NASA - CR - 188077

NASA-CR-188077
19910010409

A Reproduced Copy

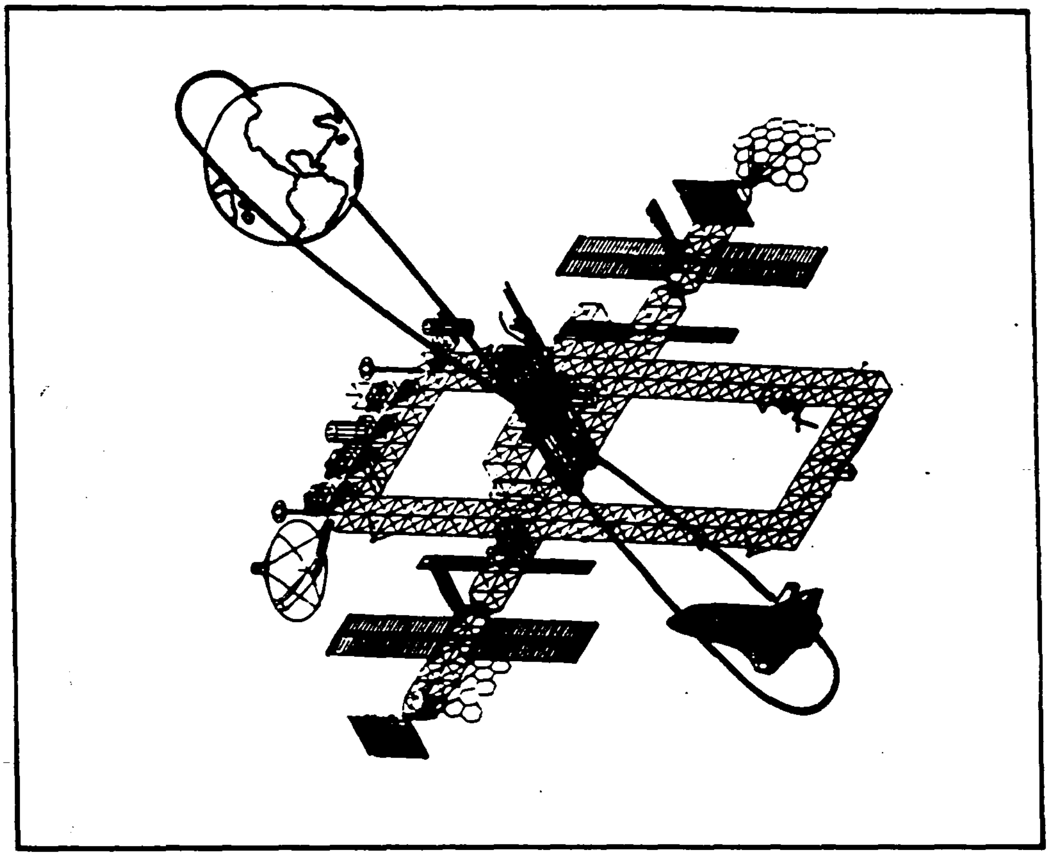
Reproduced for NASA

by the

NASA **Scientific and Technical Information Facility**

FFNo 672 Aug 65

NF01201



RICIS '88 SYMPOSIUM

CONFERENCE PRESENTATION APPENDIX

(NASA-CR-1[illegible]) RICIS 1988 SYMPOSIUM [illegible] PRESENTATION APPENDIX (Houston Univ.) 13[illegible] p CSCL 09B
N91-19722
--THRU--
N91-19730
Unclas
G3/61 [illegible]

N91-19722#
thru
N91-19730#

RICIS '88

CONFERENCE PRESENTATION APPENDIX

Table of Contents

Notes

The Research Institute for Computing and Information Systems' 2nd annual RICIS Symposium was held on November 9-10, 1988 at the South Shore Harbor Resort Hotel in Houston. While the majority of presentations were included in the RICIS '88 Symposium Proceedings, there were some presentations that were not included. Therefore, we have collected the presentation papers and slides that were not in the original proceedings and included them in this volume for your reference.

If you have any questions or require additional copies, please contact:

Software Engineering Professional Education Center
UH-Clear Lake, Box 270
2700 Bay Area Blvd.
Houston, Texas, 77058-1088
(713) 488-9433.

Environment Concerns

- User
  - conceptual integrity
  - tool integration
  - new tool additions
  - life-cycle coverage
  - method supported
  - language(s) supported
  - hardware base
- Environment architect
  - software architecture
  - representation of objects
  - ...
- Tool builder
  - interfaces
  - ...

