zone EW (number of buckets $b-w \geq 0$)

$b < k \leq N$: the k^{th} bucket is in zone EB (number of buckets $N-b \geq 0$)

This can be illustrated by the following figure.



Initially, r=1, and w=b=N, so that the zones ER, EW, and EB are empty. The program then proceeds by incrementing r, and decrementing w and b while making the necessary swaps, until the area marked “X” is empty because $r = w+1$.

```
program DutchNationalFlag (input, output);
const
   N = 10;

type
   color = (red, white, blue, blank);
   buckets = array [1..N] of color;

var
   v : buckets;
   i : integer;

function LOOK(i : integer) : color;
   begin
      LOOK := v[i]
   end;

procedure PUT(i : integer; c : color);
   begin
      v[i] := c
   end;

procedure SWAP(i, j : integer);
   var
      t : color;

   begin
      if ((i > N) or (i < 1) or (j > N) or (j < 1)) then
            writeln ('wrong index passed to SWAP')
      else
         begin
            t := v[i];
            v[i] := v[j];
            v[j] := t
         end
   end;  {SWAP}
```

```
procedure Decrease(var r, w, b : integer);
   var
      colr, colw : color;

   begin
      colr := LOOK(r);
      while ((colr = red) and (r < w)) do
         begin
            r := r + 1;
            colr := LOOK(r)
         end;
      if (r < w) then
         begin
            {DecW}
            colw := LOOK(w);
            while ((colw = white) and ((r+1) < w)) do
               begin
                  w := w - 1;
                  colw := LOOK(w)
               end;

            case colw of
               red:    begin
                          SWAP(r, w); r := r + 1
                       end;
               white:  w := w - 1;
               blue:   begin
                          SWAP(w, b); w := w - 1; b := b - 1;
                          SWAP(r, w)
                       end
            end
         end;
      case colr of
         red:    r := r + 1;
         white:  w := w - 1;
         blue:   begin
                    SWAP(w, b); w := w - 1;
                    b := b - 1;
                 end
      end
   end;  {Decrease}
```

```
procedure Rearrange(var r, w, b : integer);
   begin
      while (w >= r) do
         Decrease(r, w, b)
   end; {Rearrange}

procedure DutchFlag;
   var
      r, w, b < integer;
   begin
      r := 1;
      w := N;
      b := N;
      Rearrange(r, w, b)
   end; {DutchFlag}
                             {MAIN PROGRAM BODY}
begin
   {initialize the object v}
    DutchFlag;
end.  {DutchNationalFlag}
```

LEXICON

A. Auxiliary functions

card: set → integer

$card(s) \overset{\text{df}}{=} |\, s \,|$ (i.e. number of elements in the set s)

flag: buckets → boolean

$flag(v) \overset{\text{df}}{=} \exists r,b\,[partial\_flag(v,r,r-1,b)]$

partial_flag: buckets × integer × integer × integer → boolean

$$
\begin{aligned}
partial\_flag(v,r,w,b) \overset{\text{df}}{=}\ & (1 \le r) \wedge (r-1 \le w) \wedge (w \le b) \wedge (b \le N) \wedge \\
& \forall i\,(1 \le i \le N)\,[\ ((i < r) \Rightarrow (v_i = \text{red})) \wedge \\
& \qquad ((w < i \le b) \Rightarrow (v_i = \text{white})) \wedge \\
& \qquad ((b < i) \Rightarrow (v_i = \text{blue}))\ ]
\end{aligned}
$$

Note: v_i is defined in part C of this Lexicon.

same_colors: buckets × buckets → boolean

$same\_colors(v1,v2) \overset{\text{df}}{=}$

$$
\begin{aligned}
& (card(\{i \mid (1 \le i \le N) \wedge (v1_i = \text{red})\}) = card(\{i \mid (1 \le i \le N) \wedge (v2_i = \text{red})\})) \wedge \\
& (card(\{i \mid (1 \le i \le N) \wedge (v1_i = \text{white})\}) = card(\{i \mid (1 \le i \le N) \wedge (v2_i = \text{white})\})) \wedge \\
& (card(\{i \mid (1 \le i \le N) \wedge (v1_i = \text{blue})\}) = card(\{i \mid (1 \le i \le N) \wedge (v2_i = \text{blue})\}))
\end{aligned}
$$

B. Pascal external definitions and declarations

```
const N = {literal non-negative integer}
type color = (red, white, blue);
type buckets =   {vector(N, color) - cf. part C of this Lexicon}
var v : buckets;
procedure LOOK(i : integer);
  {cf. part C of this Lexicon}
procedure SWAP(i, j : integer);
  {cf. part C of this Lexicon}
```

C. vector(n,elem) Module Interface Specification

(0) CHARACTERISTICS

- type specified: vector(n,elem)
- features: single-object, generic
- foreign types: elem, <integer>, <positive_integer>
- foreign types: n: <positive_integer>, elem

(1) SYNTAX

ACCESS-PROGRAMS

Program Name	Arg#1	Arg#2	Value Type
LOOK	<integer>:V		elem
PUT	<integer>:V	elem:V	
SWAP	<integer>:V	<integer>:V	

(2) CANONICAL TRACES

canonical(T) $\Leftrightarrow$ T = $[\mathrm{PUT}(i, e_i)]_{i=1}^{n}$

_ = $[\mathrm{PUT}(i, \_)]_{i=1}^{n}$

EQUIVALENT NOTATION FOR TRACES

Trace	Equivalent notation
v.LOOK(i)	v_i

(3) EQUIVALENCES.

T.LOOK(i) ⇒ T

T.PUT(i, e) ⇒

Condition	Equivalence
$\neg(1 \le i \le n)$	%wrong_index%
$1 \le i \le n$	T1.PUT(i,e).T2 where T=T1.PUT(i,x).T2

T.SWAP(i, j) ⇒

Condition		Equivalence
$\neg((1 \le i \le n) \wedge (1 \le j \le n))$		%wrong_index%
$(1 \le i \le n) \wedge (1 \le j \le n) \wedge$	$(i < j)$	T1.PUT(i,x).T2.PUT(j,y).T3 where T= T1.PUT(i,y).T2.PUT(j,x).T3
	$(i = j)$	T
	$(i > j)$	T1.PUT(j,x).T2.PUT(i,y).T3 where T = T1.PUT(j,y).T2.PUT(i,x).T3

(4) RETURN VALUES

Program Name	Argument No	Value
LOOK	Value	e where #0 = T1.PUT(#1,e).T2

Precise Documentation of Well-Structured Programs

David Lorge Parnas, Jan Madey[1], Michal Iglewski[2]

Telecommunications Research Institute of Ontario (TRIO)
CRL, McMaster University, Hamilton, Ontario, Canada L8S 4K1

ABSTRACT

This paper describes a new form of program documentation that is precise, systematic and readable. This documentation comprises a *set of displays* supplemented by a *lexicon* and an *index*. Each *display* presents a program fragment in such a way that its correctness can be examined without looking at any other display. Each display has three parts: (1) the specification of the program presented in the display, (2) the program itself, and (3) the specifications of programs invoked by this program. The displays are intended to be used by Software Engineers as a reference document during inspection and maintenance. This paper also introduces a specification technique that is a refinement of Mills' functional approach to program documentation and verification; programs are specified and described in tabular form.

1 Introduction

The process of program development has been thoroughly studied for nearly 30 years and useful insights have been gained. However, the focus of this work has been on designing the *first* version of a program. If a software product is successful, the program will have many more readers than writers and will be studied and revised many times. Moreover, while the writers have had the time to become closely familiar with the program, most readers will not have that luxury. We consider the needs of readers, e.g. reviewers and maintainers, to be at least as important as the needs of program designers. Although proper decomposition of the software into modules will reduce the complexity and length of programs, there will still be programs whose length makes them difficult to understand. This paper presents a method that can be used by developers to present their programs in a way that makes review and maintenance easier. The heart of the method is a way of precisely summarizing the effects of a program component, so that reviewers and maintainers do not have to study that code when looking at components that interact with it. The program and documentation are organized in such a way that the information needed to study a component is presented together with that component. This method is intended for programs that are well-structured in the sense defined later in this paper.

The present report is a revised version of [24]; it will appear in IEEE Transactions on Software Engineering.

1.1 On the role of documentation

Anyone who has ever seriously read a lengthy program produced by others (for example to inspect it or to make changes to it) realizes the importance of documentation. Some argue that well-written programs are self documenting. Practical experience suggests that this is true only for small programs; human beings cannot easily understand long programs. When asked to study such programs, we tend to focus on little details while making use of inaccurate

[1] Permanent address: Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland

[2] Permanent address: Département d'informatique, Université du Québec à Hull, Hull, Québec, Canada J8X 3X7

descriptions of the overall structure. The combination of a large amount of detail with inaccurate or vague descriptions of the structure makes it quite common for serious errors to escape the reviewers' attention.

A design concept or algorithmic method that was obvious to the programmer at the time the program was written will not be obvious to other programmers, or even to the same programmer, one year later. Even if the program was developed using a systematic refinement process, there are few traces of that process in the final code. Although the program's author may have thought of the program in terms of a set of building blocks, each with a clearly defined function, it is not easy to identify those blocks and induce their functions by looking at the final code.

1.2 Studying long programs

When studying a long program, we must decompose it into small parts and then, provisionally, associate a function with each one. We must then convince ourselves of two things: (1) if each part implements its assigned function, the whole program will be correct, (2) that each part implements its assigned function. Frequently, we find that our provisional assumptions were not exactly what the programmer intended. Then, after revising our initial division and function descriptions, we try again. In principle, this iterative process converges and we learn whether or not the program is correct. In practice, we usually give up before we have a complete understanding of the program. The process terminates when we run out of time or patience.

1.3 Conventional documentation

Experienced development organizations have long recognized the need for documentation and there are extensive documentation standards. Unfortunately, when one tries to use this documentation, it is not found to be very useful. Often, the document includes a narrative description of the program - a translation of the program into a "natural" language. For people with an understanding of programming, it is usually easier to read the program itself than prose that attempts to say the same thing. Our natural languages were not intended to be used for precise descriptions where small details are critical. Most documentation encountered in industry is vague, inaccurate, and incomplete.

When documenting programs, there seems to be a tendency to focus on the details that we think will be hard to remember while ignoring the basic structural decisions, which seem obvious. Later, readers find that the structure is not obvious and the details are overwhelming. Moreover, most documentation is informally organized. Even when the desired information is present, it is not obvious where it will be found. When the information is found, it is often inconsistent or inaccurate. Industrial experience suggests that a huge portion of the "maintenance effort" goes into finding information and then finding an expert who can confirm or correct the information that was found.

The inadequacies of most software documentation can, in part, be blamed on the differences between standard engineering practice and the way that software systems are designed. In engineering, the production of design documents plays a key role - it is rare to find an engineer proceeding by building first and documenting later. In engineering, mathematics is extensively used to provide accurate and detailed descriptions of the products to be built; the need for precise descriptions of each component of larger products is almost universally accepted. In contrast, software systems are commonly produced *before* proper documentation is written; documentation is not viewed as a part of the design activity but as an additional task required by bureaucratic regulations or ignorant customers. The use of mathematics in describing programs is rare. As a result, the documentation is of limited value for programmers, reviewers and maintainers.

1.4 Design through documentation

The methods presented in this paper must be understood in the context of the complete documentation scheme described in [19].

It is widely accepted that the documentation of a computer system must include a *software requirements docu-*

ment (consisting of a *system requirements document* and a *system design document*). These documents provide a black-box description of the system as a whole, a description of the hardware structure, and a black-box description of the software. Detailed discussions of these documents can be found in [5, 6, 28, 29].

Because large software systems are seldom the product of a single person, the task of constructing them must be split into several smaller work assignments. Each assignment is to design and implement a group of one or more programs, which we call a *module*. In well-structured systems, the programs in a module share access to a private data structure and implement one or more abstract *objects*. We call programs that are part of the module, and can be used from outside the module, the *access-programs* of the module. Programs that belong to other modules never read directly from, or write directly to, the internal data structure of a module; they always use a module's access-programs to get information about, or change the state of, any objects created by that module [16]. We recommend a *software module guide*, which describes the structure of the software system by indicating the design decisions hidden in each one [23]. For each module identified in the module guide, there should be a *module interface specification*, which provides a black-box description of the behavior of the objects created by that module. Our approach to specification of module interfaces (the *trace assertion method*) is illustrated in Appendix B and described in [21, 8].

For every implementation of a module interface specification (there may be several), there should be a document describing the *module internal design*; that document must describe the internal data structures and the effect of the module's access-programs on the state of that structure. The contents of these documents are defined in [19], which contains a more general discussion of the role and structure of documentation in software engineering. Examples of a detailed software requirements document can be found in [5, 28].

This paper focuses on the documentation of programs *within* a module. The documentation described here complements the documents mentioned above.

1.5 The responsibilities of program designers and reviewers

We believe that the reviewer or maintainer of a program should never have to guess its structure. The iterative process described in Section 1.2 must be eliminated. Programs should be presented to the reviewer and maintainer as a collection of small parts, each with a precise description of its function. The structure should be explicitly and precisely described in the documentation. It should be possible to review the small parts separately and know that, if each of the components is correct, the whole program is correct. In other words, the decomposition phase of the review process should not be repeated by the reviewer; it should be communicated by the designer. The reviewers must check that the structure is a good one, but their primary responsibility should be checking each of the small fragments against the description of its function.

It is clear that we are asking more work from the designers than they usually do. We are asking them to write down, systematically, information that reviewers and maintainers would otherwise have to discover for themselves. Because there will be more readers than writers, and because the writer already knows the information, we believe that the combined cost of developing and maintaining the product will be lower if the writer presents the program as proposed in this paper and the documentation is kept live by revising it each time that the program is revised. Moreover, our experience suggests that the quality of the program will be improved as a result of requiring the programmer to produce the documentation.

1.6 The use of mathematics in documentation

Our method is based on a mathematical model of programs and uses mathematical notation to provide precise descriptions of programs. Although mathematics is not commonly used in programming practice, we believe that the ability to use mathematics in this way will be the hallmark of Software Engineers in the future.

Most demonstrations of the use of mathematical methods in software engineering emphasize program development or verification. This paper focuses on documentation. While we believe in systematic development, we believe that the documentation delivered with a program should not depend on the program development process. This paper discusses the documentation that should be associated with a program, not the procedure for developing the program.

Many papers on formal methods for program development emphasize the idea of proving a program to be correct. Our paper is less ambitious. Although we believe that the mathematical documentation we describe could be used as input to a program verification process (our notation is close to classical predicate logic), our emphasis is on documentation that is valuable whether or not formal proof is attempted. We have used this type of documentation as input to an inspection process [22], but this paper does not discuss formal verification.

1.7 Introduction to the "Display Method"

This paper introduces a method of documenting well-structured programs called the Display Method. It requires designers and implementers to present their programs as sets of displays. The method is based on the well-known fact that a well-structured program can always be written as a short text in which the names of other programs[3] may appear and the programs named can also be short. The down-side of such an organization is that there will be many programs and to understand any one of them, one must understand several others. We overcome this by presenting the material in displays. A *display* is a document in which a program is presented in such a way that its correctness can be examined without looking at other displays.

Though the Display Method can be used with any specification technique (and any imperative programming language), we decided to illustrate it using a refinement of Mills' approach[4] [13, 14]. We have chosen to base our work on Mills' method, rather than approaches that are more popular, because we find it more suitable for large programs. Unlike Floyd [3], Hoare [7], Dijkstra [2], and their followers, Mills, although equally rigorous, does not include axiomatic descriptions of programming language statements among his basic definitions. Instead, he assumes that the programs, from which other programs are constructed, can be described by mathematical functions. Since this assumption is valid for all deterministic programs, one can apply Mills' approach even when the component programs are quite long and complex. This allows the same method to be used for well-structured programs of any size.

Many other methods do not deal with the problem of how to assemble small programs into large ones. For example, if one were to mimic the techniques used by Wirth for the eight queens problem [27], one would keep repeating the parts of the text that were developed early in the refinement process. For a long program, this would not be practical. Program texts would grow so long that no one could keep them under full intellectual control. Other presentations of moderate-sized programs are confusing because it is not clear how the small sections fit together (cf. e.g. chapters 14 and 24 in [2]). Our method avoids both problems.

In documentation, the notation is very important; documents are to be read by experts from a variety of fields and should be easily understood. We must apply the principle of "divide and conquer" when designing notation; readers should not have to parse long expressions. Our approach is based on the use of tables to describe mathematical functions, relations, and sets [18]; such tabular notation has already been used in practice (e.g. in safety-critical software for a nuclear plant [22]) and has proven practical.

Some readers will observe that, in our examples, the volume of the documentation is much greater than the volume of program code. This is a consequence of the need to use small, but nontrivial, examples in a paper of this sort. The length and complexity of a precise description of a program's effect does not necessarily increase with the length of the program. In fact, it often happens that the description of the effects of a part of a program is more complex than the description of the whole program. Consequently, the ratio of program size to program documentation size is under

[3] Note that these named programs need not be subroutines. In the text submitted to the compiler some of the program names may have been replaced by the text of the program itself.

[4] Although Mills is the best known proponent of this approach, similar ideas were independently discovered by many others.

the control of the document's author. When documenting long, but easily understood, programs, it is not necessary to describe the behavior of small components; consequently, the ratio of code size to documentation size increases. In practice, the components identified will be longer than those in this paper's examples.

1.8 Organization of this paper

In the next section, we review some old issues about the structure of programs. Section 3 contains some basic definitions used in our approach to program description. Section 4 presents the main ideas of the Display Method and introduces important notational conventions. The method is illustrated by two complete examples (presented in appendices). A discussion of these examples and some sample displays are presented in Section 5. The lessons learned from previous experience with the proposed approach, and some future plans, are described in the final section.

2 Well-structured programs

This section motivates restricting the structure of programs, and then states the constraints proposed. While some researchers consider the themes in this section obvious, many practitioners continue to ignore them.

2.1 Hierarchical control structure in programs

The well known "structured programming" constructs, such as "while" and "if then else" have two very useful properties:

(1) programs constructed using them can be decomposed into a hierarchy of parts (with lower level parts completely contained in an upper level part) using simple parsers; those parsers need not even distinguish one identifier from another,

(2) the semantics of the total program can be determined from the semantics of its parts, using simple operations (cf. e.g. [17, 20]).

Further, the semantics of the program can be determined in a flexible sequence, finding the semantics of inner parts first and finding the semantics of a sequence of programs constructed using ";" either left to right, right to left, or a mixture - as one prefers. In fact, the work need not be sequential. In contrast, the use of "go to" and labels makes it difficult to find a decomposition in which the components have simple semantics.

The above properties are important because they make it easier to study a long program one small part at a time, and to do so without a previous understanding of the overall structure of that program. In contrast, when a program is constructed using labels and unrestricted jumps, considerable understanding of the program is needed in order to decompose it into parts that can be studied independently.

Programs having the desired properties are often referred to as having a *hierarchical control structure* or as *well-structured programs*. The Display Method is intended to be used for such programs.

2.2 Use of data abstractions

Even the best structured program will be difficult to explain and understand if it is presented in terms of complex data structures. Essential information about the nature of the data and algorithm can be obscured by representational details.

Complex data structures should be encapsulated (or hidden) by the introduction of new data types that have been designed specifically for the type of data being stored. Such specially designed data types, known as *abstract data types* (because they allow the reader to abstract from the actual representation of the data), were introduced into the literature by Dijkstra [1]. The principle of information hiding, long used by very good programmers, was first discussed explicitly in [15].

Precise documentation of a program that uses abstract data types is not possible unless the properties of the abstract operations are also precisely documented. In this paper we presume that the abstract types are implemented by modules whose properties have been specified by a module specification method such as that discussed in [21] or by one of the algebraic methods. However, the examples in this paper have been selected so that they can be understood without an understanding of module specifications.

2.3 Discipline vs. notation

It will be seen that the usability of the discipline proposed in this paper is independent of:

(1) the notation used to present the information in a display,

(2) the language used for coding the program, and

(3) the method used to verify the displays.

The present paper focuses on the contents of the displays, using one programming language and one of many possible notations for presenting specifications. We have chosen the Pascal language [9] for the initial examples, not because it is ideal but because it is familiar. We have chosen to use tabular representations of LD-relations for reasons explained in Section 3, but we believe that the display method could be adapted for use with other notations such as VDM [10]. While we do not present formal verifications, we claim that the information necessary for verification of any display is contained in that display and the lexicon.

3 Mathematical description of program effects

In this section we show how to use standard mathematical concepts to describe the effect of program execution. We introduce the LD-relation [17, 20] and its application to program description and specification. Those wanting to use this method must read this section carefully. The literature contains many notations that are similar but differ from this one in subtle ways. In particular, the meaning of our notation is (necessarily) different from that of both Hehner [4] and VDM [10][5]; confusion can arise if one assumes otherwise.

3.1 Finite state machine approach

A digital computer can usefully be viewed as a finite state machine. For our purposes such a machine is one that is always in one of a finite set of states and whose operation consists of a sequence of state-changes, i.e. transitions from state to state.

Definition 1:

We will use the term "*program*" to denote a description of state-change sequences in a finite state machine. Programs may describe both finite (terminating) and infinite (non-terminating) state-change sequences.

□

Let P be a program and let U be the set of states of a digital computer. The following terminology and notation will be used in the sequel:

Definition 2:

- A complete state-change sequence described by P is called an *execution of P*.
- The set of executions of P that begin with the state x, $(x \in U)$, is denoted by $e_P(x)$, and x is called the *starting state* of the sequences in that set. The set of all executions of P is denoted by Exec(P,U).

[5] The work described in [4] stresses the description of programs by a single predicate, which limits the ability to provide complete descriptions of non-deterministic programs. VDM only describes the behavior of a program when started in states that satisfy a precondition that guarantees termination. We chose a method that allows complete description of any program.

- If there exists a finite execution in $e_P(x)$ with final element z, then:
  - we write $<x,\ldots,z> \in e_P(x)$,
  - we say that this *execution terminates (in z)* and call z the *final state (of this execution)*.
- If $<x,\ldots,z> \in e_P(x)$, we also say that the *program P* may *start in x* and *terminate in z*.
- If $e_P(x)$ contains an infinite sequence, we say that this is a *non-terminating execution*, and denote it by $<x, \ldots>$.
- If there exists a state x, $(x \in U)$, such that $e_P(x)$ contains two or more distinct executions, then P is a *non-deterministic program*.
- If for a given state x, $(x \in U)$, every member of $e_P(x)$ terminates, then x is called a *safe state* for P. The set of safe states for P is denoted by S_P.

□

3.2 Limited-domain relations (LD-relations)

If we are not interested in the intermediate states of executions, then every deterministic program can be described by a *program function*, a function whose domain is the set of safe states and whose range is the set of final states [13]. Non-deterministic programs cannot be fully described by program functions. First, a program started in a safe state may terminate in one of several distinct final states; thus a relation must be used and not a function. Second, a program started in a state that is not a safe one may sometimes terminate and sometimes not; a relation on the set of states does not provide sufficient information to distinguish between safe and unsafe states.

In [17] one possible solution[6] to the latter problem was suggested: instead of a relation we use a pair, (relation, set). This set will be used to provide the necessary additional information. The definitions that follow describe this solution. We begin by defining some formal structures, and describe how these can be used to describe and specify programs.

Definition 3:

- A *binary relation* R on a given set U is a set of ordered pairs with both elements from U, *i.e.* $R \subseteq U \times U$. The set U is called the *Universe*.
- The set of pairs R could also be defined by its *characteristic predicate*, R(p,q), i.e. $R = \{(p,q): U \times U \mid R(p,q)\}$.
- The *domain* and the *range* of R can be expressed as follows:

$$Dom(R) = \{p \mid \exists q\, [R(p,q)]\}, \qquad Range(R) = \{q \mid \exists p\, [R(p,q)]\}.$$

□

In the sequel the term "relation" means "binary relation".

Definition 4:

Let U be a set. A *limited-domain relation* (LD-relation) on U is an ordered pair $L = (R_L, C_L)$, where:

- R_L, the *relational component* of L, is a relation on U, i.e. $R_L \subseteq U \times U$,
- C_L, the *competence set* of L, is a subset of the domain of R_L, i.e. $C_L \subseteq Dom(R_L)$. □

3.3 Applications of LD-relations

An LD-relation can be used both to specify and to describe programs. A program specification is a statement of the requirements that an acceptable program must satisfy. A program description is a representation of the visible behavior of a specific program. A specification may allow behavior that is not actually exhibited by the program. Since the same mathematical structure is used for both descriptions and specifications, each must be labelled to indicate the intended interpretation of the information. The following sections explain our usage of these terms more precisely.

[6] Other, mathematically equivalent, approaches introduce a special symbol to represent non-termination, cf. e.g. [12]. The approach chosen here allows representation in terms of variable values without the addition of any special symbols or states.

3.3.1 Program descriptions

As was mentioned in Section 3.2 a deterministic program can be described by a program function. We can generalize this notion, as follows:

Definition 5:

- Let P be a program, let U be a set of states, and let $L_P = (R_P, C_P)$ be an LD-relation on U such, that:
  - $(x,y) \in R_P \Leftrightarrow \langle x,\ldots,y\rangle \in \mathrm{Exec}(P,U)$,
  - $C_P = S_P$.[7]

  L_p is called the *LD-relation of P* and *the description of* P.
- If $C_P = \mathrm{Dom}(R_P)$, then (by convention) the competence set need not be given explicitly. In other words, if C_P is not given, then it is, by default, $\mathrm{Dom}(R_P)$.

□

One should note the following consequences of this definition:

- if $x \in C_P$, P always terminates when started in x and if $(x, y) \in R_P$, P may terminate in y,
- if $x \in (\mathrm{Dom}(R_P) - C_P)$, the termination of P when started in x is non-deterministic; in that case if $(x, y) \in R_P$, P may terminate in y, but it might not terminate at all,
- if $x \in \mathrm{Dom}(R_P)$ and P starts in x, then P will never terminate.
- If P is a deterministic program, then the relational component, R_P, is a function, $C_P = \mathrm{Dom}(R_P)$, and hence L_P is the program function defined in [13]. Hence, our approach is "upward compatible" with that of Mills.

3.3.2 Specification of programs

We can also use LD-relations to specify a program. In the general case one may be given a set of LD-relations and be asked to write a program that satisfies at least one of them.

Definition 6:

Let $L_p = (R_P, C_P)$ be the LD-relation of a program P (where U is the set of states). Let S, called a *specification*, be a set of LD-relations on U, and let $L_S = (R_S, C_S)$ be an element of S. We say that:

- P *satisfies the LD-relation* L_S, iff $C_S \subseteq C_P$ and $R_P \subseteq R_S$,
- P *satisfies the specification* S, iff P satisfies at least one element of S.

□

Often, S has only one element. If S is a specification and $S = \{L_S\}$, then we can also call L_S a specification. This is the usual case and the only one illustrated in this paper.

If L_p is used as a specification, P will satisfy it. However, P will satisfy many other specifications and other programs may satisfy L_p.

4 The Display Method of program documentation

In the Display Method, program documentation consists of a set of *displays*, supplemented by a *lexicon* and an *index*. This section explains these concepts.

4.1 Displays

A well-structured program can usually be written as a short text in which names of other programs may appear.

[7] Please note that C_P is not the same as the precondition used in VDM [10] and other methods. LD-relations provide a complete description of the behavior of a program, not just a description of its behavior when the starting state is in C_P. R_P is a description of the behavior within its domain, not just within C_P.

These named programs can also be short and can include the names of other programs. By a *display* we mean a concise document, (preferably 1-2 pages), in which a short program is presented in such a way, that its correctness can be determined without examining other displays. More precisely:

<u>Definition 7:</u>

A *display* is a document that consists of the following three parts:

- P1: a specification for the program presented in this display,
- P2: the program itself. The names of other programs may appear in this text; we say that the these programs are *invoked* in this display,
- P3: specifications of all programs (other than that specified in P1[8]) invoked in P2 that are not known[9].

□

Note, that the terms "program" and "invocation" are to be understood in a generic sense. A name appearing in the program P2 may represent a procedure call (in which case it will usually be followed by actual parameters) but may also be treated as a macro call, to be replaced by a sequence of instructions. In either case, the construction of the resulting program by merging the P2 parts of all displays should be a simple operation that can be done automatically. As discussed below (cf. Section 4.4), if an invoked program is not an available[10] program, its specification must appear as P1 in another display.

4.2 The lexicon

To avoid repetition of information in several displays, and the maintenance problems that result from redundant information, we place that information in a separate document, called the lexicon.

<u>Definition 8:</u>

A *lexicon* is a dictionary containing definitions of terms used in the program being documented. It will contain the definitions of any mathematical functions, programs constants, types, etc. that are used in more than one display.

□

We refer readers to the lexicon wherever the information that it contains would have appeared.

4.3 The index

To help those studying a program we also recommend an index.

<u>Definition 9:</u>

An *index* is a list of all the variables, programs, etc. indicating where those items appear in the displays. If some names are used with more than one meaning, we also describe the category of each name.

□

4.4 Completeness and correctness

Each display can be reviewed without any reference to other displays; its correctness can be verified without looking at the implementation of either the programs that are invoked in that display or the programs that invoke the program it describes.

[8] Note that if a program invokes itself recursively, one should not include the specification of that program in its own P3.

[9] A *known* program is one that does not require a specification. The semantics of known programs are assumed to be understood. Every project should have a list of programs that are considered to be known.

[10] An *available* program is one that exists in a project or system library. We need not have a display for an available program. Available programs are not necessarily known programs. Known programs are usually, but not always, available.

Definition 10:

- A *display is correct* if the program in P2 will satisfy the specification in P1, provided that the programs invoked in P2 satisfy the specifications given in P3.
- A *set of displays is complete* if, for each specification of a program (except an available program) that is found in P3 of a display, there exists another display in which this specification is in P1[11].
- A *set of displays is correct* if (1) the set of displays is complete, and (2) all displays are correct.

□

A display can be supplemented by an additional part, P4, that contains a demonstration of its correctness. This could be either a description of the informal reasoning routinely done by a programmer, or a more formal argument. The existence of this additional section would make the reviewer's task simpler – one would not have to invent a "proof", only to check one. In the present paper we do not supply P4.

4.5 Notation

In the examples of displays in this paper we will use LD-relations for program specifications and the Pascal language for programs. The LD-relations will be represented in a *tabular* form [18]. The basis of such representation is the fact that every relation can be understood as a set of ordered pairs defined by its characteristic predicate (cf. Definition 3, Section 3.2). A predicate is also used to represent the competence set of an LD-relation.

4.5.1 Introductory conventions

This section introduces some useful notational conventions. It is usual to describe predicates using boolean expressions. The tabular notation used in the present paper will be explained by means of examples.

Convention 1:

Let P be a program specified by an LD-relation L = (R, C), and let $(v_1, ..., v_k)$ be the variables in P that constitute its data structure, v. Then:

- " $'v_i$ " (to be read "v_i *before*") denotes the value of the program variable v_i before an execution of P,
- " v_i' " (to be read "v_i *after*") denotes the value of the program variable v_i after a terminating execution of P,
- " 'v " (to be read "v *before*") denotes the value of the data structure v before an execution of P,
- " v' " (to be read "v *after*") denotes the value of the data structure v after a terminating execution of P.

□

Each pair in R will be of the form $('v_i, v_i')$. Note that $'v_i$ and v_i', as mathematical variables, could have been replaced in the definition of R by other symbols, but we would then have to establish an explicit correspondence between those symbols and the components of program data structure. Our notational convention makes the correspondence implicit in the variable names.

Convention 2:

If it is clear from the context that the programming variables are a, b, c, ..., then one may write "R(,)" instead of "R(('a, 'b, 'c, ...), (a', b', c', ...))".

□

Convention 3:

In examples we will often need to express the fact that some variables do not change their values during the execution of a program. We found it useful to introduce a predicate symbol NC ("Not Changed").

$$NC(v_1, ..., v_k) \Leftrightarrow (v_1' = 'v_1) \wedge ... \wedge (v_k' = 'v_k)$$

□

[11] Note that completeness of the set of displays can easily be checked mechanically.

Convention 4:

When we write a boolean expression to characterize a set of program variable values, we always assume that programming variables can only have values appropriate to their types and do not state those restrictions explicitly.

□

Convention 5:

The variables that form the domain and range for a given LD-relation can be listed in the heading preceding the LD-relation and need not be repeated in the characteristic predicates.

□

4.5.2 Tabular representations

To explain the tabular notation used in this paper, we introduce the following simple problem:

PROBLEM

Write a program which finds the maximum of two integer values stored in programming variables.

Discussion:

The data structure of this program will consist of three variables of integer[12] type named a, b, and max. The first two will be used to store the input values, while the third one will store the result. We will require that the final values of a and b be the same as the initial ones. Note, that the initial value of max (i.e. 'max) is irrelevant.

The above considerations lead to the following specification of this program by an LD-relation, $L_S = (R_S, C_S)$:

- $R_S(,) = \{ (a' = {'a}) \wedge (b' = {'b}) \wedge ((({'a} \leq {'b}) \wedge (max' = {'b})) \vee (({'a} \geq {'b}) \wedge (max' = {'a})))] \}$,
- $C_S = Dom(R_S)$.

□

Tabular form:

The characteristic predicate of the relation R_S can be given in tabular form.

- A direct representation of this predicate as a table, is as follows:

T1	'a ≤ 'b	'a ≥ 'b
a' =	'a	'a
b' =	'b	'b
max' =	'b	'a

- For ease of checking tables, we usually require that conditions that head columns be mutually exclusive[13]. In this case we should replace "≤" by "<", or "≥" by ">". The first replacement leads to the following table:

T2	'a ≤ 'b	'a > 'b
a' =	'a	'a
b' =	'b	'b
max' =	'b	'a

[12] We will use different fonts to distinguish between programming language elements (e.g. "integer", "true"), and mathematical terms (e.g. "integer", "*true*").

[13] This requirement is not strictly necessary, just useful. Eliminating heading overlap for tables that represent functions, cannot change their meaning and, consequently, does not result in overspecification. We show how to describe relations below.

- The first two rows of *T2* can easily be expressed conventionally. We can combine both notations as follows:

$$(a' = {}'a) \wedge (b' = {}'b) \wedge$$

T3	'a ≤ 'b	'a > 'b
max' =	'b	'a

- Using "NC" we can rewrite the above expression as follows:

T3	'a ≤ 'b	'a > 'b
max' =	'b	'a

$\wedge$ NC(a, b)

- The conditions in *T3* itself can be written in another way (which may make the table easier to read when expressions are long) - the string above a dotted line is treated as if it were repeated in each column below that line:

T4	('a ≤ 'b) =	
	true	*false*
max' =	'b	'a

$\wedge$ NC(a, b)

- The conditions heading columns in *T4* can be written in yet another form, as follows:

T5	'a	
	≤ 'b	> 'b
max' =	'b	'a

$\wedge$ NC(a, b)

- The equality operator in the "value after" phrase can be replaced by any other relational operator or by the vertical bar, "|". The latter is to be read "such that". When "|" is used, the entries in that row must be boolean expressions; the value of the variable must satisfy the predicate described in the relevant column. For instance, the row defining max' in the table *T3* could have been written as follows: Note that the use of "|" allows the description of relations

T6	'a ≤ 'b	'a > 'b
max' |	max' = 'b	max' = 'a

$\wedge$ NC(a, b)

or non-deterministic programs without having overlapping column headings.

- The table identifiers: *T1*, *T2*, ... are optional and have no formal meaning.

□

4.6 Parameters and side-effects

Programs presented in displays will often use procedures. Procedures are not programs in the sense described above; they are program schemata, which cannot be described by functions or LD-relations. Procedures with formal parameters can be represented by program function schema, mappings from actual parameters to program functions, as described and illustrated in [8]. A procedure invocation, including the actual parameters, is a program in the sense of this paper. Here, we provide the program function corresponding to each actual invocation.

(1) The specification of the procedure invocation will be written in terms of *actual* parameters. In the declaration of this procedure, however, *formal* parameters will be used. Both the specifications of invoked programs appearing in the declaration, and statements in the declaration body must be written in terms of the formal parameters of the procedure (and its other local or non-local objects). The binding of parameters is done according to semantics of the given programming language (Pascal, for the examples in Section 5).

(2) For simplicity's sake, we will avoid any form of aliasing[14] in our examples, e.g.:

- If more than one parameter is called by variable, then the actual parameters will be different variables.
- If there are side-effects, then a variable external to the procedure body will not be passed as a parameter called by variable.

5 Examples

In this section we will illustrate the Display Method on two simple but complete examples written in Standard Pascal [9]. We decided to use simple and well-known problems to emphasize the main ideas of the proposed approach. The complete sets of displays with the lexicons and the indices are presented in appendices.

5.1 "Binary search"

We begin with a problem familiar to all programmers, so that we can focus on the display method.

5.1.1 Informal description of the problem

Given an integer x, and a list of $n \geq 1$ integers, $a_1, \ldots, a_n$ in non-decreasing order:

- check whether x is among $a_1, \ldots, a_n$ and return this information,
- if x is among $a_1, \ldots, a_n$, find an index j such that $x = a_j$.

If the list is empty or not sorted, we require program termination but do not care what the program does because we assume that the program will not be invoked under such conditions[15].

5.1.2 Discussion

(1) A solution to this problem (by the well-known "binary search" method) will be presented as a Pascal procedure declaration and its invocation. It is the invocation that must satisfy the specification. This procedure declaration should be preceded by definitions and declarations of needed constants, types and variables, to set up the data structure whose values will form the state space.

(2) The following assumptions are made about the correspondence between the description of the problem and Pascal programming language objects:

- Integer numbers are represented by values of the standard type integer[16].
- The length of the list is represented by the constant n.
- The list itself is represented by the value of the variable A of a type vector, defined as array[1..n] of integer.
- The integer x is represented by the value of the variable x of type integer.
- The results are represented by the values of two variables: j of type integer, and present of type Boolean.

[14] Aliasing does not invalidate the basic theory or model used in our work. However, it complicates the representation of data states. In our examples, there is a 1:1 correspondence between identifiers and elements of the data structure at any point in the program. This allows us to represent state by a list of values in which each element corresponds to one identifier. If aliasing is allowed, or with dynamic data structures, one needs a more elaborate scheme for identifying data states.

[15] This is undoubtedly a foolish assumption in practice, but it is useful for illustrating the meaning of the notation. In this example, if a program is called when the assumptions are not satisfied, even the values of the variables x and A are allowed to change.

[16] Recall that by convention we use different fonts to distinguish Pascal objects from mathematical ones.

(3) We will specify, that:

- The values of A and x should not change if the program is invoked under normal conditions.
- If the desired index exists, then j will return its value and present will be true. If the index does not exist, present will be false and j can have any integer value.

(4) The following observations and conventions are related to the data state:

- Initially, the data state is determined by the values of the constant n and the variables A, x, j, and present.
- The relational component R of the LD-relation should specify acceptable changes of these values (however constants, by definition, do not change and their values need not be mentioned).
- For variables we will use the conventions introduced in the previous section.

5.1.3 Example of a display

We will present one display (the complete set is to be found in Appendix A). To help in understanding specifications, we begin by discussing P1 of this display in detail. We have numbered each line of part P1 and explain those lines in the notes below.

Specification

(1) Find(x, A, j, present)

(2) $R_0(,) = ((1 \le n) \wedge \forall i\,[\,(1 \le i < n) \Rightarrow ('A[i] \le 'A[i+1])\,]) \Rightarrow$

(3)		$\exists i\,[\,(1 \le i \le n) \wedge ('A[i] = 'x)\,] =$		
(4)		*true*	*false*	
(5)	j' |	'A[j'] = 'x	*true*	
(6)	present' =	true	false	$\wedge$ NC(x, A)

Notes on P1:

(1) The procedure invocation "Find(x,A,j,present)" lists actual parameters which form the data structure. If external[17] variables were used, they need to be listed in this line.

(2) Since the elements of the data structure are listed in line 1, we do not need to repeat them (Convention 2, Section 4). Without that convention we would have to write "R_0(('x, 'A, 'j, 'present), (x', A', j', present'))" instead of "$R_0(,)$". Next note, that the expression "$((1 \le n) \wedge \forall i\,[\,(1 \le i < n) \Rightarrow ('A[i] \le 'A[i+1])\,])$" is *true* if the input sequence is non-decreasingly ordered.

(3,4)This and the next line could have been written as one entry but we would have to repeat the long expression twice.

(5) The phrase "j' | *true*" expresses the fact that the program will satisfy the specification no matter what the value of j is when the program terminates.

(6) Notice that the logical values written here are Pascal constants. The other "*true*" and "*false*" were mathematical constants. The phrase "NC(x, A)" expresses the requirement that the variables with input values remain unchanged.

In P3 of the display, the rows for low and high are not strictly necessary because the new values of those variables are not constrained. Since these tables represent the characteristic predicate of the relation, variables that are not mentioned are not constrained. We sometimes include such rows to make this more explicit.

[17] We will use the term *external* to denote objects that are not local to a given program.

DISPLAY 1

Display 1 Specification

Find(x, A, j, present)
$R_0(.) = ((1 \le n) \wedge \forall i\,[\,(1 \le i < n) \Rightarrow ('A[i] \le 'A[i+1])\,]) \Rightarrow$

	$\exists i\,[\,(1 \le i \le n) \wedge ('A[i] = 'x)\,] =$		
	true	*false*	
$j' \mid$	$'A[j'] = 'x$	*true*	
present' =	true	false	$\wedge$ NC(x, A)

Display 1 Program

Procedure declaration:

```
procedure Find(e : integer; V : vector; var index : integer; var found : Boolean);
var low, high : integer;
begin
  Initialization; Body
end {Find}
```

Display 1 Specifications of Invoked Programs

Initialization	external variables: e, V, found, low, high
$R_1(.) = (low' = 1) \wedge (high' = n) \wedge (found' = false) \wedge NC(e, V)$	

(on Display 4)

Body	external variables: e, V, index, found, low, high
$R_2(.) =$ $(('low \le 'high) \wedge ('found = false) \wedge \forall i\,[\,('low \le i < 'high) \Rightarrow ('V[i] \le 'V[i+1])\,]) \Rightarrow$	

(on Display 2)

	$\exists i\,[\,('low \le i \le 'high) \wedge ('V[i] = 'e)\,] =$		
	true	*false*	
$index' \mid$	$'V[index'] = 'e$	*true*	
found' =	true	false	
$low' \mid$	*true*	*true*	
$high' \mid$	*true*	*true*	$\wedge$ NC(e, V)

END OF DISPLAY 1

5.2 "Dutch national flag" example

This example is based on [2], chapter 14.

5.2.1 Informal description of the problem

(1) There is an abstract data type "buckets". A value of this type may be used as a vector of N elements of type "color", where $N \geq 0$ is a fixed integer, and color $\overset{df}{=}$ {blue, red, white}. Each element is called a "pebble" by Dijkstra. We introduce a variable of type buckets, v, c of type color, and i,j of type integer. The operations on v are:

- PUT(i,c), which sets the value of i^{th} element of v to c, if N>0, (i.e. puts the c-colored pebble into the i^{th} bucket) and does nothing if N=0 or i is out of range.
- LOOK(i), which returns the color of the pebble in the i^{th} bucket and does nothing if i is out of range.
- SWAP(i,j), which swaps pebbles between the i^{th} and j^{th} bucket, if i≠j, and does nothing if i and j are equal or the arguments are out of range.

(2) The type buckets and the operations PUT, LOOK and SWAP are defined more formally in Appendix B (in the lexicon) by a parameterized module interface specification using the trace assertion method [21, 8]. The initial value of v is assumed to be set externally.

(3) We want to design a Pascal procedure that, given any initial arrangements of pebbles in v, "will rearrange (if necessary) the pebbles in the order of the Dutch national flag, i.e. in order from low to high bucket number first the red, then the white, and finally the blue pebbles." [2]. This procedure should:

- cope with all possible special cases, including missing colors and N=0,
- not introduce arrays of any sort, only a fixed number of variables of type integer and color, and
- not use the operation LOOK(i) more than once for each value of i.

5.2.2 Discussion

Our solution (and the description in this section) is based on the original proposal by Dijkstra. We will assume the existence of the external Pascal variable v of type buckets, as presented in the problem description above, and that the Pascal procedures PUT, LOOK, and SWAP are both available and known.

Although the pebbles are of only three different colors, the fact that we can only inspect pebbles one at a time, together with the requirement that we can only inspect each pebble once, implies that throughout the arrangement process, we have to distinguish between pebbles of four different categories, viz. *established red* (ER), *established white* (EW), *established blue* (EB), and *as yet uninspected* (X). We will divide the row of buckets into four (possibly empty) zones of consecutively numbered buckets, each zone being reserved for pebbles of a specific category. For keeping track of the place of the zone boundaries we will use three integer variables, r, w, b, with the meanings:

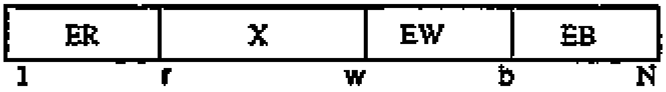
$1 \leq k < r$: the k^{th} bucket is in zone ER (number of buckets $r-1 \geq 0$)

$r \leq k \leq w$: the k^{th} bucket is in zone X (number of buckets $w-r+1 \geq 0$)

$w < k \leq b$: the k^{th} bucket is in zone EW (number of buckets $b-w \geq 0$)

$b < k \leq N$: the k^{th} bucket is in zone EB (number of buckets $N-b \geq 0$)

This is illustrated by the following figure:



Initially, r=1, and w = b = N, so that the zones ER, EW, and EB are empty. The program then proceeds by incrementing r, and decrementing w and b while making the necessary swaps, until the area marked "X" is empty because r = w+1.

5.2.3 Example of a display

The complete set of displays including the lexicon and index is to be found in Appendix B. In the display below there are three auxiliary functions (predicates) used: *flag*, *partial_flag*, and *same_colors*. Their formal definition is given in the lexicon. Intuitively, *flag*(v) is *true* if the colors in v form the required final configuration (zone X is empty); *partial_flag*(v,r,w,b) is *true* if colors are grouped as on the above figure. The predicate *same_colors*(x,y) is *true* if x and y have the same number of red, white, and blue pebbles.

DISPLAY 1

Display 1 Specification

DutchFlag	external variable: v
$R_0(,) = \mathit{flag}(v') \wedge \mathit{same\_colors}('v,v')$	

Display 1 Program

Procedure declaration:

```
procedure DutchFlag;
var r, w, b : integer;
begin
   r := 1; w := N; b := N;
   Rearrange(r, w, b)
end {DutchFlag}
```

Display 1 Specifications of Invoked Programs

Rearrange(r, w, b)	external variable: v
$R_1(,) = (('r = 1) \wedge ('w = N) \wedge ('b = N))$ $\Rightarrow$ $(\mathit{partial\_flag}(v',r',w',b') \wedge (w' = r'-1) \wedge \mathit{same\_colors}('v,v'))$	

END OF DISPLAY 1

6 Experience

The ideas reported in this paper are motivated more by practical experience than by theory. The theory has been introduced only to the extent that it was needed to provide a precise meaning for the notation. We have all had the frustrating experience of trying to read the mind of a programmer when trying to correct a program. The proposals in this paper represent our thoughts about what the programmer should have given us.

The method described in this paper is an improved version of the technique used in the inspection of safety-critical software for the Darlington (Ontario) Nuclear Power Generation Station [22]. It is important to understand that the Darlington experience was not an experiment; we did not gather data or make scientific observations. There was a job to be done and it had to be done as quickly as safety considerations would permit.

At the Darlington station, two safety-critical systems were, for the first time, implemented in software. The Atomic Energy Control Board of Canada (AECB) was not willing to allow the plant to operate until they were convinced of the correctness of the programs. Delays were very expensive for the owners of the plant, Ontario Hydro. The software had been ready for several years (because the rest of the plant was even further behind schedule), had been tested thoroughly, and was considered by its owners to be safe to use. However, the usual informal approaches to inspection did not provide the confidence level demanded by the AECB. The code, while not huge[18], was sufficiently complex that the engineers who inspected it using informal methods could not be confident that they had considered all of the possibilities and found all of the errors.

One of the preliminary inspections demonstrated that the requirements documentation was not complete or precise. An error caused by misinterpretation of a sentence was discovered. As a result, the manufacturer was asked to produce a mathematical requirements document using [5] as a model. This document, which also used tabular representations of mathematical functions, was reviewed by nuclear safety experts.

It was also agreed that precise program documentation would be produced and used as the basis for an inspection procedure. Because the correctness of this code was considered vital to the safety of the plant, AECB, Ontario Hydro, and Atomic Energy of Canada Ltd. (AECL), were able to train approximately 60 engineers to produce and review tabular documentation. The inspectors had to identify program components and document them. The resulting tables were then used as the input to an open inspection process. Each table was presented to a review group and the authors had to demonstrate that it was a correct description of the code. Generally, this involved going through the table on a column-by column, row-by-row basis. The tabular organization was extremely valuable because it made it easy to take breaks (the process went on for months) without losing context or continuity.

In addition to demonstrating that the tabular documentation of the programs accurately described the code, it was necessary to demonstrate that the tables describing the code described behavior that satisfied the requirements represented by tables in the requirements document. Generally, this involved a step-by-step transformation of one table until it matched the corresponding table in the other requirements document. The transformations were not mechanical; their correctness depended on properties of the functions used in the expressions and required human insight. Again, the tabular organization proved essential to allowing human beings with finite attention spans to compare two very detailed documents

In the Darlington work the documentation was not formally organized into displays. This led to a lot of page flip-

[18] While line-counts are notoriously subjective, an outside expert ([11]) estimates the programs as containing about 2500 lines of FORTRAN and Pascal, plus about the same amount of code in assembler.

ping during the inspection process. Technological limitations also prevented us from using some of the notation in this paper. The work was done without the precise definitions in this paper and demonstrated the need for those definitions. In the Darlington work, for example, we did not use quantifiers and this led to problems when dealing with arrays in the program.

The methods described in this paper result from our reflection on the Darlington experience. The notations used here are the ones that we now believe we should have used in Darlington. The notation presented here has been used in more academic experiments including work done at Warsaw University and at McMaster University. Our conclusions are supported by experience gained when the Display Method was applied to examples larger than those presented in this paper (e.g. a simple data base) and implemented in different programming languages (Sun Pascal, Turbo Pascal, FORTRAN, C), cf. [26]. One interesting aspect of this McMaster University work was that it was done by an undergraduate with no prior exposure to formal methods or mathematical logic. He was able to document and repair a FORTRAN program that had been frustrating its owners in their attempts to repair it for many months. Our success did not surprise us, but it surprised the owners of the FORTRAN program who had reluctantly concluded that the program could not be salvaged.

The extensive experience gained in the Darlington work, and in subsequent uses of the method, has revealed where users of these ideas spend their time. We have found that much of the Engineer's time was spent on tasks that could be done by relatively simple tools. This work has led to tool projects at McMaster University, the Université du Québec à Hull, and Warsaw University, which will be described in the next Section.

7 Concluding remarks

We base this method on a very simple idea. Programs can only be understood in small chunks, so they should always be presented in small pieces. Each presentation must be complete in itself so that it can be studied without looking at the others. However, one can not follow this simple precept without finding a way to express the connections between the small sections. It does no good to have a collection of small programs, each one of them correct, if they do not fit together to make a large correct program. This observation led us to use a relational/functional model, both to specify the requirements that a program must meet, and to describe the behavior of a given program. While we found that conventional mathematical concepts were theoretically sufficient to describe these relations, conventional notation resulted in complex expressions that were hard to parse and understand. This led us to introduce a tabular notation that allowed us to describe the programs in a more readable manner. Without this notational progress, the original simple idea would not be as practical.

We began our work on the assumption that we were studying a method of program presentation. It soon became clear that the method was also a way of developing programs. Programs that had been developed before we began to document them, were found to have defects that became obvious when we started to present them in displays. Documenting programs using the display method can result in significant improvements in the quality of the program.

One advantage of this method is that one can speed up a review by employing more reviewers. The displays do not have to be reviewed in any special order and can be reviewed in parallel because they are independent. Even more important, if an error is found in Part 2 of a display, that part can be changed without necessitating modifications to any other displays unless Part 3 is changed. If we do find it necessary to change Part 3 of a display, other displays will have to be changed but we will know exactly which displays must be revised and checked.

The package of ideas that we have presented has proven valuable, but we believe that tool support is needed to make it practical for "everyday" programs. With current tools, it takes an excessive amount of effort to make sure that our expressions are syntactically correct and to achieve neat formatting. Moreover, it requires a high degree of discipline to perform simple checks on the displays, and to make sure that the specifications that are "copied" from the bottom of one display to the top of another are, and remain, identical. Checking lexicon entries requires annoying page-flipping or frustrating delays on the screen. Assembling the program segments to produce executable code by hand is also a time-consuming process in which it is easy to introduce careless errors.

We believe that the situation can be ameliorated by building a set of tools that are designed to support this method of program development and documentation. We envision a system in which the central window presents a display, and other windows provide the relevant lexicon entries. In such a system, the "copying" of the specifications would be automatic and it would be impossible to change one without changing the other. The system would be capable of performing a completeness check and would remind us of specifications that could be found in Part 3 of one display but were not yet developed as Part 1 of another. Checking correctness remains a task for humans. We now have a prototype of such a tool. Other tools would provide syntactic and semantic checks and help us to format the displays. Work on direct support of the Display Method is being carried out at both McMaster University and Warsaw University. At the Université du Québec à Hull editors to support other types of formal documentation have been completed.

A system of this sort would be extremely valuable for people who develop software and even more valuable for those who maintain software products. It would be valuable even without any verification capability, but a simple theorem prover would allow us to make basic checks on the tables. In the future, documentation in this style could be used as input to more sophisticated provers. The information necessary for verification is present in these documents.

Because the documentation is mathematical in nature, it can be used to support testing. The tabular representations can be converted to "oracles", i.e. programs that evaluate the results of tests. If a program is tested against programs generated from it's documentation, developers are more likely to keep the program and documentation consistent. Work of this sort is described in [25].

Tools to make it easier to produce tabular representations of functions and relations in any kind of documentation are being studied and developed at McMaster University.

If readers take the time to compare our presentation of the problem of the Dutch National Flag with Dijkstra's original proposal [2] they will see the benefit of our approach. Dijkstra's presentation, though very illuminating, is dangerously unclear. Although he shows great discipline in developing the small program fragments that are presented in the text, he relies on informal discussions to describe how these are to be assembled into a complete working program. Four essential lines of program text in our solution cannot be found in the program fragments in the original version. Three of these lines are implied by an easily overlooked English sentence in Dijkstra's discussion of the program development. The fourth covers a simple case that seems to have been overlooked because the complete program structure was never presented. We know of several occasions where readers have been asked to examine the original description of the algorithm and then assemble working Pascal programs. Some readers simply assembled Dijkstra's program fragments - producing programs that were not correct. Others noted the conditions in the English text and produced correct programs. We consider Dijkstra's description to be unclear; some have argued that it is wrong[19]. While no method guarantees error-free programs, we believe that the use of the Display Method with careful reviews of each display, makes such errors much less likely.

The problem of the Dutch National Flag reveals one of the limitations of our specification method. LD-relations, like predicate transformers and pre/post conditions, are unable to express the fact that the program is only permitted to inspect the contents of a bucket once. Relational methods limit the final state of the program, but the number of "LOOK" operations that have been carried out is not reflected in the final state with the data structure given. The definition of the buckets abstraction could easily be modified to distinguish between inspected and uninspected buckets, but this would be modifying the data structure only to make the specification easier.

The binary search example illustrates the subtle ways in which programming language restrictions can affect the documentation. In Display 2 we had to introduce "med" but, because we were using Pascal, this variable's declaration should have been included in Display 1. If we had been using Pascal's predecessor, Algol 60, the declaration could

[19] Dijkstra advised against bothering to assemble the final program, apparently because there was no need to look at it.

have been made where it was needed and kept local to the block in which it was used.

Acknowledgements

Dr. Gordon Stuart, then a Ph.D. student at the University of Victoria, contributed to the early development of these ideas. Wm. Wadge helped to clarify the concepts of LD-relations. Graydon Saunders worked on a diagrammatic form of displays for a mythical programming language that we call DAD. The many employees and consultants of AECB, AECL, and Ontario Hydro who applied some of these ideas to the review of the software for the Darlington Nuclear Plant [22] helped us by showing what was needed to turn some academic conceptions into a more mature technology. Mr. P. Filip Sawicki's work on his MSc Thesis [26] was an important step in exploration of the Display Method. We are grateful to David Weiss and to the many people who were kind enough to offer us comments on earlier versions of this paper. An excruciatingly detailed review by referee #4 helped us to clarify our explanations in many ways. The editor's final suggestions were unusually helpful.

This research was primarily funded by the Telecommunications Research Institute of Ontario (TRIO). Other support was provided by the Canadian Institute of Telecommunications Research (CITR), by the State Committee for Scientific Research in Poland (KBN), by Digital Equipment's European External Research Programme (EERP) and by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Dijkstra, E.W., "The Structure of the 'THE' Multiprogramming System", *Communications of the ACM*, Vol. 11, No. 5, May 1968, pp. 341-346.
2. Dijkstra, E.W., *Discipline of Programming*, Prentice-Hall, 1976.
3. Floyd, R.W., "Assigning Meanings to Programs", *Proceedings of the Symposium of Applied Mathematics*, Vol. 19, 1968. Also in: Schwartz, J.T. (editor), *Mathematical Aspects of Computer Science*, American Mathematical Society, 1967, pp. 19-32.
4. Hehner, E.C.R., "Predicative Programming, Part 1", *Communications of the ACM*, Vol. 27, No. 2, February 1984, pp. 134-143.
5. Heninger, K.L., Kallander, J., Parnas, D.L., Shore, J.E., "Software Requirements for the A-7E Aircraft", *NRL Memorandum Report 3876*, United States Naval Research Lab., Washington D.C., November 1978, 523 pp.
6. Heninger, K.L., "Specifying Software Requirements for Complex Systems: New Techniques and their Application", *IEEE Transactions Software Engineering*, Vol. SE-6, No. 1, January 1980, pp. 2-13
7. Hoare, C.A.R., "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, Vol. 12, No. 10, October 1969, pp. 576-580.]
8. Iglewski, M., Madey, J., Parnas, D.L., Kelly P. C., "Documentation Paradigms", *CRL Report* 270, McMaster University, CRL, Telecommunications Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada; July 1993, 45 pp.
9. Jensen, K., Wirth, N., "Pascal User Manual and Report", *Lecture Notes in Computer Science*, Vol. 18, New York, Springer-Verlag, 1974 (second corrected edition 1976).
10. Jones, C. B., *Systematic Software Development Using VDM*, Prentice-Hall, 1986.
11. Leveson, N., *Personal Communication*, 10 September 1994.
12. Majster-Cederbaum, M.E., "A Simple Relation Between Relational and Predicate Transformer Semantics for Nondeterministic Programs", *Information Processing Letters*, Vol. 11, No. 4,5, December 1980, pp. 190-192.
13. Mills, H.D.,"The New Math of Computer Programming", *Communications of the ACM*, Vol. 18, No. 1, January 1975, pp. 43-48.
14. Mills, H.D., "Function Semantics for Sequential Programs", *Proceedings of the IFIP Congress 1980*, North Hol-

land 1980, pp. 241-250.

15. Parnas, D.L., "Information Distributions Aspects of Design Methodology", *Proceedings of the IFIP Congress '71*, Booklet TA-3, 1971, pp. 26-30.

16. Parnas, D.L., "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM*, Vol. 15, No. 12, December 1972, pp. 1053-1058.

17. Parnas, D.L., "A Generalized Control Structure and Its Formal Definition", *Communications of the ACM*, Vol. 26, No. 8, August 1983, pp. 527-581.

18. Parnas, D.L., "Tabular Representation of Relations", *CRL Report* 260, McMaster University, CRL, Telecommunications Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada; October 1992, 17 pp.

19. Parnas, D.L., Madey, J., "Functional Documentation for Computer Systems Engineering. (Version 2)", *CRL Report* 237, McMaster University, CRL, Telecommunications Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada; September 1991, 14 pp.

20. Parnas, D.L., Wadge, W.W., "Less Restrictive Constructs for Structured Programs", *Technical Report* 86-186, Queen's, C&IS, Kingston, Ontario, Canada, October 1986, 16 pp.

21. Parnas, D.L., Wang, Y., "The Trace Assertion Method of Module Interface Specification", *Technical Report* 89-261, Queen's, C&IS, Telecommunications Research Institute of Ontario (TRIO), Kingston, Ontario, Canada, October 1989, 39 pp. (*Available from McMaster University*).

22. Parnas, D.L., Asmis, G.J.K., Madey, J., "Assessment of Safety-Critical Software in Nuclear Power Plants", *Nuclear Safety*, Vol. 32, No. 2, 1991, pp. 189-198.

23. Parnas, D. L., Clements, P. C., Weiss, D. M., "The Modular Structure of Complex Systems", *IEEE Transactions on Software Engineering*, March 1985, Vol. SE-11 No. 3, pp. 259-266.

24. Parnas, D.L., Madey, J., Iglewski, M., "Formal Documentation of Well-Structured Programs", *CRL Report* 259, McMaster University, CRL, Telecommunications Research Institute of Ontario (TRIO), Hamilton, Ontario, Canada; September 1992, 37 pp.

25. Peters, D., Parnas, D. L. "Generating a Test Oracle from Program Documentation", published in *Proceedings of the International Symposium on Software Testing and Analysis*, August 17 - 19, 1994.

26. Sawicki, P.F., "Analiza metody SPS(R) weryfikacji programow" (*in Polish*), ["An Analysis of the SPS(R) Method of Program Verification"], MSc Thesis, Warsaw University, Institute of Informatics, 1992, 124 pp.

27. Wirth, N., "Program Development by Stepwise Refinement", *Communications of the ACM*, Vol. 14, No. 4, April 1971, pp. 221-227.

28. van Schouwen, A. J., "The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems", *Technical Report* 90-276, Queen's, C&IS, Telecommunications Research Institute of Ontario (TRIO), Kingston, Ontario, Canada, May 1990, 93 pp. (*Available from McMaster University*).

29. van Schouwen, A. J., Parnas, D. L., Madey, J., "Documentation of Requirements for Computer Systems", presented at *RE '93 IEEE International Symposium on Requirements Engineering*, San Diego, CA, 4 - 6 January, 1993.

Appendix A

"Binary search" example presented on displays

The description of the problem and the discussion of the solution were given in Section 5.1. What follows is the formal documentation for the complete solution.

DISPLAY 1

Display 1 Specification

Find(x, A, j, present)

$R_0(.) = ((1 \le n) \wedge \forall i\, [\,(1 \le i < n) \Rightarrow ('A[i] \le 'A[i+1])\,]) \Rightarrow$

	$\exists i\, [\,(1 \le i \le n) \wedge ('A[i] = 'x)\,] =$	
	true	*false*
j' |	$'A[j'] = 'x$	*true*
present' =	true	false

$\wedge\, NC(x, A)$

Display 1 Program

Procedure declaration:

```
procedure Find(e : integer; V : vector; var index : integer; var found : Boolean);
var low, high : integer;
begin
  Initialization; Body
end {Find}
```

Display 1 Specifications of Invoked Programs

Initialization	external variables: e, V, found, low, high

$R_1(.) = (low' = 1) \wedge (high' = n) \wedge (found' = false) \wedge NC(e, V)$

(on Display 4)

Body	external variables: e, V, index, found, low, high

(on Display 2)

$R_2(.) =$

$(('low \le 'high) \wedge (\,'found = false) \wedge \forall i\, [\,('low \le i < 'high) \Rightarrow ('V[i] \le 'V[i+1])\,]) \Rightarrow$

	$\exists i\, [\,('low \le i \le 'high) \wedge ('V[i] = 'e)\,] =$	
	true	*false*
index' |	$'V[index'] = 'e$	*true*
found' =	true	false
low' |	*true*	*true*
high' |	*true*	*true*

$\wedge\, NC(e, V)$

END OF DISPLAY 1

DISPLAY 2

Display 2 Specification

Body	external variables: e, V, index, found, low, high

(from Display 1)

R₂(,) =

(('found = false) ∧ ('low ≤ 'high) ∧ ∀i [('low ≤ i < 'high) ⇒ ('V[i] ≤ 'V[i+1])]) ⇒

		∃i [('low ≤ i ≤ 'high) ∧ ('V[i] = 'e)] =	
		true	*false*
index'	∣	'V[index'] = 'e	*true*
found'	=	true	false
low'	∣	*true*	*true*
high'	∣	*true*	*true*

∧ NC(e, V)

Display 2 Program

New variable (to be declared in the embedding block): var med : integer;

Program statements:

```
{Body}
while not found and (low ≤ high) do begin
  med := (low + high) div 2;
  Test
end
```

Display 2 Specifications of Invoked Programs

Test	external variables: e, V, index, found, low, high, med

(on Display 3)

R₃(,) = ('low ≤ 'med ≤ 'high) ⇒

		'V['med]		
		< 'e	= 'e	> 'e
index'	∣	*true*	index' = 'med	*true*
found'	=	'found	true	'found
low'	=	'med + 1	'low	'low
high'	=	'high	'high	'med – 1

∧ NC(e, V, med)

END OF DISPLAY 2

DISPLAY 3

Display 3 Specification

Test	external variables: e, V, index, found, low, high, med

(from Display 2)

$R_3(.)$ = ('low ≤ 'med ≤ 'high) ⇒

	'V['med]		
	< 'e	= 'e	> 'e
index' |	*true*	index' = 'med	*true*
found' =	'found	true	'found
low' =	'med + 1	'low	'low
high' =	'high	'high	'med – 1

∧ NC(e, V, med)

Display 3 Program

```
{Test}
if V[med] < e then
  low := med + 1
else
  if V[med] > e then
    high := med – 1
  else begin
    index := med;
    found := true
  end
```

Display 3 Specifications of Invoked Programs

Empty

END OF DISPLAY 3

DISPLAY 4

Display 4 Specification

Initialization	external variables: e, V, found, low, high
$\mathbf{R}_1(,) = (low' = 1) \wedge (high' = n) \wedge (found' = false) \wedge NC(e, V)$	

(from Display 1)

Display 4 Program

```
{Initialization}
low := 1;
high := n;
found := false;
```

Display 4 Specifications of Invoked Programs

<u>Empty</u>

END OF DISPLAY 4

LEXICON

A. Pascal external definitions and declarations

```
const n = n; {literal integer is to be written here}
type vector = array[1..n] of integer;
var x, j : integer; A : vector; present : Boolean;
```

INDEX

Name	Used in
A	D0, $D1_1$, L_A
Body	$D1_{2,3}$, $D2_{1,2}$
c	$D1_{2,3}$, $D2_{1,3}$, D3, $D4_1$
Find	$D1_{1,2}$
found	$D1_{2,3}$, D2, D3, D4
high	$D1_{2,3}$, D2, D3, D4
index	$D1_{2,3}$, $D2_{1,3}$, D3, D4
Initialization	$D1_{2,3}$, D4
j	D0, $D1_1$, L_A
low	$D1_{2,3}$, D2, D3, D4
med	$D2_{2,3}$, D3
n	D0, $D1_{1,3}$, D4, L_A
present	D0, $D1_1$, L_A
Test	$D2_{2,3}$, D3
V	$D1_{2,3}$, $D2_{1,3}$, D3, $D4_1$
vector	D0, $D1_2$, L_A
x	D0, $D1_1$, L_A

Legend:

• D0			denotes the introduction,
• Di,	i=1,2, ...		denotes Display i,
• Di_j,	i=1,2, ...,	$j \in \{1,2,3\}$	denotes Display i, part Pj,
• $Di_{j,k}$,	i=1,2, ...,	$j,k \in \{1,2,3)$	denotes Display i, parts Pj and Pk,
• L_x,	x=A,B, ...		denotes the lexicon, part x.

Appendix B

"Dutch national flag" example presented on displays

The description of the problem (based on [2], chapter 14), and the discussion of the solution were given in Section 5.2. What follows is the formal documentation for the complete solution. The notation used to specify "buckets" is explained in [8] and [21].

DISPLAY 1

Display 1 Specification

DutchFlag	external variable: v
$R_0(.) = \mathit{flag}(v') \wedge \mathit{same\_colors}('v,v')$	

Display 1 Program

Procedure declaration:

```
procedure DutchFlag;
var r, w, b : integer;
begin
  r := 1; w := N; b := N;
  Rearrange(r, w, b)
end {DutchFlag}
```

Display 1 Specifications of Invoked Programs

Rearrange(r, w, b)	external variable: v
$R_1(.) = (('r = 1) \wedge ('w = N) \wedge ('b = N))$ $\Rightarrow$ $(\mathit{partial\_flag}(v',r',w',b') \wedge (w' = r'-1) \wedge \mathit{same\_colors}('v,v'))$	

(on Display 2)

END OF DISPLAY 1

DISPLAY 2

Display 2 Specification

Rearrange(r, w, b)	external variable: v
$R_1(,) = (('r = 1) \wedge ('w = N) \wedge ('b = N))$ $\Rightarrow$ $(\textit{partial\_flag}(v',r',w',b') \wedge (w' = r'-1) \wedge \textit{same\_colors}('v,v'))$	

(from Display 1)

Display 2 Program

Procedure declaration:

```
procedure Rearrange(var r, w, b : integer);
begin
  while w ≥ r do
    Decrease(r, w, b)
end {Rearrange}
```

Display 2 Specifications of Invoked Programs

Decrease(r, w, b)	external variable: v
$R_1(,) = (\textit{partial\_flag}('v,'r,'w,'b) \wedge ('r \leq 'w))$ $\Rightarrow$ $(\textit{partial\_flag}(v',r',w',b') \wedge ((w'-r') < ('w-'r)) \wedge$ $\textit{same\_colors}('v,v'))$	

(on Display 3)

END OF DISPLAY 2

DISPLAY 3

Display 3 Specification

Decrease(r, w, b)	external variable: v

$$R_2(.) = (\mathit{partial\_flag}('v,'r,'w,'b) \wedge ('r \le 'w)) \Rightarrow (\mathit{partial\_flag}(v',r',w',b') \wedge ((w'-r') < ('w-'r)) \wedge \mathit{same\_colors}('v,v'))$$

(from Display 2)

Display 3 Program

Procedure declaration:

```
procedure Decrease(var r, w, b : integer);
var colr, colw : color;
begin
  IncR;
  if r < w then begin
    DecW;
    UseColw
  end {if};
  UseColr
end {Decrease}
```

Display 3 Specifications of Invoked Programs[1]

DecW	external variables: v, r, w, b, colr, colw

(on Display 4)

$$R_3(.) = \mathit{partial\_flag}('v, 'r, 'w, 'b) \wedge ('r < 'w) \Rightarrow \mathit{partial\_flag}(v', r', w', b') \wedge$$

	true
w' |	$(r' < w') \wedge$ $(((r'+1) < w') \Rightarrow (v'_{w'} \neq \mathrm{while}))$
colw' =	$v'_{w'}$

$\wedge$ NC(v,r,b,colr)

Display to be continued

[1] *Note:* v_i is defined in part C of the lexicon.

IncR	external variables: v, r, w, b, colr, colw

(on Display 5)

$R_4(.) = \mathit{partial\_flag}(`v, `r, `w, `b) \land (`r \le `w) \Rightarrow$

$\mathit{partial\_flag}(v', r', w', b') \land$

	true
r' |	$(r' \le w') \land$ $((r' < w') \Rightarrow (v'_{r'} \ne \text{red}))$
colr' =	$v'_{r'}$

$\land\ NC(v,w,b,colw)$

UseColr	external variables: v, r, w, b, colr, colw

(on Display 6)

$R_5(.) = \mathit{partial\_flag}(`v, `r, `w, `b) \land (`colr = `v_{`w}) \land$

$(`r \le `w) \land ((`r < `w) \Rightarrow (`colr \ne \text{red}))$

$\Rightarrow$

$\mathit{partial\_flag}(v', r', w', b') \land \mathit{same\_colors}(`v,v') \land$

	`colr =		
	red	white	blue
r' |	$r' = `r + 1$	NC(r)	NC(r)
w' |	NC(w)	$w' = `w - 1$	$w' = `w - 1$
b' |	NC(b)	NC(b)	$b' = `b - 1$

$\land\ NC(colr,colw)$

UseColw	external variables: v, r, w, b, colr, colw

(on Display 7)

$R_6(.) = \mathit{partial\_flag}(`v, `r, `w, `b) \land (`colr = `v_{`r}) \land (`colw = `v_{`w}) \land$

$(`r < `w) \land (((`r{+}1) < `w) \Rightarrow (`colw \ne \text{white}))$

$\Rightarrow$

$\mathit{partial\_flag}(v', r', w', b') \land \mathit{same\_colors}(`v,v') \land (v'_{w'} = colr') \land$

	`colw =		
	red	white	blue
r' |	$r' = `r + 1$	NC(r)	NC(r)
w' |	NC(w)	$w' = `w - 1$	$w' = `w - 1$
b' |	NC(b)	NC(b)	$b' = `b - 1$

$\land\ NC(colr,colw)$

END OF DISPLAY 3

DISPLAY 4

Display 4 Specification

DecW	external variables: v, r, w, b, colr, colw

(from Display 3)

$R_3(,) = \mathit{partial\_flag}('v, 'r, 'w, 'b) \wedge ('r < 'w) \Rightarrow$

$\mathit{partial\_flag}(v', r', w', b') \wedge$

	true
w' |	$(r' < w') \wedge$ $(((r'+1) < w') \Rightarrow (v'_{w'} \neq \text{white}))$
colw' =	$v'_{w'}$

$\wedge$ NC(v,r,b,colr)

Display 4 Program

```
{DecW}
colw := LOOK(w);
while (colw = white) and ((r+1) < w) do begin
  w := w-1; colw := LOOK(w)
end
```

Display 4 Specifications of Invoked Programs

Empty

END OF DISPLAY 4

DISPLAY 5

Display 5 Specification

IncR	external variables: v, r, w, b, colr, colw

(from Display 3)

$R_4(,) = \mathit{partial\_flag}({}'v, {}'r, {}'w, {}'b) \wedge ({}'r \le {}'w) \Rightarrow$

$\mathit{partial\_flag}(v', r', w', b') \wedge$

	true
r' |	$(r' \le w') \wedge$ $((r' < w') \Rightarrow (v'_{r'} \ne \mathrm{red}))$
colr' =	$v'_{r'}$

$\wedge\ NC(v,w,b,colw)$

Display 5 Specification

```
(IncR)
colr := LOOK(r);                  { v is an implicit variable used by LOOK }
while (colr = red) and (r < w) do begin
  r := r+1; colr := LOOK(r)
end
```

Display 5 Specifications of Invoked Programs

Empty

END OF DISPLAY 5

DISPLAY 6

Display 6 Specification

UseColr	external variables: v, r, w, b, colr, colw

(from Display 3)

$R_5(,) = \textit{partial\_flag}('v, 'r, 'w, 'b) \wedge ('colr = 'v_{'w}) \wedge$

$('r \leq 'w) \wedge (('r < 'w) \Rightarrow ('colr \neq red))$

$\Rightarrow$

$\textit{partial\_flag}(v', r', w', b') \wedge \textit{same\_colors}('v, v') \wedge$

	'colr =		
	red	white	blue
$r' \mid$	$r' = 'r + 1$	NC(r)	NC(r)
$w' \mid$	NC(w)	$w' = 'w - 1$	$w' = 'w - 1$
$b' \mid$	NC(b)	NC(b)	$b' = 'b - 1$

$\wedge$ NC(colr,colw)

Display 6 Program

```
{UseColr}
case colr of
  red:    r := r+1;
  white:  w := w-1;
  blue:   begin SWAP(w,b); w := w-1; b := b-1 end
end
```

Display 6 Specifications of Invoked Programs

Empty

END OF DISPLAY 6

DISPLAY 7

Display 7 Specification

UseColw	external variables: v, r, w, b, colr, colw

(from Display 3)

$$R_6(,) = \textit{partial\_flag}('v, 'r, 'w, 'b) \wedge ('colr = 'v_{'r}) \wedge ('colw = 'v_{'w}) \wedge$$

$$('r < 'w) \wedge ((('r+1) < 'w) \Rightarrow ('colw \neq white))$$

$$\Rightarrow$$

$$\textit{partial\_flag}(v', r', w', b') \wedge \textit{same\_colors}('v,v') \wedge (v'_{w'} = colr')^{a} \wedge$$

	'colw =		
	red	white	blue
r' |	r' = 'r + 1	NC(r)	NC(r)
w' |	NC(w)	w' = 'w − 1	w' = 'w − 1
b' |	NC(b)	NC(b)	b' = 'b − 1

$\wedge$ NC(colr,colw)

a. The post-condition $v'_{w'} = colr'$ is redundant and has been added for ease of comprehension.

Display 7 Program

```
{UseColw}
case colw of
  red:     begin SWAP(r, w); r := r+1 end;
  white:   w := w−1;
  blue:    begin SWAP(w, b); w := w−1; b := b−1; SWAP(r,w) end
end
```

Display 7 Specifications of Invoked Programs

Empty

END OF DISPLAY 7

LEXICON

A. Auxiliary functions

$card$: set → integer

$card(s) \overset{\text{df}}{=} |\, s \,|$ (i.e. number of elements in the set s)

$flag$: buckets → boolean

$flag(v) \overset{\text{df}}{=} \exists r,b\ [partial\_flag(v,r,r{-}1,b)]$

$partial\_flag$: buckets × integer × integer × integer → boolean

$$\begin{aligned} partial\_flag(v,r,w,b) \overset{\text{df}}{=}\ & (1 \le r) \wedge (r{-}1 \le w) \wedge (w \le b) \wedge (b \le N) \wedge \\ & \forall i\, (1 \le i \le N)\ [\ ((i < r) \Rightarrow (v_i = \text{red})) \wedge \\ & \qquad ((w < i \le b) \Rightarrow (v_i = \text{white})) \wedge \\ & \qquad ((b < i) \Rightarrow (v_i = \text{blue}))\ ] \end{aligned}$$

Note: v_i is defined in part C of this lexicon.

$same\_colors$: buckets × buckets → boolean

$same\_colors(v1,v2) \overset{\text{df}}{=}$

$$\begin{aligned} & (card(\{i \mid (1 \le i \le N) \wedge (v1_i = \text{red})\}) = card(\{i \mid (1 \le i \le N) \wedge (v2_i = \text{red})\})) \wedge \\ & (card(\{i \mid (1 \le i \le N) \wedge (v1_i = \text{white})\}) = card(\{i \mid (1 \le i \le N) \wedge (v2_i = \text{white})\})) \wedge \\ & (card(\{i \mid (1 \le i \le N) \wedge (v1_i = \text{blue})\}) = card(\{i \mid (1 \le i \le N) \wedge (v2_i = \text{blue})\})) \end{aligned}$$

B. Pascal external definitions and declarations

```
const N = {literal non-negative integer}
type color = (red, white, blue);
type buckets =    {vector(N, color) - cf. part C of this lexicon}
var v : buckets;
procedure LOOK(i : integer);
  {cf. part C of this lexicon}
procedure SWAP(i, j : integer);
  {cf. part C of this lexicon}
```

C. vector(n,elem) Module Interface Specification

(0) CHARACTERISTICS

- type specified: vector(n,elem)
- features: single-object, generic
- foreign types: elem, <integer>, <positive_integer>
- generic parameters: n: <positive_integer>, elem

(1) SYNTAX

ACCESS-PROGRAMS

Program Name	Arg#1	Arg#2	Value Type
LOOK	<integer>:V		elem
PUT	<integer>:V	elem:V	
SWAP	<integer>:V	<integer>:V	

(2) CANONICAL TRACES

canonical(T) $\Leftrightarrow$ $T = [PUT(i, e_i)]_{i=1}^{n}$

$\_ = [PUT(i, \_)]_{i=1}^{n}$

EQUIVALENT NOTATION FOR TRACES

Trace	Equivalent notation
v.LOOK(i)	v_i

(3) EQUIVALENCES

T.LOOK(i) $\Rightarrow$ T

T.PUT(i, e) $\Rightarrow$

Condition	Equivalence
$\neg(1 \le i \le n)$	%wrong_index%
$1 \le i \le n$	T1.PUT(i,e).T2 where T=T1.PUT(i,x).T2

T.SWAP(i, j) $\Rightarrow$

<table>
<tr><th colspan="2">Condition</th><th>Equivalence</th></tr>
<tr><td colspan="2">$\neg((1 \le i \le n) \wedge (1 \le j \le n))$</td><td>%wrong_index%</td></tr>
<tr><td rowspan="3">$(1 \le i \le n) \wedge (1 \le j \le n) \wedge$</td><td>$(i < j)$</td><td>T1.PUT(i,x).T2.PUT(j,y).T3
where
T = T1.PUT(i,y).T2.PUT(j,x).T3</td></tr>
<tr><td>$(i = j)$</td><td>T</td></tr>
<tr><td>$(i > j)$</td><td>T1.PUT(j,x).T2.PUT(i,y).T3
where
T = T1.PUT(j,y).T2.PUT(i,x).T3</td></tr>
</table>

(4) RETURN VALUES

Program Name	Argument No	Value
LOOK(i)	Value	e where vector(n,elem) = T1.PUT(i,e).T2

INDEX

Name	Category	Used in
v		D0, $D1_{1,3}$, $D2_{1,3}$, $D3_{1,3}$, $D4_1$, $D5_1$, $D6_1$, $D7_1$, L_B
vector		D0, $L_{B,C}$
w	variable in DutchFlag	$D1_2$
w	formal parameter in Rearrange	$D1_3$, $D2_{1,2}$
w	formal parameter in Decrease	$D2_3$, D3, $D4_{1,2}$, $D5_{1,2}$, $D6_{1,2}$, $D7_{1,2}$
white		D0, $D3_3$, D4, D6, D7, $L_{A,B}$

Legend:

- D0 denotes the introduction,
- Di, i=1,2, ... denotes Display i,
- Di_j, i=1,2, ..., $j \in \{1,2,3\}$ denotes Display i, part Pj,
- D$i_{j,k}$, i=1,2, ..., $j,k \in \{1,2,3\}$ denotes Display i, parts Pj and Pk,
- L_x, x=A,B, ... denotes the lexicon, part x.

Session W1: Natural Language Modeling

Dr. John Sharp
Sandia National Laboratories

Natural Language Modeling

John K. Sharp, PhD
Sandia National Laboratories

This seminar describes a process and methodology that uses structured natural language to enable the construction of precise information requirements directly from users, experts, and mangers. The main focus of this natural language approach is to create the precise information requirements and to do it in such a way that the business and technical experts are fully accountable for the results. These requirements can then be implemented using appropriate tools and technology. This requirement set is also a universal learning tool because it has all of the knowledge that is needed to understand a particular process (e.g., expense vouchers, project management, budget reviews, tax laws, machine function)

Personal accountability for results is established with the expert that is specifying the design and the implementor is accountable for meeting the design requirements. This is done through a systematic procedure based on a common understanding of the requirements and the ability to communicate effectively. In other words, if the craftsman produced the part according to the requirements then he did the correct job. The accountability for form, fit, and function resides with the engineer who created the design. The craftsman is only accountable for meeting the requirements. The center of this accountability process is a communication channel that is completely understood by all of the participants. Natural language modeling processes allow information technology to achieve this same high quality level.

The advantage of this procedure is that it takes an informal, possibly incomplete, possibly redundant, possibly inconsistent and possibly indeterminate description of a user problem and turns it into a precise set of facts and constraints that contain all of the knowledge and business rules that are necessary for completely solving a user problem. The sentences are created and analyzed by the subject matter expert with the analyst being a facilitator or scribe of the knowledge that is created. The expert is fully accountable for the specification and the knowledge can be transformed into desired graphical and textual presentations that become part of the design specification for the implementor.

This seminar will be an overview of the procedure for creating natural language models. Examples will be provided for every step in the procedure. The procedure starts with the subject matter expert verbalizing sentences about the subject area. Placeholders or variables are then assigned within the created sentences. The sentences are then qualified by assigning names to the placeholder and the object. Constraints are then identified and tested. Finally, the results can then be specified in a number of ways, including relational tables. The focus of the seminar shows how low quality initial inputs are turned into high quality requirements that can hold the subject matter expert accountable for the requirements and the implementor accountable for meeting them.

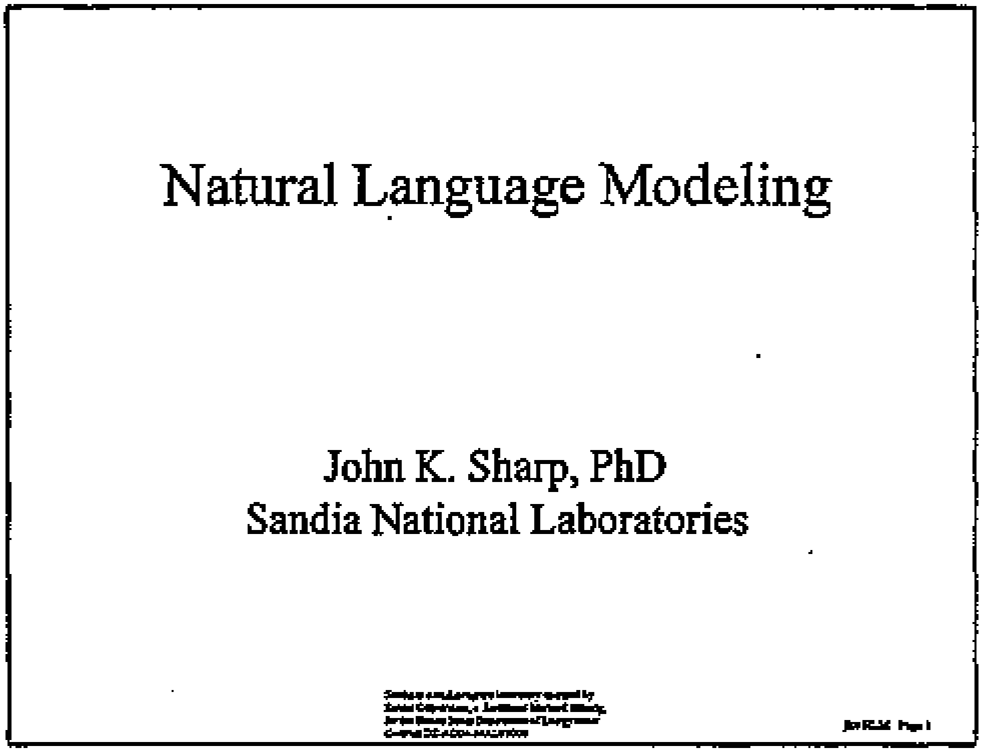
Simple examples will be used throughout the seminar to show how unary, binary and n-ary sentences are analyzed. All possible procedure steps will be presented using examples. Several examples will be used as interactive problems to help attendees understand the procedure.

BIOGRAPHY

John K. Sharp, PhD

John has performed information analysis in various positions at Sandia for fifteen years. He has worked closely with Prof. Shir Nijssen of the Netherlands for several years to establish the procedure to develop and analyze information problems using structured natural language. They are currently finishing a text on this topic. This procedure was originally based on the NIAM (Natural language Information Analysis Methodology) modeling technique. John and Prof. Nijssen have co-chaired two international conferences on natural language modeling. John is also the editor of the international standard on conceptual schemas.

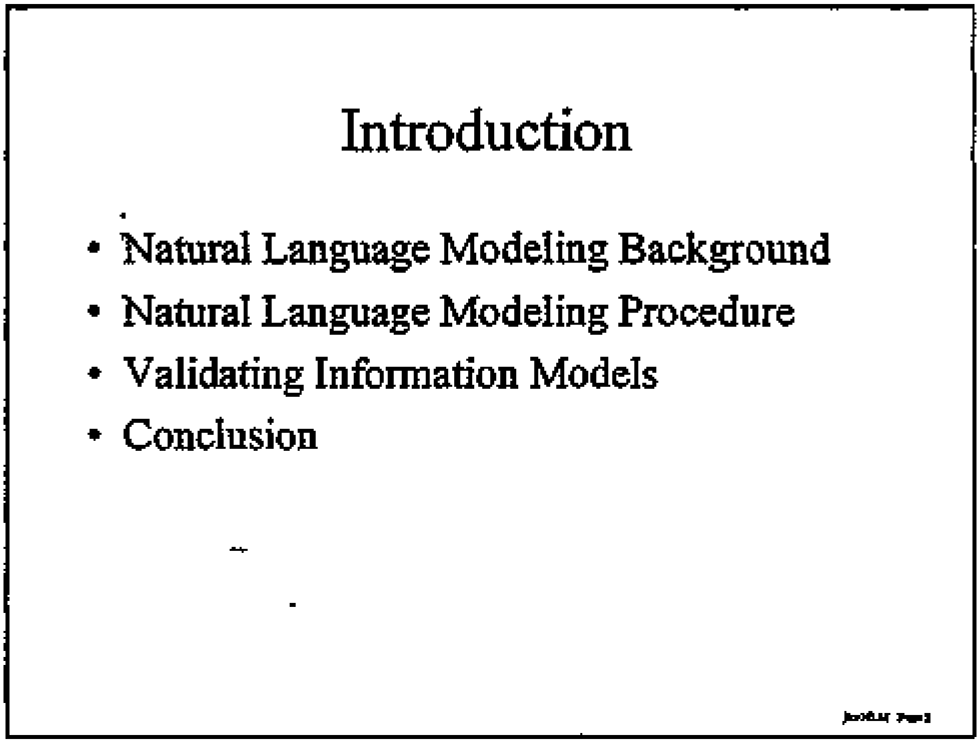
Sandia National Laboratories
Reengineering Center
P.O. Box 5800, MS-0803
Albuquerque, NM 87185-0803
Voice: 505-844-5428
Fax: 505-844-7501
E-mail: jksharp@sandia.gov



Natural Language Modeling

John K. Sharp, PhD
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under
Contract DE-AC04-94AL85000



Introduction

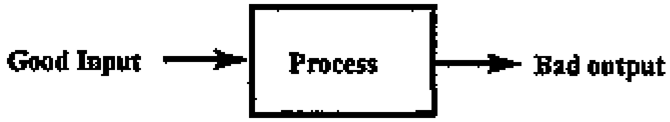
- Natural Language Modeling Background
- Natural Language Modeling Procedure
- Validating Information Models
- Conclusion

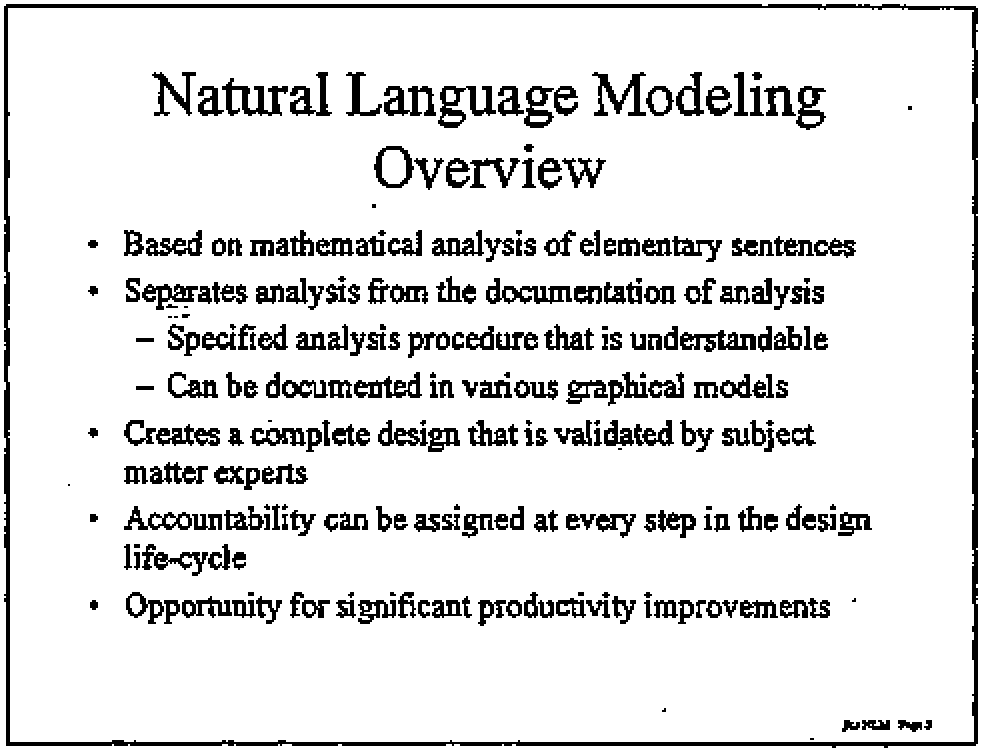
Natural Language Modeling Background

Information Modeling Processes Must Limit Analyst Liability

- Every information analyst must have the ability to make the users/owners fully□ accountable for their information system design

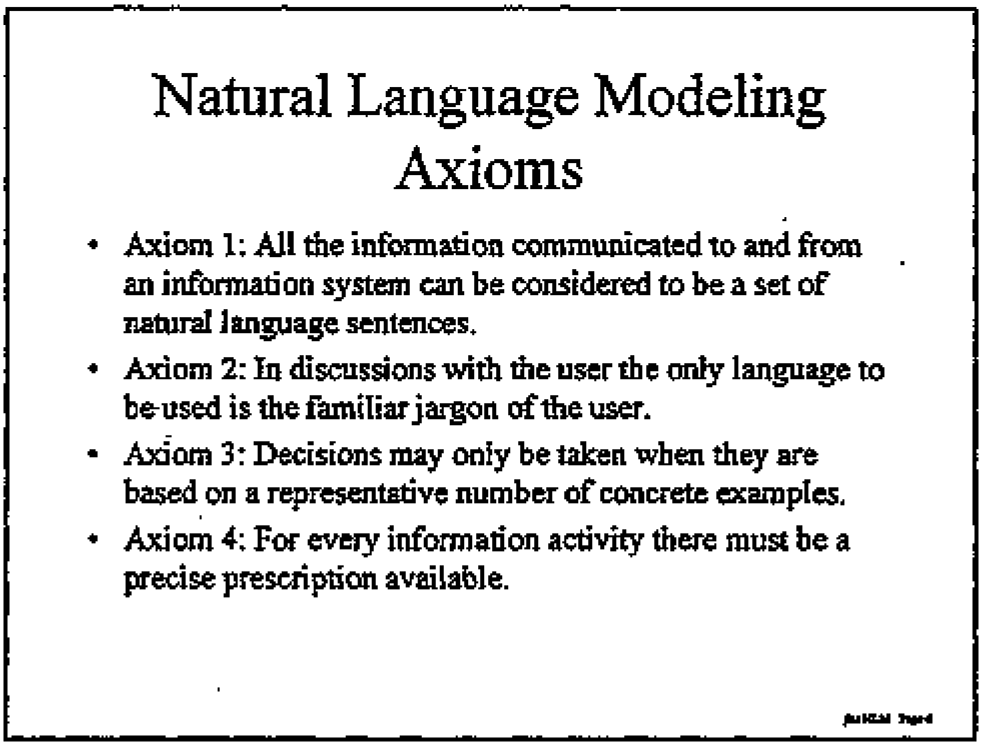
- No more of the following





Natural Language Modeling Overview

- Based on mathematical analysis of elementary sentences
- Separates analysis from the documentation of analysis
  - Specified analysis procedure that is understandable
  - Can be documented in various graphical models
- Creates a complete design that is validated by subject matter experts
- Accountability can be assigned at every step in the design life-cycle
- Opportunity for significant productivity improvements



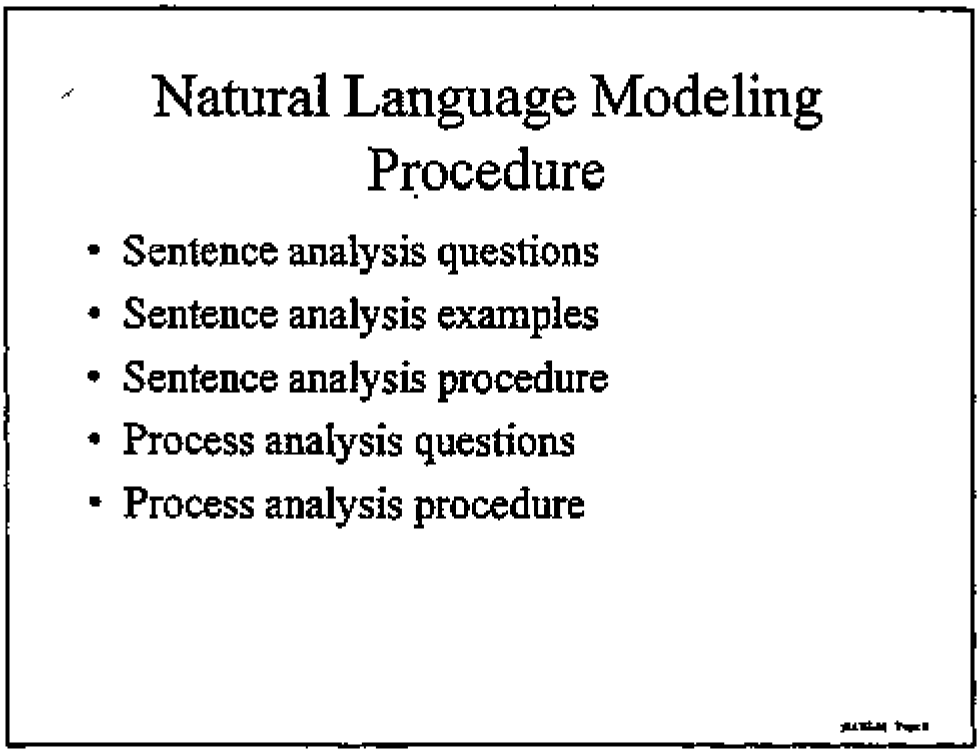
Natural Language Modeling Axioms

- Axiom 1: All the information communicated to and from an information system can be considered to be a set of natural language sentences.
- Axiom 2: In discussions with the user the only language to be used is the familiar jargon of the user.
- Axiom 3: Decisions may only be taken when they are based on a representative number of concrete examples.
- Axiom 4: For every information activity there must be a precise prescription available.

Accountability is available for information technology

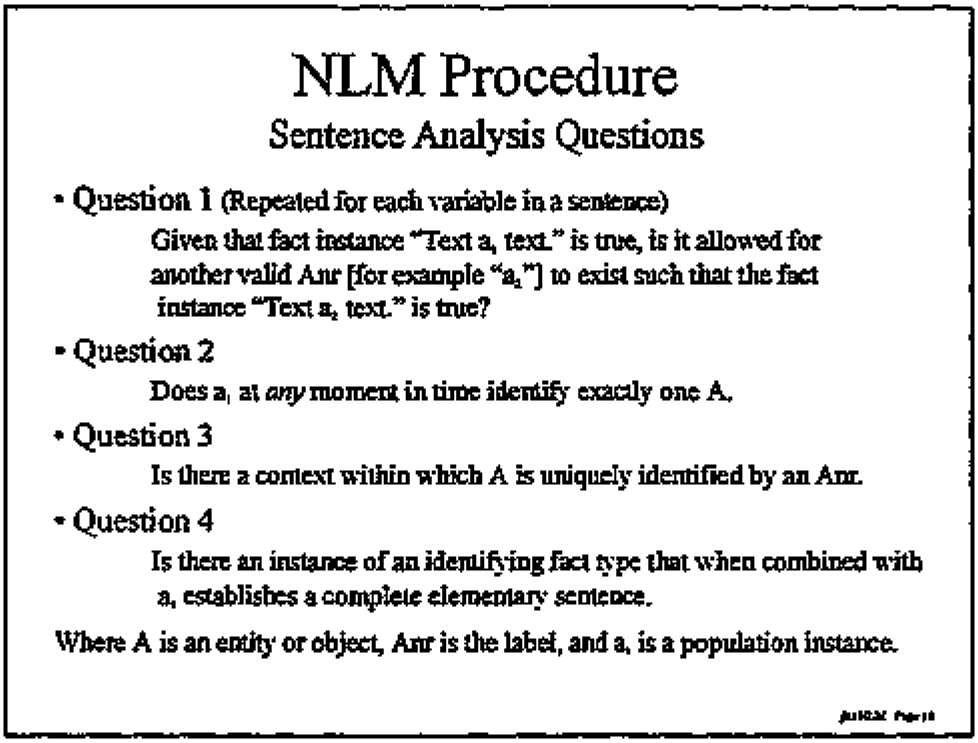
- Subject matter experts become accountable for the requirements.
- Analysts are accountable for a logically complete set of requirements.
- Implementators are accountable for implementing the requirements.
- Management is accountable for the delivery of the application based on validated requirements.

Natural Language Modeling Procedure



Natural Language Modeling Procedure

- Sentence analysis questions
- Sentence analysis examples
- Sentence analysis procedure
- Process analysis questions
- Process analysis procedure



NLM Procedure

Sentence Analysis Questions

- Question 1 (Repeated for each variable in a sentence)

  Given that fact instance "Text a_1 text." is true, is it allowed for another valid Anr [for example "a_2"] to exist such that the fact instance "Text a_2 text." is true?

- Question 2

  Does a_1 at *any* moment in time identify exactly one A.

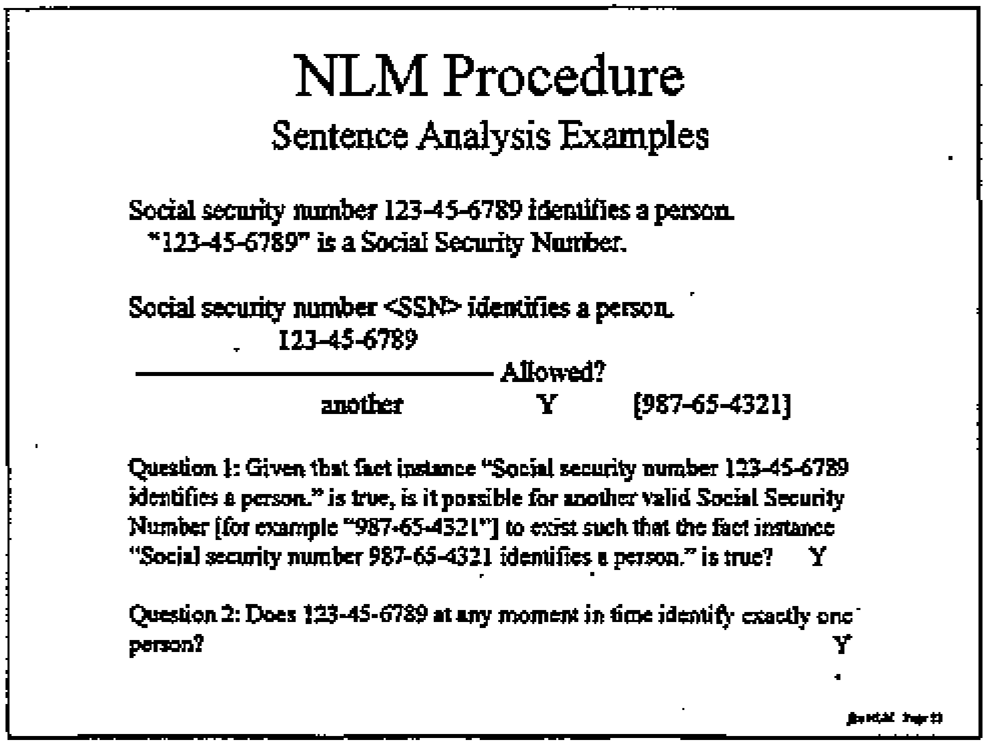
- Question 3

  Is there a context within which A is uniquely identified by an Anr.

- Question 4

  Is there an instance of an identifying fact type that when combined with a_1 establishes a complete elementary sentence.

Where A is an entity or object, Anr is the label, and a_1 is a population instance.



NLM Procedure

Sentence Analysis Examples

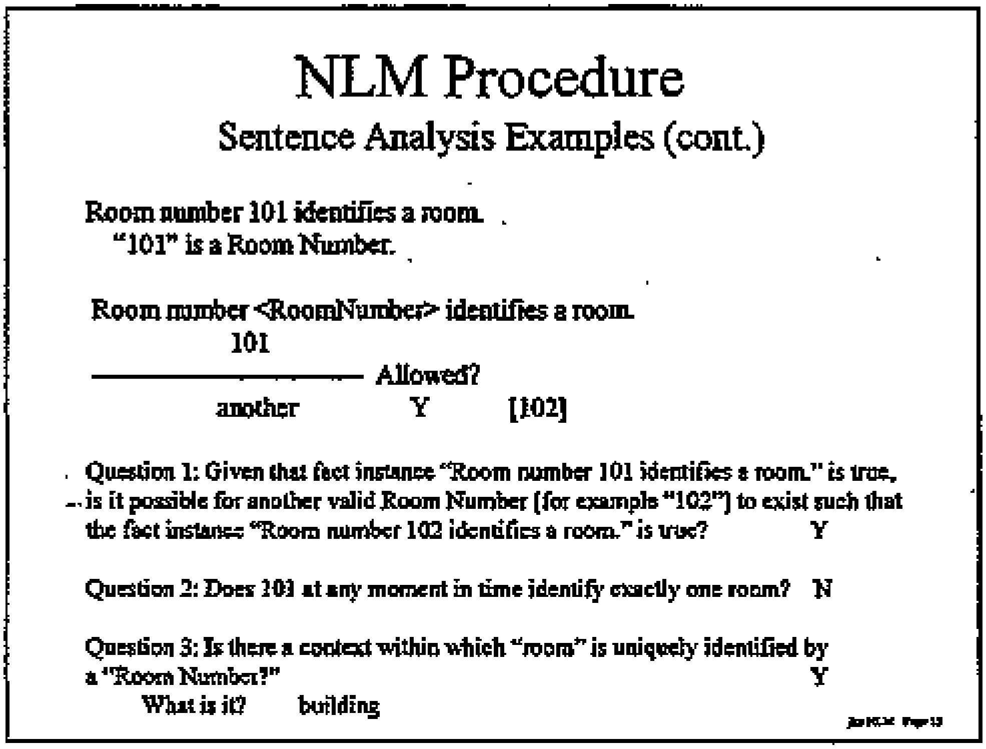
Social security number 123-45-6789 identifies a person.
"123-45-6789" is a Social Security Number.

Social security number <SSN> identifies a person.
123-45-6789

——————————— Allowed?
another Y [987-65-4321]

Question 1: Given that fact instance "Social security number 123-45-6789 identifies a person." is true, is it possible for another valid Social Security Number [for example "987-65-4321"] to exist such that the fact instance "Social security number 987-65-4321 identifies a person." is true? Y

Question 2: Does 123-45-6789 at any moment in time identify exactly one person? Y



NLM Procedure

Sentence Analysis Examples (cont.)

Room number 101 identifies a room.
"101" is a Room Number.

Room number <RoomNumber> identifies a room.
101

——————————— Allowed?
another Y [102]

Question 1: Given that fact instance "Room number 101 identifies a room." is true, is it possible for another valid Room Number [for example "102"] to exist such that the fact instance "Room number 102 identifies a room." is true? Y

Question 2: Does 101 at any moment in time identify exactly one room? N

Question 3: Is there a context within which "room" is uniquely identified by a "Room Number?" Y
What is it? building

NLM Procedure

Sentence Analysis Examples (cont.)

Person name John Smith identifies a person.
"John Smith" is an Person Name.

Person name <Person Name> identifies a person.
John Smith

———————————— Allowed?

another Y [Sue Jones]

Question 1: Given that fact instance "Person name John Smith identifies a person." is true, is it possible for another valid Person Name [for example "Sue Jones"] to exist such that the fact instance "Person name Sue Jones identifies a person." is true? Y

Question 2: Does John Smith at any moment in time identify exactly one person? N

Question 3: Is there a context within which "person" is uniquely identified by a "Person Name?" N

Question 4: Is there an instance of an identifying fact type that when combined with person name establishes a complete elementary sentence? Y

What is it? Social security number 123-45-6789 identifies a person.

NLM Procedure

Sentence Analysis Examples (cont.)

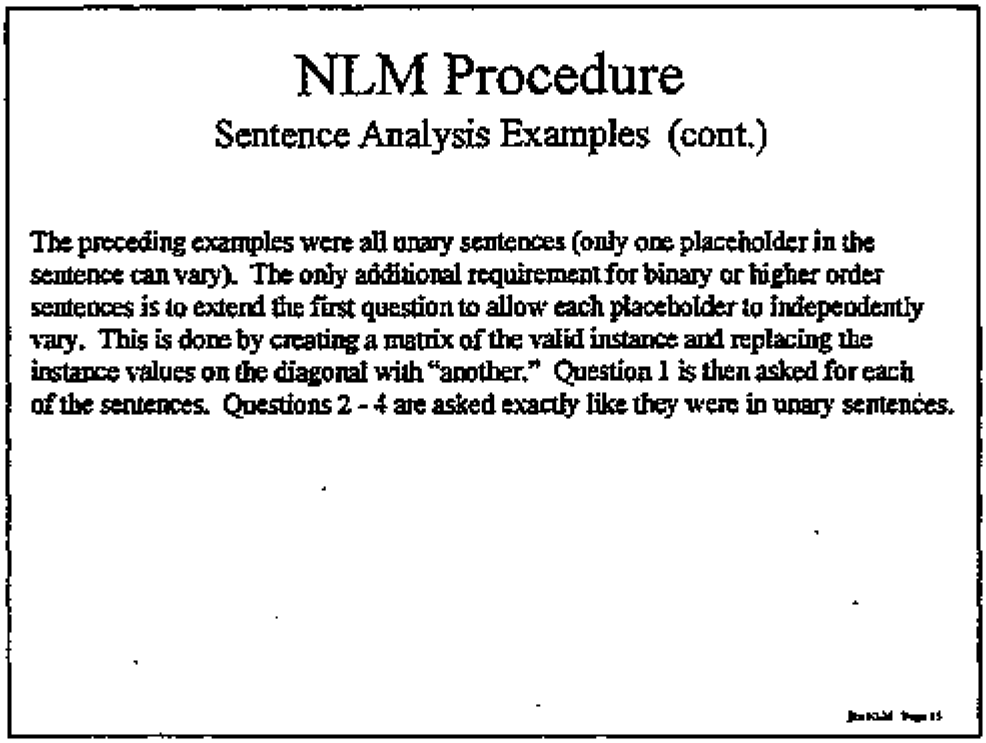
Company name Sandia National Laboratories identifies a company.
"Sandia National Laboratories" is a Company Name.

Company name <CompanyName> identifies a company.
Sandia National Laboratories

———————————— Allowed?

another N [Intel]

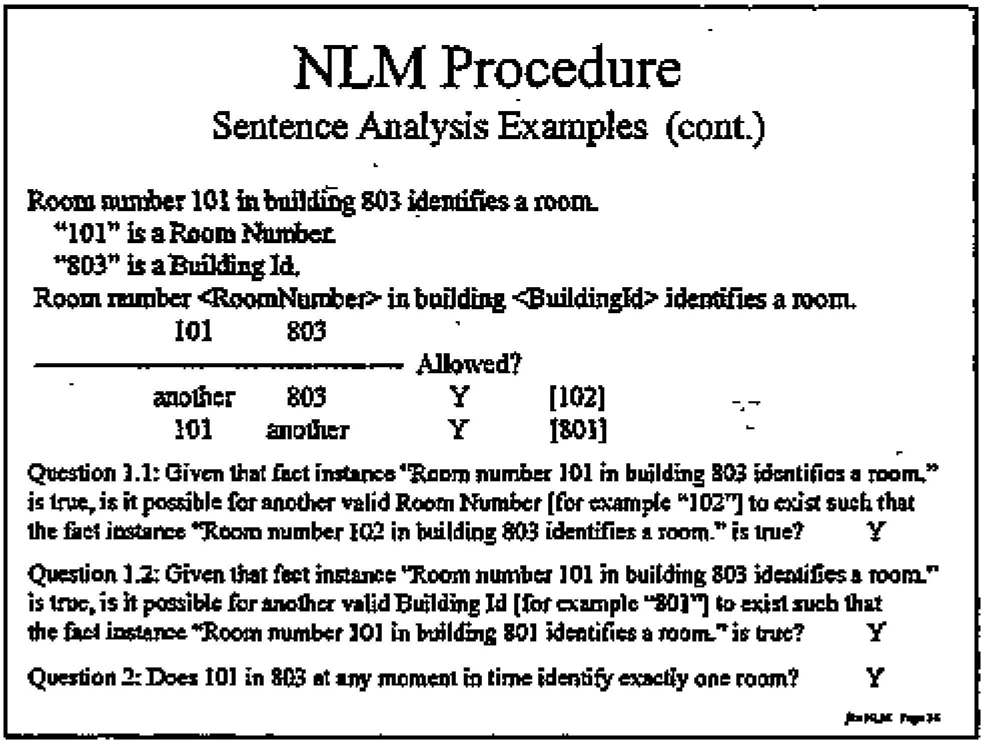
Question 1: Given that fact instance "Company name Sandia National Laboratories identifies a company." is true, is it possible for another valid Company Name [for example "Intel"] to exist such that the fact instance "Company name Intel identifies a company." is true? N



NLM Procedure

Sentence Analysis Examples (cont.)

The preceding examples were all unary sentences (only one placeholder in the sentence can vary). The only additional requirement for binary or higher order sentences is to extend the first question to allow each placeholder to independently vary. This is done by creating a matrix of the valid instance and replacing the instance values on the diagonal with "another." Question 1 is then asked for each of the sentences. Questions 2 - 4 are asked exactly like they were in unary sentences.



NLM Procedure

Sentence Analysis Examples (cont.)

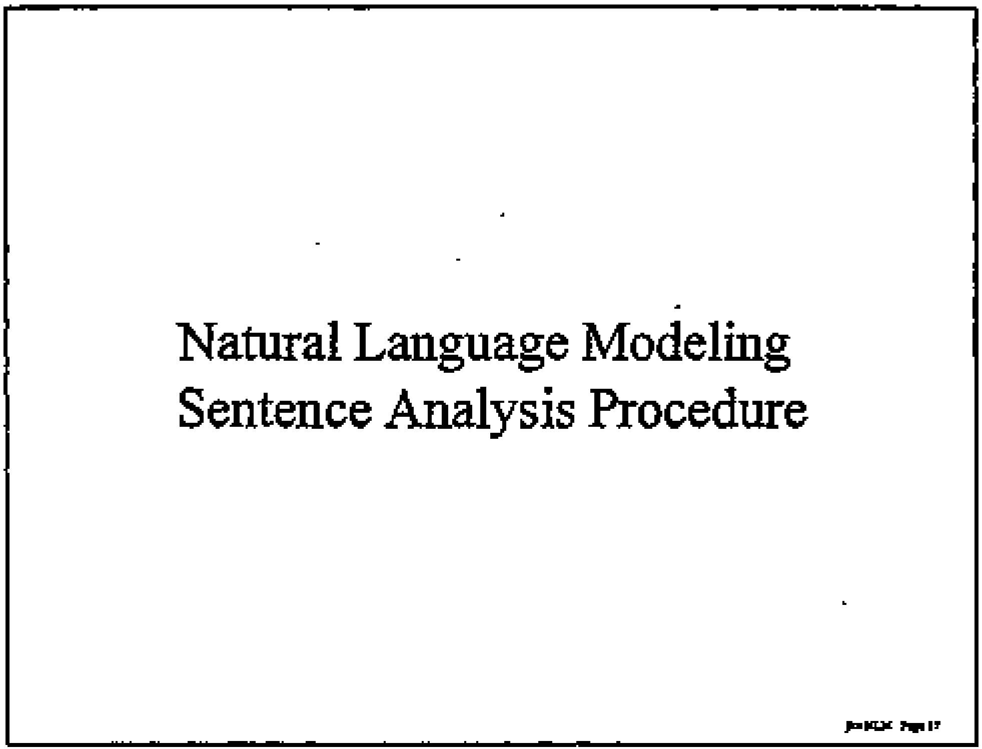
Room number 101 in building 803 identifies a room.
"101" is a Room Number.
"803" is a Building Id.
Room number <RoomNumber> in building <BuildingId> identifies a room.

101	803	Allowed?	
another	803	Y	[102]
101	another	Y	[801]

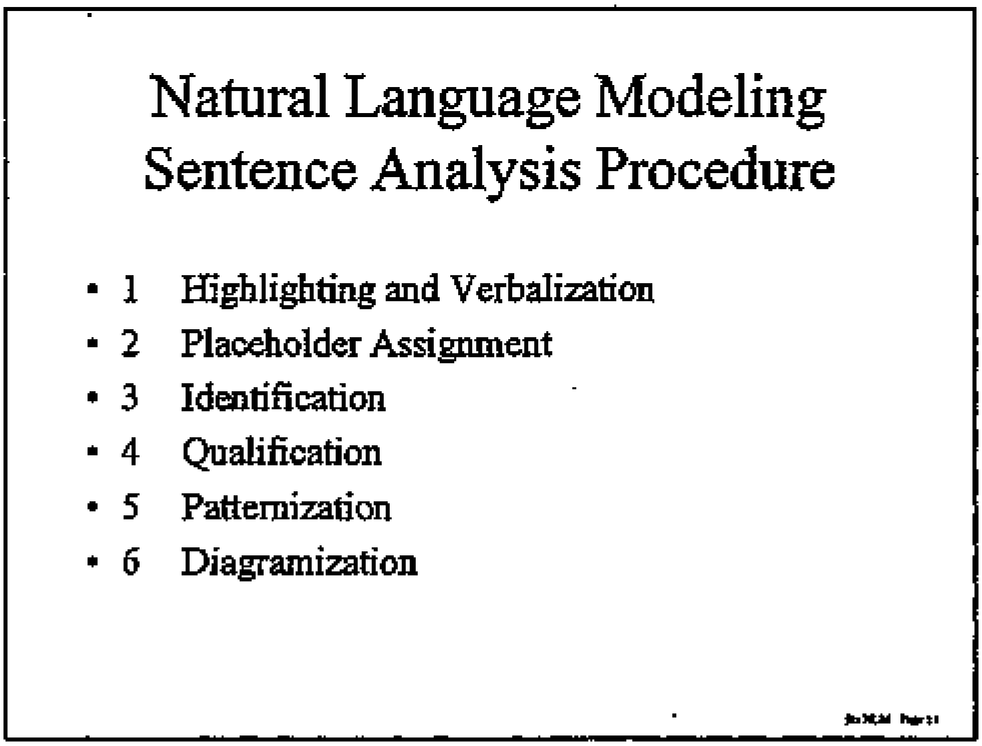
Question 1.1: Given that fact instance "Room number 101 in building 803 identifies a room." is true, is it possible for another valid Room Number [for example "102"] to exist such that the fact instance "Room number 102 in building 803 identifies a room." is true? Y

Question 1.2: Given that fact instance "Room number 101 in building 803 identifies a room." is true, is it possible for another valid Building Id [for example "801"] to exist such that the fact instance "Room number 101 in building 801 identifies a room." is true? Y

Question 2: Does 101 in 803 at any moment in time identify exactly one room? Y



Natural Language Modeling Sentence Analysis Procedure

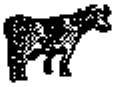


Natural Language Modeling Sentence Analysis Procedure

- 1 Highlighting and Verbalization
- 2 Placeholder Assignment
- 3 Identification
- 4 Qualification
- 5 Patternization
- 6 Diagramization

Example 1
Business Card

Udder, Inc.



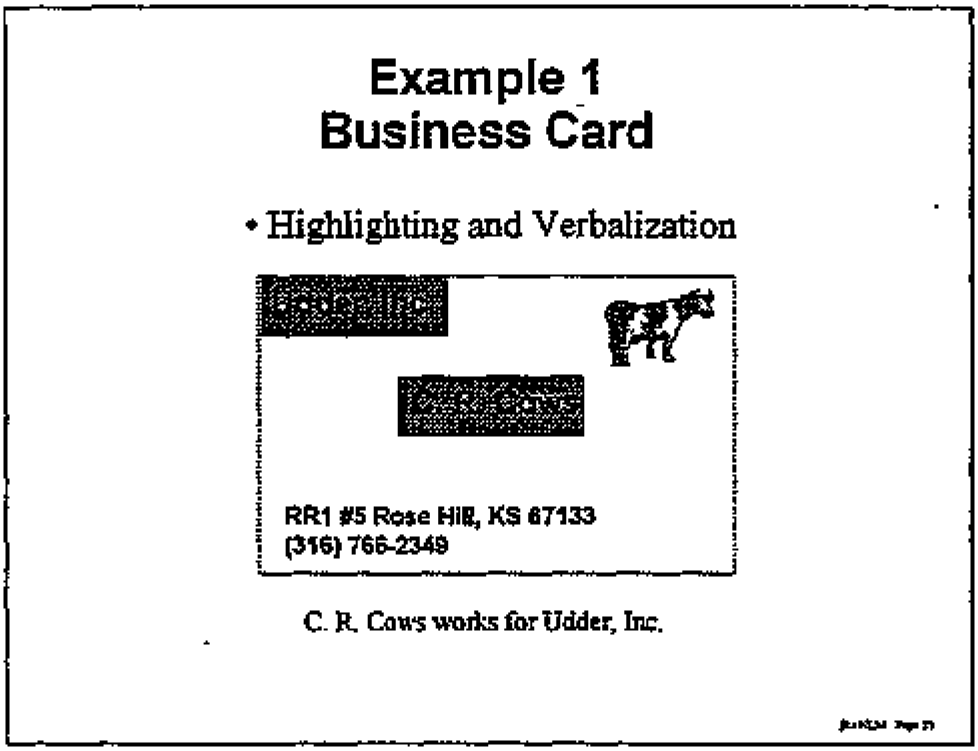
C. R. Cows

RR1 #5 Rose Hill, KS 67133
(316) 766-2349

Example 1
Business Card

Problem statement:

Replace a stack of business cards with an electronic version that provides easier access to the information.



Example 1
Business Card

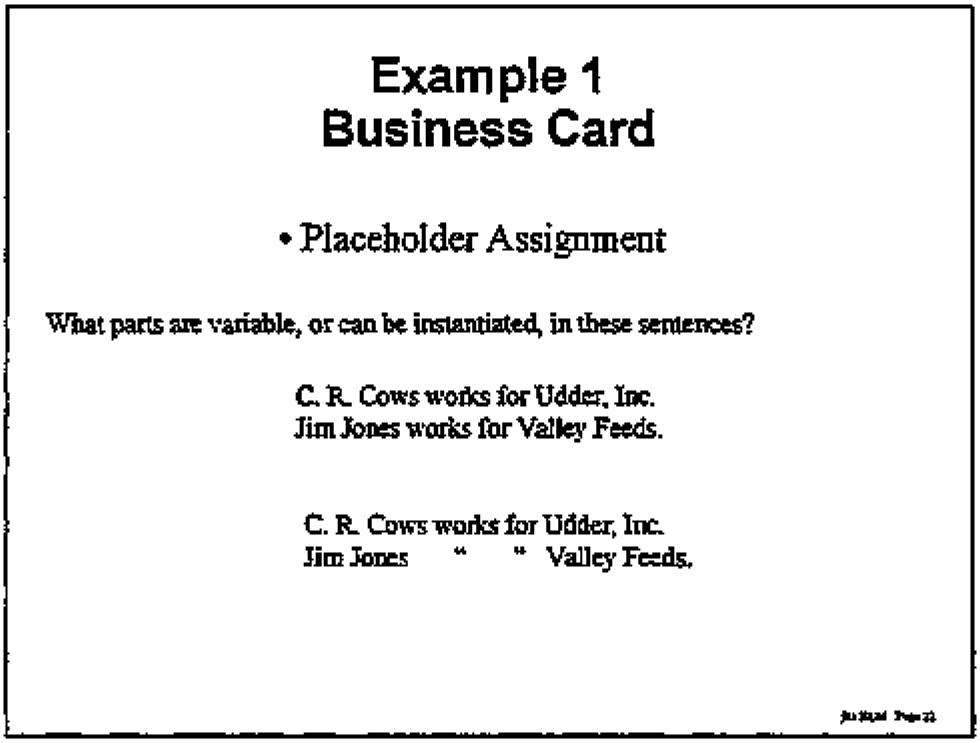
- Highlighting and Verbalization

Udder, Inc.

C. R. Cows

RR1 #5 Rose Hill, KS 67133
(316) 766-2349

C. R. Cows works for Udder, Inc.



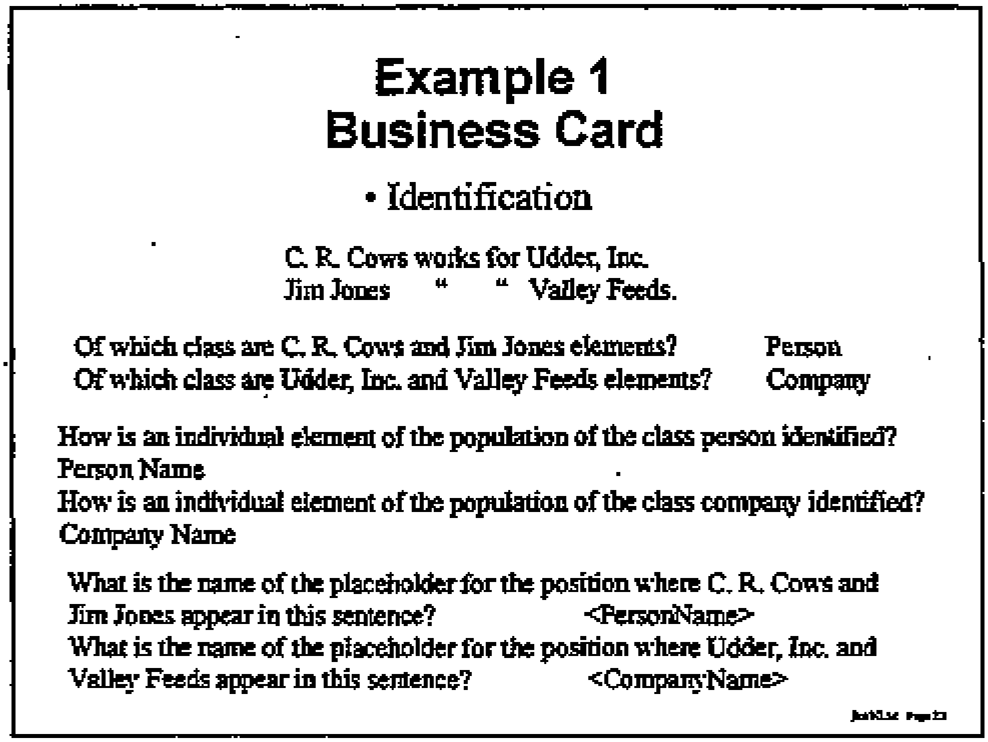
Example 1
Business Card

- Placeholder Assignment

What parts are variable, or can be instantiated, in these sentences?

C. R. Cows works for Udder, Inc.
Jim Jones works for Valley Feeds.

C. R. Cows works for Udder, Inc.
Jim Jones " " Valley Feeds.



Example 1
Business Card

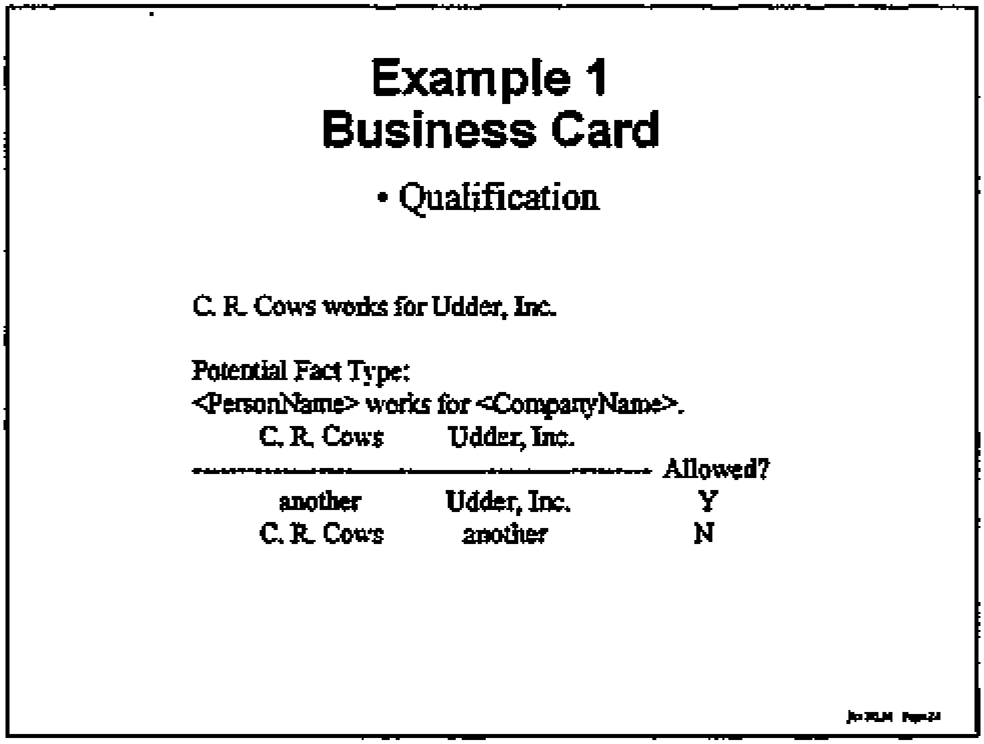
- Identification

C. R. Cows works for Udder, Inc.
Jim Jones " " Valley Feeds.

Of which class are C. R. Cows and Jim Jones elements?	Person
Of which class are Udder, Inc. and Valley Feeds elements?	Company

How is an individual element of the population of the class person identified?
Person Name
How is an individual element of the population of the class company identified?
Company Name

What is the name of the placeholder for the position where C. R. Cows and Jim Jones appear in this sentence? <PersonName>
What is the name of the placeholder for the position where Udder, Inc. and Valley Feeds appear in this sentence? <CompanyName>



Example 1
Business Card

- Qualification

C. R. Cows works for Udder, Inc.

Potential Fact Type:
<PersonName> works for <CompanyName>.

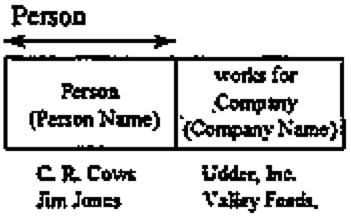
C. R. Cows	Udder, Inc.	Allowed?
another	Udder, Inc.	Y
C. R. Cows	another	N

Example 1
Business Card

- Patternization

Fact type:
FT-1 <PersonName> works for <CompanyName>.

- Diagramization



Person

Person
(Person Name)

works for
Company
(Company Name)

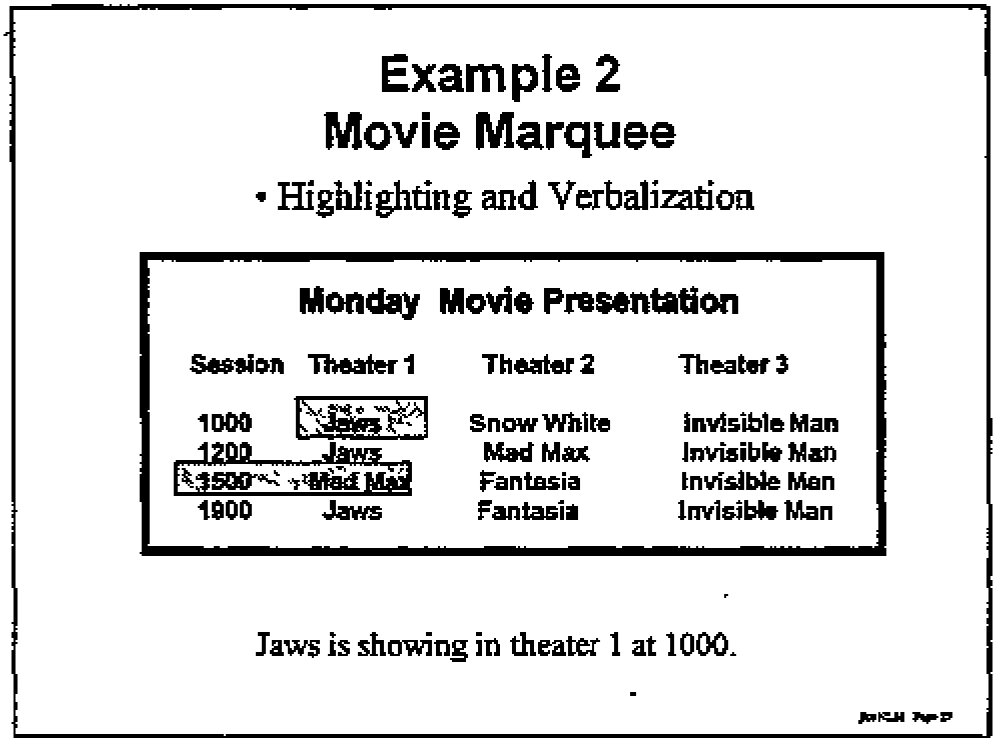
C. R. Cows
Jim Jones

Udder, Inc.
Valley Foods.

Example 2
Movie Marquee

Monday Movie Presentation

Session	Theater 1	Theater 2	Theater 3
1000	Jaws	Snow White	Invisible Man
1200	Jaws	Mad Max	Invisible Man
1500	Mad Max	Fantasia	Invisible Man
1900	Jaws	Fantasia	Invisible Man



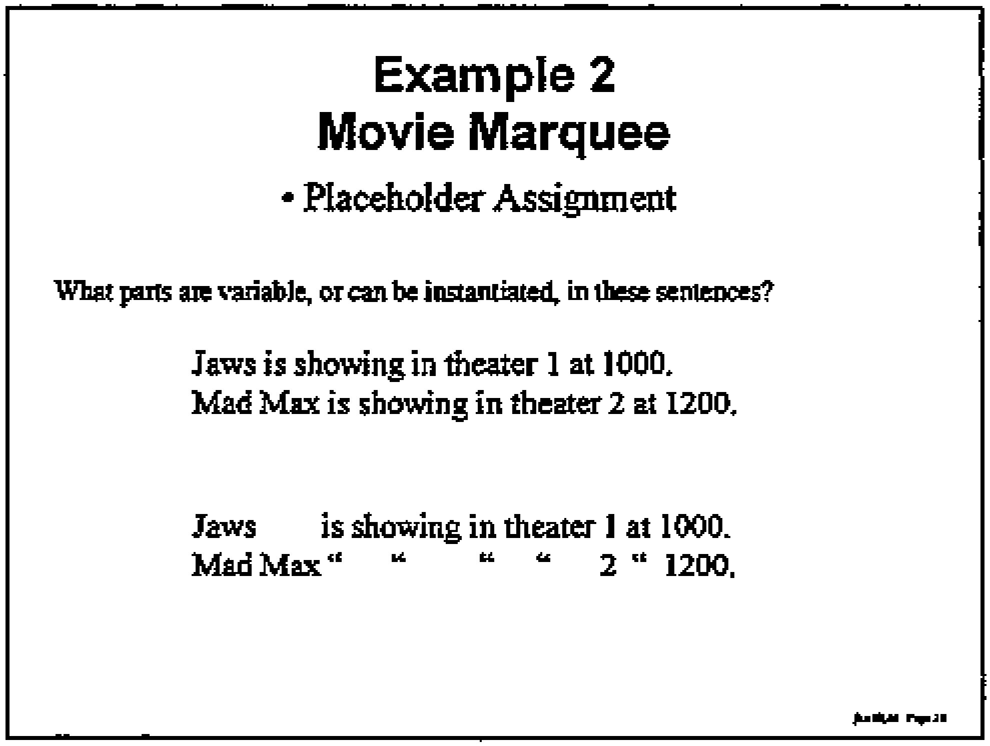
Example 2
Movie Marquee

- Highlighting and Verbalization

Monday Movie Presentation

Session	Theater 1	Theater 2	Theater 3
1000	Jaws	Snow White	Invisible Man
1200	Jaws	Mad Max	Invisible Man
1500	Mad Max	Fantasia	Invisible Man
1900	Jaws	Fantasia	Invisible Man

Jaws is showing in theater 1 at 1000.



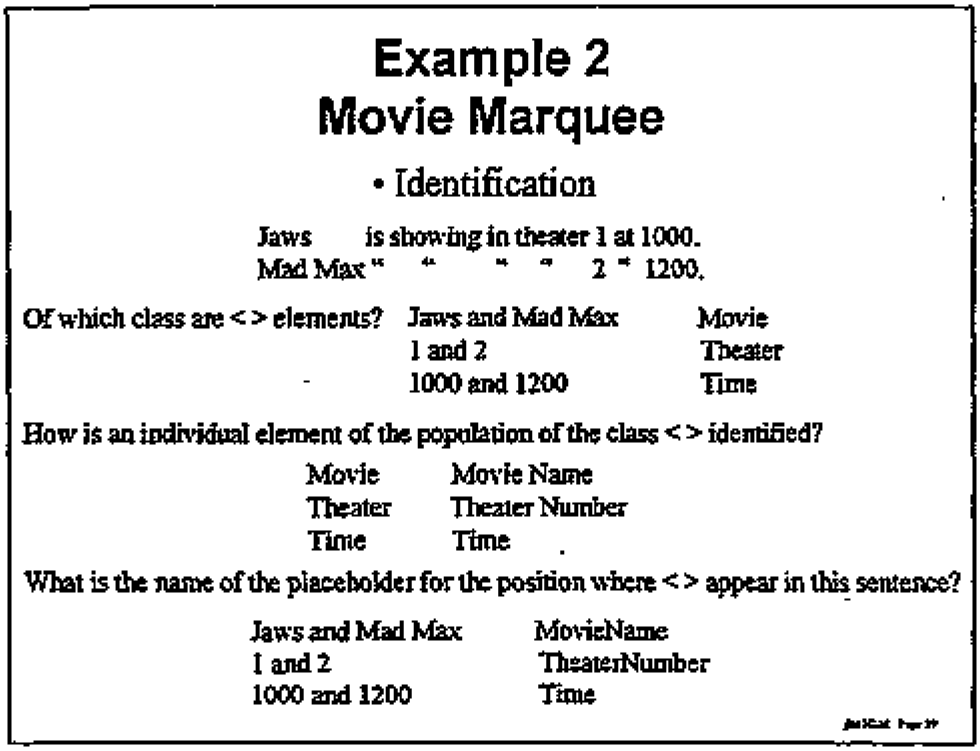
Example 2
Movie Marquee

- Placeholder Assignment

What parts are variable, or can be instantiated, in these sentences?

Jaws is showing in theater 1 at 1000.
Mad Max is showing in theater 2 at 1200.

Jaws is showing in theater 1 at 1000.
Mad Max " " " " 2 " 1200.



Example 2
Movie Marquee

- Identification

Jaws is showing in theater 1 at 1000.
Mad Max " " " " 2 " 1200.

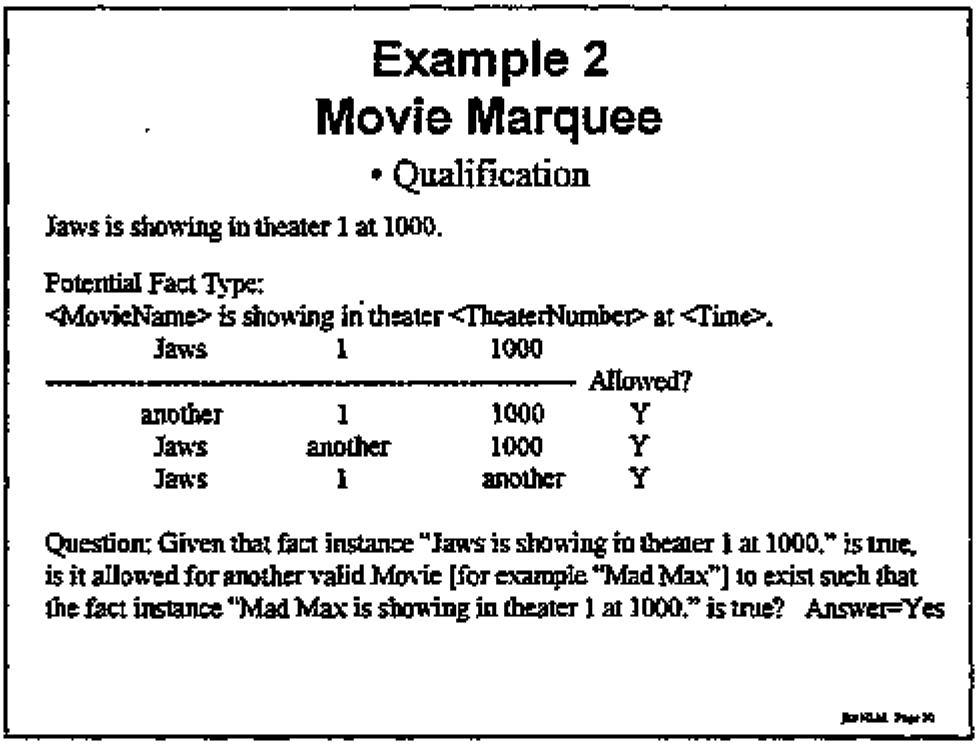
Of which class are < > elements?		
	Jaws and Mad Max	Movie
	1 and 2	Theater
	1000 and 1200	Time

How is an individual element of the population of the class < > identified?

Movie	Movie Name
Theater	Theater Number
Time	Time

What is the name of the placeholder for the position where < > appear in this sentence?

Jaws and Mad Max	MovieName
1 and 2	TheaterNumber
1000 and 1200	Time



Example 2
Movie Marquee

- Qualification

Jaws is showing in theater 1 at 1000.

Potential Fact Type:
<MovieName> is showing in theater <TheaterNumber> at <Time>.

Jaws	1	1000	
			Allowed?
another	1	1000	Y
Jaws	another	1000	Y
Jaws	1	another	Y

Question: Given that fact instance "Jaws is showing in theater 1 at 1000." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing in theater 1 at 1000." is true? Answer=Yes

Example 2
Movie Marquee

- Qualification (cont.)

Since the answer to all three question Is was Yes, The sentence needs to be tested to determine if it is an identification fact type.

Question: Does Jaws, 1, and 1000 at any moment in time identify exactly one movie showing in theater at time. Answer=No

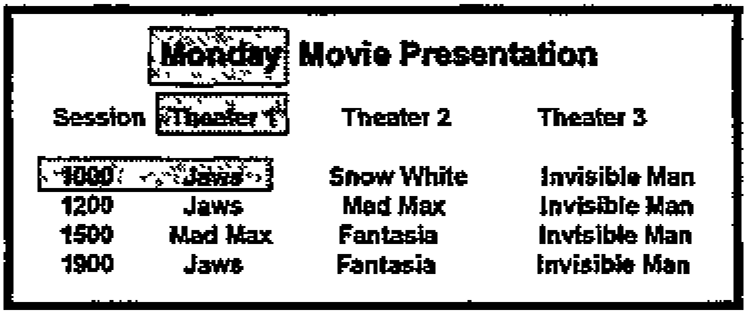
Is there a context within which "movie showing in theater at time" is uniquely identified by a "movie name, theater number, and time?" Answer=Yes

What is it? Day

This has established that the original sentence is not a instance of a valid fact type in this subject area. A new sentence needs to be created which contains "Day."

Example 2
Movie Marquee

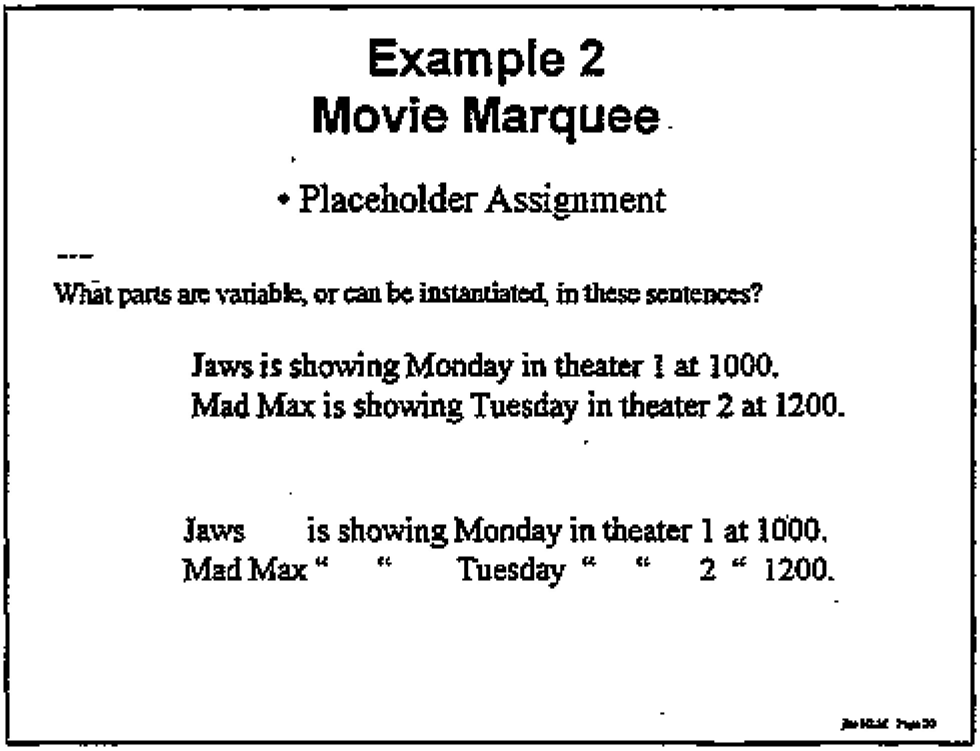
- Highlighting and Verbalization



Monday Movie Presentation

Session	Theater 1	Theater 2	Theater 3
1000	Jaws	Snow White	Invisible Man
1200	Jaws	Mad Max	Invisible Man
1500	Mad Max	Fantasia	Invisible Man
1900	Jaws	Fantasia	Invisible Man

Jaws is showing Monday in theater 1 at 1000.



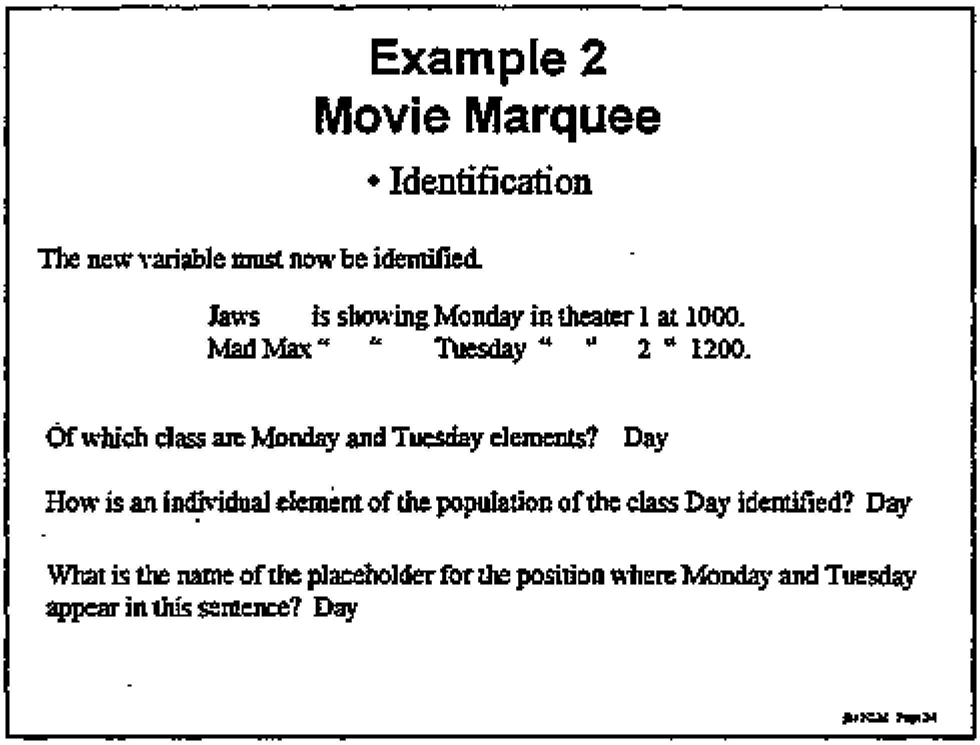
Example 2
Movie Marquee

- Placeholder Assignment

What parts are variable, or can be instantiated, in these sentences?

Jaws is showing Monday in theater 1 at 1000.
Mad Max is showing Tuesday in theater 2 at 1200.

Jaws is showing Monday in theater 1 at 1000.
Mad Max " " Tuesday " " 2 " 1200.



Example 2
Movie Marquee

- Identification

The new variable must now be identified.

Jaws is showing Monday in theater 1 at 1000.
Mad Max " " Tuesday " " 2 " 1200.

Of which class are Monday and Tuesday elements? Day

How is an individual element of the population of the class Day identified? Day

What is the name of the placeholder for the position where Monday and Tuesday appear in this sentence? Day

Example 2
Movie Marquee

- Qualification

Jaws is showing Monday in theater 1 at 1000.

Potential Fact Type:
.<MovieName> is showing <Day> in theater <TheaterNumber> at <Time>.

Jaws	Monday	1	1000	Allowed?
another	Monday	1	1000	N
Jaws	another	1	1000	Y
Jaws	Monday	another	1000	N
Jaws	Monday	1	another	Y

Question 1.1: Given that fact instance "Jaws is showing Monday in theater 1 at 1000." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing Monday in theater 1 at 1000." is true?
Answer=No

Example 2
Movie Marquee

- Qualification (cont.)

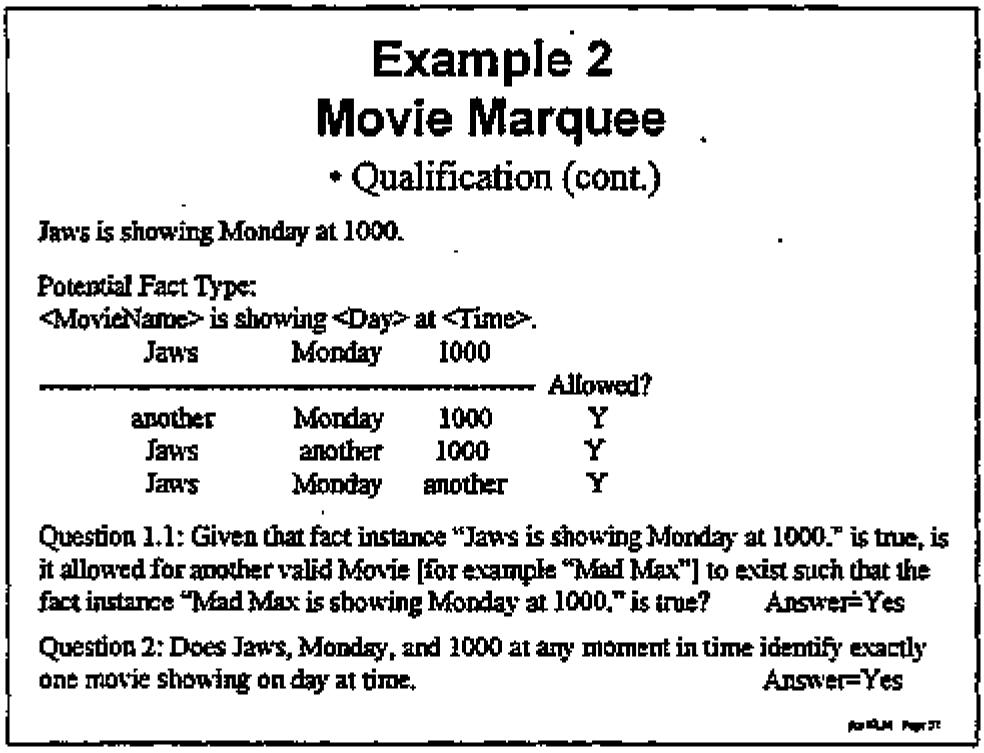
The sentence analysis produced two "N" answers so the corresponding objects must be analyzed together in a sentence to determine if they are independent.

<MovieName> is showing in theater <TheaterNumber>.

Jaws	1	Allowed?
another	1	Y
Jaws	another	Y

Question 1.1: Given that fact instance "Jaws is showing in theater 1." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing in theater 1." is true? Answer=Yes

Result: Movie and theater are independent of each other, so two sentences must be created from the two previous "Y" answers and either movie or theater.



Example 2
Movie Marquee

- Qualification (cont.)

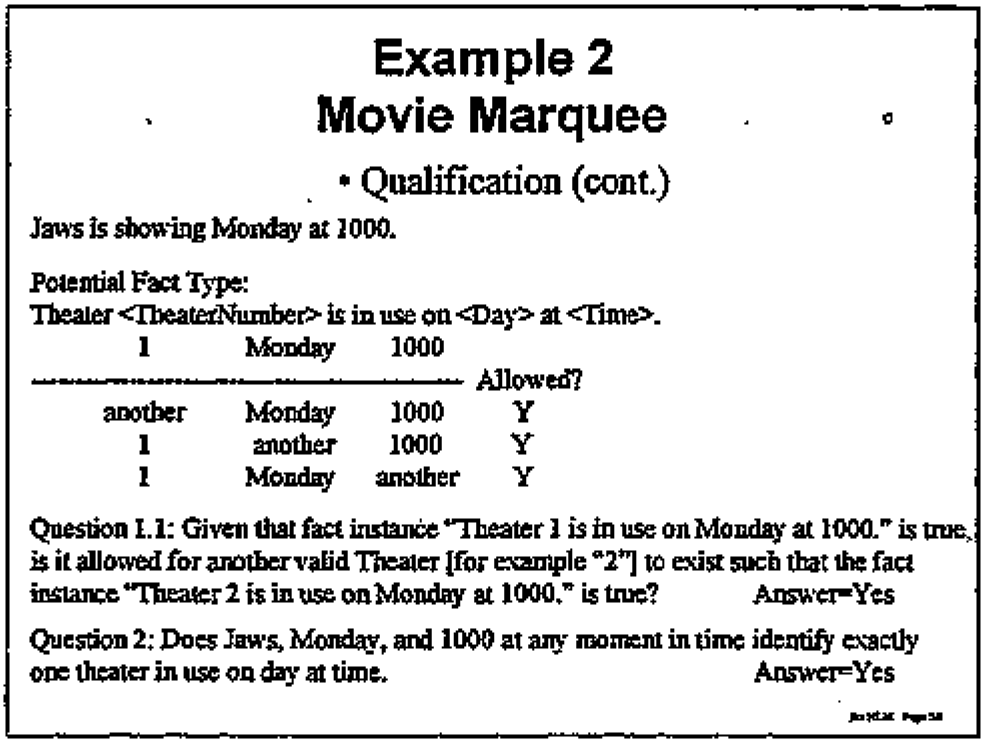
Jaws is showing Monday at 1000.

Potential Fact Type:
<MovieName> is showing <Day> at <Time>.

Jaws	Monday	1000	Allowed?
another	Monday	1000	Y
Jaws	another	1000	Y
Jaws	Monday	another	Y

Question 1.1: Given that fact instance "Jaws is showing Monday at 1000." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing Monday at 1000." is true? Answer=Yes

Question 2: Does Jaws, Monday, and 1000 at any moment in time identify exactly one movie showing on day at time. Answer=Yes



Example 2
Movie Marquee

- Qualification (cont.)

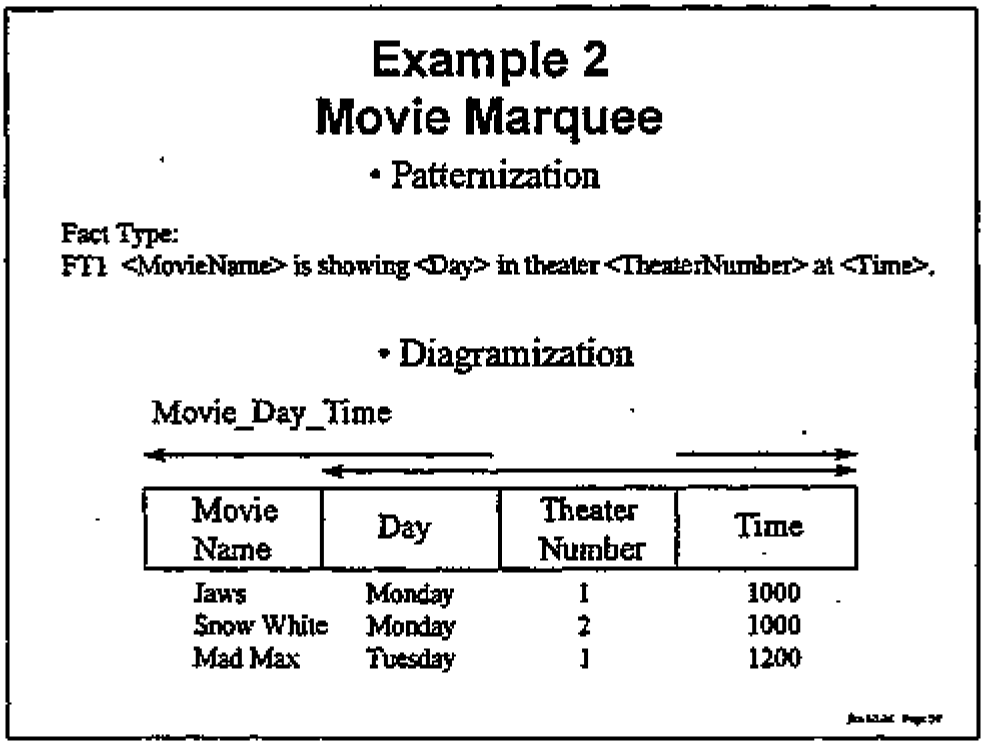
Jaws is showing Monday at 1000.

Potential Fact Type:
Theater <TheaterNumber> is in use on <Day> at <Time>.

1	Monday	1000	Allowed?
another	Monday	1000	Y
1	another	1000	Y
1	Monday	another	Y

Question 1.1: Given that fact instance "Theater 1 is in use on Monday at 1000." is true, is it allowed for another valid Theater [for example "2"] to exist such that the fact instance "Theater 2 is in use on Monday at 1000." is true? Answer=Yes

Question 2: Does Jaws, Monday, and 1000 at any moment in time identify exactly one theater in use on day at time. Answer=Yes



Example 2
Movie Marquee

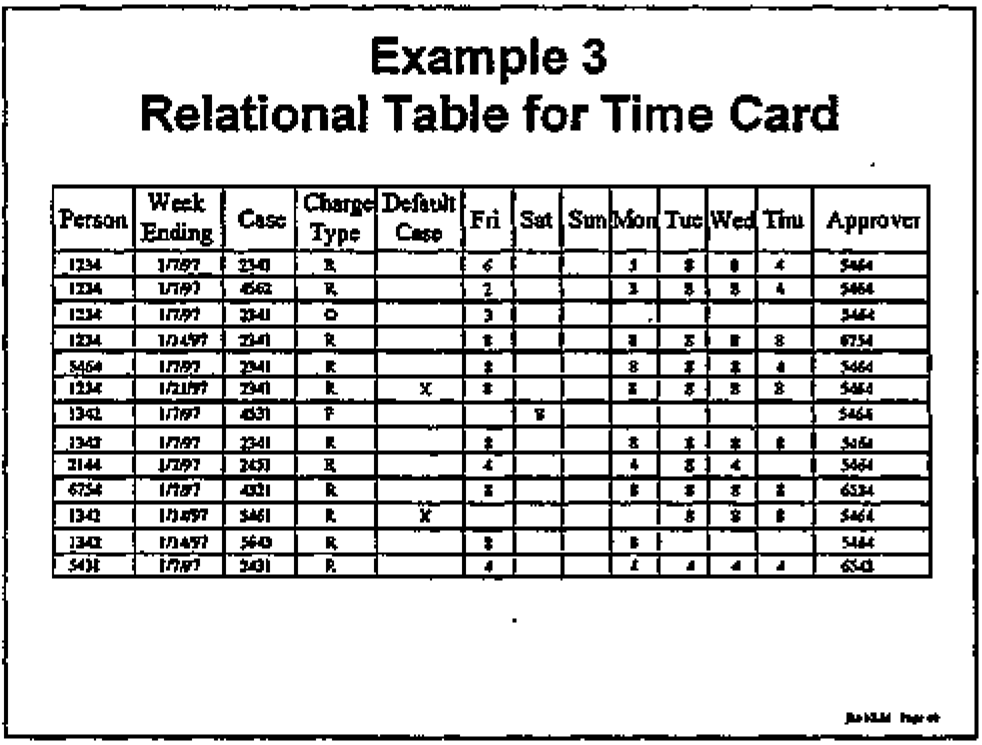
- Patternization

Fact Type:
FT1 <MovieName> is showing <Day> in theater <TheaterNumber> at <Time>.

- Diagramization

Movie_Day_Time

Movie Name	Day	Theater Number	Time
Jaws	Monday	1	1000
Snow White	Monday	2	1000
Mad Max	Tuesday	1	1200



Example 3
Relational Table for Time Card

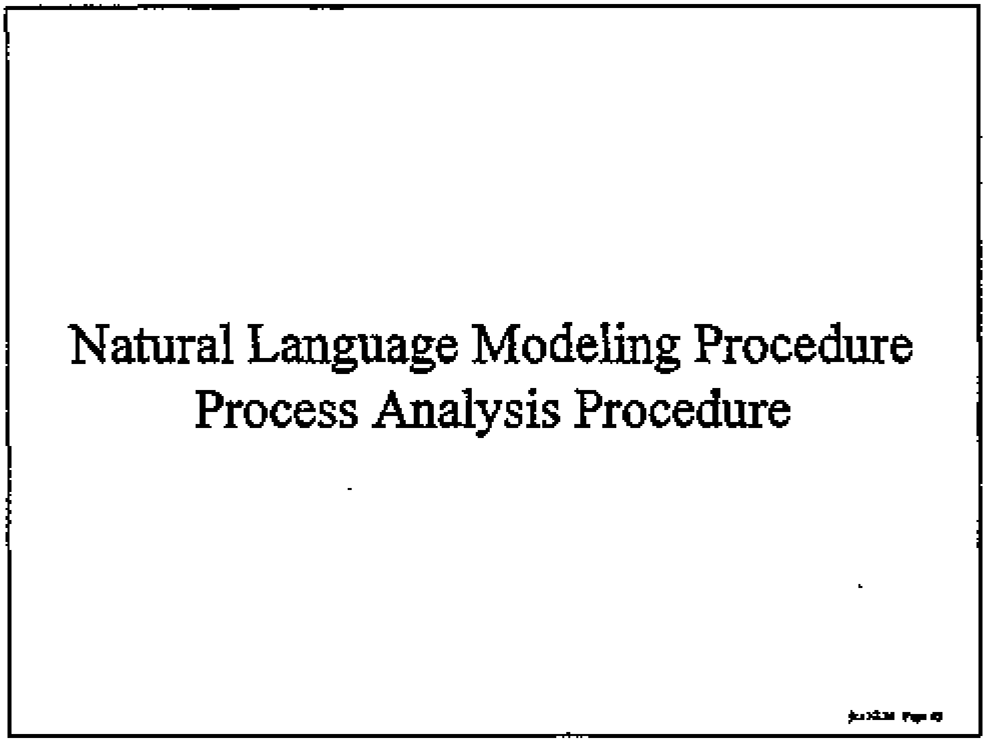
Person	Week Ending	Case	Charge Type	Default Case	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Approver
1234	1/7/97	2340	R		6			1	8	8	4	5464
1234	1/7/97	4562	R		2			3	5	5	4	5464
1234	1/7/97	2341	O		3							5464
1234	1/14/97	2341	R		8			8	5	8	8	6754
5464	1/7/97	2341	R		8			8	8	8	4	5464
1234	1/21/97	2341	R	X	8			8	8	8	8	5464
1342	1/7/97	4531	P			8						5464
1342	1/7/97	2341	R		8			8	8	8	8	5464
2144	1/7/97	2451	R		4			4	8	4		5464
6754	1/7/97	4321	R		8			8	8	8	8	6534
1342	1/14/97	5461	R	X					8	8	8	5464
1342	1/14/97	5640	R		8			8				5464
5431	1/7/97	2431	R		4			4	4	4	4	6542

Natural Language Modeling Procedure
Process Analysis Sentences

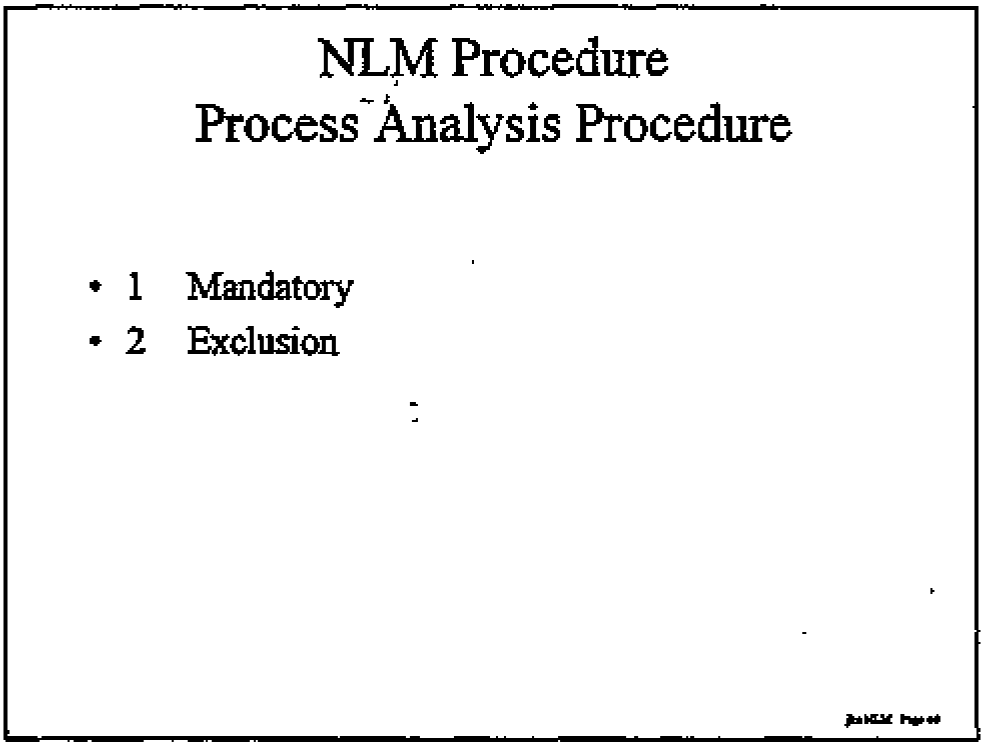
NLM Procedure
Process Analysis Questions

Question 1: Given that instance a_1 exists in A for fact type FT-1, then must a_1 exist in A for fact type FT-2?

Question 2: Given that instance a_1 exists in A for fact type FT-1, then may a_1 exist in A for fact type FT-3?

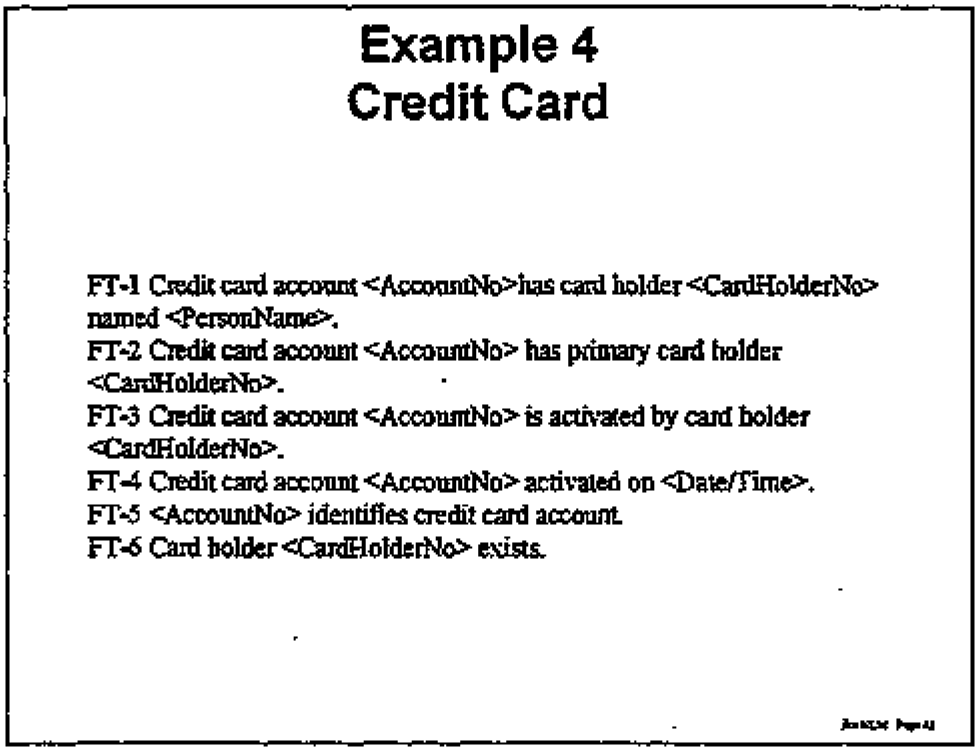


Natural Language Modeling Procedure
Process Analysis Procedure



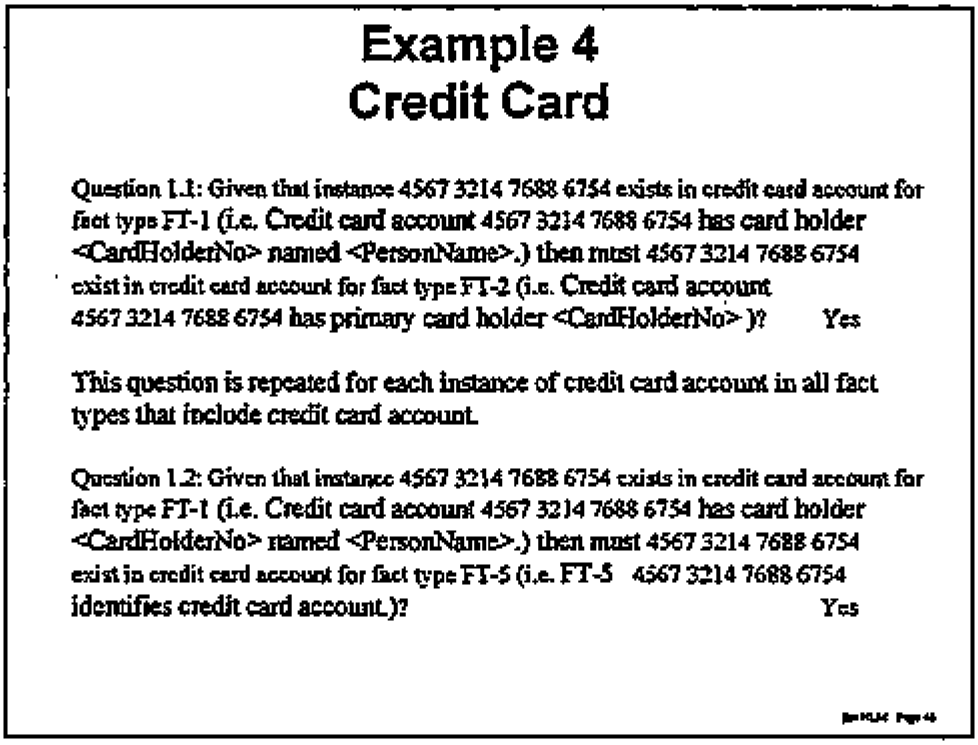
NLM Procedure
Process Analysis Procedure

- 1 Mandatory
- 2 Exclusion



Example 4
Credit Card

FT-1 Credit card account <AccountNo>has card holder <CardHolderNo> named <PersonName>.
FT-2 Credit card account <AccountNo> has primary card holder <CardHolderNo>.
FT-3 Credit card account <AccountNo> is activated by card holder <CardHolderNo>.
FT-4 Credit card account <AccountNo> activated on <Date/Time>.
FT-5 <AccountNo> identifies credit card account.
FT-6 Card holder <CardHolderNo> exists.



Example 4
Credit Card

Question 1.1: Given that instance 4567 3214 7688 6754 exists in credit card account for fact type FT-1 (i.e. Credit card account 4567 3214 7688 6754 has card holder <CardHolderNo> named <PersonName>.) then must 4567 3214 7688 6754 exist in credit card account for fact type FT-2 (i.e. Credit card account 4567 3214 7688 6754 has primary card holder <CardHolderNo>)? Yes

This question is repeated for each instance of credit card account in all fact types that include credit card account.

Question 1.2: Given that instance 4567 3214 7688 6754 exists in credit card account for fact type FT-1 (i.e. Credit card account 4567 3214 7688 6754 has card holder <CardHolderNo> named <PersonName>.) then must 4567 3214 7688 6754 exist in credit card account for fact type FT-5 (i.e. FT-5 4567 3214 7688 6754 identifies credit card account.)? Yes

Example 4
Credit Card

Question 1.1: Given that instance 4567 3214 7688 6754 exists in credit card account for fact type FT-1 (i.e. Credit card account 4567 3214 7688 6754 has card holder <CardHolderNo> named <PersonName>.) then must 4567 3214 7688 6754 exist in credit card account for fact type FT-4 (i.e. Credit card account 4567 3214 7688 6754 activated on <Date/Time>.)? No

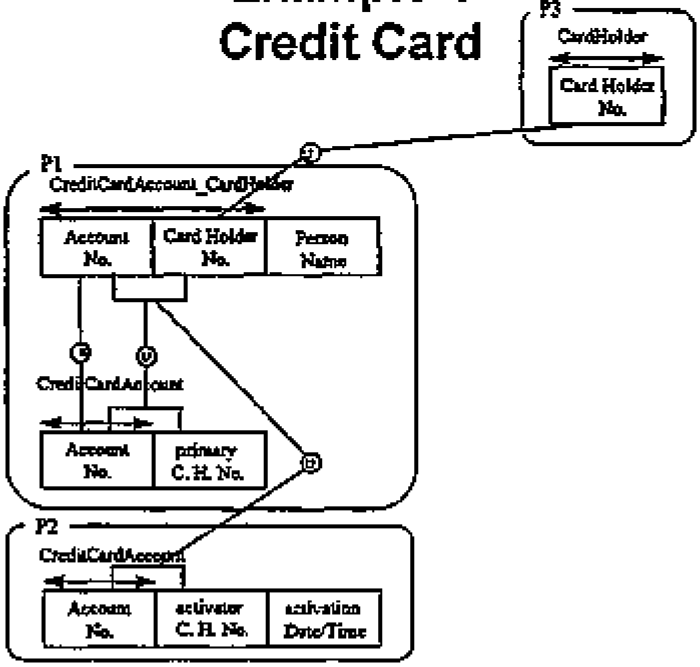
Question 2 must now be asked for this pair of fact types.

Question 1.1: Given that instance 4567 3214 7688 6754 exists in credit card account for fact type FT-1 (i.e. Credit card account 4567 3214 7688 6754 has card holder <CardHolderNo> named <PersonName>.) then may 4567 3214 7688 6754 exist in credit card account for fact type FT-4 (i.e. Credit card account 4567 3214 7688 6754 activated on <Date/Time>.)? Yes

Example 4
Credit Card

		Account									Card Holder			
		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]					[illegible]	[illegible]	[illegible]	[illegible]
P1	Establish Account	●	m	m					P3	Establish Cardholder	●			
		m	●	m						Included in P1	m	●		
		m	m	●						Included in P1	m	m	●	
P2	Activate Account	m	m	m	●	m				Included in P2	m	m		●
		m	m	m	m	●								

Example 4
Credit Card

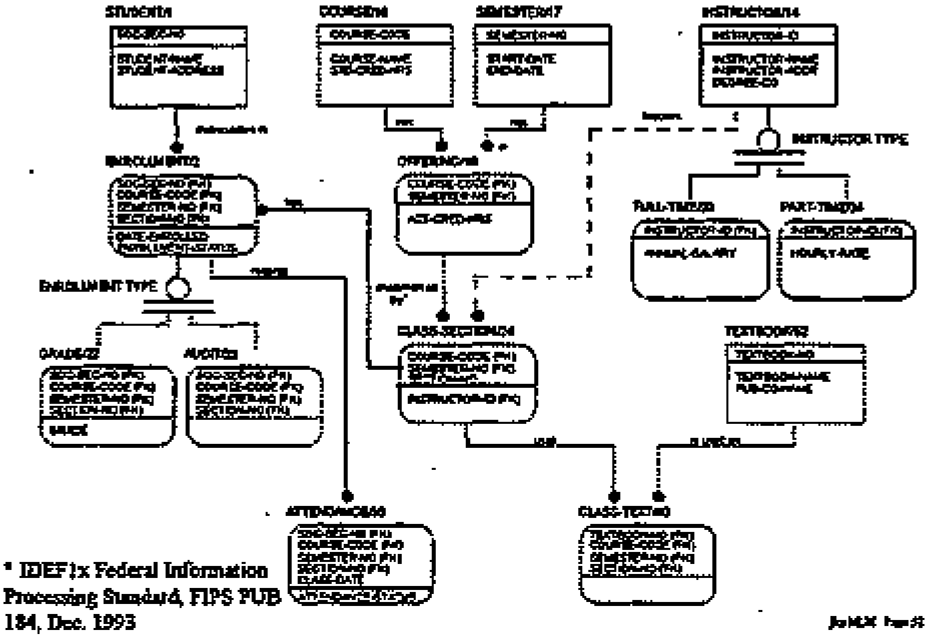


Validating Information Models

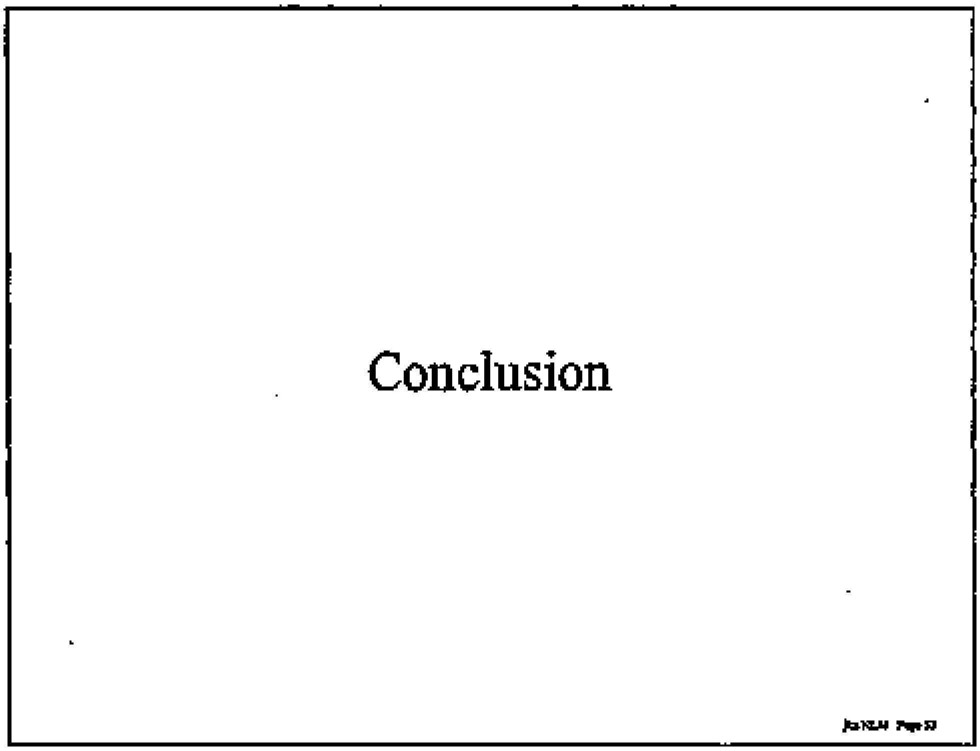
Validating Information Models

Natural Language Analysis can be used to validate any information model (ER, O-O, etc.).

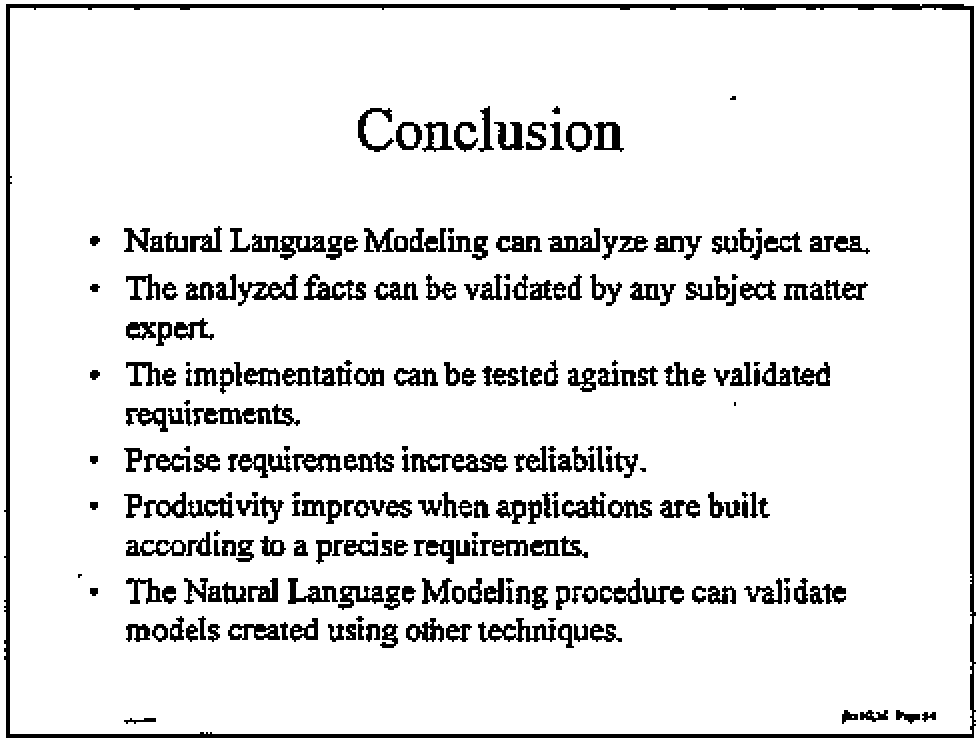
IDEF1X Student Class Model *



* IDEF1x Federal Information Processing Standard, FIPS PUB 184, Dec. 1993



Conclusion



Conclusion

- Natural Language Modeling can analyze any subject area.
- The analyzed facts can be validated by any subject matter expert.
- The implementation can be tested against the validated requirements.
- Precise requirements increase reliability.
- Productivity improves when applications are built according to a precise requirements.
- The Natural Language Modeling procedure can validate models created using other techniques.

Session X1: Defining Software Processes

Dr. Gerald McDonald
Consultant, Sandia National Laboratories

DEFINITION AND DOCUMENTATION OF ENGINEERING PROCESSES

GERALD W. MCDONALD, Ph.D.

This tutorial is an extract of a two-day workshop developed under the auspices of the Quality Engineering Department at Sandia National Laboratories. The presentation starts with basic definitions and addresses why processes should be defined and documented. It covers three primary topics: (1) process considerations and rationale, (2) approach to defining and documenting engineering processes, and (3) an IDEF0 model of the process for defining engineering processes.

Process considerations and rationale introduce models for documenting processes; describe the general architecture for product development; and define implications of immature processes versus those for mature processes.

The approach describes the top-level subprocesses that make up the methodology for definition and documentation of engineering processes; namely: planning, gaining management approval for a process definition project, collecting data on the as-is process to capture current best practices within the organization, constructing a model of the as-is process, and verifying and validating that model.

The final portion presents a four-level, hierarchical model that describes HOW to define and document an engineering process.

BIOGRAPHY

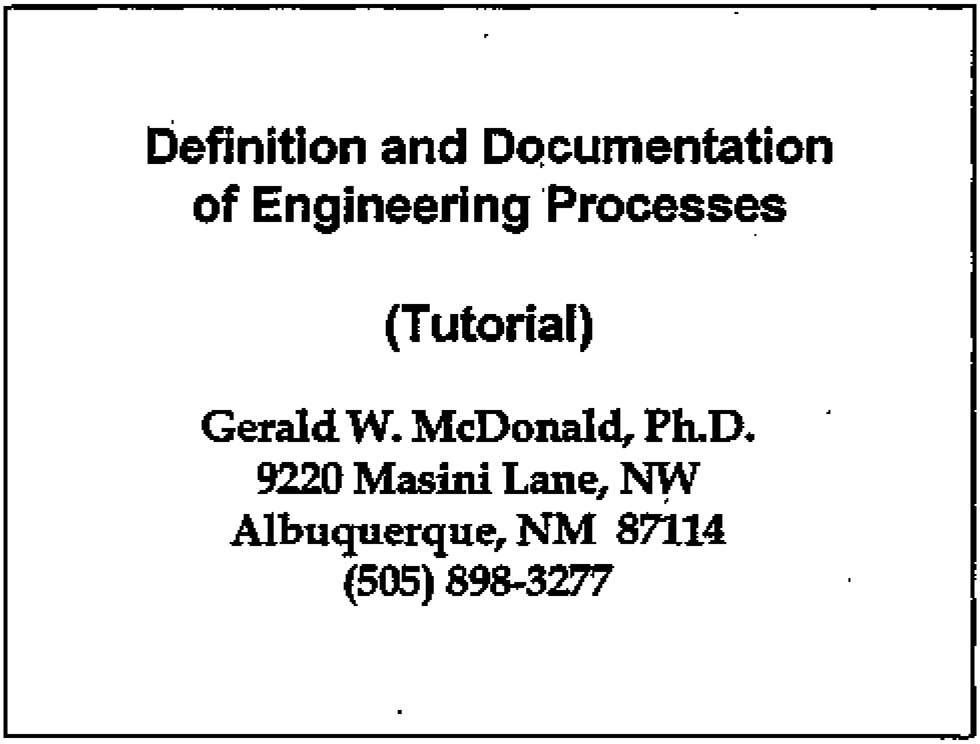
GERALD W. MCDONALD, Ph.D.

Dr. McDonald has a Bachelor of Science in Engineering Science and a Master of Science in Computer Systems Management from the Naval Postgraduate School. Following his retirement the Navy he received a Master of Engineering in Industrial and Systems Engineering and a Ph.D. in Quantitative Management Science (Operations Research) from the University of Florida.

Following receipt of his Ph.D. he worked for BDM International as an executive-level Program and/or Project Manager and technical leader. During his thirteen years with that firm he led both software and non-software projects.

During the three years since his retirement from BDM he has acted as consultant to Sandia, SEMATECH, and a number of other organizations. As a consultant he has worked primarily in the field of Software Process Improvement. Besides direct technical assistance he has presented training and workshops in software areas such as: quality engineering, software inspections, process definition and documentation, and metrics.

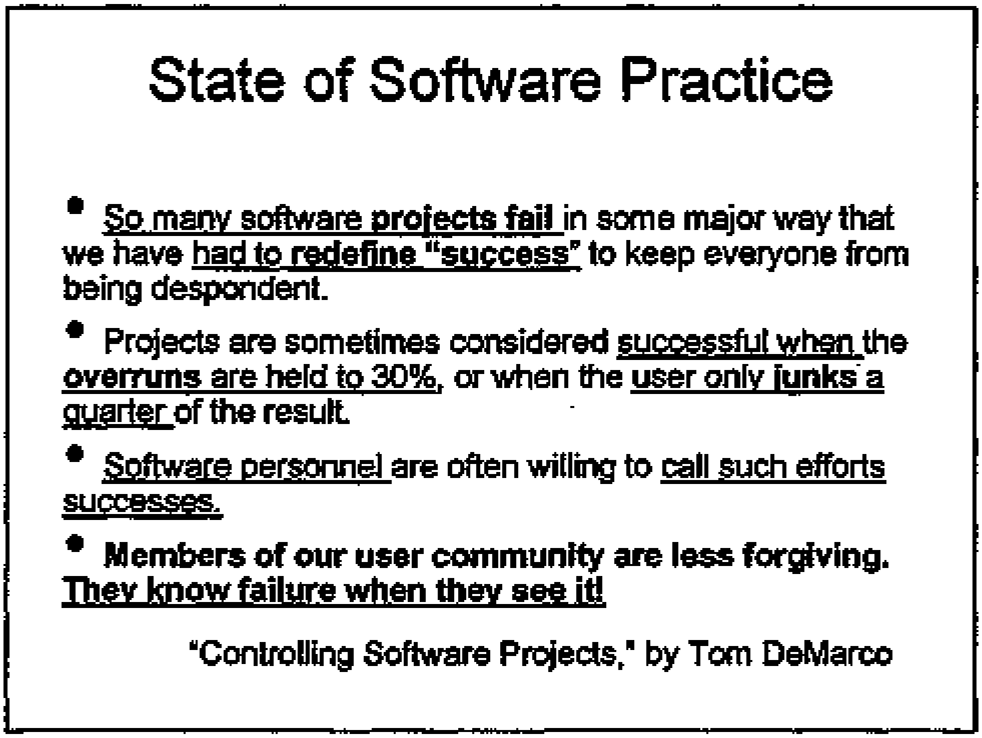
9220 Masini Lane, NW
Albuquerque, NM 87114-6001
Voice: 505-898-3277
E-mail: GeraldWMcDonald@juno.com



Definition and Documentation of Engineering Processes

(Tutorial)

Gerald W. McDonald, Ph.D.
9220 Masini Lane, NW
Albuquerque, NM 87114
(505) 898-3277



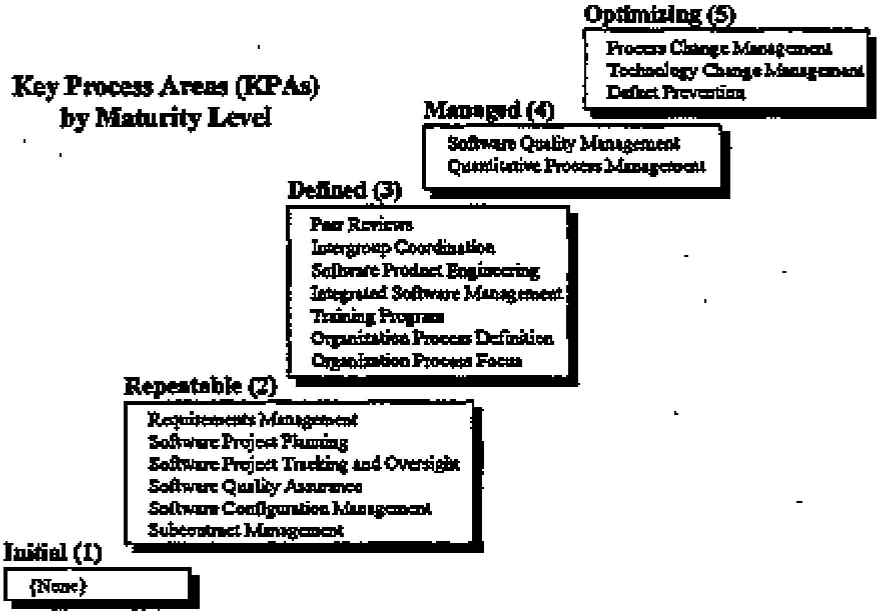
State of Software Practice

- So many software projects fail in some major way that we have had to redefine "success" to keep everyone from being despondent.
- Projects are sometimes considered successful when the overruns are held to 30%, or when the user only junks a quarter of the result.
- Software personnel are often willing to call such efforts successes.
- **Members of our user community are less forgiving. They know failure when they see it!**

"Controlling Software Projects," by Tom DeMarco

SEI CAPABILITY MATURITY MODEL (CMM)

Key Process Areas (KPAs)
by Maturity Level



Optimizing (5)
- Process Change Management
- Technology Change Management
- Defect Prevention

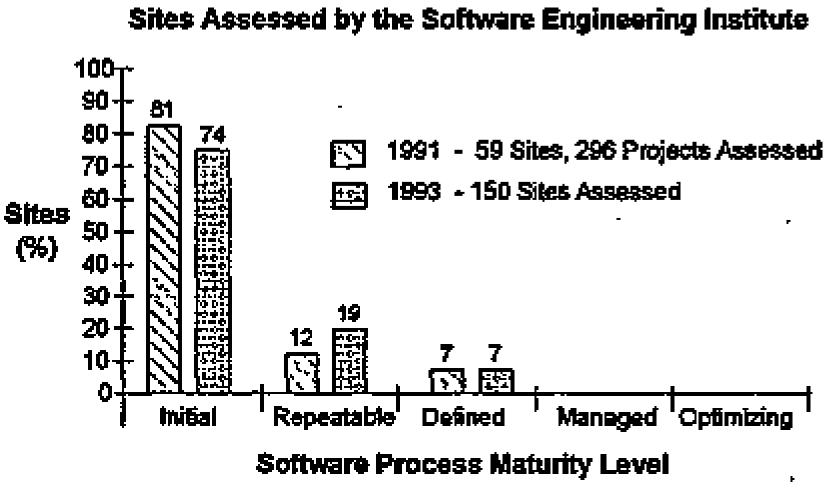
Managed (4)
- Software Quality Management
- Quantitative Process Management

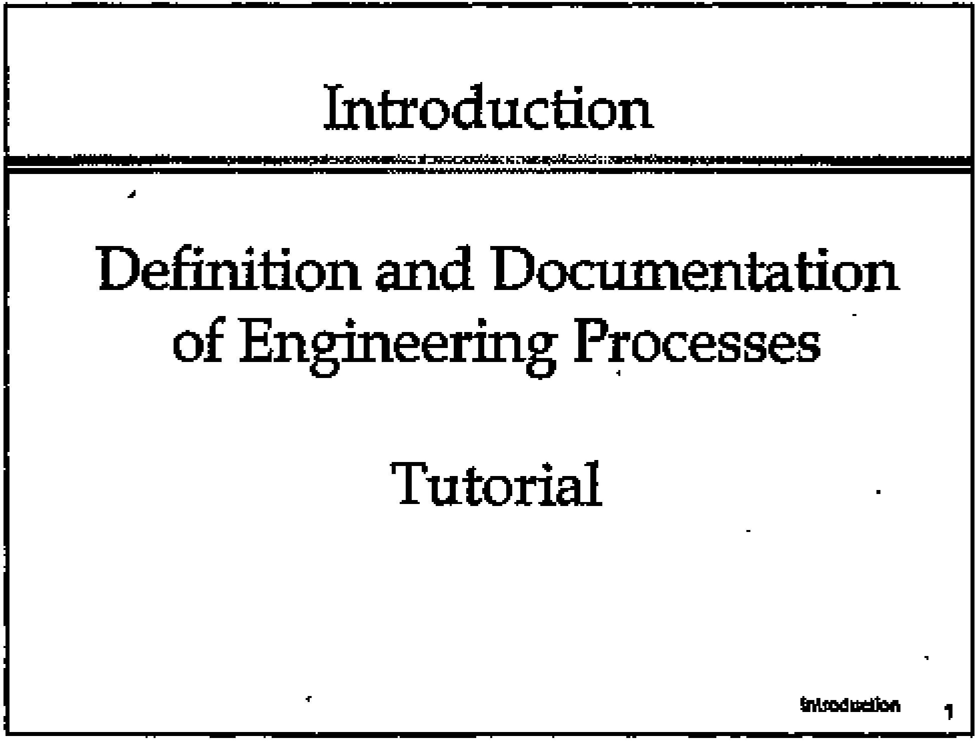
Defined (3)
- Peer Reviews
- Intergroup Coordination
- Software Product Engineering
- Integrated Software Management
- Training Program
- Organization Process Definition
- Organization Process Focus

Repeatable (2)
- Requirements Management
- Software Project Planning
- Software Project Tracking and Oversight
- Software Quality Assurance
- Software Configuration Management
- Subcontract Management

Initial (1)
- {None}

State of Process Maturity

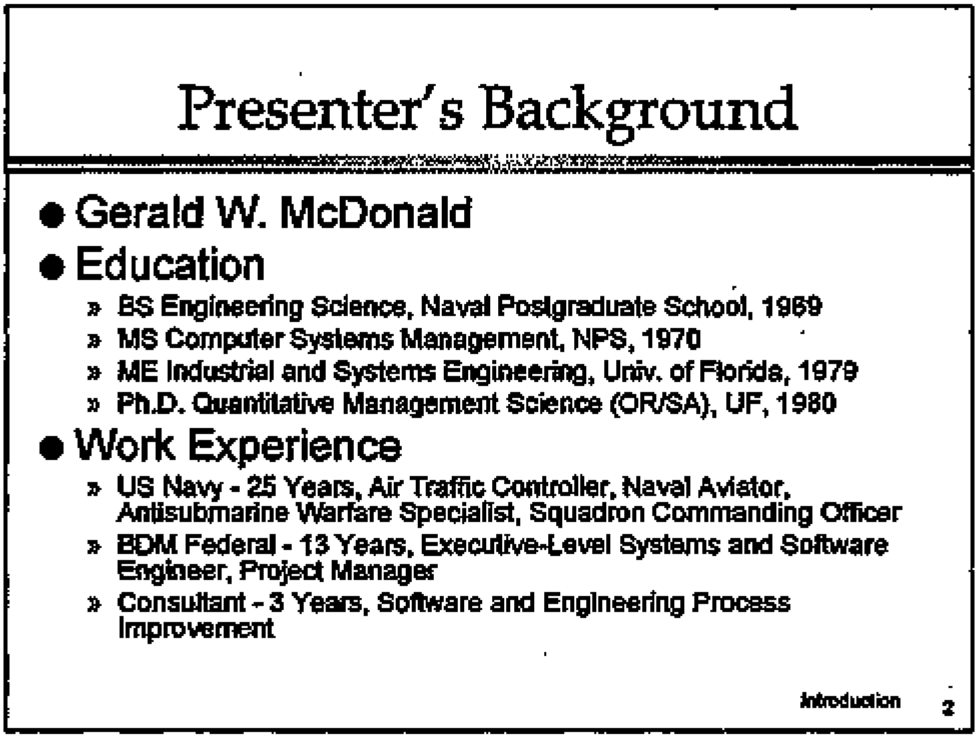




Introduction

Definition and Documentation of Engineering Processes

Tutorial



Presenter's Background

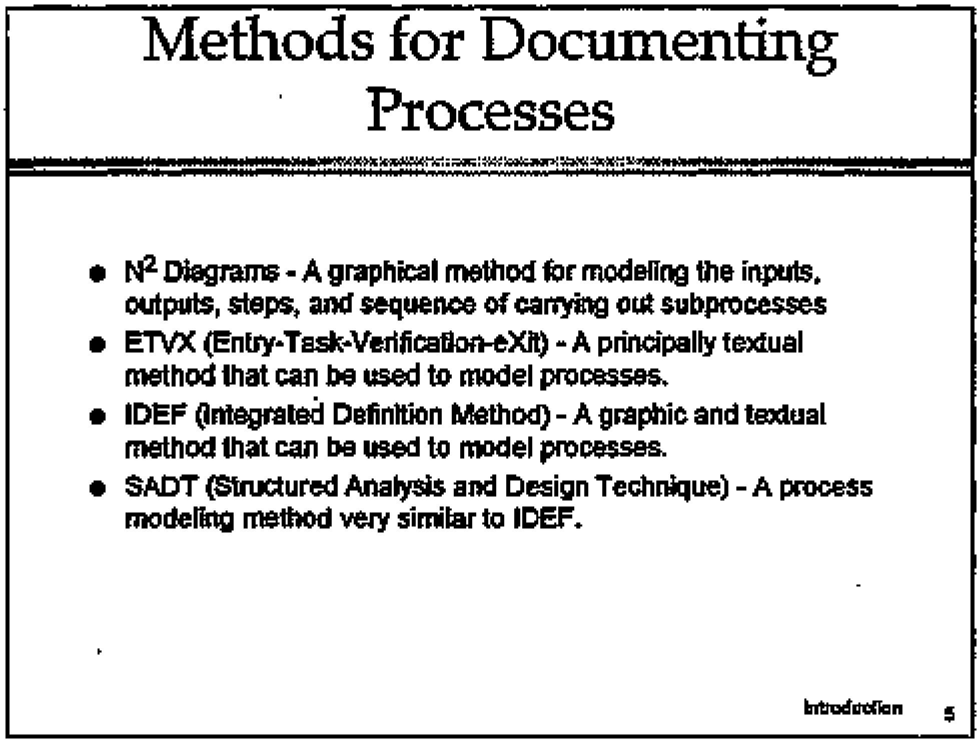
- Gerald W. McDonald
- Education
  - » BS Engineering Science, Naval Postgraduate School, 1969
  - » MS Computer Systems Management, NPS, 1970
  - » ME Industrial and Systems Engineering, Univ. of Florida, 1979
  - » Ph.D. Quantitative Management Science (OR/SA), UF, 1980
- Work Experience
  - » US Navy - 25 Years, Air Traffic Controller, Naval Aviator, Antisubmarine Warfare Specialist, Squadron Commanding Officer
  - » BDM Federal - 13 Years, Executive-Level Systems and Software Engineer, Project Manager
  - » Consultant - 3 Years, Software and Engineering Process Improvement

Process Definitions

- Process (Activity) - A set of partially ordered steps by which people apply technology and work activities to transform information, materials, and energy into a product(s) to reach a specified goal.
- Subprocesses (Sub-Activities) - The steps that make up a process or a higher level subprocess. (Depending on the context, a subprocess is often referred to as a process.)
- Engineering Process - The process involved in the management and engineering of one or more engineering work products.

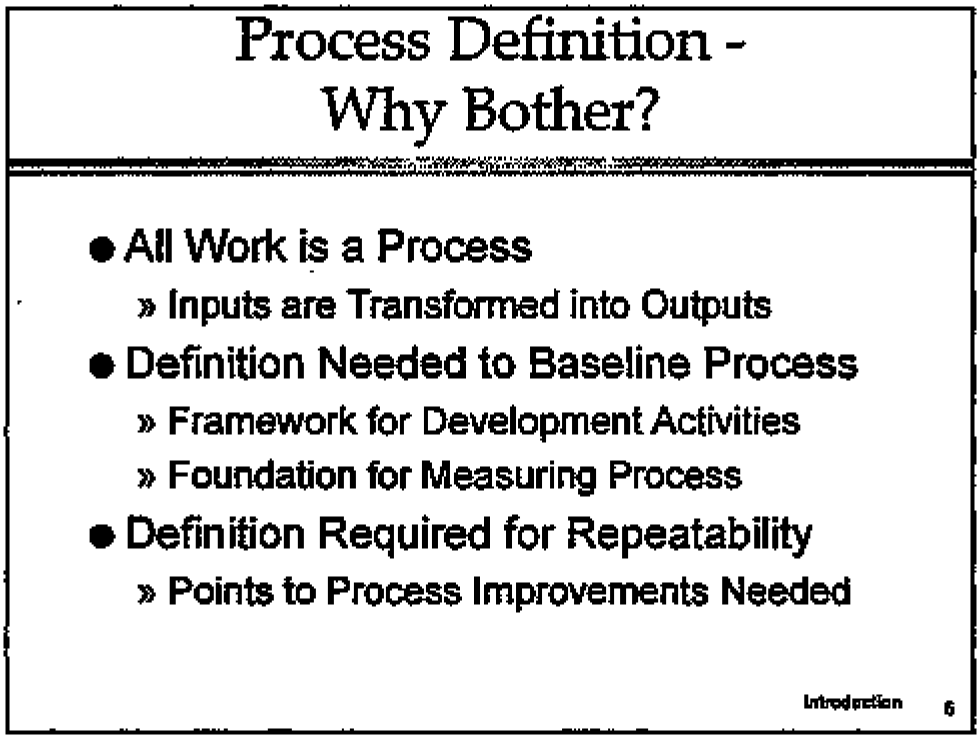
Elements Associated With a Process

- Inputs - Elements that are transformed into outputs by execution of a process
- Outputs (Work Products) - Elements that are produced as the result of executing a process/subprocess; e.g., plans documents, code, schedules, etc. They are typically represented in process models as inputs to and outputs from processes/ subprocesses.
- Controls - Elements that control and constrain engineering processes; e.g., policies, standards, schedules, budgets, etc.
- Mechanisms - Agents that perform the actions necessary to carry out a subprocess.



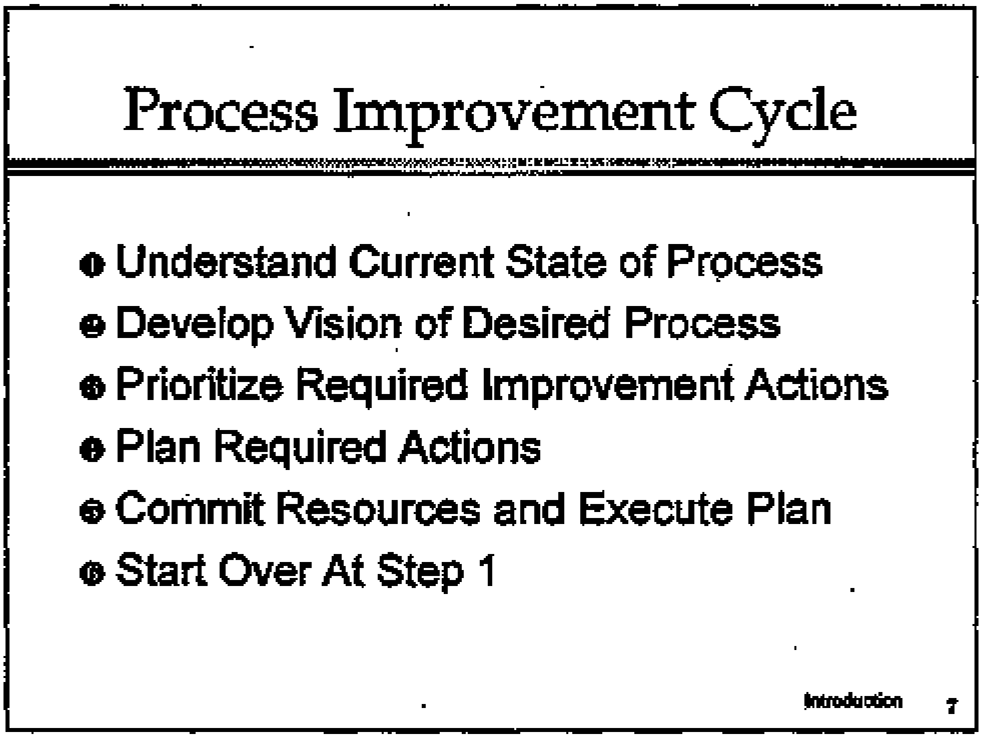
Methods for Documenting Processes

- N^2 Diagrams - A graphical method for modeling the inputs, outputs, steps, and sequence of carrying out subprocesses
- ETVX (Entry-Task-Verification-eXit) - A principally textual method that can be used to model processes.
- IDEF (Integrated Definition Method) - A graphic and textual method that can be used to model processes.
- SADT (Structured Analysis and Design Technique) - A process modeling method very similar to IDEF.



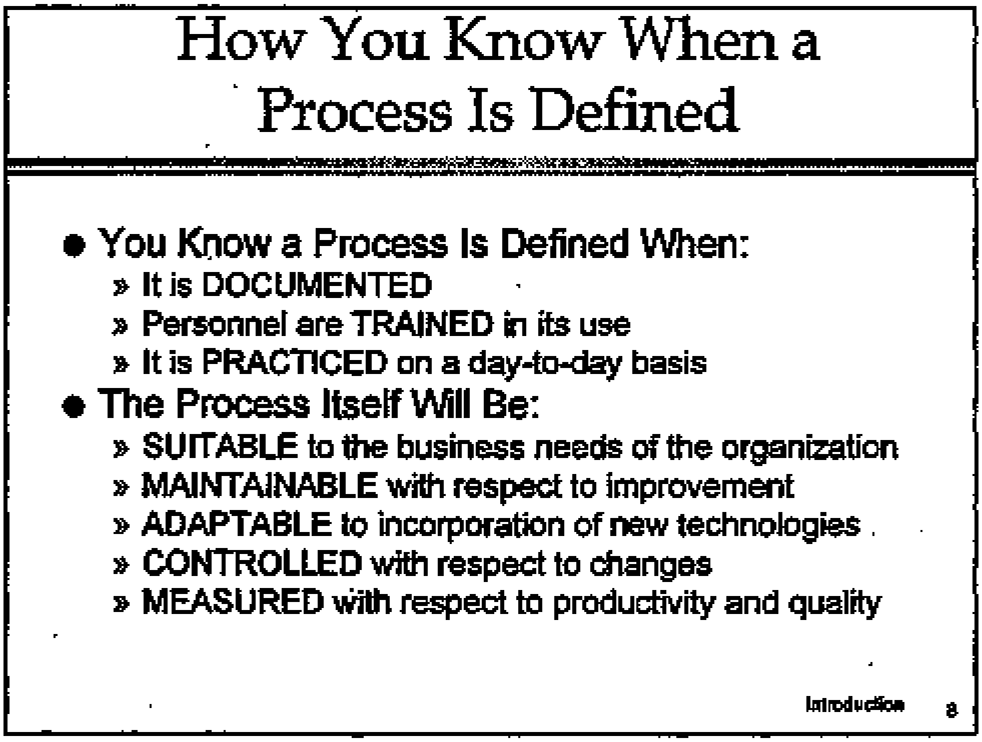
Process Definition - Why Bother?

- All Work is a Process
  - » Inputs are Transformed into Outputs
- Definition Needed to Baseline Process
  - » Framework for Development Activities
  - » Foundation for Measuring Process
- Definition Required for Repeatability
  - » Points to Process Improvements Needed



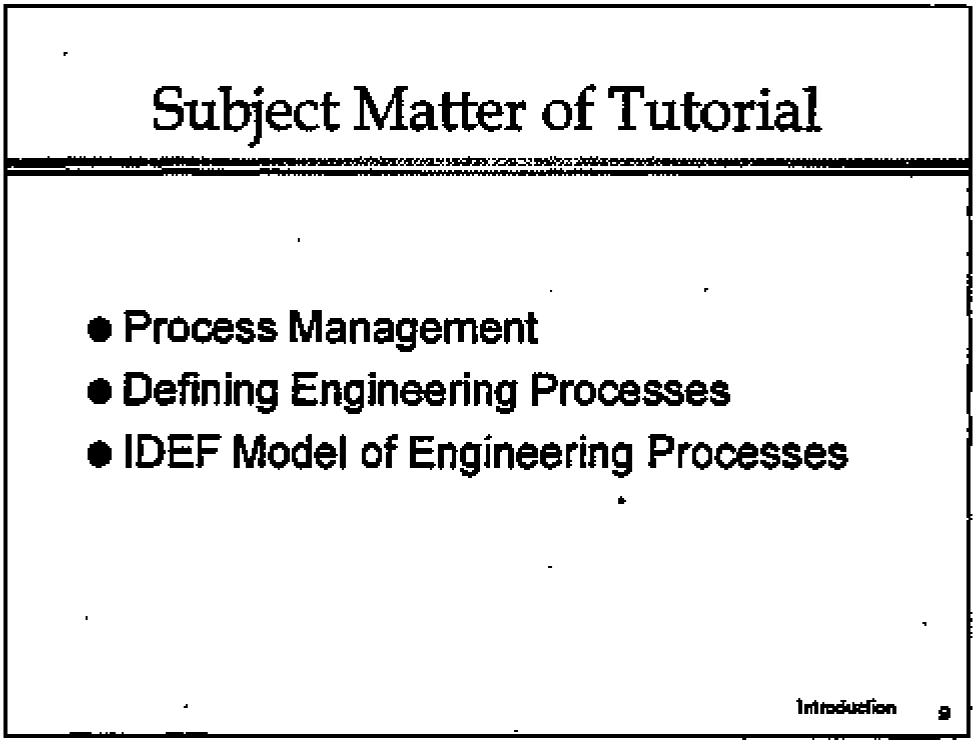
Process Improvement Cycle

- ❶ Understand Current State of Process
- ❷ Develop Vision of Desired Process
- ❸ Prioritize Required Improvement Actions
- ❹ Plan Required Actions
- ❺ Commit Resources and Execute Plan
- ❻ Start Over At Step 1



How You Know When a Process Is Defined

- You Know a Process Is Defined When:
  - » It is DOCUMENTED
  - » Personnel are TRAINED in its use
  - » It is PRACTICED on a day-to-day basis
- The Process Itself Will Be:
  - » SUITABLE to the business needs of the organization
  - » MAINTAINABLE with respect to improvement
  - » ADAPTABLE to incorporation of new technologies
  - » CONTROLLED with respect to changes
  - » MEASURED with respect to productivity and quality



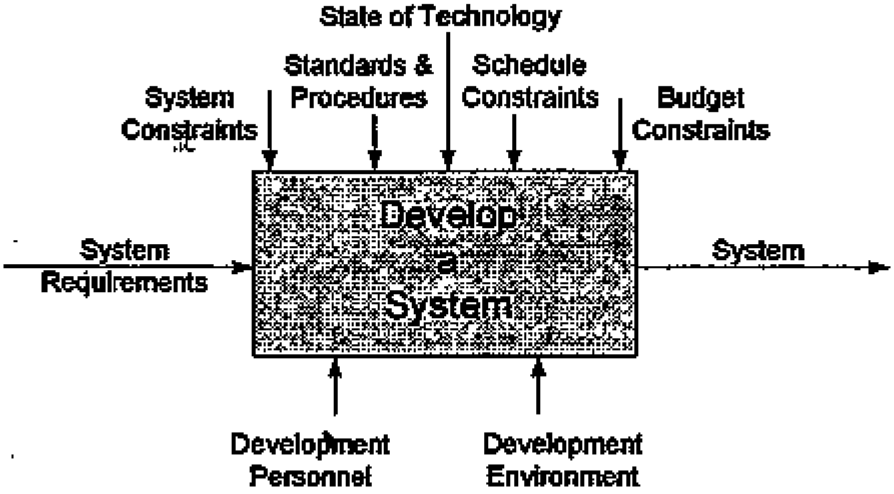
Subject Matter of Tutorial

- Process Management
- Defining Engineering Processes
- IDEF Model of Engineering Processes

Process Background

Process Considerations and Rationale

Top-Level View of Process For System Development



State of Technology

Standards & Procedures

System Constraints

Schedule Constraints

Budget Constraints

System Requirements

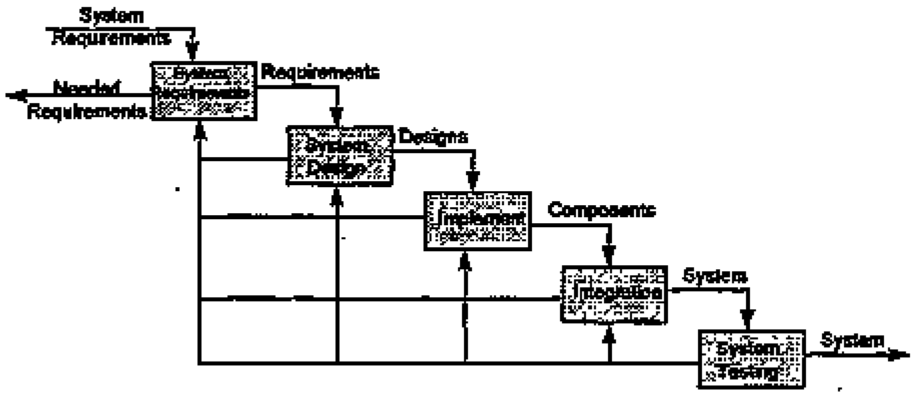
Develop a System

System

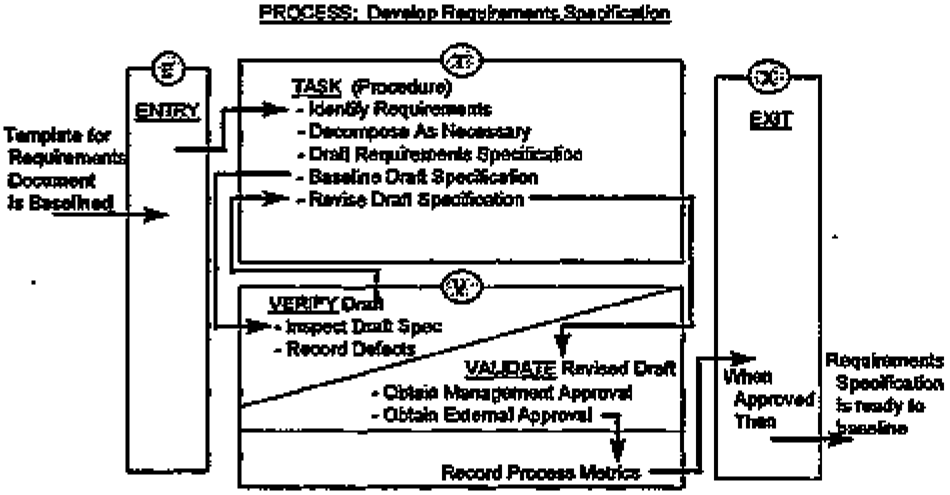
Development Personnel

Development Environment

N² Version - Second-Level of System Development Processes



Entry-Task-Verification-eXit (ETVX) Example



PROCESS: Develop Requirements Specification

Template for Requirements Document Is Baselined

ENTRY

TASK (Procedure)
- Identify Requirements
- Decompose As Necessary
- Draft Requirements Specification
- Baseline Draft Specification
- Revise Draft Specification

VERIFY Draft
- Inspect Draft Spec
- Record Defects

VALIDATE Revised Draft
- Obtain Management Approval
- Obtain External Approval

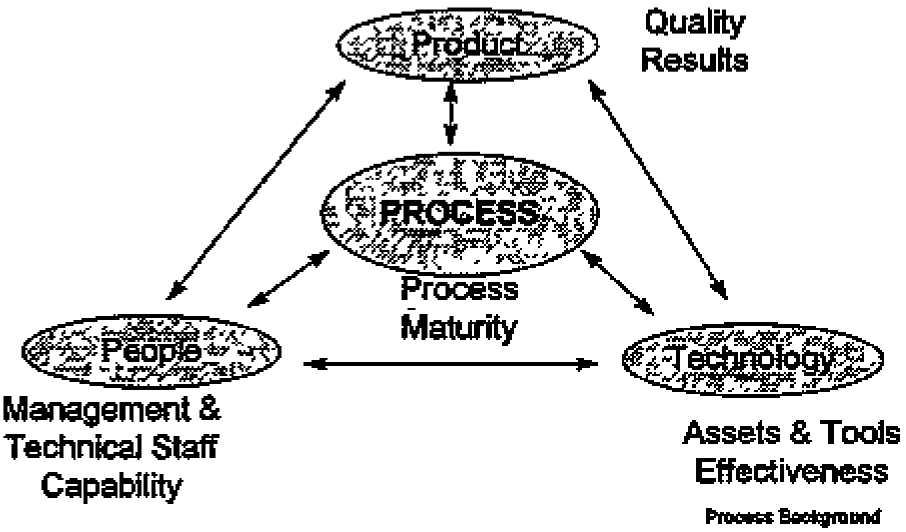
Record Process Metrics

EXIT

When Approved Then

Requirements Specification is ready to baseline

Elements and Factors of System Development



Product

Quality
Results

PROCESS

Process
Maturity

People

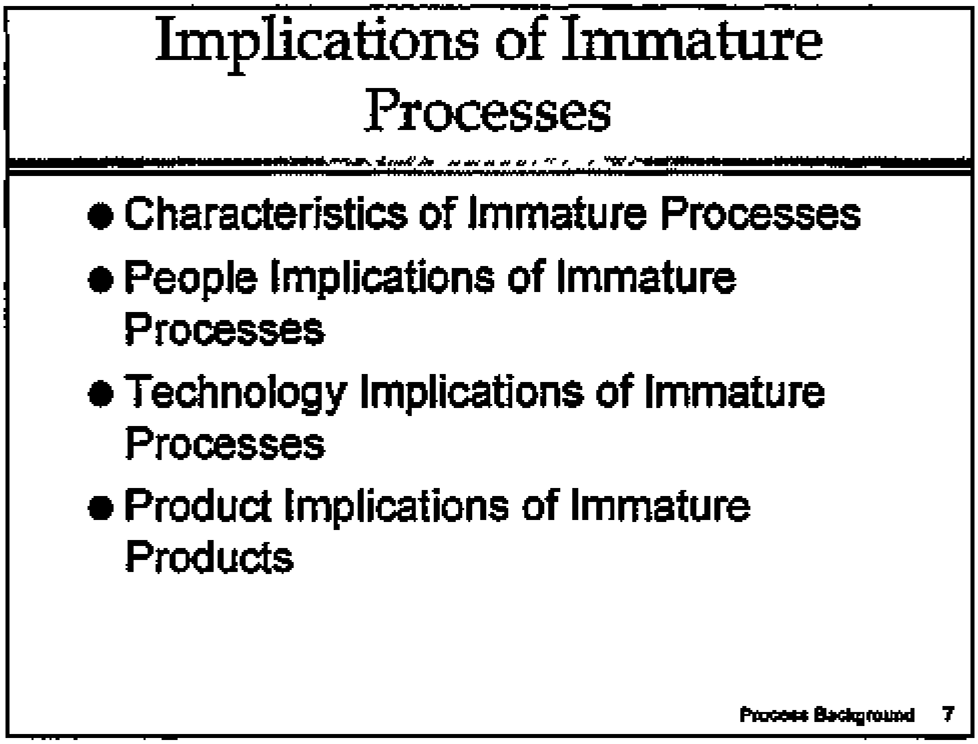
Management &
Technical Staff
Capability

Technology

Assets & Tools
Effectiveness

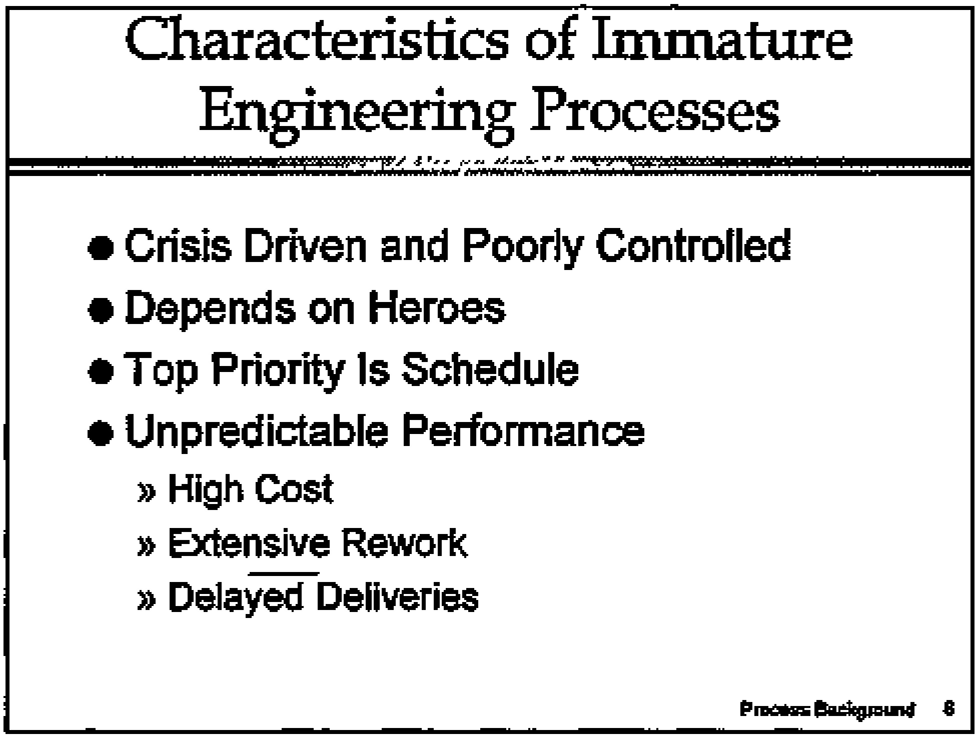
Motivation for Defining Engineering Processes

- Implications of Immature Processes
- Motivation for Improving Engineering Processes
- Implications of Mature Processes



Implications of Immature Processes

- Characteristics of Immature Processes
- People Implications of Immature Processes
- Technology Implications of Immature Processes
- Product Implications of Immature Products



Characteristics of Immature Engineering Processes

- Crisis Driven and Poorly Controlled
- Depends on Heroes
- Top Priority Is Schedule
- Unpredictable Performance
  - » High Cost
  - » Extensive Rework
  - » Delayed Deliveries

People Implications of Immature Process

- Focus on Fire Fighting and Crisis Management
- Process Steps Depend on Individual Performing Work
- Low Effectiveness, High Frustration, and Adversarial Relationships

Technology Implications of Immature Process

- Technology Needs Are Difficult to Identify
- Implementation of New Technology Is Seldom Cost Effective
- Implementation of New Technology Is Usually Difficult

Product Implications of Immature Processes

- Quality Is Dependent On Individual Performing Work
- Rework Is Often Extensive
- Requirements Creep During Development
- Customer Is Often Dissatisfied

Motivation for Improving Engineering Processes

- Characteristics of Mature Engineering Processes
- People Implications of Mature Engineering Processes
- Technology Implications of Mature Engineering Processes
- Product Implications of Mature Engineering Processes

Characteristics of Mature Engineering Processes

- Defined, Documented, Controlled, and Improved
- Processes Are Corporate Assets
- Focus Is On Product and Process Improvement
- Performance Is Well Controlled
  - » Cost
  - » Low Rework
  - » On-Time Deliveries

People Implications of Mature Processes

- High Sense of Teamwork
- Reliance on Defined Process Rather Than Ad Hoc Methods
- Development Is Driven By Events Rather Than Crisis or Schedule
- Little Crisis Management Required

Technology Implications of Mature Processes

- Technology Needs Can Be Identified
- Quantitative Basis Can Be Developed to Support Automation Needs and Selection
- Potential Impacts of New Technology Can Be Estimated More Accurately

Product Implications of Mature Processes

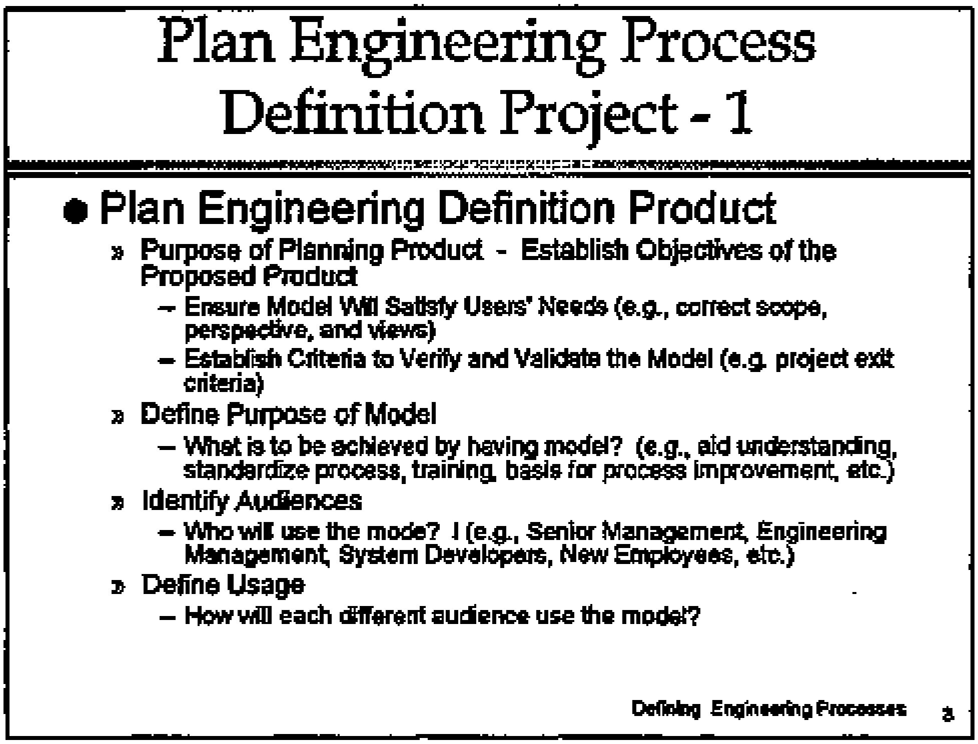
- Cost of Quality Is Very Low and Independent of Individuals Performing Work
- Customer Is More Often Satisfied With Products
- Rework Requirements Are Often Negligible

Approach to Defining Engineering Processes

Preparations for and Modeling of Engineering Processes

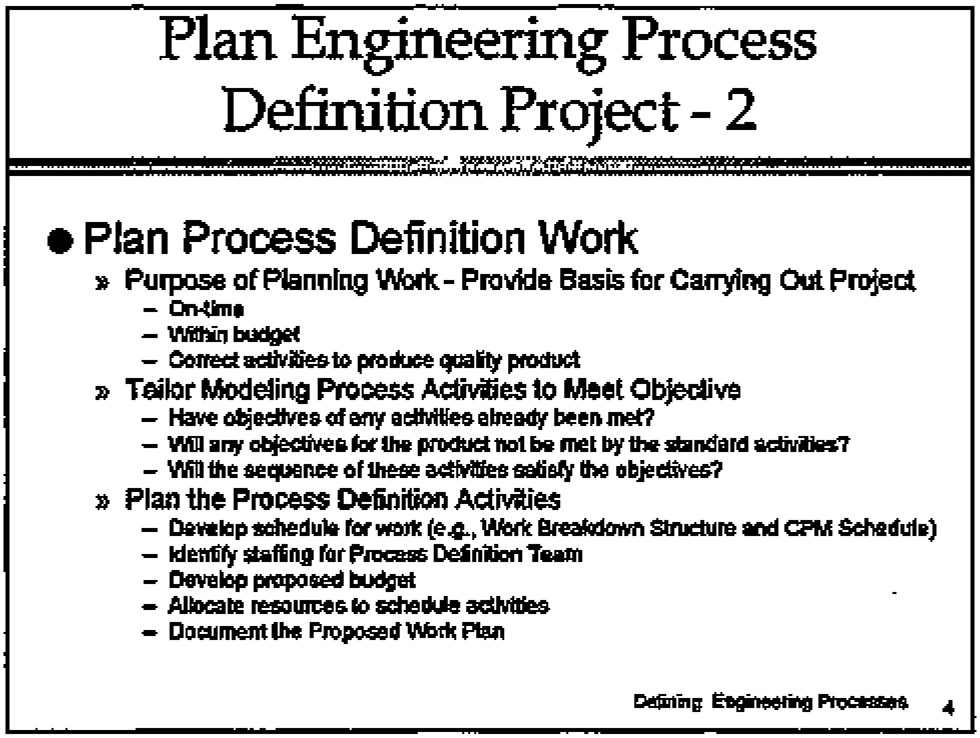
Outline - Approach to Defining Engineering Processes

- Prepare for Engineering Process Modeling
  - » Plan Process Definition Project
  - » Gain Management Approval
- Model Engineering Process
  - » Collect Data on Engineering Process
  - » Construct Engineering Process Model
  - » Verify and Validate Process Model



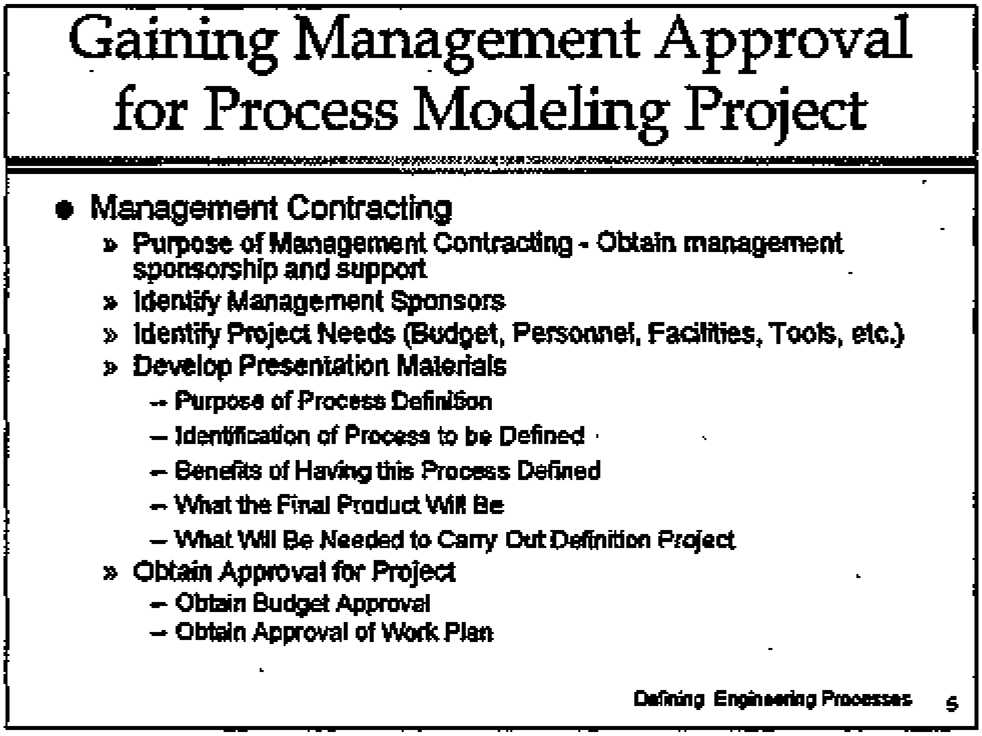
Plan Engineering Process Definition Project - 1

- Plan Engineering Definition Product
  - » Purpose of Planning Product - Establish Objectives of the Proposed Product
    - – Ensure Model Will Satisfy Users' Needs (e.g., correct scope, perspective, and views)
    - – Establish Criteria to Verify and Validate the Model (e.g. project exit criteria)
  - » Define Purpose of Model
    - – What is to be achieved by having model? (e.g., aid understanding, standardize process, training, basis for process improvement, etc.)
  - » Identify Audiences
    - – Who will use the mode? l (e.g., Senior Management, Engineering Management, System Developers, New Employees, etc.)
  - » Define Usage
    - – How will each different audience use the model?