Environment Roulette

- Backing into environments - incrementally
- False economy - focus on hardware
- Lure of the PC - scaling up
- Heterogeneity

Tool Integration and Tailoring

- Because of heterogeneity and risk factor of monolithic, single vendor environment:

  - assembly of components, e.g., design tool, code generator, document preparation, mailer, editor
  - tailorability of tools and their interaction

Software Heterogeneity

- Host target software development and maintenance, e.g., Ada embedded systems
- Distribution of life-cycle support across machines requiring integration, e.g., NASA Space Station
- Different services on different hardware
- Different models, e.g., access control, project management, configuration management

Implications

- Architecture
  - language-centered
  - process-driven
- Tailorability
- Software maintenance

Problems

- Remote resource management (not centralized)
- Integration of hardware for different services
- Data interoperability:
  - life cycle or life-cycle phase
  - between tools and between machines
- Hardware changes over life cycle
- Need transparent view via uniform interface

Structure-Oriented

- Common representation
- Editor-controlled
- Multiple views
- Semantic-directed browsing
- Examples - Gandalf, Rational, Cornell Synthesizer

Method/Process-Based

Method-based
- Support specific method
- Often include graphical representation
- Some formal foundation
- Examples - JSD, SADT, SA/SD, Statemate, Refine

Process-based
- Support a specific process model
- Enforce a discipline
- Language independent
- Examples - Refine, ISTAR

Toolkit

- Operating system extensions
- Language independence
- Standard interface
- Generality - tools applied to files
- Team cooperation requires discipline
- Examples - UNIX PWB, CAIS, PCTE

Language-Centered

- Support for semantics of specific language
- Interactive
- Incremental
- Encourages exploratory development style
- Examples - Interlisp, Smalltalk, Cedar, Rational

Management Support

- Management of resources
- Management of product
- Management of process
- Management of environment

Environment Trends

- Toolkit
- Language-centered
- Structure-oriented
- Method/process-based

Motivation for Software Development Environments

- Programming support tools
- Management of complexity
- Support for the process

Integration

- Conceptual - across life cycle phases
- Tool - permit tools to pass data
- User interface - user interacts in consistent manner
- Language centered - assumes activities in specific language
- Incremental - tools are finer grained - spreadsheet
- View - allow multiple views

* Not necessarily mutually exclusive

Strengths

- Usual benefits of automation: consistency, repeatability (plus some inflexibility)
- Working representations are captured, online, and deliverable
- Increasing ability to not only analyze, but also query and browse
- Less time spent during inspections and walkthroughs on syntax errors, and more time spent on errors of substance

Trends

- Animation of state transition models of behavior
- Performance modeling
- Enthusiasm for object-oriented design
- Integration of tool sets with different capabilities from different vendors
- Deliverables satisfying 2167

Tools Taxonomy

Development Phase / Operation on Object	Project management	System/SW Req'ts Analysis	SW Requirements Analysis	Preliminary Design	Detailed Design	Coding and Unit Testing	CSC Integration and Testing	CSCI Testing	Sys Integ/Test	Other
Create										
Transform										
Group										
Analyze										
Refine										
Import										
Export										
Other										

Evaluation Attributes

- Ease of use
- Power
- Robustness
- Functionality
- Ease of insertion
- Quality of support

Classification of Methods

		Stages of Development		
View of Problem		specification	design	implementation
	functional	data flow diagrams	PDL	
	structural	entity relationship diagrams	heirarchical structure charts	
	behavioral	state transition diagrams		

- Different views dominate at different stages
- Views are complementary

Tools

- Software supporting the software development process
- Publicly available and supported
  - Offered in expanding commercial market
- Value provided through
  - Relevance to required development activities
  - Assistance to human labor

N91
19723
UNCLAS

Carnegie Mellon University
Software Engineering Institute

Dr. Larry E. Duffel

RICIS Symposium '88

November 9, 1988

Software Development Environments Status and Trends

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Sponsored by the
U.S. Department of Defense

Design Approach for Transformation Procedures

- Identify Common Subset of Tool Capabilities
  - Requires Detailed Understanding of the Tool Suite as well as Application Domain

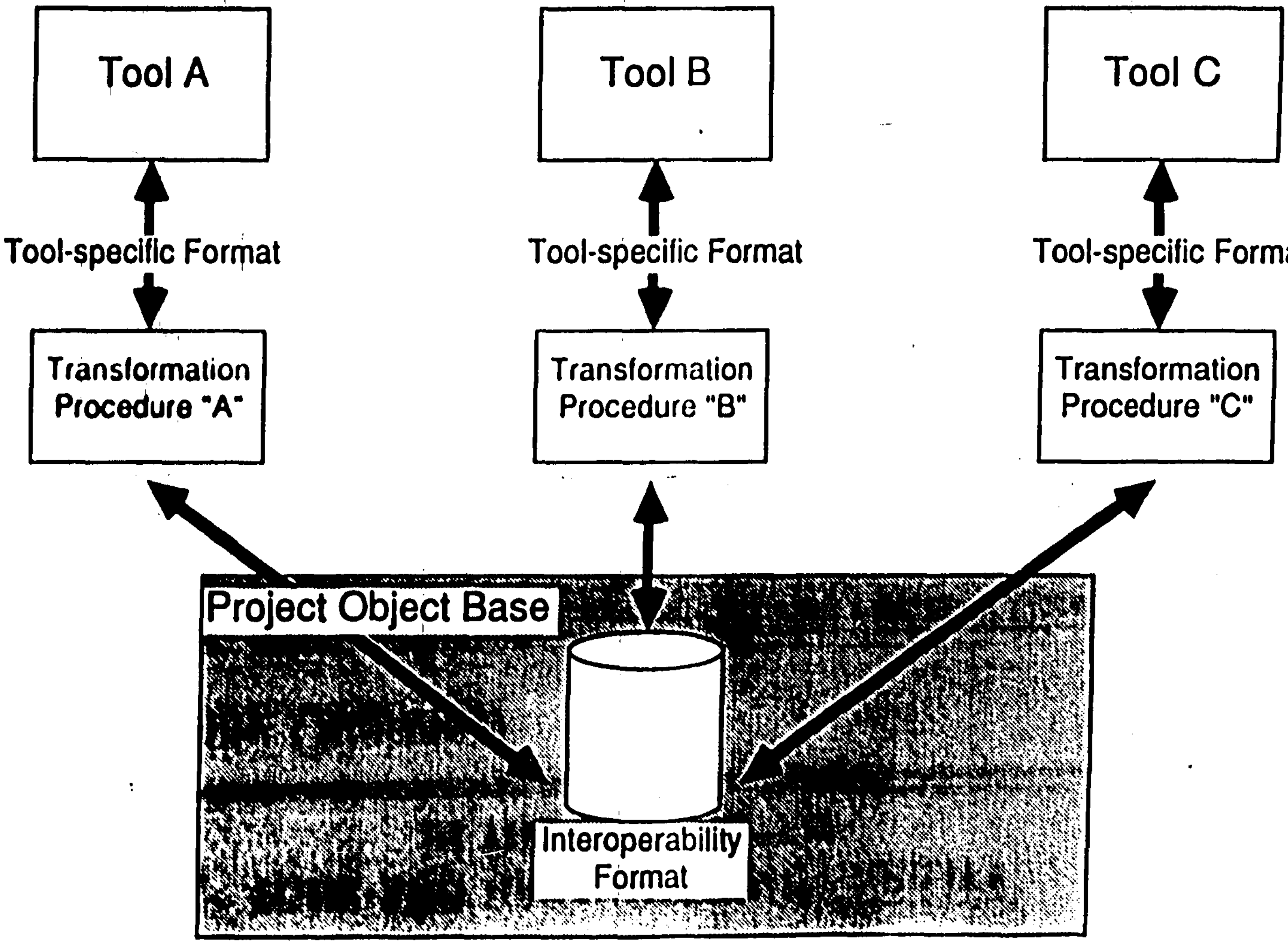
- Develop Text-based, Machine-readable Representation
  - Text-based format avoids machine-dependencies
  - Compiler Technology can be Applied in most Cases

- Common Interoperability Format should be Hidden from Applications, unless it is their Native Format
  - Allows easy modification of Interoperability Format

- Transformation Procedures Require Similar support routines: Design for Portability and Reuse
  - Up to 75% of code in an Interoperability to Tool Transformation Procedure is common.