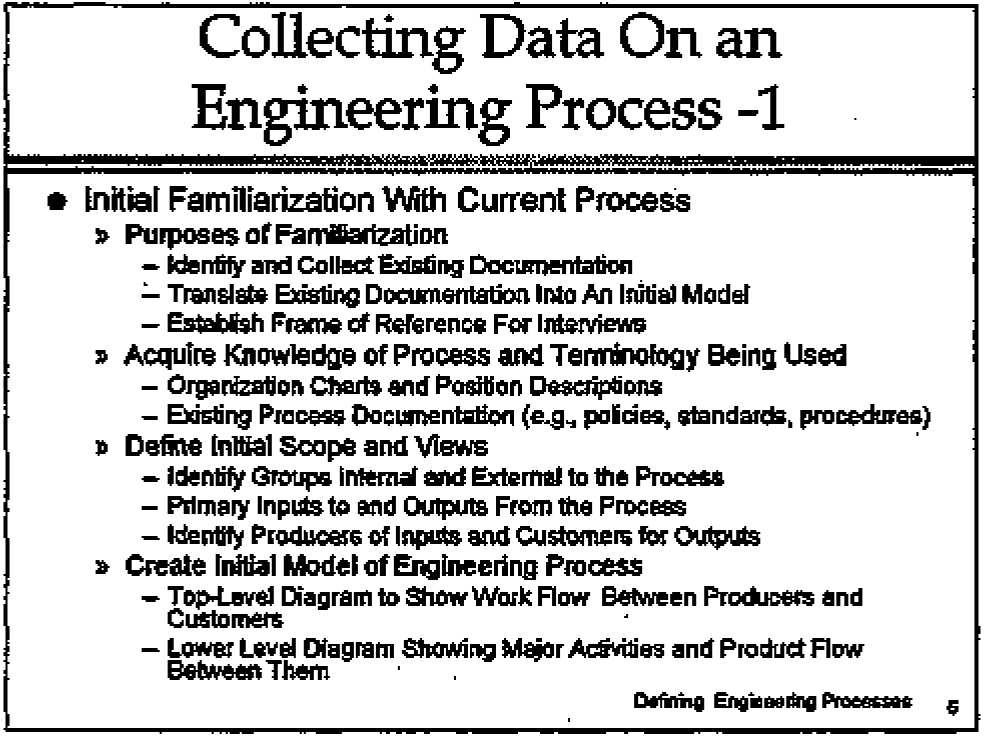
Plan Engineering Process Definition Project - 2

- Plan Process Definition Work
  - » Purpose of Planning Work - Provide Basis for Carrying Out Project
    - – On-time
    - – Within budget
    - – Correct activities to produce quality product
  - » Tailor Modeling Process Activities to Meet Objective
    - – Have objectives of any activities already been met?
    - – Will any objectives for the product not be met by the standard activities?
    - – Will the sequence of these activities satisfy the objectives?
  - » Plan the Process Definition Activities
    - – Develop schedule for work (e.g., Work Breakdown Structure and CPM Schedule)
    - – Identify staffing for Process Definition Team
    - – Develop proposed budget
    - – Allocate resources to schedule activities
    - – Document the Proposed Work Plan



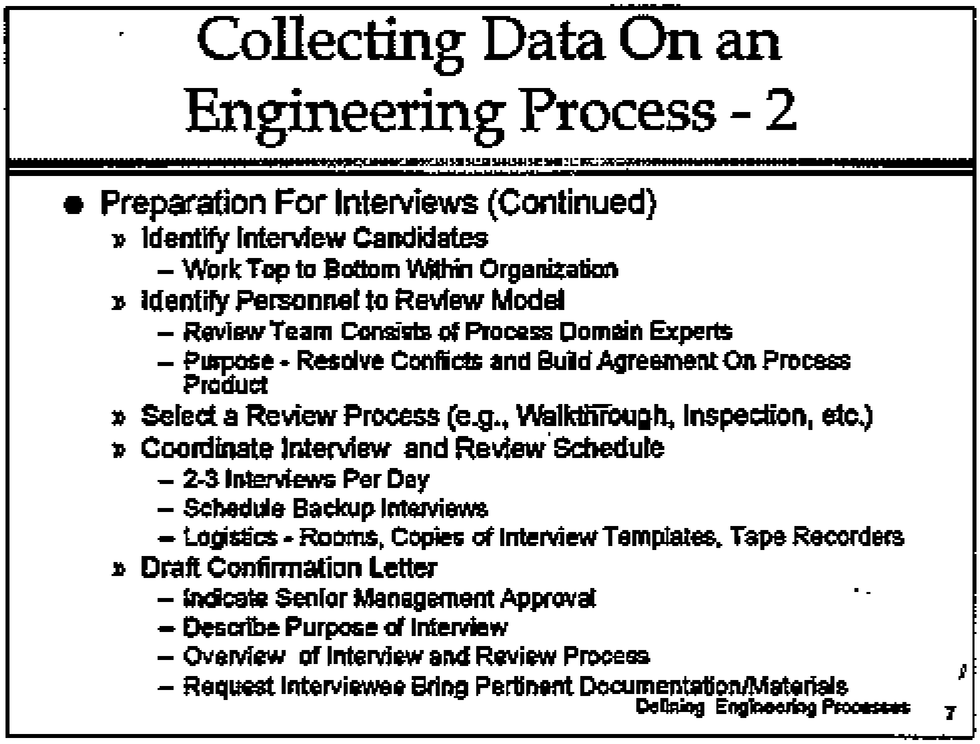
Gaining Management Approval for Process Modeling Project

- Management Contracting
  - » Purpose of Management Contracting - Obtain management sponsorship and support
  - » Identify Management Sponsors
  - » Identify Project Needs (Budget, Personnel, Facilities, Tools, etc.)
  - » Develop Presentation Materials
    - – Purpose of Process Definition
    - – Identification of Process to be Defined
    - – Benefits of Having this Process Defined
    - – What the Final Product Will Be
    - – What Will Be Needed to Carry Out Definition Project
  - » Obtain Approval for Project
    - – Obtain Budget Approval
    - – Obtain Approval of Work Plan



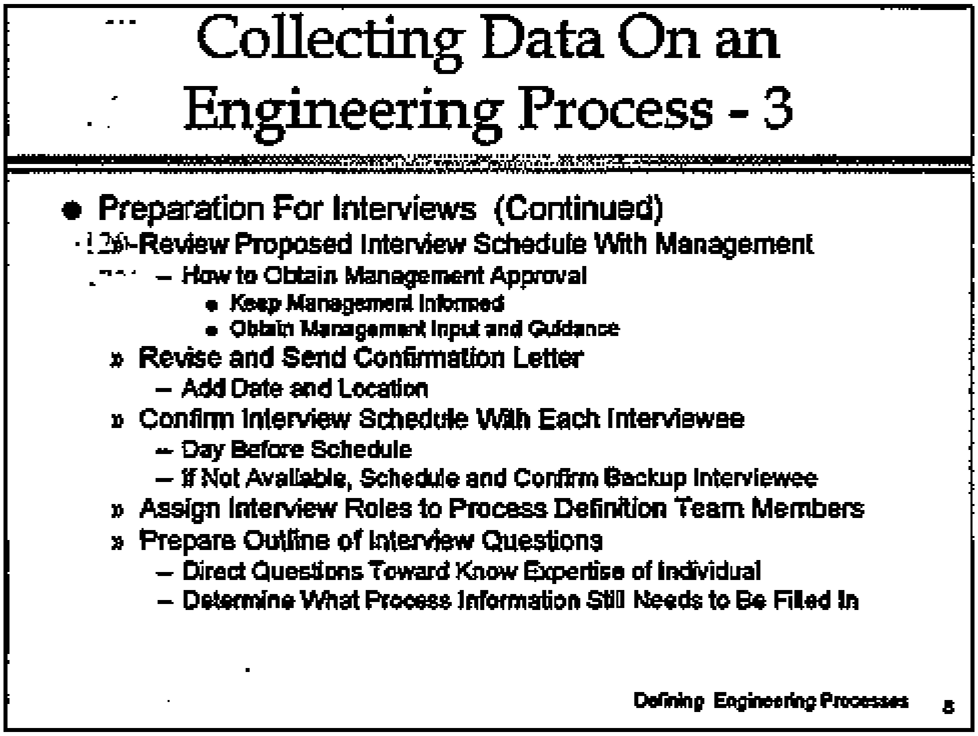
Collecting Data On an Engineering Process -1

- Initial Familiarization With Current Process
  - » Purposes of Familiarization
    - – Identify and Collect Existing Documentation
    - – Translate Existing Documentation Into An Initial Model
    - – Establish Frame of Reference For Interviews
  - » Acquire Knowledge of Process and Terminology Being Used
    - – Organization Charts and Position Descriptions
    - – Existing Process Documentation (e.g., policies, standards, procedures)
  - » Define Initial Scope and Views
    - – Identify Groups Internal and External to the Process
    - – Primary Inputs to and Outputs From the Process
    - – Identify Producers of Inputs and Customers for Outputs
  - » Create Initial Model of Engineering Process
    - – Top-Level Diagram to Show Work Flow Between Producers and Customers
    - – Lower Level Diagram Showing Major Activities and Product Flow Between Them



Collecting Data On an Engineering Process - 2

- Preparation For Interviews (Continued)
  - » Identify Interview Candidates
    - – Work Top to Bottom Within Organization
  - » Identify Personnel to Review Model
    - – Review Team Consists of Process Domain Experts
    - – Purpose - Resolve Conflicts and Build Agreement On Process Product
  - » Select a Review Process (e.g., Walkthrough, Inspection, etc.)
  - » Coordinate Interview and Review Schedule
    - – 2-3 Interviews Per Day
    - – Schedule Backup Interviews
    - – Logistics - Rooms, Copies of Interview Templates, Tape Recorders
  - » Draft Confirmation Letter
    - – Indicate Senior Management Approval
    - – Describe Purpose of Interview
    - – Overview of Interview and Review Process
    - – Request Interviewee Bring Pertinent Documentation/Materials



Collecting Data On an Engineering Process - 3

- Preparation For Interviews (Continued)
  - » Review Proposed Interview Schedule With Management
    - – How to Obtain Management Approval
      - Keep Management Informed
      - Obtain Management Input and Guidance
  - » Revise and Send Confirmation Letter
    - – Add Date and Location
  - » Confirm Interview Schedule With Each Interviewee
    - – Day Before Schedule
    - – If Not Available, Schedule and Confirm Backup Interviewee
  - » Assign Interview Roles to Process Definition Team Members
  - » Prepare Outline of Interview Questions
    - – Direct Questions Toward Know Expertise of Individual
    - – Determine What Process Information Still Needs to Be Filed In

Collecting Data On an Engineering Process - 4

- Interviewing Process Domain Experts
  - » Introductions by Point of Contact
  - » State of Purpose of Interviews
  - » State of Ground Rules
    - – Non-attribution, confidentiality
    - – If interviewee has no objections, tape interview
  - » State Scope and Perspective
    - – Focus on normal activity, not exceptions
    - – Proposed breadth and depth of process being discussed
  - » Describe Interview Process
    - – Ask interviewee to think in terms of
      - Activities/sub-activities, and their sequence
      - Product flows through those activities
      - Inputs to and outputs from Each activity
      - Controls and Constraints on each activity
      - Standards and procedures applied
      - Templates and forms used

Collecting Data On an Engineering Process - 5

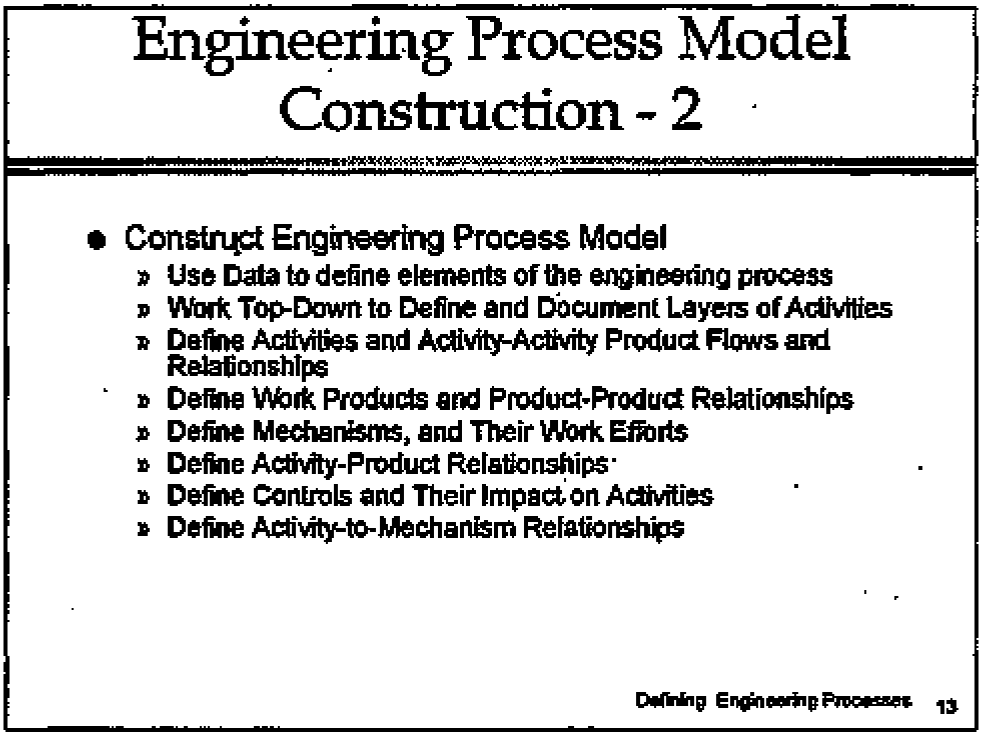
- Interviewing Process Domain Experts
  - » Gather Data - (Refer to Building Blocks Chart in Session 4)
    - – Collect personal data (name, address, phone number) for follow-up
    - – Establish interviewee's role in process
    - – Related information to Initial Process Model
    - – Document essential process elements for each activity/sub-activity
    - – Identify pending issues requiring further investigation
    - – Define action items, and assign individual responsibilities and due dates
  - » Summarize Information Gathered
    - – Restate issues and action items
    - – Request suggestions for process improvement

Collecting Data On an Engineering Process - 6

- Analysis of Interview Results
  - » After Each Interview Process Definition Team
    - – Review s Results Interview
    - – Correlates/Consolidate Team Member Understandings, Notes, Perceptions
    - – Consolidates Findings Into Master Template
    - – Identifies Additional Needs for
      - New Data
      - Confirmation of Data Gathered to Eliminate Conflicts
  - » After Interviews Have Been Completed
    - – Analyze Data Gathered For
      - Completeness
      - Correctness
      - Consistency
      - Significance
    - – Document Issues, Findings, and Assumptions
    - – Elaborate Initial Model

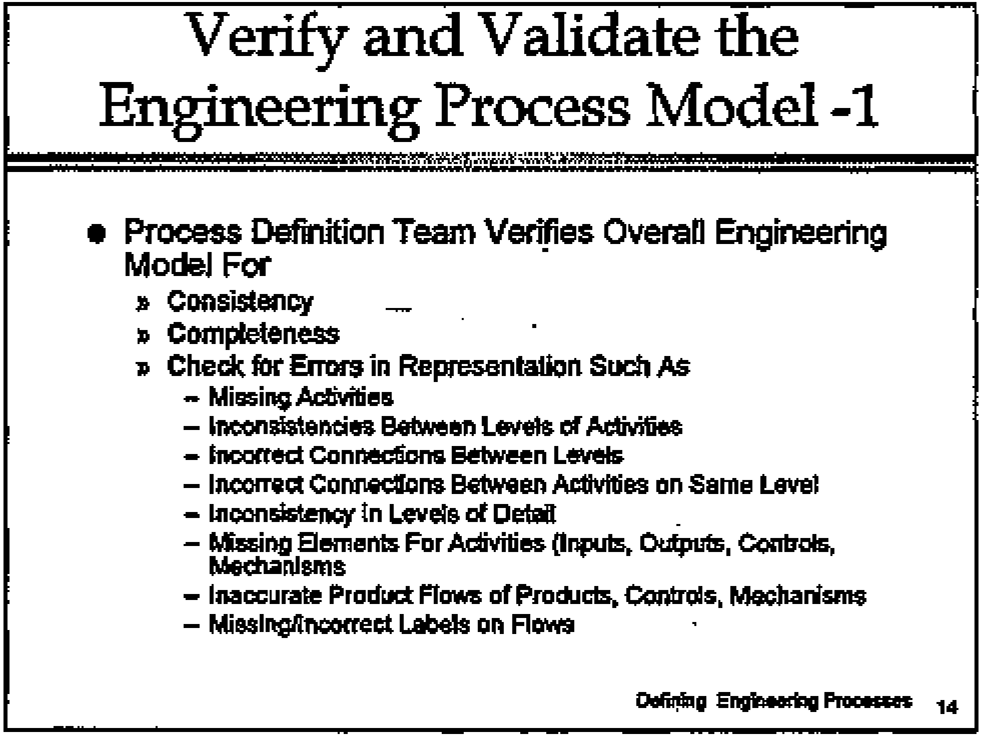
Engineering Process Model Construction -1

- Verify Engineering Process Data to Identify
  - » Missing Data
  - » Incorrect Data
  - » Inconsistent/Conflicting Data
  - » Insignificant Data
- Resolve Data Shortcomings
  - » Additional Research
  - » Additional Interviews
  - » Pre-interview Interviewees Involved in Inconsistent/Conflicting Data
  - » Discussions With Engineering Managers



Engineering Process Model Construction - 2

- Construct Engineering Process Model
  - » Use Data to define elements of the engineering process
  - » Work Top-Down to Define and Document Layers of Activities
  - » Define Activities and Activity-Activity Product Flows and Relationships
  - » Define Work Products and Product-Product Relationships
  - » Define Mechanisms, and Their Work Efforts
  - » Define Activity-Product Relationships
  - » Define Controls and Their Impact on Activities
  - » Define Activity-to-Mechanism Relationships



Verify and Validate the Engineering Process Model -1

- Process Definition Team Verifies Overall Engineering Model For
  - » Consistency
  - » Completeness
  - » Check for Errors in Representation Such As
    - – Missing Activities
    - – Inconsistencies Between Levels of Activities
    - – Incorrect Connections Between Levels
    - – Incorrect Connections Between Activities on Same Level
    - – Inconsistency in Levels of Detail
    - – Missing Elements For Activities (Inputs, Outputs, Controls, Mechanisms
    - – Inaccurate Product Flows of Products, Controls, Mechanisms
    - – Missing/Incorrect Labels on Flows

Verify and Validate the Engineering Process Model - 2

- Conduct Engineering Model Review
  - » Introduce Process Definition Team and Review Team
  - » Describe Review Purpose and Methodology
  - » Conduct Step-by-Step Walkthrough, Inspection, or Audit of Engineering Process Model
- Review Team Validates Model By Determining If
  - » Model Meets Objectives (Refer to Exit Criteria Developed During Planning Efforts)
  - » Model Describes Current Behavior of the Process Within the Specified Perspective, Scope, and Purpose
- Outbrief Engineering Process Model
  - » Present Validated Process Model to Management
  - » Review Outstanding Issues and Action Items
  - » Present Findings on Potential Improvements
  - » Define Proposal For Next Iteration of Improvement

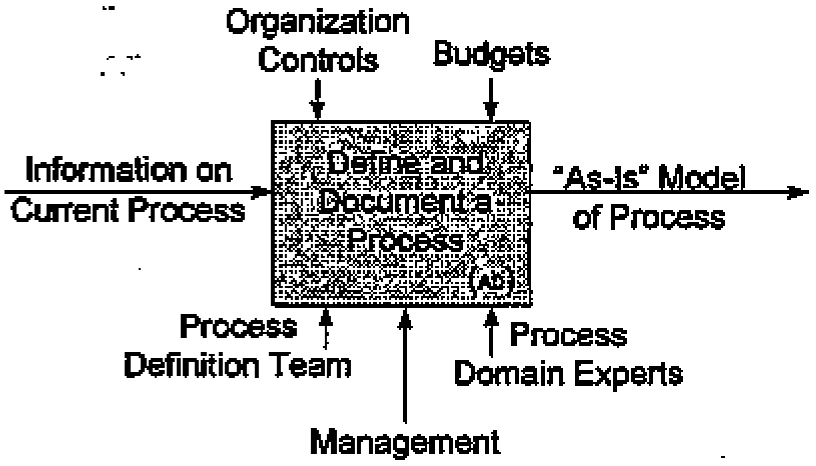
Examples of Verification and Validation Considerations

- Elements of Verification
  - » Model Is Understandable
  - » Model Accurately Portrays Either
    - – "As-Is" Process
    - – "To-Be" Process
  - » Model is Complete, Internally Consistent, Concise, and Accurate
  - » Models Demonstrating Different Perspectives and Viewpoints Are Consistent With Each Other
  - » Model Perspectives and Viewpoints Are Correct for Their Intended Audiences
- Elements of Validation
  - » Model Versions Meet the Needs of Their Associated Audience
  - » Scope of Model Is Correct
  - » Model Will Support Planning, Performing, Quality Evaluation, and Process Improvement
  - » Model Is Documented in Formally Defined Syntax and Semantics

IDEF Model

Process for Defining Engineering Processes

First-Level-Model Process Definition Process



Organization Controls

Budgets

Information on Current Process

Define and Document a Process

(A0)

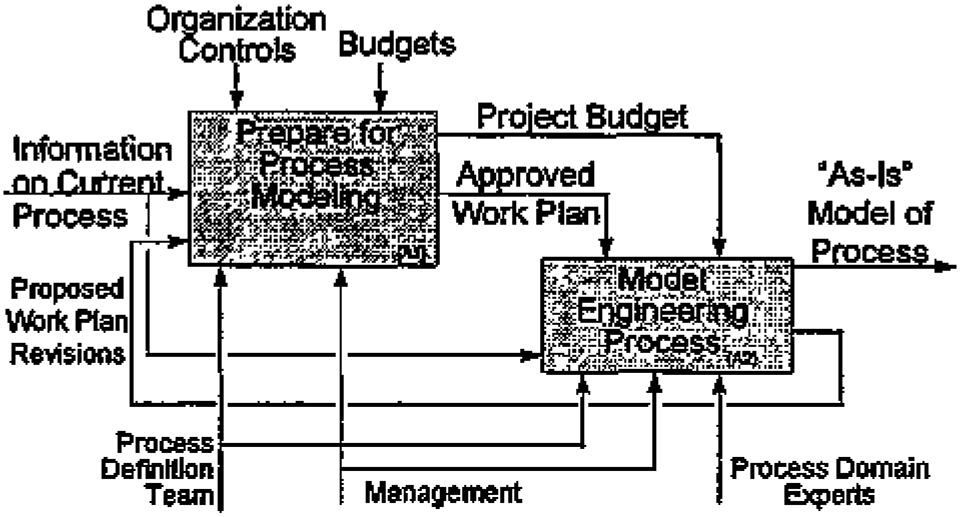
"As-Is" Model of Process

Process Definition Team

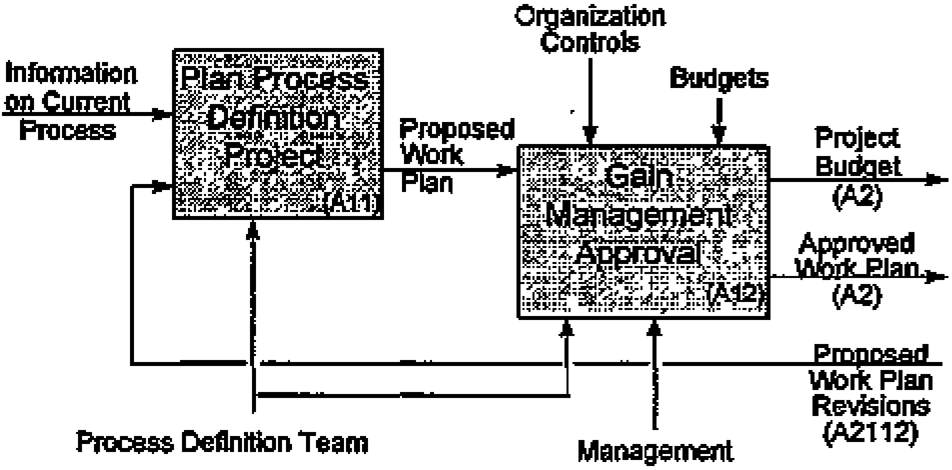
Management

Process Domain Experts

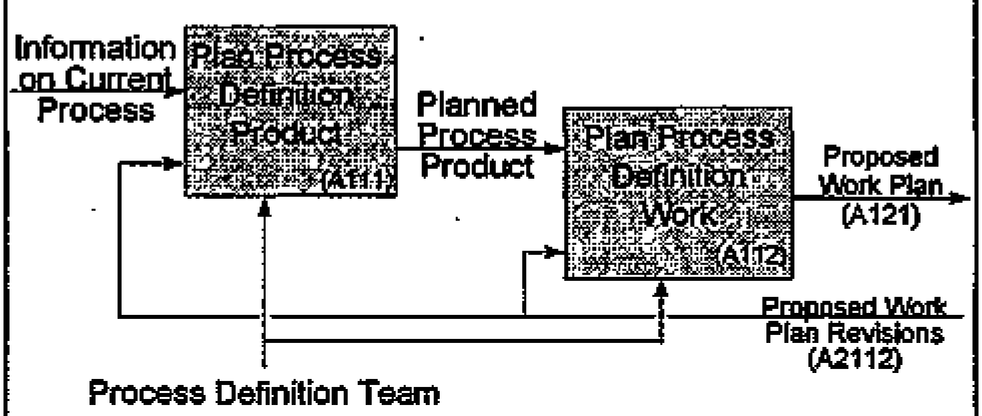
Define and Document an Engineering Process (A0)



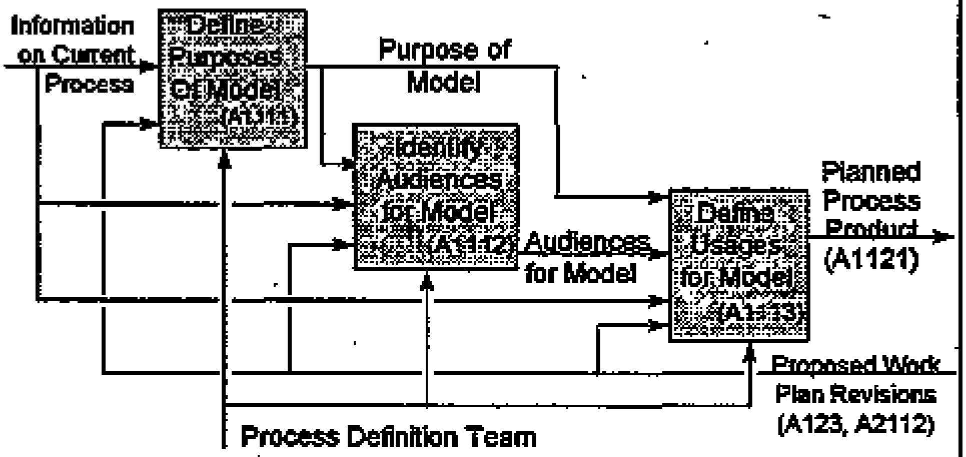
Prepare For Engineering Process Modeling (A1)



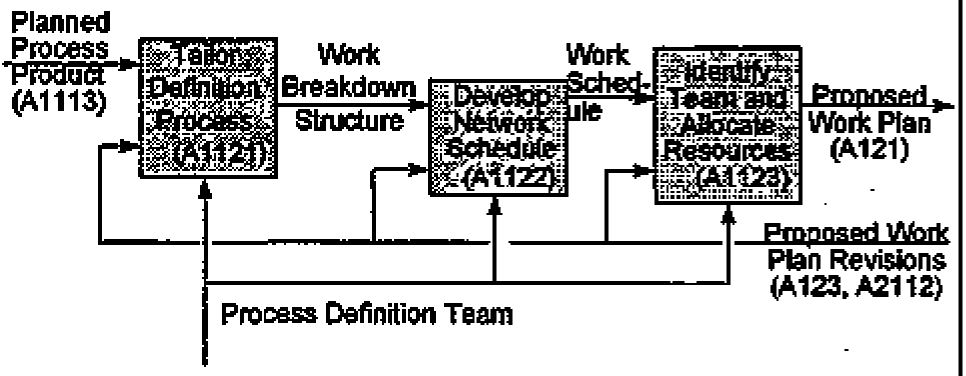
Plan Process Definition Project (A11)


Information on Current Process
Plan Process Definition Product
(A111)
Planned Process Product
Plan Process Definition Work
(A112)
Proposed Work Plan (A121)
Proposed Work Plan Revisions (A2112)
Process Definition Team

Plan Process Definition Product (A111)


Information on Current Process
Define Purposes Of Model
(A1111)
Purpose of Model
Identify Audiences for Model
(A1112)
Audiences for Model
Define Usages for Model
(A1113)
Planned Process Product (A1121)
Proposed Work Plan Revisions (A123, A2112)
Process Definition Team

Plan Process Definition Work (A112)



Planned Process Product (A1113)

Tailor Definition Process (A1121)

Work Breakdown Structure

Develop Network Schedule (A1122)

Work Schedule

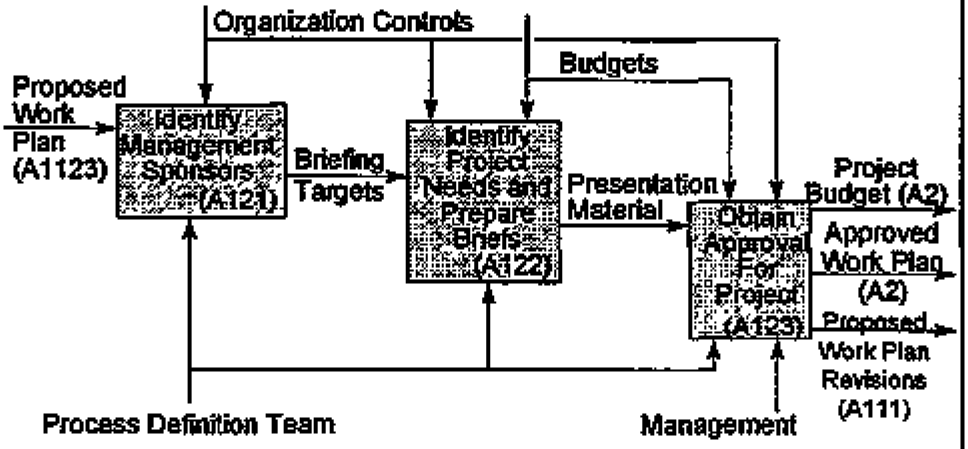
Identify Teams and Allocate Resources (A1123)

Proposed Work Plan (A121)

Proposed Work Plan Revisions (A123, A2112)

Process Definition Team

Gain Management Approval (A12)



Organization Controls

Budgets

Proposed Work Plan (A1123)

Identify Management Sponsors (A121)

Briefing Targets

Identify Project Needs and Prepare Briefs (A122)

Presentation Material

Obtain Approval For Project (A123)

Project Budget (A2)

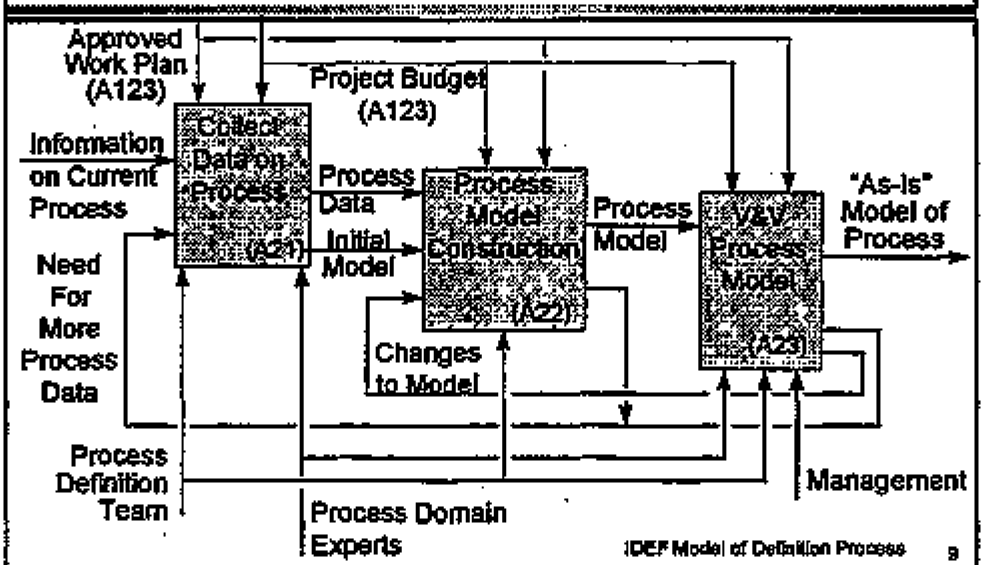
Approved Work Plan (A2)

Proposed Work Plan Revisions (A111)

Process Definition Team

Management

Model Engineering Process (A2)



Approved Work Plan (A123)

Project Budget (A123)

Information on Current Process

Collect Data on Process (A21)

Process Data

Initial Model

Process Model Construction (A22)

Process Model

V&V Process Model (A23)

"As-Is" Model of Process

Need For More Process Data

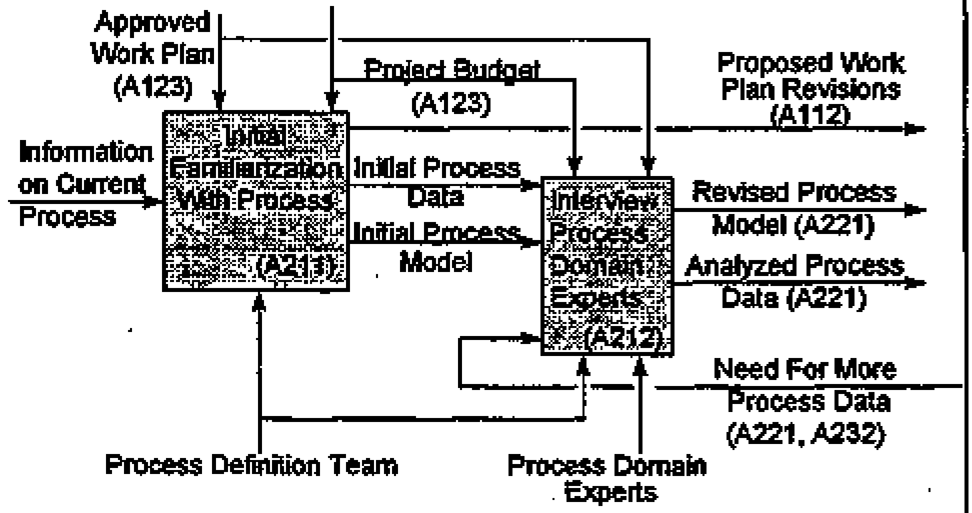
Changes to Model

Process Definition Team

Process Domain Experts

Management

Collect Data on Engineering Process (A21)



Approved Work Plan (A123)

Project Budget (A123)

Proposed Work Plan Revisions (A112)

Information on Current Process

Initial Familiarization With Process (A211)

Initial Process Data

Initial Process Model

Interview Process Domain Experts (A212)

Revised Process Model (A221)

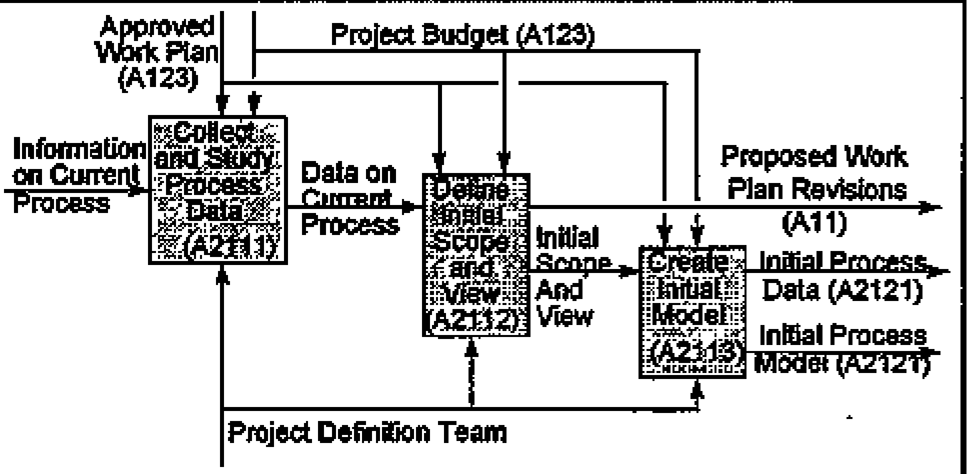
Analyzed Process Data (A221)

Need For More Process Data (A221, A232)

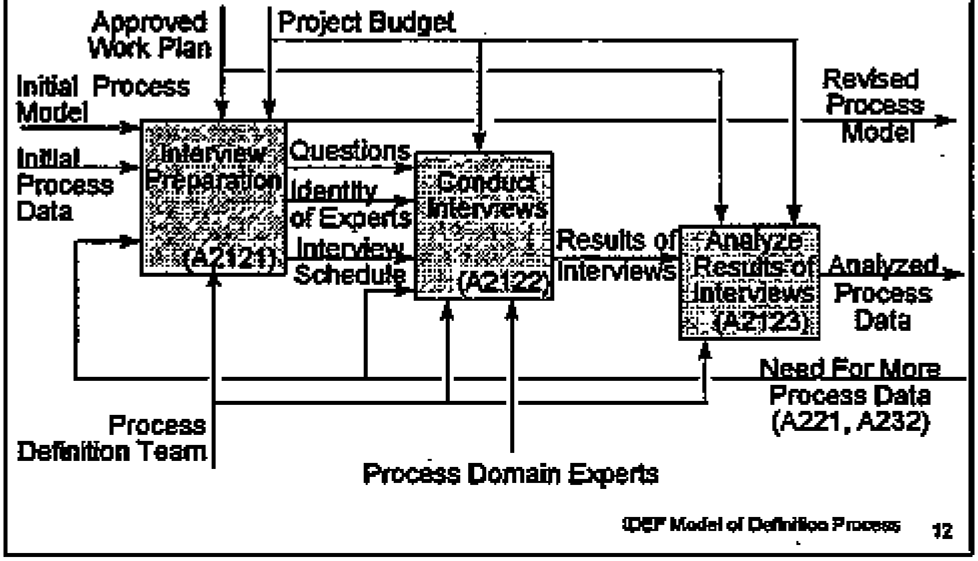
Process Definition Team

Process Domain Experts

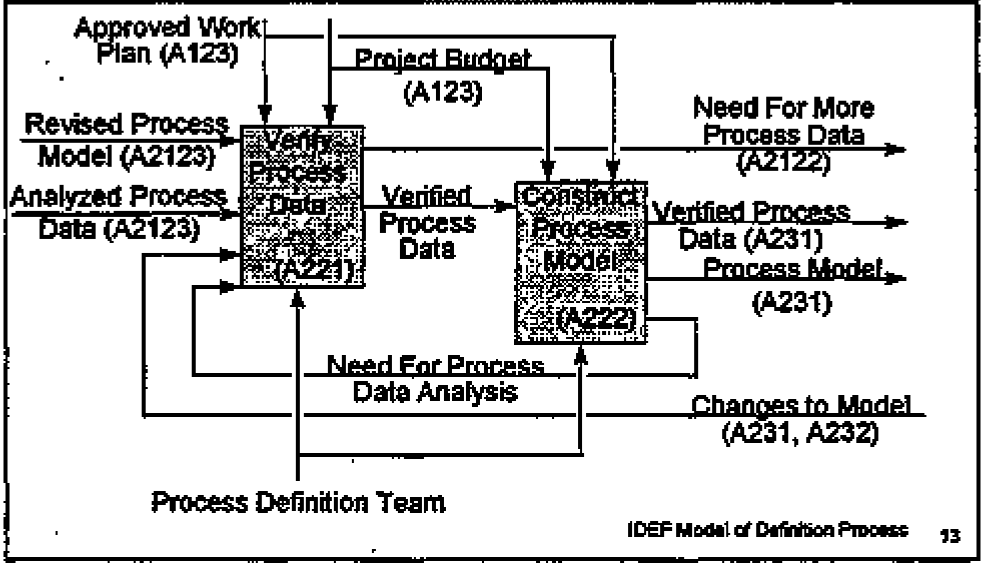
Initial Familiarization With Engineering Process (A211)


Approved Work Plan (A123)
Project Budget (A123)
Information on Current Process
Collect and Study Process Data (A2111)
Data on Current Process
Define Initial Scope and View (A2112)
Initial Scope And View
Create Initial Model (A2113)
Proposed Work Plan Revisions (A11)
Initial Process Data (A2121)
Initial Process Model (A2121)
Project Definition Team

Interview Process Domain Experts (A212)



Engineering Process Model Construction (A22)



Approved Work Plan (A123)

Project Budget (A123)

Revised Process Model (A2123)

Analyzed Process Data (A2123)

Verify Process Data (A221)

Verified Process Data

Construct Process Model (A222)

Need For More Process Data (A2122)

Verified Process Data (A231)

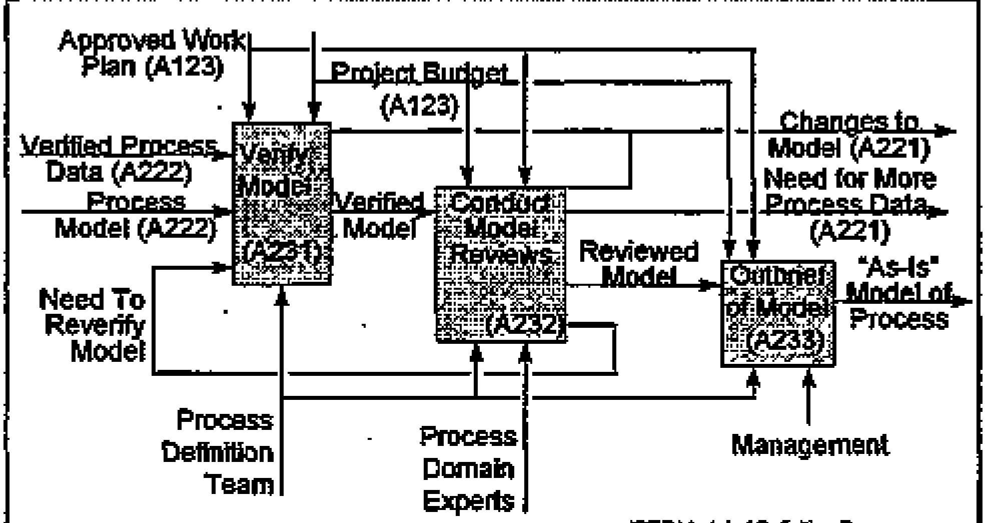
Process Model (A231)

Need For Process Data Analysis

Changes to Model (A231, A232)

Process Definition Team

Verify and Validate Engineering Process Model (A23)



Approved Work Plan (A123)

Project Budget (A123)

Verified Process Data (A222)

Process Model (A222)

Verify Model (A231)

Verified Model

Conduct Model Reviews (A232)

Reviewed Model

Outbrief of Model (A233)

Changes to Model (A221)

Need for More Process Data (A221)

"As-Is" Model of Process

Need To Reverify Model

Process Definition Team

Process Domain Experts

Management

1997 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Session Y1: How the NWC Handles Software as Product

Chair Dave Vinson

Pantex Plant
Software Quality Assurance Subcommittee
Work Item #16

How the NWC Handles Software as Product

Software Quality Assurance Subcommittee

Work Item #16

Presenters: Member(s) of SQAS WI#16: Management and Control of Product Software

Summary: This tutorial provides a hands-on view of how the Nuclear Weapons Complex projects should be handling (or planning to handle) software as a product in response to Engineering Procedure 401099. The SQAS has published the document SQAS96-002, "Guidelines for NWC Processes for Handling Software Product," that will be the basis for the tutorial. The primary scope of the tutorial is on software products that result from weapons and weapons-related projects, although the information presented is applicable to many software projects. Processes that involve the exchange, review, or evaluation of software product between or among NWC sites, DOE, and external customers will be described. These processes include:

1. **Identification:** what are software product items, how are the product and items identified, how does software identification relate to system identification.
2. **Qualification:** what is software qualification in accordance with EP401099, who is involved, how does a software Process Realization Team work, what is in a Qualification Plan and how does this Plan lead to a Qualification Evaluation Release.
3. **Acceptance:** how does DOE accept software product, what is a Quality Assurance Inspection Procedure, how are product qualification and acceptance related, what are site and DOE roles, what is needed for customer use (interagency and external end-use).
4. **Delivery:** what is the mechanism for shipping and receiving software product, how is delivery accomplished between NWC sites, how is delivery accomplished between a site and external customer.

A Case Study of a recently completed project will be given to each participant for hands-on review of how the guidelines for handling software product have been applied. In particular, examples of project products used in the handling processes that will be reviewed include: Material List, Qualification Plan, Software Requirements, Test Plan, Maintenance Plan, Software Production Requirements, Traveler, Product-Disk Labels, Integrated Contractor Order, Certificate of Inspection, Shipper Label, Package Label, Complete Engineering Release, Qualification Evaluation Release, and Quality Assurance Inspection Procedure.

Site-specific issues and the tailoring of the handling guidelines for use in non-weapons applications will be discussed. Members from several sites who are on the SQAS WI#16 Working Group will be available to discuss the site-specific issues.

Hand-Out Material:

1. Tutorial Slides
2. SQAS96-002, "Guidelines for NWC Processes for Handling Software Product," June 1996.
3. Case Study Notebook

Audience/Restrictions:

This tutorial is primarily intended for personnel who are or will be managing, developing or supporting software that will be delivered to or used by external customers. Tutorial participants must be a Department of Energy or Nuclear Weapons Complex employee. Although none of the material in this tutorial is classified, its content may be sensitive. A valid badge will be required for participants in this tutorial. If you have a question as to whether you can participate, contact a Forum representative.

Contact Information:

Dr. David E. Peercy
Sandia National Laboratories
P.O. Box 5800, MS0638
Albuquerque, NM 87185-0638
505-844-7965(voice), 505-844-3920(fax), depeerc@sandia.gov

BIOGRAPHY

The Software Quality Assurance Subcommittee (SQAS) operates under the DOE Nuclear Weapons Complex (NWC) Quality Managers to identify and resolve Software Quality issues and problems common to all DOE sites and facilities. This tutorial is the result of an NWC SQAS work item to define how to manage and control software as product. The work item was established to satisfy a need to define a consistent process for handling product software. The Nuclear Weapons Complex-wide participants and presenters of this tutorial include:

Chair David Vinson, Mason & Hanger, Pantex Plant
Phil Huffman, Mason & Hanger, Pantex Plant
Alvin Cowen, Mason & Hanger, Pantex Plant
Catherine Kuhn, AlliedSignal Aerospace, Federal Manufacturing & Technologies
Donald Schilling, AlliedSignal Aerospace, Federal Manufacturing & Technologies
Dave Peercy, Sandia National Laboratories
Mike Blackledge, Sandia National Laboratories
Orval Hart, Los Alamos National Laboratory
John Cerutti, Los Alamos National Laboratory
Bill Warren, Lawrence Livermore National Laboratory
Charles Chow, Lawrence Livermore National Laboratory
Ellis Sykes, Department of Energy, Kansas City Office
Gary Echert, Department of Energy, Albuquerque Office
Kathleen Canal, Department of Energy, Headquarters
Ray Cullen, Westinghouse, Savannah River Site
Faye Brown, Lockheed Martin Energy Systems, Oak Ridge, Y-12 Plant

Dave Vinson, Chair WI#16
Mason & Hanger Pantex Plant
Bldg 12-102
P.O. Box 30020
Amarillo, TX 79120-0020
Voice: 806-477-4739
Fax: 806-477-4350
E-mail: dvinson@pantex.com

How the NWC Handles

4/1/97 Pg. 1

Software as Product

presented by

Software Quality Assurance Subcommittee

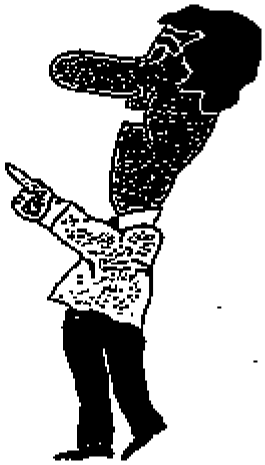
Work Item #16

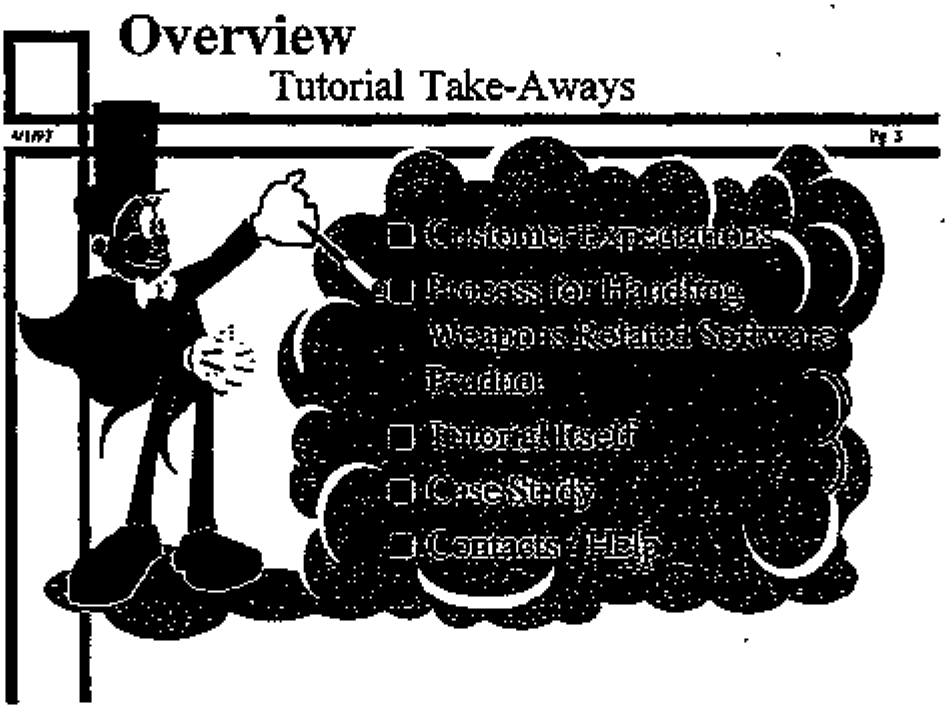
1997 Software Quality Forum
April 1, 1997

Agenda

4/1/97 Pg. 2

- Overview
- Introduction to Case Study
- Identification Process
- Qualification Process
- Acceptance Process
- Delivery Process
- Specific Concerns
- Future Direction





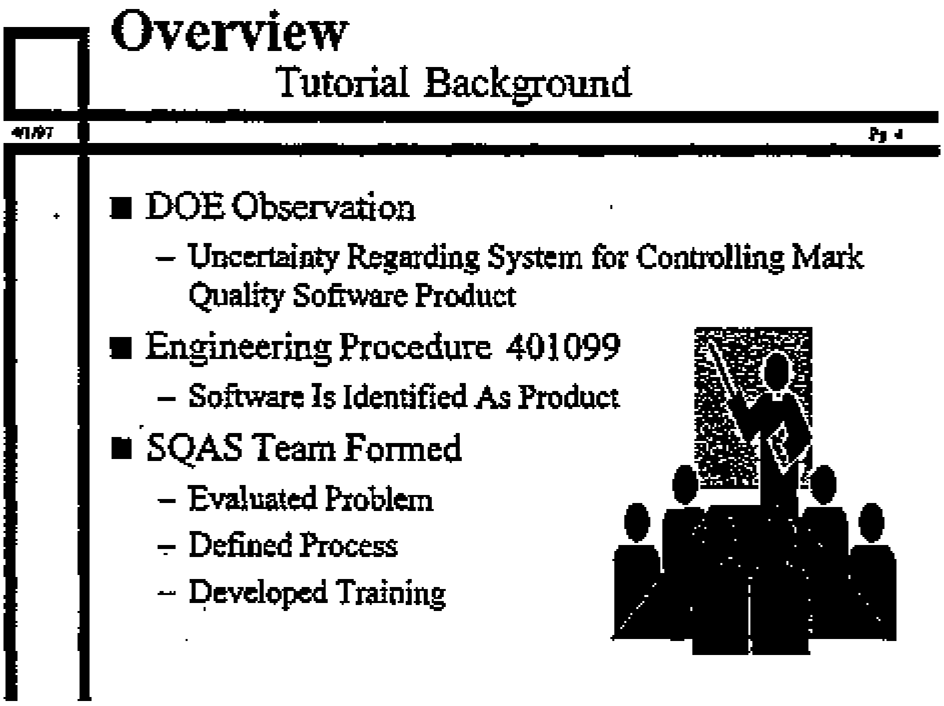
Overview

Tutorial Take-Aways

4/1/97

Pg 3

- □ Customer Expectations
- □ Process for Handling Weapons Related Software Product
- □ Tutorial Itself
- □ Case Study
- □ Contacts / Help



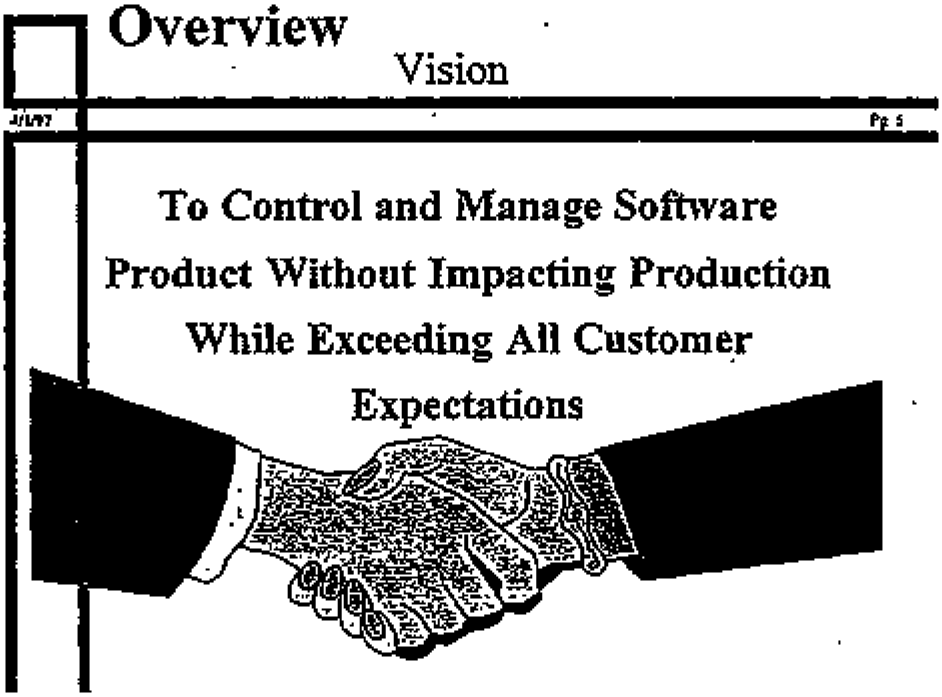
Overview

Tutorial Background

4/1/97

Pg 4

- ■ DOE Observation
  - – Uncertainty Regarding System for Controlling Mark Quality Software Product
- ■ Engineering Procedure 401099
  - – Software Is Identified As Product
- ■ SQAS Team Formed
  - – Evaluated Problem
  - – Defined Process
  - – Developed Training



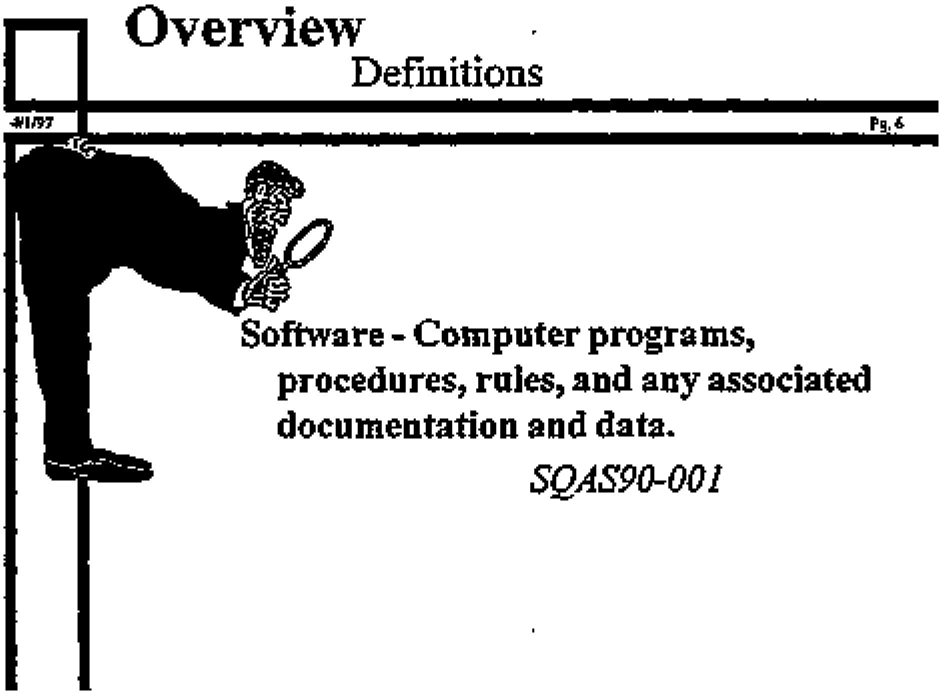
Overview

Vision

4/1/97

Pg 5

To Control and Manage Software Product Without Impacting Production While Exceeding All Customer Expectations



Overview

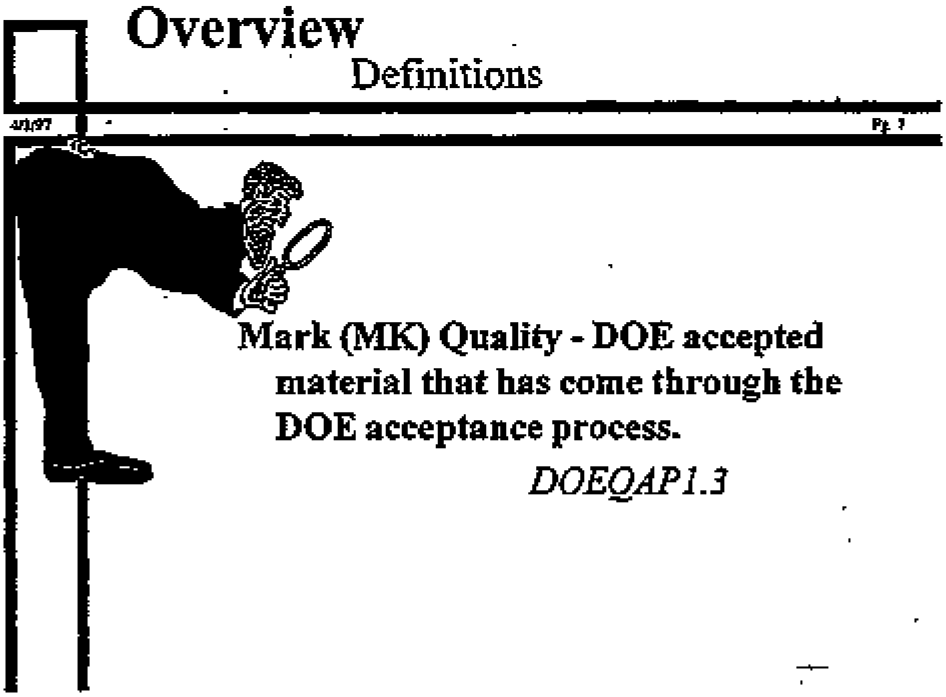
Definitions

4/1/97

Pg. 6

Software - Computer programs, procedures, rules, and any associated documentation and data.

SQAS90-001



Overview

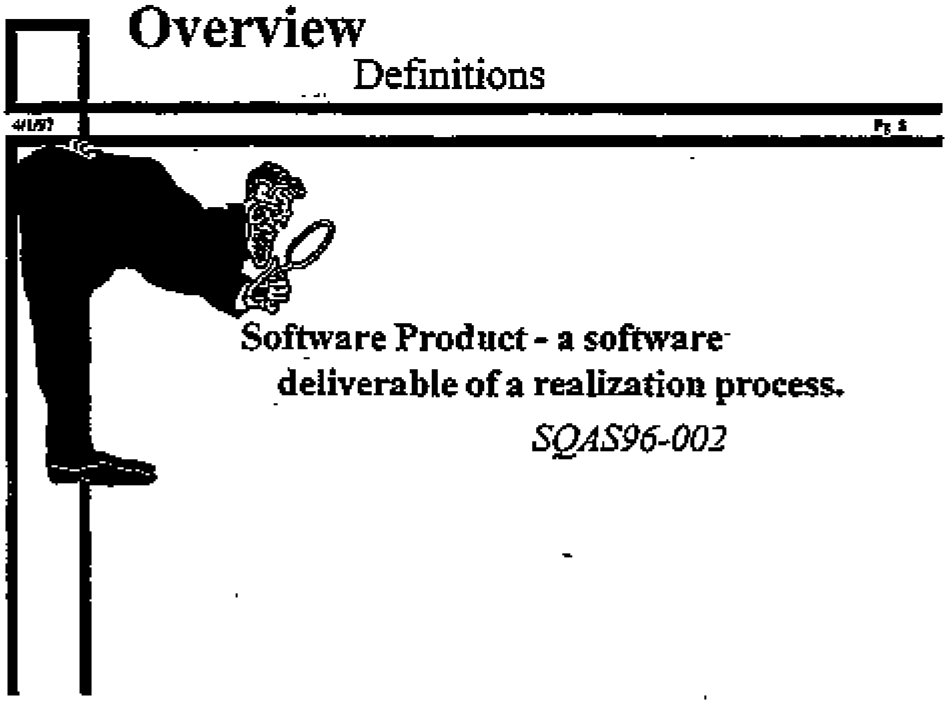
Definitions

4/1/97

Pg 7

Mark (MK) Quality - DOE accepted material that has come through the DOE acceptance process.

DOEQAP1.3



Overview

Definitions

4/1/97

Pg 8

Software Product - a software deliverable of a realization process.

SQAS96-002

Overview
Tutorial Scope

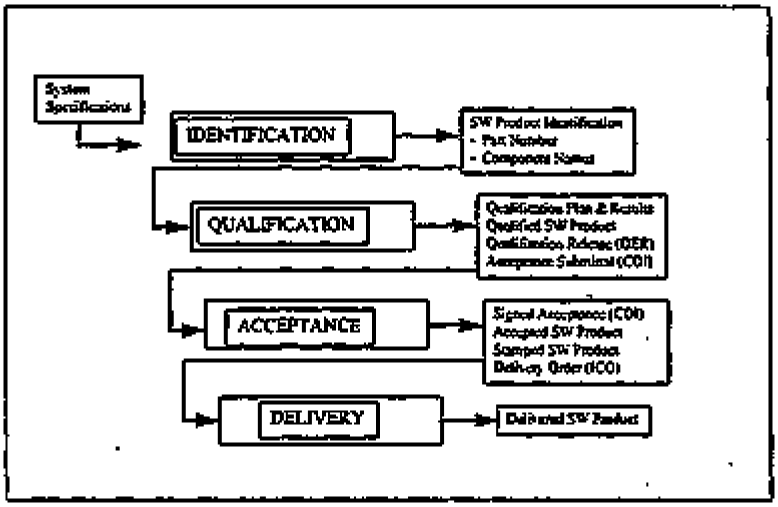
4/1/97 Pg 9

Software Product is Software that's:

- Created by a DOE Contractor
- Qualified by the Contractor
- Accepted and Stamped by DOE
- Shipped as a Product

Overview
Process Summary

4/1/97 Pg. 10



System Specifications

IDENTIFICATION

SW Product Identification
- Part Number
- Component Names

QUALIFICATION

Qualification Plan & Results
Qualified SW Product
Qualification Release (DER)
Acceptance Submittal (COI)

ACCEPTANCE

Signed Acceptance (COI)
Accepted SW Product
Stamped SW Product
Delivery Order (ICO)

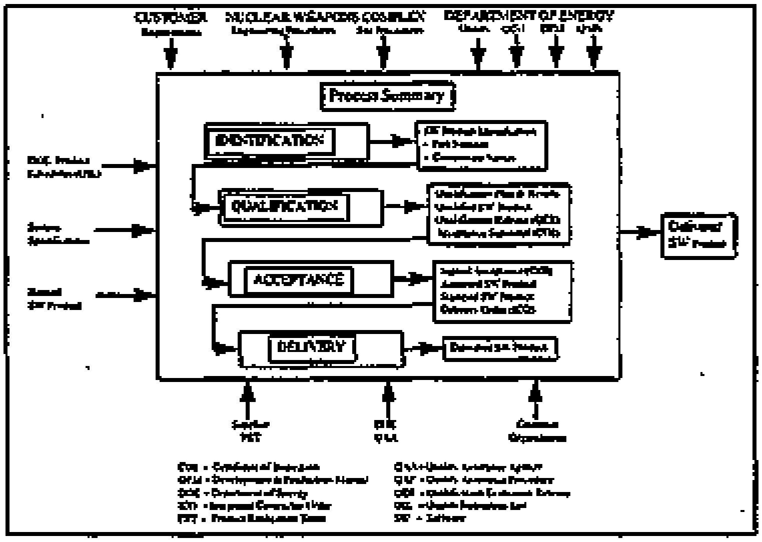
DELIVERY

Delivered SW Product

Overview

Process Summary

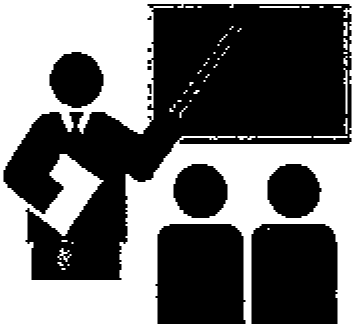
4/1/97 Pg. 11



Introduction to Case Study

4/1/97 Pg. 12

- Overview of Application
- Identification Materials
- Qualification Materials
- Acceptance Materials
- Delivery Materials



Introduction to Case Study

Overview of Application

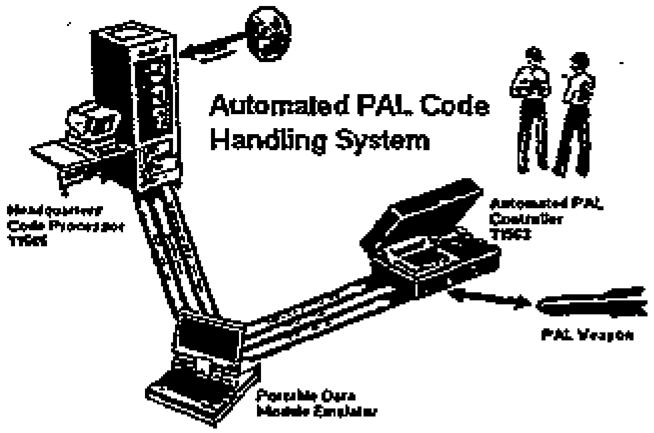
4/1/97 Pg. 13

- General Application System
  - Use control
  - T1565A, replacement for T1565 Headquarters Code Processor
- Case Study Component
  - Cryptographic Processor Firmware Software
- Life Cycle Logistics
  - Developed at Sandia National Labs
  - Qualified by Sandia, accepted by DOE/AL
  - Delivered to Kansas City Plant for loading into programmable read only memory

Introduction to Case Study

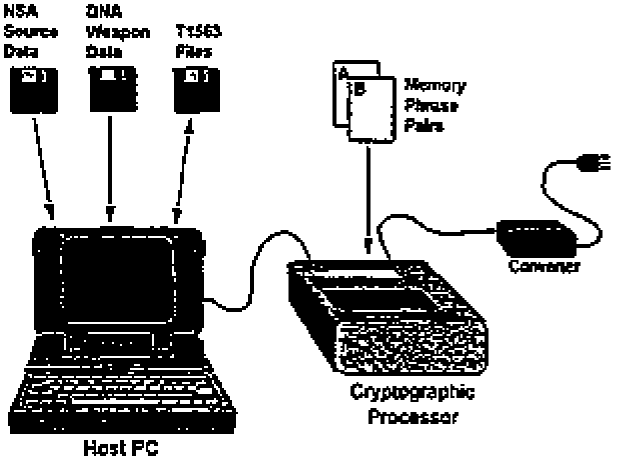
APCHS Topology

4/1/97 Pg. 14



Introduction to Case Study

T1565A Operational Topology



Introduction to Case Study

Firmware Software Functions

- Hardware
  - Initializes/activates some hardware devices
  - Verifies firmware integrity
  - Performs self-test on hardware components
- Software
  - Verifies integrity of operational code
  - Copies operational code from NVRAM to RAM
  - Initiates execution of operational code
  - Provides RAM clear and SHA functions

Introduction to Case Study

Identification Materials

4/1/97 Pg 17

- Part Number
  - FWSW: 704308-00
- Software Development Support Drawings
  - SR704308, Software Requirements
  - SD704308, Software Documentation (Design)
  - TK704308, Test Plan
  - AM704308, Control Program

Introduction to Case Study

Identification Materials

4/1/97 Pg 18

- Software Production Support Drawings
  - MP704308, Maintenance Procedure
  - SS704308, Software Production Requirements
  - TR704308, Traveler (Secure Production Procedure)
- Software Deliverable Product
  - AT704308, Executable Program

Introduction to Case Study

Identification Materials

4/1/97 Pg. 19

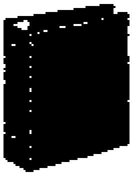
- Software Qualification Support Drawings
  - QP704308, Qualification Plan
  - CER 960061SA, Complete Engineering Release
  - QER 951006SA, Qualification Evaluation Release
- Material List for Part Number 704308-00
  - References all software product materials
  - References all software build materials
- Disk Media Labels
  - Film Bank Materials
  - Deliverable Software Product

Introduction to Case Study

Qualification Materials

4/1/97 Pg. 20

- Qualification Plan
  - QP704308
- Complete Engineering Release
  - CER 960061SA
- Qualification Evaluation Release
  - QER 951006SA
- Source Inspection (Sandia Specific)
  - Source Inspection Request (SIR)
  - Qualification Operations Instructions (QOI)
  - Qualification Verification Report (QVR)



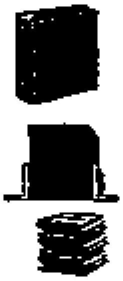
Introduction to Case Study

Acceptance Materials

4/1/97

Pg 21

- Contractual Mechanism
  - Integrated Contractor Order (ICO)
- Software Product
  - Deliverable Media (AT704308)
  - Support Drawings
- Deliverable Support Materials
  - Package Labels
  - Shipper
- DOE Inspection Materials
  - Certificate of Inspection (COI)
  - Quality Assurance Inspection Procedure (QAIP)



Introduction to Case Study

Delivery Materials

4/1/97

Pg 22

- Contractual Mechanism
  - Integrated Contractor Order (ICO)
- Shipping Instructions and Labels
  - Shipper with InterProject (IP) stamp
  - Package label with InterProject (IP) stamp
- Deliverable Software Product
  - Disk media with diamond stamp selected at QAIP

Identification Process

4/1/97 Pg. 23

The **Identification Process** provides a mechanism for uniquely numbering and labeling each of the software component elements and relating those elements to the system in which the software is executed.

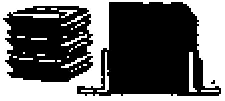
The identification process answers these questions

- What needs to be identified?
- How are they identified?
- How are changes identified and tracked?
- How are certain delivery issues resolved?

Identification Process

What Needs To Be Identified?

4/1/97 Pg. 24



Things to be Identified include:

- Software products
- Software product components (e.g.documentation)

Each of these must be given unique identifiers and labeled in accordance with naming and product numbering standards and procedures.

For Software Products within the NWC, the identification process used is the Part Drawing System and Materials List.

Identification Process

How Are Thev Identified?

4/1/97 Pg 25

NWC - Software Products

- The NWC Drawing system identifies a software product with a six digit alpha numeric identification number (drawing number) and a two digit version number (initially 00)
- The identification number is primarily associated with the part of the software that is delivered to the end use customer
- Example from Case Study:

  704308-00

Identification Process

How are Thev Identified?

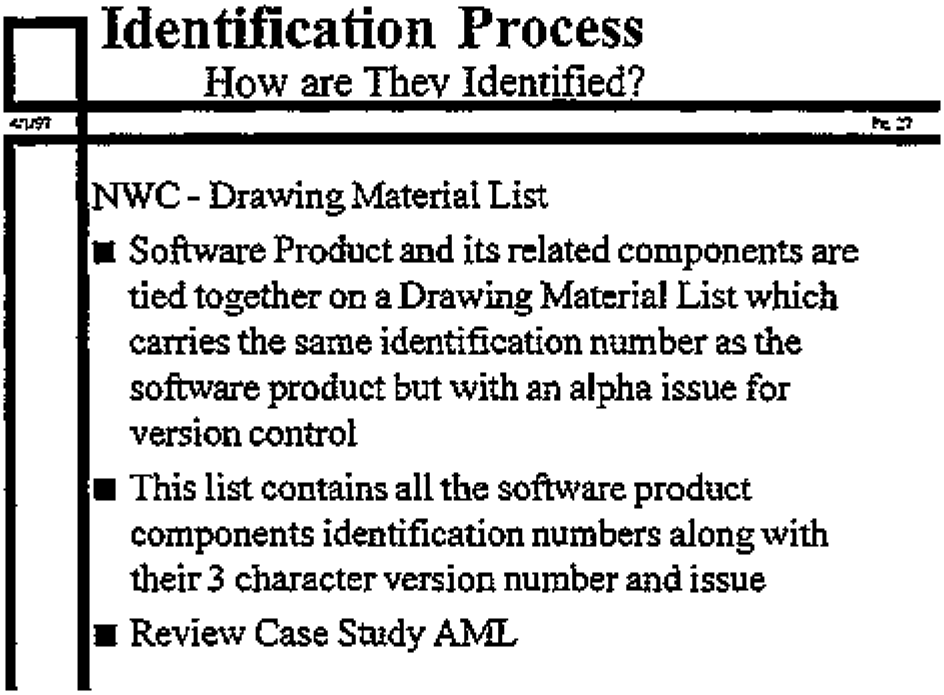
4/1/97 Pg 26

NWC - Software Product Components

- All related software product components identification numbers are derived from the associated software product identification number
- We do this by adding a 2 digit prefix indicating the software product component, a 3 character version number, and an alpha "Issue"
- Example from Case Study

  SR704308-000, Issue A

  AT704308-000, Issue A (See Case Study)



Identification Process

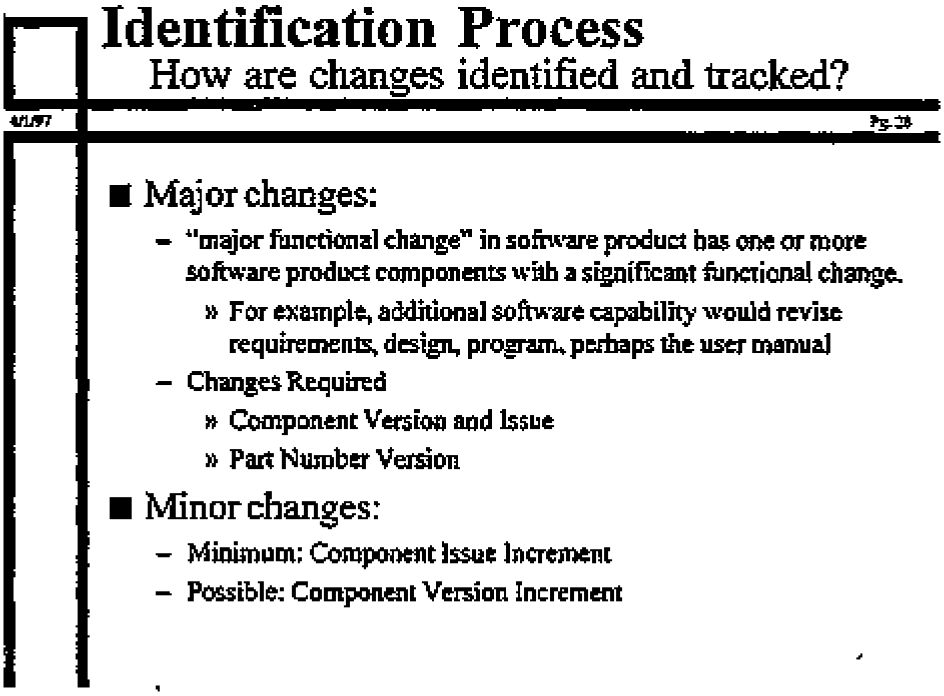
How are They Identified?

4/1/97

Pg. 27

NWC - Drawing Material List

- Software Product and its related components are tied together on a Drawing Material List which carries the same identification number as the software product but with an alpha issue for version control
- This list contains all the software product components identification numbers along with their 3 character version number and issue
- Review Case Study AML



Identification Process

How are changes identified and tracked?

4/1/97

Pg. 28

- Major changes:
  - "major functional change" in software product has one or more software product components with a significant functional change.
    - » For example, additional software capability would revise requirements, design, program, perhaps the user manual
  - Changes Required
    - » Component Version and Issue
    - » Part Number Version
- Minor changes:
  - Minimum: Component Issue Increment
  - Possible: Component Version Increment

Identification Process

How Are Certain Delivery Issues Resolved?

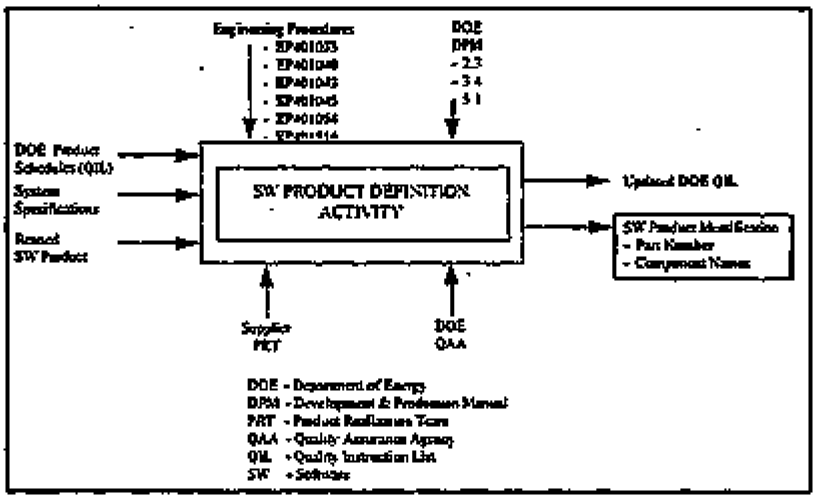
4/1/97 Pg. 29

- When a deliverable is broken into parts, either for physical necessity or for convenience, the order and existence of the parts must be specified.
- This need is satisfied by a change to the software component identification number version number: the first digit on the version suffix:
  - 0 (zero) indicates no partition
  - A, B, C, ... indicate as many partitions as there are letters and in alphabetical order.
- Discuss Examples

Identification Process

Process Summary Vs Case Study

4/1/97 Pg. 30



Engineering Procedures
- EP401033
- EP401040
- EP401043
- EP401045
- EP401054
- EP401114

DOE
DPM
- 2.3
- 3.4
3.1

DOE Product Schedules (QIL)

System Specifications

Reused SW Product

SW PRODUCT DEFINITION ACTIVITY

Updated DOE QIL

SW Product Identification
- Part Number
- Component Names

Supplier PRT

DOE QAA

DOE - Department of Energy
DPM - Development & Production Manual
PRT - Product Realization Team
QAA - Quality Assurance Agency
QIL - Quality Instruction List
SW - Software

Qualification Process

4/1/97 Pg 31

The Qualification Process includes all verification and validation activities by the software supplier and customer to ensure the software meets it stated requirements and satisfies applicable standards.

The qualification process answers these questions

- What needs to be qualified?
- How is this accomplished?
- Who does all the qualification work?

Qualification Process

4/1/97 Pg 32

Why Qualify a Product or Process?

- To See That It Does What It's Supposed To Do!
- DOE Requires It!!!!!!!!!

Qualification Process

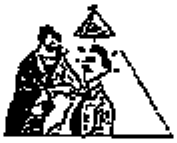
4/1/97 Pg. 33

What Needs To Be Qualified?

For software:

Anything That's Been Identified During the Identification Phases.

(Weren't You Paying Attention Earlier?????????)



Qualification Process

What Needs To Be Qualified?

4/1/97 Pg. 34

Both, Products and Processes!!!!

Products are Qualified to Ensure:

- Correct Identification
- Functional Requirements are met
- Defined software components are available

Qualification Process

What Needs To Be Qualified?

4/1/97 Pg. 35

Processes are qualified to

- Assure required engineering activities were performed
- Assure product is produced per our customer requirements.
- Assure configuration management and quality activities were performed

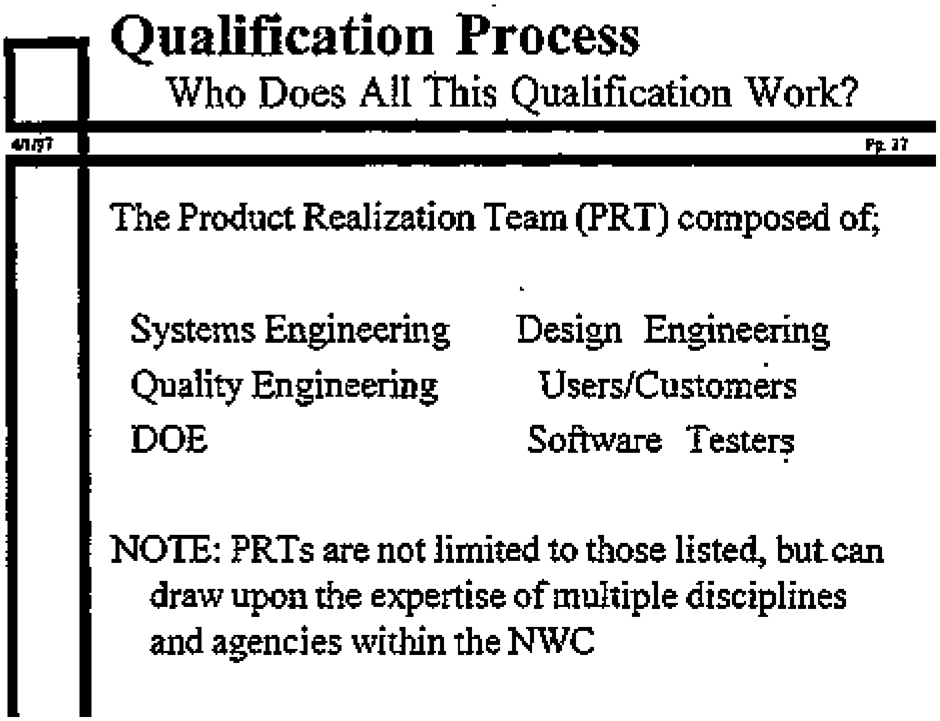
Qualification Process

How Is This Accomplished??

4/1/97 Pg. 36

Typical Activities Performed Are:

- Reviews and inspections of software development documentation
- Reviews of software test plans and results
- Reviews of configuration management, testing and design practices
- Reviews of product production documentation



Qualification Process

Who Does All This Qualification Work?

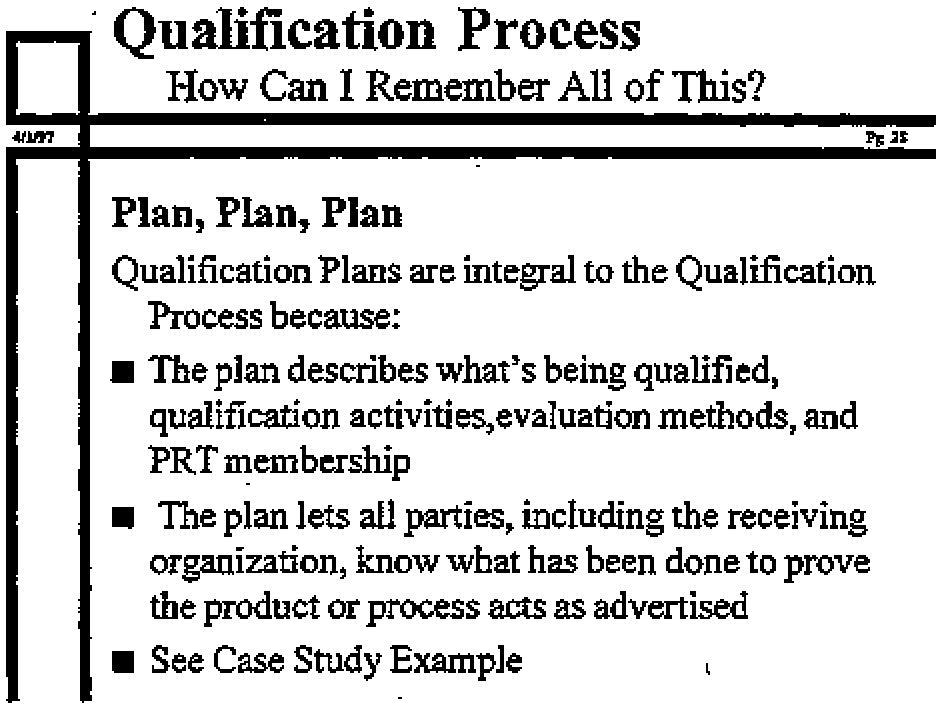
4/1/97

Pg 27

The Product Realization Team (PRT) composed of;

- Systems Engineering
- Quality Engineering
- DOE
- Design Engineering
- Users/Customers
- Software Testers

NOTE: PRTs are not limited to those listed, but can draw upon the expertise of multiple disciplines and agencies within the NWC



Qualification Process

How Can I Remember All of This?

4/1/97

Pg 28

Plan, Plan, Plan

Qualification Plans are integral to the Qualification Process because:

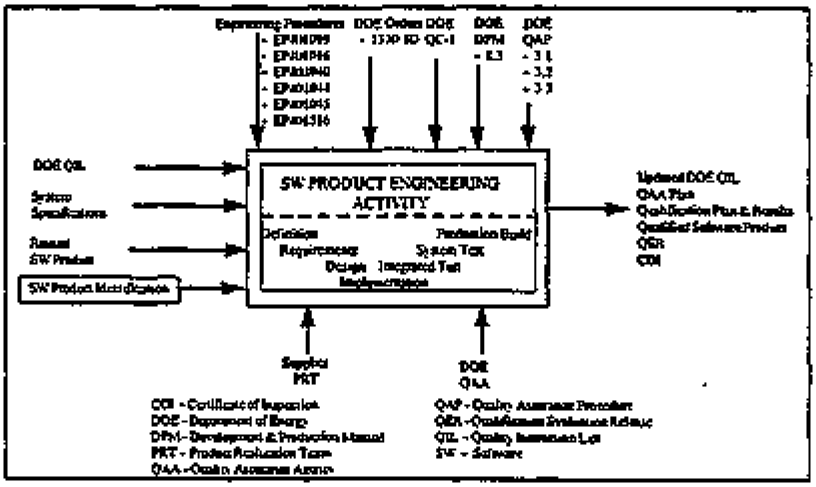
- The plan describes what's being qualified, qualification activities,evaluation methods, and PRT membership
- The plan lets all parties, including the receiving organization, know what has been done to prove the product or process acts as advertised
- See Case Study Example

Qualification Process

Process Summary Vs Case Study

4/1/97

Pg 39



COI - Certificate of Inspection
DOE - Department of Energy
DPM - Development & Production Manual
PRT - Product Realization Team
QAA - Quality Assurance Agency

QAP - Quality Assurance Procedure
QER - Qualification Evaluation Release
QIL - Quality Inspection List
SW - Software

4/1/97

Pg 40

Acceptance Process

4/1/97 Pg 41

The **Acceptance Process** includes activities that ensure software product has been adequately qualified for delivery to the specified ("next") customer

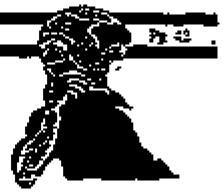
The acceptance process answers the questions:

- Why do acceptance?
- What needs acceptance?
- How do you do acceptance?

Acceptance Process

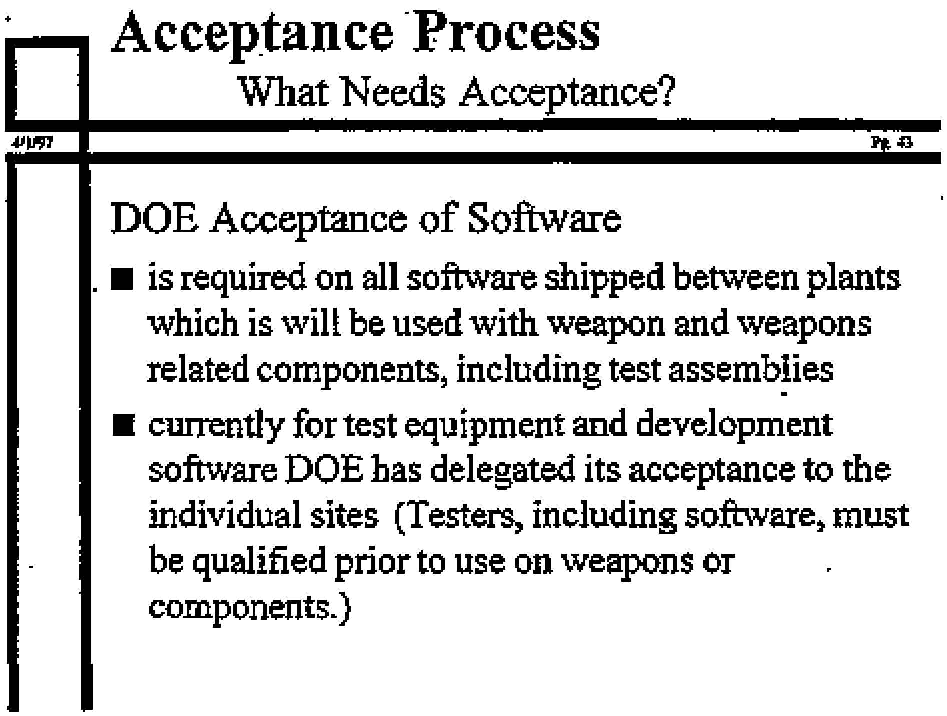
Why Do Acceptance?

4/1/97 Pg 42



DOE Policy on Software Product

- Software has become more complex and a more important element in weapon/test assembly performance
- DOE's policy has evolved to consider software as product as opposed to part of the product definition



Acceptance Process

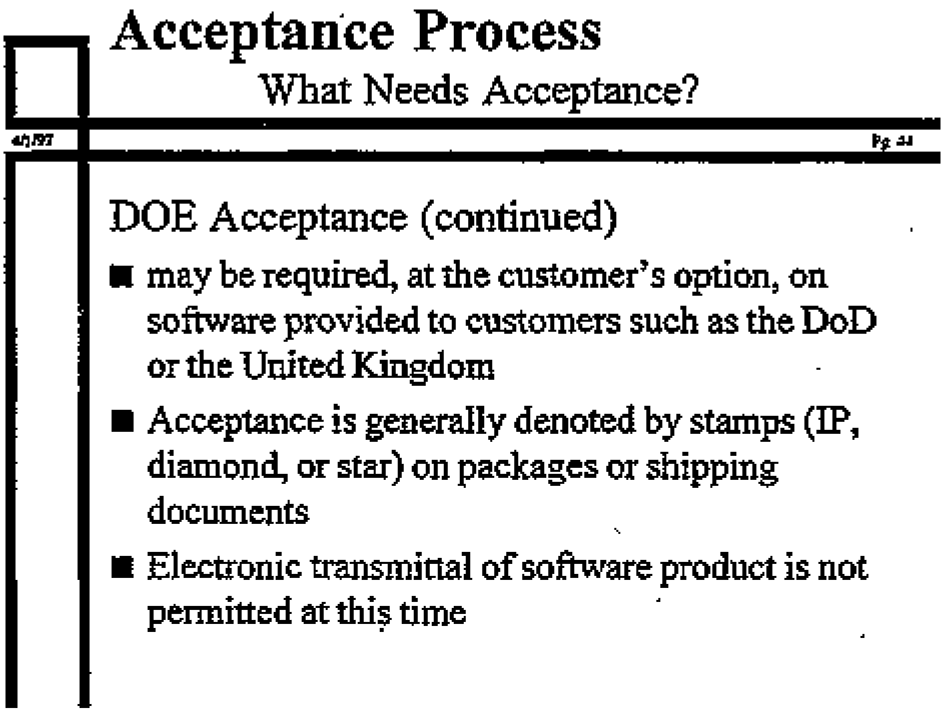
What Needs Acceptance?

4/1/97

Pg 43

DOE Acceptance of Software

- is required on all software shipped between plants which is will be used with weapon and weapons related components, including test assemblies
- currently for test equipment and development software DOE has delegated its acceptance to the individual sites (Testers, including software, must be qualified prior to use on weapons or components.)



Acceptance Process

What Needs Acceptance?

4/1/97

Pg 44

DOE Acceptance (continued)

- may be required, at the customer's option, on software provided to customers such as the DoD or the United Kingdom
- Acceptance is generally denoted by stamps (IP, diamond, or star) on packages or shipping documents
- Electronic transmittal of software product is not permitted at this time

Acceptance Process
How Do You Do Acceptance?

4/1/97 Pg 45

Submission to DOE

- The Certificate of Inspection (COI) is the form used by the contractor to submit software and other product to DOE, to identify the product definition requirements, and to certify that it meets those requirements.

Acceptance Process
How Do You Do Acceptance?

4/1/97 Pg. 46

DOE Inspection

- The DOE Quality Assurance Inspection Procedure (QAIP) describes the inspection process that DOE personnel may use as part of software acceptance
- In general the QAIP will specify verification
  - that proper labels are on media
  - that content of media is consistent
  - that software has been formally qualified (e.g., QER or equivalent)

Acceptance Process

Summary

4/1/97 Pg. 42

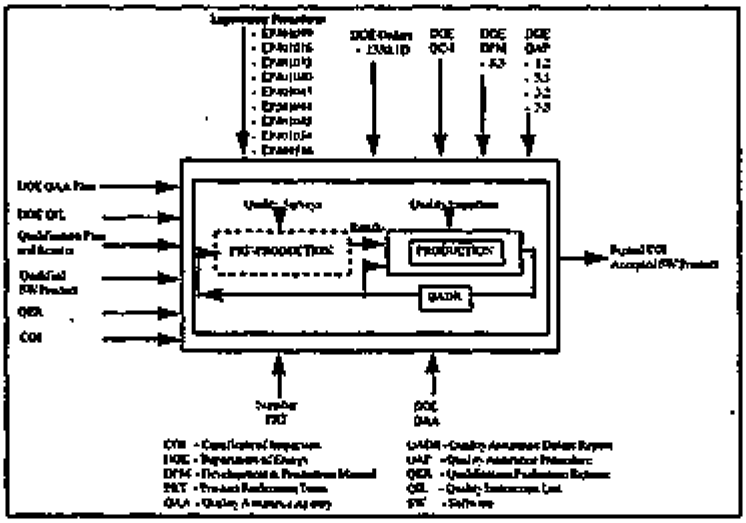
DOE Policy on Software Product

- Acceptance is essential for weapon software to provide an independent assessment that requirements have been met.
- The receiving agency requires an indication of DOE acceptance if software is intended for use in weapon product.

Acceptance Process

Process Summary Vs Case Study

4/1/97 Pg. 43



COI - Certificate of Inspection
DOE - Department of Energy
DPM - Development & Production Manual
PRT - Product Realization Team
QAA - Quality Assurance Agency
QADR - Quality Assurance Defect Report
QAP - Quality Assurance Procedures
QER - Qualification Evaluation Release
QIL - Quality Inspection List
SW - Software

Delivery Process

The **Delivery Process** includes all supplier and customer logistic activities of shipping and receiving. The Delivery Process should be sensitive to the variations in delivery of developmental software product, prove-in software product, and production software product.

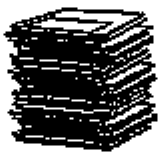
The delivery process answers the questions

- What will be delivered?
- How is software product delivered?
- How are software product components delivered?

Delivery Process

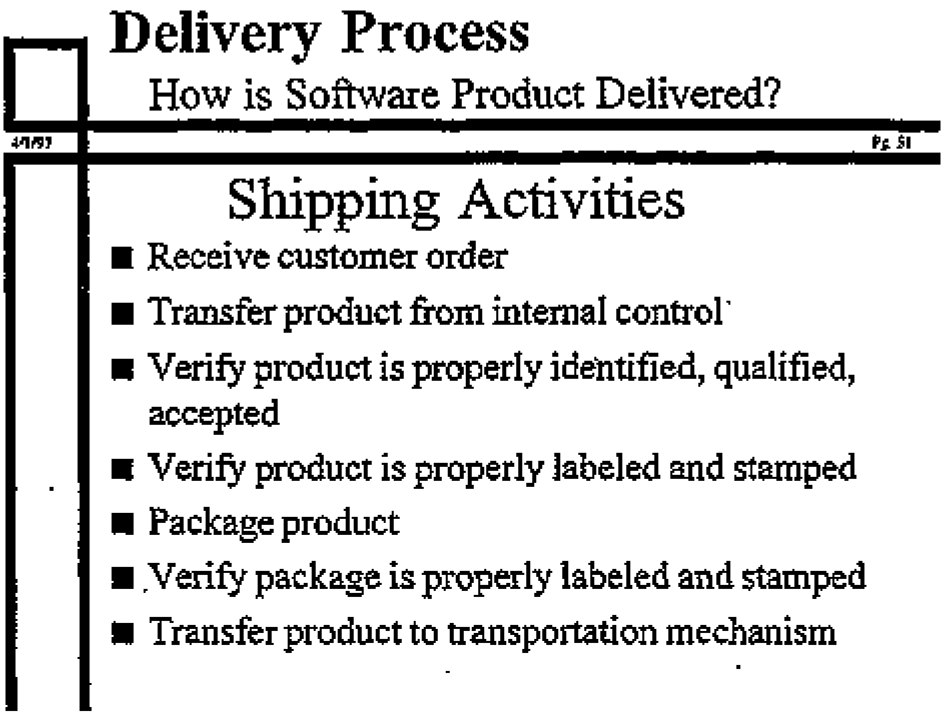
What will be Delivered?

- Software Product
- Acceptance Documentation
- Transfer Paperwork



Maybe:

- Software Product Components
  - See Case Study



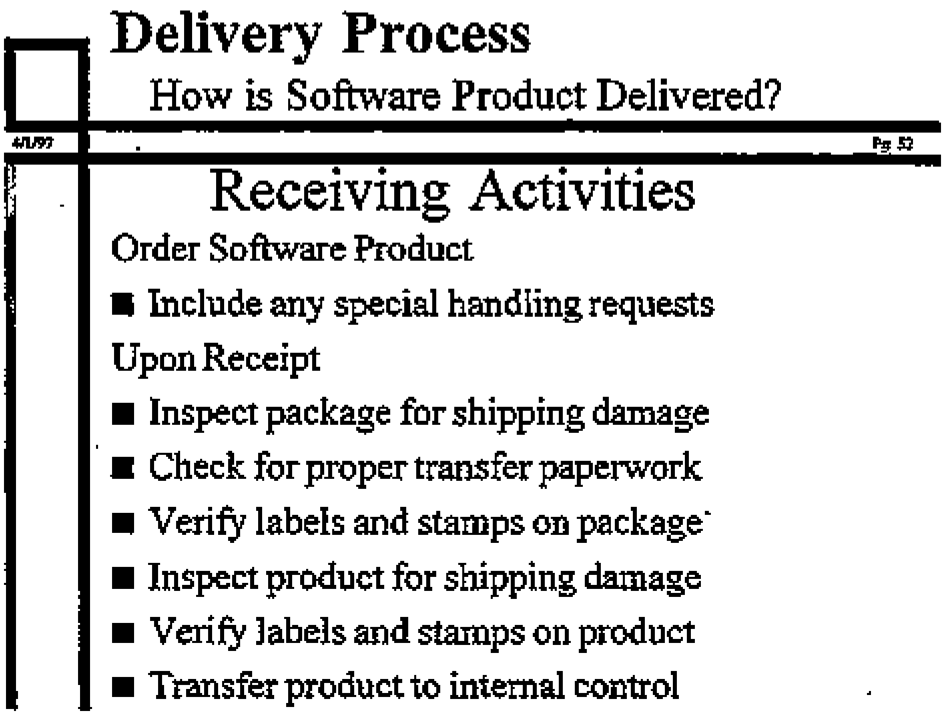
Delivery Process

How is Software Product Delivered?

4/1/97 Pg 51

Shipping Activities

- Receive customer order
- Transfer product from internal control
- Verify product is properly identified, qualified, accepted
- Verify product is properly labeled and stamped
- Package product
- Verify package is properly labeled and stamped
- Transfer product to transportation mechanism



Delivery Process

How is Software Product Delivered?

4/1/97 Pg 52

Receiving Activities

Order Software Product

- Include any special handling requests

Upon Receipt

- Inspect package for shipping damage
- Check for proper transfer paperwork
- Verify labels and stamps on package
- Inspect product for shipping damage
- Verify labels and stamps on product
- Transfer product to internal control

Delivery Process

How are Software Product Components Delivered?

4/1/97 Pg 53

Shipping Organization

- Verify product acceptance documentation is complete
- Transfer any support documentation including drawings to the receiving organization

Receiving Organization

- Verify product acceptance documentation is complete
- Verify that support documentation is released and available for use

Delivery Process

Case Study Summary

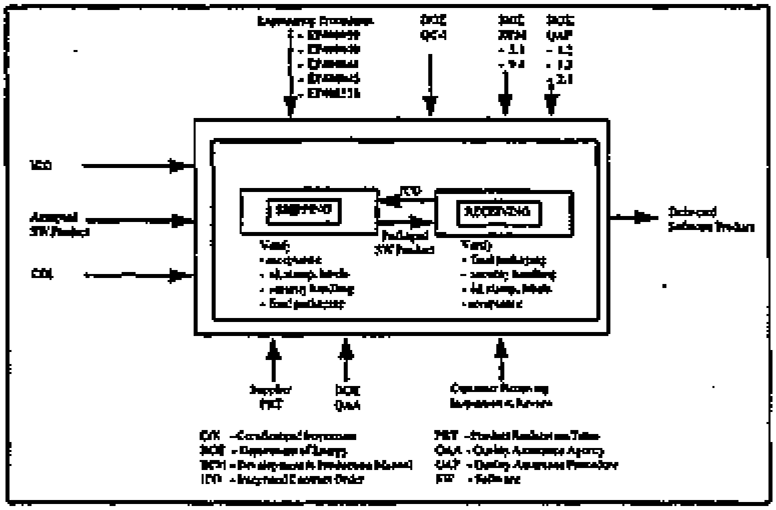
4/1/97 Pg 54

- Software Product Order
  - Integrated Contract Order (ICO)
- Software Product Delivered
  - P/N 704308-00
- Support Documentation
  - Drawings Transferred via Drawing System
  - Acceptance/Qualification Documentation Transferred with Product

Delivery Process

Process Summary Vs Case Study

4/1/97 Pg. 56


Engineering Procedures
DOE QC-1
DOE DPM - 3.1 - 9.4
DOE QAP - 1.2 - 1.3 - 2.1
ICO
Accepted SW Product
C/C
SHIPPING
ICO
Packaged SW Product
RECEIVING
Verify:
- acceptable
- id, stamp, labels
- security handling
- final packaging
Verify:
- final packaging
- security handling
- id, stamp, labels
- acceptable
Delivered Software Product
Supplier PRT
DOE QAA
Customer Receiving Inspection & Review
C/C - Certificate of Conformance
DOE - Department of Energy
DPM - Development & Production Manual
ICO - Integrated Contract Order
PRT - Product Realization Team
QAA - Quality Assurance Agency
QAP - Quality Assurance Procedure
SW - Software

Specific Concerns

4/1/97 Pg. 56

- Requirement Not Well Known Across NWC
- Complex-wide Process NOT Completely Defined
- Engineering Procedures (EPs) Mostly Do Not Address This Process
- Only Addressing Software Embedded in Product - What About Test Equipment, Numerical Control, Development, Process Equipment, Inherited, Legacy, Simulation, Scientific Codes ?..?..?

Specific Concerns

4/1/97 Pg 57

- Receiving processes may vary
- Customer may require different delivery processes
- Do these processes apply to my site's software products?

Specific Concerns

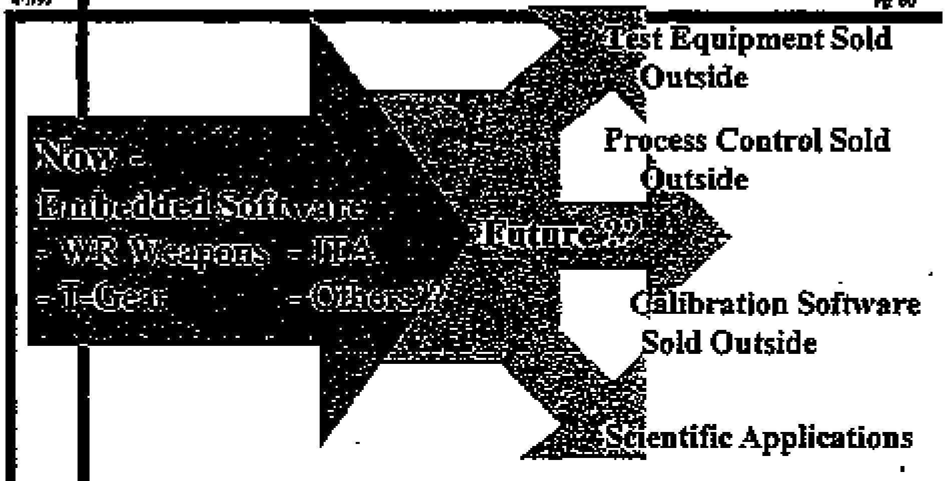
4/1/97 Pg 58

- Software Qualification Relationships
  - Quality Engineer (Role?)
  - Product Realization Team (Scope?)
  - Qualification Plan and QER (Format & Content?)
- Software Acceptance Relationships
  - Internal Inspection: Pre QAIP
  - External Inspection: QAIP Interface with DOE

Future Direction

- Reengineering of Engineering Procedures
  - EP401016,33,34,35,40,43,44,45,54, EP401516
- DOE Mission Statement
  - Include software product in statement
  - Define production, higher product integration responsibilities
- Software Product Scope
  - QAIP-like mechanisms will apply to all software
  - Not all software will have the same mechanisms as the WR software product

Summary Tutorial Focus



Now -
Embedded Software
- WR Weapons - JTA
- T-Gear - Others??

Future??

Test Equipment Sold Outside

Process Control Sold Outside

Calibration Software Sold Outside

Scientific Applications

Summary of Process

4/1/97 Pg 61

- **Identification** - 8 (or 9) Digit Part No. / Equivalent 6 Digit Drawings, support components identification
- **Development/Qualification** - PRT Controlled
- **Acceptance** - Final Acceptance by DOE or Customer
- **Delivery** - Transfer Like Any Product (e.g. ICO)

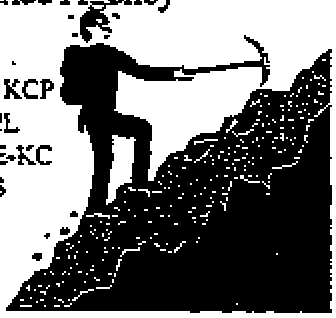
Help is Available From

4/1/97 Pg 62

- SQAS96-002 *Guideline for NWC Processes for Handling Software Product*
- The DOE Quality Assurance Procedures (QAP) Manual
- The local DOE Quality Assurance Agency
- Contacts

David Vinson, PX
Dave Peercy, SNL
Bill Warren, LLNL
Gary Echert, DOE-AL
Faye Brown, Y-12

Catherine Kuhn, KCP
Orval Hart. LANL
Ellis Sykes. DOE-KC
Ray Cullen, SRS



1947 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX •

SOFTWARE ENGINEERING QUALITY

Session W2: Writing Testable Software Requirements

Dr. Dwayne Knirk
Sandia National Laboratories

Writing Testable Software Requirements

Dr. Dwayne L. Knirk
Sandia National Laboratories

This tutorial identifies common problems in analyzing requirements in the problem and constructing a written specification of what the software is to do. It deals with two main problem areas: identifying and describing *problem requirements*; and analyzing and describing *behavior specifications*.

Software-intensive systems are expected to work in a particular environment to bring about desired effects in that environment. To accomplish these effects, the computing system must have a variety of interactions with that environment. Its capabilities and features are directed to establishing a variety of relationships between those interactions, including stimulus-response, constraint, and historical reference. To establish such relationships are the services provided by the computing system. The given environment and required effects in the problem are collectively documented as Problem Requirements. The computing system interactions and services are documented Behavior Specification. The relationship between these two sets of information is an explicit and verifiable *behavior design* task.

The Behavior Specification characterizes a computing system independently of its application context. Having a behavioral specification enables a true concurrence in development and testing processes. It provides a single reference point for all decisions of software architecture and implementation as well as for test case and testware architecture and implementation.

This tutorial focuses on determining what facts about a computing system are to be documented, how they should be expressed, and how they are related to facts about the application environment. It provides an overview of these basic specification techniques:

- the application of standard problem frames for classifying and organizing the various requirements,
- the application of stimulus/response and client/server viewpoints for structuring the description of computing system behavior,
- the expression of unique, testable action statements with the help of pre- and post-conditions, state models, and datastore models,
- the description of behaviors of components and their architectural composition into the behaviors of assemblies, and
- the use of these descriptions in Software Requirements Specification documents.

Much of this material in this tutorial is being developed as part of the next revision of IEEE Std 1175. Part of that standard is a system behavior meta model. Various parts of the material are undergoing refinement by application in various Sandia projects.

BIOGRAPHY

Dr. Dwayne L. Knirk

Dr. Knirk is a member of the software quality engineering department at Sandia National Laboratories. He provides in-house consulting to line organization projects for software engineering processes, methods, standards, tools, and training. He participates in process assessments and improvement programs, and provides support for configuration management, software inspections, and process automation. Dr. Knirk's primary focus is on the two complementary areas of software specification and testing, in which he works to bring more formal methods into more practical applications. He works actively on IEEE software engineering standards groups. He is a member of the ASQC Software Division Methods Committee.

Dr. Knirk previously worked for Programming Environments, Inc., where he was the architect and principal developer of the automated software test design tool, T. That commercial product analyzed a formal software behavior description for testability, designed test cases for demonstrating that behavior, and generated actual test case data.

Tutorial: Writing Testable Software Requirements

Software Quality Forum
Albuquerque, NM

1 April 1997

Presented by

Dr. Dwayne L. Knirk
Quality Engineering Department
Sandia National Laboratories, Albuquerque, NM

SAND97-XXXC

This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000.
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy

1947 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

Session X2: Using COTS Software in Development Projects

Lt Col Nancy Crowley
Acting Chief, Space System Technologies
Phillips Laboratory
Kirtland AFB, NM

The Use of COTS in the Multimission Advanced Ground Intelligent Control (MAGIC) Program

Lt Col Nancy L. Crowley, Phillips Laboratory PL/VTS

The use of commercial software and standards has been touted as a potential for significant cost and time savings in developing military systems, specifically, satellite control systems. And while the savings do exist, commercial software and standards must be carefully evaluated prior to selection, carefully integrated, and used where appropriate to reap their benefits. For example, not all Commercial Off-The-Shelf (COTS) products are suitable because they encompass too may inseparable functions, have a very narrow customer base and/or have no possible replacement COTS products. A COTS-based system should consist of small components that do one contained task and integrate with other components through some sort of message passing, such as files, DDE, OLE, DLL or other appropriate middleware protocols such as provided in the CORBA environment. A component should be able to be replaced with no, or minimal, impact on other components in the system. Commercial protocols can be unstable and change rapidly over time, forcing decisions on when to upgrade the components to new versions, and evaluating the impact of doing so. Also, COTS components have bugs, and are usually not tested to the stringent standards seen for some military systems. The features in COTS components are often not exactly what is needed, necessitating decisions on whether they are good enough, or if some custom code should be developed and integrated.

The tutorial will discuss the experiences of the Space System Technologies Division of the USAF Phillips Laboratory (PL/VTS) in developing a COTS-based satellite control system. The system's primary use is a testbed for new technologies that are intended for future integration into the operational satellite control system. As such, the control system architecture must be extremely open and flexible so we can integrate new components and functions easily and also provide our system to contractors for their component work. The system is based on commercial hardware, is based on Windows NT, and makes the maximum use of COTS components and industry standards.

BIOGRAPHY

Nancy L. Crowley, Lt Col

Acting Chief, Space System Technologies

Lt Col Nancy Crowley is the Acting Chief of the Space System Technologies Division (PL/VTS), Kirtland AFB, New Mexico. The focus of Space System Technologies Division is on the innovative application of software technologies to improve performance and reduce operations and maintenance costs for satellite control systems, including telemetry, tracking and commanding (TT&C), mission data dissemination, data processing, and satellite autonomy. Lt Col Crowley is also the program manager for the Multimission Advanced Ground Intelligent Control (MAGIC) program. MAGIC is developing the architecture for the next generation satellite control system that provides a low cost, flexible software architecture that allows plug and play of COTS products in a vendor independent manner.

Lt Col Crowley was born May 13, 1955 in the Bronx, New York. She graduated from Theills High School in Theills NY, in 1973. She received a Bachelor of Science in Electrical Engineering from the University of New Hampshire in 1977 where she was a ROTC distinguished graduate. She later received the Master of Science in Digital Engineering and the Doctor of Philosophy (major of software engineering, minor of artificial intelligence) from the Air Force Institute of Technology in 1982 and 1994 respectively. Her research was in object-oriented methods for software requirements analysis.

Lt Col Crowley entered the Air Force in 1972 and was a flight test engineer for Tactical Air Command. There she conducted operational test and evaluation and flew in fighter aircraft in support of projects. After her masters degree, she was assigned to the Flight Dynamics Laboratory, where she was the software engineer for the digital flight control system of the X-29 Advanced Technology Demonstrator and the Ada focal point for the laboratory. There and in subsequent assignments she was a technical consultant to the Swedish government on the development of the digital flight control system for the JAS-39. Her next assignment was at the Systems Acquisition School, Brooks AFB Texas where she was a course developer and instructor of software acquisition courses. There she was also a system administrator for a UNIX and PC-based networked system that serviced the students and staff at the school. After completing her Ph.D., she came to her current assignment in Oct 94.

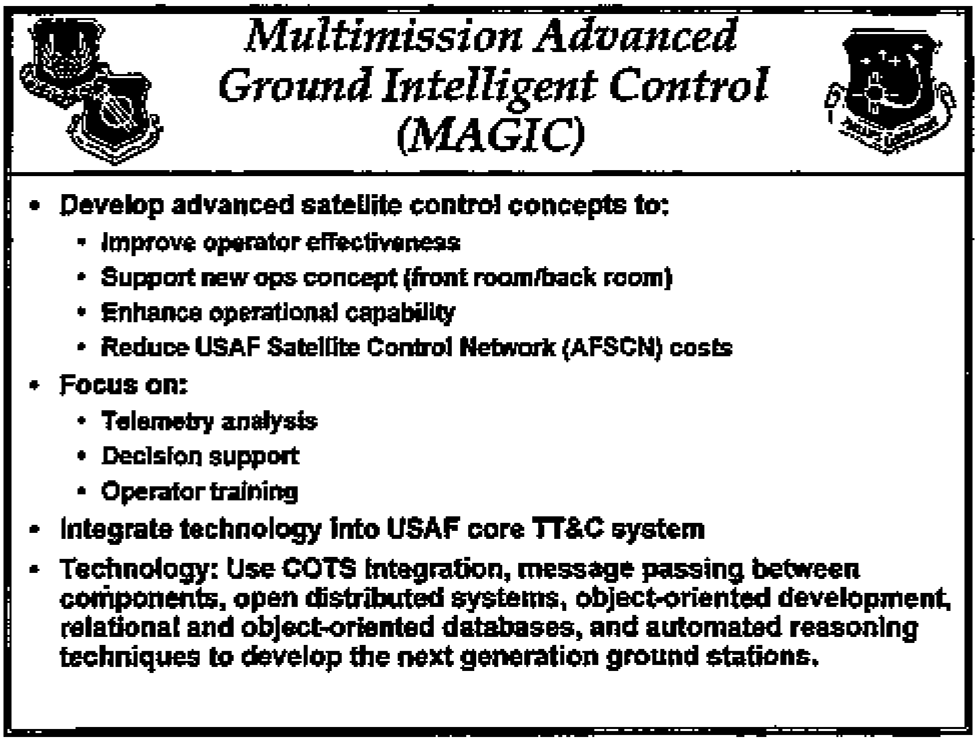
Outside her Air Force duties, Lt Col Crowley teaches software engineering, software management, and computer science courses at local Universities. Her and her husband own a computer consulting business. Both her and her husband enjoy riding horses.

Phillips Laboratory PL/VTS
3550 Aberdeen Ave SE
Kirtland AFB NM 87117-5776
Voice: 505-846-0461, ext 313
Fax: 505-846-6053
E-mail: crowleyn@plk.af.mil

SATELLITE CONTROL

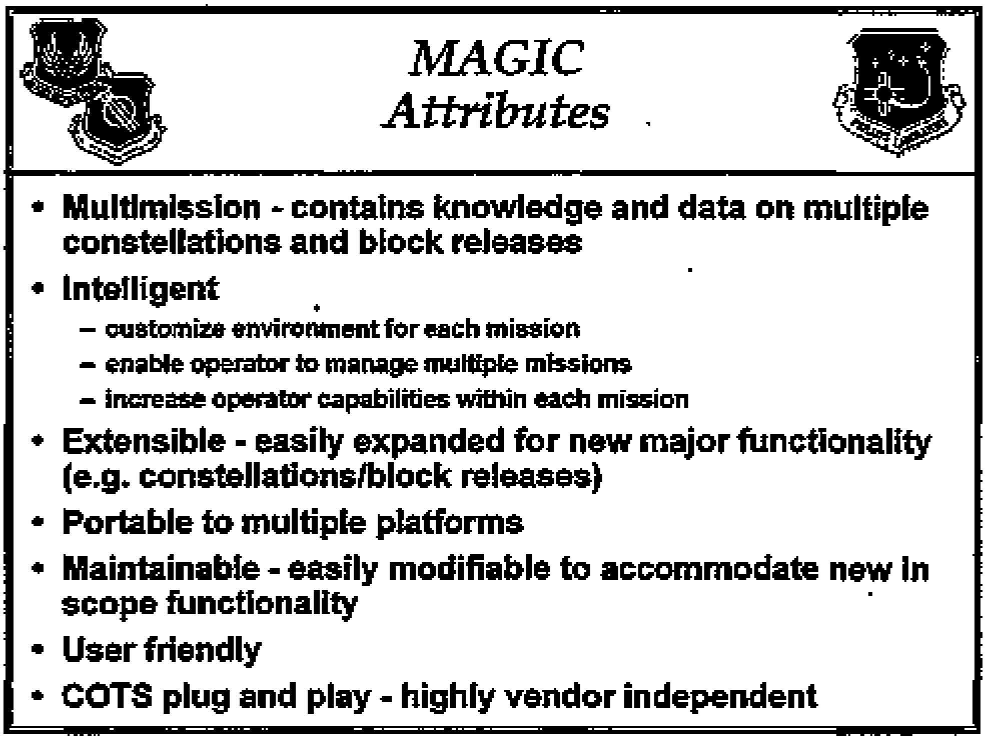
Satellite Control and
Simulation Division
MAGIC Program

Lt Col Nancy Crowley, Acting Chief
Space System Technologies Division
PL/VTS, (505)846-0461 ext 313, DSN 246-0461
crowleyn@plk.af.mil



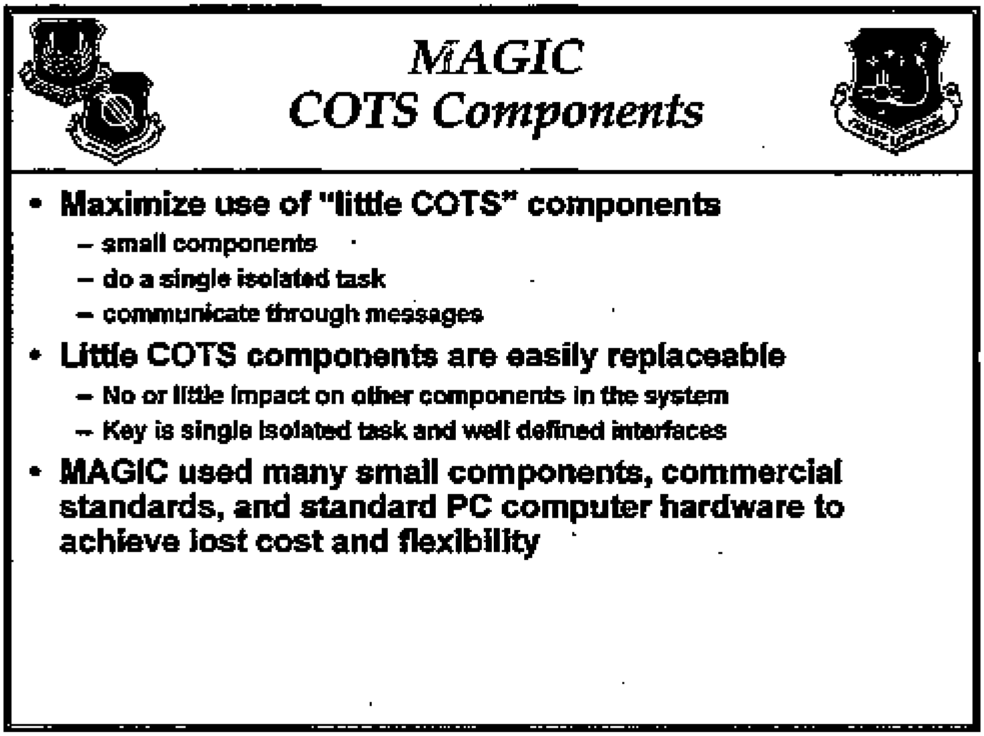
Multimission Advanced Ground Intelligent Control (MAGIC)

- Develop advanced satellite control concepts to:
  - Improve operator effectiveness
  - Support new ops concept (front room/back room)
  - Enhance operational capability
  - Reduce USAF Satellite Control Network (AFSCN) costs
- Focus on:
  - Telemetry analysis
  - Decision support
  - Operator training
- Integrate technology into USAF core TT&C system
- Technology: Use COTS integration, message passing between components, open distributed systems, object-oriented development, relational and object-oriented databases, and automated reasoning techniques to develop the next generation ground stations.



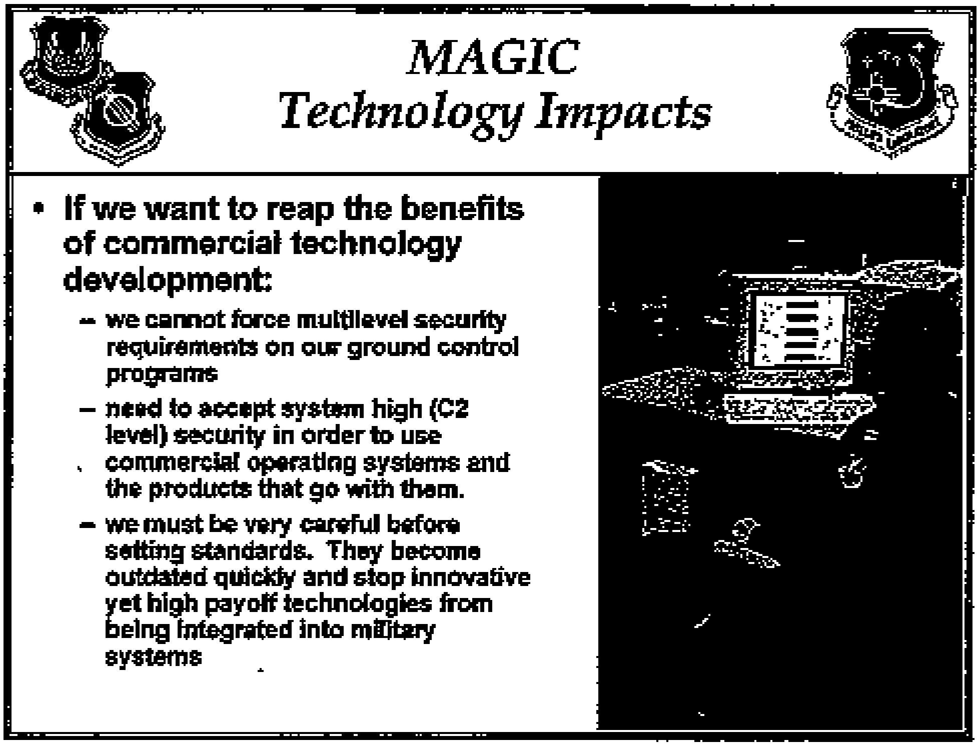
MAGIC
Attributes

- Multimission - contains knowledge and data on multiple constellations and block releases
- Intelligent
  - customize environment for each mission
  - enable operator to manage multiple missions
  - increase operator capabilities within each mission
- Extensible - easily expanded for new major functionality (e.g. constellations/block releases)
- Portable to multiple platforms
- Maintainable - easily modifiable to accommodate new in scope functionality
- User friendly
- COTS plug and play - highly vendor independent



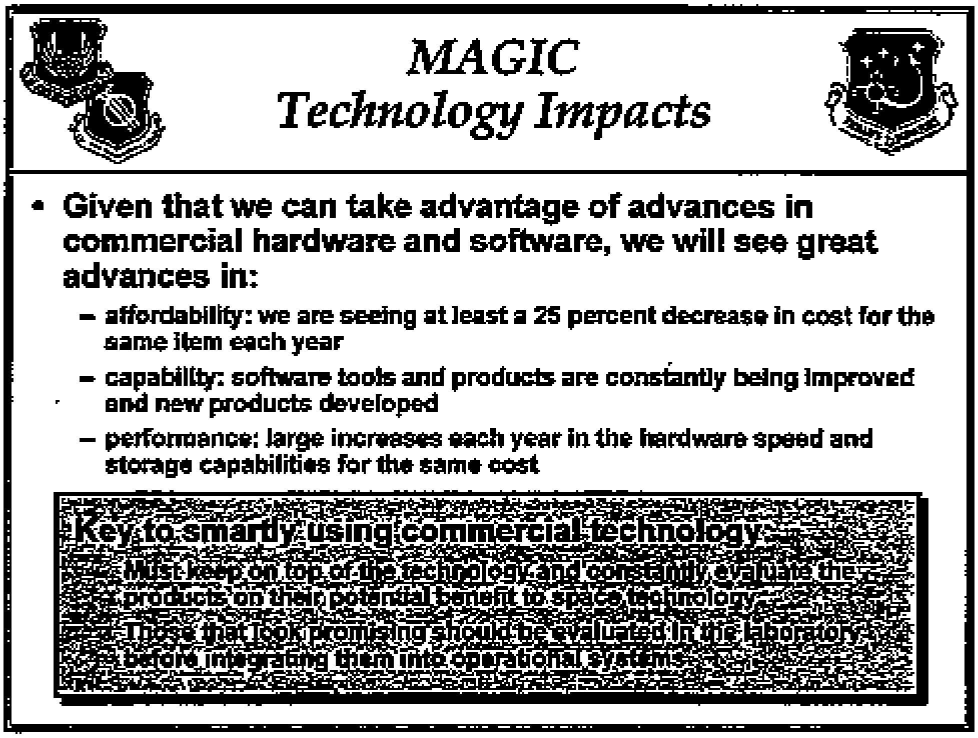
MAGIC
COTS Components

- Maximize use of "little COTS" components
  - small components
  - do a single isolated task
  - communicate through messages
- Little COTS components are easily replaceable
  - No or little impact on other components in the system
  - Key is single isolated task and well defined interfaces
- MAGIC used many small components, commercial standards, and standard PC computer hardware to achieve lost cost and flexibility



MAGIC
Technology Impacts

- If we want to reap the benefits of commercial technology development:
  - we cannot force multilevel security requirements on our ground control programs
  - need to accept system high (C2 level) security in order to use commercial operating systems and the products that go with them.
  - we must be very careful before setting standards. They become outdated quickly and stop innovative yet high payoff technologies from being integrated into military systems

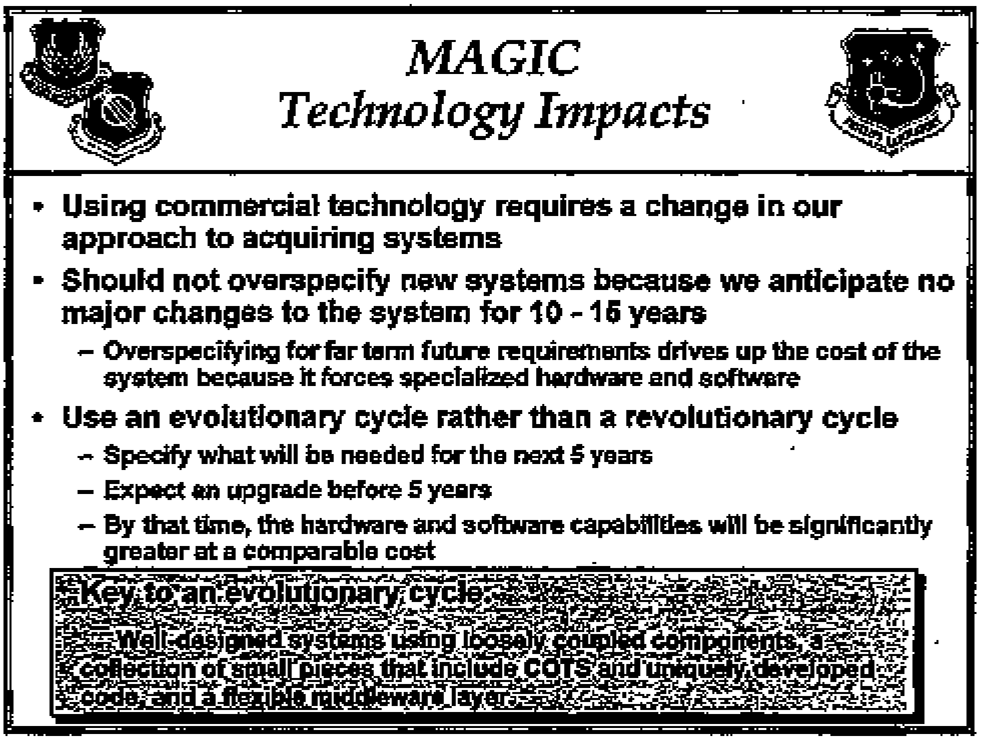


MAGIC
Technology Impacts

- Given that we can take advantage of advances in commercial hardware and software, we will see great advances in:
  - affordability: we are seeing at least a 25 percent decrease in cost for the same item each year
  - capability: software tools and products are constantly being improved and new products developed
  - performance: large increases each year in the hardware speed and storage capabilities for the same cost

Key to smartly using commercial technology

- Must keep on top of the technology and constantly evaluate the products on their potential benefit to space technology
- Those that look promising should be evaluated in the laboratory before integrating them into operational systems

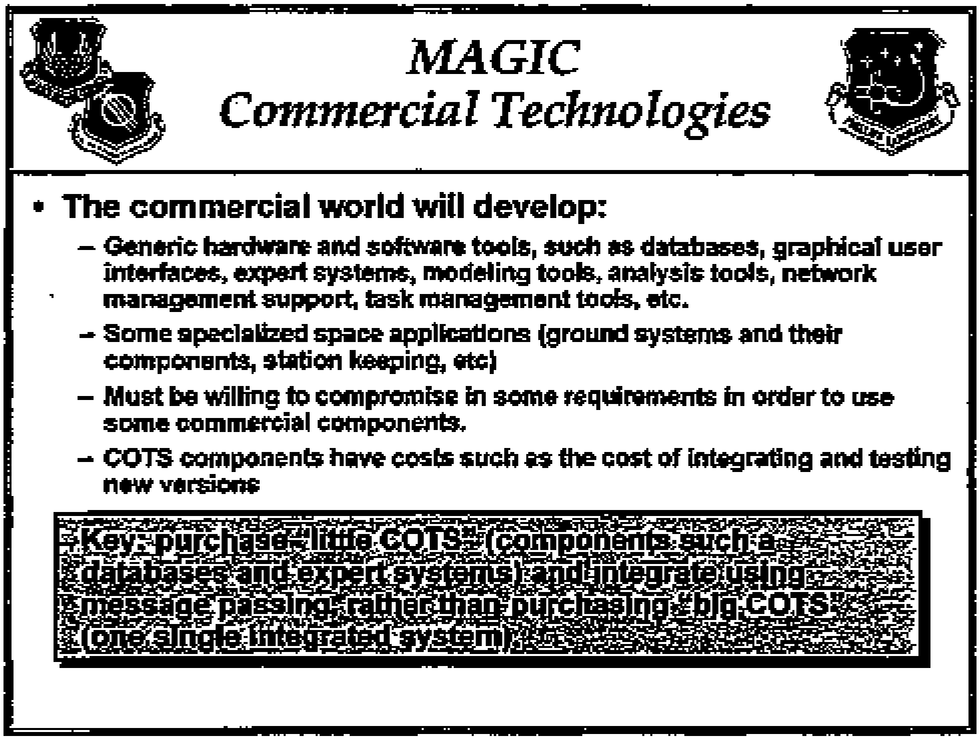


MAGIC
Technology Impacts

- Using commercial technology requires a change in our approach to acquiring systems
- Should not overspecify new systems because we anticipate no major changes to the system for 10 - 15 years
  - Overspecifying for far term future requirements drives up the cost of the system because it forces specialized hardware and software
- Use an evolutionary cycle rather than a revolutionary cycle
  - Specify what will be needed for the next 5 years
  - Expect an upgrade before 5 years
  - By that time, the hardware and software capabilities will be significantly greater at a comparable cost

Key to an evolutionary cycle:

Well-designed systems using loosely coupled components, a collection of small pieces that include COTS and uniquely developed code, and a flexible middleware layer



MAGIC
Commercial Technologies

- The commercial world will develop:
  - Generic hardware and software tools, such as databases, graphical user interfaces, expert systems, modeling tools, analysis tools, network management support, task management tools, etc.
  - Some specialized space applications (ground systems and their components, station keeping, etc)
  - Must be willing to compromise in some requirements in order to use some commercial components.
  - COTS components have costs such as the cost of integrating and testing new versions

Key: purchase "little COTS" (components such a databases and expert systems) and integrate using message passing, rather than purchasing "big COTS" (one single integrated system)

MAGIC Phases

- Phase 1: Telemetry Analysis
- Phase 2: Decision Support
  - Known anomaly decision support system (operator in the loop)
  - Known anomaly independent decision support (autonomous)
  - Unknown anomaly resolution (operator in the loop)
- Phase 3
  - Operator training
- Phase 2 and 3 are conducted in parallel

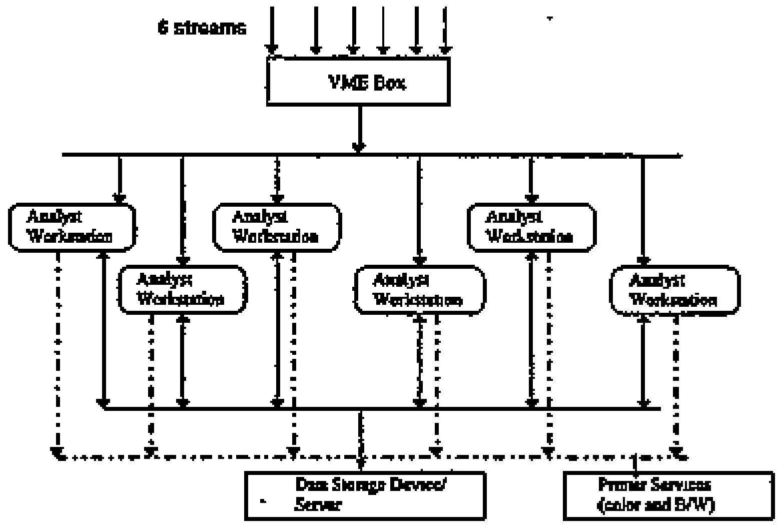
MAGIC-1

- MAGIC-1 is the first phase of MAGIC. MAGIC-1 established the basic architecture which will be used throughout the multi-year MAGIC program.
- MAGIC-1 is currently installed in Space Operation Complex (SOC) 33 at Falcon AFB CO.
- MAGIC-1 is a real-time telemetry capture and display system, as well as a post-pass telemetry analysis system.

MAGIC-1 Requirements

- MAGIC-1 is a real-time telemetry analysis system that meets the following requirements:
  - Capable of archiving 6 simultaneous telemetry streams
  - Keep telemetry data for the operational life of the satellite
  - Keep one year's worth of data on-line
  - Uses six analyst workstations
  - Provides telemetry plotting and display real-time
  - Provides operator warning of events
  - Provides trending and analysis post-pass
  - Has two-level password protection
  - Is C2 functional
  - Provides color print capability

MAGIC-1 Architecture



6 streams

VME Box

Analyst Workstation

Analyst Workstation

Analyst Workstation

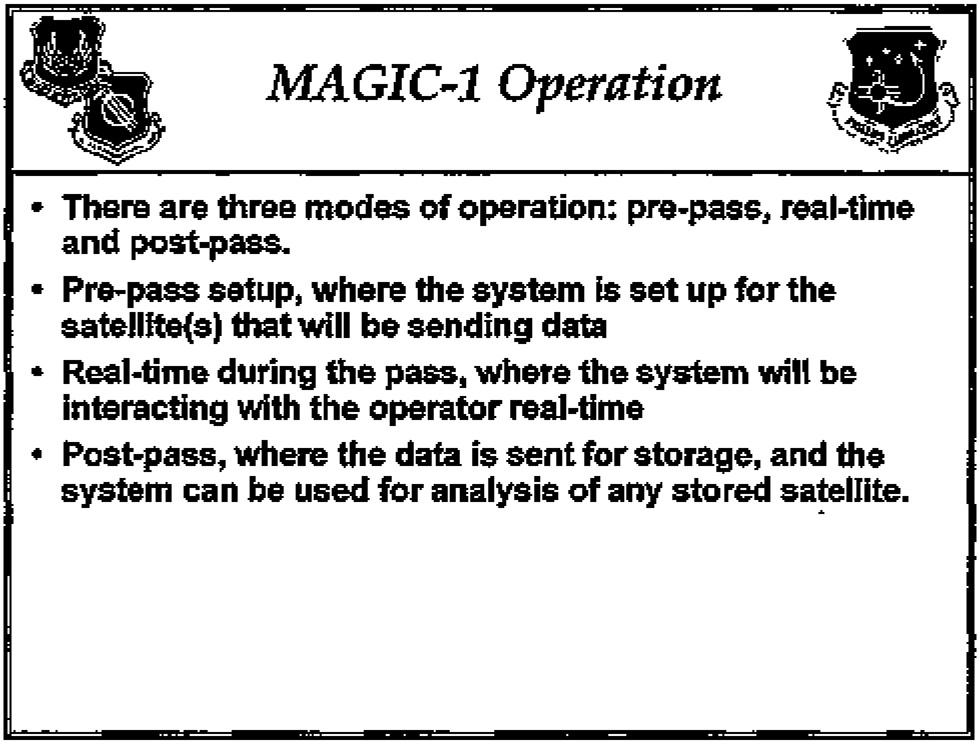
Analyst Workstation

Analyst Workstation

Analyst Workstation

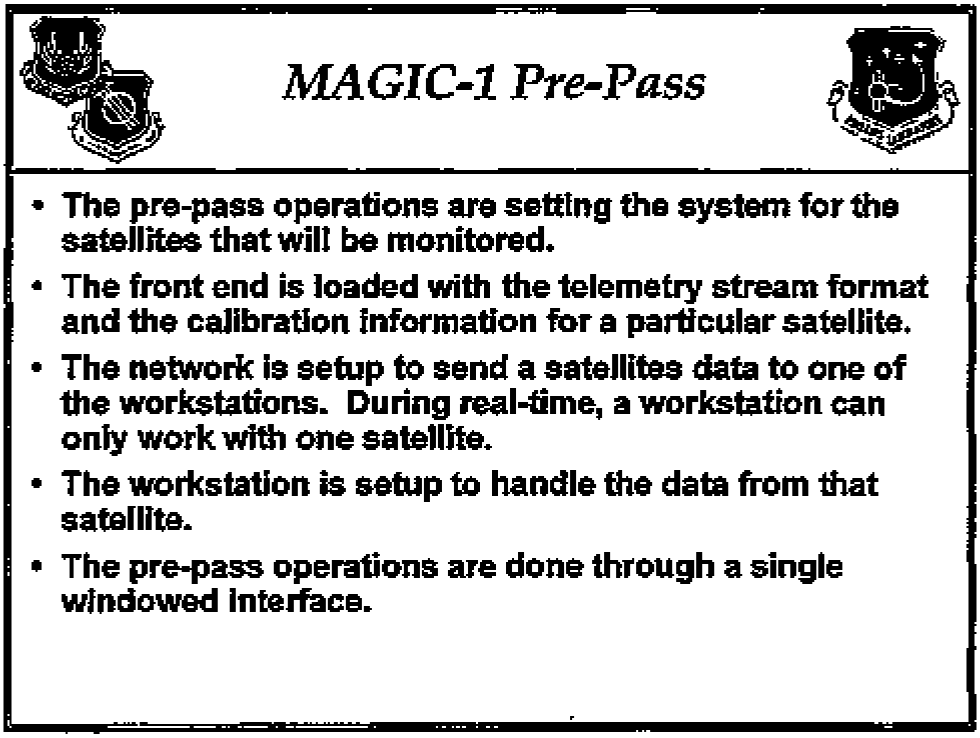
Data Storage Device/ Server

Printer Services (color and B/W)



MAGIC-1 Operation

- There are three modes of operation: pre-pass, real-time and post-pass.
- Pre-pass setup, where the system is set up for the satellite(s) that will be sending data
- Real-time during the pass, where the system will be interacting with the operator real-time
- Post-pass, where the data is sent for storage, and the system can be used for analysis of any stored satellite.



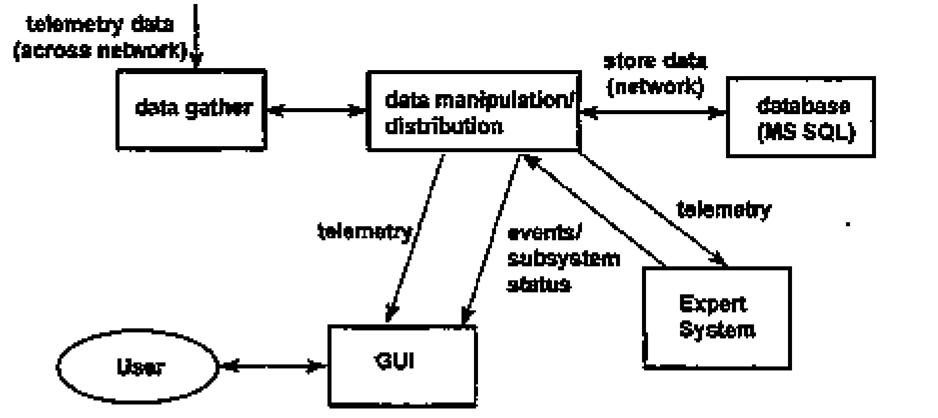
MAGIC-1 Pre-Pass

- The pre-pass operations are setting the system for the satellites that will be monitored.
- The front end is loaded with the telemetry stream format and the calibration information for a particular satellite.
- The network is setup to send a satellites data to one of the workstations. During real-time, a workstation can only work with one satellite.
- The workstation is setup to handle the data from that satellite.
- The pre-pass operations are done through a single windowed interface.

MAGIC-1 Real-Time

- During real-time operations, an operator is using the analyst workstation to monitor the satellite.
- The system will either present any of a number of preset data screens, or a custom screen.
- Each screen will consist of plots of analog data and displays of discrete points.
- ALL telemetry points, regardless of which are currently being displayed, will be examined by the system. If any go out of the normal range, the operator will be informed so corrective action can be taken.
- The status of each subsystem is displayed
- MAGIC-1 does not contain intelligence on diagnosis of potential problems.

MAGIC-1 Workstation Architecture : Real-Time



telemetry data (across network)

data gather

data manipulation/ distribution

store data (network)

database (MS SQL)

telemetry

events/ subsystem status

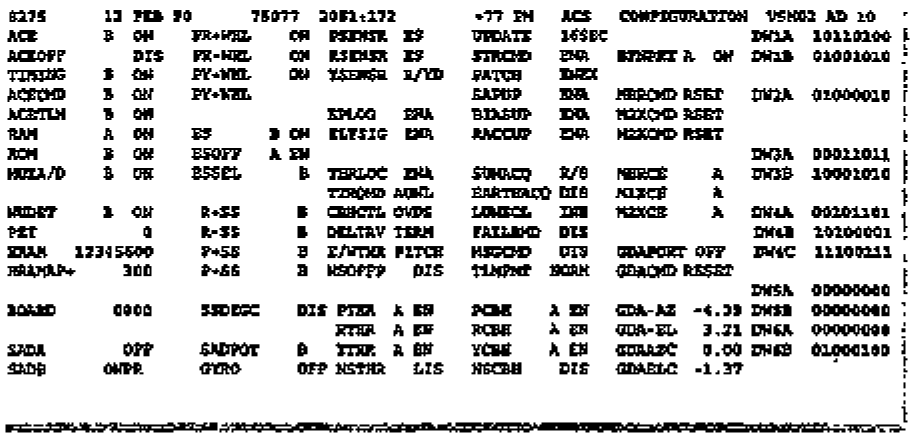
telemetry

Expert System

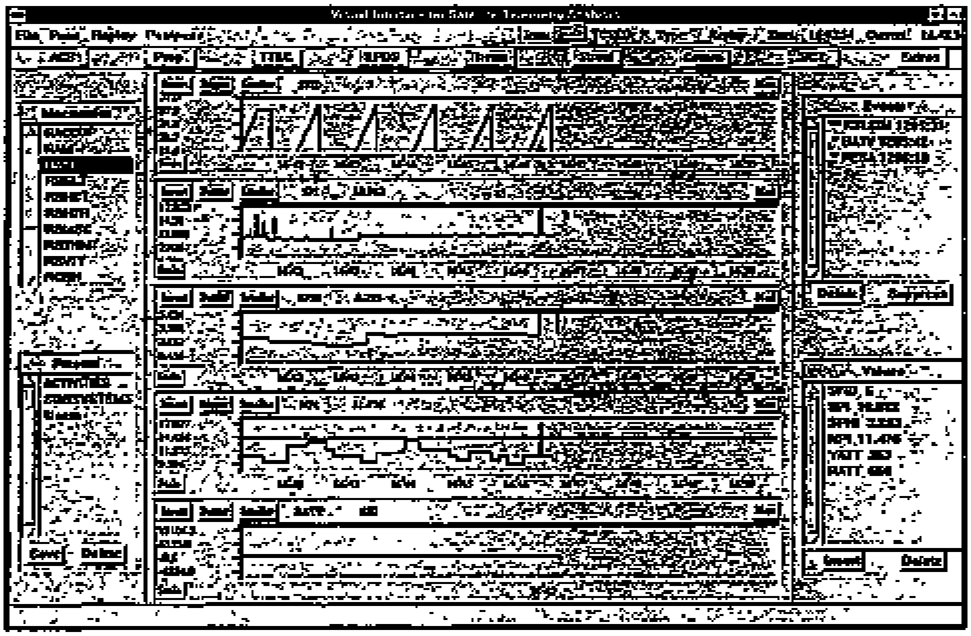
User

GUI

Current AFSCN Display Screen



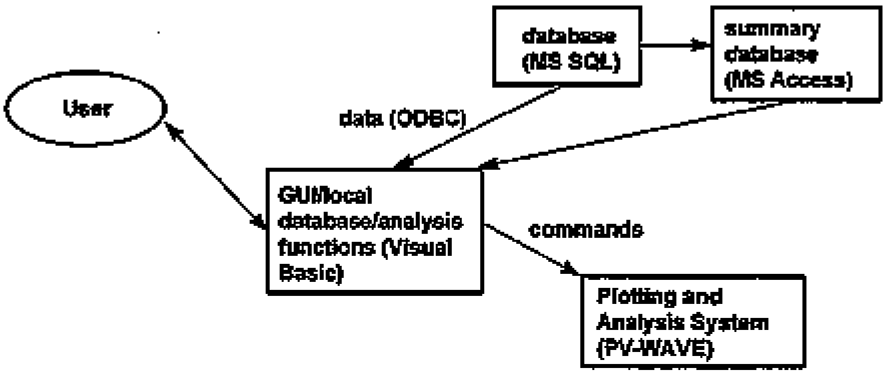
```
8275        13 FEB 90       75077  2081:172        +77 IM   ACS   CONFIGURATION  VSM02 AD 10
ACE         B  ON    FR+WHL    ON  PSENSR  ES      UPDATE   16SEC                DW1A  10110100
ACEOFF         DIS   FR-WHL    ON  RSENSR  ES      STRCMD   ENA   RTRREF A  ON   DW1B  01001010
TIMING      B  ON    PY+WHL    ON  YSENSR  R/YD    PATCH    INEX
ACECMD      B  ON    PY+WHL                        EAPUP    ENA   MECMD RSET     DW2A  01000010
ACETLM      B  ON                  EPLOG   ENA     BIASUP   ENA   MIXCMD RSET
RAM         A  ON    ES       B ON ELTSIG  ENA     RACCUP   ENA   M2XCMD RSET
ROM         B  ON    ESOFF    A EN                                               DW3A  00011011
MUXA/D      B  ON    ESSEL       B TERLOC  ENA     SUNACQ   R/8   MERCE      A   DW3B  10001010
                                   TIRCMD  AUNL    EARTHACQ DIS   MIXCE      A
WIDET       B  ON     R+SS       B CRHCTL  OVDS    LONECL   INH   M2XCE      A   DW4A  00201101
PET             0     R-SS       B DELTAV  TERM    FAILMD   DIS                  DW4B  20100001
XRAM  12345600        P+SS       B E/WTHR  PITCH   MEGCMD   DIS   GDAPORT OFF    DW4C  11100211
RRAMAP+       300     P+66       B NSOFFP    DIS   TIMPNT   NORM  GDACMD RESET
                                                                                 DW5A  00000000
BOARD        0000     SSDEGC   DIS PTHR    A EN    PCBE     A EN  GDA-AZ  -4.39  DW5B  00000000
                                   RTHR    A EN    RCBE     A EN  GDA-EL   3.21  DW6A  00000000
SADA          OFF     SADPOT     B YTHR    A EN    YCBE     A EN  GDAAZC   0.00  DW6B  01000100
SADB        ONPR      GYRO     OFF NSTHR     LIS   NSCBE      DIS GDAELC  -1.37
```

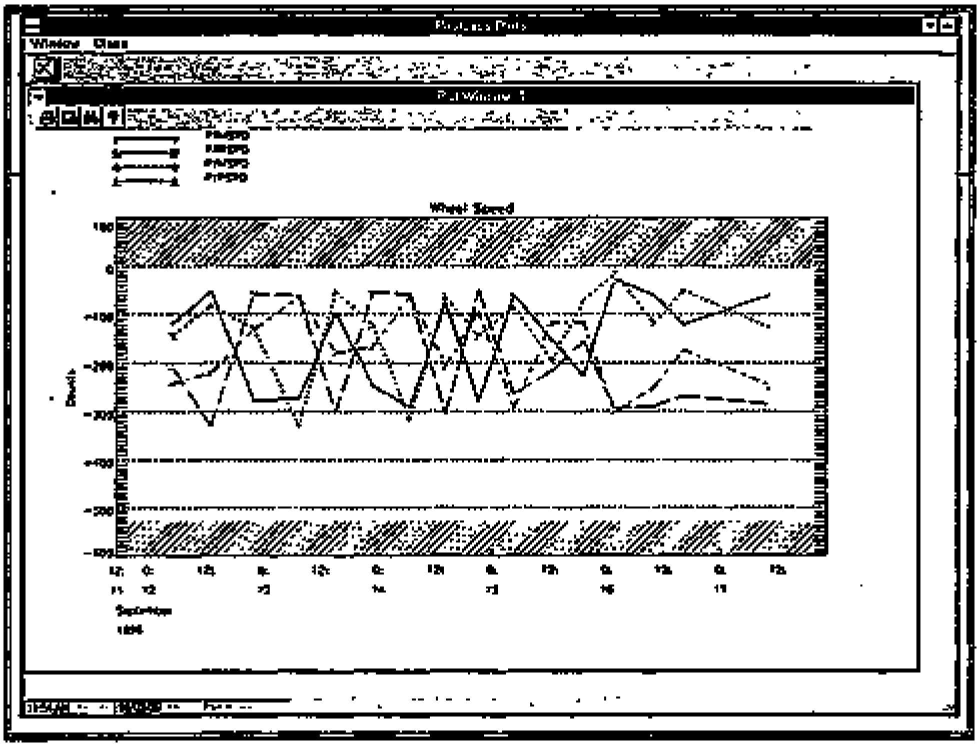


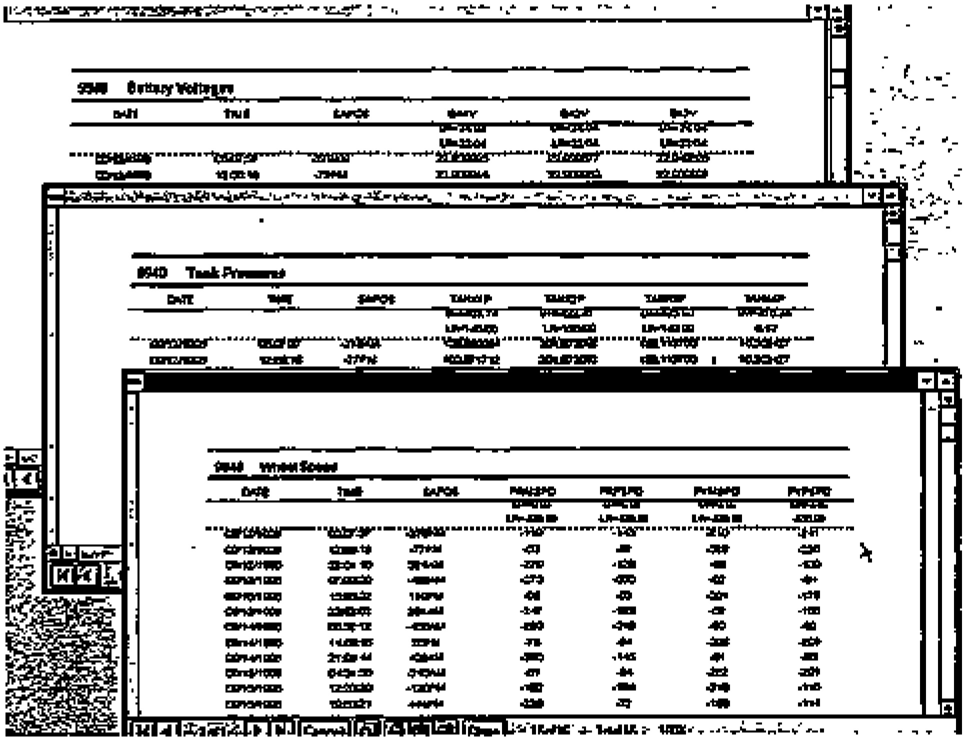
MAGIC-1 Post-Pass

- During each pass, the data is downloaded to the Data Storage Device for permanent storage
- Key data points are summarized and stored in a separate database for trending and analysis
- An operator can use any of the workstations to do analysis on data. This includes trending between satellites of the same family, as well as analysis of one satellite. Each workstation will have access to all satellite information stored on the Data Storage Device and the summarized key data points.
- The data is available during an on-going pass. This allows real-time display to occur on one workstation while post-pass analysis is conducted on the same pass on another workstation.

MAGIC-1 Workstation Architecture : Post-Pass


database (MS SQL)
summary database (MS Access)
User
data (ODBC)
GUI/local database/analysis functions (Visual Basic)
commands
Plotting and Analysis System (PV-WAVE)





9540 Battery Voltages

DATE	TIME	SAPOS	BATV	BAOV	BAJV
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

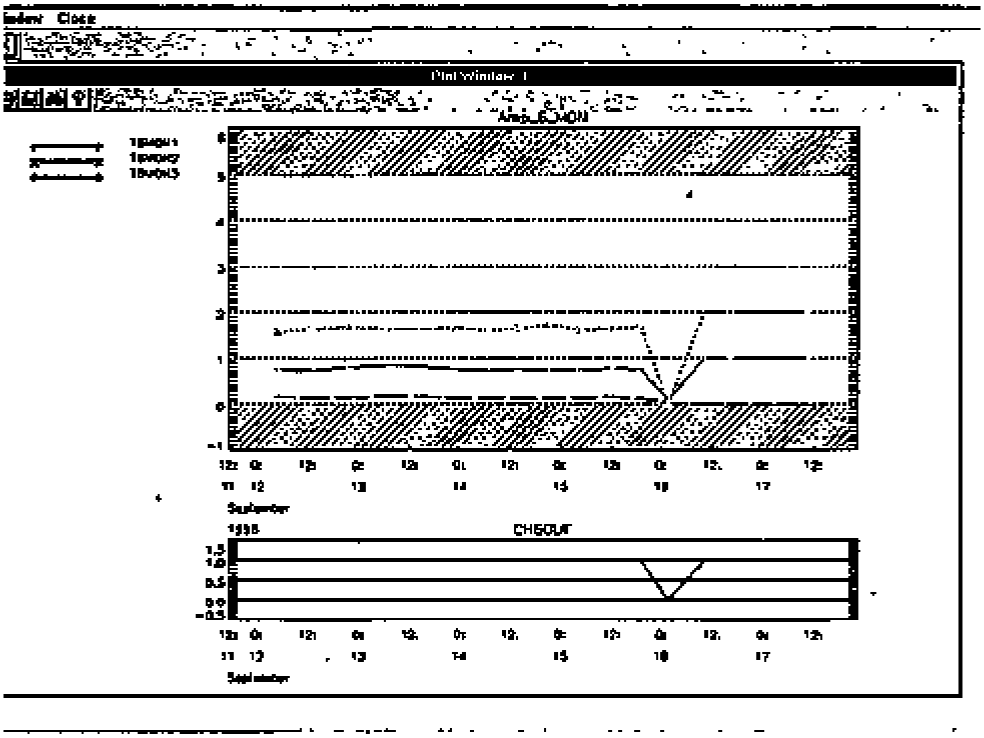
8540 Tank Pressures

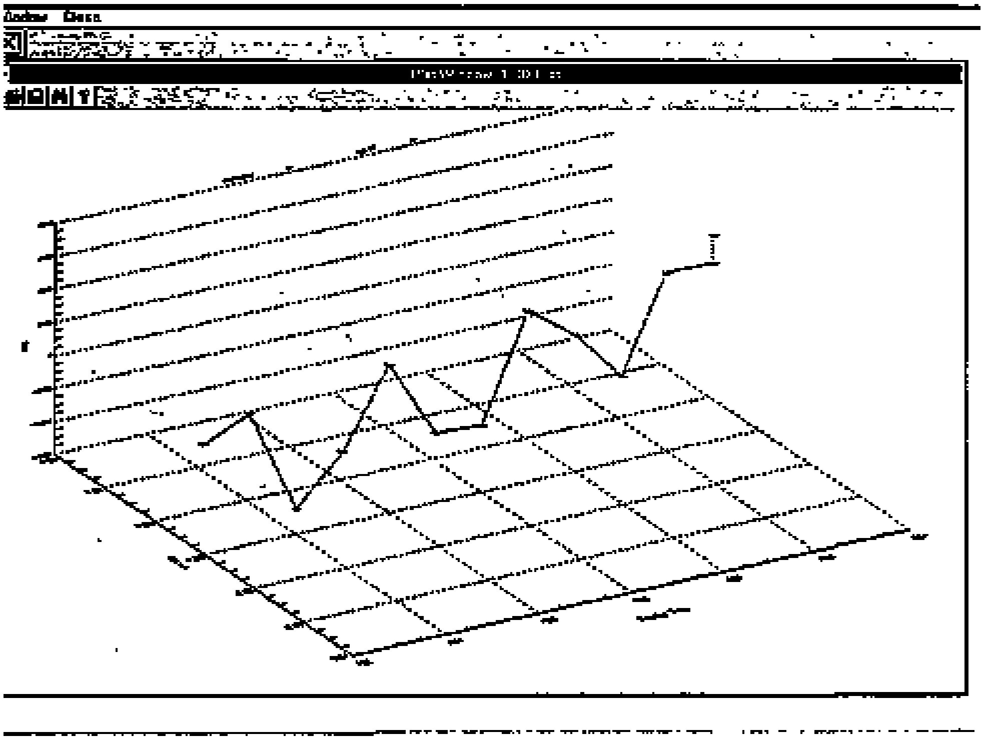
DATE	TIME	SAPOS	TANK1P	TANK2P	TANK3P	TANK4P
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

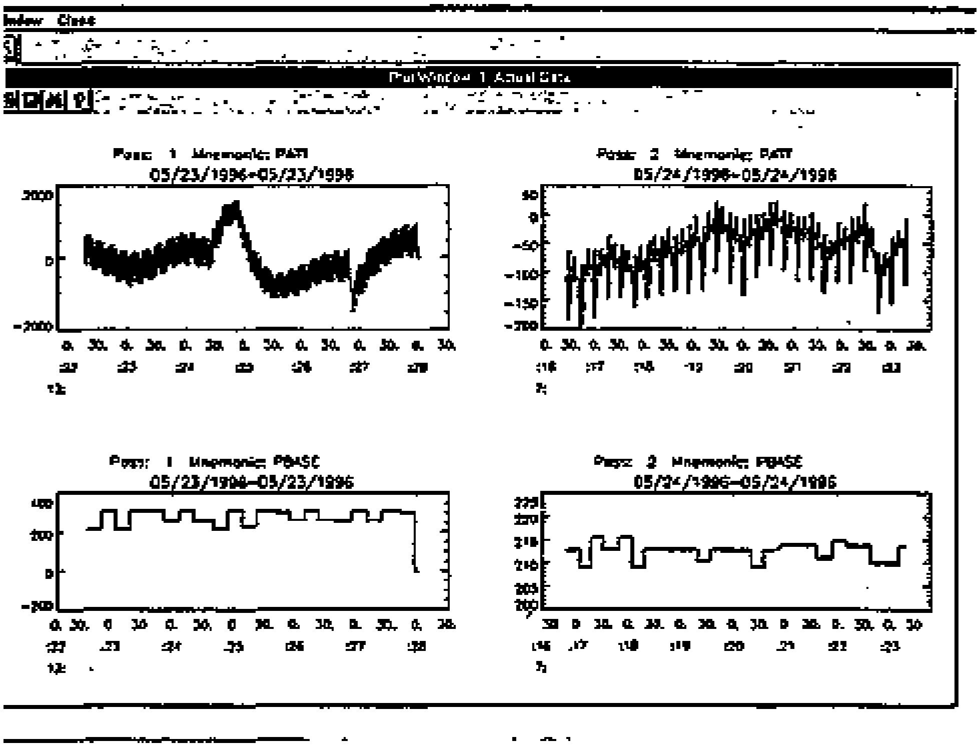
9540 Wheel Speed

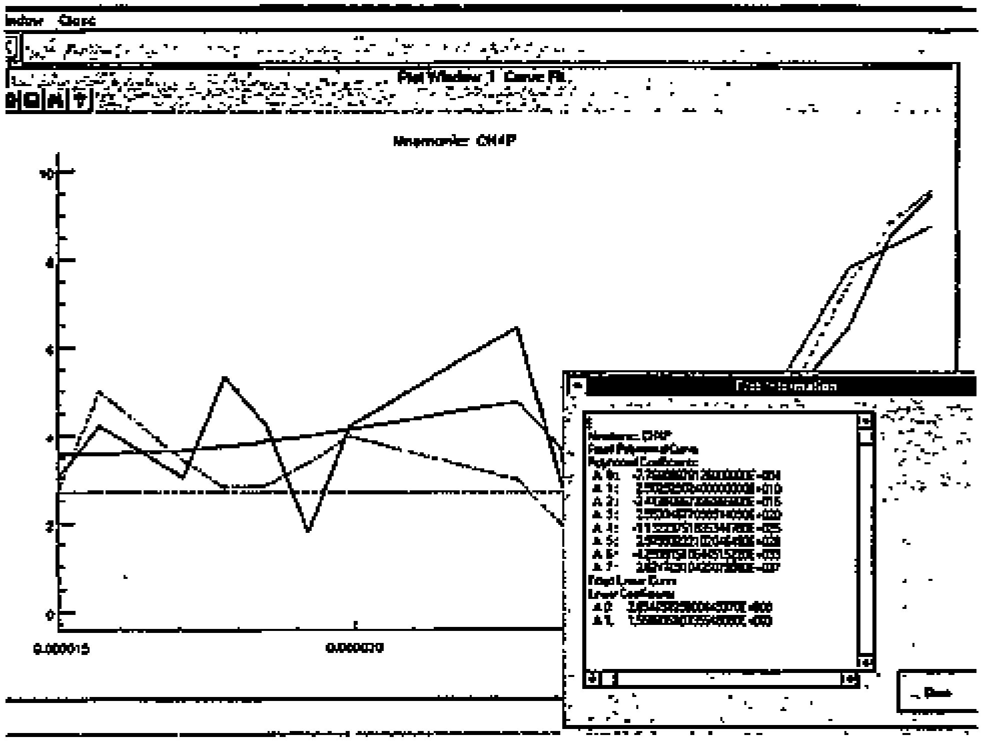
DATE	TIME	SAPOS	FRWSPD	FLWSPD	RRWSPD	RLWSPD
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Page 11

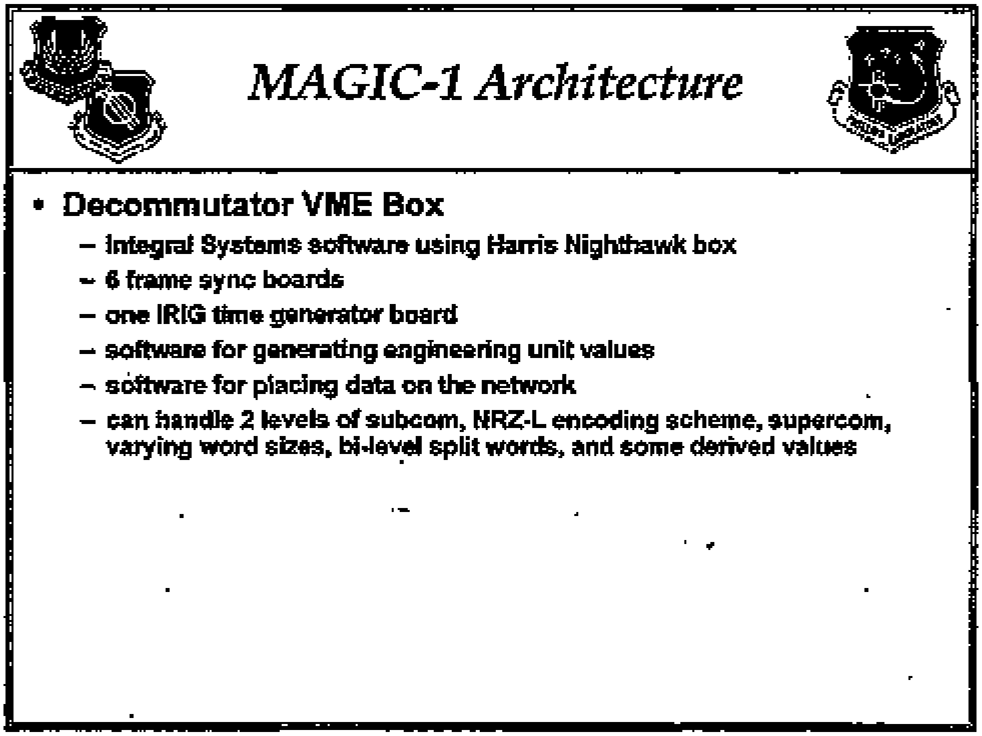

Close
Plot Window 1
CHGOUT
6
5
4
3
2
1
0
-1
1.5
1.0
0.5
0.0
-0.5
12h
0h
11
12
13
14
15
16
17
September
1998


Close



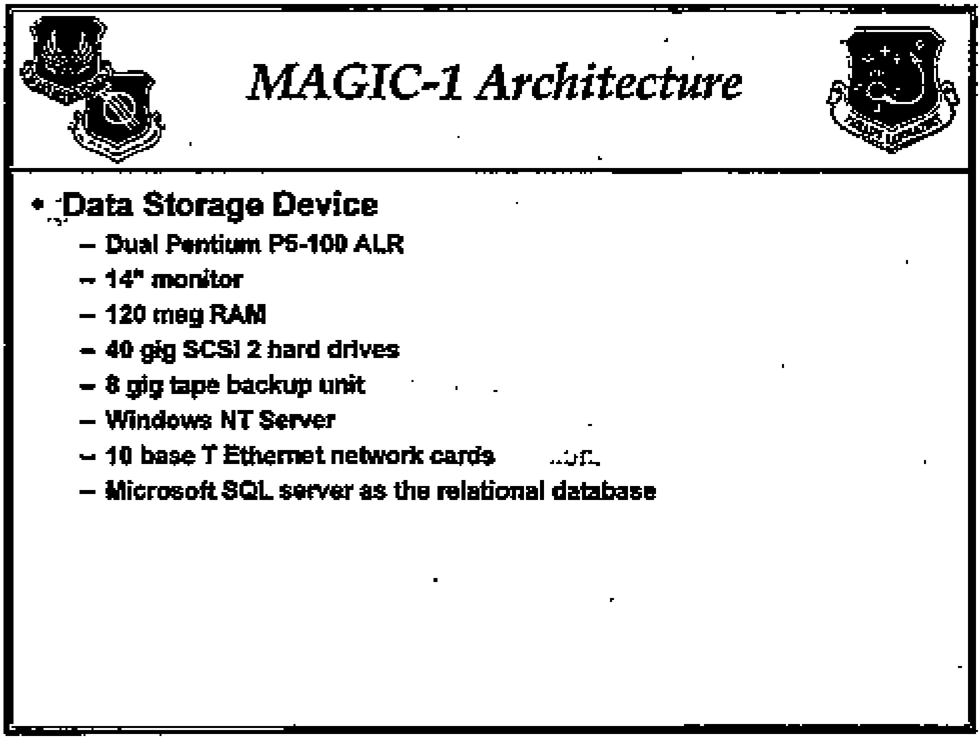


Mnemonic: CH4P
Fitted Polynomial Curve
Polynomial Coefficients:
A 0: [illegible]
A 1: [illegible]
A 2: [illegible]
A 3: [illegible]
A 4: [illegible]
A 5: [illegible]
A 6: [illegible]
A 7: [illegible]
Fitted Linear Curve
Linear Coefficients:
A 0: [illegible]
A 1: [illegible]



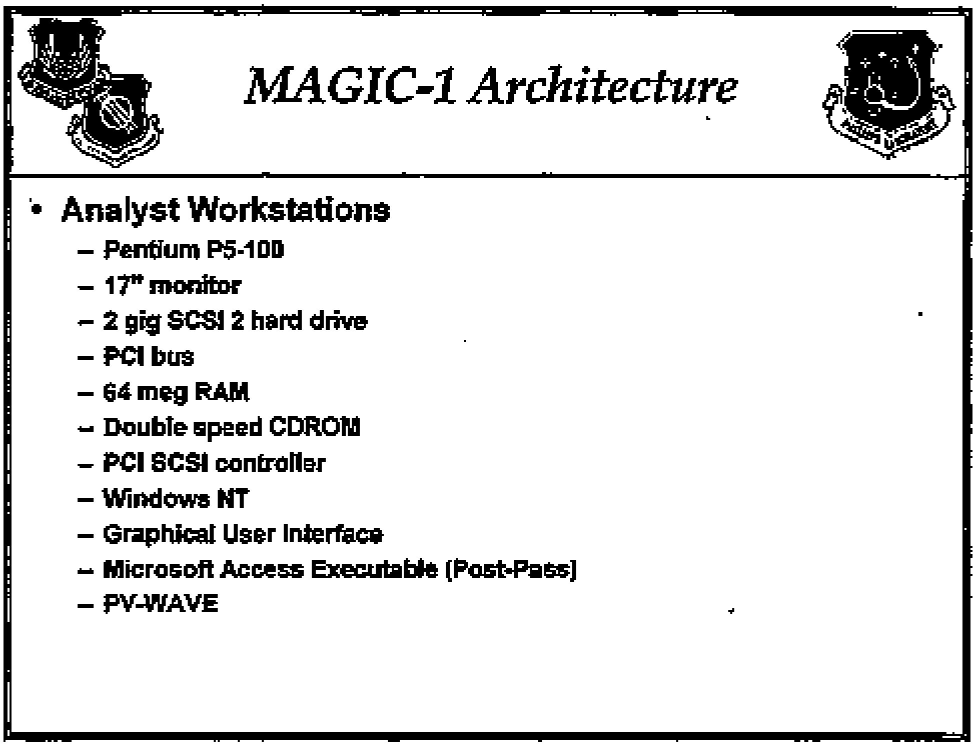
MAGIC-1 Architecture

- Decommutator VME Box
  - Integral Systems software using Harris Nighthawk box
  - 6 frame sync boards
  - one IRIG time generator board
  - software for generating engineering unit values
  - software for placing data on the network
  - can handle 2 levels of subcom, NRZ-L encoding scheme, supercom, varying word sizes, bi-level split words, and some derived values



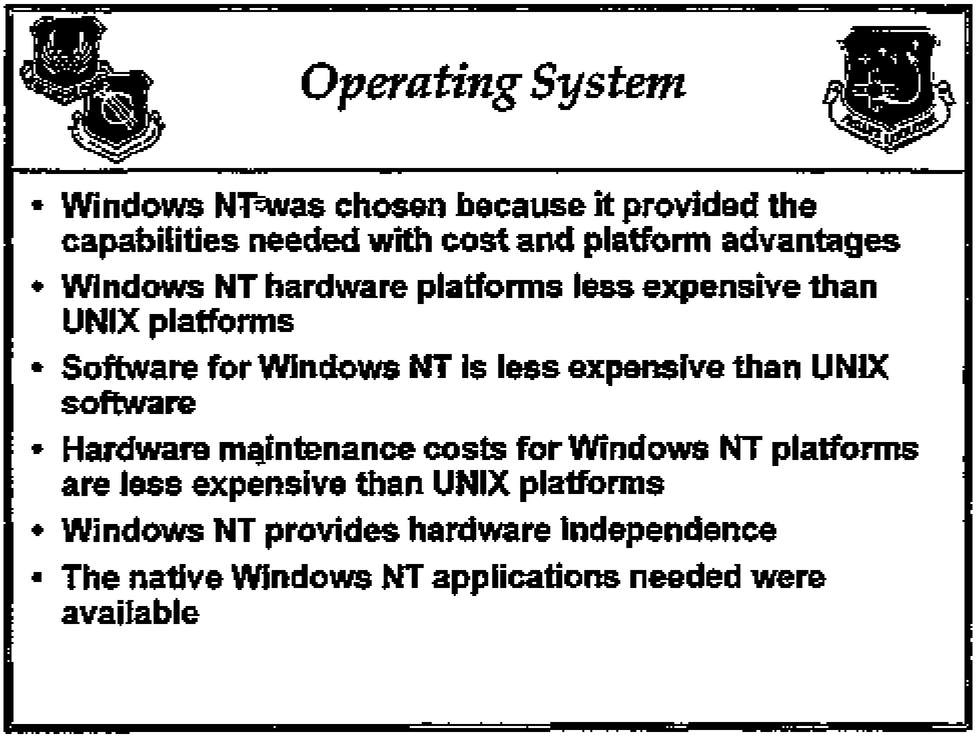
MAGIC-1 Architecture

- Data Storage Device
  - Dual Pentium P5-100 ALR
  - 14" monitor
  - 120 meg RAM
  - 40 gig SCSI 2 hard drives
  - 8 gig tape backup unit
  - Windows NT Server
  - 10 base T Ethernet network cards
  - Microsoft SQL server as the relational database



MAGIC-1 Architecture

- Analyst Workstations
  - Pentium P5-100
  - 17" monitor
  - 2 gig SCSI 2 hard drive
  - PCI bus
  - 64 meg RAM
  - Double speed CDROM
  - PCI SCSI controller
  - Windows NT
  - Graphical User Interface
  - Microsoft Access Executable (Post-Pass)
  - PV-WAVE



Operating System

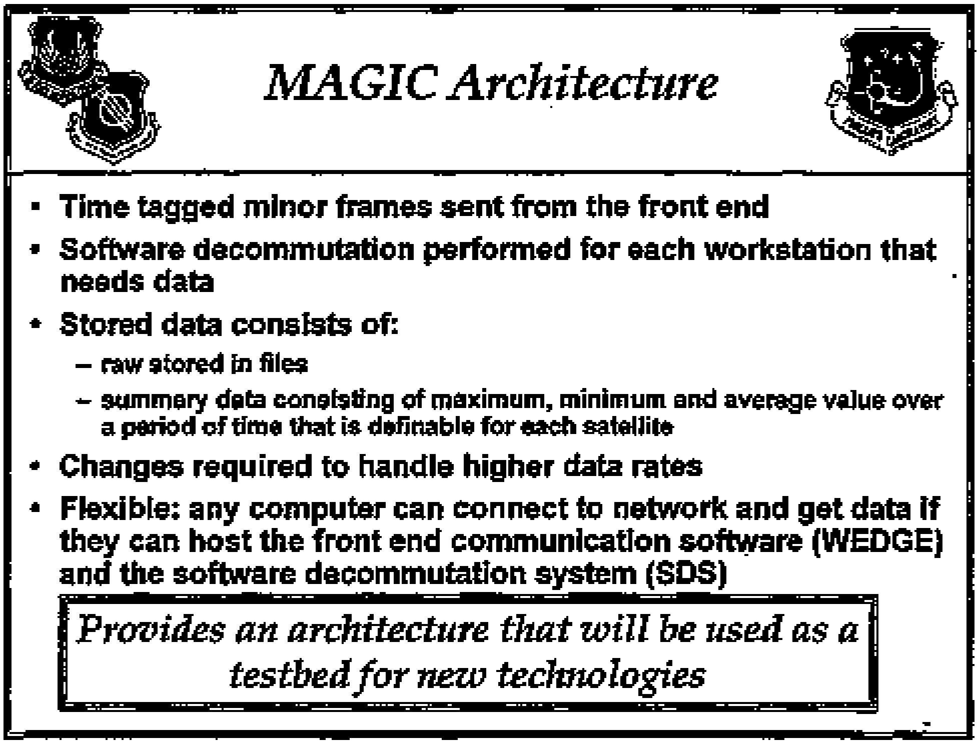
- Windows NT was chosen because it provided the capabilities needed with cost and platform advantages
- Windows NT hardware platforms less expensive than UNIX platforms
- Software for Windows NT is less expensive than UNIX software
- Hardware maintenance costs for Windows NT platforms are less expensive than UNIX platforms
- Windows NT provides hardware independence
- The native Windows NT applications needed were available

MAGIC-1 Cost

- Approximate hardware/software cost for 6 stream system is $300,000
- Cost kept down by using Windows NT-based systems, instead of a UNIX based system
- All components open. Maintenance costs low.

MAGIC-2 - Decision Support

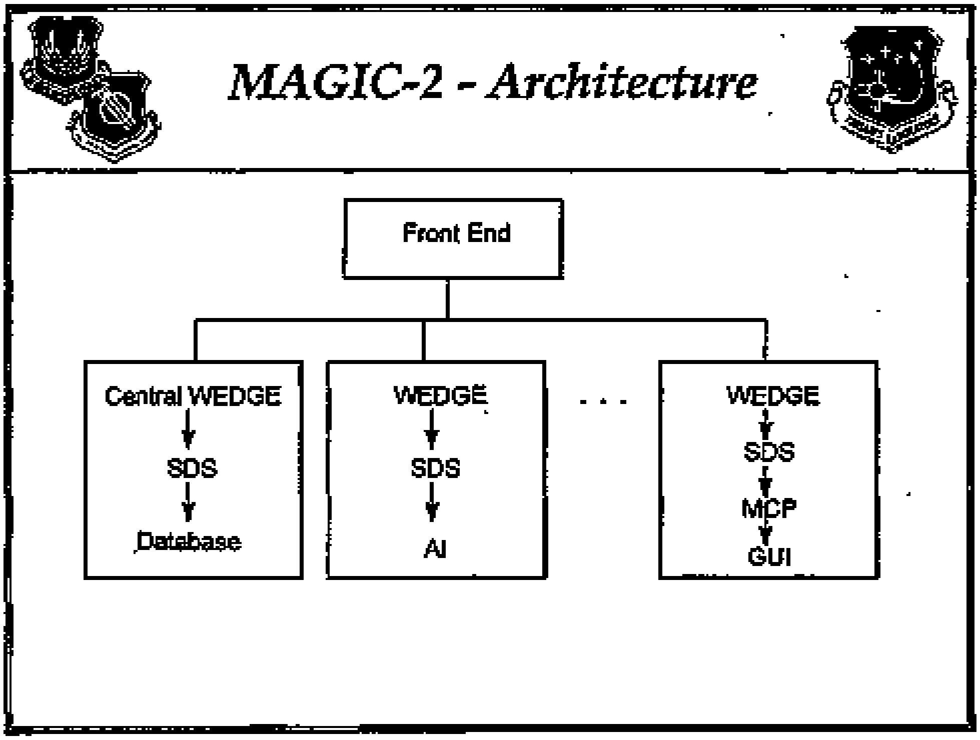
- The expert system will be enhanced to examine out-of-limit conditions and other system information to determine if it can identify a known anomaly
- If it is a known anomaly, the expert system will have a defined solution
- If it is a known anomaly, the expert system will tell the operator for approval of its decision and the proposed solution
- Only with the operator's permission will the expert system implement the proposed solution
- If it is an unknown anomaly, the expert system will give information to the operator for resolution
- Additional functions will be added, such as orbit analysis, planning, and commanding.



MAGIC Architecture

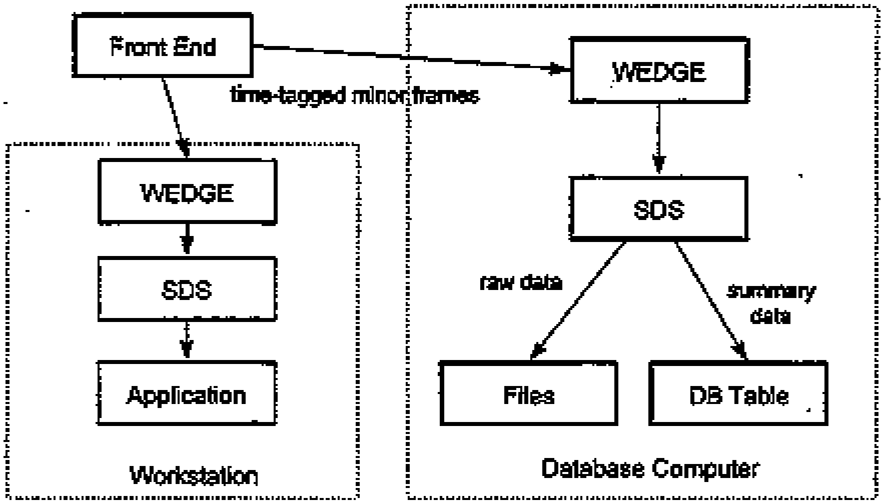
- Time tagged minor frames sent from the front end
- Software decommutation performed for each workstation that needs data
- Stored data consists of:
  - raw stored in files
  - summary data consisting of maximum, minimum and average value over a period of time that is definable for each satellite
- Changes required to handle higher data rates
- Flexible: any computer can connect to network and get data if they can host the front end communication software (WEDGE) and the software decommutation system (SDS)

Provides an architecture that will be used as a testbed for new technologies

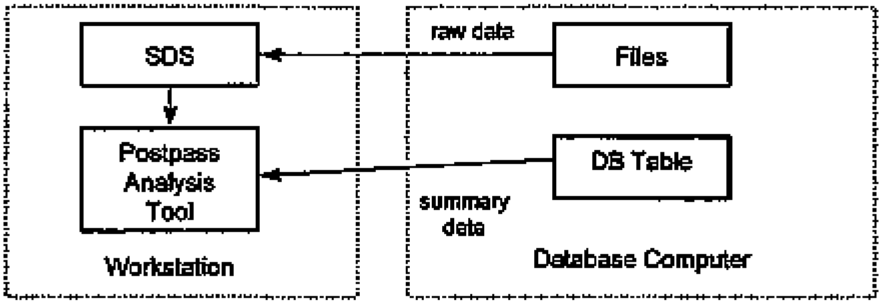


MAGIC-2 - Architecture

MAGIC-2 Real-Time



MAGIC-2 Postpass



If user requests detailed data, postpass requests SDS decommutate from the raw files.

If user requests data over a period of time, postpass uses the summary data stored in tables in the database.

MAGIC-2 - Independent Decisions

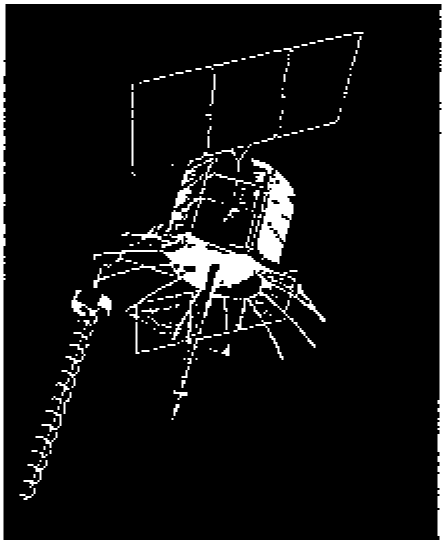
- The decisions the expert system makes will be compared to those of the satellite experts to ensure that expert system is mature
- When confidence in the expert system is achieved, the system will be permitted to make independent decisions on known anomalies without prior operator approval
- For an unknown anomaly, the expert system will provide information to the operator and provide support in anomaly resolution
- Note: known anomalies are those that have been identified and have defined solutions before the anomaly occurs

MAGIC - 3 Intelligent Operator Training

- Current training approach
  - Canned simulations (rote learning)
  - Separate from the operational system (non-realistic training)
  - Human trainer presence required
- MAGIC approach
  - Reactive, dynamic training (AI-based)
  - Integrated with the operational system
  - No human trainer required
  - Computers maintains model of student progress to customize training
  - Reduces cost by using the operational system as its core

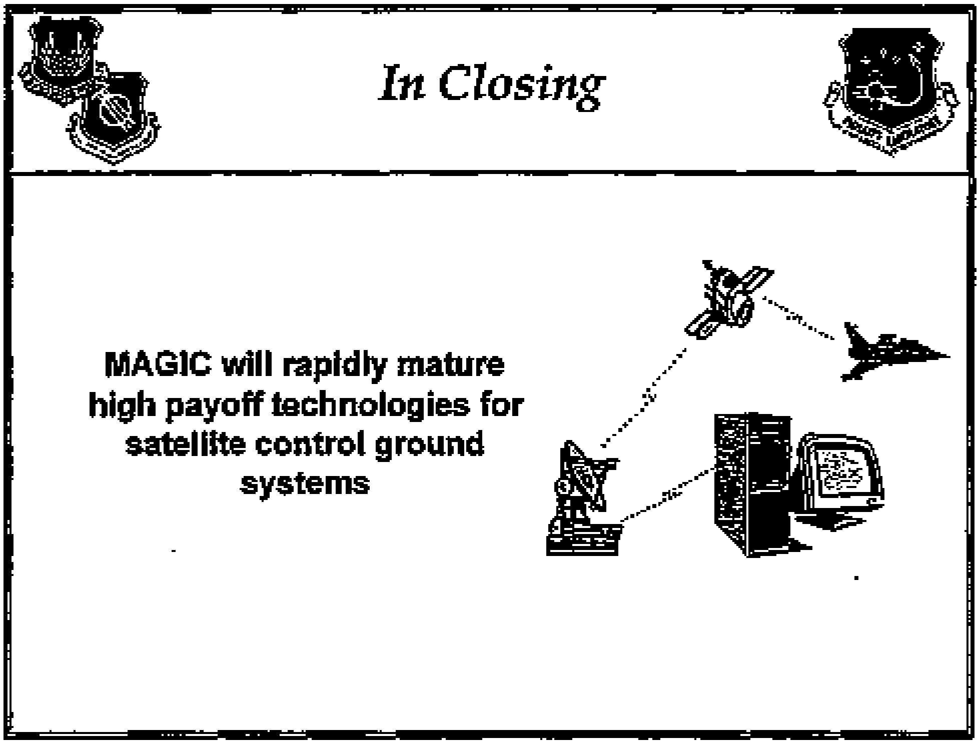
Future Programs

- **Satellite Autonomy**
  - Once the expert system has been verified and validated, portions can move to the satellite
  - Placing an expert system on the satellite will reduce the amount of data that must be sent from the satellite to the ground
  - The first area that will be examined for autonomy is health and status



Future Programs

- **Machine Learning in Ground Control and Autonomous Satellites**
  - The knowledge in ground control stations and intelligent satellites will have to be continuously updated.
  - Updates are required to:
    - add the increasing available knowledge about the satellites gathered as they age
    - Changes that occur in the satellites as they age
  - The knowledge can be manually changed, but it would be better for the system to learn as it gains experience with the satellite.
  - Techniques for machine learning will be investigated and a prototype ground/satellite system will be developed and tested.



In Closing

MAGIC will rapidly mature high payoff technologies for satellite control ground systems

1997 • SOFTWARE QUALITY FORUM • 1997 •
DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Session Y2: Software Inspection Process Overview

Larry Lane & Randy Dabbs
Sandia National Laboratories

Overview of the Software Inspection Process

G. Lawrence Lane and Randy Dabbs
Sandia National Laboratories

The Software Inspection is a formal in-process review method that provides immediate improvement in software product quality and produces metrics that indicate opportunities for process improvement. When adopted as a part of a defined, repeatable software development methodology, Software Inspections provide a mechanism for process control. The Software Inspection Process is not limited to formal reviews of code but applies to all software products. Software Inspections have consistently been shown to be very cost effective and is one of the most efficient ways to remove defects in all software products.

This tutorial introduces attendees to the Inspection Process and teaches them how to organize and participate in a software inspection. The tutorial advocates the benefits of inspections and encourages attendees to socialize the inspection process in their organizations.

The processes which are introduced in this tutorial agree with the methods recommended in the Sandia Preferred Processes for Software Development.

BIOGRAPHIES

G. Lawrence Lane

Larry Lane is a Senior Member of the Technical Staff at Sandia National Laboratories. He earned a Master of Arts Degree in mathematics from the University of Kansas. Larry joined Sandia Corporation in 1959 as an assembly language programmer in the field data reduction department. He has also worked as a operating systems programmer and was responsible for the selection and installation of Sandia's first general purpose time sharing computer. Larry also worked as a computer consultant for large scientific computers, as the second computer ombudsman, and was responsible for the development of an electronic tracking system for electrical testing of radiation-hardened microcircuits.

Larry moved to his current position in the Quality Engineering Department in 1991, where he is an instructor for the Software Quality Engineering course and the Software Inspection Class. As a software quality engineer, Larry has led numerous qualification efforts for new and upgraded software projects, particularly in the areas of use control and weapon security. He has helped develop and teach a customized version of the software inspection course to meet specific Sandia organizational needs.

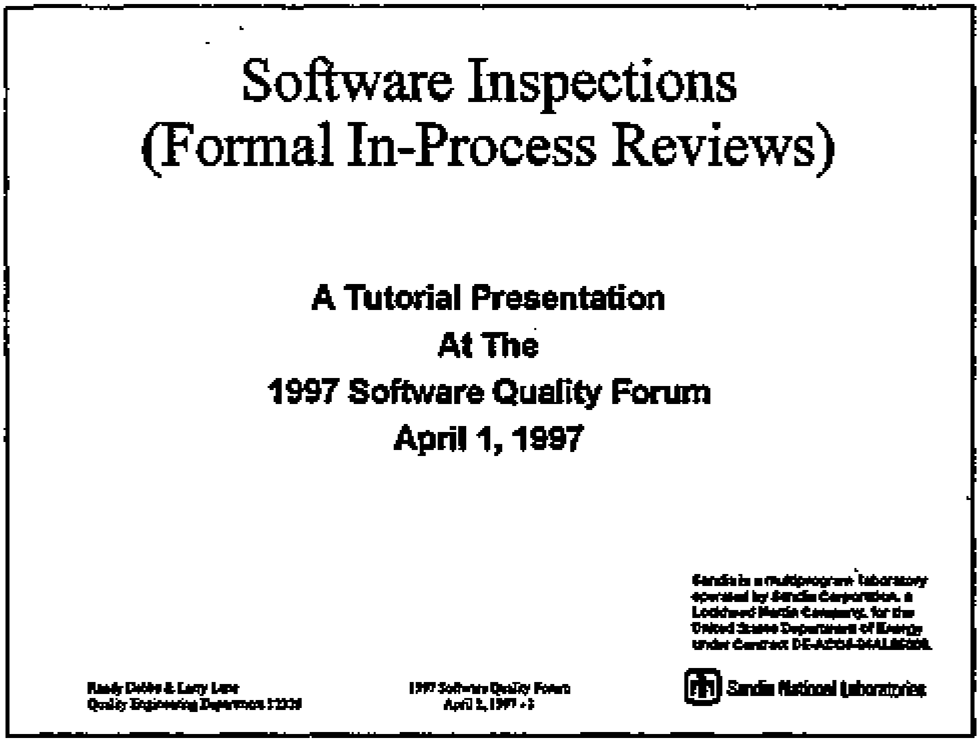
Randy Dabbs

Randy Dabbs is a Senior Member of Technical Staff at Sandia National Laboratories. He has earned a Master of Science in Electrical Engineering from the University of New Mexico. He has held positions at the Sandia Particle Beam Fusion Accelerator in the areas of data acquisition and signal processing; the Kwajalein Missile Range in the areas of range computer systems engineering, range operations, tracking software modeling and development, reentry mission project engineering, digital radar signal processing, radar controller real time software, and software configuration management; and the Sandia Kauai Test Facility in the areas of range computer support and operations, range safety software development, countdown software development, CASE tool selection and modeling of range operational software.

In his current position with the Sandia Quality Engineering Department, he has participated in instructing the Software Quality Engineering course and the Software Inspections course. In his role as software quality assurance engineer, he has participated in numerous software inspections for both internal and external customers. In addition, he has helped develop and teach a customized version of the software inspection course to meet the specific needs of Sandia organizations.

G. Lawrence "Larry" Lane
Sandia National Laboratories
PO Box 5800, MS0638
Albuquerque, NM, 87185-0638
Voice: (505)845-9122
email: gllane@sandia.gov

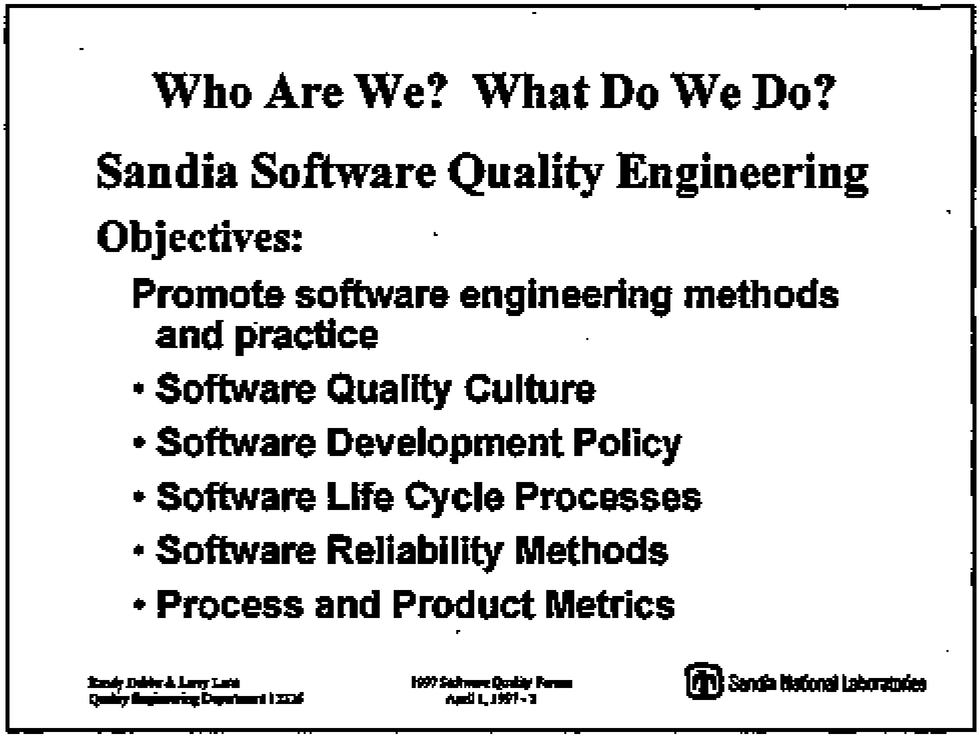
Randy Dabbs
Sandia National Laboratories
P.O. Box 5800 MS-0638
Albuquerque, NM 87185-0638
Voice: 505-845-9232
email: rddabbs@sandia.gov



Software Inspections
(Formal In-Process Reviews)

A Tutorial Presentation
At The
1997 Software Quality Forum
April 1, 1997

Sandia is a multiprogram laboratory
operated by Sandia Corporation, a
Lockheed Martin Company, for the
United States Department of Energy
under Contract DE-AC04-94AL85000.



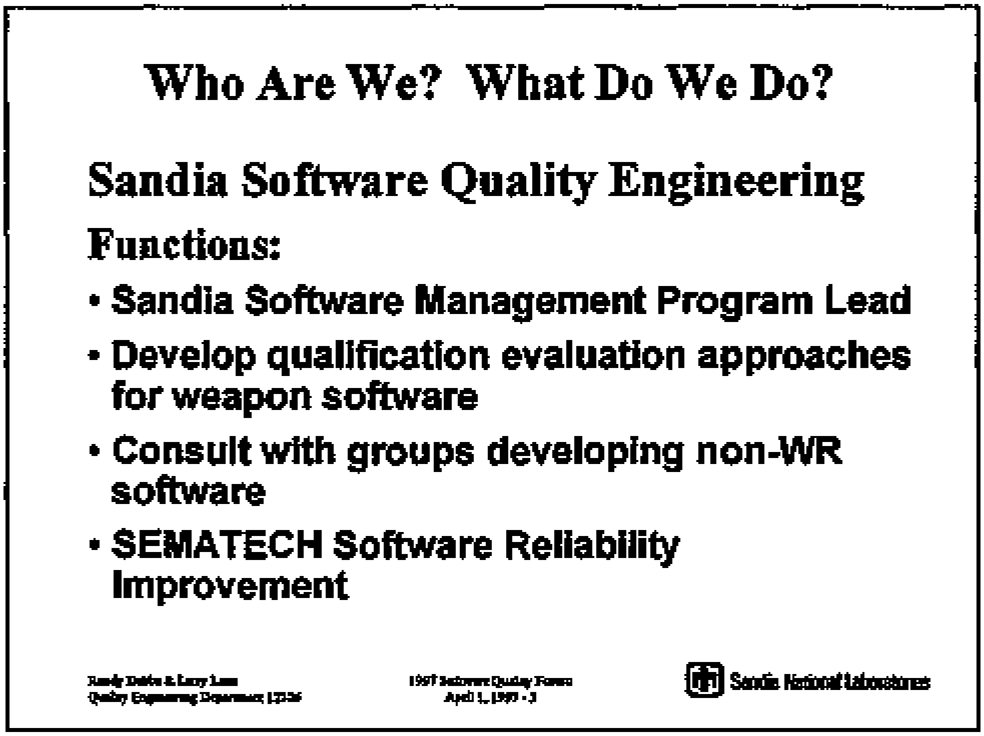
Who Are We? What Do We Do?

Sandia Software Quality Engineering

Objectives:

Promote software engineering methods and practice

- **Software Quality Culture**
- **Software Development Policy**
- **Software Life Cycle Processes**
- **Software Reliability Methods**
- **Process and Product Metrics**



Who Are We? What Do We Do?

Sandia Software Quality Engineering

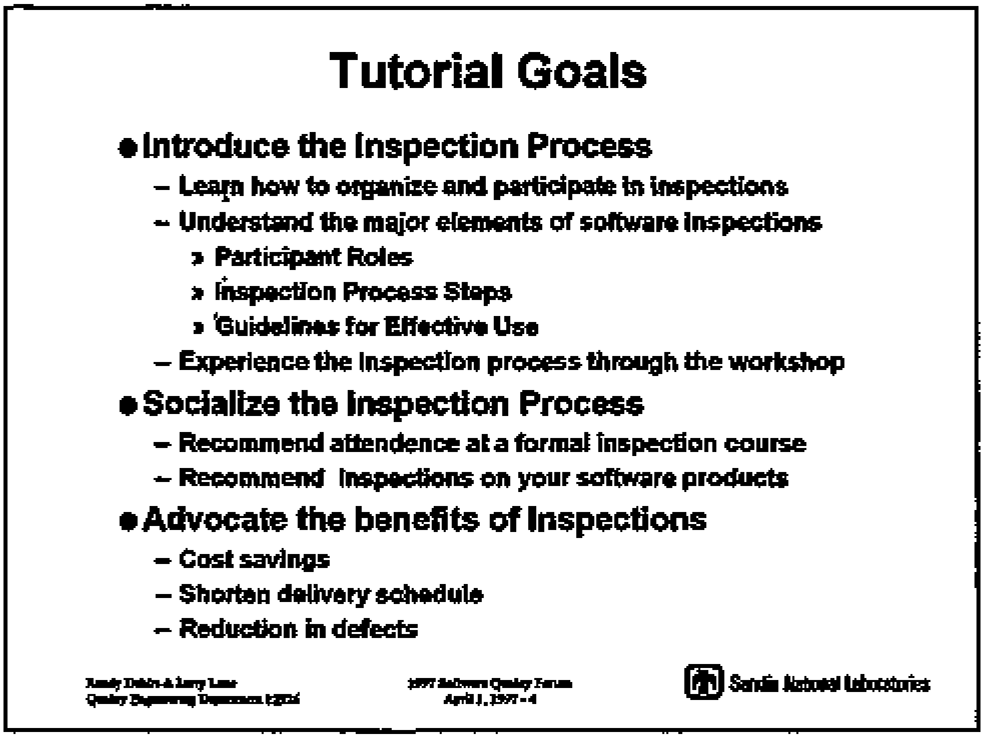
Functions:

- Sandia Software Management Program Lead
- Develop qualification evaluation approaches for weapon software
- Consult with groups developing non-WR software
- SEMATECH Software Reliability Improvement

Randy Dable & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 3

Sandia National Laboratories



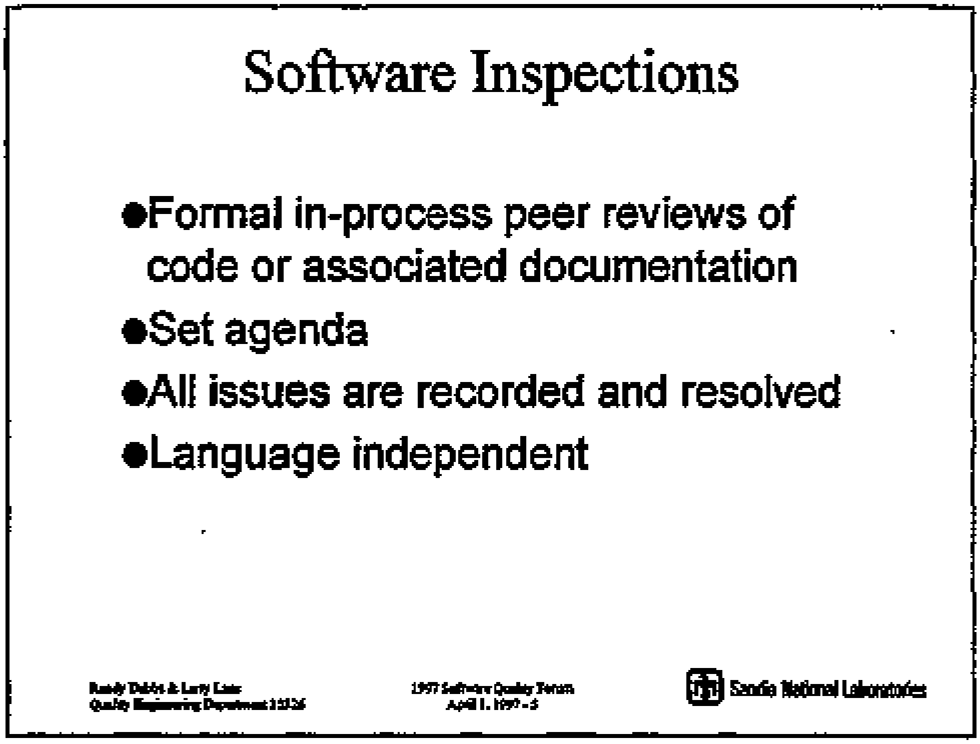
Tutorial Goals

- Introduce the Inspection Process
  - Learn how to organize and participate in inspections
  - Understand the major elements of software inspections
    - Participant Roles
    - Inspection Process Steps
    - Guidelines for Effective Use
  - Experience the inspection process through the workshop
- Socialize the Inspection Process
  - Recommend attendence at a formal inspection course
  - Recommend inspections on your software products
- Advocate the benefits of Inspections
  - Cost savings
  - Shorten delivery schedule
  - Reduction in defects

Randy Dable & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 4

Sandia National Laboratories



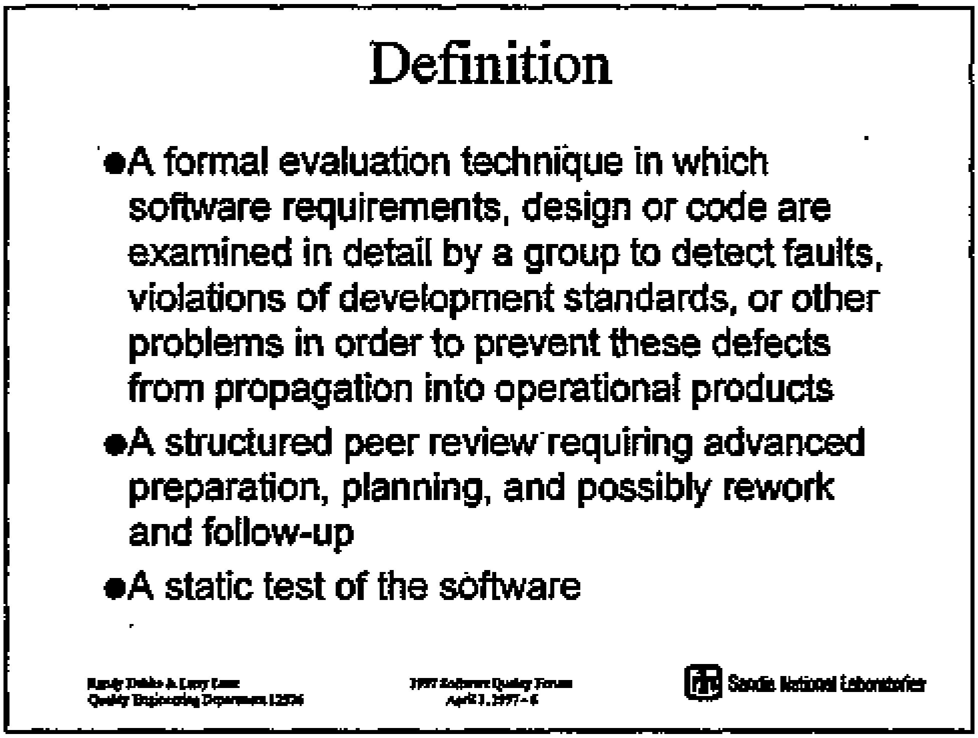
Software Inspections

- Formal in-process peer reviews of code or associated documentation
- Set agenda
- All issues are recorded and resolved
- Language independent

Randy Dabbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 5

Sandia National Laboratories



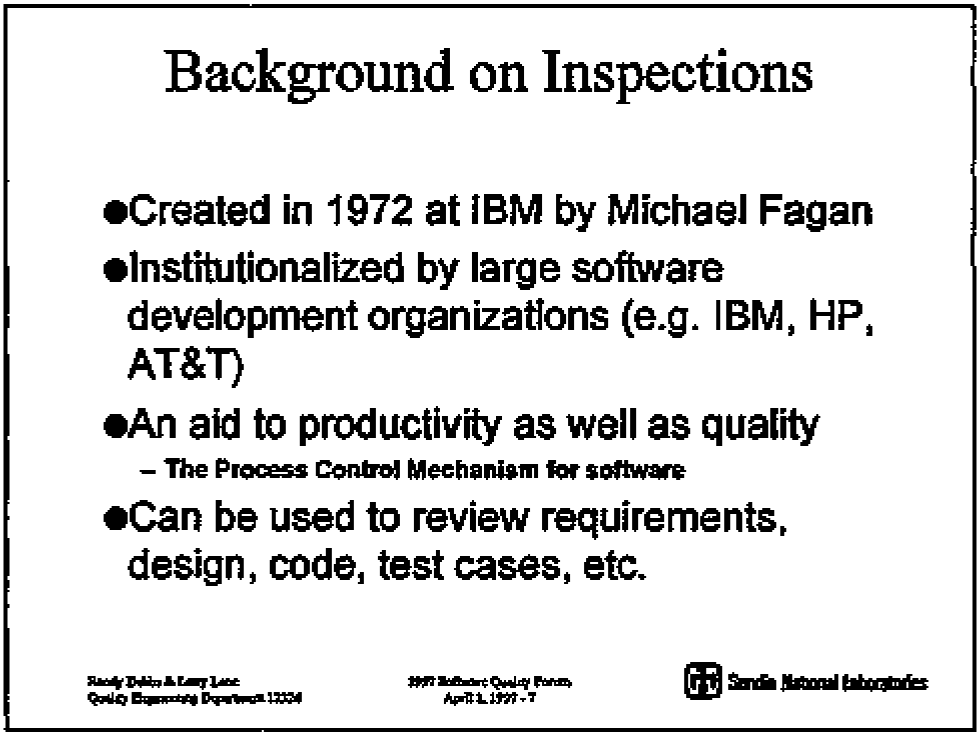
Definition

- A formal evaluation technique in which software requirements, design or code are examined in detail by a group to detect faults, violations of development standards, or other problems in order to prevent these defects from propagation into operational products
- A structured peer review requiring advanced preparation, planning, and possibly rework and follow-up
- A static test of the software

Randy Dabbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 6

Sandia National Laboratories



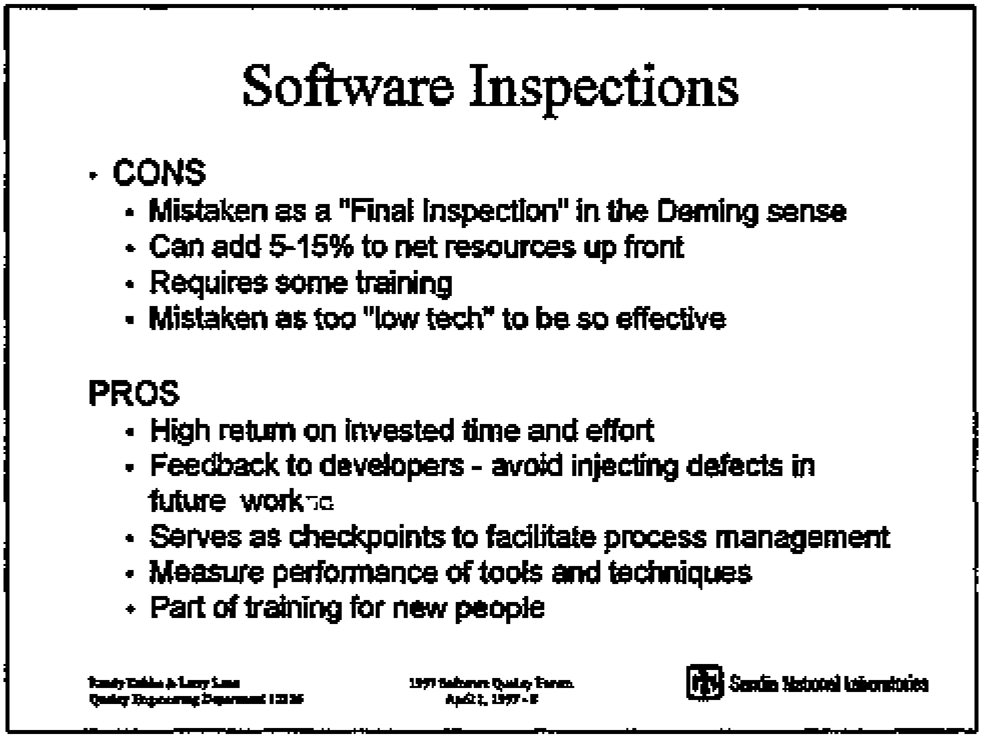
Background on Inspections

- ●Created in 1972 at IBM by Michael Fagan
- ●Institutionalized by large software development organizations (e.g. IBM, HP, AT&T)
- ●An aid to productivity as well as quality
  - The Process Control Mechanism for software
- ●Can be used to review requirements, design, code, test cases, etc.

Randy Dobbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 7

Sandia National Laboratories



Software Inspections

- CONS
  - Mistaken as a "Final Inspection" in the Deming sense
  - Can add 5-15% to net resources up front
  - Requires some training
  - Mistaken as too "low tech" to be so effective

PROS
- High return on invested time and effort
- Feedback to developers - avoid injecting defects in future work
- Serves as checkpoints to facilitate process management
- Measure performance of tools and techniques
- Part of training for new people

Randy Dobbs & Larry Lane
Quality Engineering Department 12326

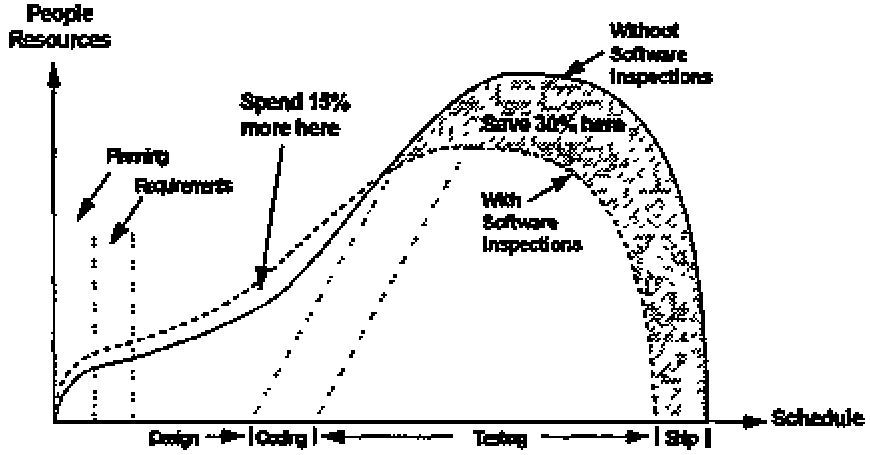
1997 Software Quality Forum
April 1, 1997 - 8

Sandia National Laboratories

Benefits of Inspections

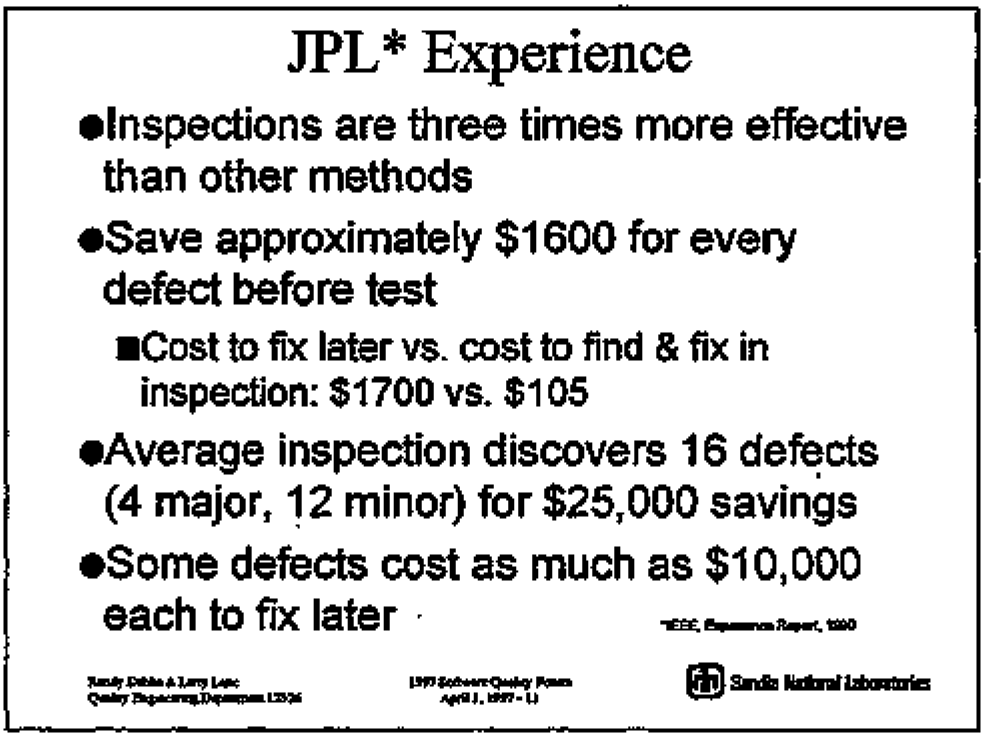
- Defect Reduction
  - 50-90% of all defects discovered by inspection
- Cost Improvement
  - 10-25% reduction in development costs
  - Up-to 95% reduction in corrective maintenance costs
- Staff Hours
  - Overall reduction by 10-40%
  - Shortens tail end of schedule

Inspection Experience Summary



(Fagan '86)

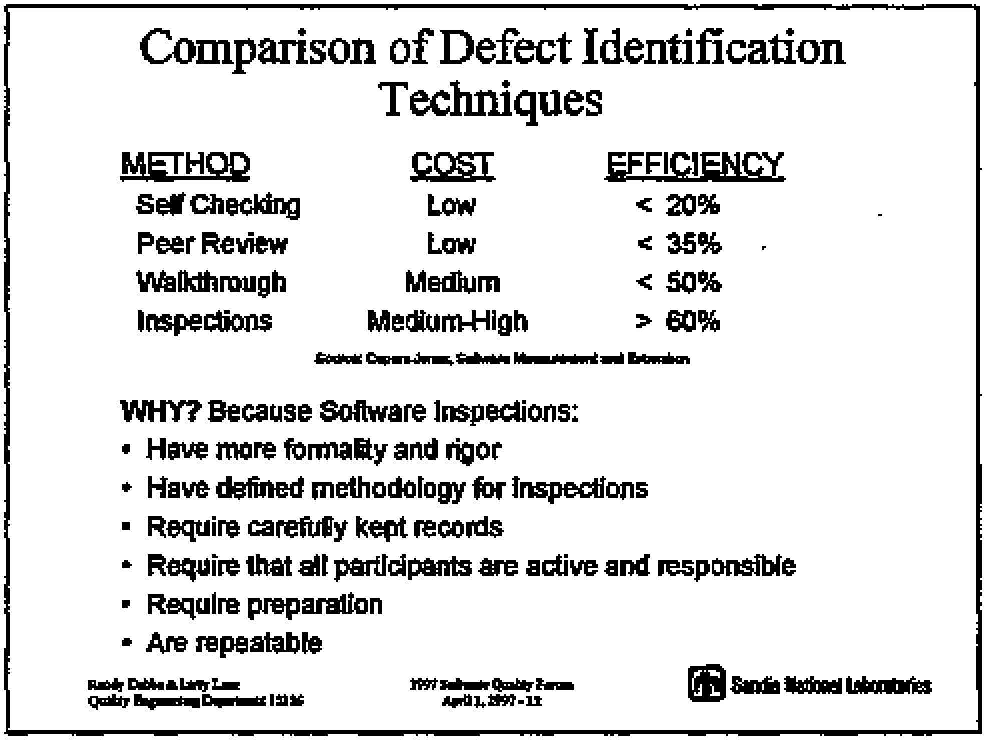
The difference in the area under these two curves after they intersect represents the additional effort expended as in a "Final Inspection" to detect all the defects introduced during a development process in which the software inspection technique is not used.



JPL* Experience

- Inspections are three times more effective than other methods
- Save approximately $1600 for every defect before test
  - Cost to fix later vs. cost to find & fix in inspection: $1700 vs. $105
- Average inspection discovers 16 defects (4 major, 12 minor) for $25,000 savings
- Some defects cost as much as $10,000 each to fix later

*IEEE, Experience Report, 1990



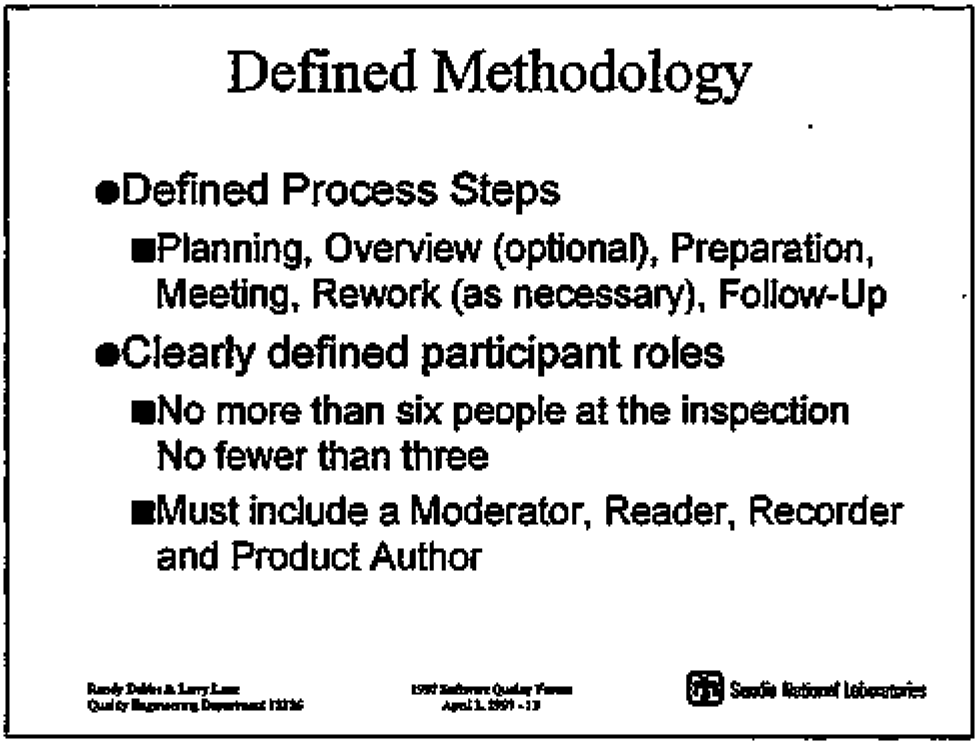
Comparison of Defect Identification Techniques

METHOD	COST	EFFICIENCY
Self Checking	Low	< 20%
Peer Review	Low	< 35%
Walkthrough	Medium	< 50%
Inspections	Medium-High	> 60%

Source: Capers Jones, Software Measurement and Estimation

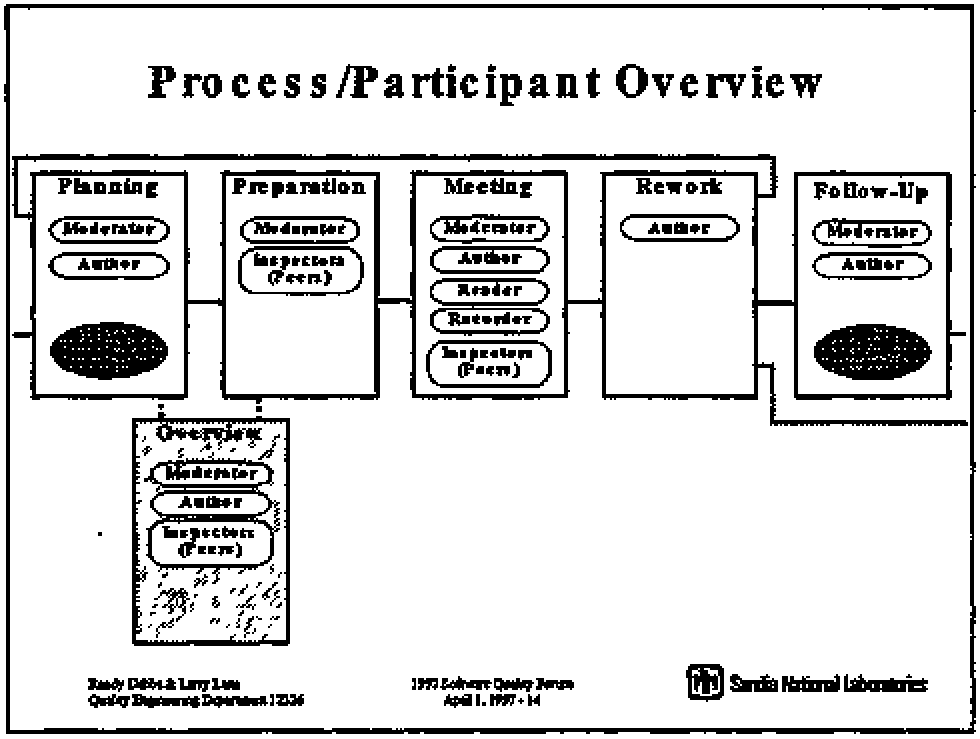
WHY? Because Software Inspections:

- Have more formality and rigor
- Have defined methodology for inspections
- Require carefully kept records
- Require that all participants are active and responsible
- Require preparation
- Are repeatable

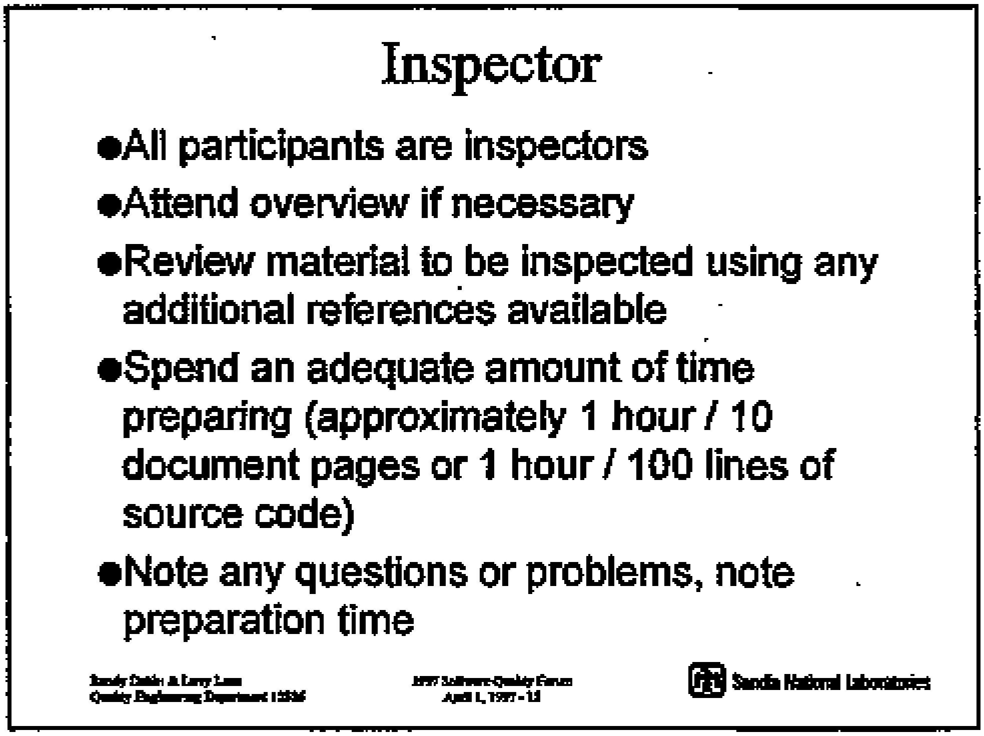


Defined Methodology

- Defined Process Steps
  - Planning, Overview (optional), Preparation, Meeting, Rework (as necessary), Follow-Up
- Clearly defined participant roles
  - No more than six people at the inspection No fewer than three
  - Must include a Moderator, Reader, Recorder and Product Author

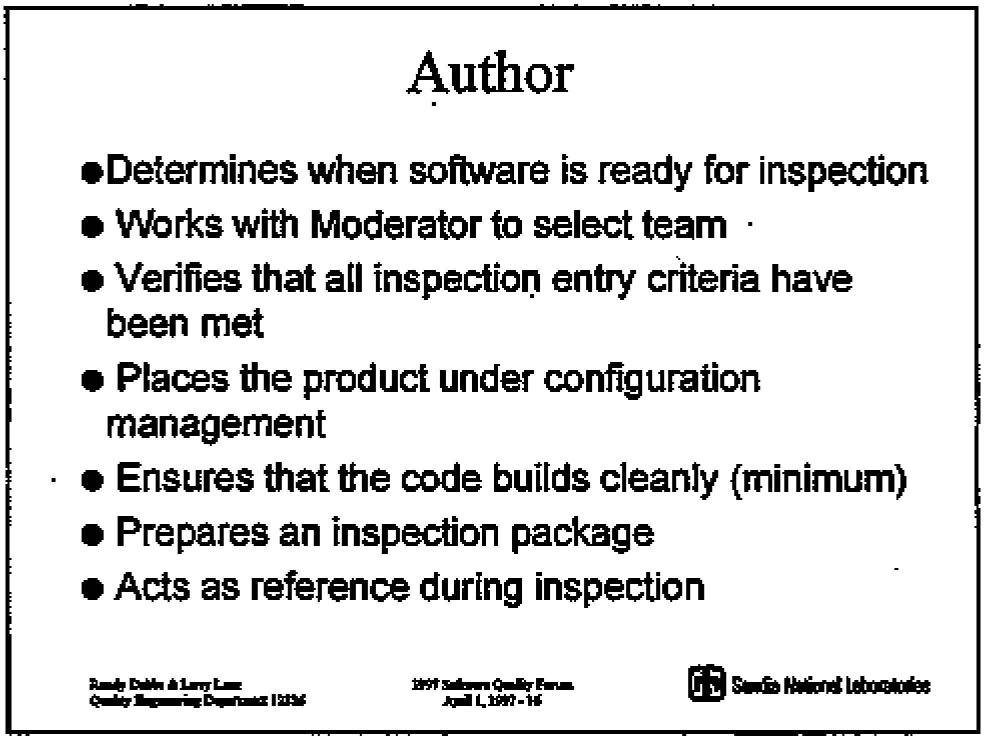


Process/Participant Overview



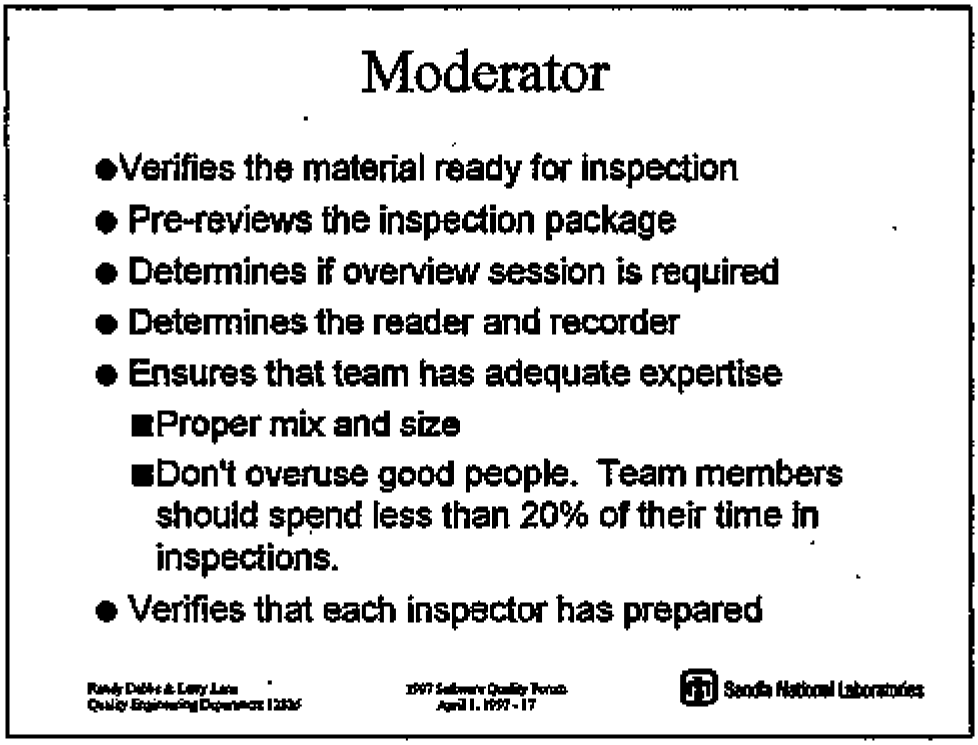
Inspector

- All participants are inspectors
- Attend overview if necessary
- Review material to be inspected using any additional references available
- Spend an adequate amount of time preparing (approximately 1 hour / 10 document pages or 1 hour / 100 lines of source code)
- Note any questions or problems, note preparation time



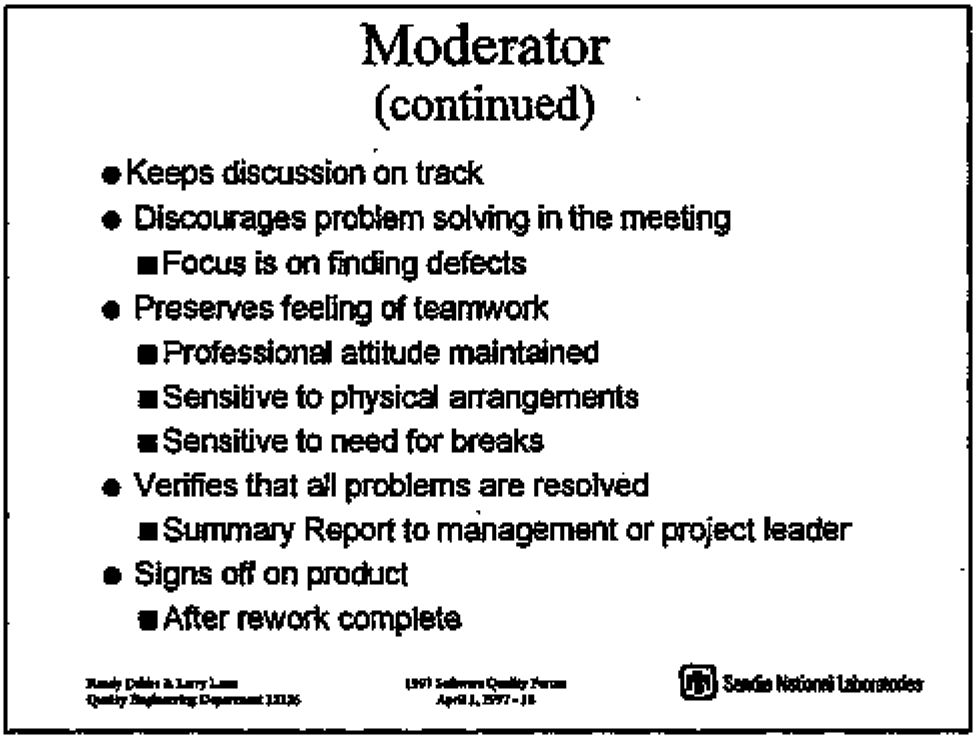
Author

- Determines when software is ready for inspection
- Works with Moderator to select team
- Verifies that all inspection entry criteria have been met
- Places the product under configuration management
- Ensures that the code builds cleanly (minimum)
- Prepares an inspection package
- Acts as reference during inspection



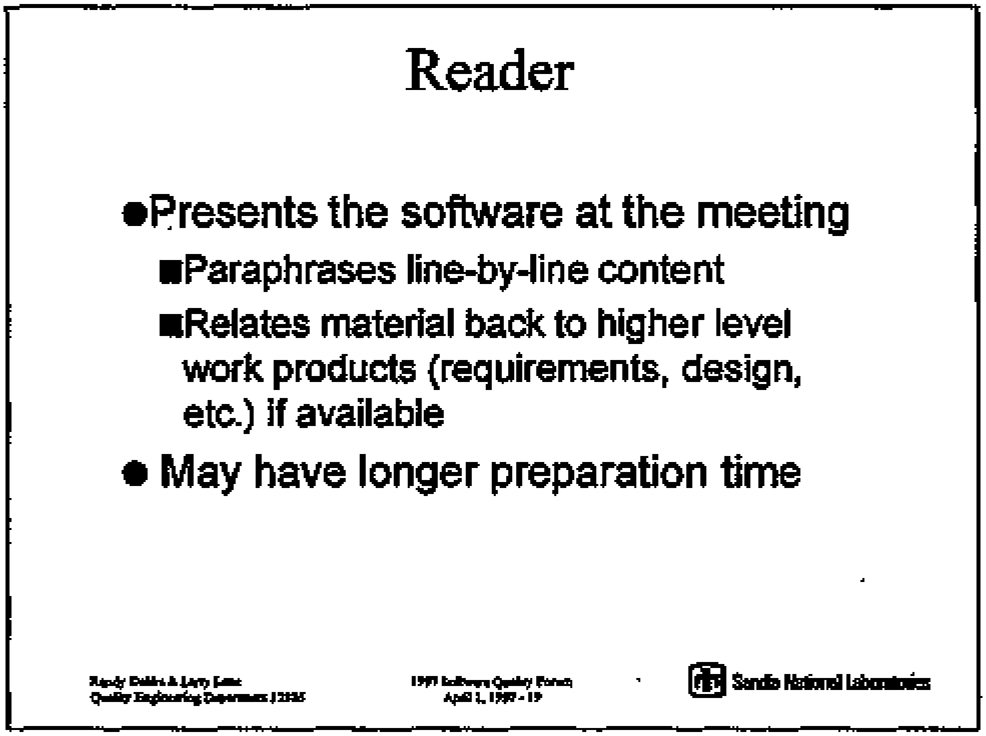
Moderator

- Verifies the material ready for inspection
- Pre-reviews the inspection package
- Determines if overview session is required
- Determines the reader and recorder
- Ensures that team has adequate expertise
  - Proper mix and size
  - Don't overuse good people. Team members should spend less than 20% of their time in inspections.
- Verifies that each inspector has prepared



Moderator
(continued)

- Keeps discussion on track
- Discourages problem solving in the meeting
  - Focus is on finding defects
- Preserves feeling of teamwork
  - Professional attitude maintained
  - Sensitive to physical arrangements
  - Sensitive to need for breaks
- Verifies that all problems are resolved
  - Summary Report to management or project leader
- Signs off on product
  - After rework complete



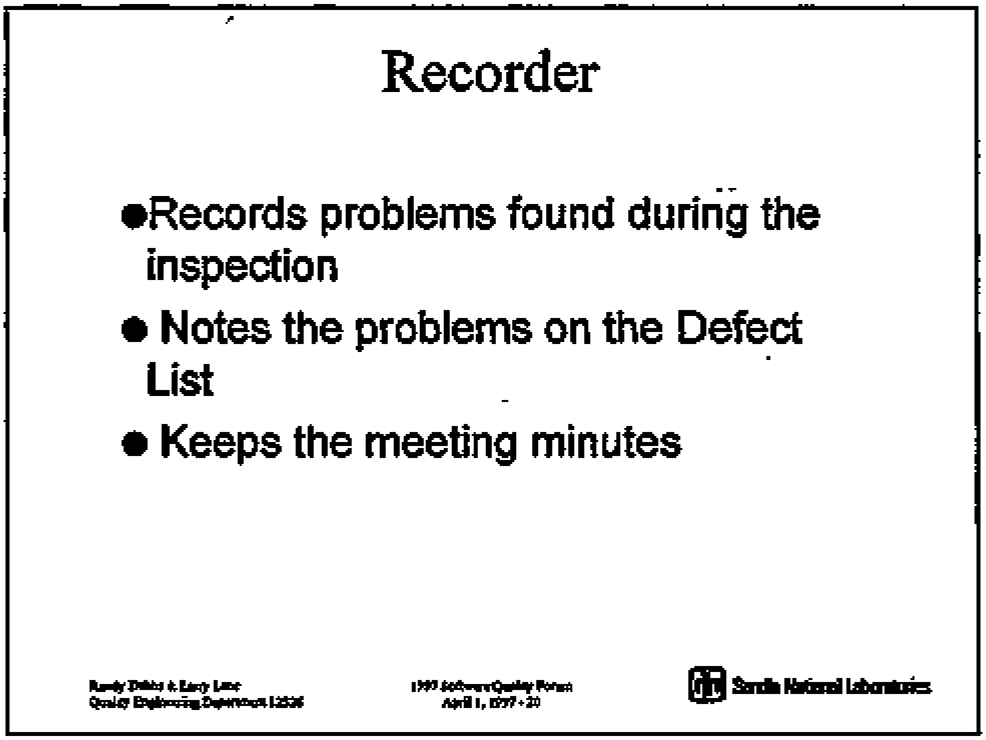
Reader

- Presents the software at the meeting
  - ■Paraphrases line-by-line content
  - ■Relates material back to higher level work products (requirements, design, etc.) if available
- May have longer preparation time

Randy Dibbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 19

Sandia National Laboratories



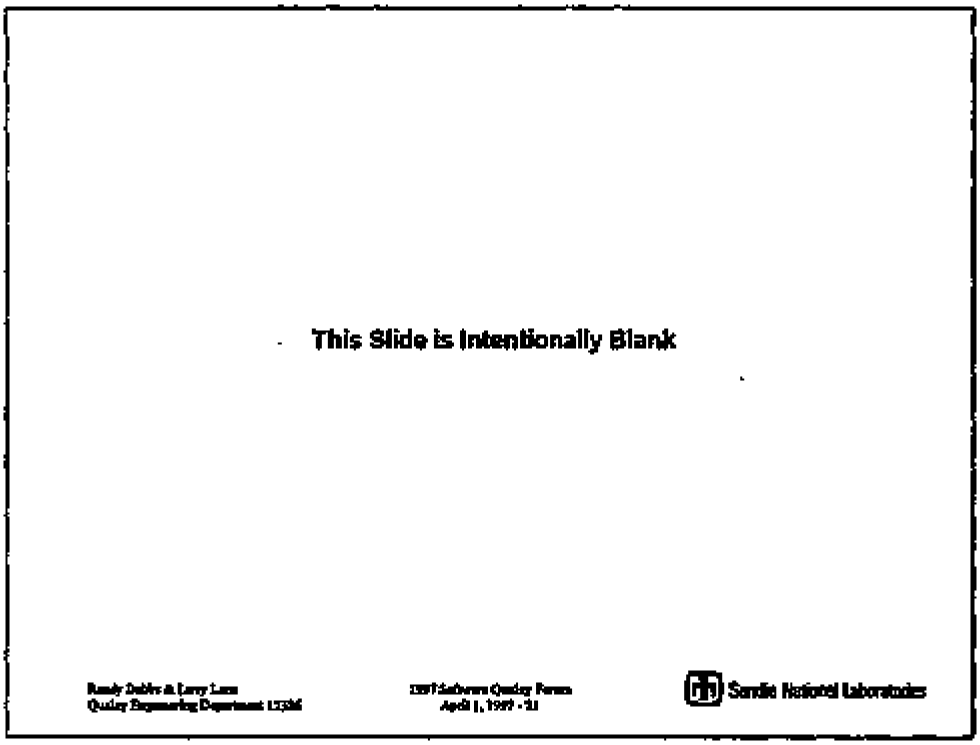
Recorder

- Records problems found during the inspection
- Notes the problems on the Defect List
- Keeps the meeting minutes

Randy Dibbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 20

Sandia National Laboratories

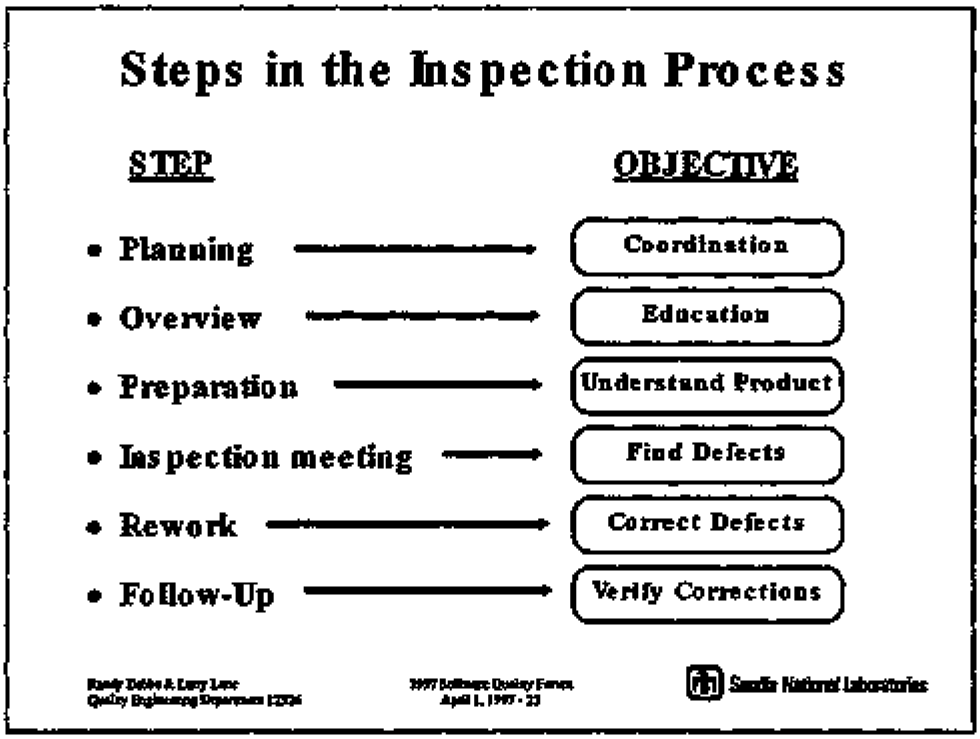


This Slide is Intentionally Blank

Randy Dobbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 21

Sandia National Laboratories



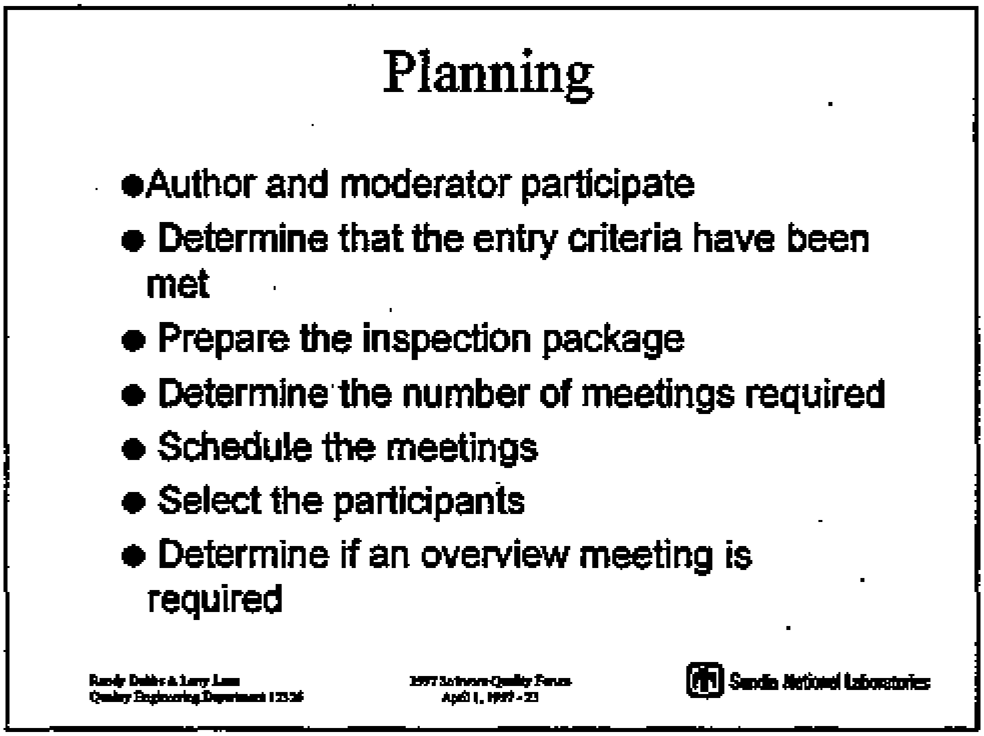
Steps in the Inspection Process

STEP	OBJECTIVE
• Planning	Coordination
• Overview	Education
• Preparation	Understand Product
• Inspection meeting	Find Defects
• Rework	Correct Defects
• Follow-Up	Verify Corrections

Randy Dobbs & Larry Lane
Quality Engineering Department 12326

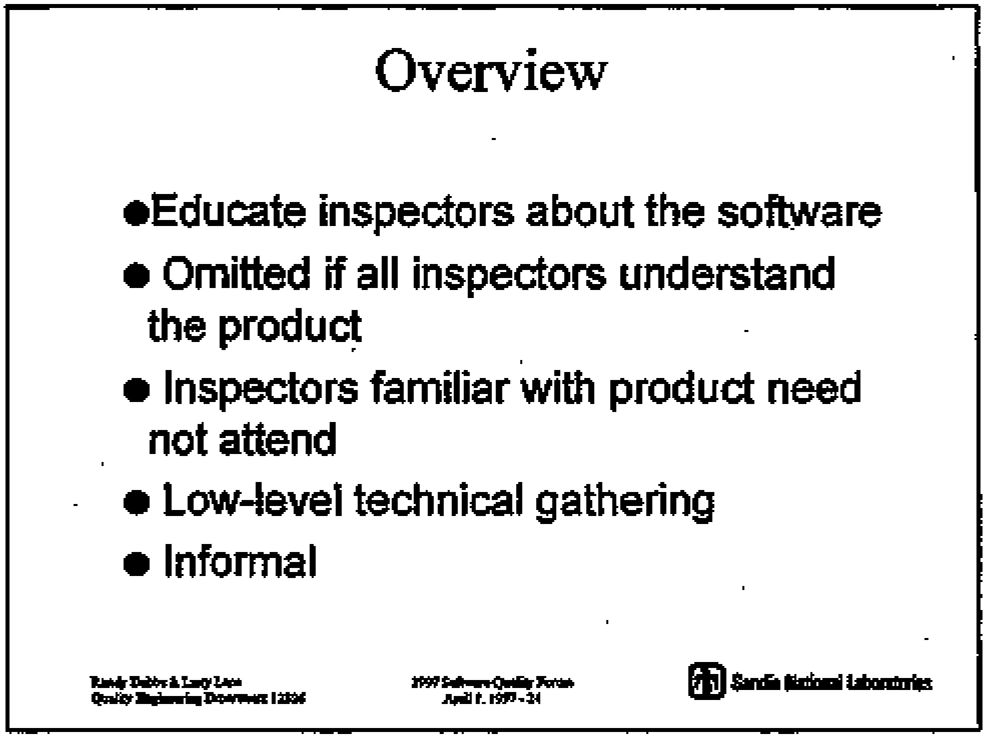
1997 Software Quality Forum
April 1, 1997 - 22

Sandia National Laboratories



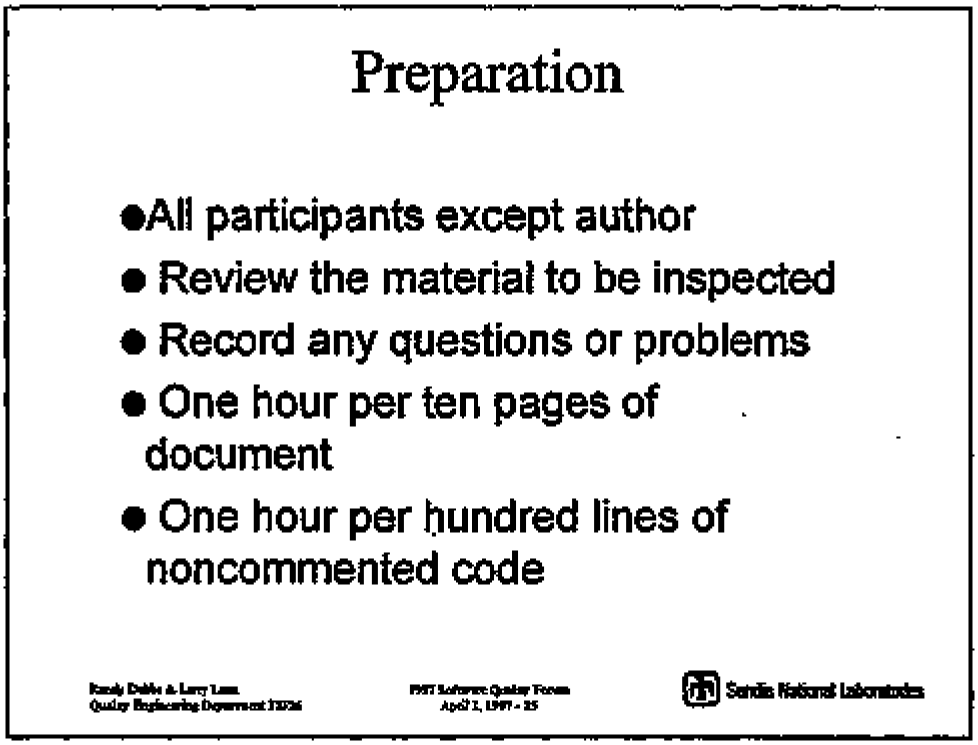
Planning

- Author and moderator participate
- Determine that the entry criteria have been met
- Prepare the inspection package
- Determine the number of meetings required
- Schedule the meetings
- Select the participants
- Determine if an overview meeting is required



Overview

- Educate inspectors about the software
- Omitted if all inspectors understand the product
- Inspectors familiar with product need not attend
- Low-level technical gathering
- Informal



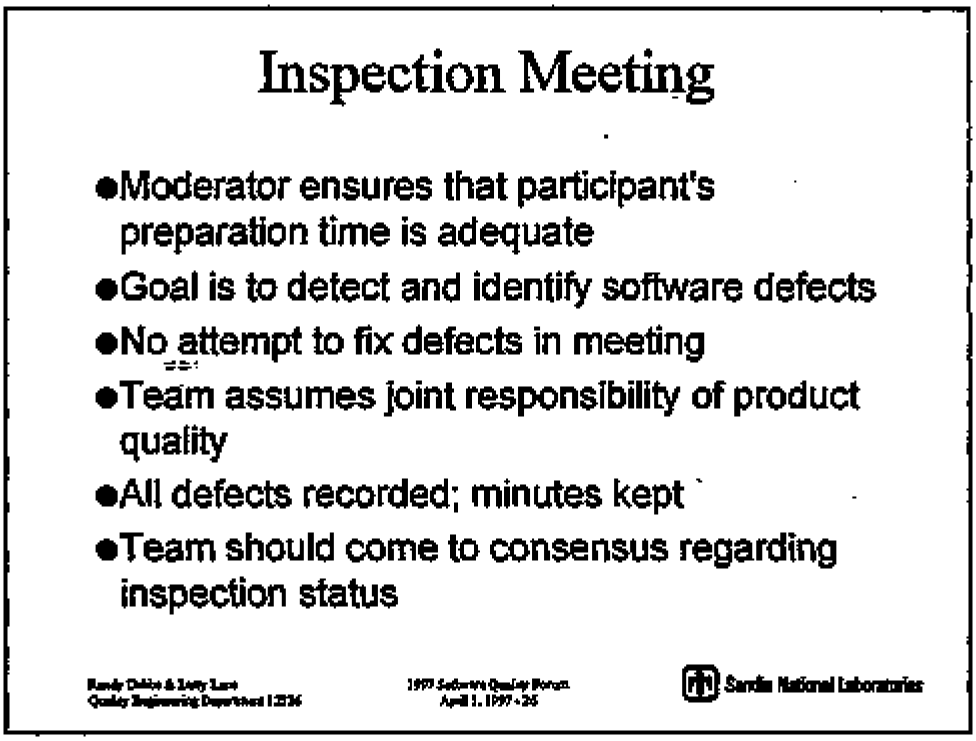
Preparation

- All participants except author
- Review the material to be inspected
- Record any questions or problems
- One hour per ten pages of document
- One hour per hundred lines of noncommented code

Randy Dabbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 25

Sandia National Laboratories



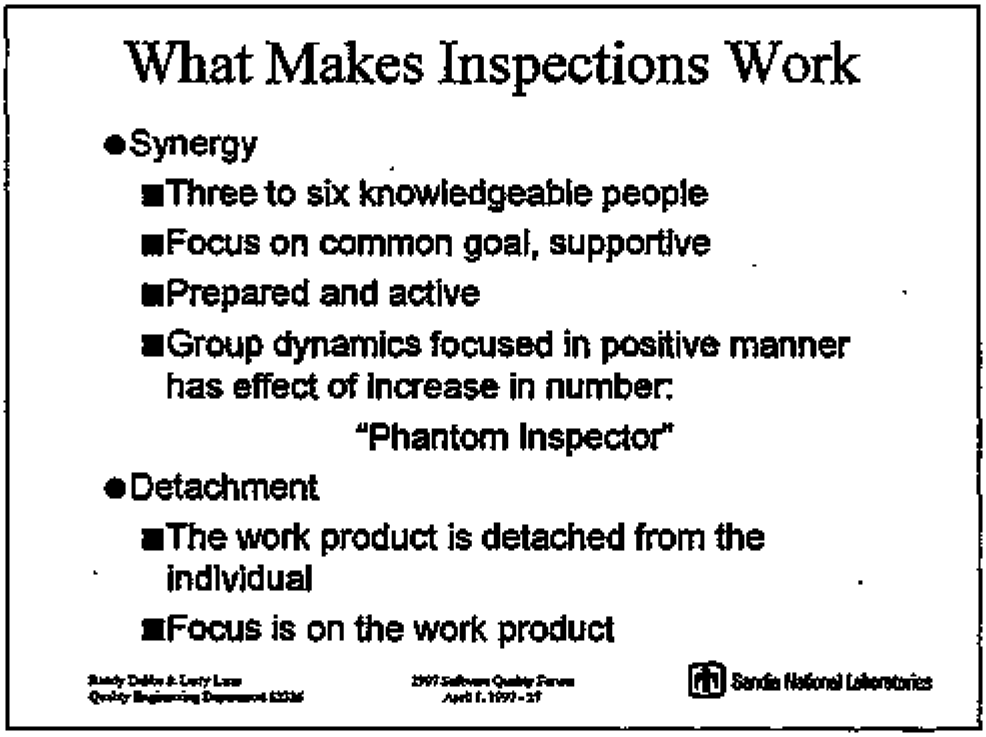
Inspection Meeting

- Moderator ensures that participant's preparation time is adequate
- Goal is to detect and identify software defects
- No attempt to fix defects in meeting
- Team assumes joint responsibility of product quality
- All defects recorded; minutes kept
- Team should come to consensus regarding inspection status

Randy Dabbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 25

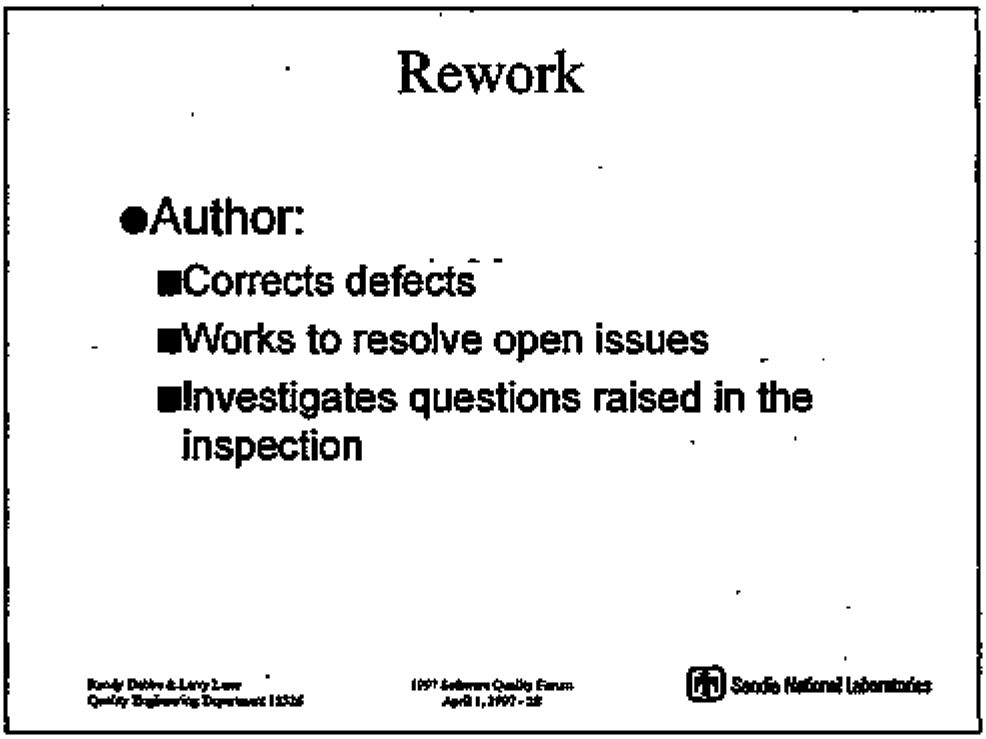
Sandia National Laboratories



What Makes Inspections Work

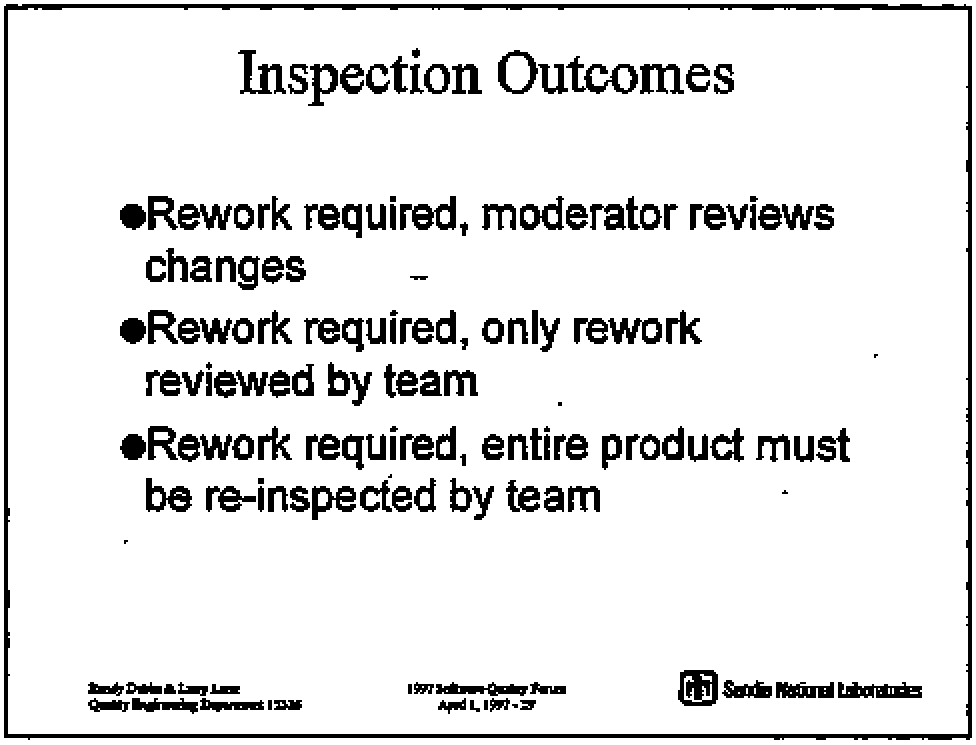
- Synergy
  - Three to six knowledgeable people
  - Focus on common goal, supportive
  - Prepared and active
  - Group dynamics focused in positive manner has effect of increase in number:

    "Phantom Inspector"
- Detachment
  - The work product is detached from the individual
  - Focus is on the work product



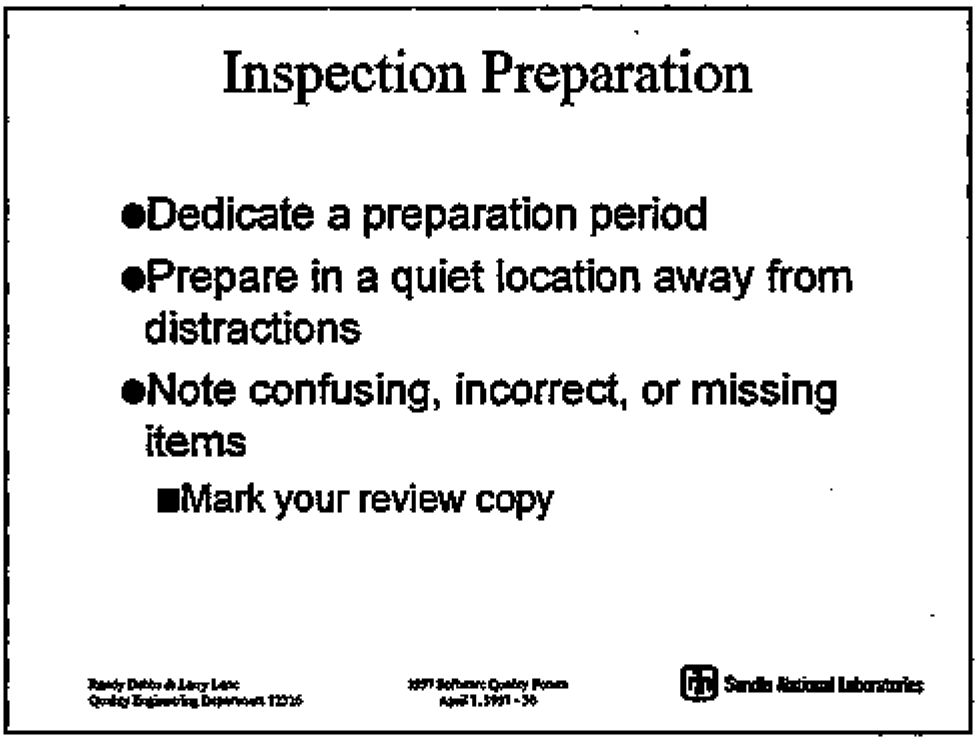
Rework

- Author:
  - Corrects defects
  - Works to resolve open issues
  - Investigates questions raised in the inspection



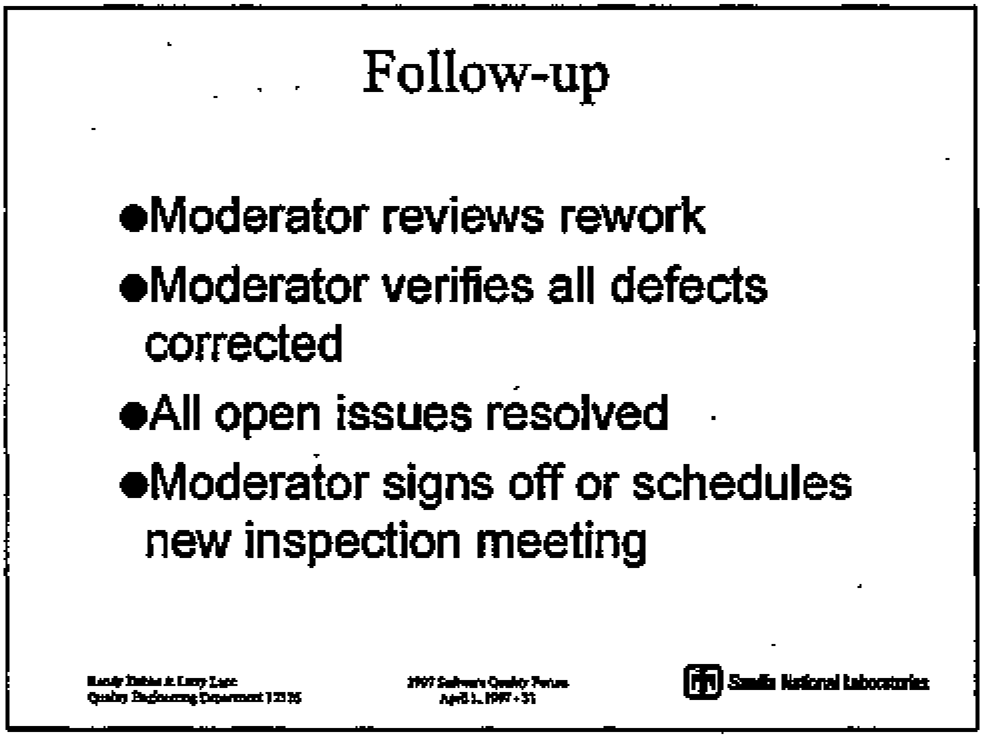
Inspection Outcomes

- Rework required, moderator reviews changes
- Rework required, only rework reviewed by team
- Rework required, entire product must be re-inspected by team



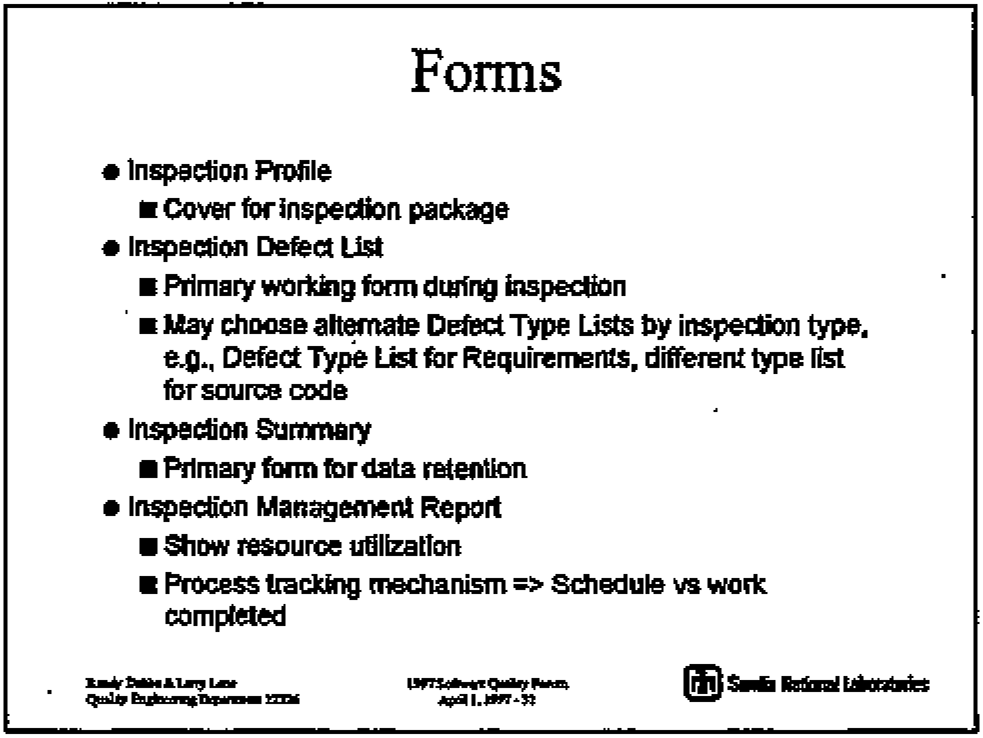
Inspection Preparation

- Dedicate a preparation period
- Prepare in a quiet location away from distractions
- Note confusing, incorrect, or missing items
  - Mark your review copy



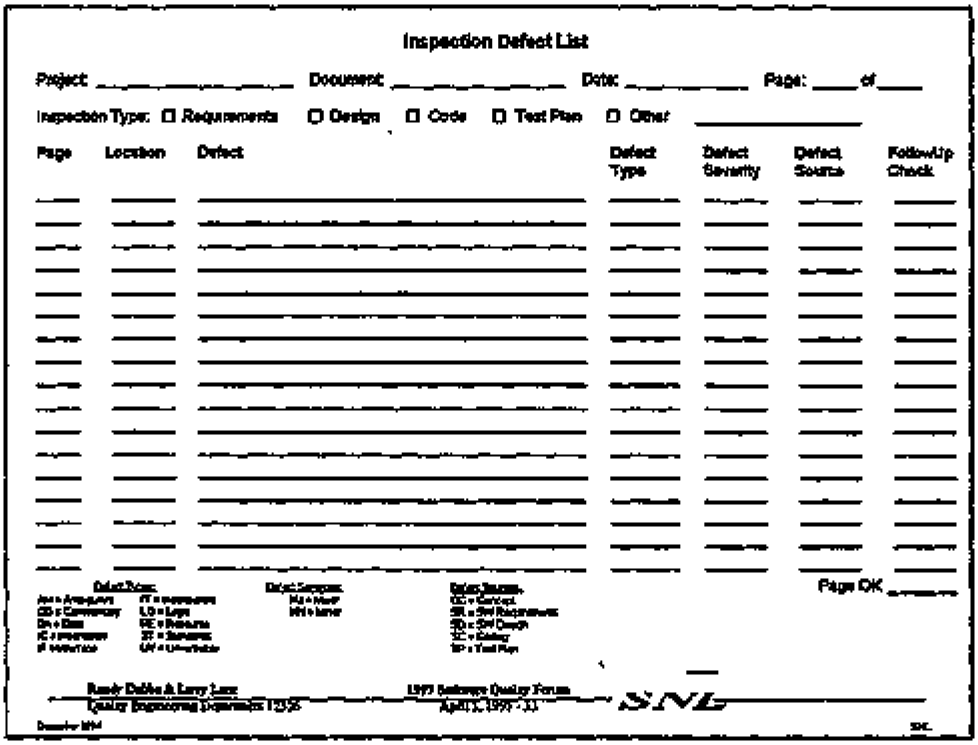
Follow-up

- Moderator reviews rework
- Moderator verifies all defects corrected
- All open issues resolved
- Moderator signs off or schedules new inspection meeting



Forms

- Inspection Profile
  - Cover for inspection package
- Inspection Defect List
  - Primary working form during inspection
  - May choose alternate Defect Type Lists by inspection type, e.g., Defect Type List for Requirements, different type list for source code
- Inspection Summary
  - Primary form for data retention
- Inspection Management Report
  - Show resource utilization
  - Process tracking mechanism => Schedule vs work completed



Inspection Defect List

Project: ____________ Document: ____________ Date: __________ Page: ____ of ____

Inspection Type: □ Requirements □ Design □ Code □ Test Plan □ Other ____________

Page | Location | Defect | Defect Type | Defect Severity | Defect Source | Follow-Up Check

Page OK ______

Defect Types:
AM = Ambiguous
CO = Consistency
DA = Data
IC = [illegible]
IF = Interface
IT = [illegible]
LO = Logic
PE = Performance
SY = Syntax
UV = Unverifiable

Defect Severity:
MA = Major
MI = Minor

Defect Sources:
CC = Concept
SR = S/W Requirements
SD = S/W Design
SC = Coding
TP = Test Plan

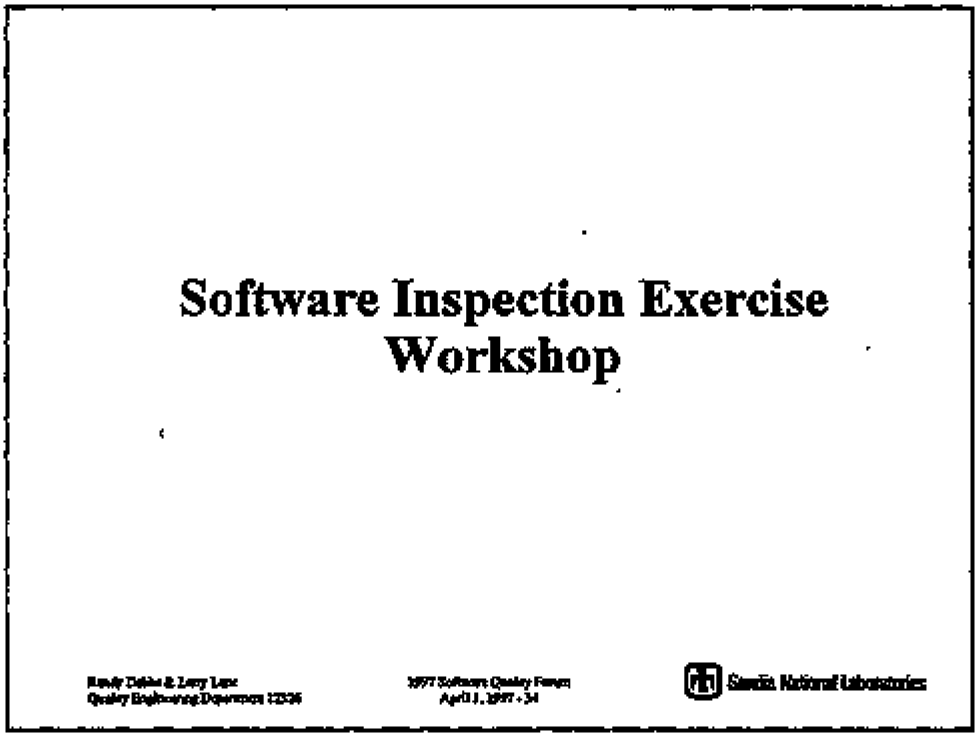
Randy Dabbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 3, 1997 - 33

SNL

December 1994

SNL

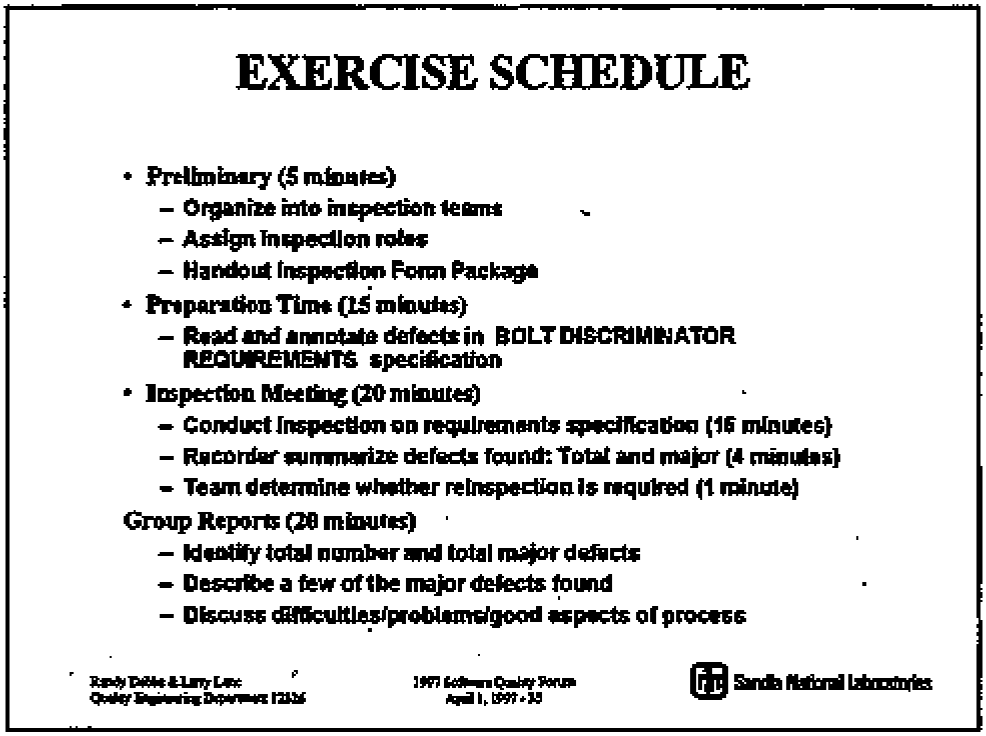


Software Inspection Exercise Workshop

Randy Dabbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 3, 1997 - 34

Sandia National Laboratories

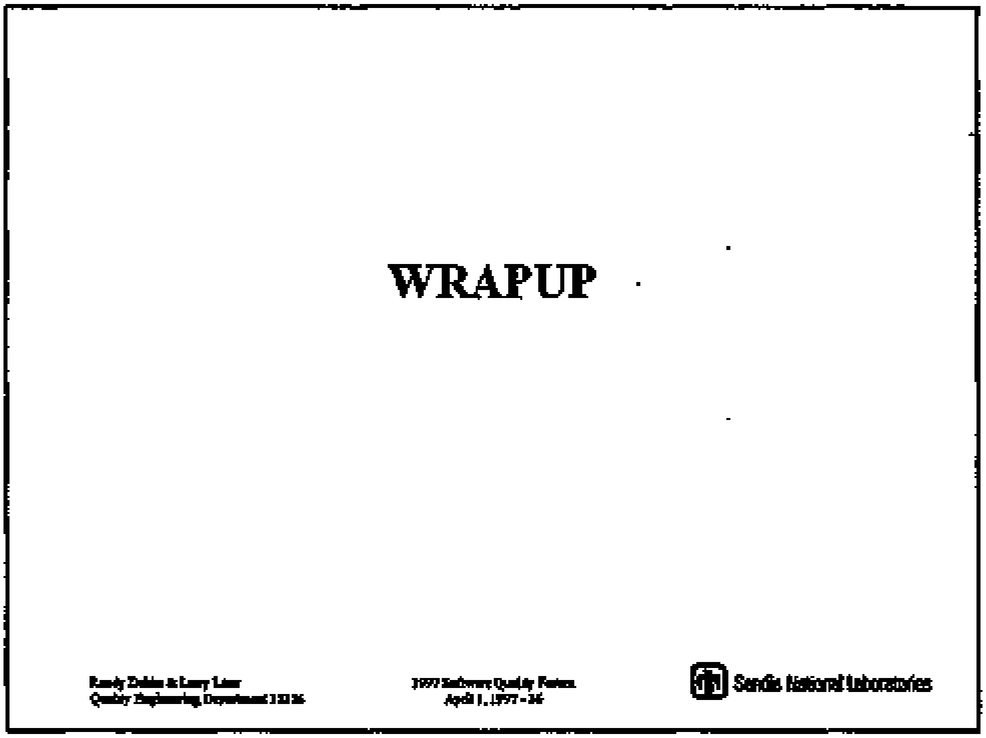


EXERCISE SCHEDULE

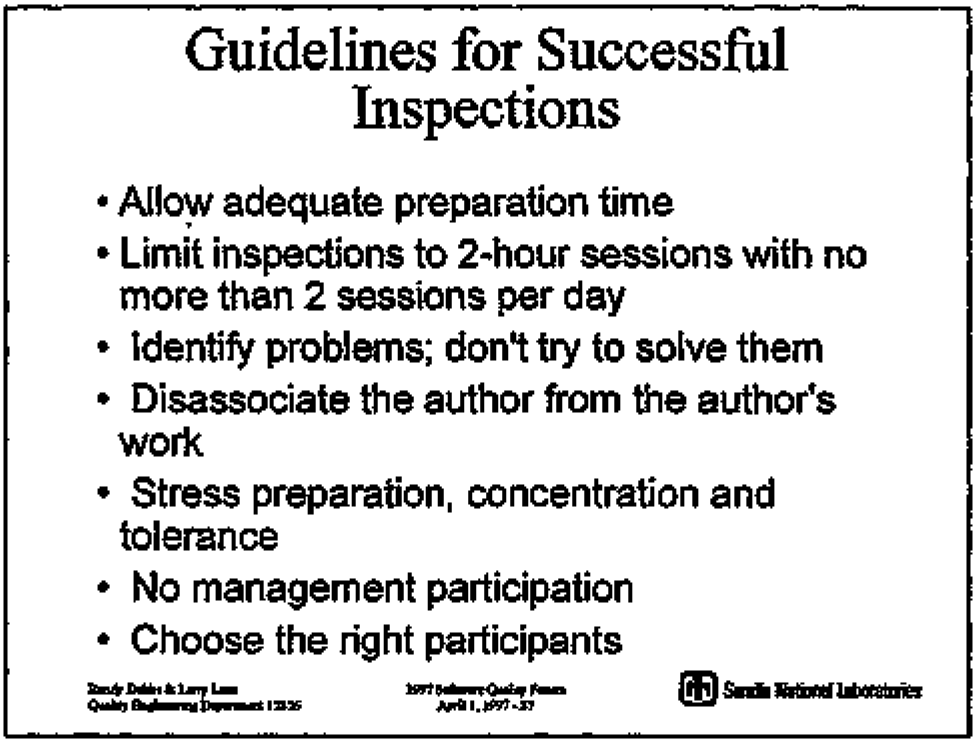
- Preliminary (5 minutes)
  - Organize into inspection teams
  - Assign inspection roles
  - Handout Inspection Form Package
- Preparation Time (15 minutes)
  - Read and annotate defects in BOLT DISCRIMINATOR REQUIREMENTS specification
- Inspection Meeting (20 minutes)
  - Conduct inspection on requirements specification (15 minutes)
  - Recorder summarize defects found: Total and major (4 minutes)
  - Team determine whether reinspection is required (1 minute)

Group Reports (20 minutes)

- Identify total number and total major defects
- Describe a few of the major defects found
- Discuss difficulties/problems/good aspects of process



WRAPUP



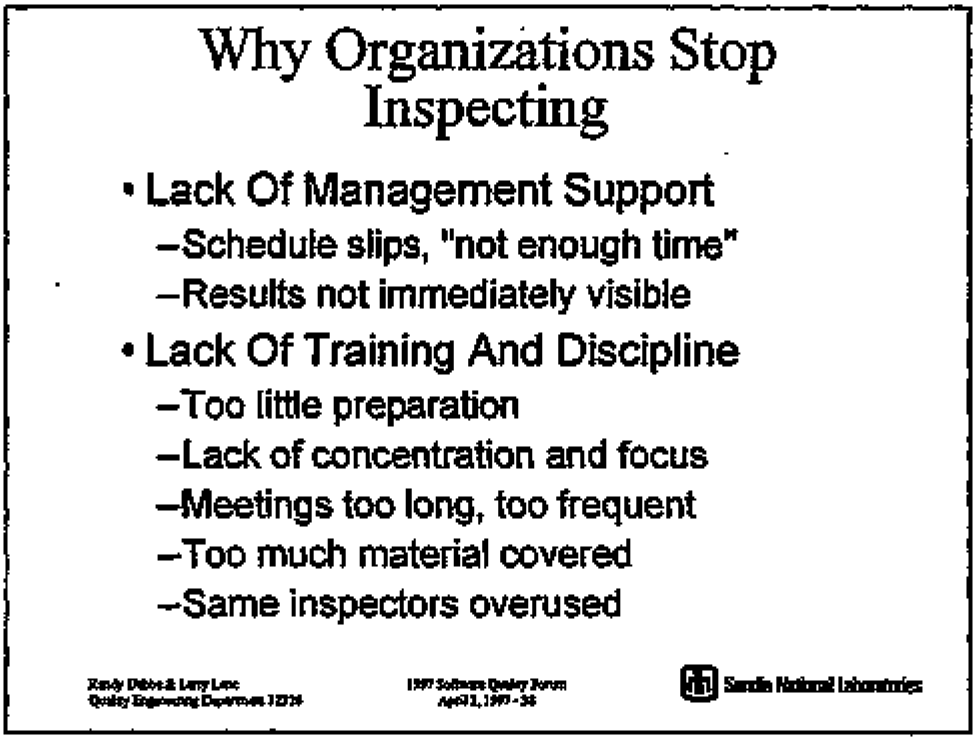
Guidelines for Successful Inspections

- Allow adequate preparation time
- Limit inspections to 2-hour sessions with no more than 2 sessions per day
- Identify problems; don't try to solve them
- Disassociate the author from the author's work
- Stress preparation, concentration and tolerance
- No management participation
- Choose the right participants

Randy Dobbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 37

Sandia National Laboratories



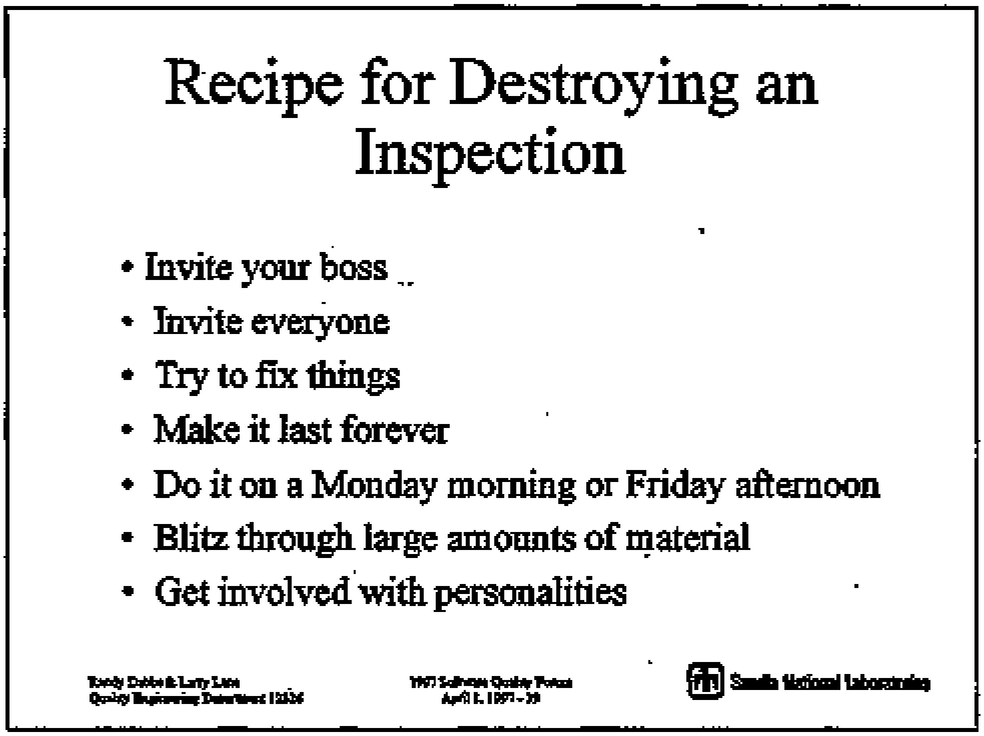
Why Organizations Stop Inspecting

- Lack Of Management Support
  - –Schedule slips, "not enough time"
  - –Results not immediately visible
- Lack Of Training And Discipline
  - –Too little preparation
  - –Lack of concentration and focus
  - –Meetings too long, too frequent
  - –Too much material covered
  - –Same inspectors overused

Randy Dobbs & Larry Lane
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 38

Sandia National Laboratories



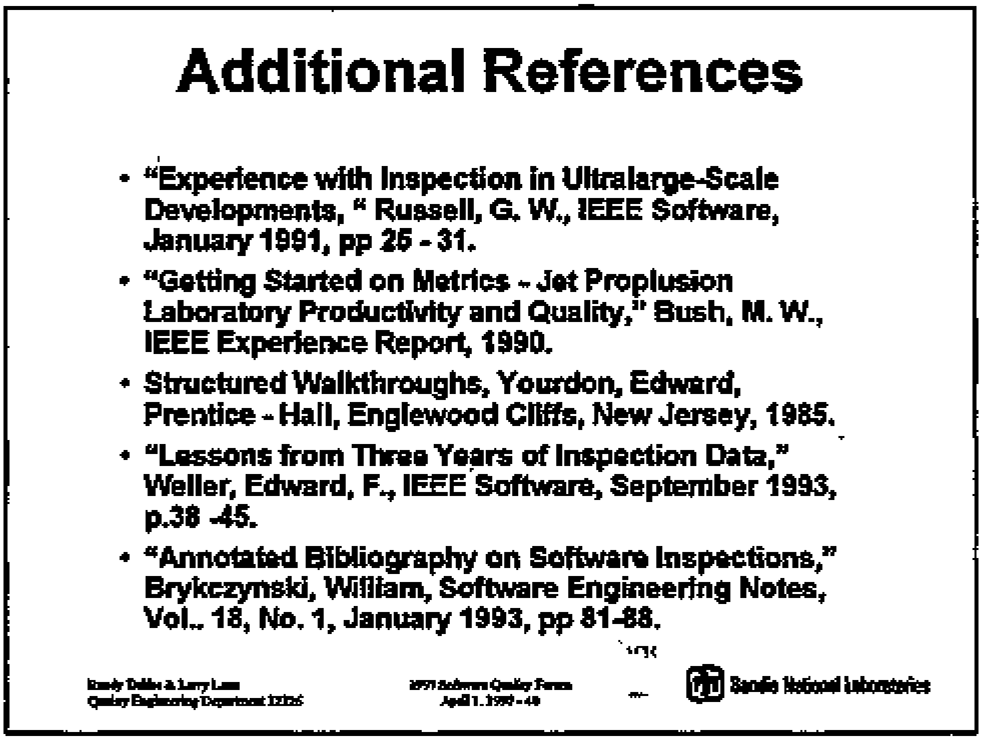
Recipe for Destroying an Inspection

- Invite your boss
- Invite everyone
- Try to fix things
- Make it last forever
- Do it on a Monday morning or Friday afternoon
- Blitz through large amounts of material
- Get involved with personalities

Randy Dobbs & Larry Lam
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 39

Sandia National Laboratories



Additional References

- "Experience with Inspection in Ultralarge-Scale Developments, " Russell, G. W., IEEE Software, January 1991, pp 25 - 31.
- "Getting Started on Metrics - Jet Proplusion Laboratory Productivity and Quality," Bush, M. W., IEEE Experience Report, 1990.
- Structured Walkthroughs, Yourdon, Edward, Prentice - Hall, Englewood Cliffs, New Jersey, 1985.
- "Lessons from Three Years of Inspection Data," Weller, Edward, F., IEEE Software, September 1993, p.38 -45.
- "Annotated Bibliography on Software Inspections," Brykczynski, William, Software Engineering Notes, Vol.. 18, No. 1, January 1993, pp 81-88.

Randy Dobbs & Larry Lam
Quality Engineering Department 12326

1997 Software Quality Forum
April 1, 1997 - 40

Sandia National Laboratories

1947 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Opening Session: Keynote Address

Capers Jones
Chairman, Software Productivity Research
Burlington, MA USA

Software Quality for 1997 - What Works and What Doesn't?

Capers Jones

Keynote Address:

Software Quality for 1997 - What Works and What Doesn't?

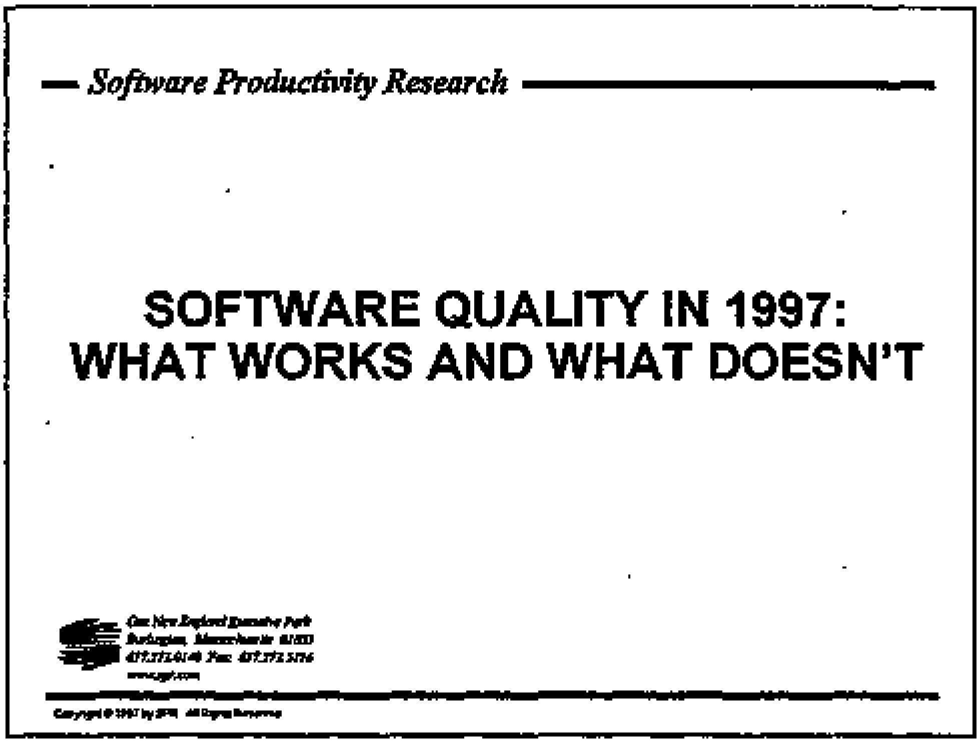
Capers Jones is an international consultant on software management topics and Chairman of Software Productivity Research, Inc. (SPR) in Burlington, MA. Following graduation from the University of Florida, Mr. Jones began his software career as a programmer in the office of the Surgeon General, Washington, D.C.. Prior to becoming Chairman at SPR, Mr. Jones also worked at the Crane Company, IBM and was Assistant Director of Programming Technology at ITT in Stratford CT. Mr. Jones has published nine books dealing with software areas, including; Programming Productivity, Software Measurement, Software Quality. His tenth book, Software Cost Estimating is scheduled for publication in early 1997. Mr. Jones will share his experience and insights in his keynote address "Software Quality for 1997 - What Works and What Doesn't".

Keynote Address: April 2 1997, 09:00 - 10:00 am, TTC Auditorium

This presentation provides a view of software quality for 1997 – what works and what doesn't. For many years, software quality assurance lagged behind hardware quality assurance in terms of methods, metrics, and successful results. New approaches such as Quality Function Deployment (QFD) the ISO 9000-9004 standards, the SEI maturity levels, and Total Quality Management (TQM) are starting to attract wide attention, and in some cases to bring software quality levels up to a parity with manufacturing quality levels. Since software is on the critical path for many engineered products, and for internal business systems as well, the new approaches are starting to affect global competition and attract widespread international interest. It can be hypothesized that success in mastering software quality will be a key strategy for dominating global software markets in the 21st century.

Capers Jones, Chairman
Software Productivity Research, Inc.
1 New England Executive Park
Burlington, MA 01803-5005

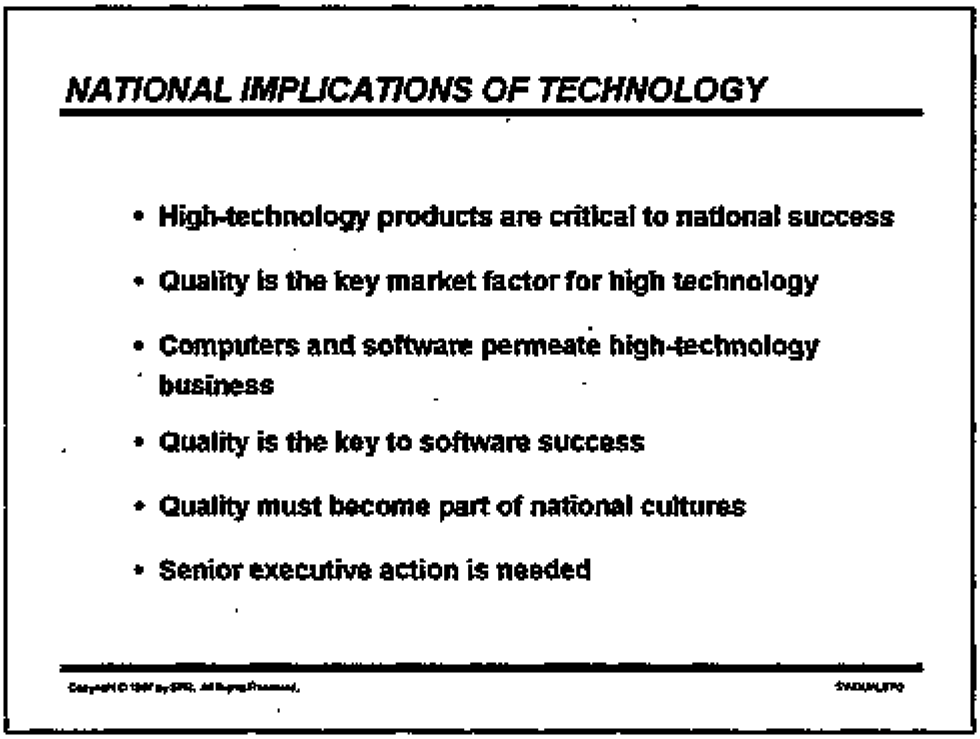
Phone 617 273 0140
FAX 617 273 5176
Email capers@spr.com



Software Productivity Research

SOFTWARE QUALITY IN 1997: WHAT WORKS AND WHAT DOESN'T

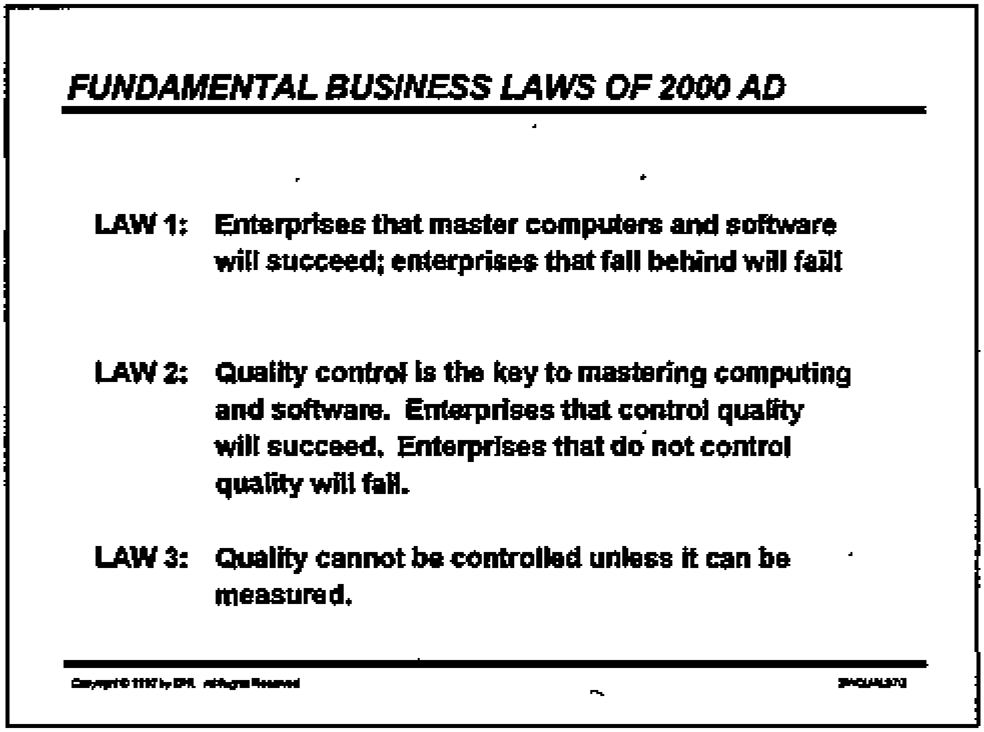
One New England Executive Park
Burlington, Massachusetts 01803
617.273.0140 Fax: 617.273.5176
www.spr.com



NATIONAL IMPLICATIONS OF TECHNOLOGY

- High-technology products are critical to national success
- Quality is the key market factor for high technology
- Computers and software permeate high-technology business
- Quality is the key to software success
- Quality must become part of national cultures
- Senior executive action is needed

[illegible]

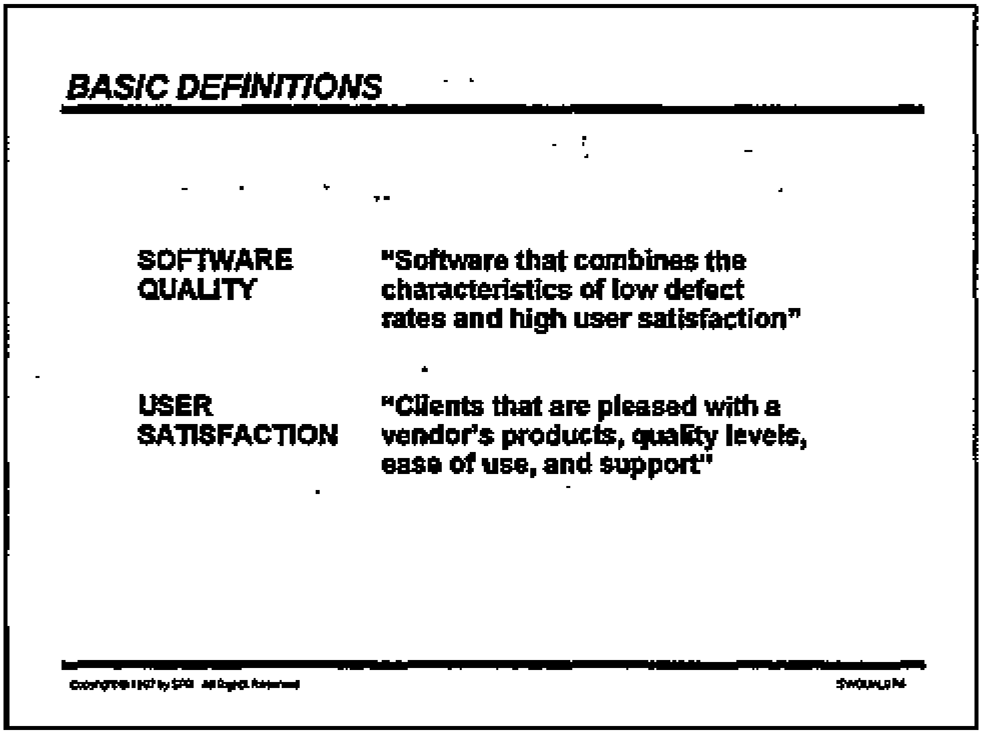


FUNDAMENTAL BUSINESS LAWS OF 2000 AD

LAW 1: Enterprises that master computers and software will succeed; enterprises that fall behind will fail!

LAW 2: Quality control is the key to mastering computing and software. Enterprises that control quality will succeed. Enterprises that do not control quality will fail.

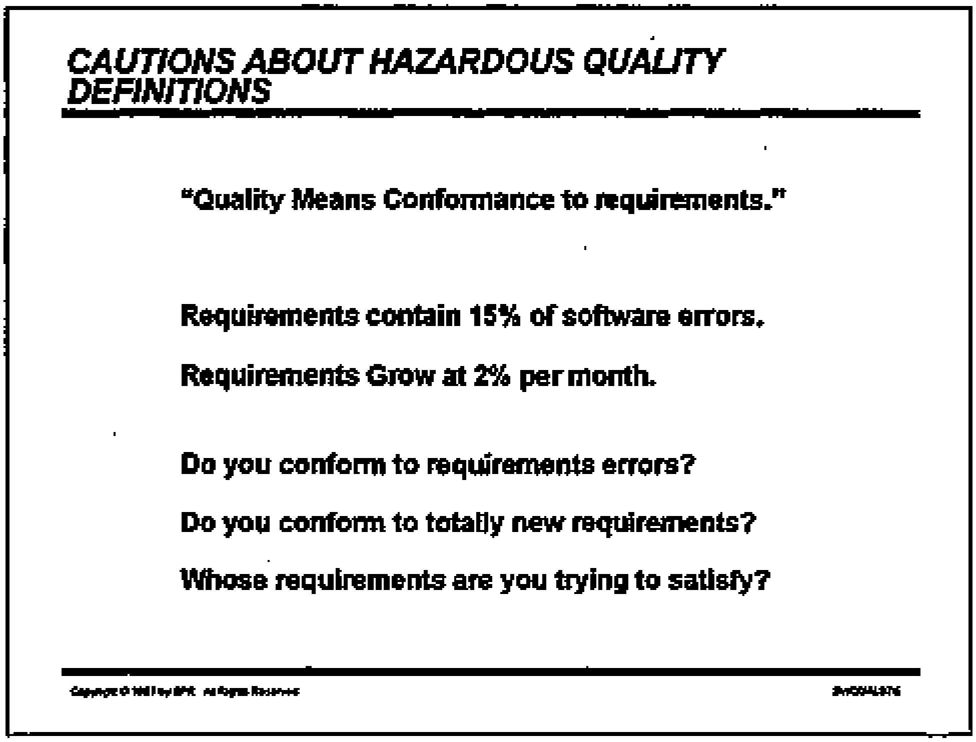
LAW 3: Quality cannot be controlled unless it can be measured.



BASIC DEFINITIONS

SOFTWARE QUALITY "Software that combines the characteristics of low defect rates and high user satisfaction"

USER SATISFACTION "Clients that are pleased with a vendor's products, quality levels, ease of use, and support"



CAUTIONS ABOUT HAZARDOUS QUALITY DEFINITIONS

"Quality Means Conformance to requirements."

Requirements contain 15% of software errors.

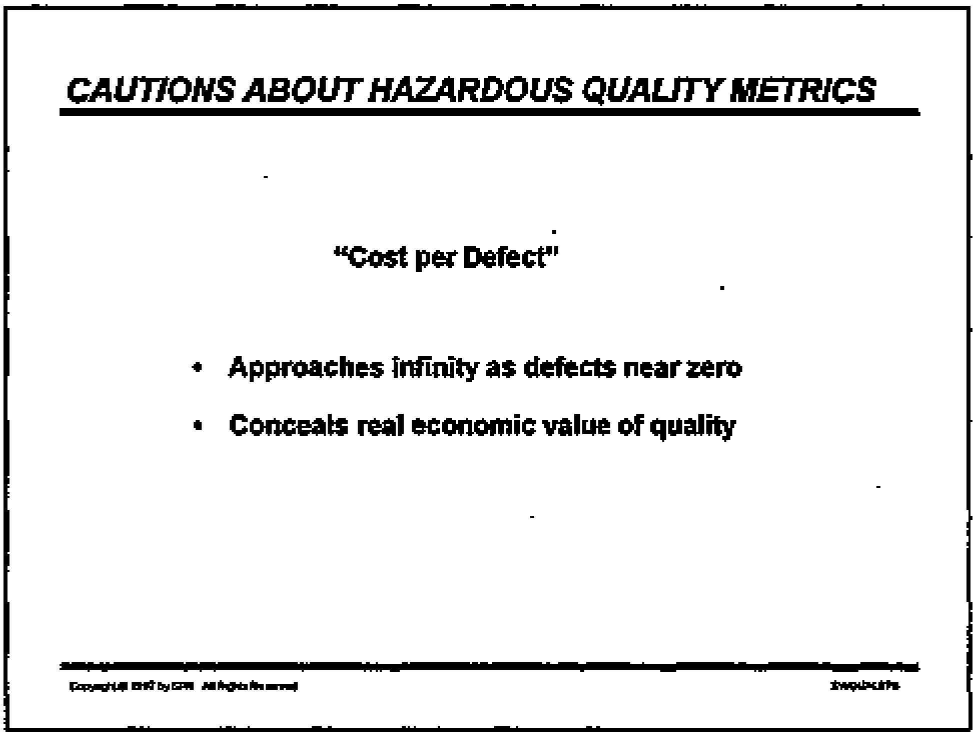
Requirements Grow at 2% per month.

Do you conform to requirements errors?

Do you conform to totally new requirements?

Whose requirements are you trying to satisfy?

[illegible]

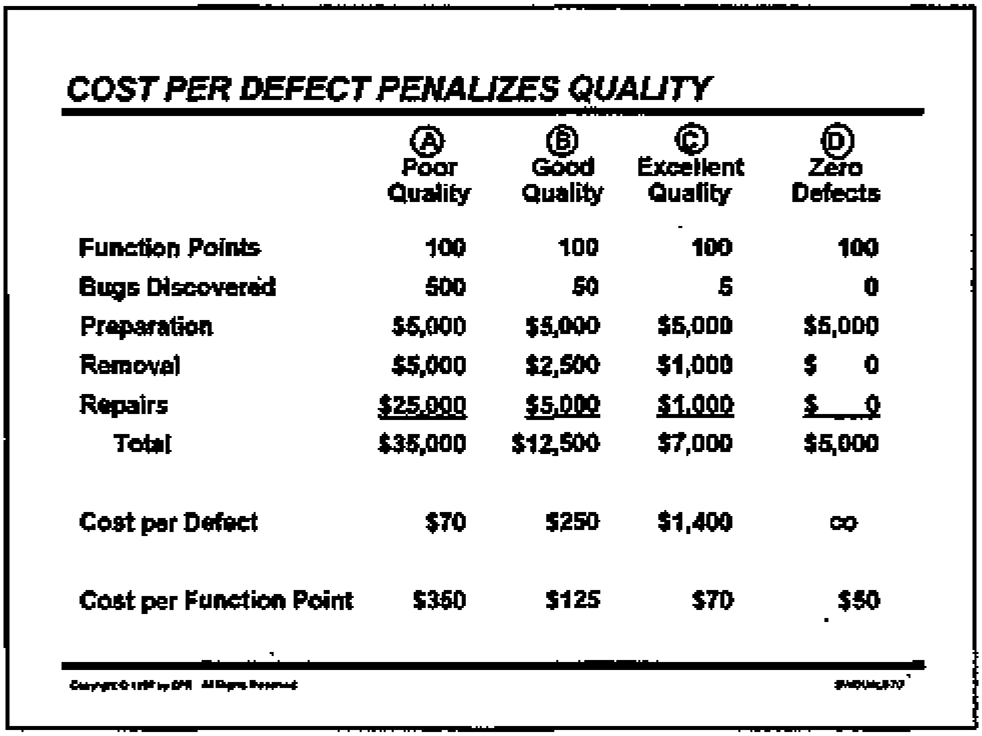


CAUTIONS ABOUT HAZARDOUS QUALITY METRICS

"Cost per Defect"

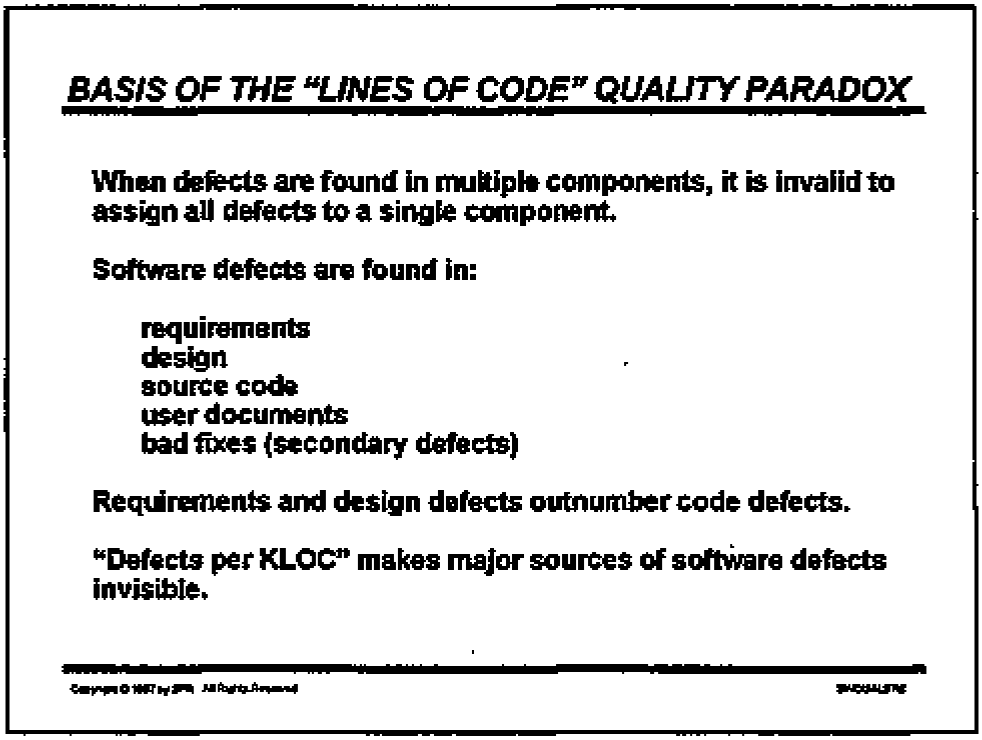
- Approaches infinity as defects near zero
- Conceals real economic value of quality

[illegible]



COST PER DEFECT PENALIZES QUALITY

	Ⓐ Poor Quality	Ⓑ Good Quality	Ⓒ Excellent Quality	Ⓓ Zero Defects
Function Points	100	100	100	100
Bugs Discovered	500	50	5	0
Preparation	$5,000	$5,000	$5,000	$5,000
Removal	$5,000	$2,500	$1,000	$ 0
Repairs	$25,000	$5,000	$1,000	$ 0
Total	$35,000	$12,500	$7,000	$5,000
Cost per Defect	$70	$250	$1,400	∞
Cost per Function Point	$350	$125	$70	$50



BASIS OF THE "LINES OF CODE" QUALITY PARADOX

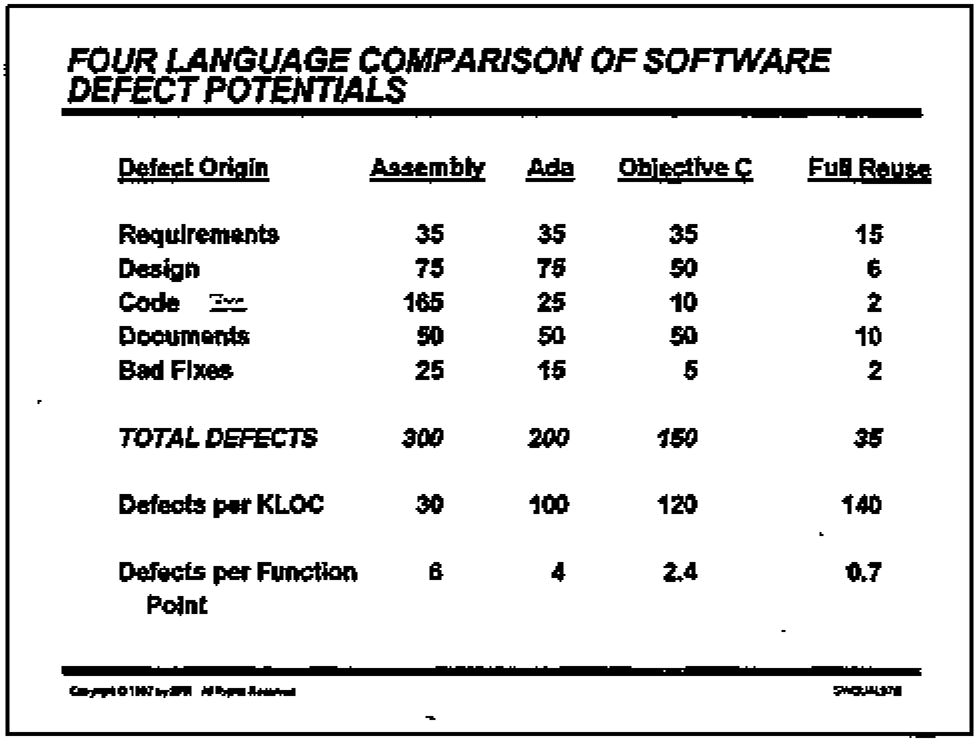
When defects are found in multiple components, it is invalid to assign all defects to a single component.

Software defects are found in:

- requirements
- design
- source code
- user documents
- bad fixes (secondary defects)

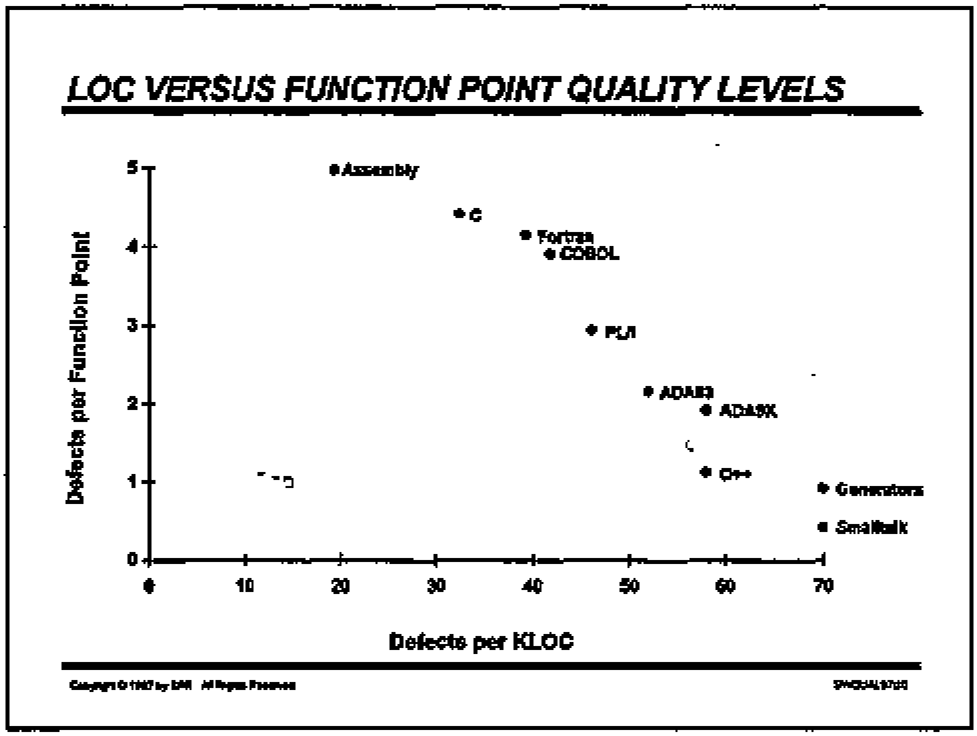
Requirements and design defects outnumber code defects.

"Defects per KLOC" makes major sources of software defects invisible.



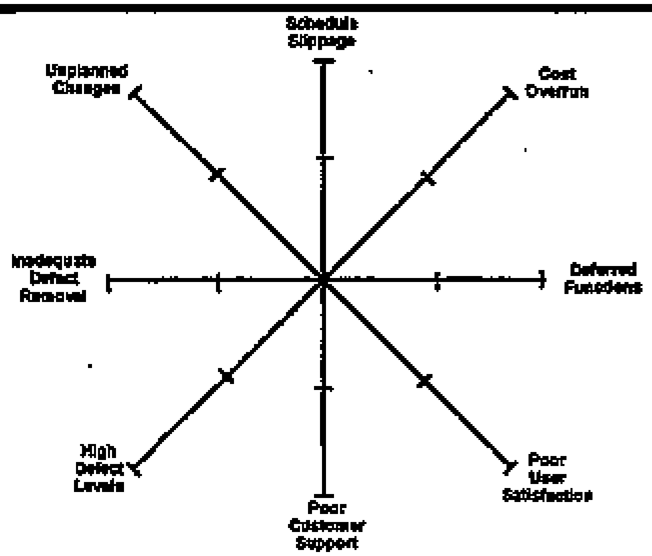
FOUR LANGUAGE COMPARISON OF SOFTWARE DEFECT POTENTIALS

Defect Origin	Assembly	Ada	Objective C	Full Reuse
Requirements	35	35	35	15
Design	75	75	50	6
Code	165	25	10	2
Documents	50	50	50	10
Bad Fixes	25	15	5	2
TOTAL DEFECTS	*300*	*200*	*150*	*35*
Defects per KLOC	30	100	120	140
Defects per Function Point	6	4	2.4	0.7



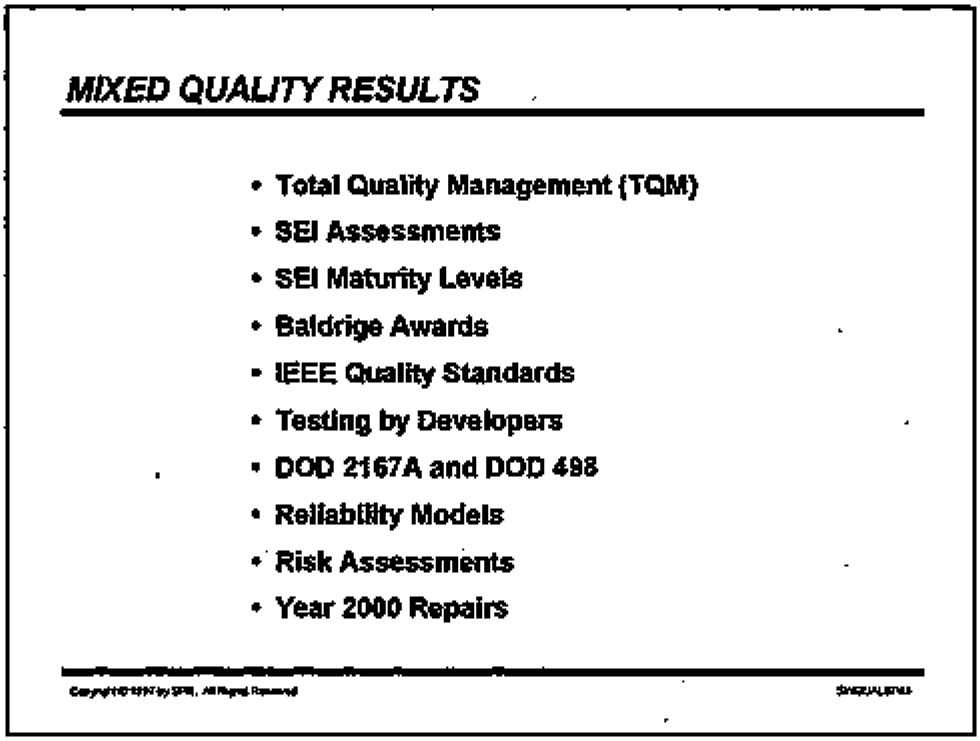
LOC VERSUS FUNCTION POINT QUALITY LEVELS

KIVIAT GRAPH OF MAJOR SOFTWARE RISKS



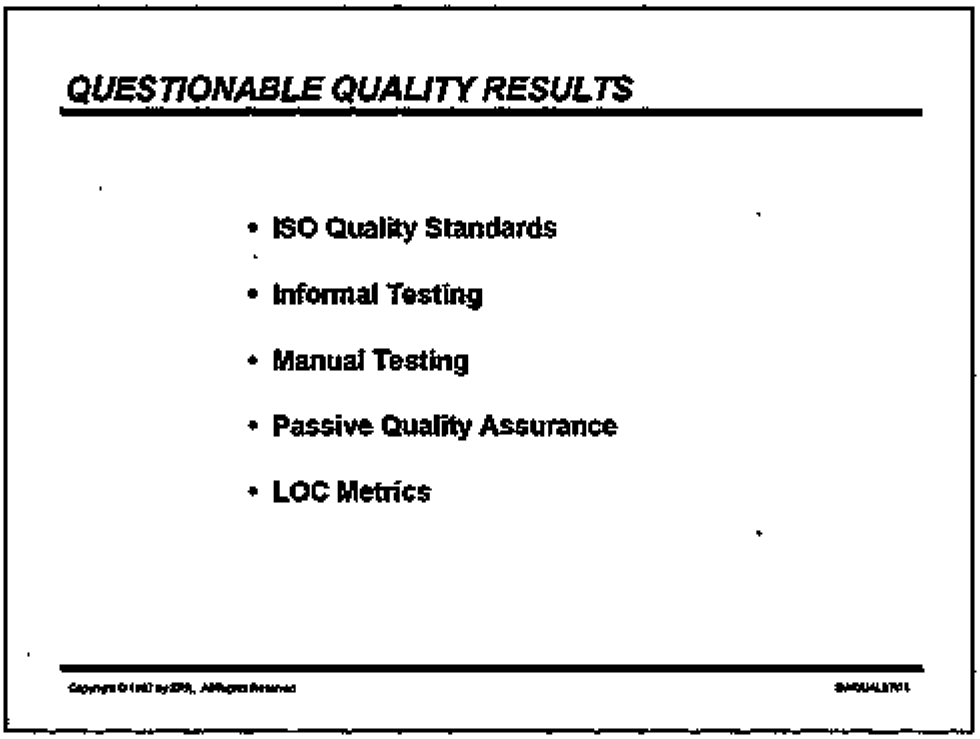
CONSISTENTLY GOOD QUALITY RESULTS

- Formal Inspections (Design and Code)
- Joint Application Design (JAD)
- Quality Function Deployment (QFD)
- Quality Metrics
- Removal Efficiency Measurements
- Functional Metrics
- Active Quality Assurance
- Formal Configuration Control
- User Satisfaction Surveys
- Formal Test Planning
- Quality Estimation Tools
- Automated Test Tools
- Testing Specialists



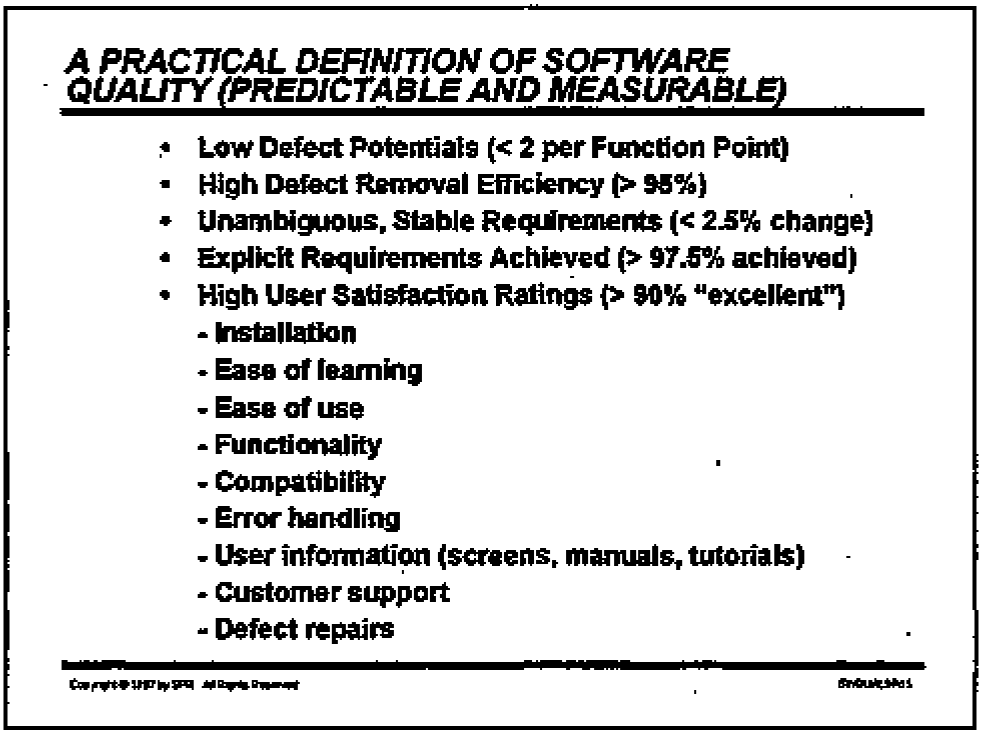
MIXED QUALITY RESULTS

- Total Quality Management (TQM)
- SEI Assessments
- SEI Maturity Levels
- Baldrige Awards
- IEEE Quality Standards
- Testing by Developers
- DOD 2167A and DOD 498
- Reliability Models
- Risk Assessments
- Year 2000 Repairs



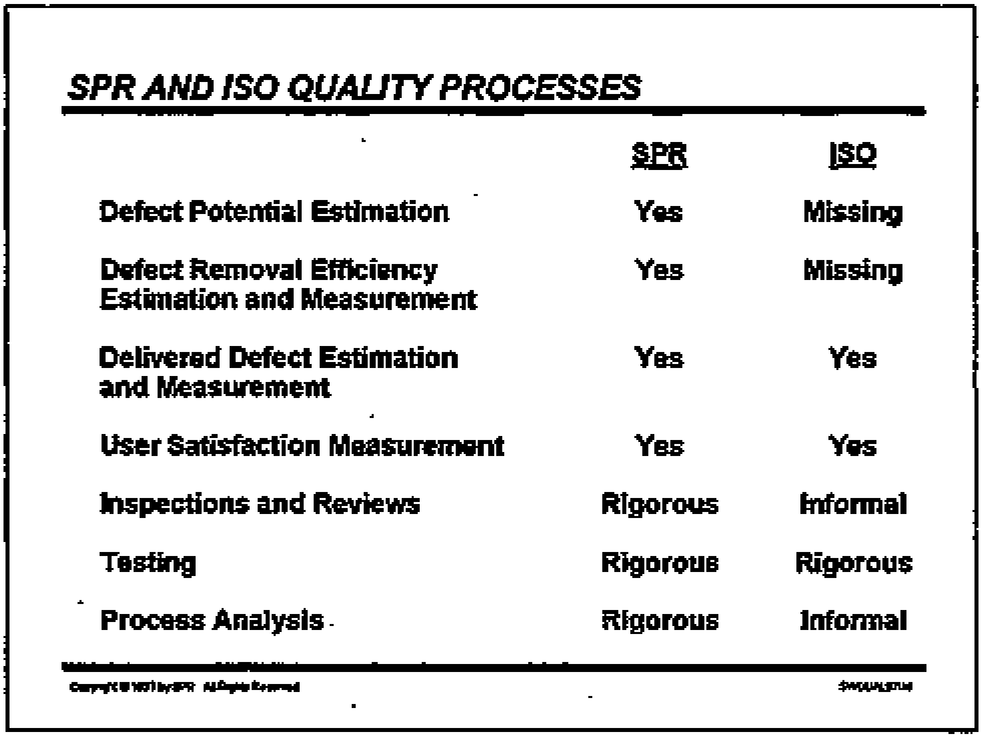
QUESTIONABLE QUALITY RESULTS

- ISO Quality Standards
- Informal Testing
- Manual Testing
- Passive Quality Assurance
- LOC Metrics



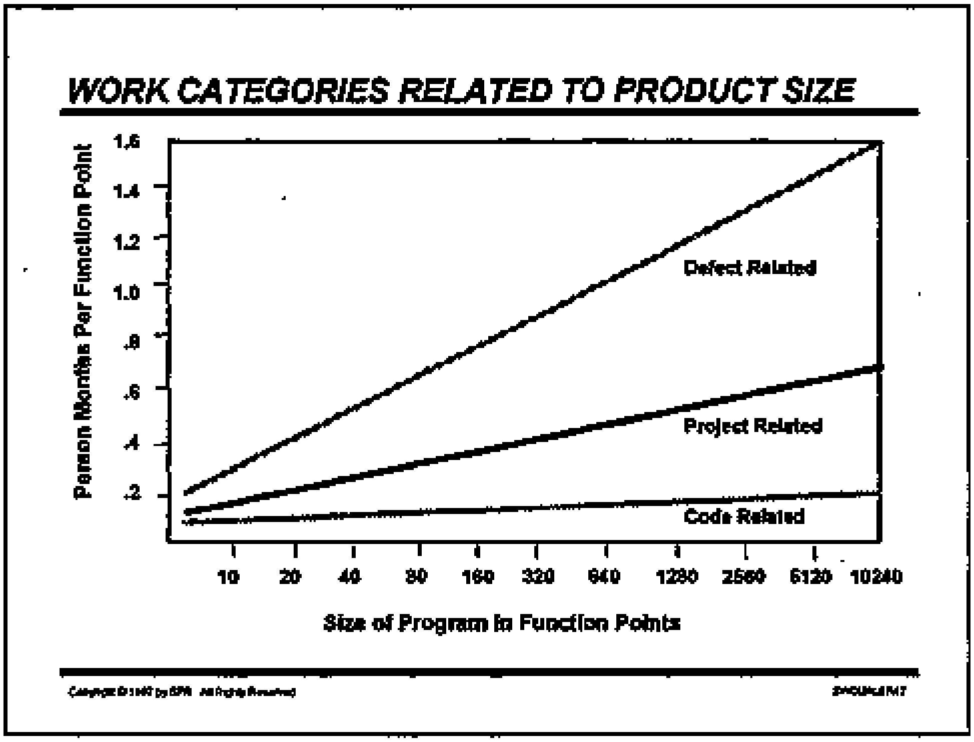
A PRACTICAL DEFINITION OF SOFTWARE QUALITY (PREDICTABLE AND MEASURABLE)

- Low Defect Potentials (< 2 per Function Point)
- High Defect Removal Efficiency (> 95%)
- Unambiguous, Stable Requirements (< 2.5% change)
- Explicit Requirements Achieved (> 97.5% achieved)
- High User Satisfaction Ratings (> 90% "excellent")
  - Installation
  - Ease of learning
  - Ease of use
  - Functionality
  - Compatibility
  - Error handling
  - User information (screens, manuals, tutorials)
  - Customer support
  - Defect repairs

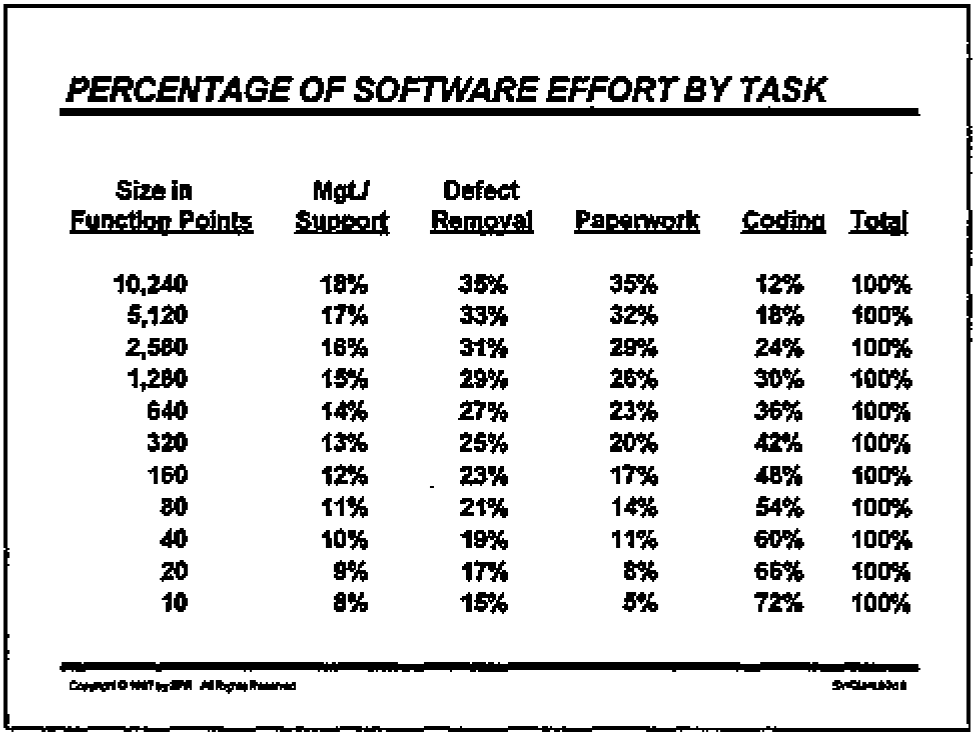


SPR AND ISO QUALITY PROCESSES

	SPR	ISO
Defect Potential Estimation	Yes	Missing
Defect Removal Efficiency Estimation and Measurement	Yes	Missing
Delivered Defect Estimation and Measurement	Yes	Yes
User Satisfaction Measurement	Yes	Yes
Inspections and Reviews	Rigorous	Informal
Testing	Rigorous	Rigorous
Process Analysis	Rigorous	Informal

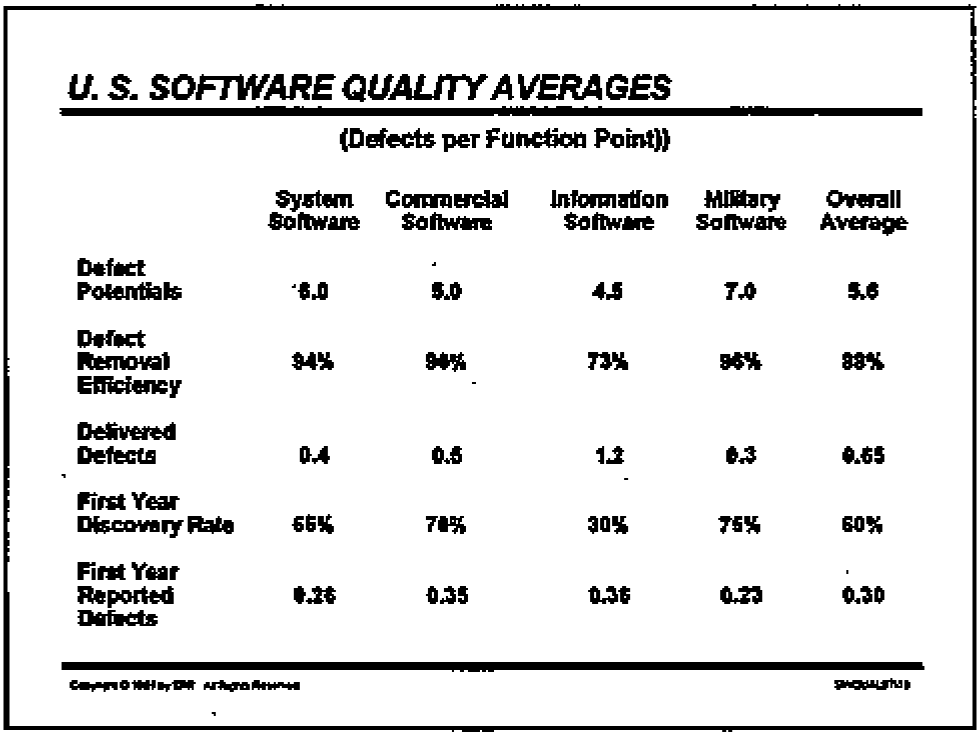


WORK CATEGORIES RELATED TO PRODUCT SIZE



PERCENTAGE OF SOFTWARE EFFORT BY TASK

Size in Function Points	Mgt./ Support	Defect Removal	Paperwork	Coding	Total
10,240	18%	35%	35%	12%	100%
5,120	17%	33%	32%	18%	100%
2,560	16%	31%	29%	24%	100%
1,280	15%	29%	26%	30%	100%
640	14%	27%	23%	36%	100%
320	13%	25%	20%	42%	100%
160	12%	23%	17%	48%	100%
80	11%	21%	14%	54%	100%
40	10%	19%	11%	60%	100%
20	9%	17%	8%	66%	100%
10	8%	15%	5%	72%	100%

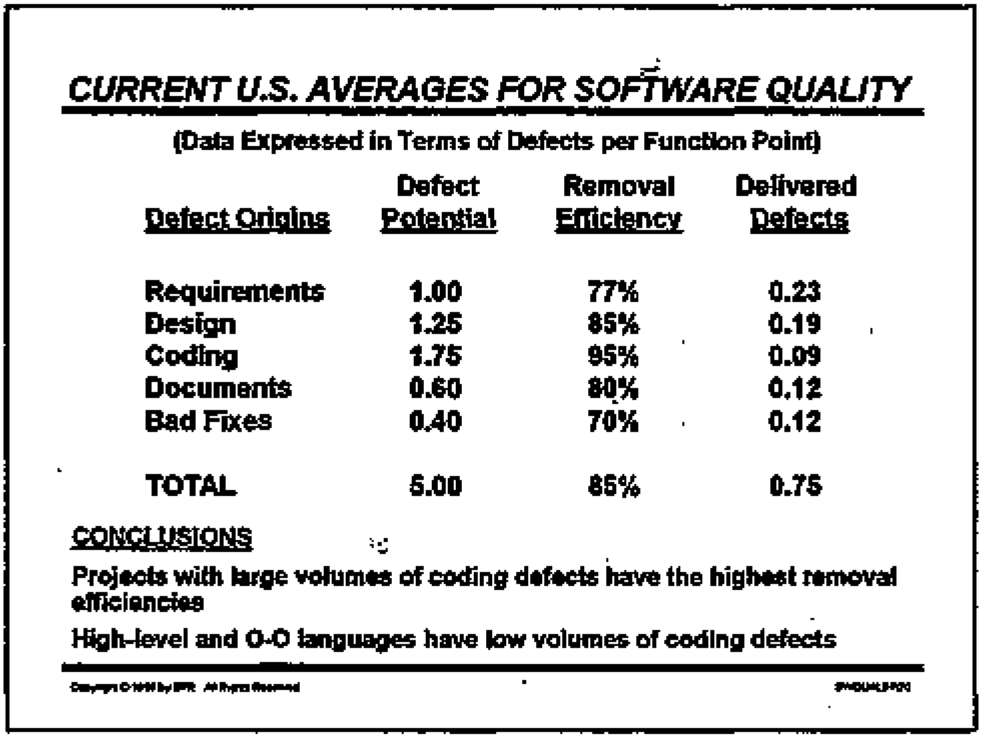


U. S. SOFTWARE QUALITY AVERAGES

(Defects per Function Point))

	System Software	Commercial Software	Information Software	Military Software	Overall Average
Defect Potentials	6.0	5.0	4.5	7.0	5.6
Defect Removal Efficiency	94%	90%	73%	96%	88%
Delivered Defects	0.4	0.5	1.2	0.3	0.65
First Year Discovery Rate	65%	70%	30%	75%	60%
First Year Reported Defects	0.26	0.35	0.36	0.23	0.30

[illegible]



CURRENT U.S. AVERAGES FOR SOFTWARE QUALITY

(Data Expressed in Terms of Defects per Function Point)

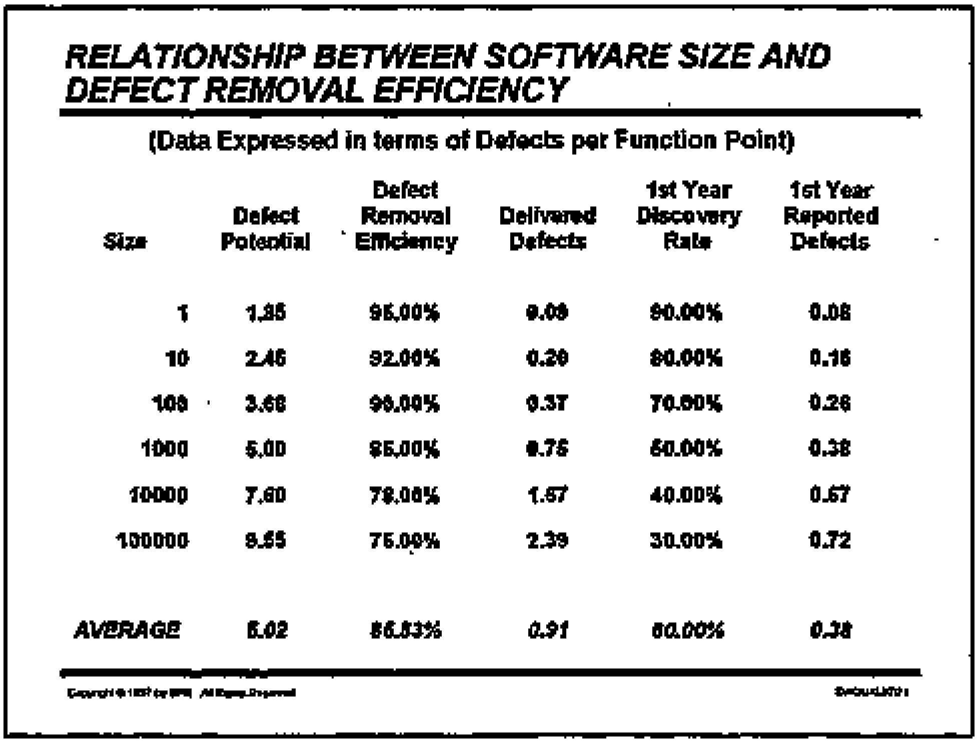
Defect Origins	Defect Potential	Removal Efficiency	Delivered Defects
Requirements	1.00	77%	0.23
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Documents	0.60	80%	0.12
Bad Fixes	0.40	70%	0.12
TOTAL	5.00	85%	0.75

CONCLUSIONS

Projects with large volumes of coding defects have the highest removal efficiencies

High-level and O-O languages have low volumes of coding defects

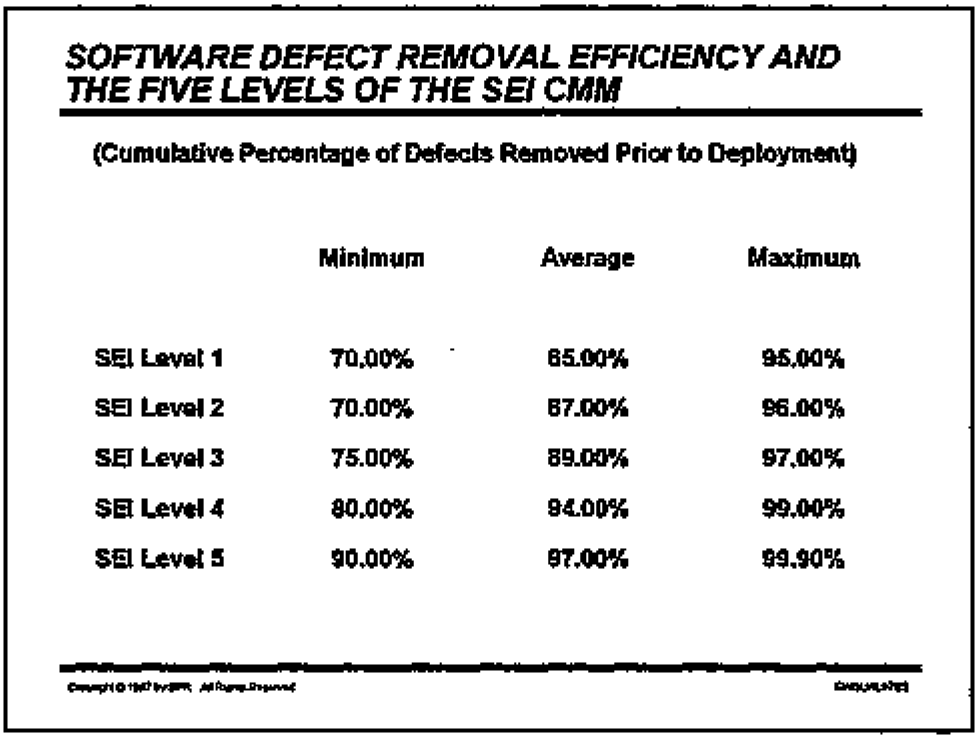
[illegible]



RELATIONSHIP BETWEEN SOFTWARE SIZE AND DEFECT REMOVAL EFFICIENCY

(Data Expressed in terms of Defects per Function Point)

Size	Defect Potential	Defect Removal Efficiency	Delivered Defects	1st Year Discovery Rate	1st Year Reported Defects
1	1.85	95.00%	0.09	90.00%	0.08
10	2.45	92.00%	0.20	80.00%	0.16
100	3.68	90.00%	0.37	70.00%	0.26
1000	5.00	85.00%	0.75	50.00%	0.38
10000	7.60	78.00%	1.67	40.00%	0.67
100000	9.55	75.00%	2.39	30.00%	0.72
AVERAGE	*5.02*	*85.83%*	*0.91*	*60.00%*	*0.38*



SOFTWARE DEFECT REMOVAL EFFICIENCY AND THE FIVE LEVELS OF THE SEI CMM

(Cumulative Percentage of Defects Removed Prior to Deployment)

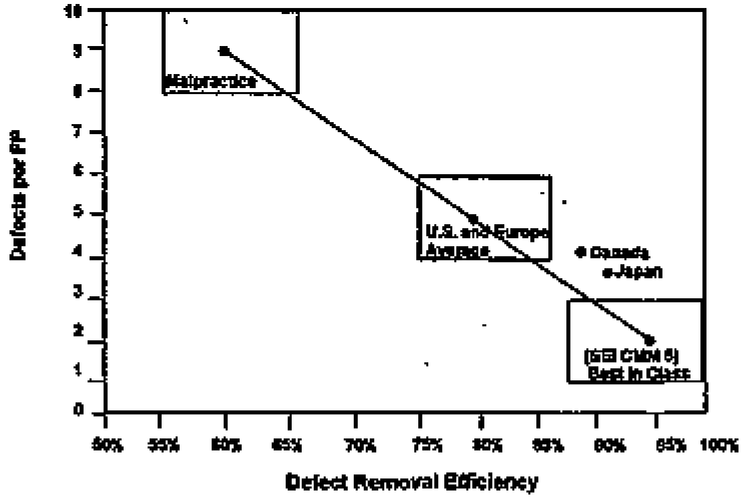
	Minimum	Average	Maximum
SEI Level 1	70.00%	85.00%	95.00%
SEI Level 2	70.00%	87.00%	96.00%
SEI Level 3	75.00%	89.00%	97.00%
SEI Level 4	80.00%	94.00%	99.00%
SEI Level 5	90.00%	97.00%	99.90%

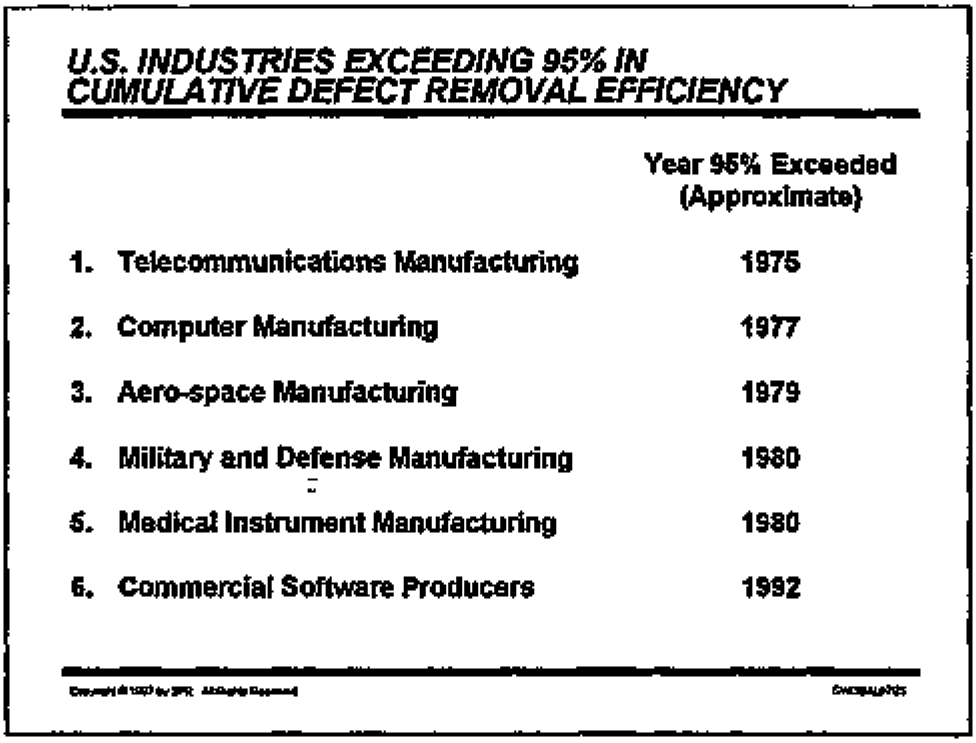
SOFTWARE DEFECT POTENTIALS & DEFECT REMOVAL EFFICIENCY SUGGESTED FOR EACH LEVEL OF SEI CMM

(Data Expressed in Terms of Defects per Function Point)

SEI CMM Levels	Defect Potentials	Removal Efficiency	Delivered Defects
SEI CMM 1	5.00	85%	0.75
SEI CMM 2	4.00	90%	0.40
SEI CMM 3	3.00	95%	0.15
SEI CMM 4	2.00	97%	0.06
SEI CMM 5	1.00	99%	0.01

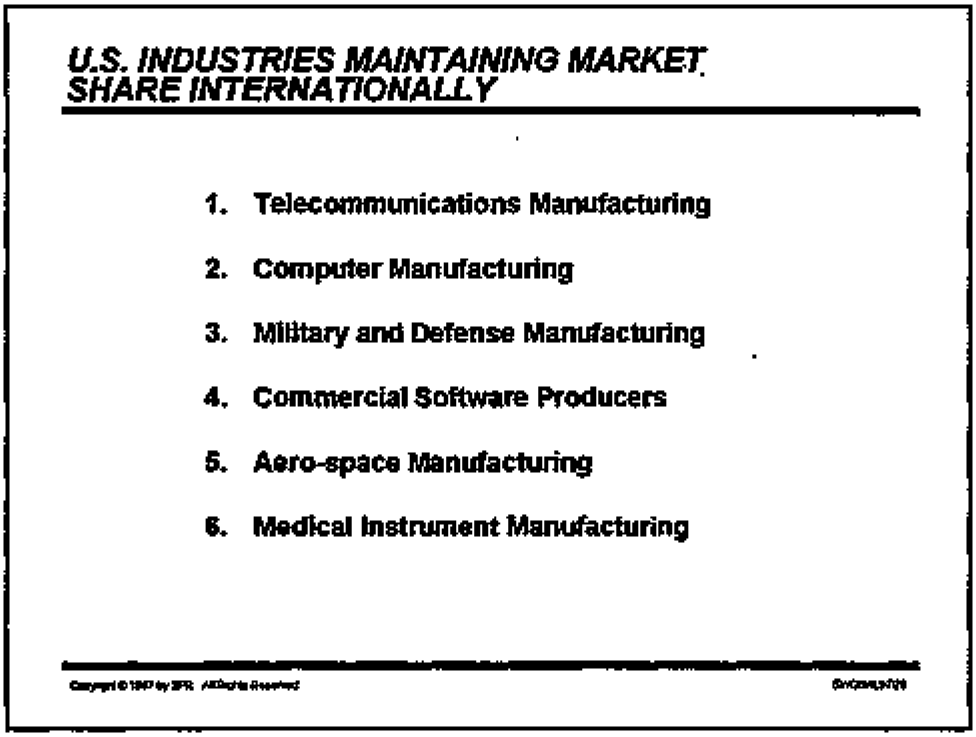
SOFTWARE QUALITY IMPROVEMENT





U.S. INDUSTRIES EXCEEDING 95% IN CUMULATIVE DEFECT REMOVAL EFFICIENCY

	Year 95% Exceeded (Approximate)
1. Telecommunications Manufacturing	1975
2. Computer Manufacturing	1977
3. Aero-space Manufacturing	1979
4. Military and Defense Manufacturing	1980
5. Medical Instrument Manufacturing	1980
6. Commercial Software Producers	1992



U.S. INDUSTRIES MAINTAINING MARKET SHARE INTERNATIONALLY

1. Telecommunications Manufacturing
2. Computer Manufacturing
3. Military and Defense Manufacturing
4. Commercial Software Producers
5. Aero-space Manufacturing
6. Medical Instrument Manufacturing

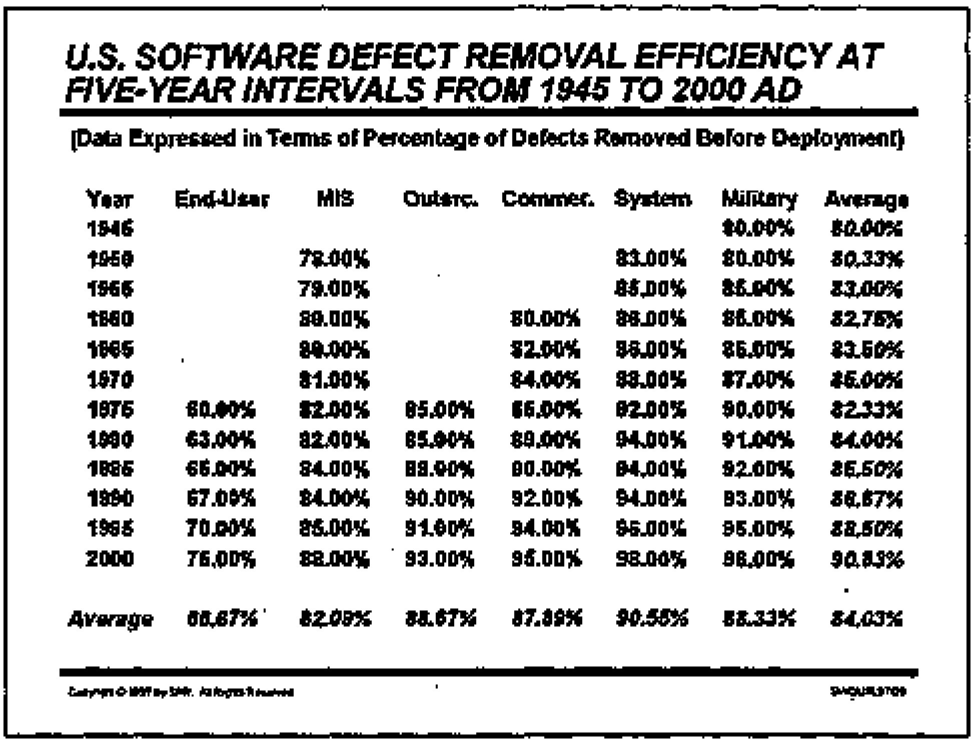
DEFECT REMOVAL AND TESTING STAGES NOTED DURING LITIGATION FOR POOR QUALITY

	Reliable Software	Software Involved in Litigation for Poor Quality
Formal design inspections	Used	Not used
Formal code inspections	Used	Not used
Subroutine testing	Used	Used
Unit testing	Used	Used
New function testing	Used	Rushed or omitted
Regression testing	Used	Rushed or omitted
Integration testing	Used	Used
System testing	Used	Rushed or omitted
Performance testing	Used	Rushed or omitted
Capacity testing	Used	Rushed or omitted

U.S. SOFTWARE DEFECT POTENTIALS AT FIVE-YEAR INTERVALS FROM 1945 TO 2000 AD

(Data Expressed in Terms of Defects Per Function Point)

Year	End-User	MIS	Outsrc.	Commer.	System	Military	Average
1945						1.50	*1.50*
1950		2.00			2.50	2.00	*2.17*
1955		2.25			2.50	2.50	*2.42*
1960		2.50		1.50	3.00	3.00	*2.50*
1965		2.50		1.75	3.25	3.50	*2.75*
1970		2.75		2.50	4.00	4.50	*3.44*
1975	1.00	3.00	3.00	3.00	6.00	5.50	*3.42*
1980	1.50	3.75	3.50	3.50	6.00	6.25	*4.08*
1985	2.00	5.00	4.50	4.50	6.00	7.00	*4.83*
1990	2.50	5.00	4.75	4.75	6.50	7.00	*5.08*
1995	2.50	5.50	5.00	5.25	6.00	6.50	*5.13*
2000	3.00	6.00	5.50	6.00	6.50	6.50	*5.58*
Average	*2.08*	*3.65*	*4.38*	*3.54*	*4.65*	*4.65*	*3.84*

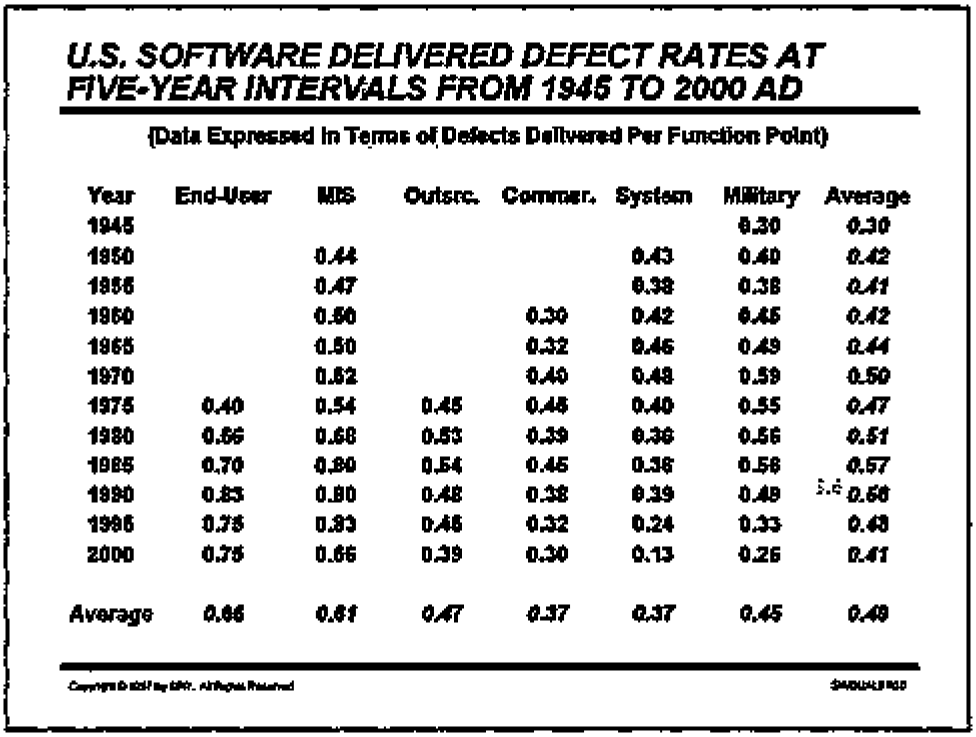


U.S. SOFTWARE DEFECT REMOVAL EFFICIENCY AT FIVE-YEAR INTERVALS FROM 1945 TO 2000 AD

(Data Expressed in Terms of Percentage of Defects Removed Before Deployment)

Year	End-User	MIS	Outsrc.	Commer.	System	Military	Average
1945						80.00%	*80.00%*
1950		78.00%			83.00%	80.00%	*80.33%*
1955		79.00%			85.00%	85.00%	*83.00%*
1960		80.00%		80.00%	85.00%	86.00%	*82.75%*
1965		80.00%		82.00%	86.00%	86.00%	*83.50%*
1970		81.00%		84.00%	88.00%	87.00%	*85.00%*
1975	60.00%	82.00%	85.00%	86.00%	92.00%	90.00%	*82.33%*
1980	63.00%	82.00%	85.00%	89.00%	94.00%	91.00%	*84.00%*
1985	65.00%	84.00%	88.00%	90.00%	94.00%	92.00%	*85.50%*
1990	67.00%	84.00%	90.00%	92.00%	94.00%	93.00%	*86.67%*
1995	70.00%	85.00%	91.00%	94.00%	96.00%	95.00%	*88.50%*
2000	75.00%	88.00%	93.00%	95.00%	98.00%	96.00%	*90.83%*
Average	*66.67%*	*82.08%*	*88.67%*	*87.89%*	*90.58%*	*88.33%*	*84.03%*

Copyright © 1997 by SPR. All Rights Reserved



U.S. SOFTWARE DELIVERED DEFECT RATES AT FIVE-YEAR INTERVALS FROM 1945 TO 2000 AD

(Data Expressed in Terms of Defects Delivered Per Function Point)

Year	End-User	MIS	Outsrc.	Commer.	System	Military	Average
1945						0.30	*0.30*
1950		0.44			0.43	0.40	*0.42*
1955		0.47			0.38	0.38	*0.41*
1960		0.50		0.30	0.42	0.45	*0.42*
1965		0.50		0.32	0.46	0.49	*0.44*
1970		0.62		0.40	0.48	0.59	0.50
1975	0.40	0.54	0.45	0.46	0.40	0.55	*0.47*
1980	0.66	0.68	0.53	0.39	0.36	0.56	*0.51*
1985	0.70	0.80	0.54	0.45	0.36	0.58	*0.57*
1990	0.83	0.80	0.48	0.38	0.39	0.49	*0.56*
1995	0.75	0.83	0.46	0.32	0.24	0.33	0.48
2000	0.75	0.66	0.39	0.30	0.13	0.26	0.41
Average	*0.66*	*0.61*	*0.47*	*0.37*	*0.37*	0.45	0.49

Copyright © 1997 by SPR. All Rights Reserved

SPR QUALITY PERFORMANCE LEVELS
CUMULATIVE DEFECT REMOVAL EFFICIENCY

(Development Defects + 1 Year of User Defect Reports)

SPR Performance Level	Efficiency Measured at One Year of Usage
1. Excellent	> 99%
2. Good	95%
3. Average	87%
4. Marginal	83%
5. Poor	< 80%

OPTIMIZING QUALITY AND PRODUCTIVITY

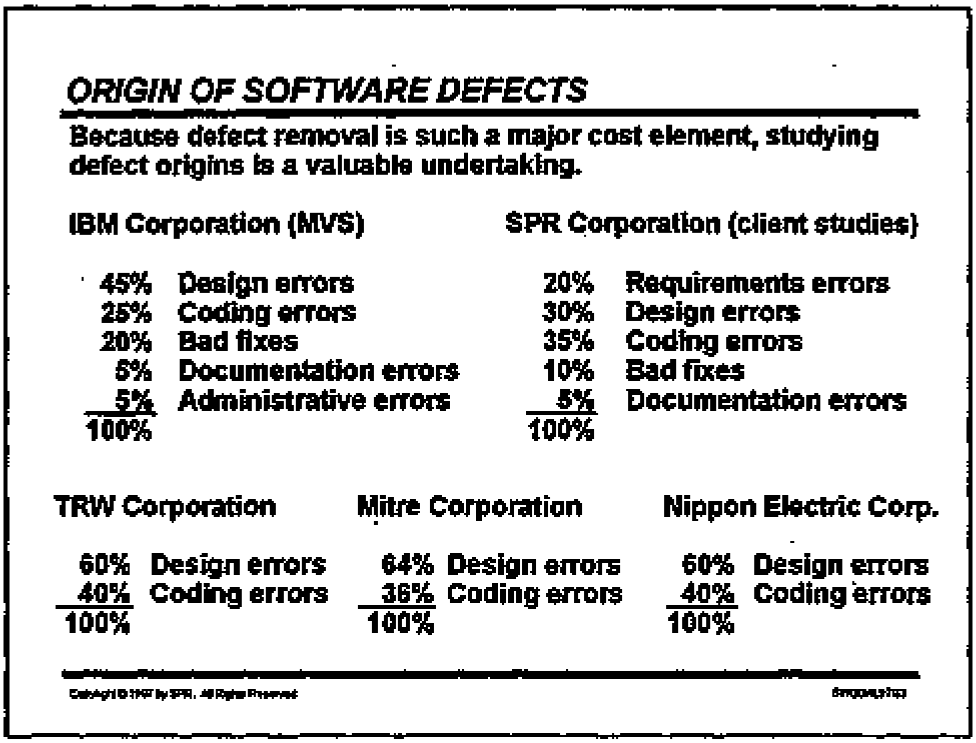
Projects that achieve 95% cumulative Defect Removal Efficiency will find:

1) Minimum schedules

2) Maximum productivity

3) High levels of user satisfaction

4) Low levels of delivered defects



ORIGIN OF SOFTWARE DEFECTS

Because defect removal is such a major cost element, studying defect origins is a valuable undertaking.

IBM Corporation (MVS)

45%	Design errors
25%	Coding errors
20%	Bad fixes
5%	Documentation errors
5%	Administrative errors
100%	

SPR Corporation (client studies)

20%	Requirements errors
30%	Design errors
35%	Coding errors
10%	Bad fixes
5%	Documentation errors
100%	

TRW Corporation

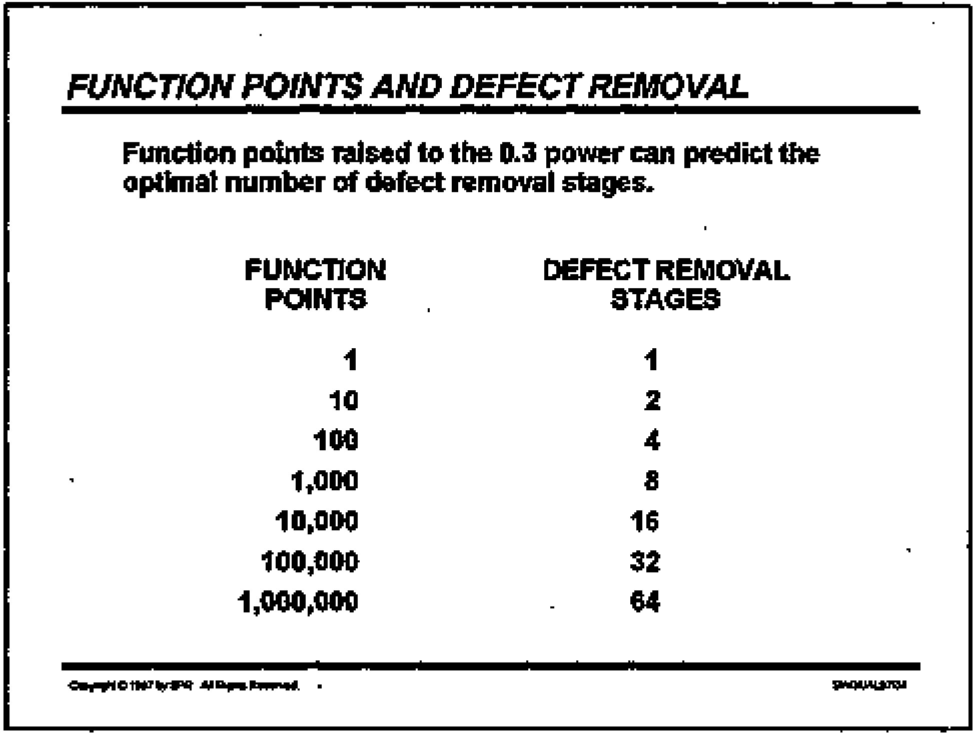
60%	Design errors
40%	Coding errors
100%	

Mitre Corporation

64%	Design errors
36%	Coding errors
100%	

Nippon Electric Corp.

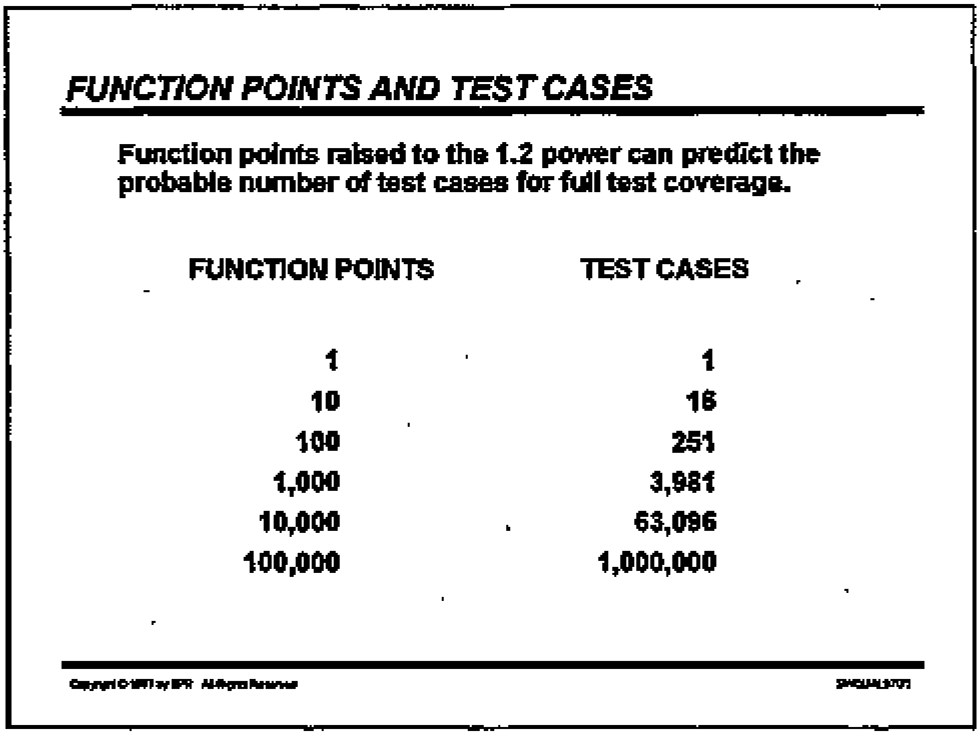
60%	Design errors
40%	Coding errors
100%	



FUNCTION POINTS AND DEFECT REMOVAL

Function points raised to the 0.3 power can predict the optimal number of defect removal stages.

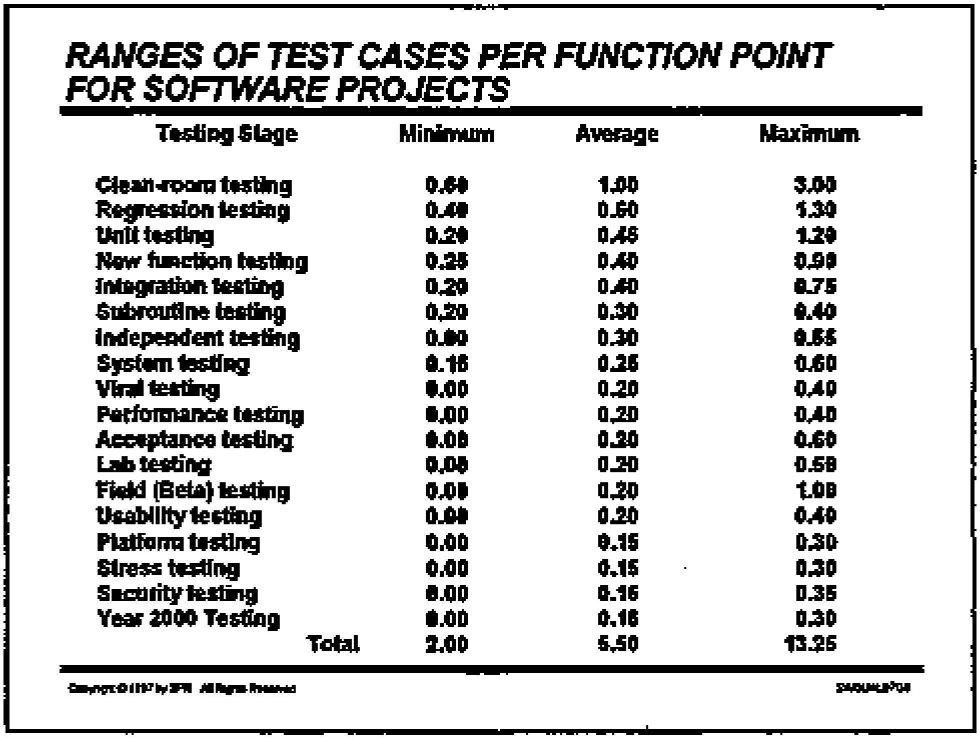
FUNCTION POINTS	DEFECT REMOVAL STAGES
1	1
10	2
100	4
1,000	8
10,000	16
100,000	32
1,000,000	64



FUNCTION POINTS AND TEST CASES

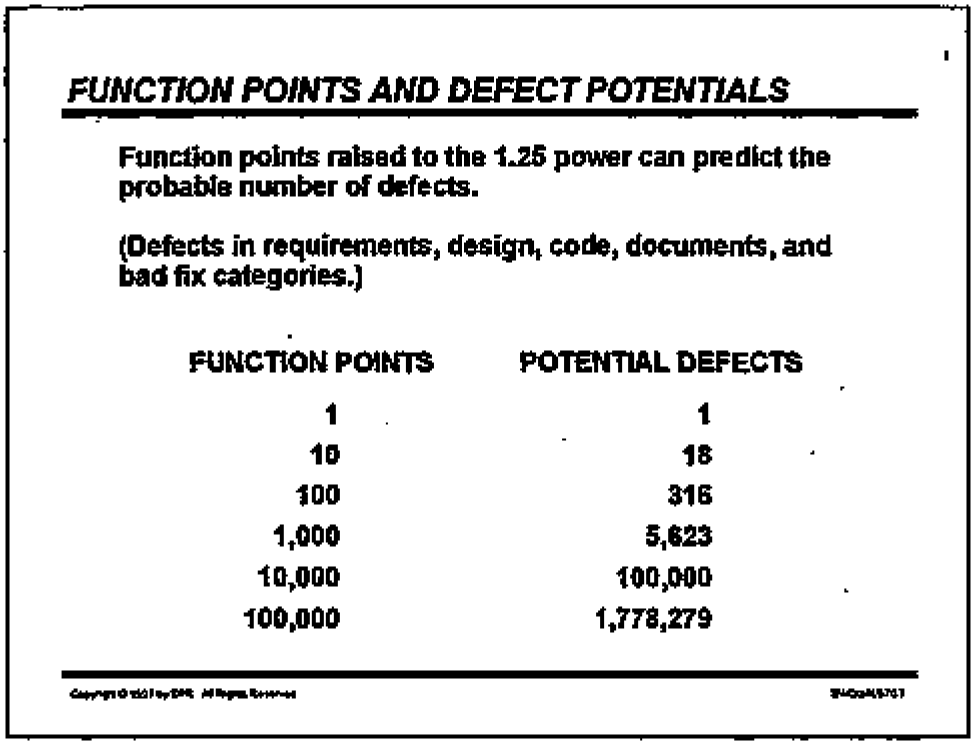
Function points raised to the 1.2 power can predict the probable number of test cases for full test coverage.

FUNCTION POINTS	TEST CASES
1	1
10	16
100	251
1,000	3,981
10,000	63,096
100,000	1,000,000



RANGES OF TEST CASES PER FUNCTION POINT FOR SOFTWARE PROJECTS

Testing Stage	Minimum	Average	Maximum
Clean-room testing	0.60	1.00	3.00
Regression testing	0.40	0.60	1.30
Unit testing	0.20	0.45	1.20
New function testing	0.25	0.40	0.90
Integration testing	0.20	0.40	0.75
Subroutine testing	0.20	0.30	0.40
Independent testing	0.00	0.30	0.55
System testing	0.15	0.25	0.60
Viral testing	0.00	0.20	0.40
Performance testing	0.00	0.20	0.40
Acceptance testing	0.00	0.20	0.60
Lab testing	0.00	0.20	0.50
Field (Beta) testing	0.00	0.20	1.00
Usability testing	0.00	0.20	0.40
Platform testing	0.00	0.15	0.30
Stress testing	0.00	0.15	0.30
Security testing	0.00	0.15	0.35
Year 2000 Testing	0.00	0.15	0.30
Total	2.00	5.50	13.25

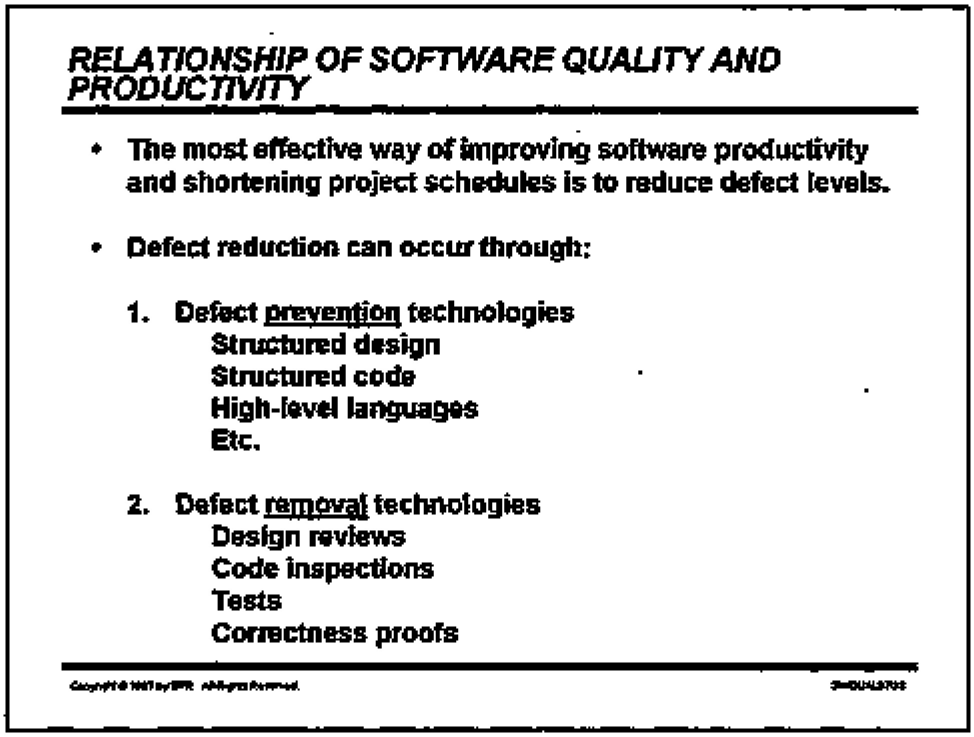


FUNCTION POINTS AND DEFECT POTENTIALS

Function points raised to the 1.25 power can predict the probable number of defects.

(Defects in requirements, design, code, documents, and bad fix categories.)

FUNCTION POINTS	POTENTIAL DEFECTS
1	1
10	18
100	316
1,000	5,623
10,000	100,000
100,000	1,778,279



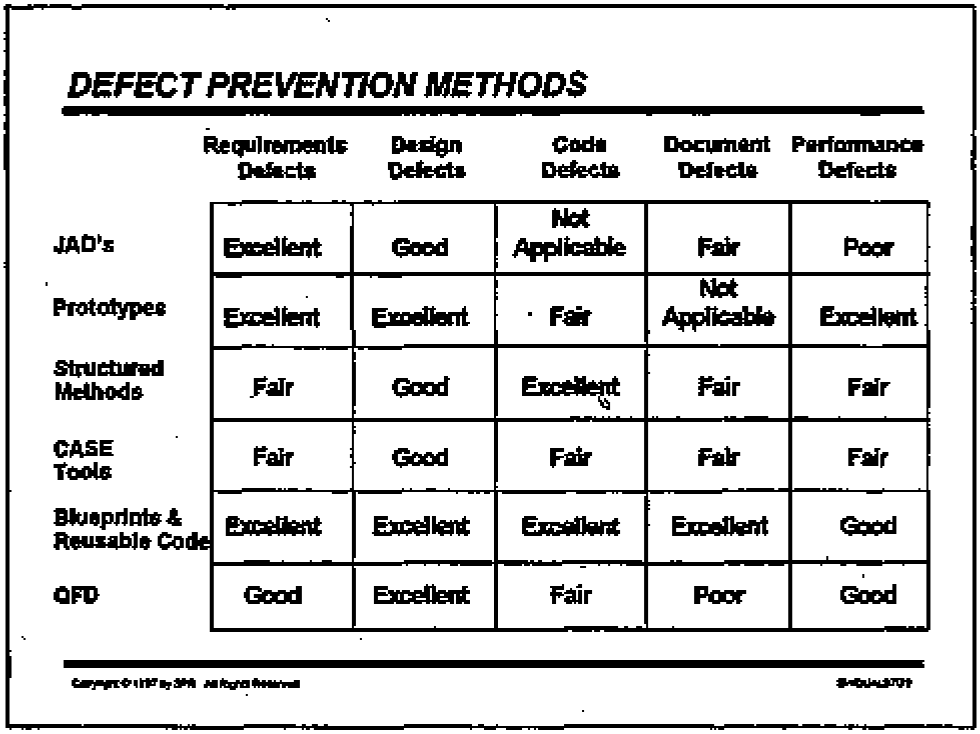
RELATIONSHIP OF SOFTWARE QUALITY AND PRODUCTIVITY

- The most effective way of improving software productivity and shortening project schedules is to reduce defect levels.

- Defect reduction can occur through:

1. Defect <u>prevention</u> technologies
   Structured design
   Structured code
   High-level languages
   Etc.

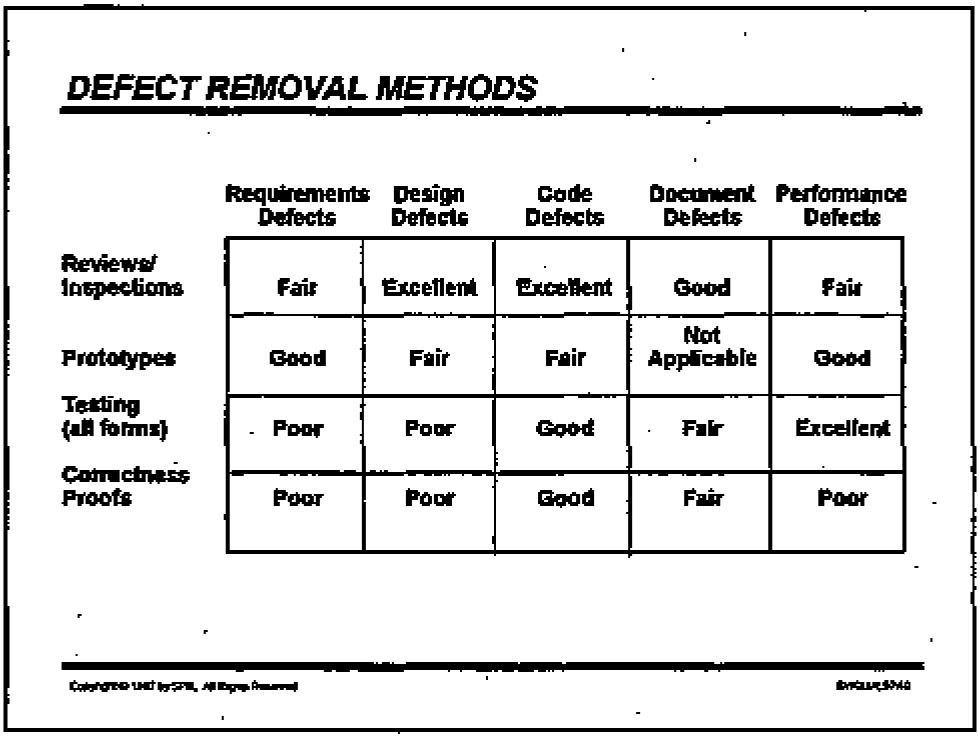
2. Defect <u>removal</u> technologies
   Design reviews
   Code inspections
   Tests
   Correctness proofs



DEFECT PREVENTION METHODS

	Requirements Defects	Design Defects	Code Defects	Document Defects	Performance Defects
JAD's	Excellent	Good	Not Applicable	Fair	Poor
Prototypes	Excellent	Excellent	Fair	Not Applicable	Excellent
Structured Methods	Fair	Good	Excellent	Fair	Fair
CASE Tools	Fair	Good	Fair	Fair	Fair
Blueprints & Reusable Code	Excellent	Excellent	Excellent	Excellent	Good
QFD	Good	Excellent	Fair	Poor	Good

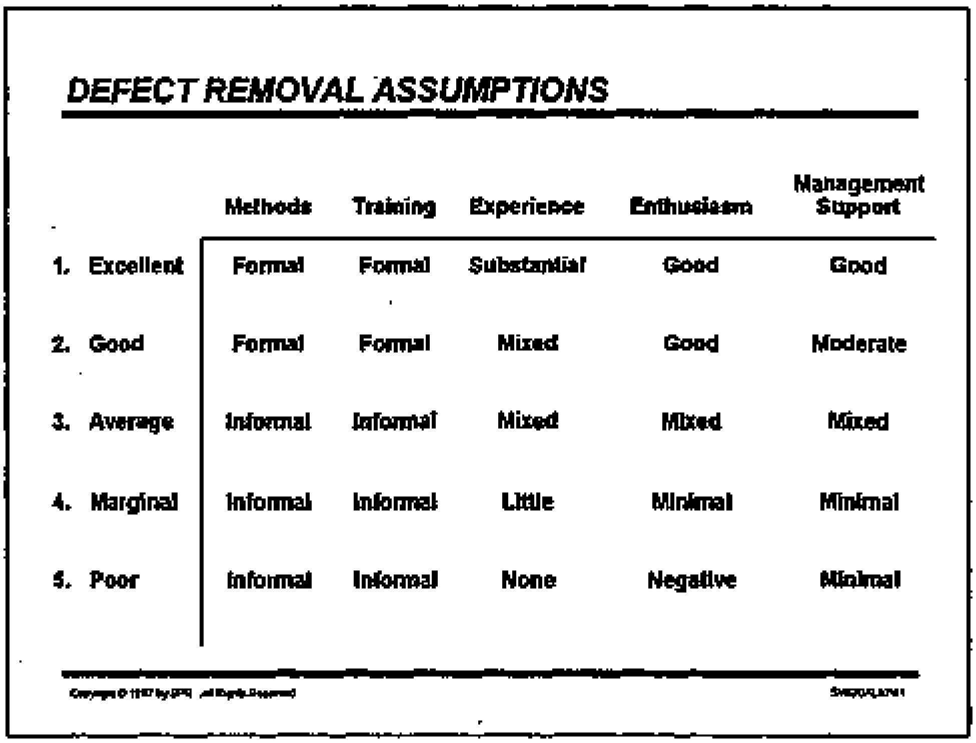
[illegible]



DEFECT REMOVAL METHODS

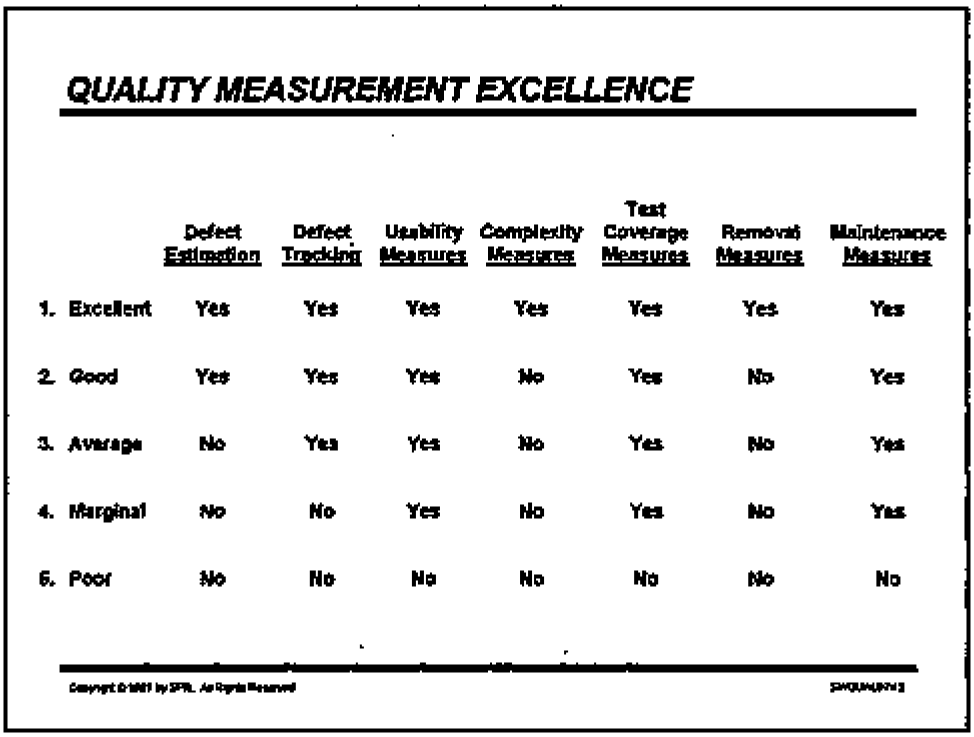
	Requirements Defects	Design Defects	Code Defects	Document Defects	Performance Defects
Reviews/ Inspections	Fair	Excellent	Excellent	Good	Fair
Prototypes	Good	Fair	Fair	Not Applicable	Good
Testing (all forms)	Poor	Poor	Good	Fair	Excellent
Correctness Proofs	Poor	Poor	Good	Fair	Poor

[illegible]



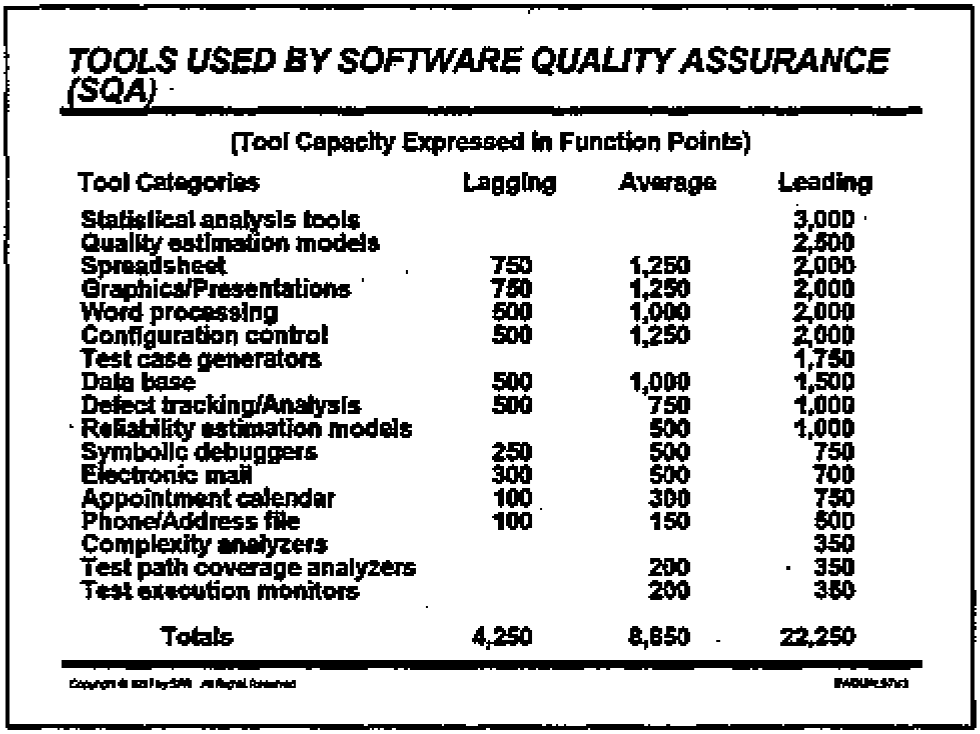
DEFECT REMOVAL ASSUMPTIONS

	Methods	Training	Experience	Enthusiasm	Management Support
1. Excellent	Formal	Formal	Substantial	Good	Good
2. Good	Formal	Formal	Mixed	Good	Moderate
3. Average	Informal	Informal	Mixed	Mixed	Mixed
4. Marginal	Informal	Informal	Little	Minimal	Minimal
5. Poor	Informal	Informal	None	Negative	Minimal



QUALITY MEASUREMENT EXCELLENCE

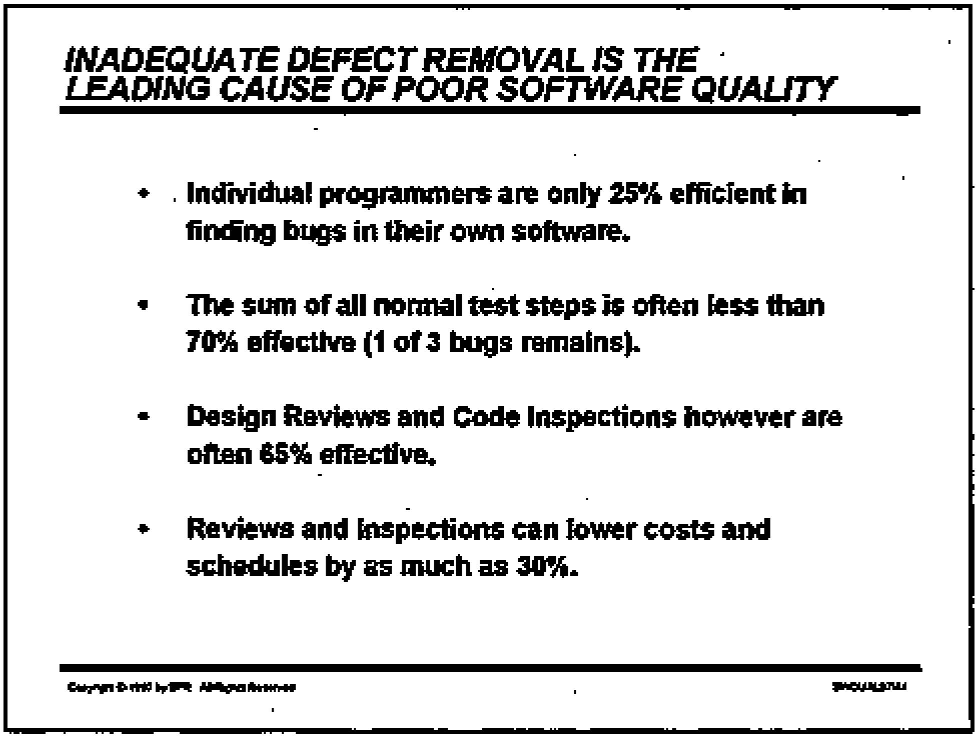
	Defect Estimation	Defect Tracking	Usability Measures	Complexity Measures	Test Coverage Measures	Removal Measures	Maintenance Measures
1. Excellent	Yes	Yes	Yes	Yes	Yes	Yes	Yes
2. Good	Yes	Yes	Yes	No	Yes	No	Yes
3. Average	No	Yes	Yes	No	Yes	No	Yes
4. Marginal	No	No	Yes	No	Yes	No	Yes
6. Poor	No	No	No	No	No	No	No



TOOLS USED BY SOFTWARE QUALITY ASSURANCE (SQA)

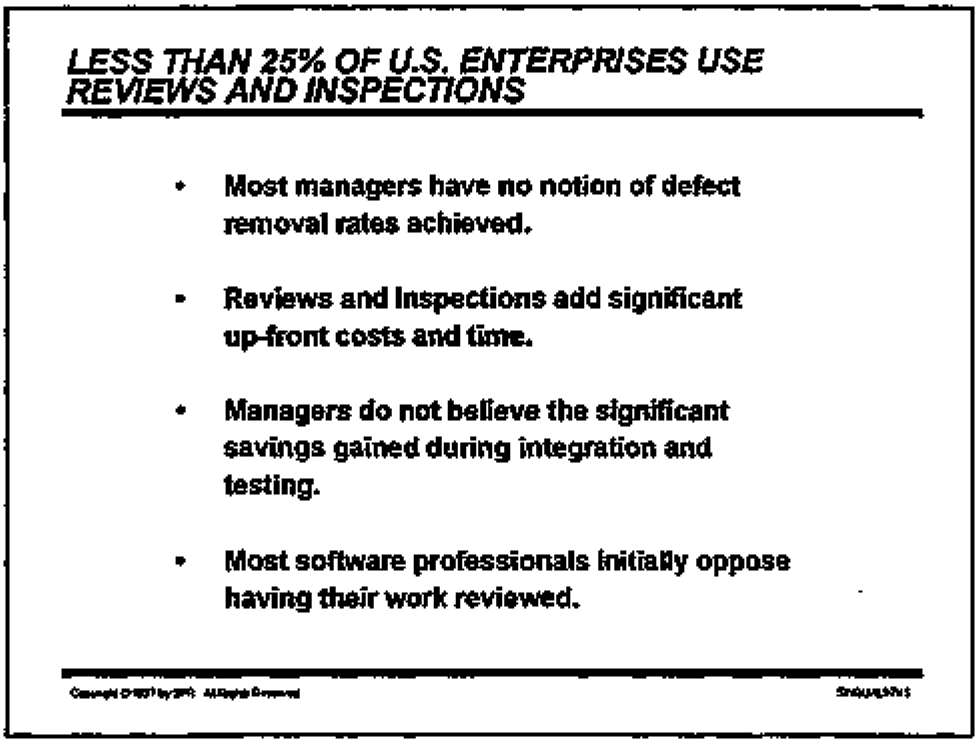
(Tool Capacity Expressed in Function Points)

Tool Categories	Lagging	Average	Leading
Statistical analysis tools			3,000
Quality estimation models			2,500
Spreadsheet	750	1,250	2,000
Graphics/Presentations	750	1,250	2,000
Word processing	500	1,000	2,000
Configuration control	500	1,250	2,000
Test case generators			1,750
Data base	500	1,000	1,500
Defect tracking/Analysis	500	750	1,000
Reliability estimation models		500	1,000
Symbolic debuggers	250	500	750
Electronic mail	300	500	700
Appointment calendar	100	300	750
Phone/Address file	100	150	500
Complexity analyzers			350
Test path coverage analyzers		200	350
Test execution monitors		200	350
Totals	4,250	8,850	22,250



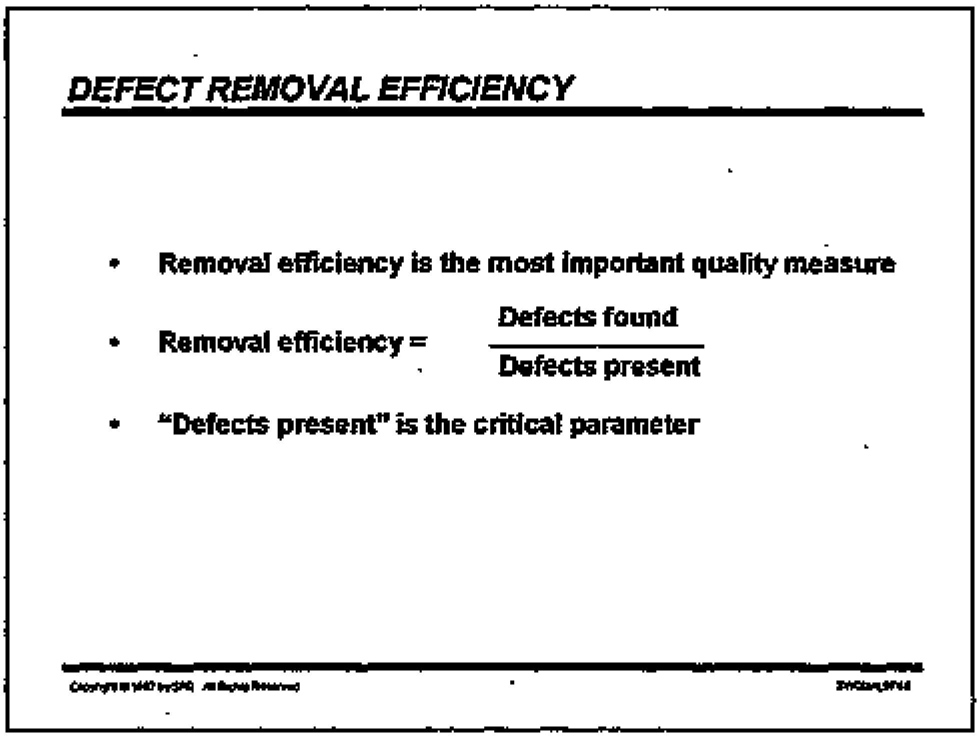
INADEQUATE DEFECT REMOVAL IS THE LEADING CAUSE OF POOR SOFTWARE QUALITY

- Individual programmers are only 25% efficient in finding bugs in their own software.
- The sum of all normal test steps is often less than 70% effective (1 of 3 bugs remains).
- Design Reviews and Code Inspections however are often 65% effective.
- Reviews and Inspections can lower costs and schedules by as much as 30%.



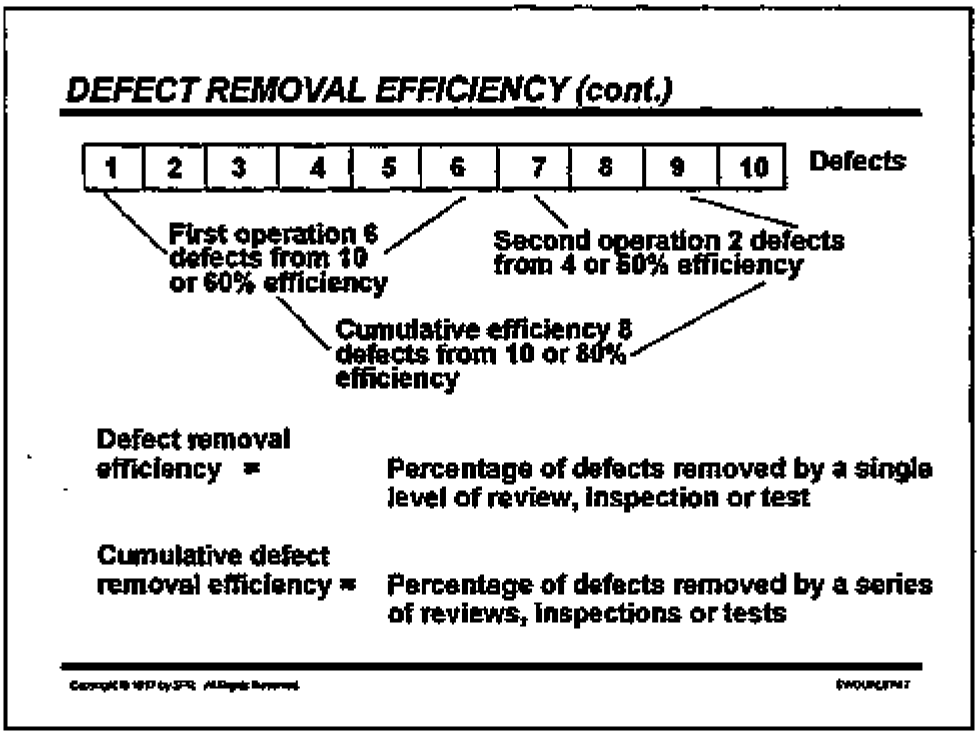
LESS THAN 25% OF U.S. ENTERPRISES USE REVIEWS AND INSPECTIONS

- Most managers have no notion of defect removal rates achieved.
- Reviews and Inspections add significant up-front costs and time.
- Managers do not believe the significant savings gained during integration and testing.
- Most software professionals initially oppose having their work reviewed.



DEFECT REMOVAL EFFICIENCY

- Removal efficiency is the most important quality measure
- Removal efficiency = $\frac{\text{Defects found}}{\text{Defects present}}$
- "Defects present" is the critical parameter



DEFECT REMOVAL EFFICIENCY (cont.)

1	2	3	4	5	6	7	8	9	10	Defects

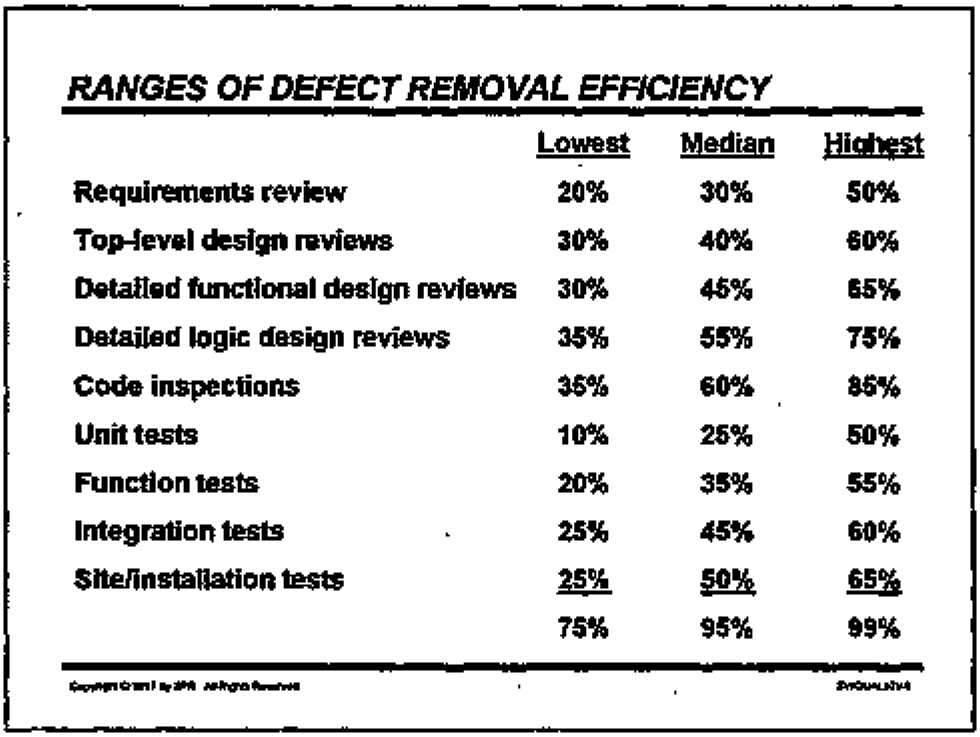
First operation 6 defects from 10 or 60% efficiency

Second operation 2 defects from 4 or 50% efficiency

Cumulative efficiency 8 defects from 10 or 80% efficiency

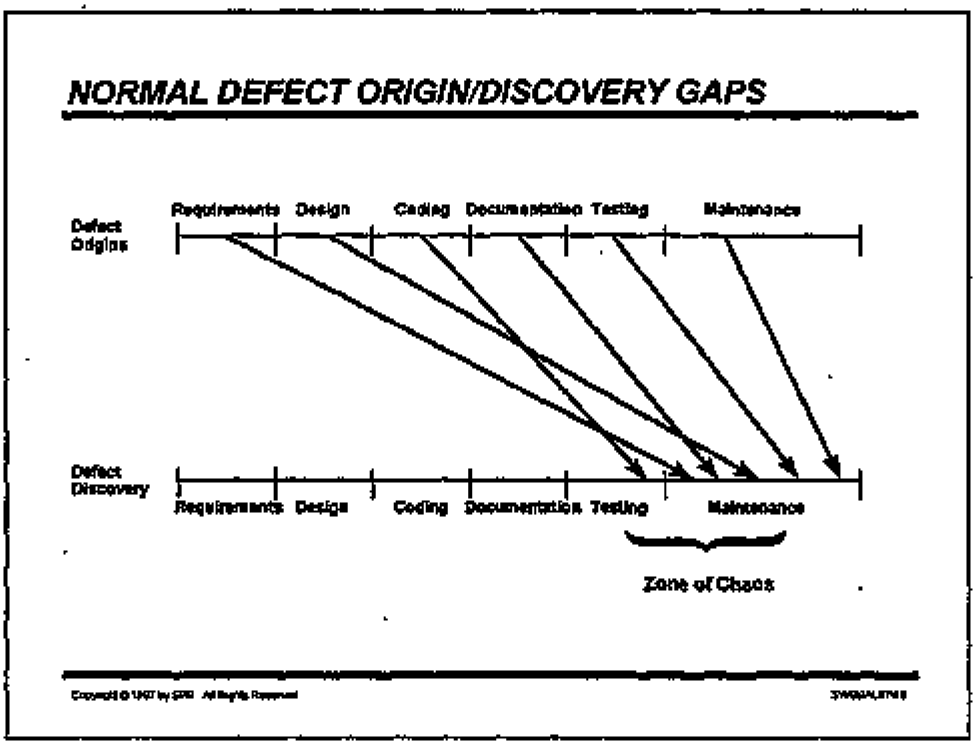
Defect removal efficiency = Percentage of defects removed by a single level of review, inspection or test

Cumulative defect removal efficiency = Percentage of defects removed by a series of reviews, inspections or tests



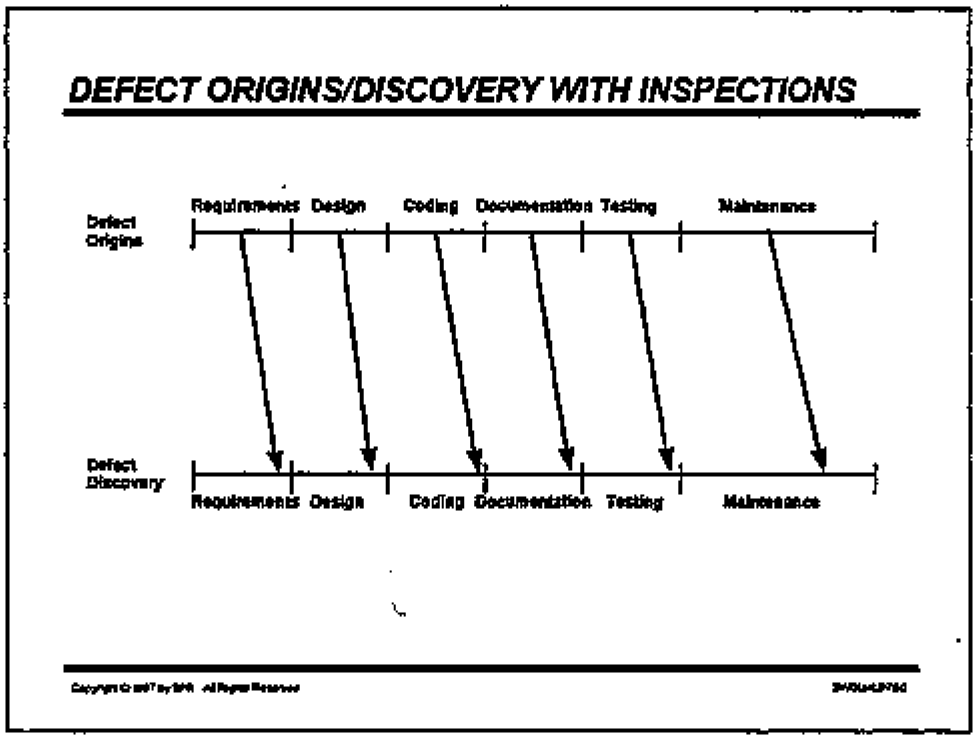
RANGES OF DEFECT REMOVAL EFFICIENCY

	Lowest	Median	Highest
Requirements review	20%	30%	50%
Top-level design reviews	30%	40%	60%
Detailed functional design reviews	30%	45%	65%
Detailed logic design reviews	35%	55%	75%
Code inspections	35%	60%	85%
Unit tests	10%	25%	50%
Function tests	20%	35%	55%
Integration tests	25%	45%	60%
Site/installation tests	25%	50%	65%
	75%	95%	99%



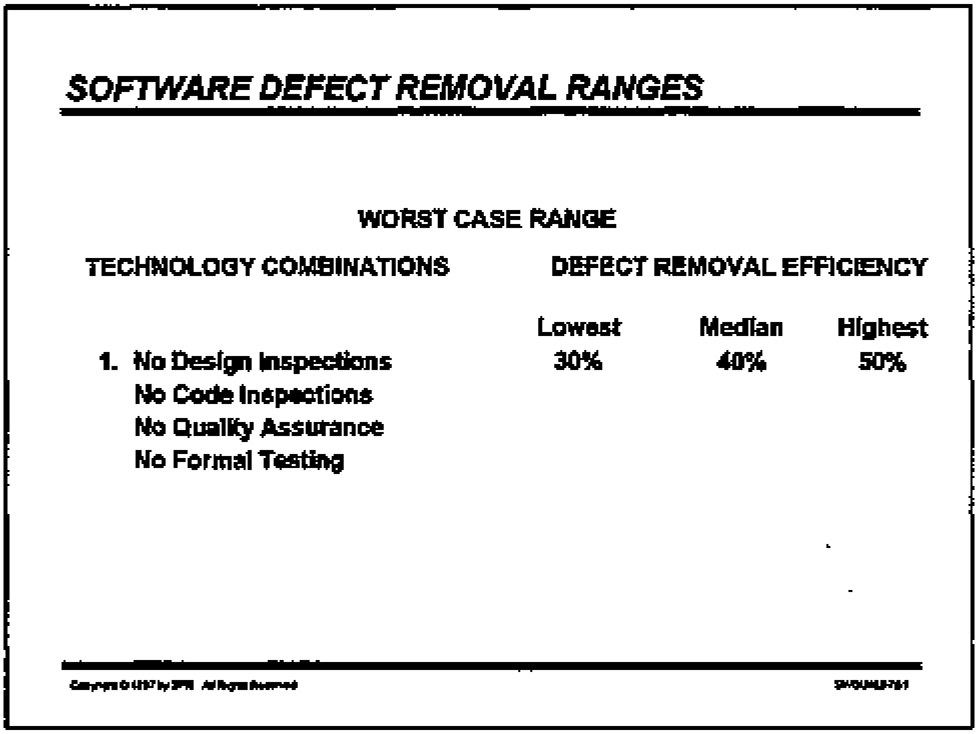
NORMAL DEFECT ORIGIN/DISCOVERY GAPS

Copyright © 1997 by SPR. All Rights Reserved [illegible]



DEFECT ORIGINS/DISCOVERY WITH INSPECTIONS

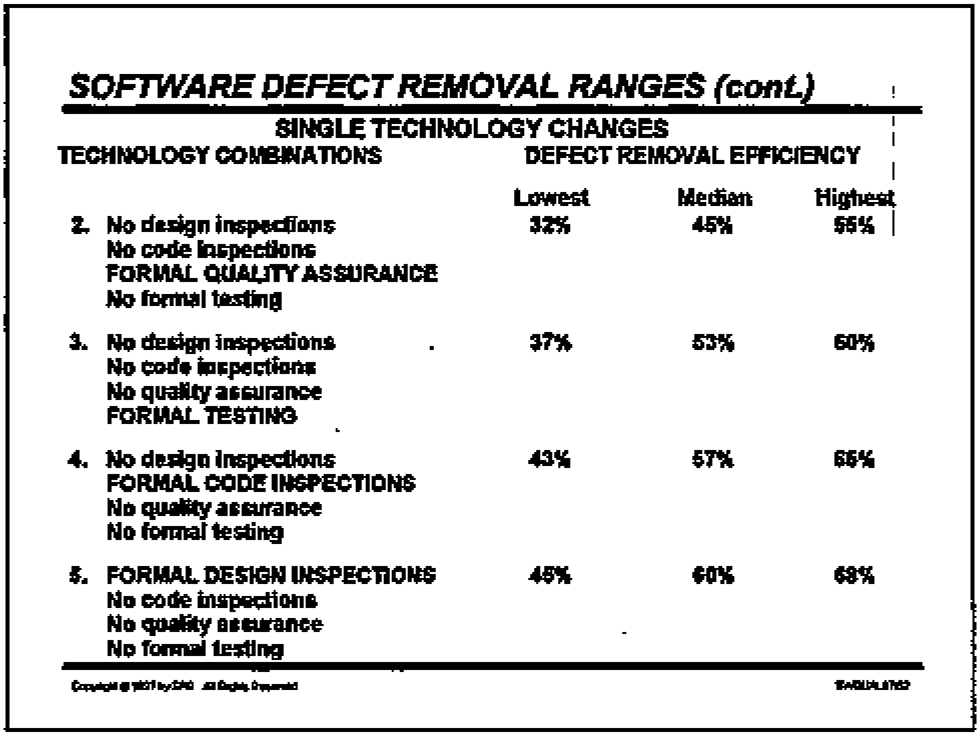
Copyright © 1997 by SPR. All Rights Reserved [illegible]



SOFTWARE DEFECT REMOVAL RANGES

WORST CASE RANGE

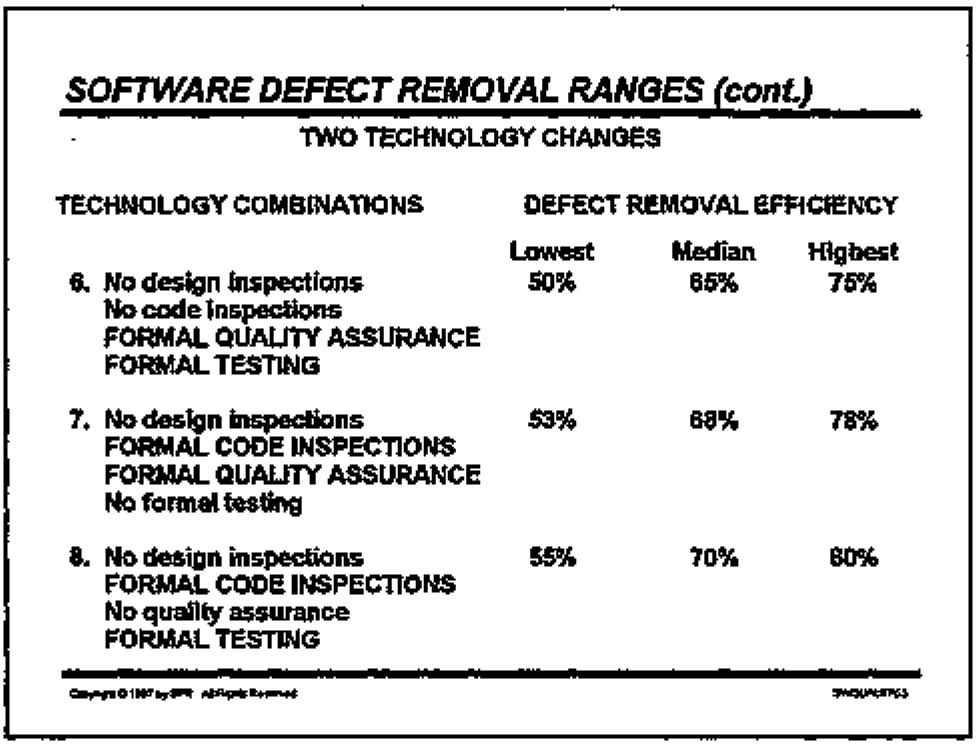
TECHNOLOGY COMBINATIONS	DEFECT REMOVAL EFFICIENCY		
	Lowest	Median	Highest
1. No Design Inspections No Code Inspections No Quality Assurance No Formal Testing	30%	40%	50%



SOFTWARE DEFECT REMOVAL RANGES (cont.)

SINGLE TECHNOLOGY CHANGES

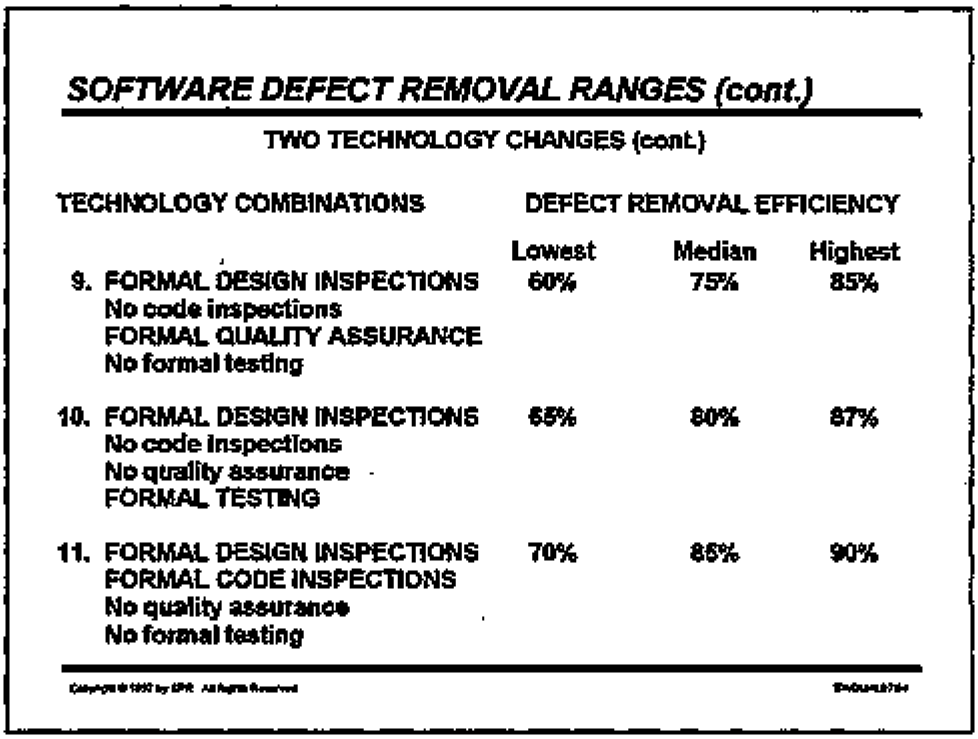
TECHNOLOGY COMBINATIONS	DEFECT REMOVAL EFFICIENCY		
	Lowest	Median	Highest
2. No design inspections No code inspections FORMAL QUALITY ASSURANCE No formal testing	32%	45%	55%
3. No design inspections No code inspections No quality assurance FORMAL TESTING	37%	53%	60%
4. No design inspections FORMAL CODE INSPECTIONS No quality assurance No formal testing	43%	57%	65%
5. FORMAL DESIGN INSPECTIONS No code inspections No quality assurance No formal testing	45%	60%	68%



SOFTWARE DEFECT REMOVAL RANGES (cont.)

TWO TECHNOLOGY CHANGES

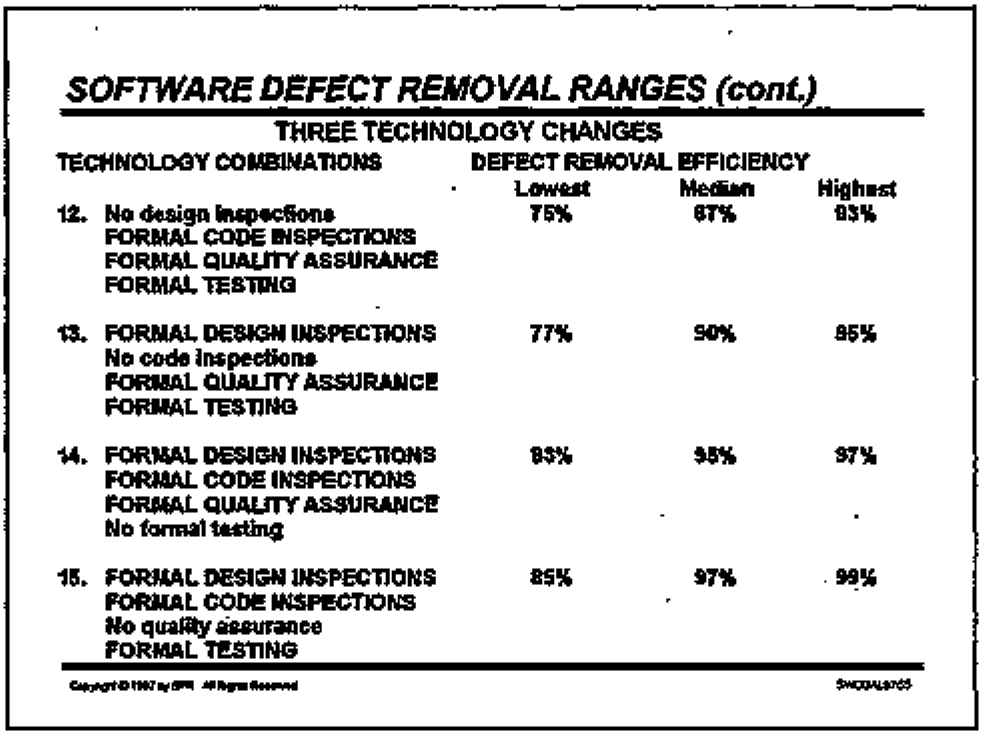
TECHNOLOGY COMBINATIONS	DEFECT REMOVAL EFFICIENCY		
	Lowest	Median	Highest
6. No design inspections No code inspections FORMAL QUALITY ASSURANCE FORMAL TESTING	50%	65%	75%
7. No design inspections FORMAL CODE INSPECTIONS FORMAL QUALITY ASSURANCE No formal testing	53%	68%	78%
8. No design inspections FORMAL CODE INSPECTIONS No quality assurance FORMAL TESTING	55%	70%	80%



SOFTWARE DEFECT REMOVAL RANGES (cont.)

TWO TECHNOLOGY CHANGES (cont.)

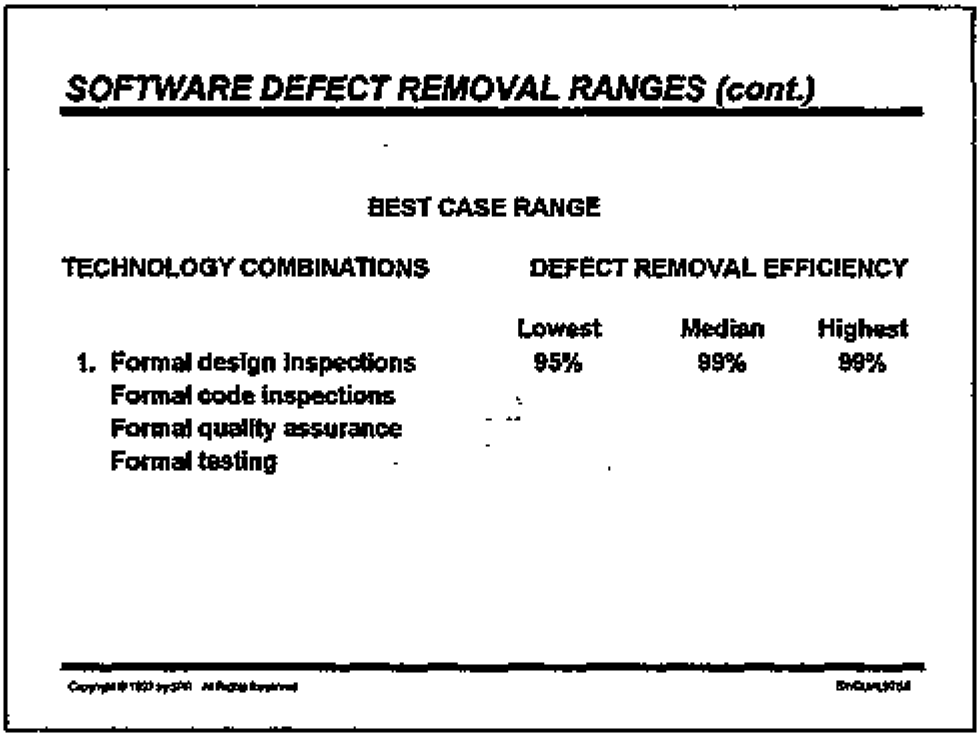
TECHNOLOGY COMBINATIONS	DEFECT REMOVAL EFFICIENCY		
	Lowest	Median	Highest
9. FORMAL DESIGN INSPECTIONS No code inspections FORMAL QUALITY ASSURANCE No formal testing	60%	75%	85%
10. FORMAL DESIGN INSPECTIONS No code inspections No quality assurance FORMAL TESTING	65%	80%	87%
11. FORMAL DESIGN INSPECTIONS FORMAL CODE INSPECTIONS No quality assurance No formal testing	70%	85%	90%



SOFTWARE DEFECT REMOVAL RANGES (cont.)

THREE TECHNOLOGY CHANGES

TECHNOLOGY COMBINATIONS	DEFECT REMOVAL EFFICIENCY		
	Lowest	Median	Highest
12. No design inspections FORMAL CODE INSPECTIONS FORMAL QUALITY ASSURANCE FORMAL TESTING	75%	87%	93%
13. FORMAL DESIGN INSPECTIONS No code inspections FORMAL QUALITY ASSURANCE FORMAL TESTING	77%	90%	95%
14. FORMAL DESIGN INSPECTIONS FORMAL CODE INSPECTIONS FORMAL QUALITY ASSURANCE No formal testing	93%	95%	97%
15. FORMAL DESIGN INSPECTIONS FORMAL CODE INSPECTIONS No quality assurance FORMAL TESTING	95%	97%	99%



SOFTWARE DEFECT REMOVAL RANGES (cont.)

BEST CASE RANGE

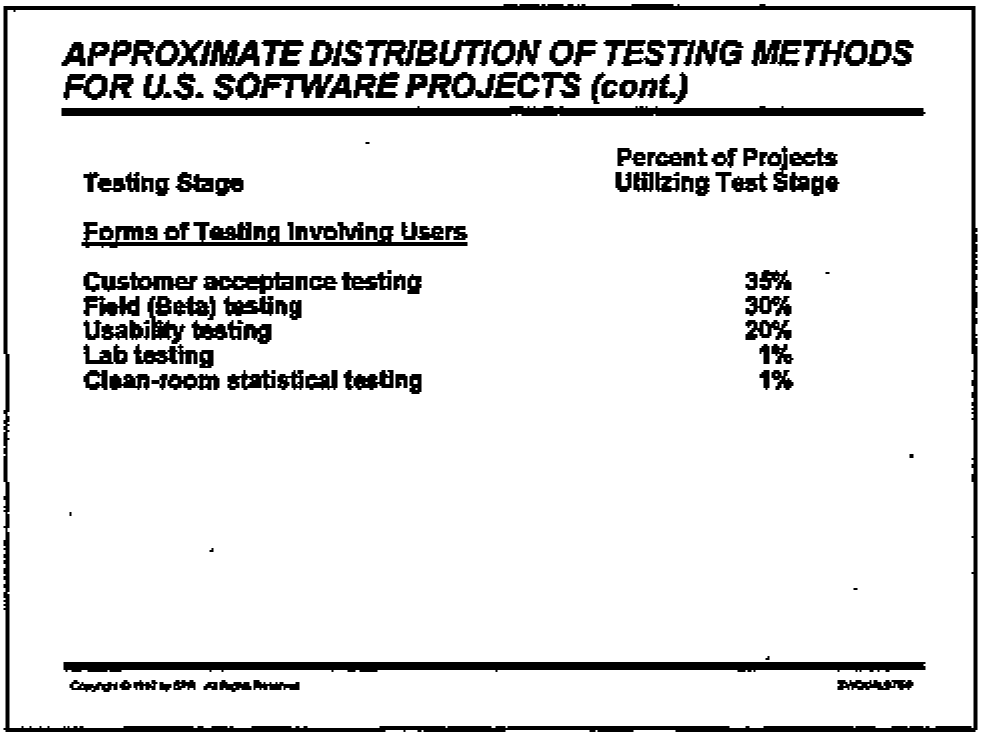
TECHNOLOGY COMBINATIONS	DEFECT REMOVAL EFFICIENCY		
	Lowest	Median	Highest
1. Formal design inspections Formal code inspections Formal quality assurance Formal testing	95%	99%	99%

DISTRIBUTION OF 1500 SOFTWARE PROJECTS BY DEFECT REMOVAL EFFICIENCY LEVEL

Defect Removal Efficiency Level (Percent)	Number of Projects	Percent of Projects
> 99	6	0.40%
95 - 99	104	6.93%
90 - 95	263	17.53%
85 - 90	559	37.26%
80 - 85	406	27.20%
< 80	161	10.73%
Total	1,500	100.00%

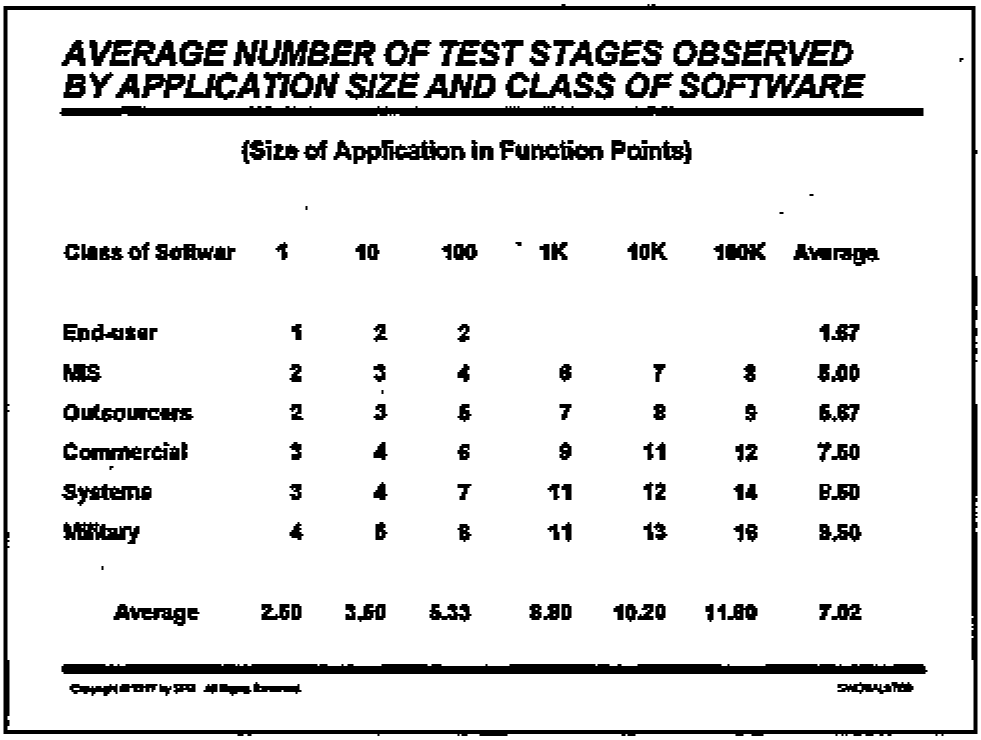
APPROXIMATE DISTRIBUTION OF TESTING METHODS FOR U.S. SOFTWARE PROJECTS

Testing Stage	Percent of Projects Utilizing Test Stage
<u>General Forms of Testing</u>	
Subroutine testing	100%
Unit testing	99%
System testing of full application	95%
New function testing	90%
Regression testing	70%
Integration testing	50%
<u>Specialized Forms of Testing</u>	
Viral protection testing	45%
Stress or capacity testing	35%
Performance testing	30%
Security testing	15%
Platform testing	5%
Year 2000 testing	5%
Independent testing	3%



APPROXIMATE DISTRIBUTION OF TESTING METHODS FOR U.S. SOFTWARE PROJECTS (cont.)

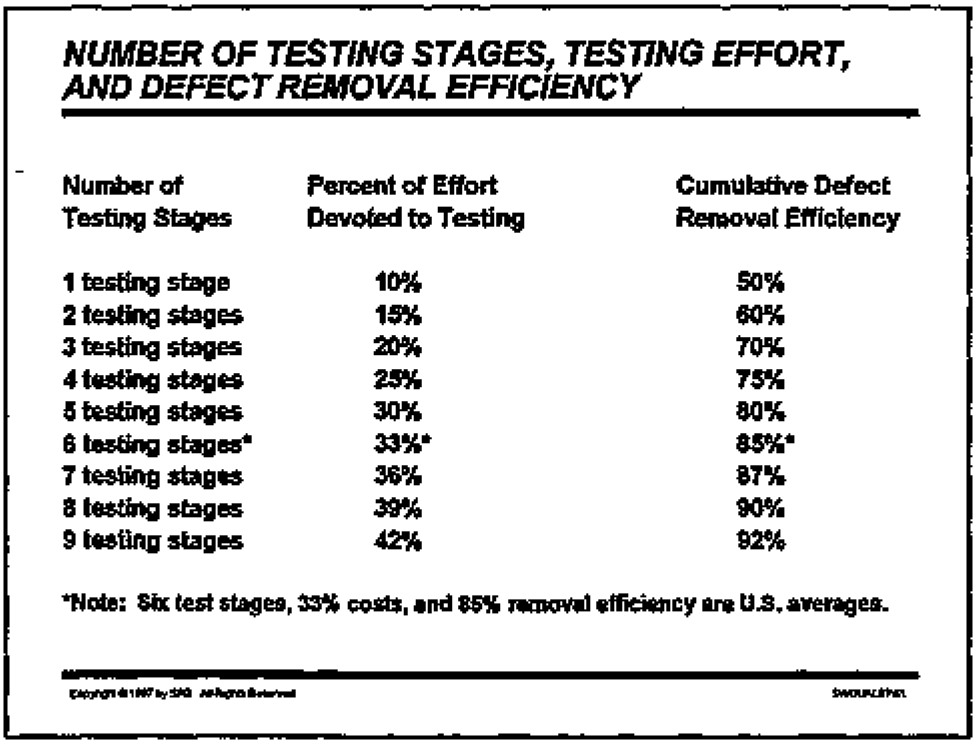
Testing Stage	Percent of Projects Utilizing Test Stage
Forms of Testing Involving Users	
Customer acceptance testing	35%
Field (Beta) testing	30%
Usability testing	20%
Lab testing	1%
Clean-room statistical testing	1%



AVERAGE NUMBER OF TEST STAGES OBSERVED BY APPLICATION SIZE AND CLASS OF SOFTWARE

(Size of Application in Function Points)

Class of Softwar	1	10	100	1K	10K	100K	Average
End-user	1	2	2				1.67
MIS	2	3	4	6	7	8	5.00
Outsourcers	2	3	5	7	8	9	5.67
Commercial	3	4	6	9	11	12	7.50
Systems	3	4	7	11	12	14	8.50
Military	4	5	8	11	13	16	9.50
Average	2.50	3.50	5.33	8.80	10.20	11.80	7.02

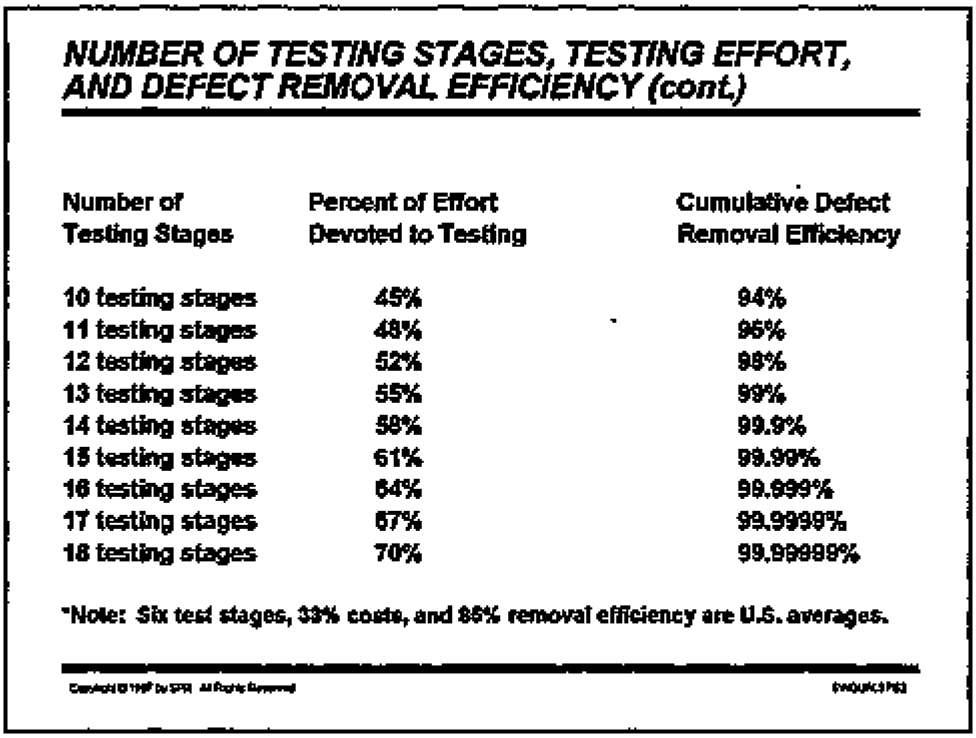


NUMBER OF TESTING STAGES, TESTING EFFORT, AND DEFECT REMOVAL EFFICIENCY

Number of Testing Stages	Percent of Effort Devoted to Testing	Cumulative Defect Removal Efficiency
1 testing stage	10%	50%
2 testing stages	15%	60%
3 testing stages	20%	70%
4 testing stages	25%	75%
5 testing stages	30%	80%
6 testing stages*	33%*	85%*
7 testing stages	36%	87%
8 testing stages	39%	90%
9 testing stages	42%	92%

*Note: Six test stages, 33% costs, and 85% removal efficiency are U.S. averages.

[illegible]

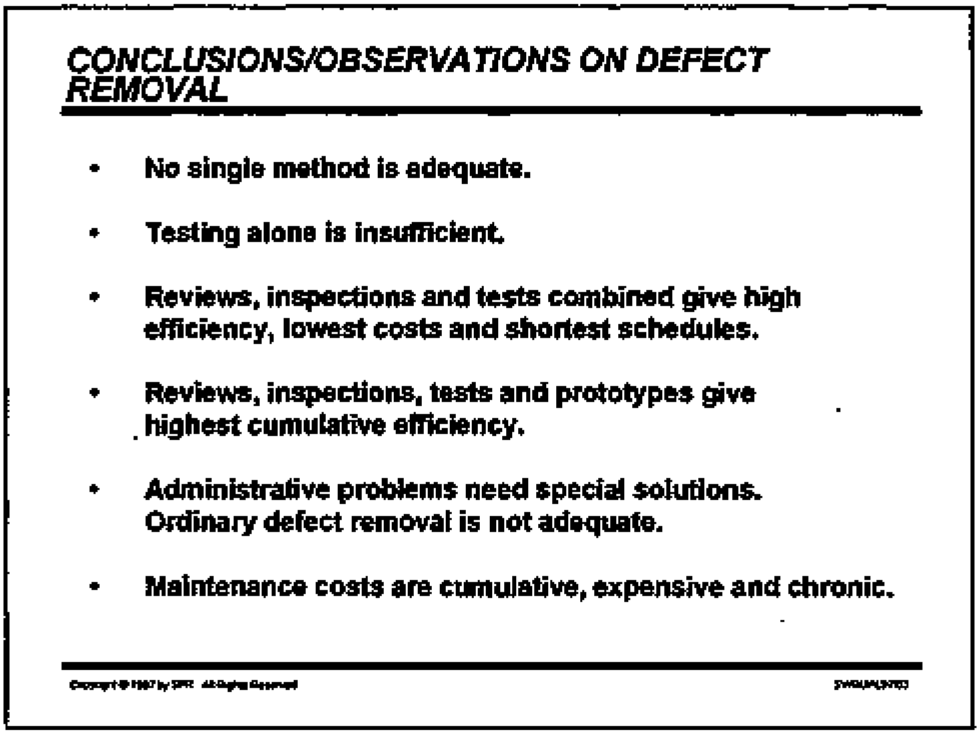


NUMBER OF TESTING STAGES, TESTING EFFORT, AND DEFECT REMOVAL EFFICIENCY (cont.)

Number of Testing Stages	Percent of Effort Devoted to Testing	Cumulative Defect Removal Efficiency
10 testing stages	45%	94%
11 testing stages	48%	96%
12 testing stages	52%	98%
13 testing stages	55%	99%
14 testing stages	58%	99.9%
15 testing stages	61%	99.99%
16 testing stages	64%	99.999%
17 testing stages	67%	99.9999%
18 testing stages	70%	99.99999%

*Note: Six test stages, 33% costs, and 85% removal efficiency are U.S. averages.

[illegible]



CONCLUSIONS/OBSERVATIONS ON DEFECT REMOVAL

- No single method is adequate.
- Testing alone is insufficient.
- Reviews, inspections and tests combined give high efficiency, lowest costs and shortest schedules.
- Reviews, inspections, tests and prototypes give highest cumulative efficiency.
- Administrative problems need special solutions. Ordinary defect removal is not adequate.
- Maintenance costs are cumulative, expensive and chronic.

Software Quality in 1997

January 9, 1997

Abstract

For many years, software quality assurance lagged behind hardware quality assurance in terms of methods, metrics, and successful results. New approaches such as Quality Function Deployment (QFD) the ISO 9000-9004 standards, the SEI maturity levels, and Total Quality Management (TQM) are starting to attract wide attention, and in some cases to bring software quality levels up to a parity with manufacturing quality levels. Since software is on the critical path for many engineered products, and for internal business systems as well, the new approaches are starting to affect global competition and attract widespread international interest. It can be hypothesized that success in mastering software quality will be a key strategy for dominating global software markets in the 21st century.

Capers Jones, Chairman
Software Productivity Research, Inc.
1 New England Executive Park
Burlington, MA 01803-5005

Phone 617 273 0140
FAX 617 273 5176
Email capers@spr.com

INTRODUCTION

Software has become one of the most pervasive technologies of the 20th century. Within the past 30 years, software has spread from a small number of comparatively specialized applications to become a critical factor in almost all engineered products. Software has also become a major factor in consumer goods, and in company operations, military operations, and government operations. Thirty years ago, poor software quality was often annoying, but today poor software quality can literally shut down a phone system, a defense system, and even a company. Any reasonable prognosis makes software even more critical in the future, and hence software quality will become more critical than today as well.

As many countries strive to compete in the international software market place, quality is now a major topic for both software vendors and for outsource contractors. Any country or company that wants to achieve a major place in world software markets must achieve and maintain high software quality levels.

Barriers to Software Quality Exploration

Progress in all forms of engineering is heavily dependent upon accurate measurement and precise metrics. Software achieved notoriety as being the worst measured engineering discipline of all time. The main barrier to software quality control in the 1950's, 60's, and 70's was a simple lack of good quantitative data about software quality levels, reliability, defect removal efficiency and other basic quality data. This lack of data was not because software managers and professionals did not care about quality, but because there were no effective metrics prior to 1979 that could actually be used to measure software quality.

Historically, software quality was measured crudely in terms of "defects found per 1000 source code statements" (normally abbreviated to KLOC). Unfortunately, that metric contained a built-in paradox which caused it to give erroneous results when used with newer and more powerful programming languages, such as Ada, object-oriented languages, or program generators. The results were so poor that several leading companies stopped trying to measure software, and lagging companies never started.

In 1979, A.J. Albrecht of IBM published a new metric for measuring both software quality and productivity, which he termed "Function Points." A Function Point is a synthetic metric derived from five visible external characteristics of software applications: 1) Inputs; 2) Outputs; 3) Inquiries; 4) Logical files; 5) Interfaces.

Function Points are completely divorced from lines of source code. In a sense, Function Points are like European Currency Units (ECU), which are synthetic metrics that allow rational economic and financial studies across multiple national currencies. Function Points allow rational quality and productivity studies across the 400 or so programming languages that have come into being.

In 1986, Function Point users formed a non-profit association, the International Function Point Users Group, or IFPUG. This organization and its affiliates now have over 500 corporations and government agencies as members in the United States, Canada, Europe, South America, and the Pacific Rim and membership is growing by more than 45% per year.

It is an interesting business phenomenon that measurement of software quality and productivity is now among the most rapidly growing technologies in the entire history of software.

One of the advantages of the Function Point metric is that it can be used to predict and measure all sources of software errors, and not just coding errors. Based on a study of more than 6700 software projects published in the book Applied Software Measurement (McGraw-Hill, 1996), the average number of software errors is about five per function point, apportioned across the following major defect origins. However, the "best in class" software organizations are achieving defect potentials of roughly half the total of "average" groups as shown in Table 1:

Table 1: U.S. Averages and "Best in Class" Defects per Function Point

Defect Origins	Average Defects per Function Point	Best in Class Defects per Function Point	Difference
Requirements	1.00	0.40	0.60
Design	1.25	0.60	0.65
Coding	1.75	1.00	0.75
Document	0.60	0.40	0.20
Bad Fixes	0.40	0.10	0.30
Total	*5.00*	*2.50*	*2.50*

These numbers represent the total numbers of defects that are found and measured from early software requirements throughout the remainder of the lifecycle of the software.

Complementing the Function Point metric are measurements of defect removal efficiency, or the percentages of software defects removed prior to delivery of the software to clients. The U.S. average for defect removal efficiency, unfortunately, is currently only about 85% although the best projects in leading companies such as Motorola, Raytheon, IBM, and Hewlett Packard achieve defect removal efficiency levels well in excess of 99%.

All software defects are not equally easy to remove. Requirements errors, design problems, and "bad fixes" tend to be the most difficult. Thus, on the day when software is actually put into production, the average quantity of latent errors or defects tends to be about 0.75 per Function Point, with the following distribution as shown in Table 2:

Table 2: U.S. Averages for Defect Potentials and Removal Efficiency Levels

Defect Origins	Defect Potentials	Removal Efficiency	Delivered Defects
Requirements	1.00	77%	0.23
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Document	0.60	80%	0.12
Bad Fixes	0.40	70%	0.12
Total	5.00	85%	0.75

The best companies are using state-of-the art methods to lower their defect potentials, and coupling that with state-of-the-art methods for removing defects with high efficiency in excess of 95%. The results can be quite impressive.

COMPARING U.S. QUALITY DATA WITH INTERNATIONAL DATA

The author's company, Software Productivity Research, collects data on both productivity and quality in more than 20 countries. Although that may sound like quite a lot, it is still only a small and partial step toward a true global survey of software quality.

From the data collected, provisional averages on international quality levels were published in 1993 in the author's book, Software Productivity and Quality Today -- The Worldwide Perspective (Information Systems Management Group, Carlsbad, CA). Following are excerpts from some of the preliminary global findings, with some data revised during 1995 and 1996:

Table 3: International Comparisons of Defect Potentials and Defect Removal

Country	Defect Potential per Function Point	Defect Removal Efficiency Levels	Delivered Defects per Function Point
Japan	4.50	93%	0.32
Canada	4.55	86%	0.64
United States	5.00	85%	0.75
Norway	4.95	84%	0.79
Sweden	5.00	84%	0.80
France	4.75	83%	0.82
Italy	4.85	83%	0.82
India	5.10	84%	0.82

Germany	4.95	83%	0.84
England	4.85	82%	0.87
South Korea	5.20	83%	0.88
Russia	5.50	80%	1.10

The margin of error of this data is very high, except for the United States, and the information is presented primarily to generate discussion about the two key topics of defect potentials and defect removal efficiency levels.

Within every country where the author and his colleagues have collected data, the ranges of defect potentials and removal efficiencies are very broad. Some companies are achieving potentials of less than 2 defects per function point and eliminating more than 95%, while other companies have defect potentials approaching 10 per function point and eliminate barely 75%.

Although for every country, the range of performance is quite broad some six industries stand out internationally as achieving the best overall software quality levels:

Industries With Best Software Quality Results

1. Computer manufacturers
2. Telecommunication equipment manufacturers
3. Defense and weapons system manufacturers
4. Aerospace manufacturers
5. Medical equipment manufacturers
6. Commercial software manufacturers

Companies within these six industries typically average more that 95% in cumulative defect removal efficiency, which places them well above the norms of the 40 industries for which SPR has collected quality data.

Four characteristics set these industries apart from industries with less effective quality control approaches: 1) Usage of formal design and code inspections; 2) Usage of formal and active quality assurance functions; 3) Usage of trained testing specialists and formal testing departments; 4) Usage of a powerful suite of defect estimation, defect tracking, and other quality control tools.

A common characteristic of these industries in every country is that much of their software controls physical devices such as computers, switching systems, weapons systems, aircraft, and the like. The single exception is that of the commercial software vendors, and in this industry it has been learned by trial and error that poor quality loses business.

TOOLS AND METHODS USED BY BEST IN CLASS QUALITY PRODUCERS

There are major variances from company to company and country to country in the sets of tools and methodologies used to approach software quality. However, the best in class organizations have a common nucleus which includes these factors:

Quality Measurements

The most striking difference between leading organizations and lagging ones in every country is that, without exception, the leaders know their quality levels and user satisfaction levels because they measure these factors very carefully.

The quality measurements in leading companies vary slightly, but usually include these elements: 1) Software defect volumes are measured from requirements or design throughout the rest of the development cycle and into the field; 2) Defect severity levels are measured, ranging from serious through minor; 3) Defect origins are measured, so that problems with requirements, design, code, documents, and secondary problems are known.

This software quality data is collected on a daily basis, and then summarized at monthly, quarterly, and annual intervals to show trends over time. In addition, the leaders also measure user satisfaction, although the frequency of user surveys is normally once or twice a year.

Quality Methods

The leading companies did not become good overnight. Most of them have been engaged in software quality control work for 20 years or more. Therefore the leading companies have developed a set of proven methods that are known to work. These methods are sometimes defined under two headings, *defect prevention* and *defect removal*. Here are some examples: 1) Formal inspections of design, code, and other deliverables are used by essentially all software quality leaders since these activities are highly effective in both preventing and removing software defects; 2) Active and energetic software quality assurance groups, which may exceed 5% of total staff, are often found in the industry leaders.

A very interesting correlation is that in every country the best in class quality producers tend to utilize formal inspections of design, code, and other deliverables. Formal inspections are one of the few kinds of defect removal operation to exceed 60% in defect removal efficiency, and on average are about twice as efficient as any common form of testing. (High-volume external Beta testing by more than 1000 clients simultaneously is the only form of testing that is more efficient in defect removal than inspections.)

Both industry leaders and laggards test their software. The most striking difference between leaders and laggards is what the leaders do before testing begins. By means of defect prevention approaches such as Joint Application Design (JAD), Quality Function Deployment (QFD), formal inspections, and various flavors of structured analysis and design, the leaders usually have far fewer problems attributable to the front of their software development life cycles. Therefore when testing begins, the code developed by the leaders is substantially free from serious problems long before testing even starts. This translates into quicker testing cycles and fewer delays of final delivery.

Two important topics do not yet have any strong empirical correlations with software quality results: ISO 9000-9004 certification and the SEI capability maturity levels. Although the ISO standards are aimed at quality, they have not yet created any significant results within the software industry.

Indeed, as this report is being drafted a world wide web conference is on-going, hosted by John Seddon of the United Kingdom, to discuss whether or not ISO certification *degrades* quality rather than enhances it. In late 1996 a British "watch dog" government agency directed the British Standards Institute to stop making claims that ISO certification improved productivity or quality without empirical evidence to support the claims.

The SEI maturity level concept is also surprisingly ambiguous in terms of quality. There is a lot of overlap among the various SEI levels, and a surprising observation is that the worst software that is created by SEI level 3 organizations in terms of quality can lag the best software created by level 1 organizations.

However, some recent studies within the past two years do indicate an overall improvement in quality as SEI levels climb upward from level 1 to 3, 4, and 5. Unfortunately, the total number of samples is too small for statistical certainty.

Following are the current ranges of software defect potentials and removal efficiency levels observed from among client organizations that have utilized the SEI CMM:

Level 1 Quality: The software defect potentials noted from several hundred projects in Level 1 organizations run from about 3 to more than 15 defects per function point but average about 5.0 defects per function point. Defect removal efficiency runs from less than 70% to more than 95% but only averages about 85%. Thus the average number of delivered defects for Level 1 organizations is about 0.75 defects per function point.

Level 2 Quality: The software defect potentials noted from about 50 projects in Level 2 organizations run from about 3 to more than 12 defects per function point but average about 4.8 defects per function point. Defect removal efficiency runs from less than 70% to more than 96% but averages about 87%. Thus the average number of delivered defects for Level 2 organizations is about 0.6 defects per function point.

Level 3 Quality: The software defect potentials noted from about 30 projects in Level 3 organizations run from about 2.5 to more than 9 defects per function point but average about 4.3 defects per function point. Defect removal efficiency runs from less than 75%

to more than 97% but averages about 89%. Thus the average number of delivered defects for Level 3 organizations is about 0.47 defects per function point.

Level 4 Quality: The software defect potentials noted from 9 projects in Level 4 organizations run from about 2.3 to more than 6 defects per function point but average about 3.8 defects per function point. Defect removal efficiency runs from less than 80% to more than 99% but averages about 94%. Thus the average number of delivered defects for Level 4 organizations is about 0.2 defects per function point.

Level 5 Quality: The software defect potentials noted from 4 projects in a Level 5 organization ran from about 2 to 5 defects per function point but currently seem to average 3.5 defects per function point. Defect removal efficiency ran from less than 90% to more than 99% but averaged about 97%. Thus the average number of delivered defects for a Level 5 organization is about 0.1 defects per function point although there is obviously an insufficient sample at this level.

To illustrate the overlap of quality among the five levels of the SEI CMM, the following table shows our minimum, average, and maximum numbers of delivered defects per function point for each of the five CMM levels. Note that the best results from Level 1 are actually better than the worst results from Levels 3 and 4, even though the average results improve as the CMM ladder is climbed.

Table 4: Software Delivered Defects at Each Level of the SEI CMM

(Defects expressed in terms of defects per function point)

	Minimum	Average	Maximum
SEI Level 1	0.150	0.750	4.500
SEI Level 2	0.120	0.624	3.600
SEI Level 3	0.075	0.473	2.250
SEI Level 4	0.023	0.228	1.200
SEI Level 5	0.002	0.105	0.500

Although samples are small for the higher levels, there is now evidence from studies such as the ones carried out by Software Productivity Research (SPR) in 1994 which indicate that when organizations do move from CMM level 1 up to the higher levels their productivity and quality levels tend to improve, although there is quite a bit of overlap among the five CMM stages.

Quality Tools

What is easily the most visible difference between industry quality leaders and quality laggards is the set of tools available to the leaders, and totally absent from the lagging organizations. The leaders usually employ a set of quality tools that include some or all of the following: 1) Quality estimation predictive tools; 2) Defect and quality measurement tools; 3) Test planning tools; 4) Test coverage analysis tools; 5) Software reliability predictive models; 6) Complexity analysis tools; 7) Statistical analysis and reporting tools.

These tools have the general characteristic of putting quality in tangible, quantitative terms so that the underlying root causes can be explored and improved. The laggards tend to have no quantitative data, and hence are unable to take any kind of carefully planned corrective actions.

Since each of the quality tools cited in this section is roughly 1000 function points in size, it can be asserted that the leading quality assurance groups have in the range of 6000 to 8000 function points of quality-related tools available. By contrast, laggards with marginal quality levels often have less than 500 function points of quality-related tools, or even none at all.

Quality Culture

A final aspect which separates the laggards from the leaders is the culture of quality among the leaders, and its absence among the laggards. The word "culture" does not have a very precise definition, so in this context the meaning is the following: when visiting the industry leaders, almost everyone you talk to cares about quality and many of them also know something about it. When visiting the laggards, you tend to find some people who care about quality of course, and a few people who know how it might be achieved, but these quality-conscious people often feel isolated and even angry that their executives have no particular interest in the subject. There is no substitute for executive awareness of the importance of quality. When you meet an executive vice president or a CEO that can carry on a serious conversation about software quality, you can be fairly sure that the company is a pretty good one. When you visit a company where the executives know nothing of quality and give the appearance of not caring either, you can be fairly sure that the company will have some tough times ahead.

SUMMARY AND CONCLUSIONS

Now that the dimensions of software quality can be measured, it is obvious that there are two powerful sets of technologies which must both be deployed in order to be successful with software: 1) Defect prevention methods; 2) Enhanced defect removal methods.

The set of defect prevention methods includes all technologies which can simplify complexity, and minimize the tendency to make errors. Examples of software defect prevention methods include Joint Application Design (JAD), prototyping, structured methods, clean-room development, Information Engineering (IE), Object-Oriented methods (OO), Quality Function Deployment (QFD), and of course software quality measurement programs. Synergistic combinations of defect prevention methods can reduce defect potentials by more than 50% across the board, with the most notable improvements being in some of the most difficult problems, such as requirements errors.

The set of defect removal methods include structured walkthroughs, formal inspections, audits, independent verification and validation, and many forms of testing. Accurate measurement of defect removal efficiency has revealed some surprising findings. One surprise is that most forms of testing are less than 30% efficient in actually finding software problems, due in part to the fact that test cases are almost worthless for finding requirements errors, and not terribly effective in finding design errors. Against front-end requirements and design defects, formal inspections often achieve more than 60% defect removal efficiency rates.

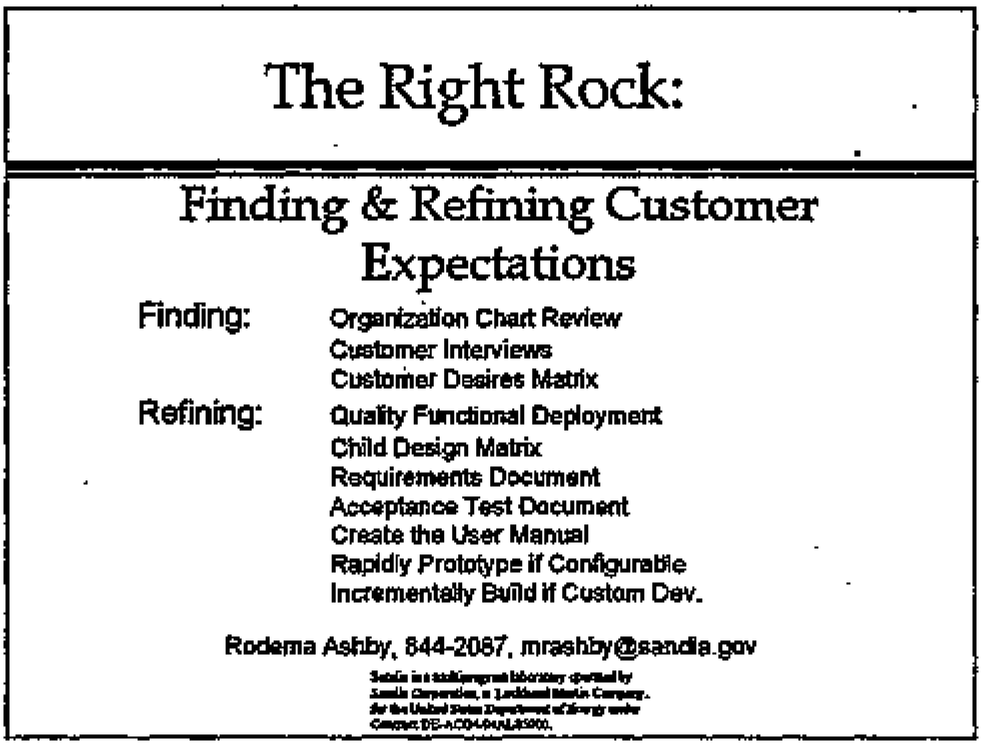
The "best in class" software producers now have defect potentials of less than 2.0 errors per Function Point, coupled with defect removal efficiencies that hover around 99% and may exceed it for mission-critical software. This combination yields delivered defect totals of only 0.02 defects per Function Point, or more than an order of magnitude better than U.S. norms and provisional international norms as well.

It can be hypothesized that international competition in the software domain will intensify as we move to the end of the 20th century. Since high levels of software quality are associated with high market shares, quality control is now a major topic of global competitiveness.

Session A1: Software Management

Chair Don Schilling
AS/FM&T

Session : Paper #	Author(s)	*Title*
A1:1	Rodema Ashby Sandia National Laboratories	*The Right Rock: Finding/Refining Customer Expectations*
A1:2	David Harris Sandia National Laboratories	*TCAMS Lessons Learned*
A1:3	Joe Schofield Sandia National Laboratories	*The Next Silver Bullet - Or Just Another Shot in the Foot?*



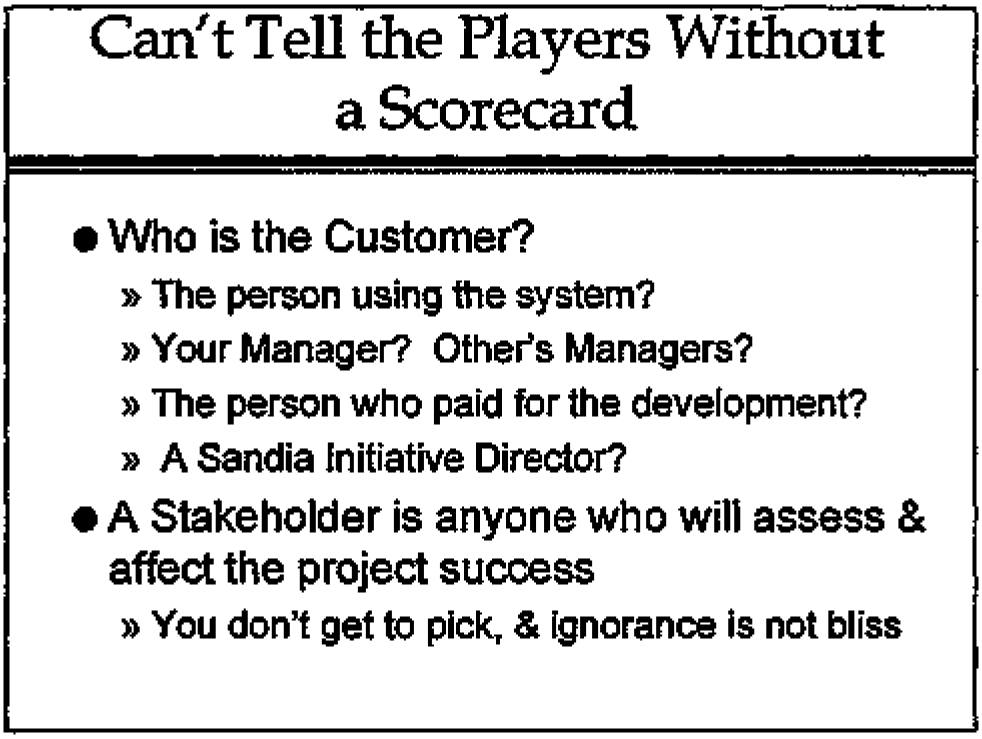
The Right Rock:

Finding & Refining Customer Expectations

Finding:	Organization Chart Review Customer Interviews Customer Desires Matrix
Refining:	Quality Functional Deployment Child Design Matrix Requirements Document Acceptance Test Document Create the User Manual Rapidly Prototype if Configurable Incrementally Build if Custom Dev.