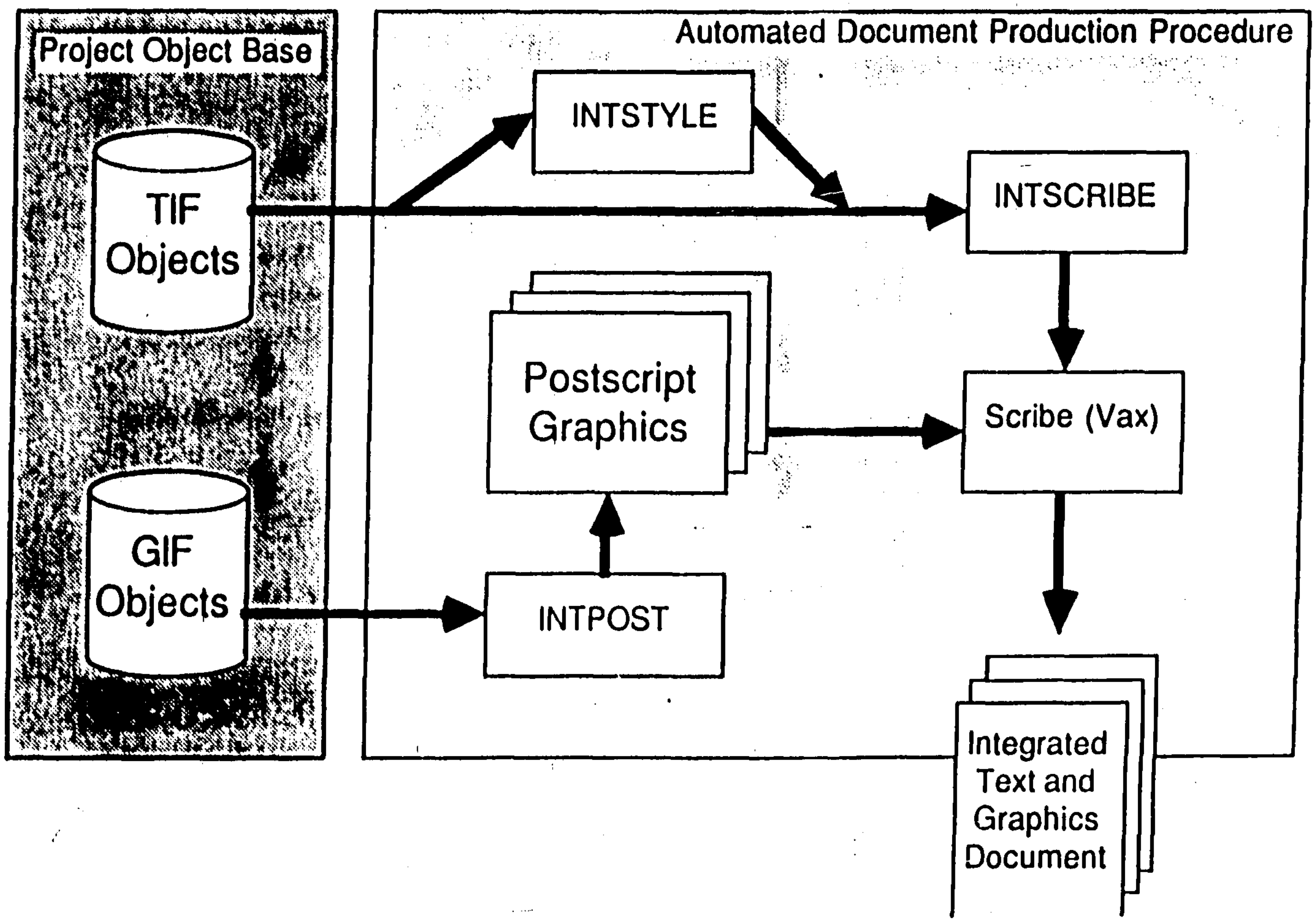
TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

Interoperability Overview



TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

Interoperability Overview



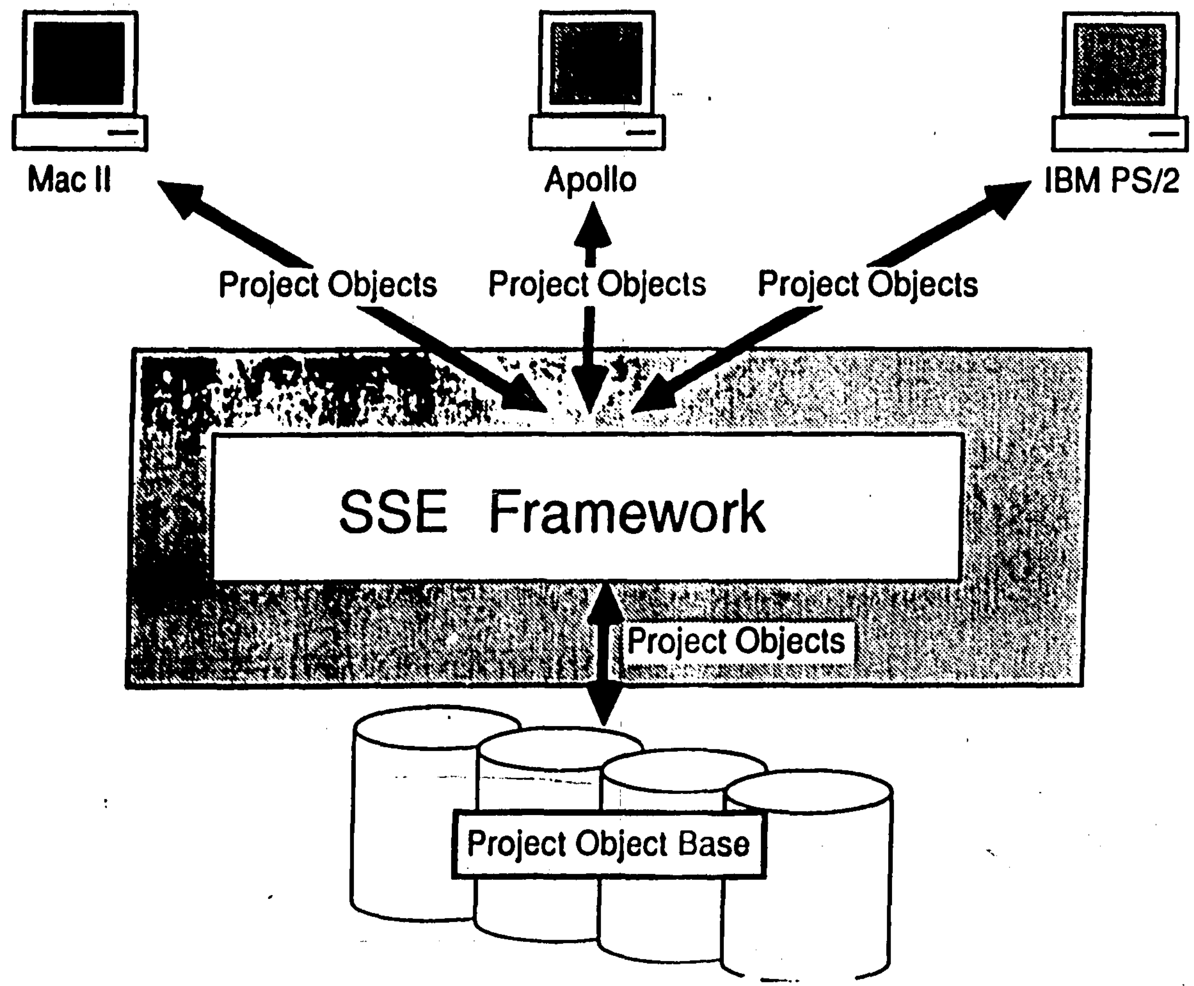
TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