Rodema Ashby, 844-2087, mrashby@sandia.gov

Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under
Contract DE-AC04-94AL85000.



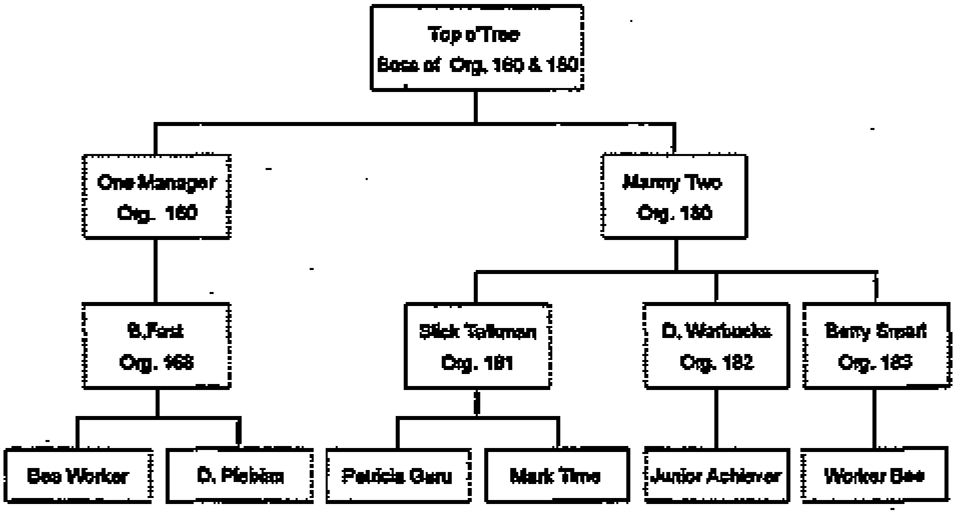
Can't Tell the Players Without a Scorecard

- Who is the Customer?
  - » The person using the system?
  - » Your Manager? Other's Managers?
  - » The person who paid for the development?
  - » A Sandia Initiative Director?
- A Stakeholder is anyone who will assess & affect the project success
  - » You don't get to pick, & ignorance is not bliss

Goal: Figure out the Politics, as best you can

- Draw an organizational chart, with everyone involved in the project.
- If there are other companies involved you need to chart that organization(s) players also.
- Review and assess the Players:
  - » Who is powerful,listened to, gets their way?
  - » Who could ruin your career?
  - » Who has money and interest?
  - » Who wants the Project to Succeed?
  - » Who wants their pet technology used...

Organizational Chart Example



Top o'Tree
Boss of Org. 160 & 180

One Manager
Org. 160

Manny Two
Org. 180

S.First
Org. 168

Slick Talkman
Org. 181

D. Warbucks
Org. 182

Barry Smart
Org. 183

Bee Worker

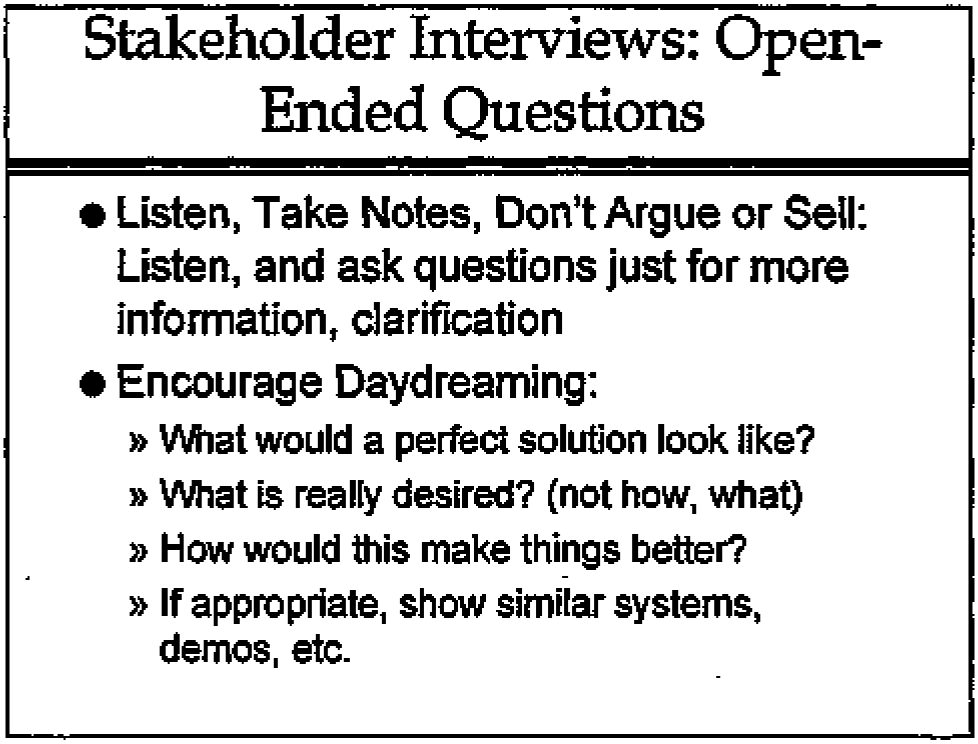
D. Plebian

Patricia Guru

Mark Time

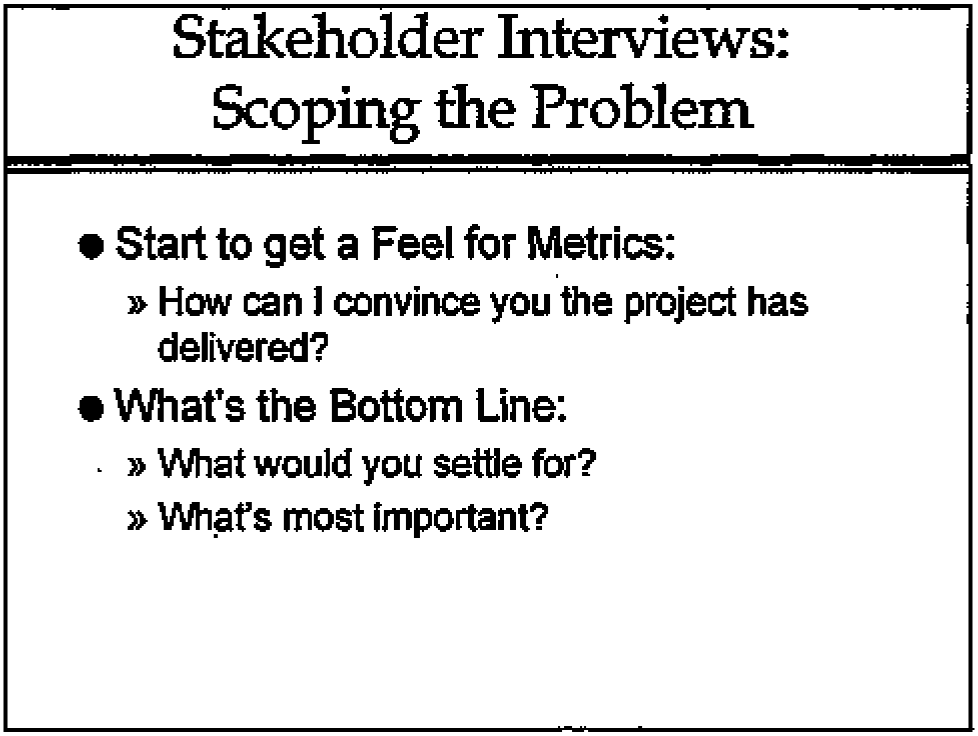
Junior Achiever

Worker Bee



Stakeholder Interviews: Open-Ended Questions

- Listen, Take Notes, Don't Argue or Sell: Listen, and ask questions just for more information, clarification
- Encourage Daydreaming:
  - » What would a perfect solution look like?
  - » What is really desired? (not how, what)
  - » How would this make things better?
  - » If appropriate, show similar systems, demos, etc.



Stakeholder Interviews: Scoping the Problem

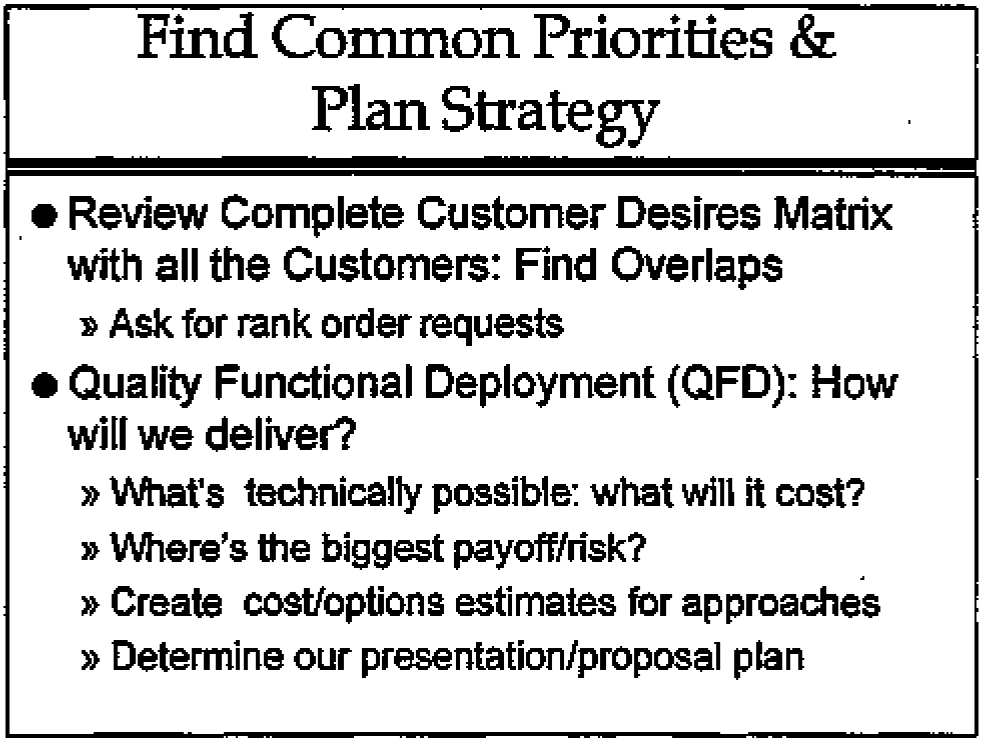
- Start to get a Feel for Metrics:
  - » How can I convince you the project has delivered?
- What's the Bottom Line:
  - » What would you settle for?
  - » What's most important?

Creating Order out of Chaos:
Matrix of Customer Desires

- Brainstorm with customer group if available
- If there are customers with very different needs, create a list of desires for each customer from your interview notes
- Create a Customer's desires matrix, noting who cares most about what

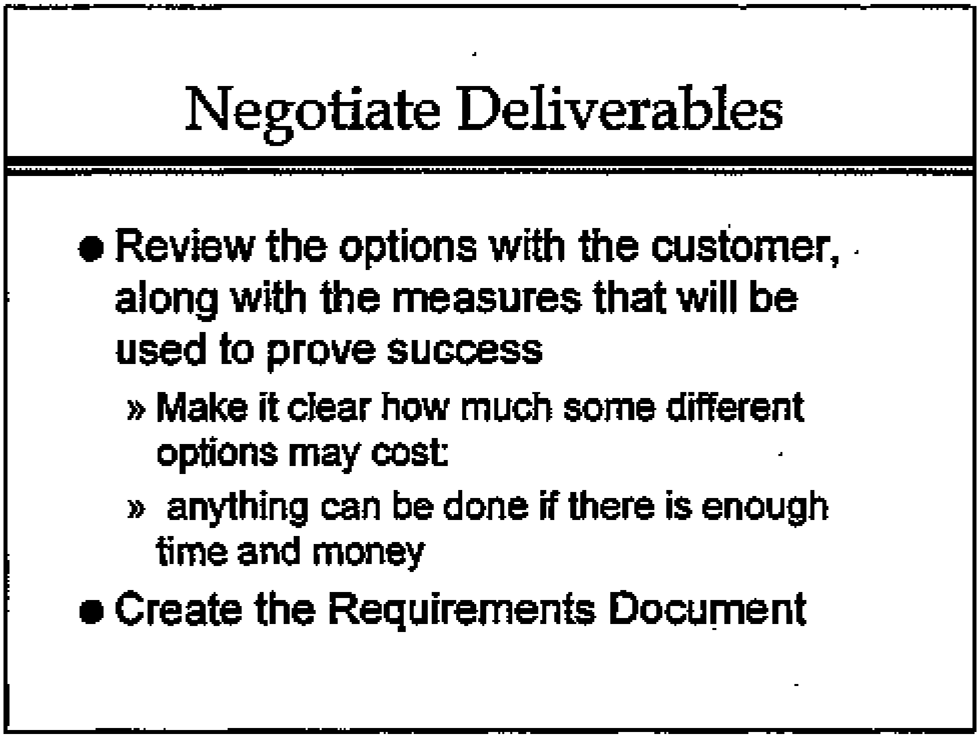
Document Customer Desires as Measurable Objectives

- Example: Instead of "User Friendly":
  - » "Novice can use the system to do x after 30 minutes of training"
  - » "Users with more than 1 hour's experience make less than 1 error per12 major operations as described in the Acceptance Test



Find Common Priorities & Plan Strategy

- Review Complete Customer Desires Matrix with all the Customers: Find Overlaps
  - » Ask for rank order requests
- Quality Functional Deployment (QFD): How will we deliver?
  - » What's technically possible: what will it cost?
  - » Where's the biggest payoff/risk?
  - » Create cost/options estimates for approaches
  - » Determine our presentation/proposal plan



Negotiate Deliverables

- Review the options with the customer, along with the measures that will be used to prove success
  - » Make it clear how much some different options may cost:
  - » anything can be done if there is enough time and money
- Create the Requirements Document

Write the Acceptance Test Before Development Starts

- Write the Requirements Document
- Write out the Acceptance Test criteria for each requirement:
  - » this defines exactly how the requirements will be measured
- Review & Renegotiate the Requirements and the AcceptanceTest Doc. with the Customer
- Create a Detailed System Test in the general design phase as implementation details arise

The User Interface is Defined/Refined during Proposal

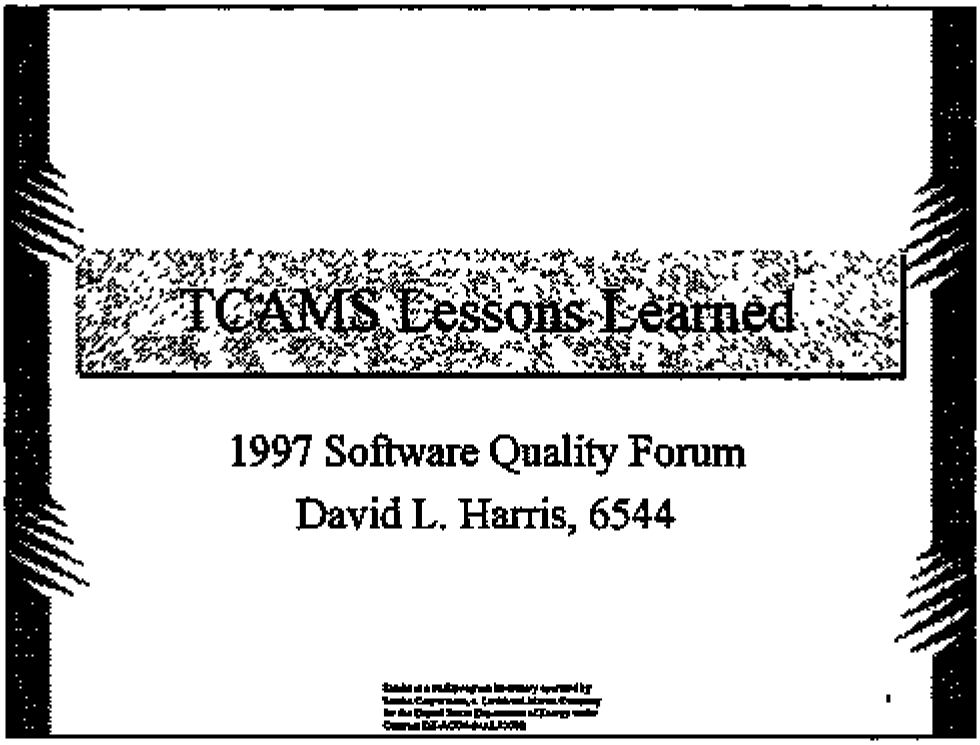
- Prototype and review the initial user interface quickly (Reusable code?)
- Use the people who will actually be using the system for the user testing:
  - » They become champions for it's acceptance
  - » They know their jobs, and how it will be used
- Complete the User Manual before coding the User Interface: It's the Requirements Document & Acceptance Test for the UI

Rapidly Prototype the whole system if possible

- Reusing a configurable system increases robustness and cuts development time
- Demonstrate and Modify System as Requirements are renegotiated.
- If New Development, Build Incrementally
- Structure the Project with Many Milestones: coordinate incremental changes to deliver new functionality

Summary: Listen, Document, Review, Update

- Find out who the customers really are: Organization Chart Review
- Find out what the customers want: Customer Interviews
- Figure out what the project needs to deliver: Customer Desires
- Figure out how the project will deliver: QFD, Proj. Plan, Child Design Matrix
- Document how we'll know we delivered: Acceptance Test, User Interface Manual & Milestone Reviews as the Project is Implemented

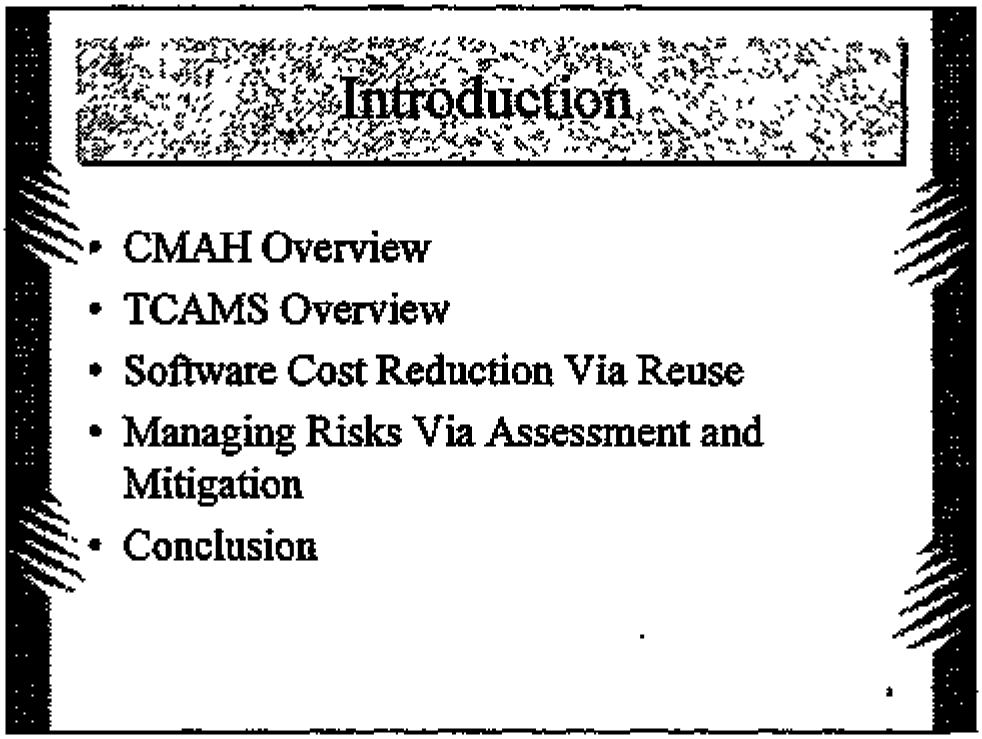


TCAMS Lessons Learned

1997 Software Quality Forum

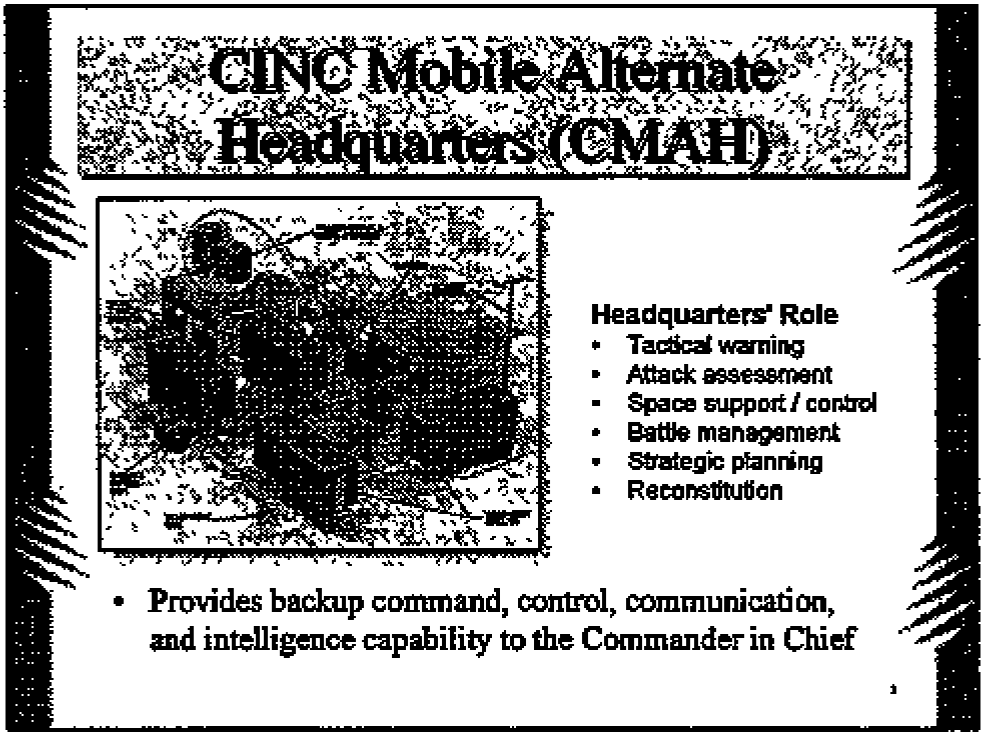
David L. Harris, 6544

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company for the United States Department of Energy under Contract DE-AC04-94AL85000



Introduction

- CMAH Overview
- TCAMS Overview
- Software Cost Reduction Via Reuse
- Managing Risks Via Assessment and Mitigation
- Conclusion



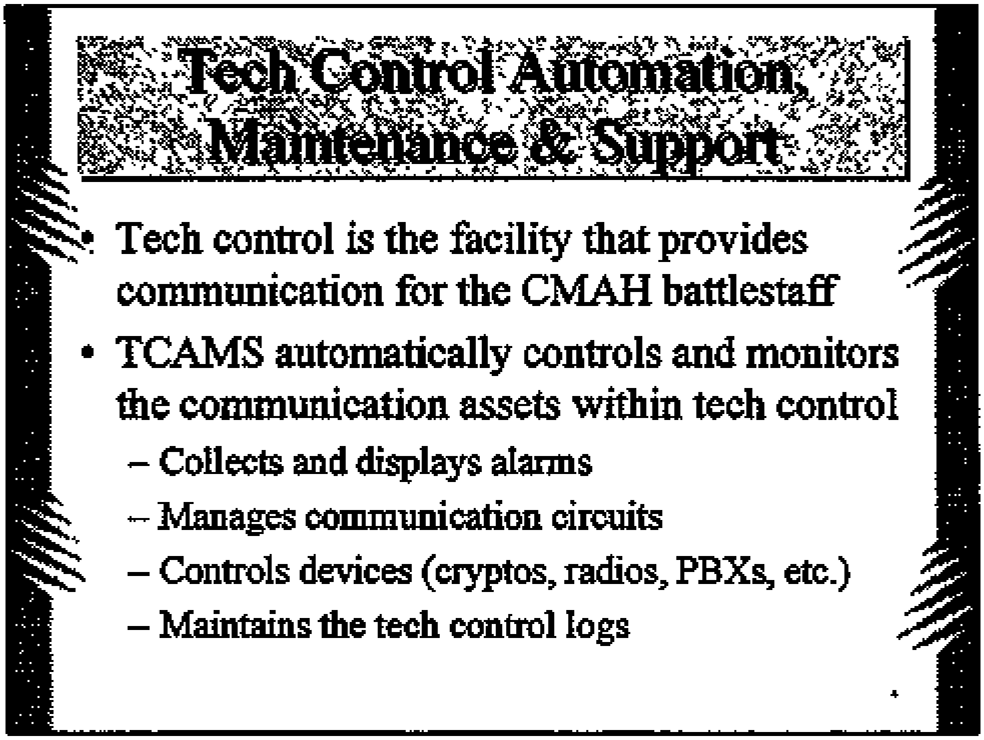
CINC Mobile Alternate Headquarters (CMAH)

Headquarters' Role

- Tactical warning
- Attack assessment
- Space support / control
- Battle management
- Strategic planning
- Reconstitution

- Provides backup command, control, communication, and intelligence capability to the Commander in Chief

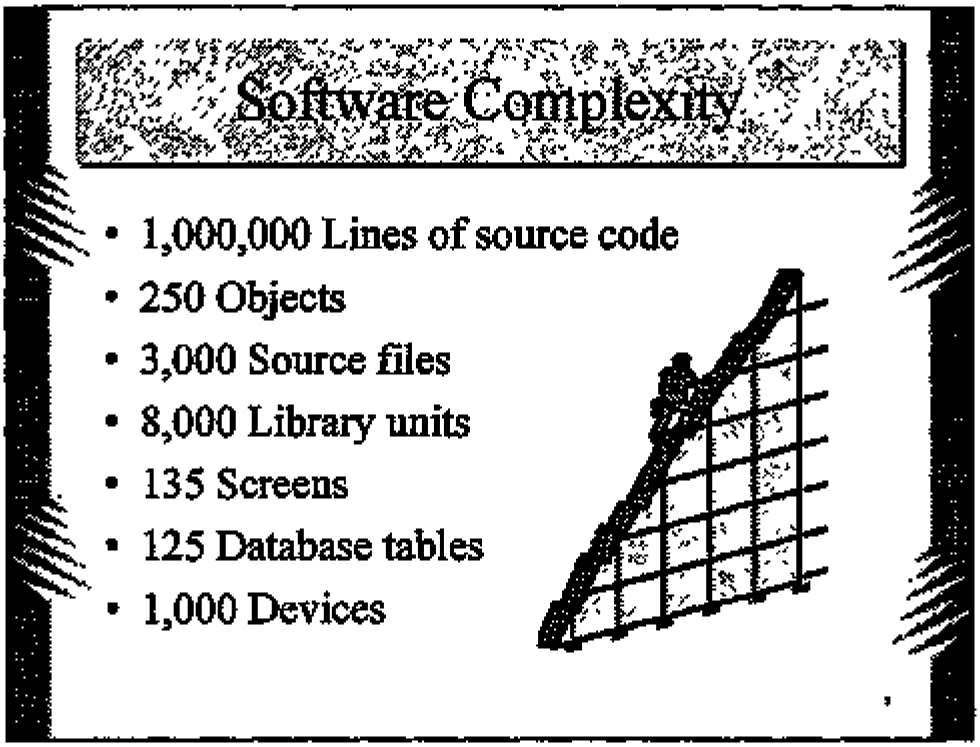
3



Tech Control Automation, Maintenance & Support

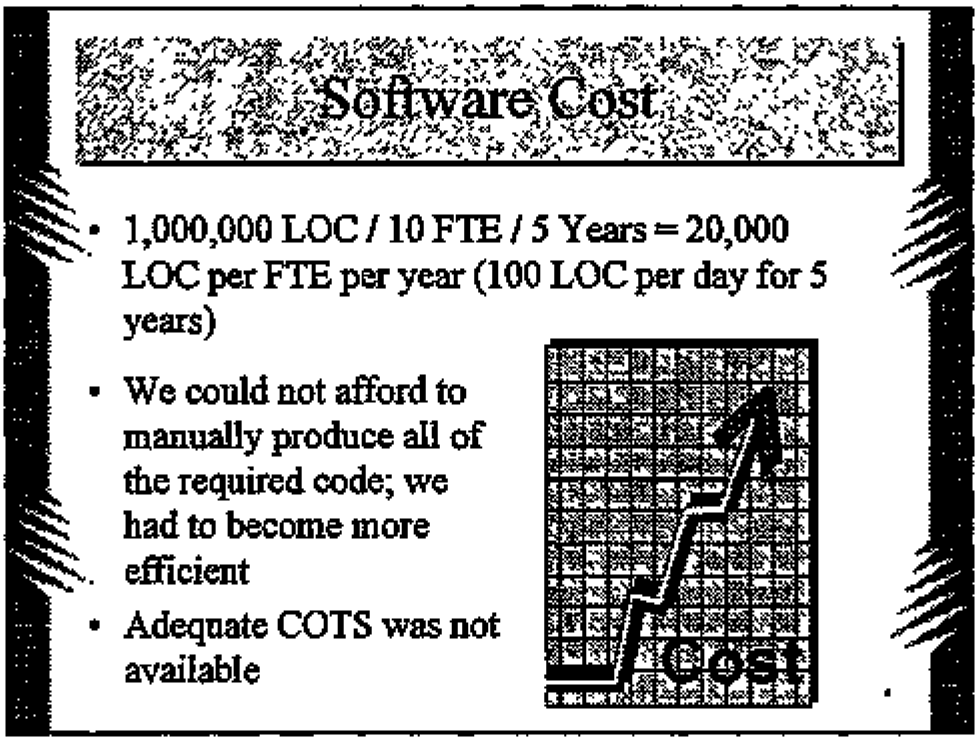
- Tech control is the facility that provides communication for the CMAH battlestaff
- TCAMS automatically controls and monitors the communication assets within tech control
  - Collects and displays alarms
  - Manages communication circuits
  - Controls devices (cryptos, radios, PBXs, etc.)
  - Maintains the tech control logs

4



Software Complexity

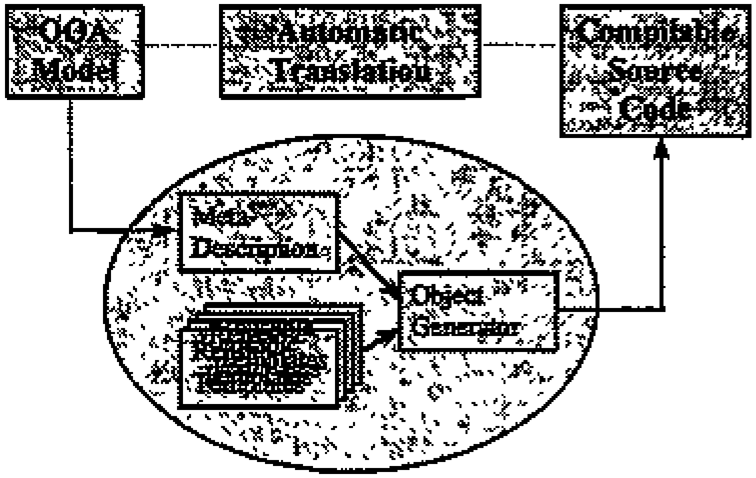
- 1,000,000 Lines of source code
- 250 Objects
- 3,000 Source files
- 8,000 Library units
- 135 Screens
- 125 Database tables
- 1,000 Devices



Software Cost

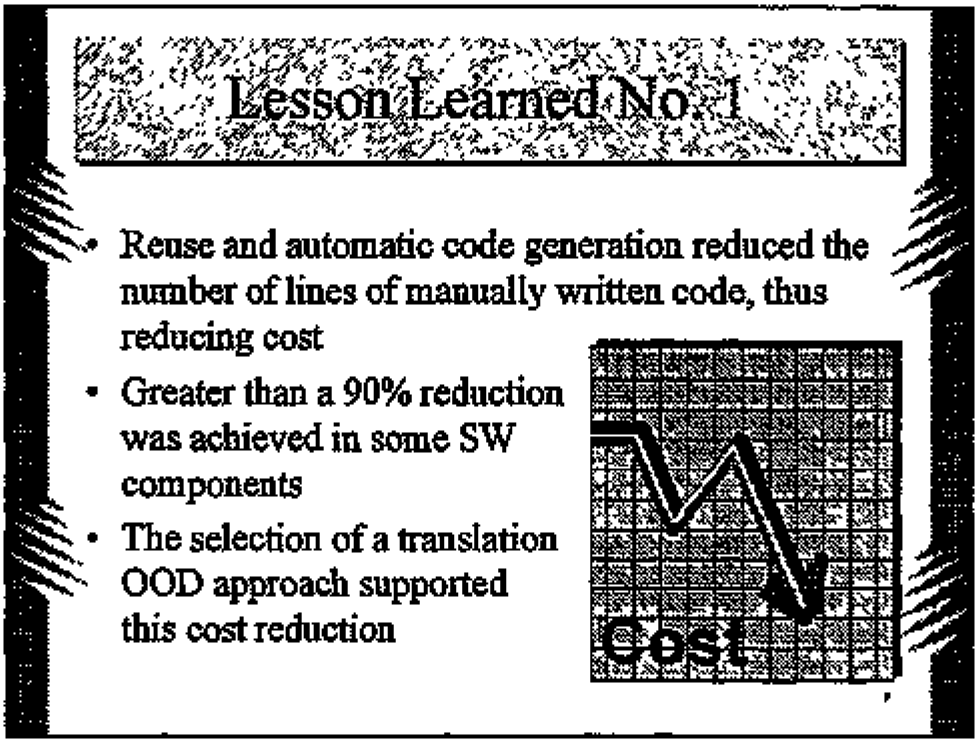
- 1,000,000 LOC / 10 FTE / 5 Years = 20,000 LOC per FTE per year (100 LOC per day for 5 years)
- We could not afford to manually produce all of the required code; we had to become more efficient
- Adequate COTS was not available

Achieving 90% Cost Reduction



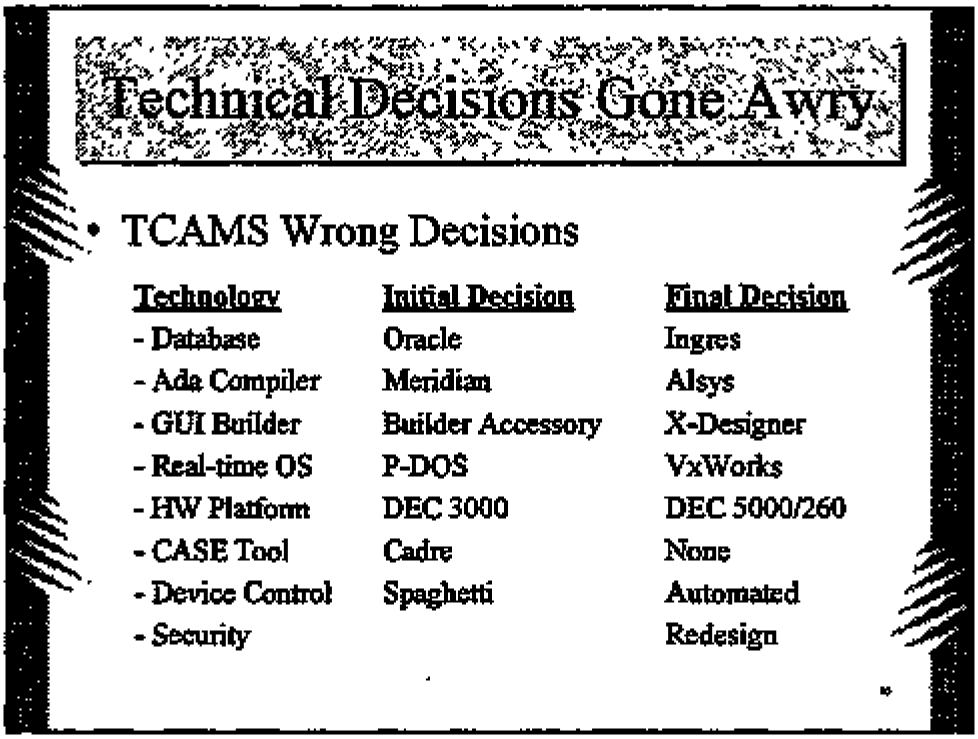
Cost Reduced Through Reuse

- Design decisions are made once, embodied within the meta-files, templates, & object generator rules, and reused for each object
- Source code templates are written once and reused for each object
- A translation (Shlaer/Mellor) vs elaboration (Rumbaugh) OOD approach was essential in achieving the large per cent of reuse



Lesson Learned No. 1

- Reuse and automatic code generation reduced the number of lines of manually written code, thus reducing cost
- Greater than a 90% reduction was achieved in some SW components
- The selection of a translation OOD approach supported this cost reduction



Technical Decisions Gone Awry

- TCAMS Wrong Decisions

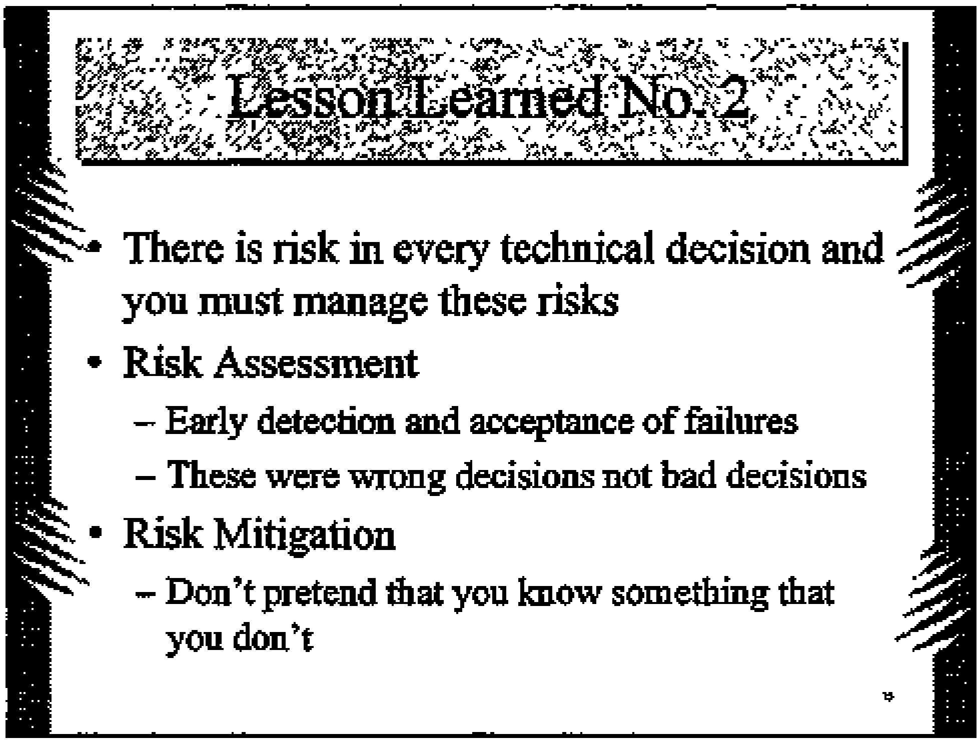
Technology	Initial Decision	Final Decision
- Database	Oracle	Ingres
- Ada Compiler	Meridian	Alsys
- GUI Builder	Builder Accessory	X-Designer
- Real-time OS	P-DOS	VxWorks
- HW Platform	DEC 3000	DEC 5000/260
- CASE Tool	Cadre	None
- Device Control	Spaghetti	Automated
- Security		Redesign

Reasons for Wrong Decisions

- Uncertainty and risk due to the lack of adequate knowledge
  - Try a technology to find out its characteristics
  - Lowest bid wasn't the lowest cost
  - Newer technology became available
  - Planned hardware upgrades
  - Changes in customer requirements

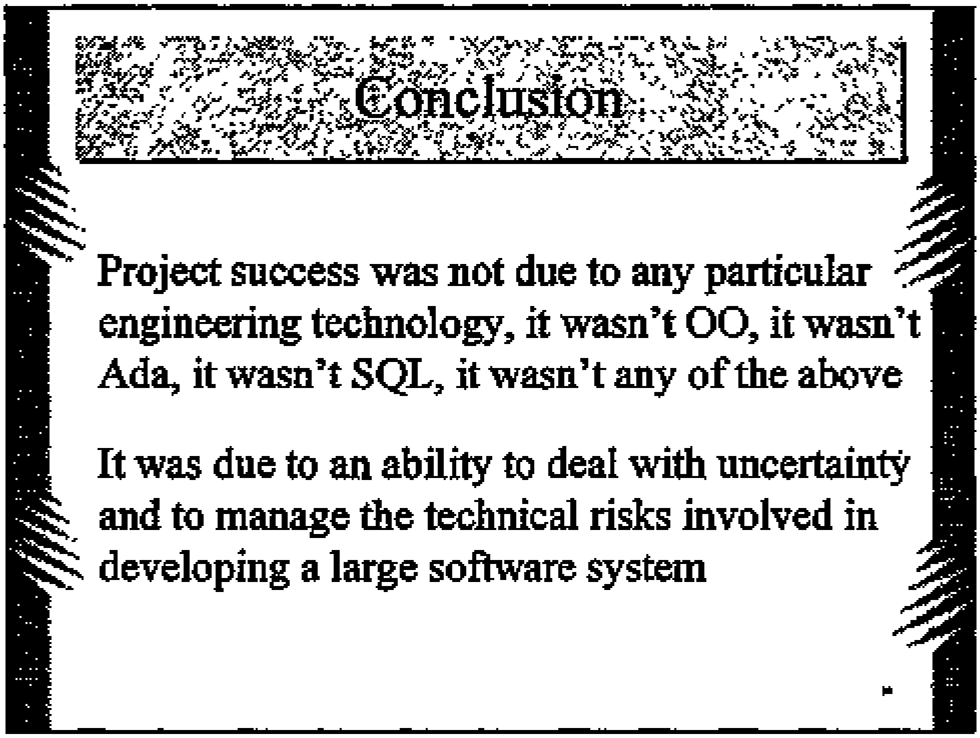
Analysis

- Each of these initial technical decisions was based on sound engineering analysis
- Things just didn't turn out as expected
- This is exactly what happens in all large software projects



Lesson Learned No. 2

- There is risk in every technical decision and you must manage these risks
- Risk Assessment
  - Early detection and acceptance of failures
  - These were wrong decisions not bad decisions
- Risk Mitigation
  - Don't pretend that you know something that you don't



Conclusion

Project success was not due to any particular engineering technology, it wasn't OO, it wasn't Ada, it wasn't SQL, it wasn't any of the above

It was due to an ability to deal with uncertainty and to manage the technical risks involved in developing a large software system

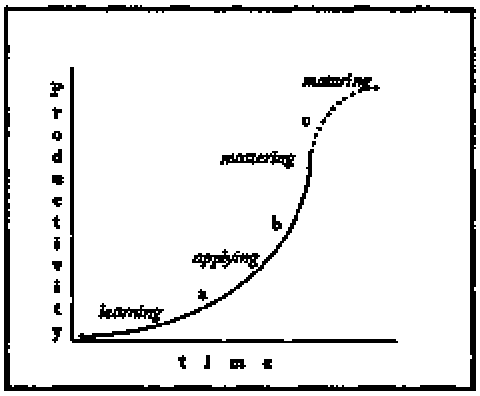
The Next Silver Bullet
(Or Just Another Shot in the Foot?)

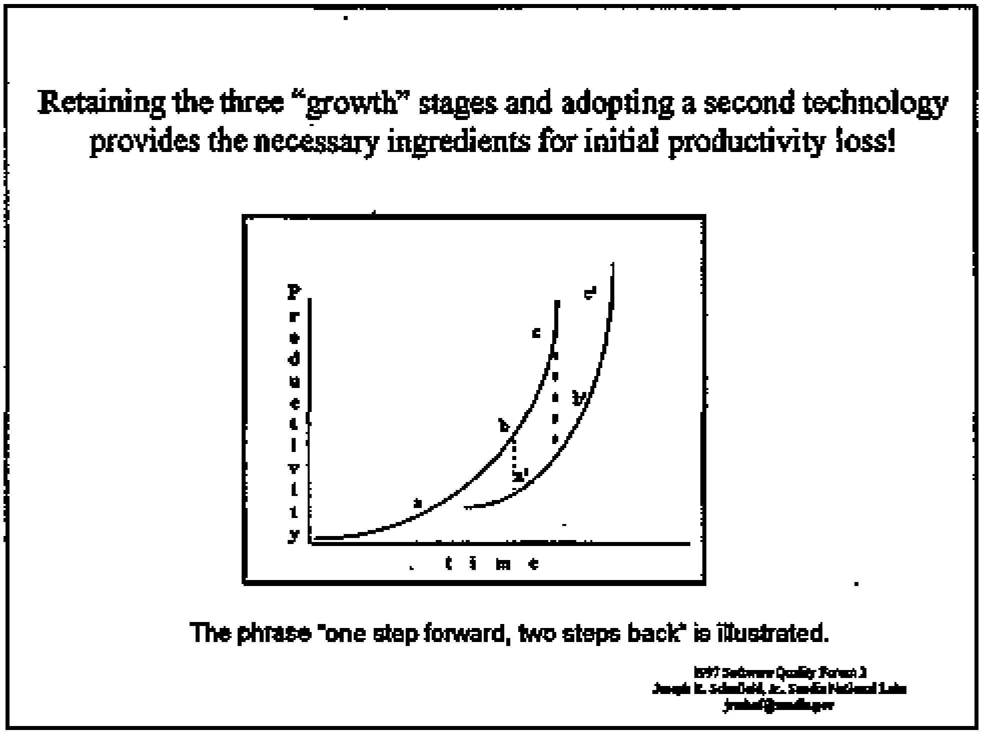
Joseph R. Schofield, Jr.,

Sandia National Laboratories

This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000. The full copyrighted version of this material is available through Applied Computer Research, Inc. (602) [illegible], *Managing System Development*, volume 15, number 8.

The productivity associated with the introduction of new technology can be depicted in four simple stages

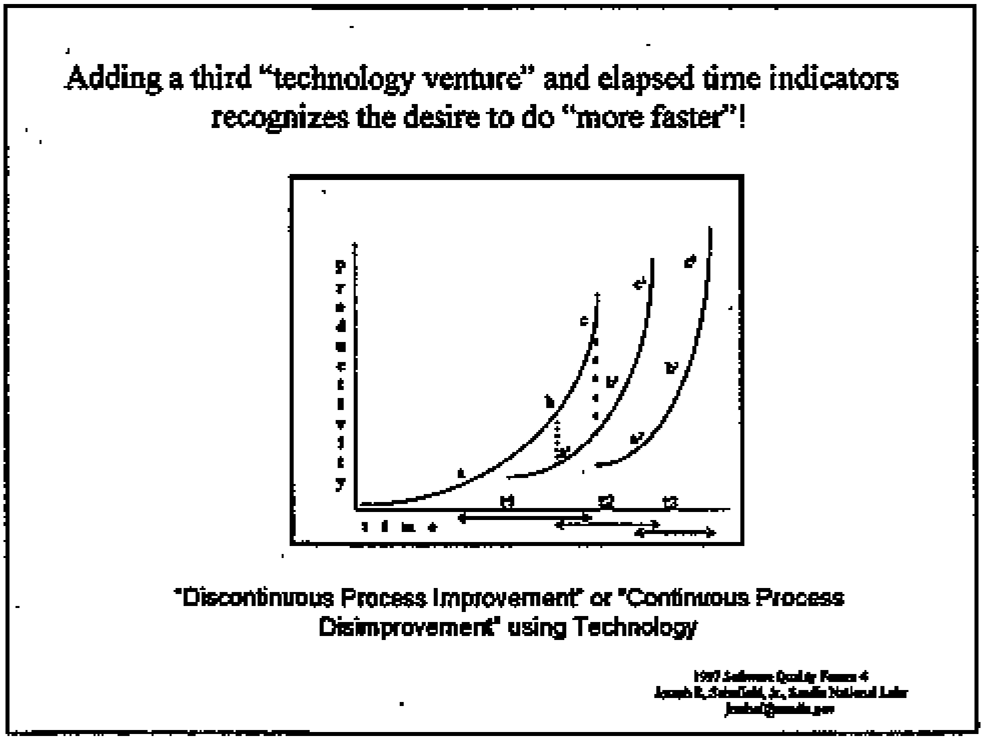




Retaining the three "growth" stages and adopting a second technology provides the necessary ingredients for initial productivity loss!

The phrase "one step forward, two steps back" is illustrated.

1997 Software Quality Forum 3
Joseph R. Schofield, Jr., Sandia National Labs
jrschof@sandia.gov

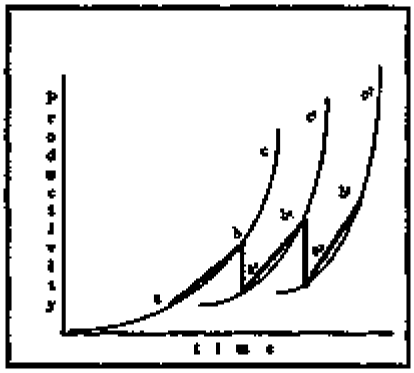


Adding a third "technology venture" and elapsed time indicators recognizes the desire to do "more faster"!

"Discontinuous Process Improvement" or "Continuous Process Disimprovement" using Technology

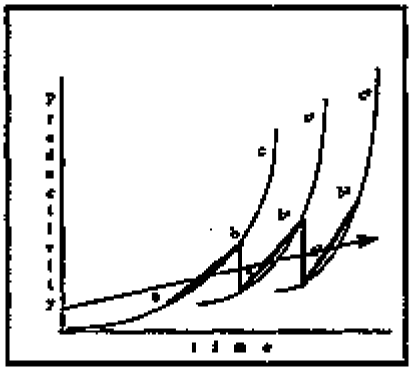
1997 Software Quality Forum 4
Joseph R. Schofield, Jr., Sandia National Labs
jrschof@sandia.gov

The quest for productivity has demonstrable ups and downs.



We're bleeding now - but expected productivity leaps lag still.

What's it all mean?



Options include:

Will be described in the session!

Session B1: Software Testing

Chair Larry Rodin
Pantex Plant

Session : Paper #	Author(s)	*Title*
B1:1	Debra Sparkman Lawrence Livermore National Laboratory	*A Working Testing Process*
B1:2	Nancy Storch Lawrence Livermore National Laboratory	*Testing the Design and Operations of a New Badging System*
B1:3	Dwayne Knirk Sandia National Laboratories	*Establishing a Three-Way Agreement: Specification, Code, Test*

1997 Software Quality Forum

A Working Testing Process

Debra Sparkman
Lawrence Livermore National Laboratory

April 1997

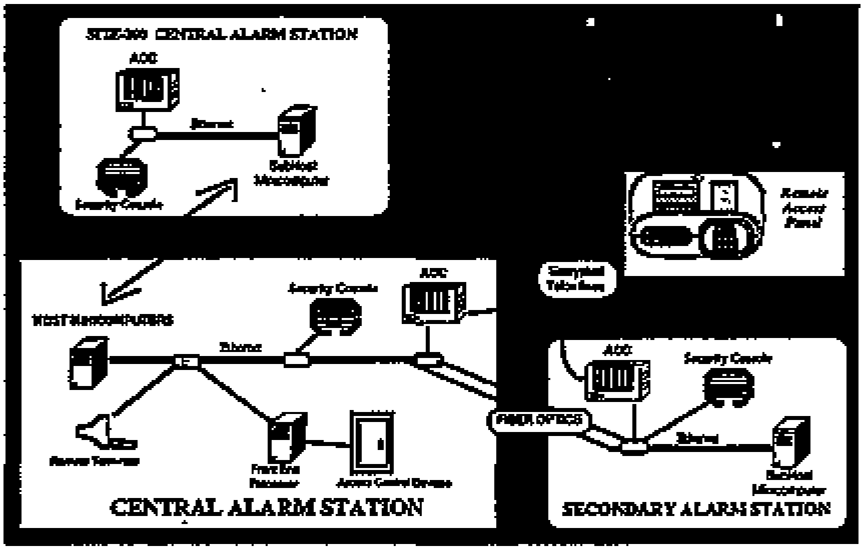
This work was performed under the US Department of Energy by Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

Argus Overview

- □ Automated Security System
- □ Intrusion Detection
- □ Access Control
- □ Dispatch Center

Argus Overview

Argus



Developer Testing

Argus

- ❑ Unit/Package Testing
  - ❑ Ada test packages for shared modules
- ❑ Integration & System
  - ❑ Performed on development system using "mock" utilities

Independent System Testing

Argus

- ❑ Conducted on separate system
  - ❑ Based upon configuration of all customer sites
  - ❑ Physical equipment in most cases
  - ❑ Flexibility to configure system to allow parallel testing for different sites
- ❑ Focus on regression testing and new major feature testing
  - ❑ manual testing
  - ❑ repeatable

Lawrence Livermore National Laboratory

Independent System Testing cont.

Argus

- ❑ Test planning based on priorities
- ❑ Time allows, perform special feature and defect correction testing
- ❑ Test anomalies tracked and reviewed by Test Leader
- ❑ Test summary
- ❑ Major coordination efforts, frequent meetings with development staff
- ❑. Metrics collection of cumulative failure profile.

Lawrence Livermore National Laboratory

Testing Process Tools

Argus

- ❑ Test procedures priority & pass/fail log
- ❑ Test incident report
- ❑ Test sequence log

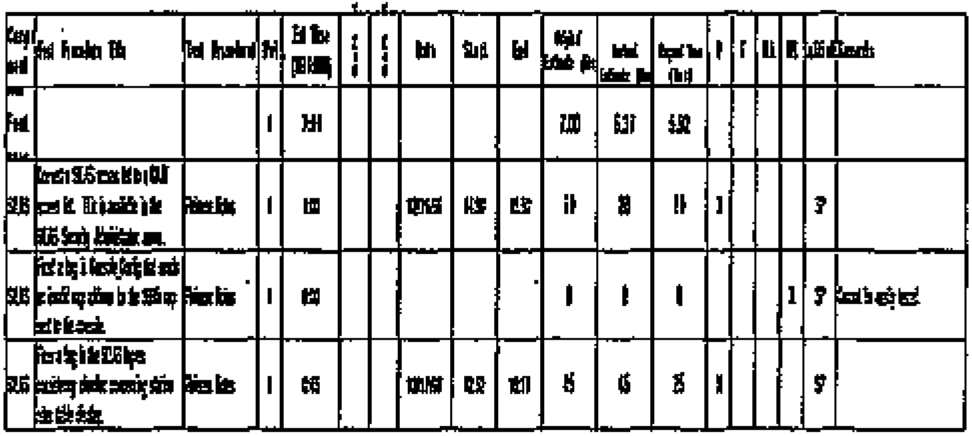
Test Procedure Priority & Pass/Fail Log

Argus

- ❑ Excel spreadsheet
- ❑ Testers input data during testing
  - ❑ start & stop times
  - ❑ pass or fail status
  - ❑ initials
  - ❑ comments
- ❑ Automatically calculated fields
  - ❑ duration
- ❑ Status reporting
  - ❑ testing completed
  - ❑ testing to be completed

Excel Spreadsheet

Argus



[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]			1	[illegible]						7.00	6.37	5.92						
[illegible]	[illegible]	[illegible]	1	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1				[illegible]	
[illegible]	[illegible]	[illegible]	1	[illegible]						[illegible]	[illegible]	[illegible]				1	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	1	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				[illegible]	

Testing Process Tools

Argus

- ☐ Test procedures priority & pass/fail log
- ☐ Test incident report
- ☐ Test sequence log

Test Incident Report

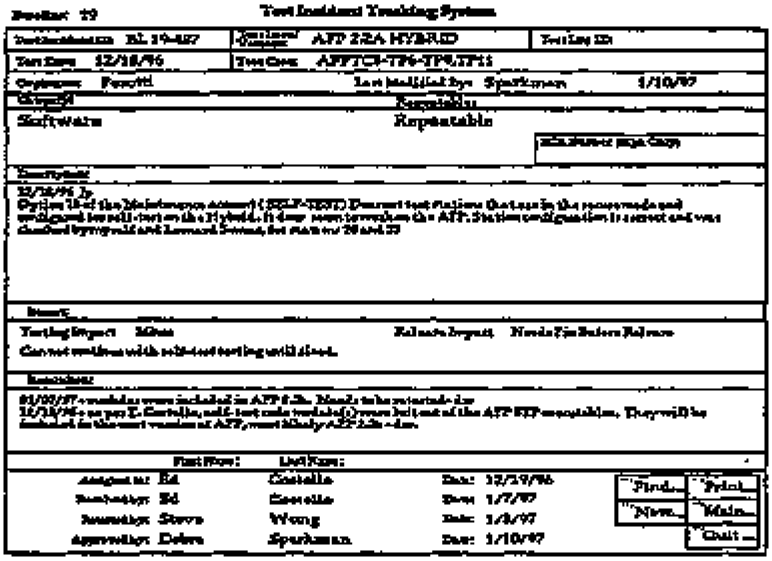
- □ FileMaker Pro Application
- □ Real-time defect reporting
- □ All defects are collected
    - □ Software
    - □ Hardware
    - □ Test Process (tester error, test procedure defects)
    - □ Test Configuration (test system specific data)

Test Incident Report cont.

- □ Two Impact Categorizations
    - □ Testing impact
    - □ Release impact
- □ Status & Approval Signatures
    - □ Assigned to
    - □ Resolved by
    - □ Retested by
    - □ Approved by

Test Incident Tracking System

Argus



Test Incident Tracking System

[illegible] 19

[illegible] RL 19-497	APP 22A HYBRID	Test Log ID:
Test Date 12/18/96	Test Case APPTCS-TP6-TP8,TP11	

[illegible] Fossett Last Modified by: Sparkman 1/10/97

[illegible] [illegible]

Software Repeatable

[illegible]

12/18/96 [illegible]

[illegible]

Testing Impact Minor Release Impact Needs Fix Before Release

Can not continue with self-test testing until fixed.

[illegible]

[illegible]

First Name: Last Name:

Assigned to:	Ed	Costello	Date:	12/19/96
[illegible]	Ed	Costello	Date:	1/7/97
[illegible]	Steve	Wong	Date:	1/8/97
Approved by:	Debra	Sparkman	Date:	1/10/97

Testing Process Tools

Argus

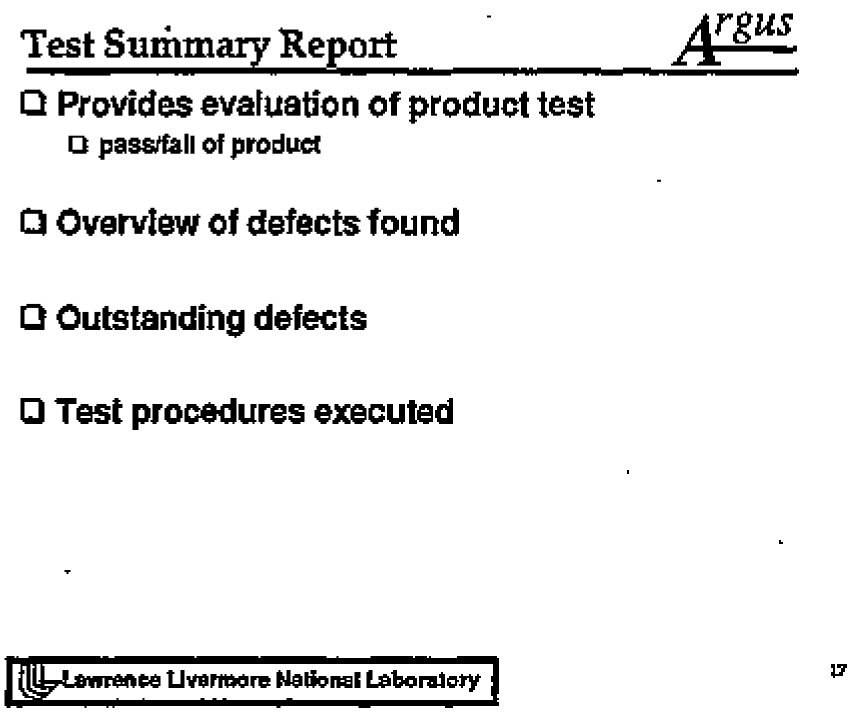
- ❑ Test procedures priority & pass/fail log
- ❑ Test incident report
- ❑ Test sequence log

Test Sequence Log

- ❑ Provides a sequence of activities
- ❑ Global viewpoint of product testing
- ❑ Logs software products version & date identifiers

Test Sequence Log

Date and Time	Test Procedure Id	Comments
961210, 14:48		Began preparation for testing
961213, 15:15	Test System Software Configuration	1. AFP 2.3a, 4-Dec, 16:07:22, checksum 25c6 (AFP_[illegible]) (Master AFP) 2. AFP 2.1b, 30-Oct, 22:43, checksum 0de9 (outboard_slave) (outboard AFP) 3. AFP 2.2a, 4 Dec95, 15:17:45, checksum e0ed (Hybrid) 4. Argus VAX Tools 4.4 (4 Dec, 12:34) 5. Argus Tools 4.4, 10-dec-1995 10:00 6. BLAS #3 Host 4.5, 15-Nov, 18:31 7. Firmway 1.1, 27-Aug, 10:56 8. CAIN 19.07 dated 12-dec-1995, 17:18. 9. Console 2.0a, 11-Nov, 16:29 10. MPO 3.04, 4-Dec-1995, 11:36 11. BLAS Host 19.7, 21-Nov, 08:29 12. CMU 2.9, 11-Sep, 11:34 13. CCTV Server 1.7, 21-Dec-1995, 15:52 14. Phonebook Server 1.5, 22-Dec, 10:13 15. Time Code 1.2, 22-Dec-1995, 08:39 16. VAU 1.4, 13-Feb, 10:49
961213, 15:15	Test box/sensor rack wiring	Test box/sensor rack wiring: [(Hybrid, sensor rack), (AFP, [illegible] test box), (RTU [illegible] test box)].
961213, 15:30		Checked system functionality.
961213, 15:25		Suspended testing preparation for the day
961214, 07:30		Resumed preparation for testing.
961215, 08:00		Began testing
961215, 12:10		Suspended testing
961214, 14:30		Resumed testing
961216, 15:15		Need to review results of tests with Rick and Bruce before [illegible] several of Release Notes tests passed/failed.
961216, 15:20		Suspended testing for the day.



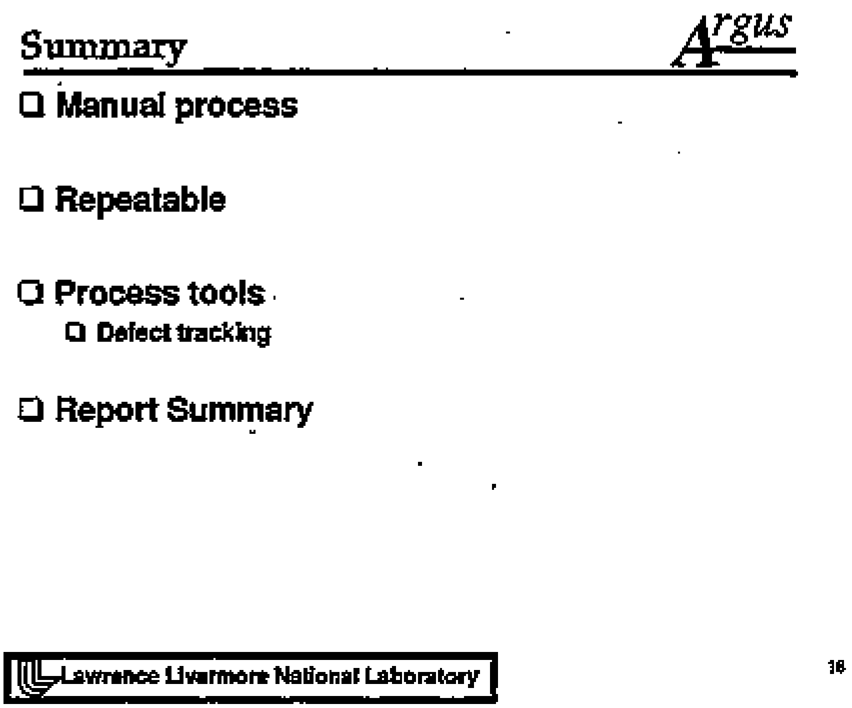
Test Summary Report

Argus

- ❑ Provides evaluation of product test
  - ❑ pass/fail of product
- ❑ Overview of defects found
- ❑ Outstanding defects
- ❑ Test procedures executed

Lawrence Livermore National Laboratory

17



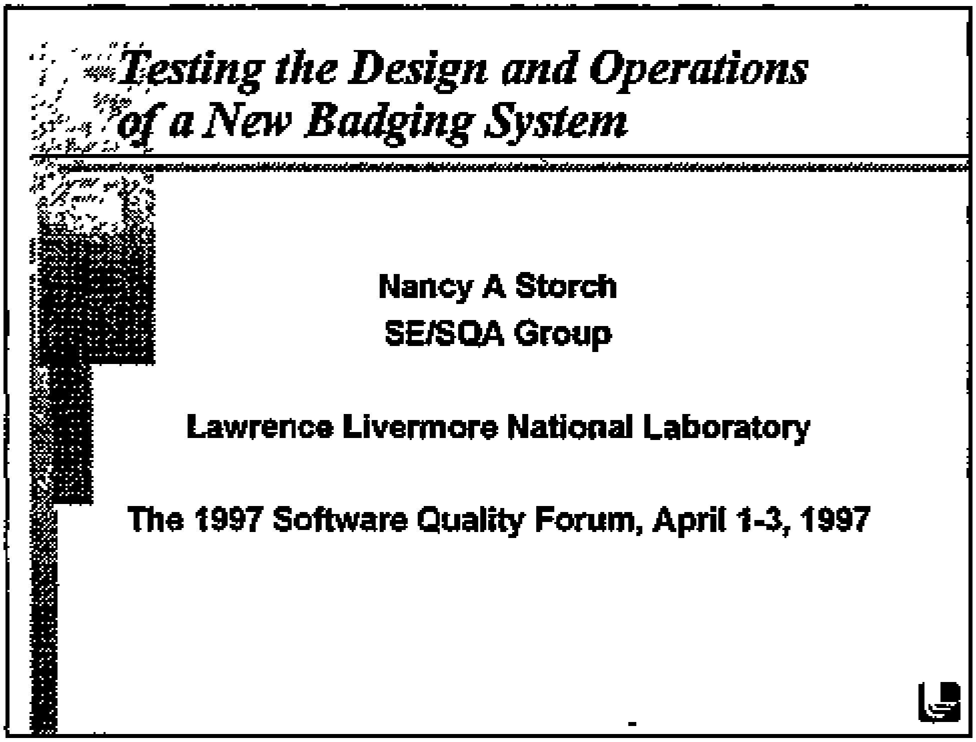
Summary

Argus

- ❑ Manual process
- ❑ Repeatable
- ❑ Process tools
  - ❑ Defect tracking
- ❑ Report Summary

Lawrence Livermore National Laboratory

18

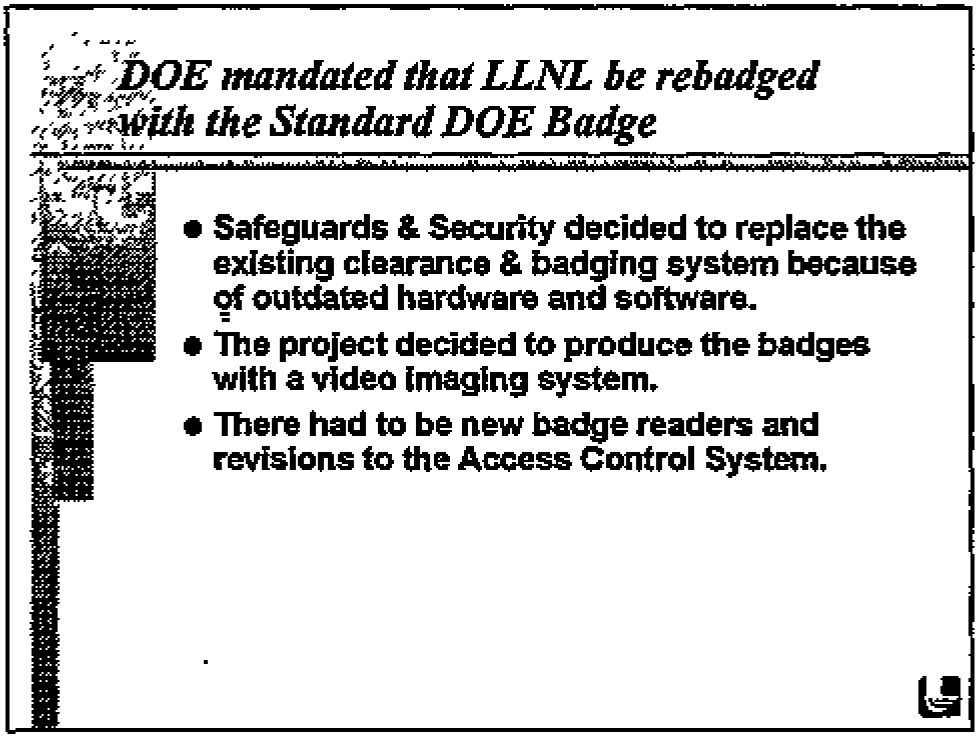


Testing the Design and Operations of a New Badging System

Nancy A Storch
SE/SQA Group

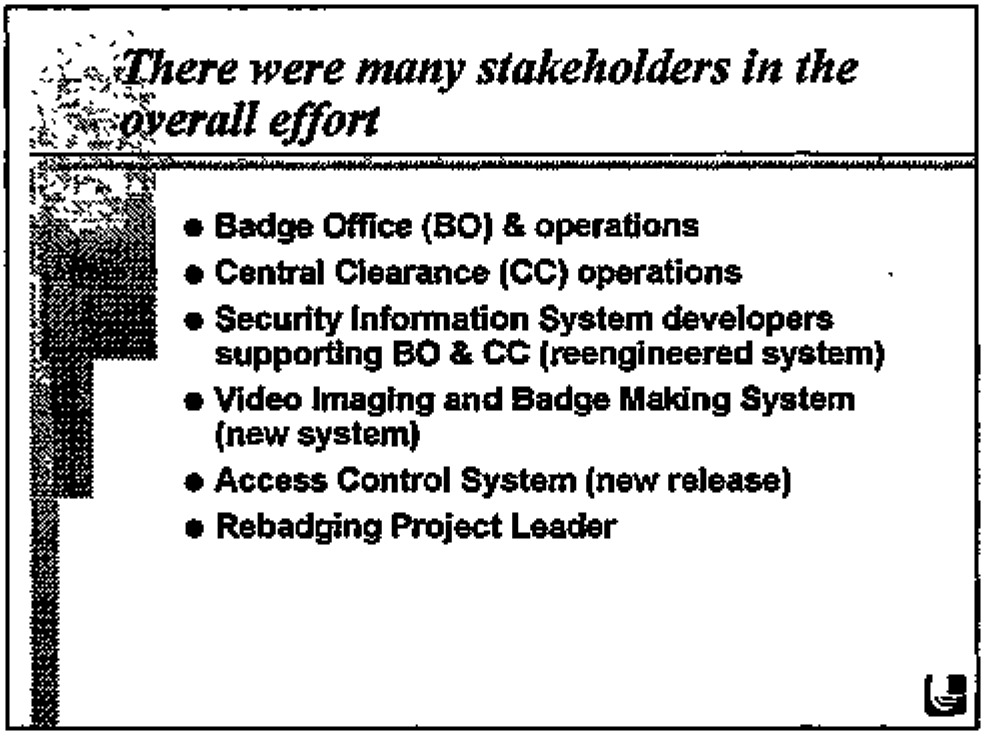
Lawrence Livermore National Laboratory

The 1997 Software Quality Forum, April 1-3, 1997



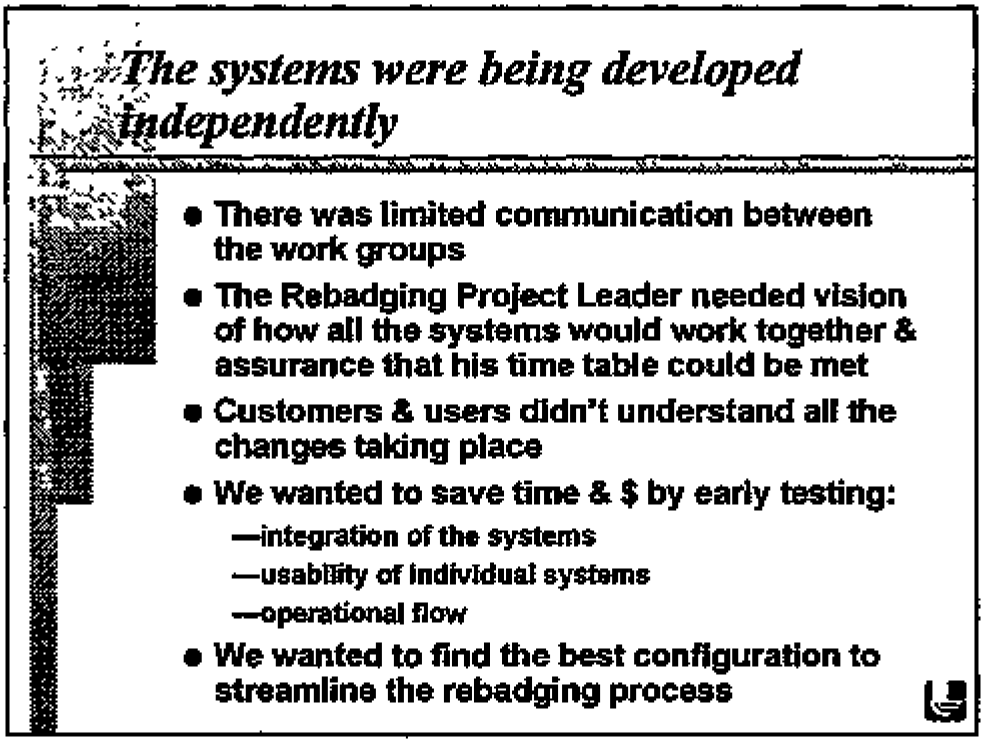
DOE mandated that LLNL be rebadged with the Standard DOE Badge

- Safeguards & Security decided to replace the existing clearance & badging system because of outdated hardware and software.
- The project decided to produce the badges with a video imaging system.
- There had to be new badge readers and revisions to the Access Control System.



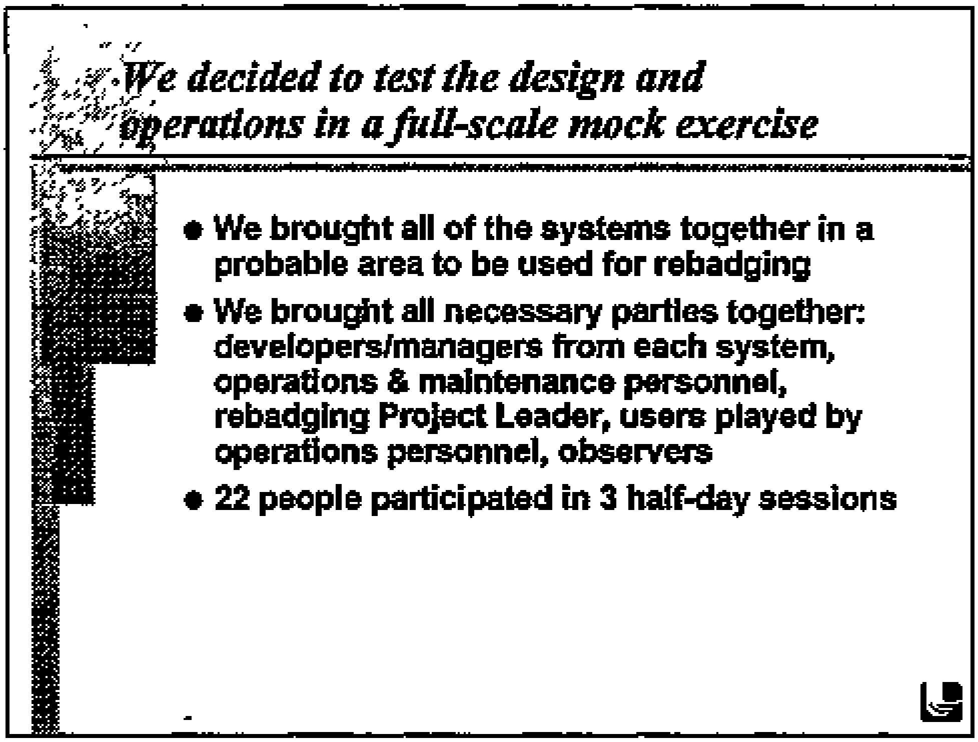
There were many stakeholders in the overall effort

- Badge Office (BO) & operations
- Central Clearance (CC) operations
- Security Information System developers supporting BO & CC (reengineered system)
- Video Imaging and Badge Making System (new system)
- Access Control System (new release)
- Rebadging Project Leader



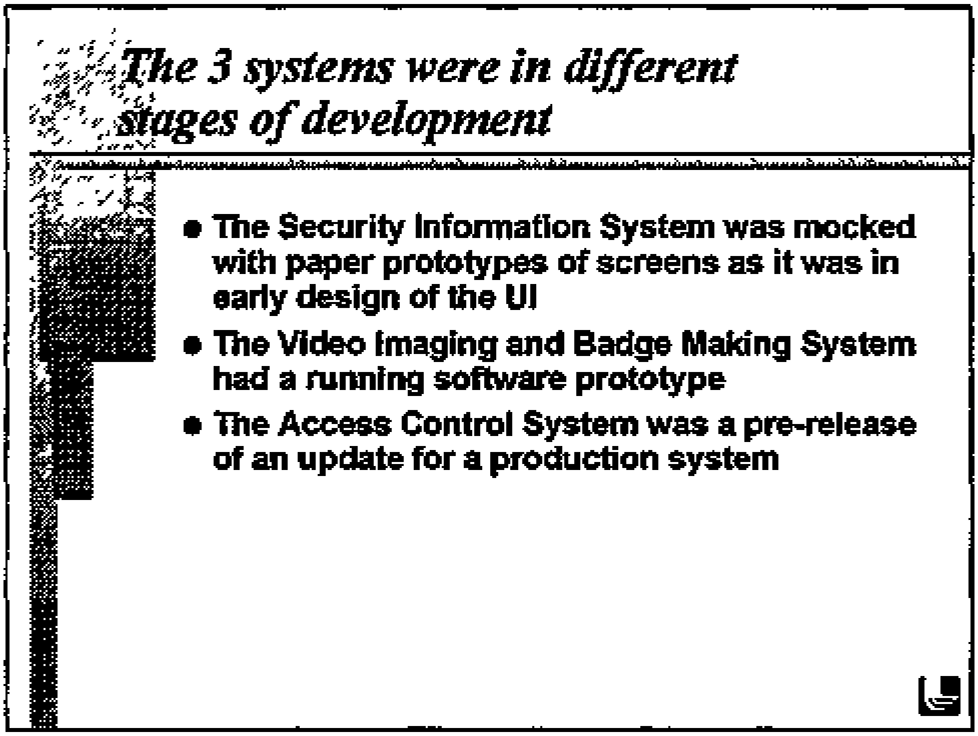
The systems were being developed independently

- There was limited communication between the work groups
- The Rebadging Project Leader needed vision of how all the systems would work together & assurance that his time table could be met
- Customers & users didn't understand all the changes taking place
- We wanted to save time & $ by early testing:
  - —integration of the systems
  - —usability of individual systems
  - —operational flow
- We wanted to find the best configuration to streamline the rebadging process



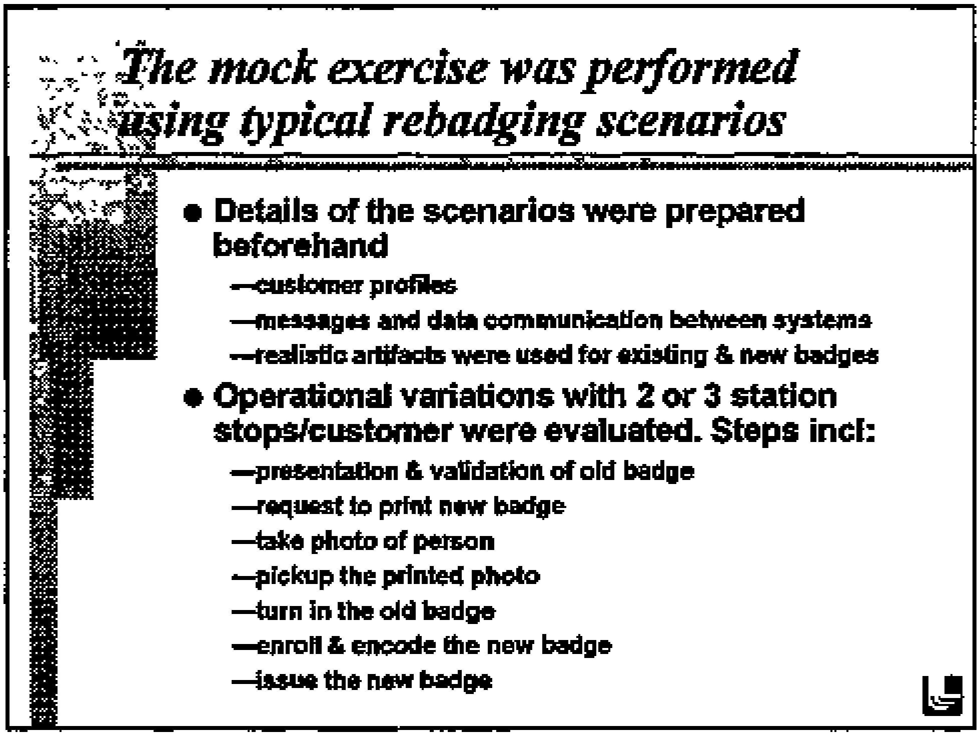
We decided to test the design and operations in a full-scale mock exercise

- We brought all of the systems together in a probable area to be used for rebadging
- We brought all necessary parties together: developers/managers from each system, operations & maintenance personnel, rebadging Project Leader, users played by operations personnel, observers
- 22 people participated in 3 half-day sessions



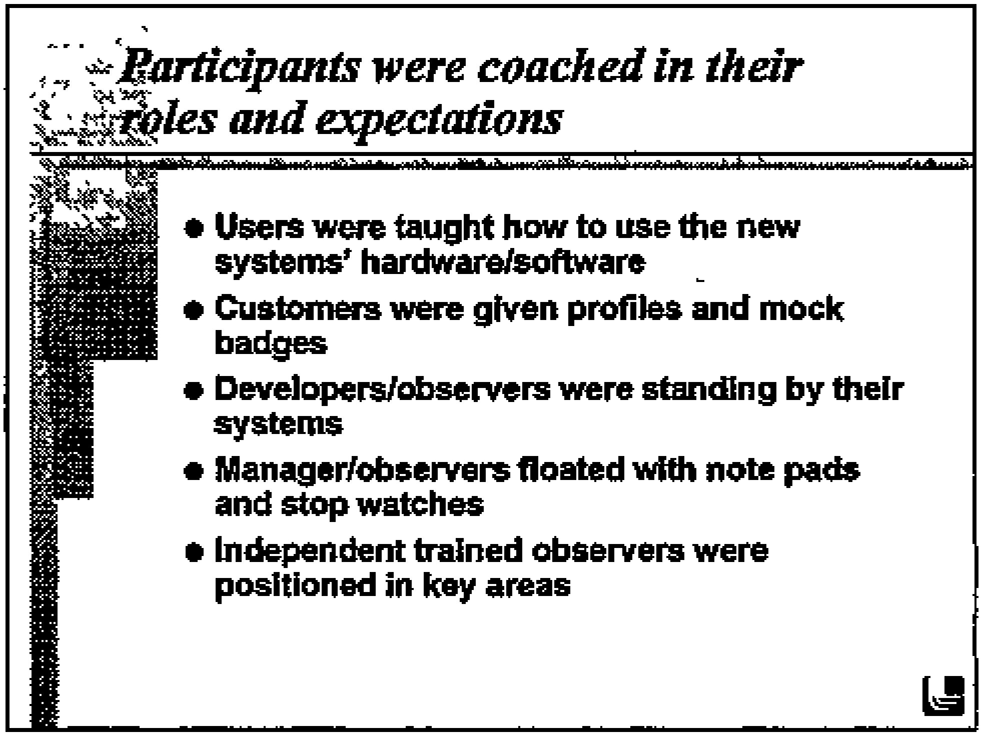
The 3 systems were in different stages of development

- The Security Information System was mocked with paper prototypes of screens as it was in early design of the UI
- The Video Imaging and Badge Making System had a running software prototype
- The Access Control System was a pre-release of an update for a production system



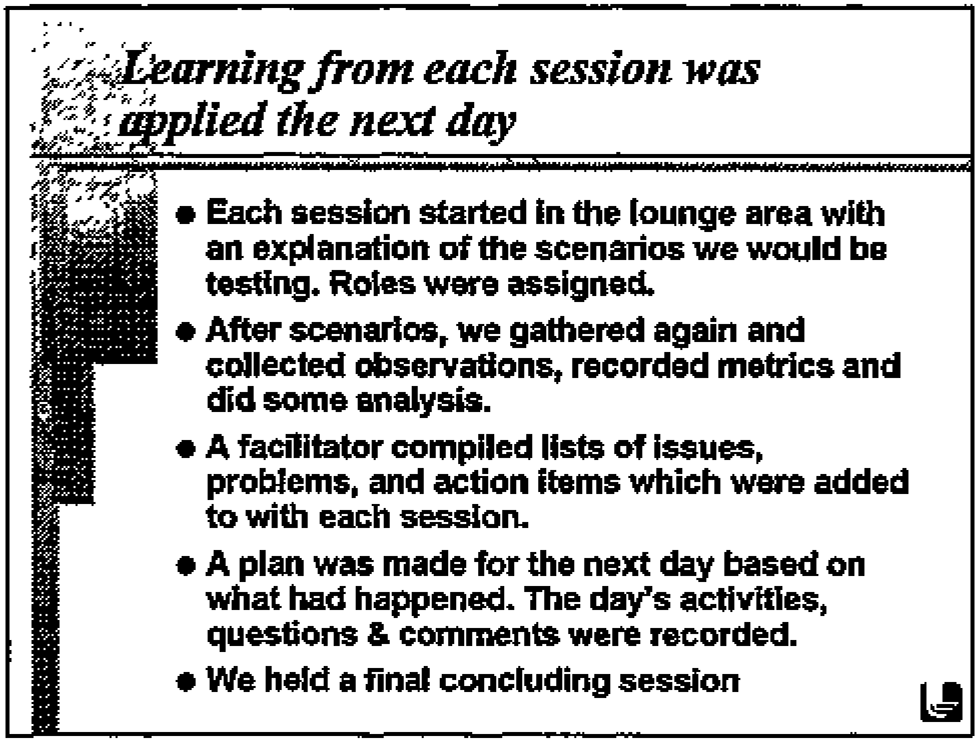
The mock exercise was performed using typical rebadging scenarios

- Details of the scenarios were prepared beforehand
  - —customer profiles
  - —messages and data communication between systems
  - —realistic artifacts were used for existing & new badges
- Operational variations with 2 or 3 station stops/customer were evaluated. Steps incl:
  - —presentation & validation of old badge
  - —request to print new badge
  - —take photo of person
  - —pickup the printed photo
  - —turn in the old badge
  - —enroll & encode the new badge
  - —issue the new badge



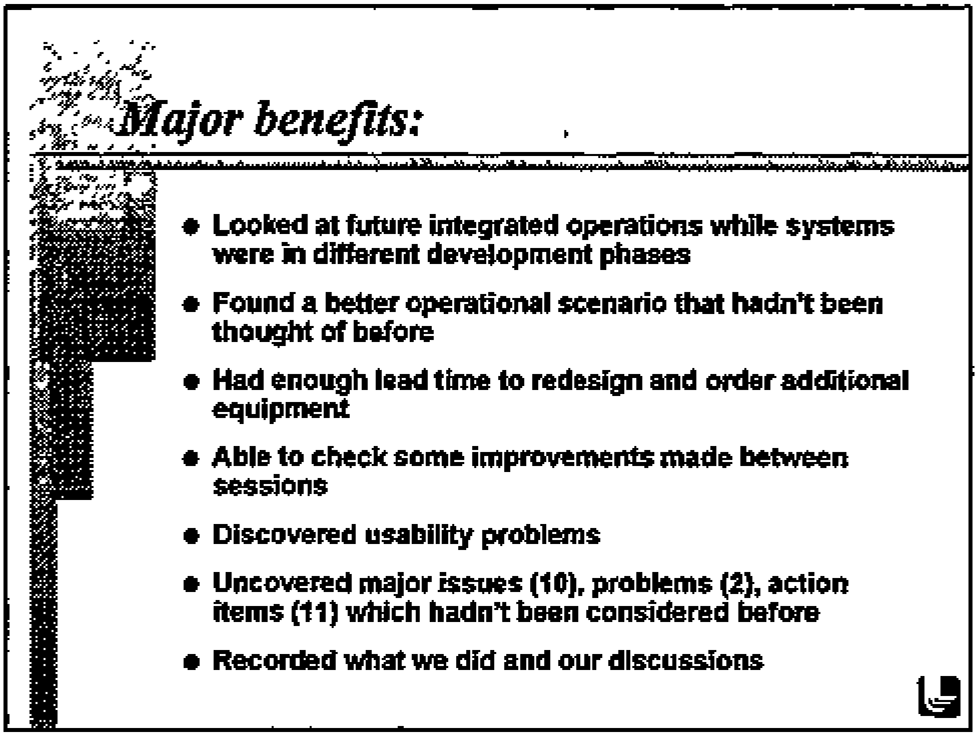
Participants were coached in their roles and expectations

- Users were taught how to use the new systems' hardware/software
- Customers were given profiles and mock badges
- Developers/observers were standing by their systems
- Manager/observers floated with note pads and stop watches
- Independent trained observers were positioned in key areas



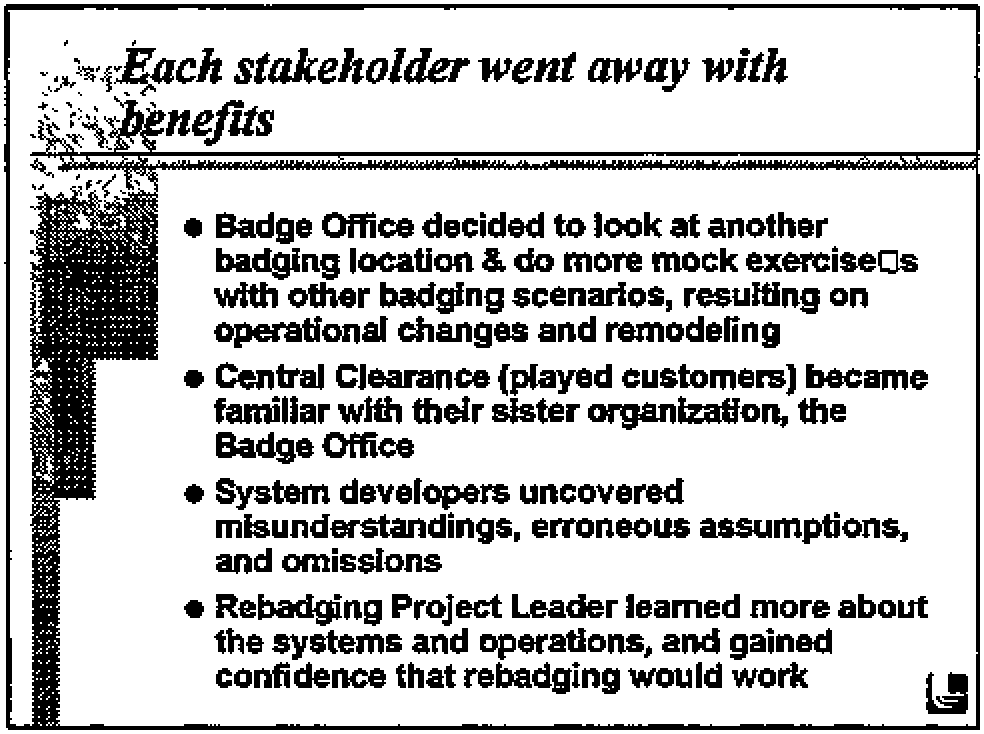
Learning from each session was applied the next day

- Each session started in the lounge area with an explanation of the scenarios we would be testing. Roles were assigned.
- After scenarios, we gathered again and collected observations, recorded metrics and did some analysis.
- A facilitator compiled lists of issues, problems, and action items which were added to with each session.
- A plan was made for the next day based on what had happened. The day's activities, questions & comments were recorded.
- We held a final concluding session



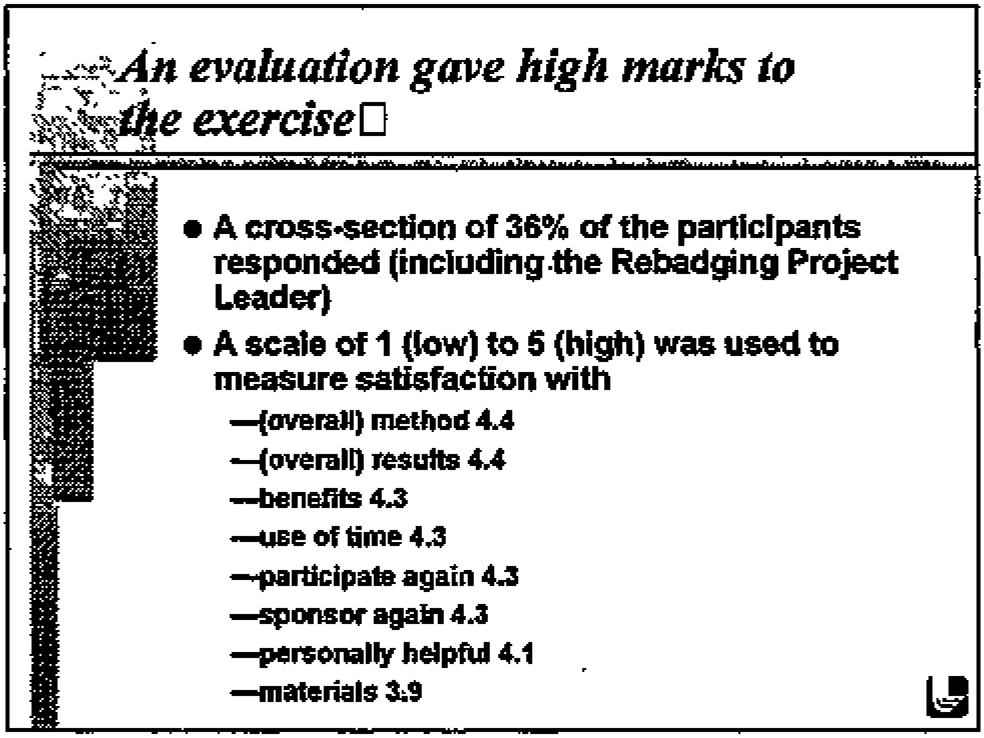
Major benefits:

- Looked at future integrated operations while systems were in different development phases
- Found a better operational scenario that hadn't been thought of before
- Had enough lead time to redesign and order additional equipment
- Able to check some improvements made between sessions
- Discovered usability problems
- Uncovered major issues (10), problems (2), action items (11) which hadn't been considered before
- Recorded what we did and our discussions



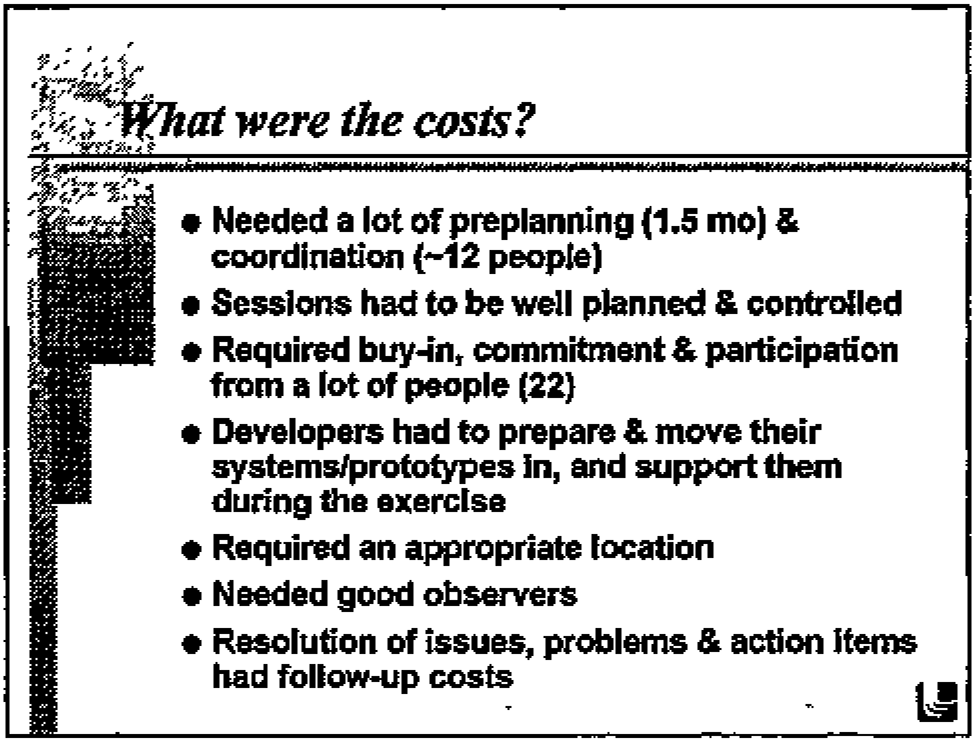
Each stakeholder went away with benefits

- Badge Office decided to look at another badging location & do more mock exercise□s with other badging scenarios, resulting on operational changes and remodeling
- Central Clearance (played customers) became familiar with their sister organization, the Badge Office
- System developers uncovered misunderstandings, erroneous assumptions, and omissions
- Rebadging Project Leader learned more about the systems and operations, and gained confidence that rebadging would work



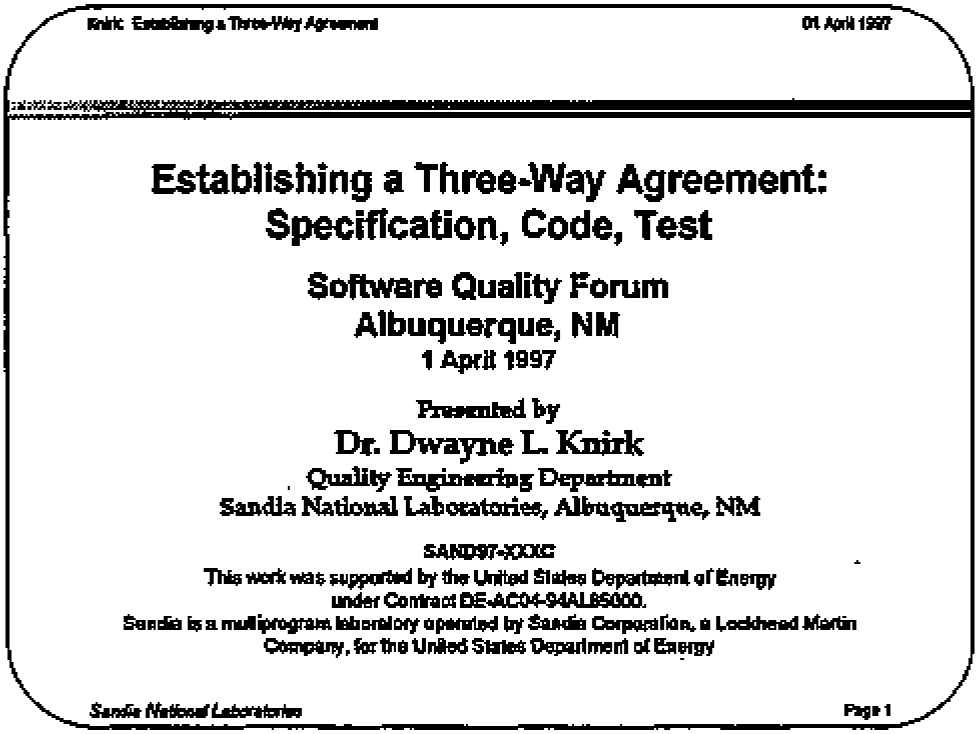
An evaluation gave high marks to the exercise□

- A cross-section of 36% of the participants responded (including the Rebadging Project Leader)
- A scale of 1 (low) to 5 (high) was used to measure satisfaction with
  - —(overall) method 4.4
  - —(overall) results 4.4
  - —benefits 4.3
  - —use of time 4.3
  - —participate again 4.3
  - —sponsor again 4.3
  - —personally helpful 4.1
  - —materials 3.9



What were the costs?

- Needed a lot of preplanning (1.5 mo) & coordination (~12 people)
- Sessions had to be well planned & controlled
- Required buy-in, commitment & participation from a lot of people (22)
- Developers had to prepare & move their systems/prototypes in, and support them during the exercise
- Required an appropriate location
- Needed good observers
- Resolution of issues, problems & action items had follow-up costs



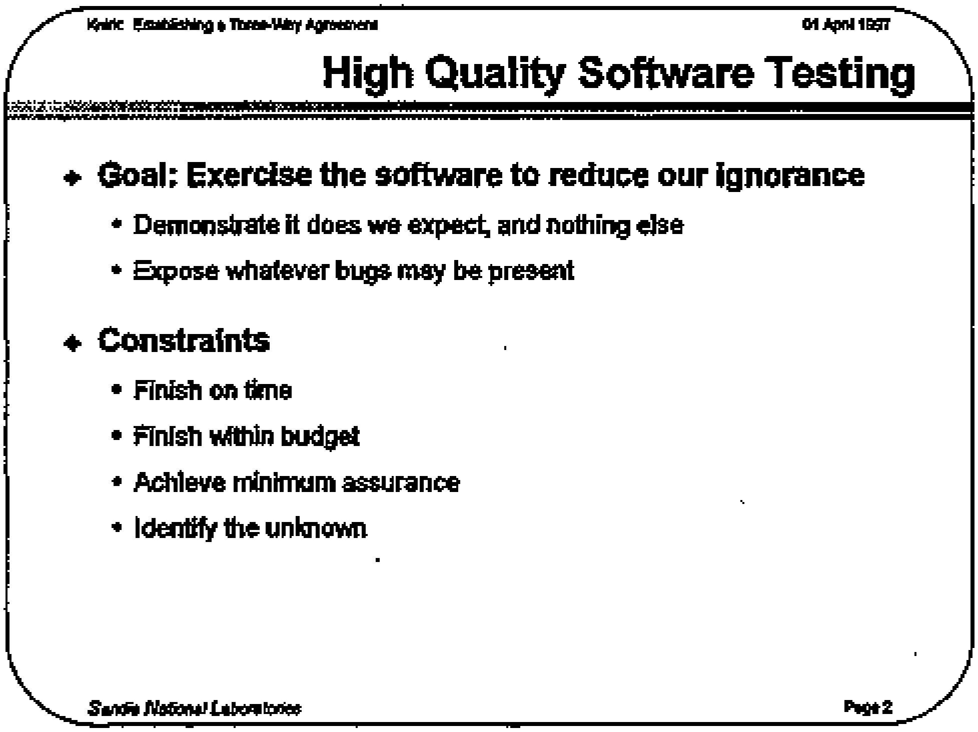
Establishing a Three-Way Agreement: Specification, Code, Test

Software Quality Forum
Albuquerque, NM
1 April 1997

Presented by

Dr. Dwayne L. Knirk
Quality Engineering Department
Sandia National Laboratories, Albuquerque, NM

SAND97-XXXC
This work was supported by the United States Department of Energy
under Contract DE-AC04-94AL85000.
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin
Company, for the United States Department of Energy



High Quality Software Testing

- **Goal: Exercise the software to reduce our ignorance**
  - Demonstrate it does we expect, and nothing else
  - Expose whatever bugs may be present
- **Constraints**
  - Finish on time
  - Finish within budget
  - Achieve minimum assurance
  - Identify the unknown

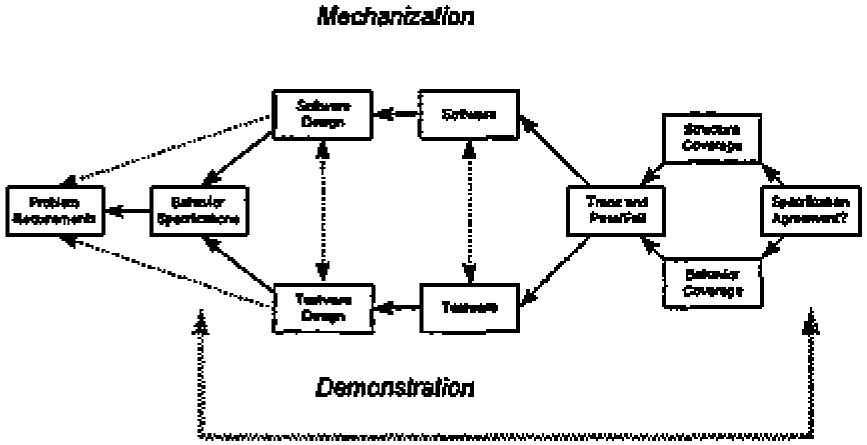
High Quality Software Testing

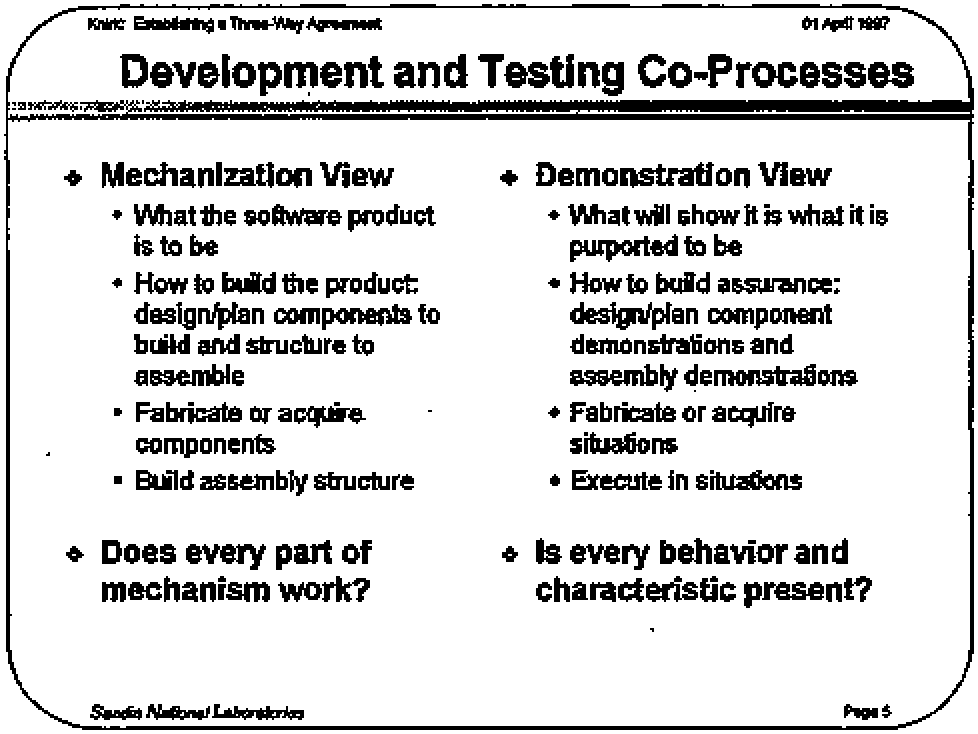
- The testing we do answers two questions:
  - Does it work?
  - What doesn't work?
- The testing we don't do reduces our confidence in the answers
- Every test case has a unique purpose

> To show something about the software that no other test case shows

Development and Testing Co-Processes

Conception Formulation Implementation Evaluation





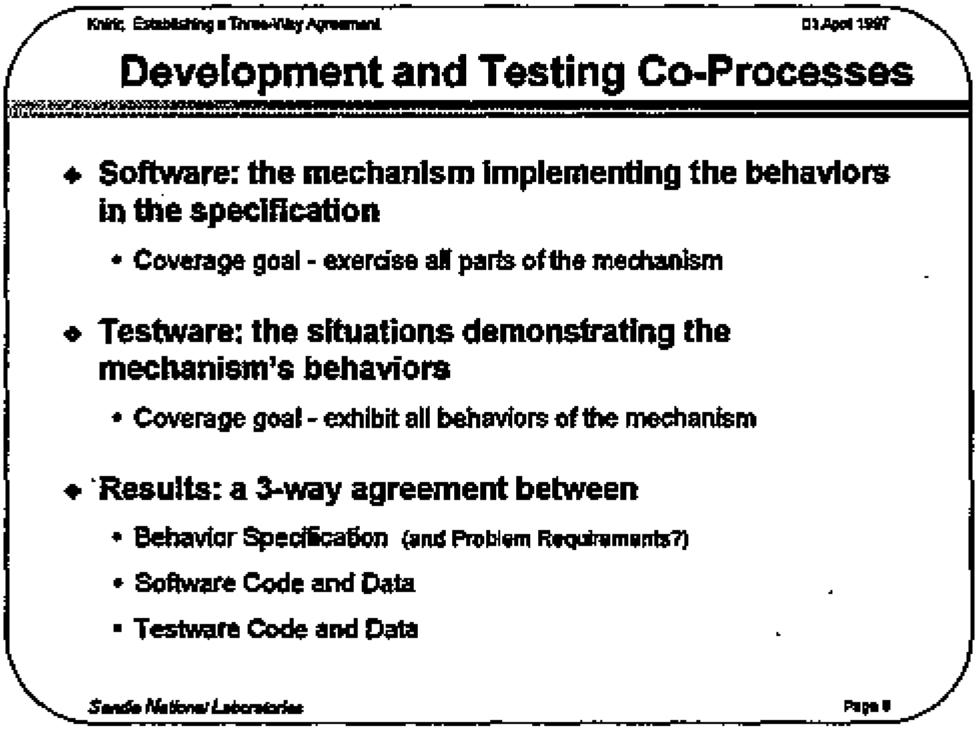
Knirk: Establishing a Three-Way Agreement 01 April 1997

Development and Testing Co-Processes

- Mechanization View
  - What the software product is to be
  - How to build the product: design/plan components to build and structure to assemble
  - Fabricate or acquire components
  - Build assembly structure
- Does every part of mechanism work?

- Demonstration View
  - What will show it is what it is purported to be
  - How to build assurance: design/plan component demonstrations and assembly demonstrations
  - Fabricate or acquire situations
  - Execute in situations
- Is every behavior and characteristic present?

Sandia National Laboratories Page 5

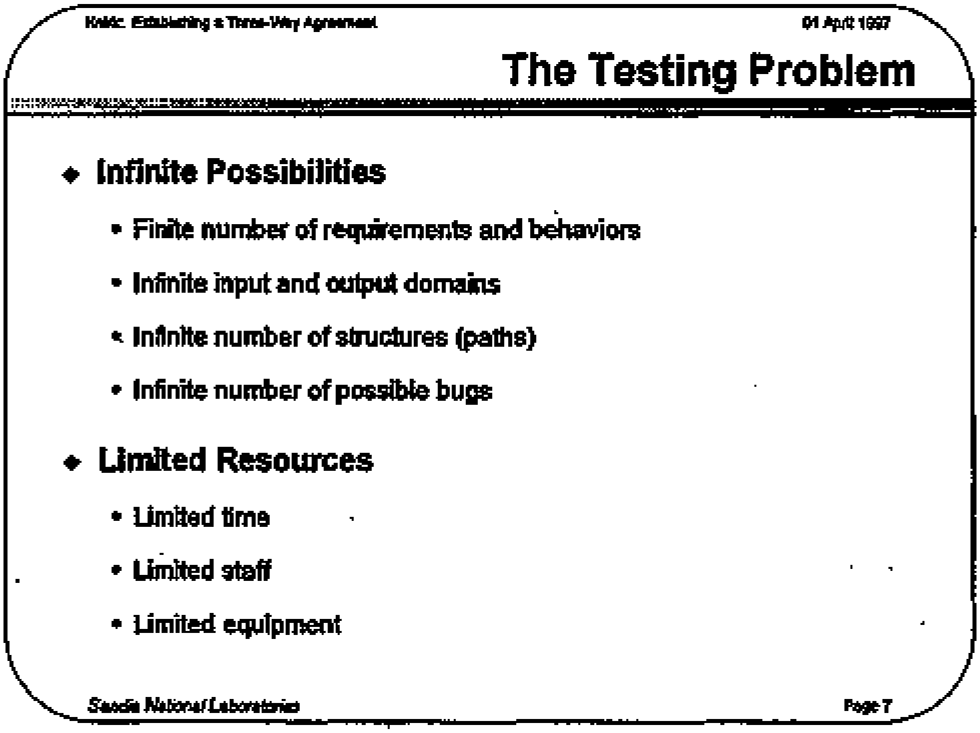


Knirk: Establishing a Three-Way Agreement 01 April 1997

Development and Testing Co-Processes

- Software: the mechanism implementing the behaviors in the specification
  - Coverage goal - exercise all parts of the mechanism
- Testware: the situations demonstrating the mechanism's behaviors
  - Coverage goal - exhibit all behaviors of the mechanism
- Results: a 3-way agreement between
  - Behavior Specification (and Problem Requirements?)
  - Software Code and Data
  - Testware Code and Data

Sandia National Laboratories Page 6



Kukic: Establishing a Three-Way Agreement

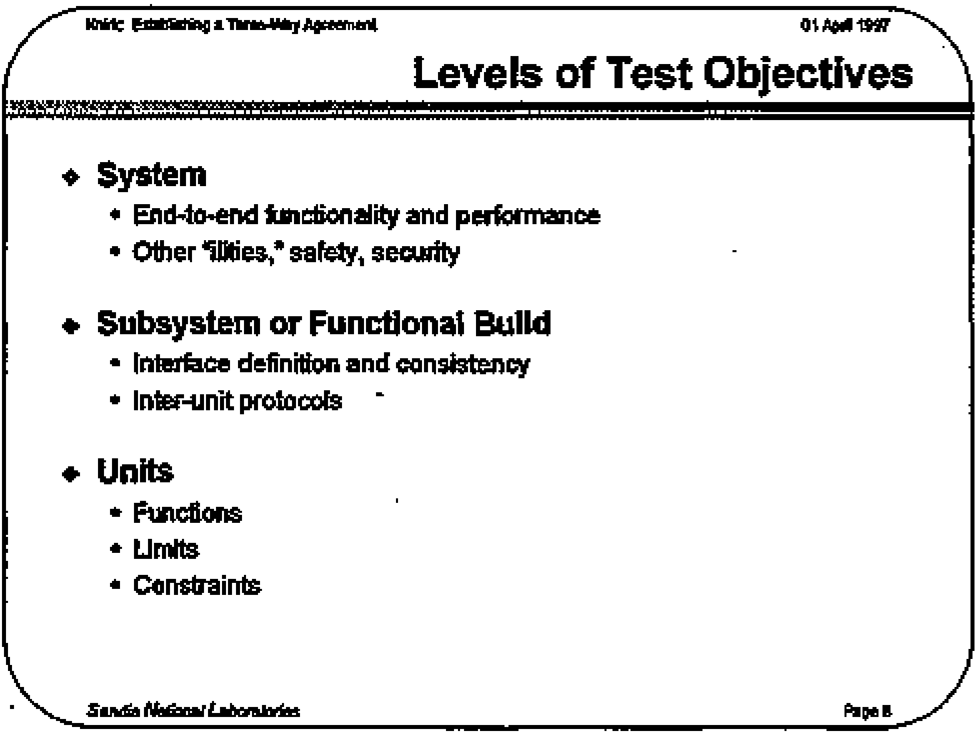
01 April 1997

The Testing Problem

- **Infinite Possibilities**
  - Finite number of requirements and behaviors
  - Infinite input and output domains
  - Infinite number of structures (paths)
  - Infinite number of possible bugs
- **Limited Resources**
  - Limited time
  - Limited staff
  - Limited equipment

Sandia National Laboratories

Page 7



Kukic: Establishing a Three-Way Agreement

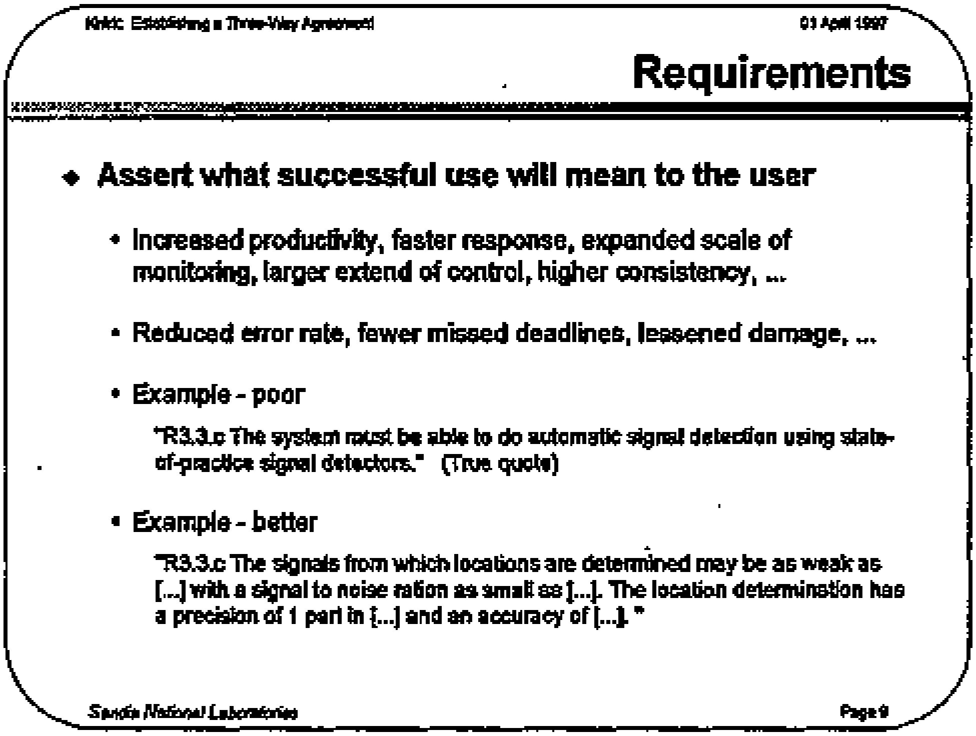
01 April 1997

Levels of Test Objectives

- **System**
  - End-to-end functionality and performance
  - Other "ilities," safety, security
- **Subsystem or Functional Build**
  - Interface definition and consistency
  - Inter-unit protocols
- **Units**
  - Functions
  - Limits
  - Constraints

Sandia National Laboratories

Page 8



Kirk: Establishing a Three-Way Agreement 01 April 1997

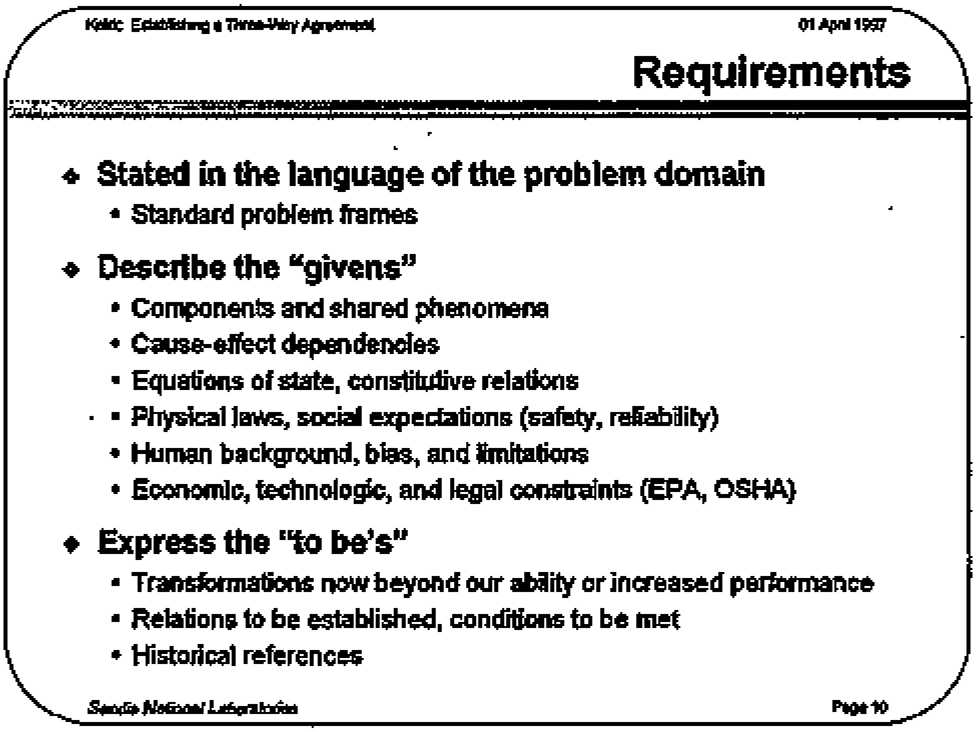
Requirements

- Assert what successful use will mean to the user
  - Increased productivity, faster response, expanded scale of monitoring, larger extend of control, higher consistency, ...
  - Reduced error rate, fewer missed deadlines, lessened damage, ...
  - Example - poor

    "R3.3.c The system must be able to do automatic signal detection using state-of-practice signal detectors." (True quote)
  - Example - better

    "R3.3.c The signals from which locations are determined may be as weak as [...] with a signal to noise ration as small as [...]. The location determination has a precision of 1 part in [...] and an accuracy of [...]. "

Sandia National Laboratories Page 9



Kirk: Establishing a Three-Way Agreement 01 April 1997

Requirements

- Stated in the language of the problem domain
  - Standard problem frames
- Describe the "givens"
  - Components and shared phenomena
  - Cause-effect dependencies
  - Equations of state, constitutive relations
  - Physical laws, social expectations (safety, reliability)
  - Human background, bias, and limitations
  - Economic, technologic, and legal constraints (EPA, OSHA)
- Express the "to be's"
  - Transformations now beyond our ability or increased performance
  - Relations to be established, conditions to be met
  - Historical references

Sandia National Laboratories Page 10

Specifications

- **Behavior**
  - Observable activity when measurable in terms of quantifiable effects on the environment whether arising from internal or external stimulus
  - The peculiar reaction of a thing under given circumstances
- **Behavior Specification**
  - Focuses on the functions required of the executing software
  - Expressed in terms of observables of software behavior
  - Allows many possible software implementations
  - Must be *predictive* to answer questions of the following sort

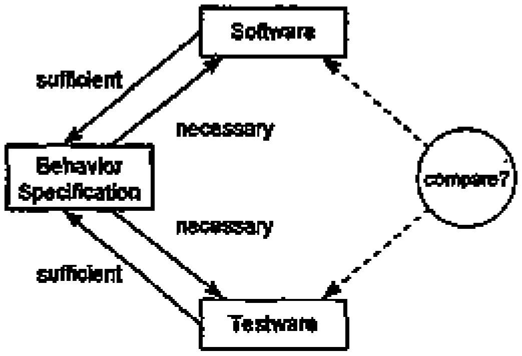
    "In situation Q, what does the computing system do when P happens?" (and P happens, often when it is least expected)

Specifications

- **Stated in the language of shared phenomena**
  - Standard interaction patterns
- **Describe the interactions between the application environment and the computing system**
  - Direction (input, output)
  - Aggregate structure (by value, by reference)
  - Representation medium (digital, analog), format, units
  - Time and value granularities (continuous, discrete)
  - Time and value domains (possible values, event times)
- **Express interaction sequences and coordination**
  - Stimulus-response interactions (cause-effect)
  - Serialization and concurrency
  - Internal "real world" model

Dependencies

Three-way agreement



Dependencies

- **Software**
  - Necessary — all specified behaviors are realized by the code
  - Sufficient — all implemented behaviors are desired

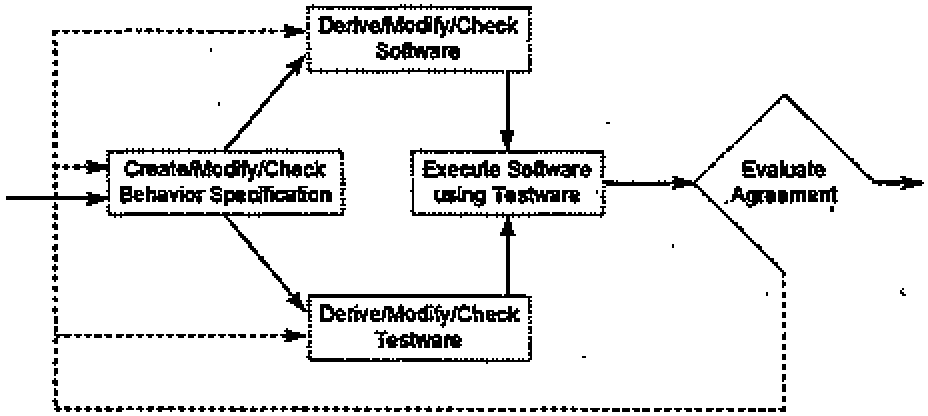
Behavior Specification ←→ Software

- **Testware**
  - Necessary — all specified behaviors are demonstrated in tests
  - Sufficient — all demonstrated behaviors are desired

Behavior Specification ←→ Testware

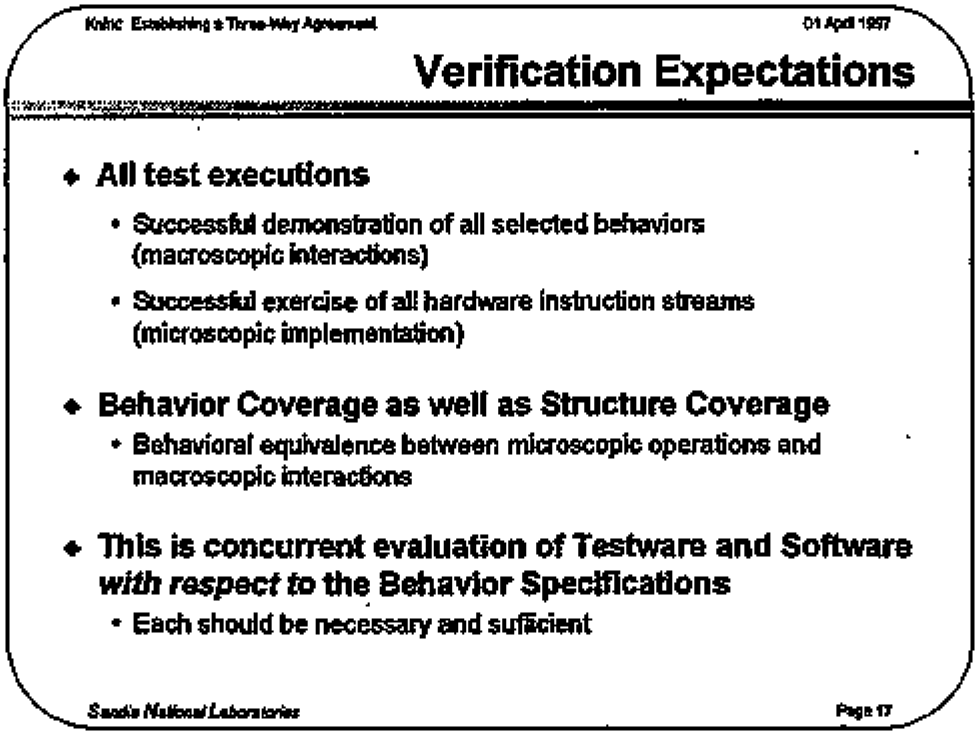
What do these two equivalencies suggest?

Establishing Three-Way Agreement


Derive/Modify/Check Software
Create/Modify/Check Behavior Specification
Execute Software using Testware
Evaluate Agreement
Derive/Modify/Check Testware

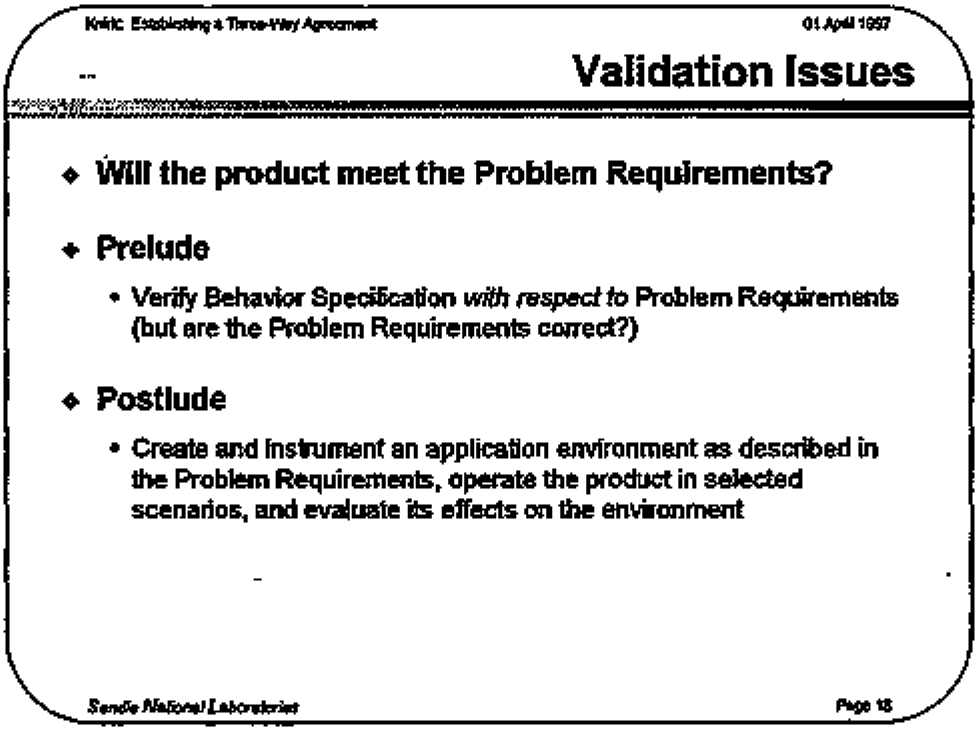
Establishing Three-Way Agreement

- **Process**
  - Design test cases from the behavior specification
  - Execute tests on an instrumented code
  - Examine test outcomes for behavior pass/fail
    missed services, missed state transitions, incorrect retained data updates
    wrong boundaries, violated constraints
  - Examine execution trace for structure coverage omissions
    missed segments, missed branches, missed branch sequences
    missed units, missed call-return pairs, missed data def-use pairs
  - Quit when all? behaviors pass and all? structures are exercised
  - Otherwise, change specification, code, or tests, and iterate



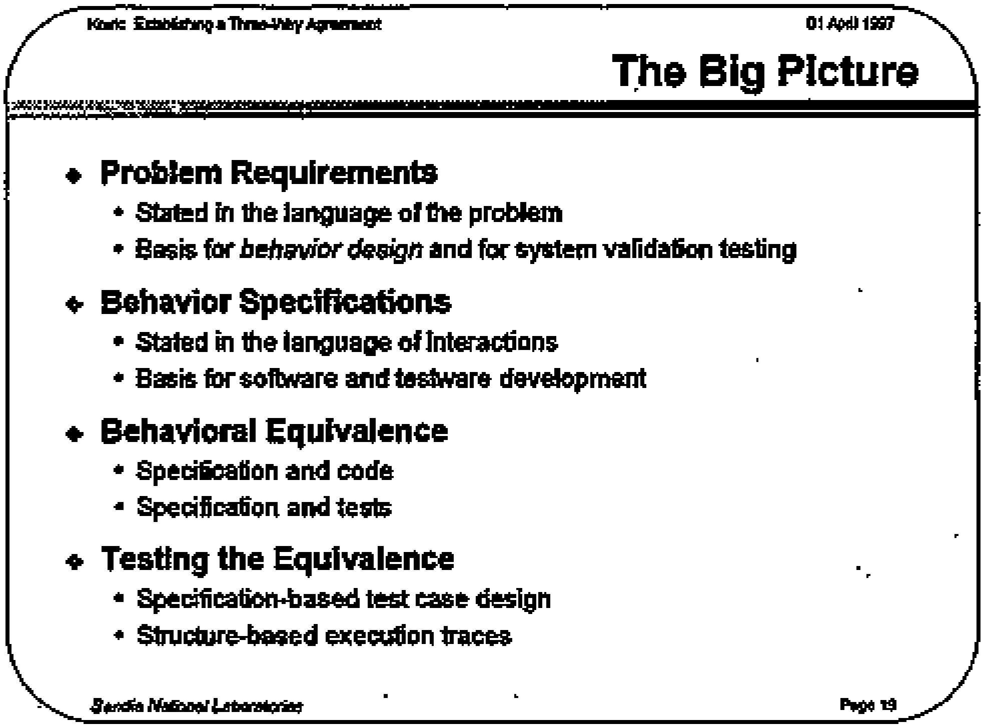
Verification Expectations

- **All test executions**
  - Successful demonstration of all selected behaviors (macroscopic interactions)
  - Successful exercise of all hardware instruction streams (microscopic implementation)
- **Behavior Coverage as well as Structure Coverage**
  - Behavioral equivalence between microscopic operations and macroscopic interactions
- **This is concurrent evaluation of Testware and Software *with respect to* the Behavior Specifications**
  - Each should be necessary and sufficient



Validation Issues

- **Will the product meet the Problem Requirements?**
- **Prelude**
  - Verify Behavior Specification *with respect to* Problem Requirements (but are the Problem Requirements correct?)
- **Postlude**
  - Create and instrument an application environment as described in the Problem Requirements, operate the product in selected scenarios, and evaluate its effects on the environment



Koric: Establishing a Three-Way Agreement 01 April 1997

The Big Picture

- **Problem Requirements**
  - Stated in the language of the problem
  - Basis for *behavior design* and for system validation testing
- **Behavior Specifications**
  - Stated in the language of interactions
  - Basis for software and testware development
- **Behavioral Equivalence**
  - Specification and code
  - Specification and tests
- **Testing the Equivalence**
  - Specification-based test case design
  - Structure-based execution traces

Sandia National Laboratories Page 19

1947 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Session C1: Software Quality for Scientific Applications

Chair John Cerutti

Los Alamos National Laboratory

Session : Paper #	Author(s)	*Title*
C1:1	John Ambrosiano & Robert Webster Los Alamos National Laboratory	*Software Quality and Process Improvement in Scientific Simulation Codes*
C1:2	Ed Russell Lawrence Livermore National Laboratory	*The SQA of Finite Element Method (FEM) Codes used for Analyses of Pit Storage/Transport Packages*
C1:3	Orval Hart Los Alamos National Laboratory	*Software Quality Assurance at the Weapons Engineering Tritium Facility*

Software Quality and Process Improvement in Scientific Simulation Codes

John Ambrosiano and Robert Webster
Computation Methods Group
Applied Theoretical and Computational Physics Division
Los Alamos National Laboratory

Motivation

- This study looks at the quest for better simulation code quality through process modeling and improvement
- Scientists often doubt the value of standardized methods for software development and QA saying they believe the process models on which they are based are not appropriate
- The goal of this study is to discover the processes by which computational scientists produce production and prototype simulation codes and to compare these processes with standard software process methodology

Background

- The authors of the study are computational scientists who have been involved in both large and small simulation code projects for many years
- The subjects of this study are scientists and computer scientists within the Applied Theoretical and Computational Physics (X) Division of Los Alamos National Laboratory
- X Division is responsible for developing and maintaining simulation codes used in nuclear weapon design and assessment
- One of the goals of this study is to try to understand our own code development processes at LANL better

How this Study was Conducted

- The study is based on the experience of the authors and interviews with 10 subjects chosen from simulation code development teams at LANL
- This study is descriptive rather than scientific
  - evidence is mainly anecdotal
  - taken from a small sample in an isolated population
- The aim is to discover and develop ideas that could lead to better and broader studies
- In order to provide a frame of reference for the study we referred to the SEI Capability Maturity Model (CMM); also used were two books by Watts Humphrey:
  - "Managing the Software Process," (1989)
  - "A Discipline for Software Engineering," (1995)

The Capability Maturity Model

- The CMM suggests incremental process improvement guidelines:
  - **Level 2, repeatable:** institute certain key practices on a per project basis
  - **Level, defined:** move toward uniform organization-wide implementation of practices
  - **Level 4, managed:** instrument key practices with appropriate measures
  - **Level 5, optimizing:** use measures to optimize the process

CMM (continued)

- The CMM suggests key practices at each level specific to software engineering
- Key practices considered essential to reach level 2 are:
  - Requirements management
  - Project planning
  - Project tracking
  - Subcontract management
  - Quality Assurance
  - Configuration management

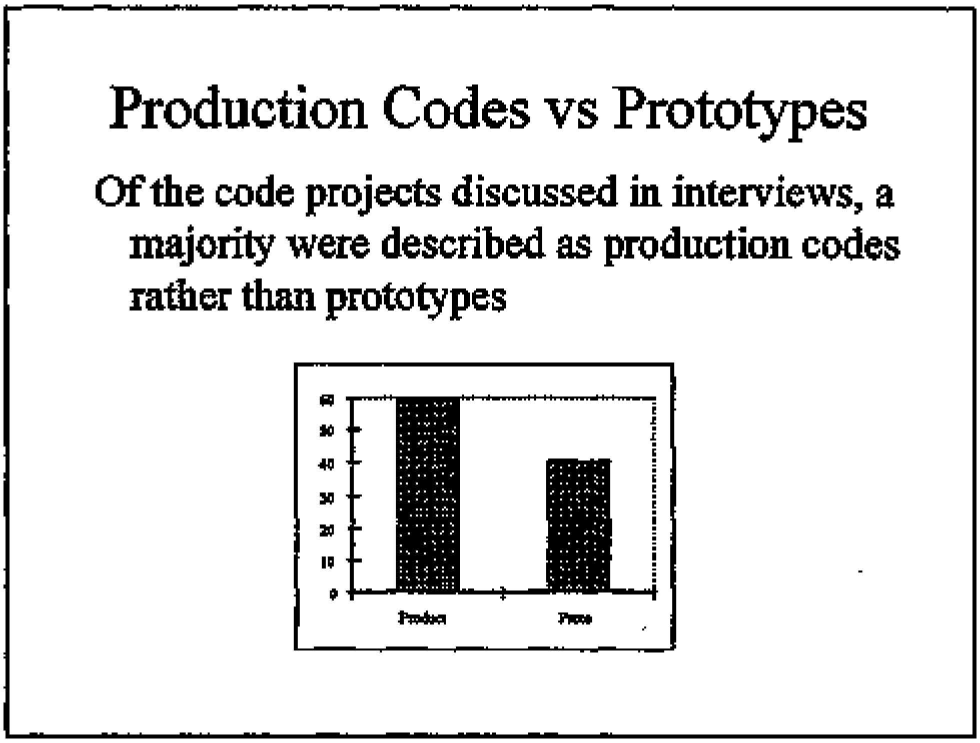
General Statisitics

- Project size: between 2 and 15; average 6
- Many projects described as ongoing for years (1 to 15); average 5.5
- Numerical application domains covered:
  - hydrodynamics, radiation transport, neutronics, computational geometry, data analysis, electromagnetics, and plasmas
- Estimated lines of code: 30,000 to one million; average about 250,000

Production Codes vs Prototypes

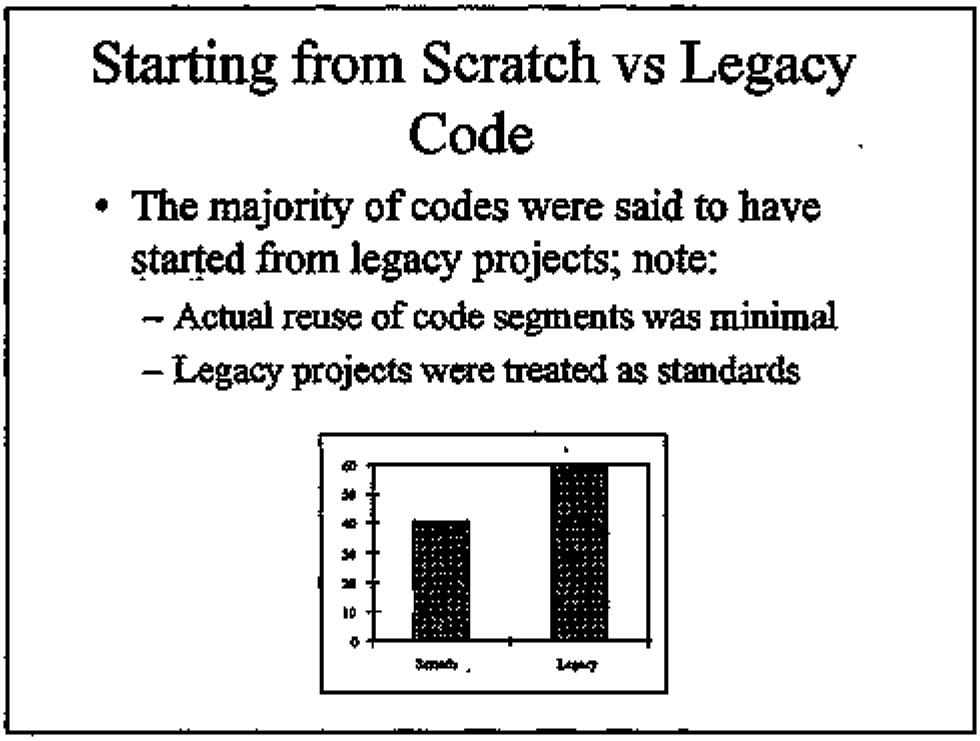
Production codes were distinguished as follows:

- Designed to be used by someone other than a developer
- Well documented; reasonable learning curve
- Serve as repository for models and algorithms proven to be useful; a historical archive of community experience in the intended application domain
- Give correct or expected answers to an agreed set of posed problems of practical interest



Production Codes vs Prototypes

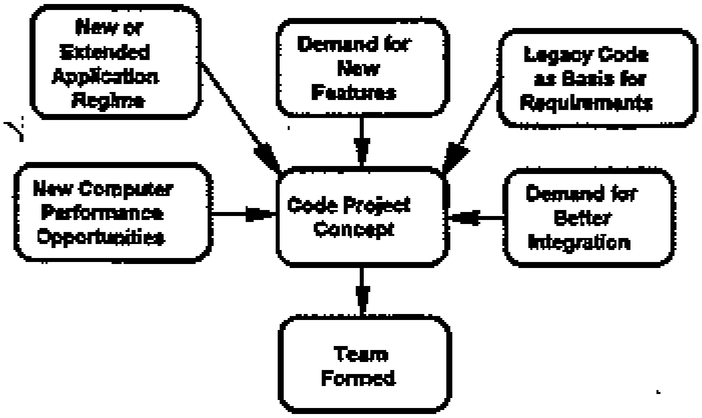
Of the code projects discussed in interviews, a majority were described as production codes rather than prototypes



Starting from Scratch vs Legacy Code

- The majority of codes were said to have started from legacy projects; note:
  - Actual reuse of code segments was minimal
  - Legacy projects were treated as standards

A Typical Development Process: Concept Phase



New or Extended Application Regime

Demand for New Features

Legacy Code as Basis for Requirements

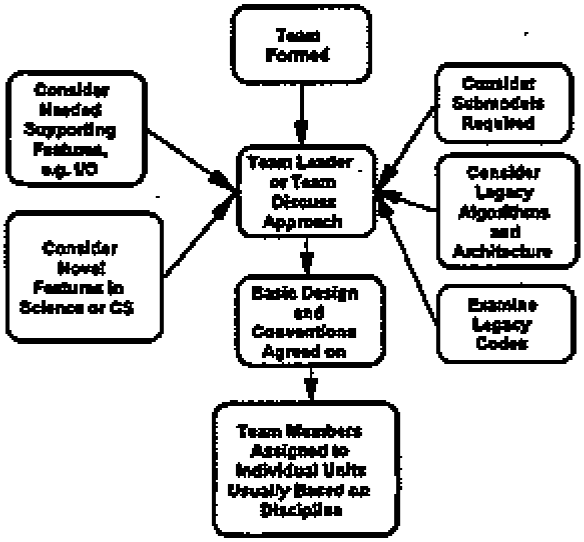
New Computer Performance Opportunities

Code Project Concept

Demand for Better Integration

Team Formed

Development Process: Design



Team Formed

Consider Needed Supporting Features, e.g. I/O

Consider Novel Features in Science or CS

Team Leader or Team Discuss Approach

Consider Submodels Required

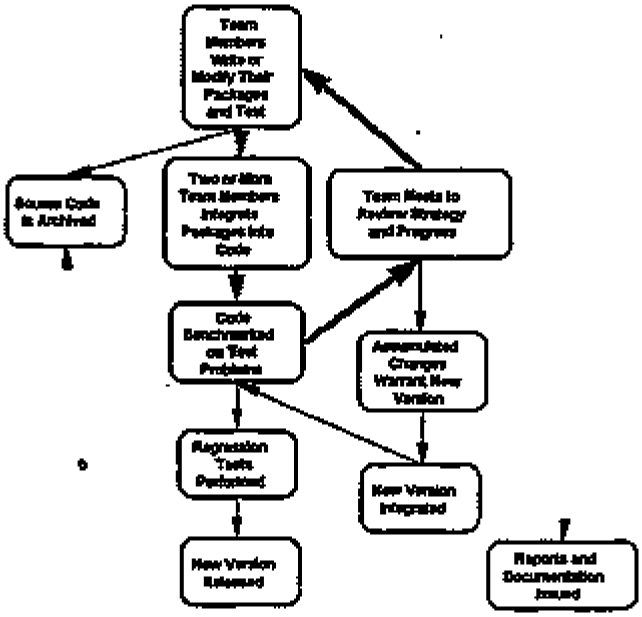
Consider Legacy Algorithms and Architecture

Examine Legacy Codes

Basic Design and Conventions Agreed on

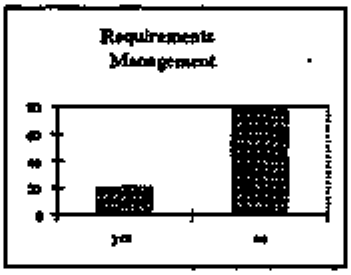
Team Members Assigned to Individual Units Usually Based on Discipline

Development Process (cont):


Team Members Write or Modify Their Packages and Test
Source Code is Archived
Two or More Team Members Integrate Packages into Code
Team Meets to Review Strategy and Progress
Code Benchmarked on Test Problems
Accumulated Changes Warrant New Version
Regression Tests Performed
New Version Integrated
New Version Released
Reports and Documentation Issued

Comparison with CMM Practices: Requirements Management

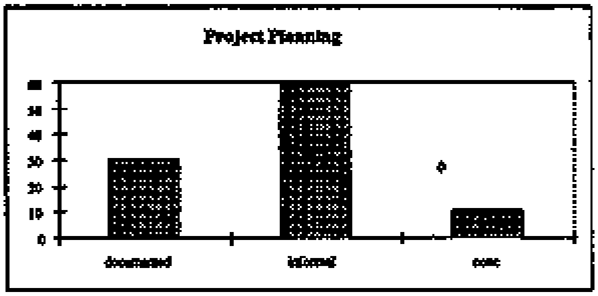
- Majority (80%) reported requirements were not developed in detail
- Interviewees told us the principle requirement is that the model produce the "correct" answer



- When questioned further told us that the codes had to reproduce the results of the legacy production codes

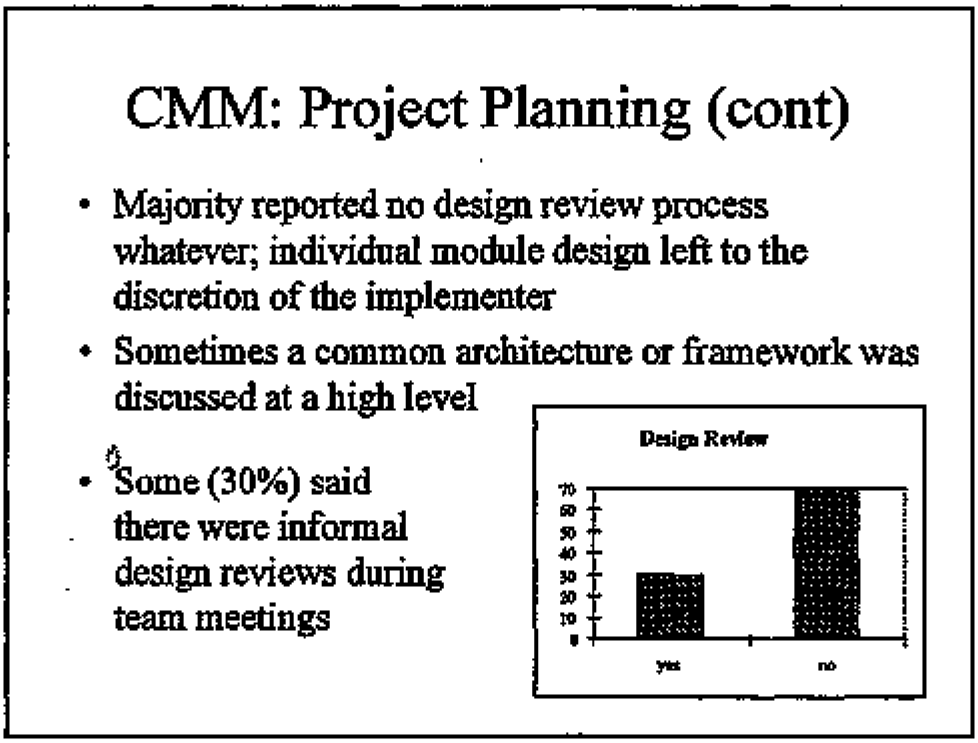
CMM Practices: Project Planning

- Most projects (70%) did not have a documented project plan with specific tasks, timelines and milestones
- 60% said the project was planned in an *informal* way



CMM: Project Planning (cont)

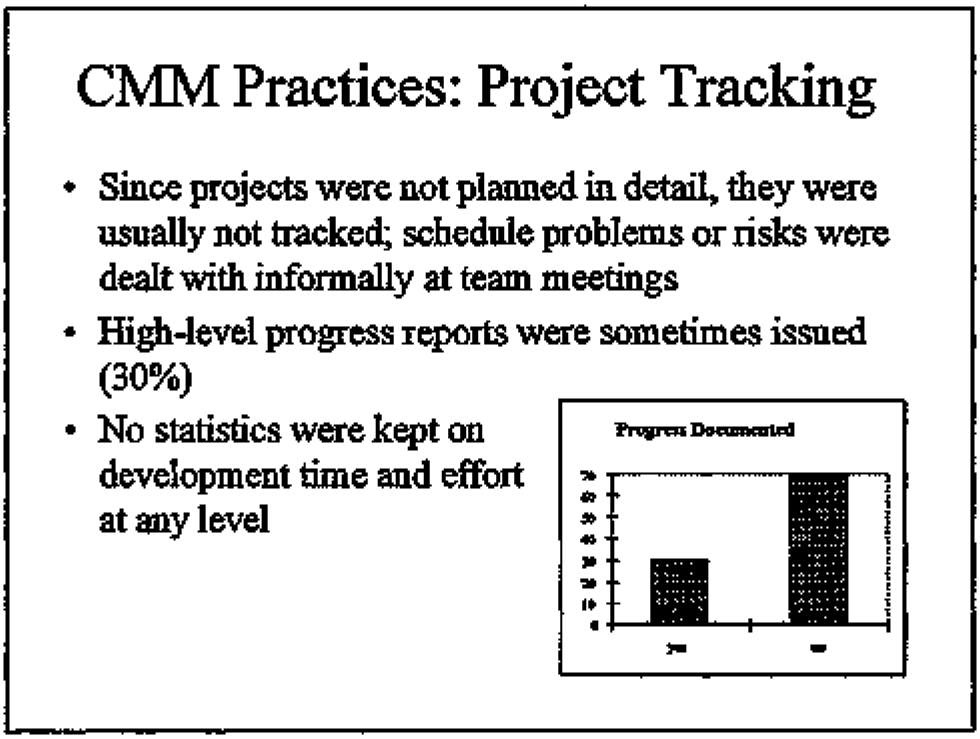
- Architecture and design broken down by scientific discipline
- Design strategies largely functional (one object-oriented design; a prototype)
- Milestones driven by user demands; in the past closely linked to the nuclear test schedule
- Little or no data (e.g. LOC) used in estimating development time or personnel costs



CMM: Project Planning (cont)

- Majority reported no design review process whatever; individual module design left to the discretion of the implementer
- Sometimes a common architecture or framework was discussed at a high level
- Some (30%) said there were informal design reviews during team meetings

Design Review
70
60
50
40
30
20
10
0
yes
no



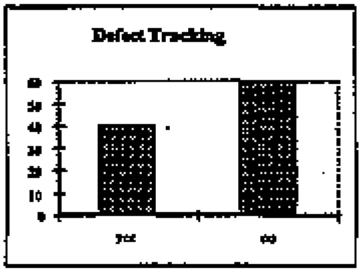
CMM Practices: Project Tracking

- Since projects were not planned in detail, they were usually not tracked; schedule problems or risks were dealt with informally at team meetings
- High-level progress reports were sometimes issued (30%)
- No statistics were kept on development time and effort at any level

Progress Documented
70
60
50
40
30
20
10
0
yes
no

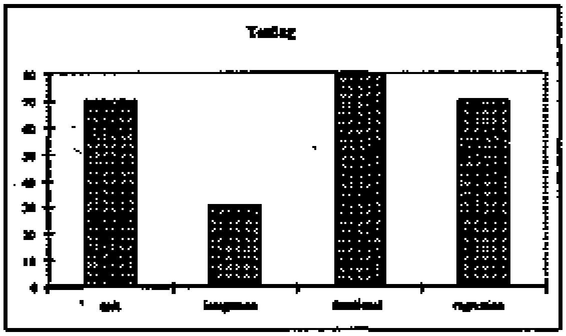
CMM Practices: Testing and Quality Assurance

- Few projects kept any statistics on defects. Those reporting some defect tracking (40%) maintained a bug report list. No statistics were kept on number of defects, type or effort expended in repair.



CMM : Testing and QA

- All projects did a fair amount of testing
- Unit tests done almost always at the discretion of the programmer
- Integration tests were unplanned
- Function tests mainly scientific or mathematical test problems
- Regression test suite was typically employed

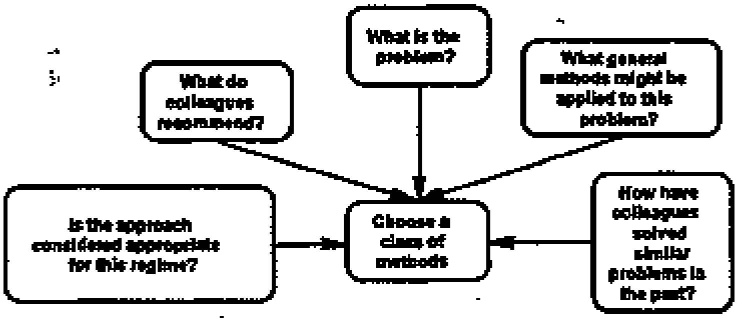


A Tale of Two Projects (cont)

- (The novel project)
  - Met its benchmarks of hypothetical test problems
  - Did not meet expectations when used on the intended design application
  - Involved new algorithms, new architectures, and new programming methods (sometimes together)
- The other project
  - Had no new methods
  - Involved 6 people for 3 years, and produced 300,000 lines of code
  - Is considered a success

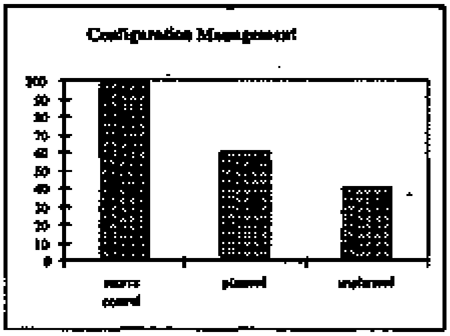
Exploration Explored: Looking at the Solution Landscape

- One interview concerned only algorithm development as opposed to code development
- The following is a process model based on that interview


What is the problem?
What do colleagues recommend?
What general methods might be applied to this problem?
Is the approach considered appropriate for this regime?
Choose a class of methods
How have colleagues solved similar problems in the past?

CMM Practices: Configuration Mangement

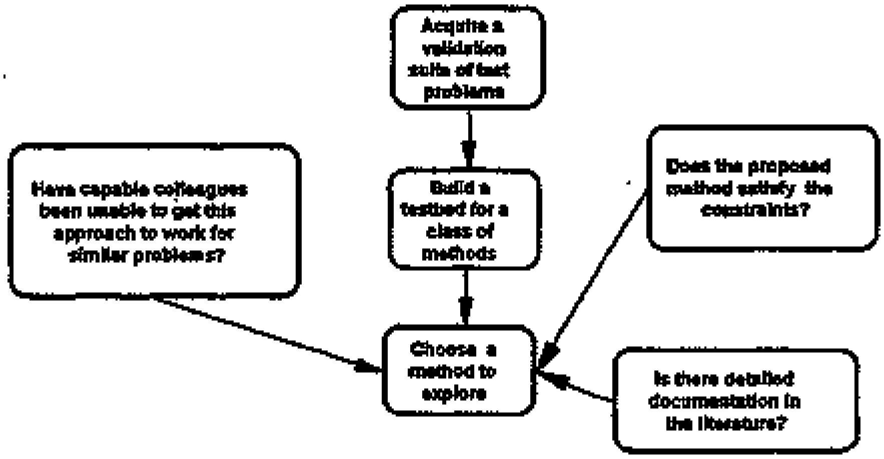
- All reported some configuration management practice
- Disciplined, planned configuration management was *not* typical
- Some sort of *source control was universal*



A Tale of Two Projects

- An interesting contrast arose between two projects:
  - A well-regarded, well-used production code
  - An ambitious and novel development effort
- The novel project
  - Had the largest team (10-15)
  - Produced the largest number of LOC (over one million)
  - Used formal methods more extensively
  - Was officially supported for about seven years
  - Was terminated by management before completion

Exploration: Narrowing the Search Space



Acquire a validation suite of test problems

Build a testbed for a class of methods

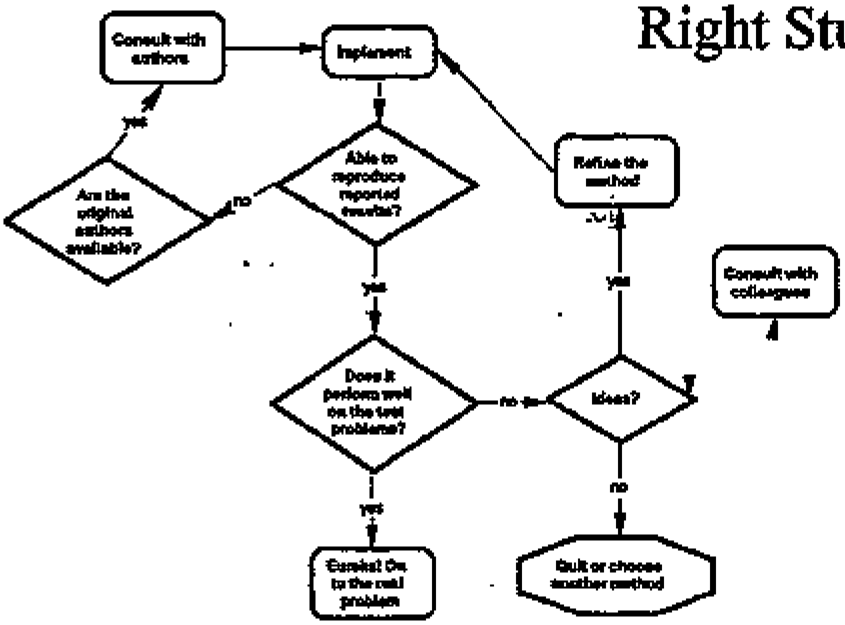
Have capable colleagues been unable to get this approach to work for similar problems?

Does the proposed method satisfy the constraints?

Choose a method to explore

Is there detailed documentation in the literature?

Exploration: Looking for the Right Stuff



Consult with authors

Implement

Are the original authors available?

Able to reproduce reported results?

Refine the method

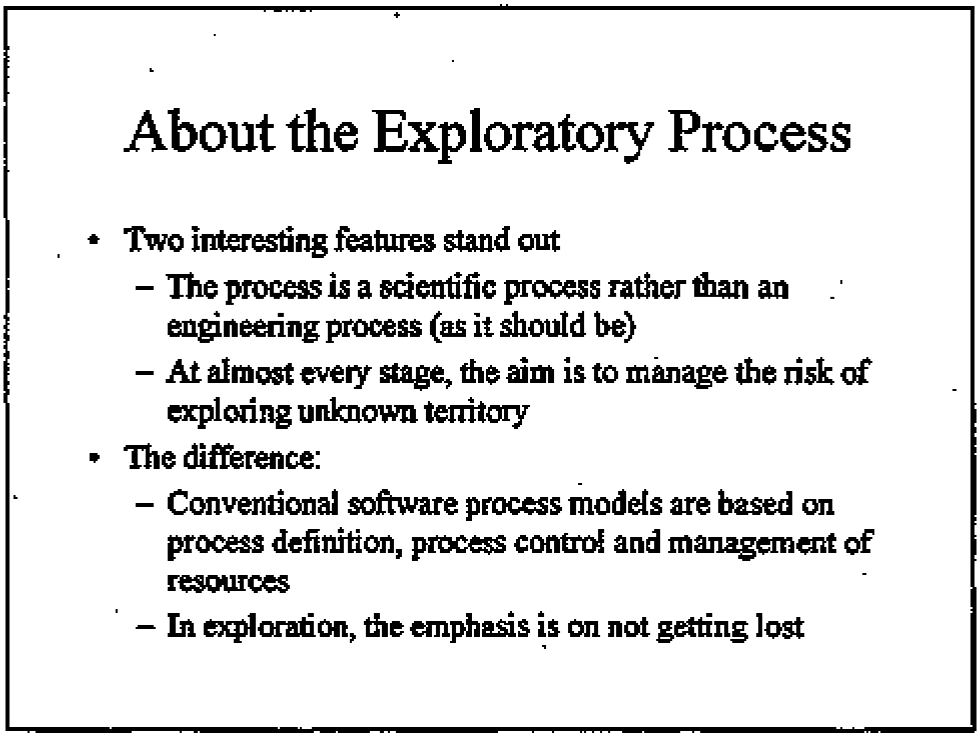
Consult with colleagues

Does it perform well on the test problems?

Ideas?

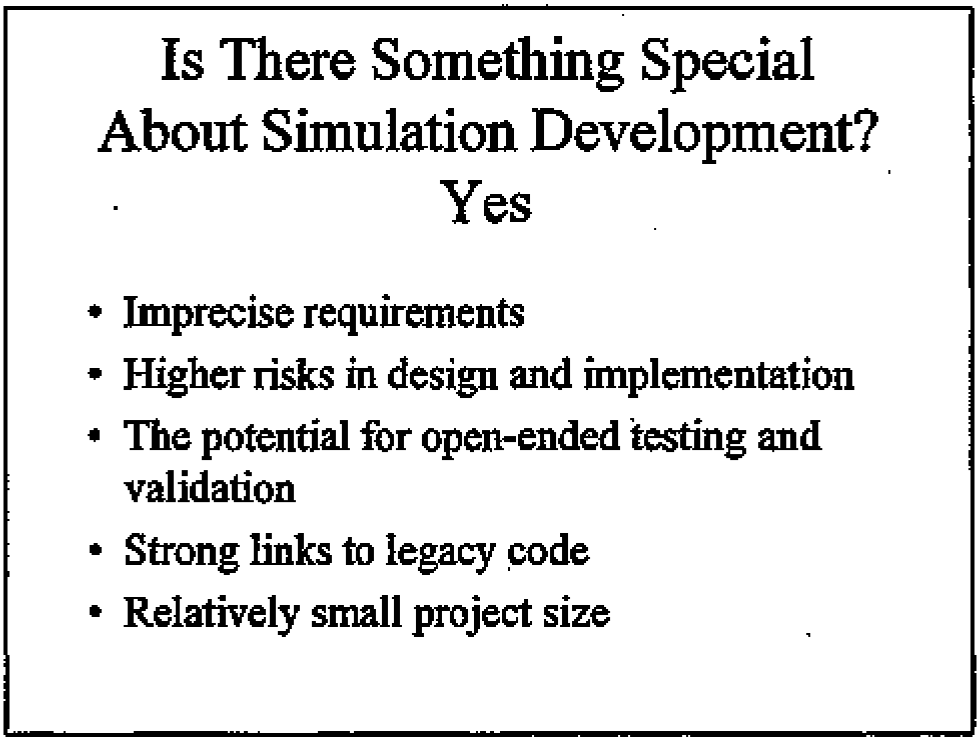
Eureka! On to the real problem

Quit or choose another method



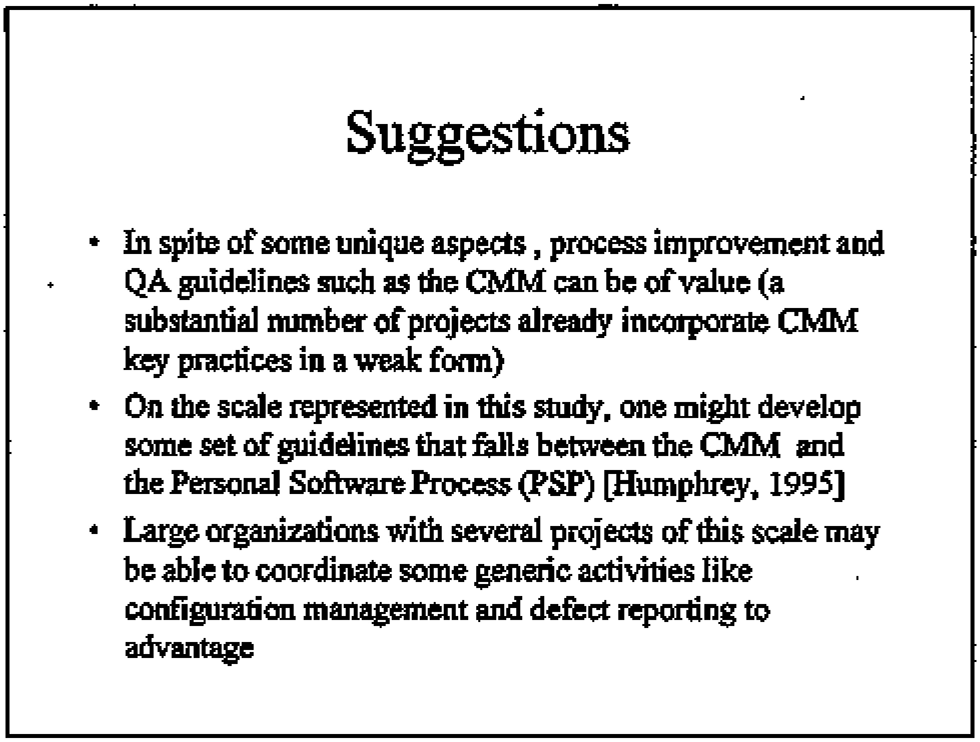
About the Exploratory Process

- Two interesting features stand out
  - The process is a scientific process rather than an engineering process (as it should be)
  - At almost every stage, the aim is to manage the risk of exploring unknown territory
- The difference:
  - Conventional software process models are based on process definition, process control and management of resources
  - In exploration, the emphasis is on not getting lost



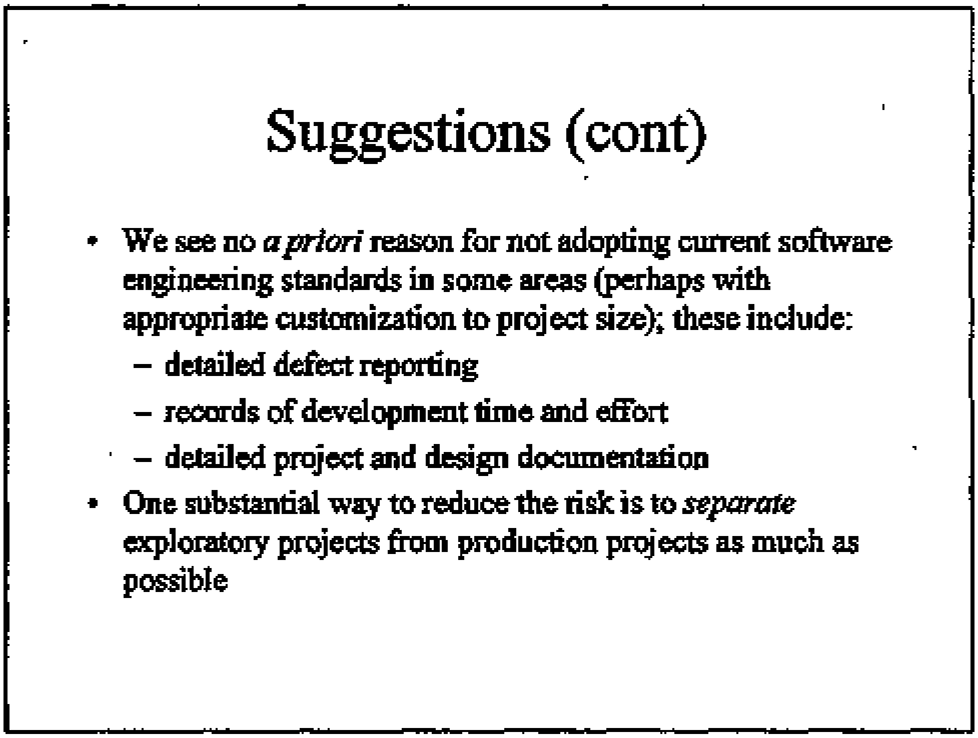
Is There Something Special About Simulation Development? Yes

- Imprecise requirements
- Higher risks in design and implementation
- The potential for open-ended testing and validation
- Strong links to legacy code
- Relatively small project size



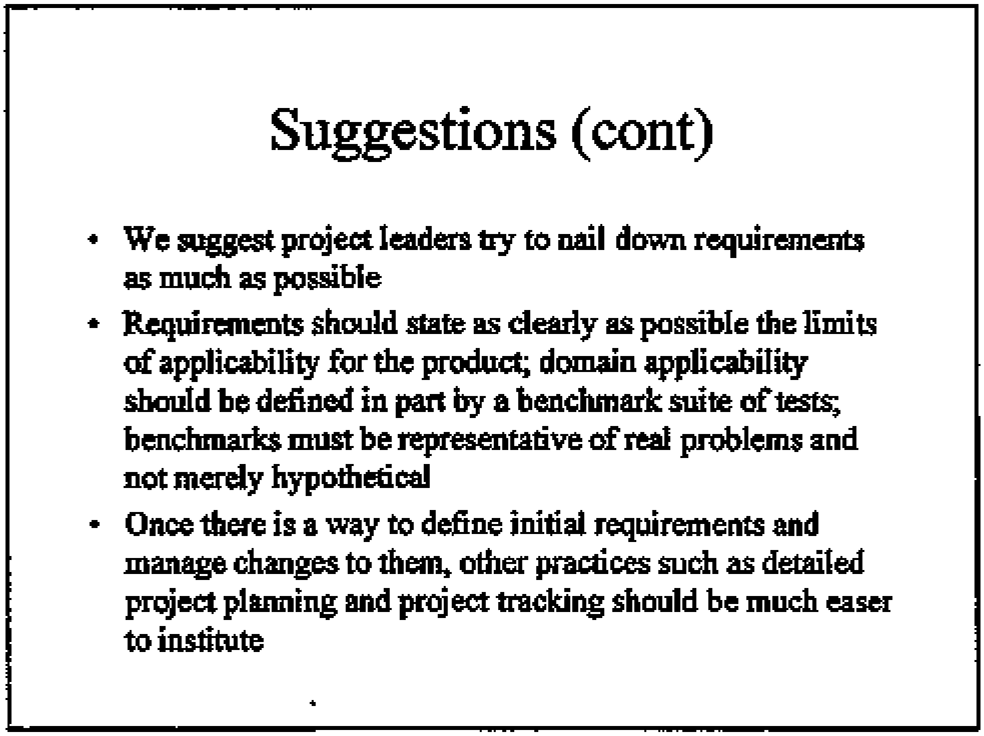
Suggestions

- In spite of some unique aspects , process improvement and QA guidelines such as the CMM can be of value (a substantial number of projects already incorporate CMM key practices in a weak form)
- On the scale represented in this study, one might develop some set of guidelines that falls between the CMM and the Personal Software Process (PSP) [Humphrey, 1995]
- Large organizations with several projects of this scale may be able to coordinate some generic activities like configuration management and defect reporting to advantage



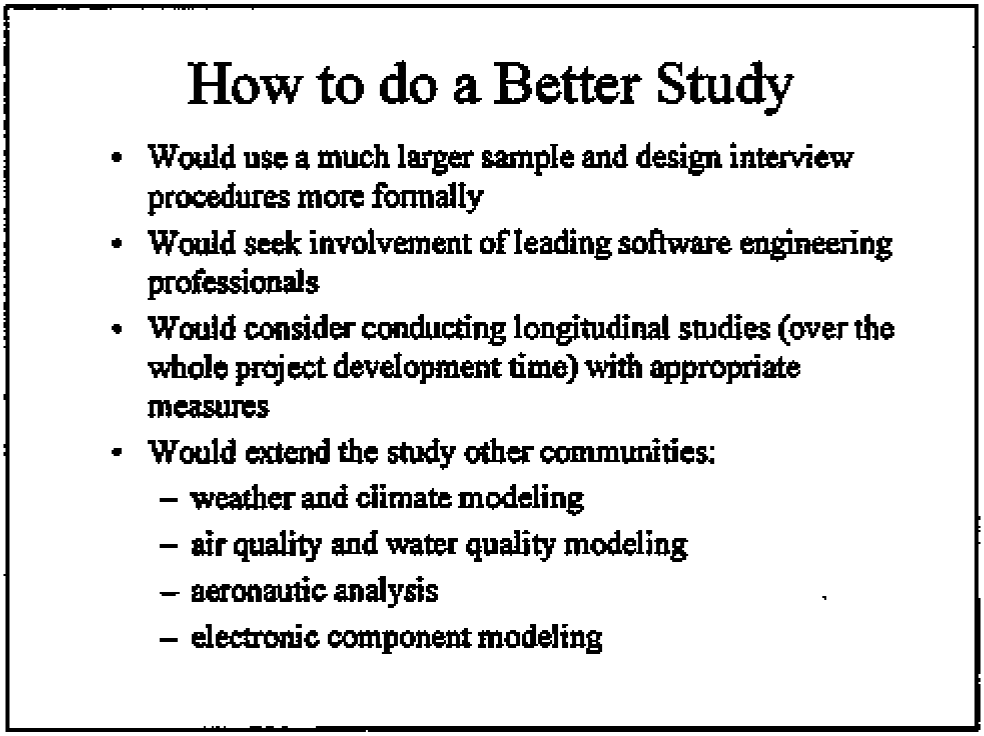
Suggestions (cont)

- We see no *a priori* reason for not adopting current software engineering standards in some areas (perhaps with appropriate customization to project size); these include:
  - detailed defect reporting
  - records of development time and effort
  - detailed project and design documentation
- One substantial way to reduce the risk is to *separate* exploratory projects from production projects as much as possible



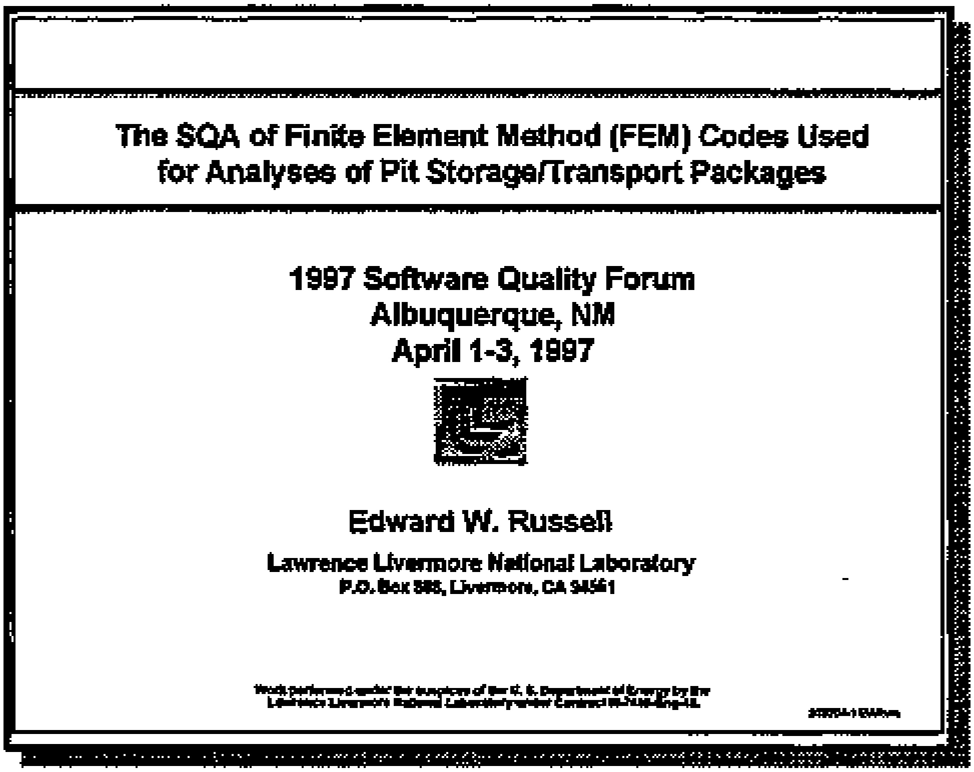
Suggestions (cont)

- We suggest project leaders try to nail down requirements as much as possible
- Requirements should state as clearly as possible the limits of applicability for the product; domain applicability should be defined in part by a benchmark suite of tests; benchmarks must be representative of real problems and not merely hypothetical
- Once there is a way to define initial requirements and manage changes to them, other practices such as detailed project planning and project tracking should be much easer to institute



How to do a Better Study

- Would use a much larger sample and design interview procedures more formally
- Would seek involvement of leading software engineering professionals
- Would consider conducting longitudinal studies (over the whole project development time) with appropriate measures
- Would extend the study other communities:
  - weather and climate modeling
  - air quality and water quality modeling
  - aeronautic analysis
  - electronic component modeling



The SQA of Finite Element Method (FEM) Codes Used for Analyses of Pit Storage/Transport Packages

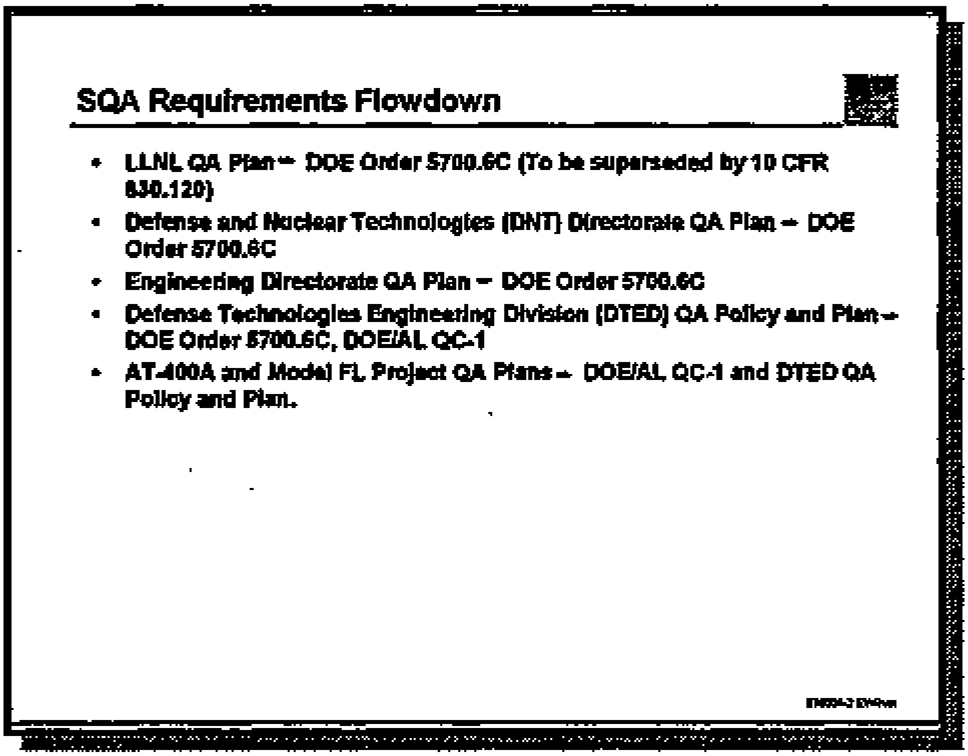
1997 Software Quality Forum
Albuquerque, NM
April 1-3, 1997

Edward W. Russell

Lawrence Livermore National Laboratory
P.O. Box 808, Livermore, CA 94551

Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

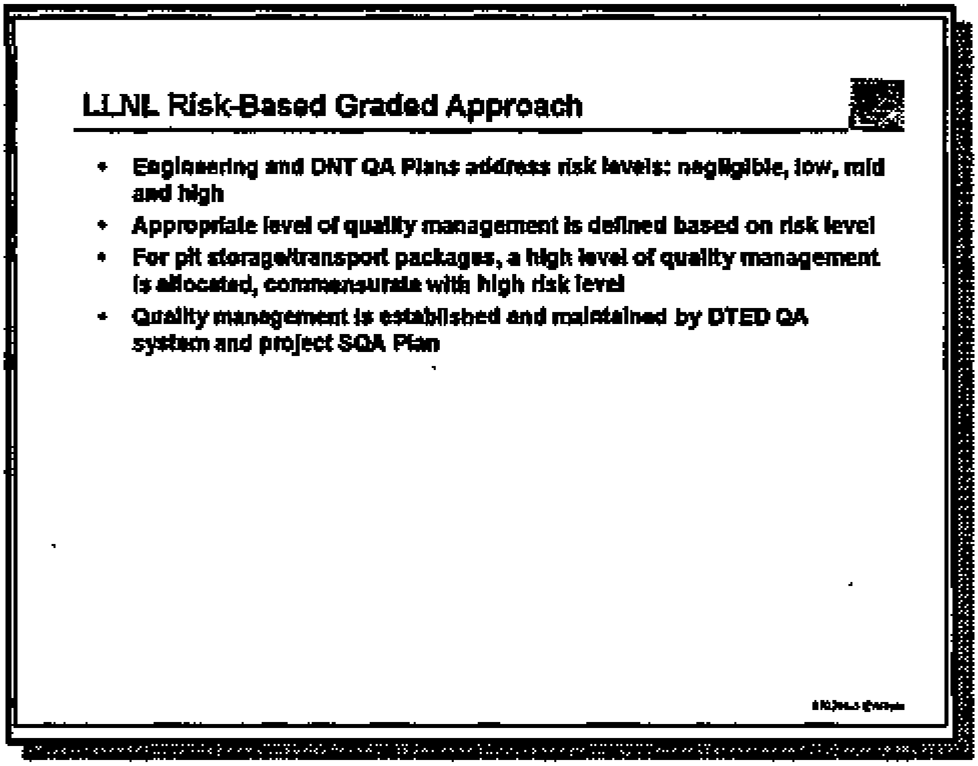
[illegible]



SQA Requirements Flowdown

- LLNL QA Plan – DOE Order 5700.6C (To be superseded by 10 CFR 830.120)
- Defense and Nuclear Technologies (DNT) Directorate QA Plan – DOE Order 5700.6C
- Engineering Directorate QA Plan – DOE Order 5700.6C
- Defense Technologies Engineering Division (DTED) QA Policy and Plan – DOE Order 5700.6C, DOE/AL QC-1
- AT-400A and Model FL Project QA Plans – DOE/AL QC-1 and DTED QA Policy and Plan.

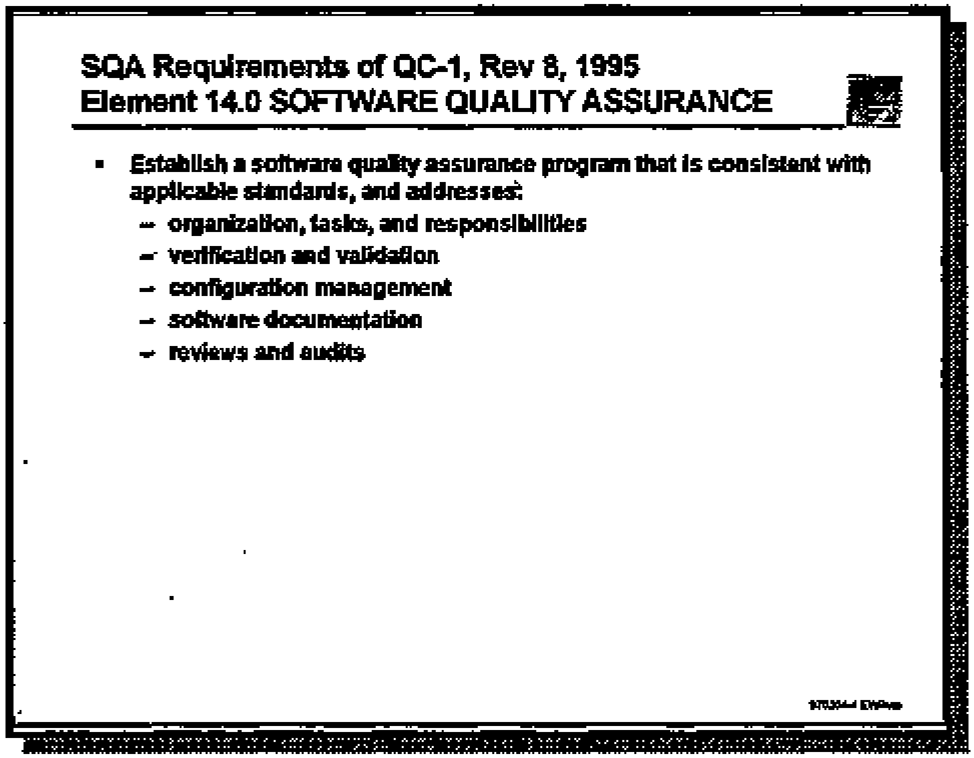
[illegible]



LLNL Risk-Based Graded Approach

- Engineering and DNT QA Plans address risk levels: negligible, low, mid and high
- Appropriate level of quality management is defined based on risk level
- For pit storage/transport packages, a high level of quality management is allocated, commensurate with high risk level
- Quality management is established and maintained by DTED QA system and project SQA Plan

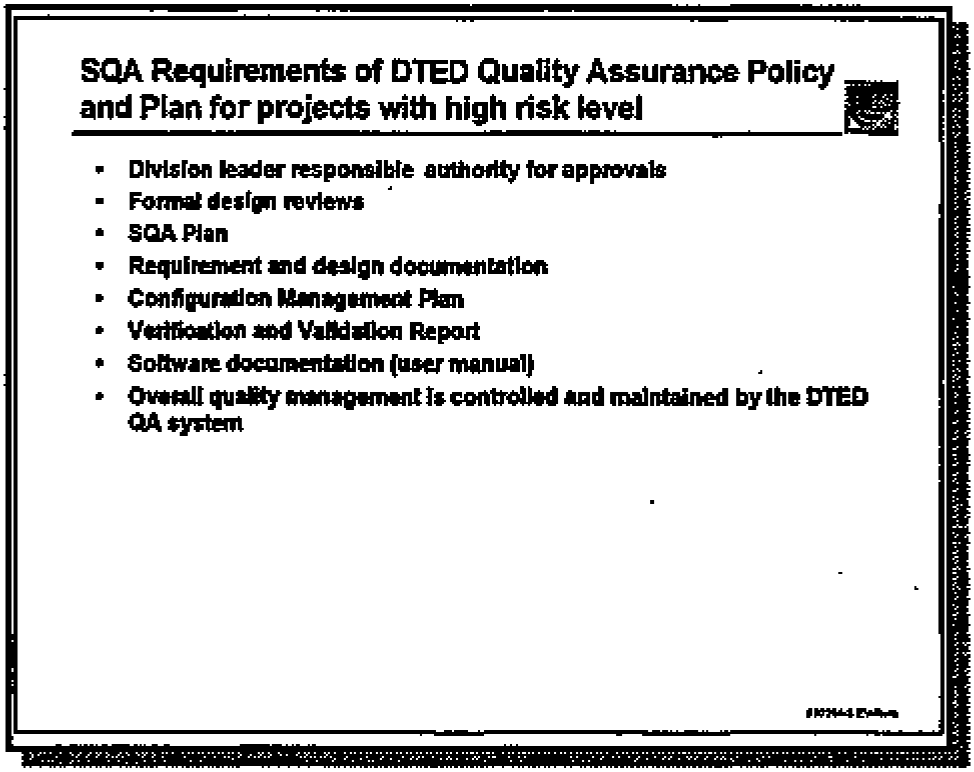
[illegible]



SQA Requirements of QC-1, Rev 8, 1995 Element 14.0 SOFTWARE QUALITY ASSURANCE

- Establish a software quality assurance program that is consistent with applicable standards, and addresses:
  - organization, tasks, and responsibilities
  - verification and validation
  - configuration management
  - software documentation
  - reviews and audits

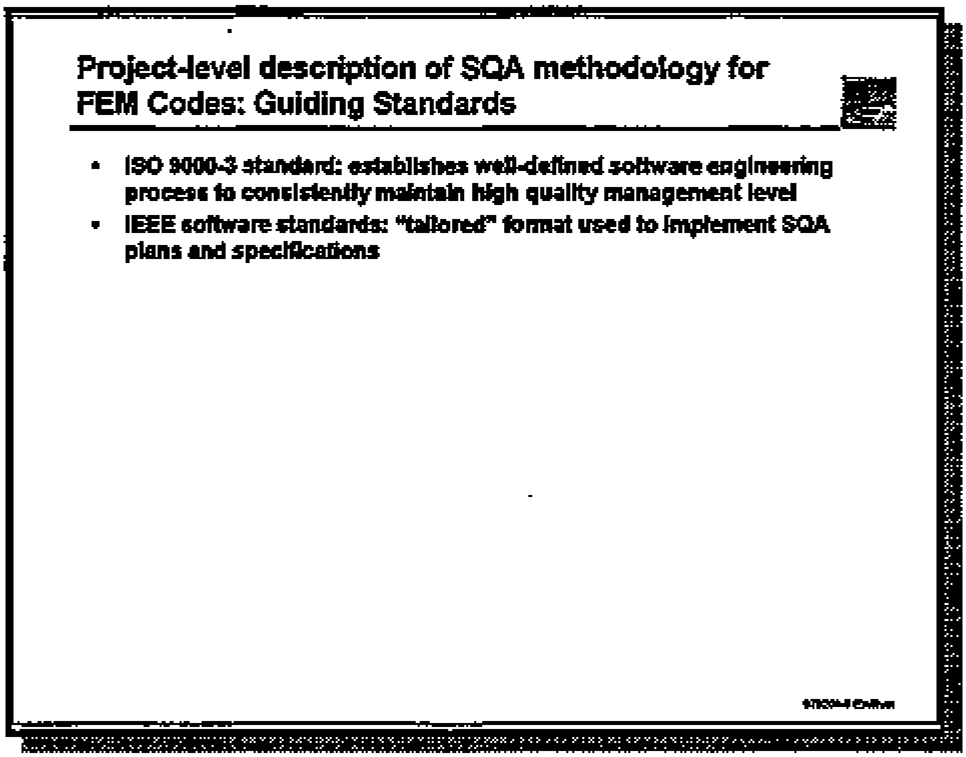
[illegible]



SQA Requirements of DTED Quality Assurance Policy and Plan for projects with high risk level

- Division leader responsible authority for approvals
- Formal design reviews
- SQA Plan
- Requirement and design documentation
- Configuration Management Plan
- Verification and Validation Report
- Software documentation (user manual)
- Overall quality management is controlled and maintained by the DTED QA system

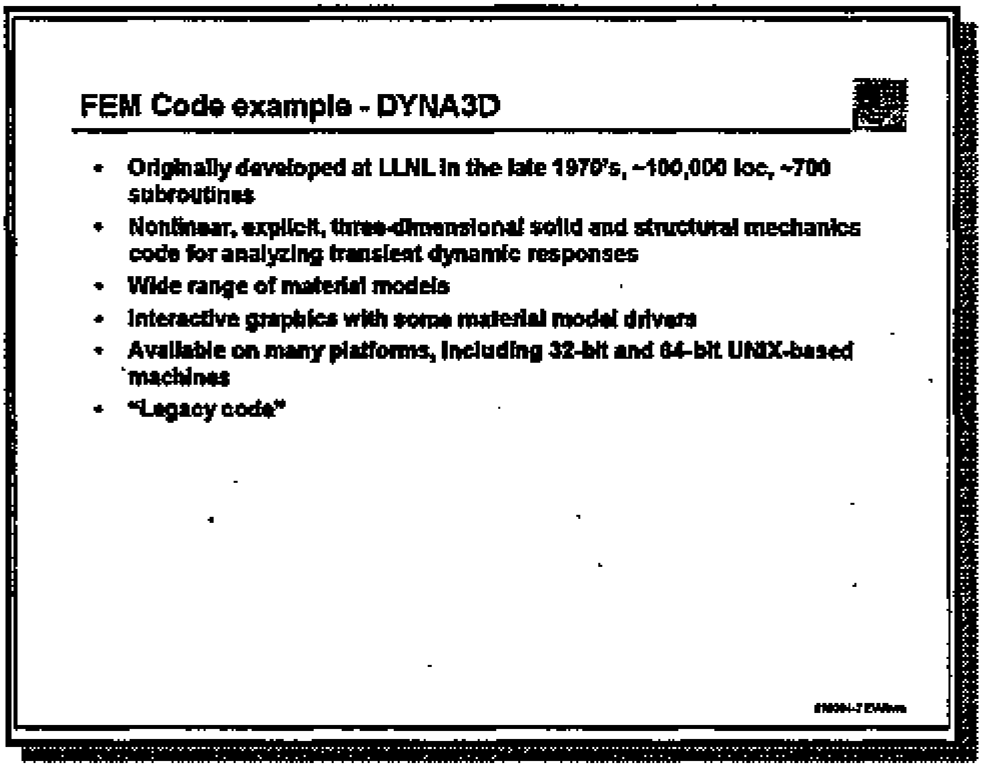
[illegible]



Project-level description of SQA methodology for FEM Codes: Guiding Standards

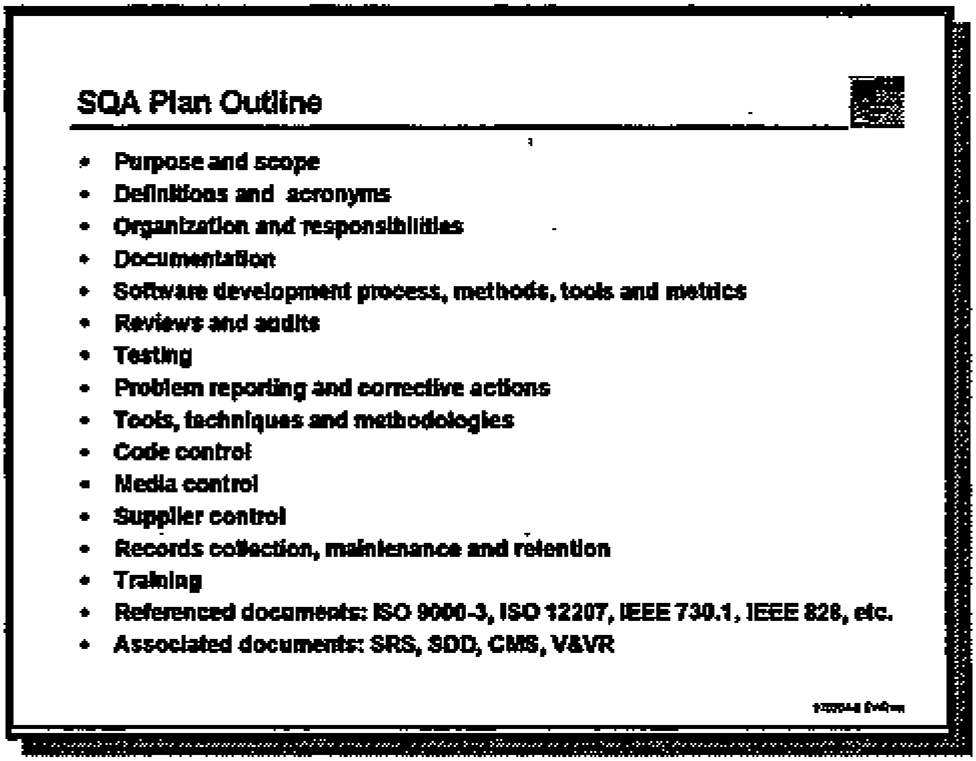
- ISO 9000-3 standard: establishes well-defined software engineering process to consistently maintain high quality management level
- IEEE software standards: "tailored" format used to implement SQA plans and specifications

[illegible]



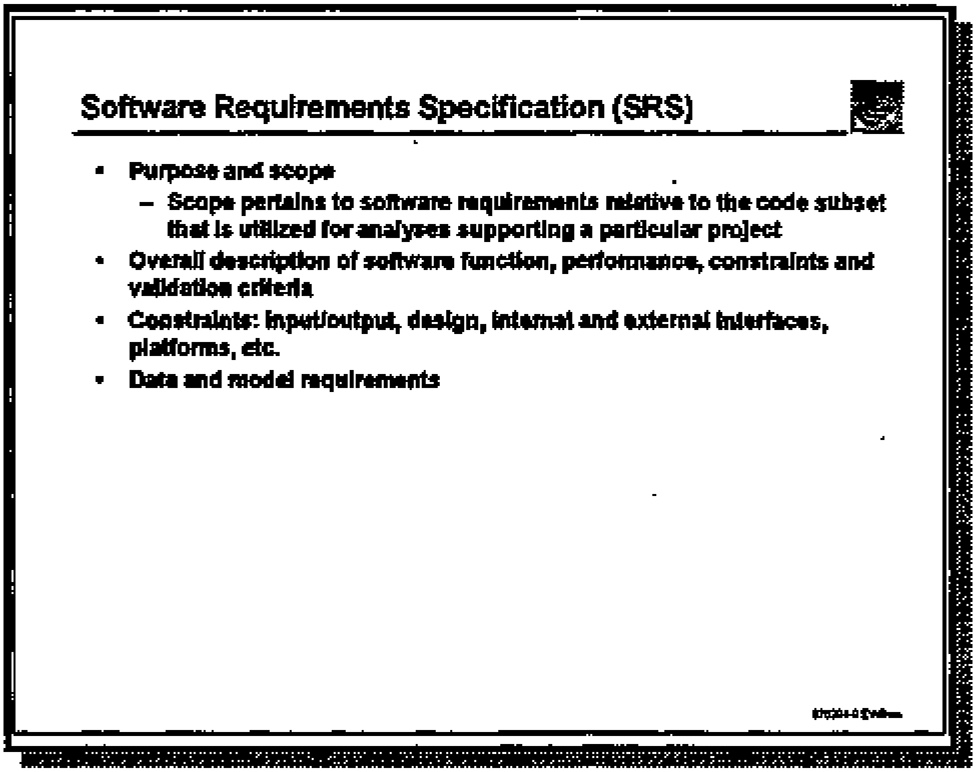
FEM Code example - DYNA3D

- Originally developed at LLNL in the late 1970's, ~100,000 loc, ~700 subroutines
- Nonlinear, explicit, three-dimensional solid and structural mechanics code for analyzing transient dynamic responses
- Wide range of material models
- Interactive graphics with some material model drivers
- Available on many platforms, including 32-bit and 64-bit UNIX-based machines
- "Legacy code"



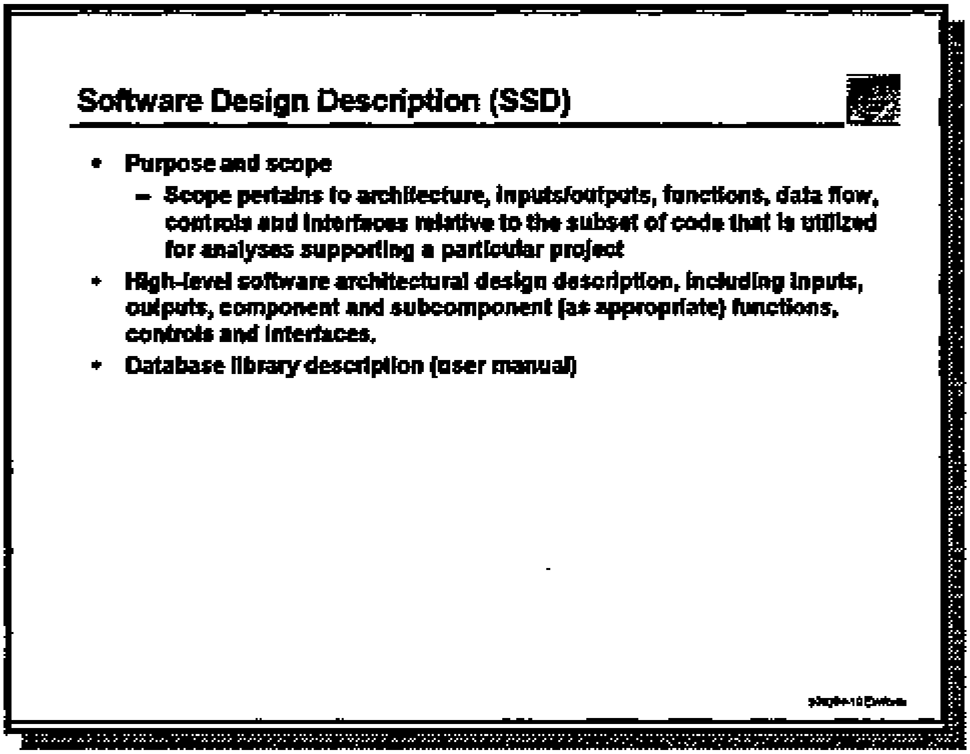
SQA Plan Outline

- Purpose and scope
- Definitions and acronyms
- Organization and responsibilities
- Documentation
- Software development process, methods, tools and metrics
- Reviews and audits
- Testing
- Problem reporting and corrective actions
- Tools, techniques and methodologies
- Code control
- Media control
- Supplier control
- Records collection, maintenance and retention
- Training
- Referenced documents: ISO 9000-3, ISO 12207, IEEE 730.1, IEEE 828, etc.
- Associated documents: SRS, SDD, CMS, V&VR



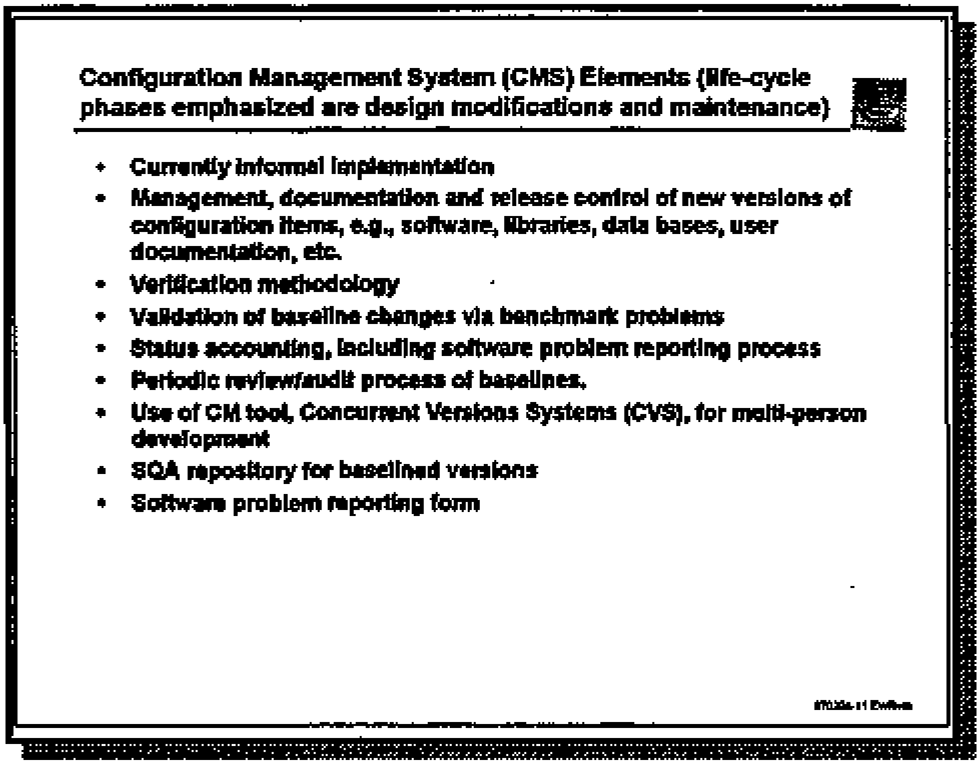
Software Requirements Specification (SRS)

- Purpose and scope
  - Scope pertains to software requirements relative to the code subset that is utilized for analyses supporting a particular project
- Overall description of software function, performance, constraints and validation criteria
- Constraints: input/output, design, internal and external interfaces, platforms, etc.
- Data and model requirements



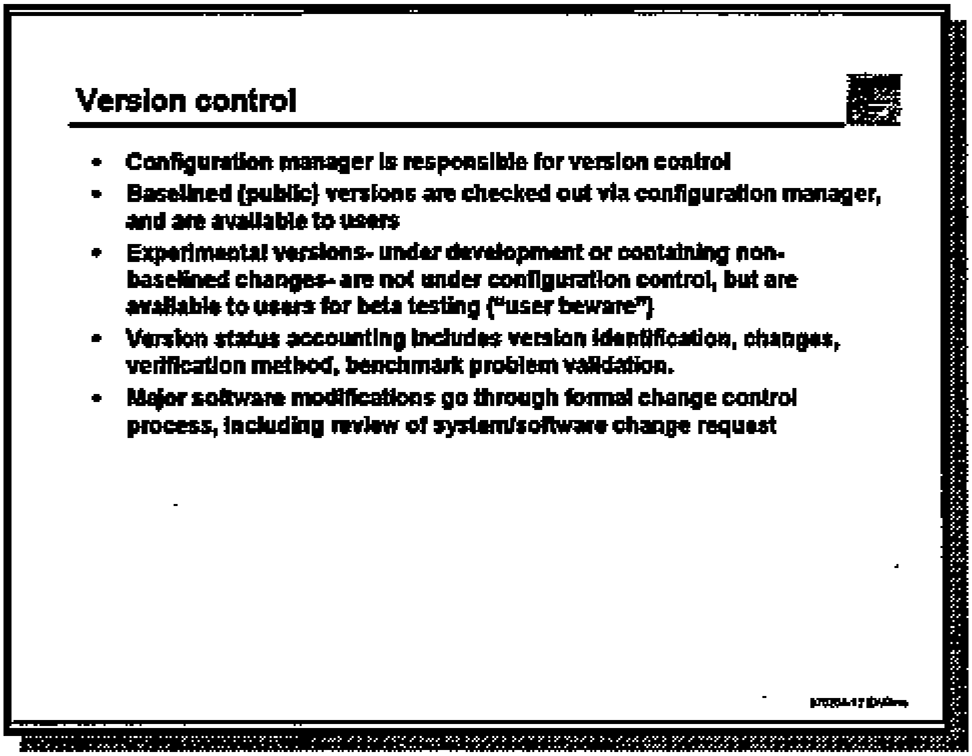
Software Design Description (SSD)

- Purpose and scope
  - Scope pertains to architecture, inputs/outputs, functions, data flow, controls and interfaces relative to the subset of code that is utilized for analyses supporting a particular project
- High-level software architectural design description, including inputs, outputs, component and subcomponent (as appropriate) functions, controls and interfaces.
- Database library description (user manual)



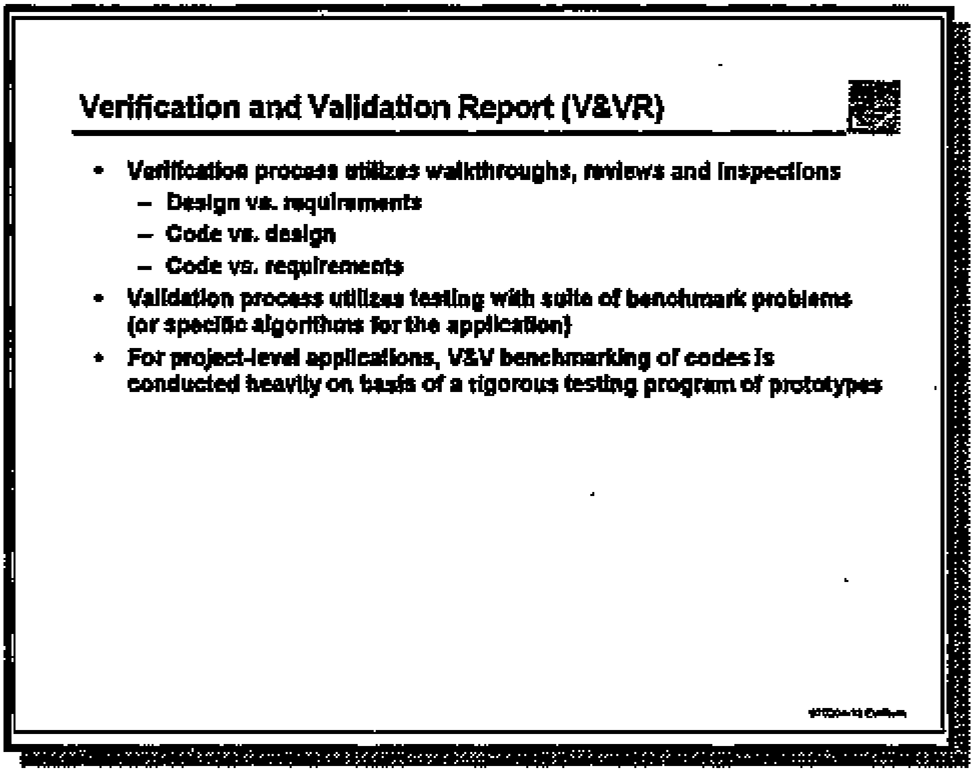
Configuration Management System (CMS) Elements (life-cycle phases emphasized are design modifications and maintenance)

- Currently informal implementation
- Management, documentation and release control of new versions of configuration items, e.g., software, libraries, data bases, user documentation, etc.
- Verification methodology
- Validation of baseline changes via benchmark problems
- Status accounting, including software problem reporting process
- Periodic review/audit process of baselines.
- Use of CM tool, Concurrent Versions Systems (CVS), for multi-person development
- SQA repository for baselined versions
- Software problem reporting form



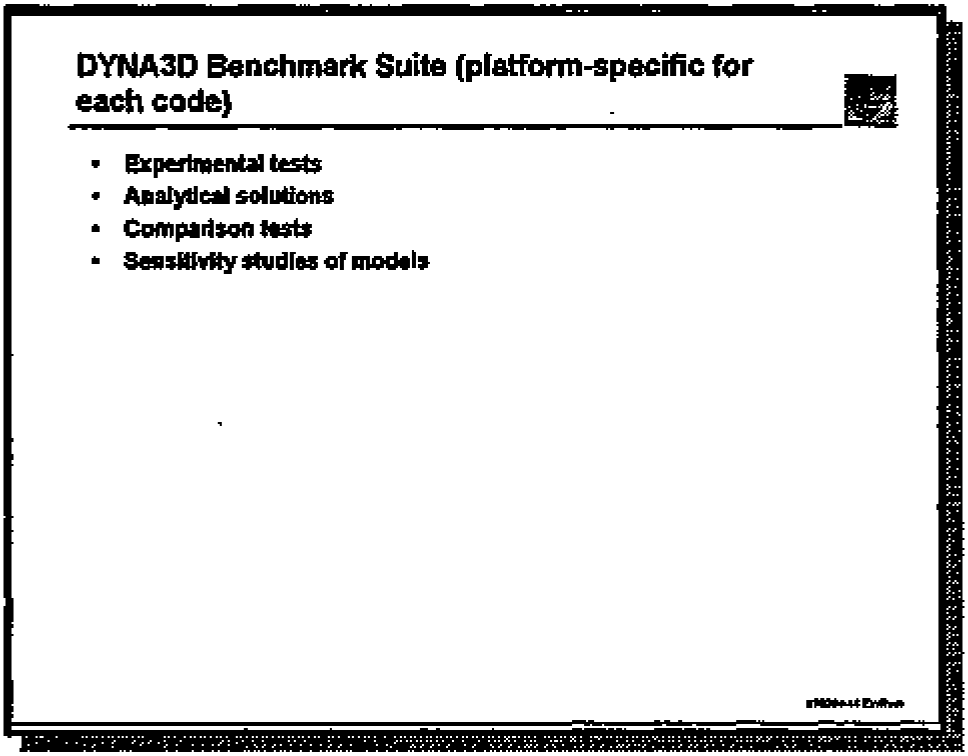
Version control

- Configuration manager is responsible for version control
- Baselined (public) versions are checked out via configuration manager, and are available to users
- Experimental versions- under development or containing non-baselined changes- are not under configuration control, but are available to users for beta testing ("user beware")
- Version status accounting includes version identification, changes, verification method, benchmark problem validation.
- Major software modifications go through formal change control process, including review of system/software change request



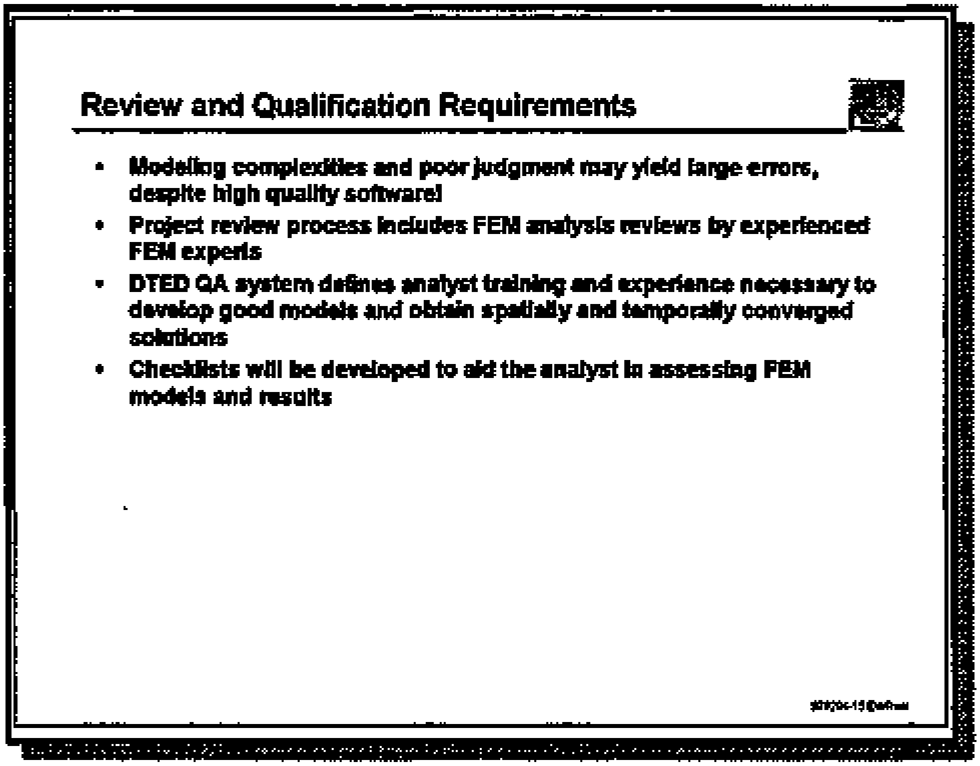
Verification and Validation Report (V&VR)

- Verification process utilizes walkthroughs, reviews and inspections
  - Design vs. requirements
  - Code vs. design
  - Code vs. requirements
- Validation process utilizes testing with suite of benchmark problems (or specific algorithms for the application)
- For project-level applications, V&V benchmarking of codes is conducted heavily on basis of a rigorous testing program of prototypes



DYNA3D Benchmark Suite (platform-specific for each code)

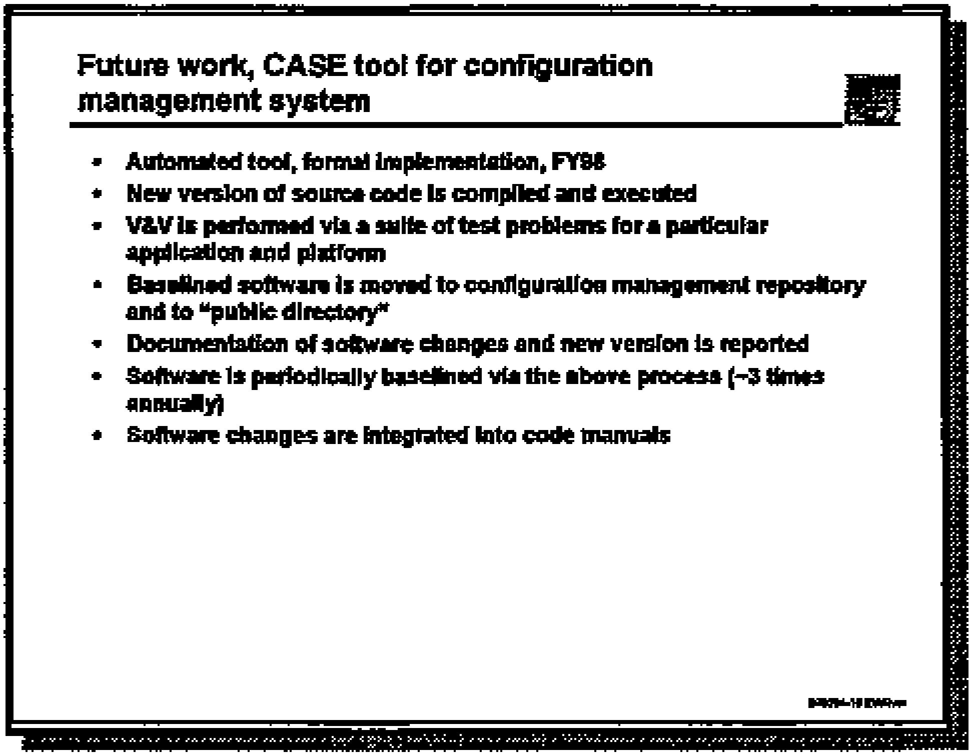
- Experimental tests
- Analytical solutions
- Comparison tests
- Sensitivity studies of models



Review and Qualification Requirements

- Modeling complexities and poor judgment may yield large errors, despite high quality software!
- Project review process includes FEM analysis reviews by experienced FEM experts
- DTED QA system defines analyst training and experience necessary to develop good models and obtain spatially and temporally converged solutions
- Checklists will be developed to aid the analyst in assessing FEM models and results

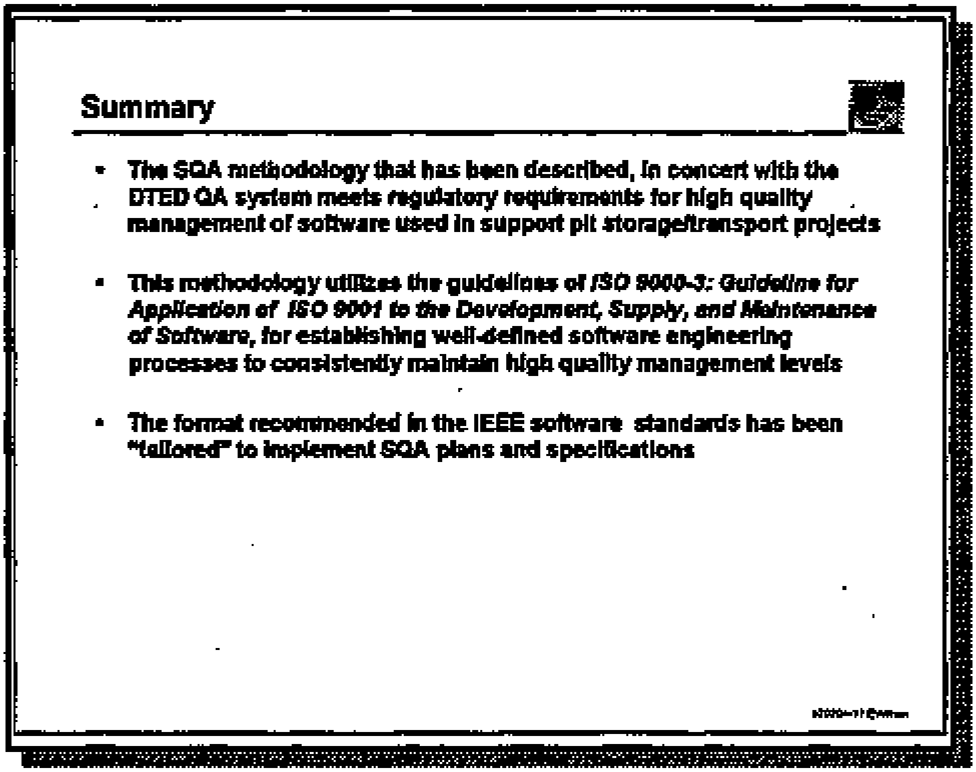
[illegible]



Future work, CASE tool for configuration management system

- Automated tool, formal implementation, FY98
- New version of source code is compiled and executed
- V&V is performed via a suite of test problems for a particular application and platform
- Baselined software is moved to configuration management repository and to "public directory"
- Documentation of software changes and new version is reported
- Software is periodically baselined via the above process (~3 times annually)
- Software changes are integrated into code manuals

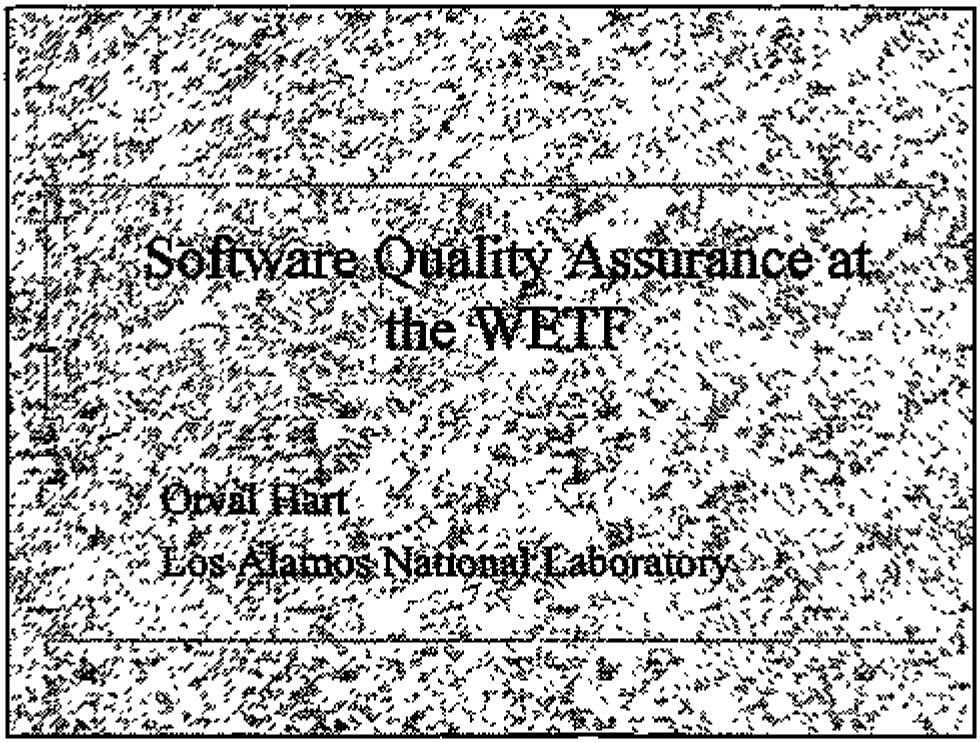
[illegible]



Summary

- The SQA methodology that has been described, in concert with the DTED QA system meets regulatory requirements for high quality management of software used in support pit storage/transport projects
- This methodology utilizes the guidelines of *ISO 9000-3: Guideline for Application of ISO 9001 to the Development, Supply, and Maintenance of Software*, for establishing well-defined software engineering processes to consistently maintain high quality management levels
- The format recommended in the IEEE software standards has been "tailored" to implement SQA plans and specifications

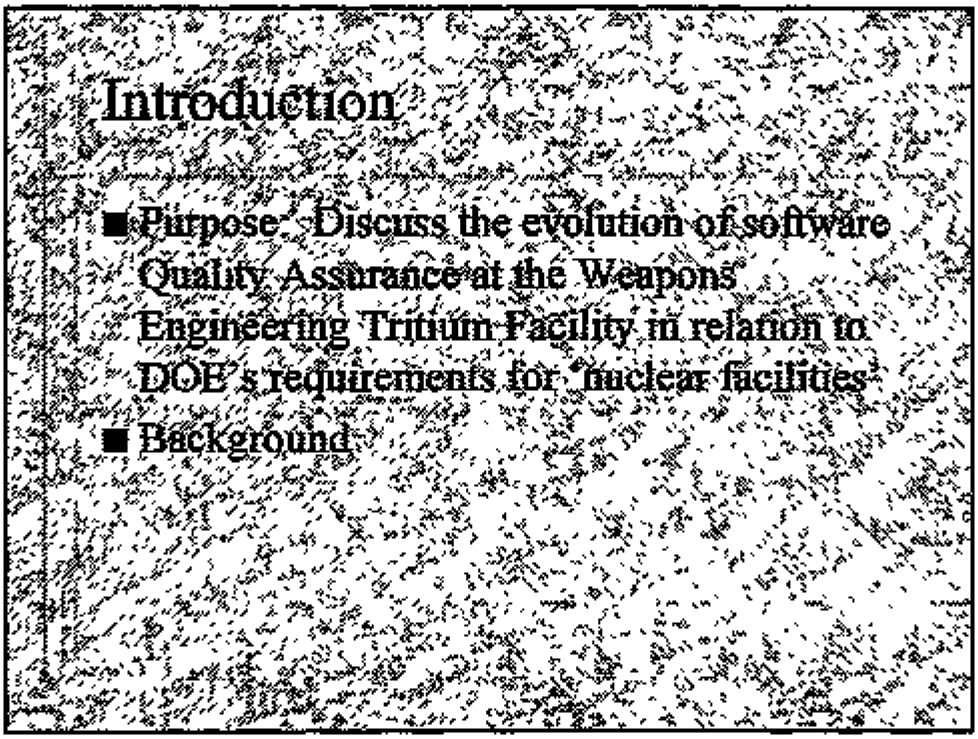
[illegible]



Software Quality Assurance at the WETF

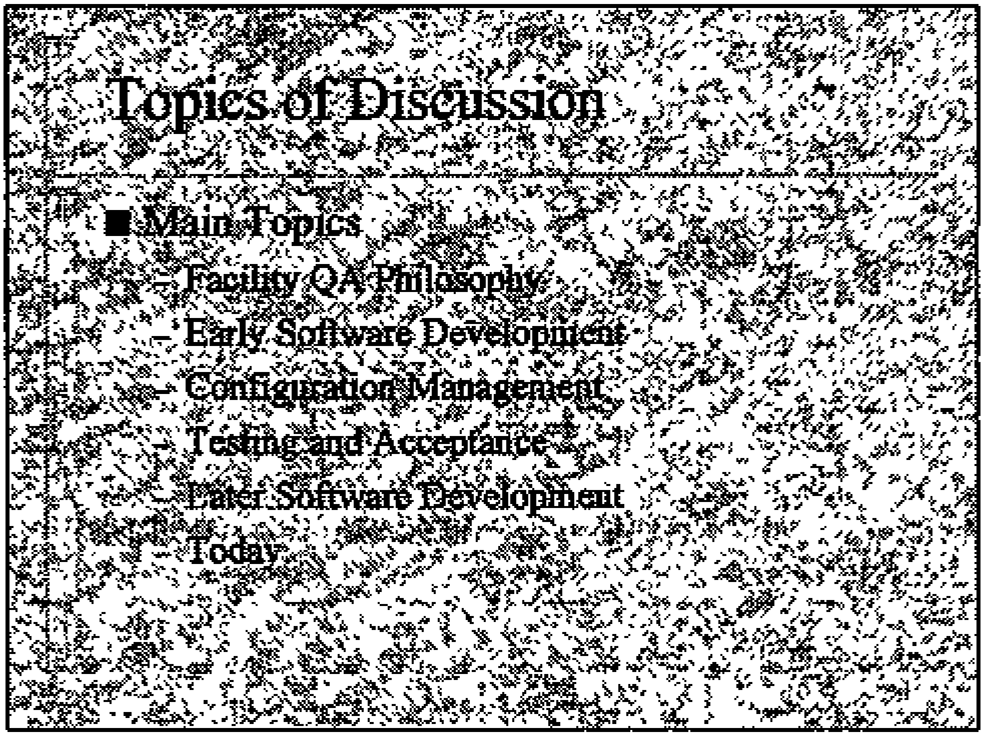
Orval Hart

Los Alamos National Laboratory



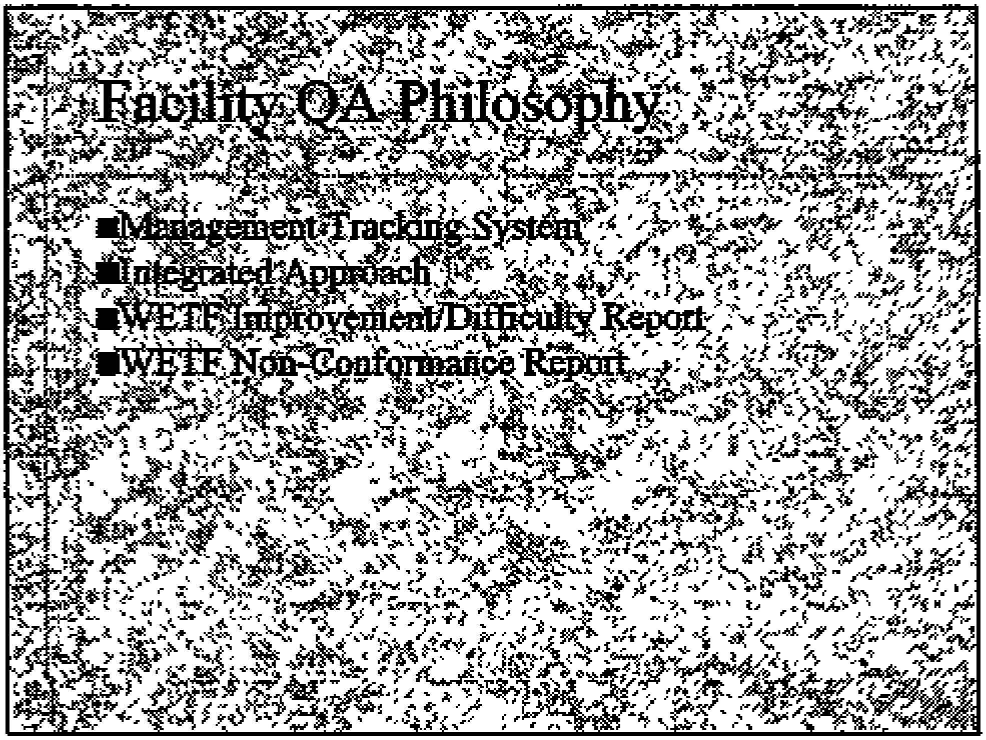
Introduction

- Purpose: Discuss the evolution of software Quality Assurance at the Weapons Engineering Tritium Facility in relation to DOE's requirements for "nuclear facilities"
- Background



Topics of Discussion

- Main Topics
  - Facility QA Philosophy
  - Early Software Development
  - Configuration Management
  - Testing and Acceptance
  - Later Software Development
  - Today



Facility QA Philosophy

- Management Tracking System
- Integrated Approach
- WETF Improvement/Difficulty Report
- WETF Non-Conformance Report

WETF IMPROVEMENT/DIFFICULTY REPORT

000448

Improvement/[✓ Difficulty] Description (circle one) | WIDR #

An intermittent stack tritium monitor flow alarm will fail the ICS stack monitor status. This, in turn, will record only "fail" on the daily stack integrated T2 report even though the flow has returned to normal.

Safety-Related?	no
Signature	
Typed Name	R.L. Hemphill
Date of Report	5-18-92

WIDR Index Description: Stack Monitor Intermittent flow fails S.M. daily report

Disposition of Improvement/Difficulty

Dispoation Steps

Diagnose & repair so data is still available. T2 monitor should latch fails but integration data should be written to file.

Configuration Control? (Software)	Yes
Corrective Maintenance?	
Non-Conformance Report	
SWP	
RWP	
Work Order	
Priority	High

TSM Signature, Date(s): Robert L. Nolen Jr. 6/22/92

Functional Titles

1. Designer/Originator
2. Section Leader
3. Tritium Systems Manager
4. Building Manager
5. ENG Division
6. Database Custodian
7. ICS Hardware Specialist
8. Equipment Custodian
9. HSE Division Representative
10. WETF Electronic Technician
11. Nuclear Material Custodian
12. Facility Coordinator
13. Database Designer
14. ICS Software Specialist
15. Mech Tech

Associated Document Changes

Databases	Procedures	Software	Drawings

WIDR Review/Assignment Documentation

Title	Review 1	Review 2	Review 3	Review 4	Action Required	Action Completed	Action Checked By
2	KL 6/22/92	[illegible] 8/4/92					
3	[illegible] 6/29/92	[illegible] 8/10/92					
1/12	[illegible]	[illegible]					[illegible] 8/3/92
10	BC 6/23/92						
14	[illegible] 7/13/92	[illegible] 8/10/92			✓	8/3/92 [illegible]	
15	WUR 6/23/92						
9	[illegible] 8/10/92						

Equipment/Component Identifiers

Serial Number		Room #		Glovebox	
Manufacturer		Measurement List Parameter Name			
Model Number		Symbol			
LANL Property Number					
Item Description					
Other Equipment/Components To Be Identified?		N/A			

Form No. WETF 01

Revision No. 3

Date December 12, 1990

Page No. 1

WIDR (continued)

Comments

WIDR #

CONDITIONS			
Stack failed	1	1	1
No voltage	1	0	0
Low flow	—	1	0
Actions			
measure RM		X	X
"fail"	X		
"f"		X	X

If the stack is failed because of low flow then
measure the RM but set a flag indicating the low flow condition

if the stack is failed because of bad instrument power then
don't measure the RM and store the failed condition in the measurement value.

Action Taken

Added the ability to set a low flow flag in the stack report file when the stack RM is measured during low flow conditions.

Added the ability to add an "f" to RM values measured during low flow conditions when printing the stack reports.

Action Completed (TSM)	Brian Jones
Completion Date	5/10/92

WETF SYSTEM TEST

Test No. 448

Page 1 of 2

Summary of Test Objectives

To verify that the stack integrator & its reports function as specified when:

a) low flow condition in stack monitor

b) no instrument power

c) normal operating conditions

Preliminary □

In-Use ☒

Other □

Responds to WIDR (s)

448

By: O. Hart
Submitter

Reviewed:

Approved:

Date: 8/3/92

TEST PREREQUISITES

Test S/W Installed	Ver.	Configuration Used for Test	Verified By	Facility Configuration	Verified By
ETC	2.27	Normal	O. Hart Test Conductor	No Alterations Required ☒ Alterations Required. Ref. □	Test Witness

TEST RESULTS

☒ Test Results are acceptable

□ Retest required.* Reference Test Log for: ____________
Date(s)

* In-Use Test Failures, Only.

Reviewed by and Date

O. Hart
Test Conductor

8/3/92
Test Witness

POST TEST ACTIONS

Op. S/W Installed	Ver.	Other Actions	Verified By and Date	Facility Configuration	Verified By and Date
yes 8/3/92	2.27	N/A	O. Hart Test Conductor	Restored to Pretest Configuration □ As Noted below □ N/A	8/3/92 Test Witness

4/22/91, Rev. 0

	WETF COMPU. .(SYSTEM TEST	Test. 148
Test Item No.	Test Procedure - Including Inputs and Anticipated Outputs	Page 2 of 2
1.	While FLIC/CAMAC running, bring up ~~FEE~~ ETC 2.27. Note time at which code comes up. Let ETC operater for a minimum of two minutes. Put ETC in RUN using 21800.	
2. →	Set stack flow alarm using 20031 (EFA-EXHSK, DI 43). This will generate a low flow alarm, as well as the "TH STACK ~~FAIL~~ FLAG". Let ETC operate for a minimum of 2 minutes, noting the time at which the low flow was initiated.	
3,	Reset ~~Clear~~ stack flow alarm using 20031. ~~Reset~~ stack monitor fail using, 21100. Note times at which performed. Let ETC operate for a minimum of 2 minutes.	
4.	Set bad voltage alarm using 20031 (EVA-EXHSK, DI 4). This will generate a "TM-STK VOLT STATUS" alarm, as well as the "TH STACK FAIL FLAG". Let ETC operate for a minimum of 2 minutes, noting the time at which the bad voltage ~~low flow~~ was initiated.	
4.	Reset voltage bad status alarm using 20031. Reset stack monitor using 21100. Note times at which performed. Let ETC operate for a minimum of 2 minutes.	
5.	Using 21860, stop the ETC code. Note time at which code stopped.	
	DEVELOPMENT SYSTEM	
6.	In the test directory (:UDD:NTF:ETC), perform ETC_REPORT_2 "filename", where filename matches the stack data filename, e.g., STACK_920731.DAT. The stack report is printed on the printer. Verify that stack report reflects actions at time taken.	

```
 7/31 10:48:34 HVM-H312A      RACK- H3 +12V PWR, ANALOG              9.42 LO        U
 7/31 10:50:14 EZS-ETCRUN     ETC SMODE RUN                             1 OK        C
 7/31 10:50:24 GOM-LI         LOAD IN- G.B. O2                      15.88 HI        A
 7/31 10:50:24 GOM-LI         LOAD IN- G.B. O2                      15.88 HI        C
 7/31 10:50:28 GOM-LI         LOAD IN- G.B. O2                        .16 OK        A
 7/31 10:50:28 GOM-LI         LOAD IN- G.B. O2                        .16 OK        C
 7/31 10:53: 6 FLM-WH2OTK     WASTE WATER TANK LEVEL              1148.23 OK        A
 7/31 10:53:23 HVM-H312A      RACK- H3 +12V PWR, ANALOG             12.19 OK        A
 7/31 10:53:23 HVM-H312A      RACK- H3 +12V PWR, ANALOG             12.19 OK        U
 10:55:32
DIID  1125 - DIIDS  - CHANGED TO:  SET
 7/31 10:55:43 WPM-TK5L1      SLP1- PR.                            826.35 HI        A
 7/31 10:56: 6 EZS-ETCRUN     ETC SMODE RUN                             0 LO        C
 10:58:27
DIID  1104 - DIIDS  - CHANGED TO:  SET
 10:58:39
DIID    43 - DIIDS  - CHANGED TO:  SET
 7/31 10:58:39 EFA-EXHSK      TM- STK FLOW ALARM                        1 HI        A
 10:59:35
DIID    43 - DIIDS  - CHANGED TO:  RESET
 7/31 10:59:36 EFA-EXHSK      TM- STK FLOW ALARM                        0 OK        A
 7/31 11: 0:10 EZS-ETCRUN     ETC SMODE RUN  ETC Code Startup           1 OK        C
 11: 0:17
DIID  1104 - DIIDS  - CHANGED TO:  SET
 7/31 11: 0:20 EZS-ETCNULL    ETC CMODE NULL  ETC in AUTO               0 OK        A
***** THE TIME IS 11:00       THE DATE IS 07/31/92 *****
 11: 1:30
`IID    43 - DIIDS  - CHANGED TO:  SET
 7/31 11: 1:31 EFA-EXHSK      TM- STK FLOW ALARM  initiate low flow     1 HI        A
 7/31 11: 1:33 EZS-EXHSKFAI   TM STACK FAIL FLAG                        1 HI        U
 11: 5:30
DIID    43 - DIIDS  - CHANGED TO:  RESET
 7/31 11: 5:31 EFA-EXHSK      TM- STK FLOW ALARM  reset low flow        0 OK        A
 11: 5:40
DIID  1144 - DIIDS  - CHANGED TO:  RESET
 7/31 11: 5:40 EZS-EXHSKFAI   TM STACK FAIL FLAG  reset fail flag       0 OK        U
 7/31 11: 6:13 HVM-H312A      RACK- H3 +12V PWR, ANALOG              9.52 LO        A
 7/31 11: 6:13 HVM-H312A      RACK- H3 +12V PWR, ANALOG              9.52 LO        U
 7/31 11: 6:42 HVM-H312A      RACK- H3 +12V PWR, ANALOG             12.19 OK        A
 7/31 11: 6:42 HVM-H312A      RACK- H3 +12V PWR, ANALOG             12.19 OK        U
 11: 8:36
DIID     4 - DIIDS  - CHANGED TO:  RESET
 7/31 11: 8:36 EVA-EXHSK      TM- STK VOLT STATUS initiate bad Power supply 0 LO    A
 7/31 11: 8:38 EZS-EXHSKFAI   TM STACK FAIL FLAG                        1 HI        U
 7/31 11: 9: 2 FLM-WH2OTK     WASTE WATER TANK LEVEL              1201.58 HI        A
 7/31 11:10:41 WPM-TK5L1      SLP1- PR.                            950.06 HI        C
 7/31 11:15:21 FLM-WH2OTK     WASTE WATER TANK LEVEL              1146.13 OK        A
 11:16:31
DIID     4 - DIIDS  - CHANGED TO:  SET
 7/31 11:16:32 EVA-EXHSK      TM- STK VOLT STATUS reset bad power supply 1 OK       A
 11:16:40
DIID  1144 - DIIDS  - CHANGED TO:  RESET       reset fail flag
 7/31 11:16:41 EZS-EXHSKFAI   TM STACK FAIL FLAG                        0 OK        U
 .1:17:31
.IID  1125 - DIIDS  - CHANGED TO:  SET
 7/31 11:17:32 EZS-ETCNULL    ETC CMODE NULL  ETC Code STOP             1 HI        A
 7/31 11:18: 4 EZS-ETCRUN     ETC SMODE RUN  ETC Code down              0 LO        C
 7/31 11:20:53 WPM-TK5U2      LPR PRESSURE (U2)                    351.71 HI        E
 7/31 11:21: 1 WPM-MB4        DRY22- TANK PRESSURE                 529.91 LO        A
 7/31 11:21: 3 WOM-HX6OT      MO-2 O2 MONITOR                         .27 LO        A
```

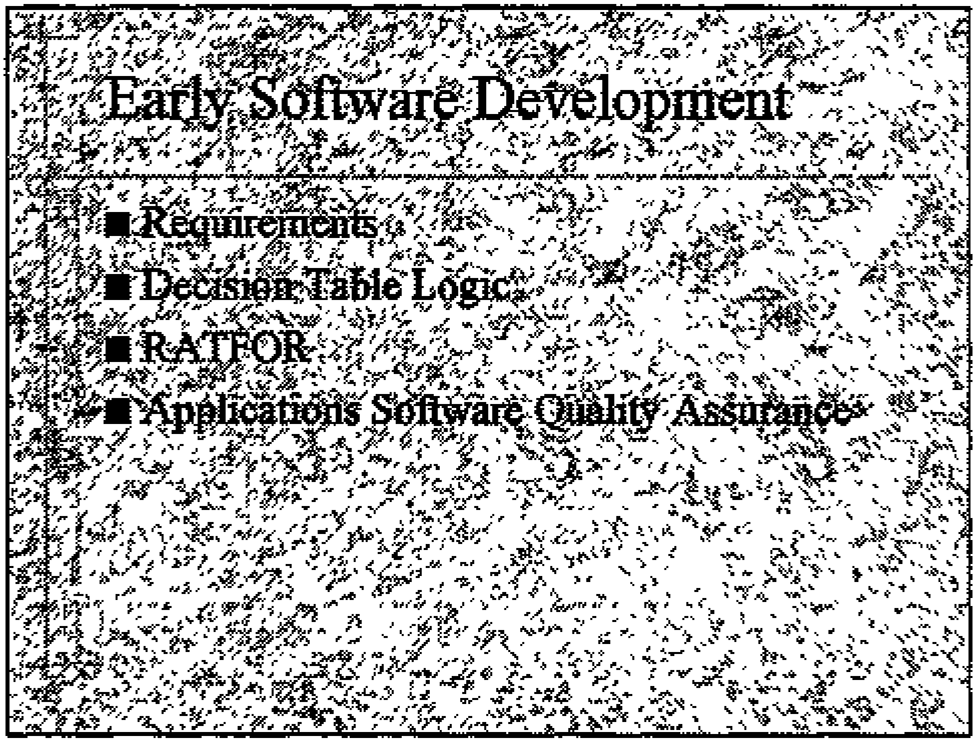
STACK INTEGRATED T2 REPORT FOR 07/31/92

```
value* at 0814 of:                             mmddyy hhmmss
24 hr accum       miss        mCi  last reset at: miss   miss
long-term accum   miss        mCi  last reset at: miss   miss
```

*increase in stack integrated T2 during the minute, in mCi

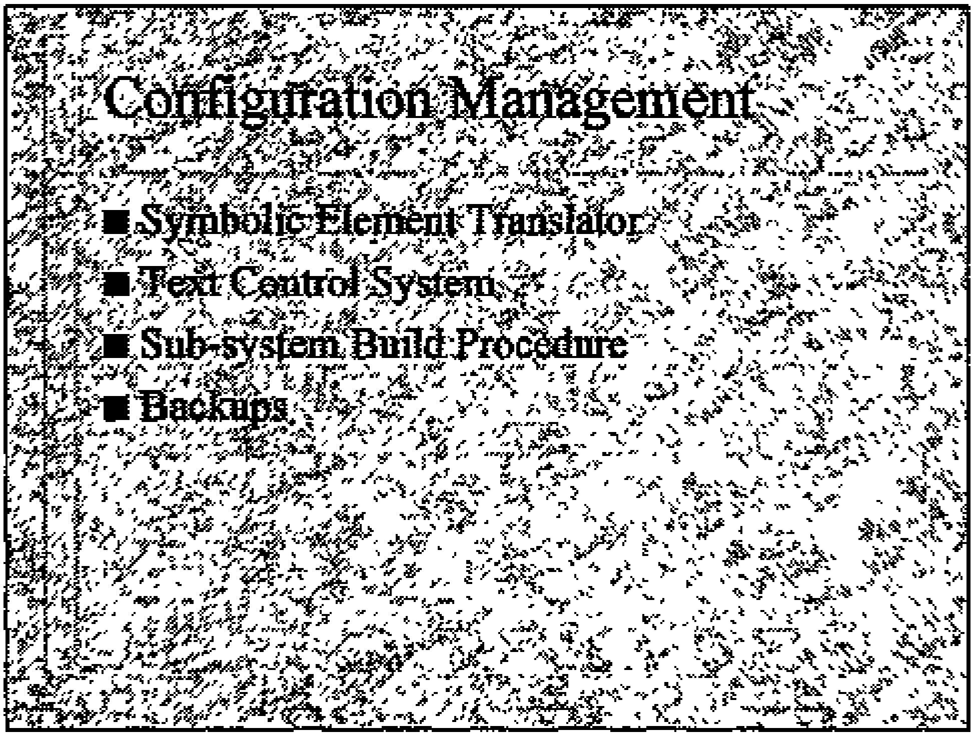
hour------

min.	00xx	01xx	02xx	03xx	04xx	05xx	06xx	07xx	08xx	09xx	10xx	11xx	12xx	13xx	14xx	15xx	16xx	17xx	18xx	19xx	20xx	21xx	22xx	23xx
00	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
01	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss 3	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
02	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss 4	0f	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
03	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss 4	0f	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
04	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss 5	0f	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
05	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss 6	0f	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
06	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
07	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
08	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
09	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
10	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
11	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
12	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
13	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
14	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
15	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
16	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	fail	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
17	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
18	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
19	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
20	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
21	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
22	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
23	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
24	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
25	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
26	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
27	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
28	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
29	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
30	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
31	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
32	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
33	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
34	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
35	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
36	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
37	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
38	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
39	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
40	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
41	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
42	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
43	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
44	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
45	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
46	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
47	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
48	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
49	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
50	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
51	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
52	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
53	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
54	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
55	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	0	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
56	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
57	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
58	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss
59	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss



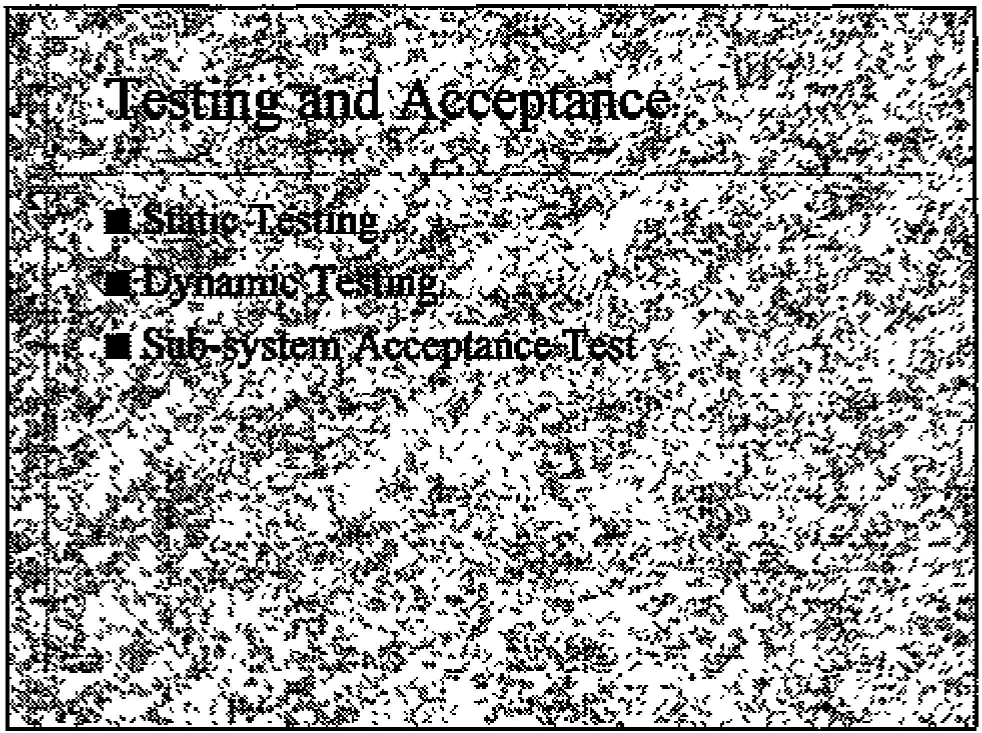
Early Software Development

- Requirements
- Decision Table Logic
- RATFOR
- Applications Software Quality Assurance



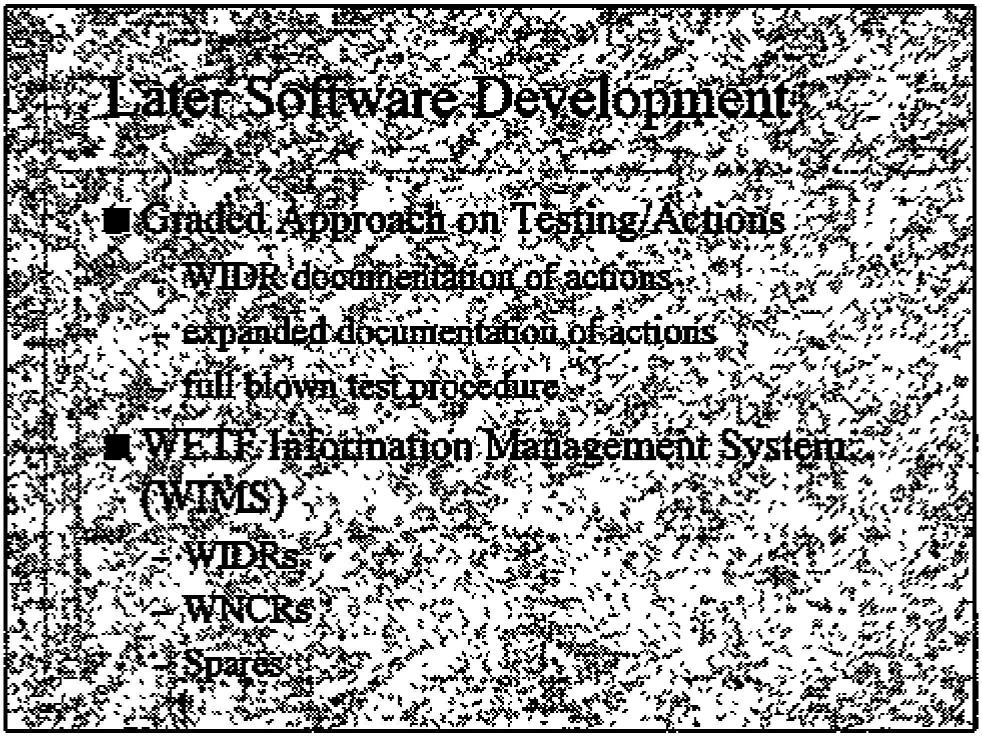
Configuration Management

- Symbolic Element Translator
- Text Control System
- Sub-system Build Procedure
- Backups



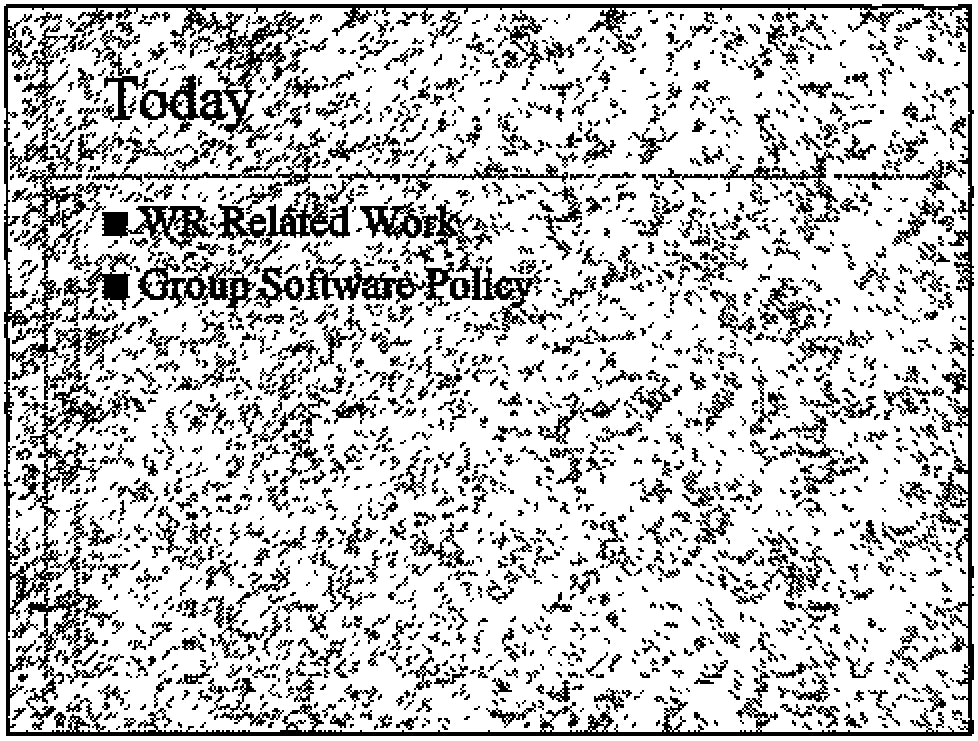
Testing and Acceptance

- ■ Static Testing
- ■ Dynamic Testing
- ■ Sub-system Acceptance Test



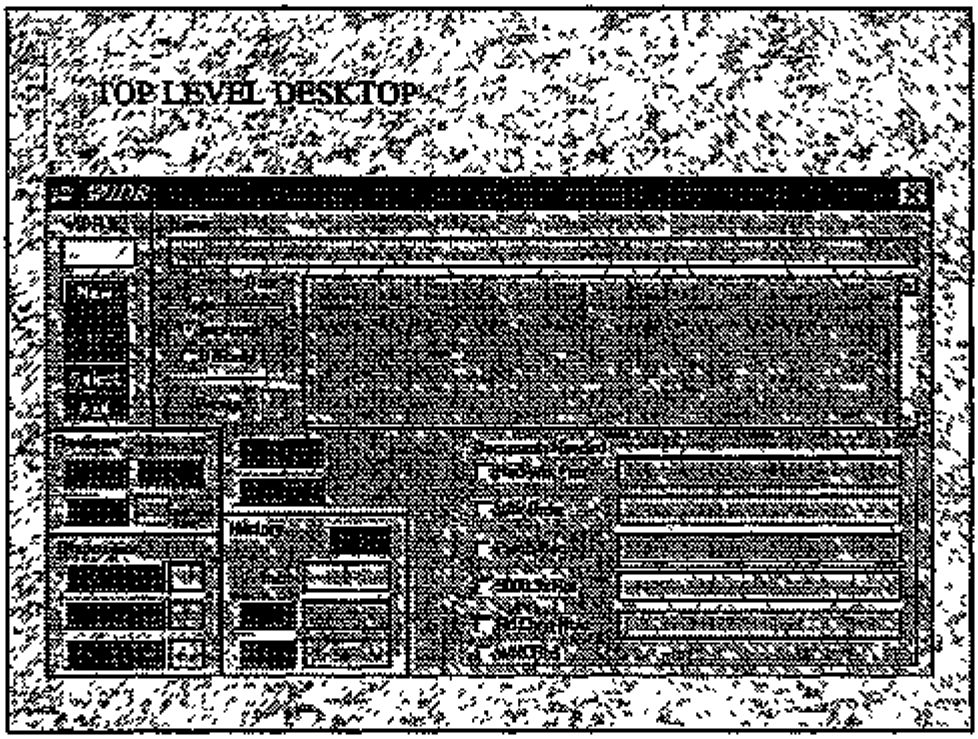
Later Software Development

- ■ Graded Approach on Testing/Actions
  - WIDR documentation of actions
  - expanded documentation of actions
  - full blown test procedure
- ■ WETF Information Management System (WIMS)
  - WIDRs
  - WNCRs
  - Spares



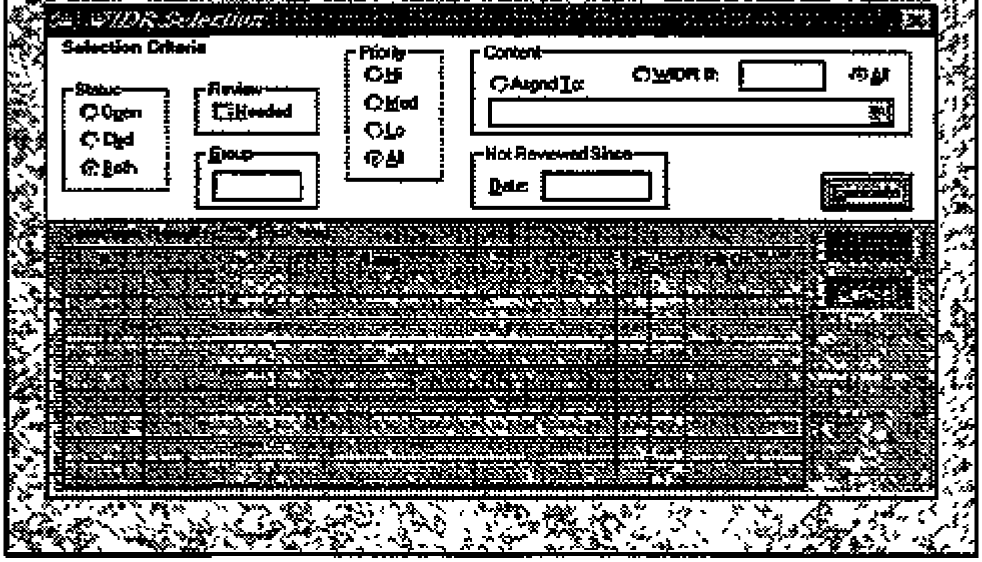
Today

- ■ WR Related Work
- ■ Group Software Policy



TOP LEVEL DESKTOP

WIDR SELECTION CRITERIA



WIDR Selection

Selection Criteria

Status: ○ Open ○ Died ● Both

Review: □ Needed

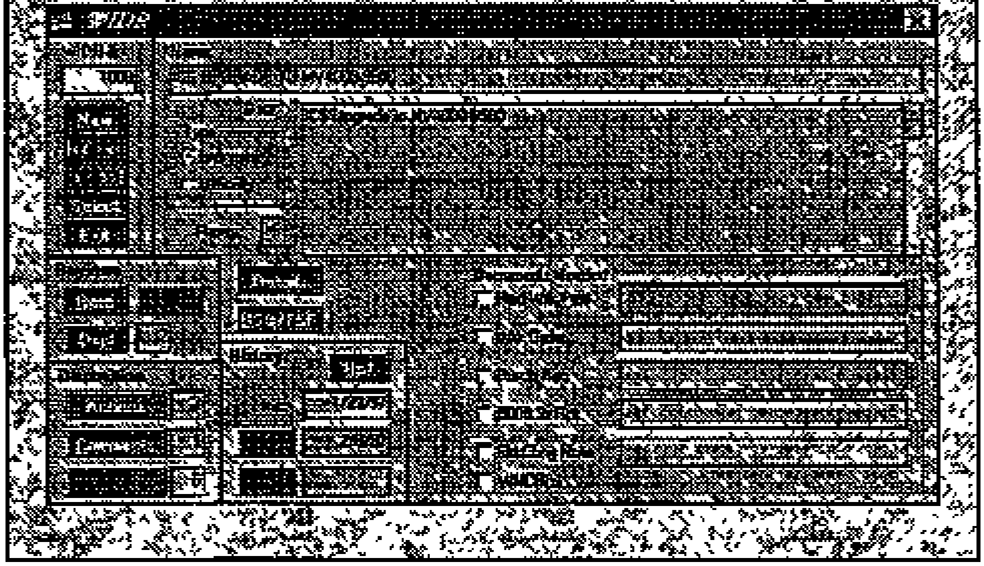
Group:

Priority: ○ Hi ○ Med ○ Lo ● All

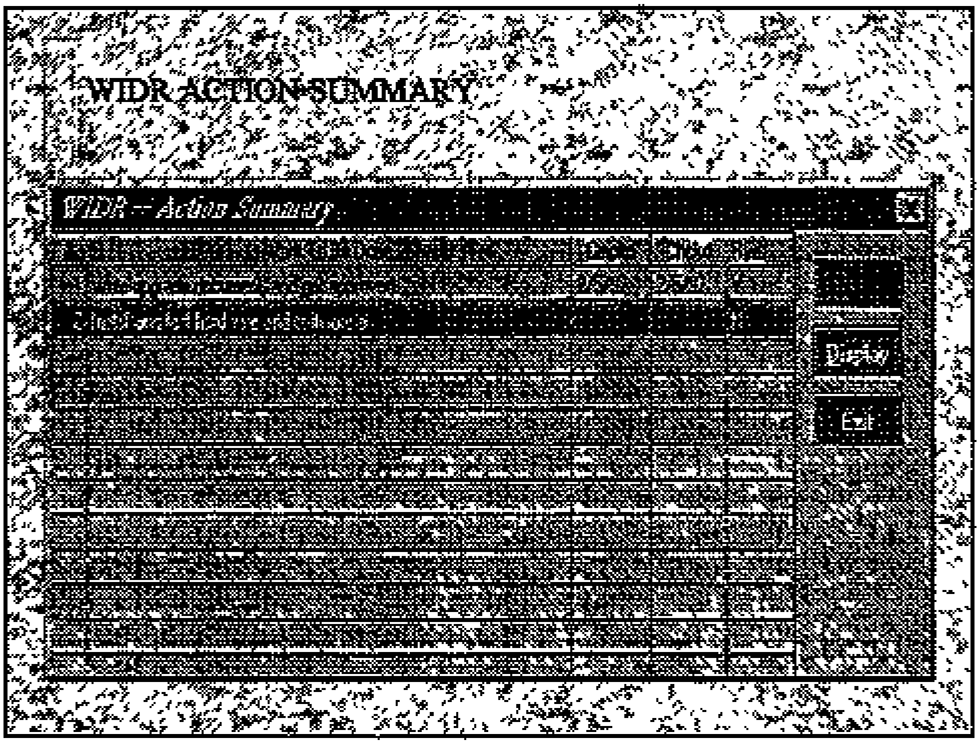
Content: ○ Asgnd To: ○ WIDR #: ● All

Not Reviewed Since: Date:

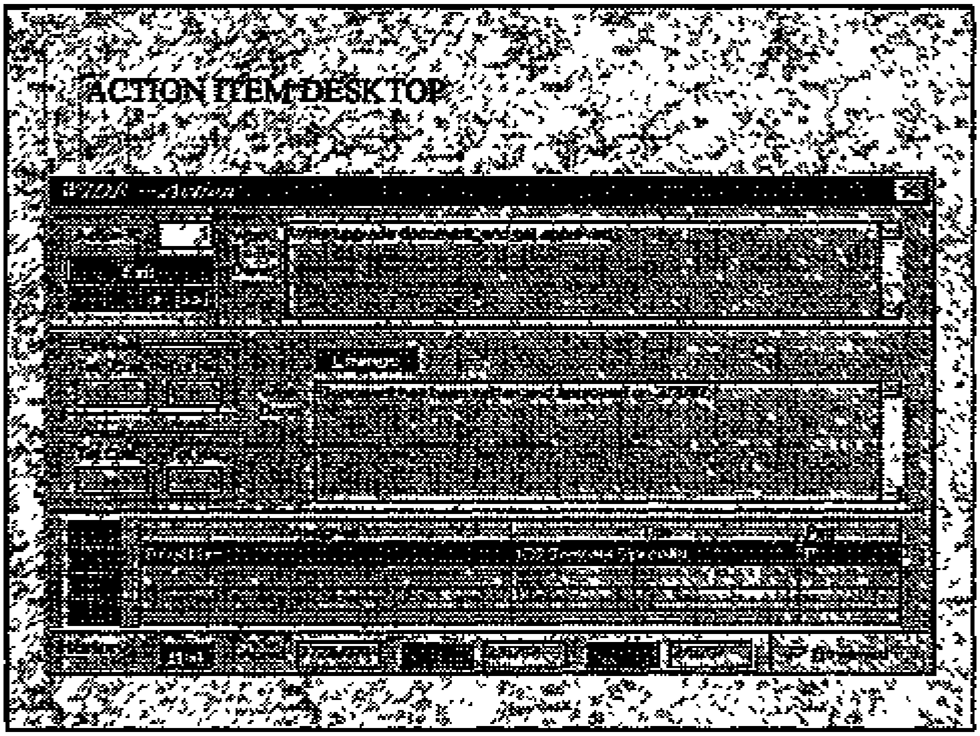
WIDR DESKTOP



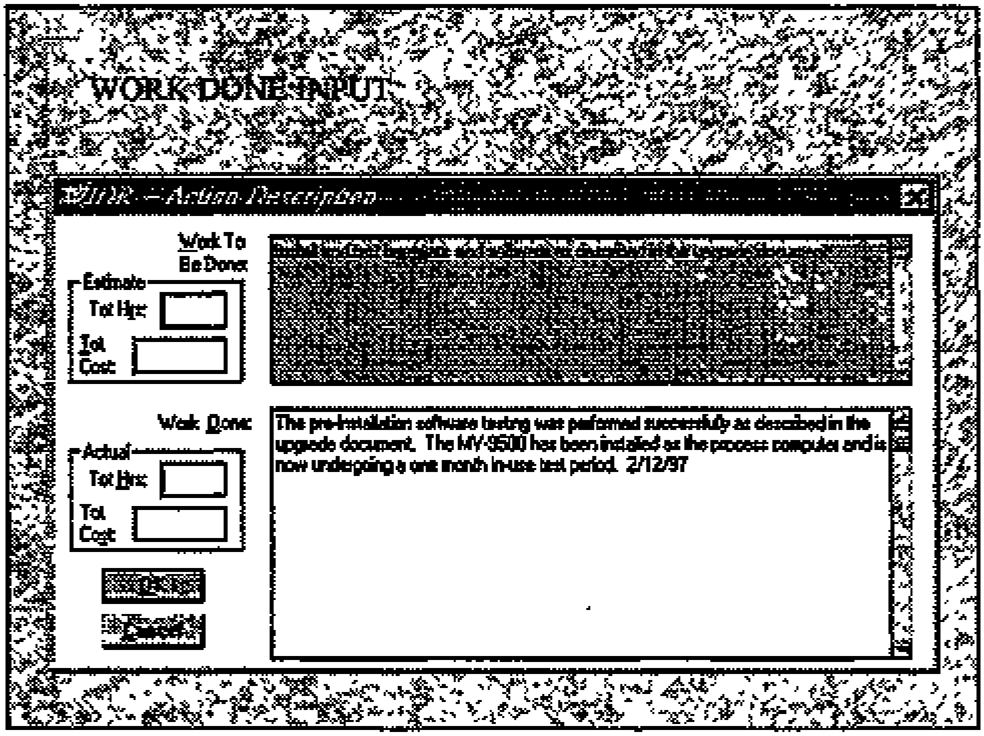
WIDR



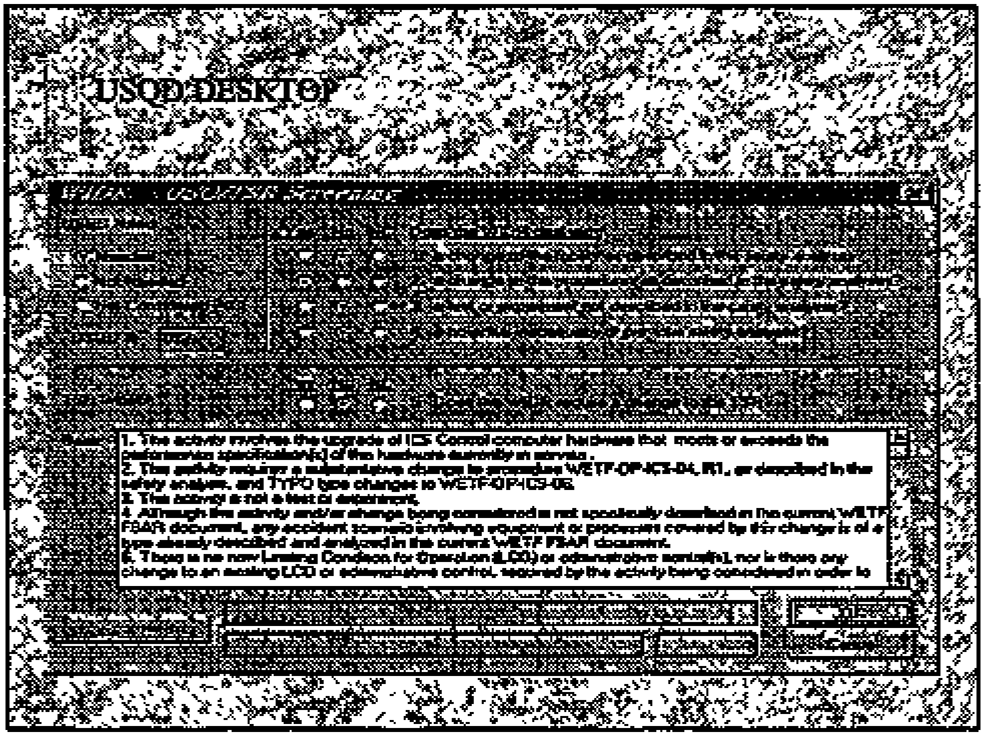
WIDR ACTION SUMMARY



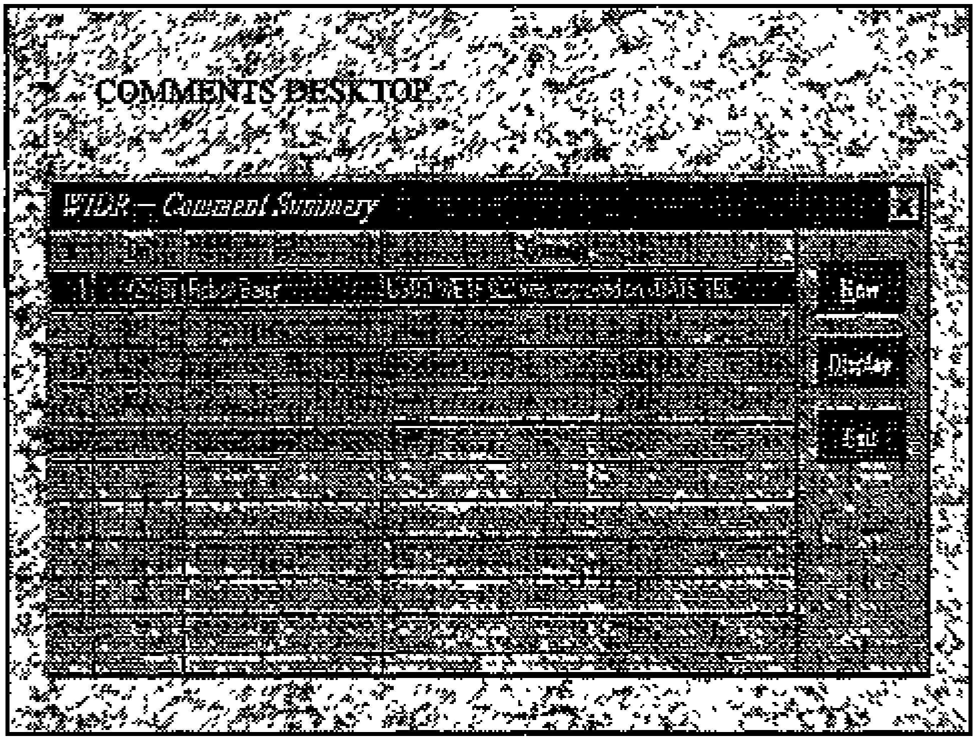
ACTION ITEM DESKTOP



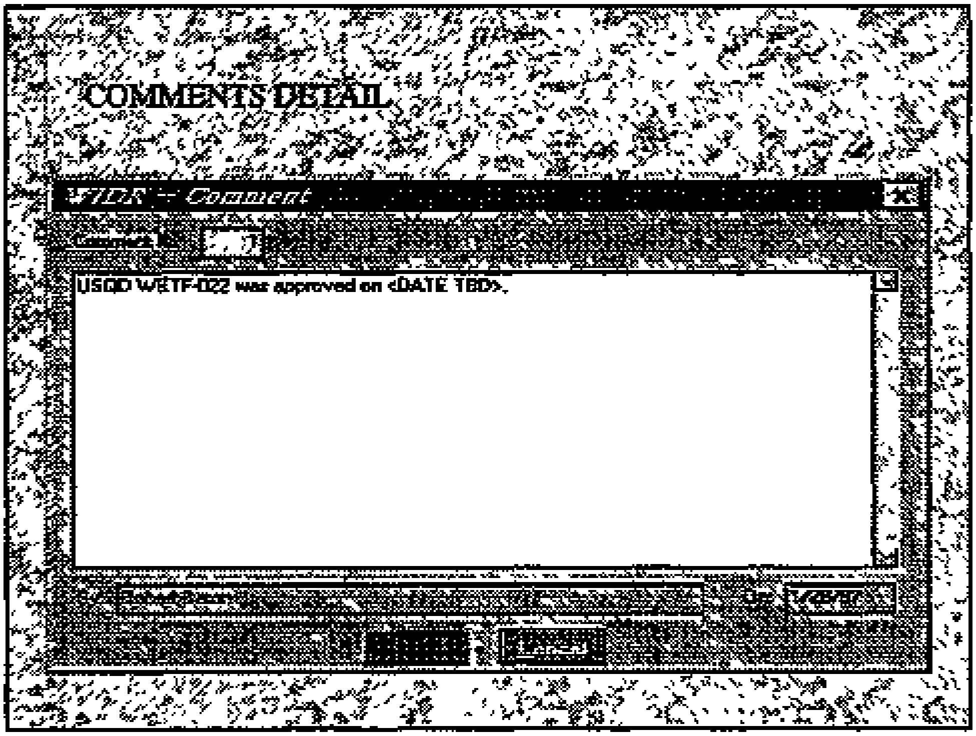
The pre-installation software testing was performed successfully as described in the upgrade document. The MV-9500 has been installed as the process computer and is now undergoing a one month in-use test period. 2/12/97



1. The activity involves the upgrade of ICS Control computer hardware that meets or exceeds the performance specification(s) of the hardware currently in service.
2. This activity requires a substantive change to procedure WETF-OP-ICS-04, R1, as described in the safety analysis, and TYPO type changes to WETF-OP-ICS-05.
3. This activity is not a test or experiment.
4. Although the activity and/or change being considered is not specifically described in the current WETF FSAR document, any accident scenario involving equipment or processes covered by this change is of a type already described and analyzed in the current WETF FSAR document.
5. There is no new Limiting Condition for Operation (LCO) or administrative control(s), nor is there any change to an existing LCO or administrative control, required by the activity being considered in order to



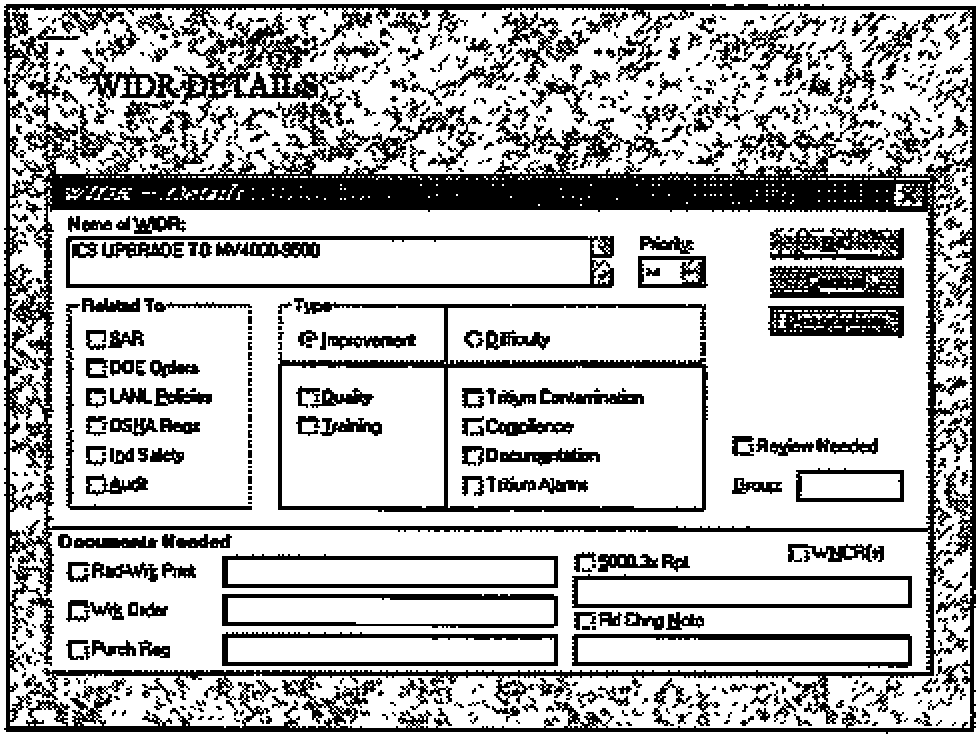
COMMENTS DESKTOP



COMMENTS DETAIL

USQD WETF-022 was approved on <DATE TBD>.

WIDR DETAILS



Name of WIDR: ICS UPGRADE TO MV4000-9500

Priority: [illegible]

Related To:
- □ SAR
- □ DOE Orders
- □ LANL Policies
- □ OSHA Regs
- □ Ind Safety
- □ Audit

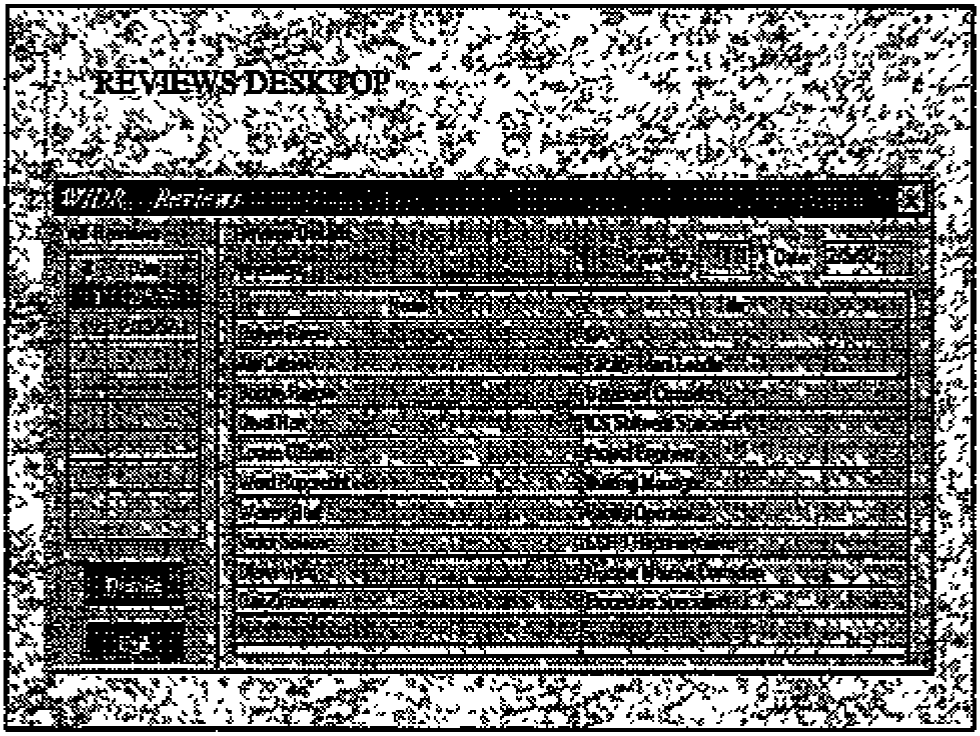
Type:
- ● Improvement
- ○ Difficulty
- □ Quality
- □ Training
- □ Tritium Contamination
- □ Compliance
- □ Documentation
- □ Tritium Alarms

□ Review Needed

Group:

Documents Needed
- □ Rad-Wrk Print
- □ Wrk Order
- □ Purch Req
- □ 5000.3x Rpt
- □ Rd Chng Note
- □ WMCR(s)

REVIEWS DESKTOP

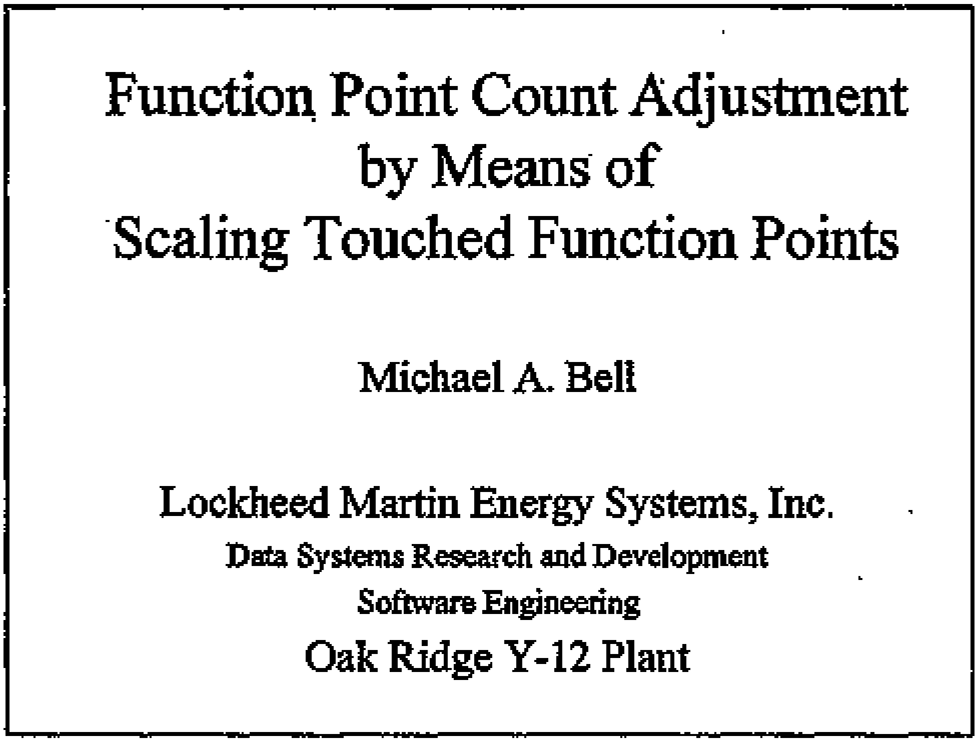


Name	Title
[illegible]	[illegible]
[illegible]	[illegible] Team Leader
[illegible]	[illegible]
Orval Hart	ICS Software Specialist
[illegible]	Project Engineer
[illegible]	Building Manager
[illegible]	WETF Operations
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	Procedure Specialist

Session A2: Software Engineering Processes

Chair Kathleen Canal
Headquarters Department of Energy

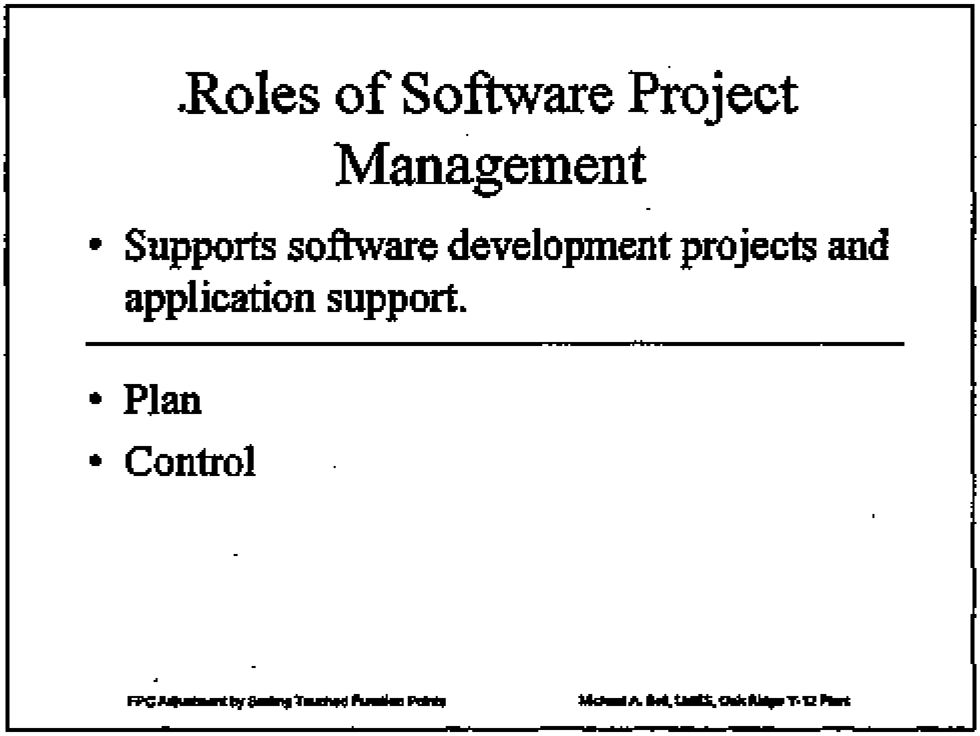
Session : Paper #	Author(s)	*Title*
A2:1	Michael Bell Lockheed Martin Energy Systems	*Function Point Count Adjustment by Means of Scaling Touched Function Points*
A2:2	Stewart Meyer Westinghouse Savannah River Co.	*Using An Automated Code Management System To Improve Configuration Control Practices*
A2:3	Karen Jefferson, Terry Porter & Todd West, Sandia National Laboratories	*Software Engineering and Graphical Programming Languages*



Function Point Count Adjustment by Means of Scaling Touched Function Points

Michael A. Bell

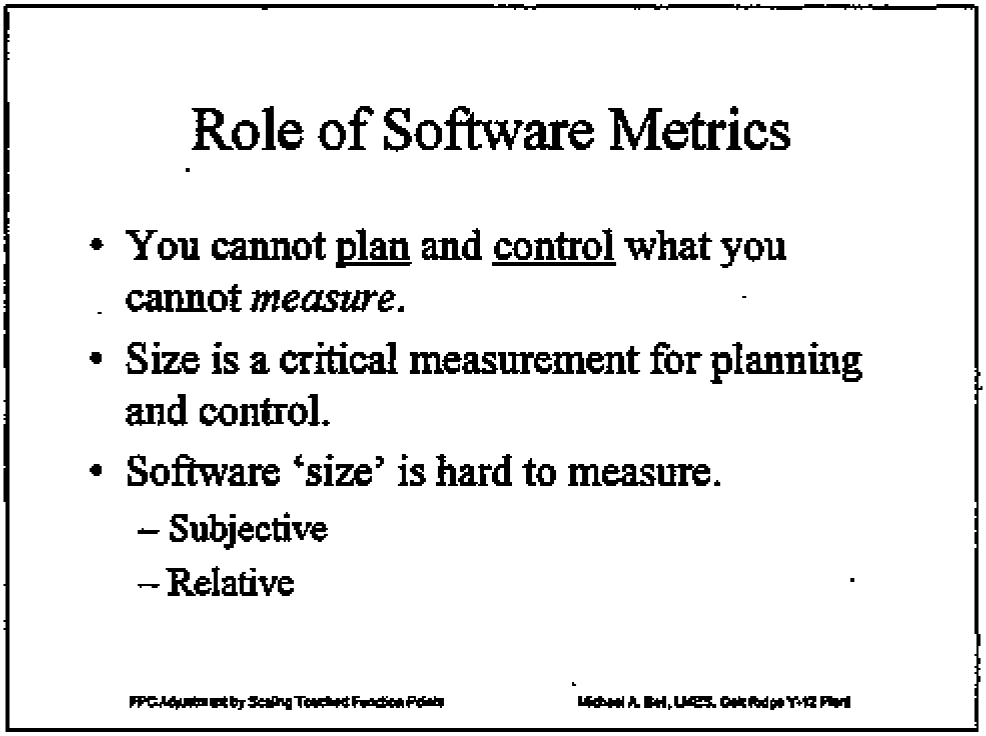
Lockheed Martin Energy Systems, Inc.
Data Systems Research and Development
Software Engineering
Oak Ridge Y-12 Plant



.Roles of Software Project Management

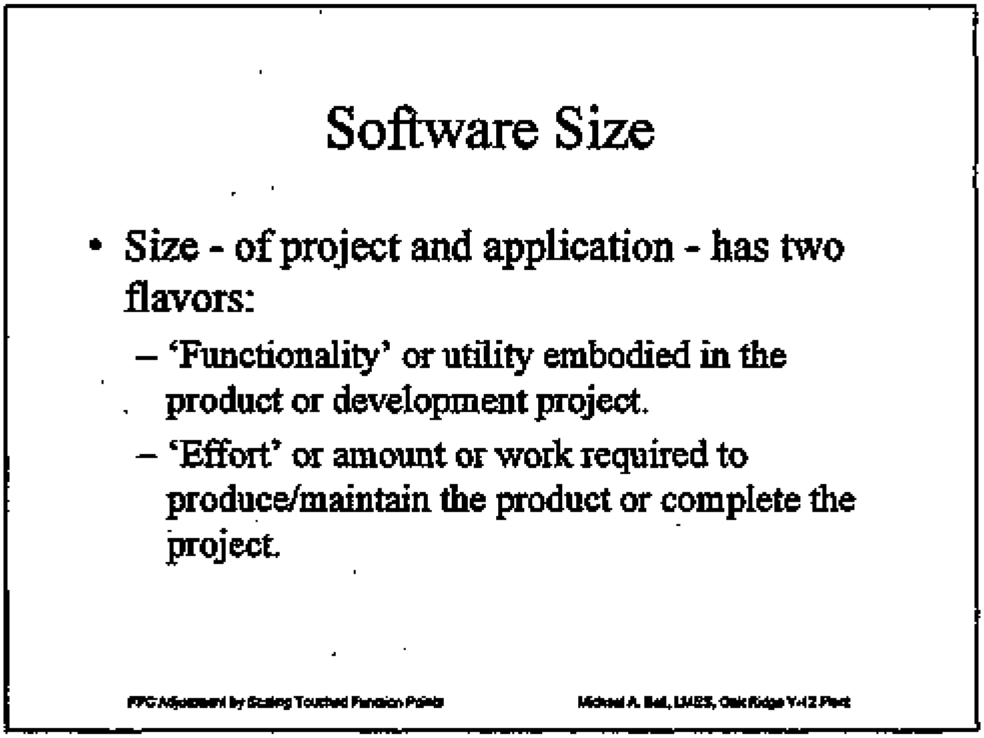
- Supports software development projects and application support.

- Plan
- Control



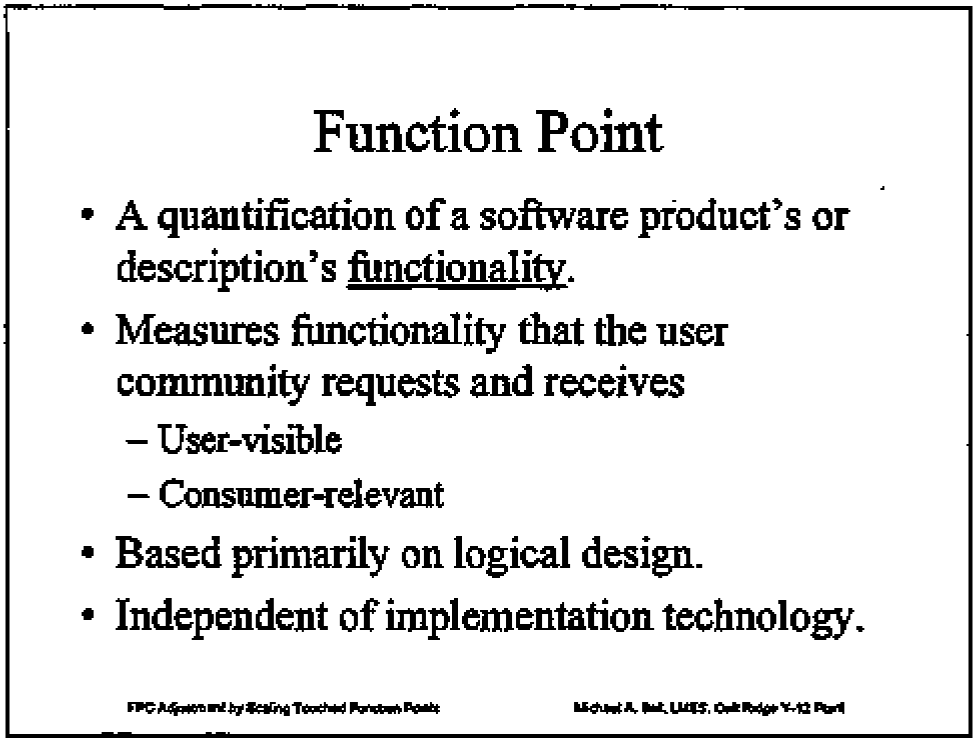
Role of Software Metrics

- You cannot <u>plan</u> and <u>control</u> what you cannot *measure*.
- Size is a critical measurement for planning and control.
- Software 'size' is hard to measure.
  - Subjective
  - Relative



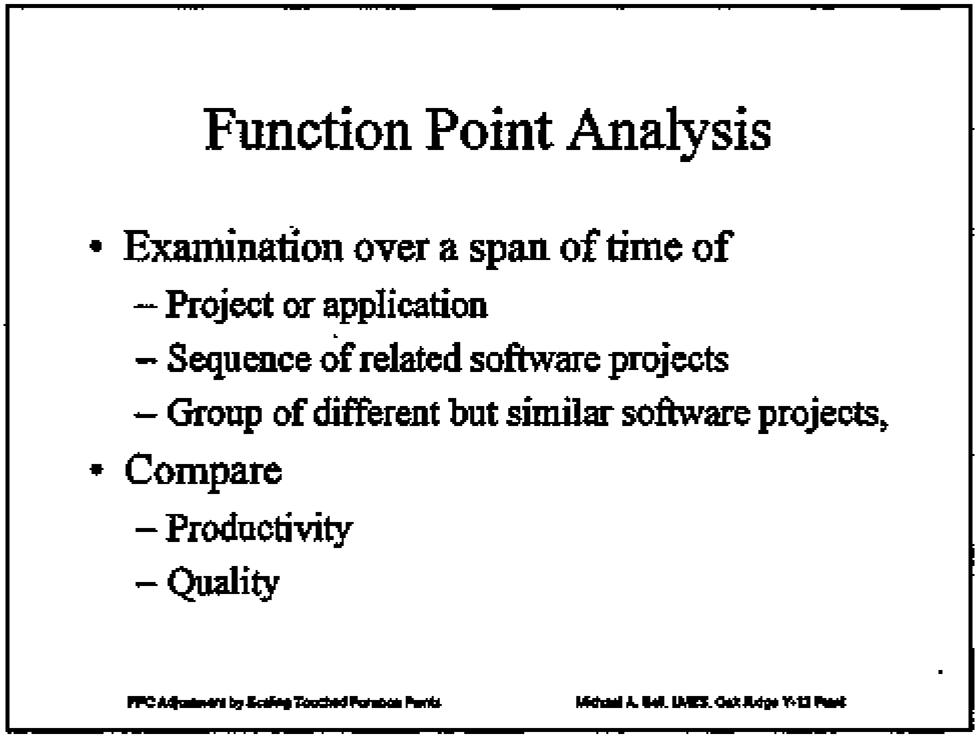
Software Size

- Size - of project and application - has two flavors:
  - 'Functionality' or utility embodied in the product or development project.
  - 'Effort' or amount or work required to produce/maintain the product or complete the project.



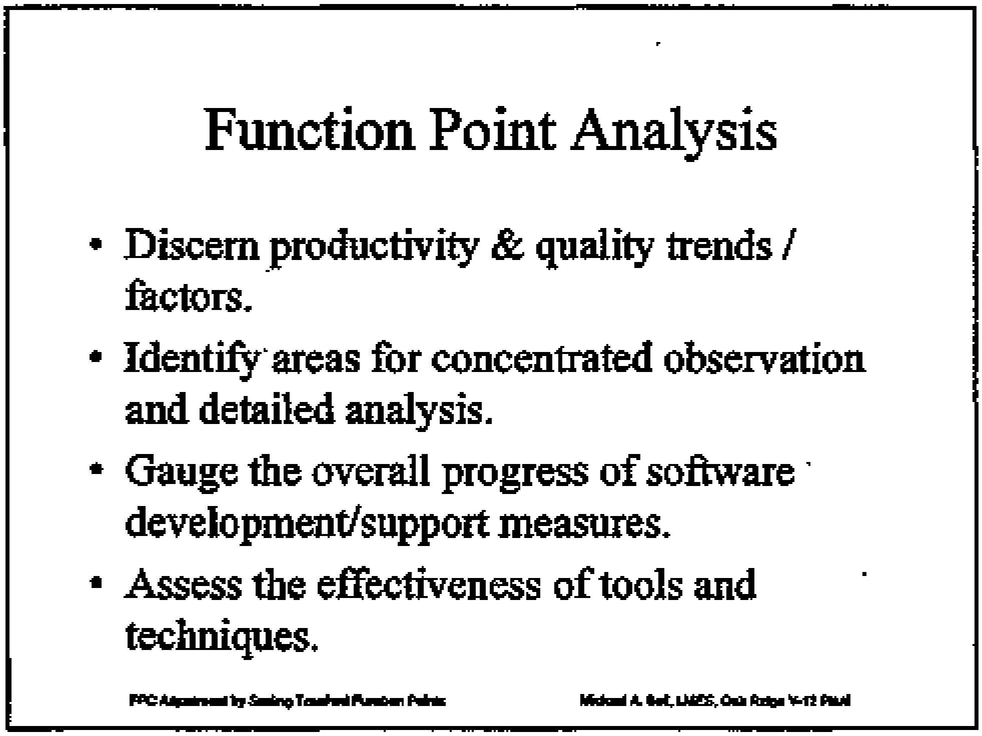
Function Point

- A quantification of a software product's or description's functionality.
- Measures functionality that the user community requests and receives
  - User-visible
  - Consumer-relevant
- Based primarily on logical design.
- Independent of implementation technology.



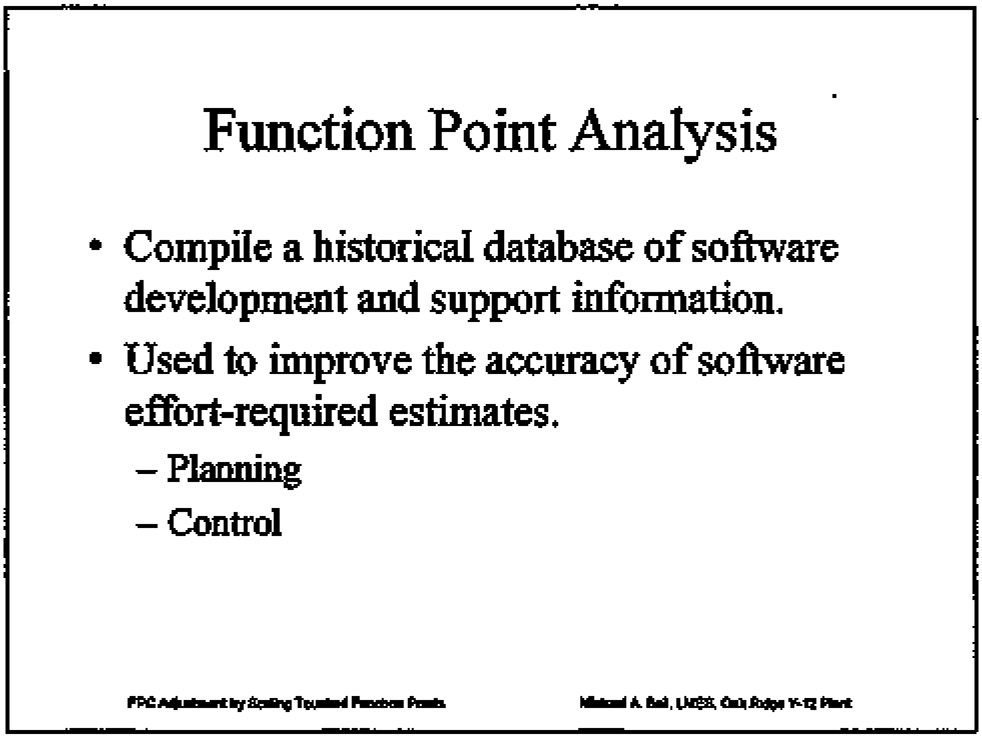
Function Point Analysis

- Examination over a span of time of
  - Project or application
  - Sequence of related software projects
  - Group of different but similar software projects,
- Compare
  - Productivity
  - Quality



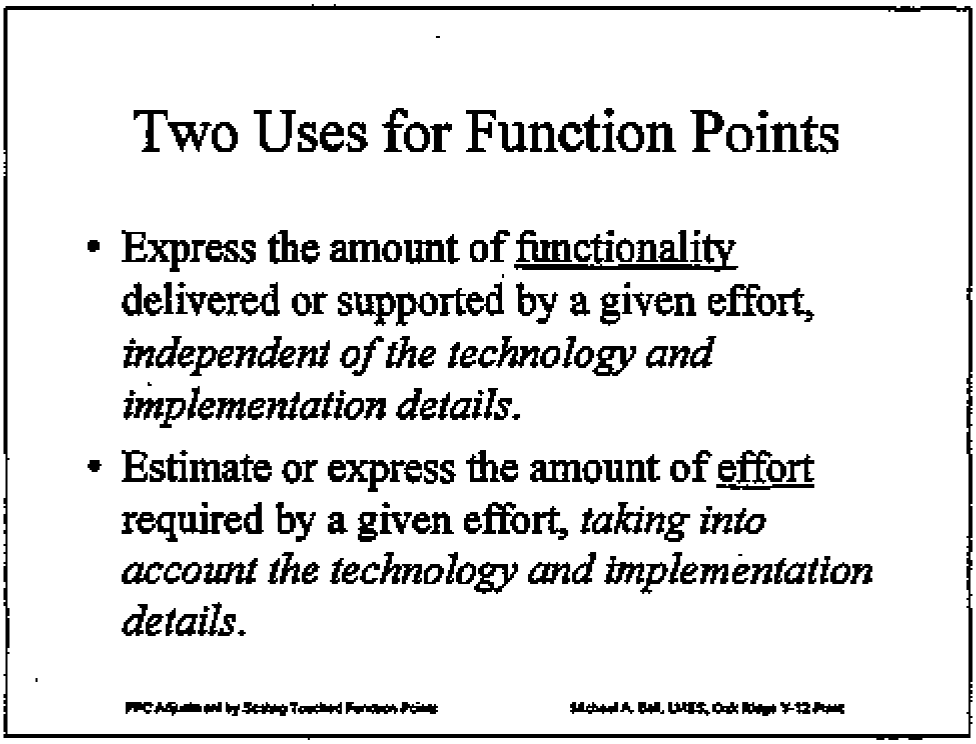
Function Point Analysis

- Discern productivity & quality trends / factors.
- Identify areas for concentrated observation and detailed analysis.
- Gauge the overall progress of software development/support measures.
- Assess the effectiveness of tools and techniques.



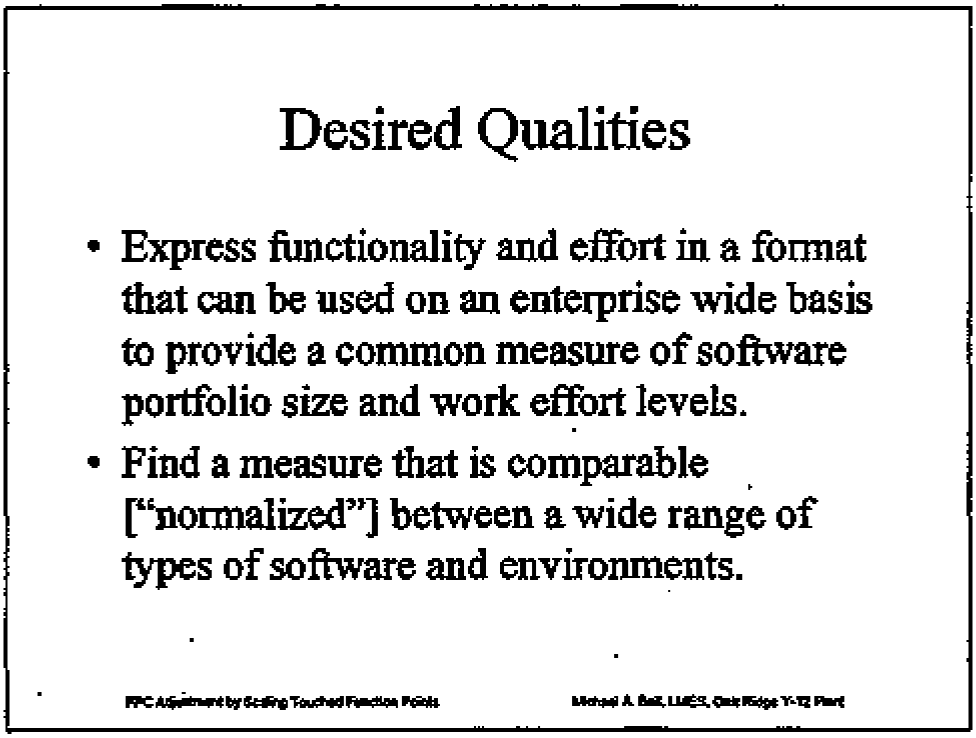
Function Point Analysis

- Compile a historical database of software development and support information.
- Used to improve the accuracy of software effort-required estimates.
  - Planning
  - Control



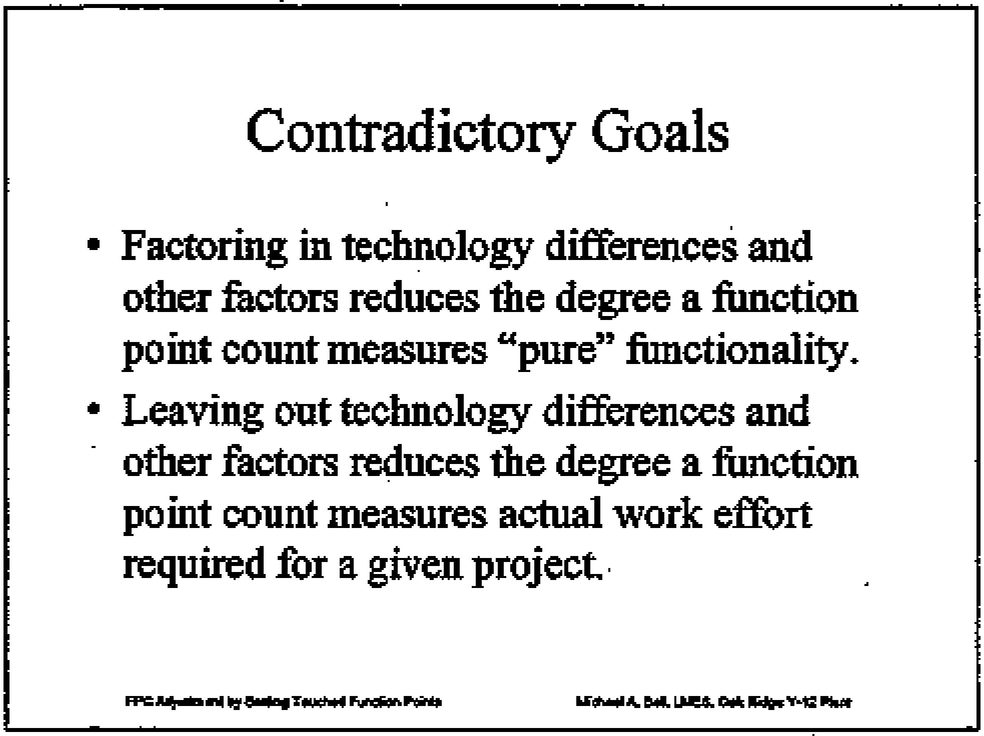
Two Uses for Function Points

- Express the amount of <u>functionality</u> delivered or supported by a given effort, *independent of the technology and implementation details.*
- Estimate or express the amount of <u>effort</u> required by a given effort, *taking into account the technology and implementation details.*



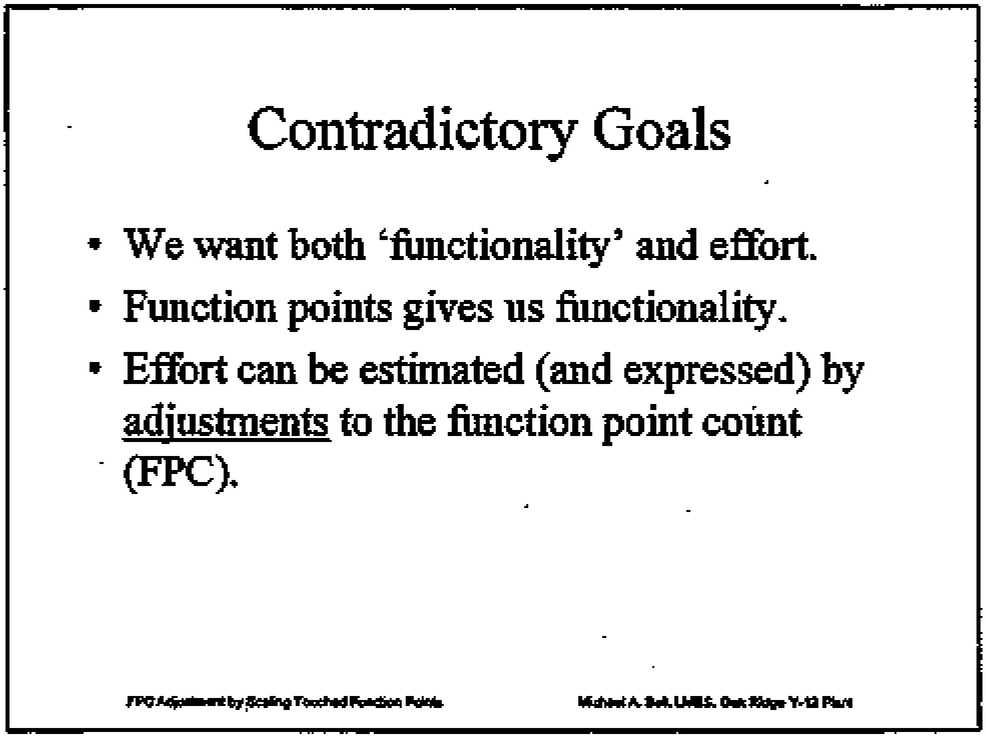
Desired Qualities

- Express functionality and effort in a format that can be used on an enterprise wide basis to provide a common measure of software portfolio size and work effort levels.
- Find a measure that is comparable ["normalized"] between a wide range of types of software and environments.



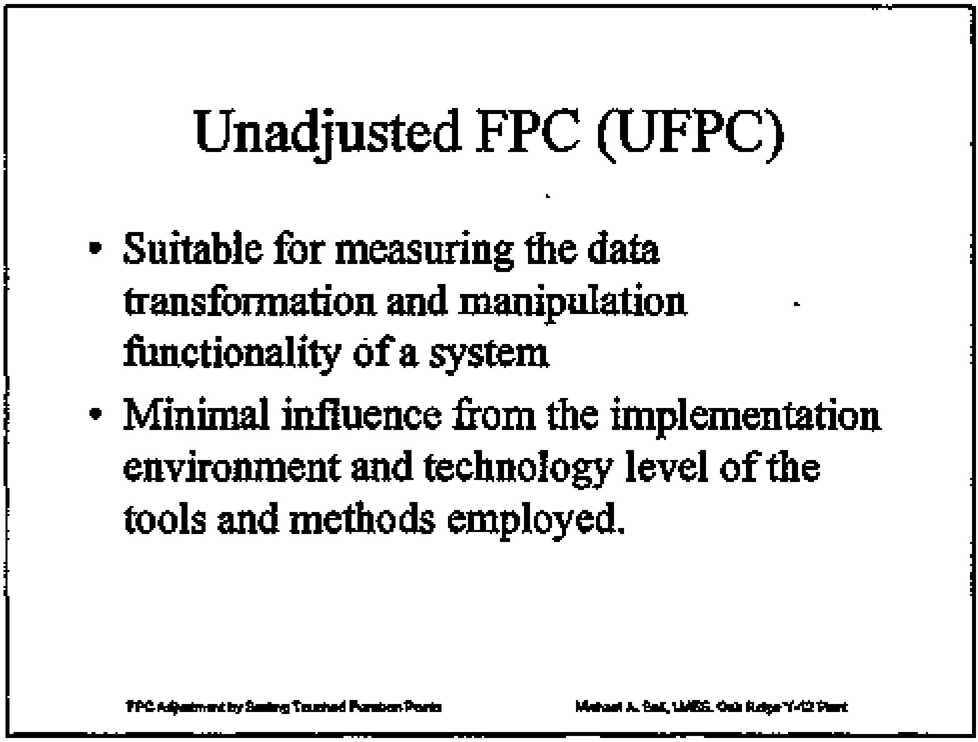
Contradictory Goals

- Factoring in technology differences and other factors reduces the degree a function point count measures “pure” functionality.
- Leaving out technology differences and other factors reduces the degree a function point count measures actual work effort required for a given project.



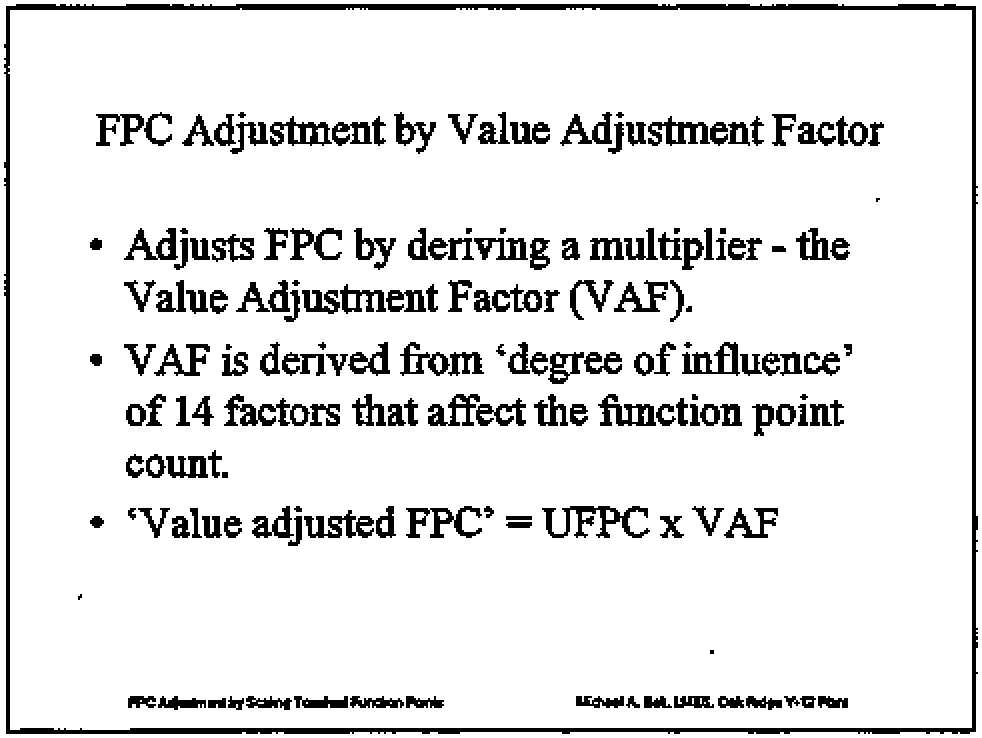
Contradictory Goals

- We want both ‘functionality’ and effort.
- Function points gives us functionality.
- Effort can be estimated (and expressed) by <u>adjustments</u> to the function point count (FPC).



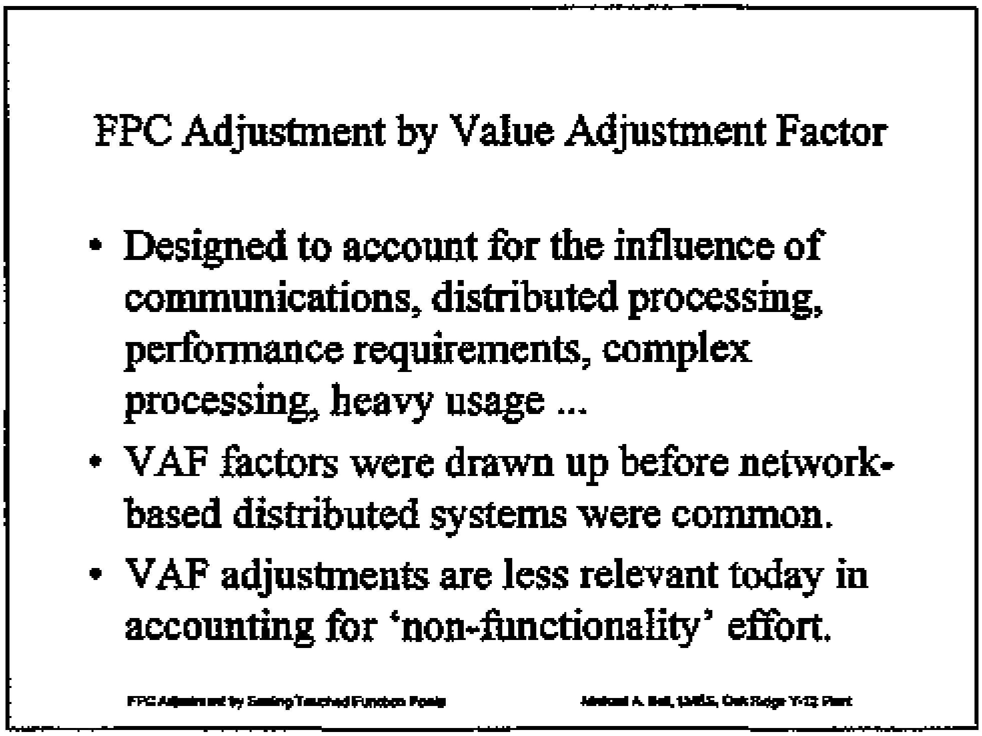
Unadjusted FPC (UFPC)

- Suitable for measuring the data transformation and manipulation functionality of a system
- Minimal influence from the implementation environment and technology level of the tools and methods employed.



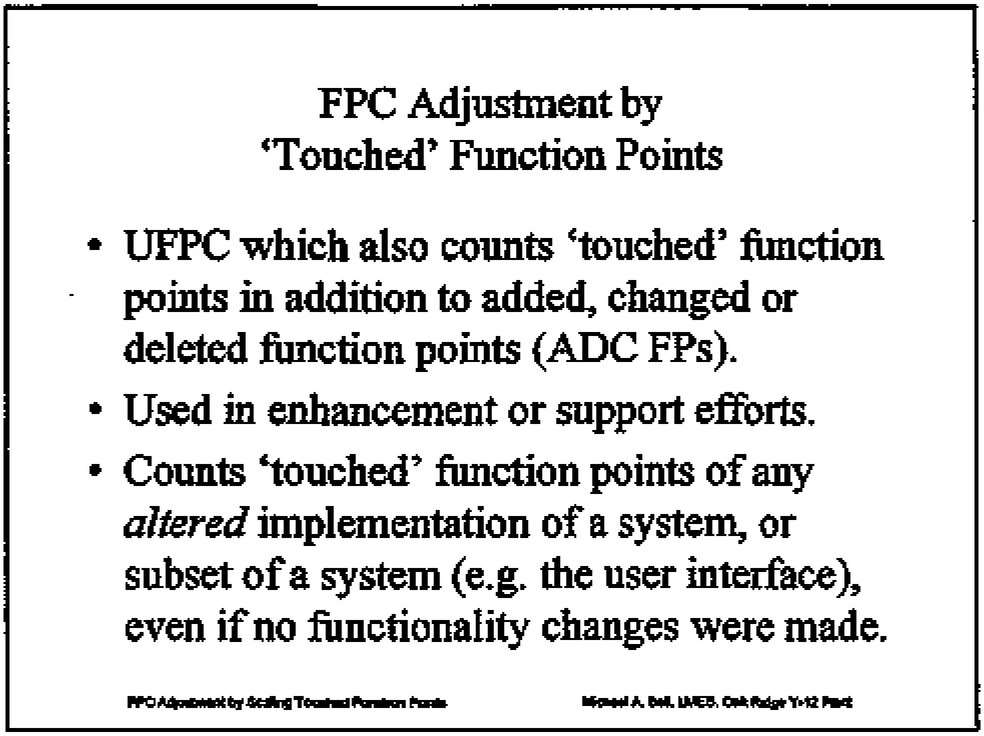
FPC Adjustment by Value Adjustment Factor

- Adjusts FPC by deriving a multiplier - the Value Adjustment Factor (VAF).
- VAF is derived from 'degree of influence' of 14 factors that affect the function point count.
- 'Value adjusted FPC' = UFPC x VAF



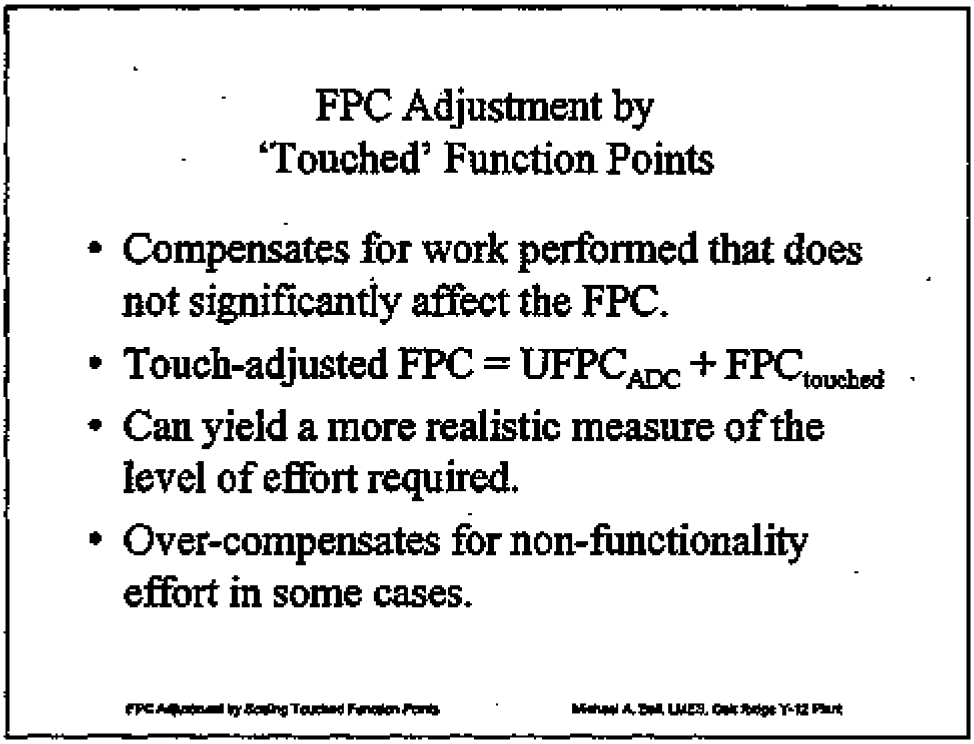
FPC Adjustment by Value Adjustment Factor

- Designed to account for the influence of communications, distributed processing, performance requirements, complex processing, heavy usage ...
- VAF factors were drawn up before network-based distributed systems were common.
- VAF adjustments are less relevant today in accounting for 'non-functionality' effort.



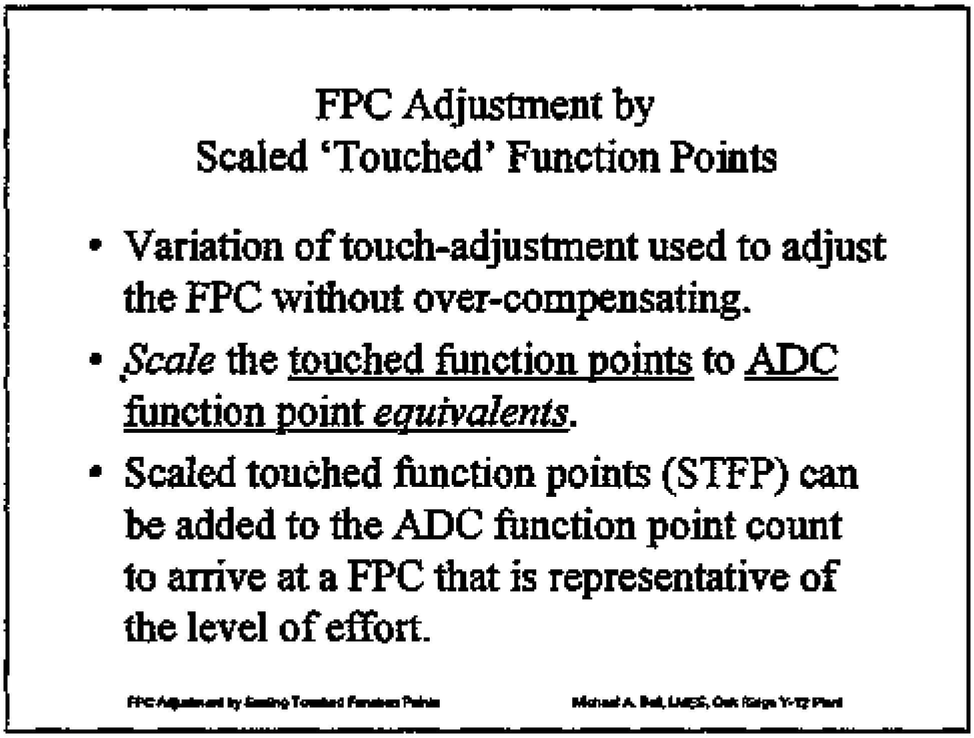
FPC Adjustment by 'Touched' Function Points

- UFPC which also counts 'touched' function points in addition to added, changed or deleted function points (ADC FPs).
- Used in enhancement or support efforts.
- Counts 'touched' function points of any *altered* implementation of a system, or subset of a system (e.g. the user interface), even if no functionality changes were made.



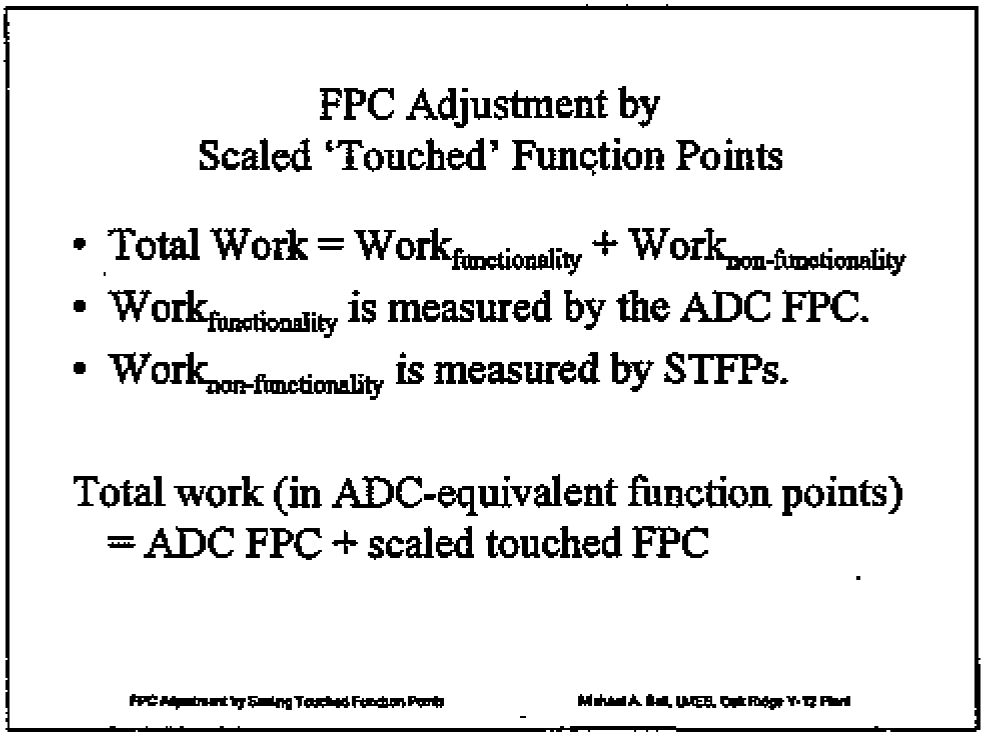
FPC Adjustment by 'Touched' Function Points

- Compensates for work performed that does not significantly affect the FPC.
- Touch-adjusted FPC = $UFPC_{ADC} + FPC_{touched}$
- Can yield a more realistic measure of the level of effort required.
- Over-compensates for non-functionality effort in some cases.



FPC Adjustment by Scaled 'Touched' Function Points

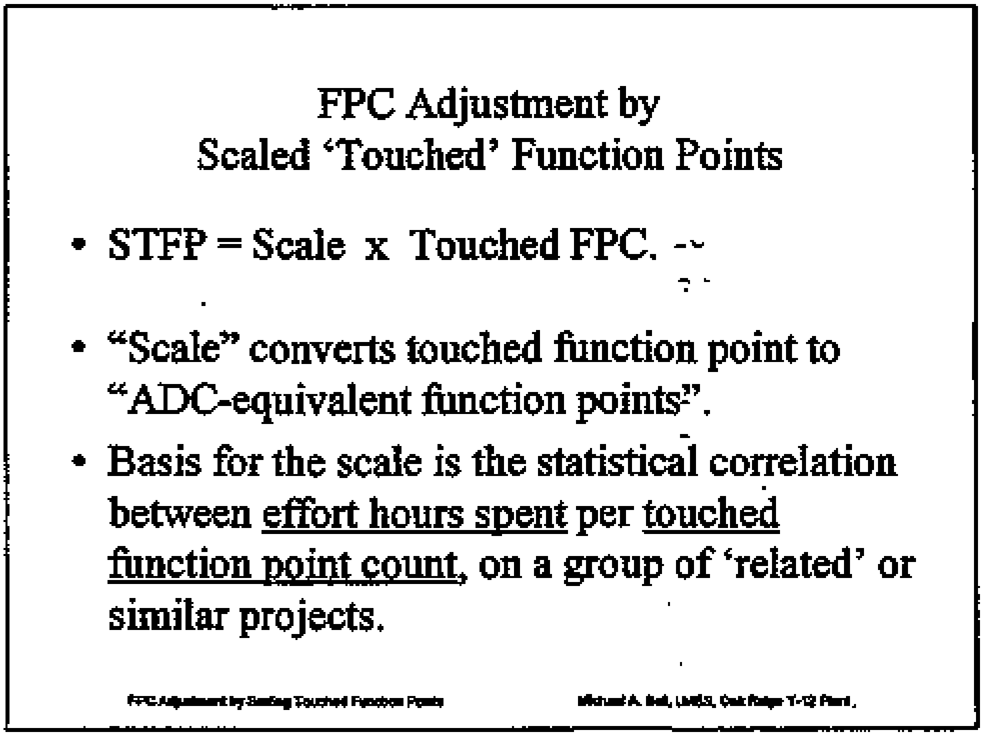
- Variation of touch-adjustment used to adjust the FPC without over-compensating.
- *Scale* the touched function points to ADC function point *equivalents*.
- Scaled touched function points (STFP) can be added to the ADC function point count to arrive at a FPC that is representative of the level of effort.



FPC Adjustment by Scaled 'Touched' Function Points

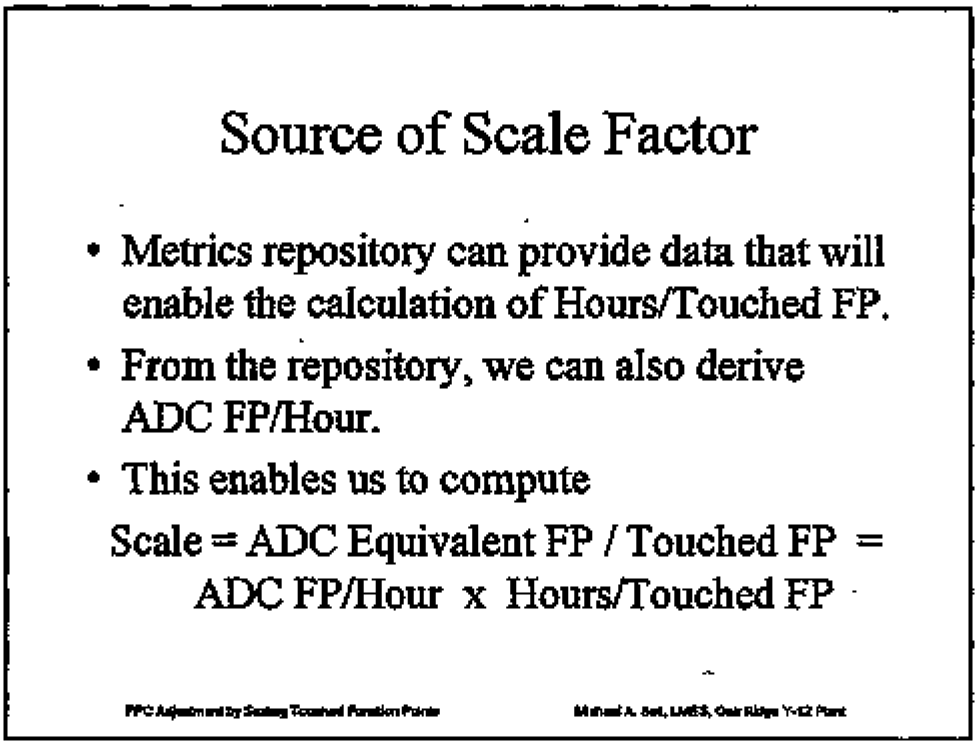
- Total Work = Work$_{\text{functionality}}$ + Work$_{\text{non-functionality}}$
- Work$_{\text{functionality}}$ is measured by the ADC FPC.
- Work$_{\text{non-functionality}}$ is measured by STFPs.

Total work (in ADC-equivalent function points)
= ADC FPC + scaled touched FPC



FPC Adjustment by Scaled 'Touched' Function Points

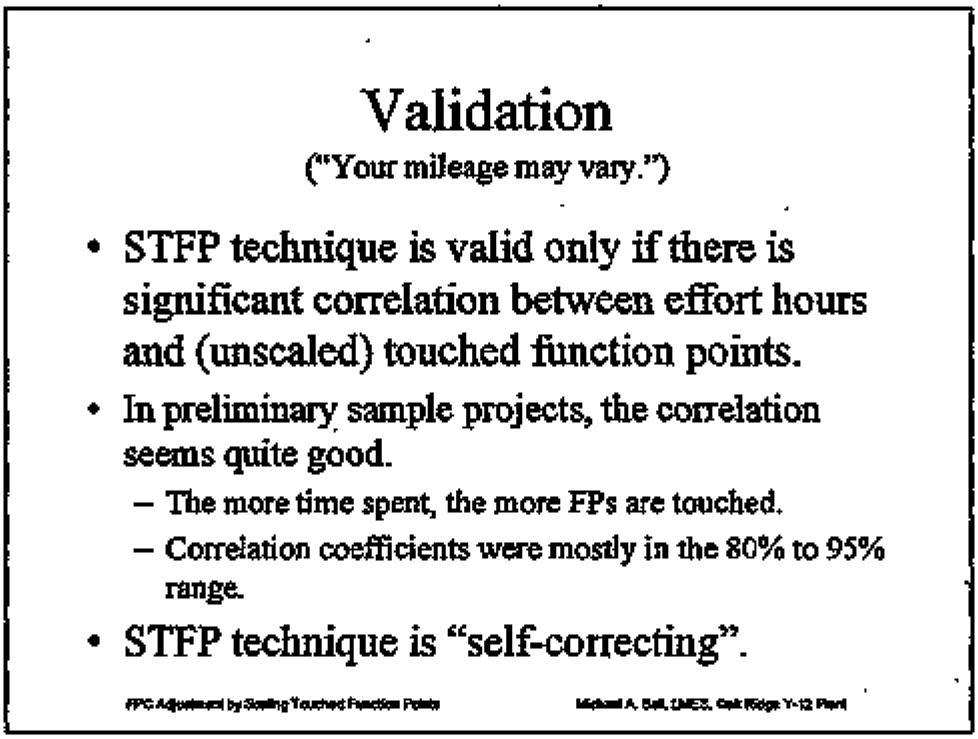
- STFP = Scale x Touched FPC.
- "Scale" converts touched function point to "ADC-equivalent function points".
- Basis for the scale is the statistical correlation between <u>effort hours spent</u> per <u>touched function point count</u>, on a group of 'related' or similar projects.



Source of Scale Factor

- Metrics repository can provide data that will enable the calculation of Hours/Touched FP.
- From the repository, we can also derive ADC FP/Hour.
- This enables us to compute

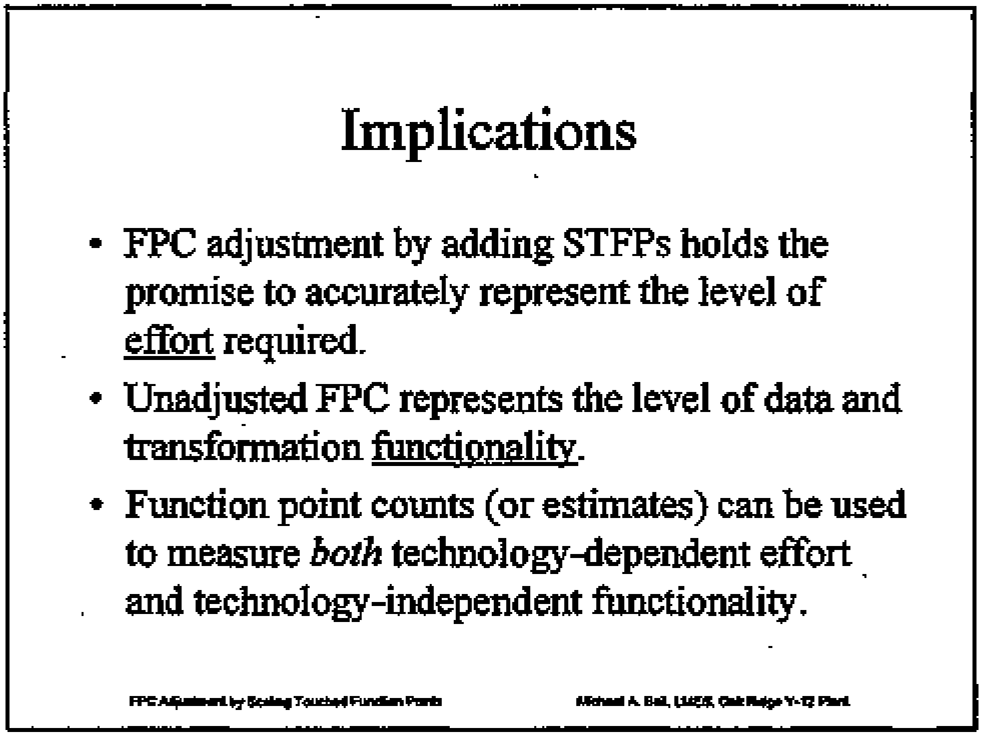
Scale = ADC Equivalent FP / Touched FP =
ADC FP/Hour x Hours/Touched FP



Validation

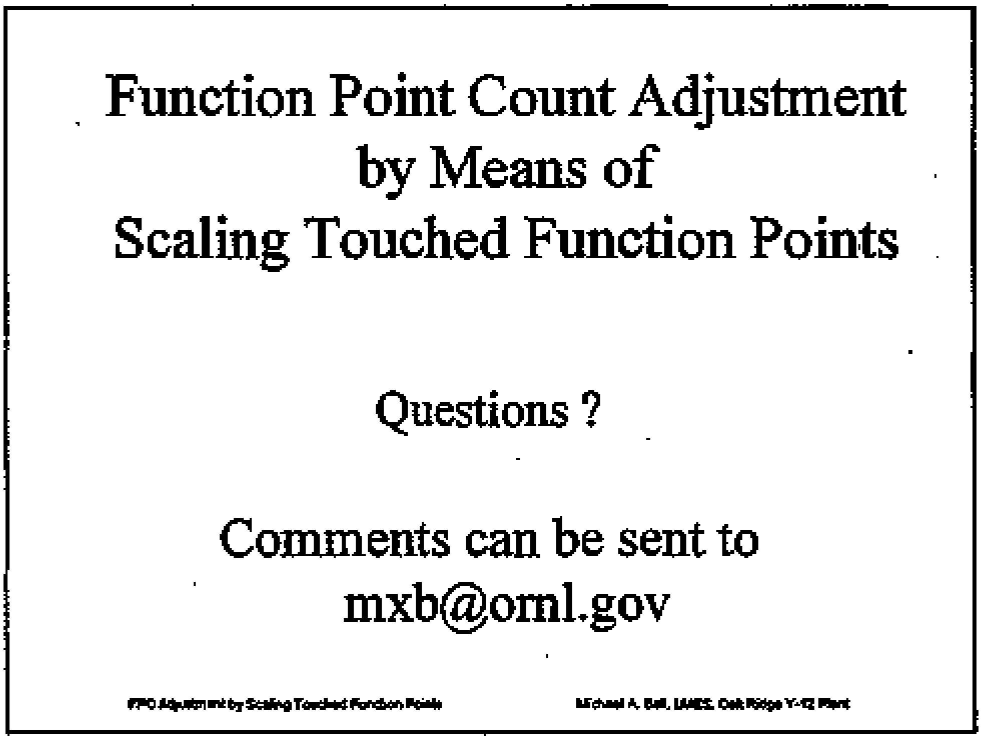
("Your mileage may vary.")

- STFP technique is valid only if there is significant correlation between effort hours and (unscaled) touched function points.
- In preliminary sample projects, the correlation seems quite good.
  - The more time spent, the more FPs are touched.
  - Correlation coefficients were mostly in the 80% to 95% range.
- STFP technique is "self-correcting".



Implications

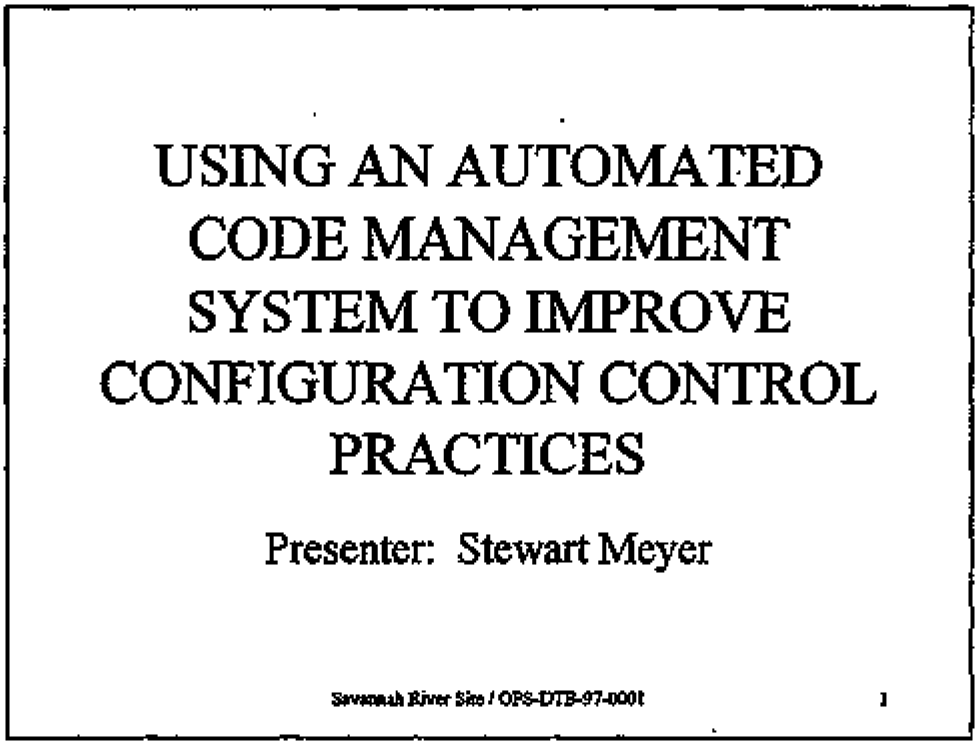
- FPC adjustment by adding STFPs holds the promise to accurately represent the level of effort required.
- Unadjusted FPC represents the level of data and transformation functionality.
- Function point counts (or estimates) can be used to measure *both* technology-dependent effort and technology-independent functionality.



Function Point Count Adjustment by Means of Scaling Touched Function Points

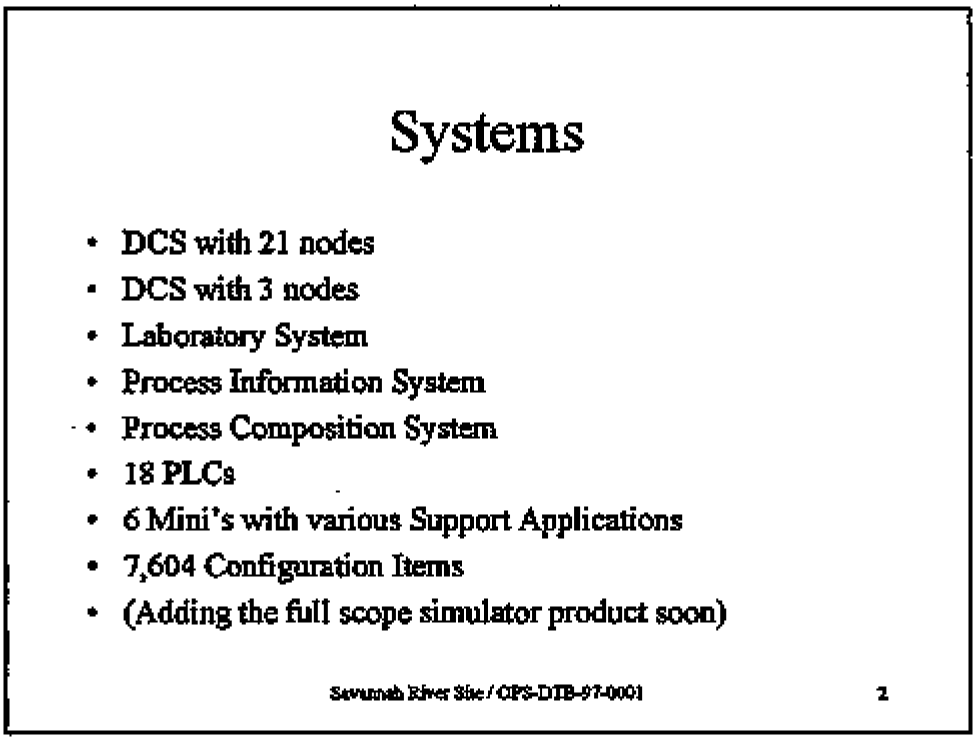
Questions ?

Comments can be sent to
mxb@ornl.gov



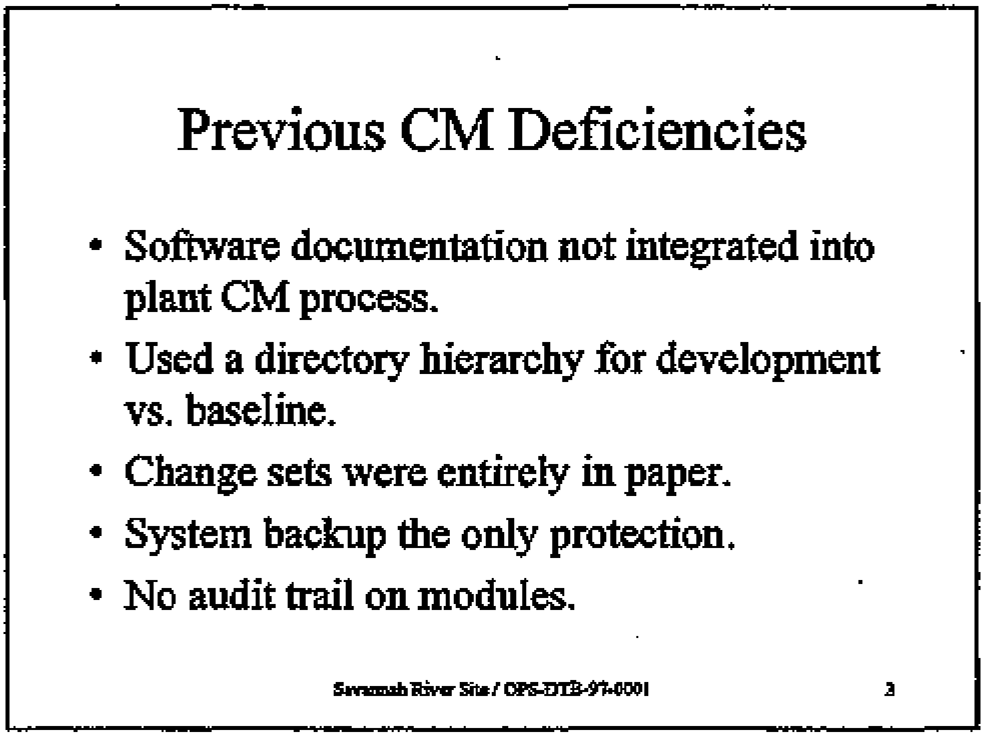
USING AN AUTOMATED CODE MANAGEMENT SYSTEM TO IMPROVE CONFIGURATION CONTROL PRACTICES

Presenter: Stewart Meyer



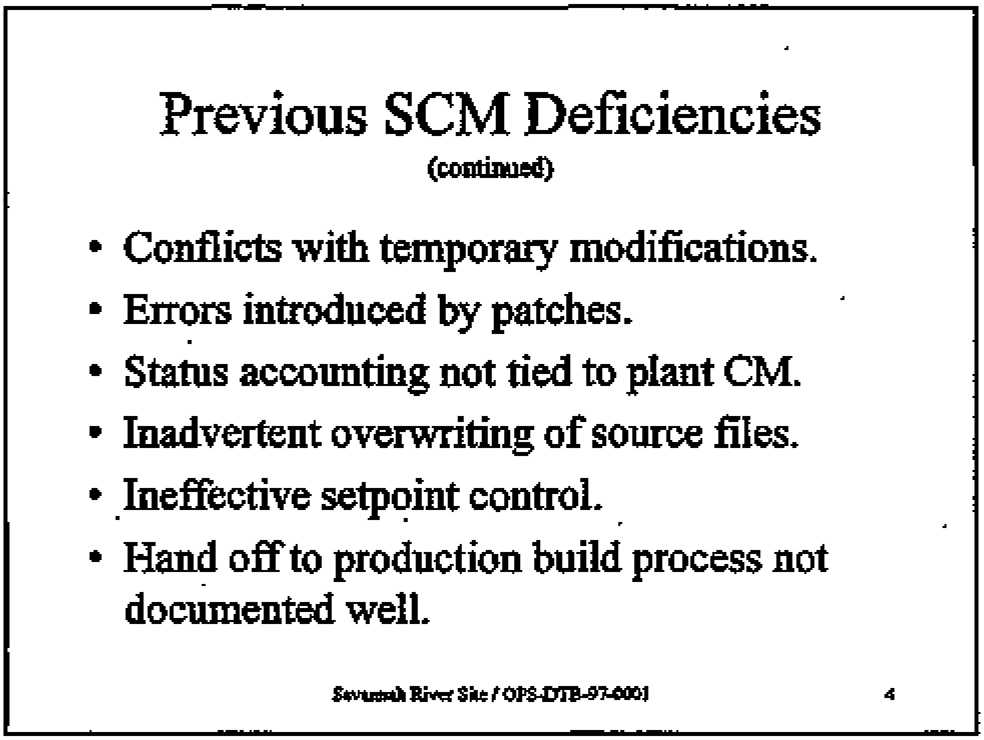
Systems

- DCS with 21 nodes
- DCS with 3 nodes
- Laboratory System
- Process Information System
- Process Composition System
- 18 PLCs
- 6 Mini's with various Support Applications
- 7,604 Configuration Items
- (Adding the full scope simulator product soon)



Previous CM Deficiencies

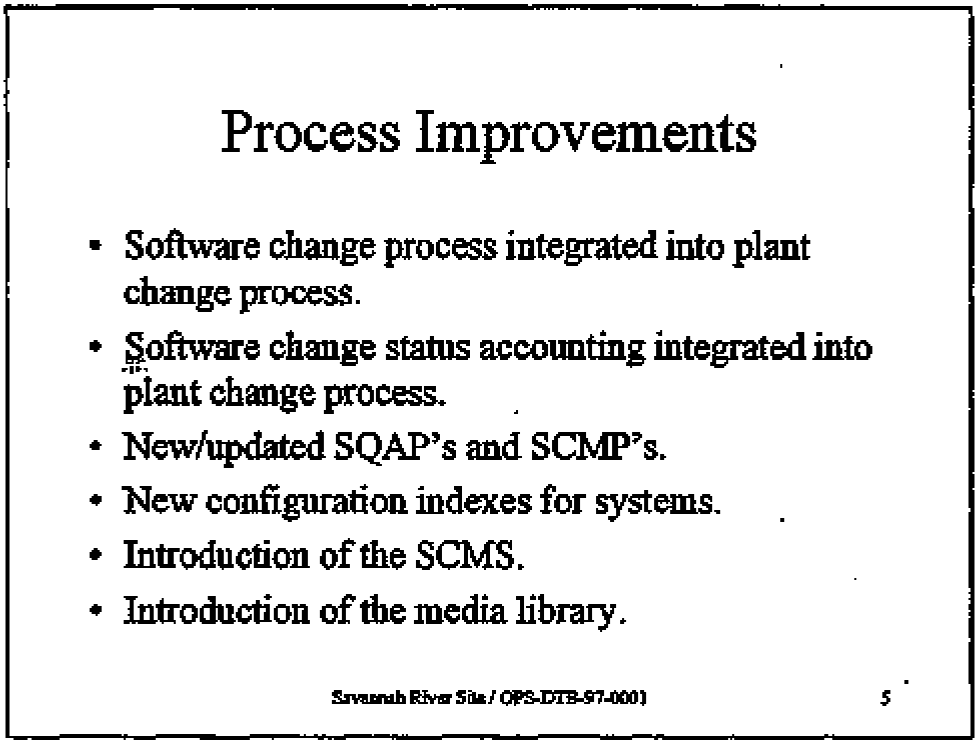
- Software documentation not integrated into plant CM process.
- Used a directory hierarchy for development vs. baseline.
- Change sets were entirely in paper.
- System backup the only protection.
- No audit trail on modules.



Previous SCM Deficiencies

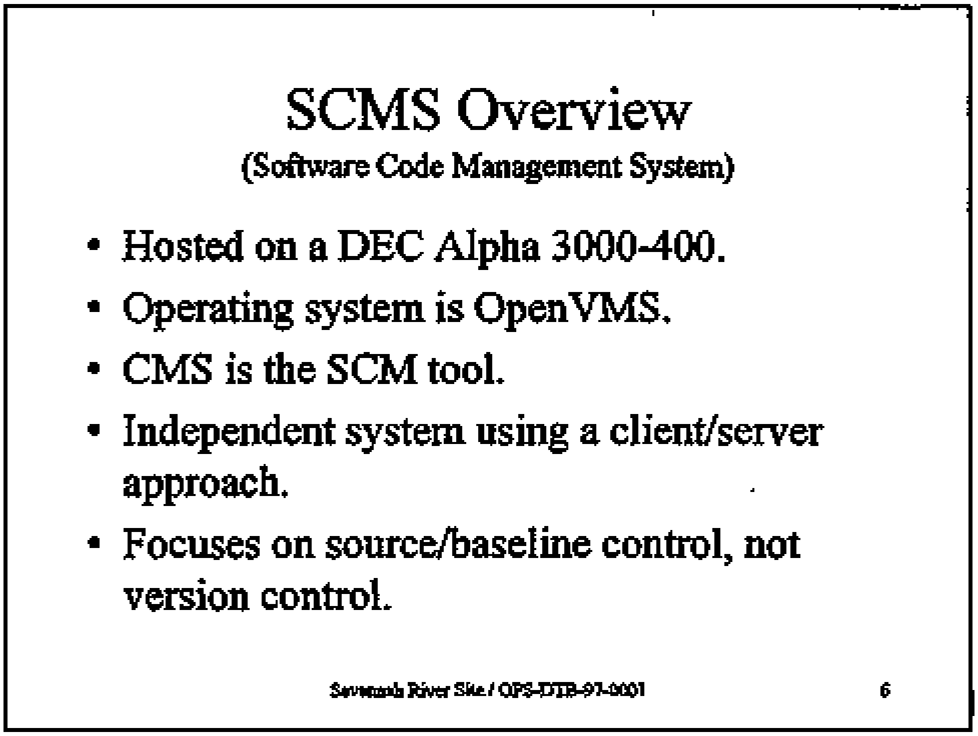
(continued)

- Conflicts with temporary modifications.
- Errors introduced by patches.
- Status accounting not tied to plant CM.
- Inadvertent overwriting of source files.
- Ineffective setpoint control.
- Hand off to production build process not documented well.



Process Improvements

- Software change process integrated into plant change process.
- Software change status accounting integrated into plant change process.
- New/updated SQAP's and SCMP's.
- New configuration indexes for systems.
- Introduction of the SCMS.
- Introduction of the media library.



SCMS Overview

(Software Code Management System)

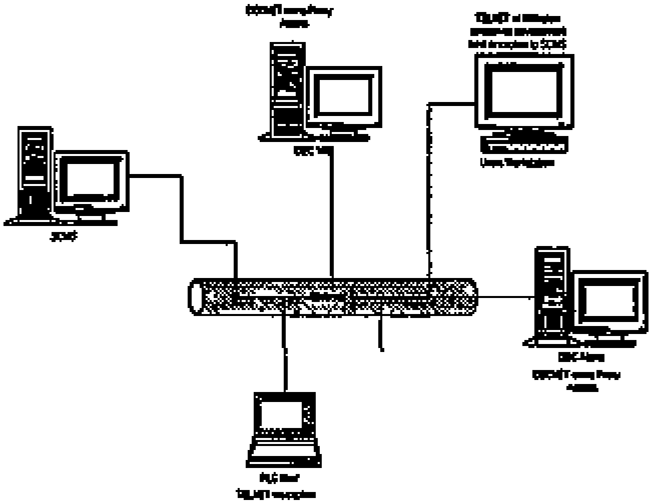
- Hosted on a DEC Alpha 3000-400.
- Operating system is OpenVMS.
- CMS is the SCM tool.
- Independent system using a client/server approach.
- Focuses on source/baseline control, not version control.

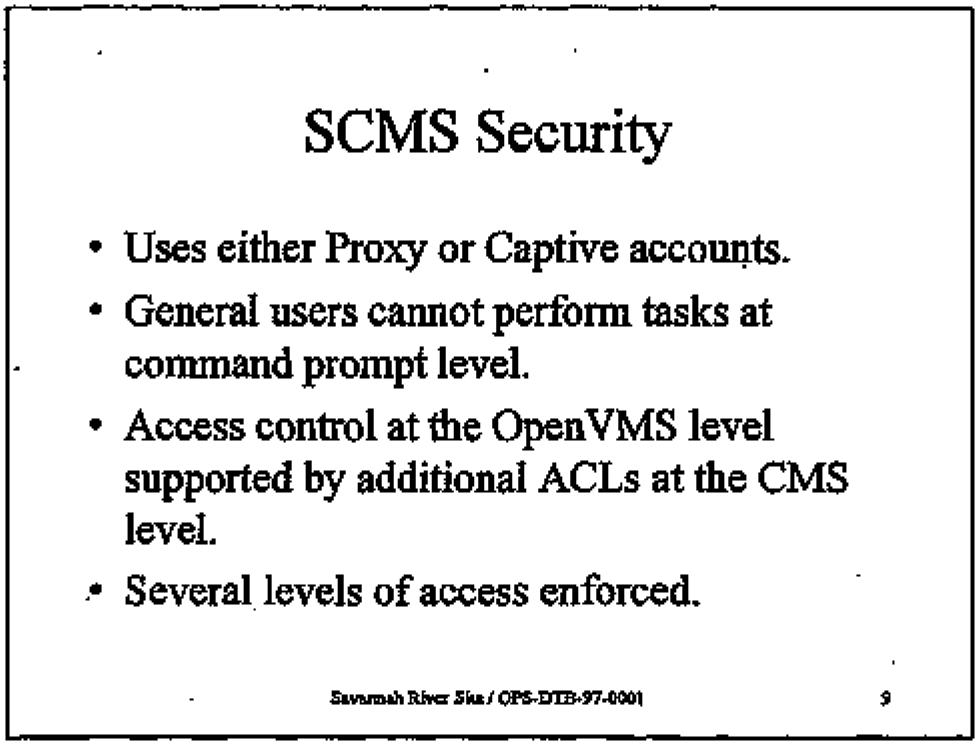
SCMS Overview

(Continued)

- Interface is a simple, in-house developed, text based menu system.
- Many multi-step functions are automated.
- Enforces policies outlined in SCMP's via pseudo functions.
- Employs very tight security and access restrictions.

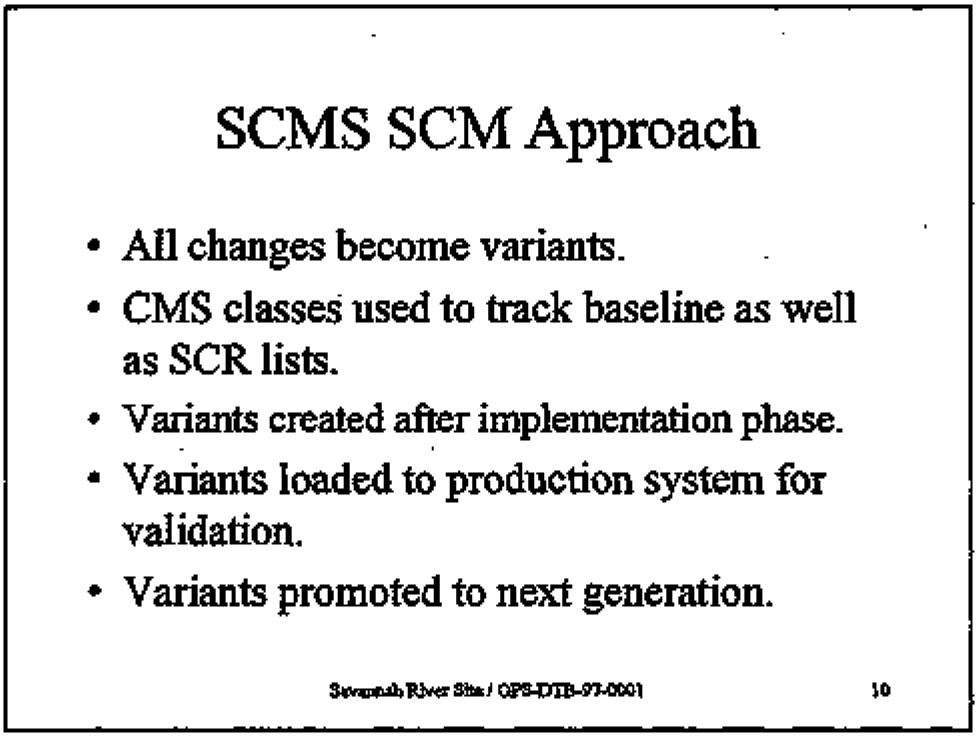
SCMS ACCESS





SCMS Security

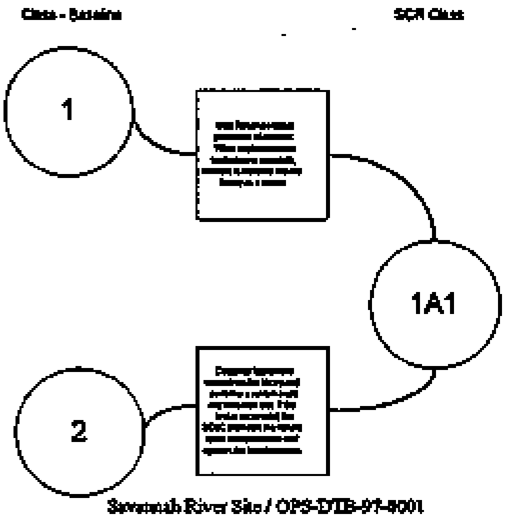
- Uses either Proxy or Captive accounts.
- General users cannot perform tasks at command prompt level.
- Access control at the OpenVMS level supported by additional ACLs at the CMS level.
- Several levels of access enforced.



SCMS SCM Approach

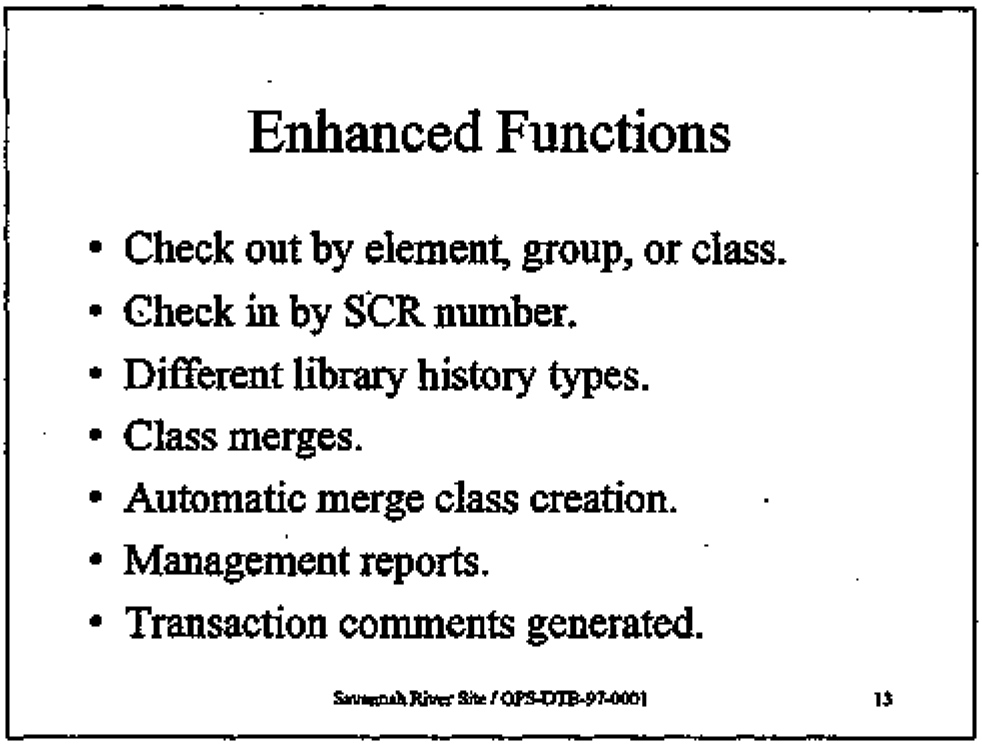
- All changes become variants.
- CMS classes used to track baseline as well as SCR lists.
- Variants created after implementation phase.
- Variants loaded to production system for validation.
- Variants promoted to next generation.

SCMS SCM Model



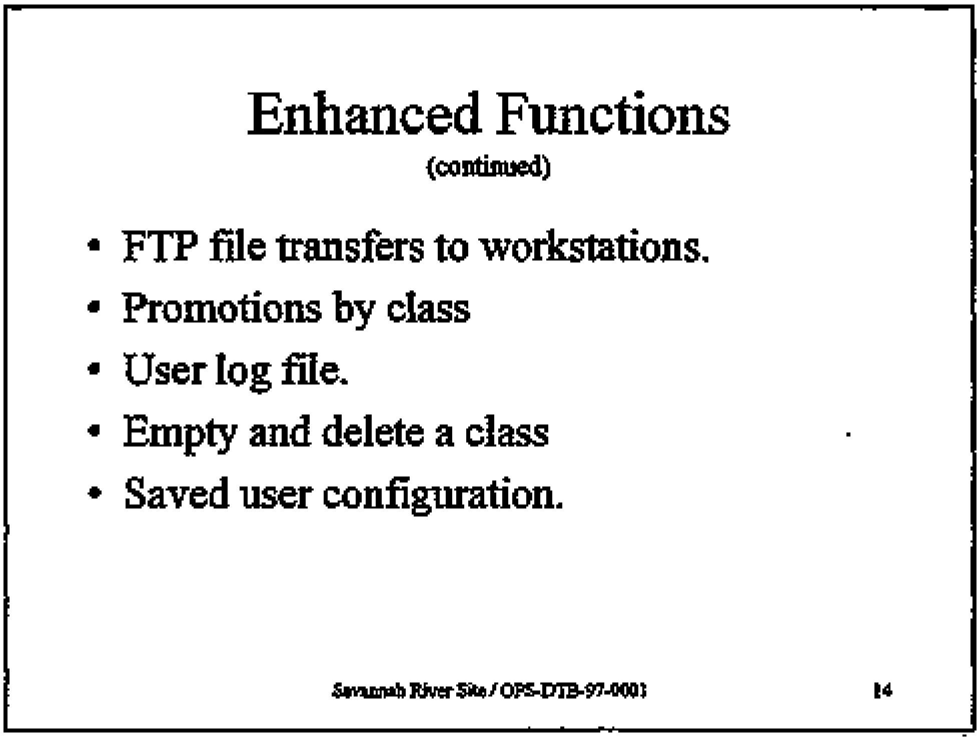
Standard Functions

- Many common commands are provided on the menu, such as show elements or generations.
- Concept is to allow full use of the tool without having to learn the command language or complex command syntax.
- Generally, there is no "programming" involved here, just converting user input to a command to the CMS. (Advanced users may use qualifiers.)



Enhanced Functions

- Check out by element, group, or class.
- Check in by SCR number.
- Different library history types.
- Class merges.
- Automatic merge class creation.
- Management reports.
- Transaction comments generated.



Enhanced Functions

(continued)

- FTP file transfers to workstations.
- Promotions by class
- User log file.
- Empty and delete a class
- Saved user configuration.

Enforced Policy Functions

- Upon check in, an SCR class is created, variants produced, and inserted into this new class. Variant letter codes are created automatically.
- For non-concurrent libraries, reservations are denied while any variants exist. Modules released after promotions.

Key Improvements

- Each product is stored in a separate library. At any time, the current baseline can be ascertained, as well as work in progress. This, plus the configuration index, have improved the identification of the product makeup, to include vendor supplied OS, tools, and other support products.

Key Improvements

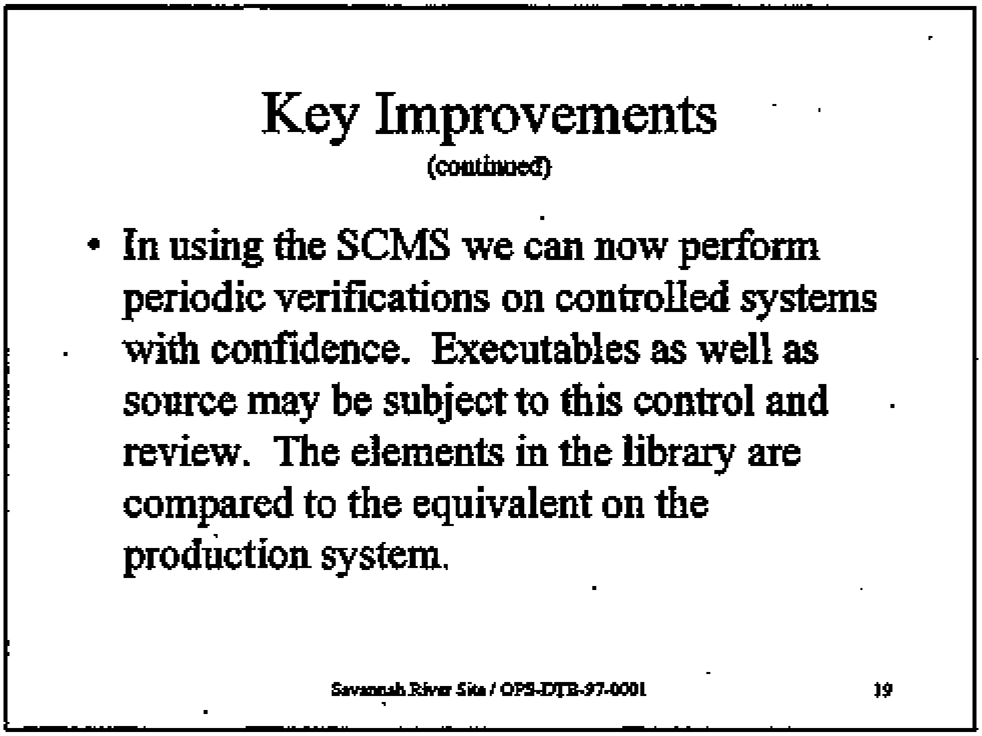
(continued)

- By using the SCMS functions provided, patches are now stored and verified. Reports from the developer produced after verification are checked against the same report run on the production system after the patch is installed. This provides instant feedback on possible errors introduced due to typo's or incorrect field modifications.

Key Improvements

(continued)

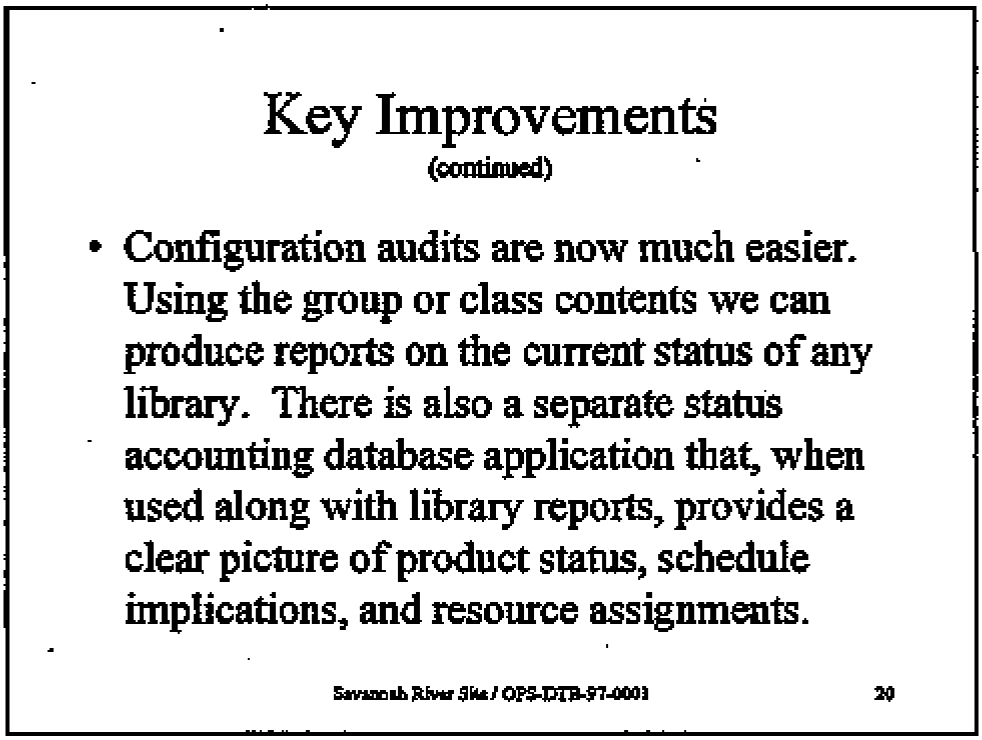
- By performing merges within the controlled environment of the library, a necessary function when allowing concurrent development, unknown file corruption has been reduced to zero incidents. There is still a chance of overwriting a file, but it will be discovered in the SCM process and final close out during the SCMC review.



Key Improvements

(continued)

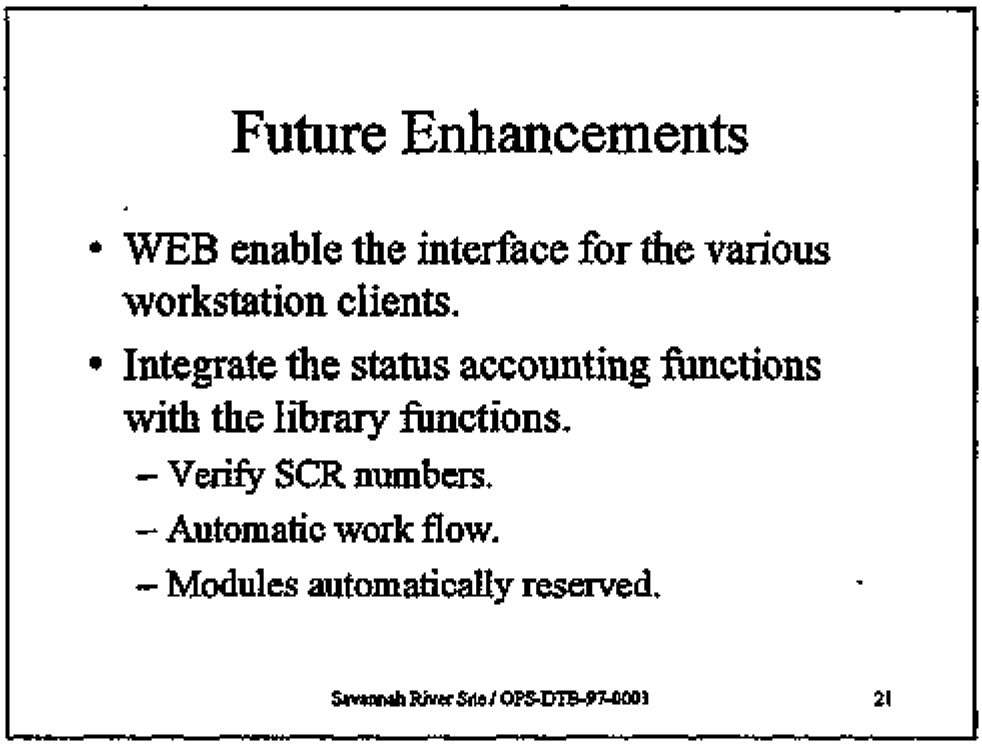
- In using the SCMS we can now perform periodic verifications on controlled systems with confidence. Executables as well as source may be subject to this control and review. The elements in the library are compared to the equivalent on the production system.



Key Improvements

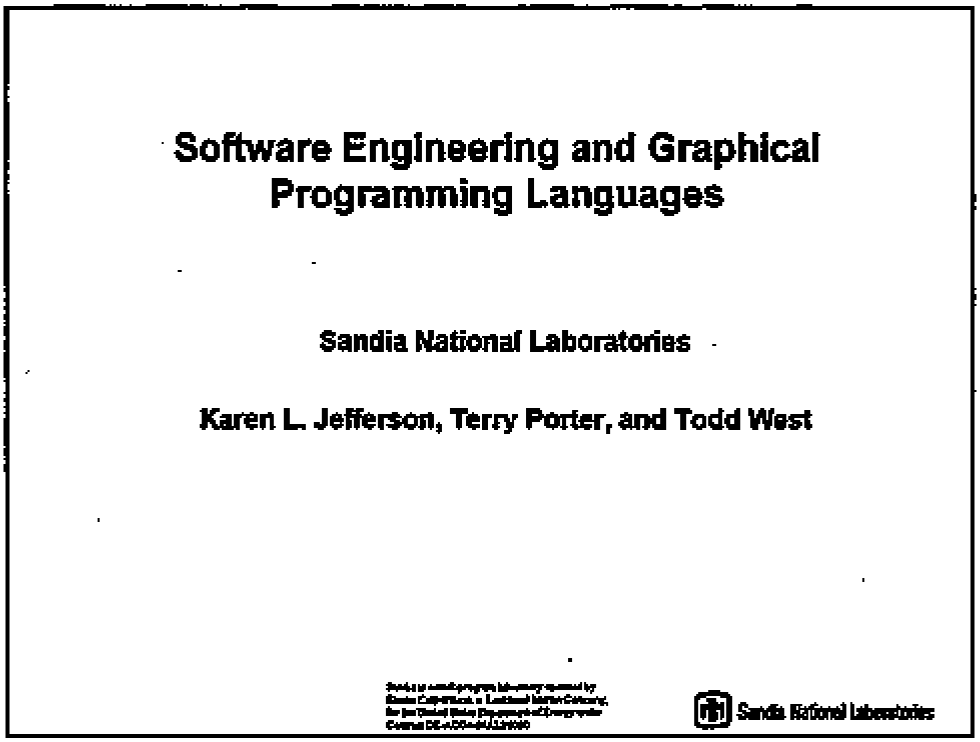
(continued)

- Configuration audits are now much easier. Using the group or class contents we can produce reports on the current status of any library. There is also a separate status accounting database application that, when used along with library reports, provides a clear picture of product status, schedule implications, and resource assignments.



Future Enhancements

- WEB enable the interface for the various workstation clients.
- Integrate the status accounting functions with the library functions.
  - Verify SCR numbers.
  - Automatic work flow.
  - Modules automatically reserved.



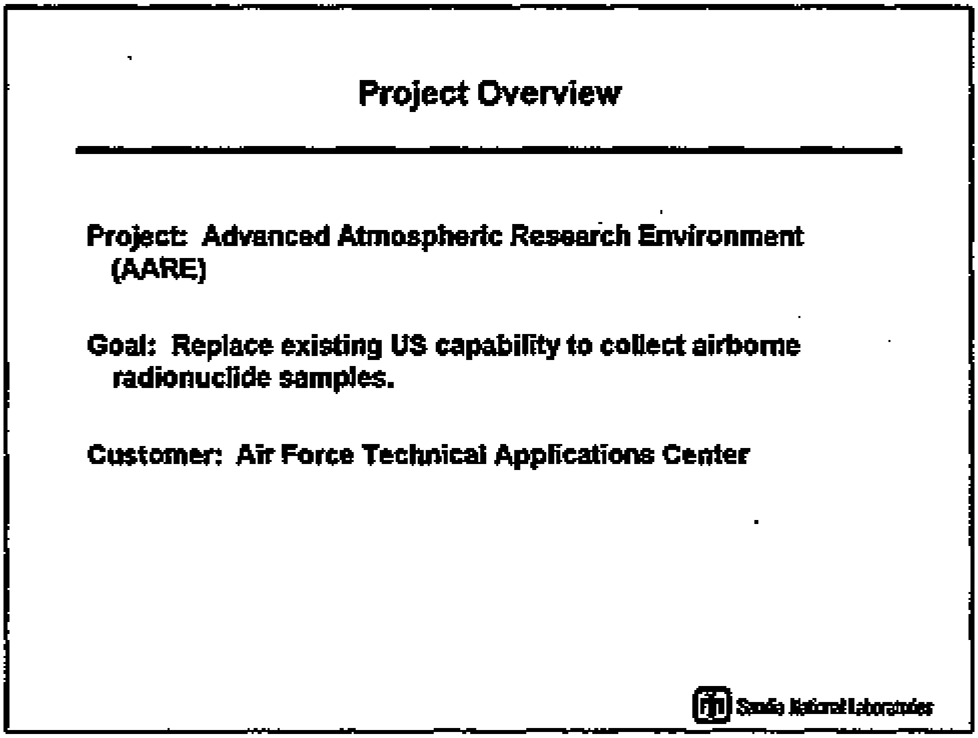
Software Engineering and Graphical Programming Languages

Sandia National Laboratories

Karen L. Jefferson, Terry Porter, and Todd West

Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under
Contract DE-AC04-94AL85000

Sandia National Laboratories



Project Overview

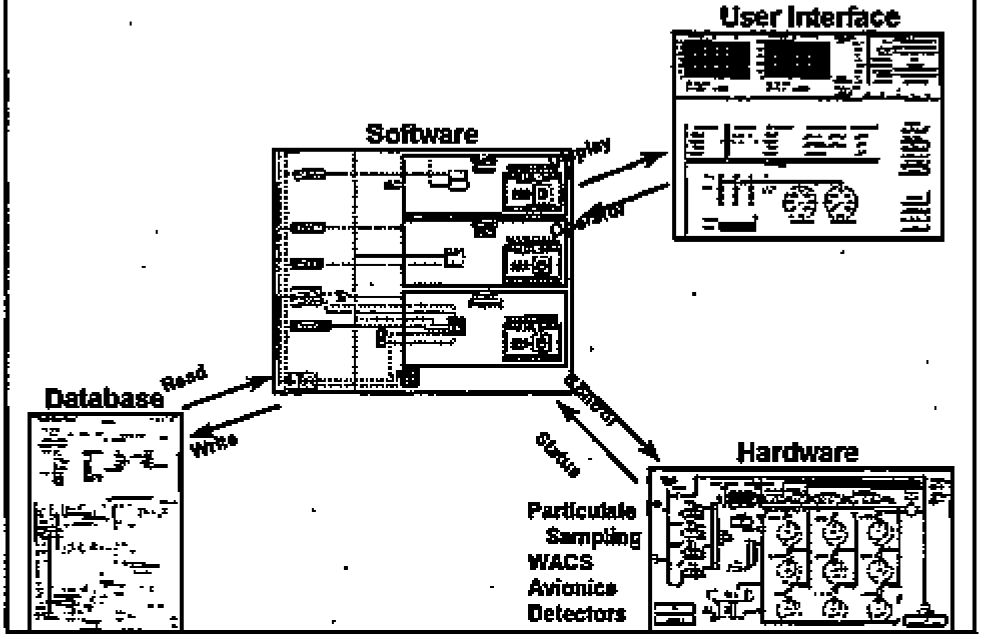
Project: Advanced Atmospheric Research Environment (AARE)

Goal: Replace existing US capability to collect airborne radionuclide samples.

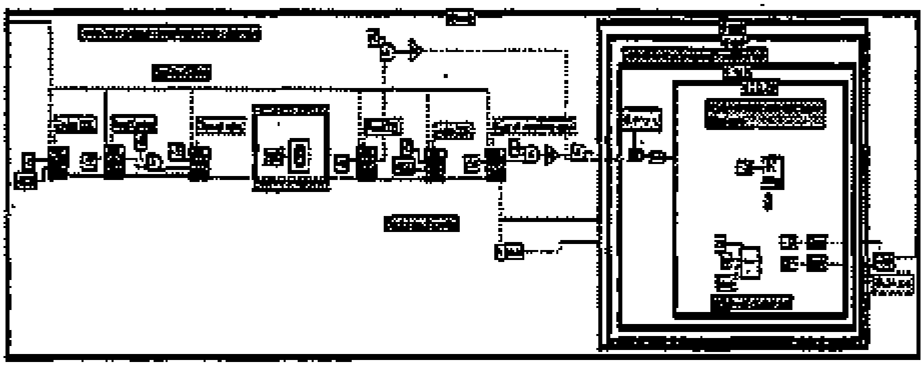
Customer: Air Force Technical Applications Center

Sandia National Laboratories

AARE Software Overview



Graphical Programming Languages



Customer Needs

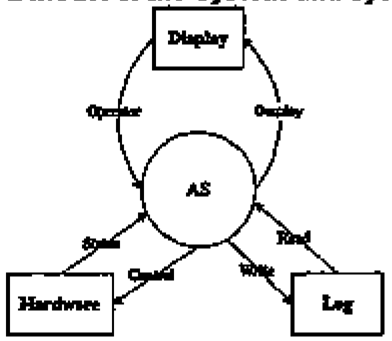
Maintainable and Reliable Software

- Required following elements from Mil-Std 491
  - Software Quality Plan
  - Software Requirements Document
  - Software Design Document
  - Software Test Plan
  - Coding Standards
  - Configuration Management
  - Test Log
  - Programmers Manual
  - Users Manual

Sandia National laboratories

Software Requirements Specification

- Developed a model of the system and system interactions.



- Developed and documented a syntax and semantics for the AARE stimulus/response language.
- Each stimulus/response pair was easily transformed into testable assertions.

Sandia National Laboratories

Software Requirements Specification

- Example

Ref.	Req Note	Stimulus	Condition
[illegible]	[illegible]	[illegible]	
		[illegible]	[illegible]
		[illegible]	[illegible]
		[illegible]	[illegible]
		[illegible]	[illegible]
		[illegible]	[illegible]

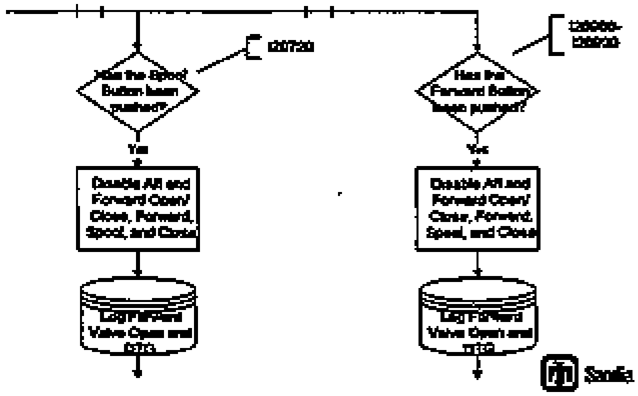
Response	Req. Note
[illegible] ← 'Close valves before forwarding pages."	[illegible]

- Each stimulus/response pair was independent which mirrored LabVIEW's undetermined execution ordering.
- Traced system requirements to software requirements.

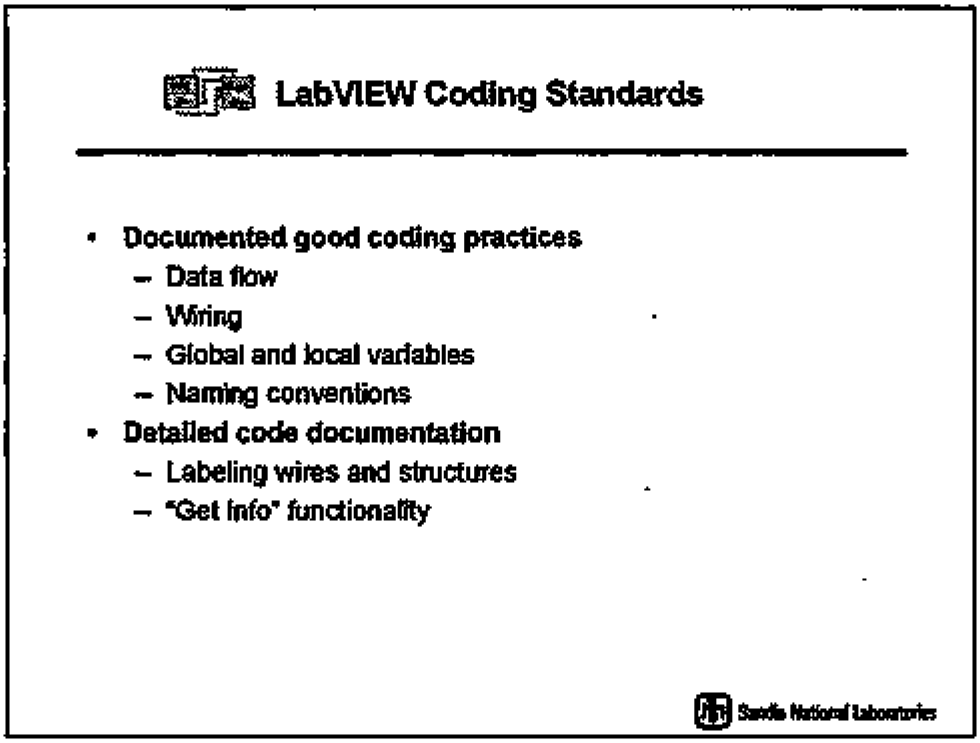
Sandia National Laboratories

Software Design Document

- Reflected data flow paradigm of LabVIEW.
- Tied design elements to specific software requirements.
- Example


120720
Has the Spool Button been pushed?
Yes
Disable All and Forward Open/ Close, Forward, Spool, and Close
Log Forward Valve Open and [illegible]
120000-120030
Has the Forward Button been pushed?
Yes
Disable All and Forward Open/ Close, Forward, Spool, and Close
Log Forward Valve Open and [illegible]

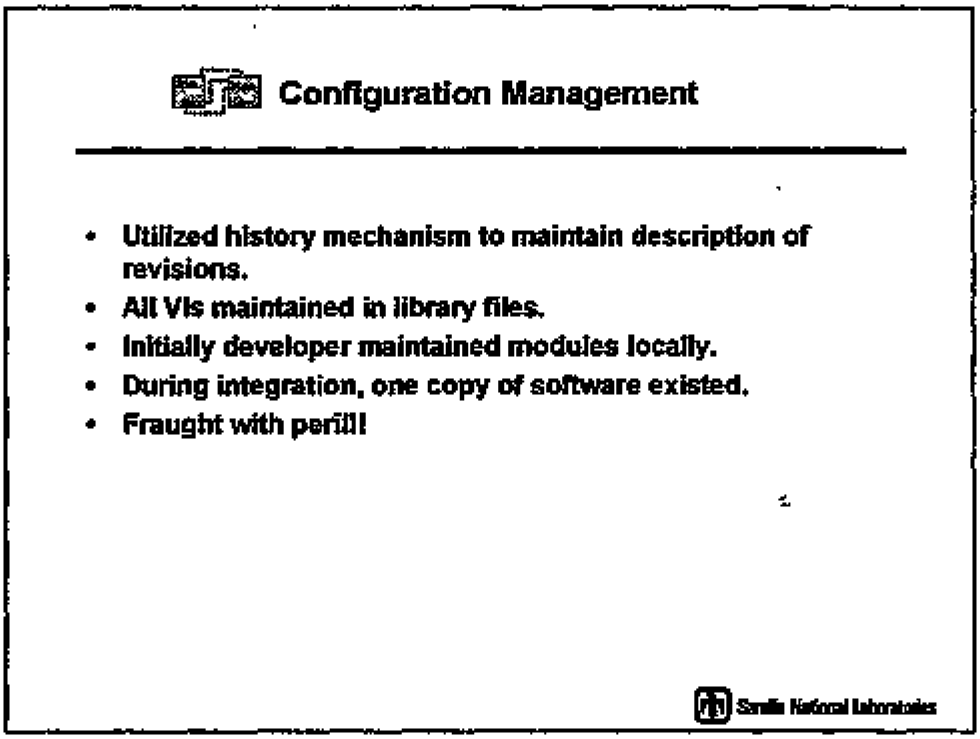
Sandia National Laboratories



LabVIEW Coding Standards

- **Documented good coding practices**
  - Data flow
  - Wiring
  - Global and local variables
  - Naming conventions
- **Detailed code documentation**
  - Labeling wires and structures
  - "Get Info" functionality

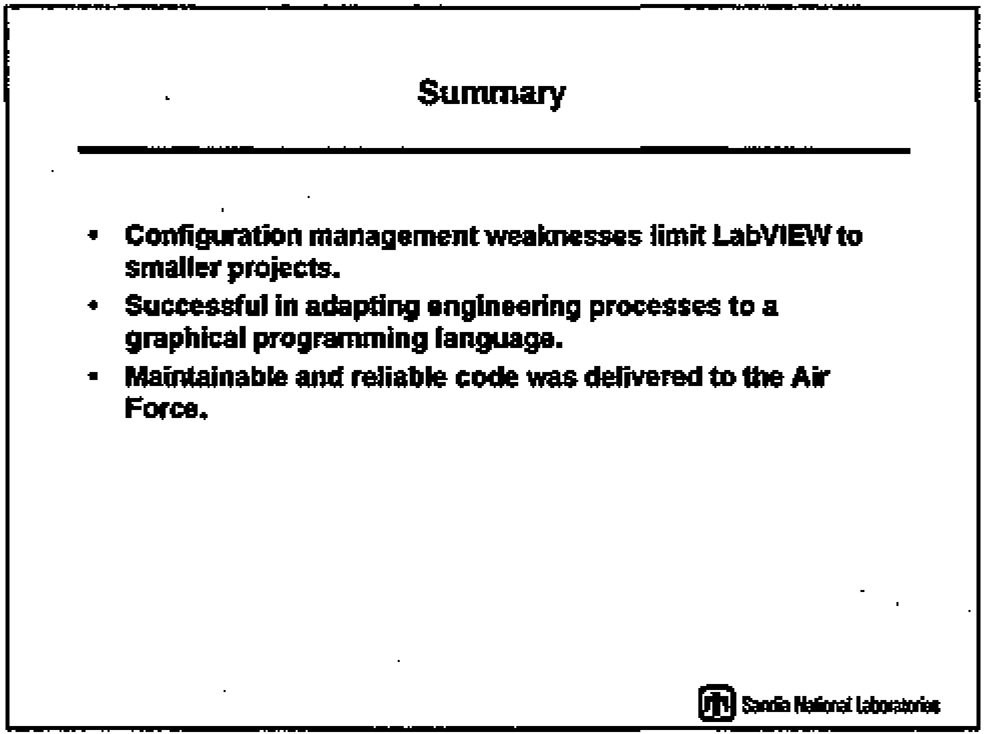
Sandia National Laboratories



Configuration Management

- **Utilized history mechanism to maintain description of revisions.**
- **All VIs maintained in library files.**
- **Initially developer maintained modules locally.**
- **During integration, one copy of software existed.**
- **Fraught with peril!**

Sandia National Laboratories



Summary

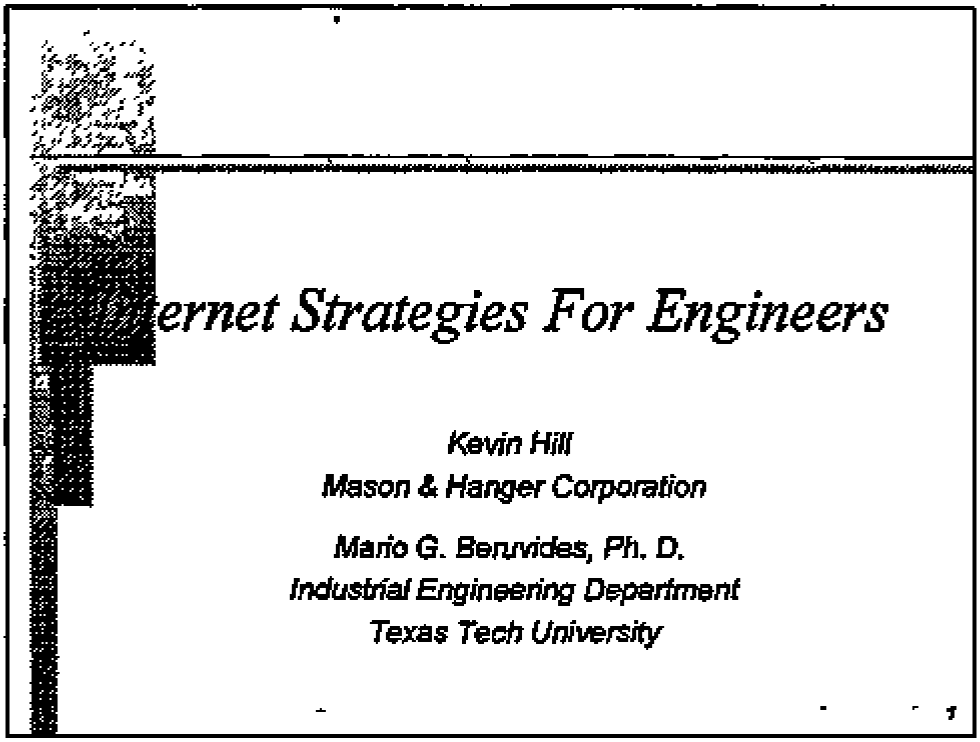
- Configuration management weaknesses limit LabVIEW to smaller projects.
- Successful in adapting engineering processes to a graphical programming language.
- Maintainable and reliable code was delivered to the Air Force.

Sandia National Laboratories

Session B2: Internet WEB Applications

Chair Faye Brown
Lockheed Martin Energy Systems

Session : Paper #	Author(s)	*Title*
B2:1	Kevin Hill Pantex Plant	*Internet Strategies for Engineers*
B2:2	David Leong & Fran Current Sandia National Laboratories	*Exploiting the Intranet: A New Architecture for Enterprise Information*
B2:3	Jennie Negin Sandia National Laboratories	*"Rightsizing" Software Quality for a Web Services Organization*

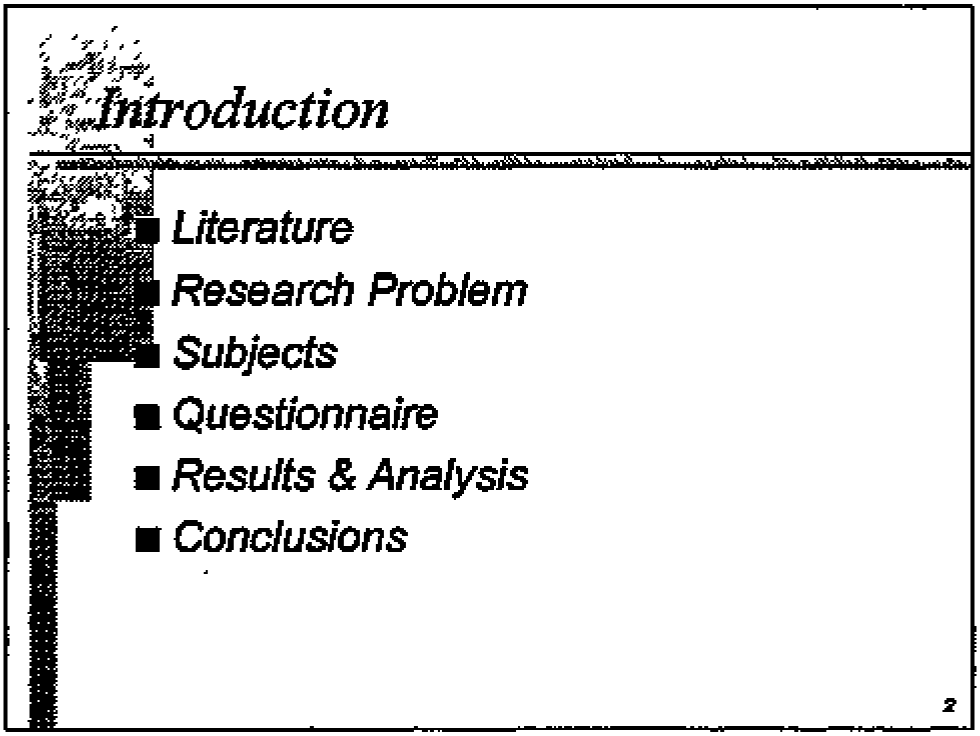


Internet Strategies For Engineers

Kevin Hill
Mason & Hanger Corporation

Mario G. Beruvides, Ph. D.
Industrial Engineering Department
Texas Tech University

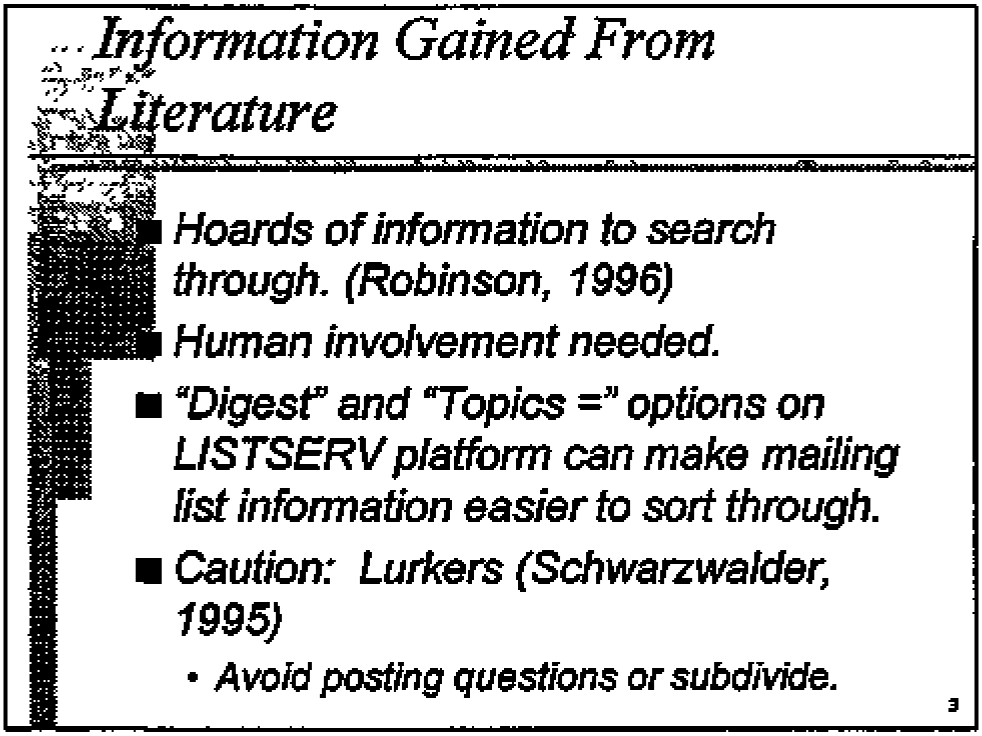
1



Introduction

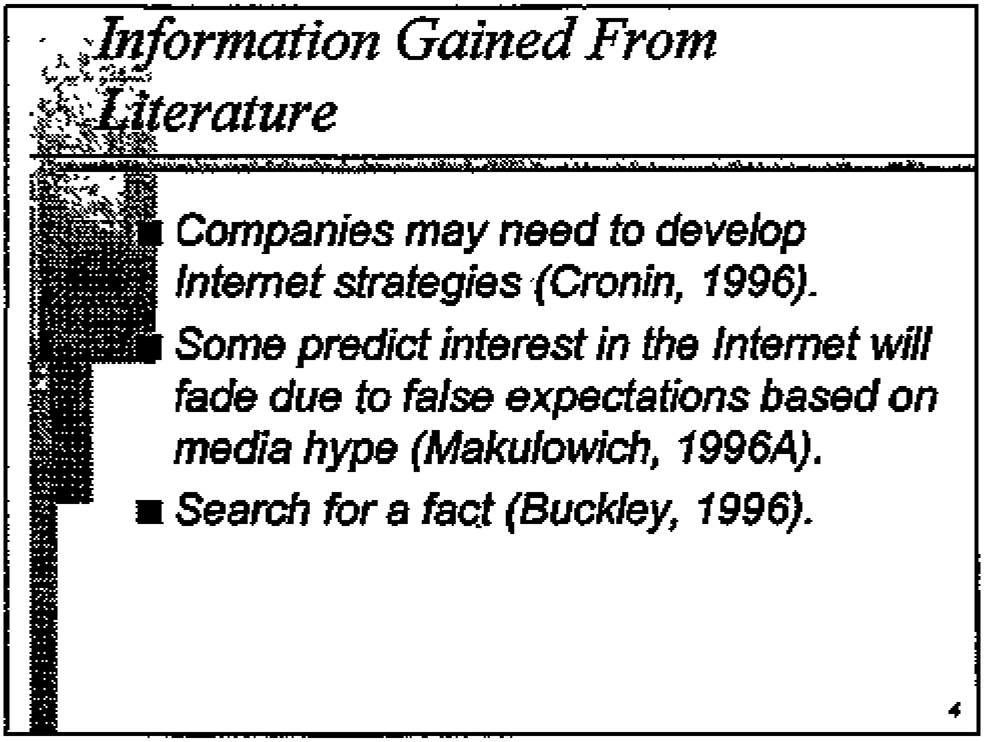
- Literature
- Research Problem
- Subjects
- Questionnaire
- Results & Analysis
- Conclusions

2



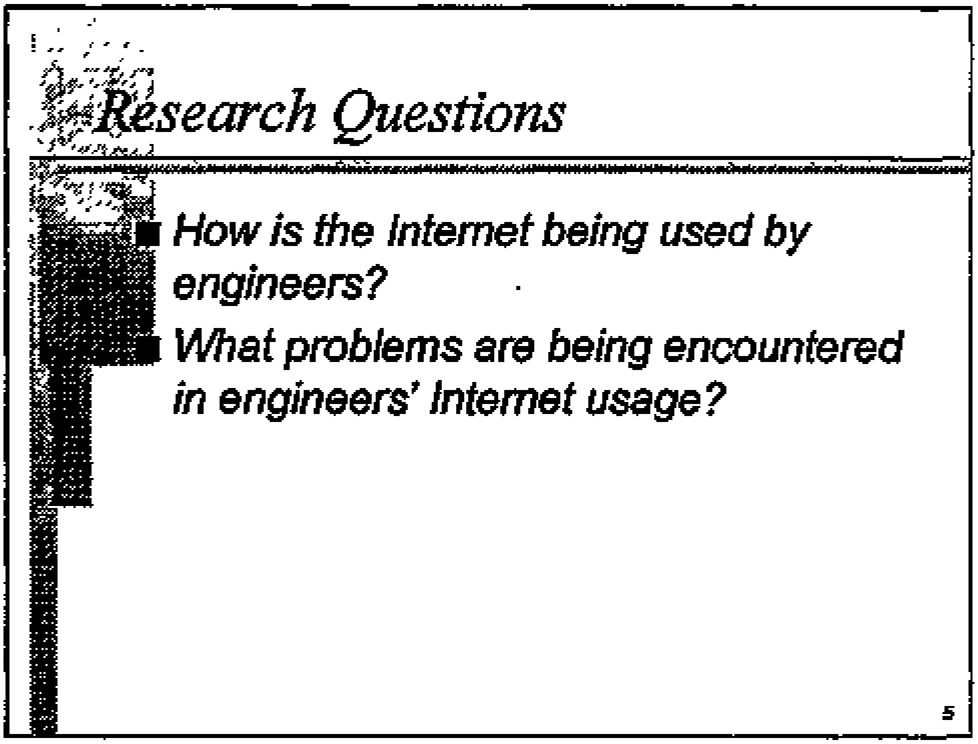
Information Gained From Literature

- Hoards of information to search through. (Robinson, 1996)
- Human involvement needed.
- "Digest" and "Topics =" options on LISTSERV platform can make mailing list information easier to sort through.
- Caution: Lurkers (Schwarzwalder, 1995)
  - Avoid posting questions or subdivide.



Information Gained From Literature

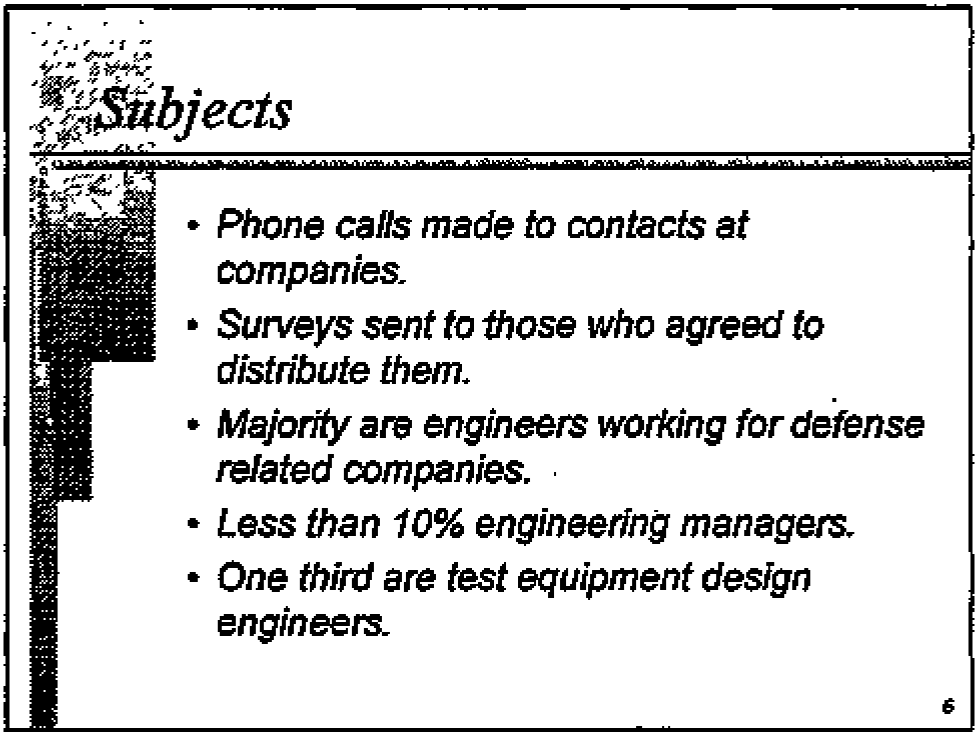
- Companies may need to develop Internet strategies (Cronin, 1996).
- Some predict interest in the Internet will fade due to false expectations based on media hype (Makulowich, 1996A).
- Search for a fact (Buckley, 1996).



Research Questions

- How is the Internet being used by engineers?
- What problems are being encountered in engineers' Internet usage?

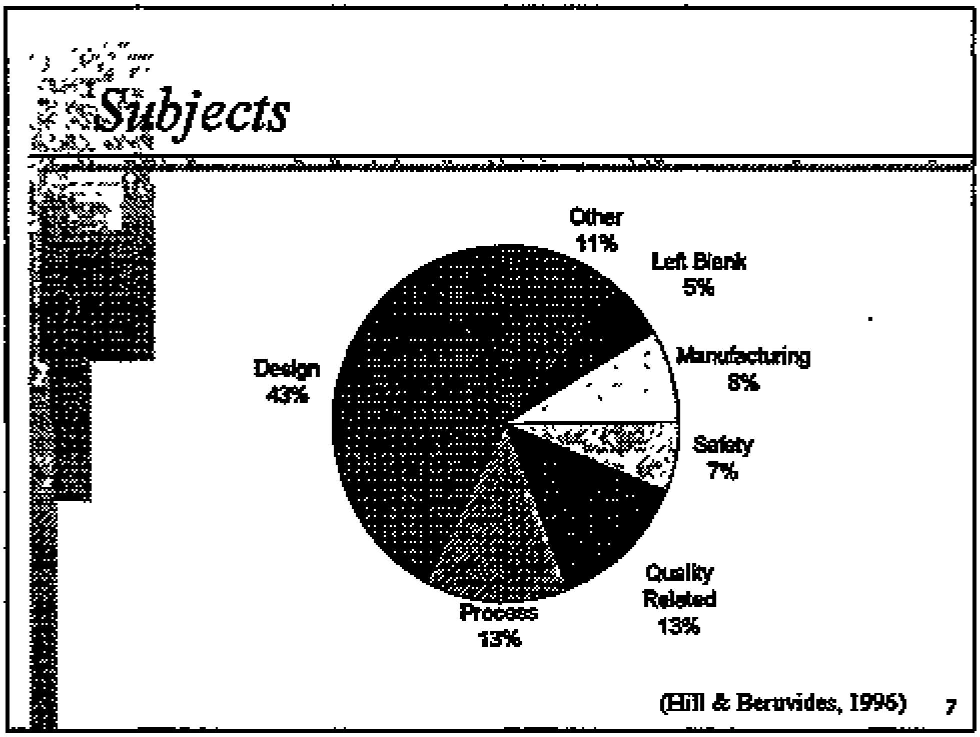
5



Subjects

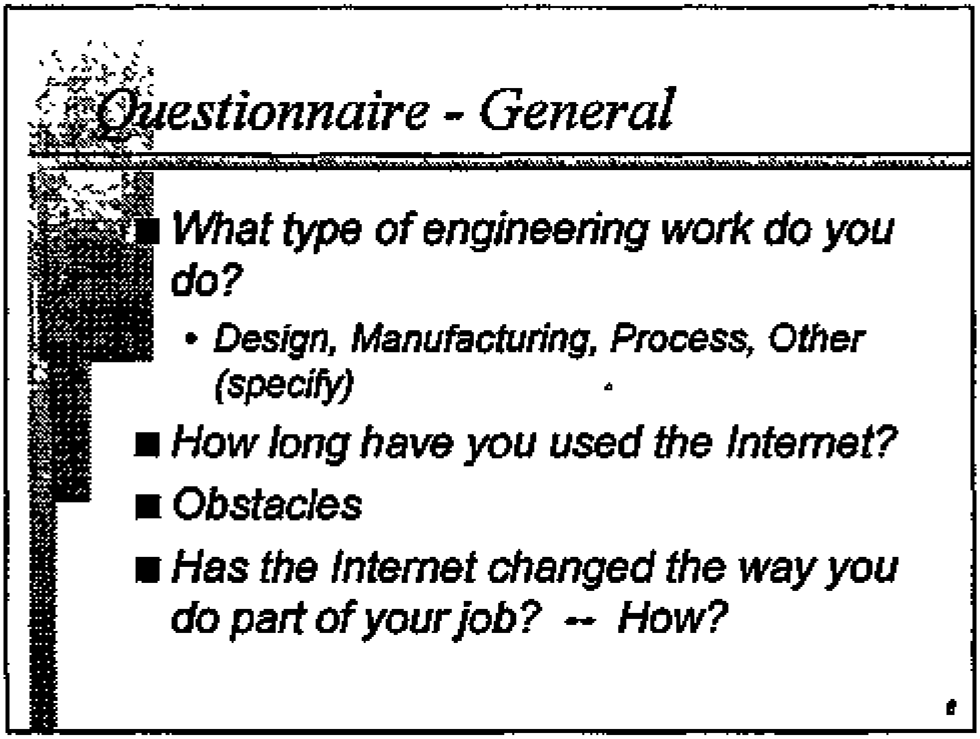
- Phone calls made to contacts at companies.
- Surveys sent to those who agreed to distribute them.
- Majority are engineers working for defense related companies.
- Less than 10% engineering managers.
- One third are test equipment design engineers.

6



Subjects

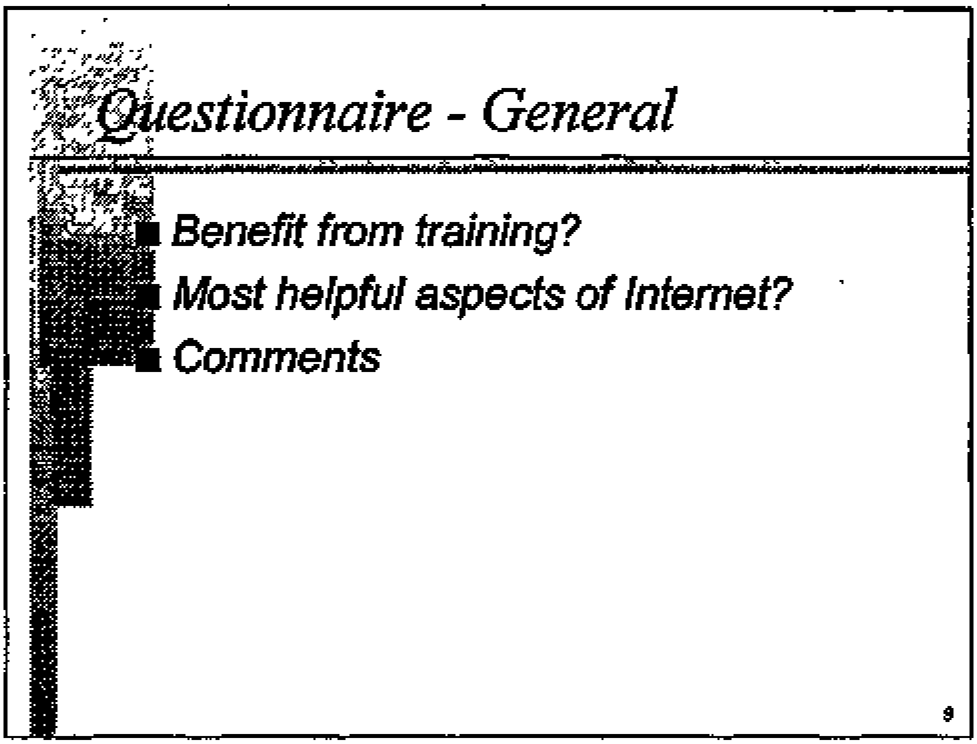
(Hill & Beruvides, 1996) 7



Questionnaire - General

- *What type of engineering work do you do?*
  - *Design, Manufacturing, Process, Other (specify)*
- *How long have you used the Internet?*
- *Obstacles*
- *Has the Internet changed the way you do part of your job? -- How?*

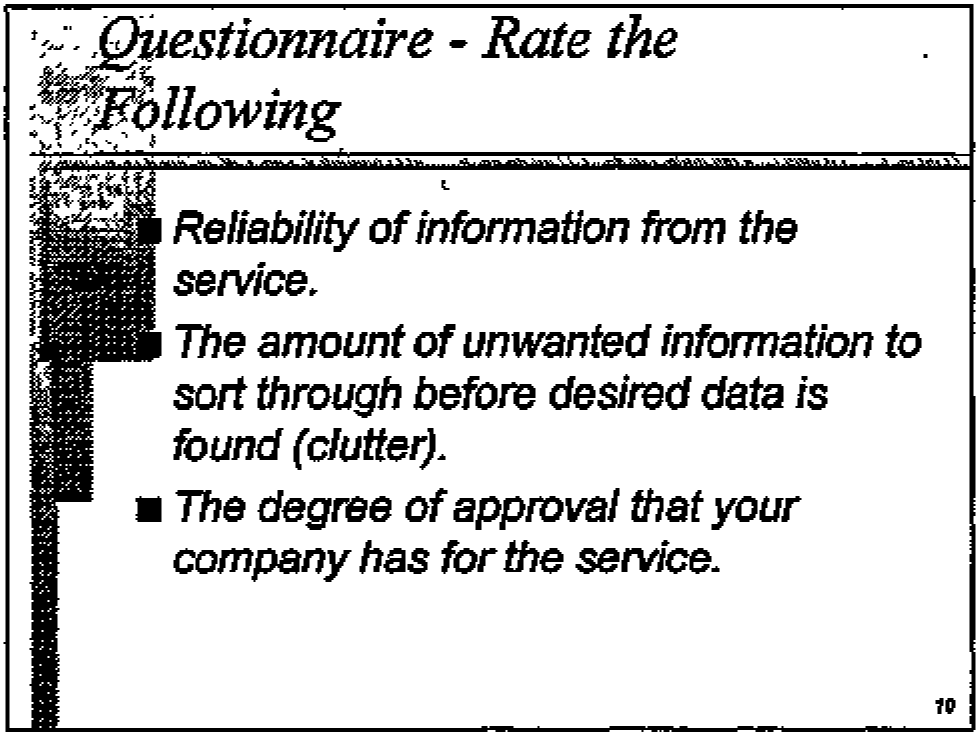
8



Questionnaire - General

- Benefit from training?
- Most helpful aspects of Internet?
- Comments

9



Questionnaire - Rate the Following

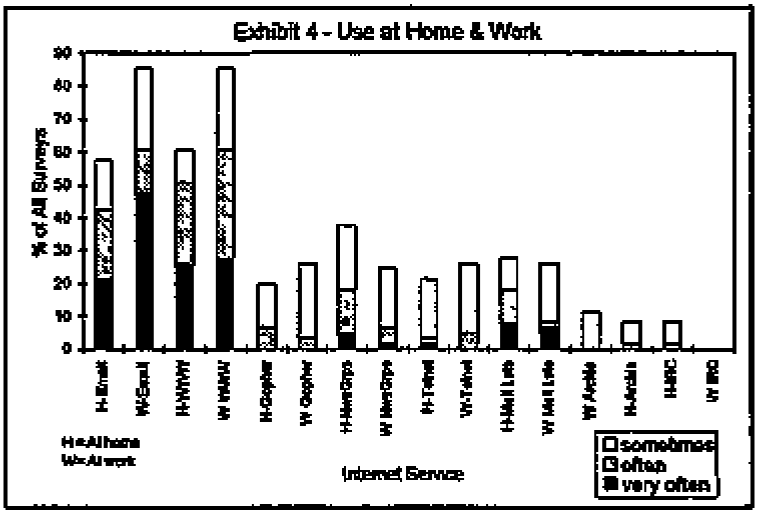
- Reliability of information from the service.
- The amount of unwanted information to sort through before desired data is found (clutter).
- The degree of approval that your company has for the service.

10

Results

- Response rate of 67% (61 of 91 surveys completed and returned).
- Surveys received from 6 states and at least 5 companies.

Results



(Hill & Benvides, 1996)

Results

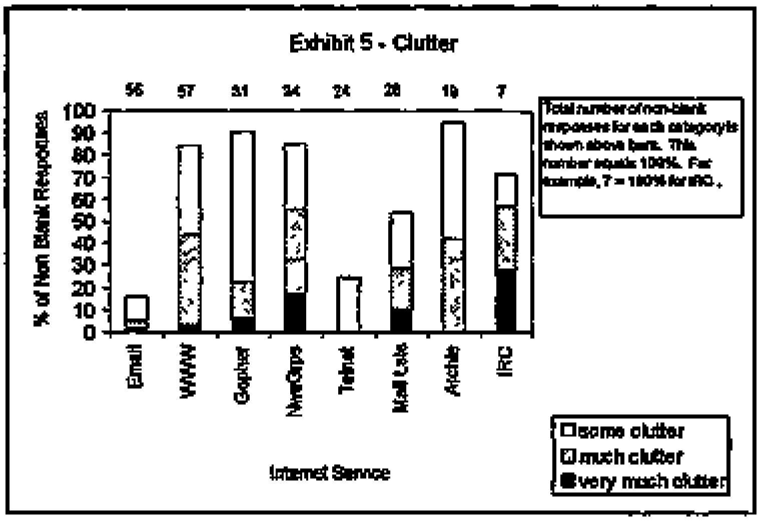


Exhibit 5 - Clutter

Total number of non-blank responses for each category is shown above bars. This number equals 100%. For example, 7 = 100% for IRC.

(Hill & Beruvides, 1996) 13

Results

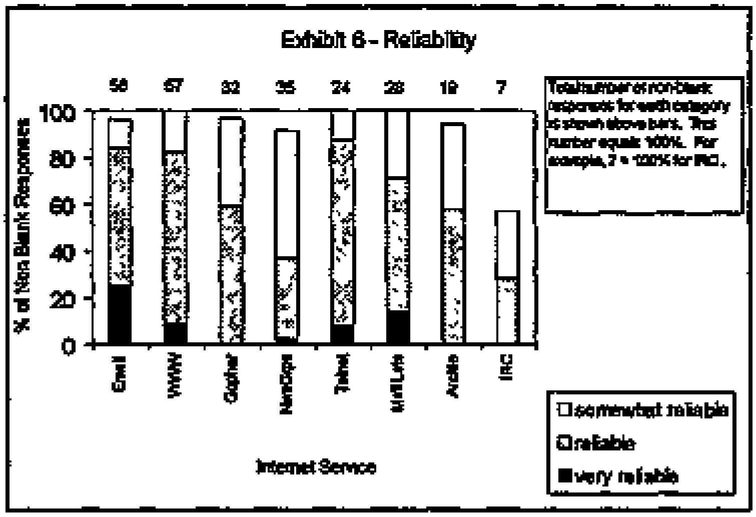
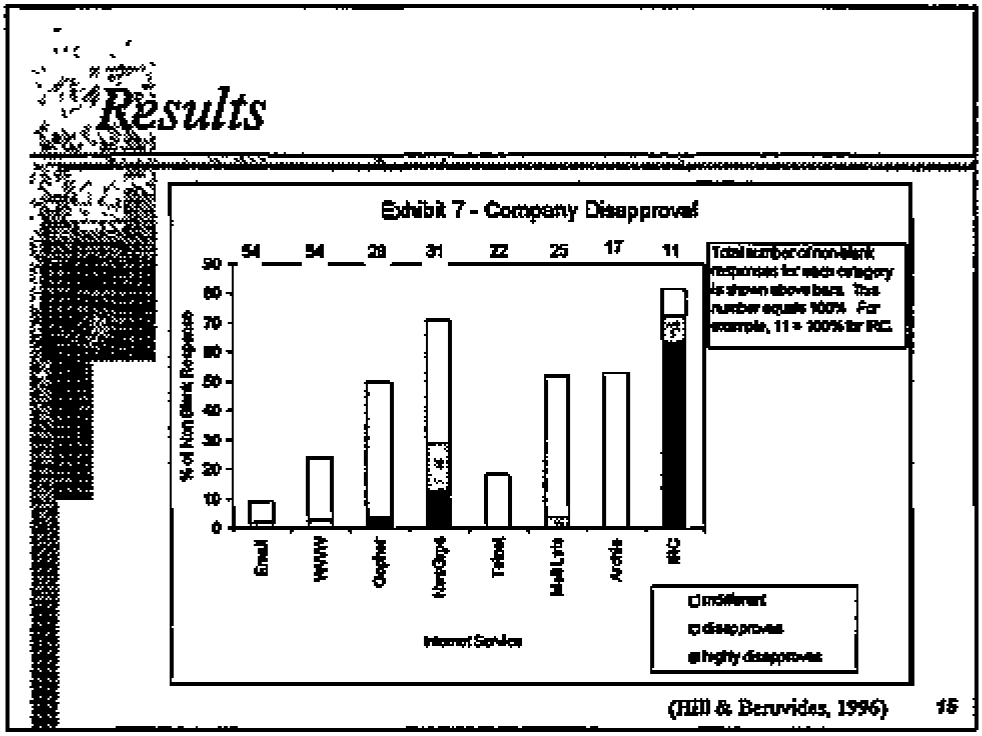


Exhibit 6 - Reliability

Total number of non-blank responses for each category is shown above bars. This number equals 100%. For example, 7 = 100% for IRC.

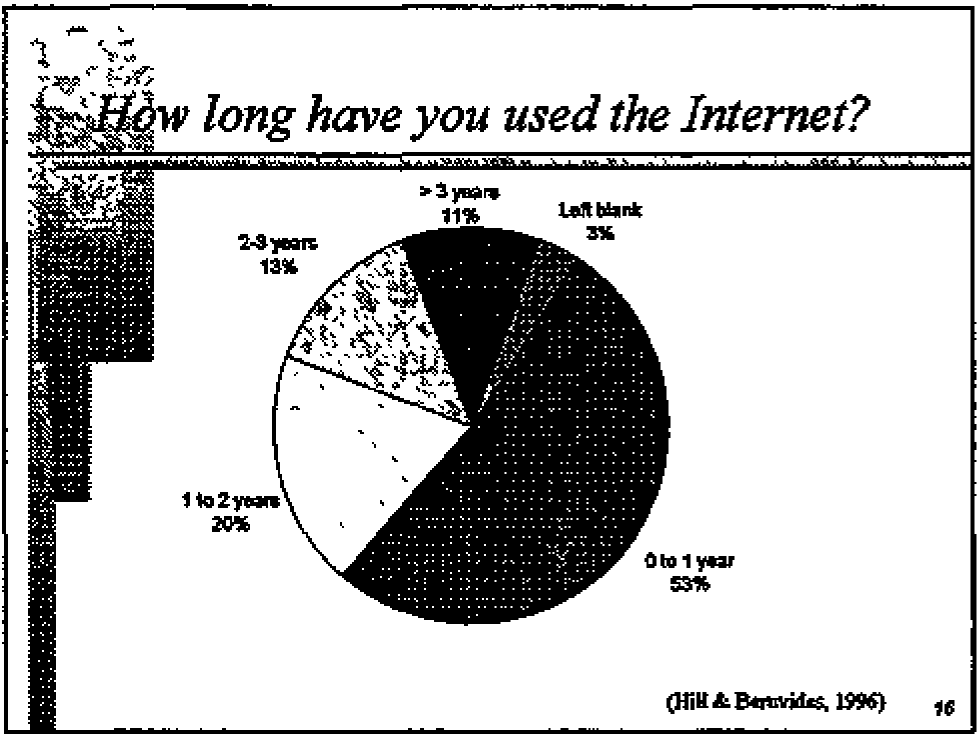
(Hill & Beruvides, 1996) 14



Results

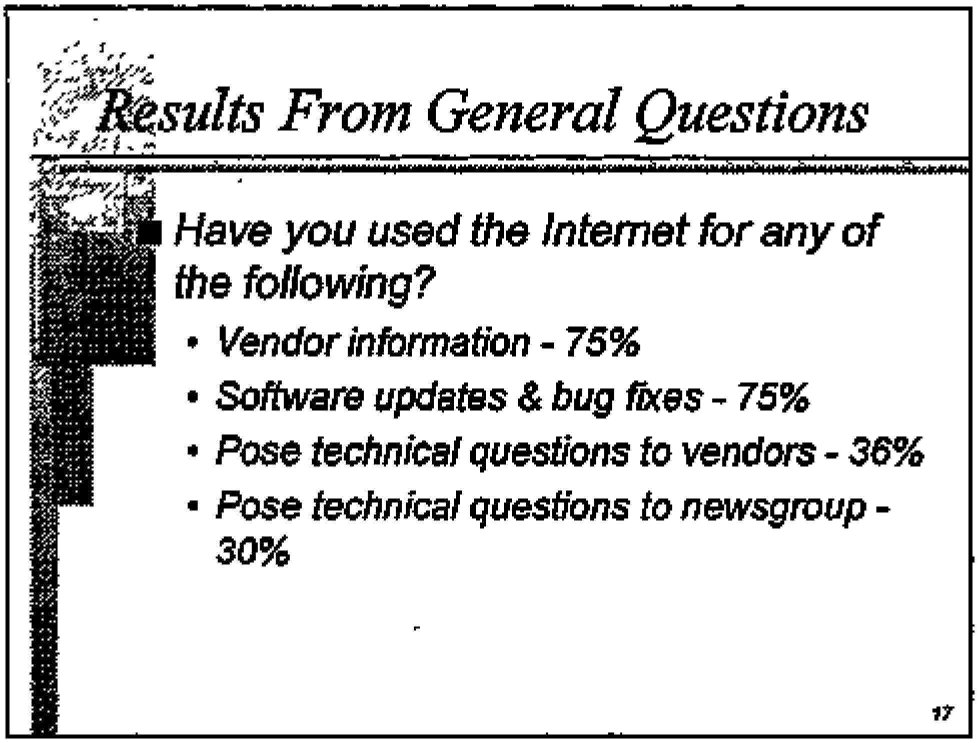
Total number of non-blank responses for each category is shown above bars. This number equals 100%. For example, 11 = 100% for IRC.

(Hill & Beruvides, 1996) 15



How long have you used the Internet?

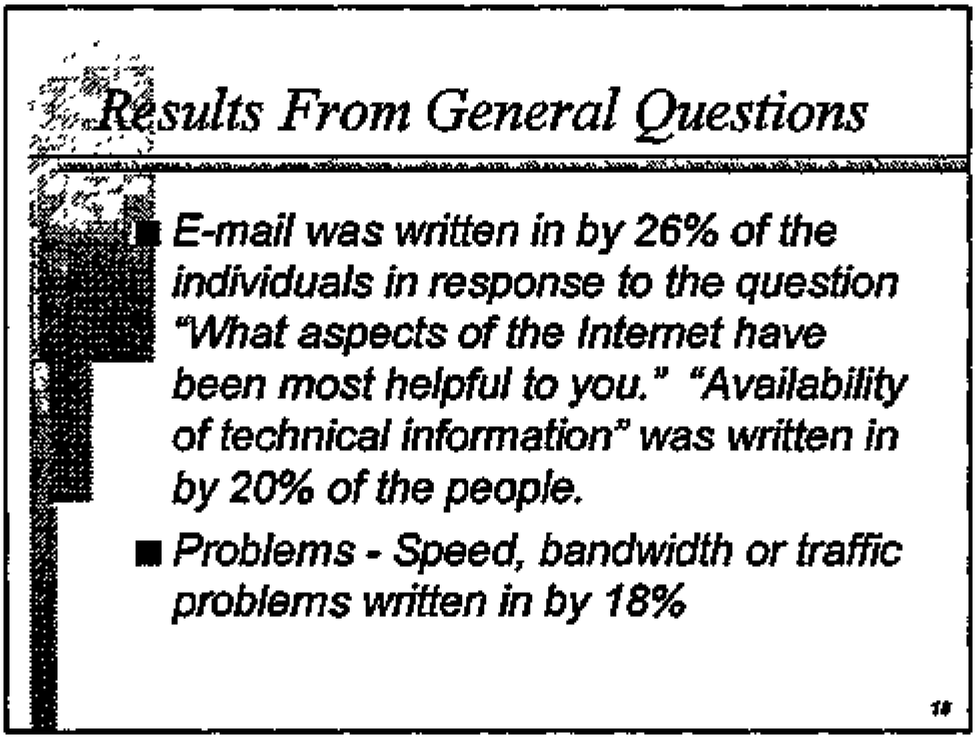
(Hill & Beruvides, 1996) 16



Results From General Questions

- Have you used the Internet for any of the following?
  - Vendor information - 75%
  - Software updates & bug fixes - 75%
  - Pose technical questions to vendors - 36%
  - Pose technical questions to newsgroup - 30%

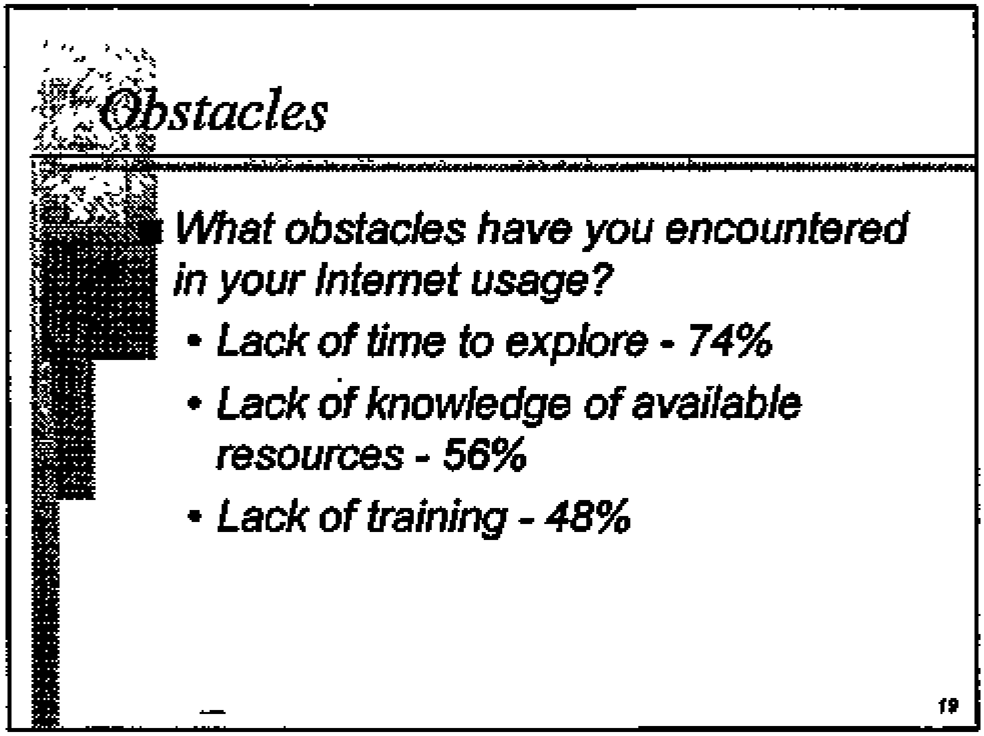
17



Results From General Questions

- E-mail was written in by 26% of the individuals in response to the question "What aspects of the Internet have been most helpful to you." "Availability of technical information" was written in by 20% of the people.
- Problems - Speed, bandwidth or traffic problems written in by 18%

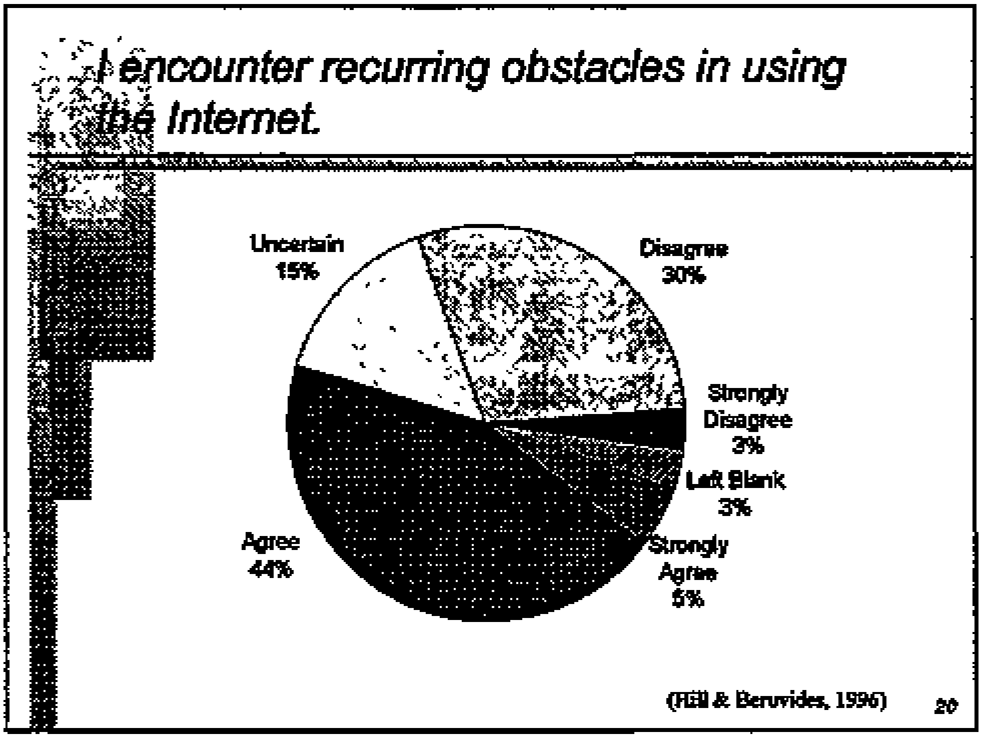
18



Obstacles

- What obstacles have you encountered in your Internet usage?
  - Lack of time to explore - 74%
  - Lack of knowledge of available resources - 56%
  - Lack of training - 48%

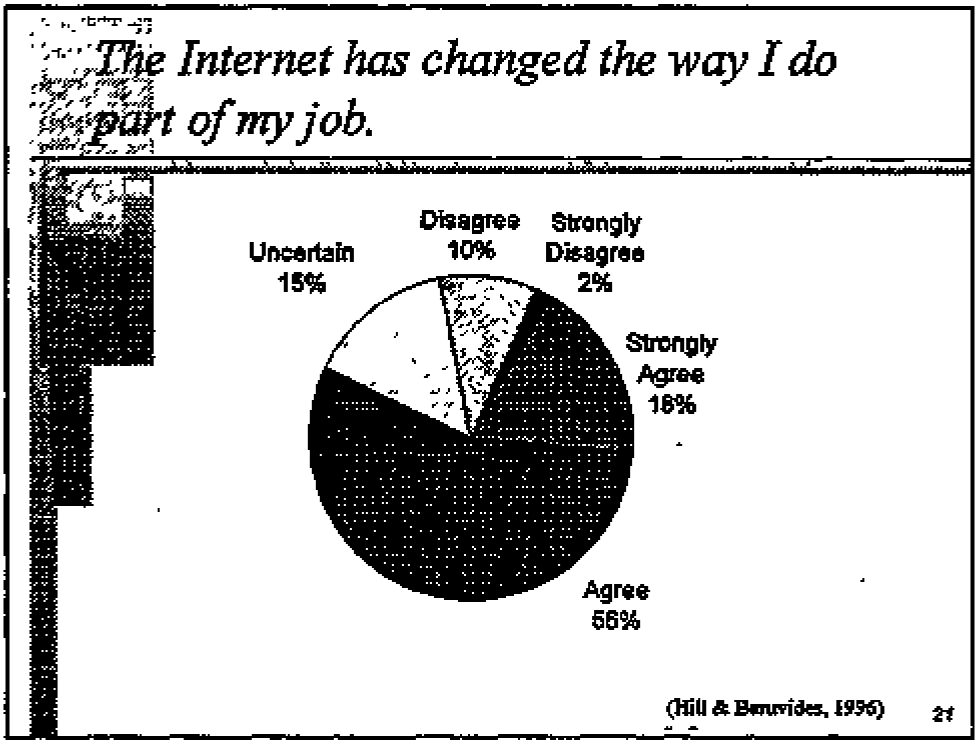
19



I encounter recurring obstacles in using the Internet.

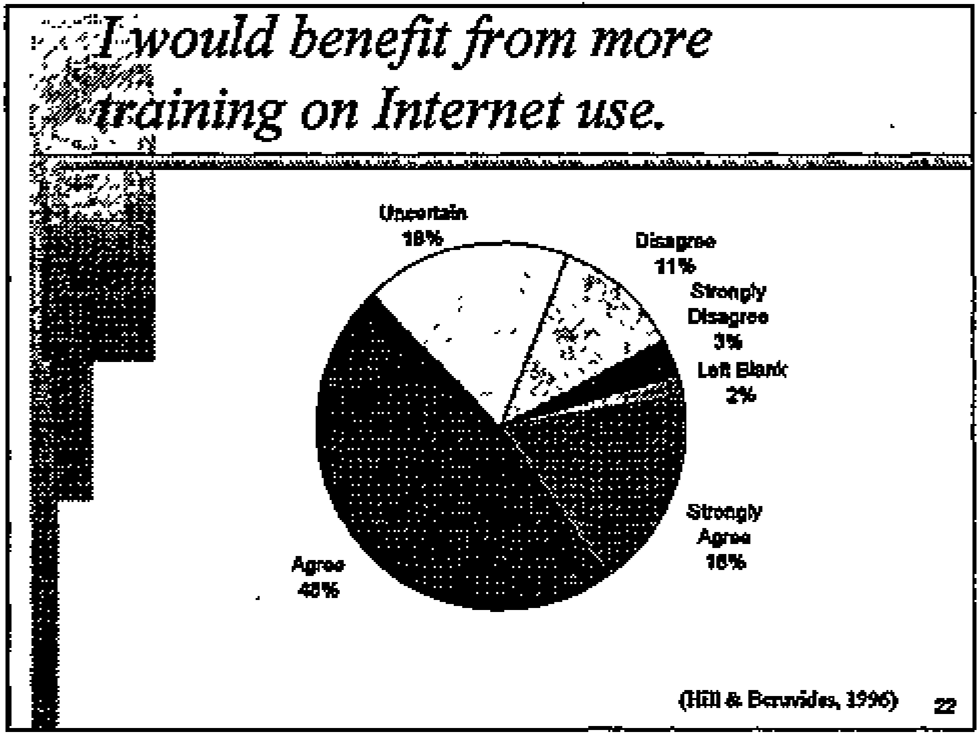
(Hill & Beruvides, 1996)

20



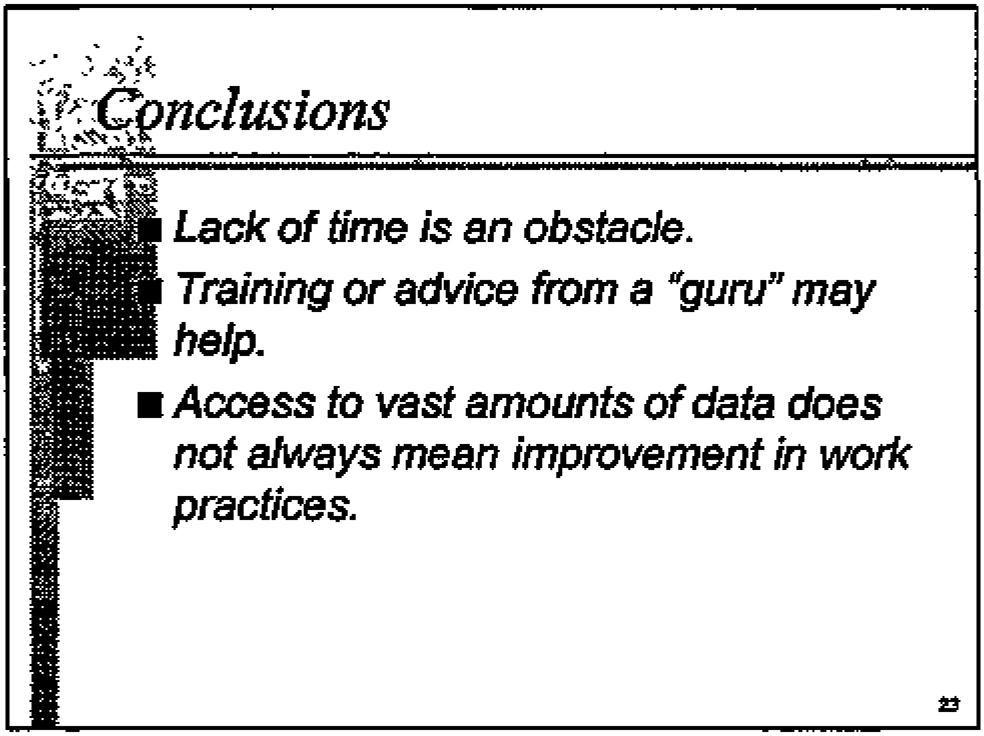
The Internet has changed the way I do part of my job.

(Hill & Benavides, 1996) 21



I would benefit from more training on Internet use.

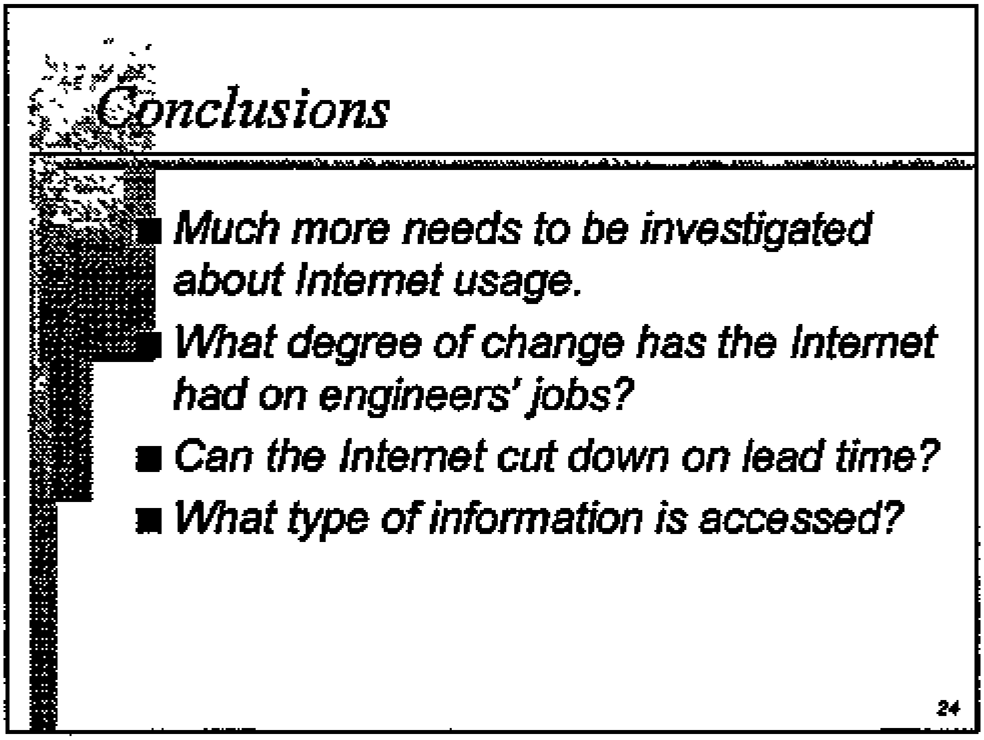
(Hill & Benavides, 1996) 22



Conclusions

- Lack of time is an obstacle.
- Training or advice from a "guru" may help.
- Access to vast amounts of data does not always mean improvement in work practices.

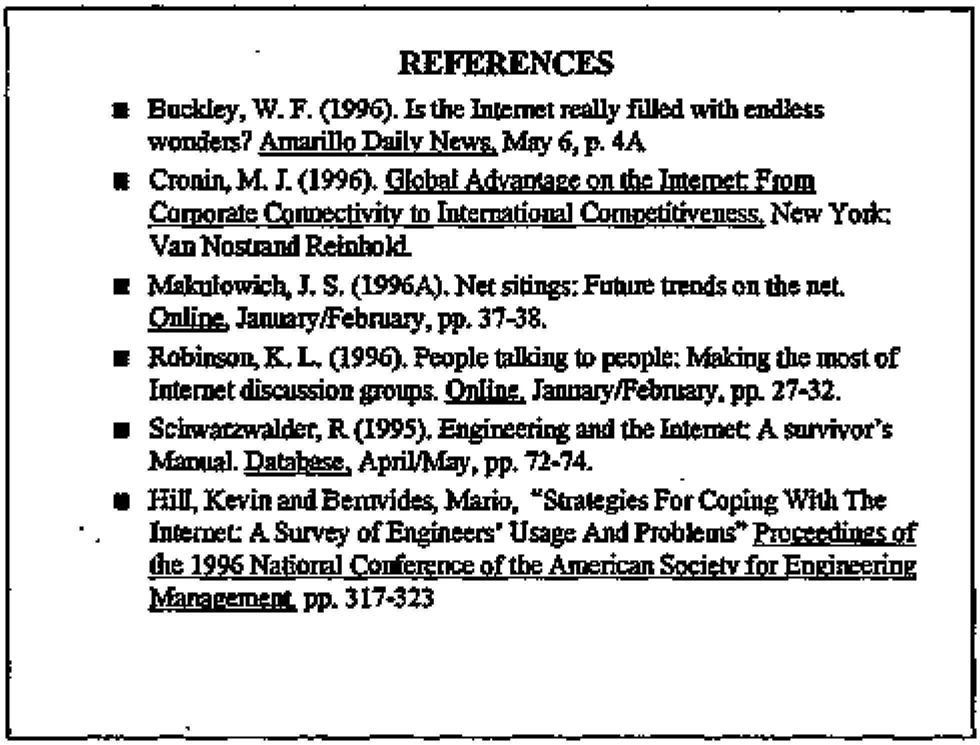
23



Conclusions

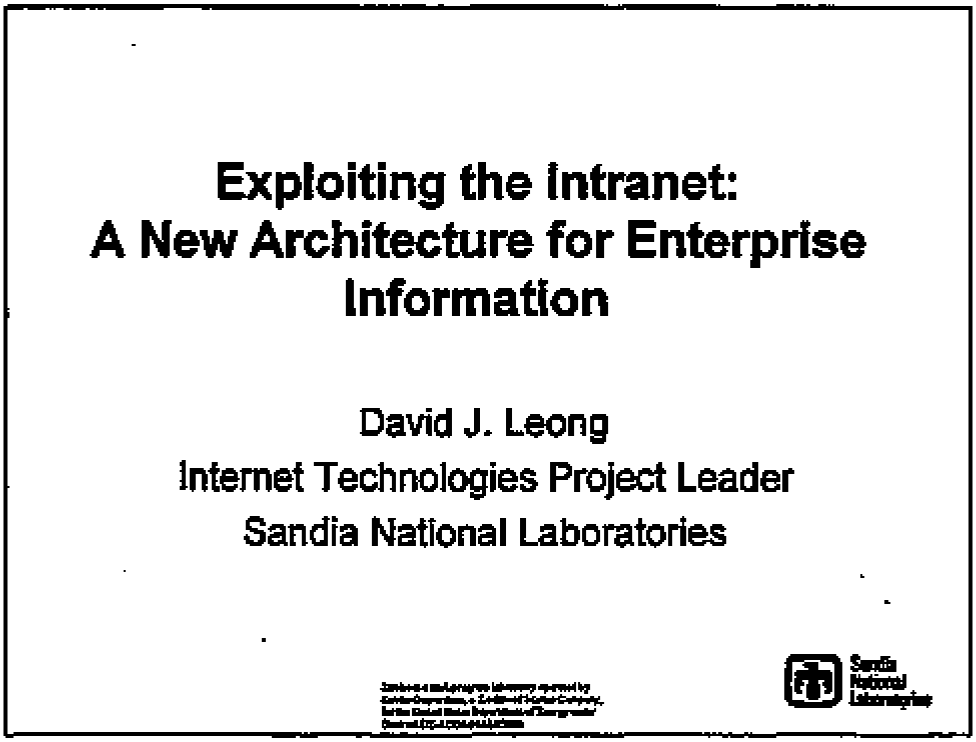
- Much more needs to be investigated about Internet usage.
- What degree of change has the Internet had on engineers' jobs?
- Can the Internet cut down on lead time?
- What type of information is accessed?

24



REFERENCES

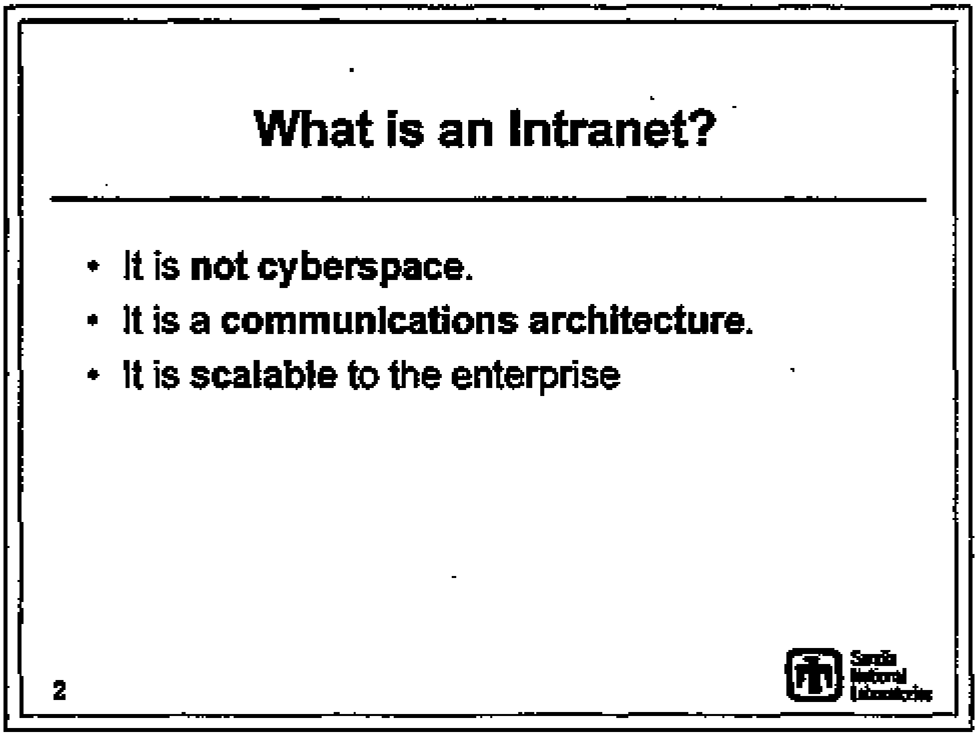
- Buckley, W. F. (1996). Is the Internet really filled with endless wonders? Amarillo Daily News, May 6, p. 4A.
- Cronin, M. J. (1996). Global Advantage on the Internet: From Corporate Connectivity to International Competitiveness. New York: Van Nostrand Reinhold.
- Makulowich, J. S. (1996A). Net sitings: Future trends on the net. Online, January/February, pp. 37-38.
- Robinson, K. L. (1996). People talking to people: Making the most of Internet discussion groups. Online, January/February, pp. 27-32.
- Schwarzwalder, R (1995). Engineering and the Internet: A survivor's Manual. Database, April/May, pp. 72-74.
- Hill, Kevin and Benvides, Mario, "Strategies For Coping With The Internet: A Survey of Engineers' Usage And Problems" Proceedings of the 1996 National Conference of the American Society for Engineering Management, pp. 317-323



Exploiting the Intranet: A New Architecture for Enterprise Information

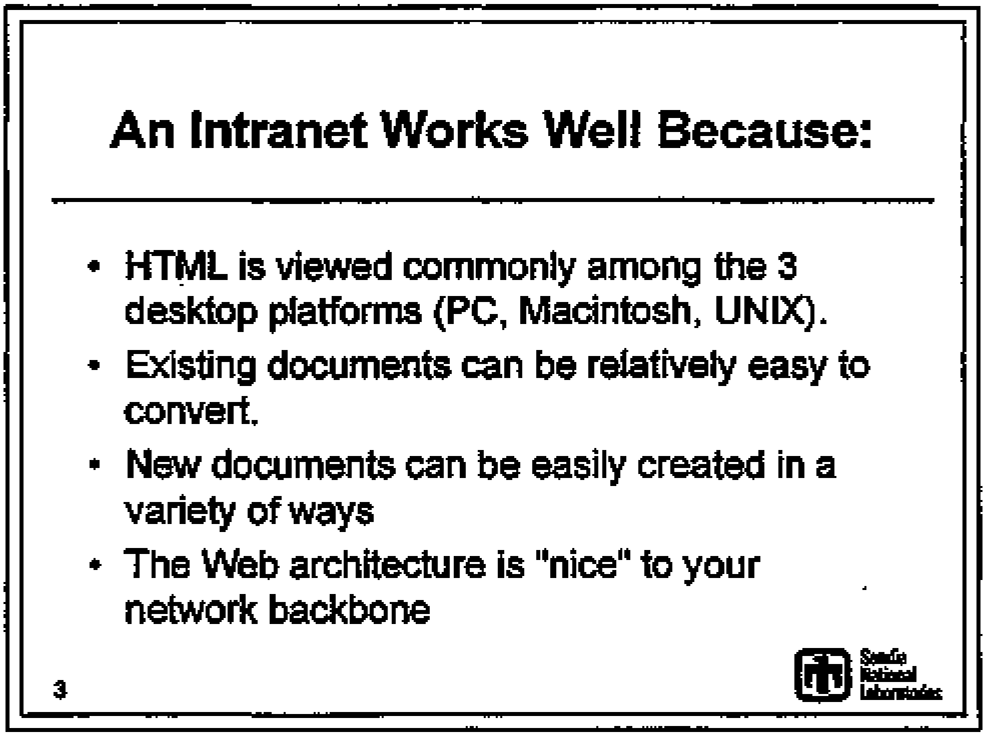
David J. Leong
Internet Technologies Project Leader
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under
Contract DE-AC04-94AL85000.



What is an Intranet?

- It is **not cyberspace.**
- It is a **communications architecture.**
- It is **scalable** to the enterprise

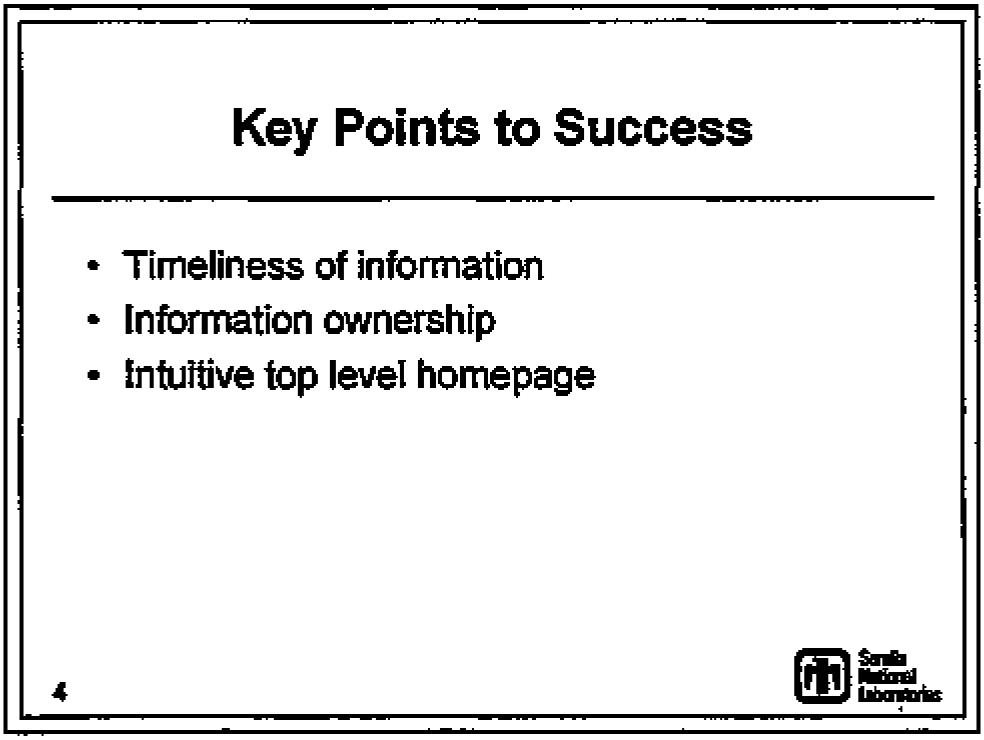


An Intranet Works Well Because:

- HTML is viewed commonly among the 3 desktop platforms (PC, Macintosh, UNIX).
- Existing documents can be relatively easy to convert.
- New documents can be easily created in a variety of ways
- The Web architecture is "nice" to your network backbone

3

Sandia National Laboratories

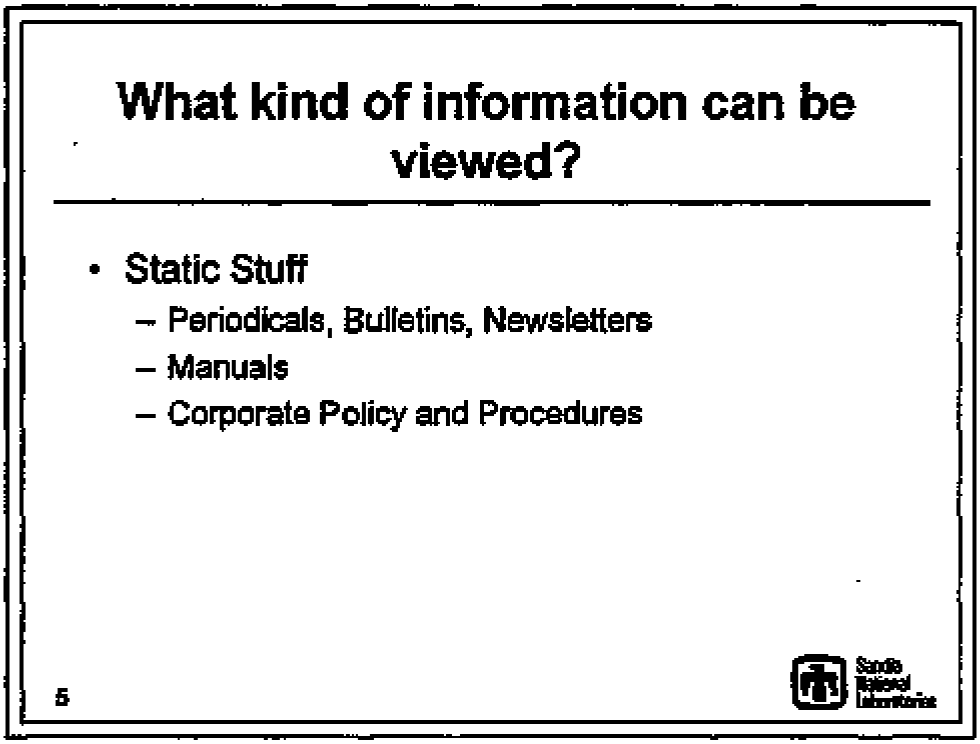


Key Points to Success

- Timeliness of information
- Information ownership
- Intuitive top level homepage

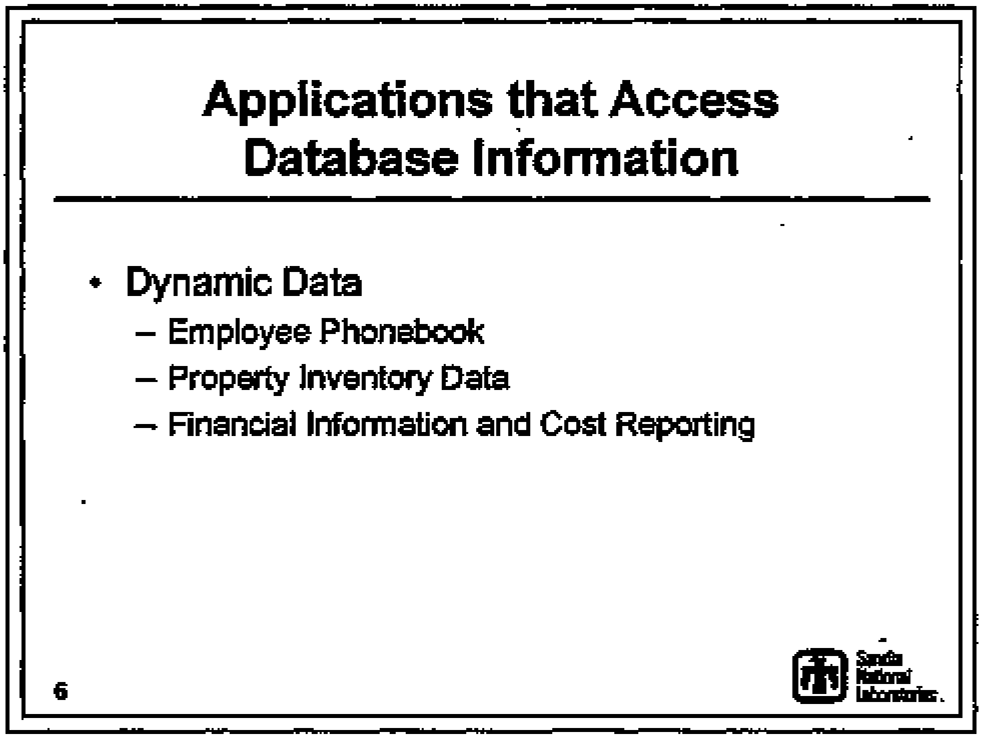
4

Sandia National Laboratories



What kind of information can be viewed?