SSE Interoperability Solution

- Develop Data Interoperability Formats for Each Class of Design and Development Tool
- Provide Application-level Views of Data, Versus Network, O/S or File System Views
- Tool/Data Interoperability Is Related to Information-bearing Entities, Not Physical Implementations or Interpretations
- Interoperability Formats Support the Intersection of Tool Capabilities, Not the Union

TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

Interoperability Overview



SSE Interoperability Issues

- Multiple Hosts in a Distributed Environment
  - Vax/VMS
  - IBM/VM
- Multiple Workstations Networked to Hosts
  - Apollo
  - Macintosh II
  - IBM PS/2

TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

SSE Interoperability Issues (cont'd)

- Design Tool Interoperability
  - Cadre Teamwork, Iconix PowerTools, Excellerator
- Graphics Development Tool Interoperability
  - Interleaf, MacDraw, GEM Draw
- Document Development Tool Interoperability
  - Interleaf, Microsoft Word (RTF and DCA Formats)
- Document Production
  - Scribe, Postscript

TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

The Interoperability Problem

- Commonality of Data and Information
- Information Exchange between Diverse Tool Sets
- Interoperability between Heterogeneous Hosts
- Interoperability between Heterogeneous Tools

TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

Past Attempts at Solving the Interoperability Problem

- Common Hardware Architecture
  - IBM 360, SDP, Various PC Standards
- Common or Standard Operating Systems
  - CP/M, MSDOS, Unix/POSIX
- Industry-developed Data Formats
  - DIFF, DCA, RTF
  - IGES, TIFF, GIF
  - EDIF
- Stand-alone Tool Integration
  - Mac O/S
  - Software Backplane

SSE SYSTEM PROJECT

Tool and Data Interoperability in the SSE System

Chuck Shotton
PRC
11/10/88

TOOL AND DATA INTEROPERABILITY IN THE SSE SYSTEM

Overview

- Industry Problems with Program and Data Interoperability
- SSE System Interoperability Issues
- SSE Solutions to Tool and Data Interoperability
- Attaining Heterogeneous Tool/Data Interoperability

N91
19724
UNCLAS

Software Development Methods

- Representations
- Deriving the representations
- Examining the representations

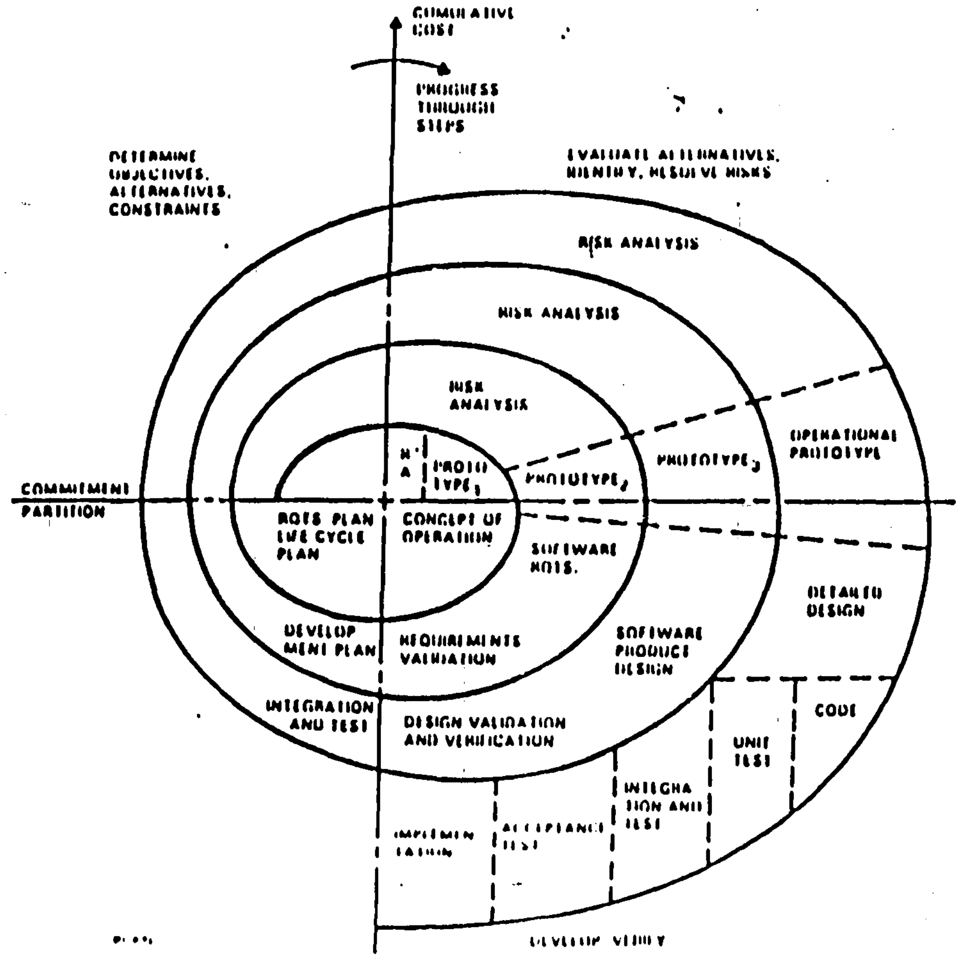
Goals

- Maintain separation of methods from tools supporting the methods
- Point of view of methods and tool users, not tool-builders
- Separate classification from evaluation
- Repository for information
- Determine "gaps" in methods and tools

Carnegie Mellon University

Software Engineering Institute

SPIRAL MODEL OF SOFTWARE PROCESS



Process Definition

- A sequence of life cycle tasks, which when properly executed produces the desired result
- An effective process must consider
  - the relationships of all the required tasks
  - the tools and methods used
  - the skills, training, motivation, and management of the people involved

Strategy

Promote the evolution of software engineering from an ad hoc, labor-intensive activity to a managed, technology-supported discipline

Implementation of Strategy

- Put process under management control
  - define
  - measure
  - optimize
- Adopt appropriate methods
- Insert technology that provides automated support for the process and methods
- Collect automated tools into an integrated environment
- Educate people

CASE

Components

- Process
- Methods
- Computers
- Tools
- Support environments
- Engineers

Currently the engineers are the essential integrating factors tying all these components together

The engineers today empower the tools
versus
the tools empowering the engineers

Issues in Software Engineering

- Quality
  - correctness
  - reliability
  - performance
- Managing the software engineering process
  - costs
  - schedules
- Productivity
  - individuals
  - groups

N91

19725

UNCLAS

PROCESSES/METHODOLOGIES

Howard Yudkin

HOW WE STAND NOW

- OK For Small Projects, Not So Good For Large Projects
- Not Good For Addressing Iterative Nature Of Requirements Resolution & Implementation (Mostly Based On Waterfall)
- Does Not Address Complexity Issues Of Requirements Stabilization (Based On Functional Decomposition)
- Does Not Explicitly Address Reuse Opportunities
- Does Not Help With People Shortages