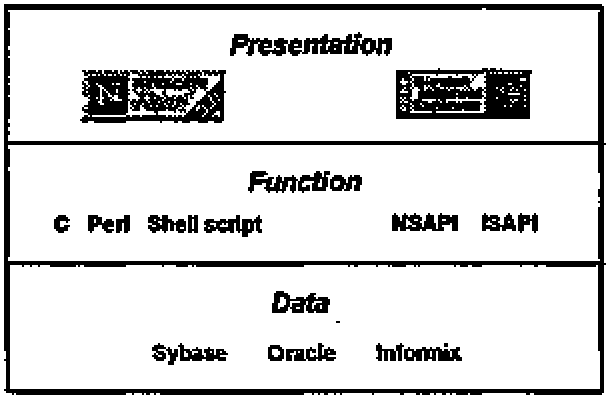
- Static Stuff
  - Periodicals, Bulletins, Newsletters
  - Manuals
  - Corporate Policy and Procedures



Applications that Access Database Information

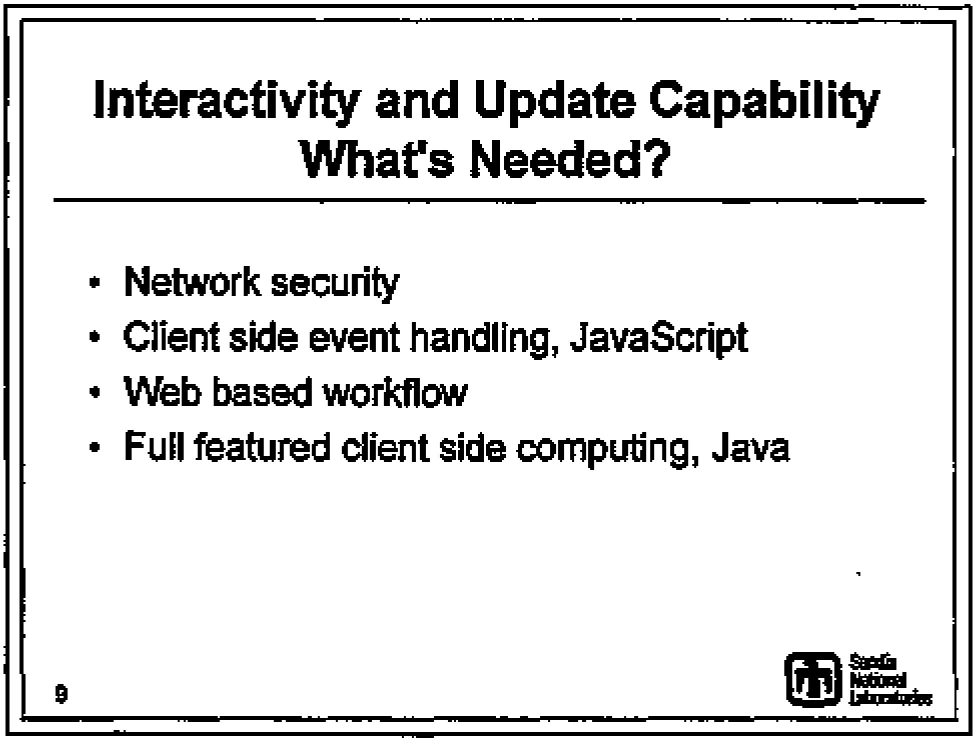
- Dynamic Data
  - Employee Phonebook
  - Property Inventory Data
  - Financial Information and Cost Reporting

Three Tiered Client/Server Architecture



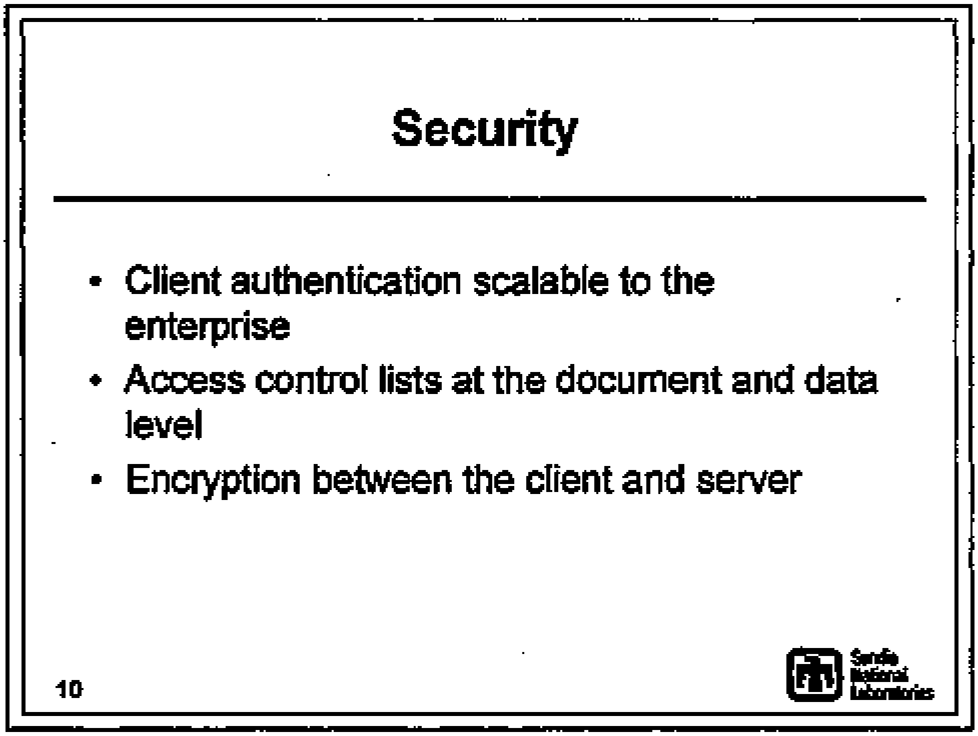
Applications that Update Information

- Interactive Interfaces
  - Conference Room Scheduler
  - Employee Timecard
  - Electronic Purchasing Requisition



Interactivity and Update Capability What's Needed?

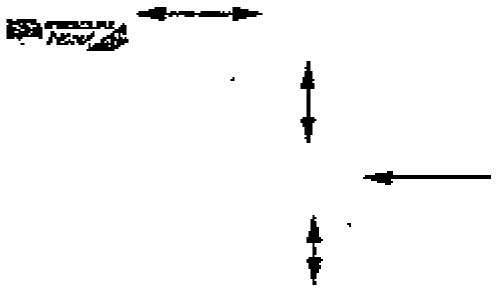
- Network security
- Client side event handling, JavaScript
- Web based workflow
- Full featured client side computing, Java



Security

- Client authentication scalable to the enterprise
- Access control lists at the document and data level
- Encryption between the client and server

Web Workflow from Action Technologies, Metro



11

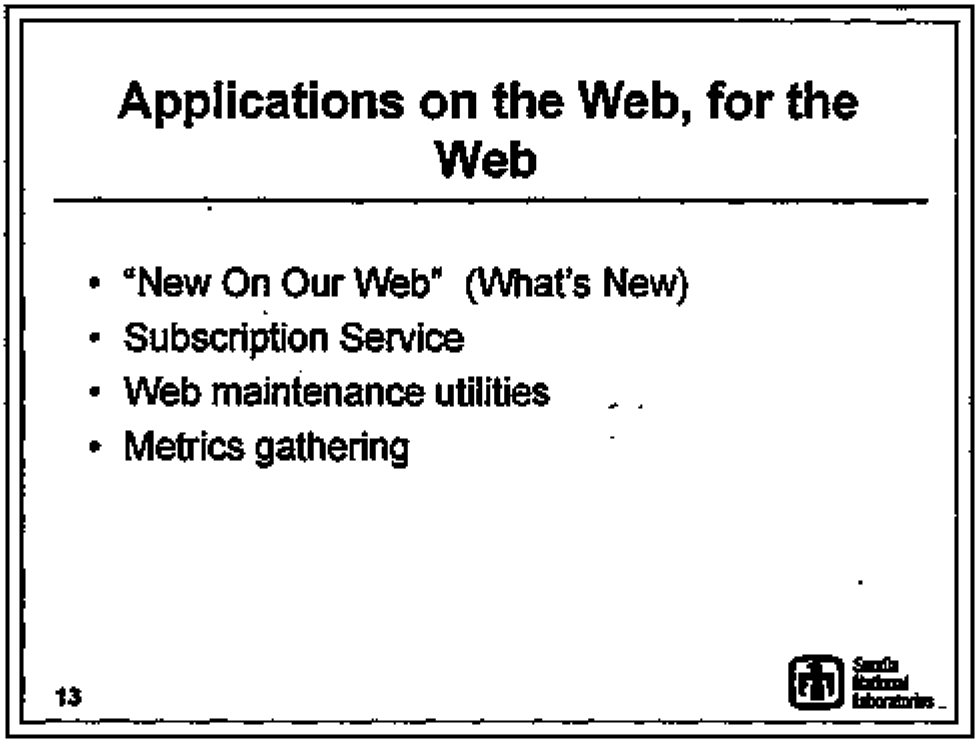
Sandia National Laboratories

Java Capabilities

- Semi-full featured programming language
- Write once, run anywhere, network-centric
- Offers socket level connections
- Security? It is getting better...

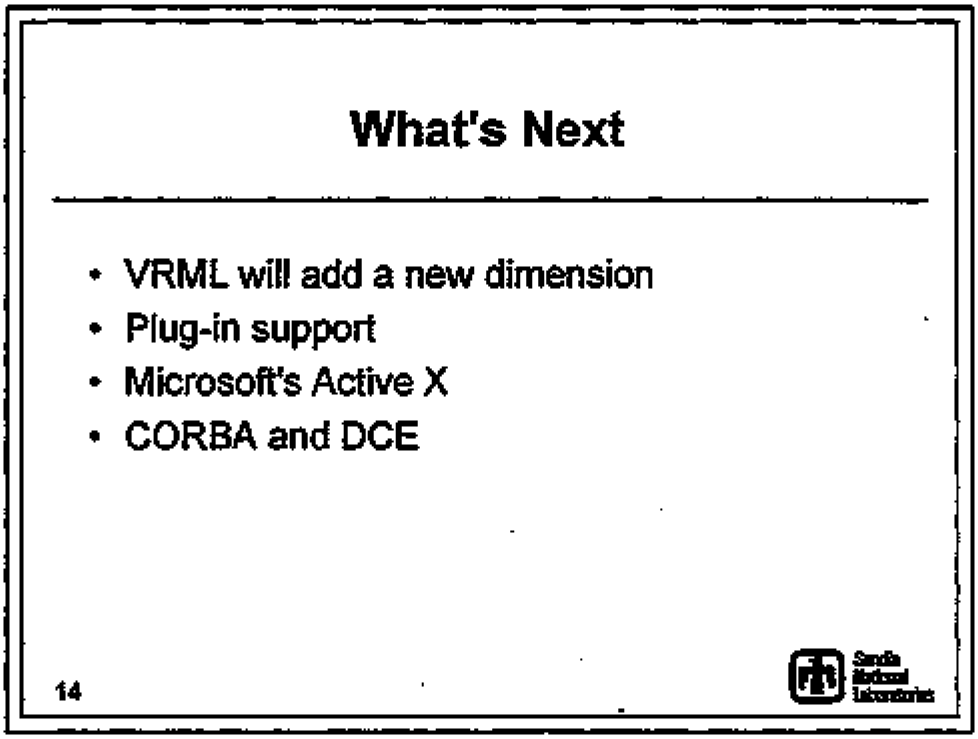
12

Sandia National Laboratories



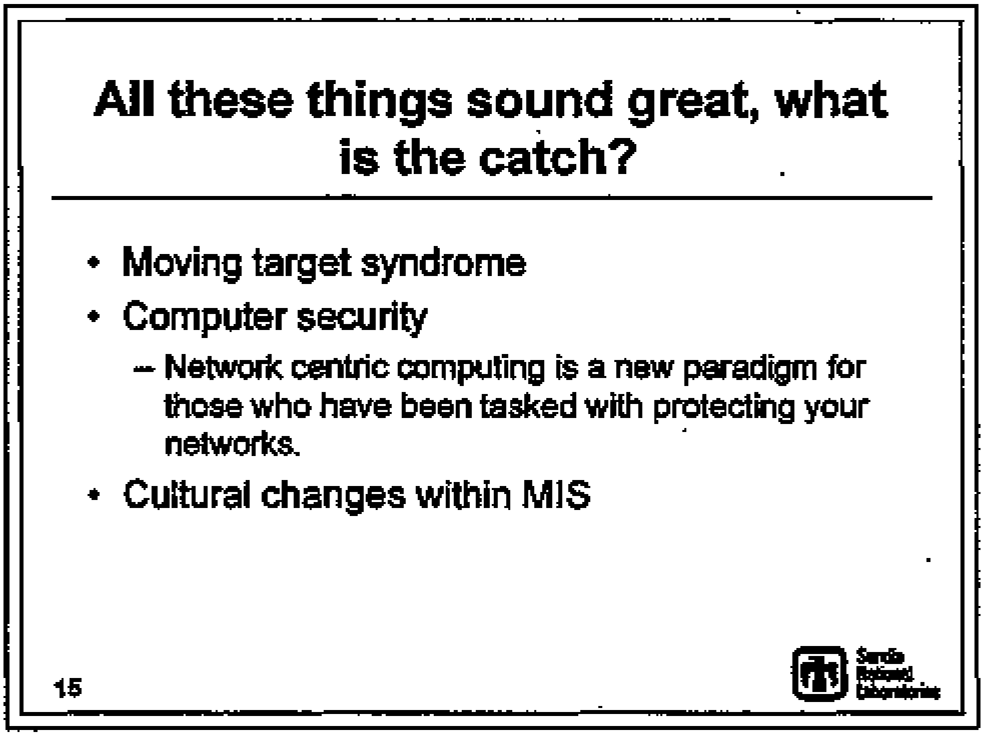
Applications on the Web, for the Web

- "New On Our Web" (What's New)
- Subscription Service
- Web maintenance utilities
- Metrics gathering



What's Next

- VRML will add a new dimension
- Plug-in support
- Microsoft's Active X
- CORBA and DCE

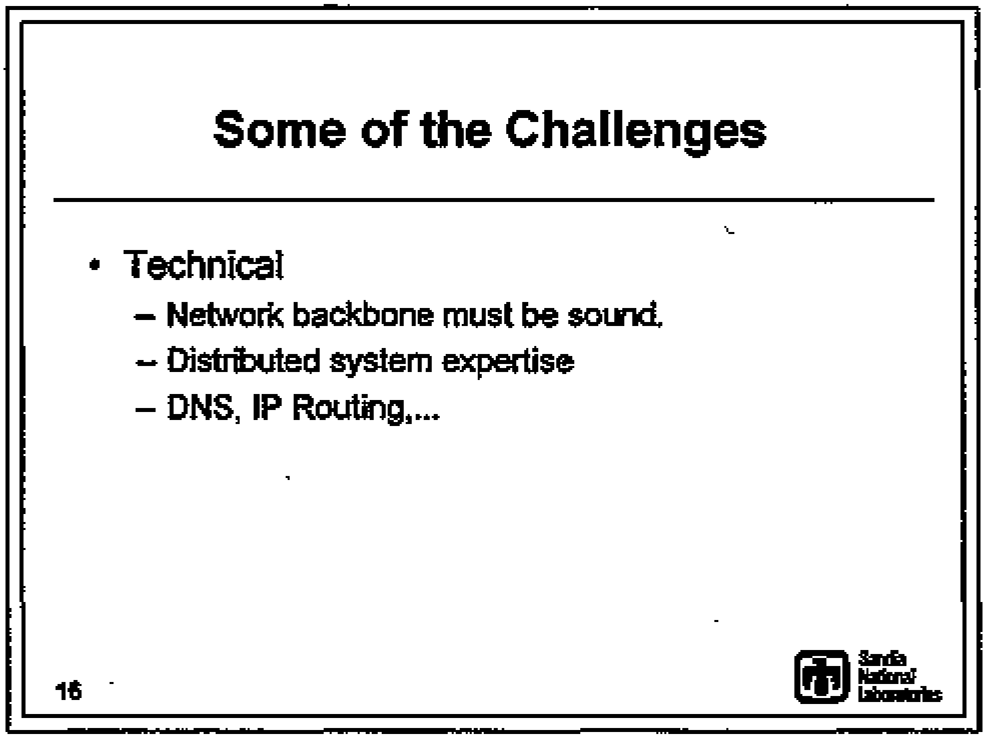


All these things sound great, what is the catch?

- Moving target syndrome
- Computer security
  - Network centric computing is a new paradigm for those who have been tasked with protecting your networks.
- Cultural changes within MIS

15

Sandia National Laboratories

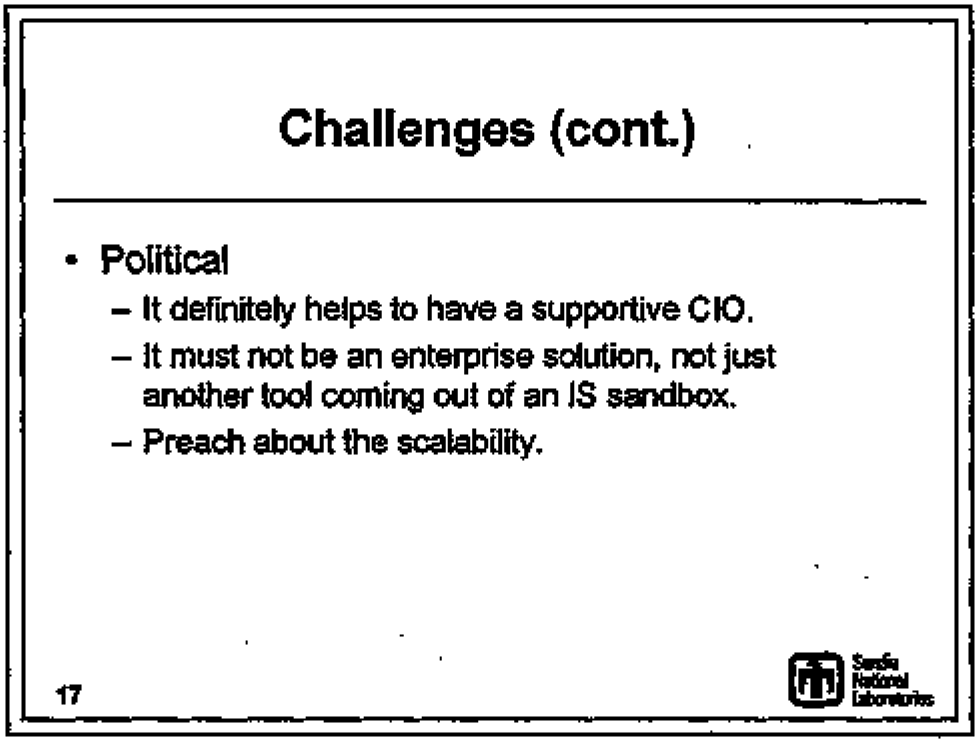


Some of the Challenges

- Technical
  - Network backbone must be sound.
  - Distributed system expertise
  - DNS, IP Routing,...

16

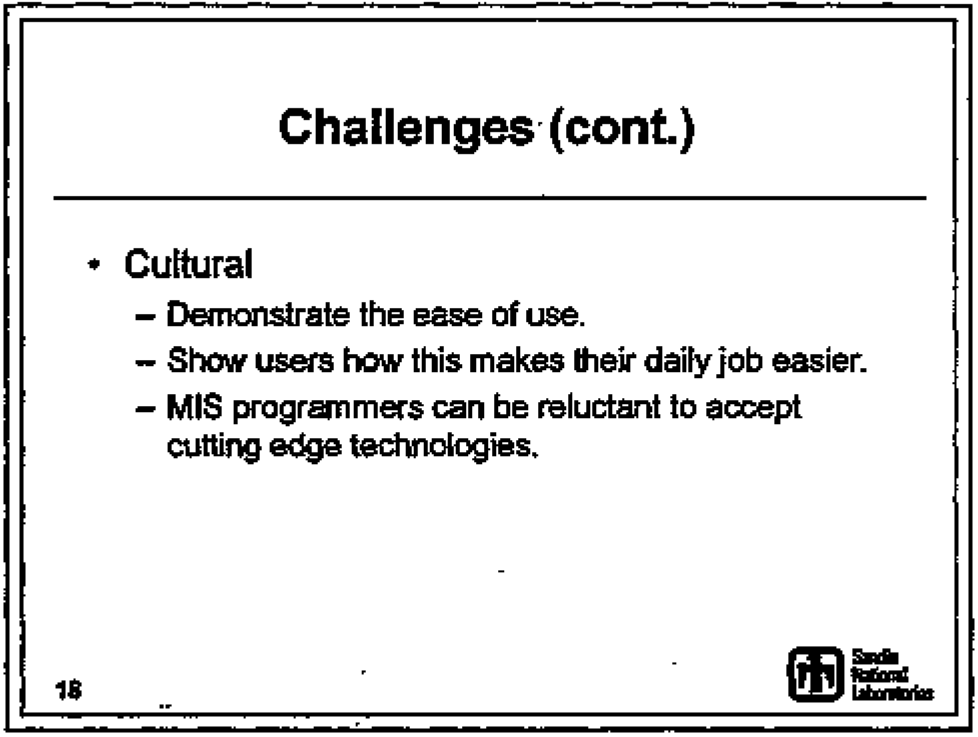
Sandia National Laboratories



Challenges (cont.)

- Political
  - It definitely helps to have a supportive CIO.
  - It must not be an enterprise solution, not just another tool coming out of an IS sandbox.
  - Preach about the scalability.

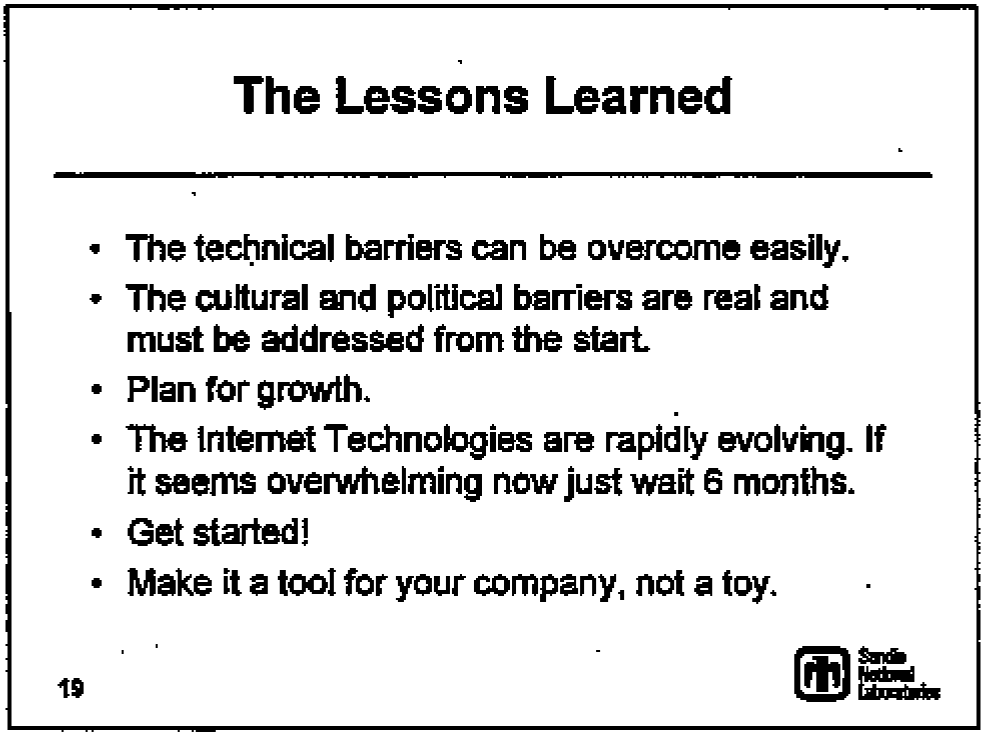
17



Challenges (cont.)

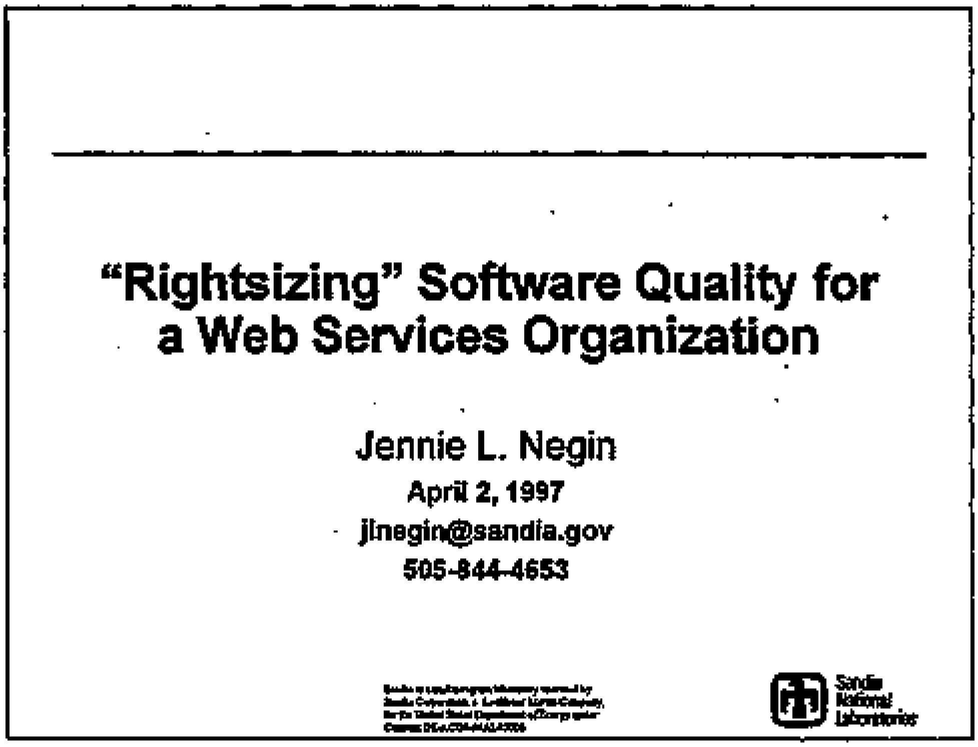
- Cultural
  - Demonstrate the ease of use.
  - Show users how this makes their daily job easier.
  - MIS programmers can be reluctant to accept cutting edge technologies.

18



The Lessons Learned

- The technical barriers can be overcome easily.
- The cultural and political barriers are real and must be addressed from the start.
- Plan for growth.
- The Internet Technologies are rapidly evolving. If it seems overwhelming now just wait 6 months.
- Get started!
- Make it a tool for your company, not a toy.

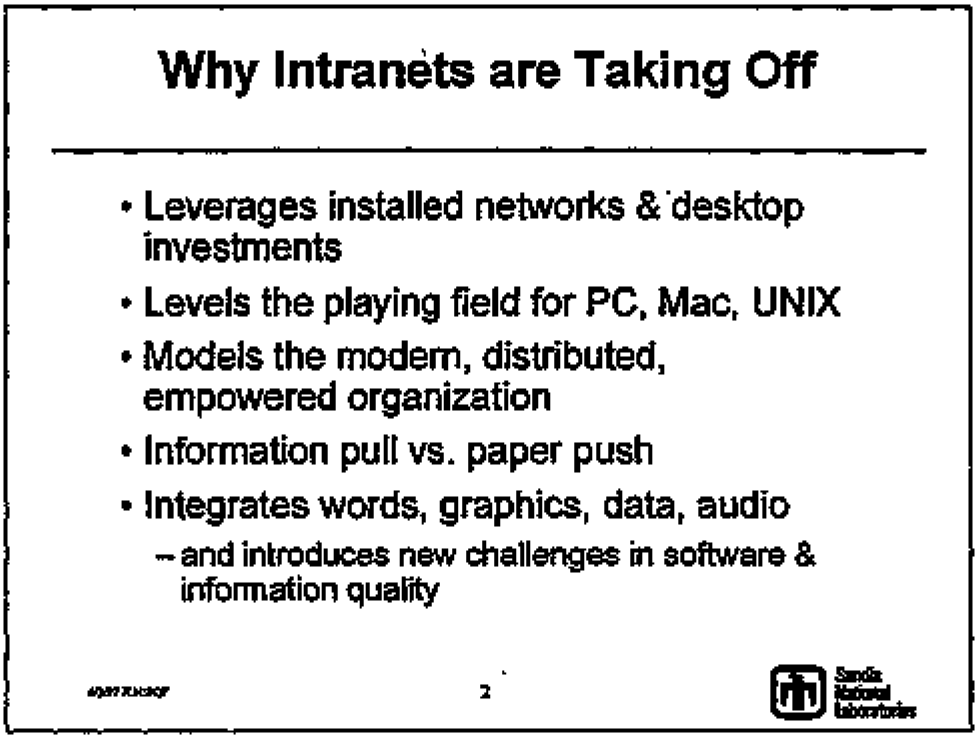


"Rightsizing" Software Quality for a Web Services Organization

Jennie L. Negin
April 2, 1997
jlnegin@sandia.gov
505-844-4653

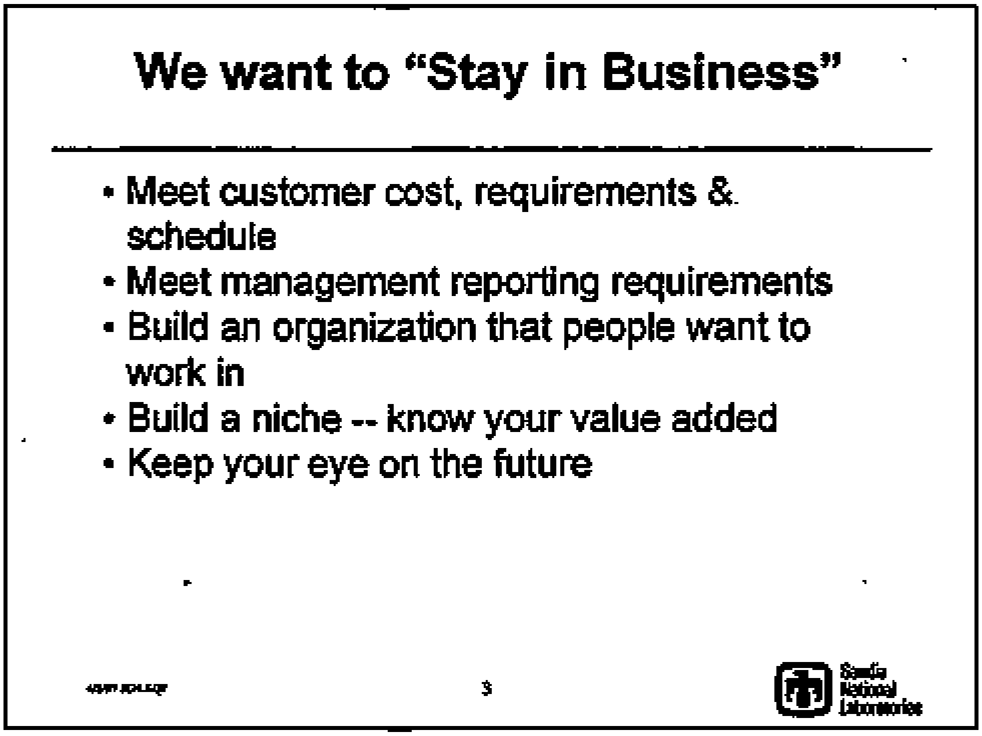
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000

Sandia National Laboratories



Why Intranets are Taking Off

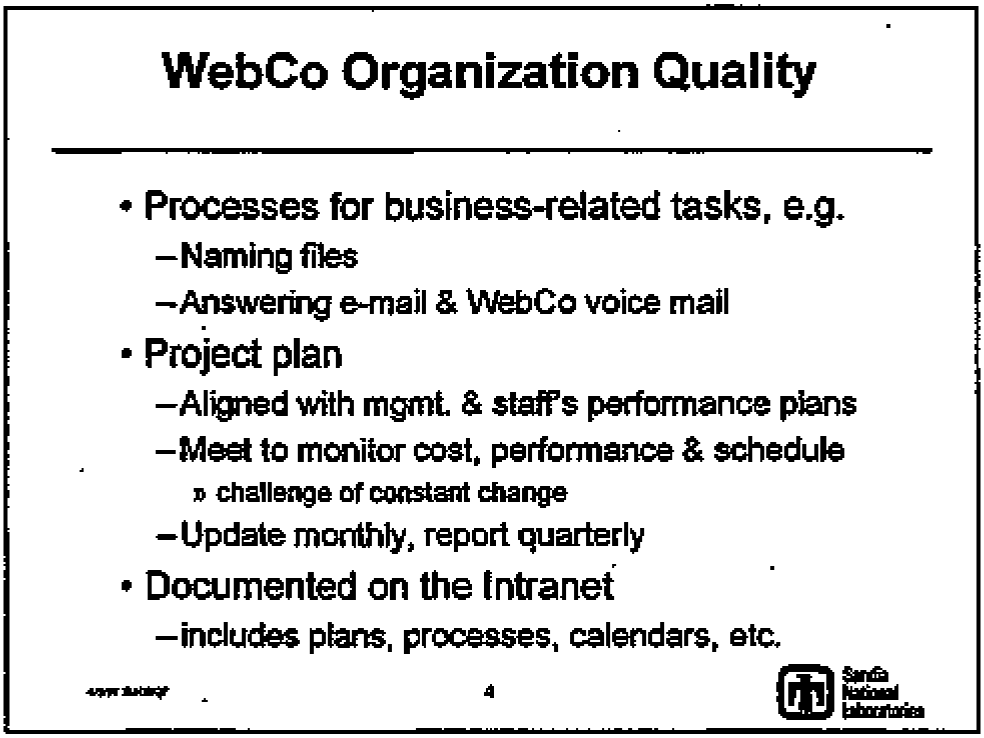
- Leverages installed networks & desktop investments
- Levels the playing field for PC, Mac, UNIX
- Models the modern, distributed, empowered organization
- Information pull vs. paper push
- Integrates words, graphics, data, audio
  - and introduces new challenges in software & information quality



We want to "Stay in Business"

- Meet customer cost, requirements & schedule
- Meet management reporting requirements
- Build an organization that people want to work in
- Build a niche -- know your value added
- Keep your eye on the future

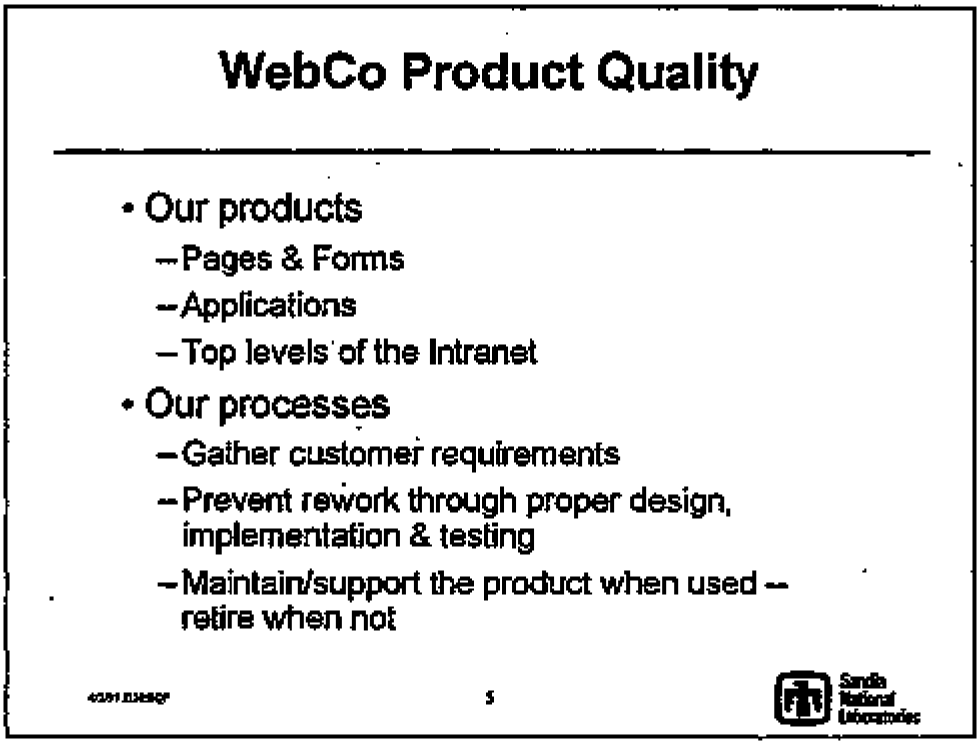
Sandia National Laboratories



WebCo Organization Quality

- Processes for business-related tasks, e.g.
  - Naming files
  - Answering e-mail & WebCo voice mail
- Project plan
  - Aligned with mgmt. & staff's performance plans
  - Meet to monitor cost, performance & schedule
    - challenge of constant change
  - Update monthly, report quarterly
- Documented on the Intranet
  - includes plans, processes, calendars, etc.

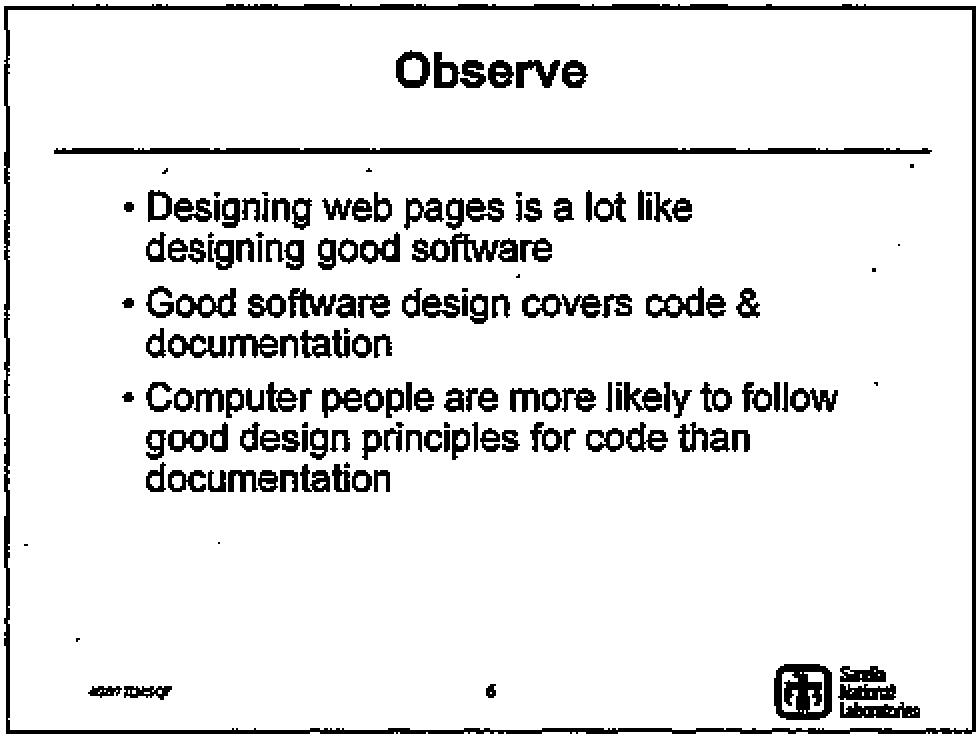
Sandia National Laboratories



WebCo Product Quality

- Our products
  - Pages & Forms
  - Applications
  - Top levels of the Intranet
- Our processes
  - Gather customer requirements
  - Prevent rework through proper design, implementation & testing
  - Maintain/support the product when used -- retire when not

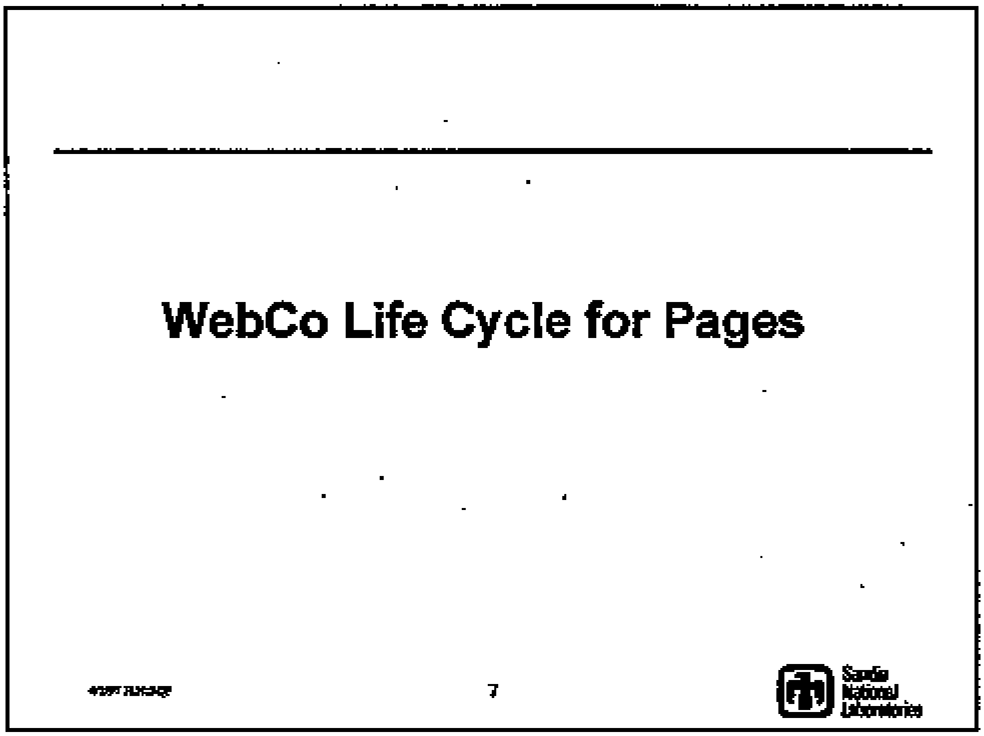
Sandia National Laboratories



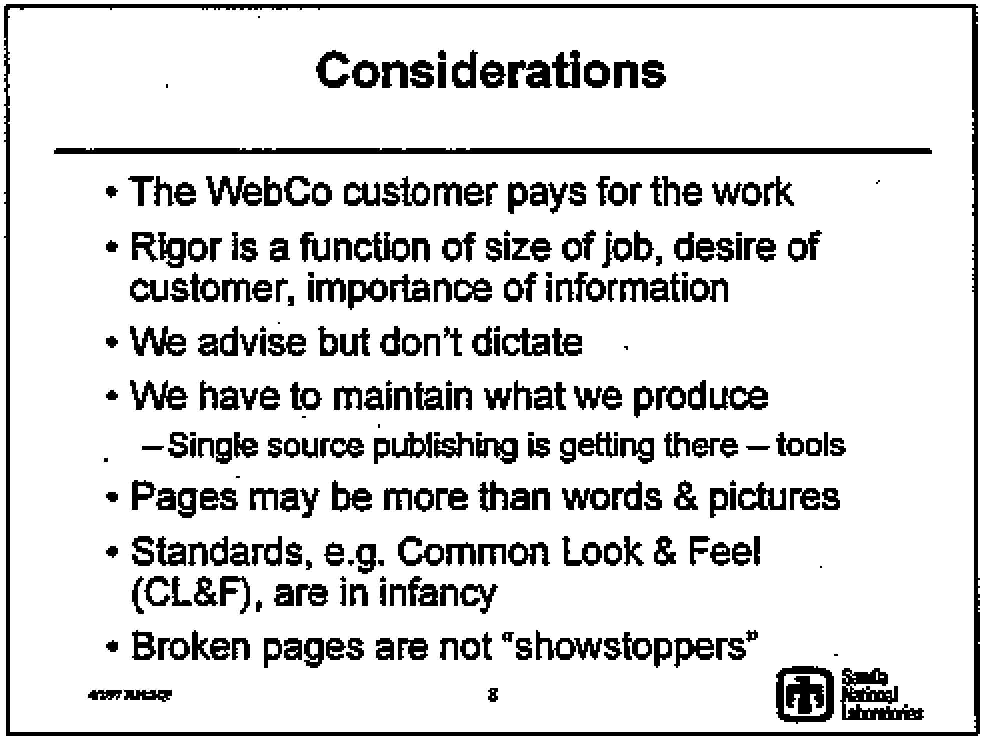
Observe

- Designing web pages is a lot like designing good software
- Good software design covers code & documentation
- Computer people are more likely to follow good design principles for code than documentation

Sandia National Laboratories

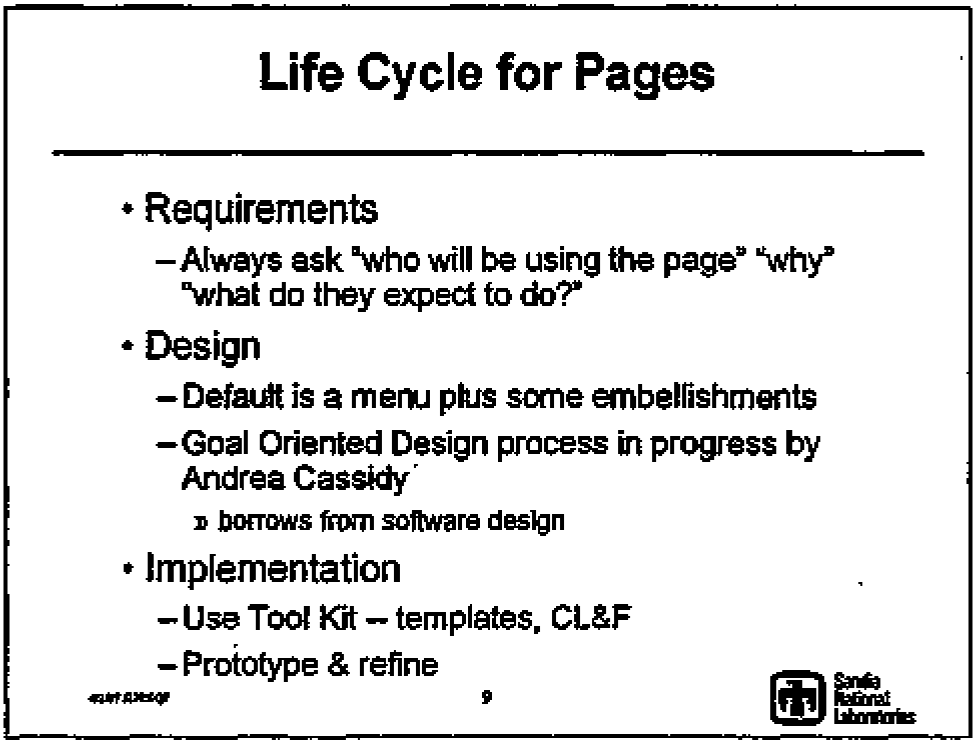


WebCo Life Cycle for Pages



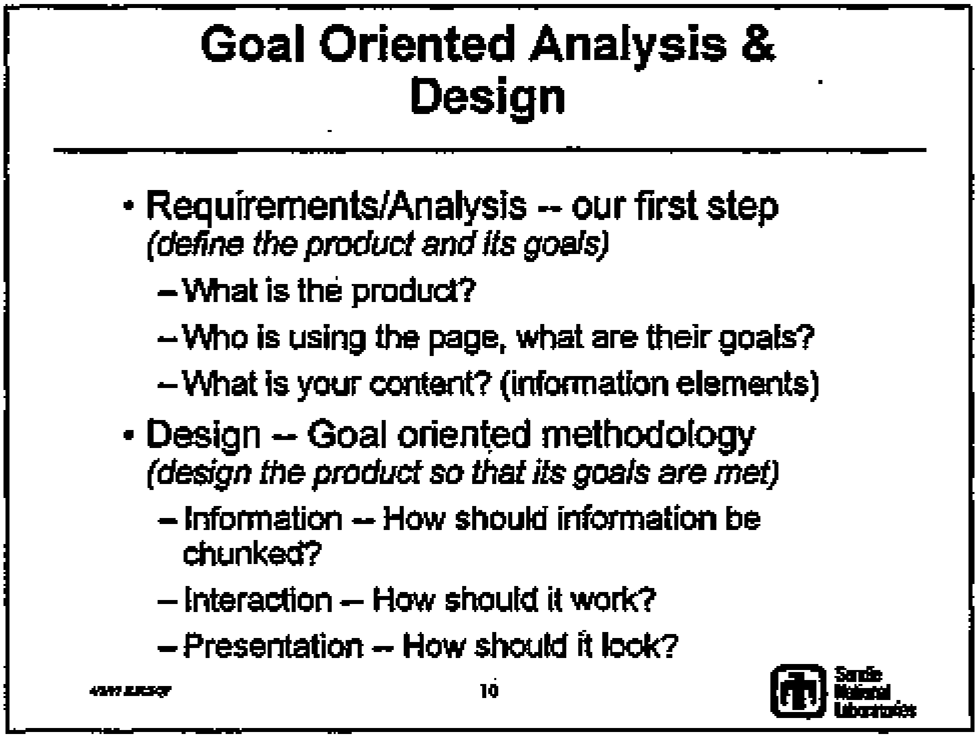
Considerations

- The WebCo customer pays for the work
- Rigor is a function of size of job, desire of customer, importance of information
- We advise but don't dictate
- We have to maintain what we produce
  - Single source publishing is getting there – tools
- Pages may be more than words & pictures
- Standards, e.g. Common Look & Feel (CL&F), are in infancy
- Broken pages are not "showstoppers"



Life Cycle for Pages

- Requirements
  - Always ask "who will be using the page" "why" "what do they expect to do?"
- Design
  - Default is a menu plus some embellishments
  - Goal Oriented Design process in progress by Andrea Cassidy
    - » borrows from software design
- Implementation
  - Use Tool Kit -- templates, CL&F
  - Prototype & refine



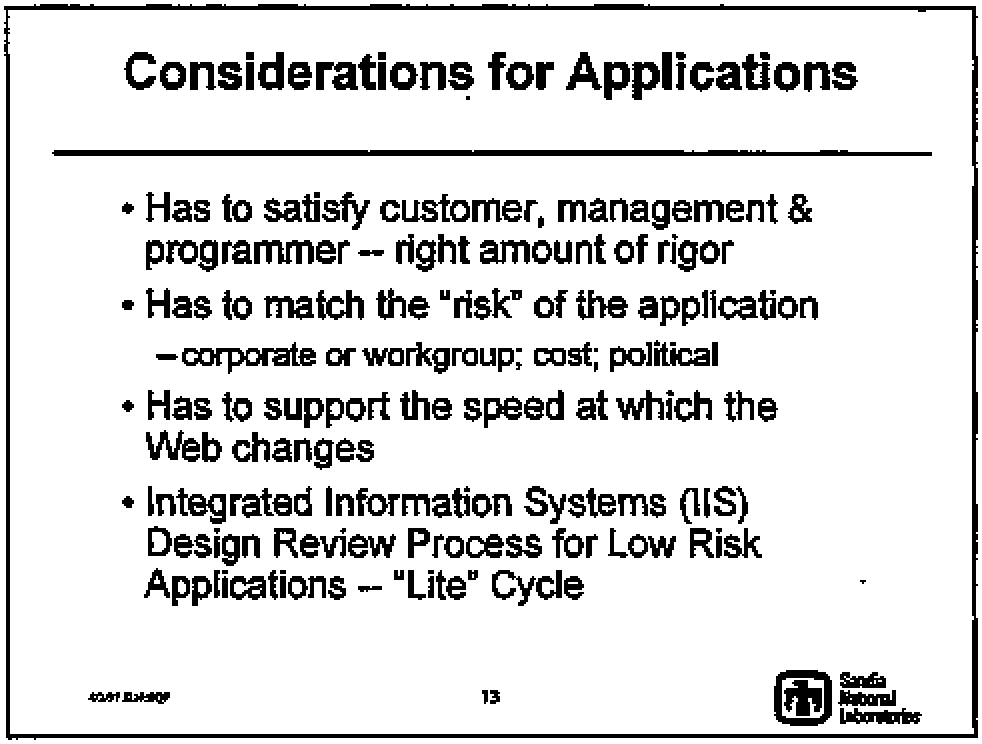
Goal Oriented Analysis & Design

- Requirements/Analysis -- our first step *(define the product and its goals)*
  - What is the product?
  - Who is using the page, what are their goals?
  - What is your content? (information elements)
- Design -- Goal oriented methodology *(design the product so that its goals are met)*
  - Information -- How should information be chunked?
  - Interaction -- How should it work?
  - Presentation -- How should it look?

Life Cycle for Pages - cont.

- Test – does it meet the customer's goals
  - Usability
  - Navigation -- tests interaction -- role of tools
- Production
  - FTP to server
  - Processes for maintenance
- Support
  - FTP from server
  - Date changed pages
  - Configuration Management is in infancy

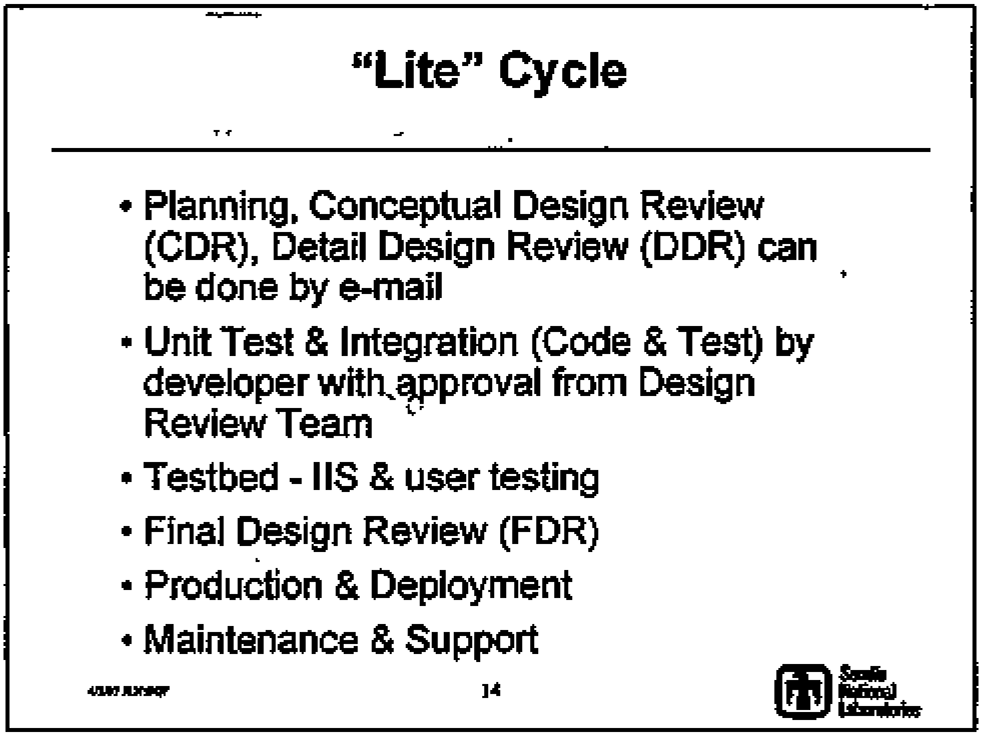
WebCo Life Cycle for Applications



Considerations for Applications

- Has to satisfy customer, management & programmer -- right amount of rigor
- Has to match the "risk" of the application
  - corporate or workgroup; cost; political
- Has to support the speed at which the Web changes
- Integrated Information Systems (IIS) Design Review Process for Low Risk Applications -- "Lite" Cycle

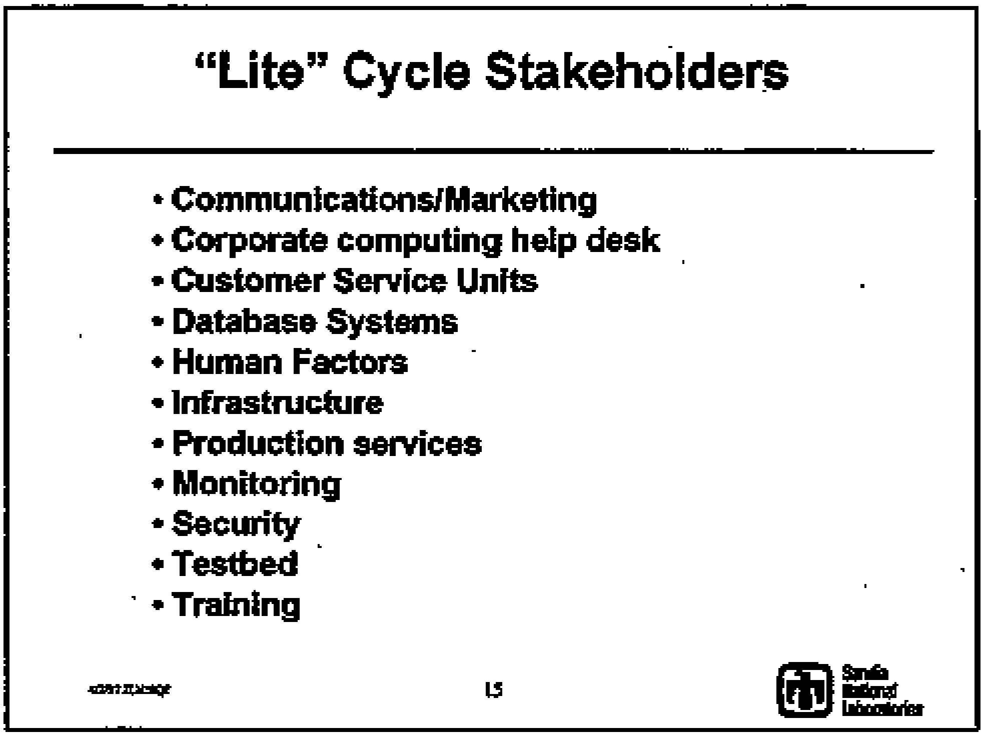
[illegible] 13 Sandia National Laboratories



"Lite" Cycle

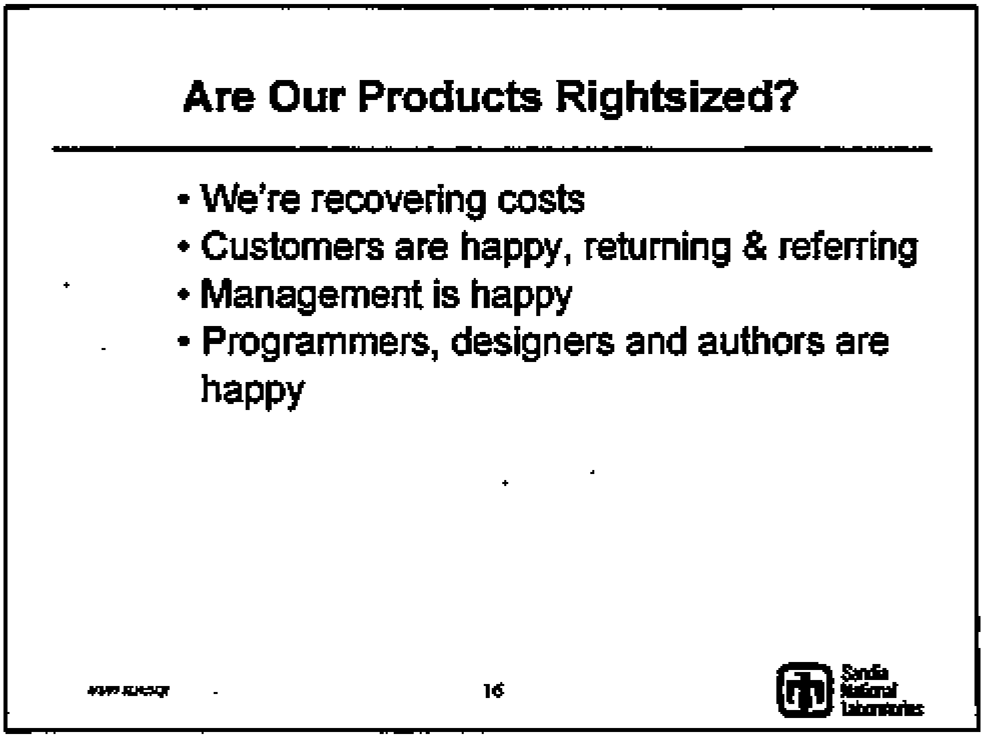
- Planning, Conceptual Design Review (CDR), Detail Design Review (DDR) can be done by e-mail
- Unit Test & Integration (Code & Test) by developer with approval from Design Review Team
- Testbed - IIS & user testing
- Final Design Review (FDR)
- Production & Deployment
- Maintenance & Support

[illegible] 14 Sandia National Laboratories



“Lite” Cycle Stakeholders

- Communications/Marketing
- Corporate computing help desk
- Customer Service Units
- Database Systems
- Human Factors
- Infrastructure
- Production services
- Monitoring
- Security
- Testbed
- Training



Are Our Products Rightsized?

- We’re recovering costs
- Customers are happy, returning & referring
- Management is happy
- Programmers, designers and authors are happy

1947 • SOFTWARE QUALITY FORUM • 1997
DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX
SOFTWARE ENGINEERING QUALITY

Session A3: Software Process Improvement I

Chair Mike Lackner
AS/FM&T

Session : Paper #	Author(s)	*Title*
A3:1	Don Schilling AS/FM&T	*Quest for Excellence 1996: Reaching for the Stars*
A3:2	Don Rathbun AS/FM&T	*Command Media System at the Kansas City Plant (KCP)*
A3:3	Michael Tiemann Headquarters Department of Energy	*Departmental Information Architecture*

Quest for Excellence 1996

Donald Schilling

AlliedSignal Federal Manufacturing & Technologies/KC*

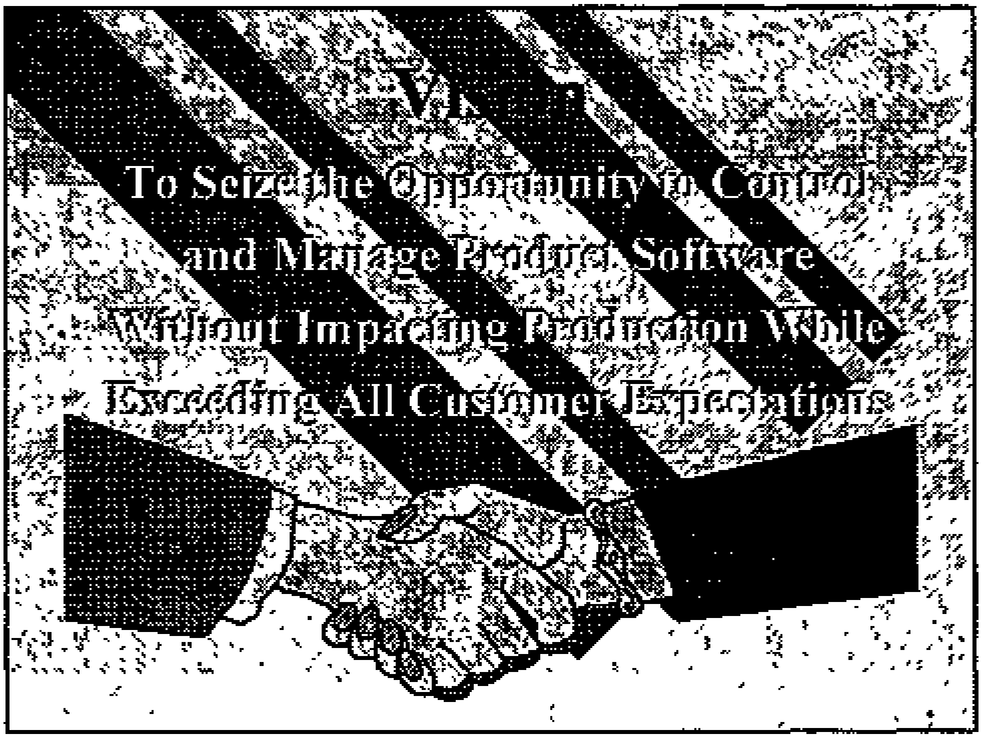
Reaching For the Stars

*Operated for the United States Department of Energy under Contract No. DE-AC04-76-DP00613.

Handling Mark Quality Software As Product

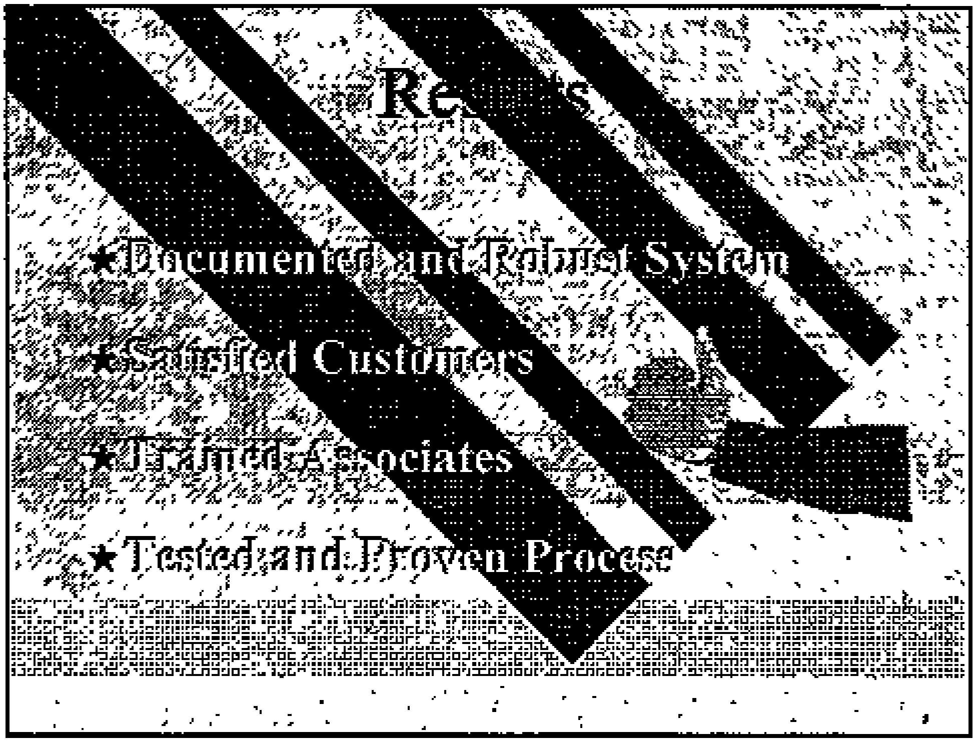
TQ Team [illegible]009

Opportunity



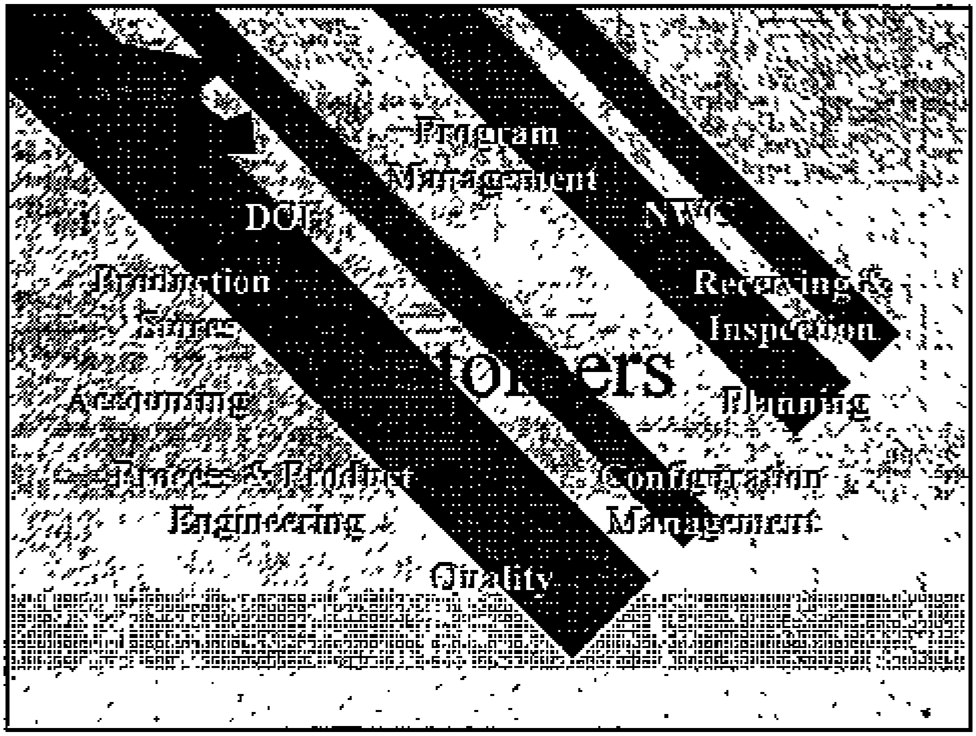
Vision

To Seize the Opportunity to Control and Manage Product Software Without Impacting Production While Exceeding All Customer Expectations



Results

- ★ Documented and Robust System
- ★ Satisfied Customers
- ★ Trained Associates
- ★ Tested and Proven Process



Customers

Program Management

DCE

NWC

Production Stores

Receiving & Inspection

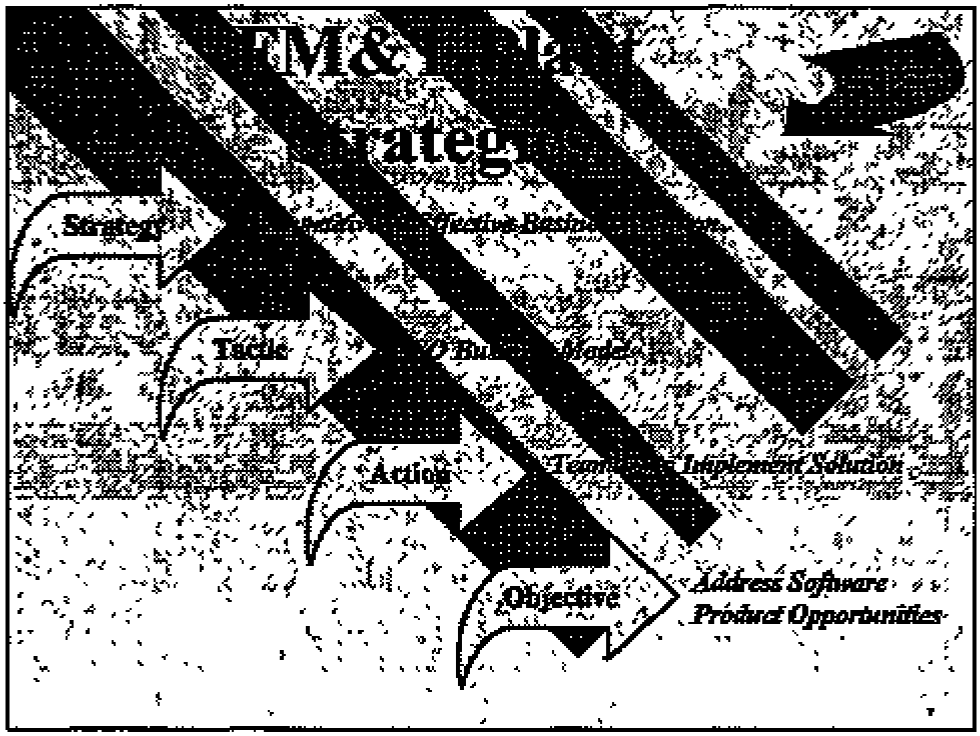
Accounting

Planning

Process & Product Engineering

Configuration Management

Quality



TM& [illegible]

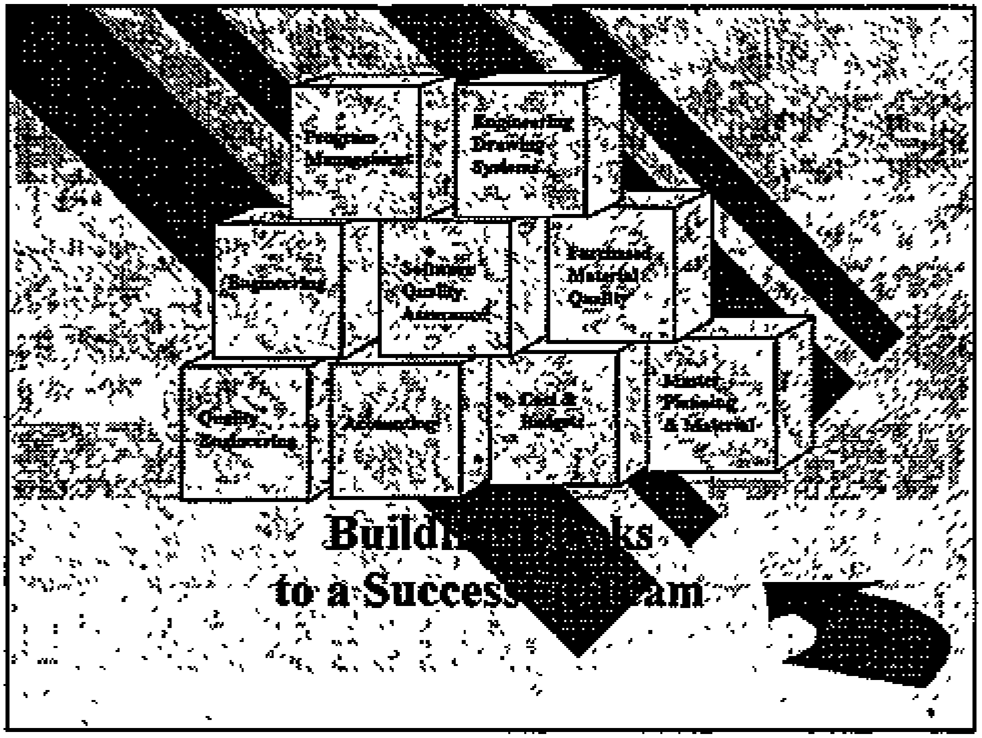
[illegible]rateg[illegible]

Strategy [illegible] Objective Business [illegible]

Tactic [illegible] Business Model

Action Team [illegible] Implement Solution

Objective Address Software Product Opportunities



Program Management

Engineering Drawing Systems

Engineering

Software Quality Assurance

Purchased Material Quality

Quality Engineering

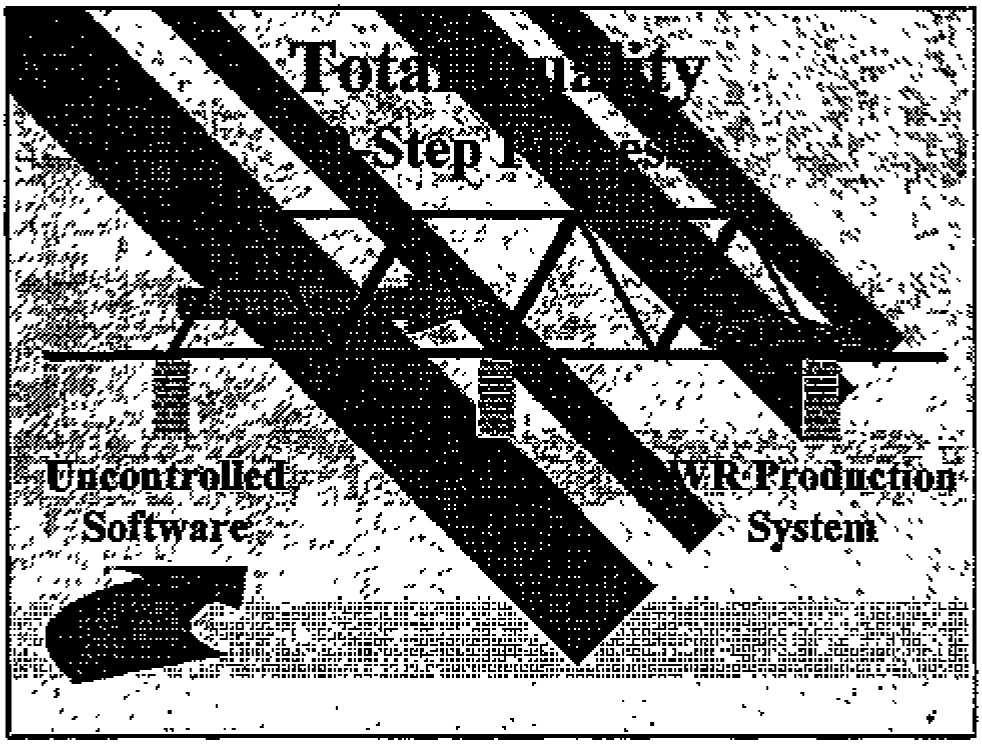
Accounting

Cost & Budgets

Master Planning & Material

Buildi[illegible]ks

to a Success[illegible]am

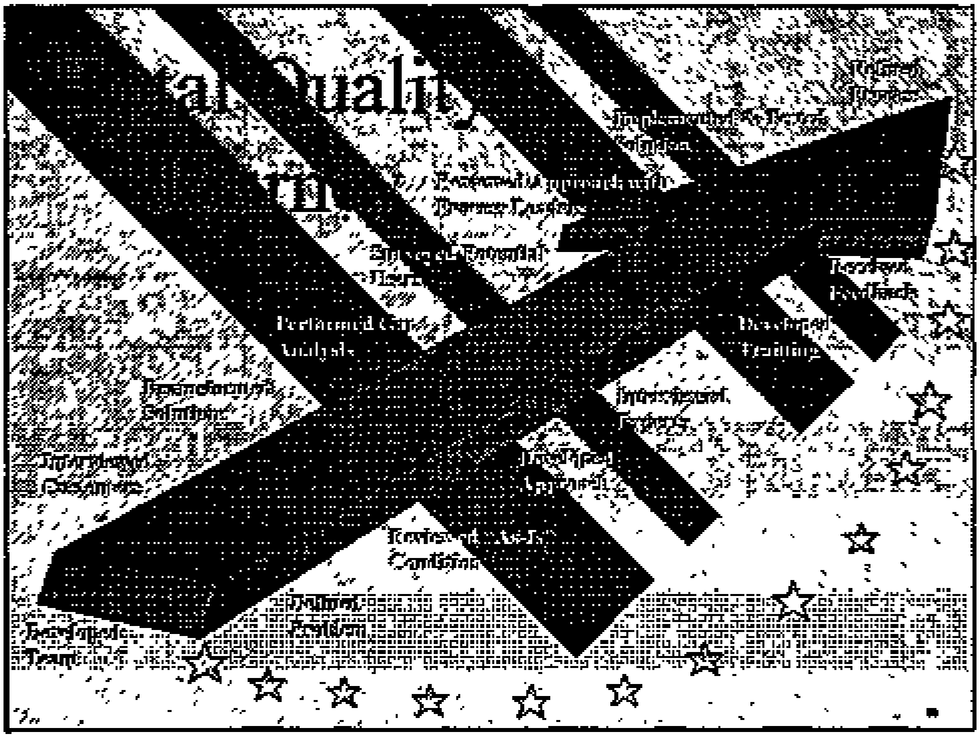


Total Quality

[illegible]-Step Process

Uncontrolled Software

[illegible]WR Production System



[illegible]al Qualit[illegible]

Developed Team

Defined Problem

Reviewed As-Is Condition

Interviewed Customers

Brainstormed Solutions

Performed Gap Analysis

Surveyed Potential Users

Reviewed Approach with Process Leaders

Developed Approach

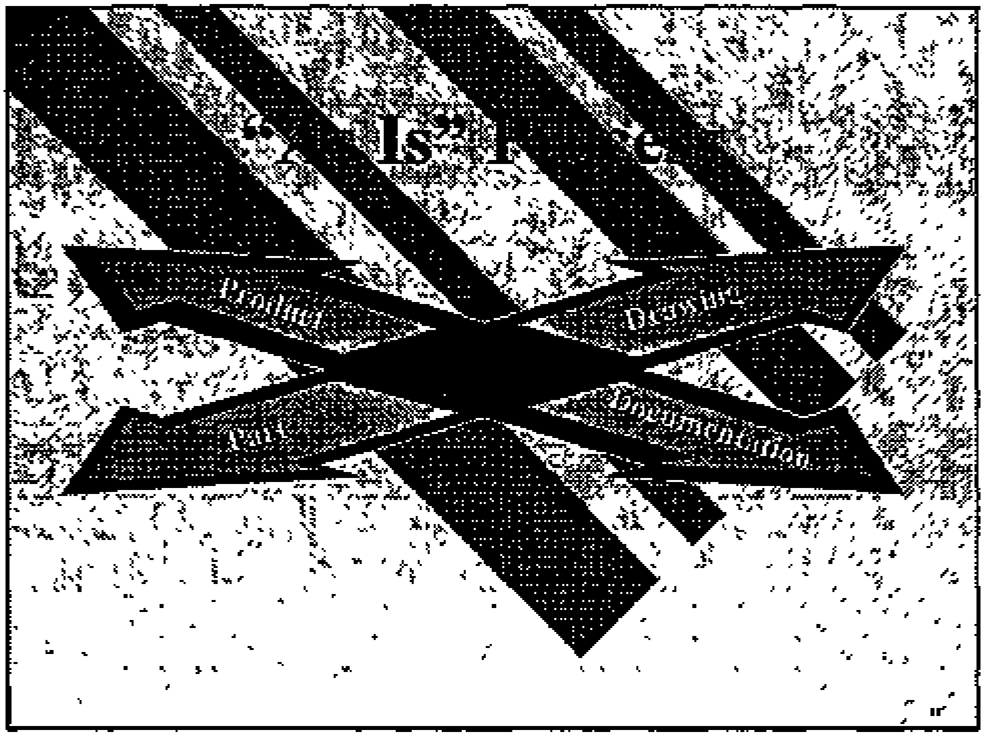
Implemented Reports

Developed Training

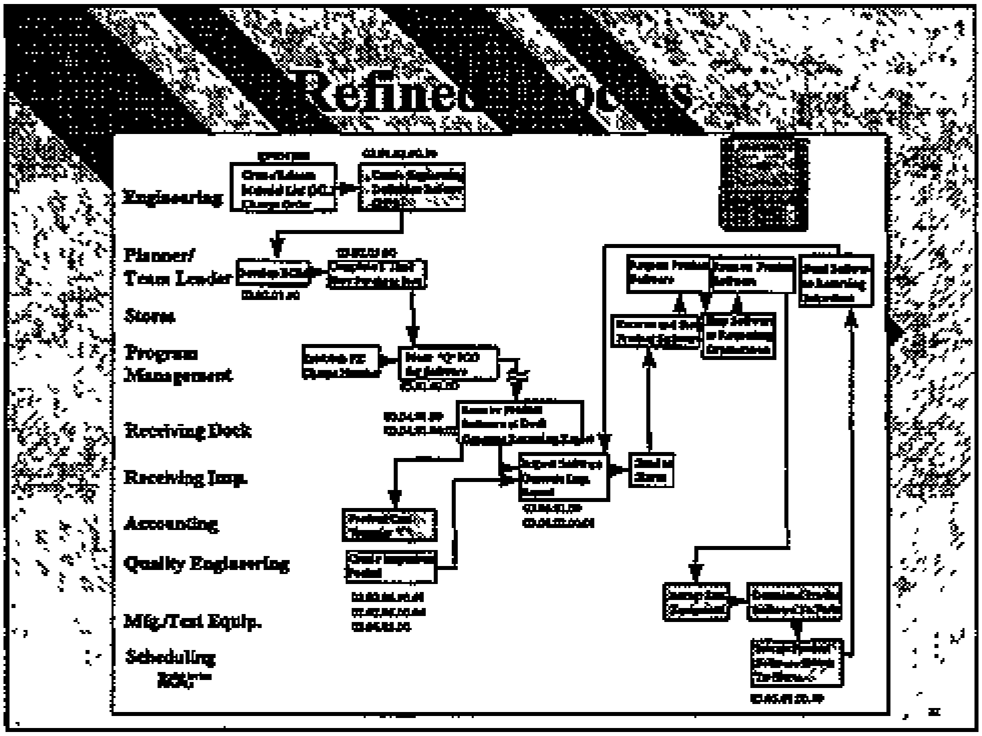
Received Feedback

Implemented Selected Solution

Refined Process

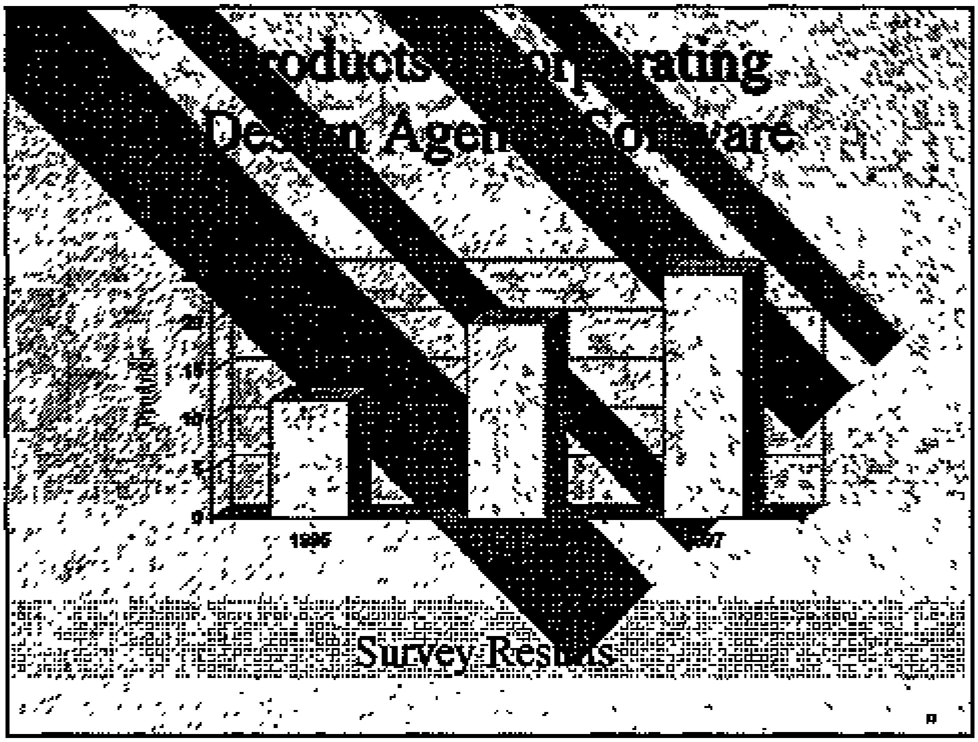


“As Is” Process



Refined Process

SQAS Forum April 1-3, 1997

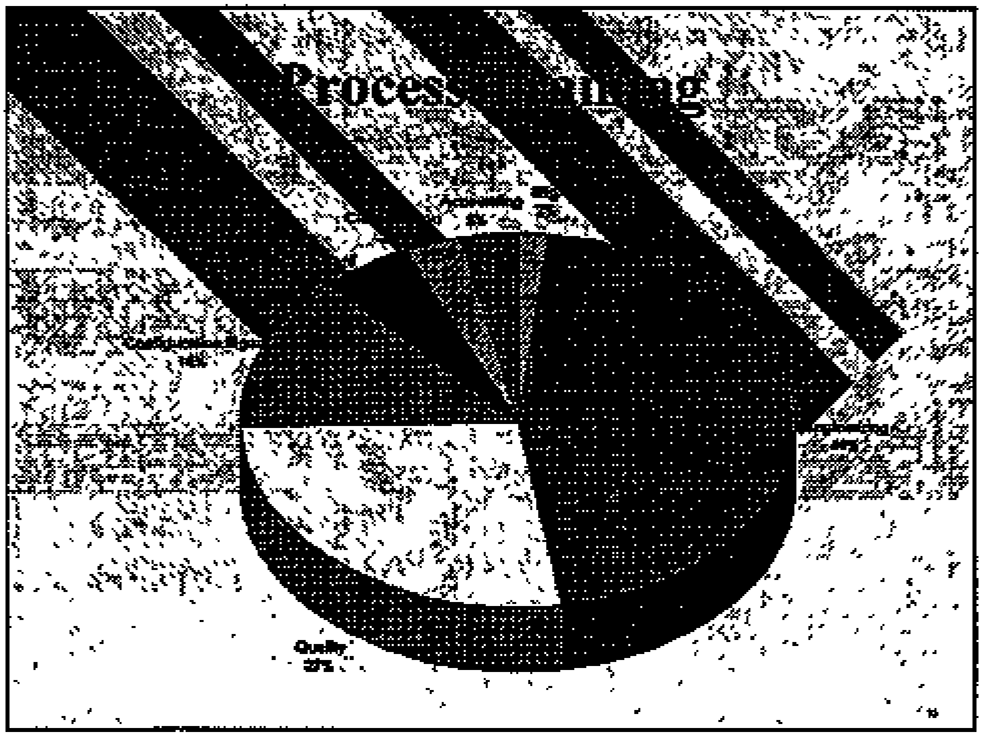


Products Incorporating Design Agency Software

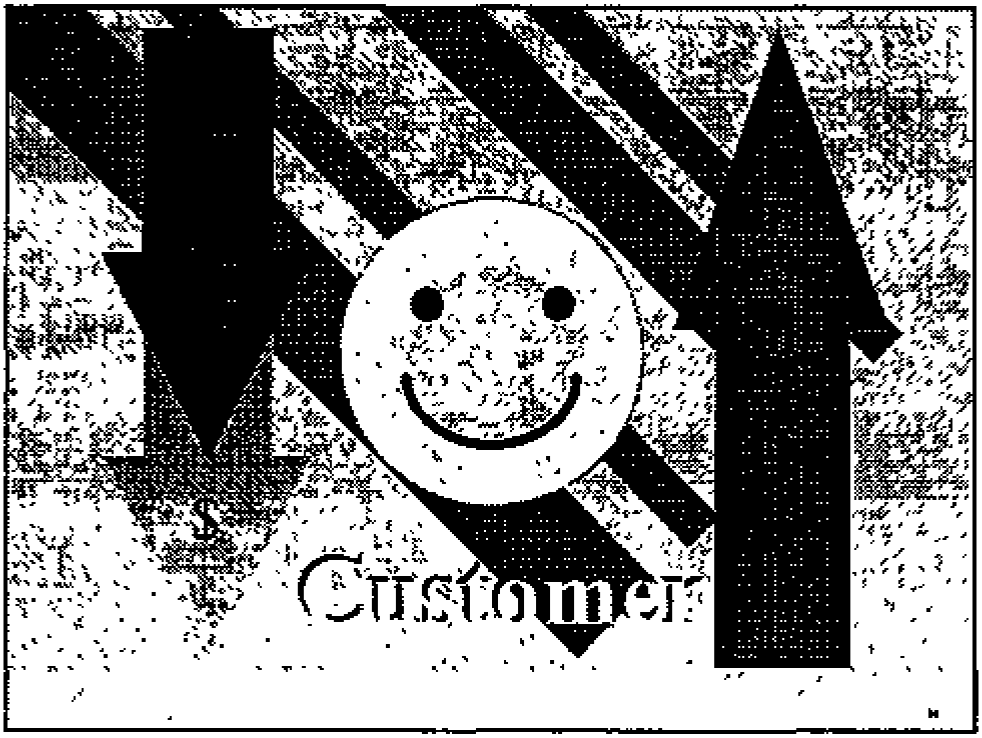
Survey Results

Gap Analysis

- ★ Need to Address Non-Consumable Product (Work Instruction 02.01.07.00.10)
- ★ Need for Process for Returning Product to Stores (Work Instruction 05.05.01.00.30)
- ★ Requirement for Product Specific SS Identified for Software
- ★ Manual Process for Amortizing Cost Highlighted.



Process Training



Customer

Command Media System at the Kansas City Plant (KCP)

Don A. Rathbun, Staff Engineer

AlliedSignal Federal Manufacturing & Technologies (FM&T)*

Presented at the 1997 Software Quality Forum, April 1-3, 1997
Kirtland Air Force Base, Albuquerque, New Mexico
Sponsored by
Department of Energy (DOE) Quality Managers
Software Quality Assurance Subcommittee of the DOE Quality Managers
Weapons Quality Division, DOE-Albuquerque Office

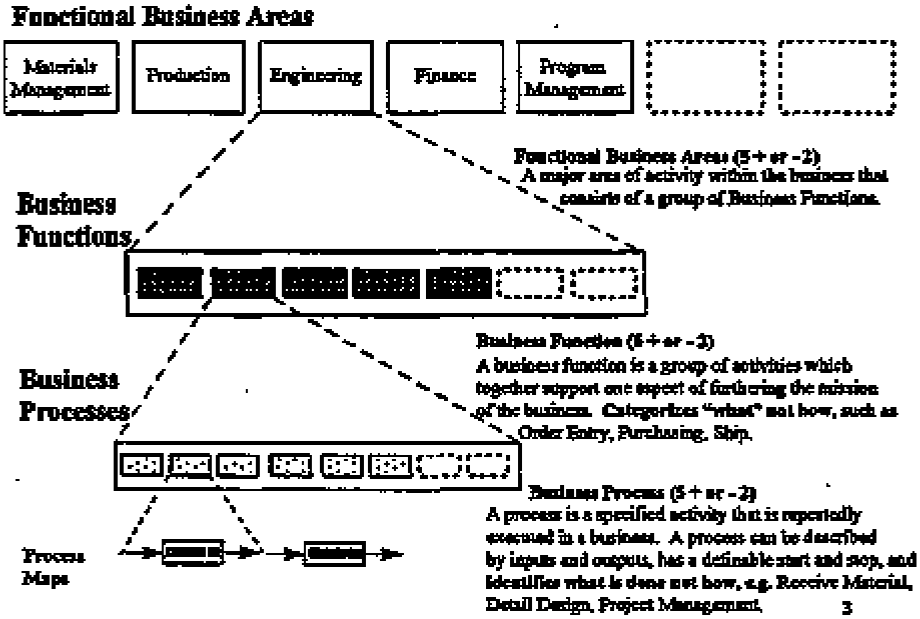
*Operated for United States Department of Energy under Contract No. DE-AC04-76-DP00613

ISO 9001 Preassessment 7/94

- Total Findings by third party auditor - 60, plus 21 Observations.
  - Document & Data Control Findings - 29

    (Business process findings, not findings against work instructions to the factory floor)
    - Corrective action taken:
      - New Command Media System implemented to replace the paper document system that was in place at the time of the preassessment. Implementation was started 9/94.
      - Training on new Command Media System.

AS FM&T Business Model Development



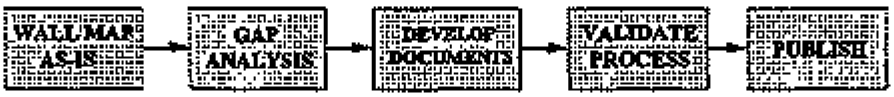
Functional Business Areas (5 + or - 2)
A major area of activity within the business that consists of a group of Business Functions.

Business Function (5 + or - 2)
A business function is a group of activities which together support one aspect of furthering the mission of the business. Categorizes "what" not how, such as Order Entry, Purchasing, Ship.

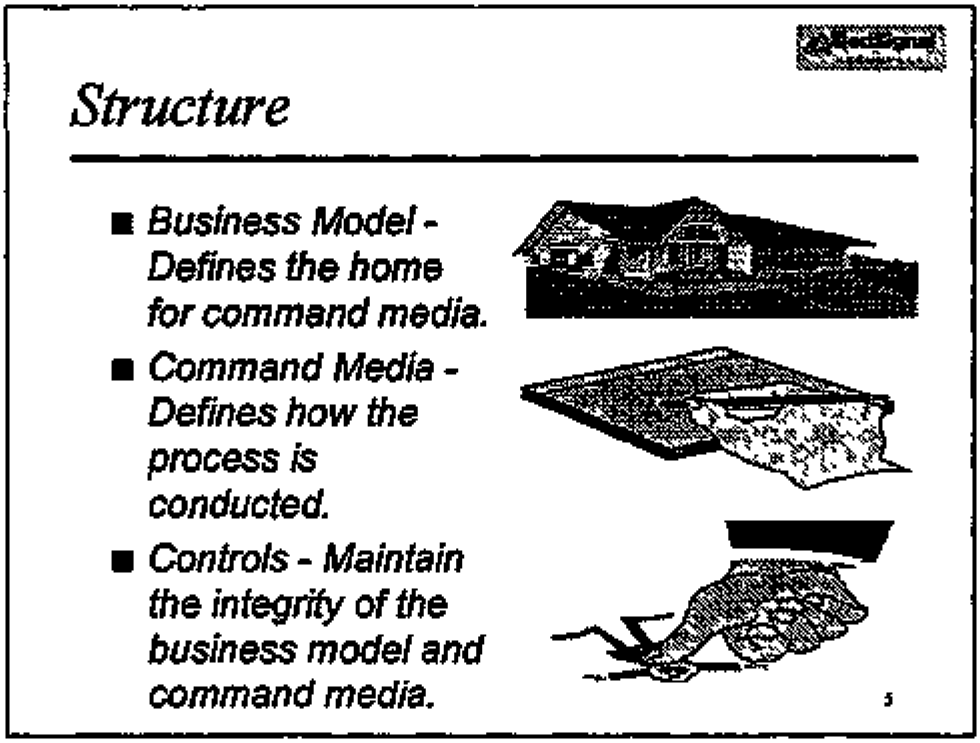
Business Process (5 + or - 2)
A process is a specified activity that is repeatedly executed in a business. A process can be described by inputs and outputs, has a definable start and stop, and identifies what is done not how, e.g. Receive Material, Detail Design, Project Management.

3

COMMAND MEDIA PROCESS

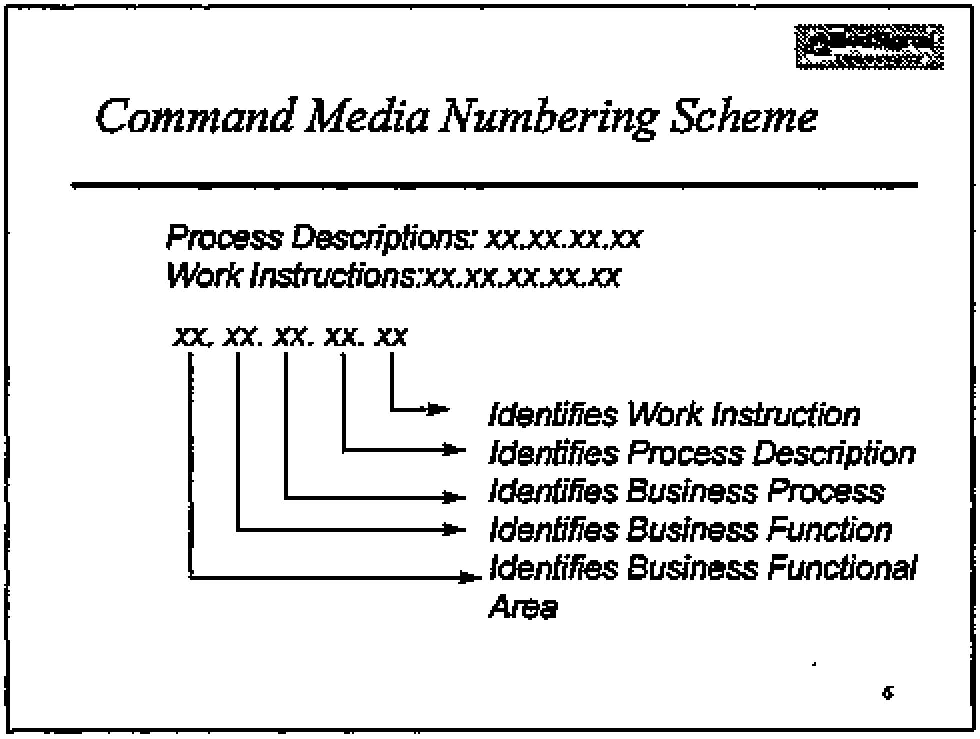


4



Structure

- Business Model - Defines the home for command media.
- Command Media - Defines how the process is conducted.
- Controls - Maintain the integrity of the business model and command media.

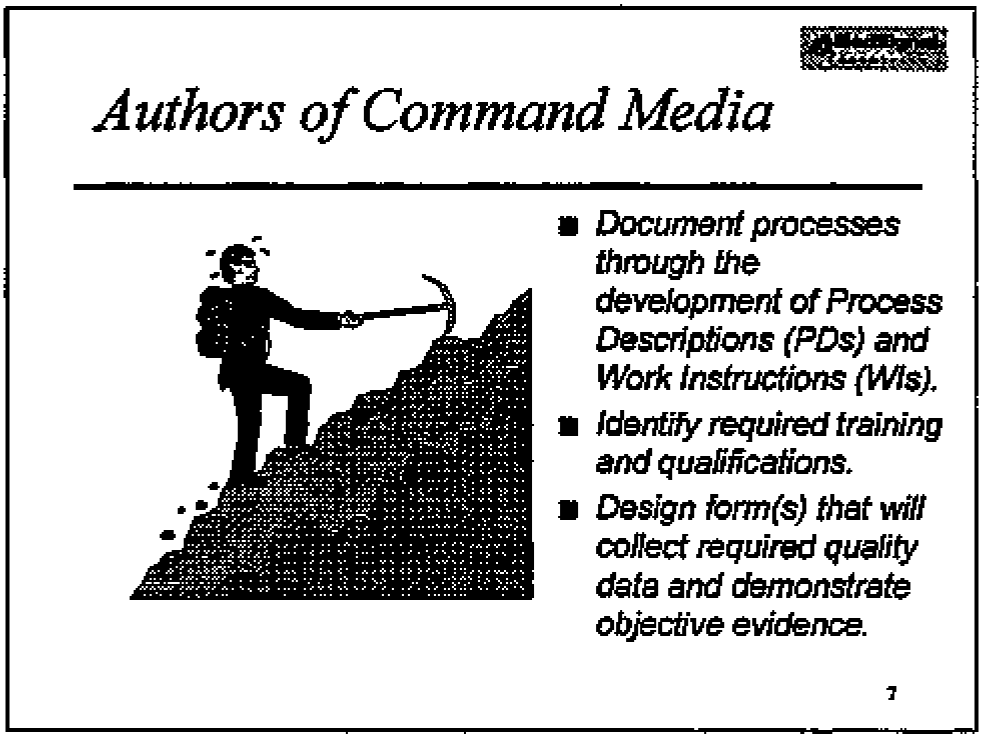


Command Media Numbering Scheme

Process Descriptions: xx.xx.xx.xx
Work Instructions:xx.xx.xx.xx.xx

xx. xx. xx. xx. xx

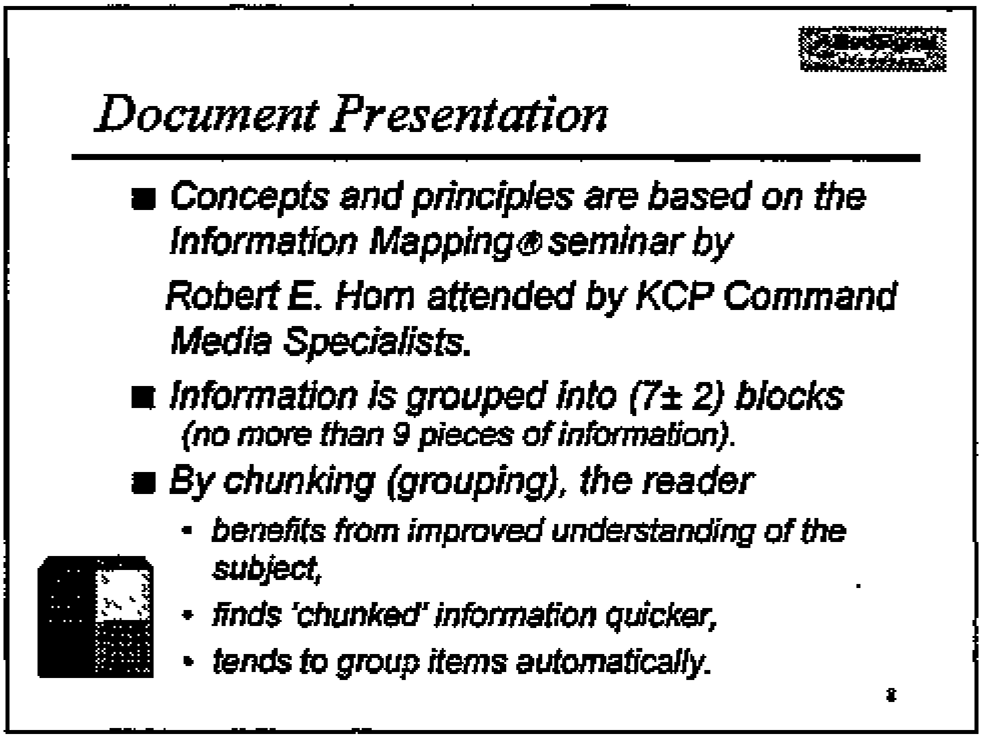
Identifies Work Instruction
Identifies Process Description
Identifies Business Process
Identifies Business Function
Identifies Business Functional Area



Authors of Command Media

- Document processes through the development of Process Descriptions (PDs) and Work Instructions (WIs).
- Identify required training and qualifications.
- Design form(s) that will collect required quality data and demonstrate objective evidence.

7



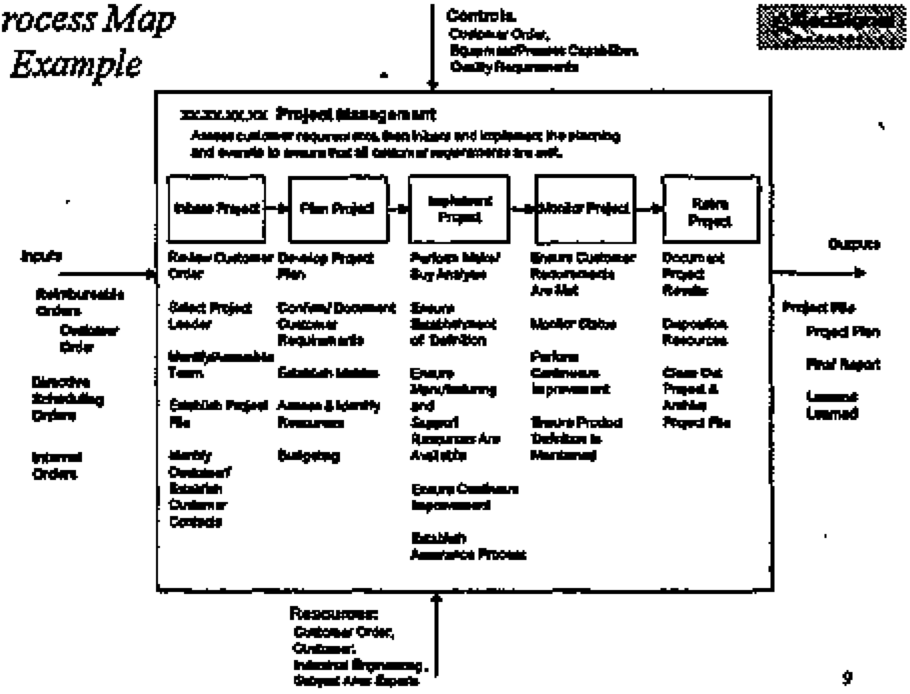
Document Presentation

- Concepts and principles are based on the Information Mapping® seminar by

  Robert E. Hom attended by KCP Command Media Specialists.
- Information is grouped into (7± 2) blocks (no more than 9 pieces of information).
- By chunking (grouping), the reader
  - benefits from improved understanding of the subject,
  - finds 'chunked' information quicker,
  - tends to group items automatically.

8

Process Map Example



Controls:
Customer Order,
Equipment/Process Capabilities,
Quality Requirements

xx.xx.xx.xx Project Management
Assess customer requirements, then initiate and implement the planning and oversite to ensure that all customer requirements are met.

Initiate Project → Plan Project → Implement Project → Monitor Project → Retire Project

Inputs
Reimbursable Orders
Customer Order
Directive Scheduling Orders
Internal Orders

Initiate Project:
Review Customer Order
Select Project Leader
Identify/Assemble Team
Establish Project File
Identify Customer/ Establish Customer Contacts

Plan Project:
Develop Project Plan
Confirm/ Document Customer Requirements
Establish Metrics
Assess & Identify Resources
Budgeting

Implement Project:
Perform Make/ Buy Analysis
Ensure Establishment of Definition
Ensure Manufacturing and Support Resources Are Available
Ensure Continuous Improvement
Establish Assurance Process

Monitor Project:
Ensure Customer Requirements Are Met
Monitor Status
Perform Continuous Improvement
Ensure Product Definition is Maintained

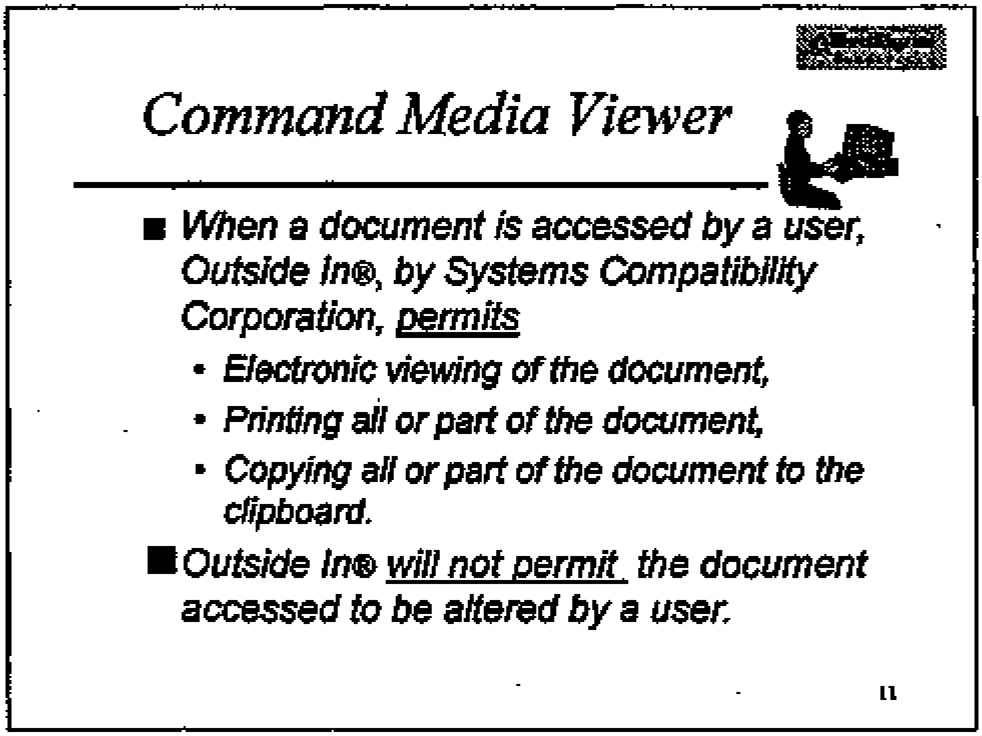
Retire Project:
Document Project Results
Disposition Resources
Close Out Project & Archive Project File

Outputs
Project File
Project Plan
Final Report
Lessons Learned

Resources:
Customer Order,
Customer,
Industrial Engineering,
Subject Area Experts

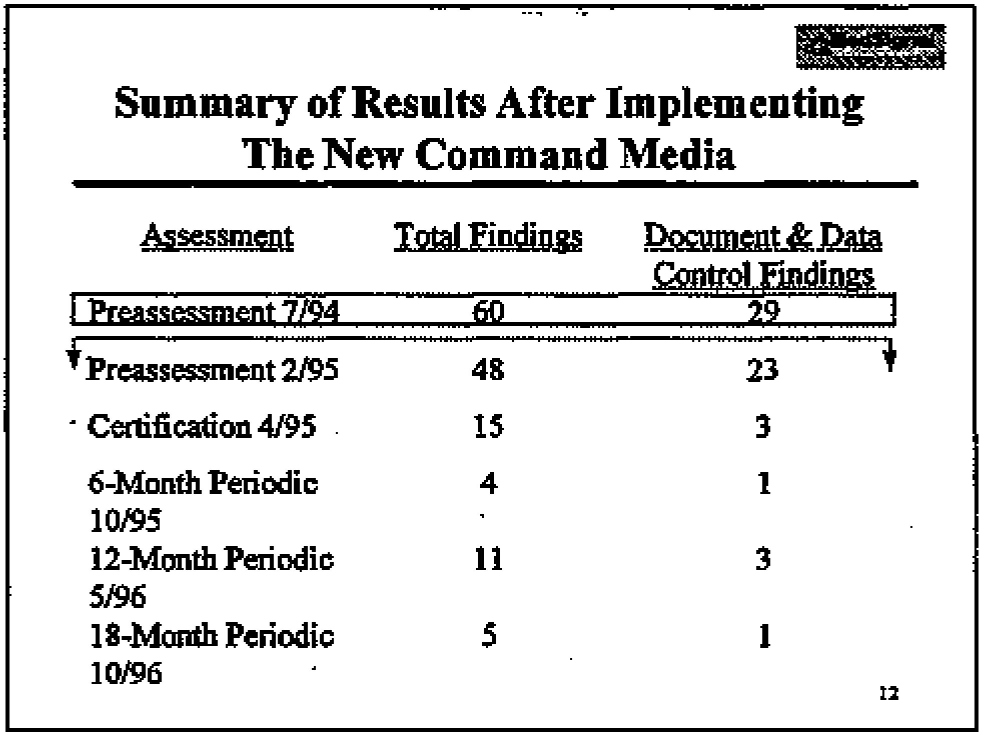
Command Media Access

- System is built on Microsoft Access® and Visual Basic®.
- Documents are created in a Microsoft Word® template and release is controlled by the **Business Systems Management** team.
- User access is through a click on an icon on a Window® of the Program Manager of a user's PC.



Command Media Viewer

- When a document is accessed by a user, Outside In®, by Systems Compatibility Corporation, permits
  - Electronic viewing of the document,
  - Printing all or part of the document,
  - Copying all or part of the document to the clipboard.
- Outside In® will not permit the document accessed to be altered by a user.

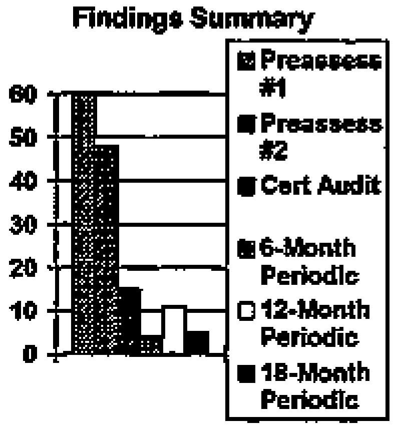


Summary of Results After Implementing The New Command Media

Assessment	Total Findings	Document & Data Control Findings
Preassessment 7/94	60	29
Preassessment 2/95	48	23
Certification 4/95	15	3
6-Month Periodic 10/95	4	1
12-Month Periodic 5/96	11	3
18-Month Periodic 10/96	5	1

Summary of Results

- Graphical representation of the 27-month history of findings during the ISO9001 certification process and from the required periodic audits to retain certification.



ISO/COMMAND MEDIA OPERATIONAL STRUCTURE

Process Steering Committee
Purpose: Oversee and champion the overall process-based philosophy of Command Media and ensure its implementation plant-wide.
Frequency: Quarterly reviews of activities and issues.
Chair: Management Representative
Members: 6 KCP Directors + Management Representative

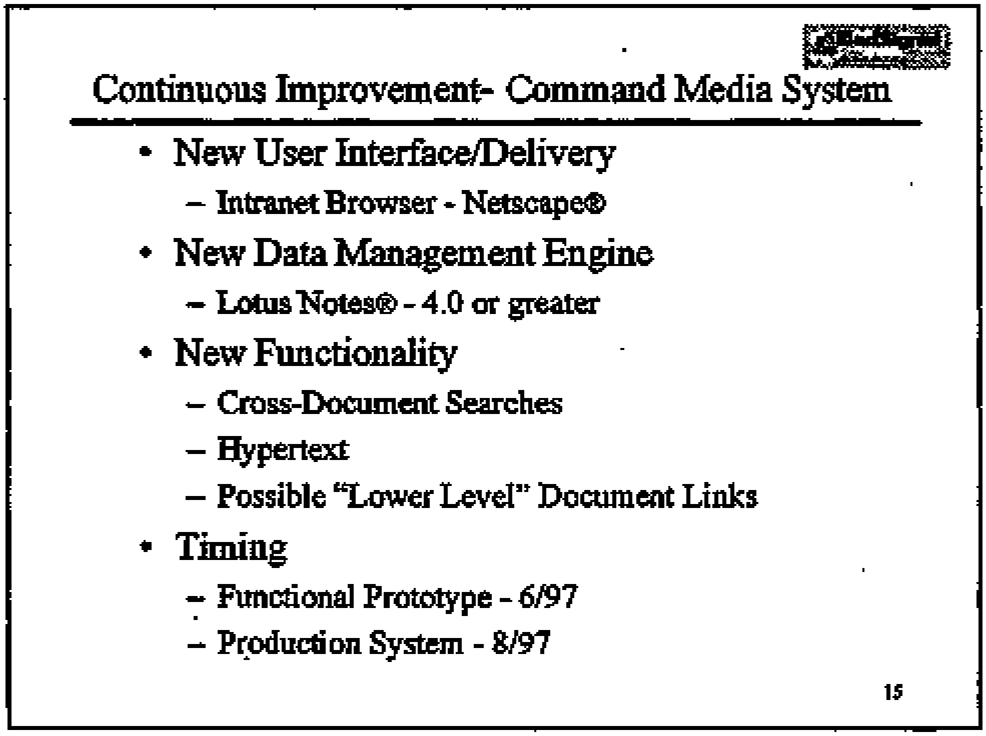
Systems Team
Purpose: Manage and oversee the day-to-day issues associated with the Business Model and Command Media placement, quality, content, and length.
Frequency: Ad hoc, once or twice monthly.
Members: 8 including ISO Coordinator

Business Systems Management
Provide administrative support for the establishment and maintenance of PDs & WIs including formal review, release control, and archival.
Members: 4

ISO Coordinator
Ensure plant-wide compliance to the ISO 9001 standard. Responsible for ensuring Command Media supports ISO 9001 and for performing reviews and evaluations throughout the ISO 9001 processes.

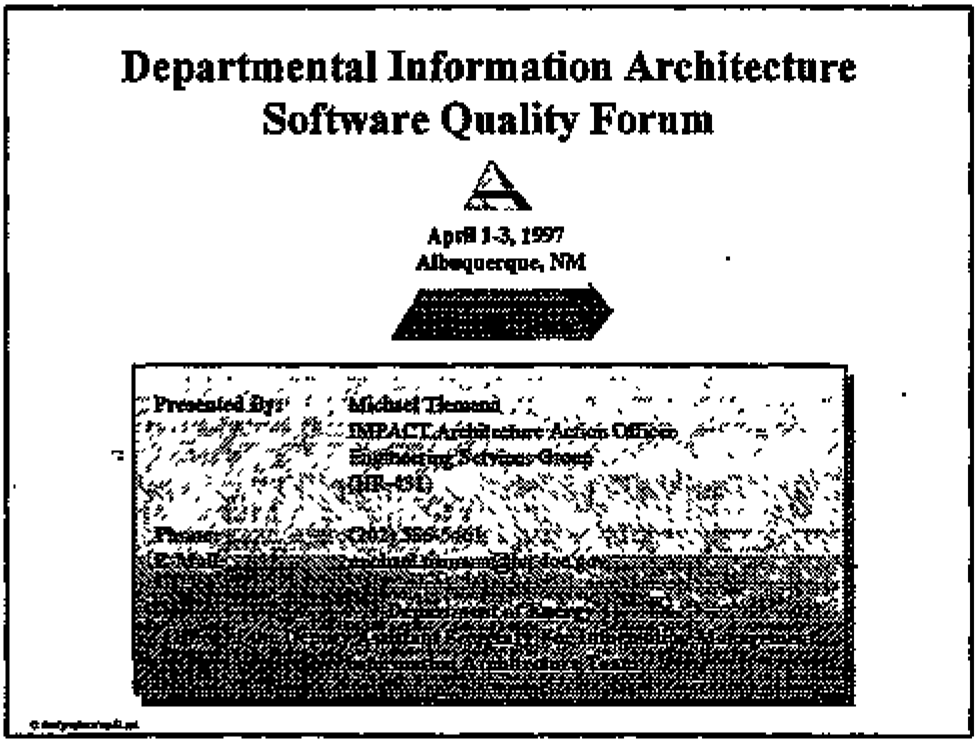
Process leaders
Owners of the plant processes.
Responsible for the technical content of PDs and WIs.

Command Media Specialists - a group of local area experts, trained in process mapping with experience in process-based philosophy. Available to provide assistance as needed.



Continuous Improvement- Command Media System

- New User Interface/Delivery
  - Intranet Browser - Netscape®
- New Data Management Engine
  - Lotus Notes® - 4.0 or greater
- New Functionality
  - Cross-Document Searches
  - Hypertext
  - Possible "Lower Level" Document Links
- Timing
  - Functional Prototype - 6/97
  - Production System - 8/97



Departmental Information Architecture Software Quality Forum

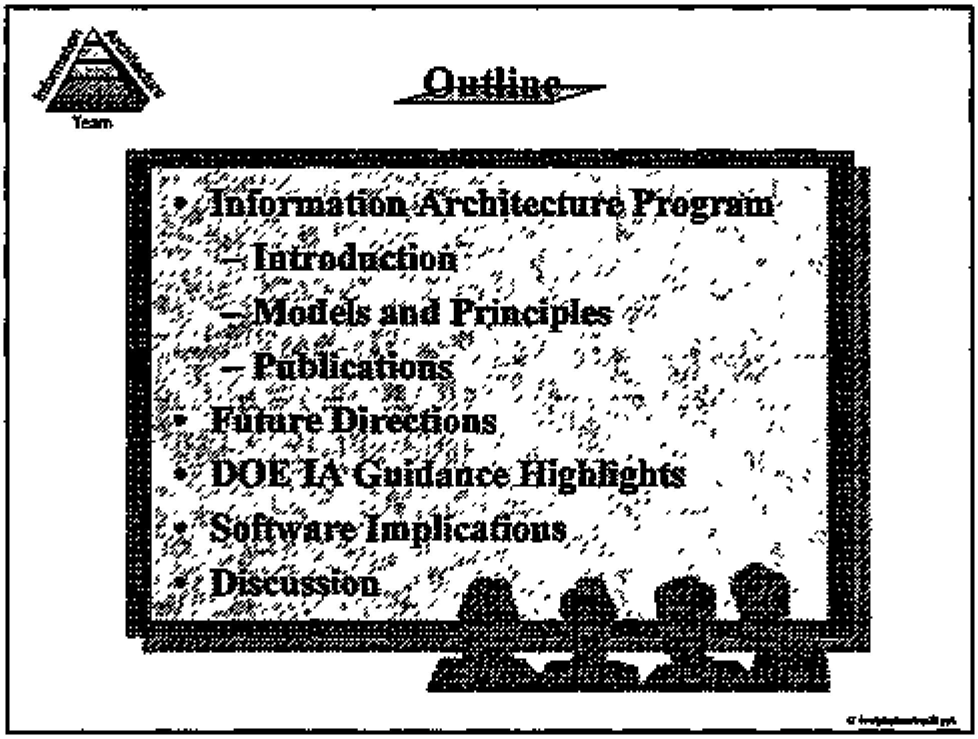
April 1-3, 1997
Albuquerque, NM

Presented By: Michael Tiemann
IMPACT Architecture Action Officer
Engineering Services Group
(HR-431)

Phone: (202) 586-[illegible]

E-Mail: [illegible]

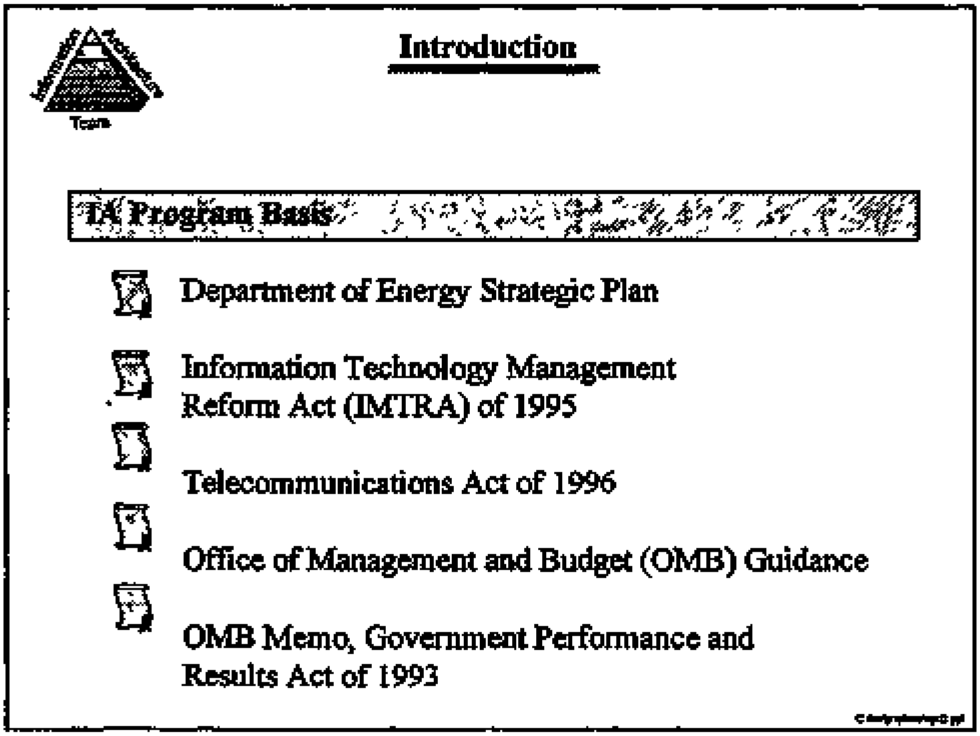
[illegible] Department of Energy [illegible]



Information Architecture Team

Outline

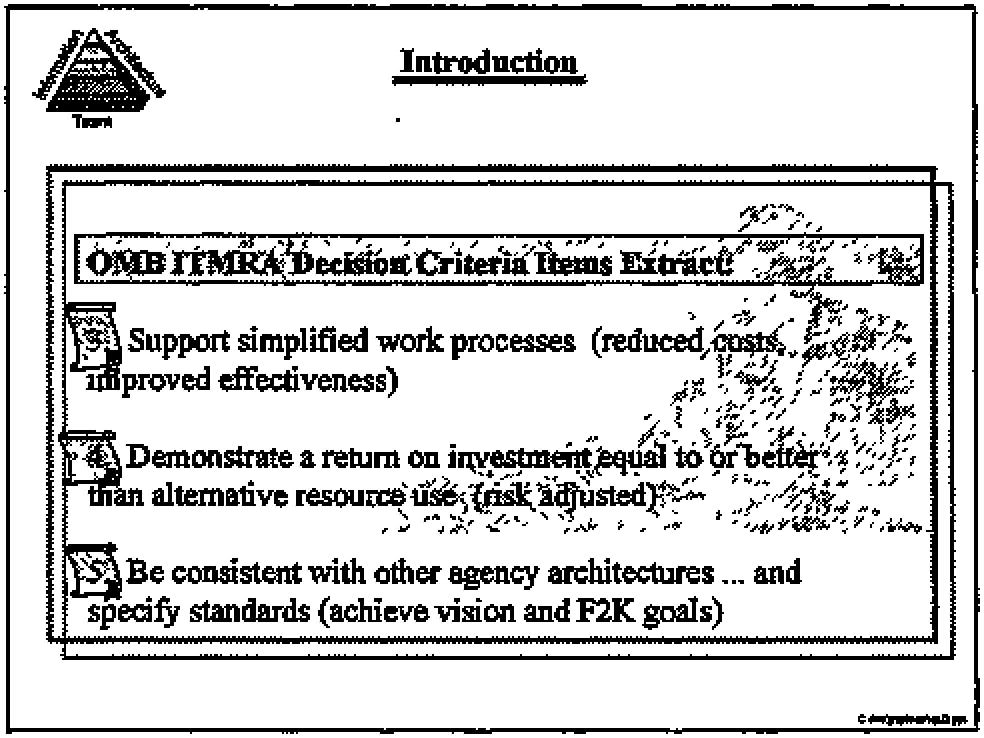
- Information Architecture Program
  - Introduction
  - Models and Principles
  - Publications
- Future Directions
- DOE IA Guidance Highlights
- Software Implications
- Discussion



Introduction

IA Program Basis

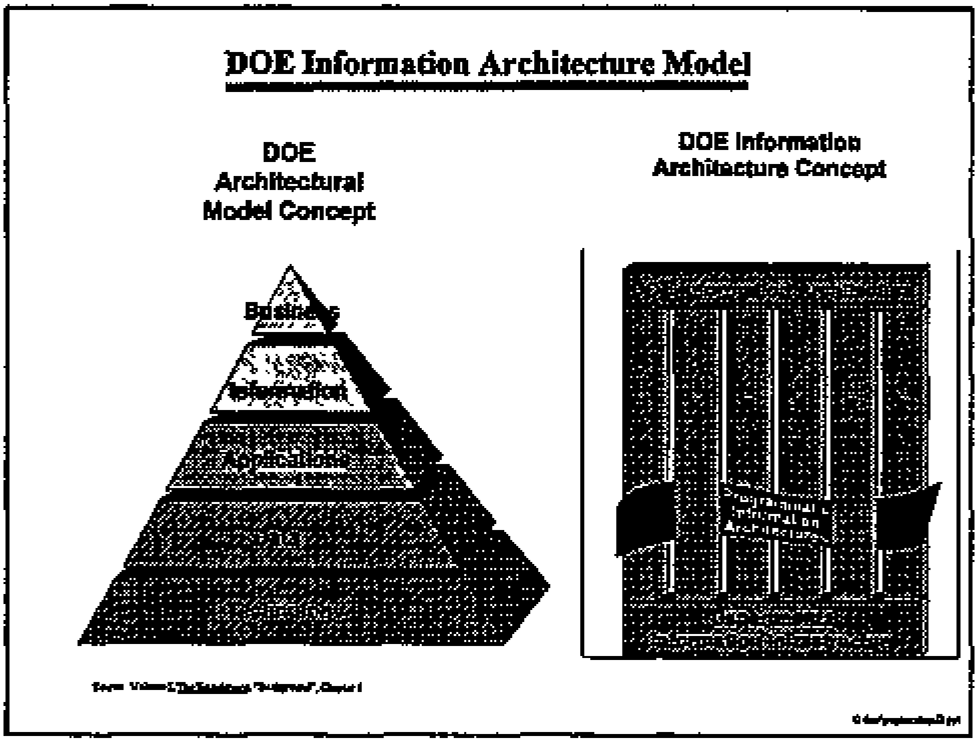
- Department of Energy Strategic Plan
- Information Technology Management Reform Act (IMTRA) of 1995
- Telecommunications Act of 1996
- Office of Management and Budget (OMB) Guidance
- OMB Memo, Government Performance and Results Act of 1993



Introduction

OMB ITMRA Decision Criteria Items Extract

3. Support simplified work processes (reduced costs, improved effectiveness)
4. Demonstrate a return on investment equal to or better than alternative resource use (risk adjusted)
5. Be consistent with other agency architectures ... and specify standards (achieve vision and F2K goals)

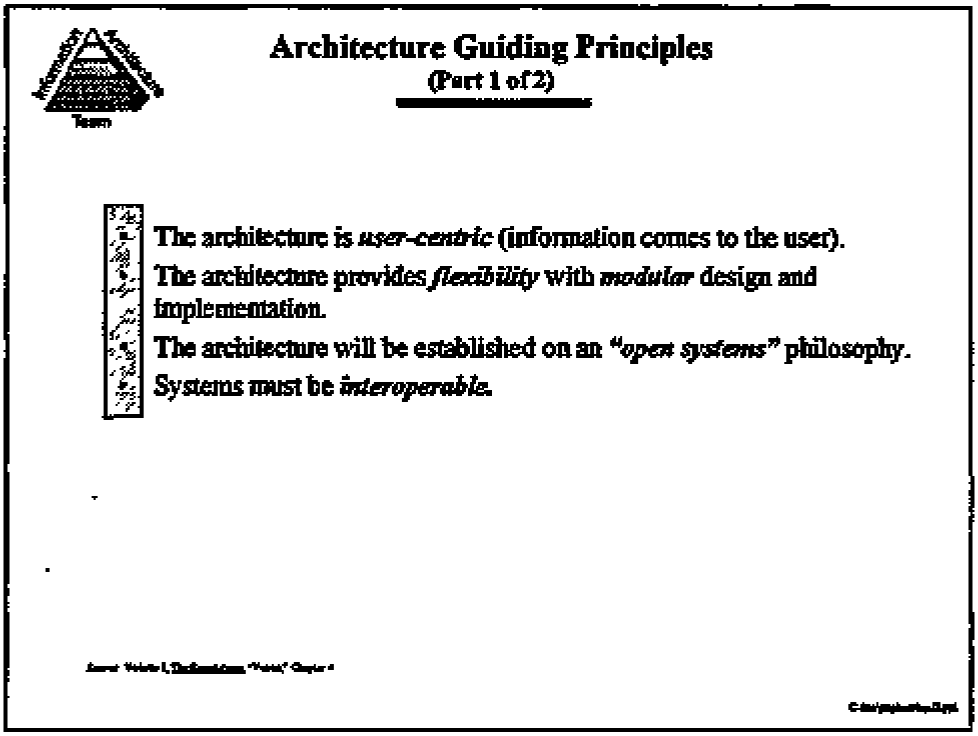


DOE Information Architecture Model

DOE Architectural Model Concept

DOE Information Architecture Concept

[illegible]



Architecture Guiding Principles

(Part 1 of 2)

Information
Architecture
Team

The architecture is *user-centric* (information comes to the user).

The architecture provides *flexibility* with *modular* design and implementation.

The architecture will be established on an *"open systems"* philosophy.

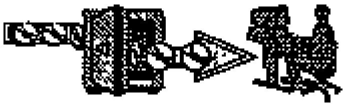
Systems must be *interoperable.*

[illegible]

Architecture Guiding Principles

(Part 2 of 2)

- *Security* is designed into all architectural elements, *balancing accessibility* and ease of use with *protection of data.*
- *Information stewards* should be identified to ensure *quality* and *accessibility* of information resources.
- DOE complexwide *access to information is the rule* rather than the exception.
- The Department's mission will be accomplished by *use of emerging technologies* to synergistically support business processes.

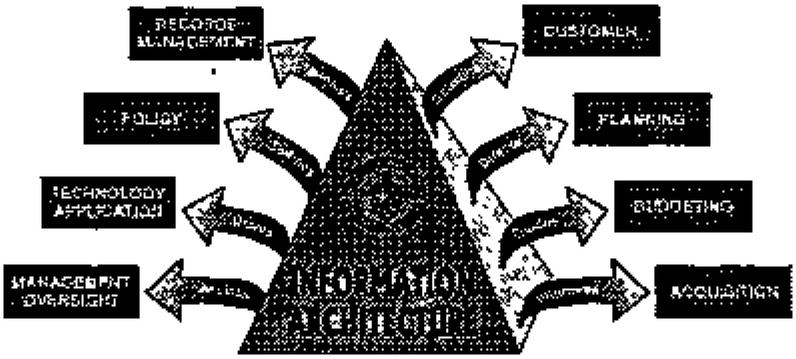


[illegible]

[illegible]

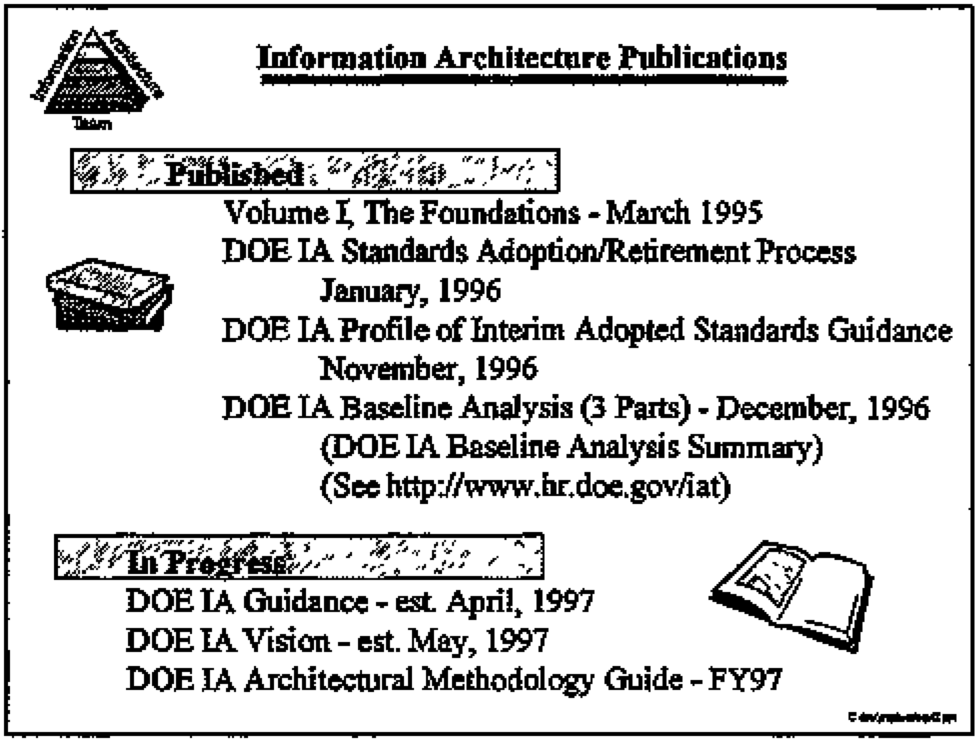
Information Architecture Integration Model

IM Functions **Business Relations**



[illegible]

[illegible]



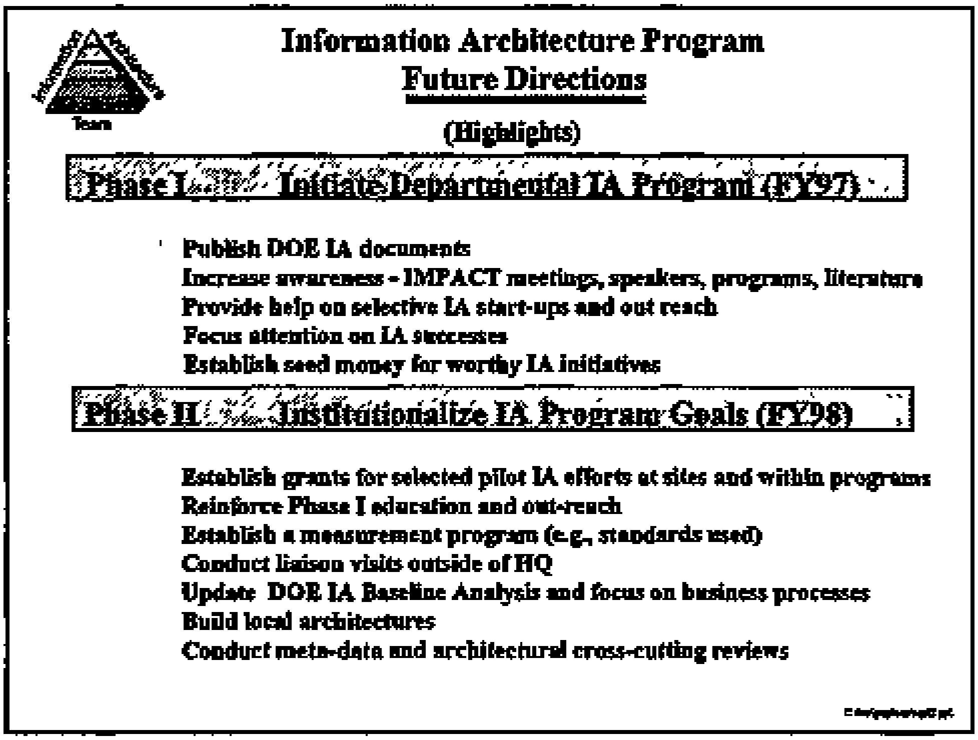
Information Architecture Publications

Published

Volume I, The Foundations - March 1995
DOE IA Standards Adoption/Retirement Process
January, 1996
DOE IA Profile of Interim Adopted Standards Guidance
November, 1996
DOE IA Baseline Analysis (3 Parts) - December, 1996
(DOE IA Baseline Analysis Summary)
(See http://www.hr.doe.gov/iat)

In Progress

DOE IA Guidance - est. April, 1997
DOE IA Vision - est. May, 1997
DOE IA Architectural Methodology Guide - FY97



Information Architecture Program Future Directions

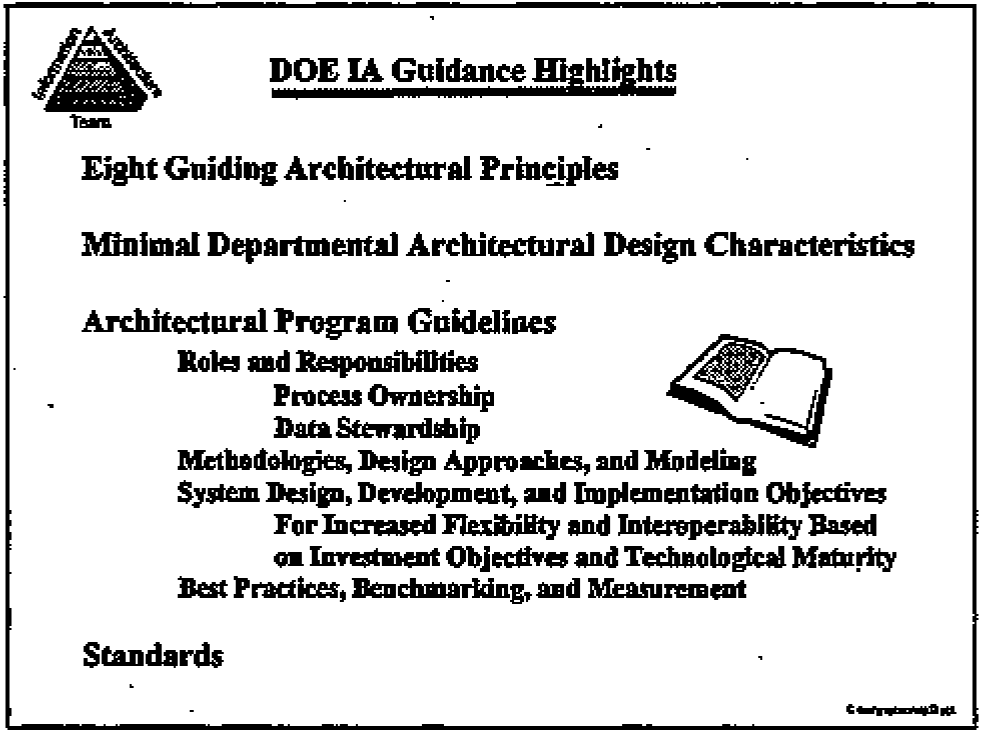
(Highlights)

Phase I Initiate Departmental IA Program (FY97)

- Publish DOE IA documents
- Increase awareness - IMPACT meetings, speakers, programs, literature
- Provide help on selective IA start-ups and out reach
- Focus attention on IA successes
- Establish seed money for worthy IA initiatives

Phase II Institutionalize IA Program Goals (FY98)

- Establish grants for selected pilot IA efforts at sites and within programs
- Reinforce Phase I education and out-reach
- Establish a measurement program (e.g., standards used)
- Conduct liaison visits outside of HQ
- Update DOE IA Baseline Analysis and focus on business processes
- Build local architectures
- Conduct meta-data and architectural cross-cutting reviews



DOE IA Guidance Highlights

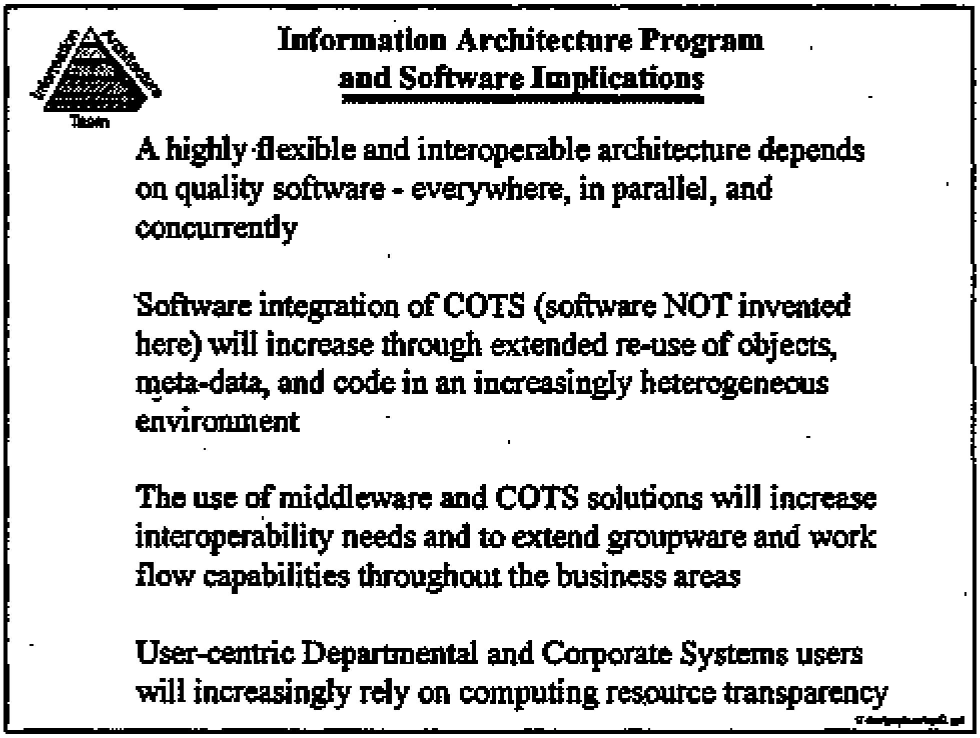
Eight Guiding Architectural Principles

Minimal Departmental Architectural Design Characteristics

Architectural Program Guidelines

- Roles and Responsibilities
  - Process Ownership
  - Data Stewardship
- Methodologies, Design Approaches, and Modeling
- System Design, Development, and Implementation Objectives
  - For Increased Flexibility and Interoperability Based on Investment Objectives and Technological Maturity
- Best Practices, Benchmarking, and Measurement

Standards



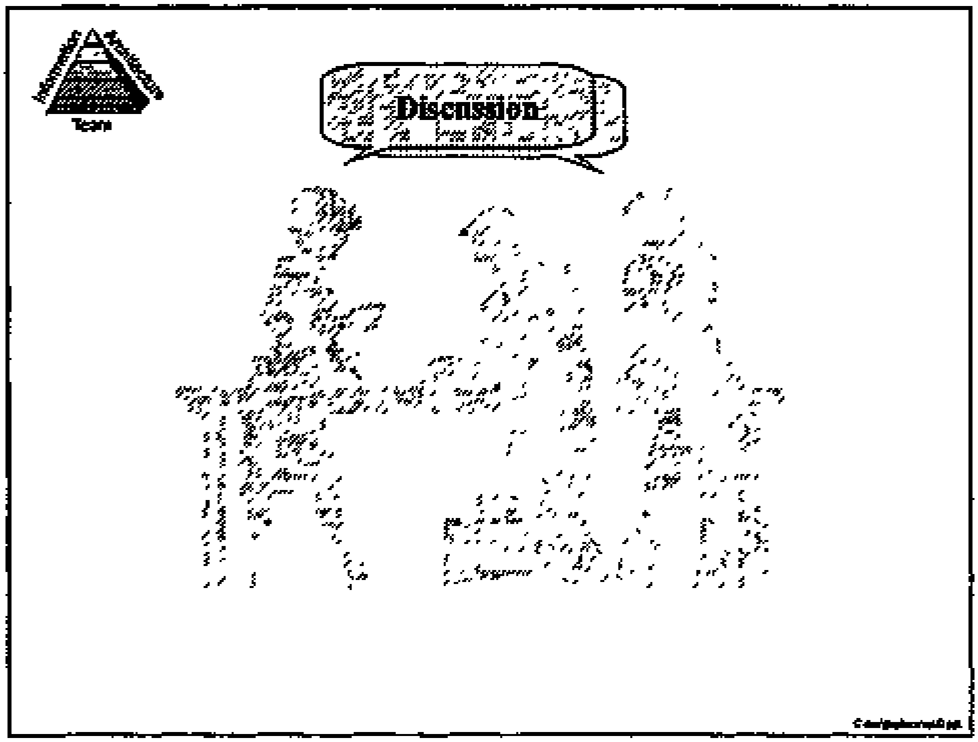
Information Architecture Program and Software Implications

A highly flexible and interoperable architecture depends on quality software - everywhere, in parallel, and concurrently

Software integration of COTS (software NOT invented here) will increase through extended re-use of objects, meta-data, and code in an increasingly heterogeneous environment

The use of middleware and COTS solutions will increase interoperability needs and to extend groupware and work flow capabilities throughout the business areas

User-centric Departmental and Corporate Systems users will increasingly rely on computing resource transparency



Discussion

Session B3: High Integrity / Formal Methods I

Chair Dave Peercy
Sandia National Laboratories

Session : Paper #	Author(s)	*Title*
B3:1	Larry J. Dalton & Marie-Elena Kidd Sandia National Laboratories	*Meeting the High Integrity Software Needs of Today and Tomorrow*
B3:2	Victor Winter Sandia National Laboratories	*An Overview of the AST Software Construction Methodology*
B3:3	Alex Yakhnis & Vladimir Yakhnis Pioneer Technologies	*Towards Automated Construction of Dependable Software/Hardware Systems*

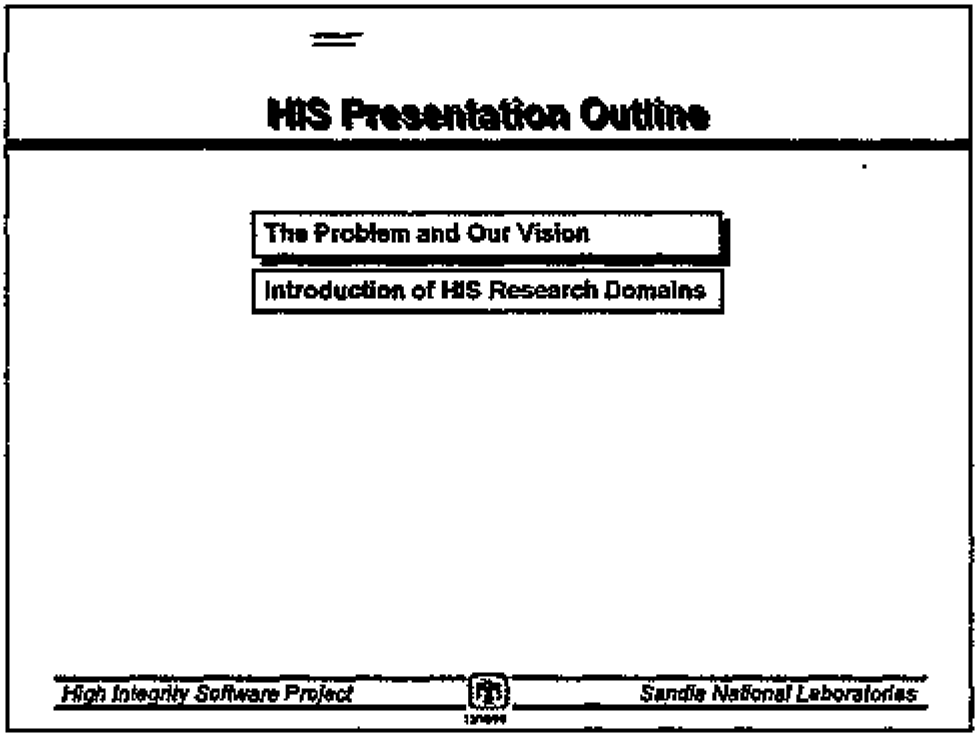
Meeting the High Integrity Software Needs of Today and Tomorrow

Presented at: The 1997 Software Quality Forum

April 1, 1997

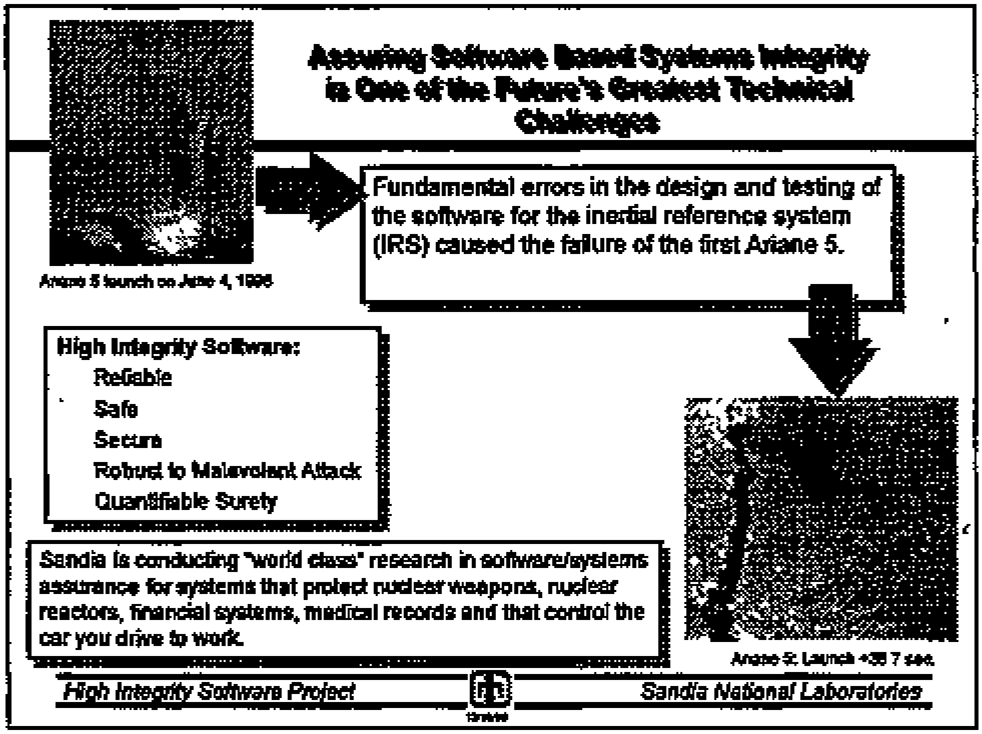
By Larry Dalton & Laney Kidd

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.



HIS Presentation Outline

- The Problem and Our Vision
- Introduction of HIS Research Domains



Assuring Software Based Systems Integrity is One of the Future's Greatest Technical Challenges

Ariane 5 launch on June 4, 1996

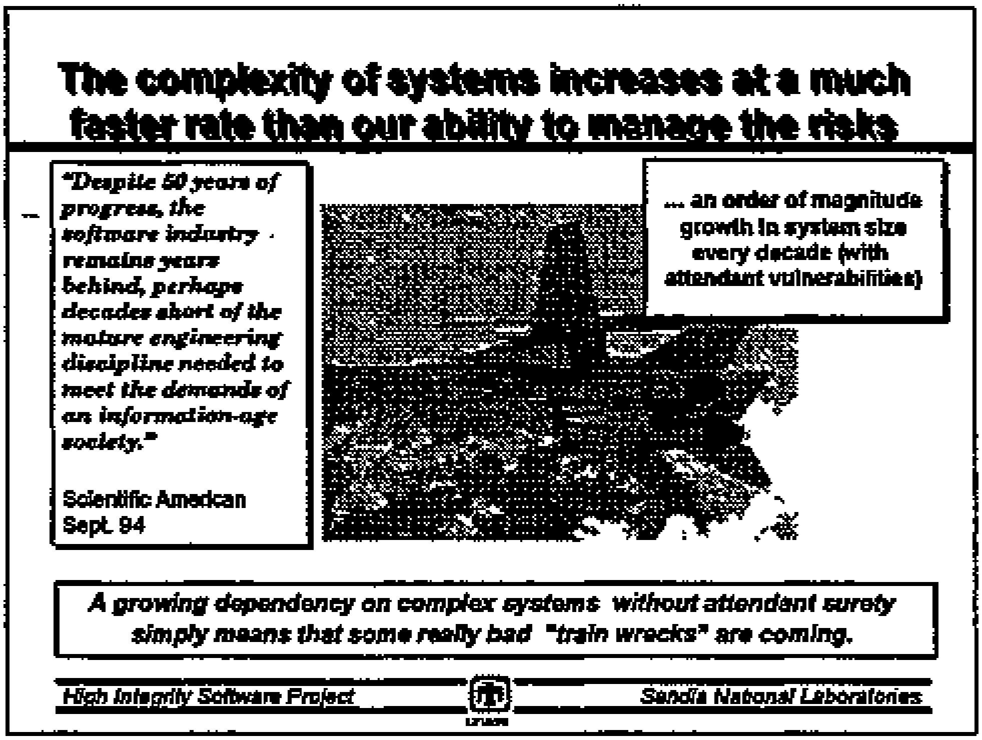
Fundamental errors in the design and testing of the software for the inertial reference system (IRS) caused the failure of the first Ariane 5.

High Integrity Software:
- Reliable
- Safe
- Secure
- Robust to Malevolent Attack
- Quantifiable Surety

Sandia is conducting "world class" research in software/systems assurance for systems that protect nuclear weapons, nuclear reactors, financial systems, medical records and that control the car you drive to work.

Ariane 5: Launch +38.7 sec.

High Integrity Software Project *Sandia National Laboratories*



The complexity of systems increases at a much faster rate than our ability to manage the risks

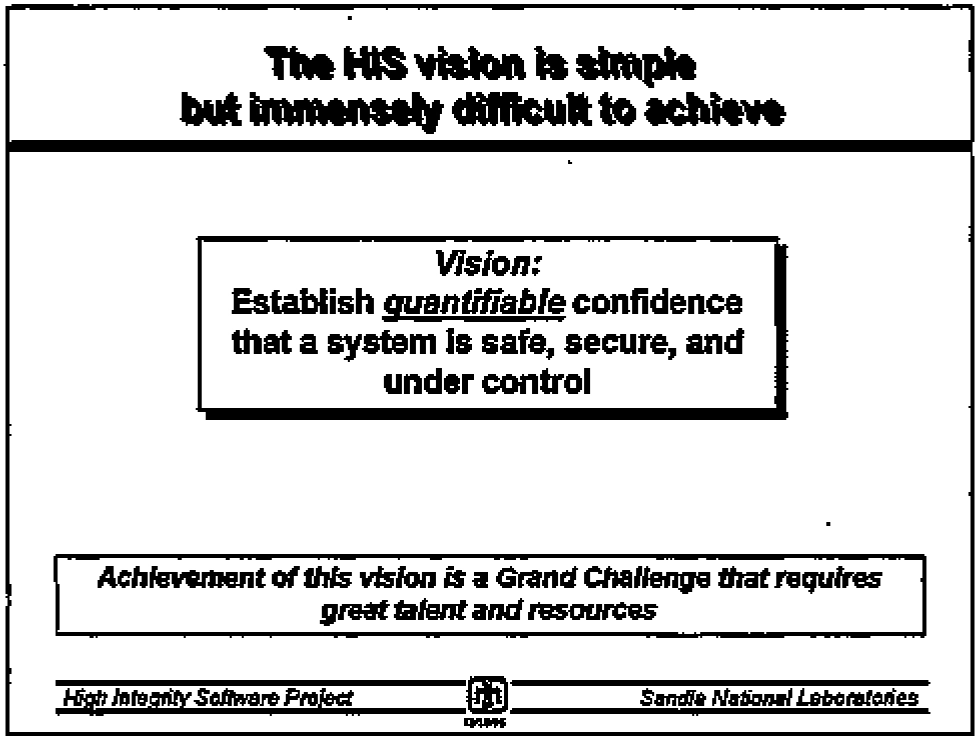
"Despite 50 years of progress, the software industry remains years behind, perhaps decades short of the mature engineering discipline needed to meet the demands of an information-age society."

Scientific American
Sept. 94

... an order of magnitude growth in system size every decade (with attendant vulnerabilities)

A growing dependency on complex systems without attendant surety simply means that some really bad "train wrecks" are coming.

High Integrity Software Project *Sandia National Laboratories*

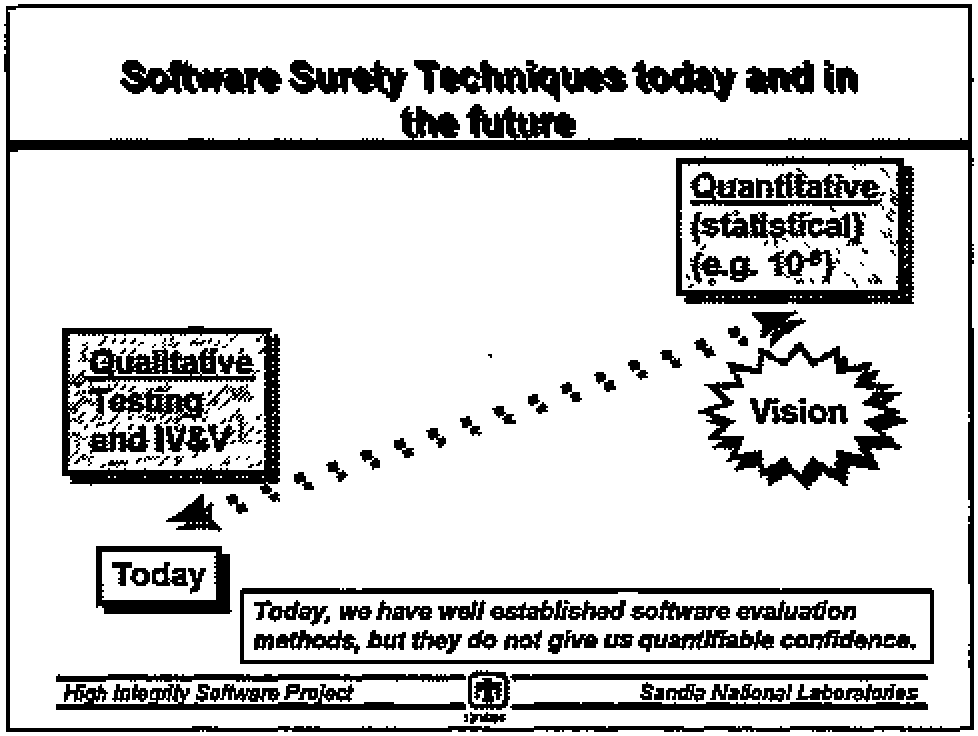


The HIS vision is simple but immensely difficult to achieve

Vision:
Establish ***quantifiable*** confidence that a system is safe, secure, and under control

Achievement of this vision is a Grand Challenge that requires great talent and resources

High Integrity Software Project *Sandia National Laboratories*

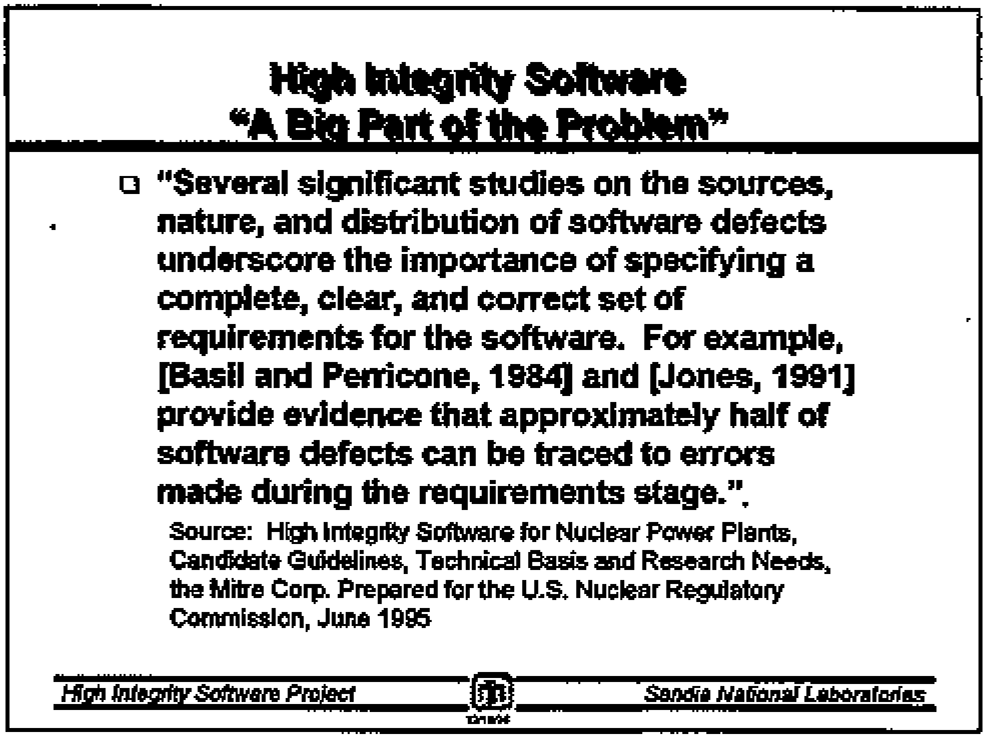


Software Surety Techniques today and in the future

Quantitative (statistical) (e.g. 10^{-6})
Qualitative Testing and IV&V
Vision
Today

Today, we have well established software evaluation methods, but they do not give us quantifiable confidence.

High Integrity Software Project *Sandia National Laboratories*

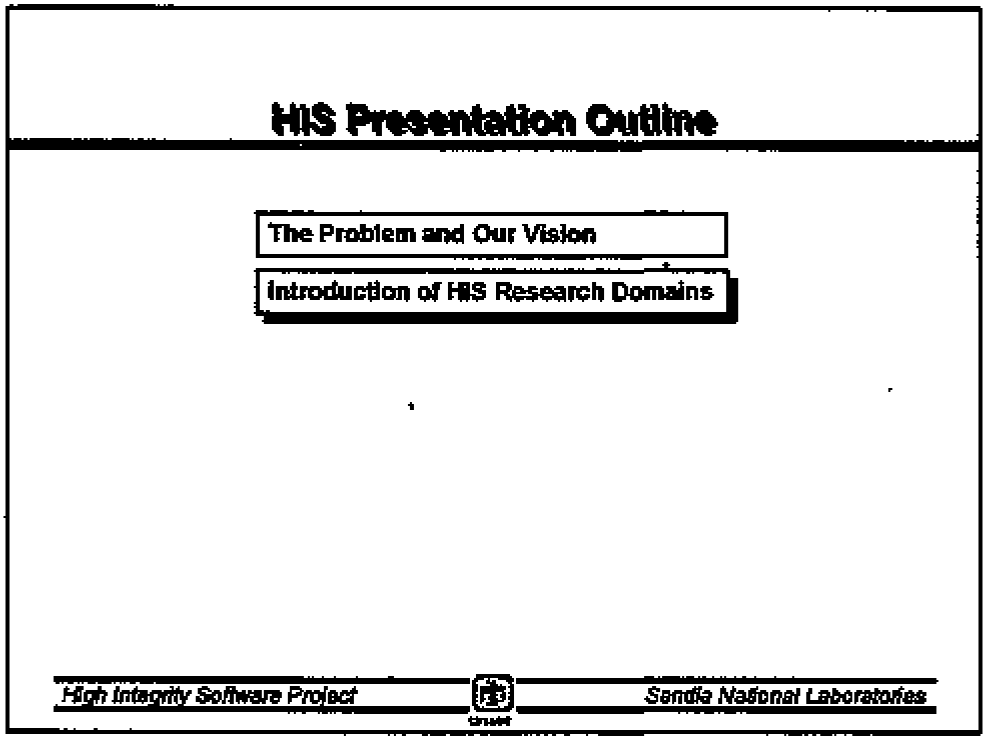


High Integrity Software
"A Big Part of the Problem"

- ❑ "Several significant studies on the sources, nature, and distribution of software defects underscore the importance of specifying a complete, clear, and correct set of requirements for the software. For example, [Basil and Perricone, 1984] and [Jones, 1991] provide evidence that approximately half of software defects can be traced to errors made during the requirements stage.".

  Source: High Integrity Software for Nuclear Power Plants, Candidate Guidelines, Technical Basis and Research Needs, the Mitre Corp. Prepared for the U.S. Nuclear Regulatory Commission, June 1995

High Integrity Software Project *Sandia National Laboratories*

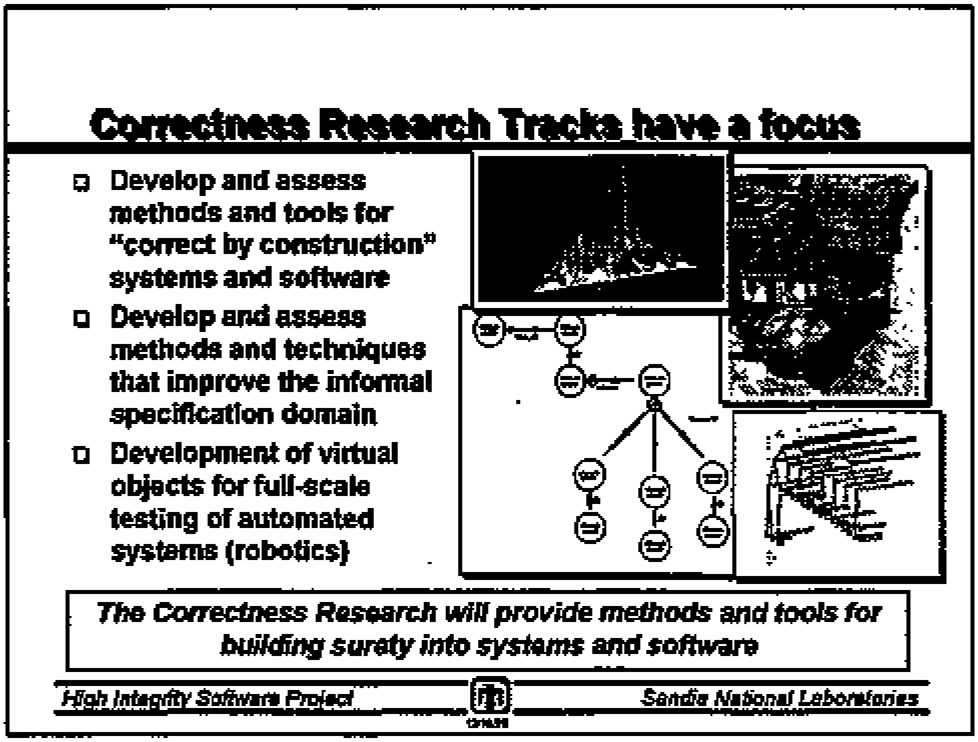


HIS Presentation Outline

The Problem and Our Vision

Introduction of HIS Research Domains

High Integrity Software Project *Sandia National Laboratories*

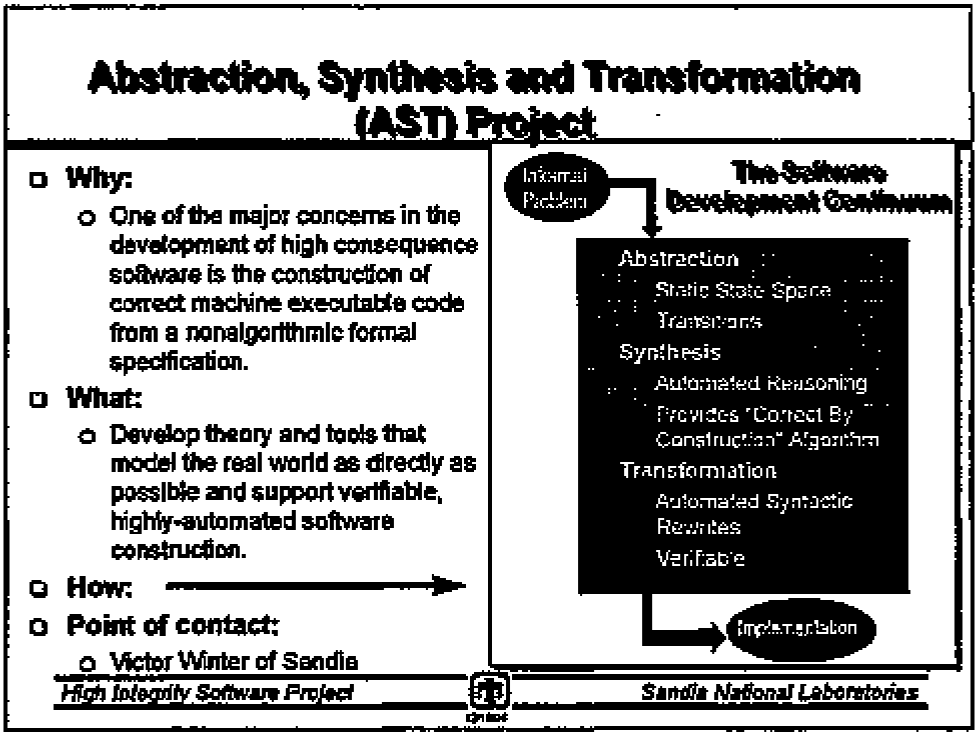


Correctness Research Tracks have a focus

- Develop and assess methods and tools for "correct by construction" systems and software
- Develop and assess methods and techniques that improve the informal specification domain
- Development of virtual objects for full-scale testing of automated systems (robotics)

The Correctness Research will provide methods and tools for building surety into systems and software

High Integrity Software Project | *Sandia National Laboratories*



Abstraction, Synthesis and Transformation (AST) Project

- Why:
  - One of the major concerns in the development of high consequence software is the construction of correct machine executable code from a nonalgorithmic formal specification.
- What:
  - Develop theory and tools that model the real world as directly as possible and support verifiable, highly-automated software construction.
- How: →
- Point of contact:
  - Victor Winter of Sandia

The Software Development Continuum

Informal Problem

Abstraction
- Static State Space
- Transitions

Synthesis
- Automated Reasoning
- Provides "Correct By Construction" Algorithm

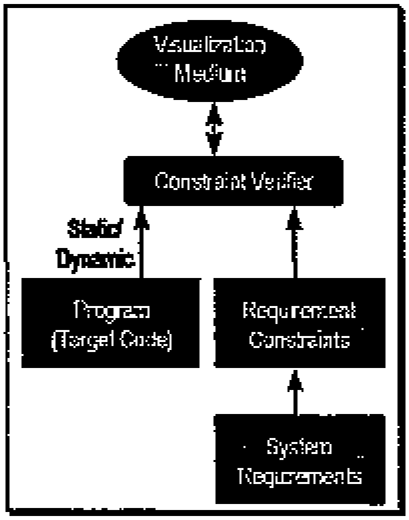
Transformation
- Automated Syntactic Rewrites
- Verifiable

Implementation

High Integrity Software Project | *Sandia National Laboratories*

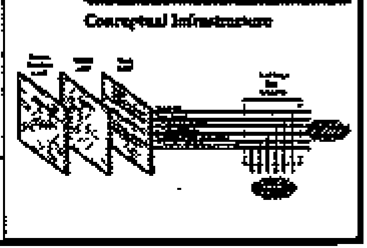
Visualization of Abstract Objects Project

- Why:
  - It is problematic to assess correct implementation of requirements for high consequence software
- What:
  - Improve cognition of software systems behavior and improve software surety confidence
- How:
  - Provide an environment that allows visualization of abstract objects and animation of program behavior incorporating requirement constraints.
- Point of contact:
  - Guylaine Pollock of Sandia



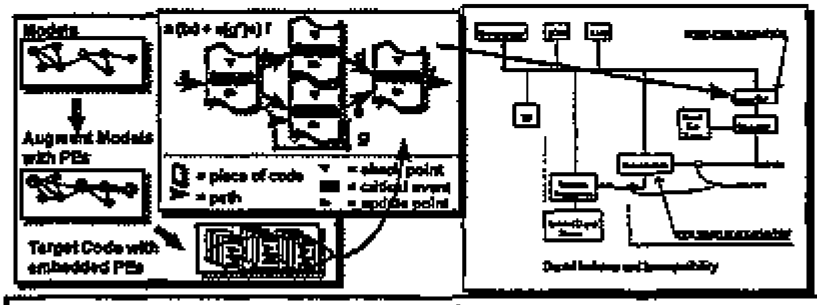
Software Testing for High Consequence Automated Systems Project

- Why: Software testing difficulties
  - not possible to run system for extended periods of time
  - not possible to operate system outside hardware design specifications
  - limited supply of raw material
  - error conditions cannot be created without causing hazards to equipment and people
- What:
  - Create the capability of testing large complex systems using *Production Control Software* through the use of a combination of virtual and real objects.
- How: →
- Point of contact:
  - Lilita Meirans of Sandia


Conceptual Infrastructure

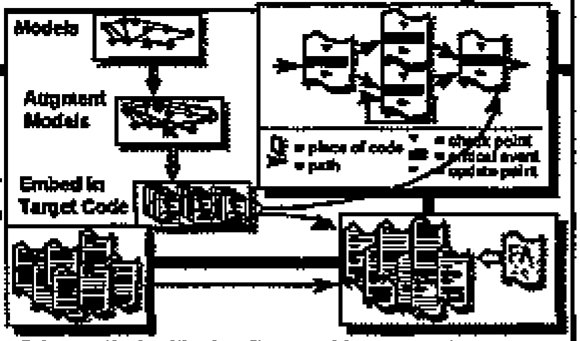
Systems Immunology™ Tracks have a focus

- Develop and assess methods and tools that "immunize" systems and software for fault conditions
- Develop and assess methods and techniques that are immediately applicable to today's high integrity software problems and needs



The Systems Immunology™ will provide in-situ (embedded) methods and tools for dynamic fault management

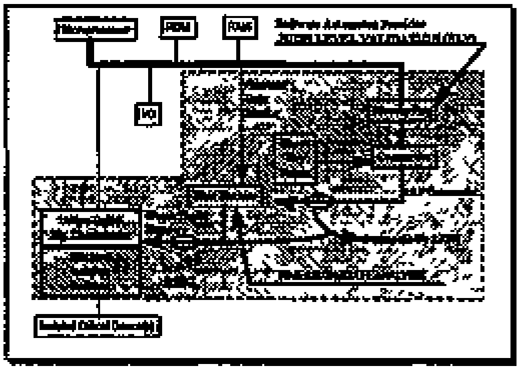
Critical Software Event Execution Reliability (SEER) Project



- Why:
  - Software developers employ ad-hoc, complex, and potentially bug-infested methods to ensure critical software event sequences.
- What:
  - Provide a high level of confidence that critical software driven event execution sequences are maintained in the face of transient software or hardware failures in both normal and abnormal operating environments.
- How:
  - Develop a repeatable, mathematical based solution using finite automata (FA) to develop a method to enforce critical event execution sequences.
- Point of contact:
  - Laney Kidd of Sandia

Digital Device Isolation & Incompatibility (DII) Project

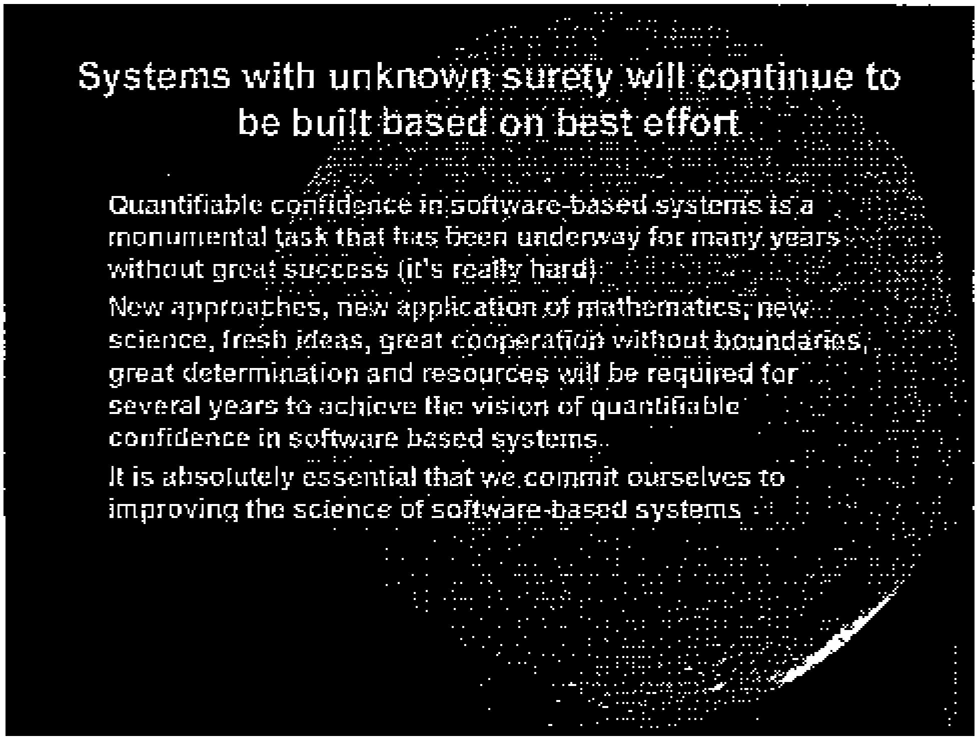
- **Why:**
  - It is not possible, with absolute certainty, to say that a computer-based system will never reach a disastrous failure state. For identified critical functions, DII can guarantee safety/security for normal operating environments.
- **What:**
  - Provide an electro-mechanical "stronglink like" device (integrated circuit size) that keeps electrical paths of critical signals open in case of faults.



- **How:** ↑
- **Point of contact:**
  - Steve Becker of Sandia

Security & System Fault Analysis Project

- **Why:**
  - Software-controlled systems can reach unacceptable states due to either hardware faults or software design faults. Analytic methods are needed to identify hardware that, should it fail, will allow the system to reach an unacceptable state.
- **What:**
  - Develop a top-down fault analysis methodology which will be the basis for a design strategy for high-consequence systems.
- **How:**
  - The analysis methodology, based on Fault-Tree Analysis, will identify high-consequence hardware failures in software-controlled systems. We would also like to extend this methodology so that it can be used to develop safe and secure software code
- **Point of contact:**
  - Edward Fronczak of Sandia

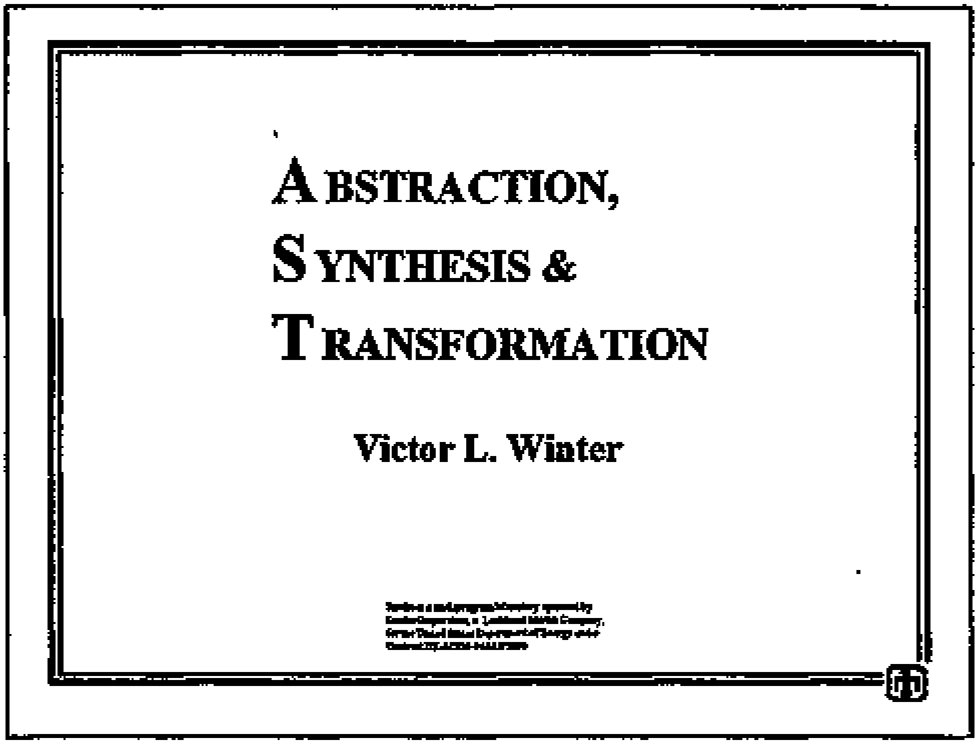


Systems with unknown surety will continue to be built based on best effort

Quantifiable confidence in software-based systems is a monumental task that has been underway for many years without great success (it's really hard)

New approaches, new application of mathematics, new science, fresh ideas, great cooperation without boundaries, great determination and resources will be required for several years to achieve the vision of quantifiable confidence in software based systems.

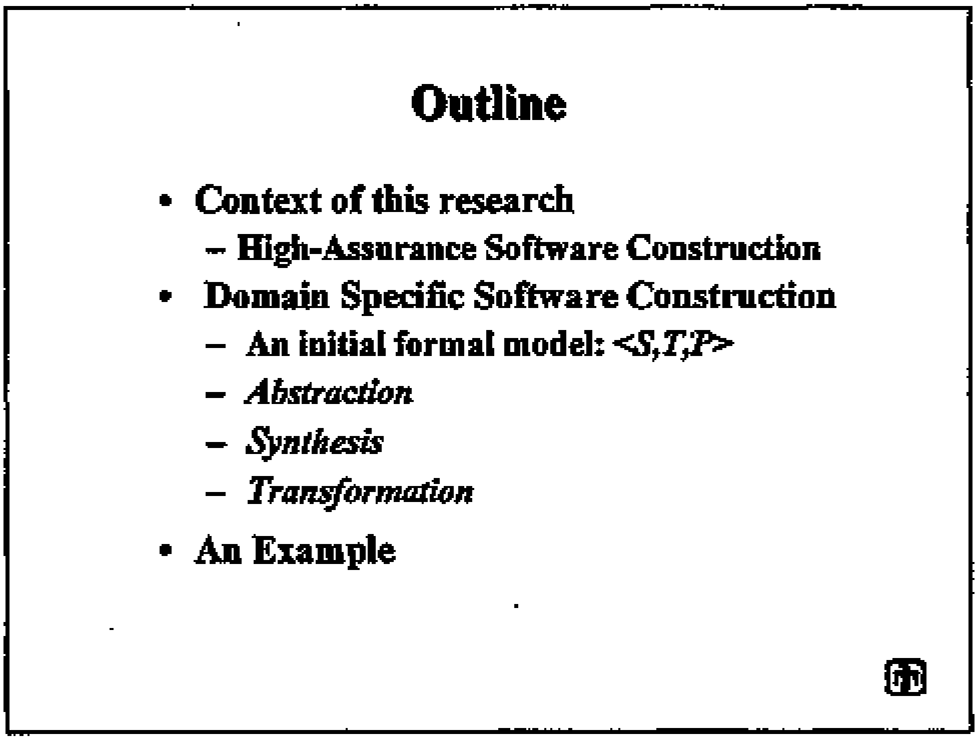
It is absolutely essential that we commit ourselves to improving the science of software-based systems



ABSTRACTION, SYNTHESIS & TRANSFORMATION

Victor L. Winter

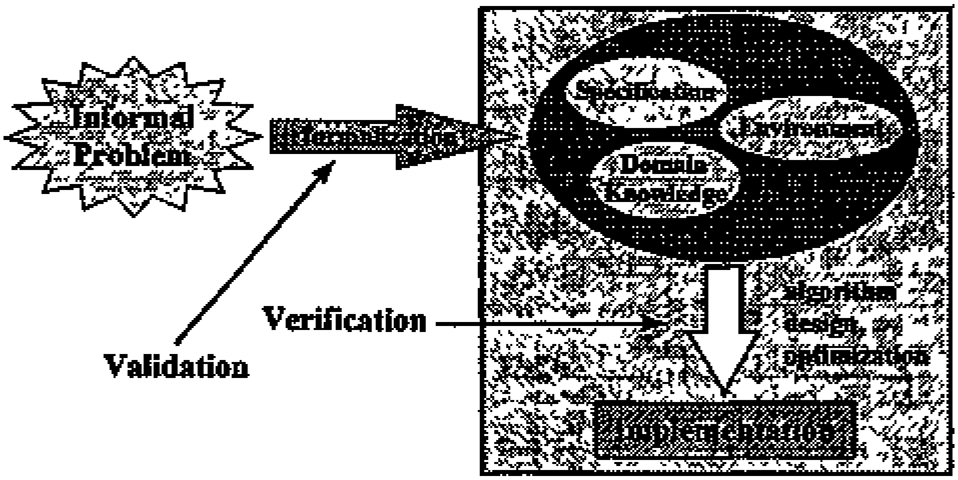
Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under
Contract DE-AC04-94AL85000



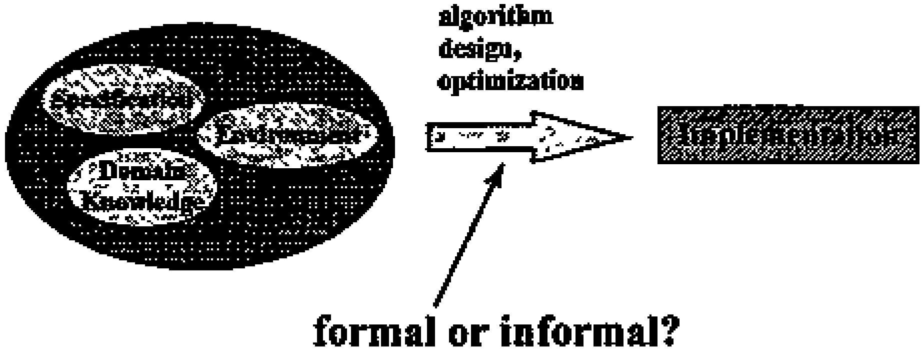
Outline

- Context of this research
  - High-Assurance Software Construction
- Domain Specific Software Construction
  - An initial formal model: <*S,T,P*>
  - *Abstraction*
  - *Synthesis*
  - *Transformation*
- An Example

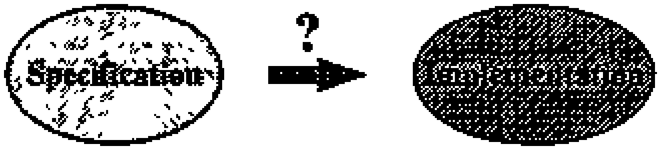
Software Construction



The Process of Constructing Software



Proof Development



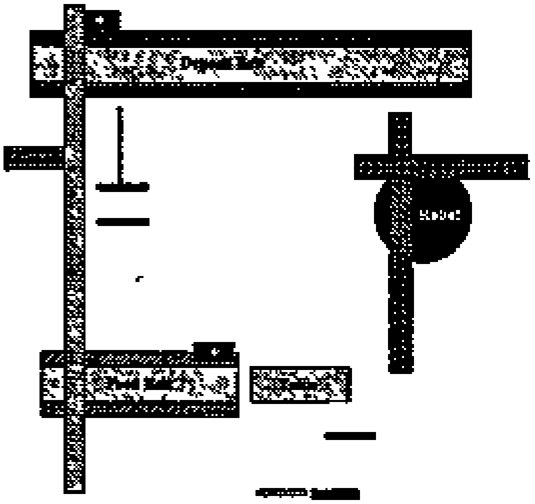
- Direct Proof: The implementation satisfies the specification
  - proofs are often at the wrong level of abstraction
  - inability to reuse proof parts in other applications
- Meta-Verification
  - prove the correctness of the software development process

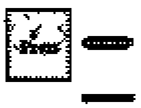
Domain Specific Formal Methods

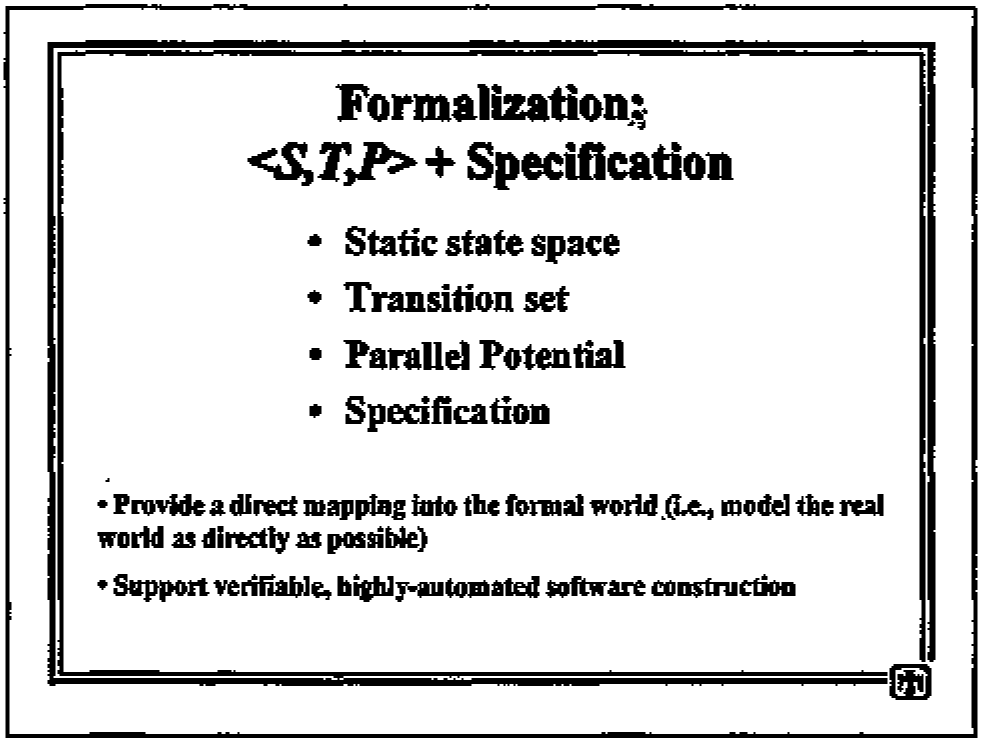
Safety-Critical *Single-Agent* Reactive Systems

- Reactive System
  - some aspect of *time* usually plays a central role (e.g., state changes may take time—they are not instantaneous)
  - controller polls sensors to determine what state the system is in
  - parallel activities are often possible
- *Single-Agent*
  - all transitions initiated by the controller
  - deterministic transitions

The Production Cell: A single-agent reactive system



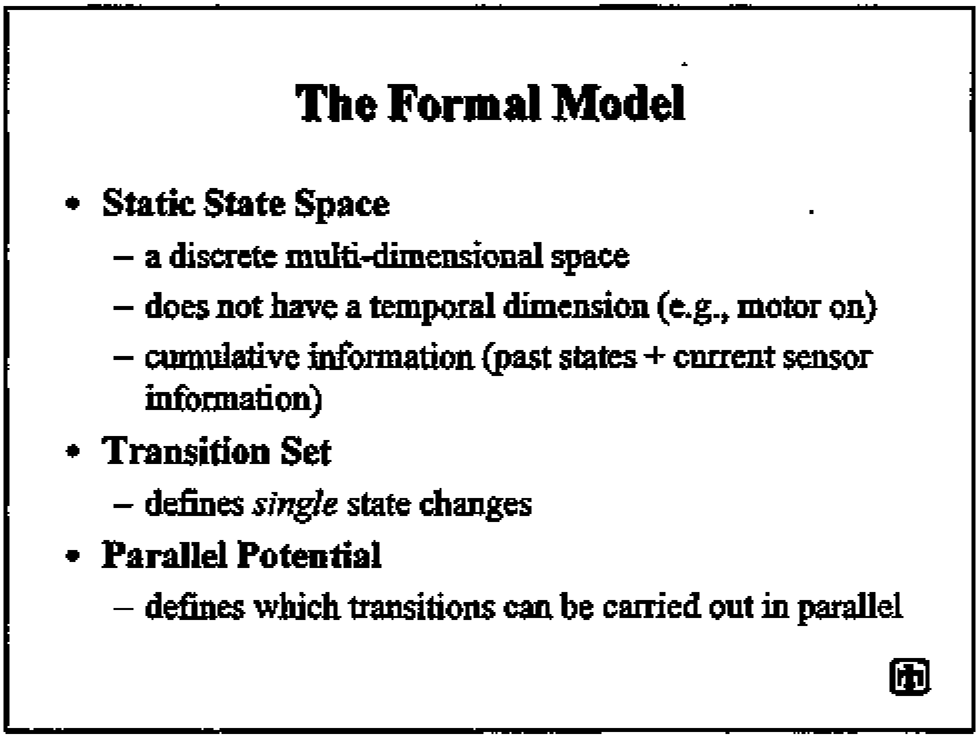




Formalization; <*S,T,P*> + Specification

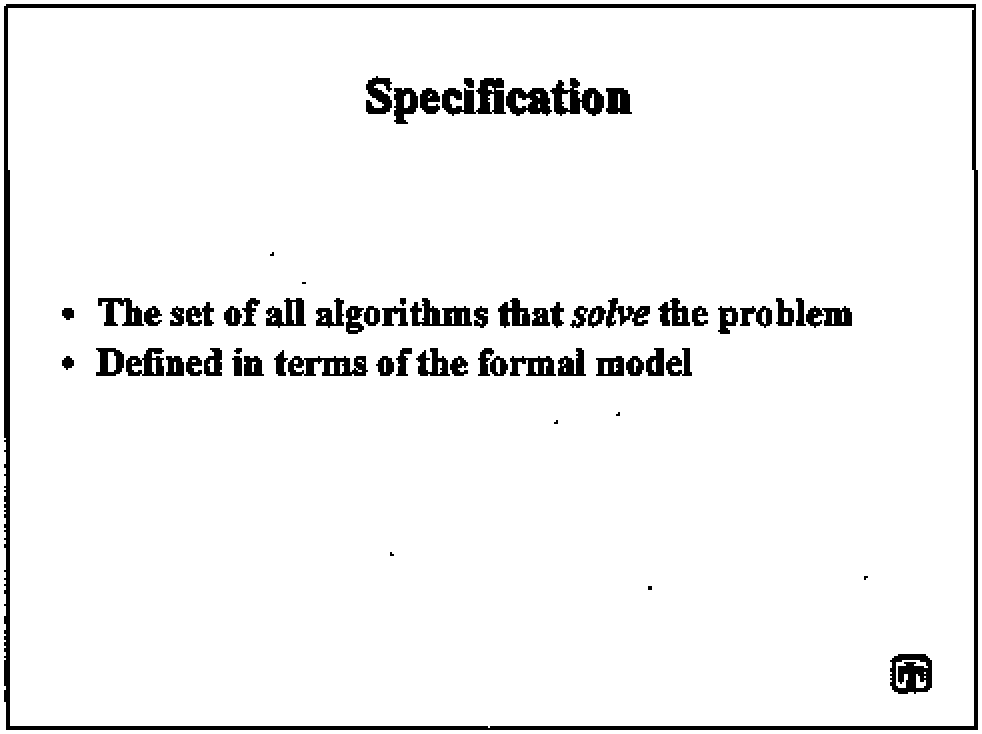
- Static state space
- Transition set
- Parallel Potential
- Specification

- Provide a direct mapping into the formal world (i.e., model the real world as directly as possible)
- Support verifiable, highly-automated software construction



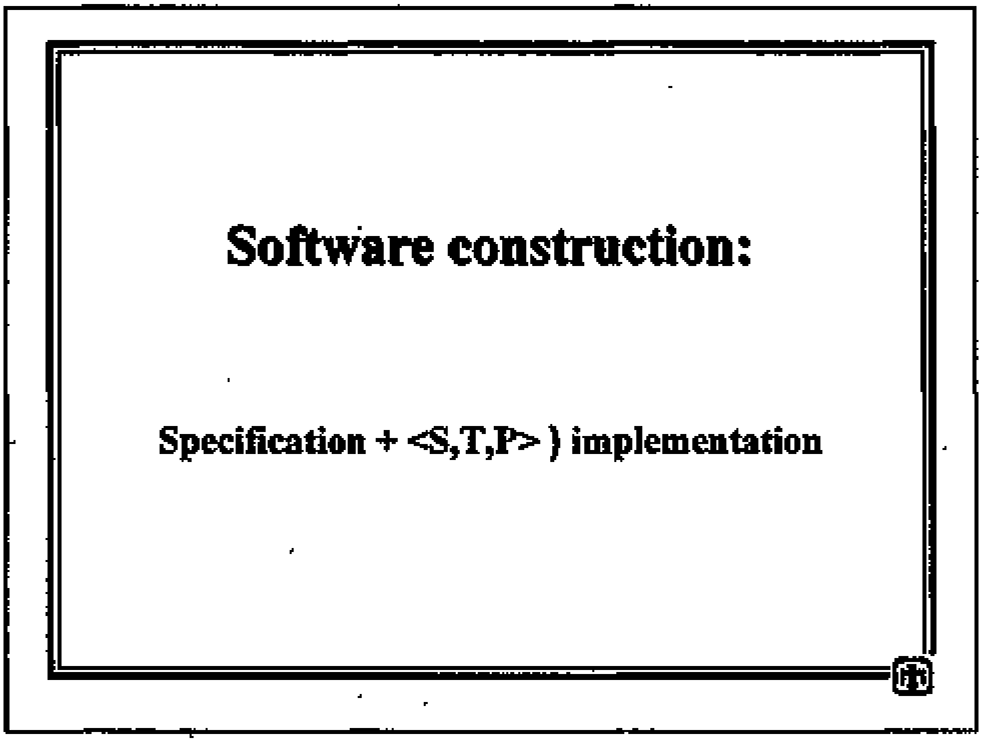
The Formal Model

- Static State Space
  - – a discrete multi-dimensional space
  - – does not have a temporal dimension (e.g., motor on)
  - – cumulative information (past states + current sensor information)
- Transition Set
  - – defines *single* state changes
- Parallel Potential
  - – defines which transitions can be carried out in parallel



Specification

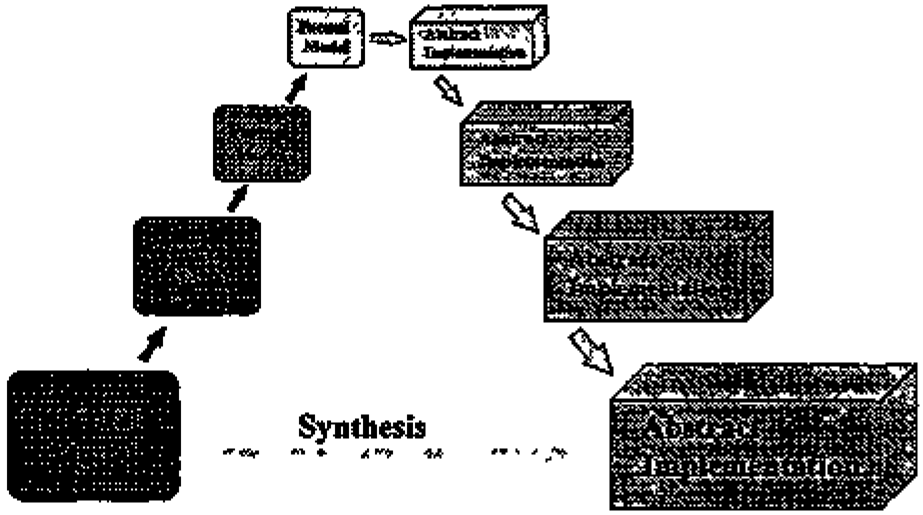
- The set of all algorithms that *solve* the problem
- Defined in terms of the formal model



Software construction:

Specification + <S,T,P>) implementation

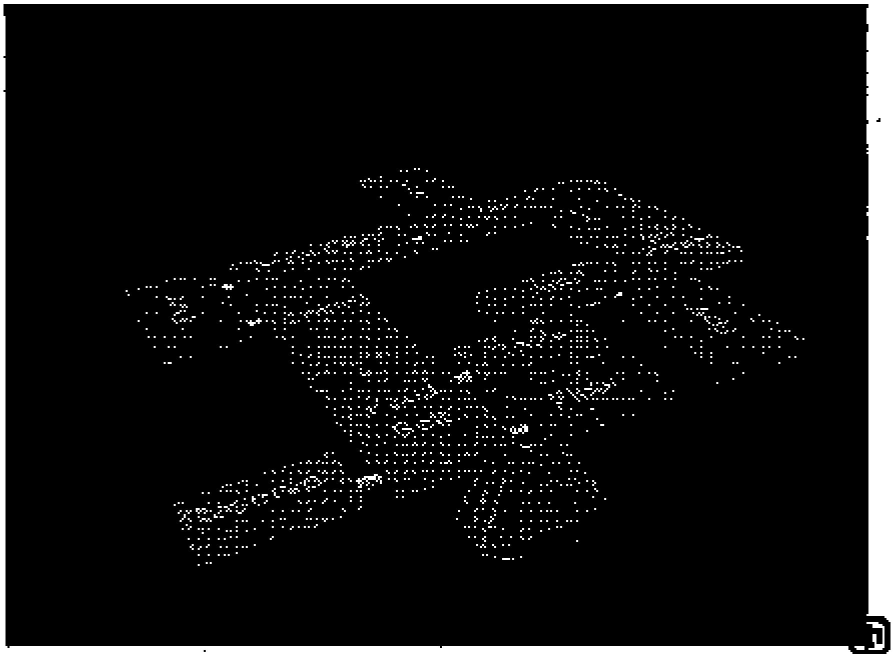
Abstraction and Synthesis



Formal Model

Abstract Implementation

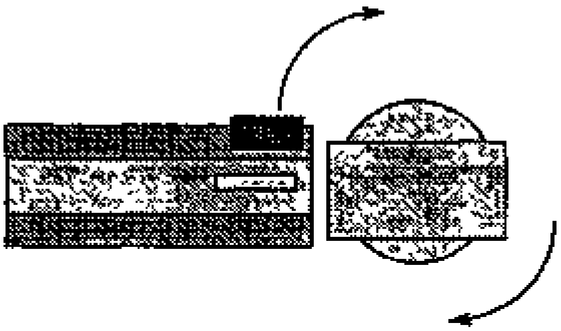
Synthesis



Transformation

- Syntactic Rewrites
- Verifiable within an extended denotational semantic framework
- Automatic Application
- Purpose:
  - optimization
  - introduction of low-level detail
  - targeting a specific computing resource (e.g., single processor, multi-processor)

Simplified Pcell: Modeling Phase



table(0,R(0,0))
table(0,R(1,0))
table(0,R(0,1))
table(0,R(1,1))
etc.

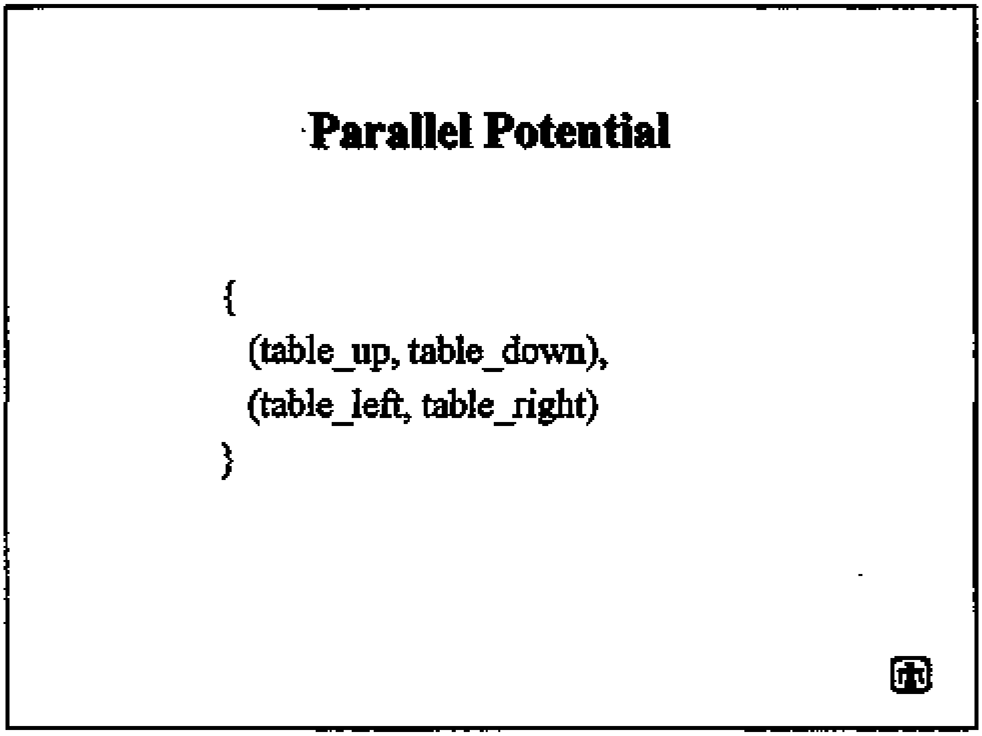
state(fb(x_1,x_2), table(y_1,R(y_2,y_3)))

Modeling Phase: Transitions

- {Precondition} move {Postcondition}

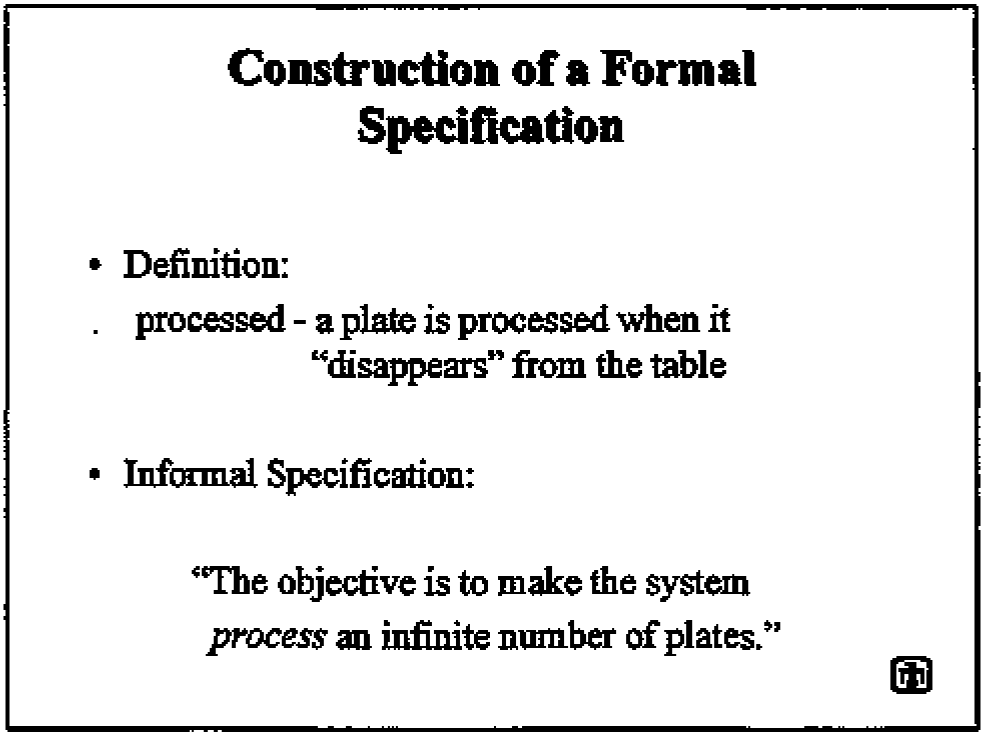
{state(fb(0, x_2), table(y_1 ,R(y_2 , y_3)))}
add_blank
{state(fb(1, x_2), table(y_1 ,R(y_2 , y_3)))}

{state(fb(x_1 , x_2), table(y_1 , R(0,1)))}
table_right
{state(fb(x_1 , x_2), table(0, R(1,1)))}



Parallel Potential

```
{
  (table_up, table_down),
  (table_left, table_right)
}
```



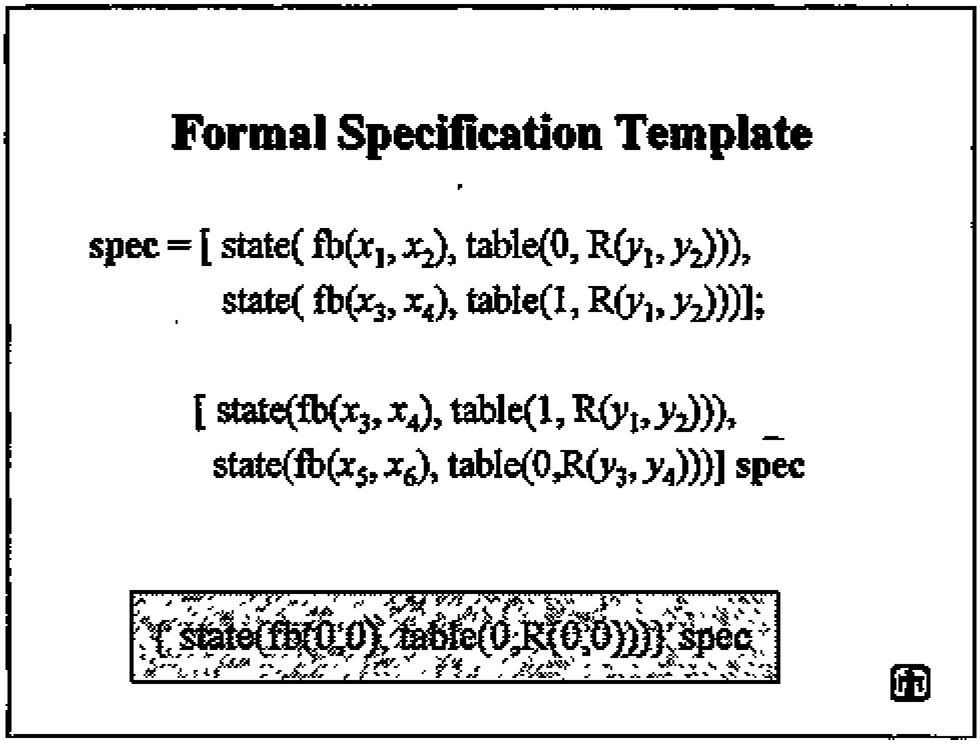
Construction of a Formal Specification

- Definition:

  processed - a plate is processed when it "disappears" from the table

- Informal Specification:

> "The objective is to make the system *process* an infinite number of plates."

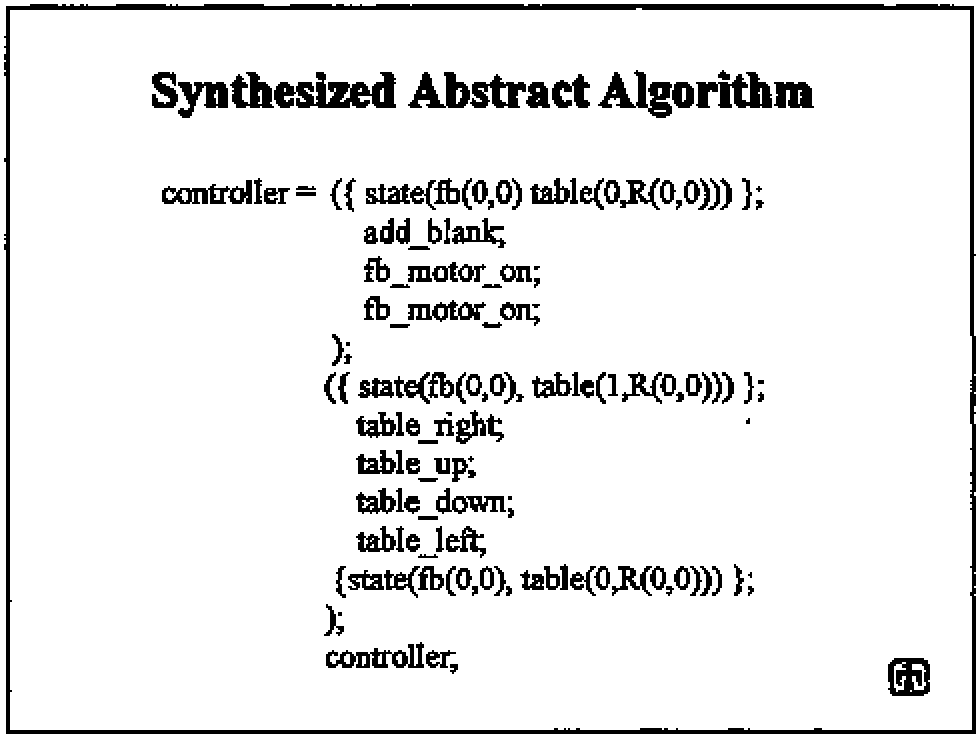


Formal Specification Template

spec = [state(fb(x_1, x_2), table(0, R(y_1, y_2))),
state(fb(x_3, x_4), table(1, R(y_1, y_2)))];

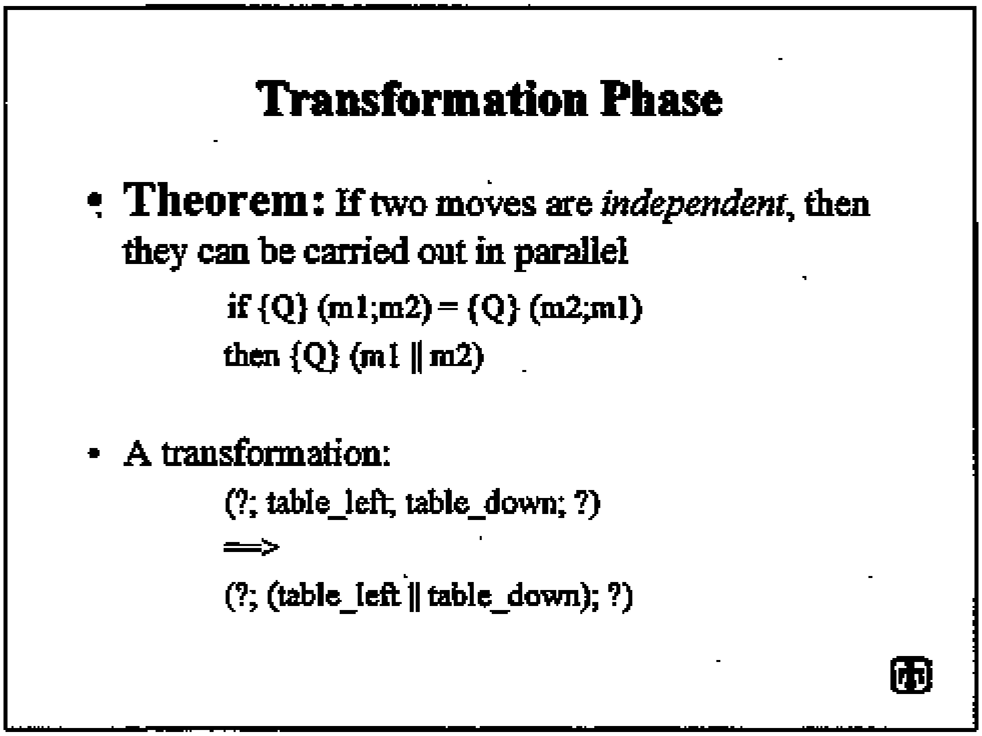
[state(fb(x_3, x_4), table(1, R(y_1, y_2))),
state(fb(x_5, x_6), table(0,R(y_3, y_4)))] spec

{ state(fb(0,0), table(0,R(0,0)))} spec



Synthesized Abstract Algorithm

```
controller =  ({ state(fb(0,0) table(0,R(0,0))) };
                  add_blank;
                  fb_motor_on;
                  fb_motor_on;
               );
               ({ state(fb(0,0), table(1,R(0,0))) };
                  table_right;
                  table_up;
                  table_down;
                  table_left;
                {state(fb(0,0), table(0,R(0,0))) };
               );
               controller;
```



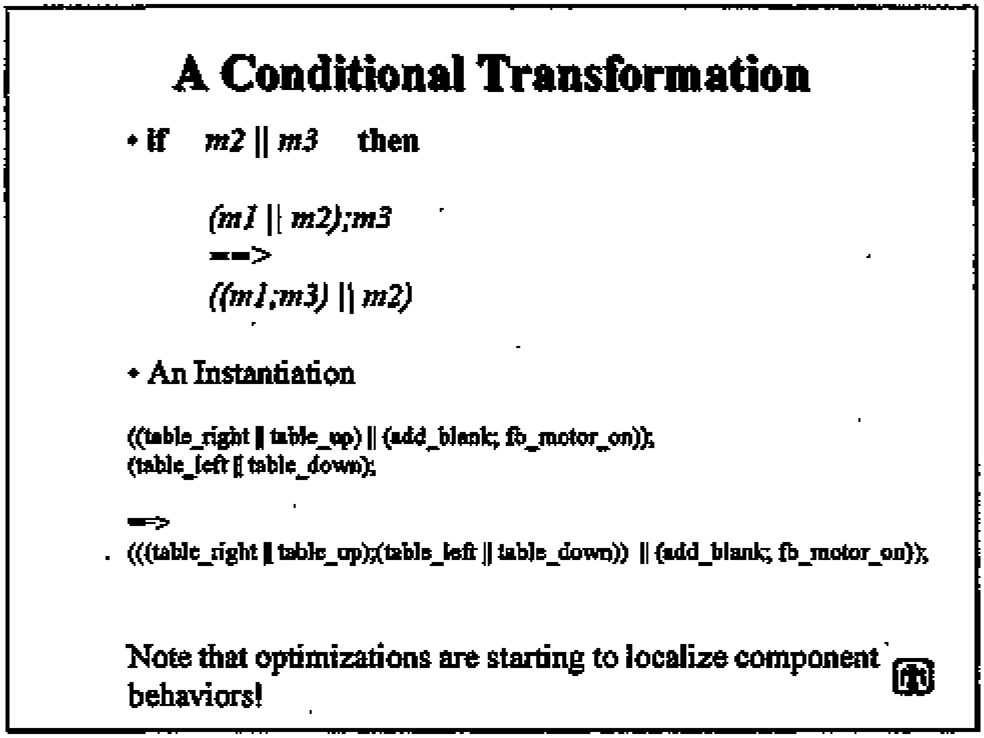
Transformation Phase

- **Theorem:** If two moves are *independent*, then they can be carried out in parallel

  if {Q} (m1;m2) = {Q} (m2;m1)
  then {Q} (m1 || m2)

- A transformation:

  (?; table_left, table_down; ?)
  ==>
  (?; (table_left || table_down); ?)



A Conditional Transformation

- **if** *m2 || m3* **then**

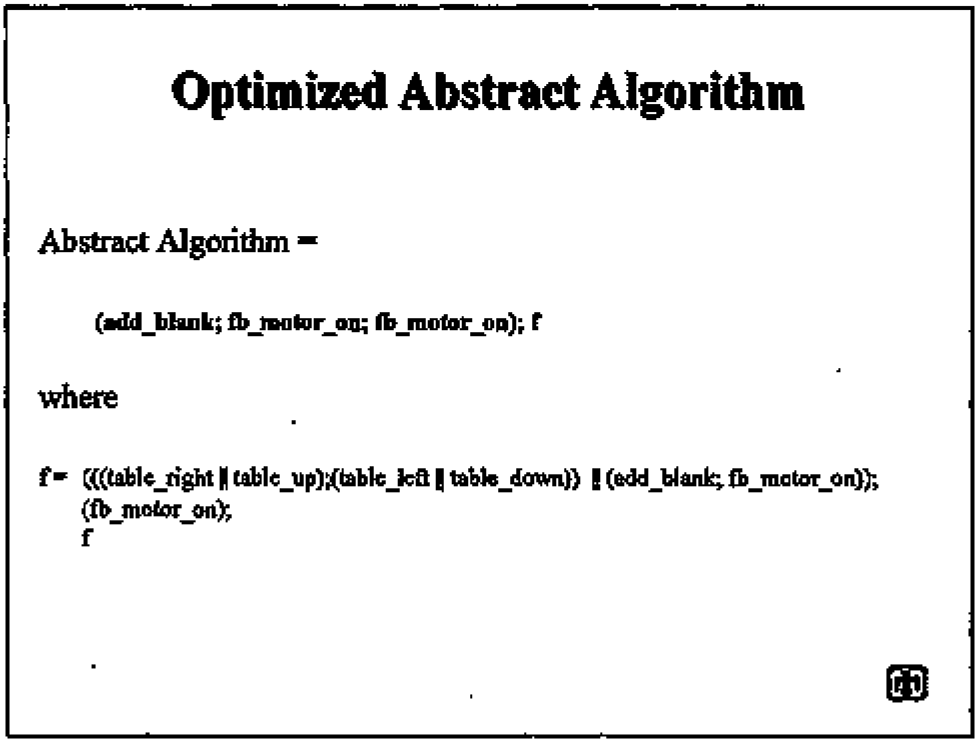
  (m1 || m2);m3
  ==>
  ((m1;m3) || m2)

- An Instantiation

```
((table_right || table_up) || (add_blank; fb_motor_on));
(table_left || table_down);

==>
(((table_right || table_up);(table_left || table_down)) || (add_blank; fb_motor_on));
```

Note that optimizations are starting to localize component behaviors!