NEED TO DEFINE AND AUTOMATE IMPROVED SOFTWARE ENGINEERING PROCESSES

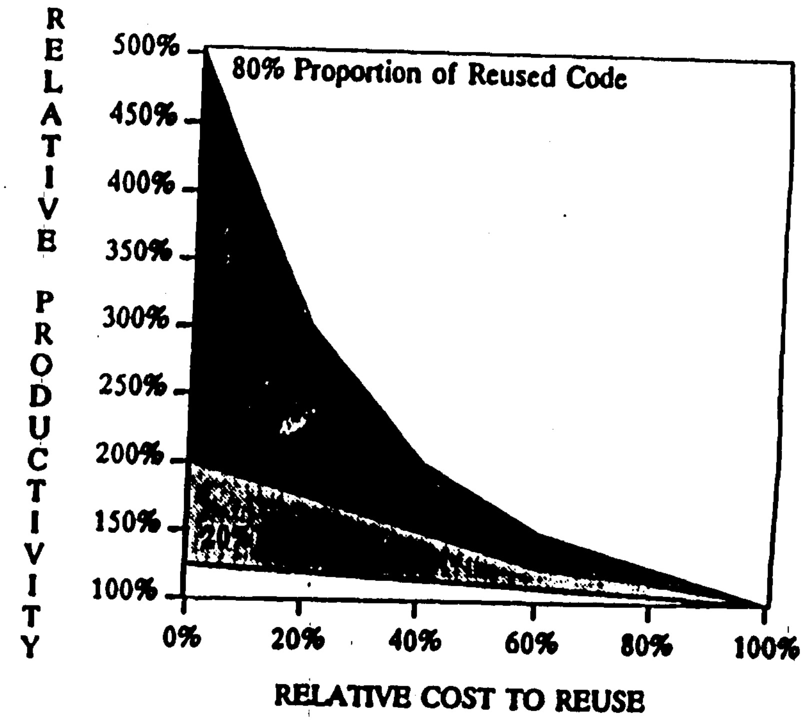
REUSE AND PROTOTYPING –TWO SIDES OF THE SAME COIN

- Reuse Library Parts Are Used To Generate Good Approximations To Desired Solutions, i.e., Prototypes
- Rapid Prototype Composition Implies Use Of Pre–existent Parts, I.E., Reusable Parts
  - Prototype Quality Depends On Fit Of The Available Parts
  - The Parts Will Often Require Some Adaptation
  - As The Set of Parts Available Becomes Richer The Prototypes Will Better Approximate Acceptable Pieces of Final Systems

REUSE PAY-OFF

- Big Gains In Productivity Will Come From Reusing Fewer Larger Parts Or Assemblies Of Smaller Parts, Not From Many Unassembled Small Parts.

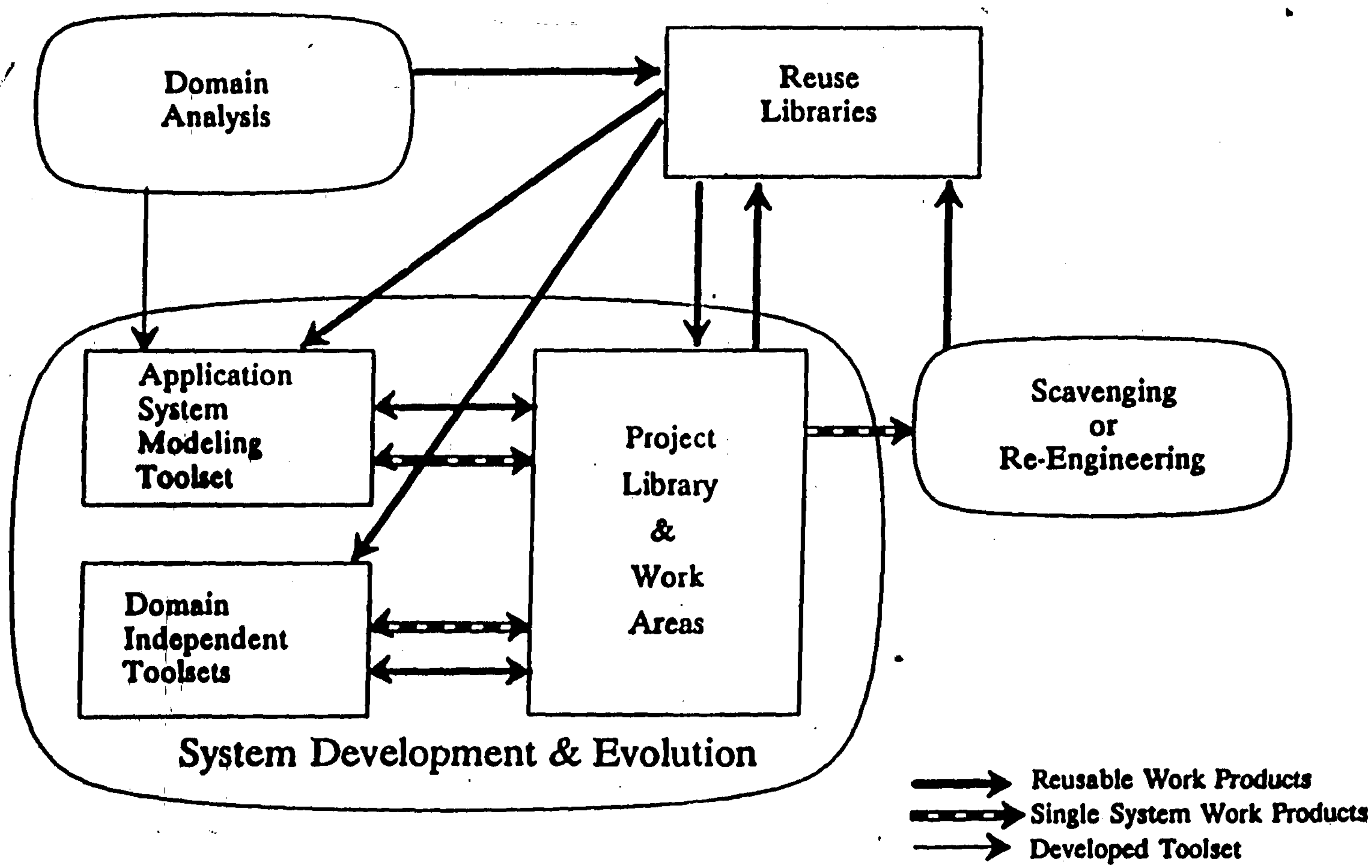
- Productivity Gain vs Cost Is Acceptable If Assemblies Of Parts Are Reused Frequently.