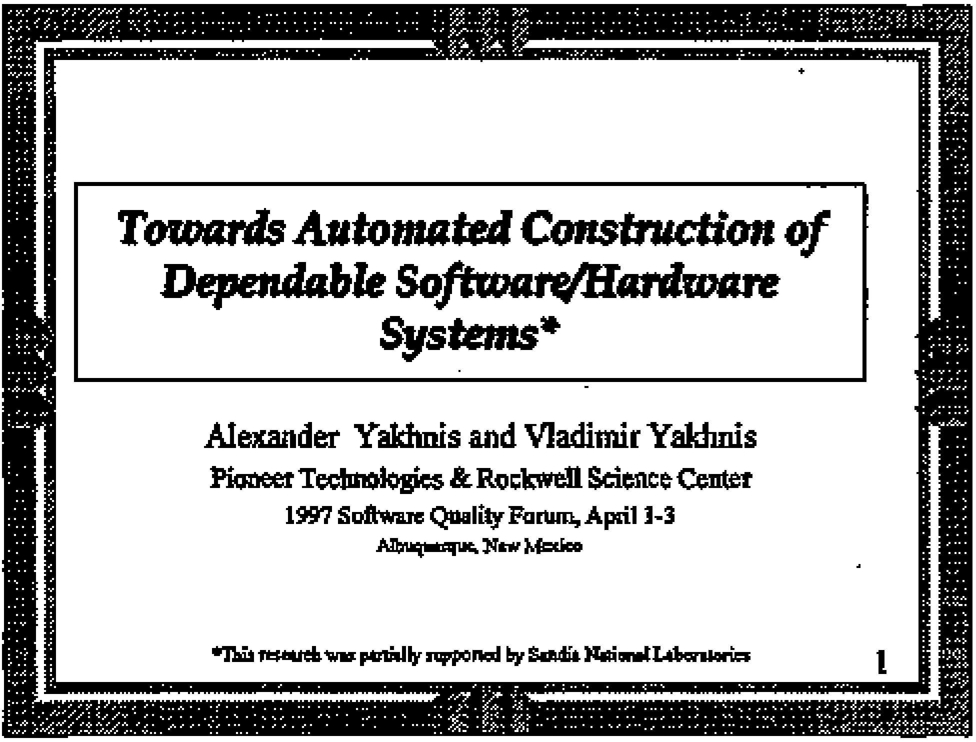
Optimized Abstract Algorithm

Abstract Algorithm =

```
    (add_blank; fb_motor_on; fb_motor_on); f
```

where

```
f =  (((table_right ∥ table_up);(table_left ∥ table_down)) ∥ (add_blank; fb_motor_on));
     (fb_motor_on);
     f
```



*Towards Automated Construction of Dependable Software/Hardware Systems**

Alexander Yakhnis and Vladimir Yakhnis

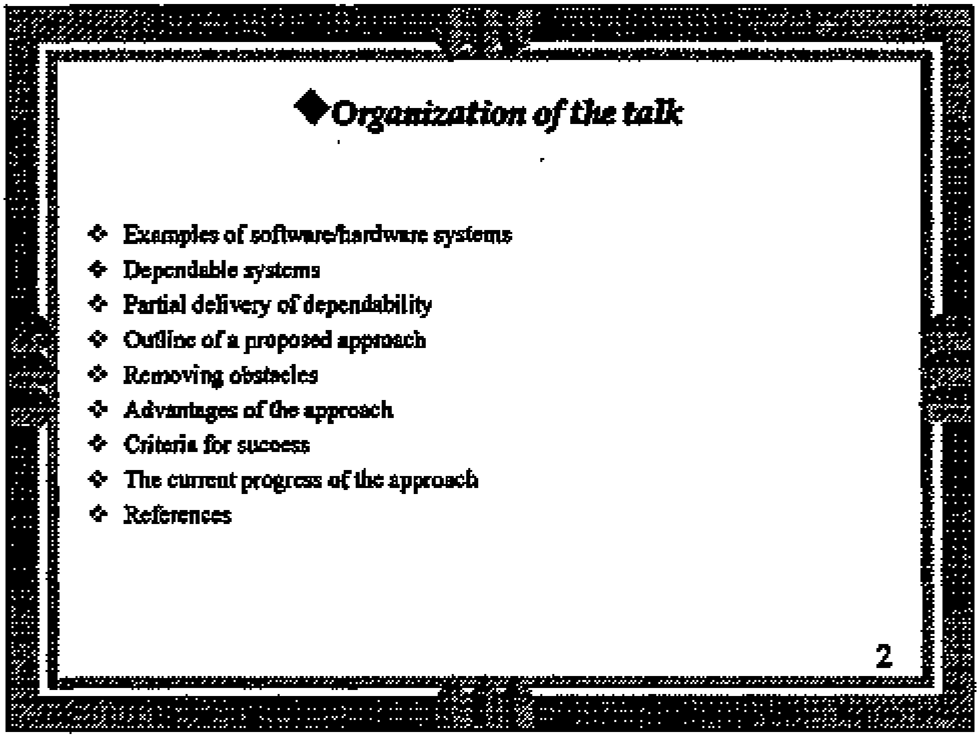
Pioneer Technologies & Rockwell Science Center

1997 Software Quality Forum, April 1-3

Albuquerque, New Mexico

*This research was partially supported by Sandia National Laboratories

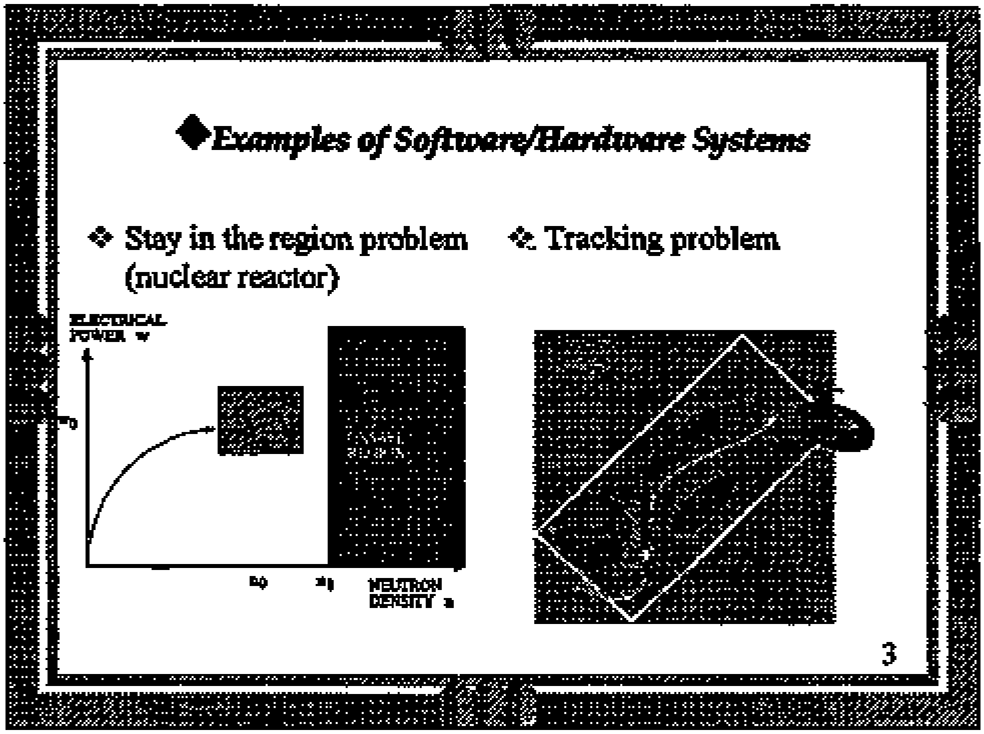
1



◆*Organization of the talk*

- ❖ Examples of software/hardware systems
- ❖ Dependable systems
- ❖ Partial delivery of dependability
- ❖ Outline of a proposed approach
- ❖ Removing obstacles
- ❖ Advantages of the approach
- ❖ Criteria for success
- ❖ The current progress of the approach
- ❖ References

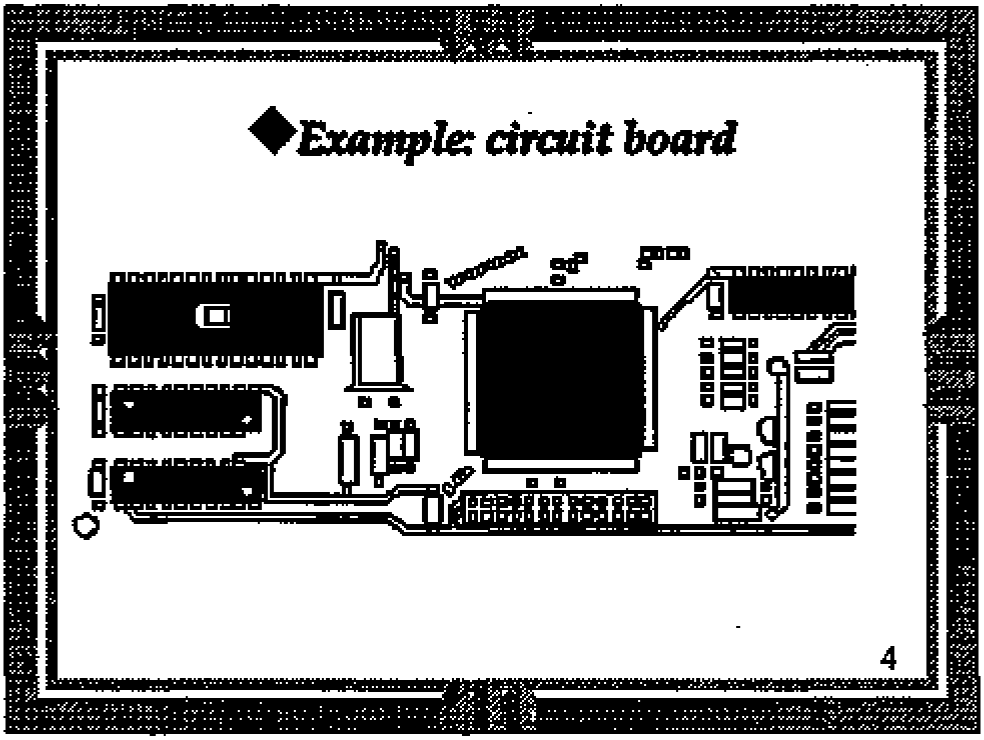
2



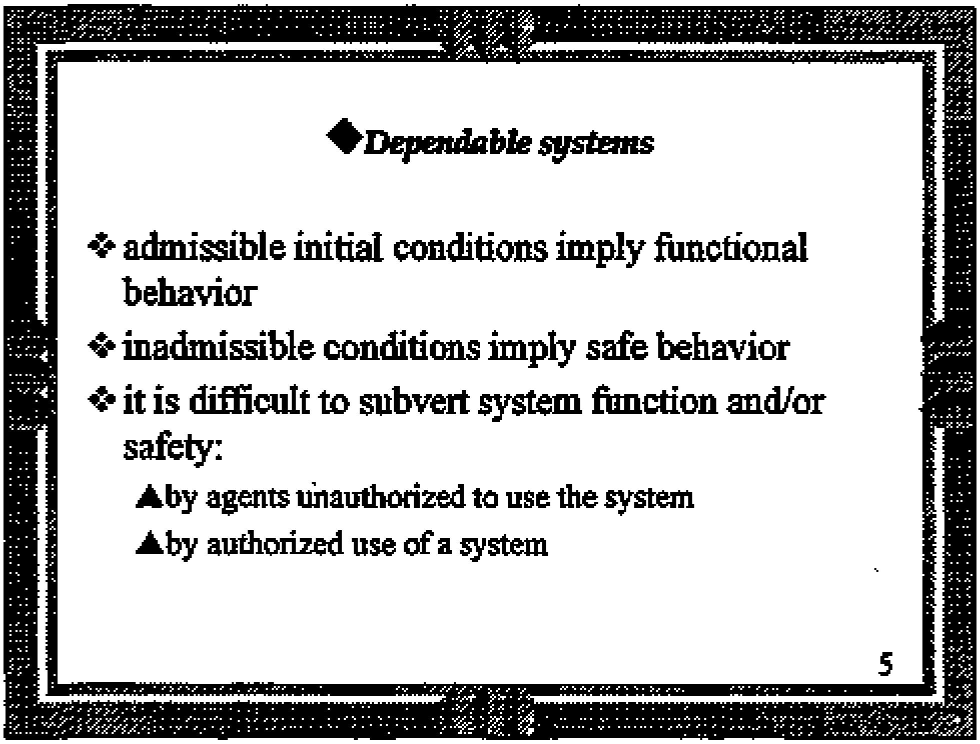
◆Examples of Software/Hardware Systems

❖ Stay in the region problem (nuclear reactor)

❖ Tracking problem

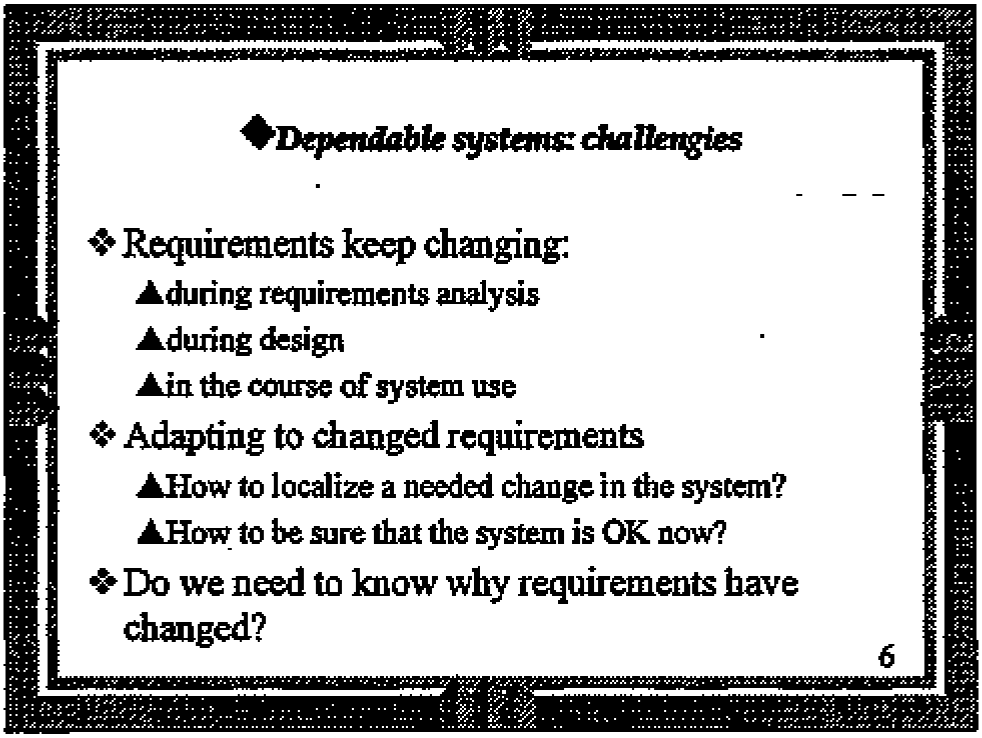


◆Example: circuit board



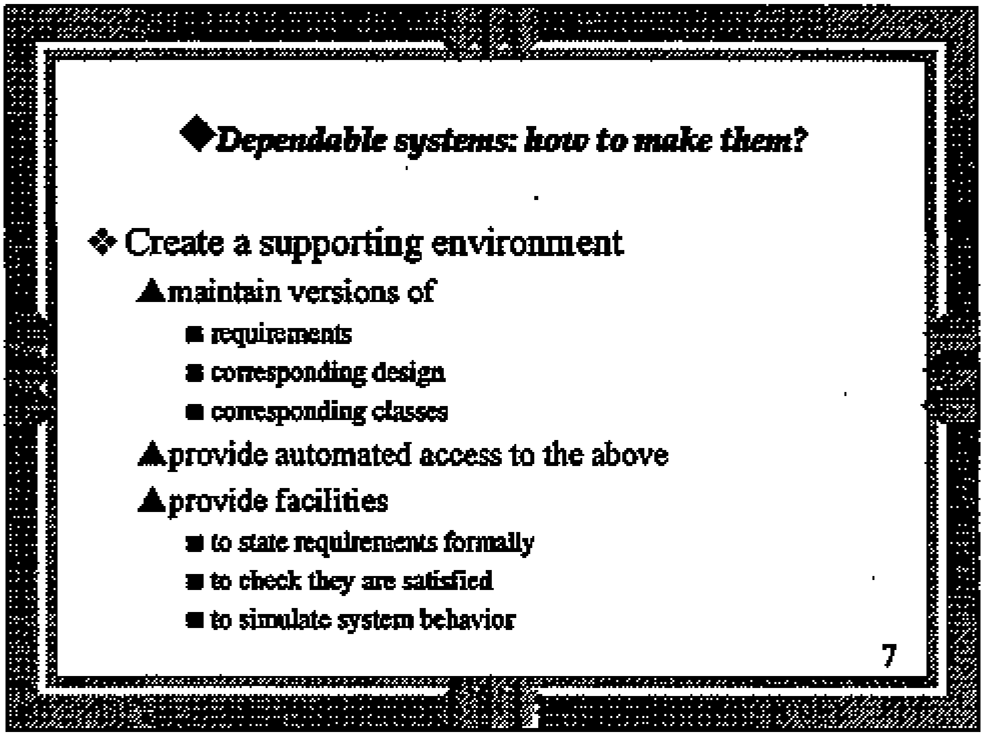
◆*Dependable systems*

- ❖ admissible initial conditions imply functional behavior
- ❖ inadmissible conditions imply safe behavior
- ❖ it is difficult to subvert system function and/or safety:
  - ▲by agents unauthorized to use the system
  - ▲by authorized use of a system



◆*Dependable systems: challengies*

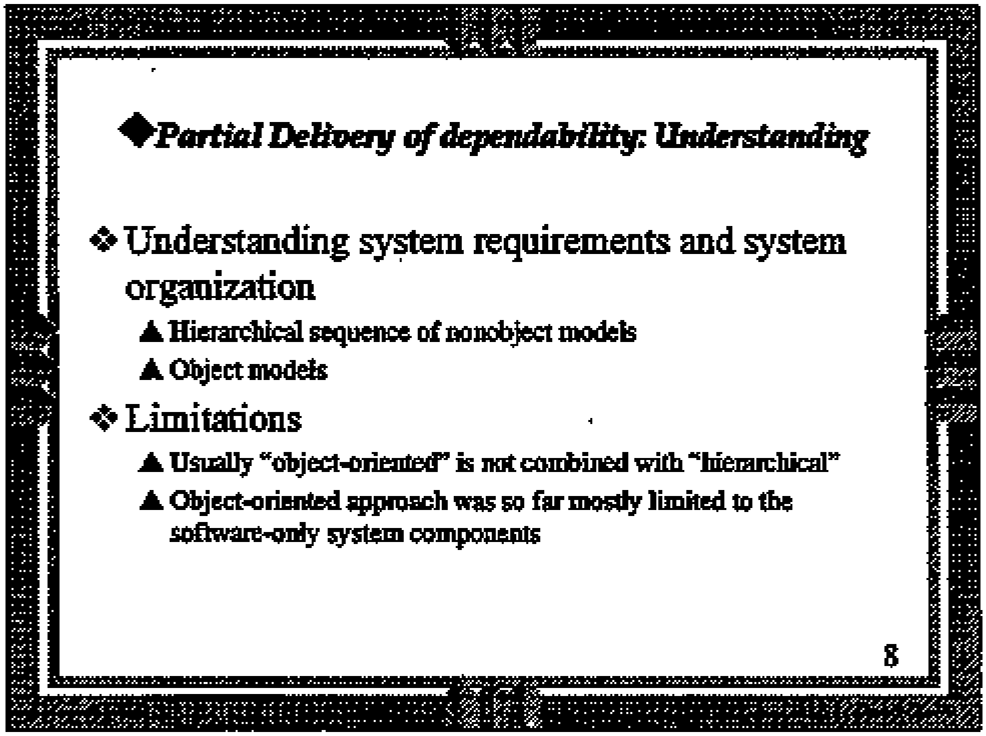
- ❖ Requirements keep changing:
  - ▲during requirements analysis
  - ▲during design
  - ▲in the course of system use
- ❖ Adapting to changed requirements
  - ▲How to localize a needed change in the system?
  - ▲How to be sure that the system is OK now?
- ❖ Do we need to know why requirements have changed?



◆*Dependable systems: how to make them?*

- ❖ Create a supporting environment
  - ▲maintain versions of
    - ■ requirements
    - ■ corresponding design
    - ■ corresponding classes
  - ▲provide automated access to the above
  - ▲provide facilities
    - ■ to state requirements formally
    - ■ to check they are satisfied
    - ■ to simulate system behavior

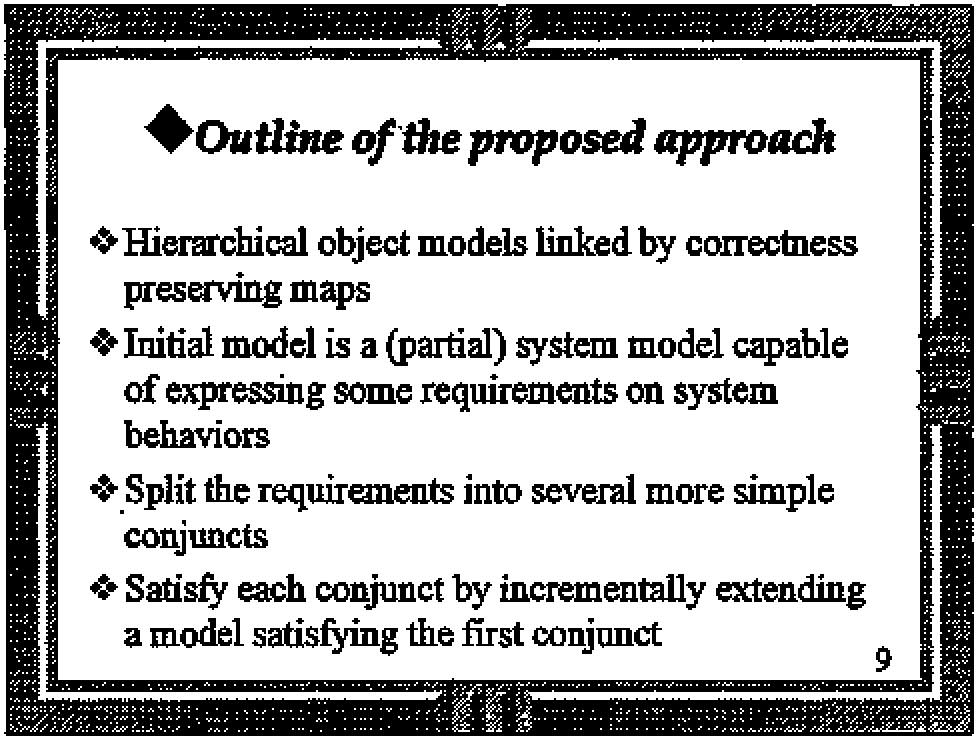
7



◆*Partial Delivery of dependability: Understanding*

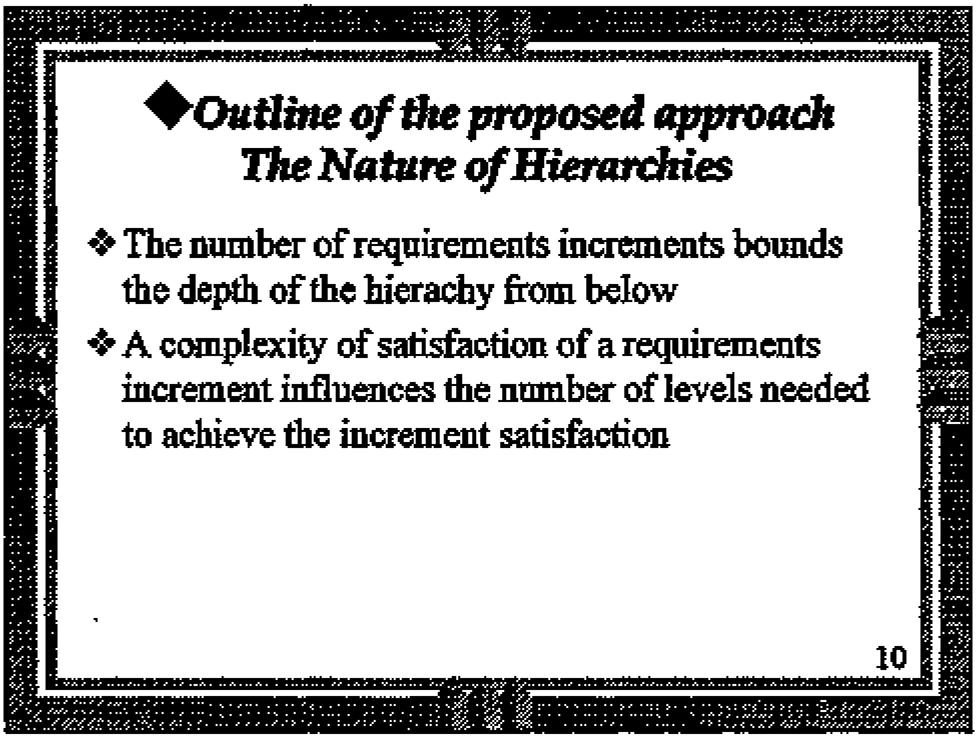
- ❖ Understanding system requirements and system organization
  - ▲ Hierarchical sequence of nonobject models
  - ▲ Object models
- ❖ Limitations
  - ▲ Usually "object-oriented" is not combined with "hierarchical"
  - ▲ Object-oriented approach was so far mostly limited to the software-only system components

8



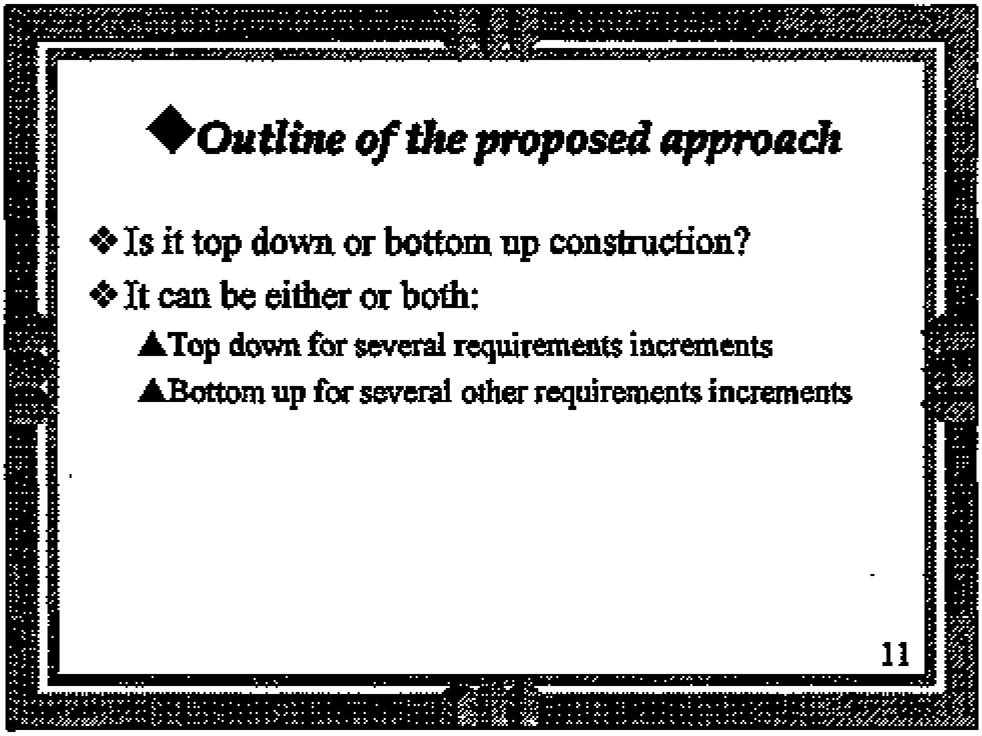
◆*Outline of the proposed approach*

- ❖ Hierarchical object models linked by correctness preserving maps
- ❖ Initial model is a (partial) system model capable of expressing some requirements on system behaviors
- ❖ Split the requirements into several more simple conjuncts
- ❖ Satisfy each conjunct by incrementally extending a model satisfying the first conjunct



◆*Outline of the proposed approach The Nature of Hierarchies*

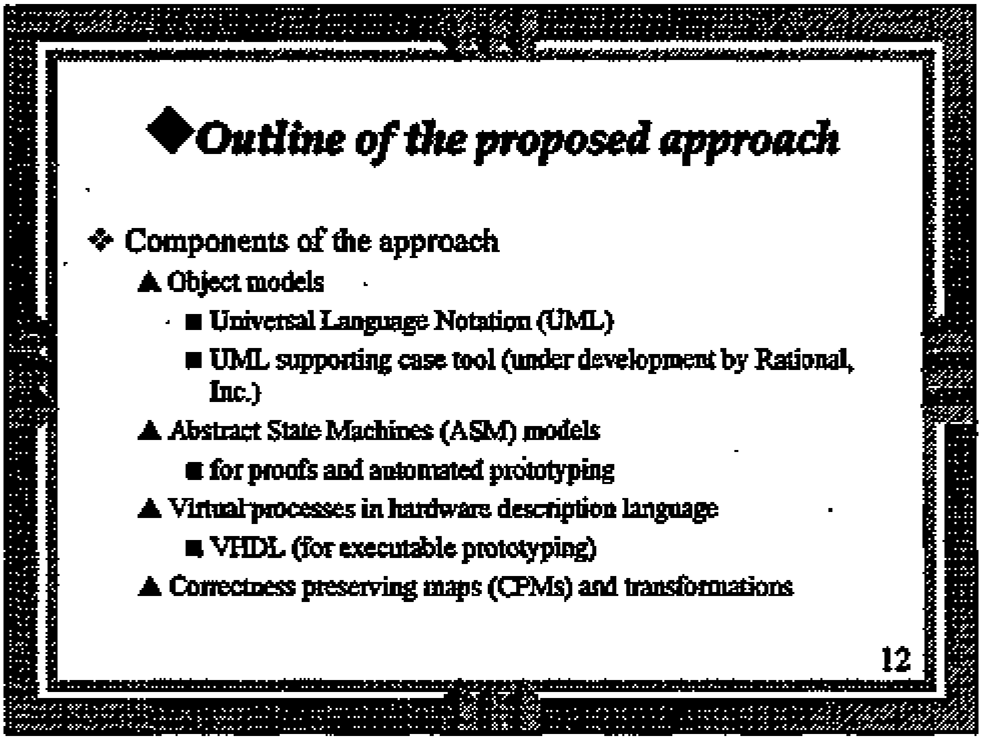
- ❖ The number of requirements increments bounds the depth of the hierachy from below
- ❖ A complexity of satisfaction of a requirements increment influences the number of levels needed to achieve the increment satisfaction



◆*Outline of the proposed approach*

- ❖ Is it top down or bottom up construction?
- ❖ It can be either or both:
  - ▲Top down for several requirements increments
  - ▲Bottom up for several other requirements increments

11



◆*Outline of the proposed approach*

- ❖ Components of the approach
  - ▲ Object models
    - ■ Universal Language Notation (UML)
    - ■ UML supporting case tool (under development by Rational, Inc.)
  - ▲ Abstract State Machines (ASM) models
    - ■ for proofs and automated prototyping
  - ▲ Virtual processes in hardware description language
    - ■ VHDL (for executable prototyping)
  - ▲ Correctness preserving maps (CPMs) and transformations

12

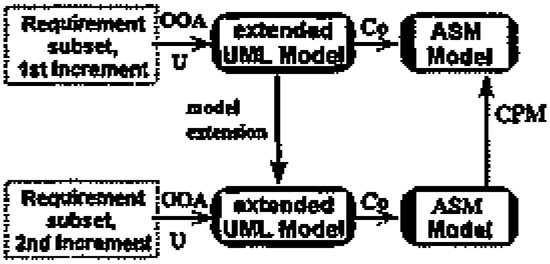
◆Outline of the proposed approach The automated tools

- Visualization tools for the approach
  - ▲ Industry animation tools
- Rigorous design tools
  - ▲ Deductive Synthesis
  - ▲ LG algorithms
  - ▲ Game-theoretic algorithms
  - ▲ Generic algorithms
- Rigorous Verification Tools adapted for partial functions
  - ▲ Formal specification languages: LARCH, Zed, Penelope
  - ▲ Software verifiers: Computational Logic, Inc., Otter
  - ▲ Other provers: PVS, HOL, LARCH PC
  - ▲ Game-throoretic based verifiers
- Validation Tool: VHDL Virtual Prototype

13

◆Outline of the proposed approach

System Modeling & Requirements Analysis



Co = compilation U = understanding

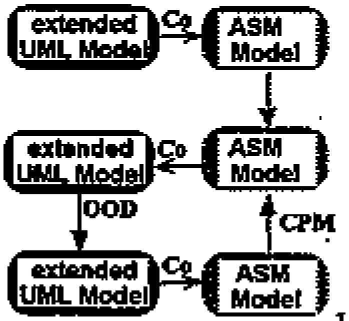
CPM = correctness preserving map UML = unified modeling language

ASM = abstract state machines OOA = object-oriented analysis

14

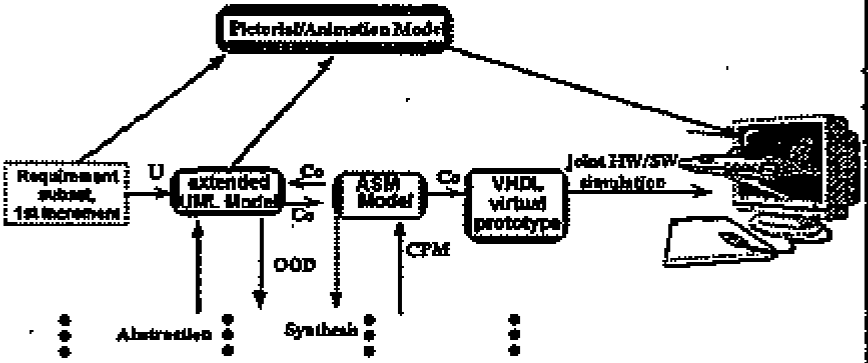
◆*Outline of the proposed approach*
Rigorous Design

- "Correct by construction" synthesis
  - "Winning strategies" via MAS
  - "Almost winning strategies" via Linguistic Geometry (LG)
  - Deductive synthesis (DS)
    - Proved refinement transformation (RT)
    - Generic algorithms
- "Formally verifiable" design
  - First design, then prove correctness by constructing correctness preserving maps (CPMs) between design levels

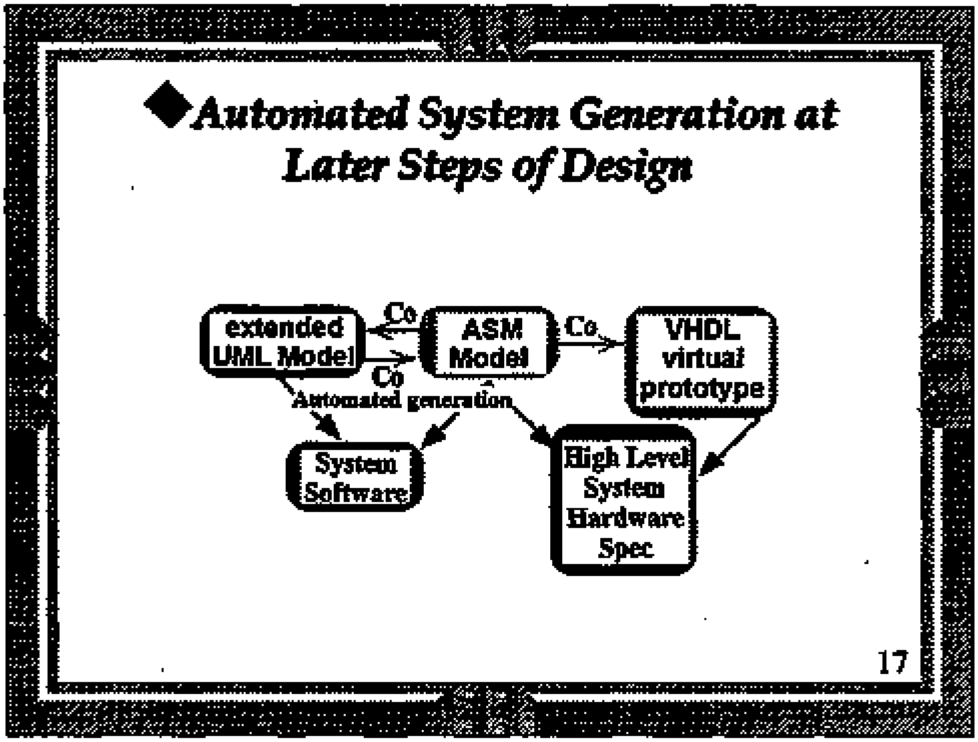


15

◆*Prototyping and Simulation in the Course of Design*

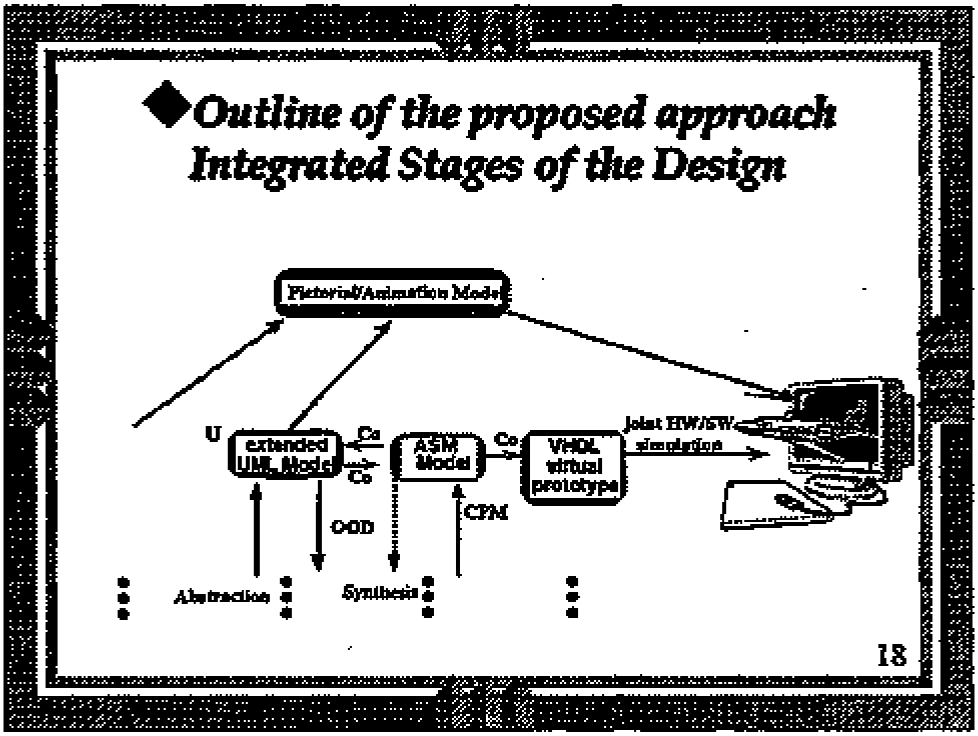


16



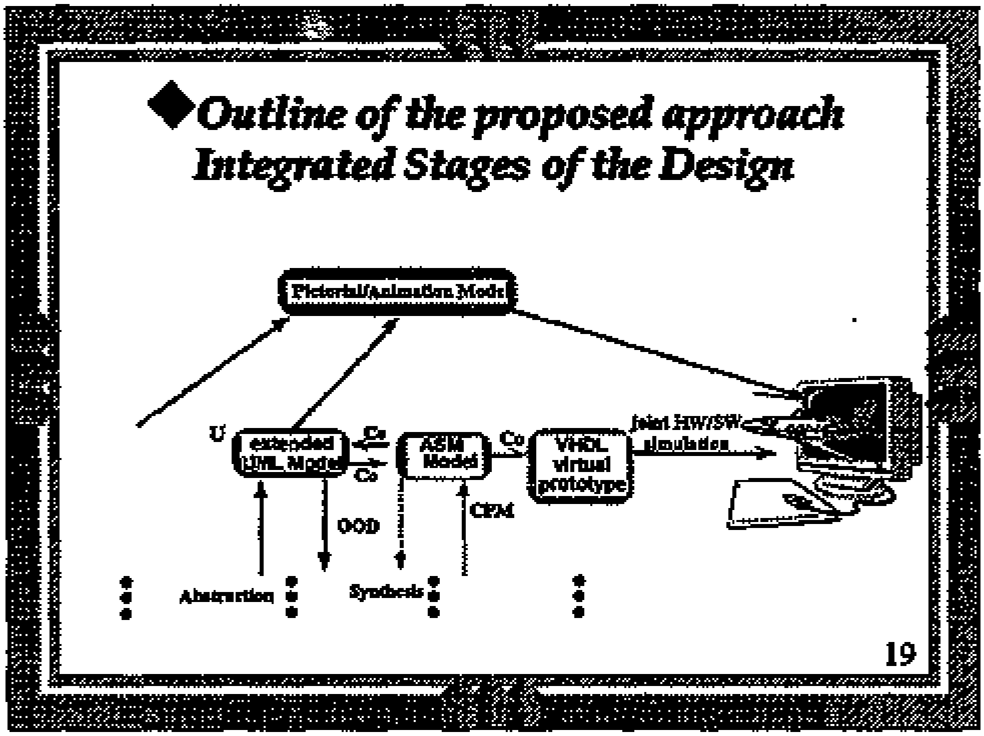
◆*Automated System Generation at Later Steps of Design*

17



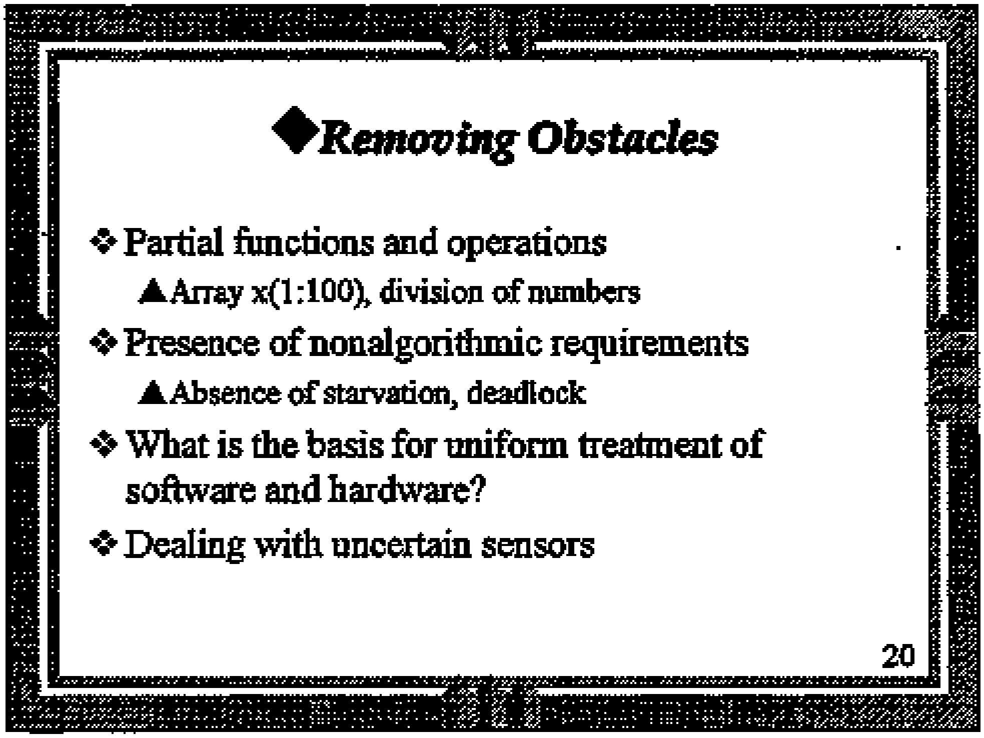
◆*Outline of the proposed approach Integrated Stages of the Design*

18



◆Outline of the proposed approach Integrated Stages of the Design

19



◆Removing Obstacles

- ❖ Partial functions and operations
  - ▲Array x(1:100), division of numbers
- ❖ Presence of nonalgorithmic requirements
  - ▲Absence of starvation, deadlock
- ❖ What is the basis for uniform treatment of software and hardware?
- ❖ Dealing with uncertain sensors

20

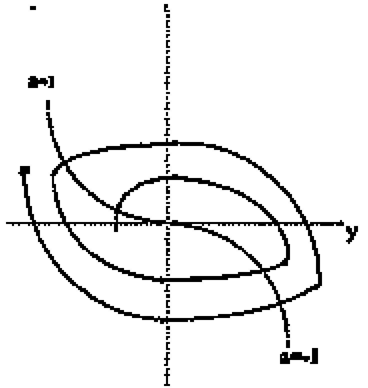
◆*A Control System with Uncertain Measurements*
Guaranteed Guidance to a Position

❖ Model vehicle dynamics

- ▲ Position x
- ▲ Velocity y
- ▲ Acceleration a=-1, 0 or 1
- ▲ Dynamics
  - ■ x'(t) =y(t)
  - ■ y'(t)=a

❖ Requirement

- ▲ Find a control strategy to reach the target at (0,0) from any point (x,y)
- ▲ using sensor with error bound e>0



Classical solution fails to reach the origin due to imprecise sensor measurements

21

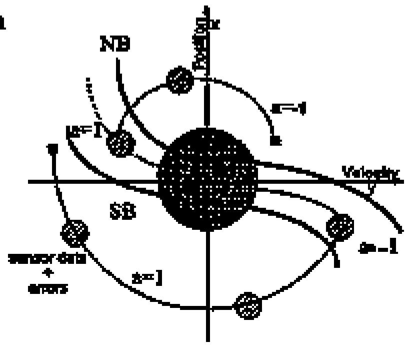
◆*A Control System with Uncertain Measurements*
Restating the Problem

❖ Reachability problem

- ▲ View a measured location p in the phase space as a disc (vs. a point)
- ▲ Golden disk: D(p, e)
- ▲-Place the golden disc completely inside the red disk D(0,r), r≥3e

❖ Stream= Bundle of all trajectories

- ▲ through disk D(0,r-2e)
- ▲ without switching



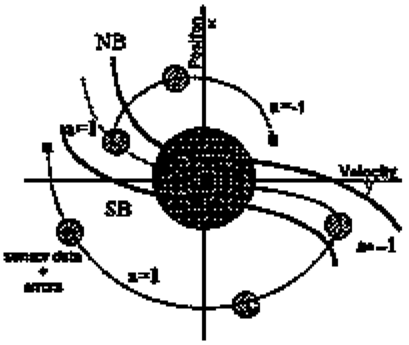
22

❖ Stream Boundaries

- ▲ South=SB

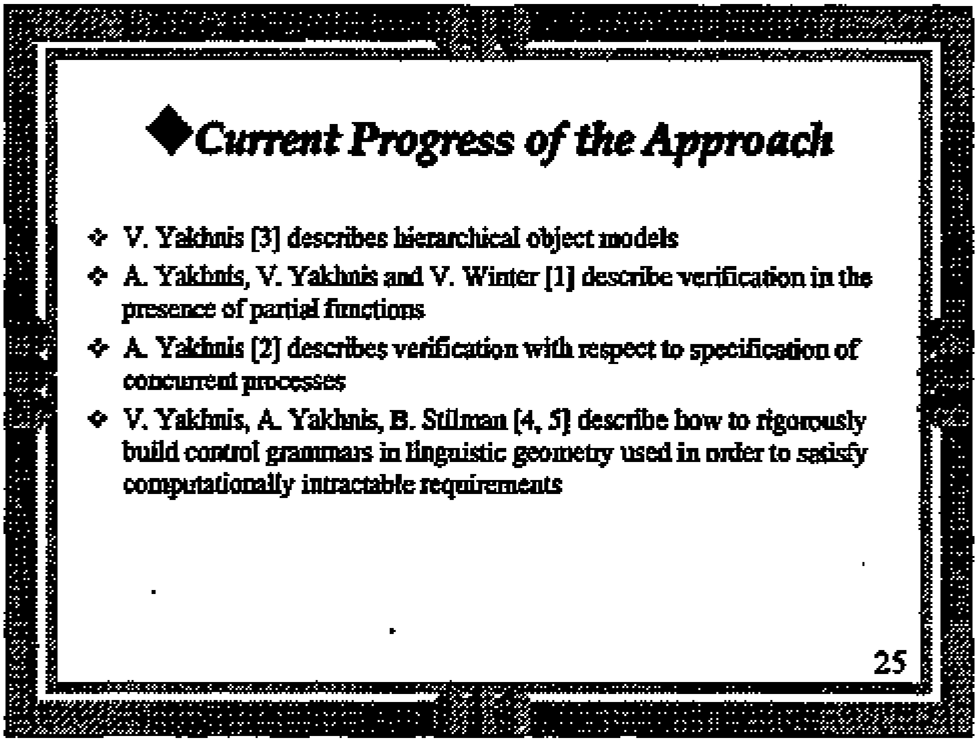
◆*A Control System with Uncertain Measurements Winning Control Strategy*

- ❖ If sensor disk is
  - ▲ below or
  - ▲ crosses SB or
  - ▲ in the West stream,
    - ■ set a to +1;
  - ▲ above or
  - ▲ crosses NB or
  - ▲ in the East stream,
    - ■ set a to -1;
- ❖ If a measurement
  - ▲ is in D(0,2e)
    - ■ the target is hit
- ❖ Lemma. The above



◆*Advantages of the Approach*

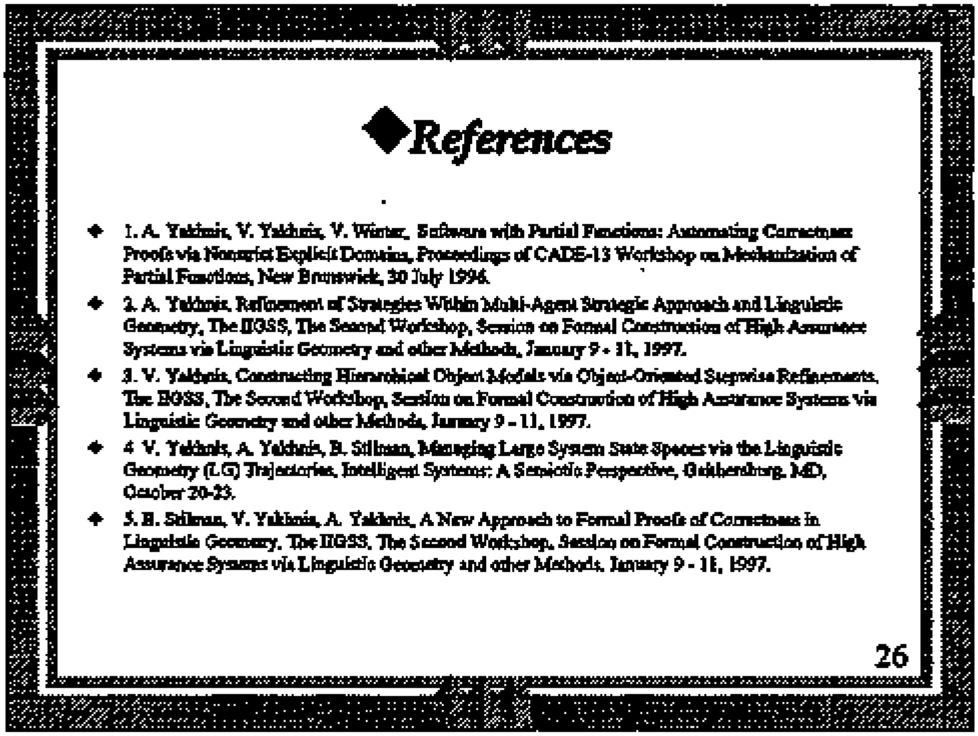
- ❖ A choice of hardware or software implementation may be postponed until later model design stages
- ❖ System prototypes and simulation of system behaviors are available at the earliest design levels and long before any hardware is built
- ❖ The approach provides a collection of parameters of confidence for system dependability



◆Current Progress of the Approach

- ❖ V. Yakhnis [3] describes hierarchical object models
- ❖ A. Yakhnis, V. Yakhnis and V. Winter [1] describe verification in the presence of partial functions
- ❖ A. Yakhnis [2] describes verification with respect to specification of concurrent processes
- ❖ V. Yakhnis, A. Yakhnis, B. Stilman [4, 5] describe how to rigorously build control grammars in linguistic geometry used in order to satisfy computationally intractable requirements

25



◆References

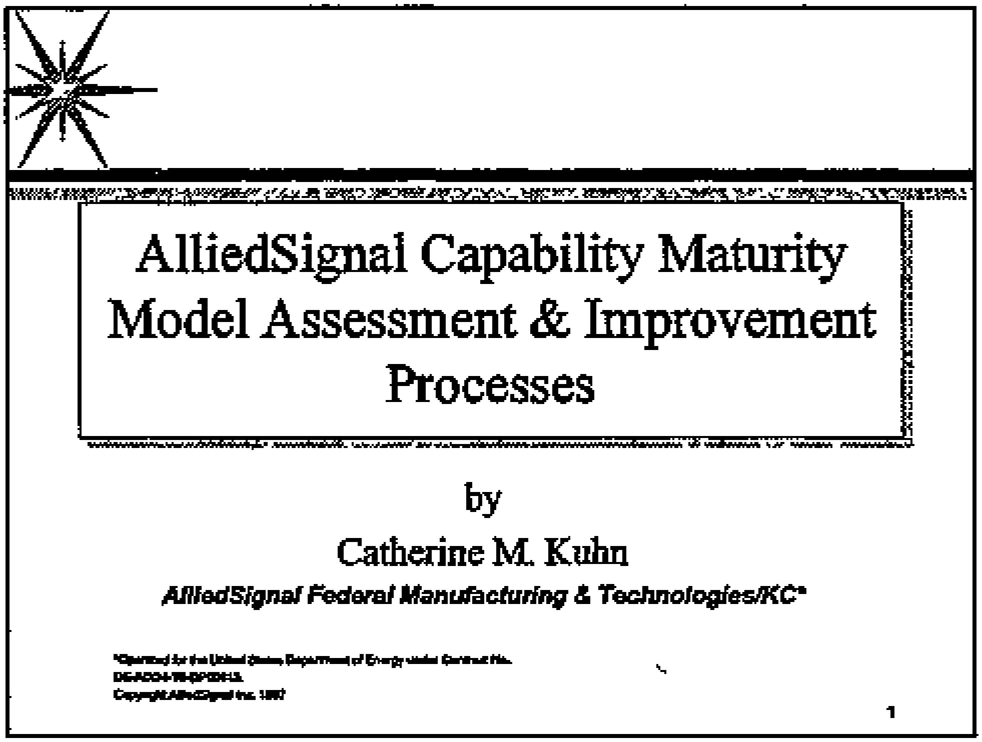
- ◆ 1. A. Yakhnis, V. Yakhnis, V. Winter, Software with Partial Functions: Automating Correctness Proofs via Nonstrict Explicit Domains, Proceedings of CADE-13 Workshop on Mechanization of Partial Functions, New Brunswick, 30 July 1996.
- ◆ 2. A. Yakhnis, Refinement of Strategies Within Multi-Agent Strategic Approach and Linguistic Geometry, The IIGSS, The Second Workshop, Session on Formal Construction of High Assurance Systems via Linguistic Geometry and other Methods, January 9 - 11, 1997.
- ◆ 3. V. Yakhnis, Constructing Hierarchical Object Models via Object-Oriented Stepwise Refinements, The IIGSS, The Second Workshop, Session on Formal Construction of High Assurance Systems via Linguistic Geometry and other Methods, January 9 - 11, 1997.
- ◆ 4 V. Yakhnis, A. Yakhnis, B. Stilman, Managing Large System State Spaces via the Linguistic Geometry (LG) Trajectories, Intelligent Systems: A Semiotic Perspective, Gaithersburg, MD, October 20-23.
- ◆ 5. B. Stilman, V. Yakhnis, A. Yakhnis, A New Approach to Formal Proofs of Correctness in Linguistic Geometry, The IIGSS, The Second Workshop, Session on Formal Construction of High Assurance Systems via Linguistic Geometry and other Methods, January 9 - 11, 1997.

26

Session A4: Software Process Improvement II

Chair John Hare
AWE United Kingdom

Session : Paper #	Author(s)	*Title*
A4:1	Cathy Kuhn AS/FM&T	*AlliedSignal Capability Maturity Model Assessment & Improvement Processes*
A4:2	Ann Stewart Lockheed Martin Energy Systems	*Lessons Learned on Utilizing the SEI/CMM in the Federal Government Work for Others Environment*
A4:3	Gail Benefield Lockheed Martin Energy Systems	*"SWiM" Your Way to Software Quality*



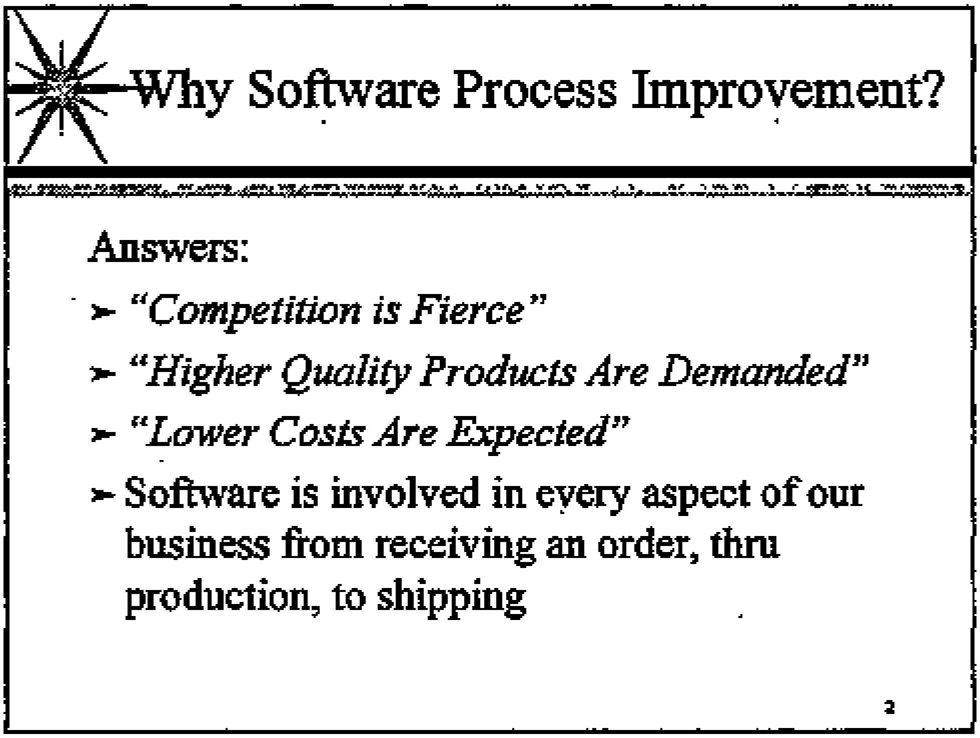
AlliedSignal Capability Maturity Model Assessment & Improvement Processes

by

Catherine M. Kuhn
*AlliedSignal Federal Manufacturing & Technologies/KC**

*Operated for the United States Department of Energy under Contract No. DE-AC04-76-DP00613.

1

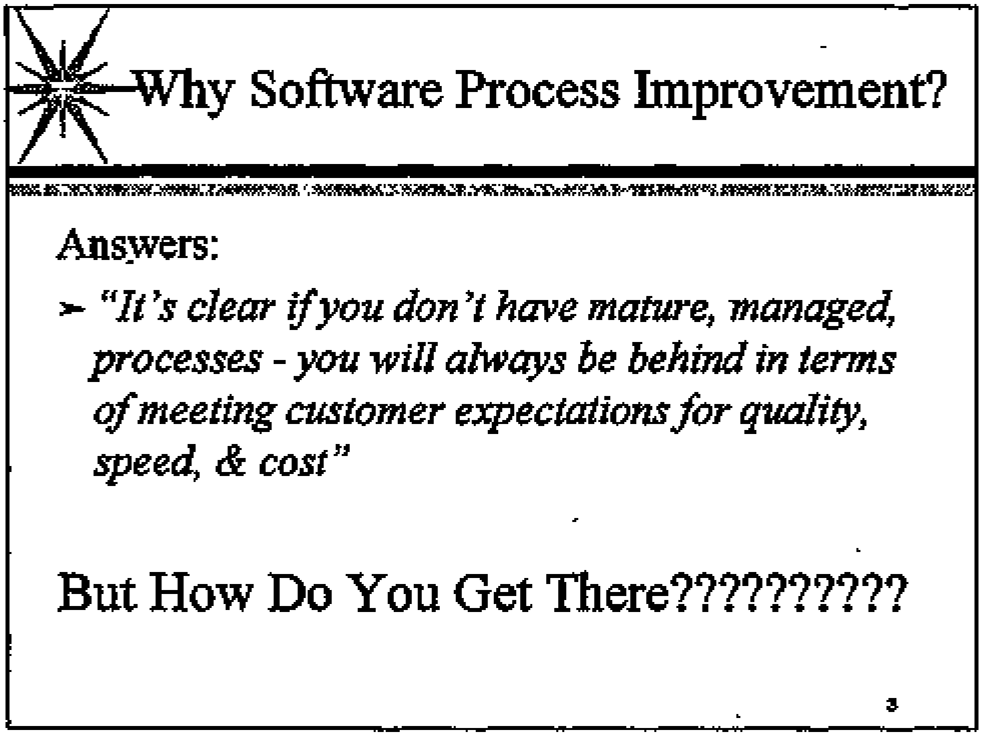


Why Software Process Improvement?

Answers:

- ➢ *"Competition is Fierce"*
- ➢ *"Higher Quality Products Are Demanded"*
- ➢ *"Lower Costs Are Expected"*
- ➢ Software is involved in every aspect of our business from receiving an order, thru production, to shipping

2



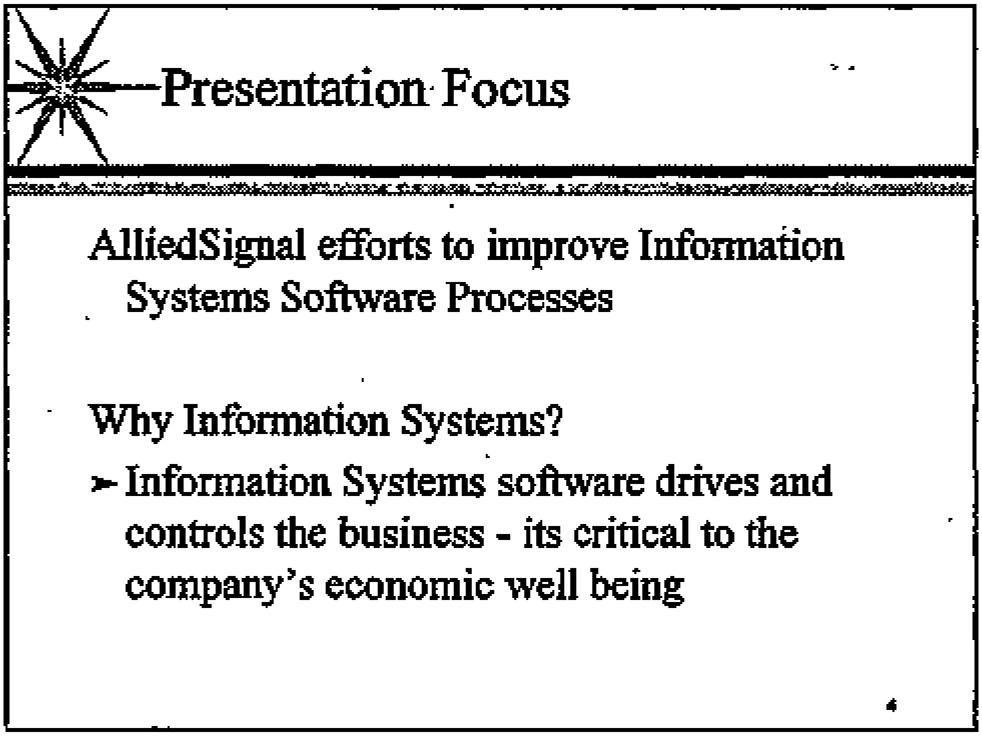
Why Software Process Improvement?

Answers:

- *"It's clear if you don't have mature, managed, processes - you will always be behind in terms of meeting customer expectations for quality, speed, & cost"*

But How Do You Get There??????????

3



Presentation Focus

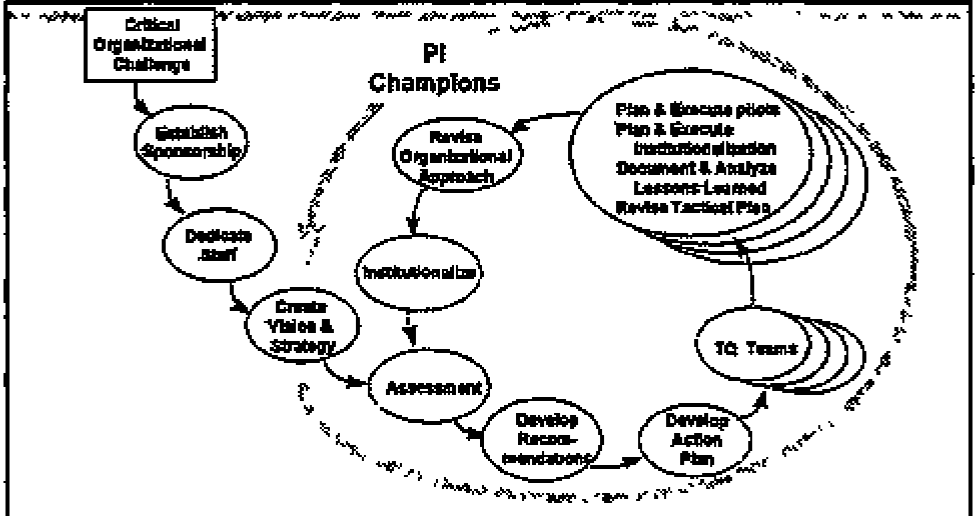
AlliedSignal efforts to improve Information Systems Software Processes

Why Information Systems?

- Information Systems software drives and controls the business - its critical to the company's economic well being

4

The AlliedSignal Process Improvement Cycle



5

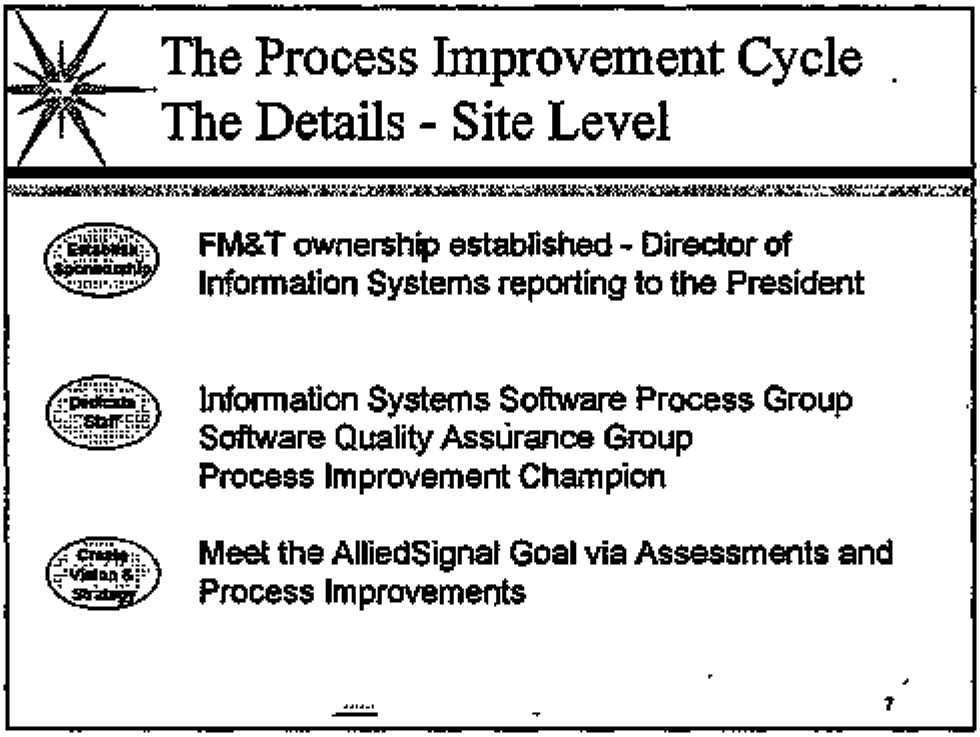
The Process Improvement Cycle The Details - Corporate Level

Critical Organizational Challenge — All major IS Sites (16) will achieve Software Engineering Institute(SEI) Capability Maturity Model (CMM) Level 2 by October 1, 1997

Establish Sponsorship — AlliedSignal ownership established - Corporate Information Officer reporting to the President

Dedicate Staff — Staff assigned to
•Train
•Assess
•Help

6


Establish Sponsorship
Dedicate Staff
Create Vision & Strategy

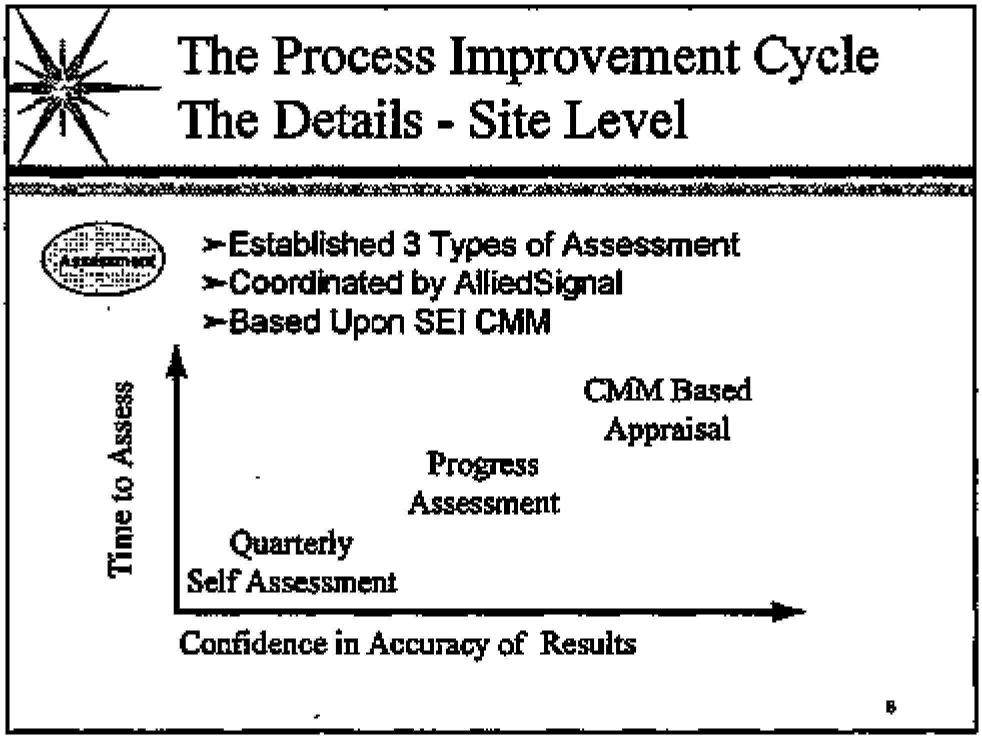
The Process Improvement Cycle
The Details - Site Level

FM&T ownership established - Director of Information Systems reporting to the President

Information Systems Software Process Group
Software Quality Assurance Group
Process Improvement Champion

Meet the AlliedSignal Goal via Assessments and Process Improvements

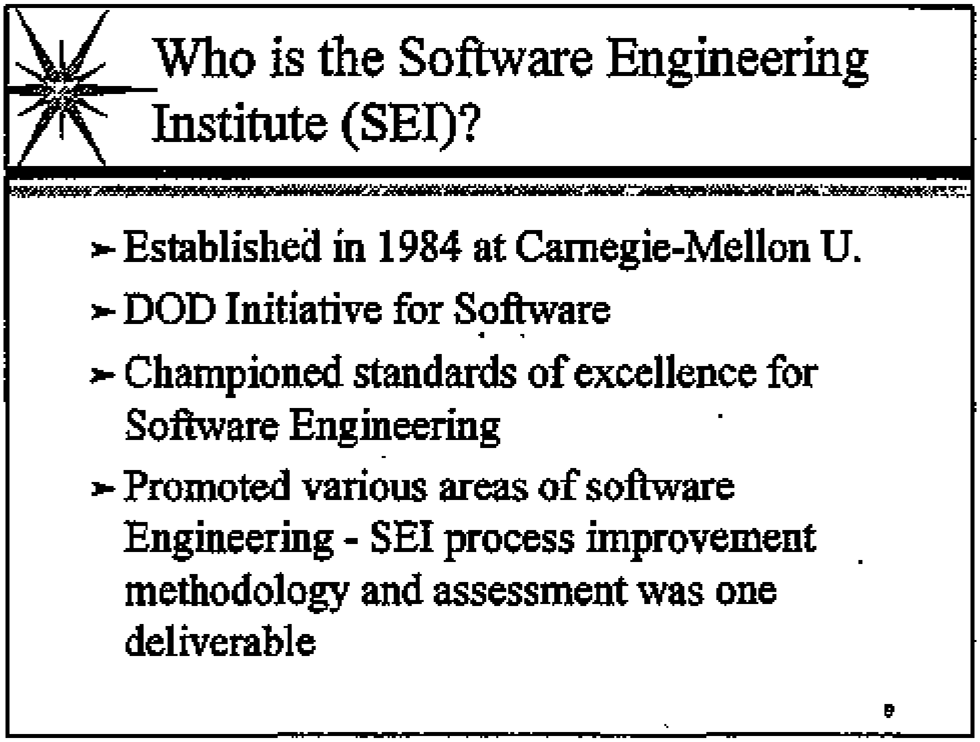
7


Assessment
Time to Assess
CMM Based Appraisal
Progress Assessment
Quarterly Self Assessment
Confidence in Accuracy of Results

The Process Improvement Cycle
The Details - Site Level

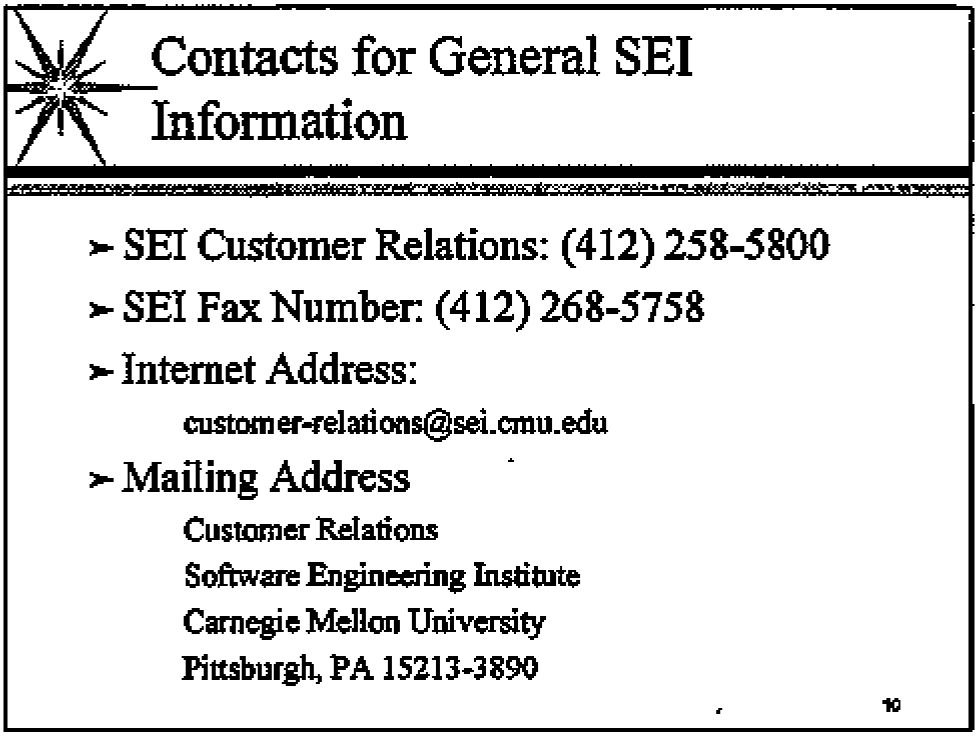
- ➢Established 3 Types of Assessment
- ➢Coordinated by AlliedSignal
- ➢Based Upon SEI CMM

8



Who is the Software Engineering Institute (SEI)?

- Established in 1984 at Carnegie-Mellon U.
- DOD Initiative for Software
- Championed standards of excellence for Software Engineering
- Promoted various areas of software Engineering - SEI process improvement methodology and assessment was one deliverable



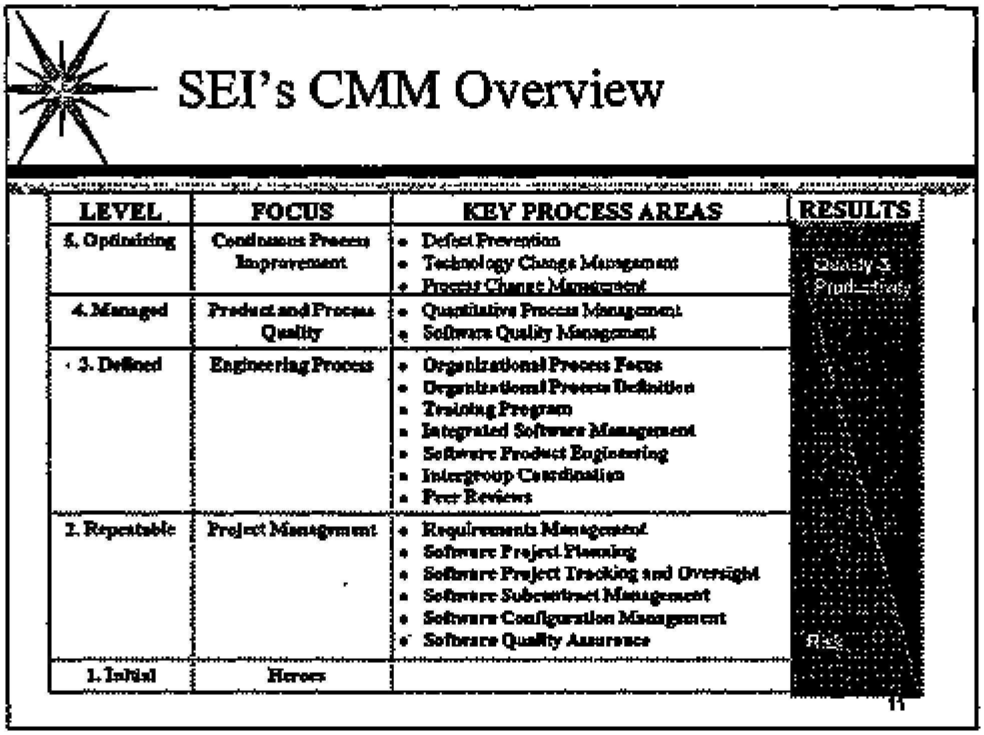
Contacts for General SEI Information

- SEI Customer Relations: (412) 258-5800
- SEI Fax Number: (412) 268-5758
- Internet Address:

  customer-relations@sei.cmu.edu

- Mailing Address

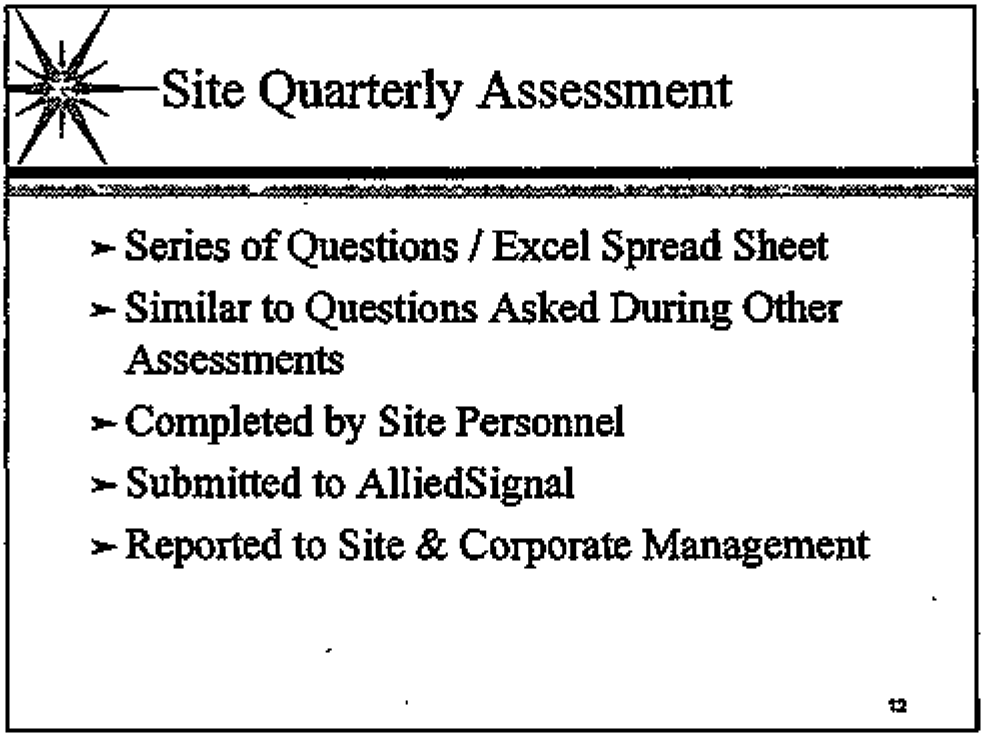
  Customer Relations
  Software Engineering Institute
  Carnegie Mellon University
  Pittsburgh, PA 15213-3890



SEI's CMM Overview

LEVEL	FOCUS	KEY PROCESS AREAS	RESULTS
5. Optimizing	Continuous Process Improvement	• Defect Prevention • Technology Change Management • Process Change Management	Quality & Productivity
4. Managed	Product and Process Quality	• Quantitative Process Management • Software Quality Management	
3. Defined	Engineering Process	• Organizational Process Focus • Organizational Process Definition • Training Program • Integrated Software Management • Software Product Engineering • Intergroup Coordination • Peer Reviews	
2. Repeatable	Project Management	• Requirements Management • Software Project Planning • Software Project Tracking and Oversight • Software Subcontract Management • Software Configuration Management • Software Quality Assurance	Risk
1. Initial	Heroes		

11



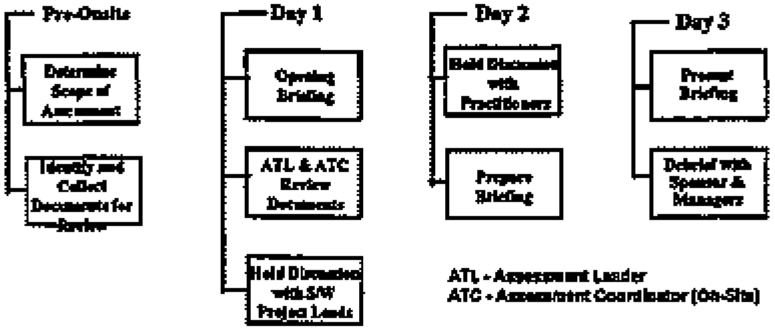
Site Quarterly Assessment

- ➢ Series of Questions / Excel Spread Sheet
- ➢ Similar to Questions Asked During Other Assessments
- ➢ Completed by Site Personnel
- ➢ Submitted to AlliedSignal
- ➢ Reported to Site & Corporate Management

12

Progress Assessment

- Led by AlliedSignal Trained Assessors
- Conducted Every 6-8 months
- Used to Guide Site Quarterly Assessment



Pre-Onsite
- Determine Scope of Assessment
- Identify and Collect Documents for Review

Day 1
- Opening Briefing
- ATL & ATC Review Documents
- Hold Discussion with S/W Project Leads

Day 2
- Hold Discussion with Practitioners
- Prepare Briefing

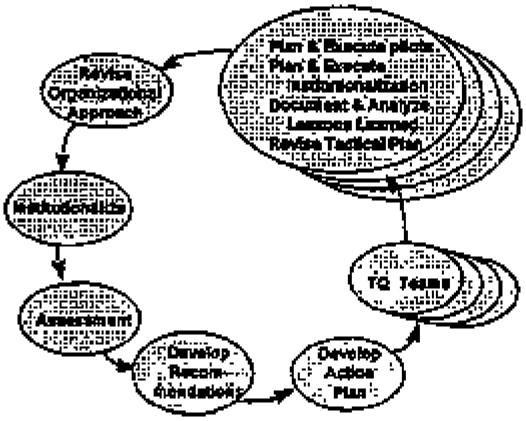
Day 3
- Present Briefing
- Debrief with Sponsor & Managers

ATL - Assessment Leader
ATC - Assessment Coordinator (On-Site)

CMM Based Appraisals

- Key Sites Will have CMM Based Appraisals
- Will be a formal CMM appraisal
- Will Only Be Done If a Business Need Exists
- Very Time Consuming

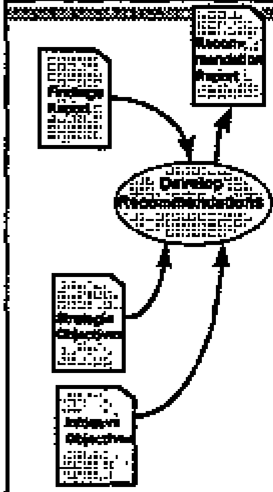
Process Improvement



RAPID - Recommendation, Action Planning, and Improvement Definition Process Guide

- Designed to improve the process of producing Action Plans after an assessment has been performed at a site

Rapid Process
Develop Recommendations

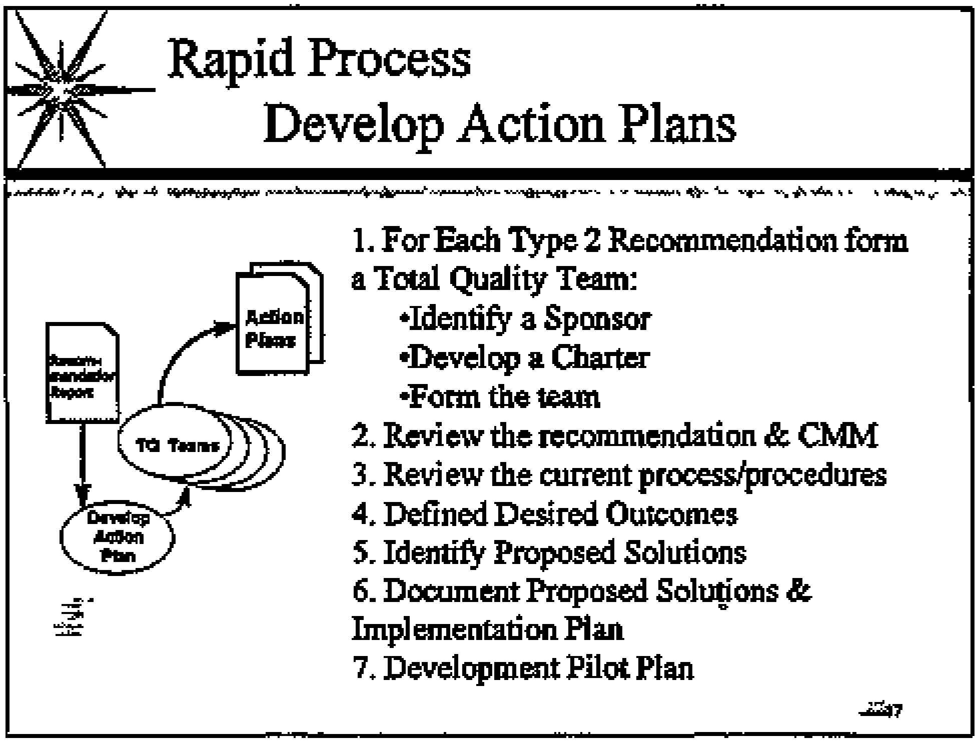


1. Prioritize on Importance & Classify Assessment Findings

Type	Definition	Action
1	Obvious Solution	Assign to Individual
2	Needs Further Investigation	Give to Recommendation Team
3	Out of Scope	Give to Management

2. For Type 2 findings Develop, Prioritize, Classify Recommendations & Report Results

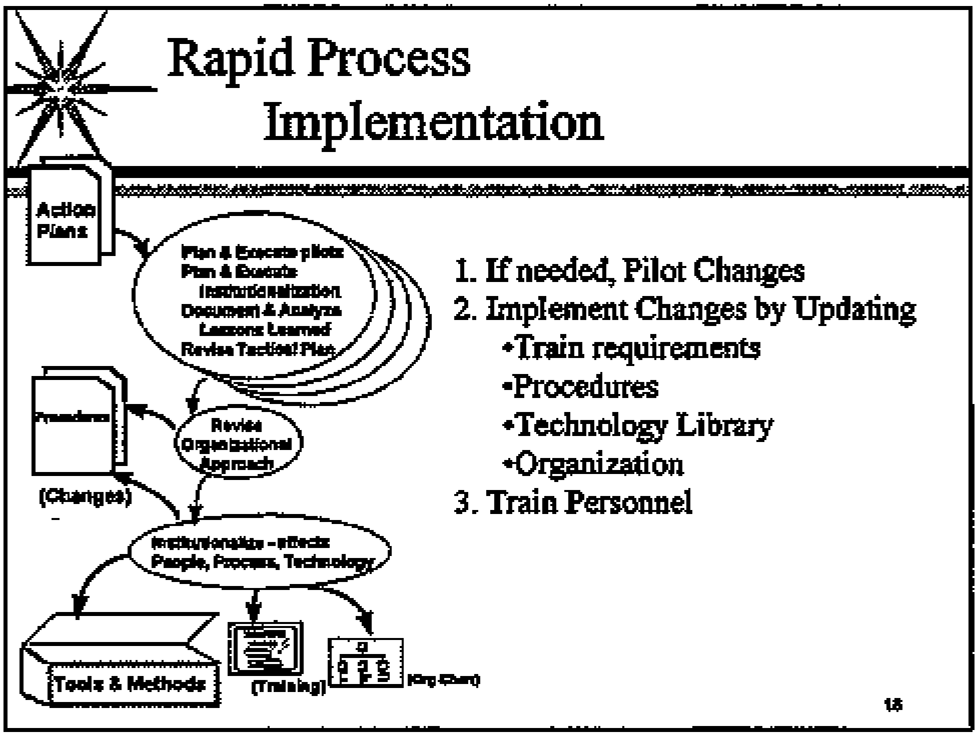
Type 2 Recommendations Continue on in the process



Rapid Process
Develop Action Plans

1. For Each Type 2 Recommendation form a Total Quality Team:
   - •Identify a Sponsor
   - •Develop a Charter
   - •Form the team
2. Review the recommendation & CMM
3. Review the current process/procedures
4. Defined Desired Outcomes
5. Identify Proposed Solutions
6. Document Proposed Solutions & Implementation Plan
7. Development Pilot Plan

17

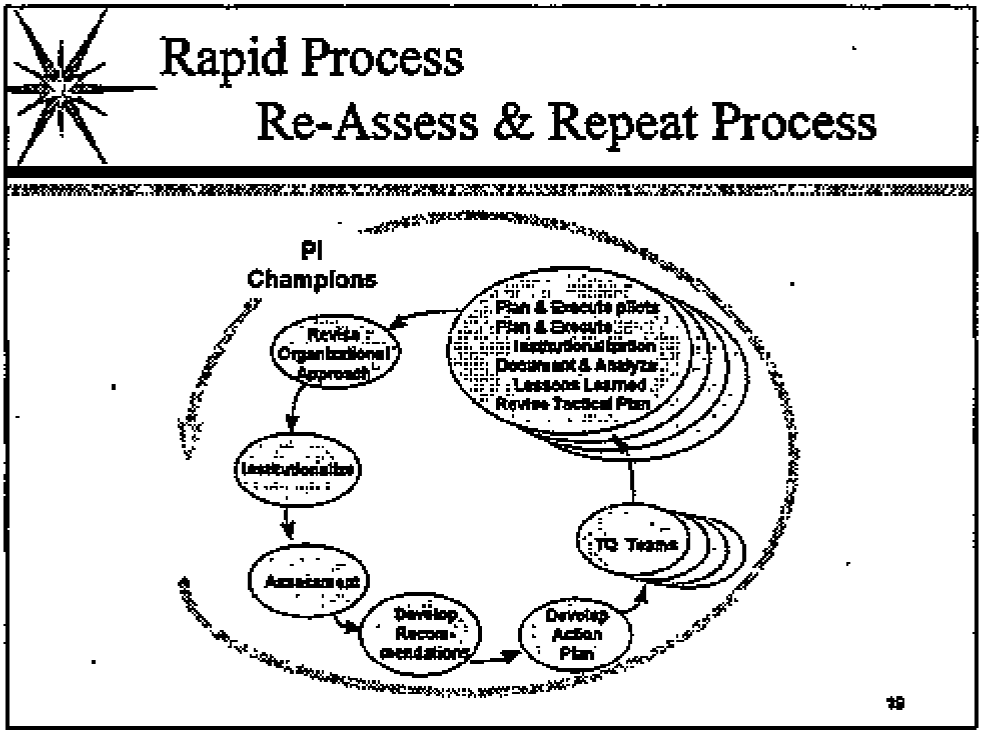


Rapid Process
Implementation

Action Plans
Plan & Execute pilots
Plan & Execute Institutionalization
Document & Analyze Lessons Learned
Revise Tactical Plan
Procedures
(Changes)
Revise Organizational Approach
Institutionalize - effects People, Process, Technology
Tools & Methods
(Training)
(Org Chart)

1. If needed, Pilot Changes
2. Implement Changes by Updating
   - •Train requirements
   - •Procedures
   - •Technology Library
   - •Organization
3. Train Personnel

18



Rapid Process
Re-Assess & Repeat Process

PI
Champions

Revise Organizational Approach

Institutionalize

Assessment

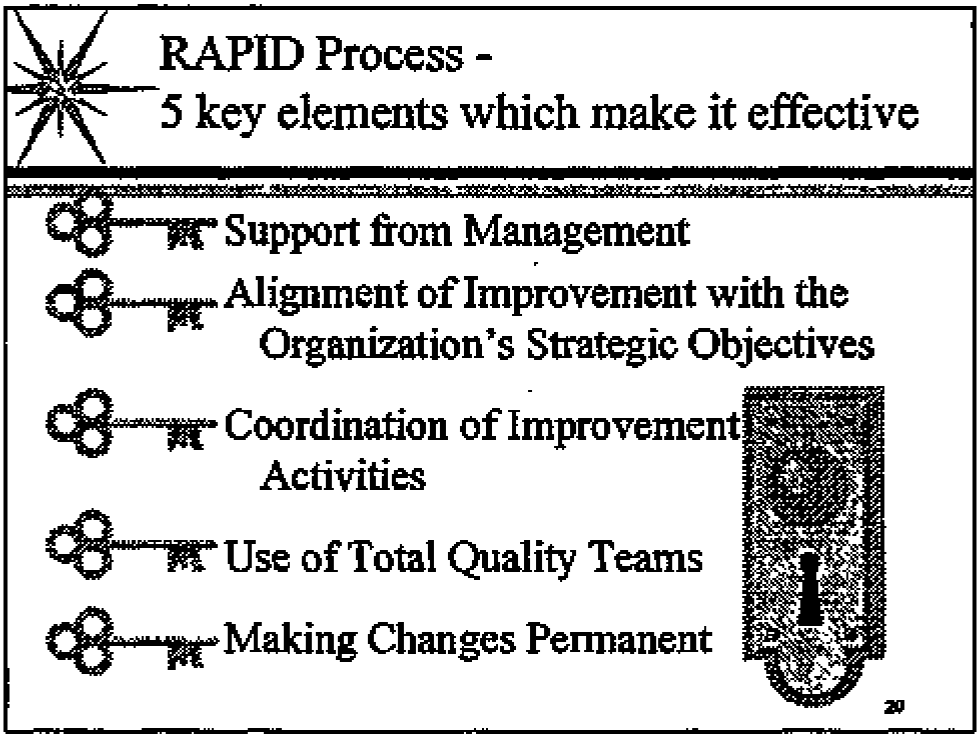
Develop Recommendations

Develop Action Plan

TQ Teams

Plan & Execute pilots
Plan & Execute Institutionalization
Document & Analyze Lessons Learned
Revise Tactical Plan

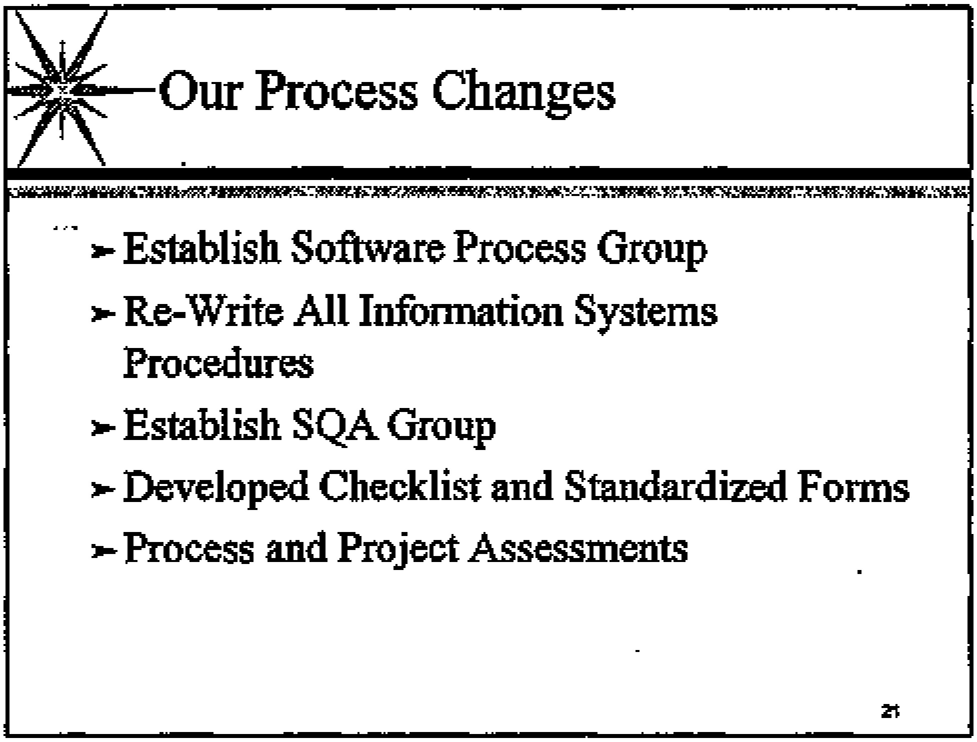
19



RAPID Process -
5 key elements which make it effective

- Support from Management
- Alignment of Improvement with the Organization's Strategic Objectives
- Coordination of Improvement Activities
- Use of Total Quality Teams
- Making Changes Permanent

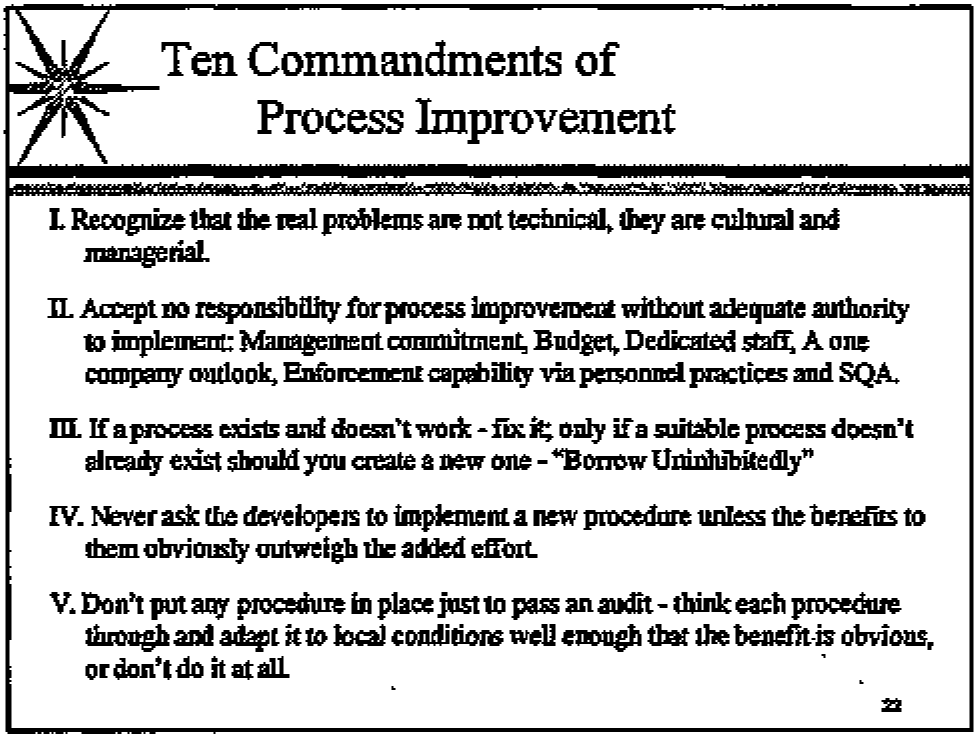
20



Our Process Changes

- ➢ Establish Software Process Group
- ➢ Re-Write All Information Systems Procedures
- ➢ Establish SQA Group
- ➢ Developed Checklist and Standardized Forms
- ➢ Process and Project Assessments

21



Ten Commandments of Process Improvement

I. Recognize that the real problems are not technical, they are cultural and managerial.

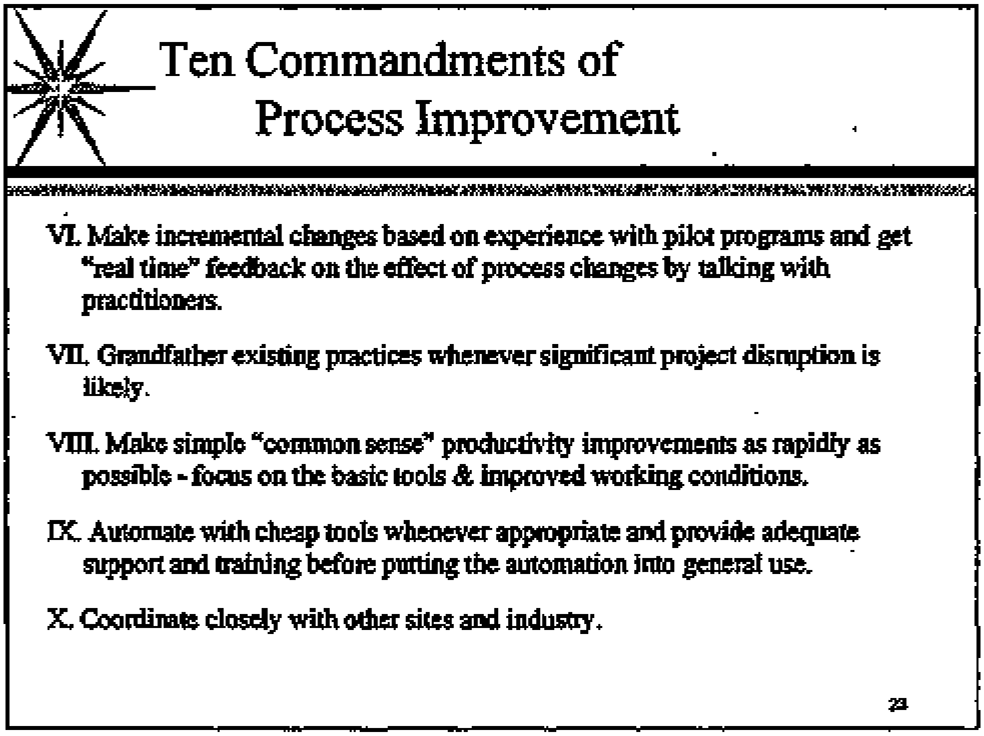
II. Accept no responsibility for process improvement without adequate authority to implement: Management commitment, Budget, Dedicated staff, A one company outlook, Enforcement capability via personnel practices and SQA.

III. If a process exists and doesn't work - fix it; only if a suitable process doesn't already exist should you create a new one - "Borrow Uninhibitedly"

IV. Never ask the developers to implement a new procedure unless the benefits to them obviously outweigh the added effort.

V. Don't put any procedure in place just to pass an audit - think each procedure through and adapt it to local conditions well enough that the benefit is obvious, or don't do it at all.

22



Ten Commandments of Process Improvement

VI. Make incremental changes based on experience with pilot programs and get "real time" feedback on the effect of process changes by talking with practitioners.

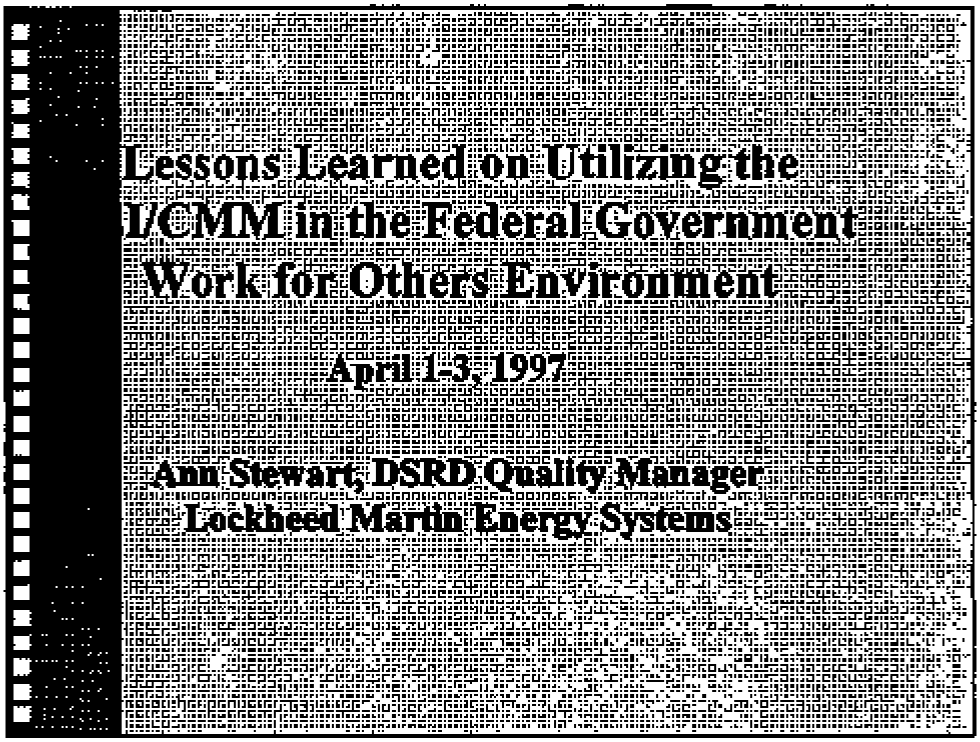
VII. Grandfather existing practices whenever significant project disruption is likely.

VIII. Make simple "common sense" productivity improvements as rapidly as possible - focus on the basic tools & improved working conditions.

IX. Automate with cheap tools whenever appropriate and provide adequate support and training before putting the automation into general use.

X. Coordinate closely with other sites and industry.

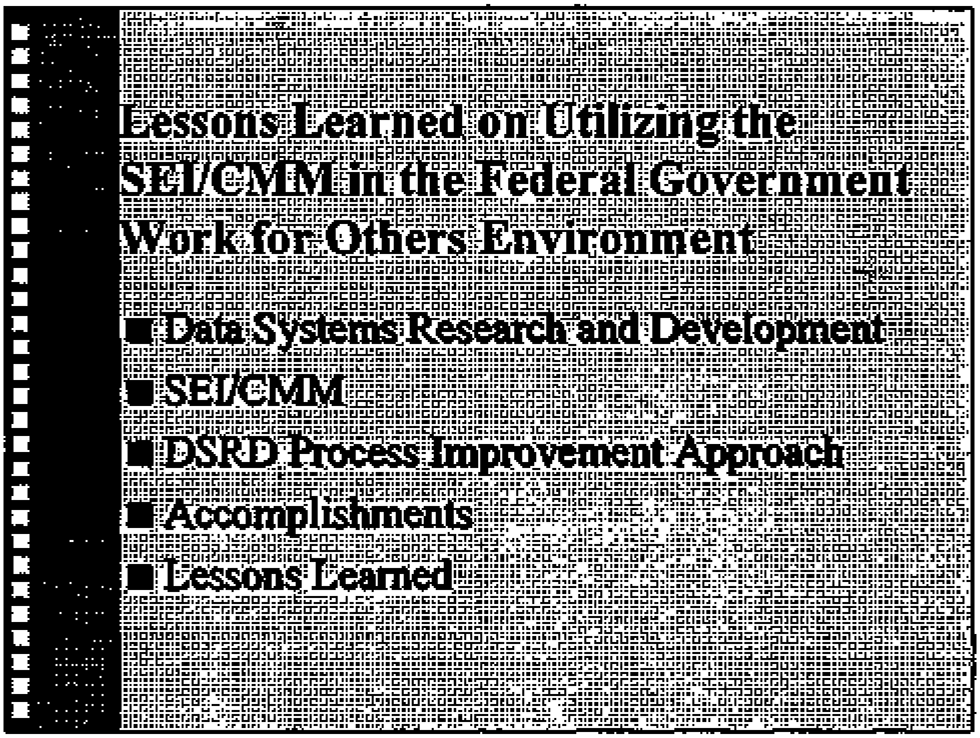
23



Lessons Learned on Utilizing the I/CMM in the Federal Government Work for Others Environment

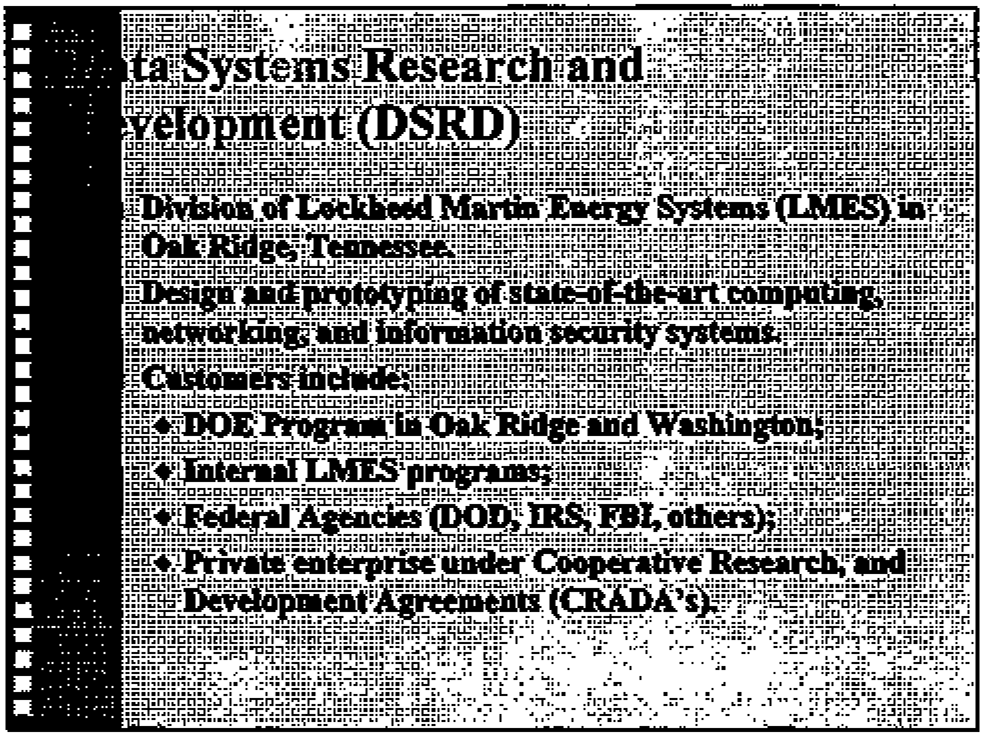
April 1-3, 1997

Ann Stewart, DSRD Quality Manager
Lockheed Martin Energy Systems



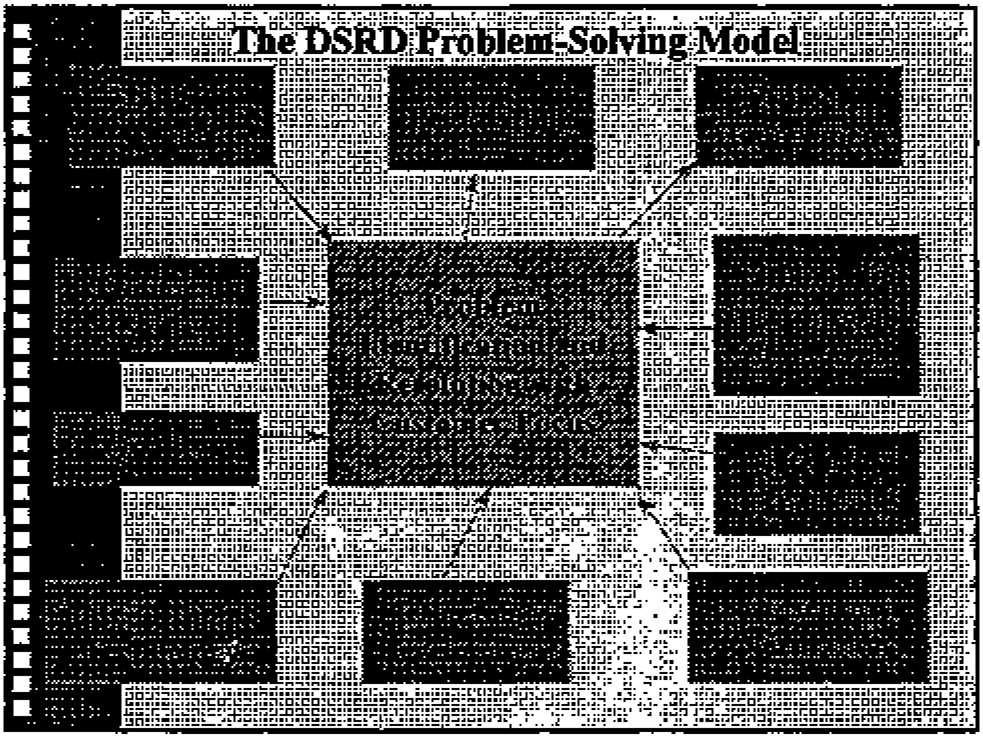
Lessons Learned on Utilizing the SEI/CMM in the Federal Government Work for Others Environment

- Data Systems Research and Development
- SEI/CMM
- DSRD Process Improvement Approach
- Accomplishments
- Lessons Learned

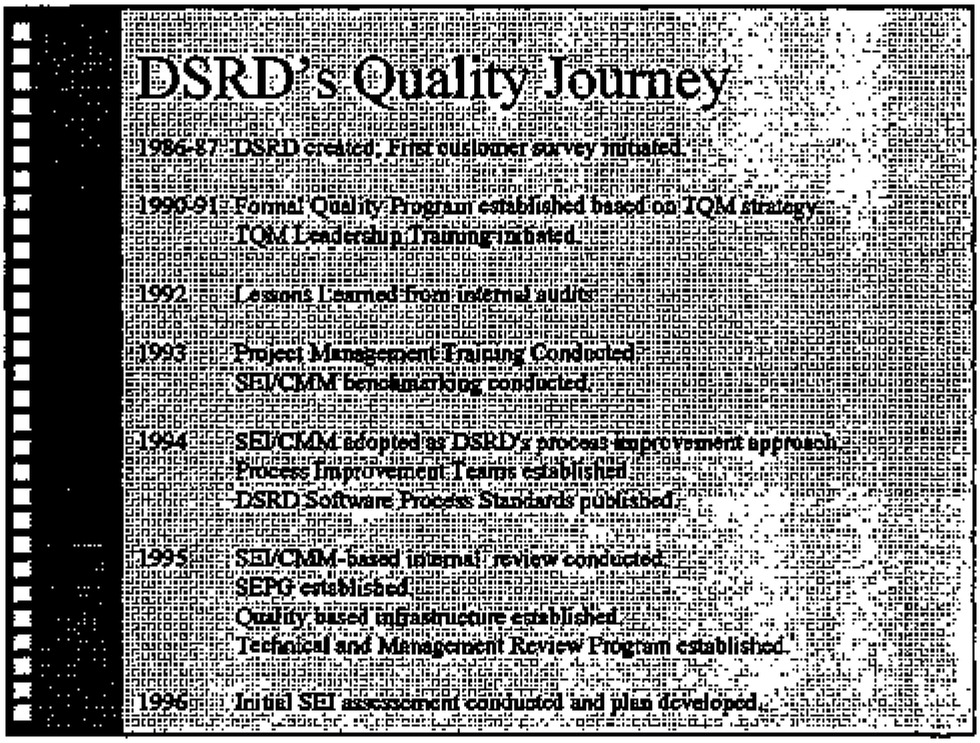


ta Systems Research and velopment (DSRD)

- Division of Lockheed Martin Energy Systems (LMES) in Oak Ridge, Tennessee.
- Design and prototyping of state-of-the-art computing, networking, and information security systems.
- Customers include:
  - DOE Program in Oak Ridge and Washington;
  - Internal LMES programs;
  - Federal Agencies (DOD, IRS, FBI, others);
  - Private enterprise under Cooperative Research, and Development Agreements (CRADA's).

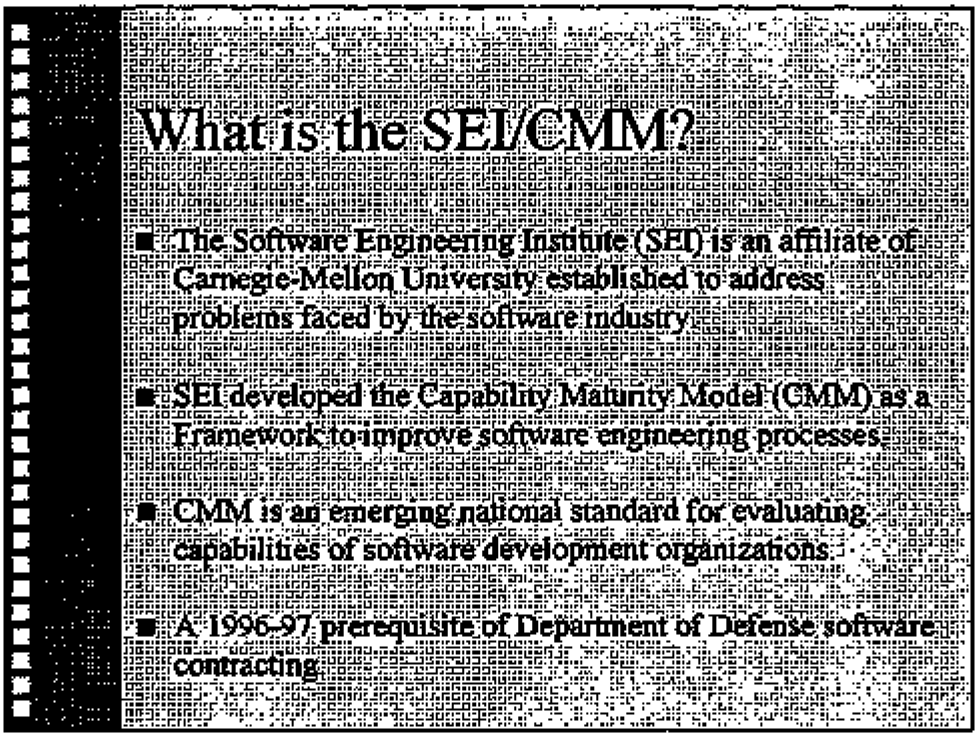


The DSRD Problem-Solving Model



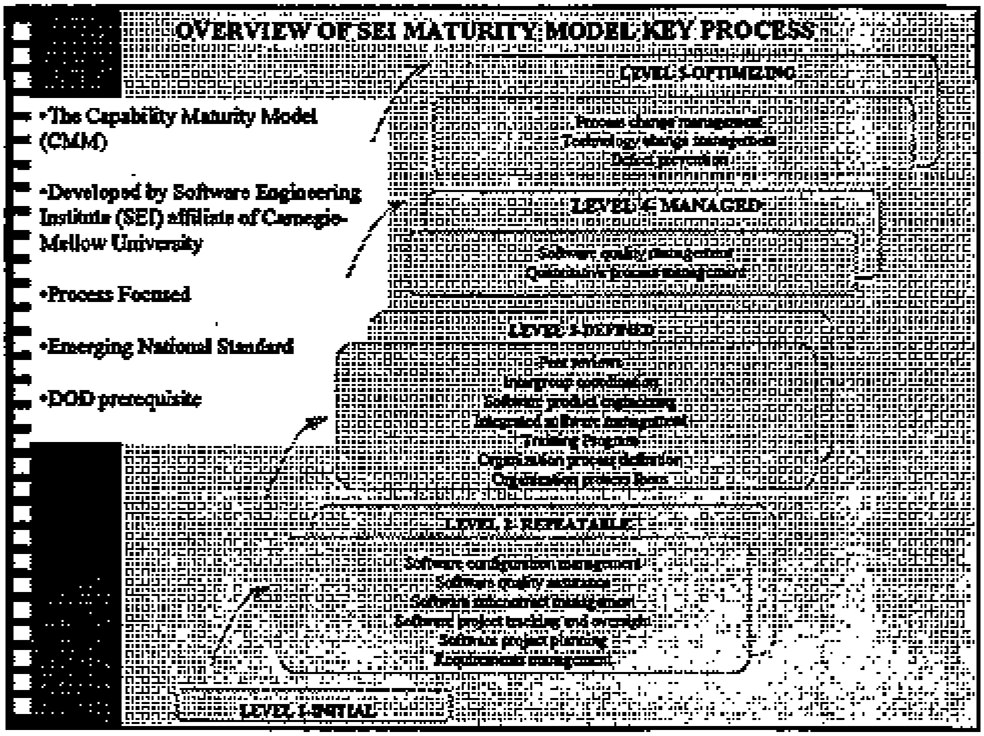
DSRD's Quality Journey

1986-87	DSRD created; First customer survey initiated.
1990-91	Formal Quality Program established based on TQM strategy. TQM Leadership Training initiated.
1992	Lessons Learned from internal audits.
1993	Project Management Training Conducted. SEI/CMM benchmarking conducted.
1994	SEI/CMM adopted as DSRD's process improvement approach. Process Improvement Teams established. DSRD Software Process Standards published.
1995	SEI/CMM-based internal review conducted. SEPG established. Quality based infrastructure established. Technical and Management Review Program established.
1996	Initial SEI assessment conducted and plan developed.



What is the SEI/CMM?

- The Software Engineering Institute (SEI) is an affiliate of Carnegie-Mellon University established to address problems faced by the software industry.
- SEI developed the Capability Maturity Model (CMM) as a Framework to improve software engineering processes.
- CMM is an emerging national standard for evaluating capabilities of software development organizations.
- A 1996-97 prerequisite of Department of Defense software contracting.



OVERVIEW OF SEI MATURITY MODEL KEY PROCESS

- •The Capability Maturity Model (CMM)
- •Developed by Software Engineering Institute (SEI) affiliate of Carnegie-Mellow University
- •Process Focused
- •Emerging National Standard
- •DOD prerequisite

LEVEL 5-OPTIMIZING

Process change management
Technology change management
Defect prevention

LEVEL 4-MANAGED

Software quality management
Quantitative process management

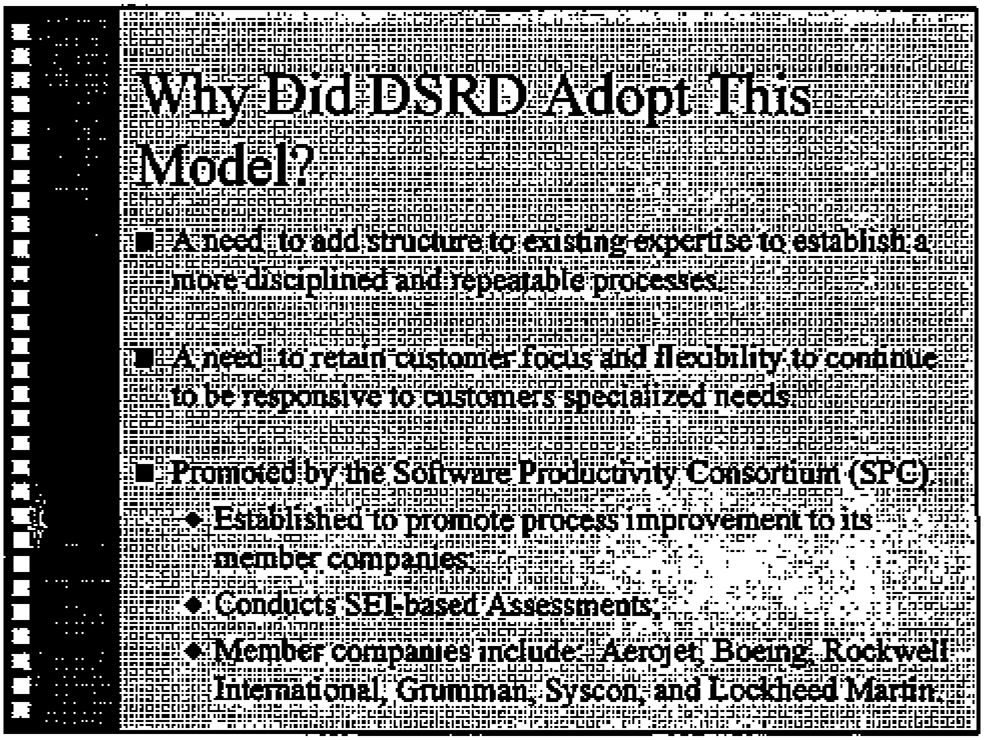
LEVEL 3-DEFINED

Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training Program
Organization process definition
Organization process focus

LEVEL 2-REPEATABLE

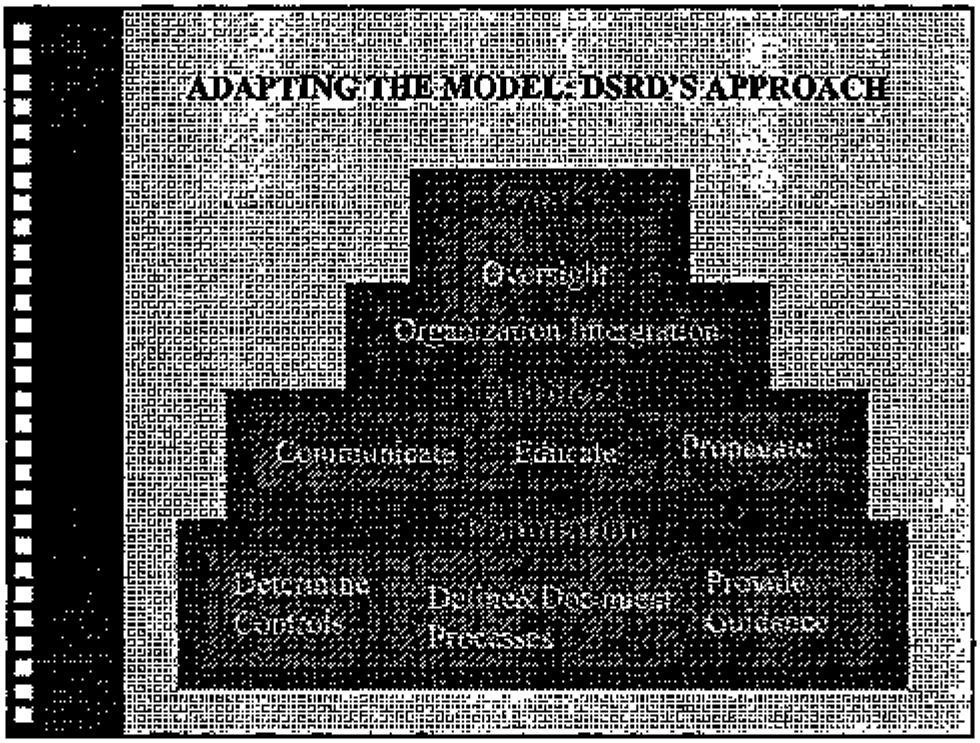
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management

LEVEL 1-INITIAL



Why Did DSRD Adopt This Model?

- ■ A need to add structure to existing expertise to establish a more disciplined and repeatable processes.
- ■ A need to retain customer focus and flexibility to continue to be responsive to customers specialized needs.
- ■ Promoted by the Software Productivity Consortium (SPC).
  - ◆ Established to promote process improvement to its member companies;
  - ◆ Conducts SEI-based Assessments;
  - ◆ Member companies include: Aerojet, Boeing, Rockwell International, Grumman, Syscon, and Lockheed Martin.



ADAPTING THE MODEL: DSRD'S APPROACH

Oversight

Organization Integration

Communicate

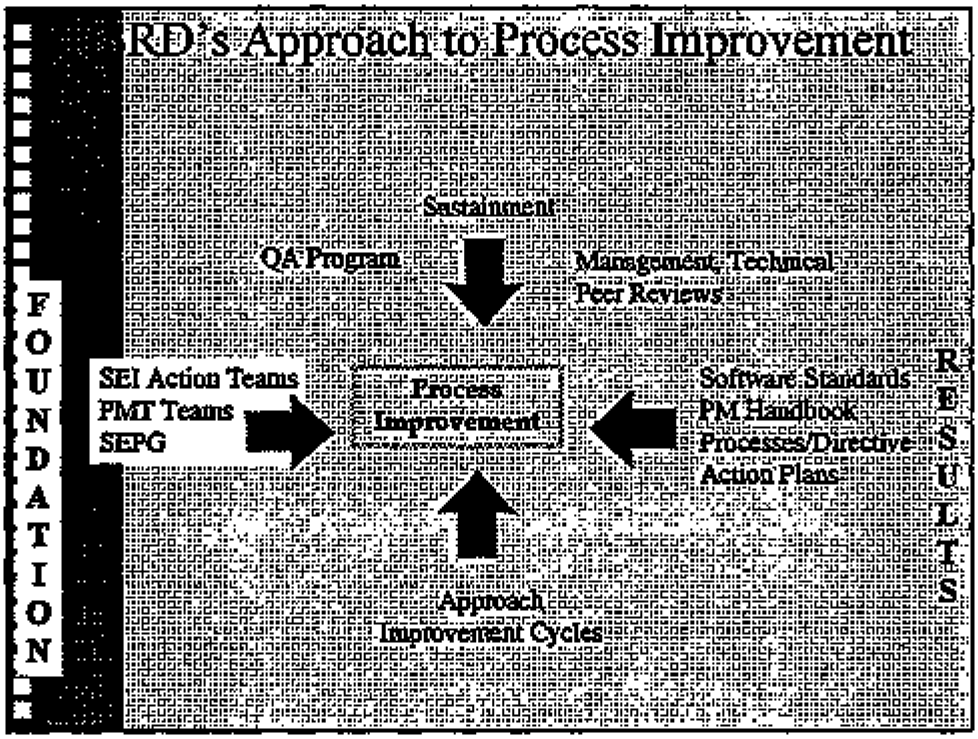
Educate

Propagate

Determine Controls

Define & Document Processes

Provide Guidance



RD's Approach to Process Improvement

FOUNDATION

Sustainment

QA Program

Management, Technical Peer Reviews

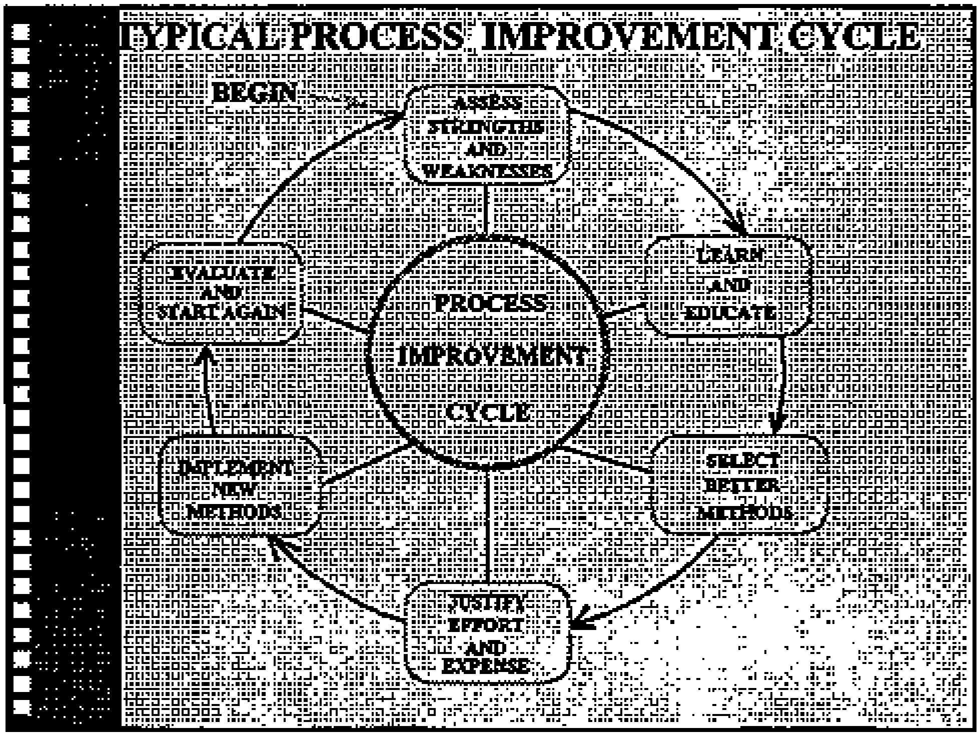
SEI Action Teams
PMT Teams
SEPG

Process Improvement

Software Standards
PM Handbook
Processes/Directive
Action Plans

Approach
Improvement Cycles

RESULTS



TYPICAL PROCESS IMPROVEMENT CYCLE

BEGIN

ASSESS STRENGTHS AND WEAKNESSES

LEARN AND EDUCATE

SELECT BETTER METHODS

JUSTIFY EFFORT AND EXPENSE

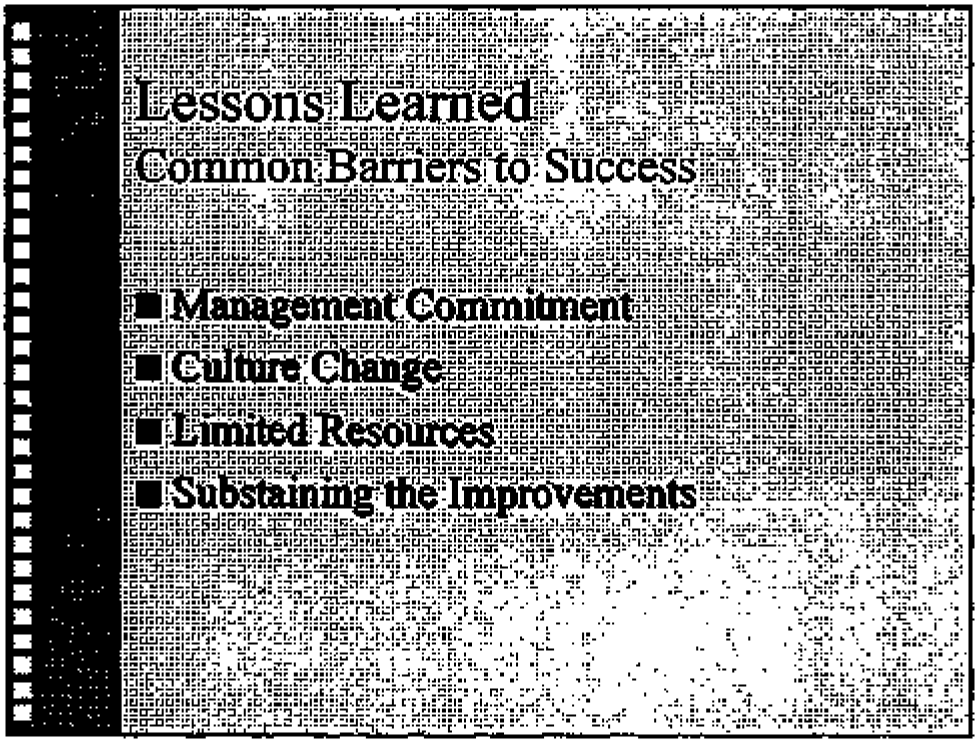
IMPLEMENT NEW METHODS

EVALUATE AND START AGAIN

PROCESS IMPROVEMENT CYCLE

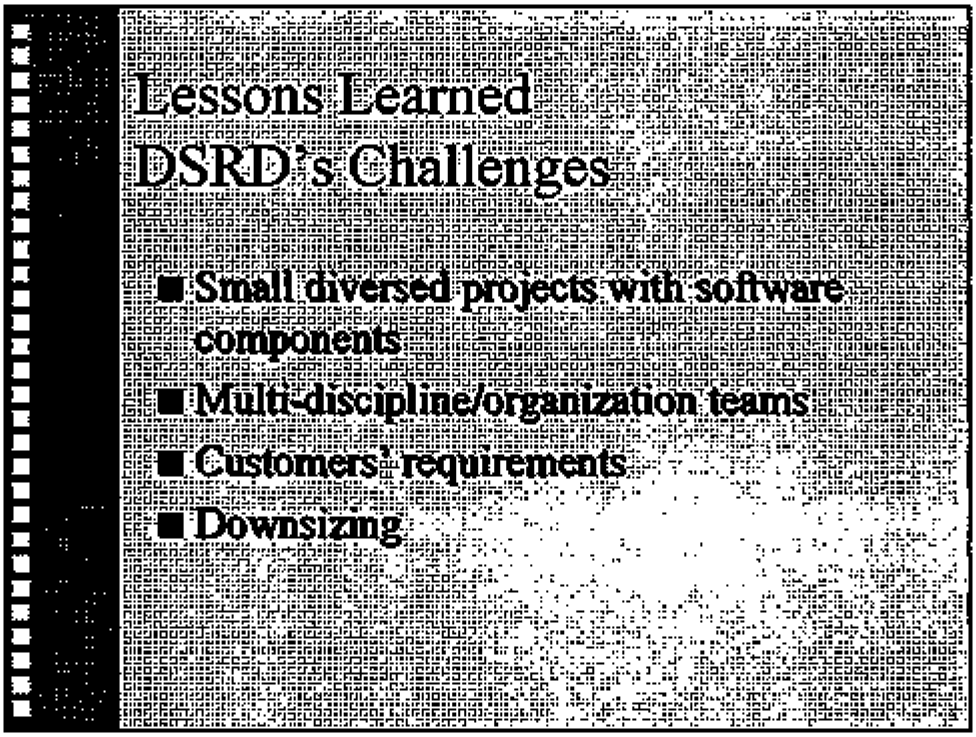
Accomplishments

- Developed Standards
- Documented Processes
- Established SEPG and Action Teams
- Performed Internal Review
- Developed Management Metrics Program
- Developed Software Engineering Process
- Completed External Assessment 1996



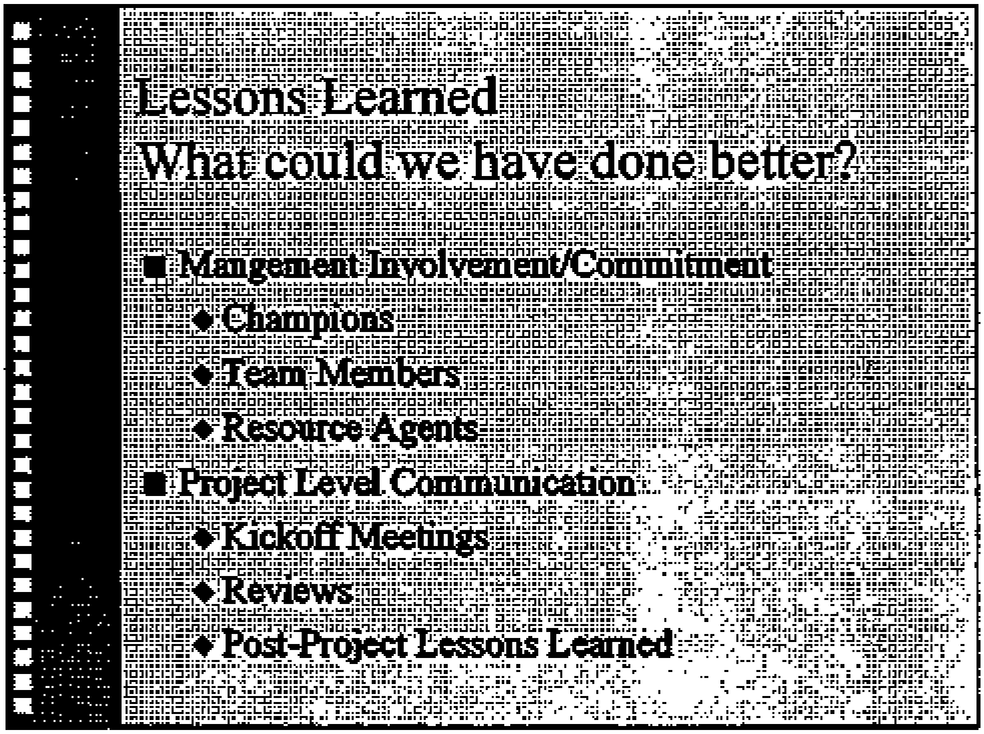
Lessons Learned
Common Barriers to Success

- Management Commitment
- Culture Change
- Limited Resources
- Substaining the Improvements



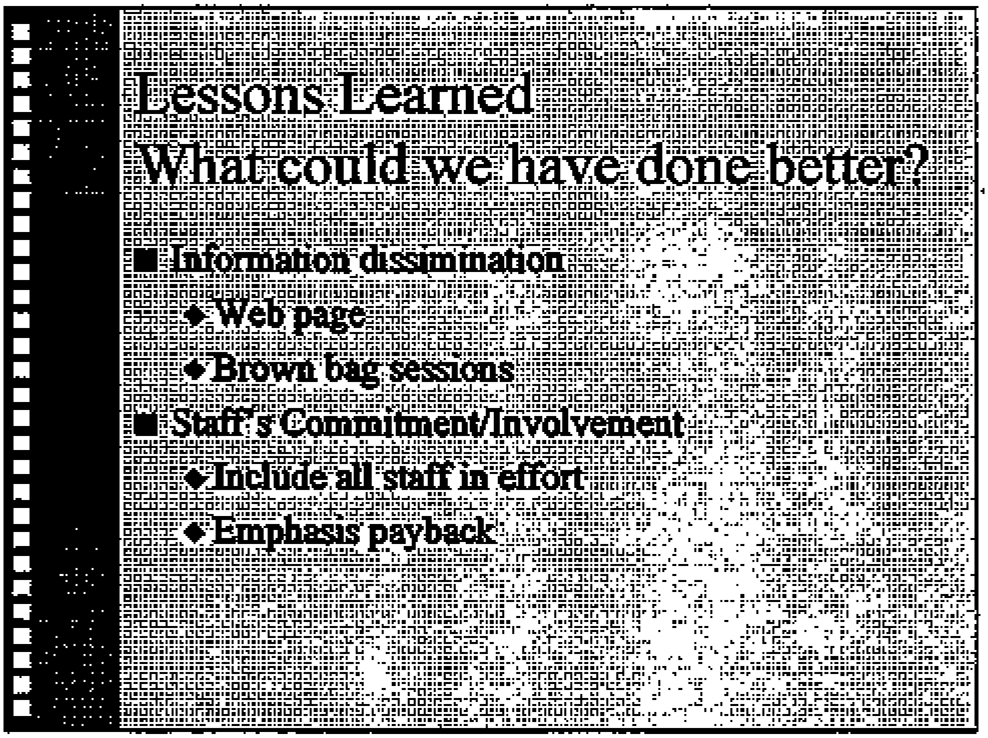
Lessons Learned
DSRD's Challenges

- Small diversed projects with software components
- Multi-discipline/organization teams
- Customers' requirements
- Downsizing



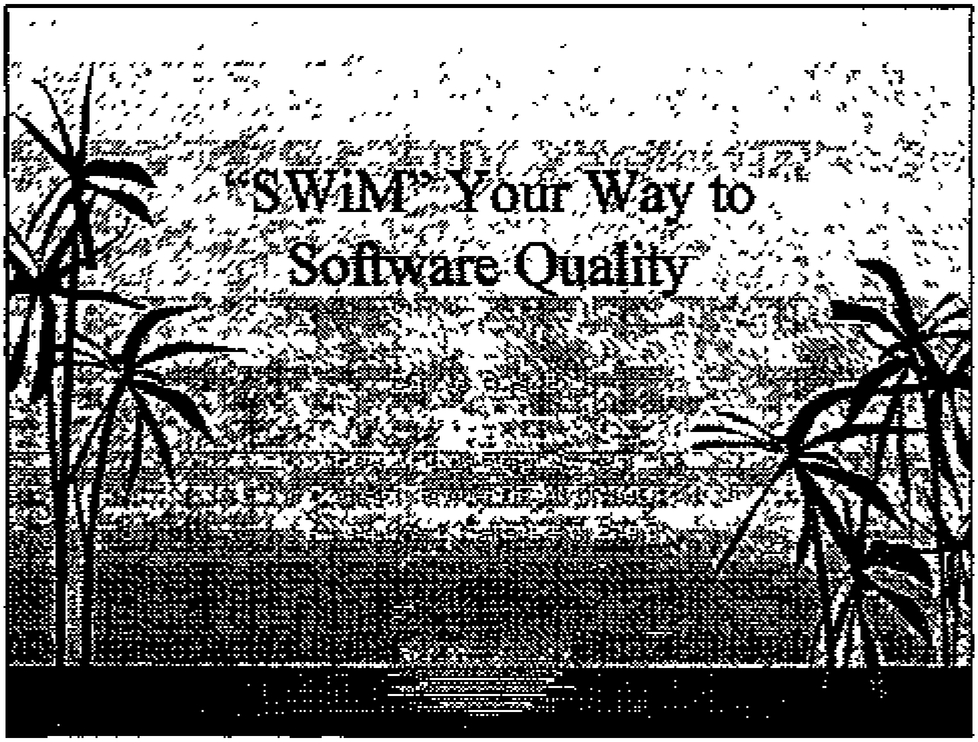
Lessons Learned
What could we have done better?

- Mangement Involvement/Commitment
  - Champions
  - Team Members
  - Resource Agents
- Project Level Communication
  - Kickoff Meetings
  - Reviews
  - Post-Project Lessons Learned

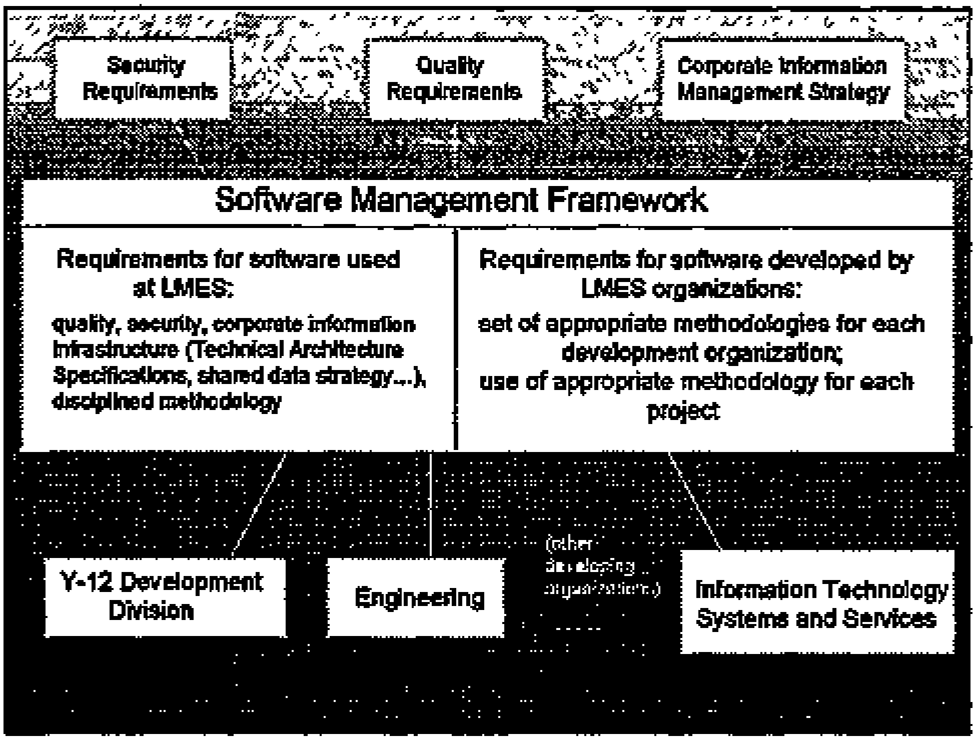


Lessons Learned
What could we have done better?

- Information dissimination
  - Web page
  - Brown bag sessions
- Staff's Commitment/Involvement
  - Include all staff in effort
  - Emphasis payback



"SWiM" Your Way to Software Quality



Software Management Framework

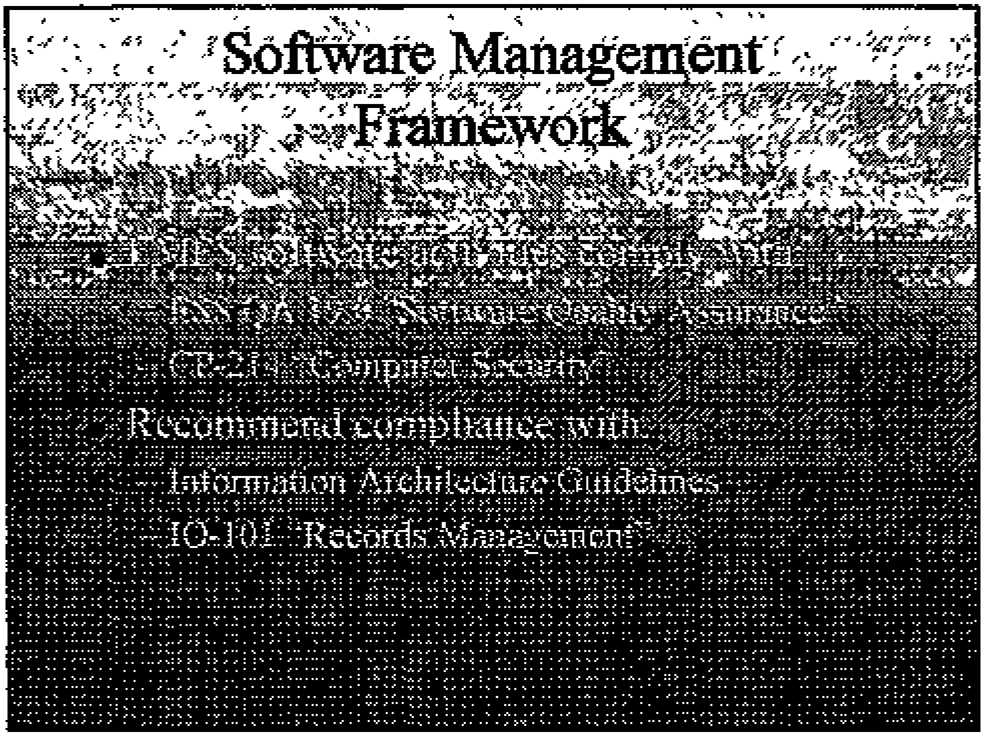
Requirements for software used at LMES:

quality, security, corporate information infrastructure (Technical Architecture Specifications, shared data strategy...), disciplined methodology

Requirements for software developed by LMES organizations:

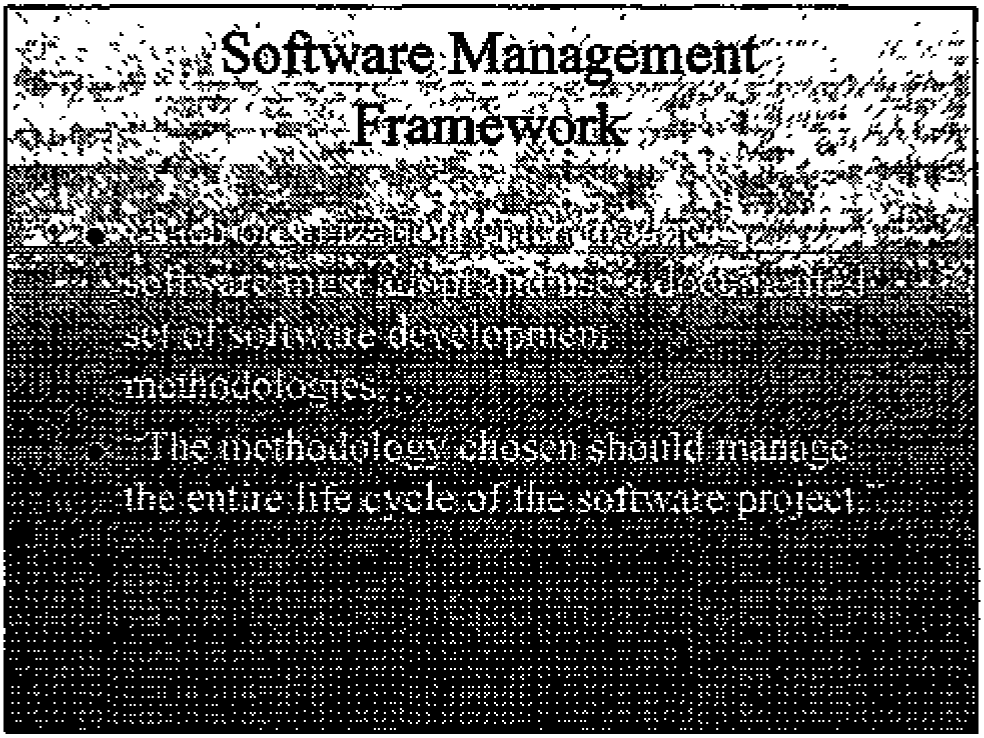
set of appropriate methodologies for each development organization;
use of appropriate methodology for each project

Y-12 Development Division
Engineering
(other developing organizations)
Information Technology Systems and Services



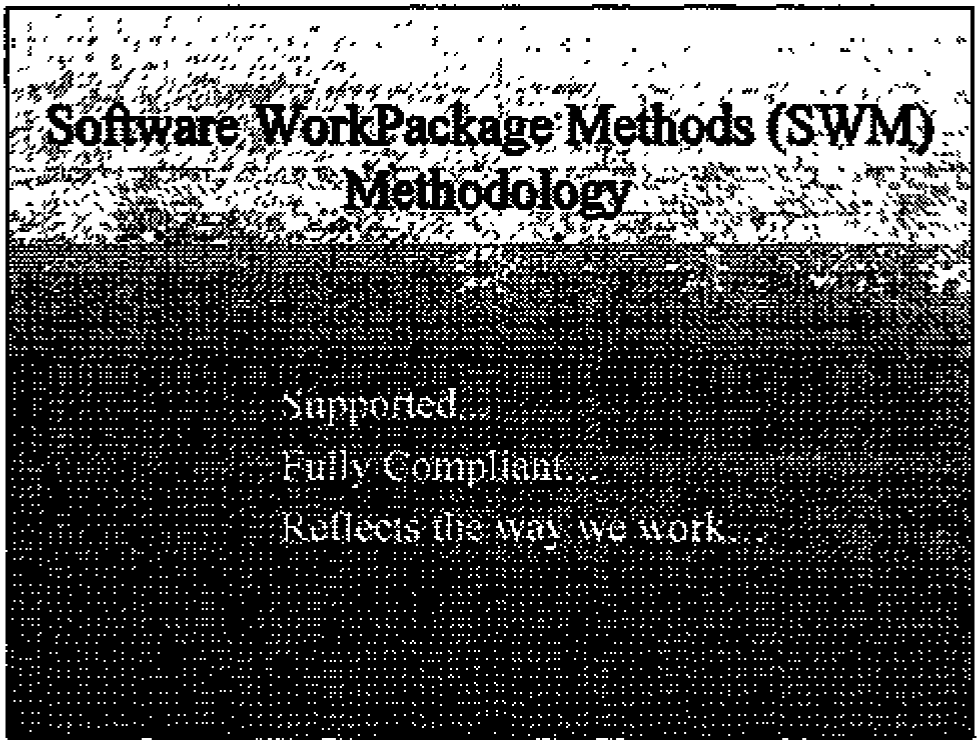
Software Management Framework

- LMES software activities comply with:
  - [illegible] "Software Quality Assurance"
  - CP-2.14 "Computer Security"
- Recommend compliance with:
  - Information Architecture Guidelines
  - IO-101 "Records Management"



Software Management Framework

- Each organization which develops software must adopt and use a documented set of software development methodologies.
- "The methodology chosen should manage the entire life cycle of the software project."

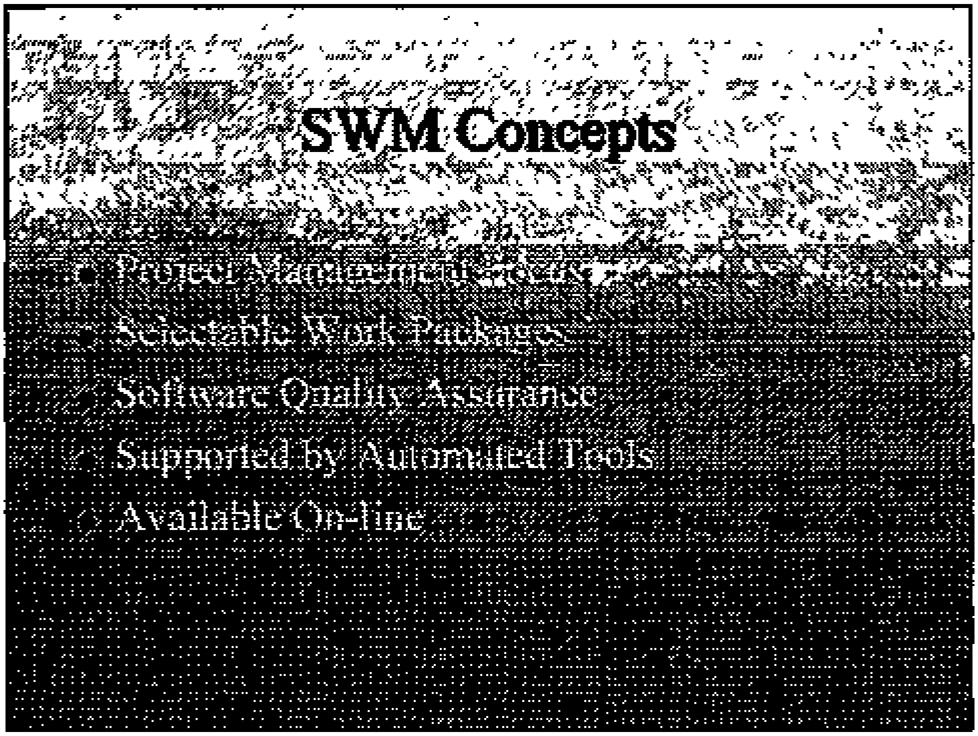


Software WorkPackage Methods (SWM) Methodology

Supported...

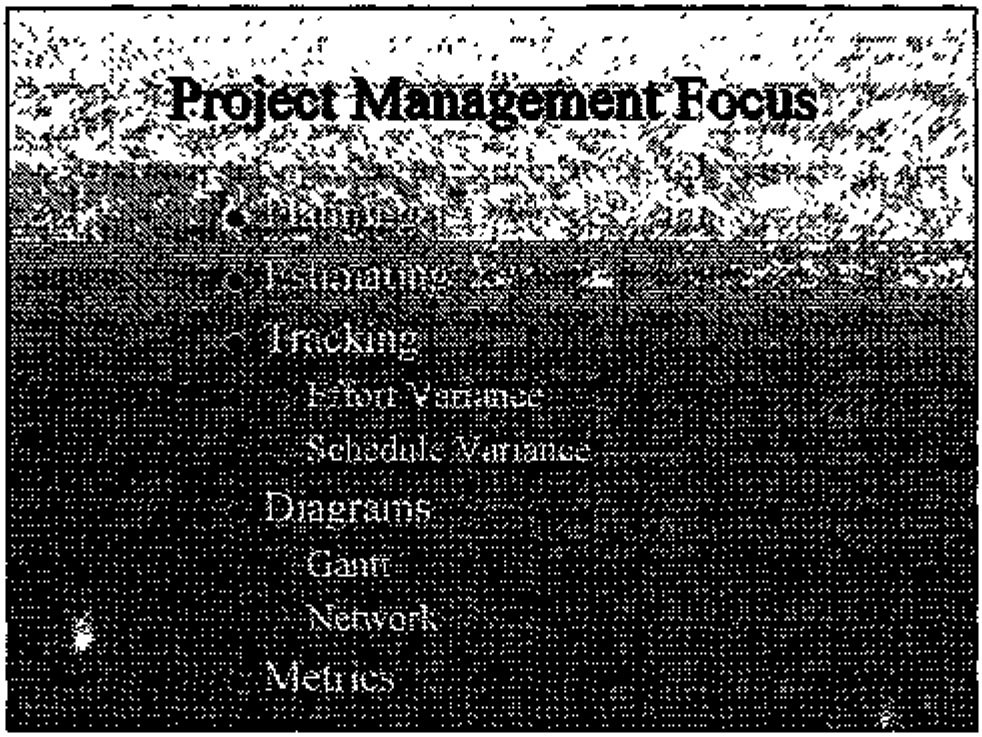
Fully Compliant...

Reflects the way we work...



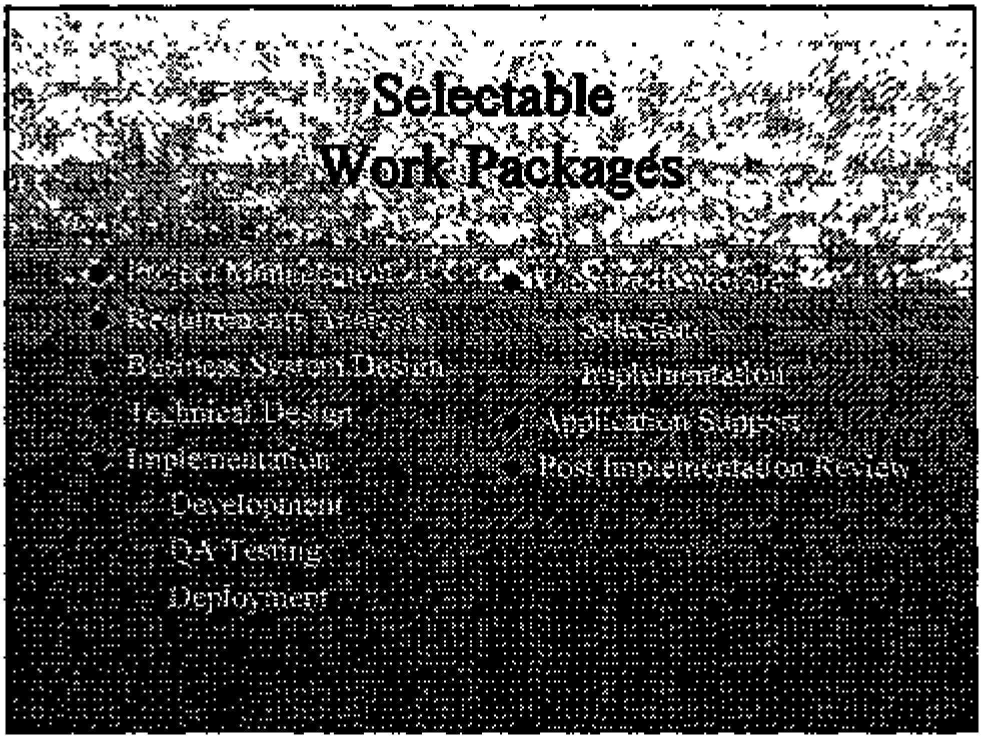
SWM Concepts

- Project Management Focus
- Selectable Work Packages
- Software Quality Assurance
- Supported by Automated Tools
- Available On-line



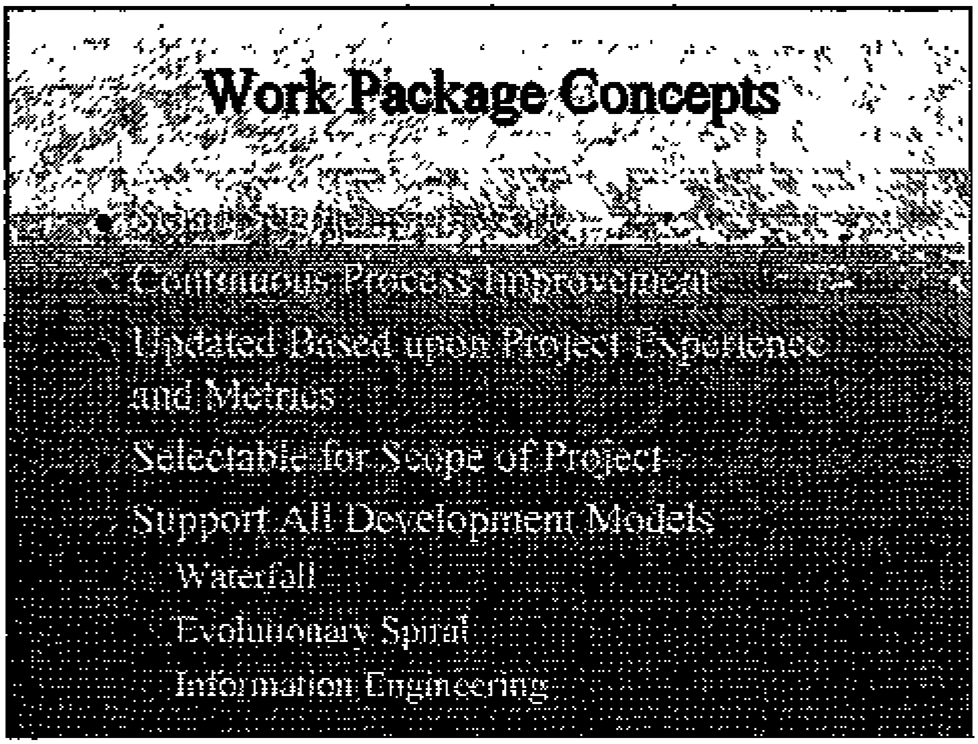
Project Management Focus

- Planning
- Estimating
- Tracking
  - Effort Variance
  - Schedule Variance
- Diagrams
  - Gantt
  - Network
- Metrics



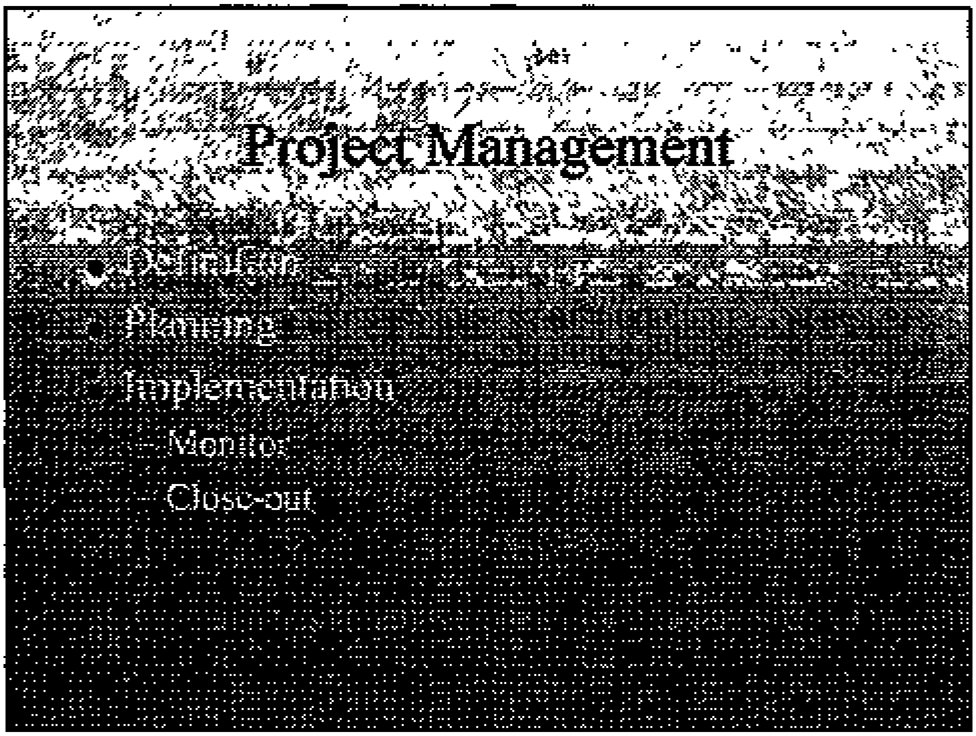
Selectable Work Packages

- Project Management
- Requirements Analysis
- Business System Design
- Technical Design
- Implementation
  - Development
  - QA Testing
  - Deployment
- Packaged Software
  - Selection
  - Implementation
- Application Support
- Post Implementation Review



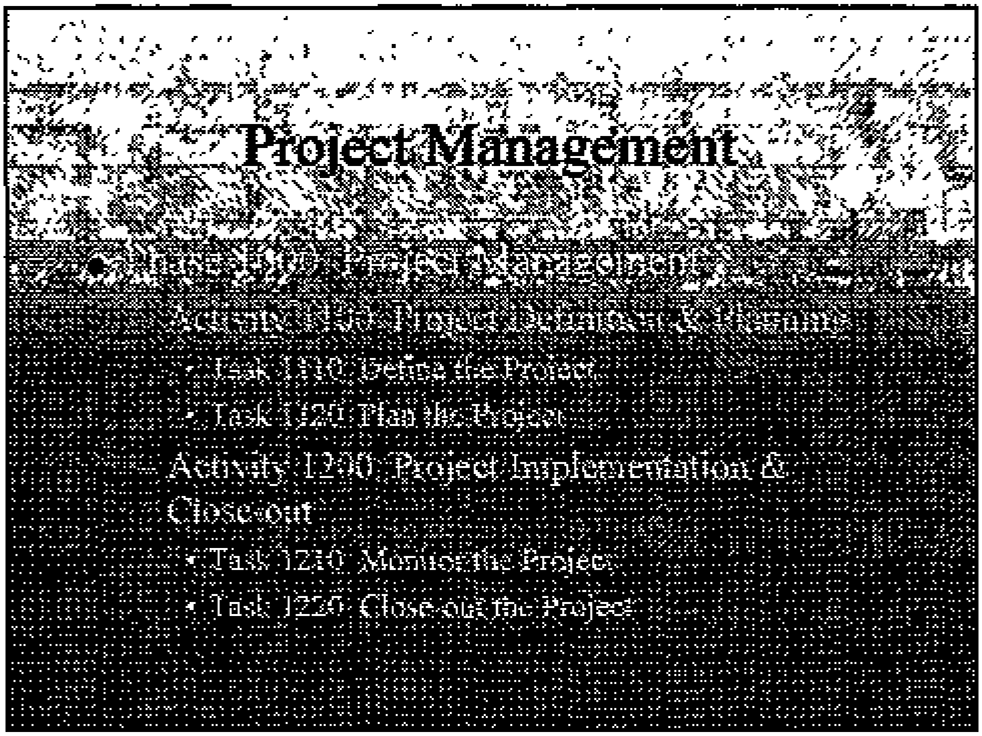
Work Package Concepts

- [illegible]
- Continuous Process Improvement
- Updated Based upon Project Experience and Metrics
- Selectable for Scope of Project
- Support All Development Models
  - Waterfall
  - Evolutionary Spiral
  - Information Engineering



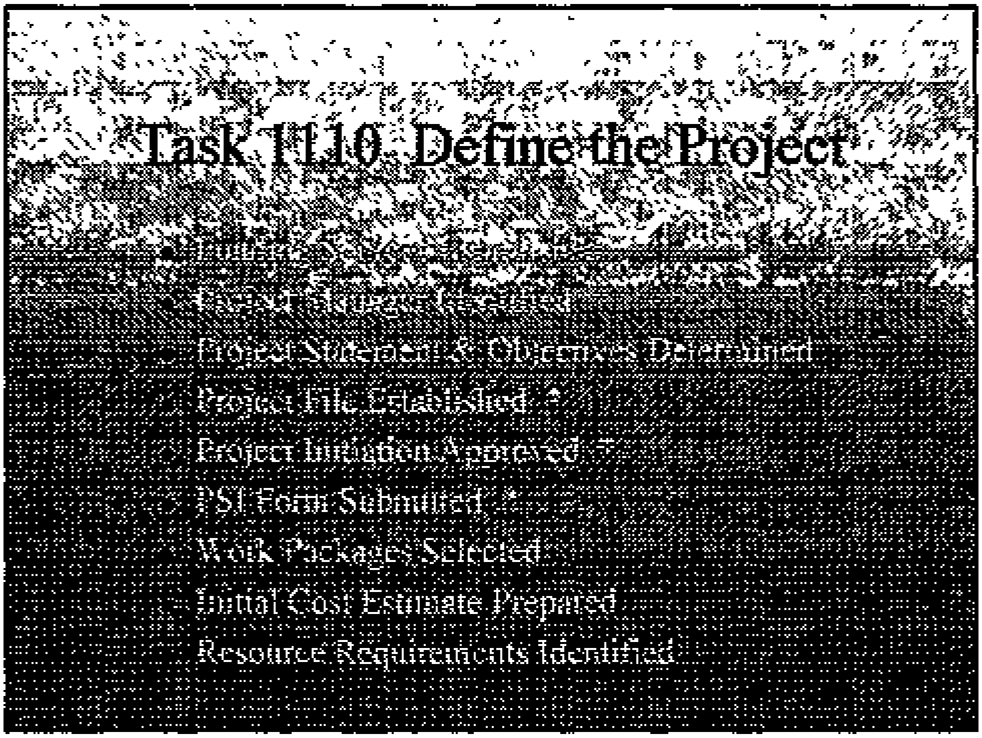
Project Management

- Definition
- Planning
- Implementation
  - Monitor
  - Close-out



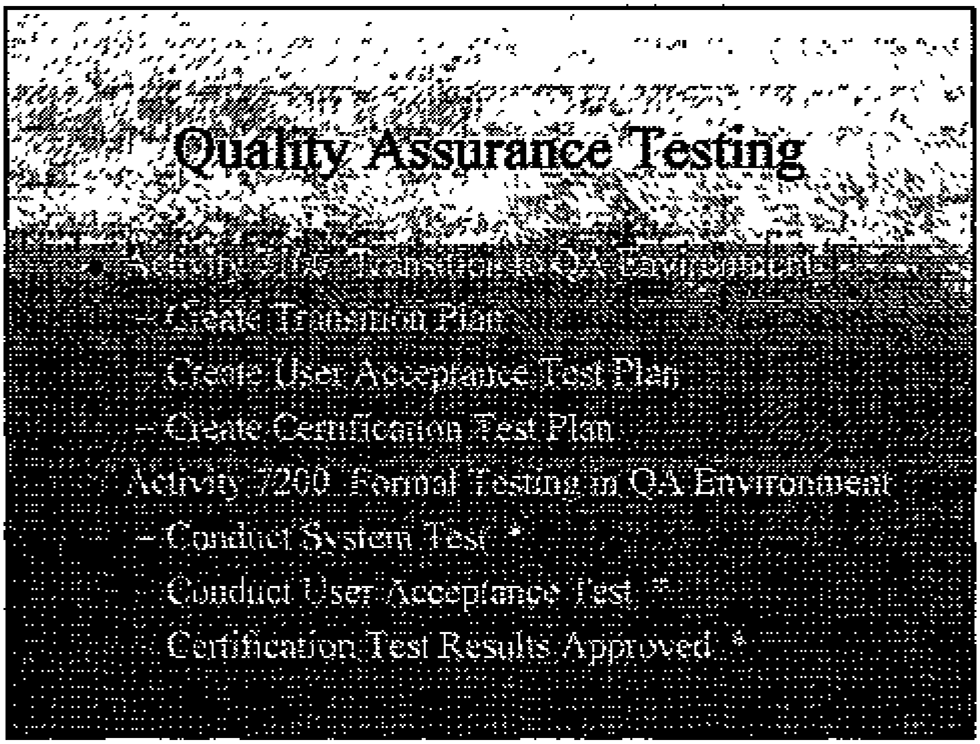
Project Management

- Phase 1000: Project Management
  - Activity 1100: Project Definition & Planning
    - Task 1110: Define the Project
    - Task 1120: Plan the Project
  - Activity 1200: Project Implementation & Close-out
    - Task 1210: Monitor the Project
    - Task 1220: Close-out the Project



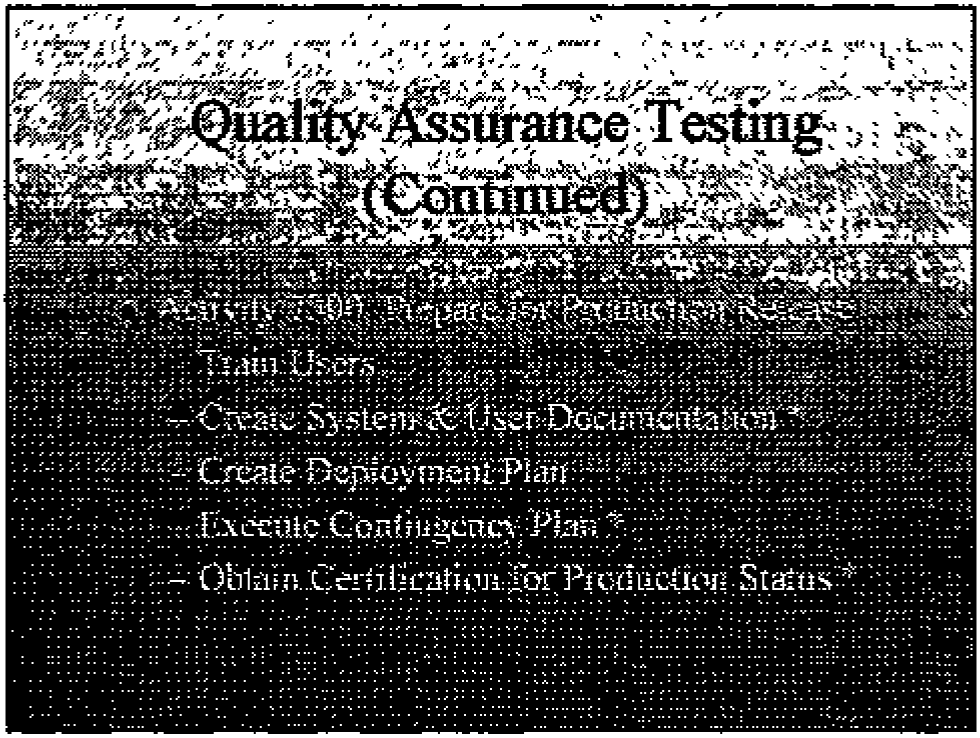
Task 1110: Define the Project

- [illegible]
- Project Manager Identified
- Project Statement & Objectives Determined
- Project File Established *
- Project Initiation Approved *
- PSI Form Submitted *
- Work Packages Selected
- Initial Cost Estimate Prepared
- Resource Requirements Identified



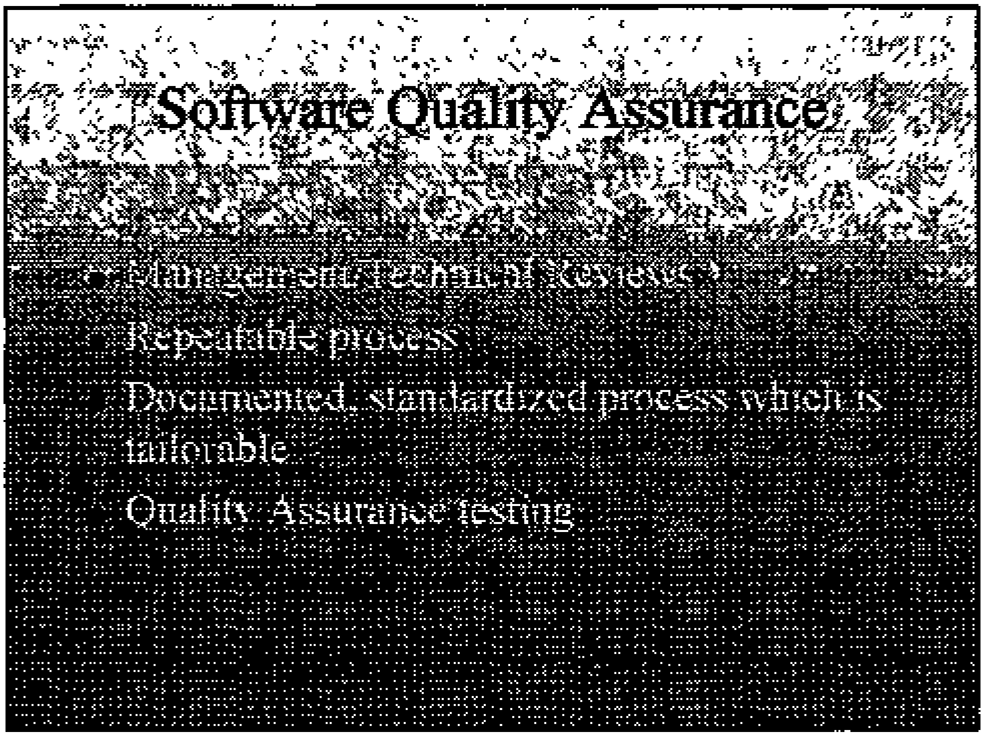
Quality Assurance Testing

- Activity 7100 Transition to QA Environment
  - Create Transition Plan
  - Create User Acceptance Test Plan
  - Create Certification Test Plan
- Activity 7200 Formal Testing in QA Environment
  - Conduct System Test *
  - Conduct User Acceptance Test *
  - Certification Test Results Approved *



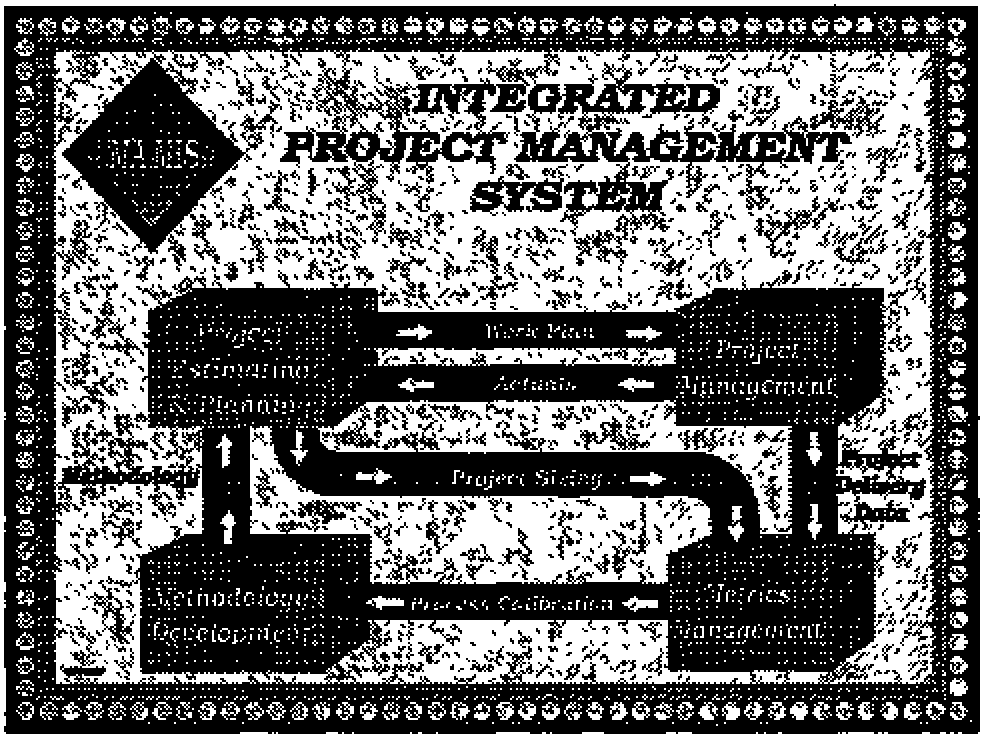
Quality Assurance Testing (Continued)

- Activity 7300 Prepare for Production Release
  - Train Users
  - Create System & User Documentation *
  - Create Deployment Plan
  - Execute Contingency Plan *
  - Obtain Certification for Production Status *



Software Quality Assurance

- Management/Technical Reviews
  - Repeatable process
  - Documented, standardized process which is tailorable
  - Quality Assurance testing



INTEGRATED PROJECT MANAGEMENT SYSTEM

Project Estimating & Planning

Work Plan

Actuals

Project Management

Project Sizing

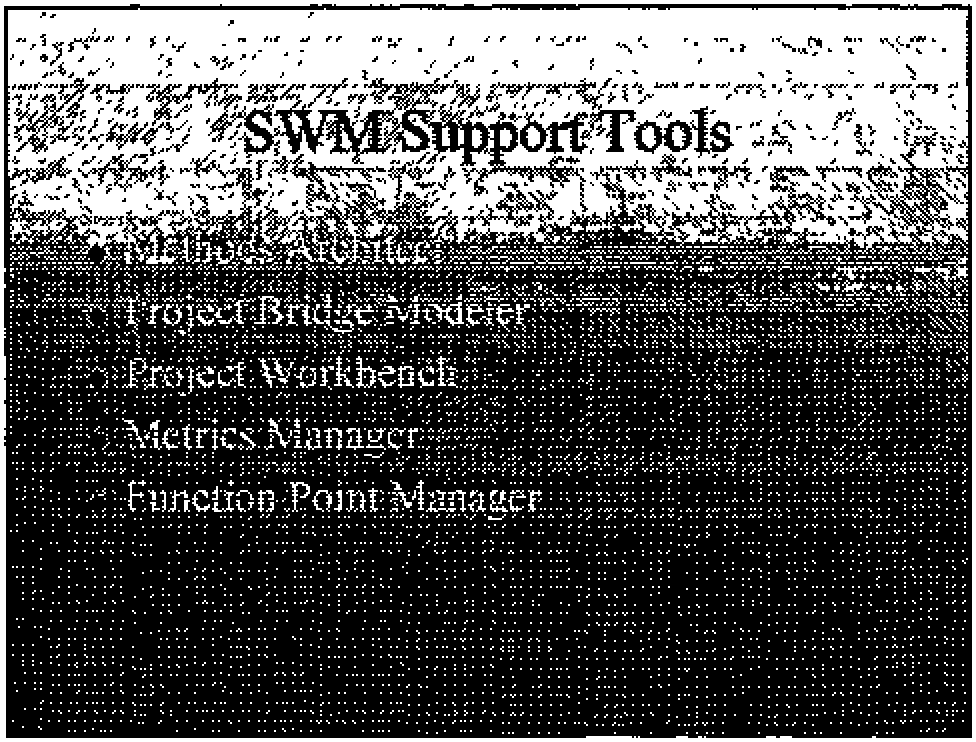
Methodology

Project Delivery Data

Methodology Development

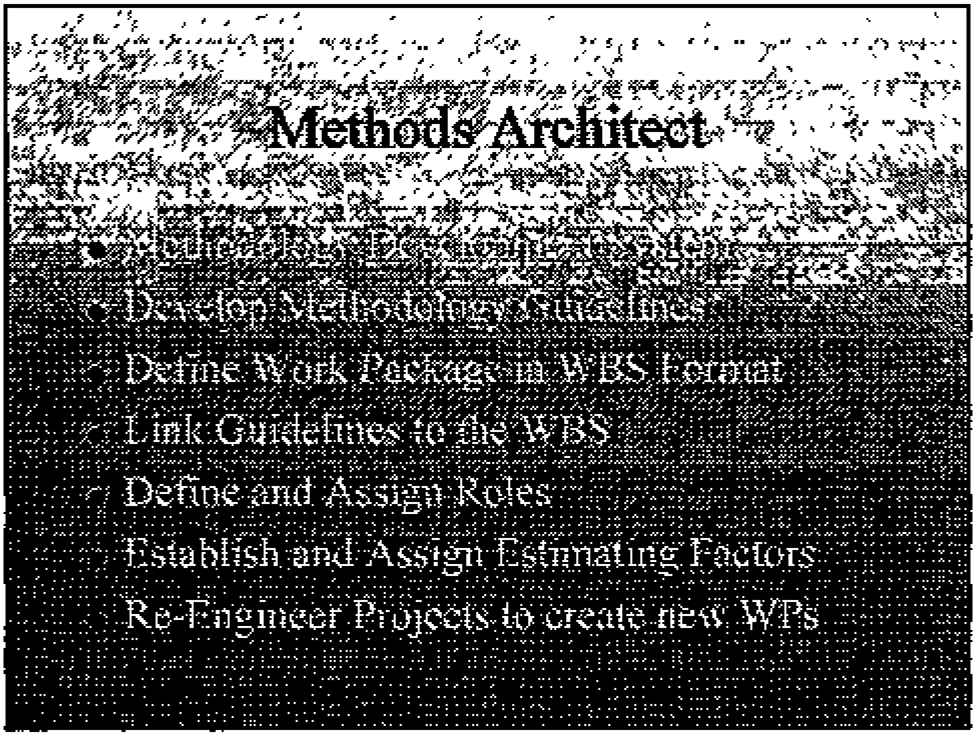
Process Calibration

Metrics Management



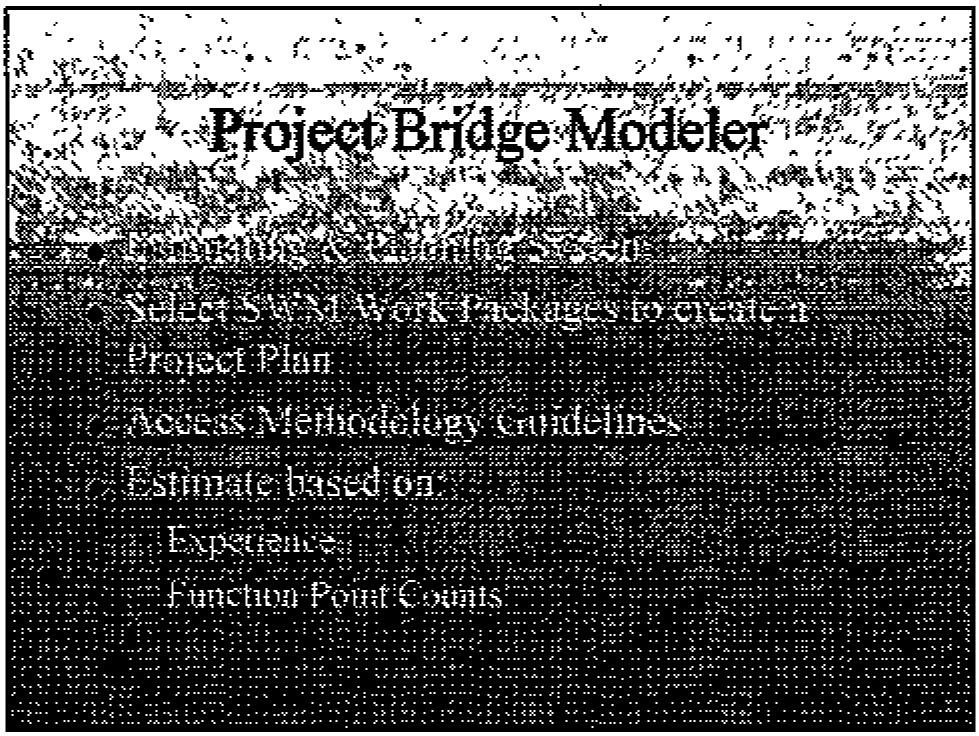
SWM Support Tools

- Methods Architect
- Project Bridge Modeler
- Project Workbench
- Metrics Manager
- Function Point Manager



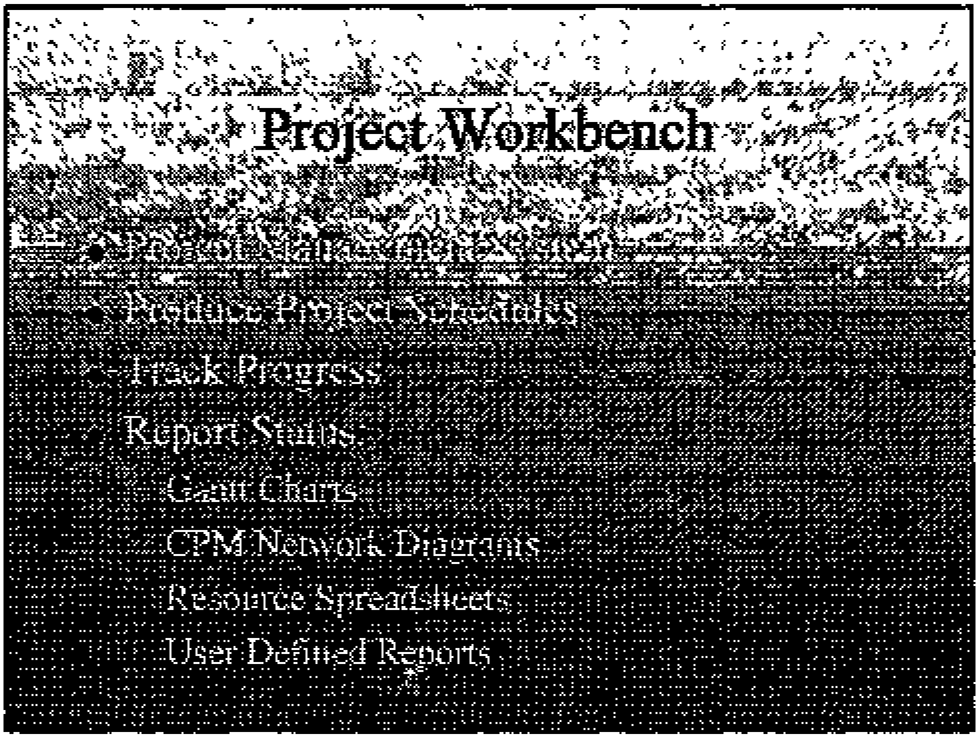
Methods Architect

- Methodology Design System
- Develop Methodology Guidelines
- Define Work Package in WBS Format
- Link Guidelines to the WBS
- Define and Assign Roles
- Establish and Assign Estimating Factors
- Re-Engineer Projects to create new WPs



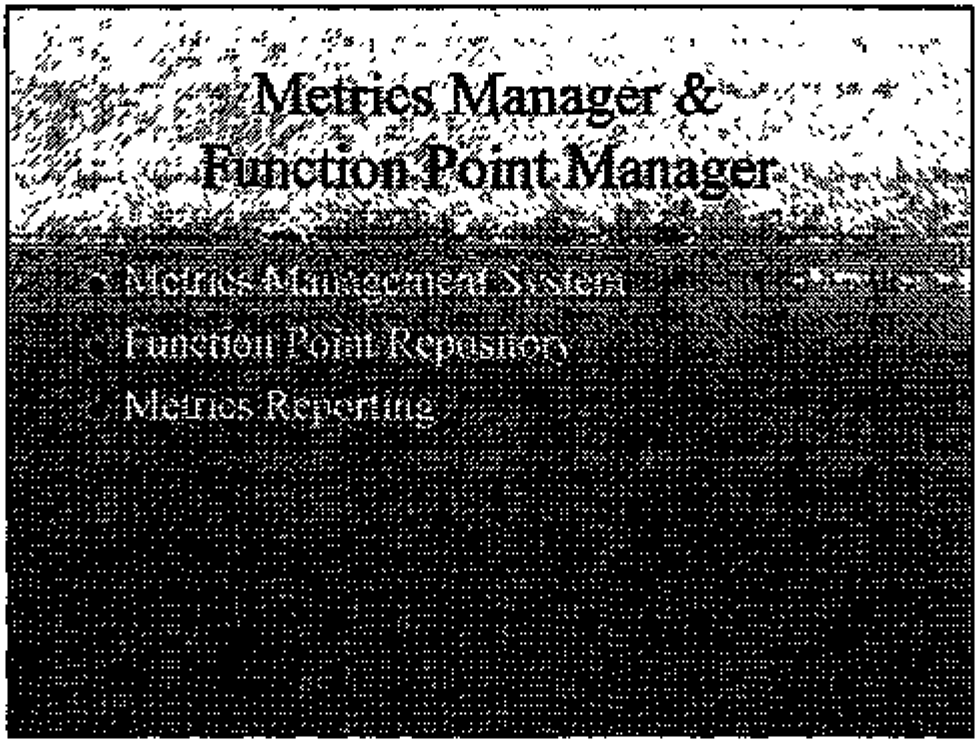
Project Bridge Modeler

- Estimating & Planning System
- Select SWM Work Packages to create a Project Plan
- Access Methodology Guidelines
- Estimate based on:
  - Experience
  - Function Point Counts



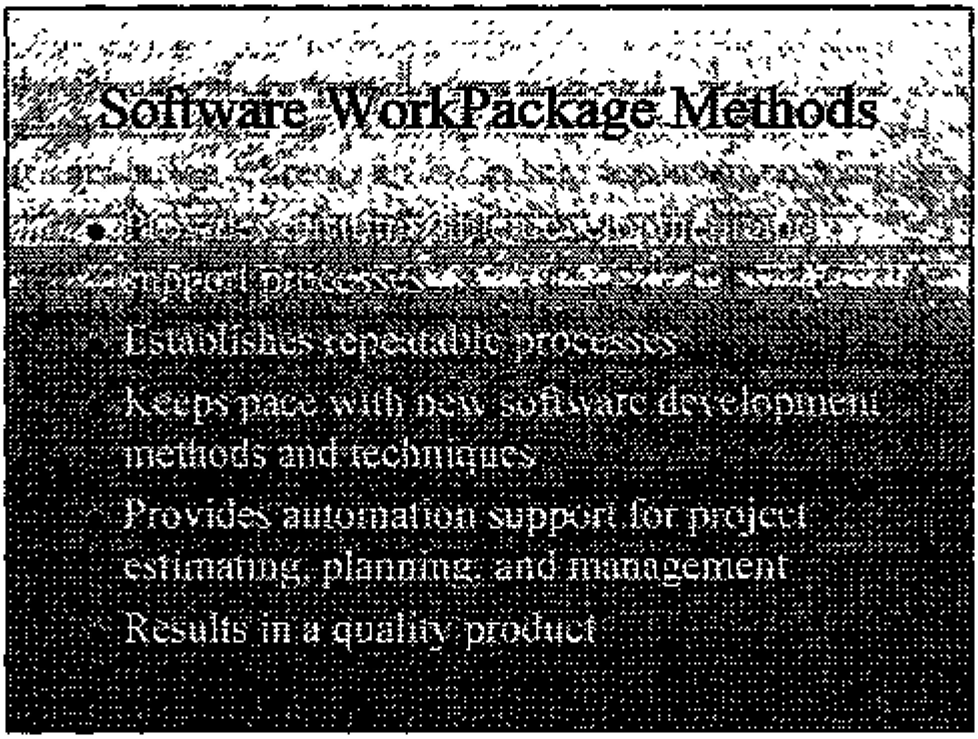
Project Workbench

- Project Management System
- Produce Project Schedules
- Track Progress
- Report Status:
  - Gantt Charts
  - CPM Network Diagrams
  - Resource Spreadsheets
  - User Defined Reports



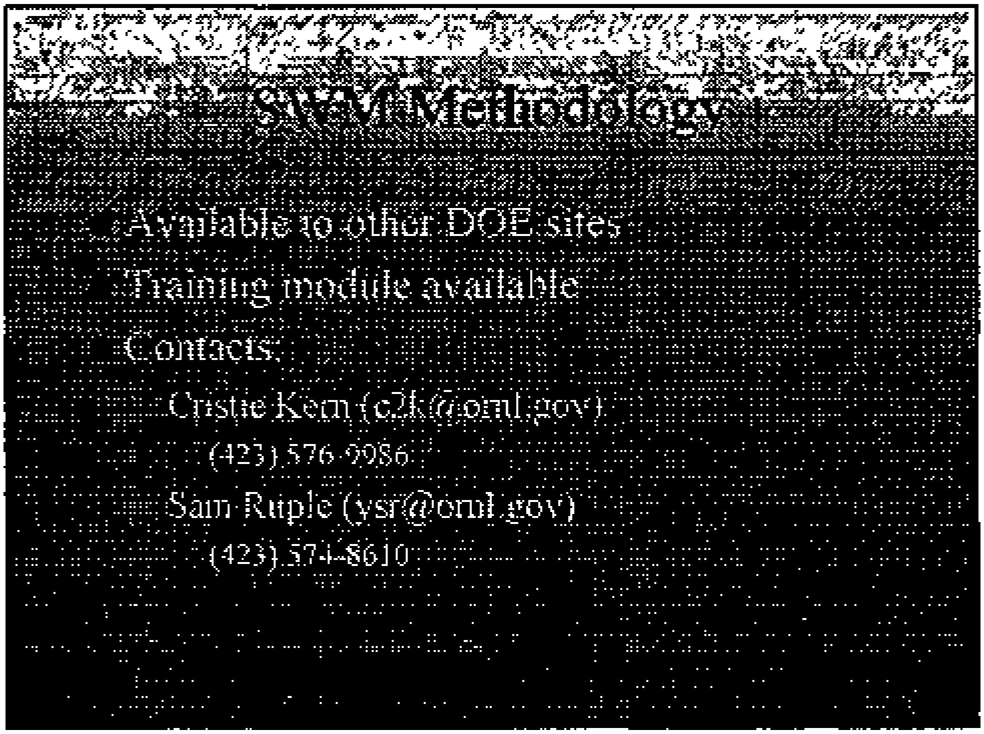
Metrics Manager & Function Point Manager

- Metrics Management System
- Function Point Repository
- Metrics Reporting



Software WorkPackage Methods

- [illegible] support processes
- Establishes repeatable processes
- Keeps pace with new software development methods and techniques
- Provides automation support for project estimating, planning, and management
- Results in a quality product



SWM Methodology

- Available to other DOE sites
- Training module available
- Contacts:
  - Cristie Kern (c2k@ornl.gov)
    - (423) 576-9986
  - Sam Ruple (ysr@ornl.gov)
    - (423) 574-8610

Session B4: High Integrity / Formal Methods II

Chair Larry Dalton
Sandia National Laboratories

Session : Paper #	Author(s)	*Title*
B4:1	Mikhail Auguston New Mexico State University	*Debugging Automation Tools Based on Event Grammars and Computations over Traces.*
B4:2	Marie-Elena Kidd Sandia National Laboratories	*A Method for Critical Software Event Execution Reliability in High Integrity Software*
B4:3	John Sharp Sandia National Laboratories	*Business Rule Enforcement Via Natural Language Modeling*

Debugging Automation Tools Based on Event Grammars and Computations over Traces

Mikhail Auguston
Department of Computer Science,
New Mexico State University, Las Cruces, New Mexico, USA
Phone: (505)-646-5286
fax: (505)-646-1002
Email: mikau@cs.nmsu.ed

Major problem in debugging automation:
the inability to express the mismatch between the expected and the observed behavior of the program on the level of abstraction maintained by the user.

Suggested solution:
to define a precise model of program behavior as a set of events with two binary relations: inclusion and precedence

Motivation for this work:
we propose to research and to design software testing and debugging automation tools, in particular, a language for computations over source program execution history

Examples of such computations:

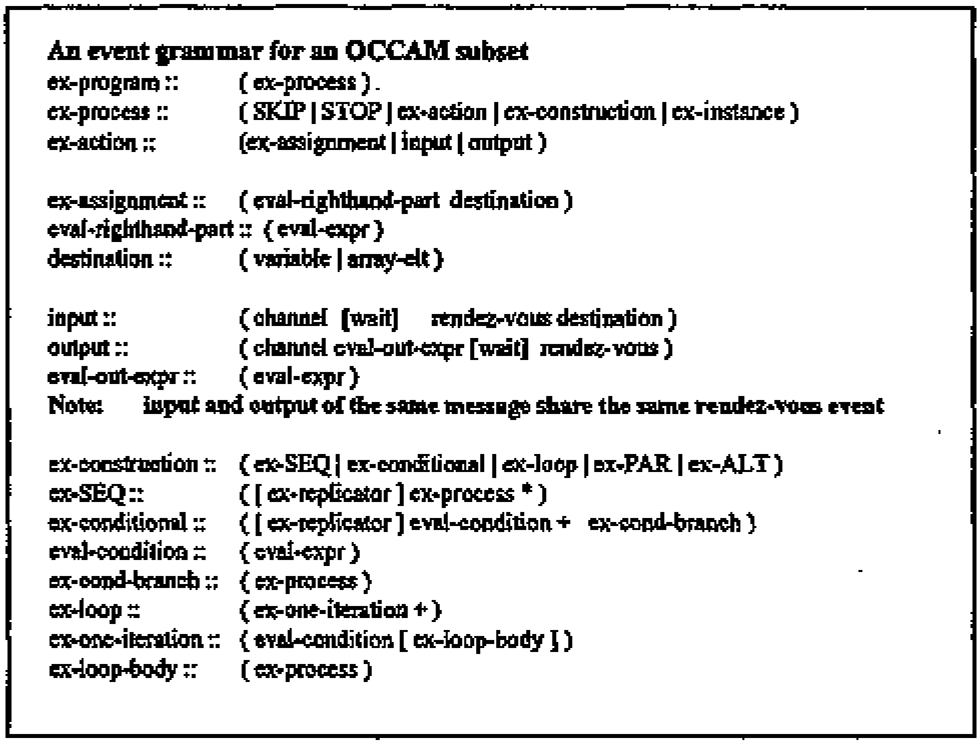
- assertion checking,
- profiles,
- performance measurement,
- debugging queries

Essential features of this approach:

- The notion of an event grammar provides a precise and formal model of parallel program behavior defined as a set of partially ordered nested events
- Event attributes provide complete access to each target program's execution state
- The inclusion relation yields a hierarchy of events; assertions can be defined at appropriate level of granularity
- Events can be detected by automatic source program instrumentation
- Patterns and aggregate operations on events describe computations over event traces
- Our approach is nondestructive: assertion text is separated from the source program's text
- Ability to formalize universal assertions and to define debugging rules and strategies

Events

- A particular action may be performed many times, but every execution of the action is denoted by a unique event.
- Every event is associated with a time-span that has a defined beginning and end.
- A composite event is a (partially ordered) set of other events.
- An event occurs when some action is performed in the target program execution process. For instance: a message is sent, a statement is executed, or an expression is evaluated.
- Each event should be detectable during the target program run time by an appropriate (automatic) instrumentation



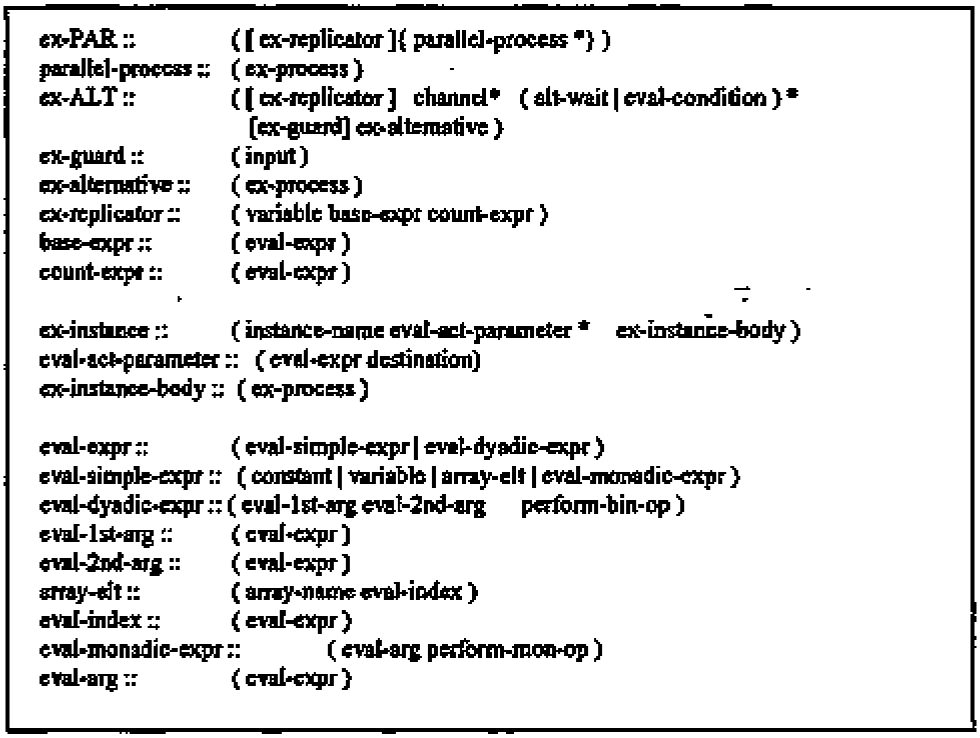
An event grammar for an OCCAM subset

```
ex-program ::          ( ex-process ) .
ex-process ::          ( SKIP | STOP | ex-action | ex-construction | ex-instance )
ex-action ::           (ex-assignment | input | output )

ex-assignment ::    ( eval-righthand-part  destination )
eval-righthand-part :: ( eval-expr )
destination ::         ( variable | array-elt )

input ::               ( channel  [wait]     rendez-vous destination )
output ::              ( channel eval-out-expr [wait]  rendez-vous )
eval-out-expr ::       ( eval-expr )
Note:      input and output of the same message share the same rendez-vous event

ex-construction ::   ( ex-SEQ | ex-conditional | ex-loop | ex-PAR | ex-ALT )
ex-SEQ ::              ( [ ex-replicator ] ex-process * )
ex-conditional ::      ( [ ex-replicator ] eval-condition +   ex-cond-branch )
eval-condition ::      ( eval-expr )
ex-cond-branch ::    ( ex-process )
ex-loop ::             ( ex-one-iteration + )
ex-one-iteration ::  ( eval-condition [ ex-loop-body ] )
ex-loop-body ::        ( ex-process )
```

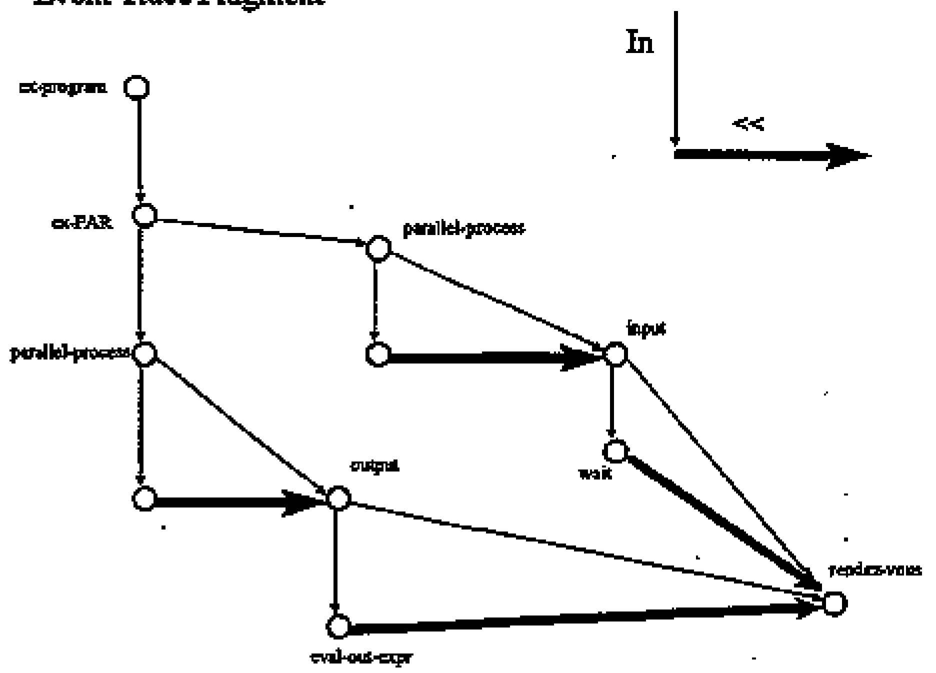


```
ex-PAR ::              ( [ ex-replicator ]( parallel-process *) )
parallel-process ::   ( ex-process )
ex-ALT ::              ( [ ex-replicator ]   channel*   ( alt-wait | eval-condition ) *
                         [ex-guard] ex-alternative )
ex-guard ::            ( input )
ex-alternative ::      ( ex-process )
ex-replicator ::       ( variable base-expr count-expr )
base-expr ::           ( eval-expr )
count-expr ::          ( eval-expr )

ex-instance ::         ( instance-name eval-act-parameter *    ex-instance-body )
eval-act-parameter ::  ( eval-expr destination)
ex-instance-body ::  ( ex-process )

eval-expr ::           ( eval-simple-expr | eval-dyadic-expr )
eval-simple-expr ::  ( constant | variable | array-elt | eval-monadic-expr )
eval-dyadic-expr :: ( eval-1st-arg eval-2nd-arg      perform-bin-op )
eval-1st-arg ::        ( eval-expr )
eval-2nd-arg ::        ( eval-expr )
array-elt ::           ( array-name eval-index )
eval-index ::          ( eval-expr )
eval-monadic-expr ::            ( eval-arg perform-mon-op )
eval-arg ::            ( eval-expr )
```

Event Trace Fragment



This model makes it possible to formalize assertions of the type:

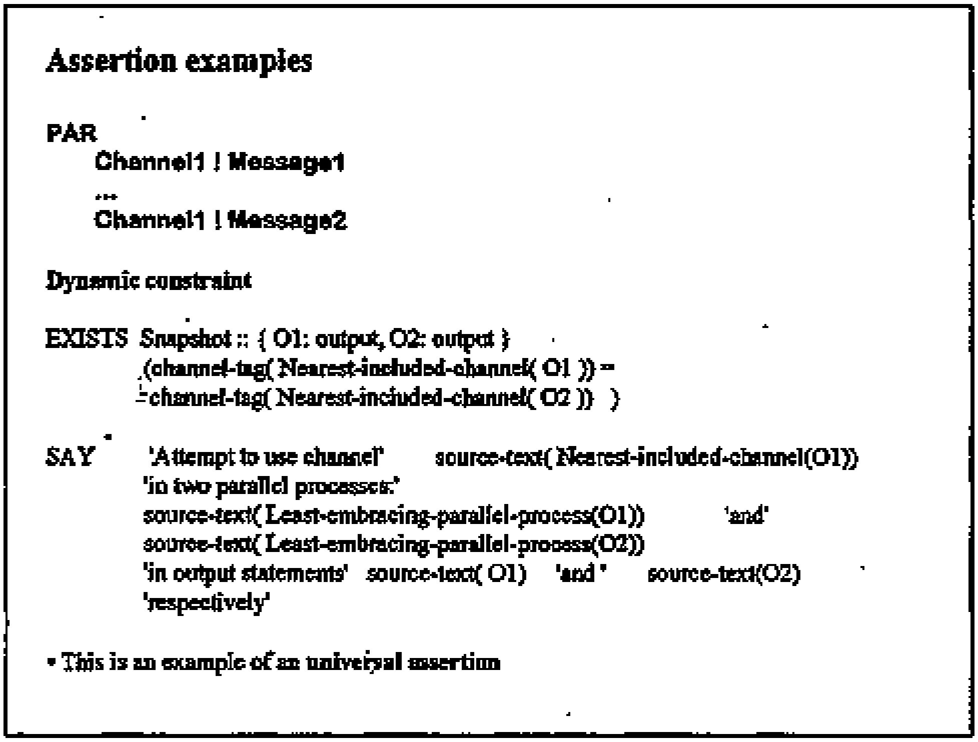
- "all variables in the program must be initialized before using in some expression",

- "file must be opened, then the read statement is performed zero or more times and after that the close statement is executed",

- "at least one variable changes its value during one loop iteration",

- "after the execution of a subprogram P the value of variable X remains unchanged",

- "there is an attempt to assign values to the same variable in two parallel processes" (data race condition).



Assertion examples

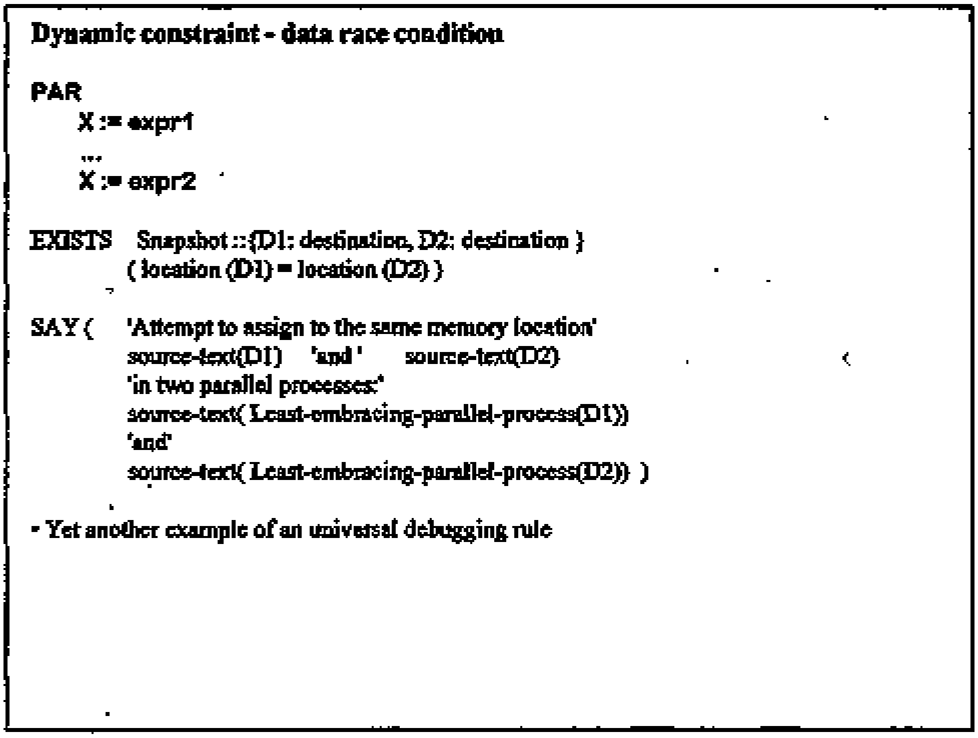
```
PAR
    Channel1 ! Message1
    ...
    Channel1 ! Message2
```

Dynamic constraint

```
EXISTS Snapshot :: { O1: output, O2: output }
          (channel-tag( Nearest-included-channel( O1 )) =
          =channel-tag( Nearest-included-channel( O2 ))   )

SAY       'Attempt to use channel'      source-text( Nearest-included-channel(O1))
          'in two parallel processes:'
          source-text( Least-embracing-parallel-process(O1))          'and'
          source-text( Least-embracing-parallel-process(O2))
          'in output statements'   source-text( O1)    'and '     source-text(O2)
          'respectively'
```

- This is an example of an universal assertion



Dynamic constraint - data race condition

```
PAR
    X := expr1
    ...
    X := expr2

EXISTS   Snapshot ::{D1: destination, D2: destination }
         ( location (D1) = location (D2) )

SAY (    'Attempt to assign to the same memory location'
         source-text(D1)    'and '     source-text(D2)
         'in two parallel processes:'
         source-text( Least-embracing-parallel-process(D1))
         'and'
         source-text( Least-embracing-parallel-process(D2))  )
```

- Yet another example of an universal debugging rule

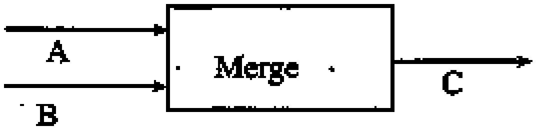
Variable X remains unchanged after each instance A call.

```
FOREACH   C: ex-instance :: ( instance-name IS 'A' )  FROM ex_program
          value-at-end ( C , 'X' ) = value-at-begin ( C , 'X')
```

Description of the process property of merging two streams:

"The number of input items equals the number of output items."

```
FOREACH P: ex-instance :: ( instance-name IS 'Merge' )   FROM ex_program
          CARD { ( input ::  ( channel IS 'A' ) | channel IS 'B' ) ) FROM  P} =
CARD {  output ::  ( channel IS 'C' ) FROM  P}
```



Performance measurement (in modeling mode)

```
SAY      'Total time is'
         +/ {| ABC: ex-instance ::( instance-name IS 'ABC')  FROM ex-program
                  APPLY duration(ABC) |}
```

Samples of possible profile request

```
SAY       'Total number of parallel processes executed is'
          CARD { ALL parallel-process    FROM    ex-program}


SAY      'Total number of assignments to the variable X executed is'
         CARD { ex-assignment :: ( destination IS 'X')     FROM ex-program }
```

References.

[Auguston, Fritzson 93] M.Auguston, P.Fritzson, PARFORMAN - an Assertion Language for Specifying Behavior when Debugging Parallel Applications, in *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing*, Gran Canaria, January 27-29, 1993, IEEE Computer Society Press.

[Auguston 94] Auguston M., A Language for Debugging Automation, in *Proceedings of 6th Int'l Conference on Software Engineering and Knowledge Engineering SEKE'94*, Jurmala, 1994, pp. 108-115

[Fritzson, Auguston, Shahmehri 94] P. Fritzson, M. Auguston, N. Shahmehri: Using Assertions in Declarative and Operational Models for Automated Debugging, *Journal of Systems and Software*, v.25, No 3, June 1994, pp. 223-239.

[Auguston 95] Mikhail Auguston, "Program Behavior Model Based on Event Grammar and its Application for Debugging Automation", in *Proceedings of the 2nd International Workshop on Automated and Algorithmic Debugging AADEBUG'95*, Saint-Malo, France, May 1995.

[Auguston, Fritzson 96] M.Auguston, P.Fritzson, PARFORMAN - an Assertion Language for Specifying Behavior when Debugging Parallel Applications, *International Journal on Software Engineering and Knowledge Engineering*, Vol. 6, No 4, 1996, pp.609-640.

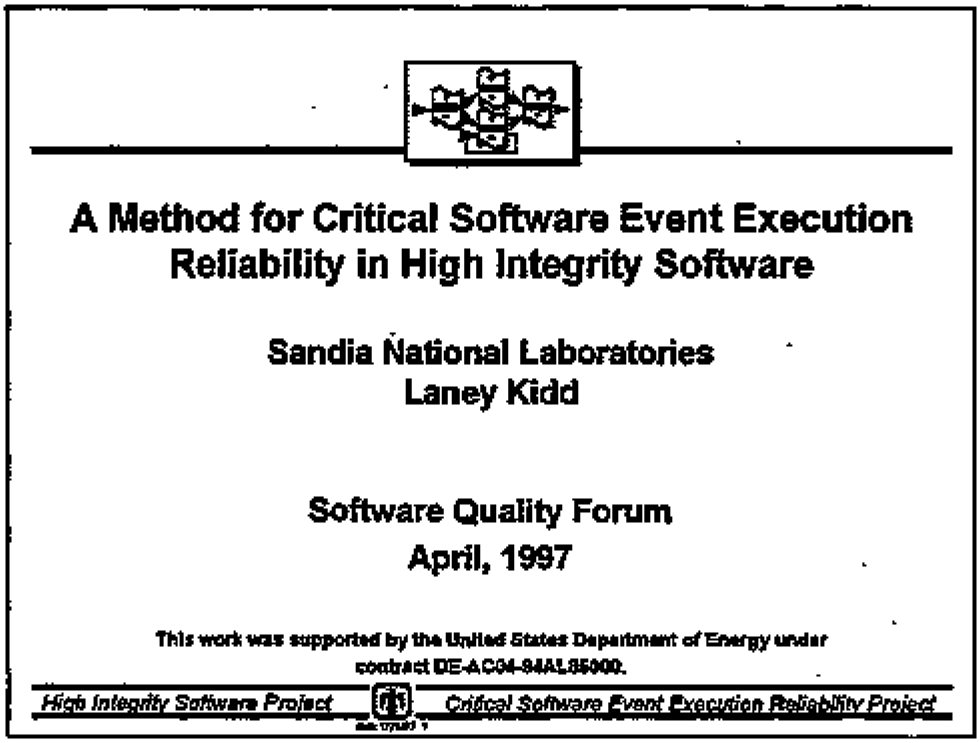
Experiments with the prototype implementation
of PASCAL assertion checker
have demonstrated some interesting features:

• different kinds of dynamic analysis can be described as an appropriate computations over the trace, e.g. debugging queries, assertion checking, profile measurement,

• computations over traces may provide values which otherwise can not be found in program states,

• informative and readable messages can be generated,

• universal assertions and debugging rules can be presented as computations over traces.

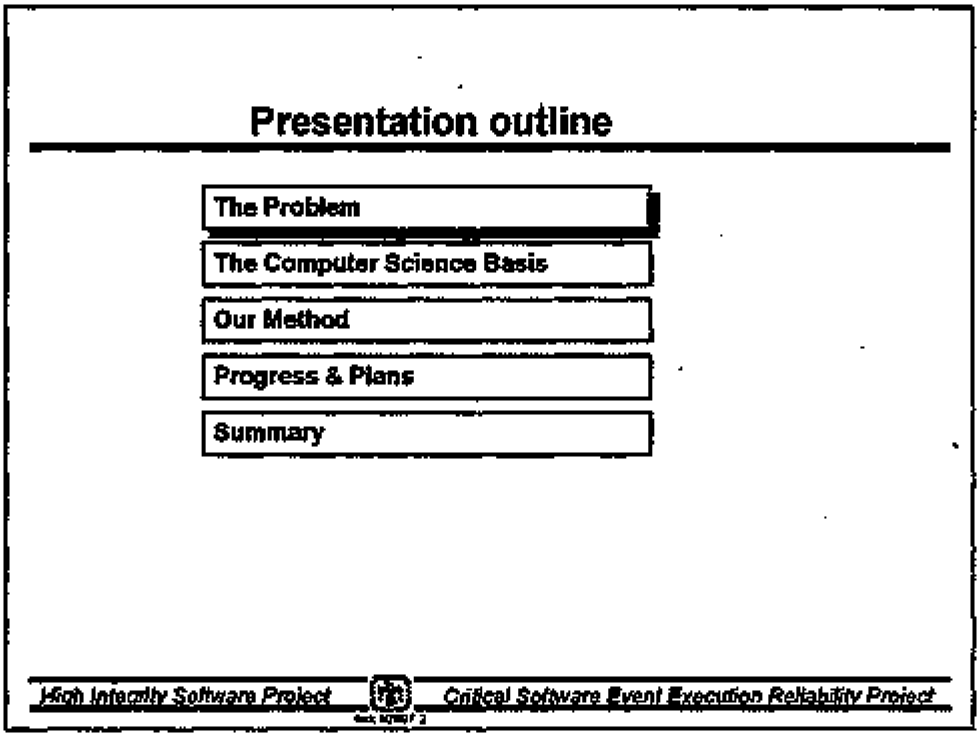


A Method for Critical Software Event Execution Reliability in High Integrity Software

Sandia National Laboratories
Laney Kidd

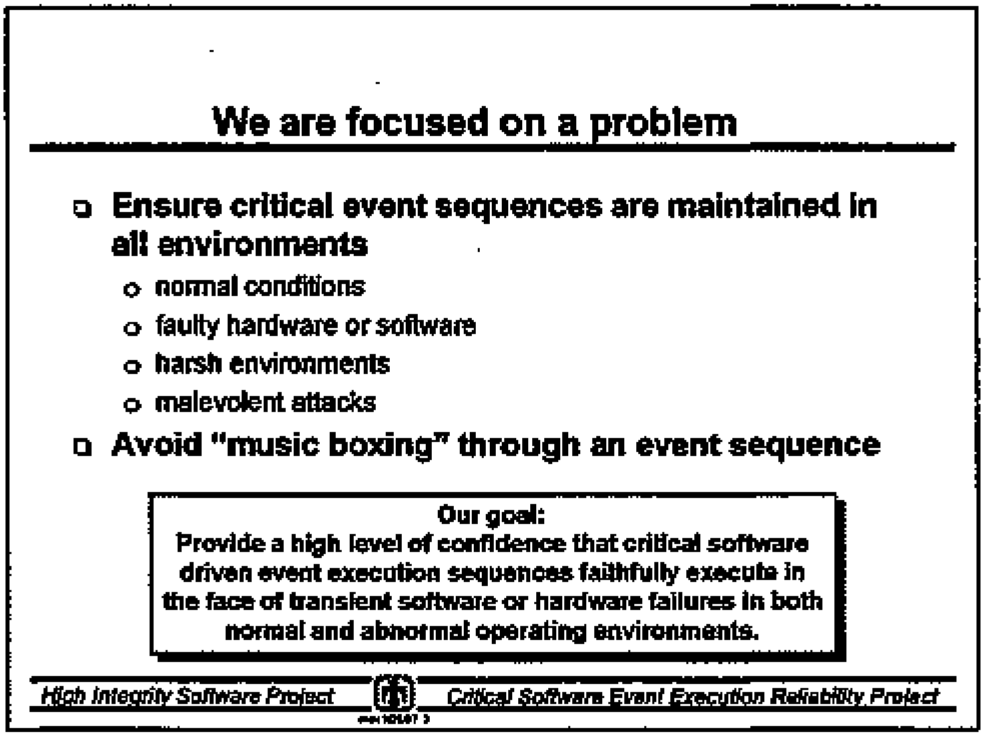
Software Quality Forum
April, 1997

This work was supported by the United States Department of Energy under contract DE-AC04-94AL85000.



Presentation outline

- The Problem
- The Computer Science Basis
- Our Method
- Progress & Plans
- Summary

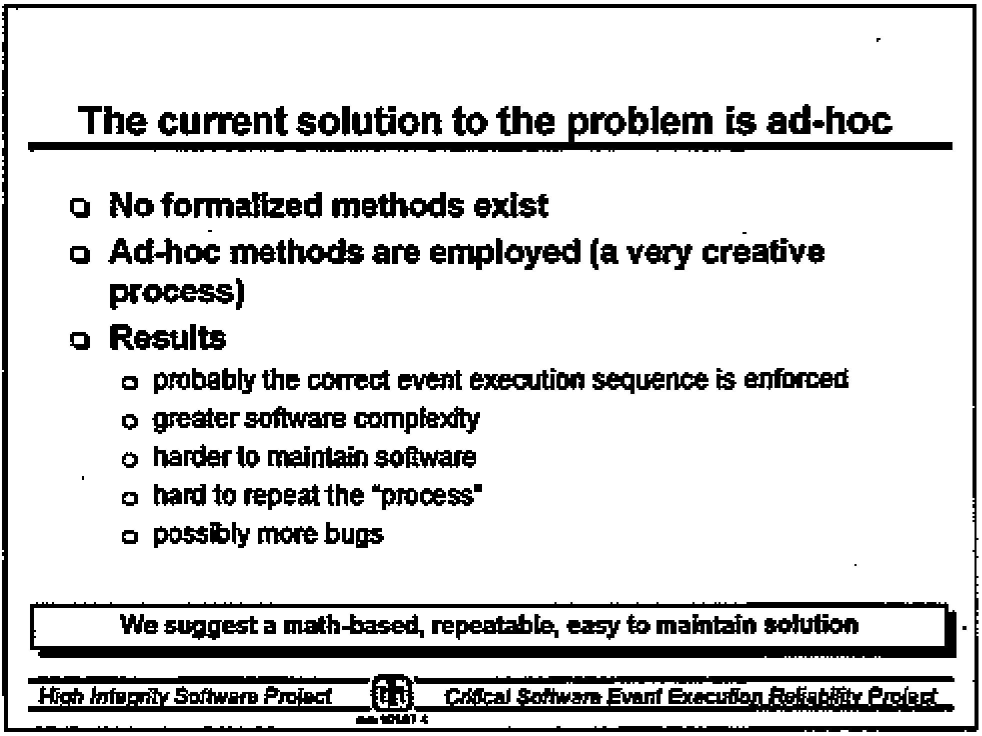


We are focused on a problem

- Ensure critical event sequences are maintained in all environments
  - normal conditions
  - faulty hardware or software
  - harsh environments
  - malevolent attacks
- Avoid "music boxing" through an event sequence

Our goal:
Provide a high level of confidence that critical software driven event execution sequences faithfully execute in the face of transient software or hardware failures in both normal and abnormal operating environments.

High Integrity Software Project *Critical Software Event Execution Reliability Project*

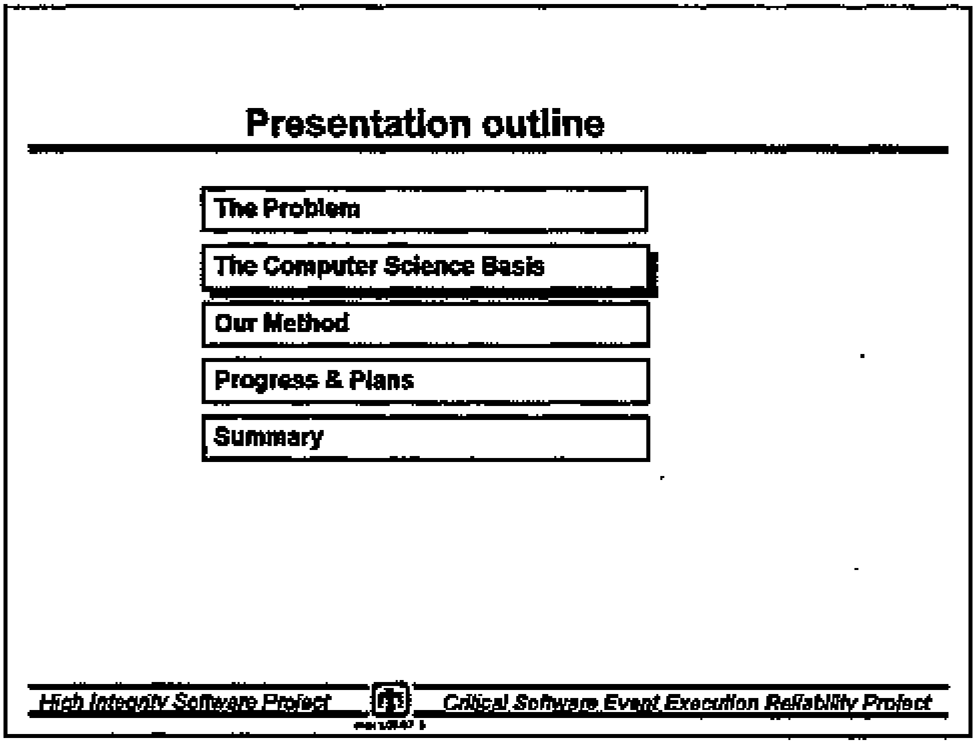


The current solution to the problem is ad-hoc

- No formalized methods exist
- Ad-hoc methods are employed (a very creative process)
- Results
  - probably the correct event execution sequence is enforced
  - greater software complexity
  - harder to maintain software
  - hard to repeat the "process"
  - possibly more bugs

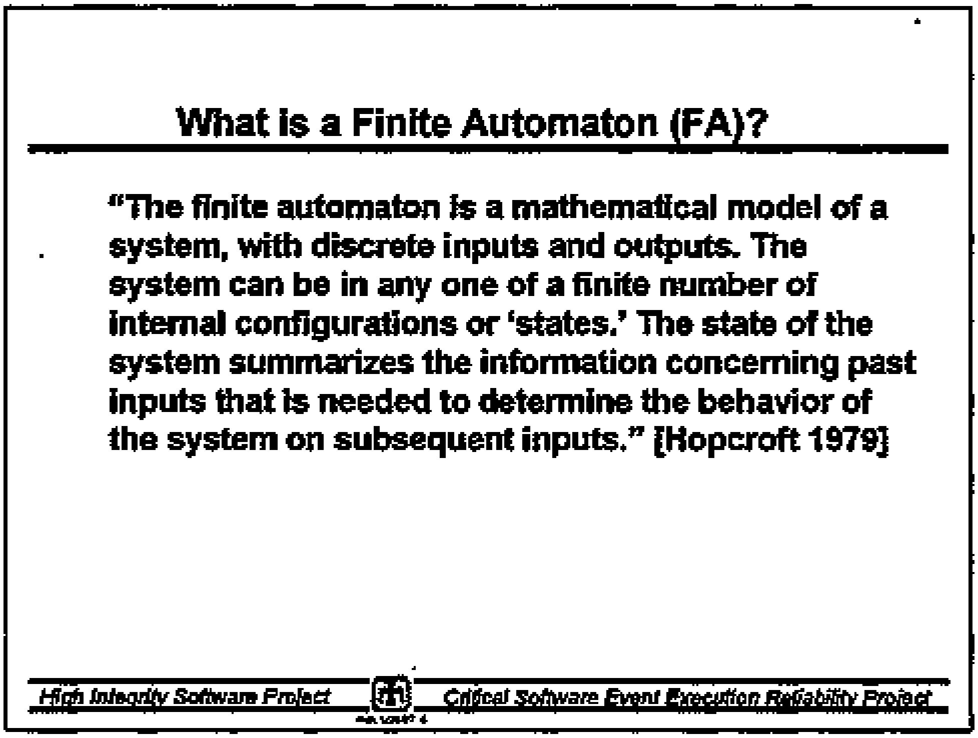
We suggest a math-based, repeatable, easy to maintain solution

High Integrity Software Project *Critical Software Event Execution Reliability Project*



Presentation outline

- The Problem
- The Computer Science Basis
- Our Method
- Progress & Plans
- Summary

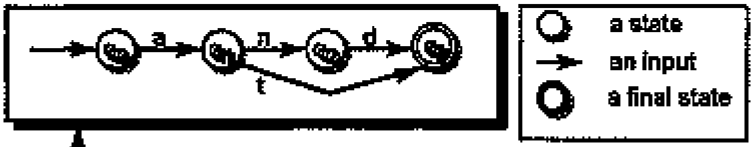


What is a Finite Automaton (FA)?

"The finite automaton is a mathematical model of a system, with discrete inputs and outputs. The system can be in any one of a finite number of internal configurations or 'states.' The state of the system summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs." [Hopcroft 1979]

Here is an example of an FA

Transition Diagram



a state

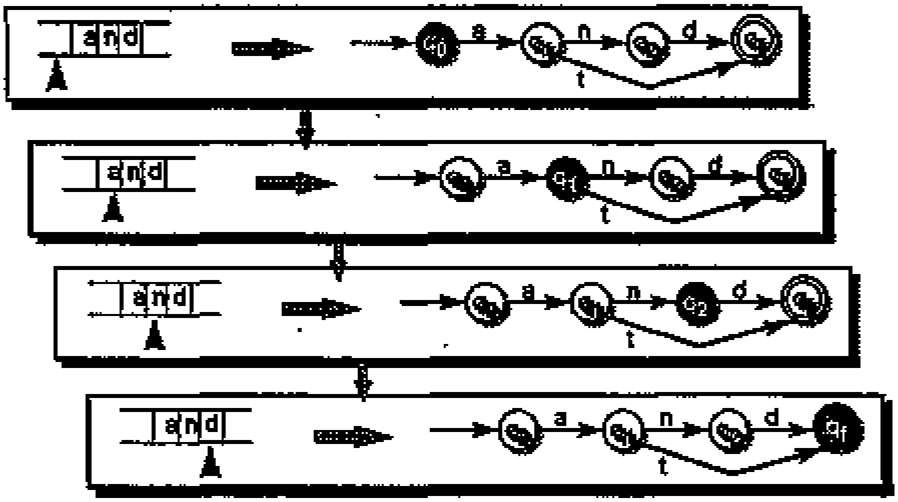
an input

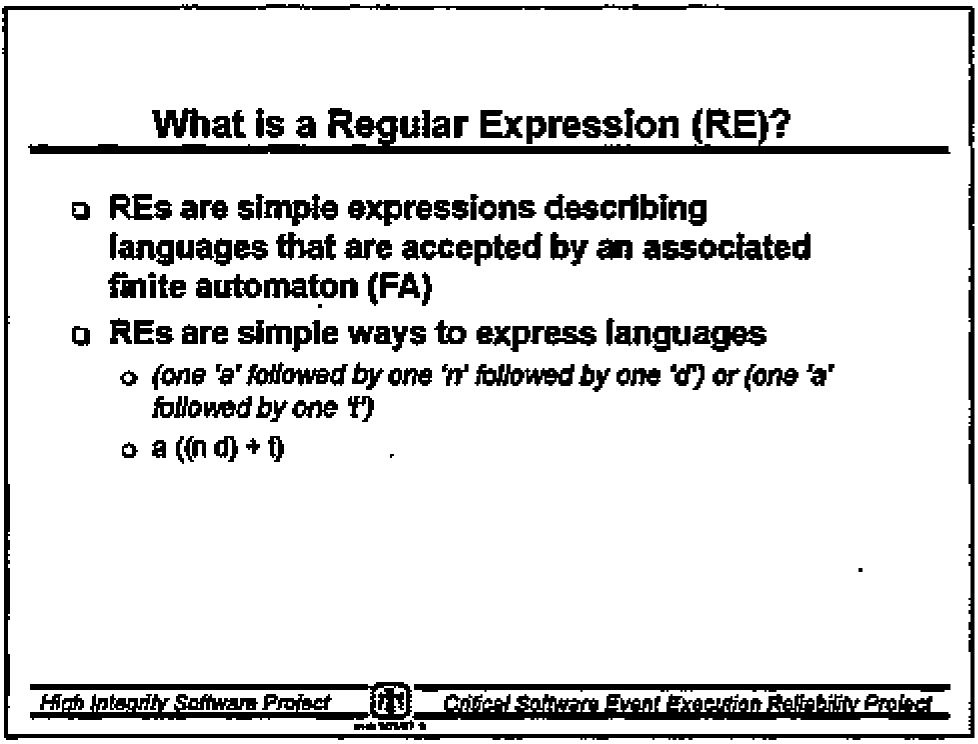
a final state

Transition Table

States in Q	inputs in Σ			
	a	n	d	t
q_0	q_1	0	0	0
q_1	0	q_2	0	q_f
q_2	0	0	q_f	0
q_f	0	0	0	0

Here is an example FA execution path

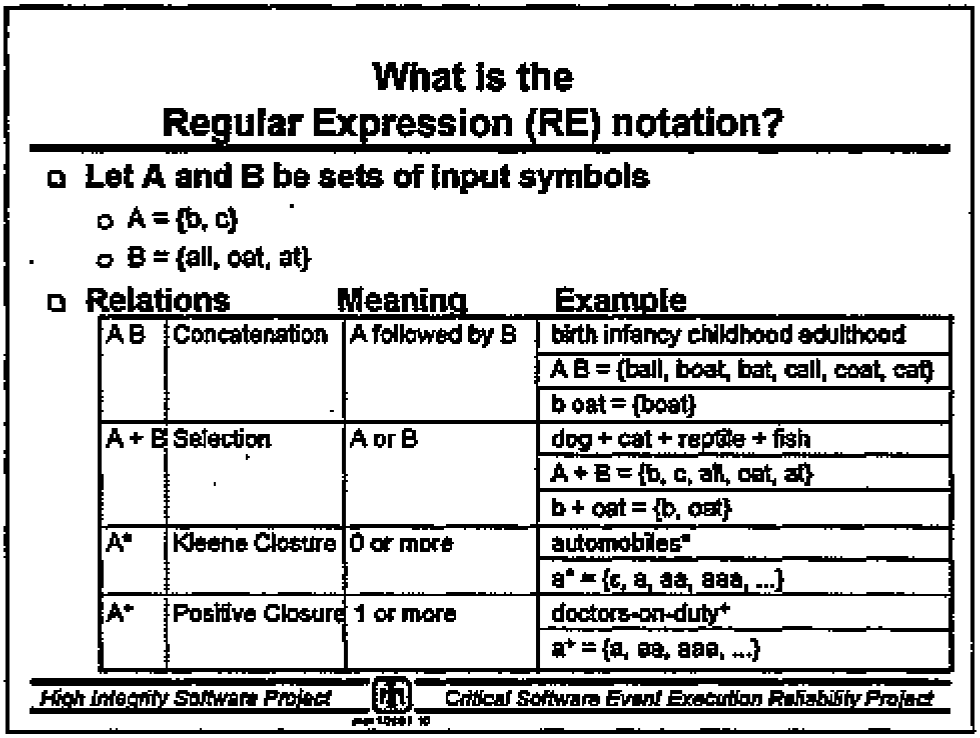




What is a Regular Expression (RE)?

- REs are simple expressions describing languages that are accepted by an associated finite automaton (FA)
- REs are simple ways to express languages
  - *(one 'a' followed by one 'n' followed by one 'd') or (one 'a' followed by one 'f')*
  - a ((n d) + f)

High Integrity Software Project *Critical Software Event Execution Reliability Project*



What is the Regular Expression (RE) notation?

- Let A and B be sets of input symbols
  - A = {b, c}
  - B = {all, oat, at}
- Relations Meaning Example

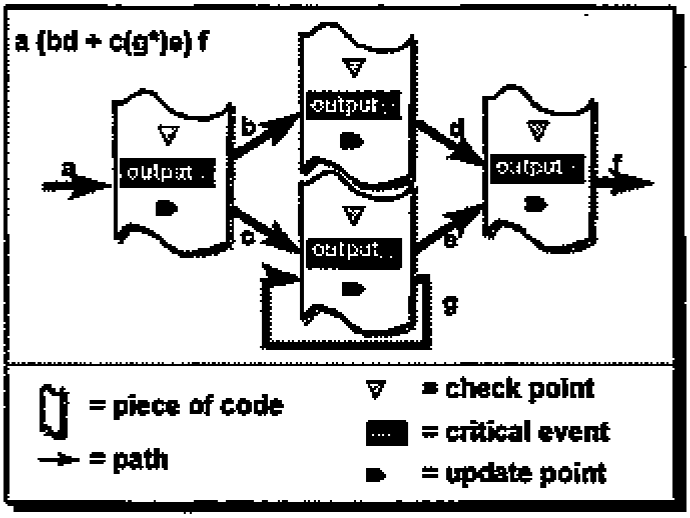
Relations		Meaning	Example
A B	Concatenation	A followed by B	birth infancy childhood adulthood
			A B = {ball, boat, bat, call, coat, cat}
			b oat = {boat}
A + B	Selection	A or B	dog + cat + reptile + fish
			A + B = {b, c, all, oat, at}
			b + oat = {b, oat}
A^*	Kleene Closure	0 or more	automobiles*
			$a^* = \{\varepsilon, a, aa, aaa, ...\}$
A^+	Positive Closure	1 or more	doctors-on-duty$^+$
			$a^+ = \{a, aa, aaa, ...\}$

High Integrity Software Project *Critical Software Event Execution Reliability Project*

Presentation outline

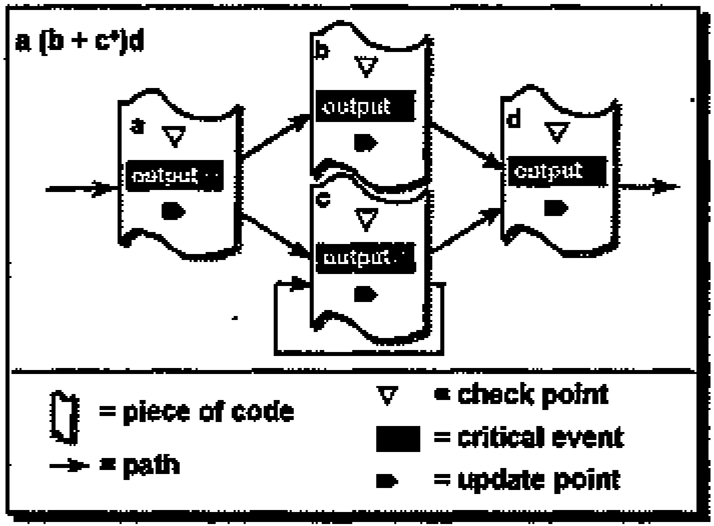
- The Problem
- The Computer Science Basis
- Our Method
- Progress & Plans
- Summary

Think of a path through pieces of code


a (bd + c(g*)e) f

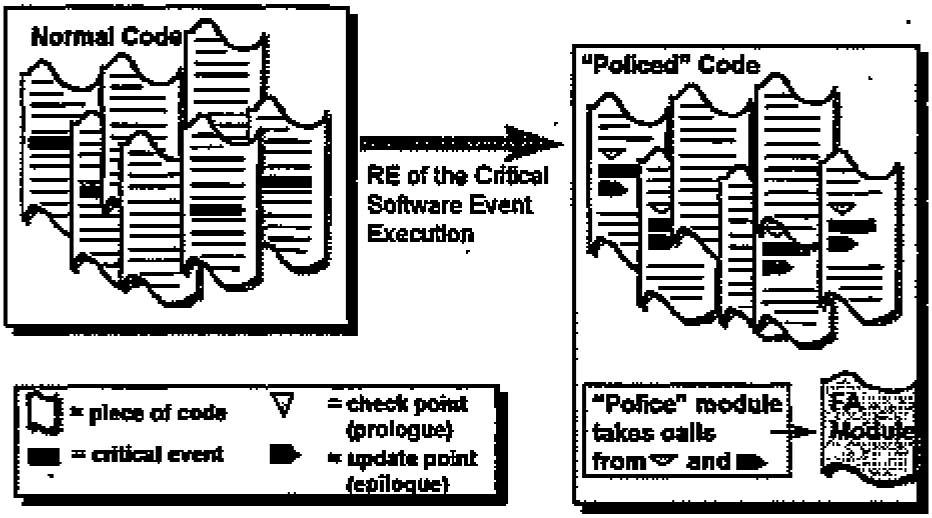
- ▯ = piece of code
- → = path
- ▽ = check point
- ■ = critical event
- ● = update point

Or, think of an event sequence through code



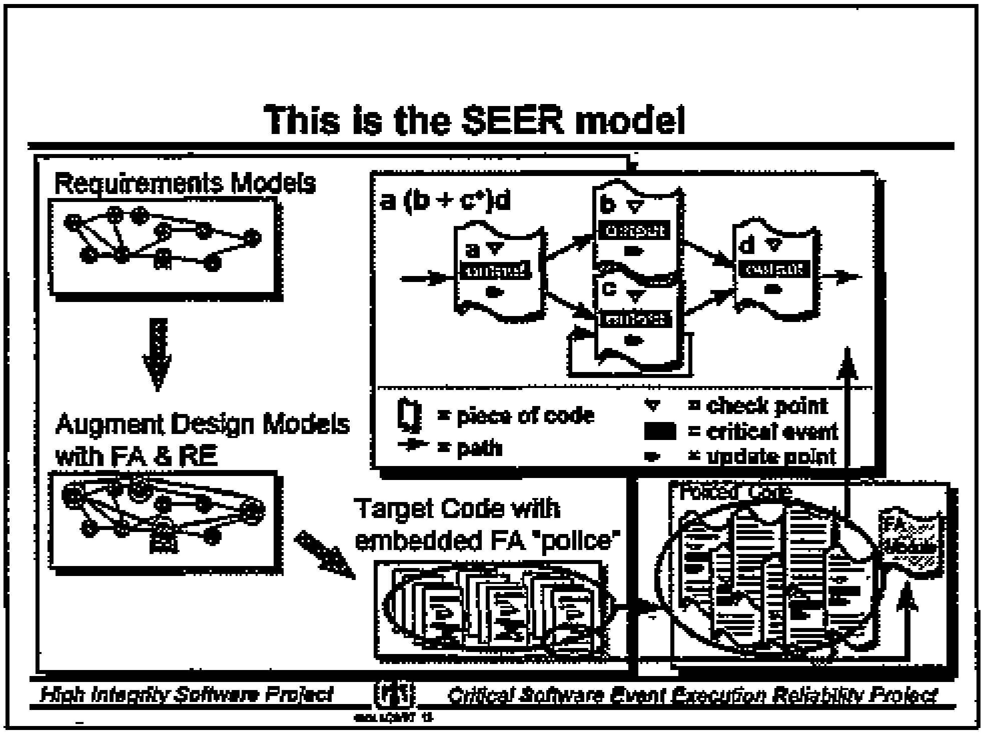
High Integrity Software Project — Critical Software Event Execution Reliability Project

The SEER method adds prologues, epilogues, and a Finite Automaton implementation module

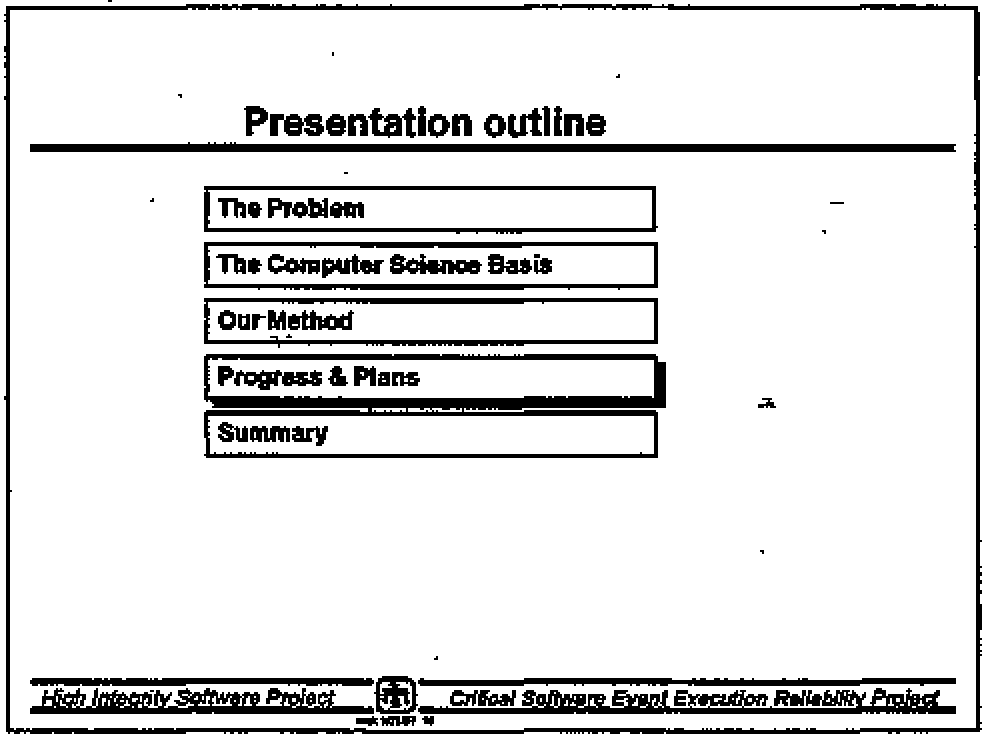


Normal Code
RE of the Critical Software Event Execution
"Policed" Code
= piece of code
= critical event
= check point (prologue)
= update point (epilogue)
"Police" module takes calls from ▽ and ▶
FA Module

High Integrity Software Project — Critical Software Event Execution Reliability Project



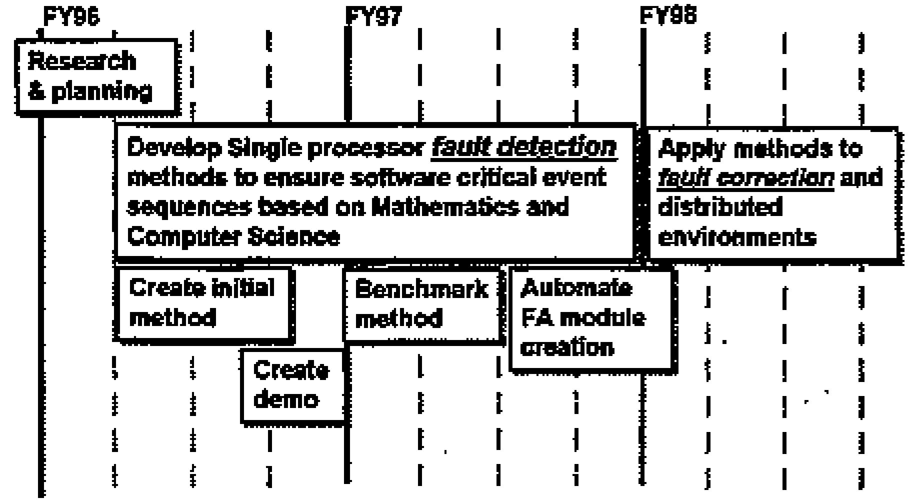
This is the SEER model



Presentation outline

- The Problem
- The Computer Science Basis
- Our Method
- Progress & Plans
- Summary

Our progress and plans at a glance


FY96
FY97
FY98
Research & planning
Develop Single processor *fault detection* methods to ensure software critical event sequences based on Mathematics and Computer Science
Apply methods to *fault correction* and distributed environments
Create initial method
Benchmark method
Automate FA module creation
Create demo

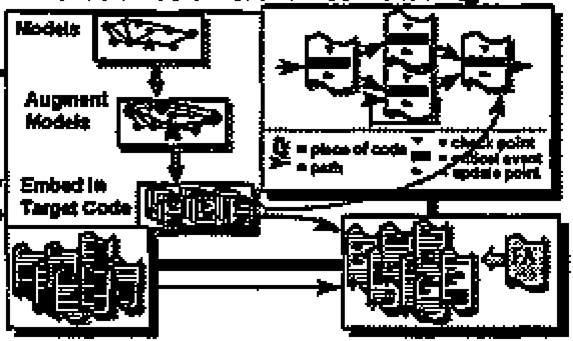
Publications to date

- "Ensuring Critical Event Sequences in High Integrity Software by Applying Path Expressions," *Proceedings of the 14th International System Safety Conference,* Albuquerque, NM, August 1996, pp. 6C2-1 - 6C2-14.
- "Ensuring Critical Event Sequences in High Consequence Computer Based Systems as Inspired by Path Expressions", *Proceedings of the International Conference and Workshop on Engineering of Computer Based Systems (ECBS),* Monterey, CA, March 1997.

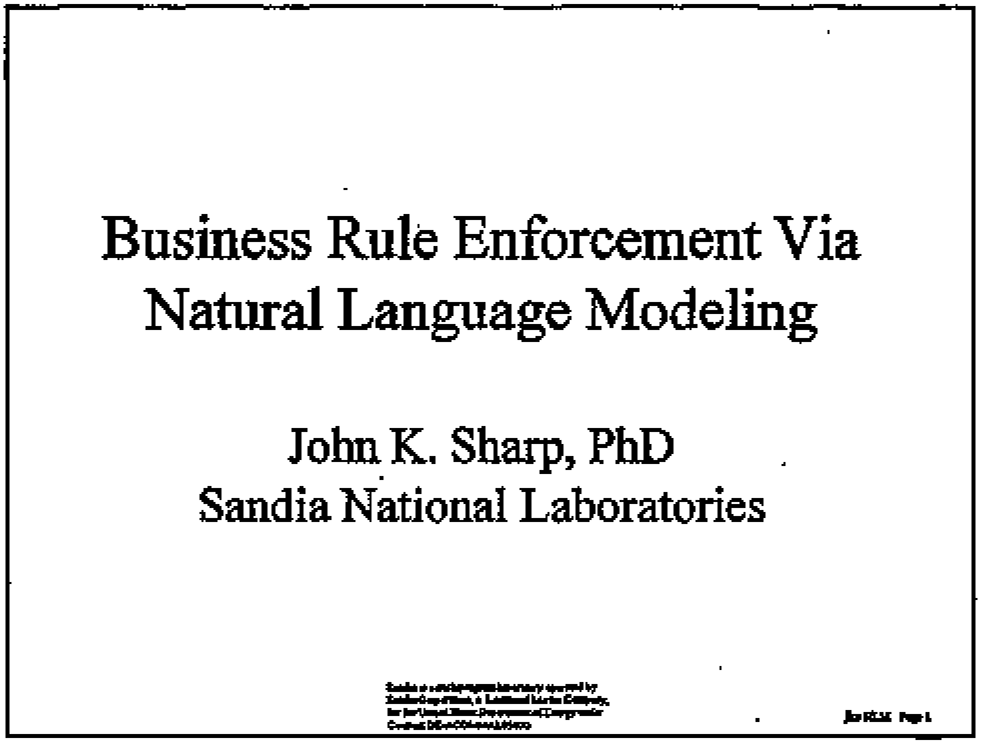
Presentation outline

- The Problem
- The Computer Science Basis
- Our Method
- Progress & Plans
- Summary

Critical Software Event Execution Reliability (SEER) Project



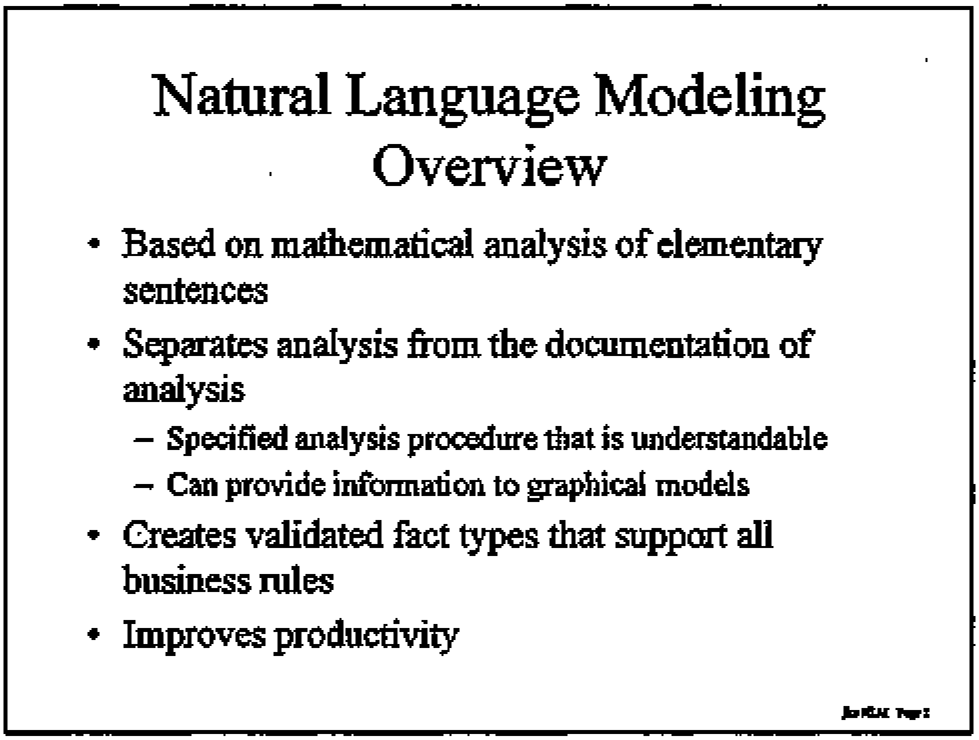
- □ Why:
  - ○ Software developers employ ad-hoc, complex, and potentially bug-infested methods to ensure critical software event sequences.
- □ What:
  - ○ Provide a high level of confidence that critical software driven event execution sequences are maintained in the face of transient software or hardware failures in both normal and abnormal operating environments.
- □ How:
  - ○ Develop a repeatable, mathematical based solution using finite automata (FA) to develop a method to enforce critical event execution sequences.
- □ Point of contact:
  - ○ Laney Kidd of Sandia



Business Rule Enforcement Via Natural Language Modeling

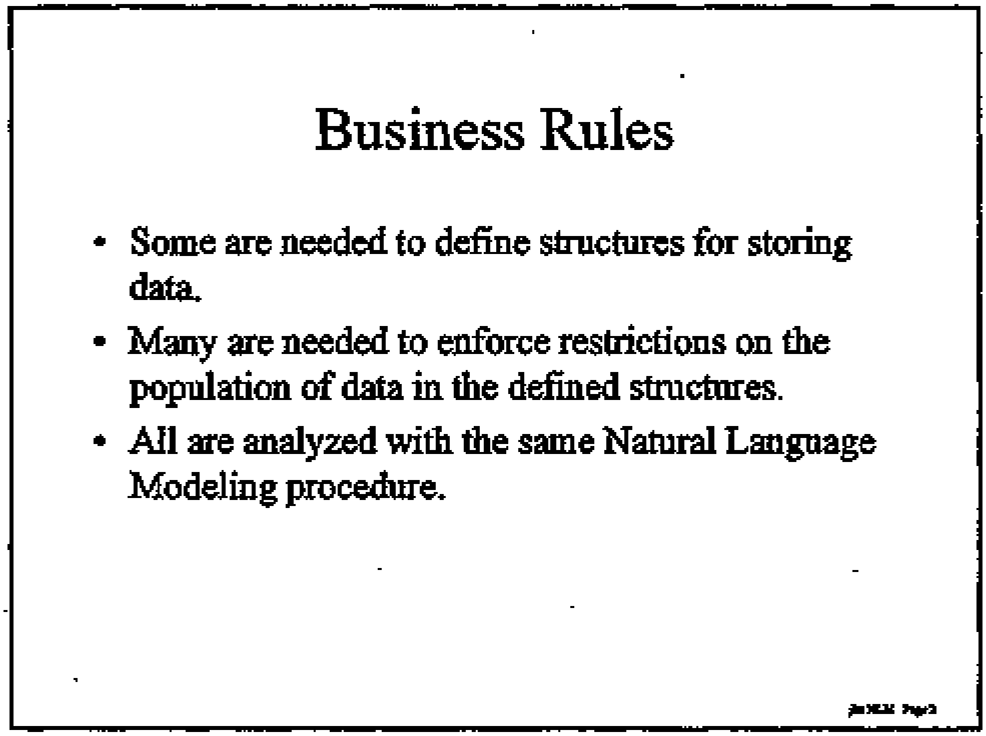
John K. Sharp, PhD
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under
Contract DE-AC04-94AL85000



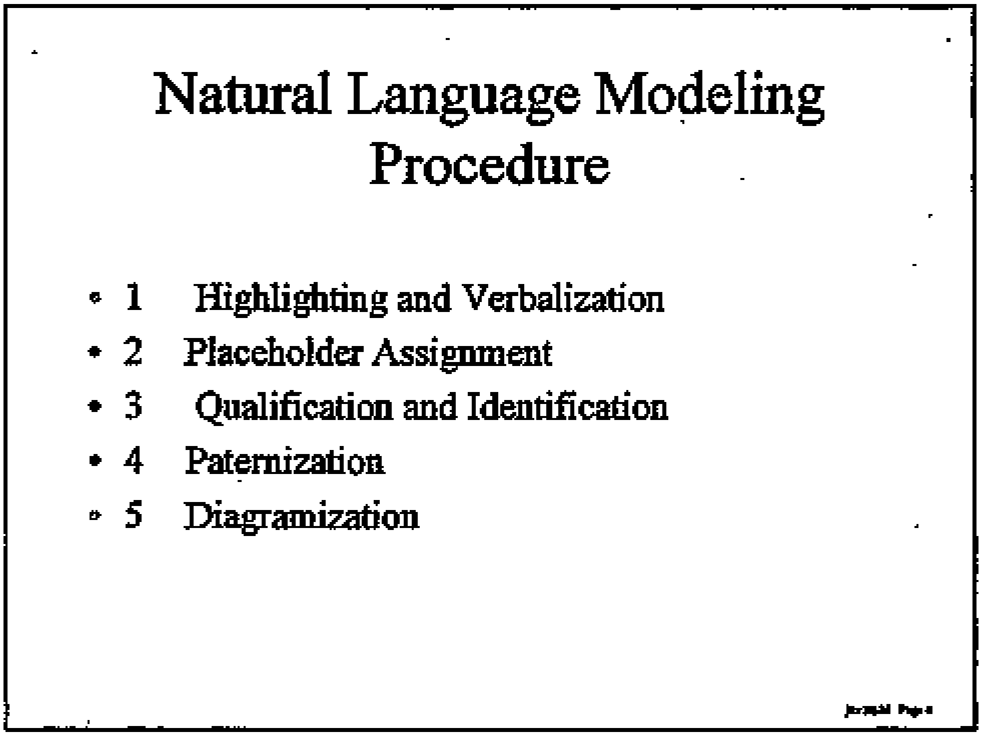
Natural Language Modeling Overview

- Based on mathematical analysis of elementary sentences
- Separates analysis from the documentation of analysis
  - Specified analysis procedure that is understandable
  - Can provide information to graphical models
- Creates validated fact types that support all business rules
- Improves productivity



Business Rules

- Some are needed to define structures for storing data.
- Many are needed to enforce restrictions on the population of data in the defined structures.
- All are analyzed with the same Natural Language Modeling procedure.



Natural Language Modeling Procedure

- 1 Highlighting and Verbalization
- 2 Placeholder Assignment
- 3 Qualification and Identification
- 4 Paternization
- 5 Diagramization

Examples Using the Natural Language Modeling Procedure

- Movie Marquee
- Sports Team
- Professor Assignment

Movie Marquee

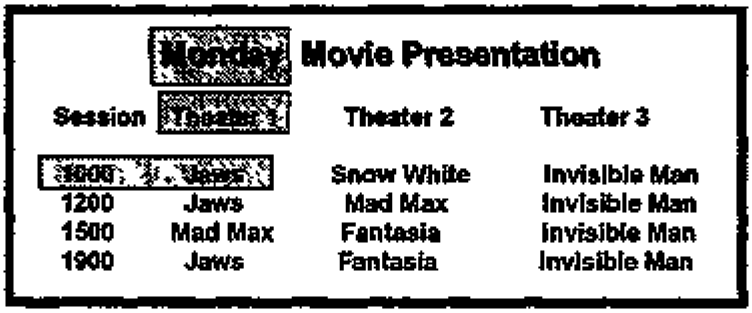
Monday Movie Presentation

Session	Theater 1	Theater 2	Theater 3
1000	Jaws	Snow White	Invisible Man
1200	Jaws	Mad Max	Invisible Man
1500	Mad Max	Fantasia	Invisible Man
1900	Jaws	Fantasia	Invisible Man

Natural Language Modeling Procedure

1 Highlighting and Verbalization

•Verbalization and highlighting is done by highlighting a limited example of information in the subject area and asking the subject matter expert to create a sentence.



Monday Movie Presentation

Session	Theater 1	Theater 2	Theater 3
1000	Jaws	Snow White	Invisible Man
1200	Jaws	Mad Max	Invisible Man
1500	Mad Max	Fantasia	Invisible Man
1900	Jaws	Fantasia	Invisible Man

Verbalization: Jaws is showing Monday in theater 1 at 1000.

Natural Language Modeling Procedure

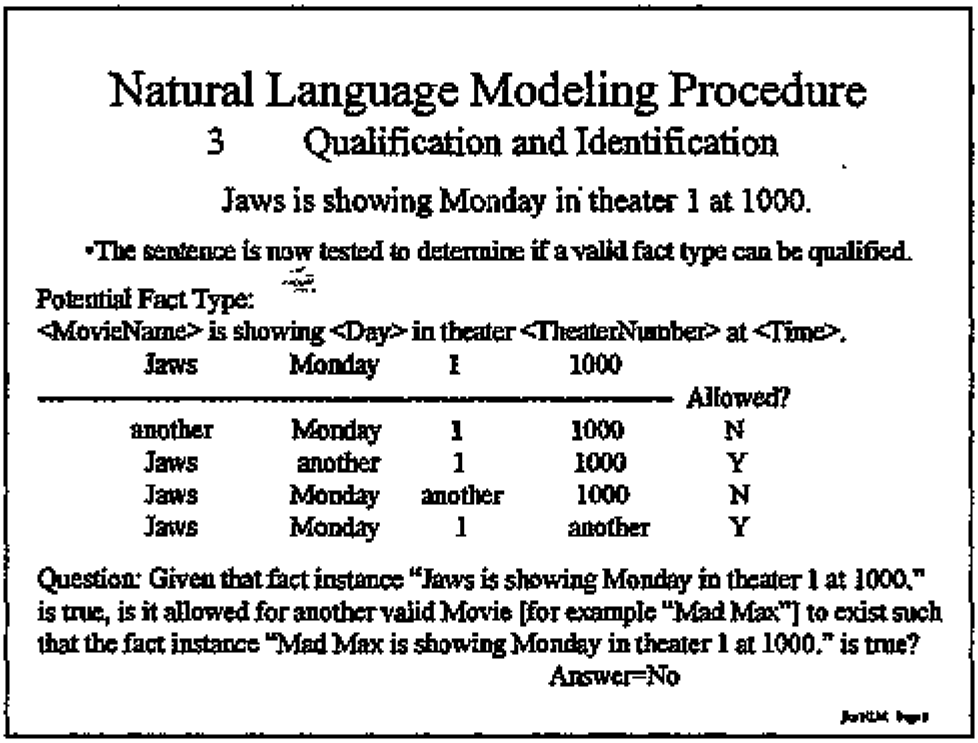
2 Placeholder Assignment

•Placeholder assignment identifies the part(s) of a sentence that can have values that change.

Jaws is showing Monday in theater 1 at 1000.
Mad Max is showing Tuesday in theater 2 at 1200.

•The values that can change are (Jaws and Mad Max), (Monday and Tuesday), (1 and 2), and (1000 and 1200).

Jaws is showing Monday in theater 1 at 1000.
Mad Max " " Tuesday " " 2 " 1200.



Natural Language Modeling Procedure

3 Qualification and Identification

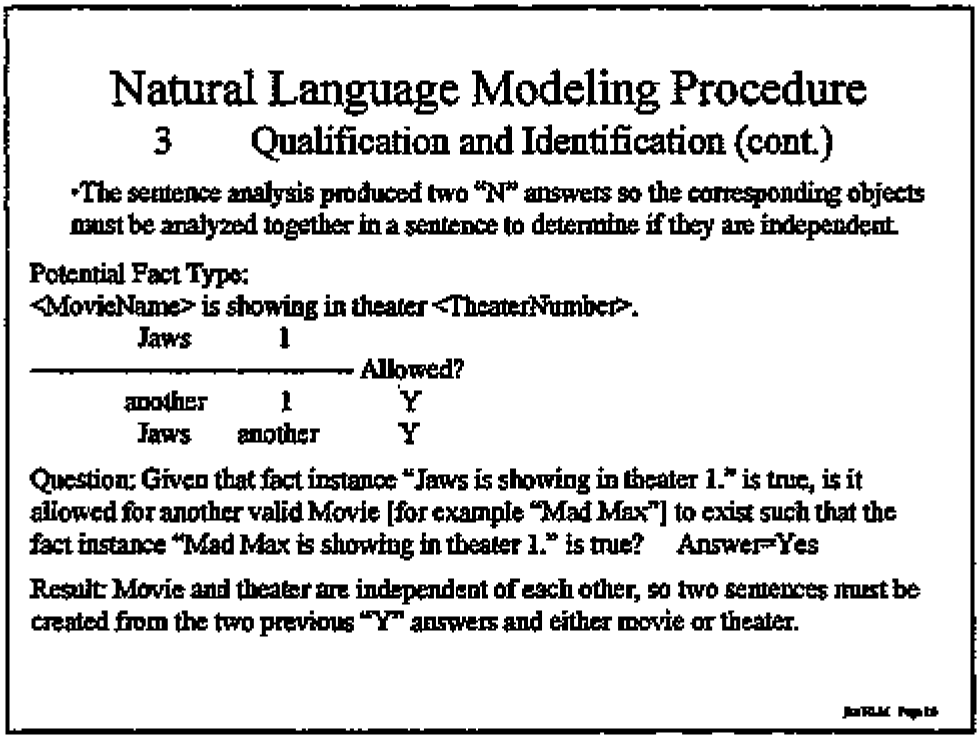
Jaws is showing Monday in theater 1 at 1000.

•The sentence is now tested to determine if a valid fact type can be qualified.

Potential Fact Type:
<MovieName> is showing <Day> in theater <TheaterNumber> at <Time>.

Jaws	Monday	1	1000	Allowed?
another	Monday	1	1000	N
Jaws	another	1	1000	Y
Jaws	Monday	another	1000	N
Jaws	Monday	1	another	Y

Question: Given that fact instance "Jaws is showing Monday in theater 1 at 1000." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing Monday in theater 1 at 1000." is true?
Answer=No



Natural Language Modeling Procedure

3 Qualification and Identification (cont.)

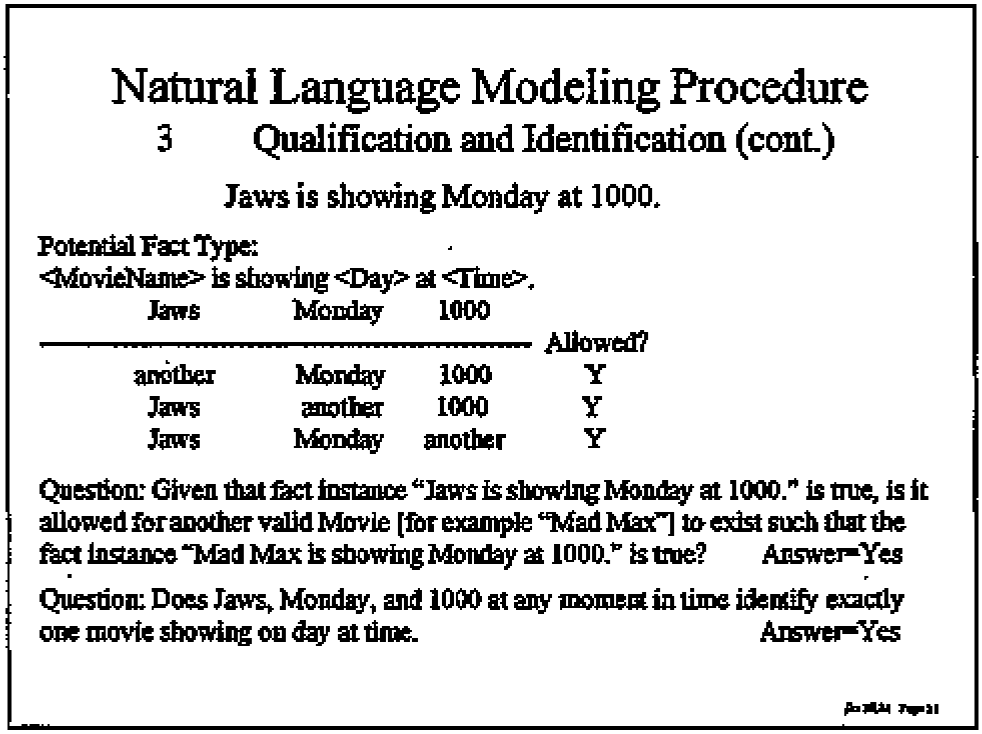
•The sentence analysis produced two "N" answers so the corresponding objects must be analyzed together in a sentence to determine if they are independent.

Potential Fact Type:
<MovieName> is showing in theater <TheaterNumber>.

Jaws	1	Allowed?
another	1	Y
Jaws	another	Y

Question: Given that fact instance "Jaws is showing in theater 1." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing in theater 1." is true? Answer=Yes

Result: Movie and theater are independent of each other, so two sentences must be created from the two previous "Y" answers and either movie or theater.



Natural Language Modeling Procedure

3 Qualification and Identification (cont.)

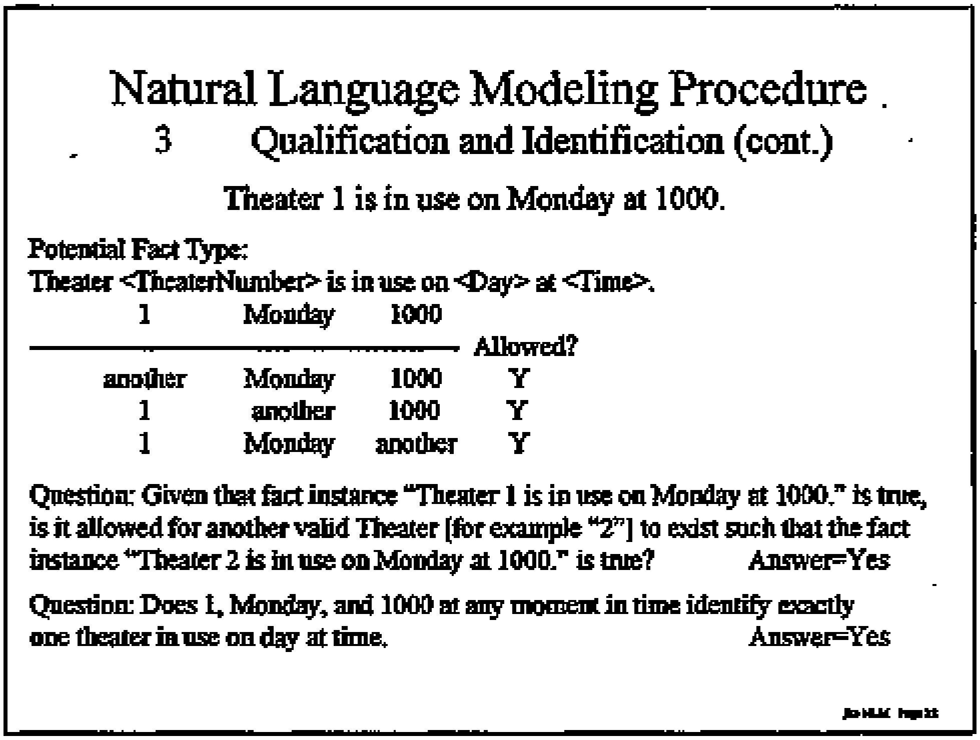
Jaws is showing Monday at 1000.

Potential Fact Type:
<MovieName> is showing <Day> at <Time>.

Jaws	Monday	1000	Allowed?
another	Monday	1000	Y
Jaws	another	1000	Y
Jaws	Monday	another	Y

Question: Given that fact instance "Jaws is showing Monday at 1000." is true, is it allowed for another valid Movie [for example "Mad Max"] to exist such that the fact instance "Mad Max is showing Monday at 1000." is true? Answer=Yes

Question: Does Jaws, Monday, and 1000 at any moment in time identify exactly one movie showing on day at time. Answer=Yes



Natural Language Modeling Procedure

3 Qualification and Identification (cont.)

Theater 1 is in use on Monday at 1000.

Potential Fact Type:
Theater <TheaterNumber> is in use on <Day> at <Time>.

1	Monday	1000	Allowed?
another	Monday	1000	Y
1	another	1000	Y
1	Monday	another	Y

Question: Given that fact instance "Theater 1 is in use on Monday at 1000." is true, is it allowed for another valid Theater [for example "2"] to exist such that the fact instance "Theater 2 is in use on Monday at 1000." is true? Answer=Yes

Question: Does 1, Monday, and 1000 at any moment in time identify exactly one theater in use on day at time. Answer=Yes

Natural Language Modeling Procedure

4 Paternization

•Paternization is the specification of the general fact type that can be populated with instances.

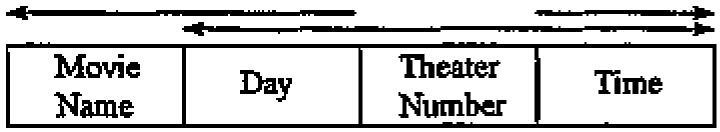
FT1 <MovieName> is showing <Day> in theater <TheaterNumber> at <Time>.

Natural Language Modeling Procedure

5 Diagramization

•Diagramization presents a relational diagram that can be populated with instances and read using the associated fact type(s).

Movie_Day_Time



Movie Name	Day	Theater Number	Time
Jaws	Monday	1	1000
Snow White	Monday	2	1000
Mad Max	Tuesday	1	1200

FT1 <MovieName> is showing <Day> in theater <TheaterNumber> at <Time>.

Movie Marquee
Enforced Business Rules

- 1 Only one movie can be shown at a time in a theater.

- 2 Only one copy of a video tape will be leased at any time.

Sports Team
Problem Statement

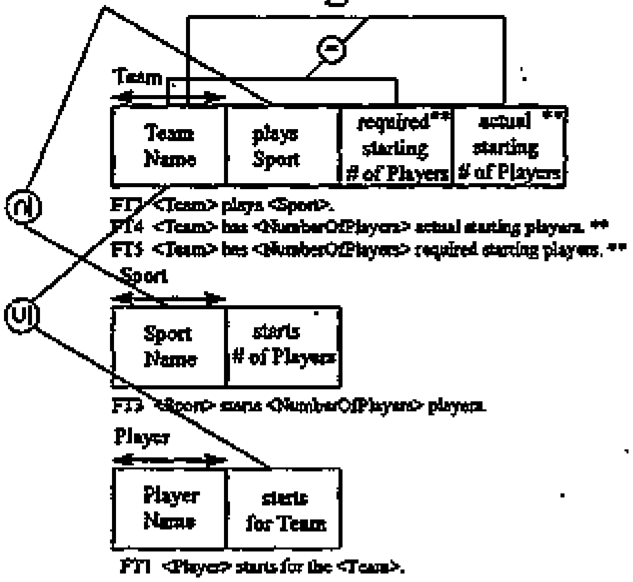
A player can start for only one team. A team plays only one sport. A sport has a required number of starting players. A team must start the number of players required for the sport the team plays.

Sports Team
Fact Types

FT1 <Player> starts for the <Team>.
FT2 <Team> plays <Sport>.
FT3 <Sport> starts <NumberOfPlayers> players.
FT4 <Team> has <NumberOfPlayers> actual starting players. **
FT5 <Team> has <NumberOfPlayers> required starting players. **

** derived facts

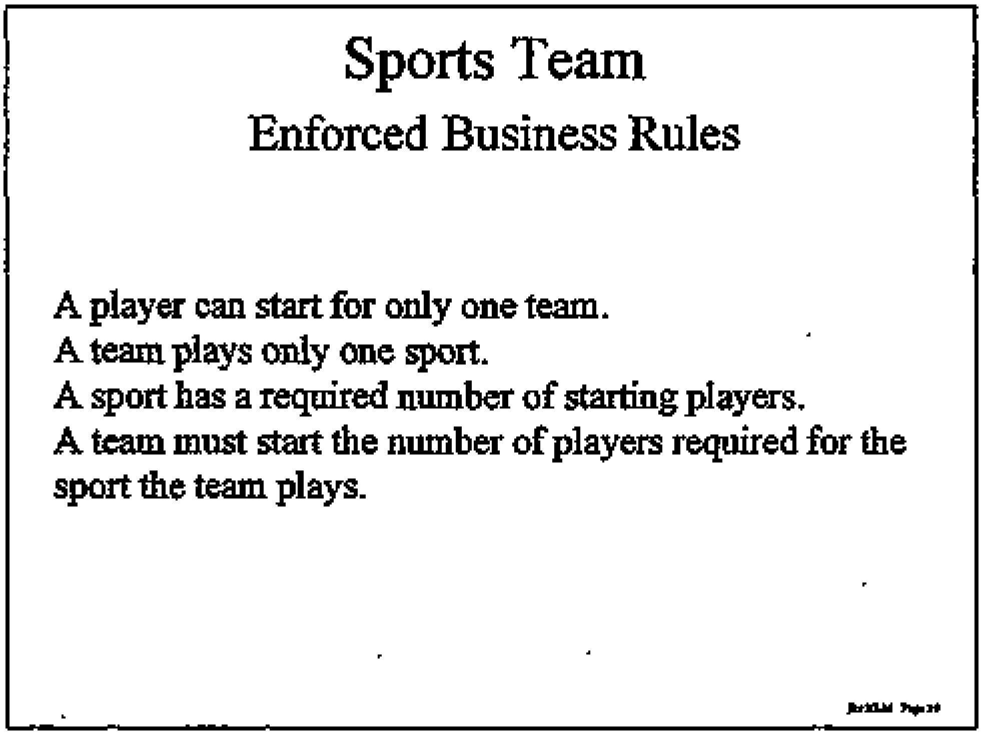
Sports Team
Diagram



FT2 <Team> plays <Sport>.
FT4 <Team> has <NumberOfPlayers> actual starting players. **
FT5 <Team> has <NumberOfPlayers> required starting players. **

FT3 <Sport> starts <NumberOfPlayers> players.

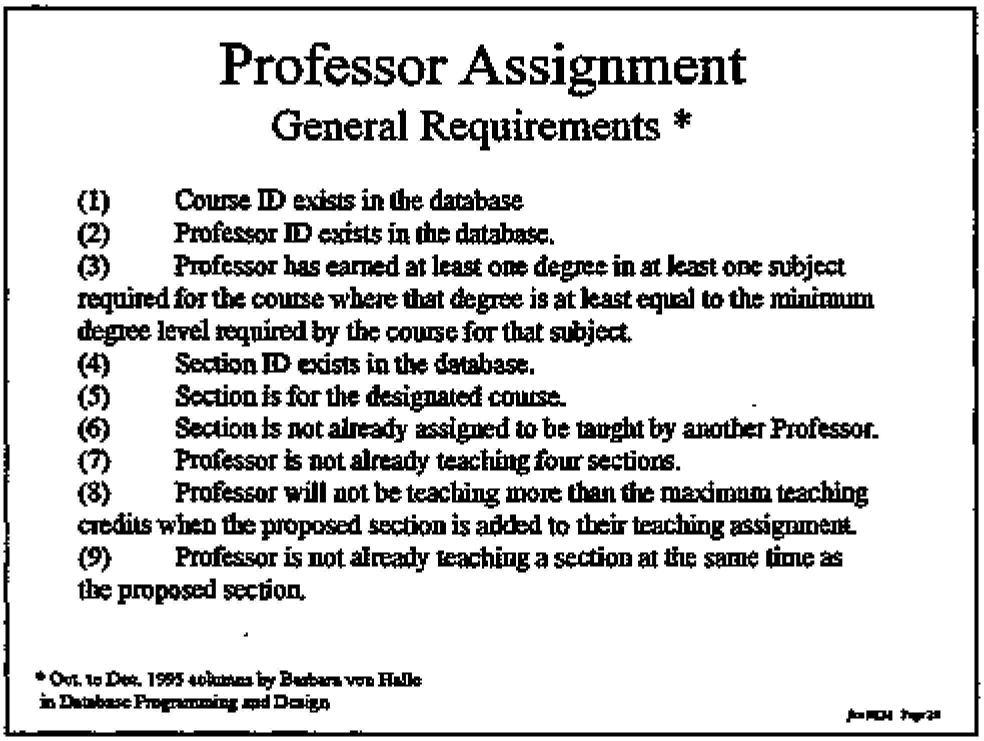
FT1 <Player> starts for the <Team>.



Sports Team

Enforced Business Rules

A player can start for only one team.
A team plays only one sport.
A sport has a required number of starting players.
A team must start the number of players required for the sport the team plays.



Professor Assignment

General Requirements *

(1) Course ID exists in the database

(2) Professor ID exists in the database.

(3) Professor has earned at least one degree in at least one subject required for the course where that degree is at least equal to the minimum degree level required by the course for that subject.

(4) Section ID exists in the database.

(5) Section is for the designated course.

(6) Section is not already assigned to be taught by another Professor.

(7) Professor is not already teaching four sections.

(8) Professor will not be teaching more than the maximum teaching credits when the proposed section is added to their teaching assignment.

(9) Professor is not already teaching a section at the same time as the proposed section.

* Oct. to Dec. 1995 columns by Barbara von Halle in Database Programming and Design

Professor Assignment

Instances**

Call No	Department Prefix	Course	Section No.	Course Title No.	Credit	Day	Time	Building	Room	Instructor
14077	MATH	121	001	College Algebra	03	M W F	0800-0850	M H	102	Staff
12615	MATH	145	004	Intro to Prob & Stat	03	T R	1100-1215	M H	120	W. Zimmer

** 1995-96 UNM course catalog

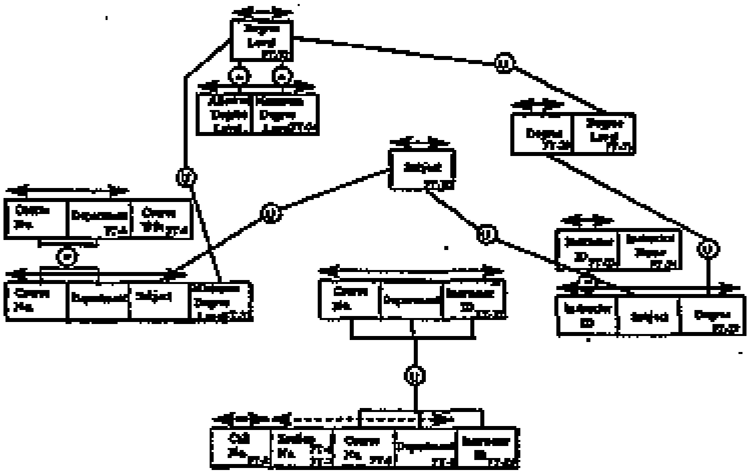
Professor Assignment

Fact Types (partial list)

FT-1 Call No. <Call No.> is in the <Department Prefix> department.
FT-2 Call No. <Call No.> exists.
FT-4 Course No. <Course No.> exists in the <Department Prefix> department.
FT-5 Call No. <Call No.> is for Course No. <Course No.> in the <Department Prefix> department.
FT-6 Section No. <Section No.> of Course No. <Course No.> is offered in the <Department Prefix> department.
FT-7 Call No. <Call No.> is for Section No. <Section No.> of Course No. <Course No.> offered in the <Department Prefix> department.
FT-8 Course No. <Course No.> in the <Department Prefix> department has the course title <Course Title>,
FT-24 Instructor <Instructor ID> has the name <Instructor Name>.
FT-25 Instructor <Instructor ID> exists.
FT-26 Call No. <Call No.> is assigned to Instructor <Instructor ID>.
FT-28 Instructor <Instructor ID> has earned a <Degree> in <Subject>.
FT-29 <Degree> degree exists.
FT-30 <Subject> subject area exists.
FT-31 <Department Prefix> <Course ID> requires a minimum of a <Minimum Degree Level> degree in <Subject>.
FT-32 <Degree Level> degree level exists.
FT-33 Instructor <Instructor ID> is allowed to teach <Department Prefix> <Course ID>.**
FT-34 <Allowed Degree Level> degree level can be substituted for a <Minimum Degree Level> degree level.
FT-35 <Degree> degree is at a <Degree Level> degree level.

Professor Assignment

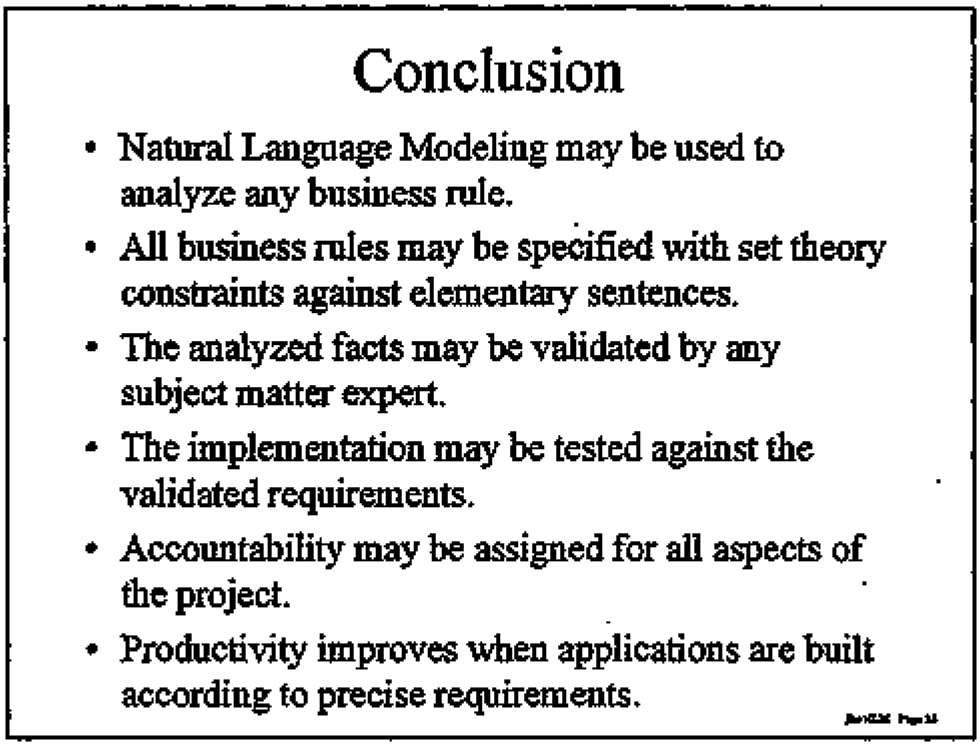
Diagram (partial)



Professor Assignment

Enforced Business Rules

Professor has earned at least one degree in at least one subject required for the course where that degree is at least equal to the minimum degree level required by the course for that subject.
Section number of course in department is identified by a call number.
Professor teaches course identified by call number.
Course in department has title.
Professor has name.



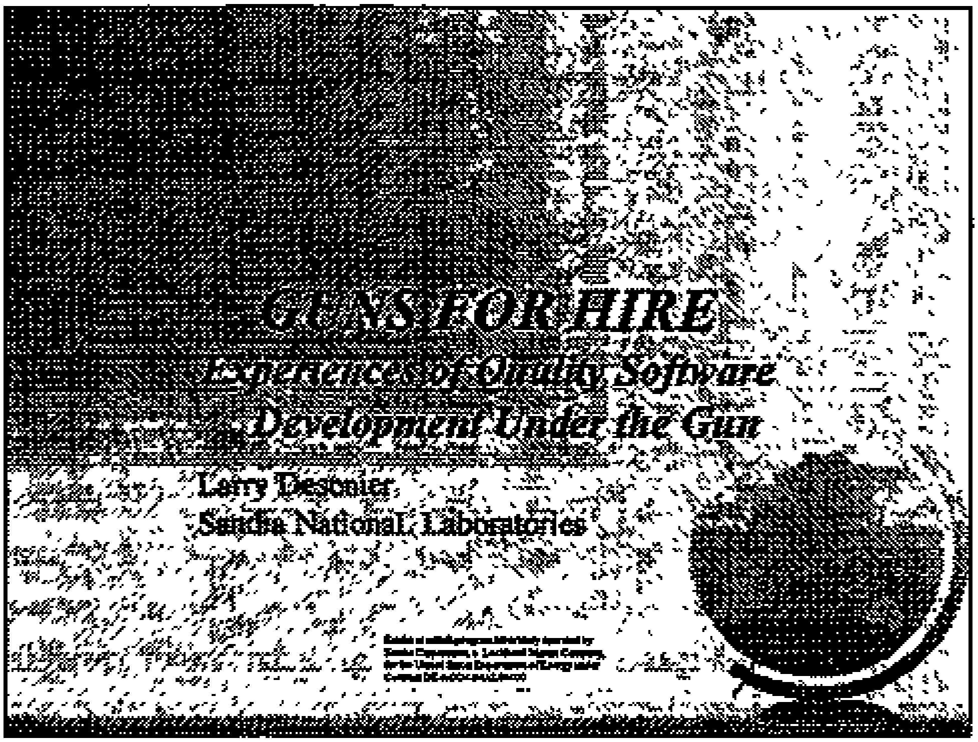
Conclusion

- Natural Language Modeling may be used to analyze any business rule.
- All business rules may be specified with set theory constraints against elementary sentences.
- The analyzed facts may be validated by any subject matter expert.
- The implementation may be tested against the validated requirements.
- Accountability may be assigned for all aspects of the project.
- Productivity improves when applications are built according to precise requirements.

Session A5: Software Quality: Experiences & Year 2000

Chair Cathy Kuhn
AS/FM&T

Session : Paper #	Author(s)	*Title*
A5:1	Larry Desonier Sandia National Laboratories	*Guns for Hire -* *Experiences of Quality Software Development Under the Gun*
A5:2	Bruce Johnston Pantex Plant	*The Year 2000 Challenge: A Project Management Perspective*
A5:3	Curt Holmes Lockheed Martin Energy Systems	*Year 2000 Awareness*

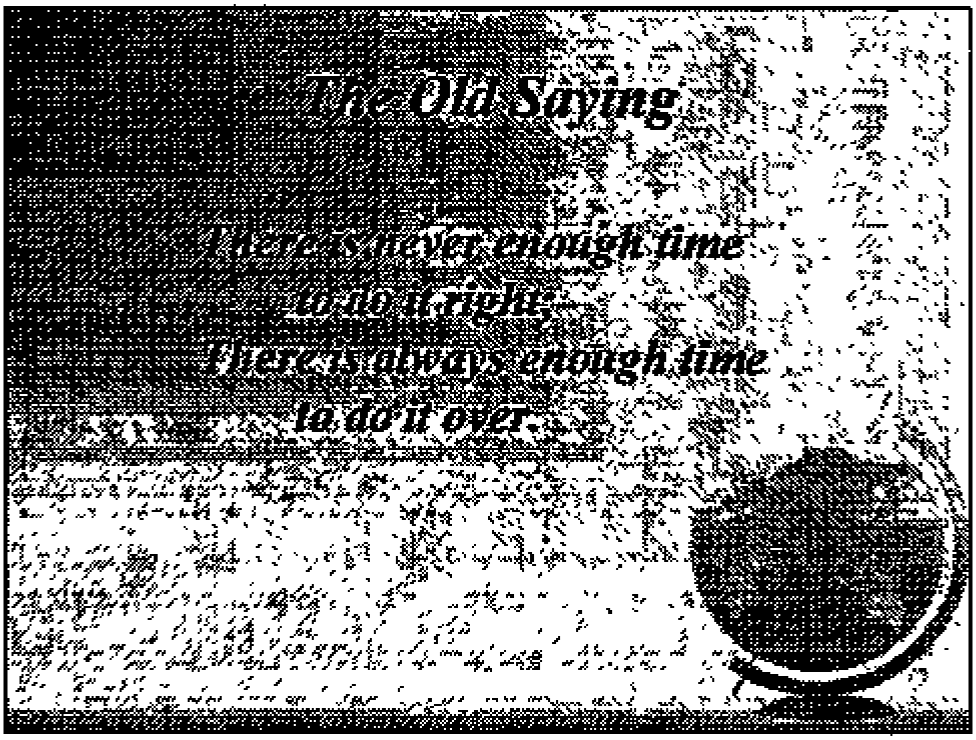


GUNS FOR HIRE

Experiences of Quality Software Development Under the Gun

Larry Desonier
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

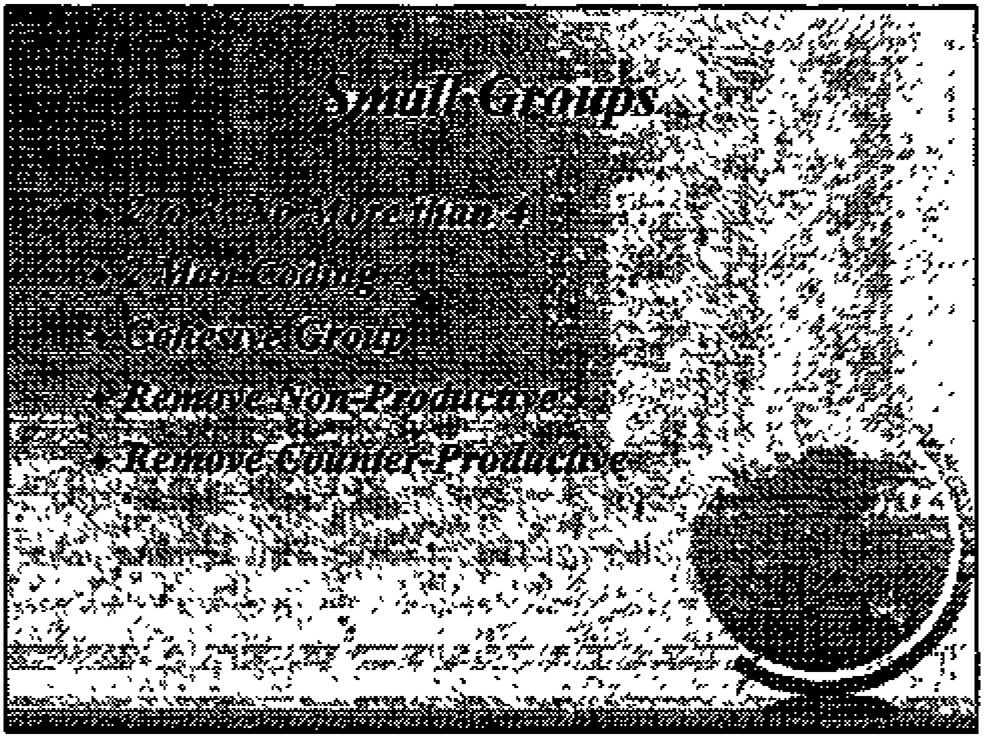


The Old Saying

There is never enough time to do it right;
There is always enough time to do it over.

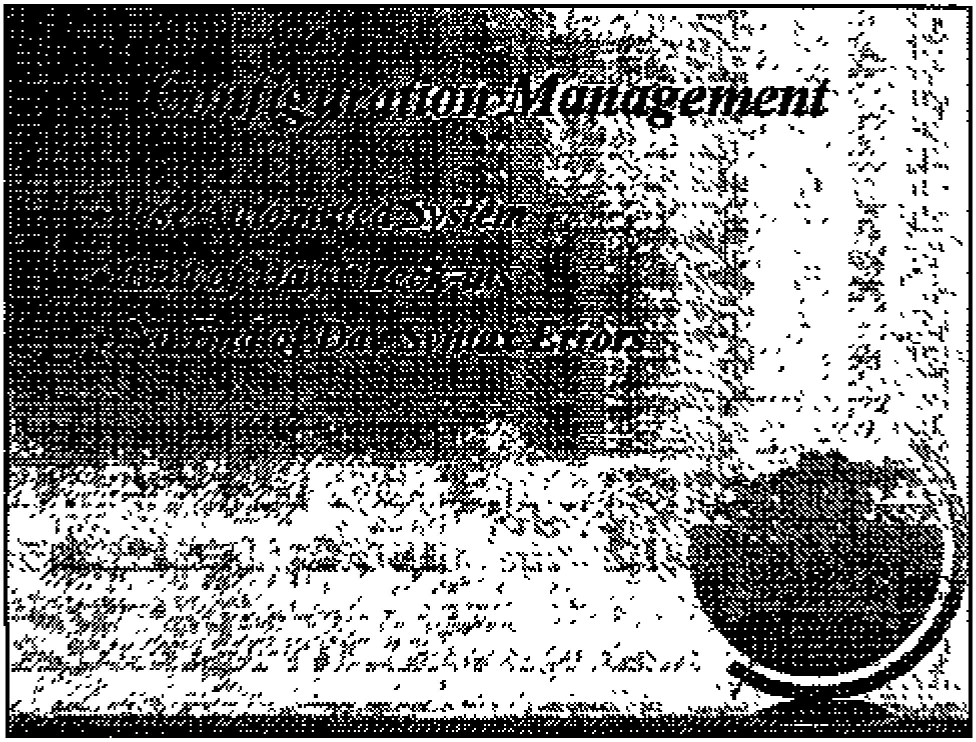
Rule 1 - Standards

- Code Style Standards
  - Naming Conventions
  - Commenting
  - Formatting
  - File Organization
  - Etc.
- Design Standards



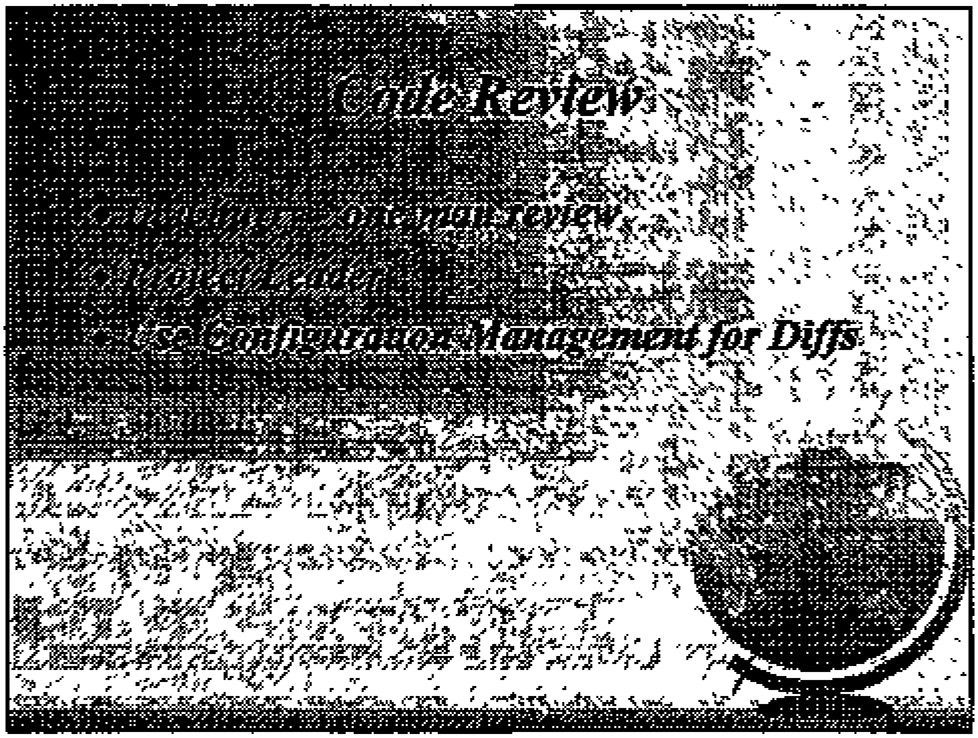
Small Groups

- [illegible] No More than 4
- 2 Man Coding
- Cohesive Group
- Remove Non-Productive
- Remove Counter-Productive



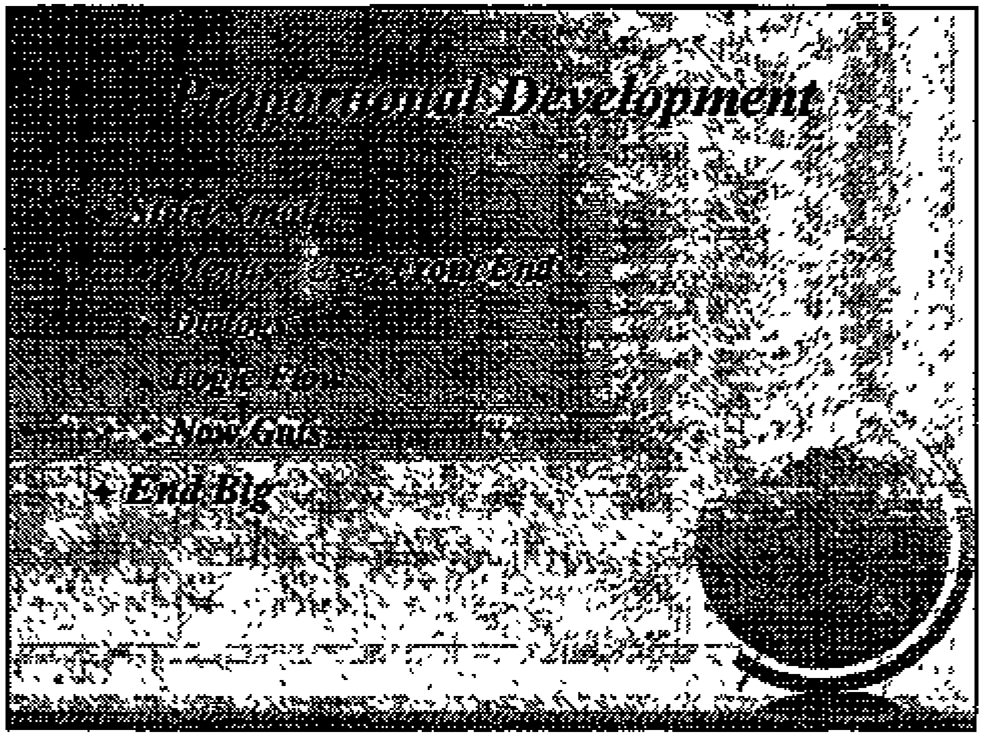
Configuration Management

- Use Automated System
- [illegible]
- [illegible] Syntax Errors



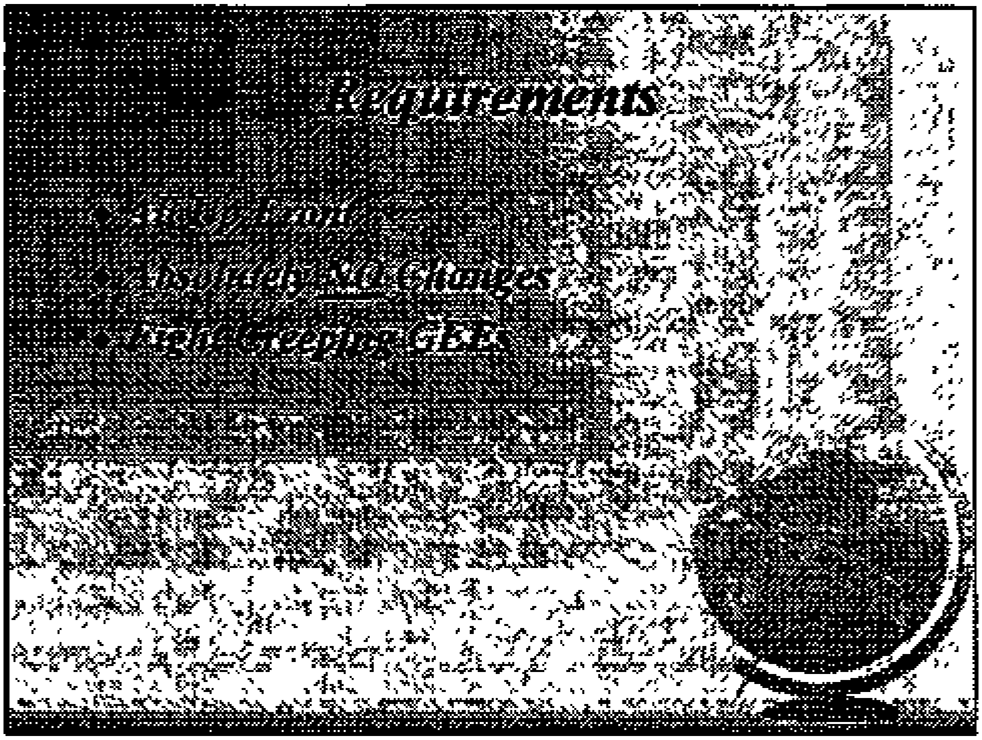
Code Review

- [illegible] review
- Project Leader
- Use Configuration Management for Diffs



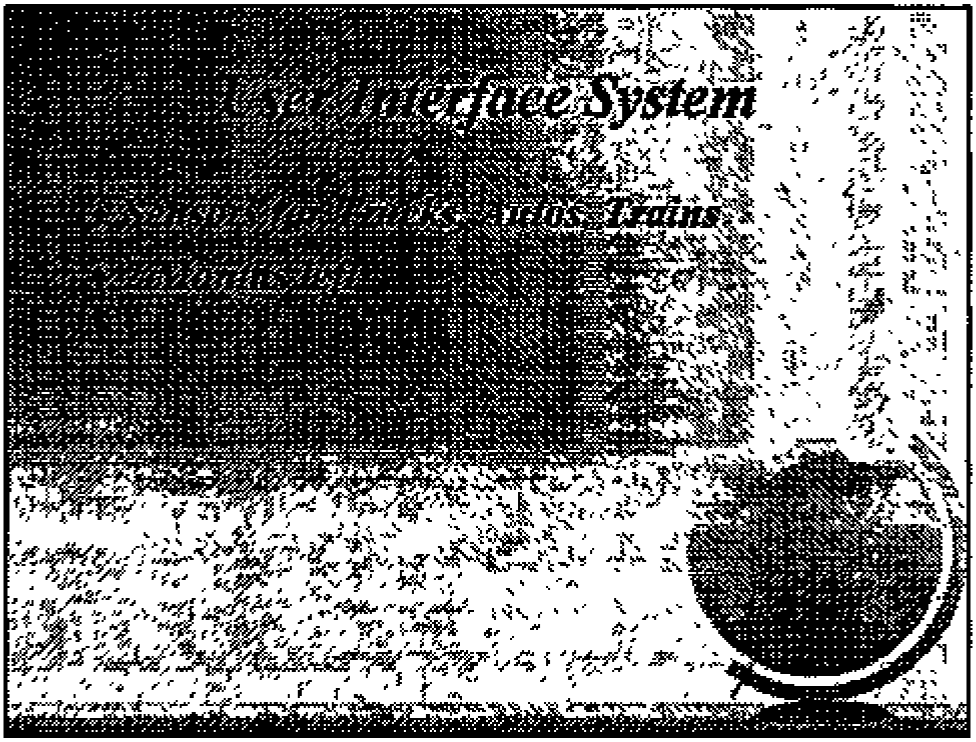
Traditional Development

- Start Small
  - Always ... Front End
  - Design
  - Logic Flow
  - New Guts
- End Big



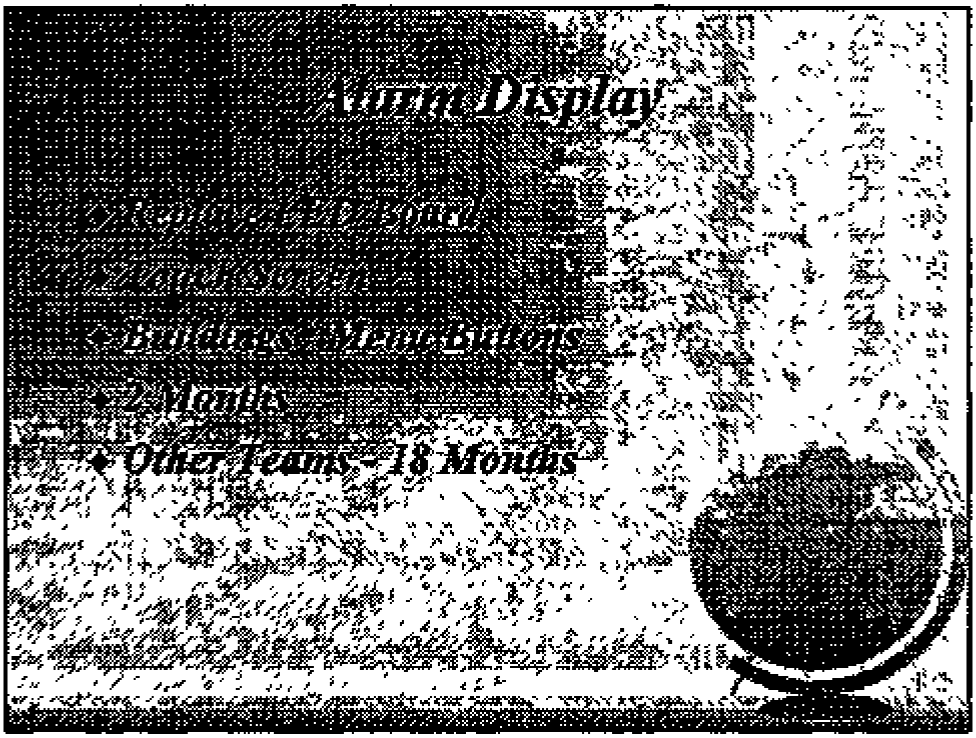
Requirements

- All Up Front
- Absolutely NO Changes
- Fight Creeping GEE



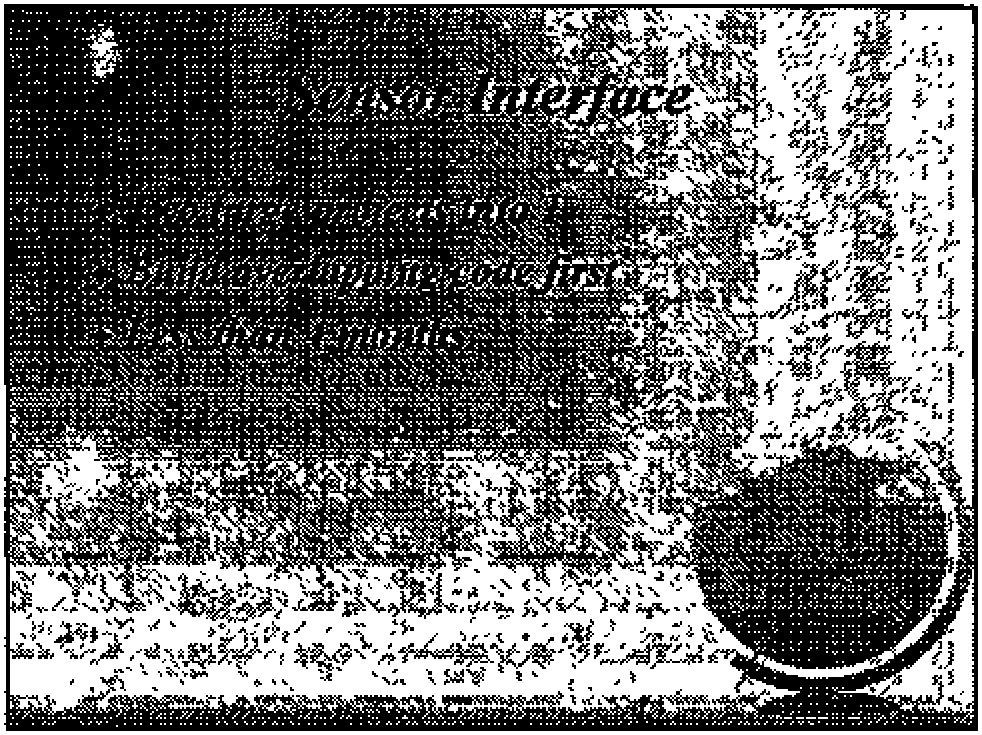
User Interface System

- [illegible] Trains
- [illegible]



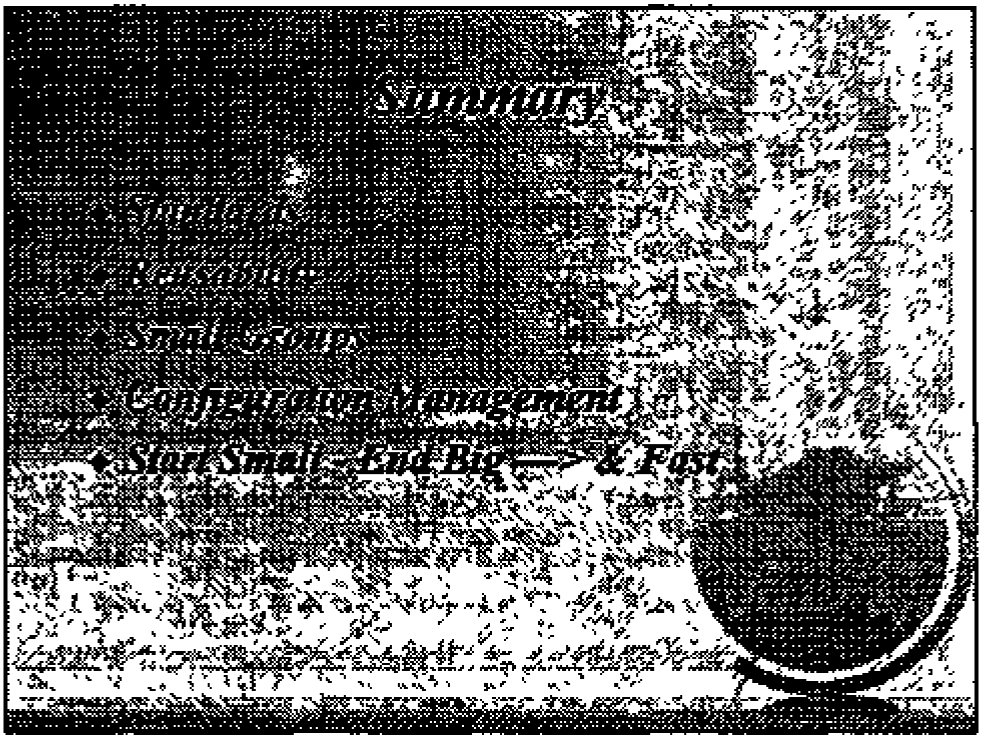
Alarm Display

- Replace LED Board
- Touch Screen
- Buildings, Menu Buttons
- 2 Months
- Other Teams – 18 Months



Sensor Interface

- Collect current info
- Build/[illegible] code first
- [illegible]



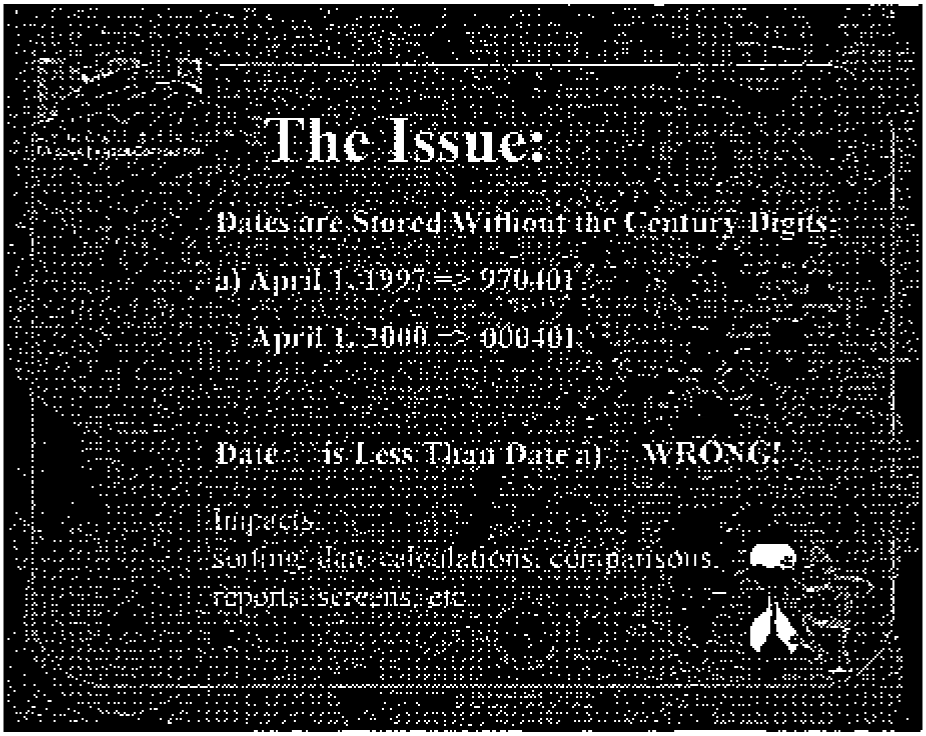
Summary

- Structure
- Personnel
- Small Groups
- Configuration Management
- Start Small - End Big ---> & Fast

THE YEAR 2000
CHALLENGE:

A PROJECT
MANAGEMENT PERSPECTIVE

Bruce Johnston
Pantex Plant Year 2000 Project Mgr.
bjohnsto@pantex.com
(806)477-3631



The Issue:

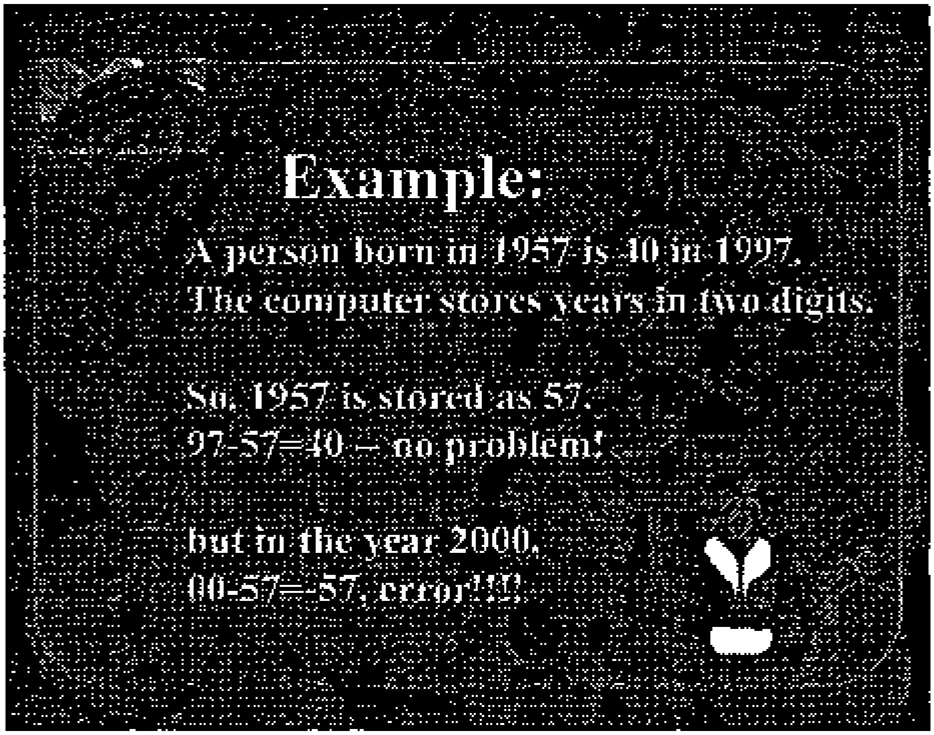
Dates are Stored Without the Century Digits:

a) April 1, 1997 => 970401

[illegible] April 1, 2000 => 000401

Date [illegible] is Less Than Date a) WRONG!

Impacts:
sorting, date calculations, comparisons,
reports, screens, etc.



Example:

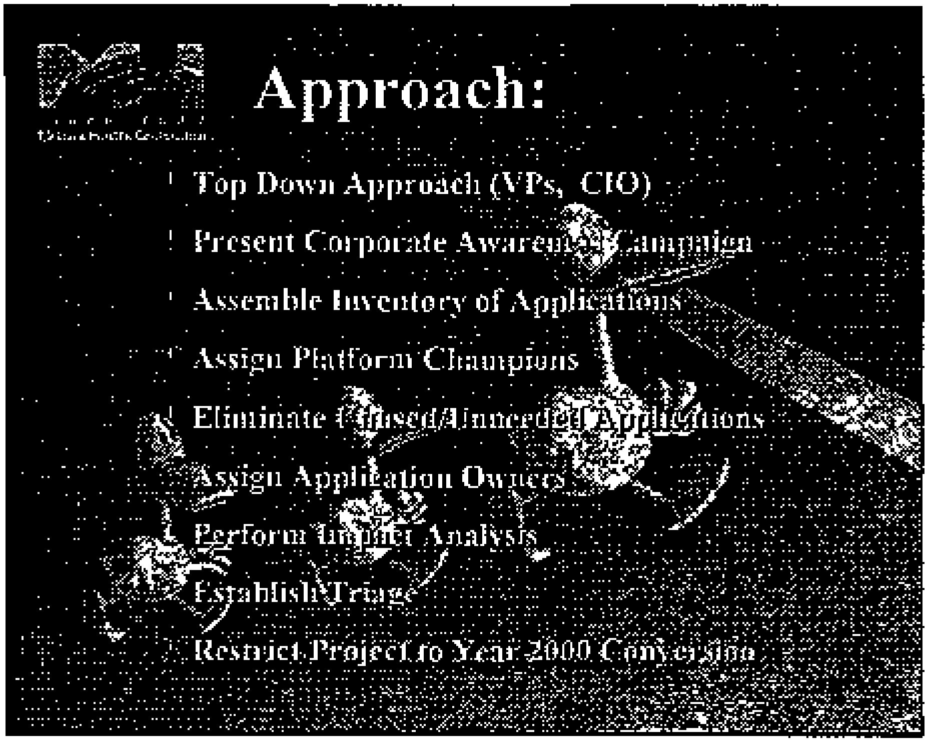
A person born in 1957 is 40 in 1997.
The computer stores years in two digits.

So, 1957 is stored as 57.
97-57=40 -- no problem!

but in the year 2000,
00-57=-57, error!!!!

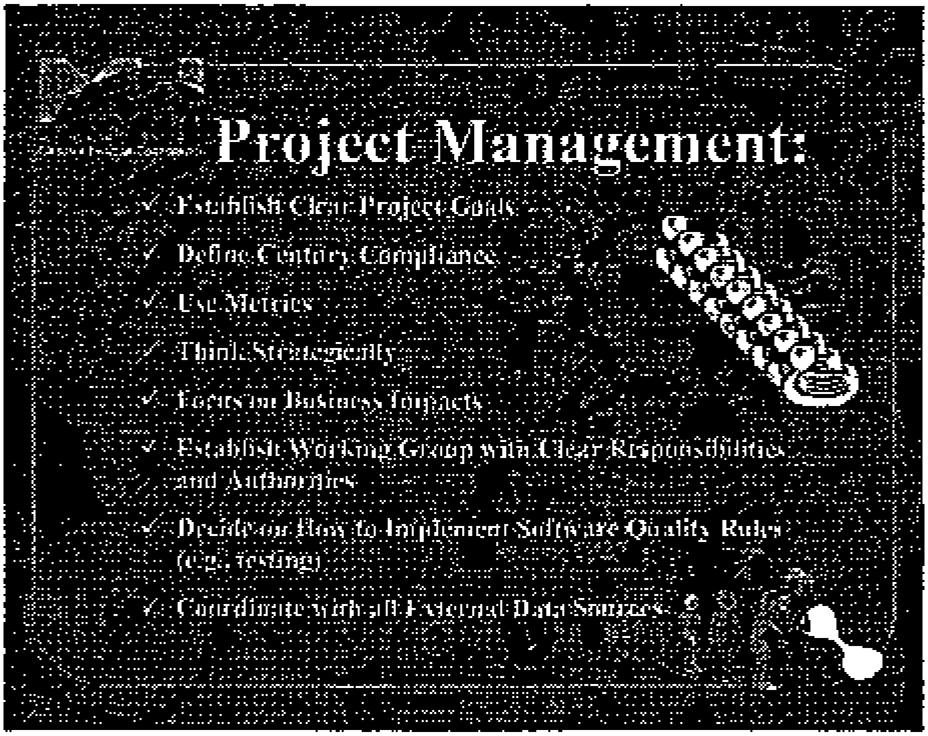
[illegible] Year 2000 Challenge:

- It is an Immovable Deadline
- The Deadline Cannot be Missed
- It Bears no Relationship to the Size of the Task
- We Share the Same Deadline
- [illegible] Problem Does Not [illegible] the Program



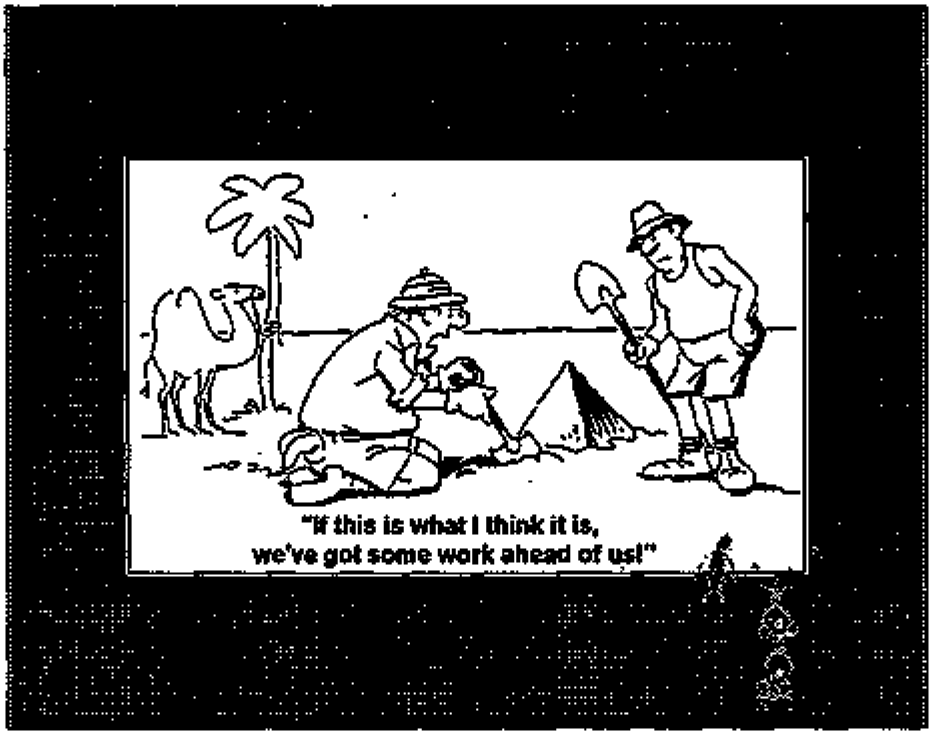
Approach:

- Top Down Approach (VPs, CIO)
- Present Corporate Awareness Campaign
- Assemble Inventory of Applications
- Assign Platform Champions
- Eliminate Unused/Unneeded Applications
- Assign Application Owners
- Perform Impact Analysis
- Establish Triage
- Restrict Project to Year 2000 Conversion

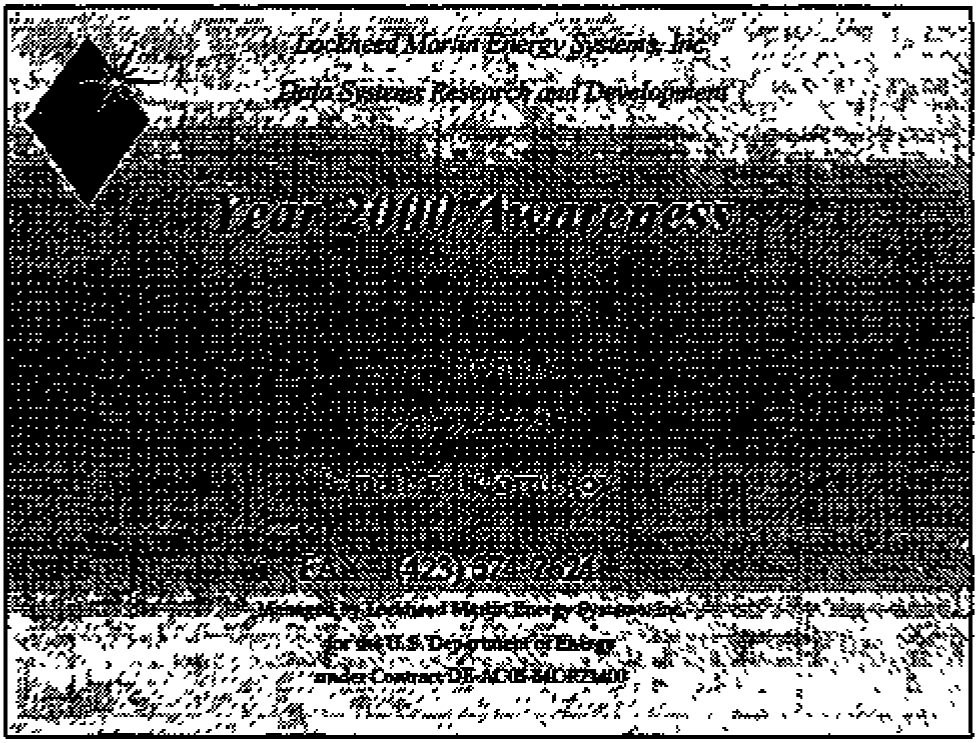


Project Management:

- ✓ Establish Clear Project Goals
- ✓ Define Century Compliance
- ✓ Use Metrics
- ✓ Think Strategically
- ✓ Focus on Business Impacts
- ✓ Establish Working Group with Clear Responsibilities and Authorities
- ✓ Decide on How to Implement Software Quality Rules (e.g., testing)
- ✓ Coordinate with all External Data Sources



"If this is what I think it is, we've got some work ahead of us!"



Lockheed Martin Energy Systems, Inc.

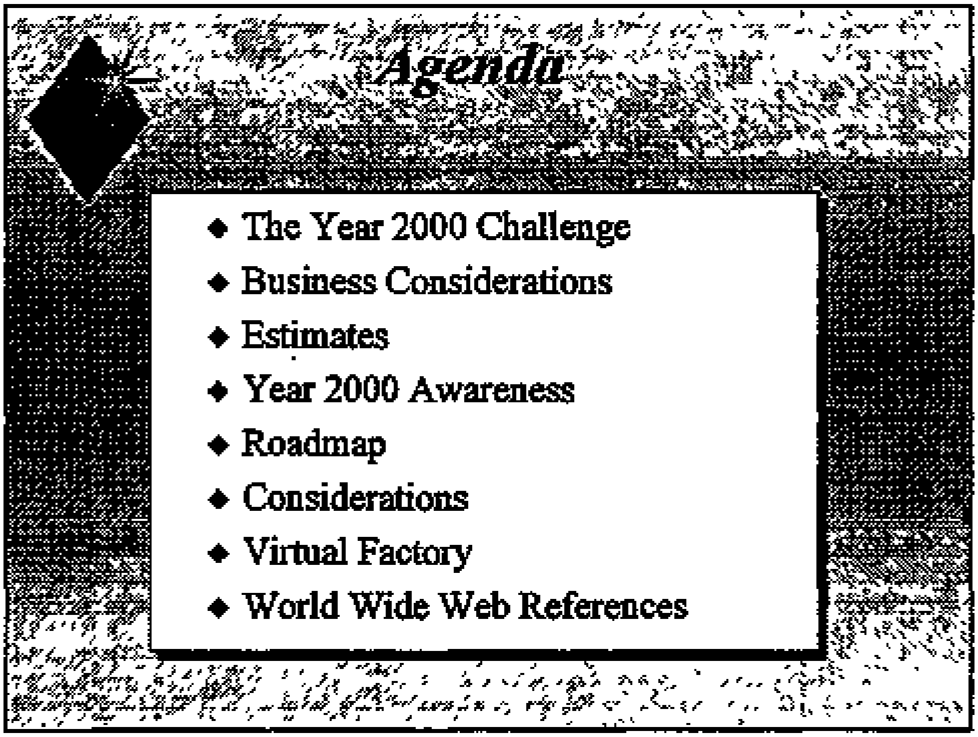
Data Systems Research and Development

Year 2000 Awareness

[illegible]

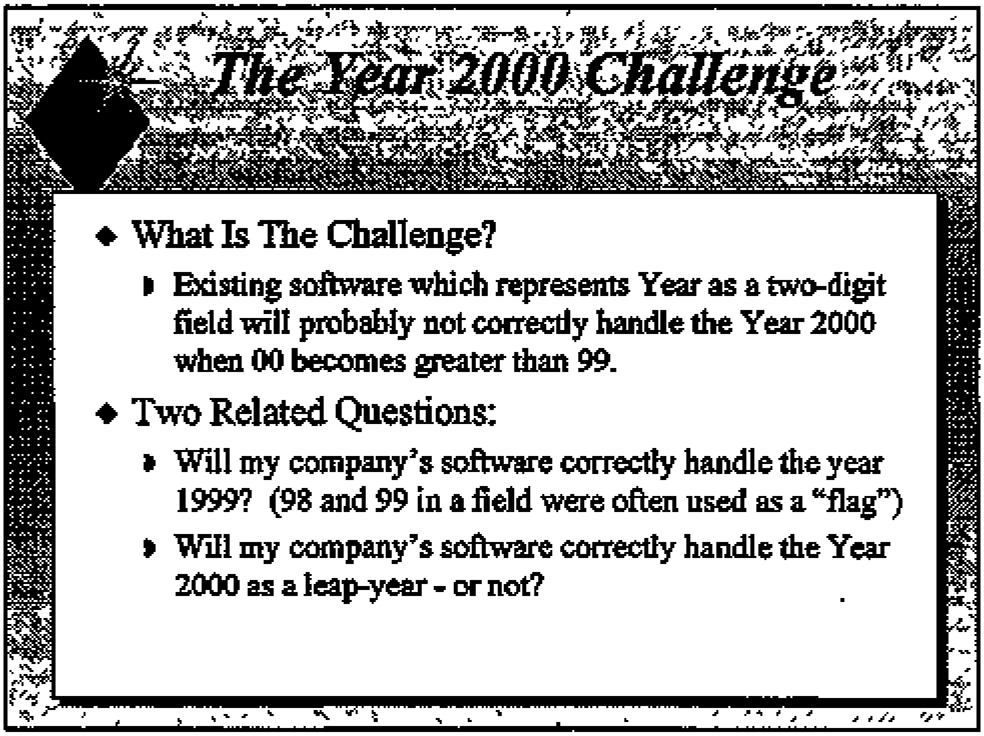
FAX (423) [illegible]

Managed by Lockheed Martin Energy Systems, Inc.
for the U.S. Department of Energy
under Contract DE-AC05-84OR21400



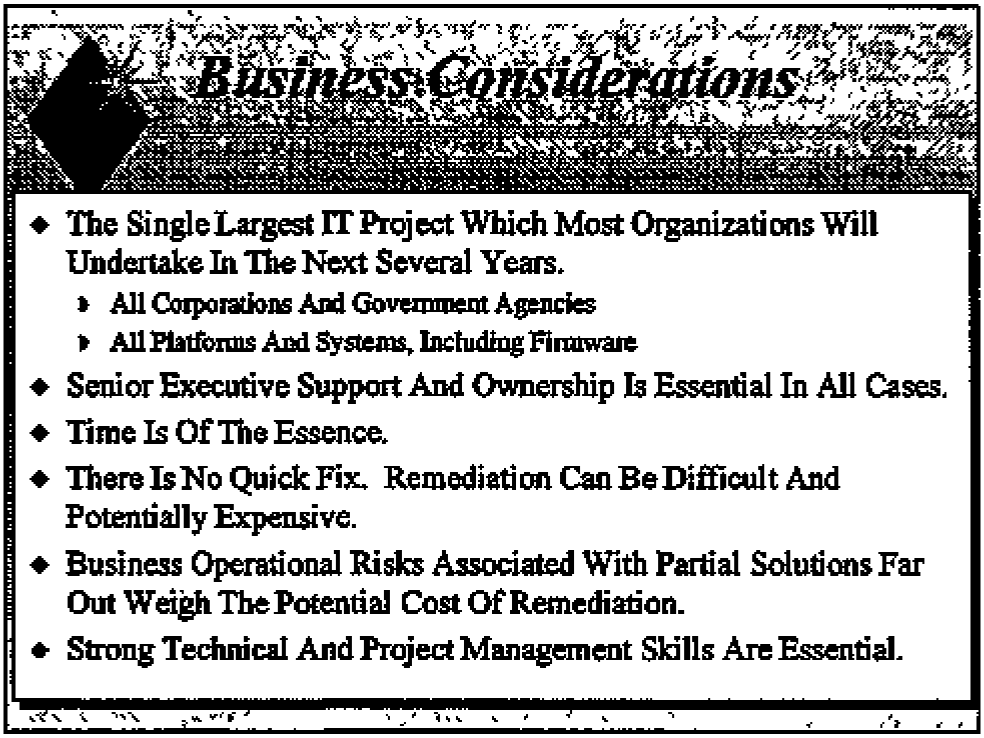
Agenda

- The Year 2000 Challenge
- Business Considerations
- Estimates
- Year 2000 Awareness
- Roadmap
- Considerations
- Virtual Factory
- World Wide Web References



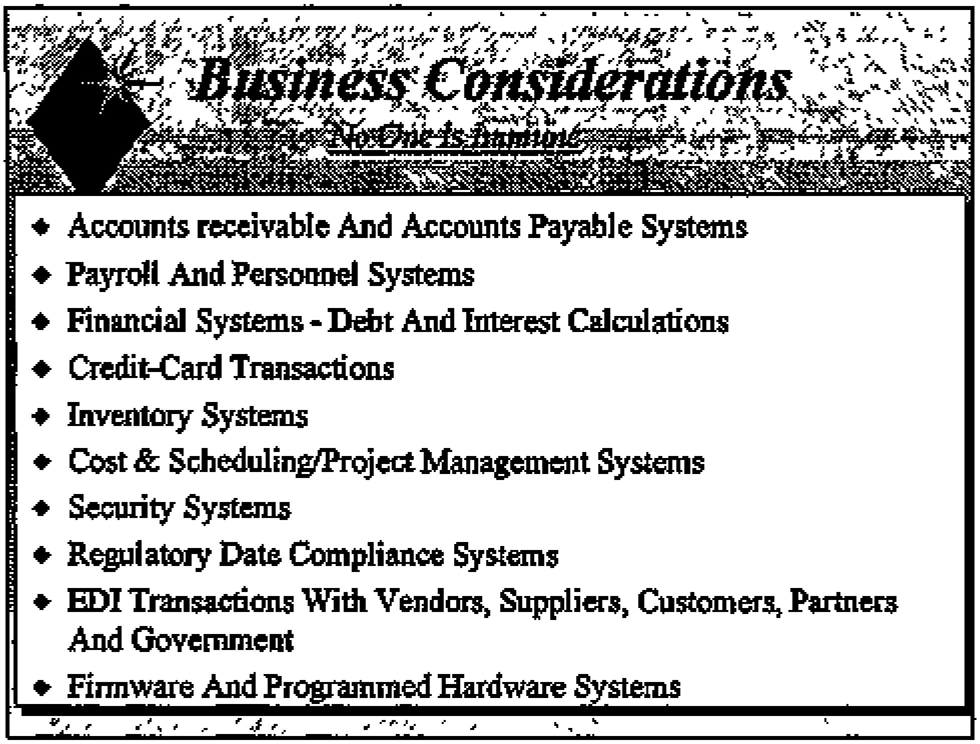
The Year 2000 Challenge

- What Is The Challenge?
  - Existing software which represents Year as a two-digit field will probably not correctly handle the Year 2000 when 00 becomes greater than 99.
- Two Related Questions:
  - Will my company's software correctly handle the year 1999? (98 and 99 in a field were often used as a "flag")
  - Will my company's software correctly handle the Year 2000 as a leap-year - or not?



Business Considerations

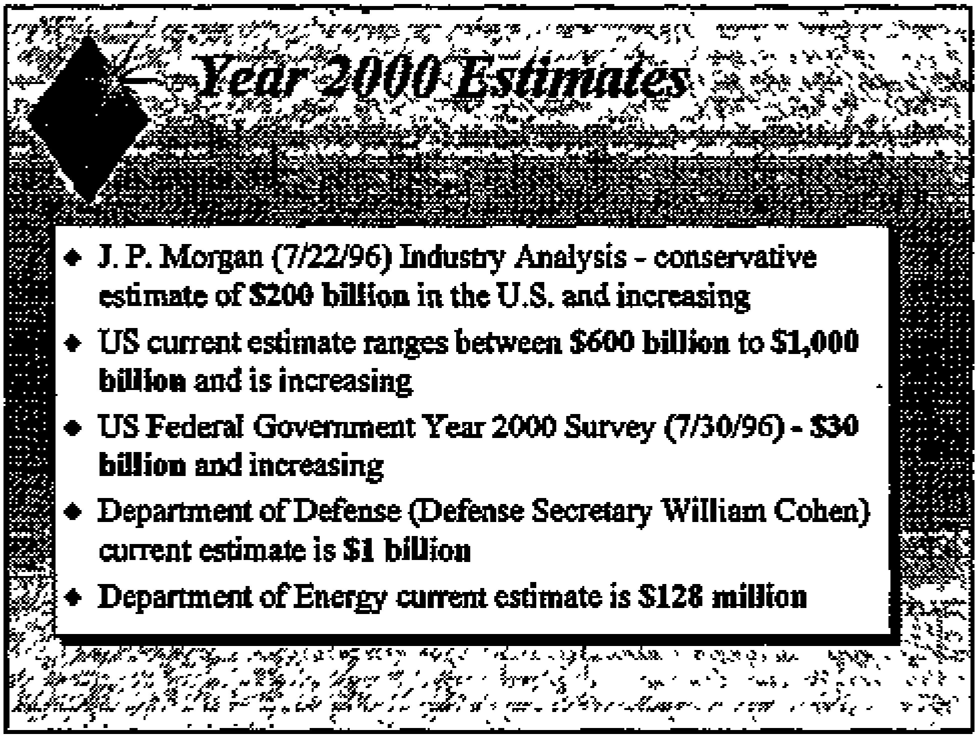
- The Single Largest IT Project Which Most Organizations Will Undertake In The Next Several Years.
  - All Corporations And Government Agencies
  - All Platforms And Systems, Including Firmware
- Senior Executive Support And Ownership Is Essential In All Cases.
- Time Is Of The Essence.
- There Is No Quick Fix. Remediation Can Be Difficult And Potentially Expensive.
- Business Operational Risks Associated With Partial Solutions Far Out Weigh The Potential Cost Of Remediation.
- Strong Technical And Project Management Skills Are Essential.



Business Considerations

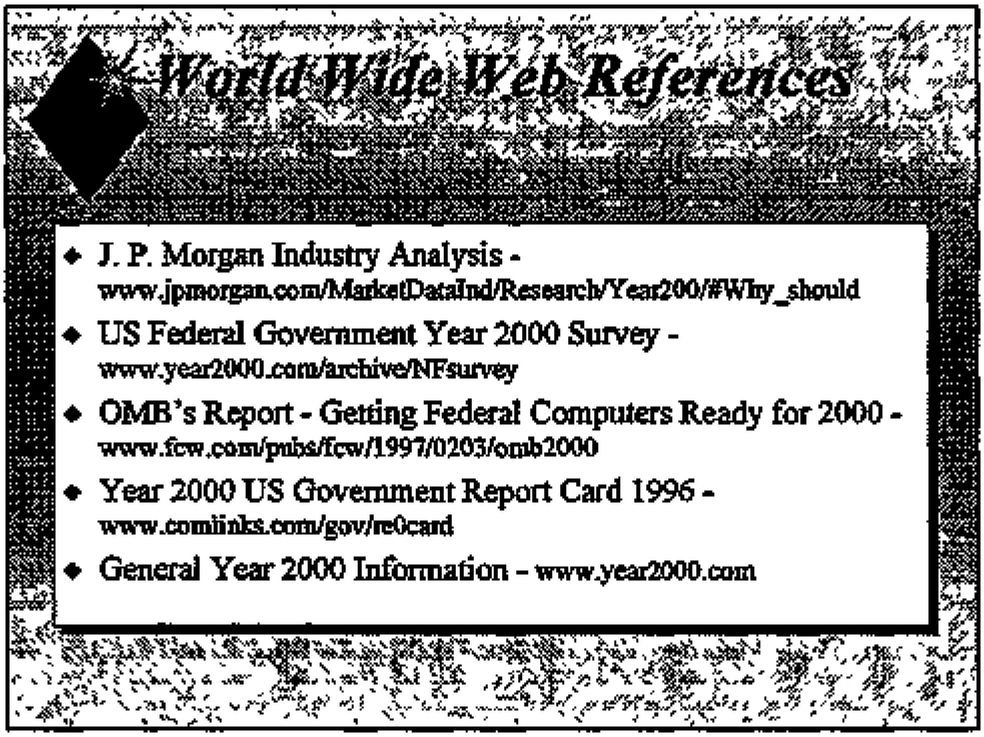
No One Is Immune

- Accounts receivable And Accounts Payable Systems
- Payroll And Personnel Systems
- Financial Systems - Debt And Interest Calculations
- Credit-Card Transactions
- Inventory Systems
- Cost & Scheduling/Project Management Systems
- Security Systems
- Regulatory Date Compliance Systems
- EDI Transactions With Vendors, Suppliers, Customers, Partners And Government
- Firmware And Programmed Hardware Systems



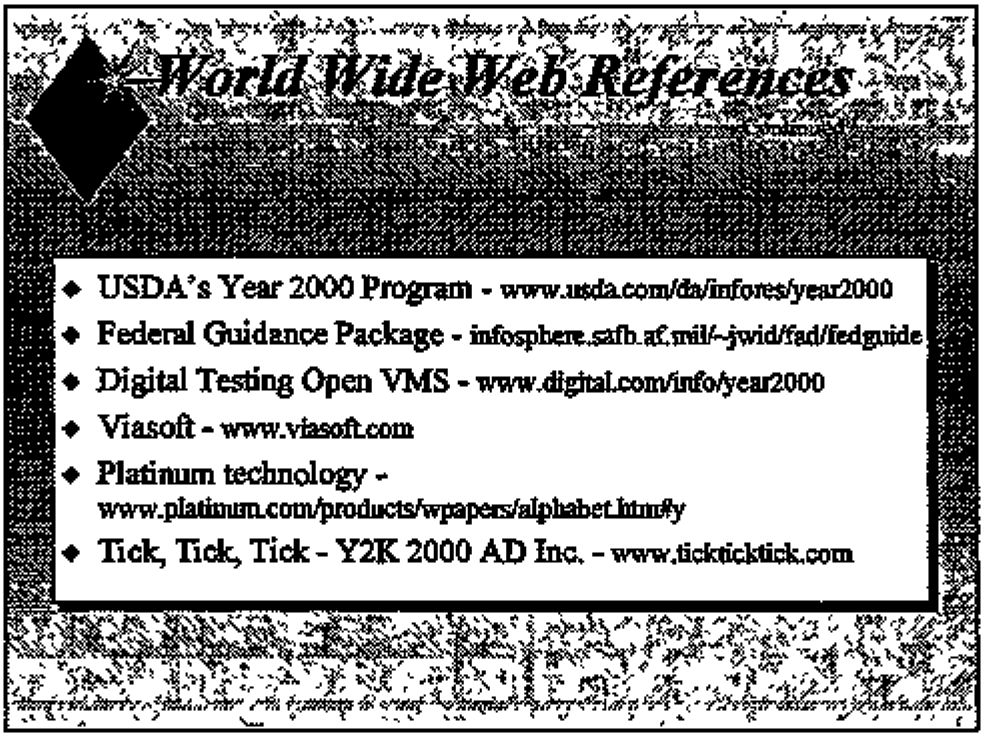
Year 2000 Estimates

- J. P. Morgan (7/22/96) Industry Analysis - conservative estimate of **$200 billion** in the U.S. and increasing
- US current estimate ranges between **$600 billion** to **$1,000 billion** and is increasing
- US Federal Government Year 2000 Survey (7/30/96) - **$30 billion** and increasing
- Department of Defense (Defense Secretary William Cohen) current estimate is **$1 billion**
- Department of Energy current estimate is **$128 million**



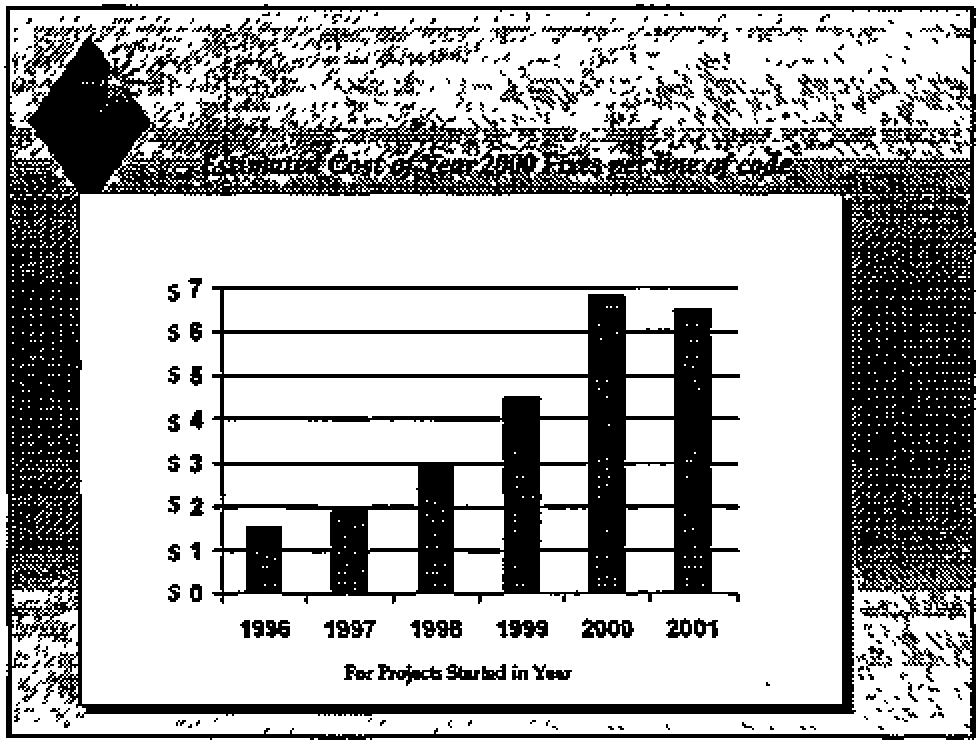
World Wide Web References

- J. P. Morgan Industry Analysis - www.jpmorgan.com/MarketDataInd/Research/Year200/#Why_should
- US Federal Government Year 2000 Survey - www.year2000.com/archive/NFsurvey
- OMB's Report - Getting Federal Computers Ready for 2000 - www.fcw.com/pubs/fcw/1997/0203/omb2000
- Year 2000 US Government Report Card 1996 - www.comlinks.com/gov/re0card
- General Year 2000 Information - www.year2000.com

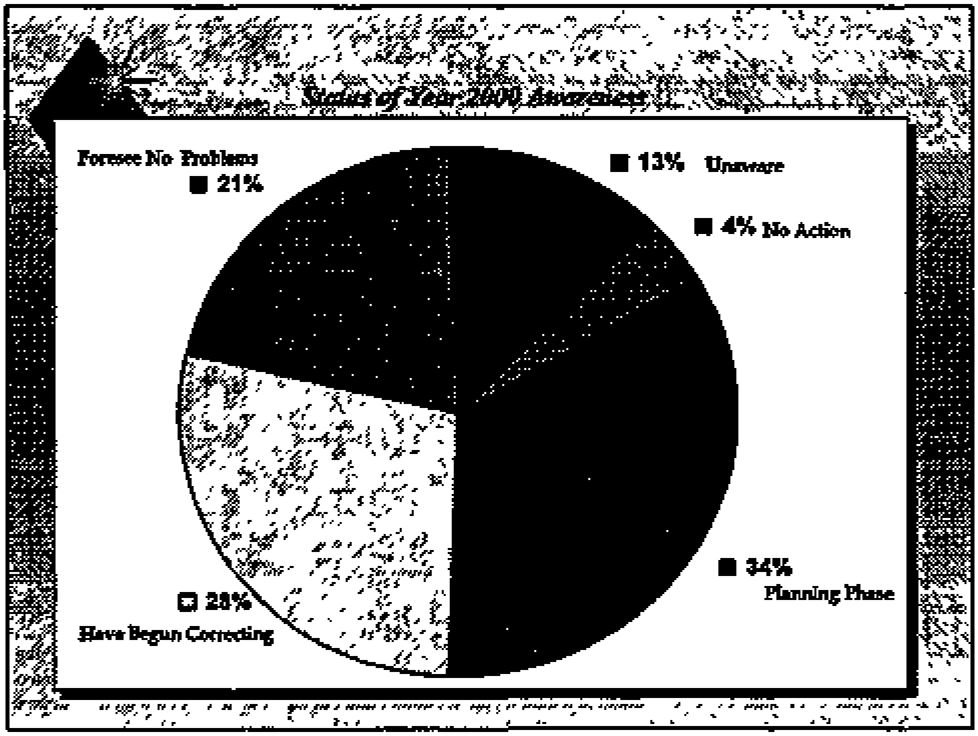


World Wide Web References

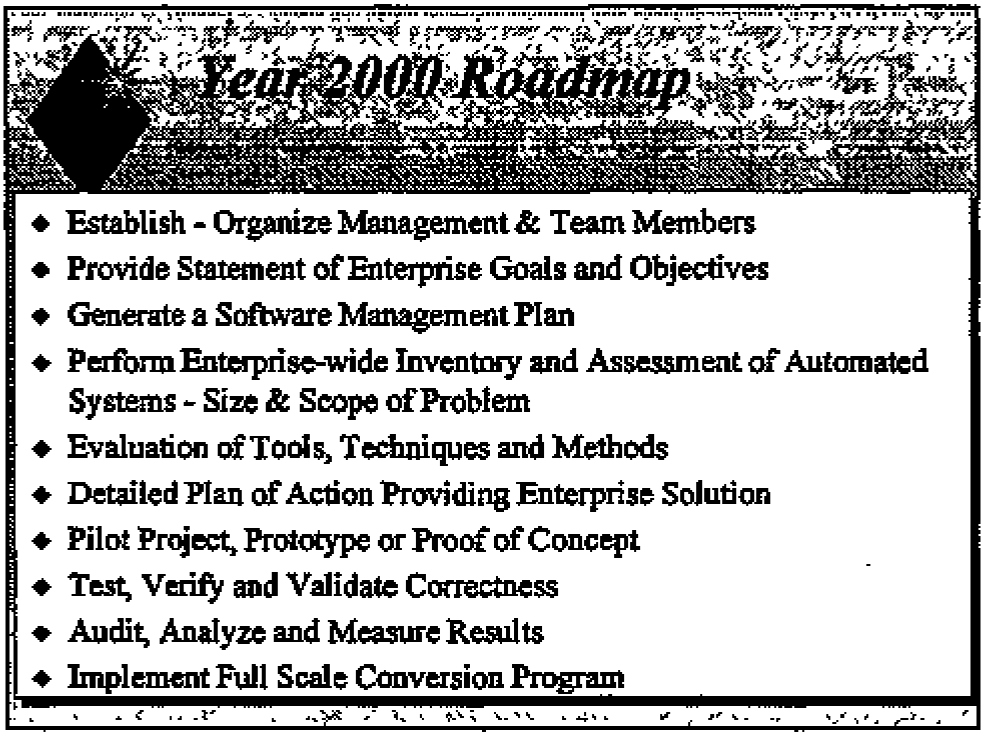
- USDA's Year 2000 Program - www.usda.com/da/infores/year2000
- Federal Guidance Package - infosphere.safb.af.mil/~jwid/fad/fedguide
- Digital Testing Open VMS - www.digital.com/info/year2000
- Viasoft - www.viasoft.com
- Platinum technology - www.platinum.com/products/wpapers/alphabet.htm#y
- Tick, Tick, Tick - Y2K 2000 AD Inc. - www.ticktick.com



Estimated Cost of Year 2000 Fixes per line of code

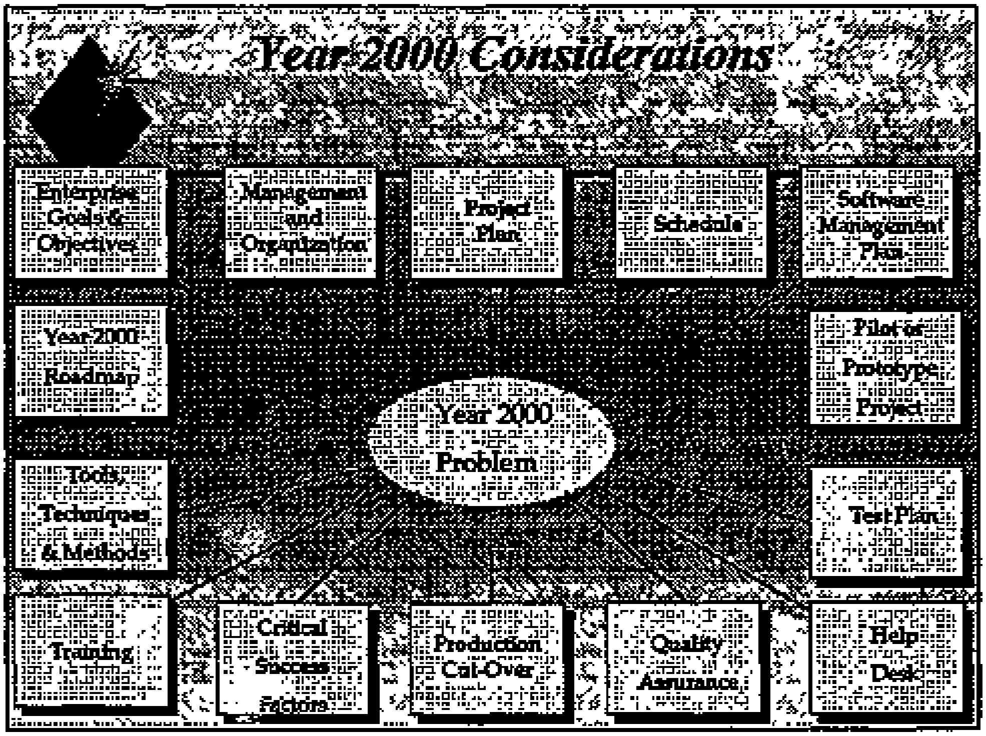


Status of Year 2000 Awareness

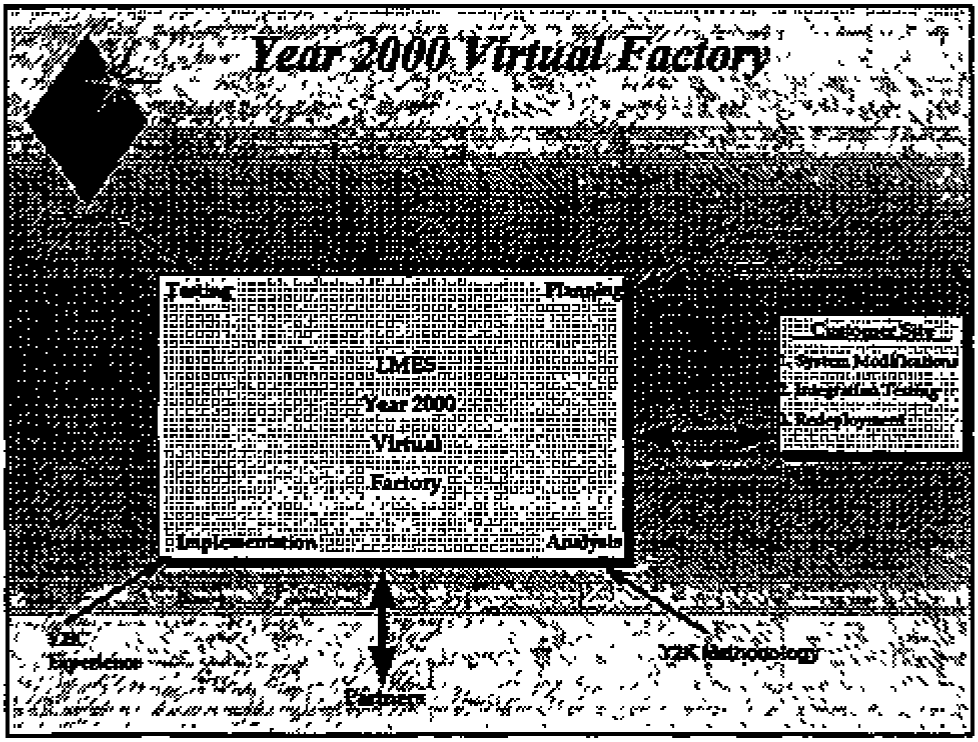


Year 2000 Roadmap

- Establish - Organize Management & Team Members
- Provide Statement of Enterprise Goals and Objectives
- Generate a Software Management Plan
- Perform Enterprise-wide Inventory and Assessment of Automated Systems - Size & Scope of Problem
- Evaluation of Tools, Techniques and Methods
- Detailed Plan of Action Providing Enterprise Solution
- Pilot Project, Prototype or Proof of Concept
- Test, Verify and Validate Correctness
- Audit, Analyze and Measure Results
- Implement Full Scale Conversion Program



Year 2000 Considerations

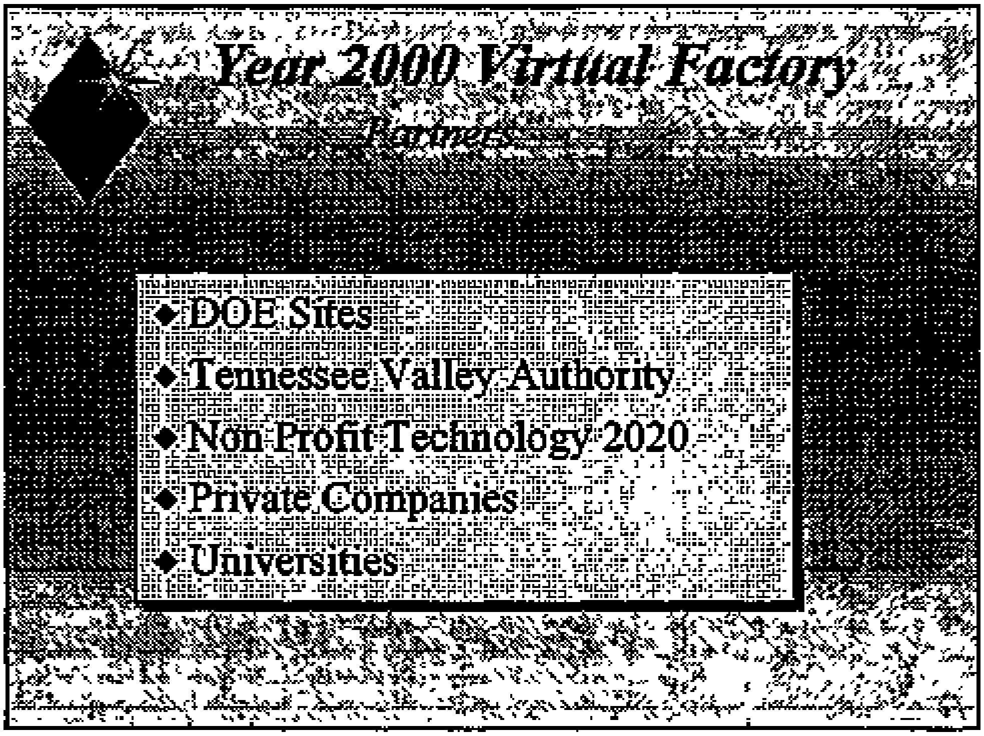


Year 2000 Virtual Factory

Year 2000 Virtual Factory

Basic Capabilities

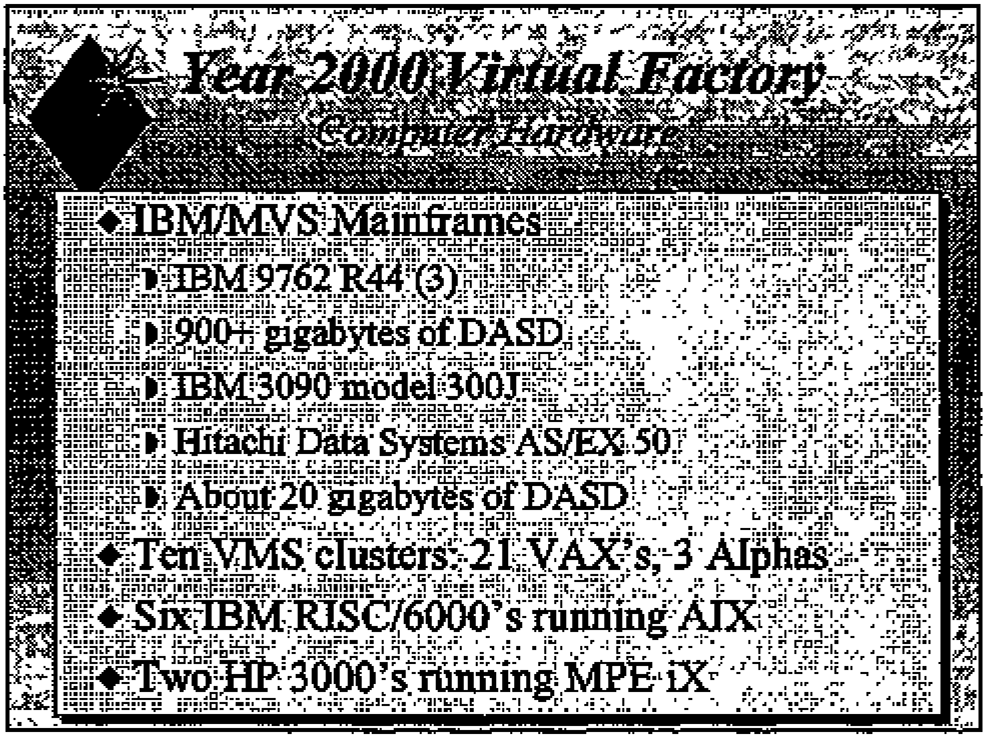
- Virtual Factory at Oak Ridge
- Technical Infrastructure
  - Three (3) Data Centers
  - Scientific Computing at ORNL
  - Experience with leading Y2K Automated Software Packages
- Telecommunications Infrastructure



Year 2000 Virtual Factory

Partners

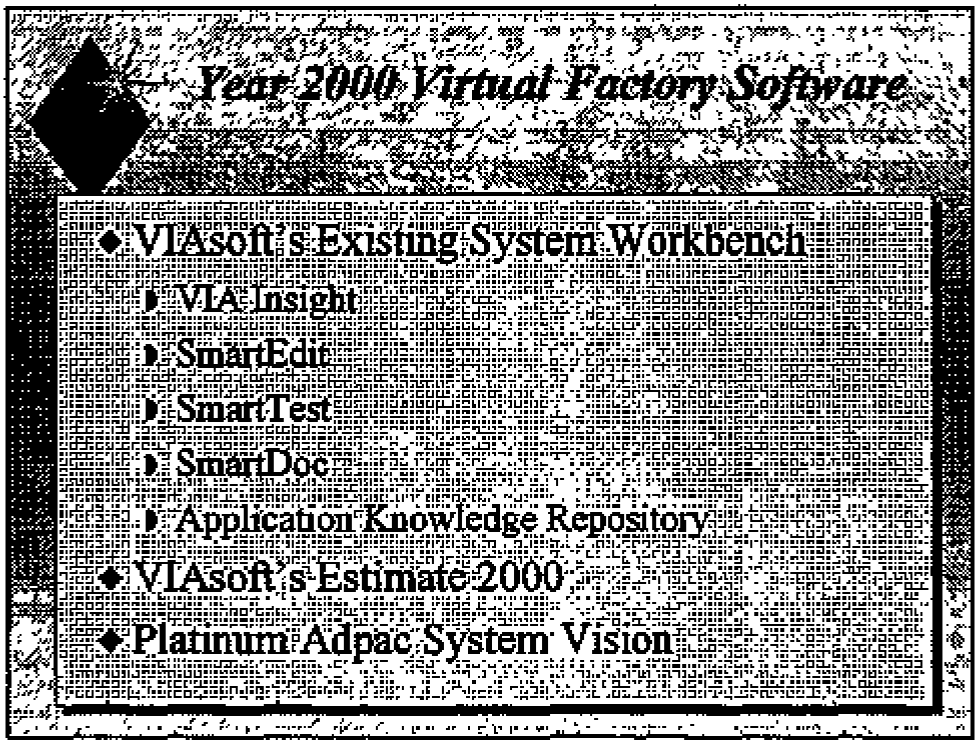
- DOE Sites
- Tennessee Valley Authority
- Non Profit Technology 2020
- Private Companies
- Universities



Year 2000 Virtual Factory

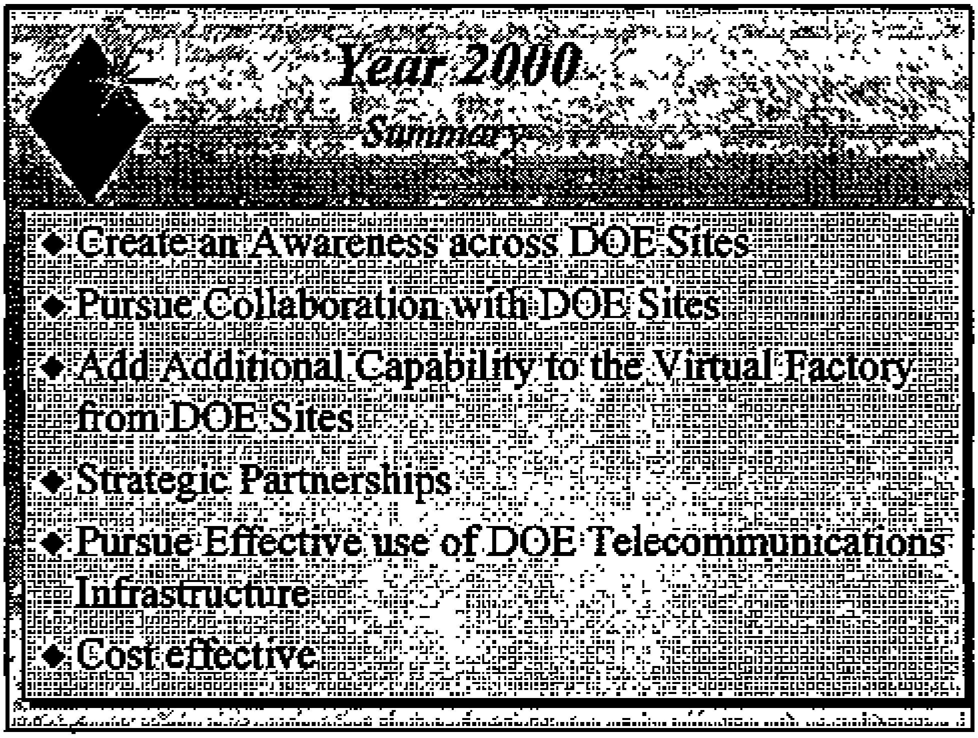
Computer Hardware

- IBM/MVS Mainframes
  - IBM 9762 R44 (3)
  - 900+ gigabytes of DASD
  - IBM 3090 model 300J
  - Hitachi Data Systems AS/EX 50
  - About 20 gigabytes of DASD
- Ten VMS clusters: 21 VAX's, 3 Alphas
- Six IBM RISC/6000's running AIX
- Two HP 3000's running MPE iX



Year 2000 Virtual Factory Software

- VIAsoft's Existing System Workbench
  - VIA Insight
  - SmartEdit
  - SmartTest
  - SmartDoc
  - Application Knowledge Repository
- VIAsoft's Estimate 2000
- Platinum Adpac System Vision



Year 2000

Summary

- Create an Awareness across DOE Sites
- Pursue Collaboration with DOE Sites
- Add Additional Capability to the Virtual Factory from DOE Sites
- Strategic Partnerships
- Pursue Effective use of DOE Telecommunications Infrastructure
- Cost effective

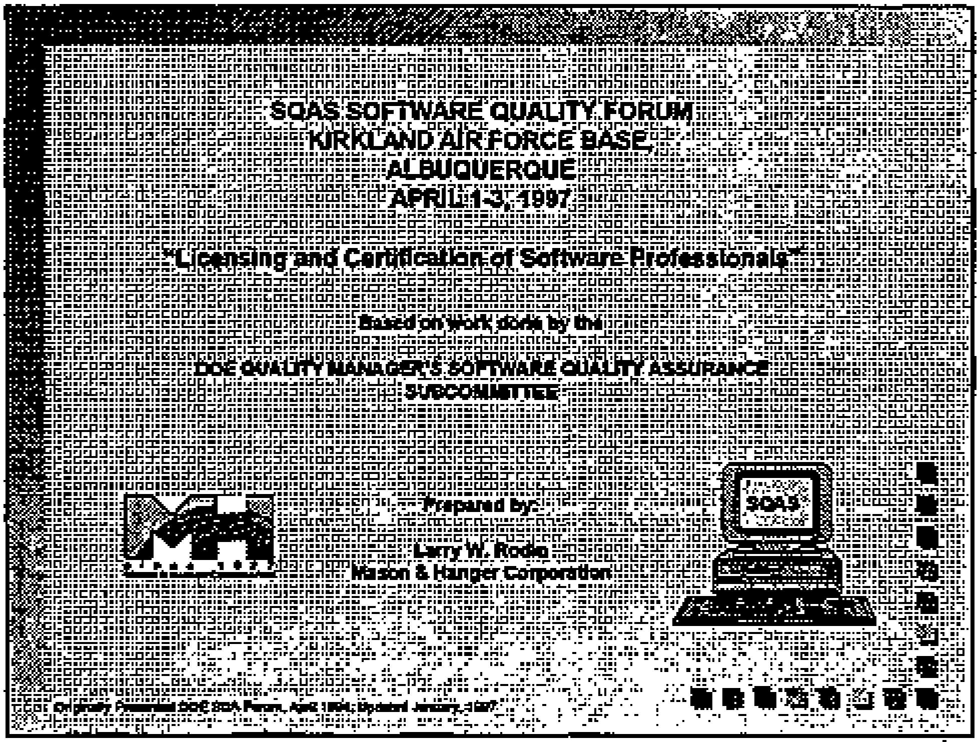
1997 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Session B5: Software Standards for Quality Engineering

Chair Patty Trellue
Sandia National Laboratories

Session : Paper #	Author(s)	*Title*
B5:1	John Hare AWE UK	*ISO and Software Quality Assurance*
B5:2	Larry Rodin Pantex Plant	*Licensing and Certification of Software Professionals*
B5:3	Michael Lackner AS/FM&T	*Operational Excellence (Six Sigma) Philosophy: Application to Software Quality Assurance*



SQAS SOFTWARE QUALITY FORUM
KIRKLAND AIR FORCE BASE,
ALBUQUERQUE
APRIL 1-3, 1997

"Licensing and Certification of Software Professionals"

Based on work done by the

DOE QUALITY MANAGER'S SOFTWARE QUALITY ASSURANCE SUBCOMMITTEE

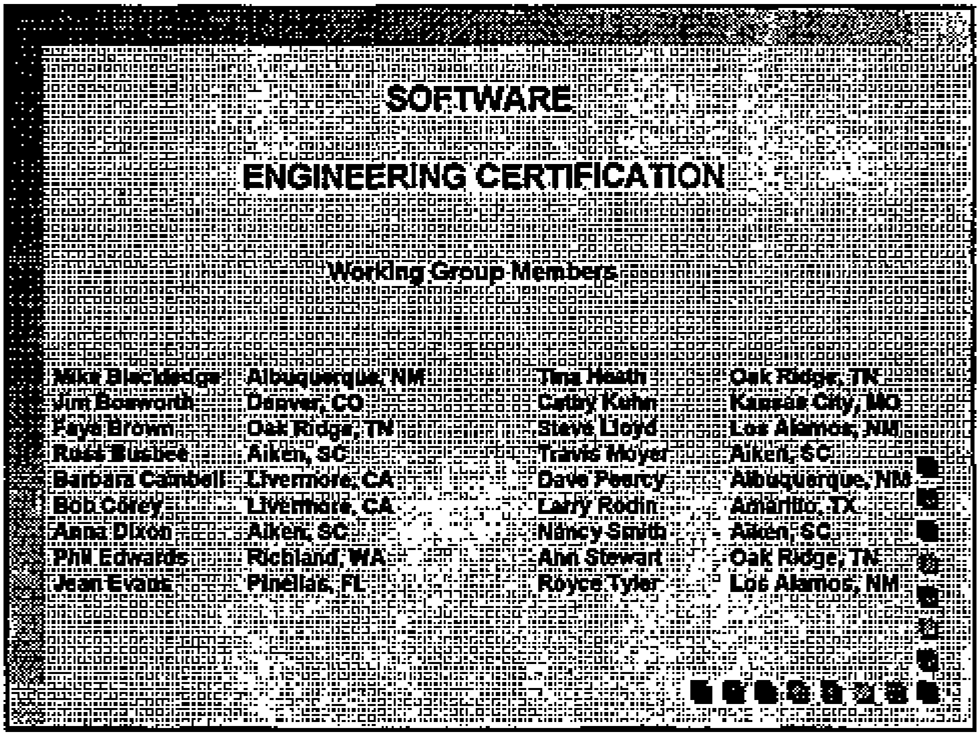
Prepared by:

Larry W. Rodin
Mason & Hanger Corporation

Originally Presented DOE SQA Forum, April 1994; Updated January, 1997

Licensing and Certification of Software Professionals

- Background for Presentation
- Certification Programs
- Licensing Programs
- Why Become Certified?
- Certification as a Condition of Employment
- Certification Requirements
- Examination Structures

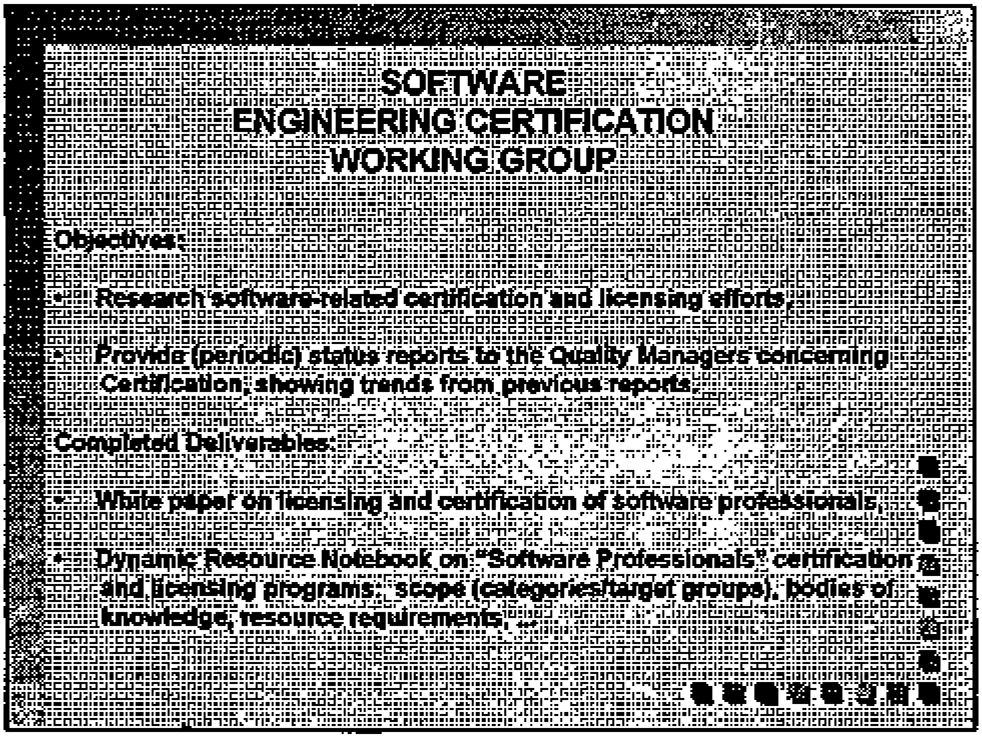


SOFTWARE

ENGINEERING CERTIFICATION

Working Group Members

Mike Blackledge	Albuquerque, NM	Tina Heath	Oak Ridge, TN
Jim Bosworth	Denver, CO	Cathy Kuhn	Kansas City, MO
Faye Brown	Oak Ridge, TN	Steve Lloyd	Los Alamos, NM
Russ Bisbee	Aiken, SC	Travis Moyer	Aiken, SC
Barbara Cambell	Livermore, CA	Dave Pearcy	Albuquerque, NM
Bob Corey	Livermore, CA	Larry Rodin	Amarillo, TX
Anna Dixon	Aiken, SC	Nancy Smith	Aiken, SC
Phil Edwards	Richland, WA	Ann Stewart	Oak Ridge, TN
Jean Evans	Pinellas, FL	Royce Tyler	Los Alamos, NM



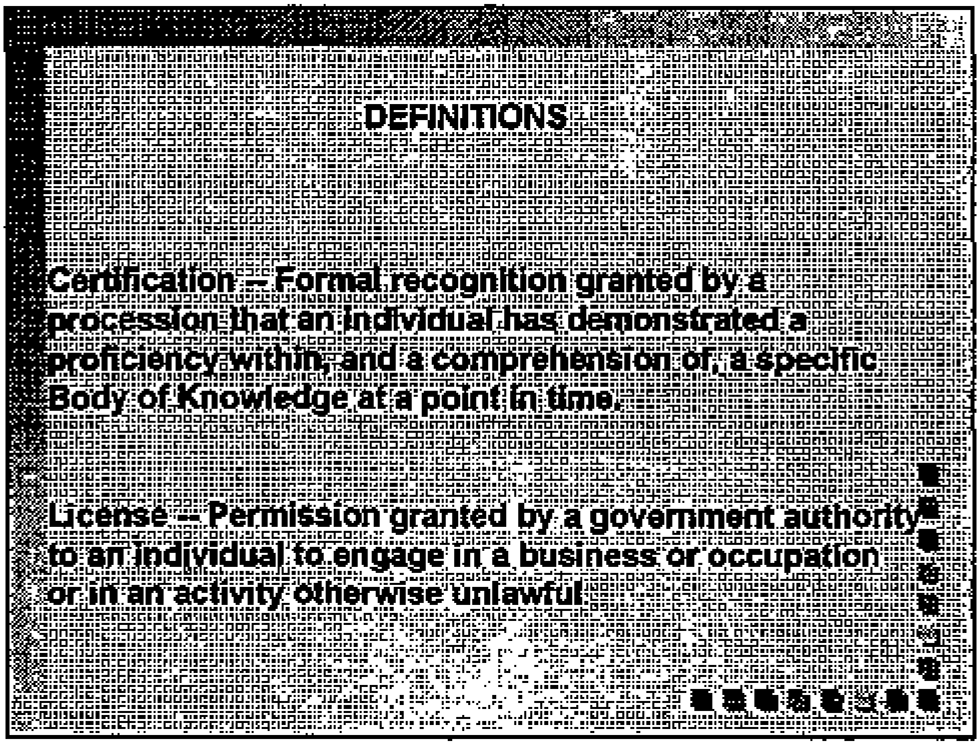
SOFTWARE ENGINEERING CERTIFICATION WORKING GROUP

Objectives:

- Research software-related certification and licensing efforts.
- Provide (periodic) status reports to the Quality Managers concerning Certification, showing trends from previous reports.

Completed Deliverables:

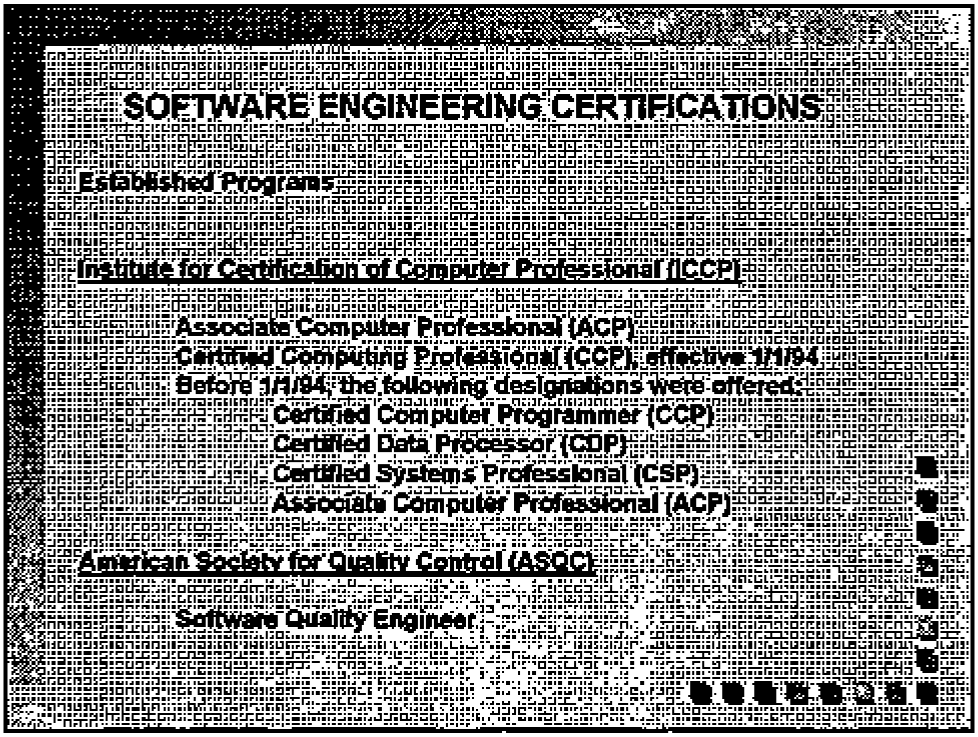
- White paper on licensing and certification of software professionals.
- Dynamic Resource Notebook on "Software Professionals" certification and licensing programs: scope (categories/target groups), bodies of knowledge, resource requirements, ...



DEFINITIONS

Certification -- Formal recognition granted by a procession that an individual has demonstrated a proficiency within, and a comprehension of, a specific Body of Knowledge at a point in time.

License -- Permission granted by a government authority to an individual to engage in a business or occupation or in an activity otherwise unlawful.



SOFTWARE ENGINEERING CERTIFICATIONS

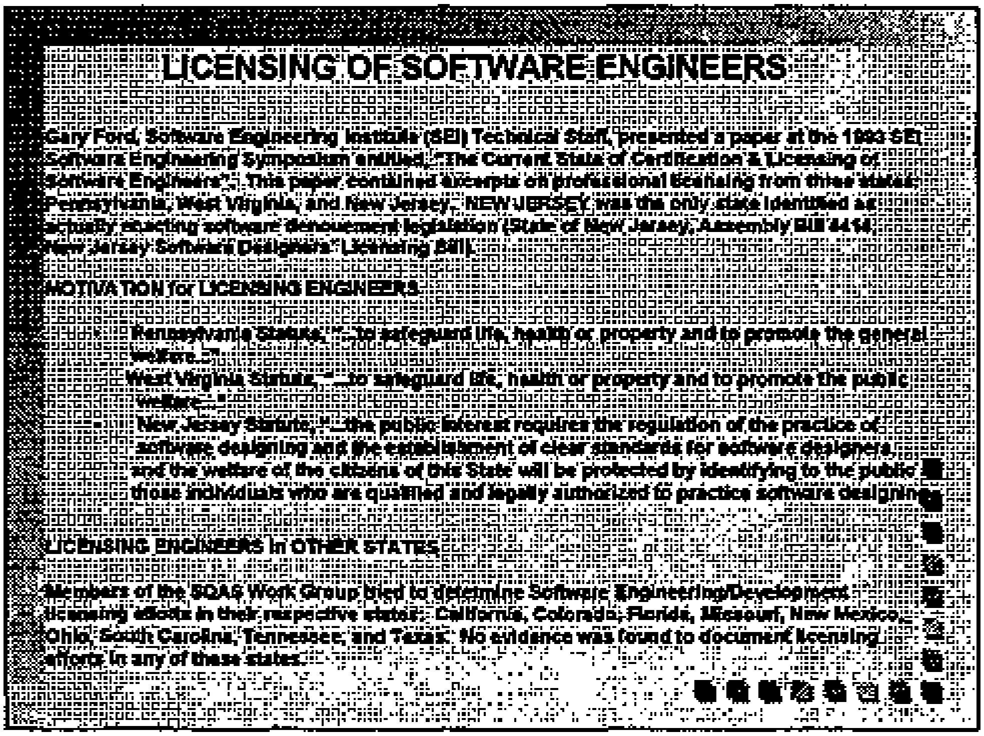
Established Programs

Institute for Certification of Computer Professional (ICCP)

- Associate Computer Professional (ACP)
- Certified Computing Professional (CCP), effective 1/1/94
- Before 1/1/94, the following designations were offered:
    - Certified Computer Programmer (CCP)
    - Certified Data Processor (CDP)
    - Certified Systems Professional (CSP)
    - Associate Computer Professional (ACP)

American Society for Quality Control (ASQC)

- Software Quality Engineer



LICENSING OF SOFTWARE ENGINEERS

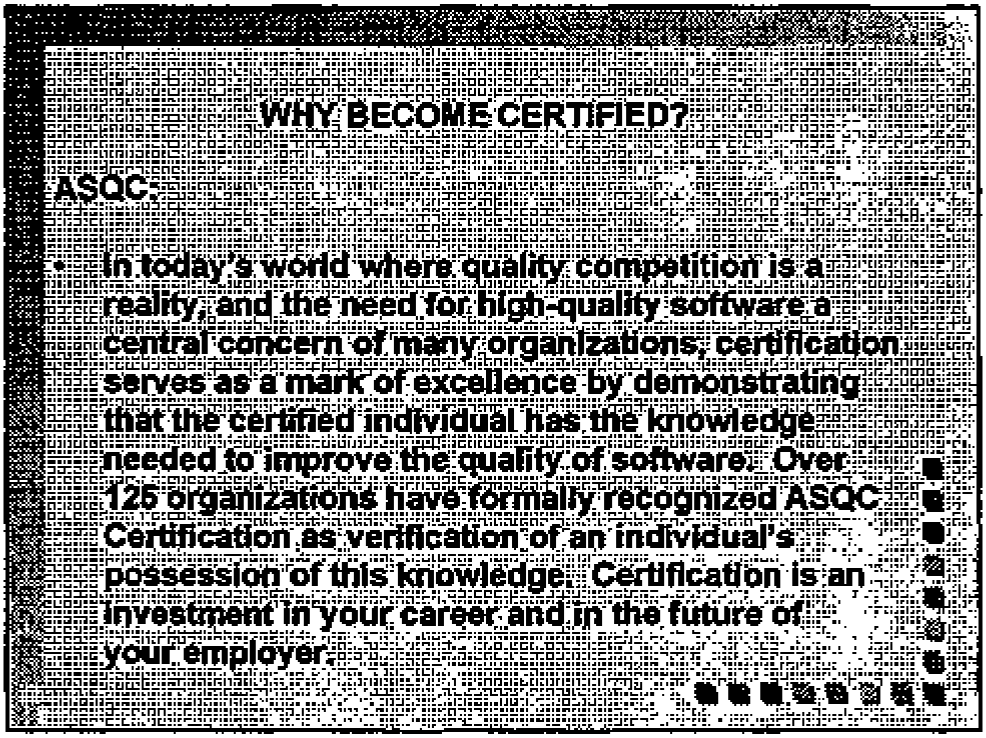
Gary Ford, Software Engineering Institute (SEI) Technical Staff, presented a paper at the 1993 SEI Software Engineering Symposium entitled, "The Current State of Certification & Licensing of Software Engineers". This paper contained excerpts on professional licensing from three states: Pennsylvania, West Virginia, and New Jersey. NEW JERSEY was the only state identified as actually enacting software development legislation (State of New Jersey, Assembly Bill 4414, New Jersey Software Designers' Licensing Bill).

MOTIVATION for LICENSING ENGINEERS

- Pennsylvania Statute, "...to safeguard life, health or property and to promote the general welfare..."
- West Virginia Statute, "...to safeguard life, health or property and to promote the public welfare..."
- New Jersey Statute, "...the public interest requires the regulation of the practice of software designing and the establishment of clear standards for software designers, and the welfare of the citizens of this State will be protected by identifying to the public those individuals who are qualified and legally authorized to practice software designing."

LICENSING ENGINEERS in OTHER STATES

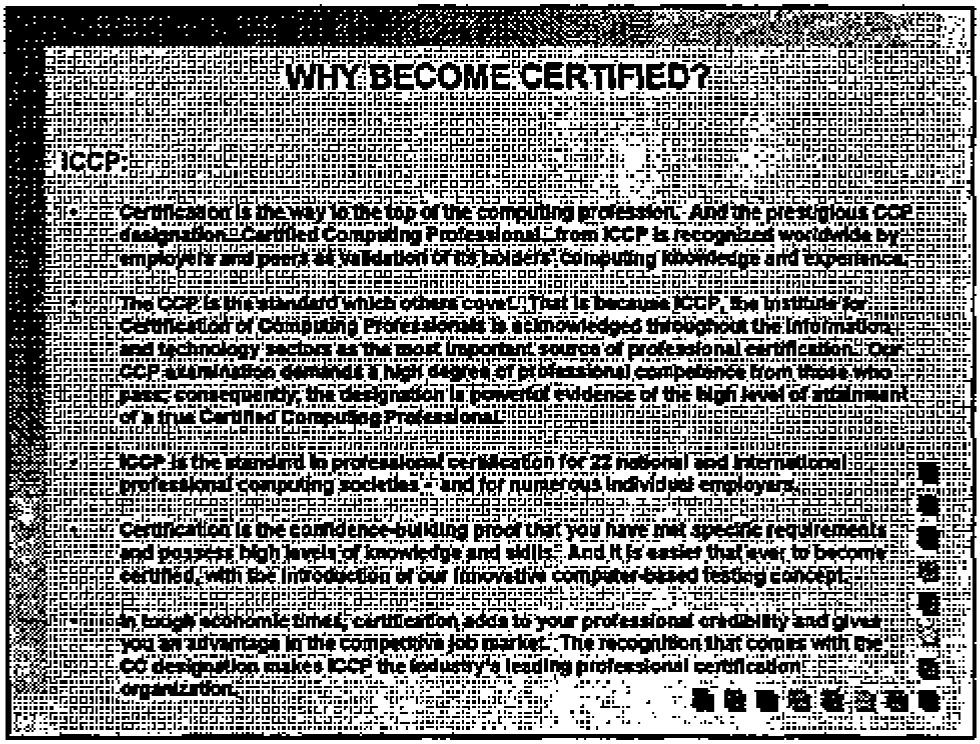
Members of the SQAS Work Group tried to determine Software Engineering/Development licensing efforts in their respective states: California, Colorado, Florida, Missouri, New Mexico, Ohio, South Carolina, Tennessee, and Texas. No evidence was found to document licensing efforts in any of these states.



WHY BECOME CERTIFIED?

ASQC:

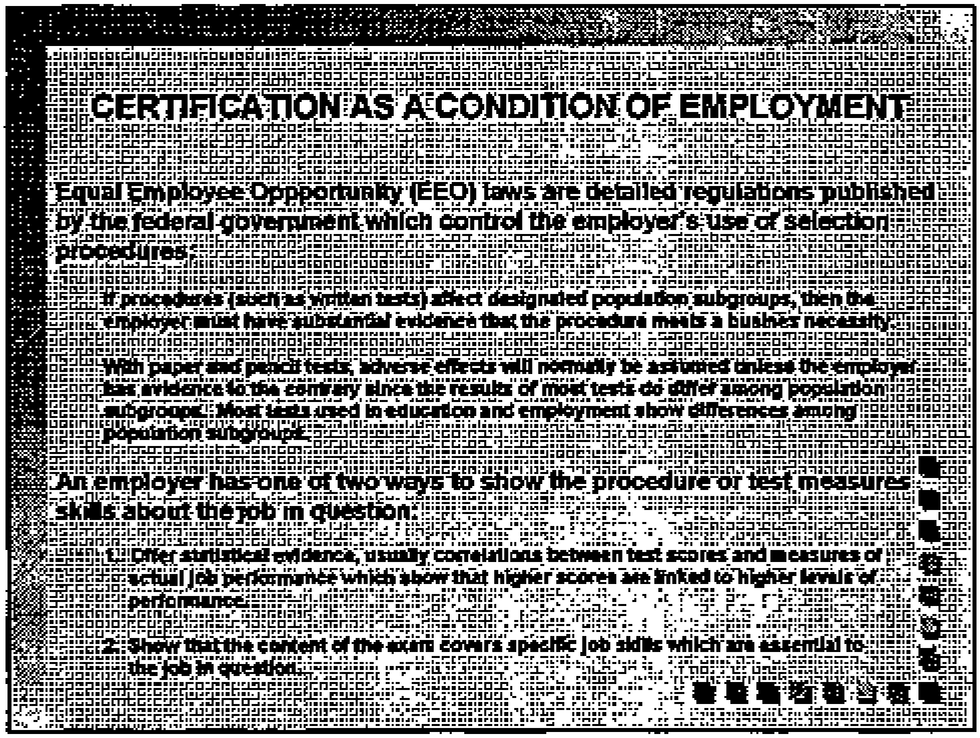
- In today's world where quality competition is a reality, and the need for high-quality software a central concern of many organizations, certification serves as a mark of excellence by demonstrating that the certified individual has the knowledge needed to improve the quality of software. Over 125 organizations have formally recognized ASQC Certification as verification of an individual's possession of this knowledge. Certification is an investment in your career and in the future of your employer.



WHY BECOME CERTIFIED?

ICCP:

- Certification is the way to the top of the computing profession. And the prestigious CCP designation—Certified Computing Professional—from ICCP is recognized worldwide by employers and peers as validation of its holders' computing knowledge and experience.
- The CCP is the standard which others covet. That is because ICCP, the Institute for Certification of Computing Professionals is acknowledged throughout the information and technology sectors as the most important source of professional certification. Our CCP examination demands a high degree of professional competence from those who pass; consequently, the designation is powerful evidence of the high level of attainment of a true Certified Computing Professional.
- ICCP is the standard in professional certification for 22 national and international professional computing societies - and for numerous individual employers.
- Certification is the confidence-building proof that you have met specific requirements and possess high levels of knowledge and skills. And it is easier that ever to become certified, with the introduction of our innovative computer-based testing concept.
- In tough economic times, certification adds to your professional credibility and gives you an advantage in the competitive job market. The recognition that comes with the CC designation makes ICCP the industry's leading professional certification organization.



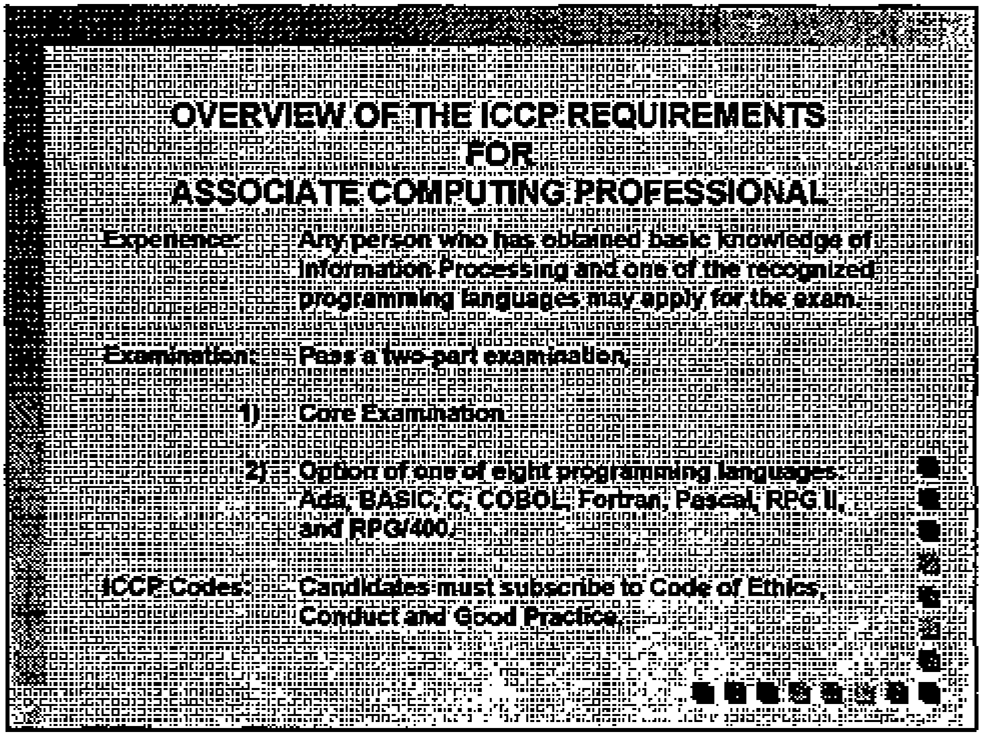
CERTIFICATION AS A CONDITION OF EMPLOYMENT

Equal Employee Oppportunity (EEO) laws are detailed regulations published by the federal government which control the employer's use of selection procedures.

- If procedures (such as written tests) affect designated population subgroups, then the employer must have substantial evidence that the procedure meets a busines necessity.
- With paper and pencil tests, adverse effects will normally be assumed unless the employer has evidence to the contrary since the results of most tests do differ among population subgroups. Most tests used in education and employment show differences among population subgroups.

An employer has one of two ways to show the procedure or test measures skills about the job in question:

1. Offer statistical evidence, usually correlations between test scores and measures of actual job performance which show that higher scores are linked to higher levels of performance.
2. Show that the content of the exam covers specific job skills which are essential to the job in question.



OVERVIEW OF THE ICCP REQUIREMENTS
FOR
ASSOCIATE COMPUTING PROFESSIONAL

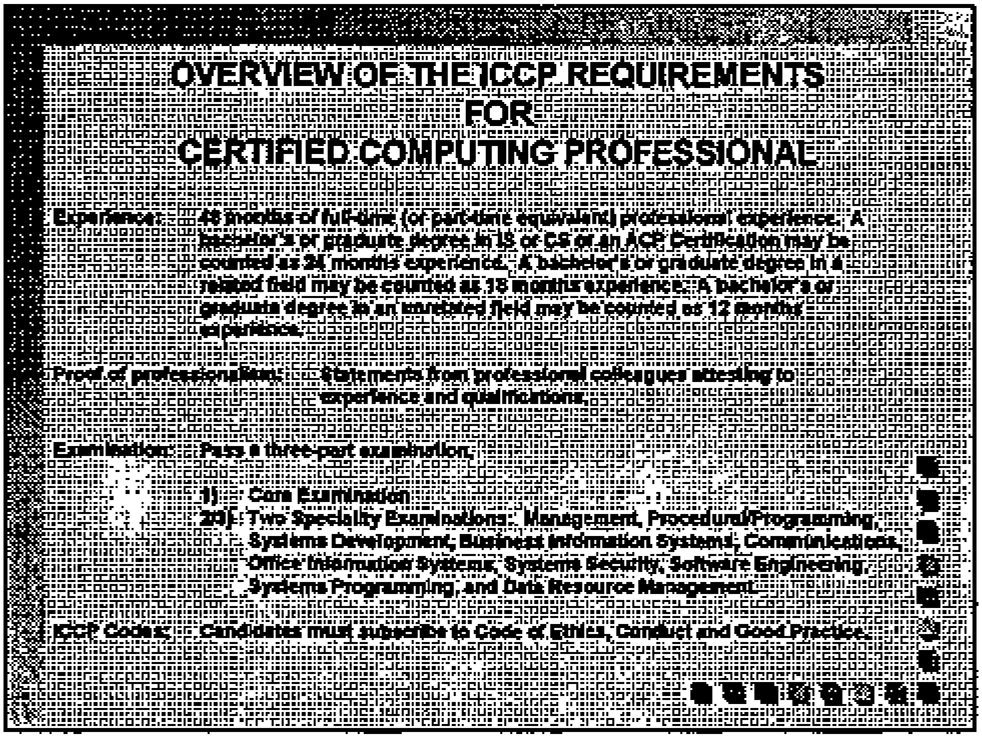
Experience: Any person who has obtained basic knowledge of Information Processing and one of the recognized programming languages may apply for the exam.

Examination: Pass a two-part examination;

1) Core Examination

2) Option of one of eight programming languages: Ada, BASIC, C, COBOL, Fortran, Pascal, RPG II, and RPG/400.

ICCP Codes: Candidates must subscribe to Code of Ethics, Conduct and Good Practice.



OVERVIEW OF THE ICCP REQUIREMENTS
FOR
CERTIFIED COMPUTING PROFESSIONAL

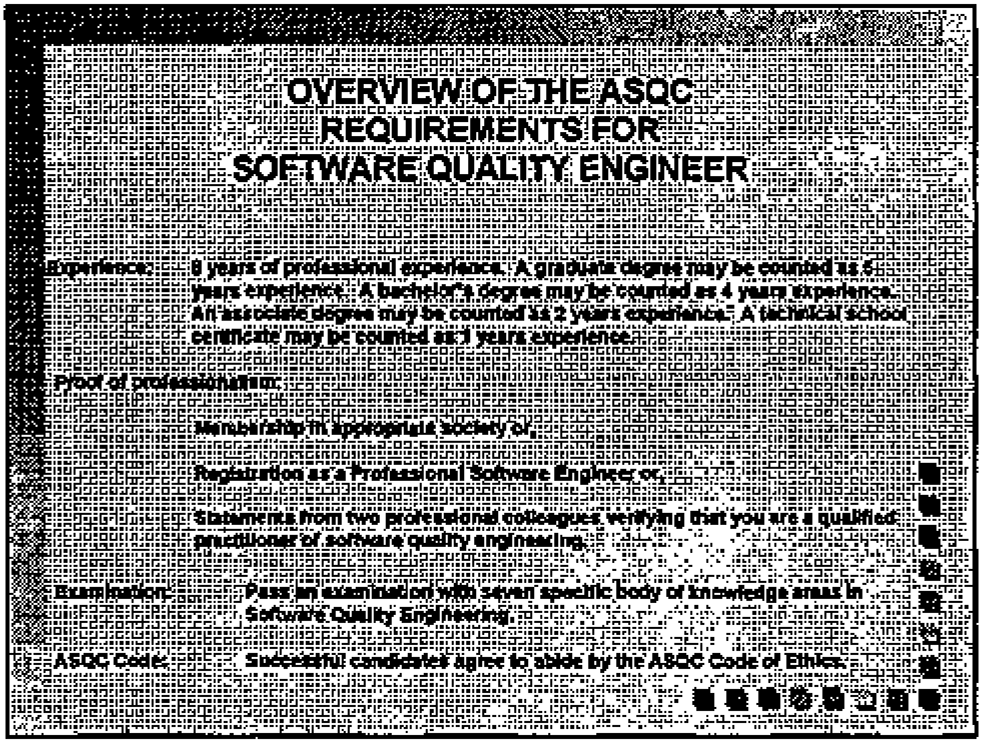
Experience: 48 months of full-time (or part-time equivalent) professional experience. A bachelor's or graduate degree in IS or CS or an ACP Certification may be counted as 24 months experience. A bachelor's or graduate degree in a related field may be counted as 18 months experience. A bachelor's or graduate degree in an unrelated field may be counted as 12 months experience.

Proof of professionalism: Statements from professional colleagues attesting to experience and qualifications.

Examination: Pass a three-part examination;

1) Core Examination
2/3) Two Speciality Examinations: Management, Procedural/Programming, Systems Development, Business Information Systems, Communications, Office Information Systems, Systems Security, Software Engineering, Systems Programming, and Data Resource Management.

ICCP Codes: Candidates must subscribe to Code of Ethics, Conduct and Good Practice.



OVERVIEW OF THE ASQC REQUIREMENTS FOR SOFTWARE QUALITY ENGINEER

Experience: 8 years of professional experience. A graduate degree may be counted as 5 years experience. A bachelor's degree may be counted as 4 years experience. An associate degree may be counted as 2 years experience. A technical school certificate may be counted as 1 years experience.

Proof of professionalism:

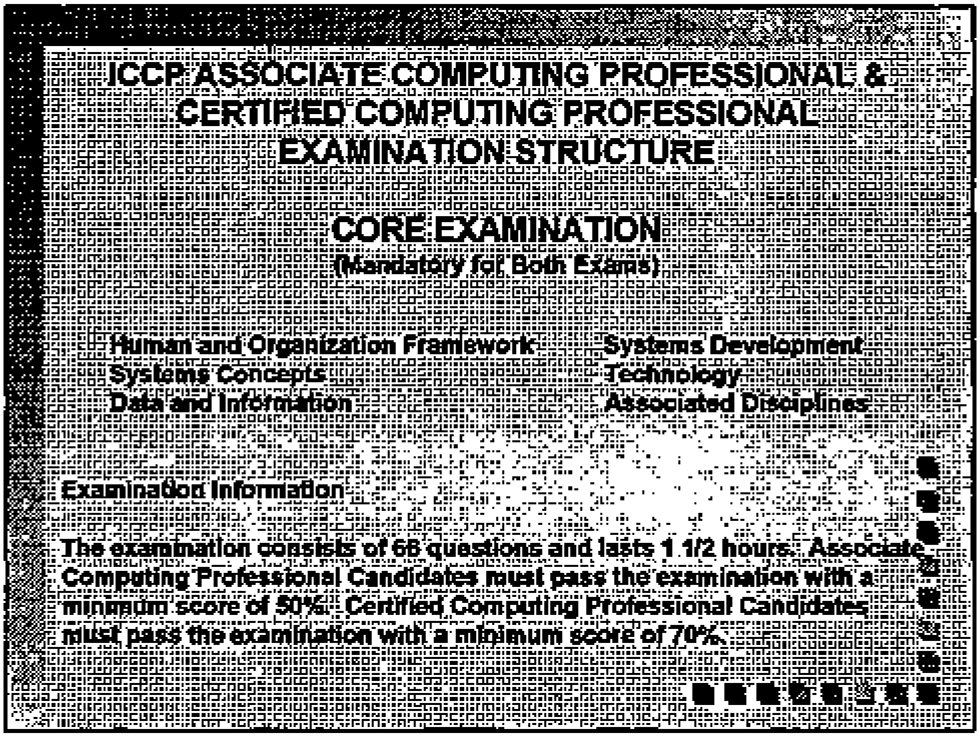
Membership in appropriate society or,

Registration as a Professional Software Engineer or,

Statements from two professional colleagues verifying that you are a qualified practitioner of software quality engineering.

Examination: Pass an examination with seven specific body of knowledge areas in Software Quality Engineering.

ASQC Code: Successful candidates agree to abide by the ASQC Code of Ethics.



ICCP ASSOCIATE COMPUTING PROFESSIONAL & CERTIFIED COMPUTING PROFESSIONAL EXAMINATION STRUCTURE

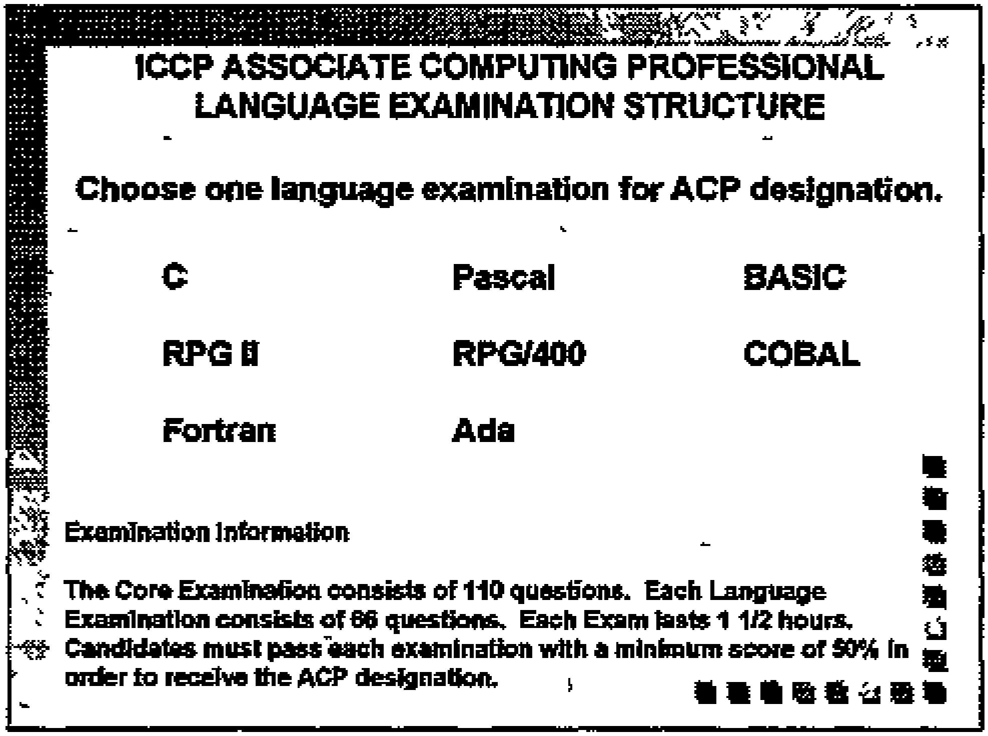
CORE EXAMINATION

(Mandatory for Both Exams)

Human and Organization Framework	Systems Development
Systems Concepts	Technology
Data and Information	Associated Disciplines

Examination Information

The examination consists of 68 questions and lasts 1 1/2 hours. Associate Computing Professional Candidates must pass the examination with a minimum score of 50%. Certified Computing Professional Candidates must pass the examination with a minimum score of 70%.



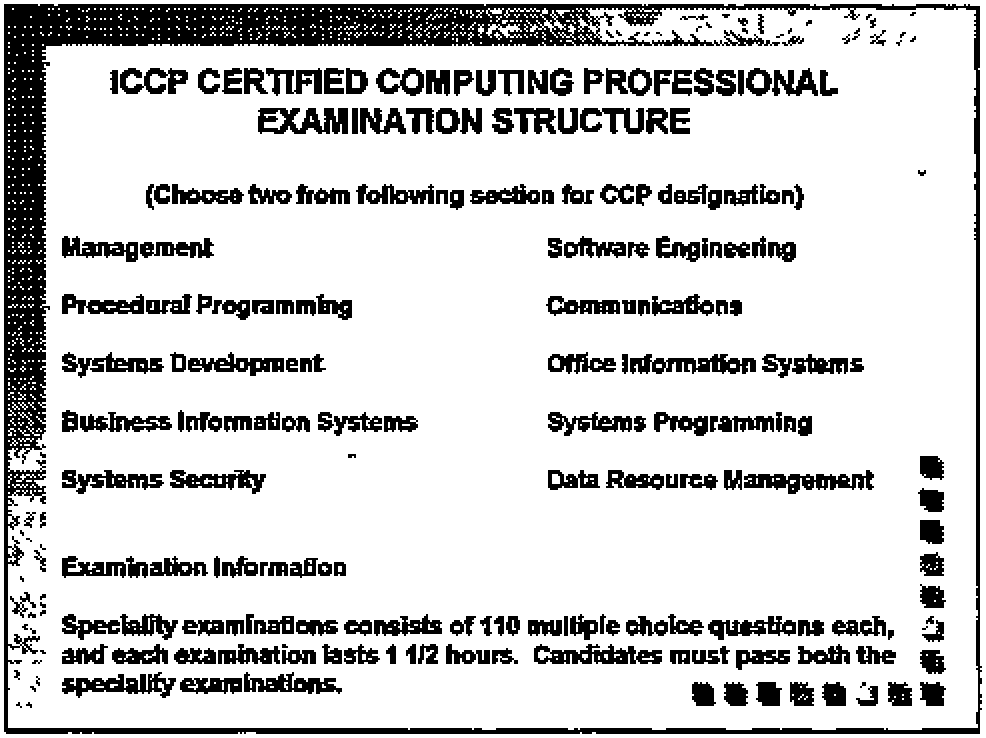
ICCP ASSOCIATE COMPUTING PROFESSIONAL LANGUAGE EXAMINATION STRUCTURE

Choose one language examination for ACP designation.

C	Pascal	BASIC
RPG II	RPG/400	COBAL
Fortran	Ada	

Examination Information

The Core Examination consists of 110 questions. Each Language Examination consists of 66 questions. Each Exam lasts 1 1/2 hours. Candidates must pass each examination with a minimum score of 50% in order to receive the ACP designation.



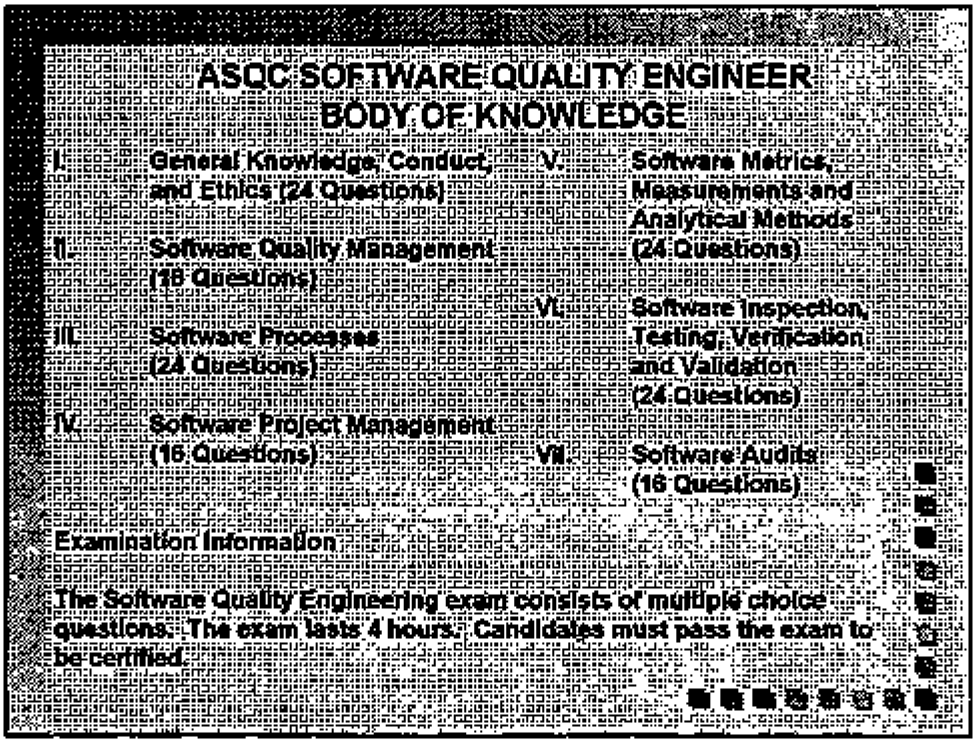
ICCP CERTIFIED COMPUTING PROFESSIONAL EXAMINATION STRUCTURE

(Choose two from following section for CCP designation)

Management	Software Engineering
Procedural Programming	Communications
Systems Development	Office Information Systems
Business Information Systems	Systems Programming
Systems Security	Data Resource Management

Examination Information

Speciality examinations consists of 110 multiple choice questions each, and each examination lasts 1 1/2 hours. Candidates must pass both the speciality examinations.



ASQC SOFTWARE QUALITY ENGINEER BODY OF KNOWLEDGE

I. General Knowledge, Conduct, and Ethics (24 Questions)

II. Software Quality Management (18 Questions)

III. Software Processes (24 Questions)

IV. Software Project Management (16 Questions)

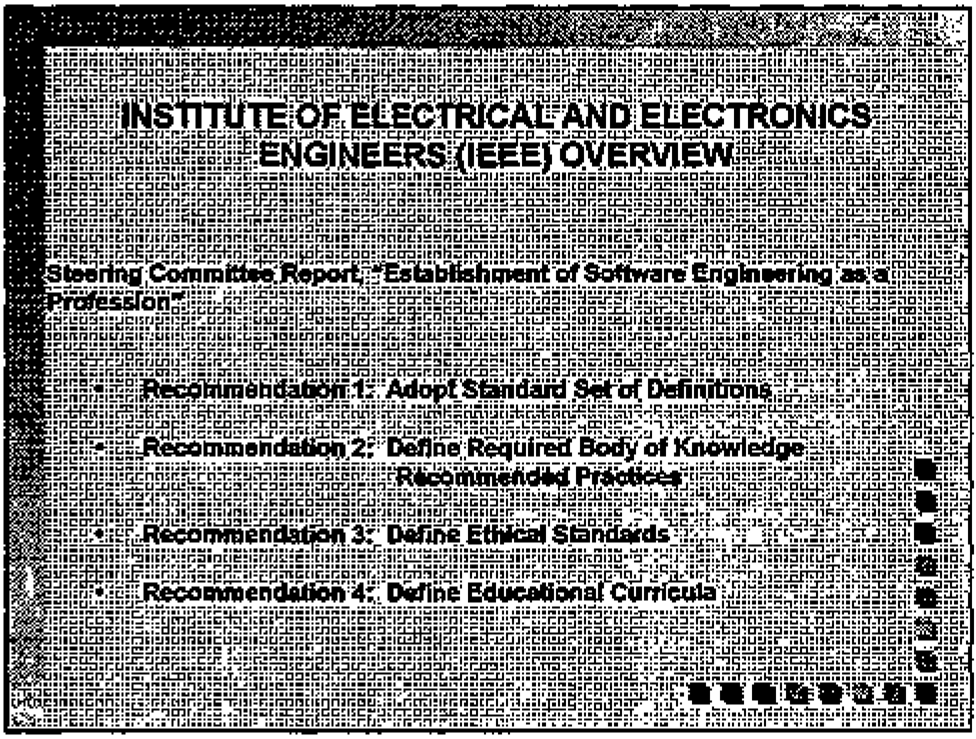
V. Software Metrics, Measurements and Analytical Methods (24 Questions)

VI. Software Inspection, Testing, Verification and Validation (24 Questions)

VII. Software Audits (16 Questions)

Examination Information

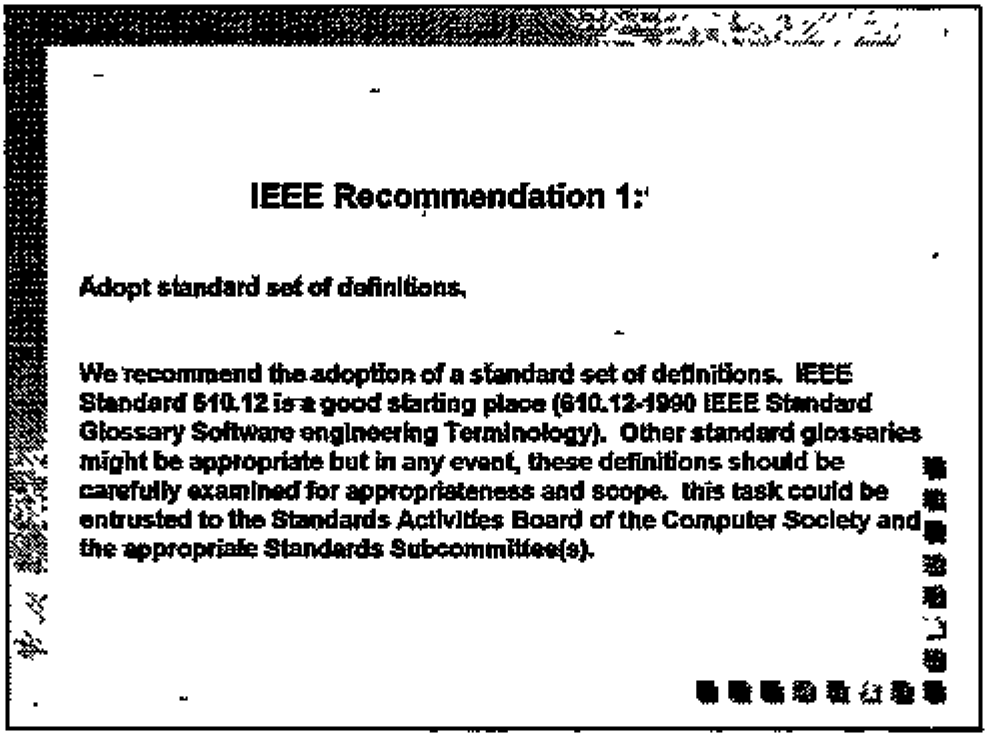
The Software Quality Engineering exam consists of multiple choice questions. The exam lasts 4 hours. Candidates must pass the exam to be certified.



INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE) OVERVIEW

Steering Committee Report, "Establishment of Software Engineering as a Profession"

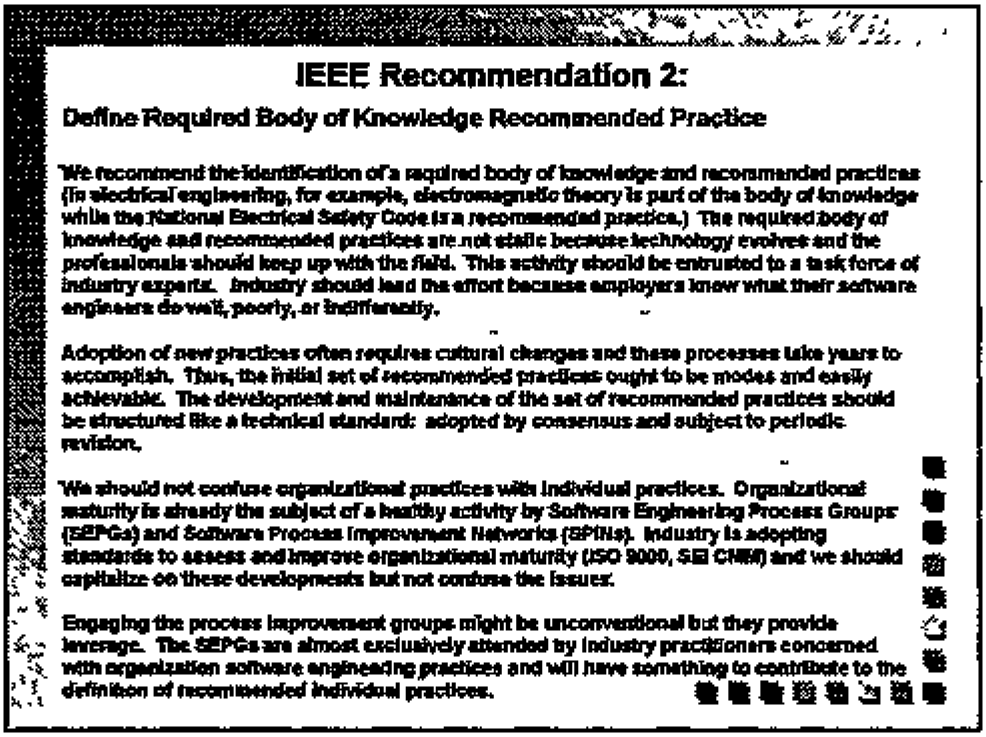
- Recommendation 1: Adopt Standard Set of Definitions
- Recommendation 2: Define Required Body of Knowledge
  Recommended Practices
- Recommendation 3: Define Ethical Standards
- Recommendation 4: Define Educational Curricula



IEEE Recommendation 1:

Adopt standard set of definitions.

We recommend the adoption of a standard set of definitions. IEEE Standard 610.12 is a good starting place (610.12-1990 IEEE Standard Glossary Software engineering Terminology). Other standard glossaries might be appropriate but in any event, these definitions should be carefully examined for appropriateness and scope. this task could be entrusted to the Standards Activities Board of the Computer Society and the appropriate Standards Subcommittee(s).



IEEE Recommendation 2:

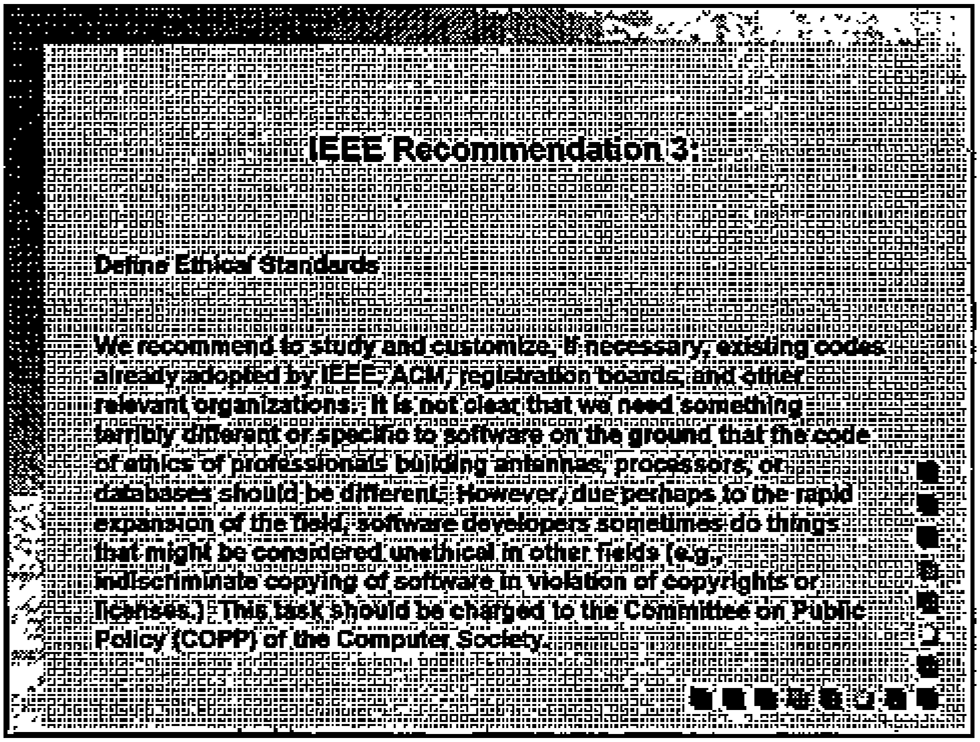
Define Required Body of Knowledge Recommended Practice

We recommend the identification of a required body of knowledge and recommended practices (in electrical engineering, for example, electromagnetic theory is part of the body of knowledge while the National Electrical Safety Code is a recommended practice.) The required body of knowledge and recommended practices are not static because technology evolves and the professionals should keep up with the field. This activity should be entrusted to a task force of industry experts. Industry should lead the effort because employers know what their software engineers do well, poorly, or indifferently.

Adoption of new practices often requires cultural changes and these processes take years to accomplish. Thus, the initial set of recommended practices ought to be modes and easily achievable. The development and maintenance of the set of recommended practices should be structured like a technical standard: adopted by consensus and subject to periodic revision.

We should not confuse organizational practices with individual practices. Organizational maturity is already the subject of a healthy activity by Software Engineering Process Groups (SEPGs) and Software Process Improvement Networks (SPINs). Industry is adopting standards to assess and improve organizational maturity (ISO 9000, SEI CMM) and we should capitalize on these developments but not confuse the issues.

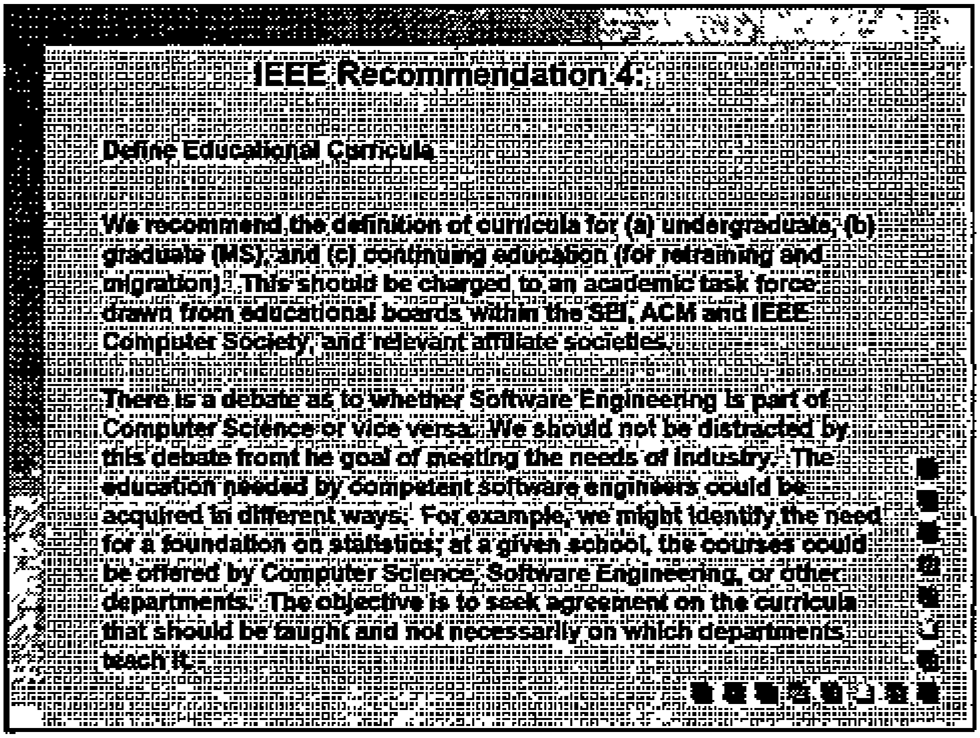
Engaging the process improvement groups might be unconventional but they provide leverage. The SEPGs are almost exclusively attended by industry practitioners concerned with organization software engineering practices and will have something to contribute to the definition of recommended individual practices.



IEEE Recommendation 3:

Define Ethical Standards

We recommend to study and customize, if necessary, existing codes already adopted by IEEE, ACM, registration boards, and other relevant organizations. It is not clear that we need something terribly different or specific to software on the ground that the code of ethics of professionals building antennas, processors, or databases should be different. However, due perhaps to the rapid expansion of the field, software developers sometimes do things that might be considered unethical in other fields (e.g., indiscriminate copying of software in violation of copyrights or licenses.) This task should be charged to the Committee on Public Policy (COPP) of the Computer Society.

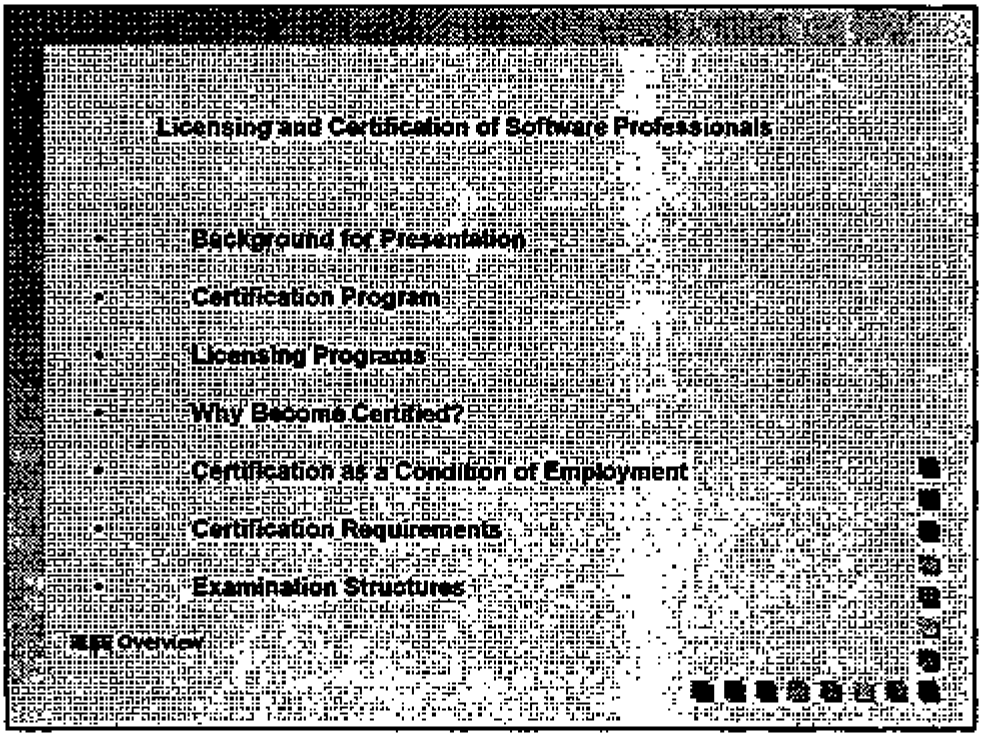


IEEE Recommendation 4:

Define Educational Curricula

We recommend the definition of curricula for (a) undergraduate, (b) graduate (MS), and (c) continuing education (for retraining and migration). This should be charged to an academic task force drawn from educational boards within the SEI, ACM and IEEE Computer Society, and relevant affiliate societies.

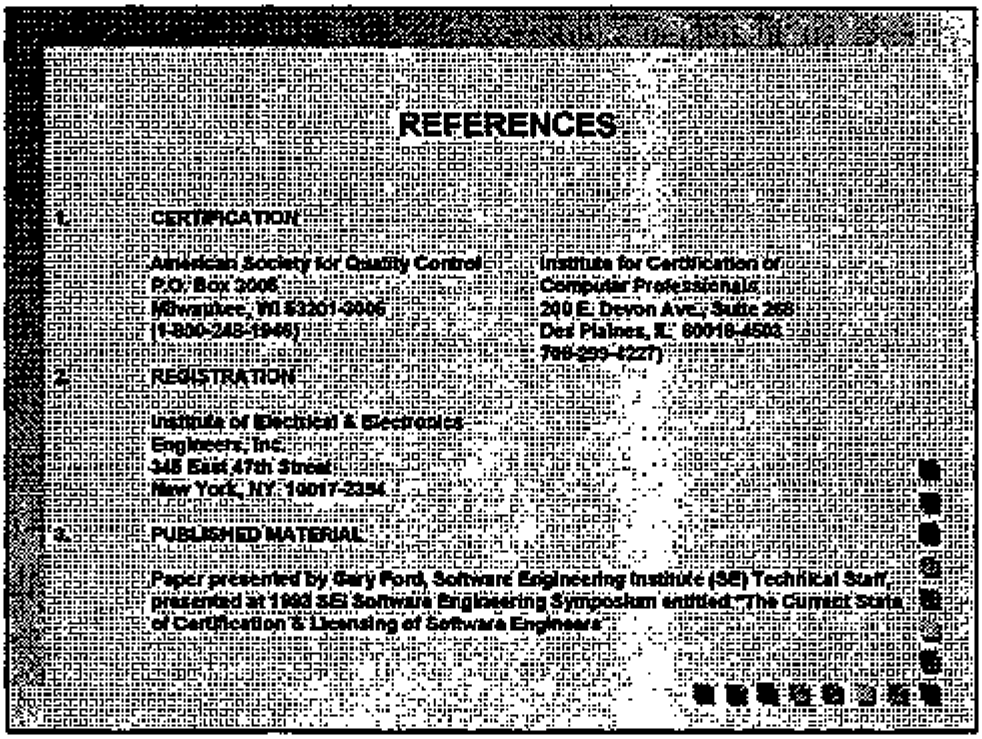
There is a debate as to whether Software Engineering is part of Computer Science or vice versa. We should not be distracted by this debate fromt he goal of meeting the needs of industry. The education needed by competent software engineers could be acquired in different ways. For example, we might identify the need for a foundation on statistics; at a given school, the courses could be offered by Computer Science, Software Engineering, or other departments. The objective is to seek agreement on the curricula that should be taught and not necessarily on which departments teach it.



Licensing and Certification of Software Professionals

- Background for Presentation
- Certification Program
- Licensing Programs
- Why Become Certified?
- Certification as a Condition of Employment
- Certification Requirements
- Examination Structures

IEEE Overview



REFERENCES

1. CERTIFICATION

American Society for Quality Control
P.O. Box 3005
Milwaukee, WI 53201-3005
(1-800-248-1946)

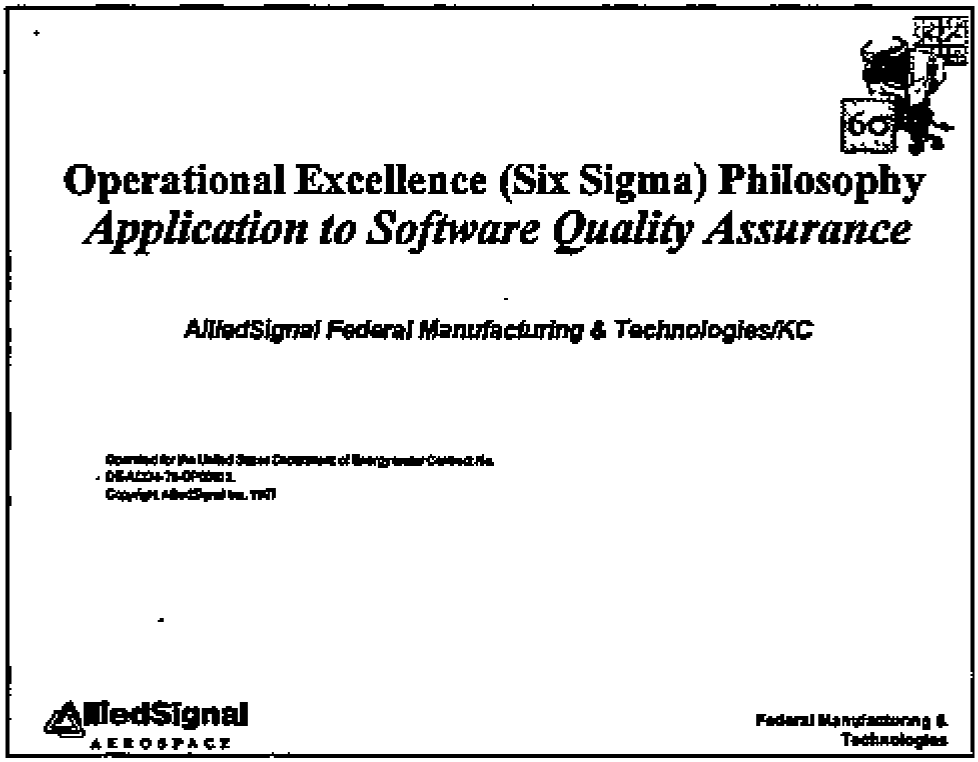
Institute for Certification of
Computer Professionals
200 E. Devon Ave., Suite 268
Des Plaines, IL 60018-4503
708-299-4227)

2. REGISTRATION

Institute of Electrical & Electronics
Engineers, Inc.
345 East 47th Street
New York, NY 10017-2394

3. PUBLISHED MATERIAL

Paper presented by Gary Ford, Software Engineering Institute (SEI) Technical Staff, presented at 1992 SEI Software Engineering Symposium entitled "The Current State of Certification & Licensing of Software Engineers"



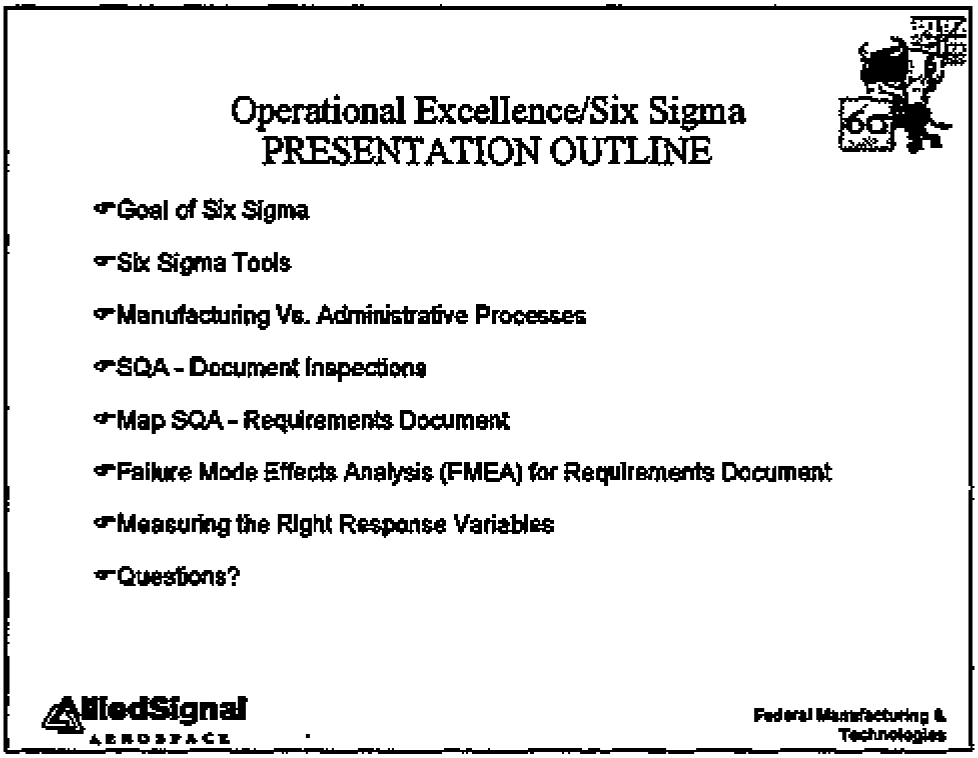
Operational Excellence (Six Sigma) Philosophy
Application to Software Quality Assurance

AlliedSignal Federal Manufacturing & Technologies/KC

Operated for the United States Department of Energy under Contract No.
DE-AC04-76-DP00613.

AlliedSignal
AEROSPACE

Federal Manufacturing &
Technologies



Operational Excellence/Six Sigma
PRESENTATION OUTLINE

☞Goal of Six Sigma

☞Six Sigma Tools

☞Manufacturing Vs. Administrative Processes

☞SQA - Document Inspections

☞Map SQA - Requirements Document

☞Failure Mode Effects Analysis (FMEA) for Requirements Document

☞Measuring the Right Response Variables

☞Questions?

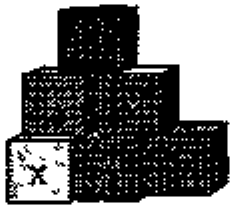
AlliedSignal
AEROSPACE

Federal Manufacturing &
Technologies

Operational Excellence/Six Sigma
GOAL OF SIX SIGMA

Understand the relationship between the critical factors (process parameters) and the response variables (process results), and then reduce the variability about the target

$$Y = F(x)$$



Knowing the 'Y's lead to identifying the 'y's, and understanding of the 'x's which control your system.

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies

Operational Excellence/Six Sigma
GOAL OF SIX SIGMA

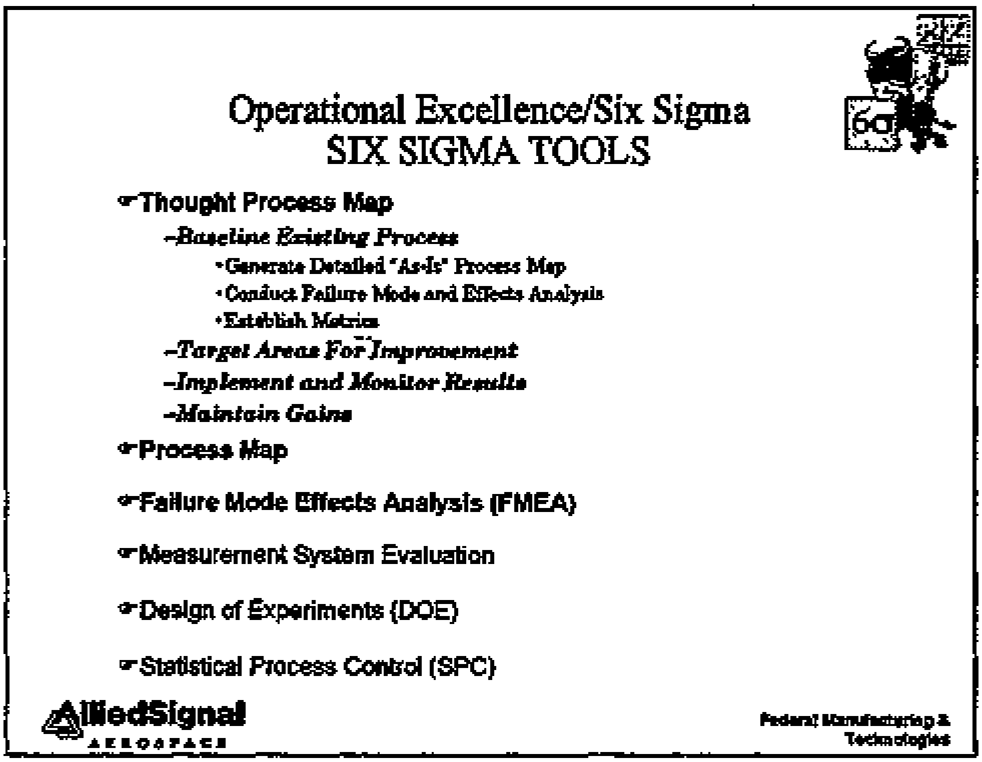
$$Y_{\text{(SQA - software project)}} = [y_{\text{(SQA - requirements)}} + y_{\text{(SQA - design)}} + y_{\text{(SQA - code)}} + \ldots]$$

$$y_{\text{(SQA - requirements)}} = [x_{\text{(customer)}} + x_{\text{(SW eng,SW developers)}} + x_{\text{(formal system)}} + x_{\text{(schedule)}} + x_{\text{(management)}}]^{\text{(experience, education/training)} \quad \text{(written, verbal)}}$$

- Use tools appropriately and discriminably (what question are you trying to answer)
- Understand the process as it now exists BEFORE any improvements are even suggested.

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies



Operational Excellence/Six Sigma
SIX SIGMA TOOLS

☞**Thought Process Map**

- ***Baseline Existing Process***
  - Generate Detailed "As-Is" Process Map
  - Conduct Failure Mode and Effects Analysis
  - Establish Metrics
- ***Target Areas For Improvement***
- ***Implement and Monitor Results***
- ***Maintain Gains***

☞**Process Map**

☞**Failure Mode Effects Analysis (FMEA)**

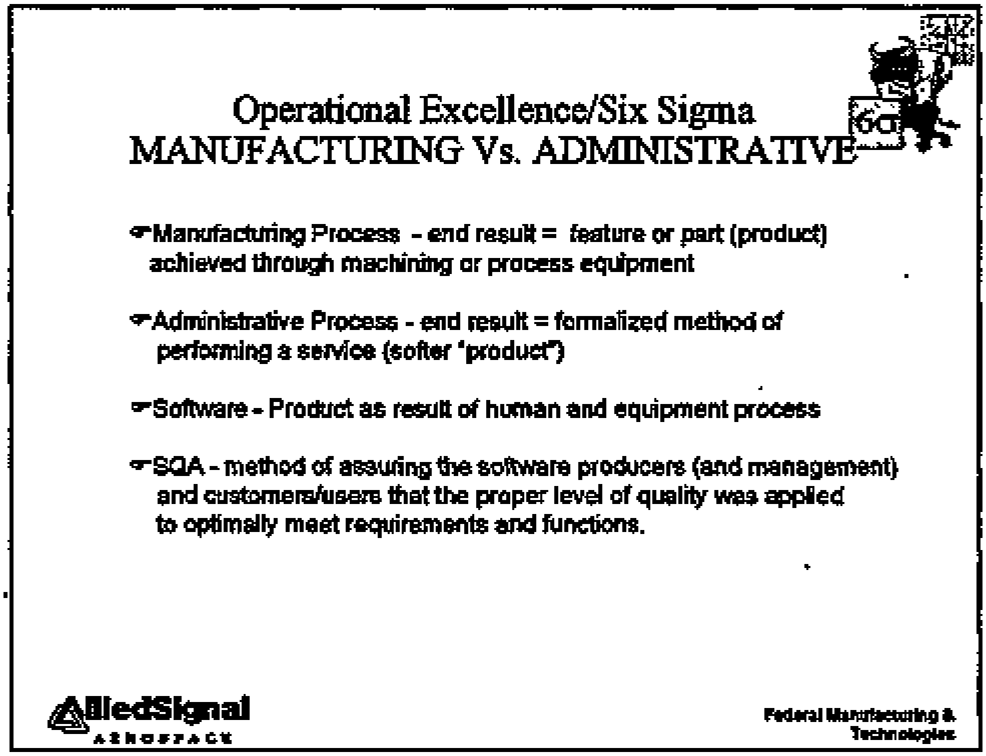
☞Measurement System Evaluation

☞Design of Experiments (DOE)

☞Statistical Process Control (SPC)

AlliedSignal
AEROSPACE

Federal Manufacturing &
Technologies



Operational Excellence/Six Sigma
MANUFACTURING Vs. ADMINISTRATIVE

☞Manufacturing Process - end result = feature or part (product) achieved through machining or process equipment

☞Administrative Process - end result = formalized method of performing a service (softer "product")

☞Software - Product as result of human and equipment process

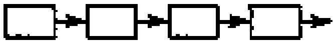
☞SQA - method of assuring the software producers (and management) and customers/users that the proper level of quality was applied to optimally meet requirements and functions.

AlliedSignal
AEROSPACE

Federal Manufacturing &
Technologies

Operational Excellence/Six Sigma
SQA - DOCUMENT INSPECTIONS

- Requirements Document

Process Map  requirements document

- Design Document

- Code

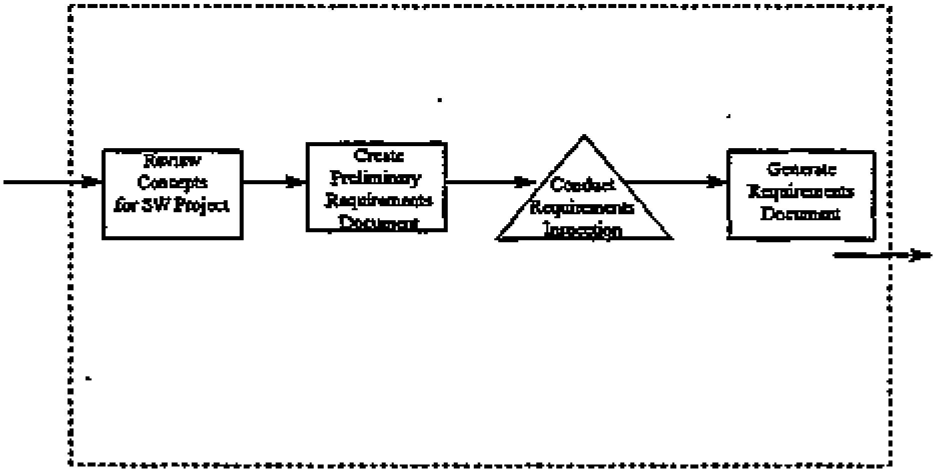
- Testing Document

- Acceptance Document

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies

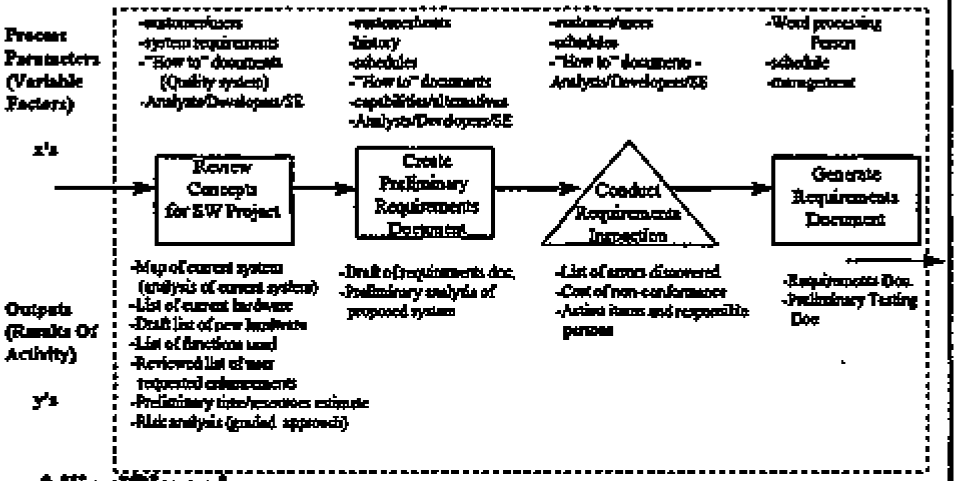
Operational Excellence/Six Sigma
MAP SQA - REQUIREMENTS DOCUMENT
1. PROCESS STEPS


Review Concepts for SW Project
Create Preliminary Requirements Document
Conduct Requirements Inspection
Generate Requirements Document

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies

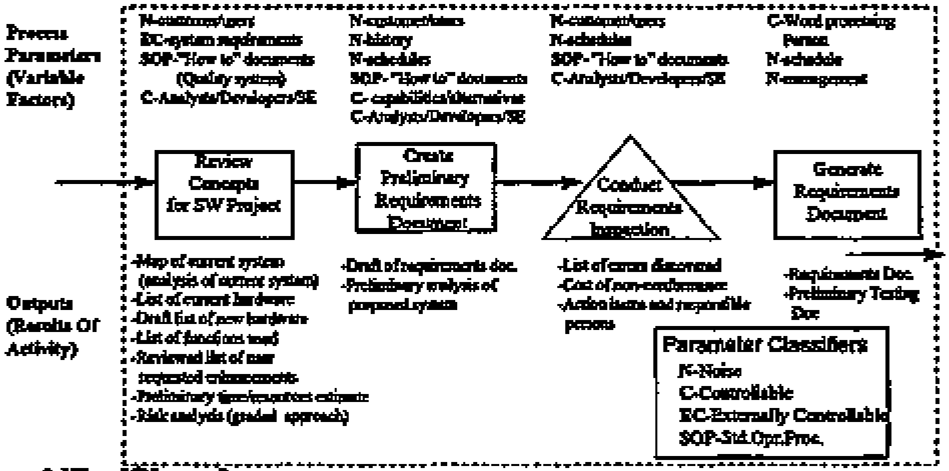
Operational Excellence/Six Sigma
MAP SQA - REQUIREMENTS DOCUMENT
2. IN-PROCESS PARAMETERS AND OUTPUTS


Process Parameters (Variable Factors)
x's
-customers/users
-system requirements
-"How to" documents (Quality system)
-Analysts/Developers/SE
-customers/users
-history
-schedules
-"How to" documents
-capabilities/alternatives
-Analysts/Developers/SE
-customers/users
-schedules
-"How to" documents -
Analysts/Developers/SE
-Word processing Person
-schedule
-management
Review Concepts for SW Project
Create Preliminary Requirements Document
Conduct Requirements Inspection
Generate Requirements Document
Outputs (Results Of Activity)
y's
-Map of current system (analysis of current system)
-List of current hardware
-Draft list of new hardware
-List of functions used
-Reviewed list of user requested enhancements
-Preliminary time/resources estimate
-Risk analysis (graded approach)
-Draft of requirements doc.
-Preliminary analysis of proposed system
-List of errors discovered
-Cost of non-conformance
-Action items and responsible persons
-Requirements Doc.
-Preliminary Testing Doc

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies

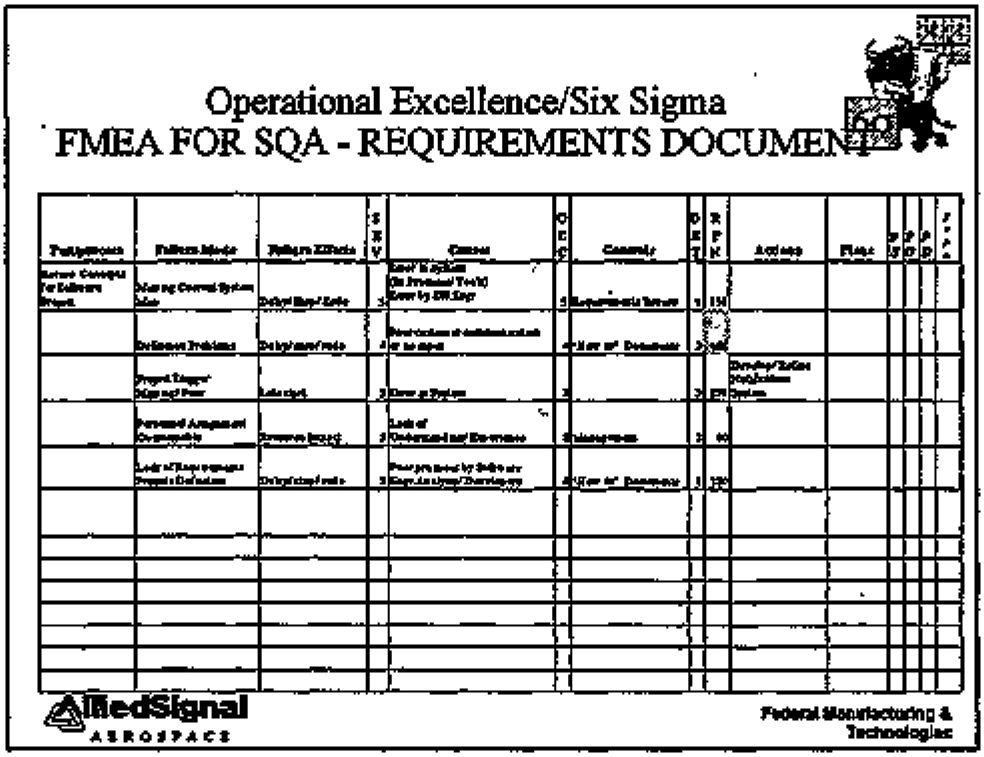
Operational Excellence/Six Sigma
MAP SQA - REQUIREMENTS DOCUMENT
3. CLASSIFY IN-PROCESS PARAMETERS


Process Parameters (Variable Factors)
N-customers/users
EC-system requirements
SOP-"How to" documents (Quality system)
C-Analysts/Developers/SE
N-customers/users
N-history
N-schedules
SOP-"How to" documents
C- capabilities/alternatives
C-Analysts/Developers/SE
N-customers/users
N-schedules
SOP-"How to" documents
C-Analysts/Developers/SE
C-Word processing Person
N-schedule
N-management
Review Concepts for SW Project
Create Preliminary Requirements Document
Conduct Requirements Inspection
Generate Requirements Document
Outputs (Results Of Activity)
-Map of current system (analysis of current system)
-List of current hardware
-Draft list of new hardware
-List of functions used
-Reviewed list of user requested enhancements
-Preliminary time/resources estimate
-Risk analysis (graded approach)
-Draft of requirements doc.
-Preliminary analysis of proposed system
-List of errors discovered
-Cost of non-conformance
-Action items and responsible persons
-Requirements Doc.
-Preliminary Testing Doc

Parameter Classifiers
- N-Noise
- C-Controllable
- EC-Externally Controllable
- SOP-Std.Opr.Proc.

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies

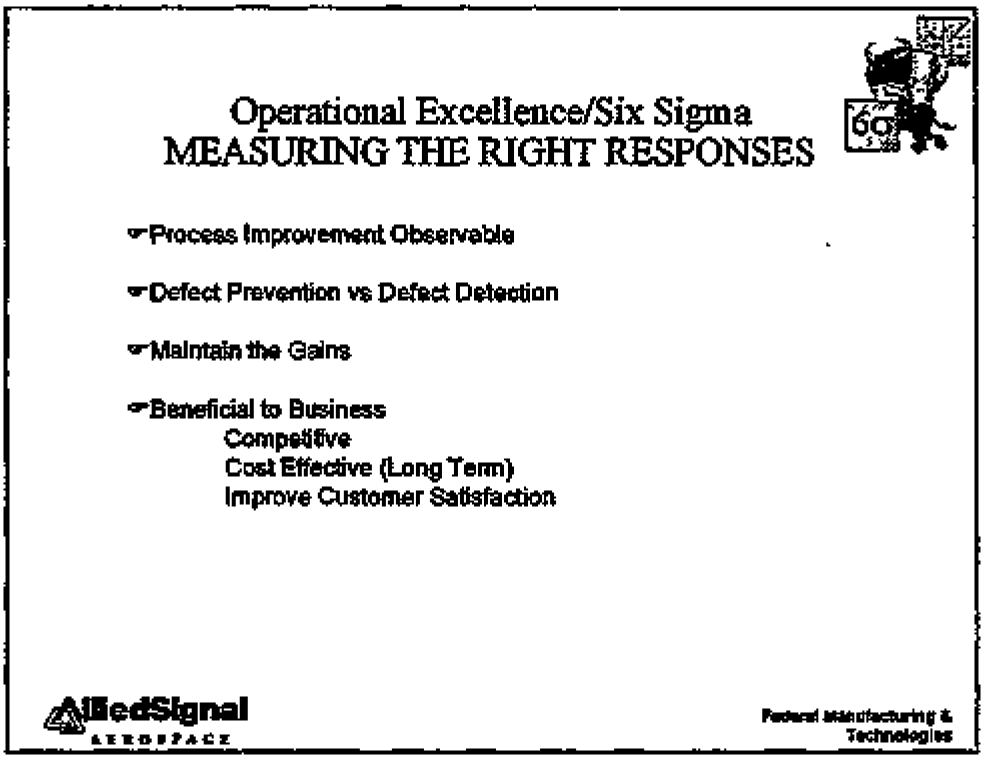


Operational Excellence/Six Sigma
FMEA FOR SQA - REQUIREMENTS DOCUMENT

[illegible]	Failure Mode	Failure Effects	SEV	Causes	OCC	Controls	DET	RPN	Actions	[illegible]	S	O	D	R P N
[illegible]	[illegible]	[illegible]	5	[illegible]	5	[illegible]	[illegible]	150						
	[illegible]	[illegible]	4	[illegible]	4	[illegible]	3	[illegible]						
	[illegible]	[illegible]	3	[illegible]	3		3	[illegible]	[illegible]					
	[illegible]	[illegible]	2	[illegible]	5	[illegible]	3	60						
	[illegible]	[illegible]	5	[illegible]	4	[illegible]	[illegible]	[illegible]						

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies

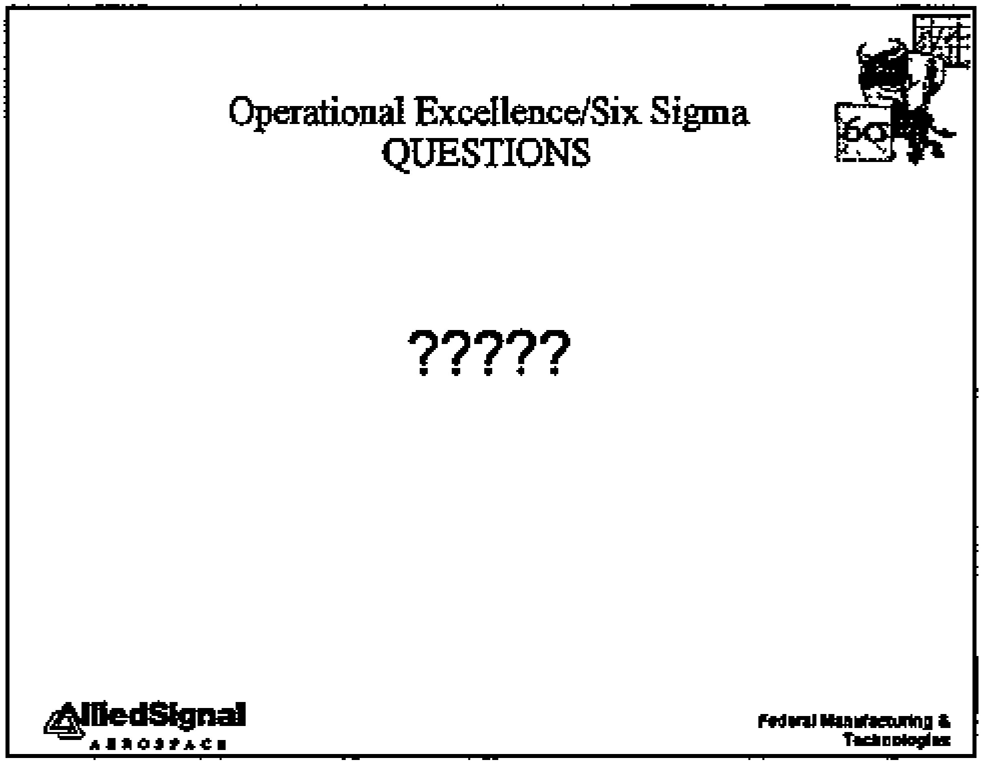


Operational Excellence/Six Sigma
MEASURING THE RIGHT RESPONSES

- Process Improvement Observable
- Defect Prevention vs Defect Detection
- Maintain the Gains
- Beneficial to Business
  - Competitive
  - Cost Effective (Long Term)
  - Improve Customer Satisfaction

AlliedSignal
AEROSPACE

Federal Manufacturing & Technologies



Operational Excellence/Six Sigma QUESTIONS

?????

AlliedSignal
AEROSPACE

Federal Manufacturing &
Technologies

1947 • SOFTWARE QUALITY FORUM • 1997 • DEPARTMENT OF ENERGY NUCLEAR WEAPONS COMPLEX

SOFTWARE ENGINEERING QUALITY

Wrapup and Awards

Best Tutorial Award
Best Presentation Award