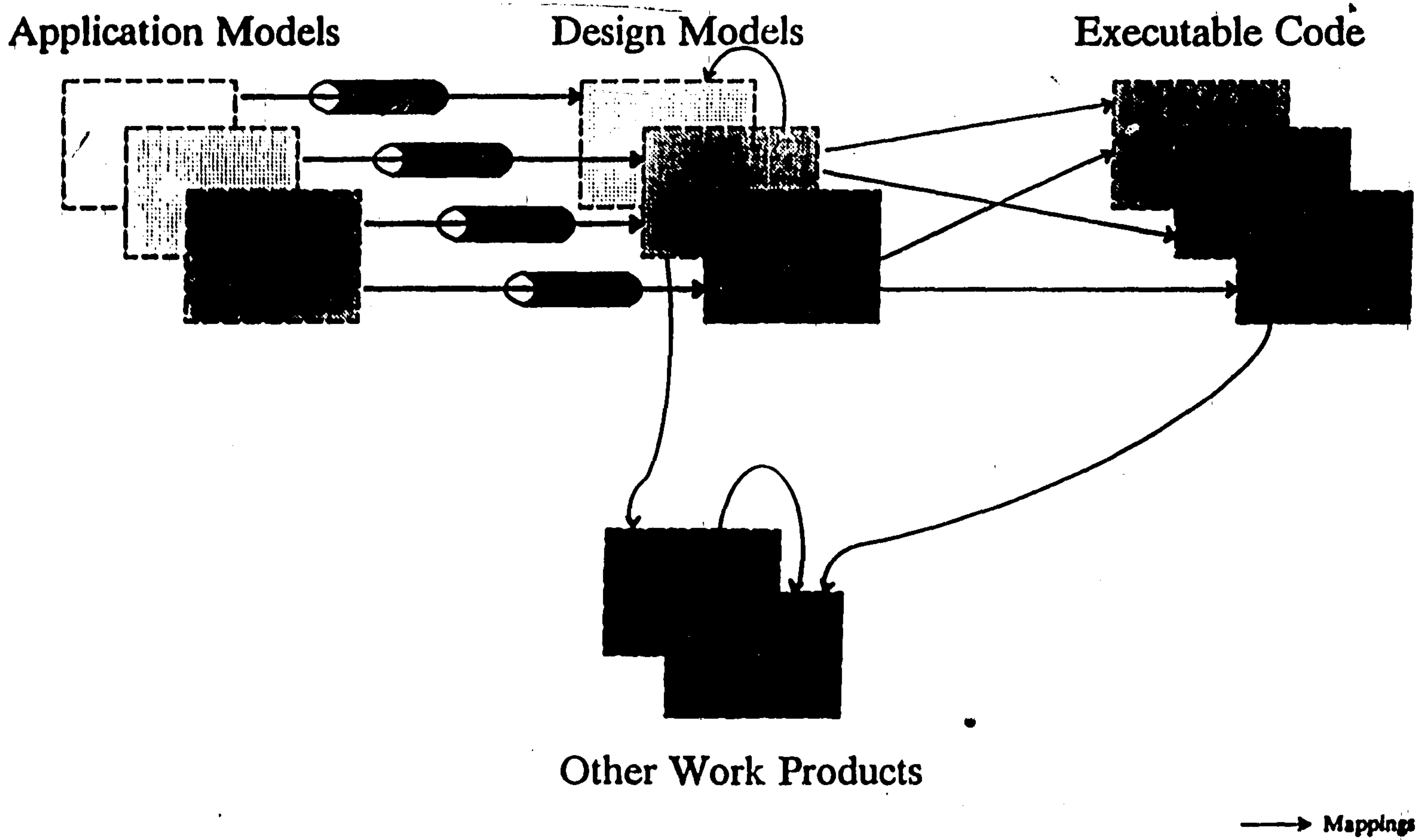
SYNTHESIS MOTIVATED BY AND ORIENTED TOWARD

- Reuse: Exploit Similarities Across Systems
- Iteration: Feedback and Enhancement
- Composition and Adaptation: Using Standard Schemes, Parts, and Designs
- Specialists: Incorporate Expertise, and Facilitating and Coordinating
- Systems View: Engineering Process
- Applying Synthesis to "Synthesizer"

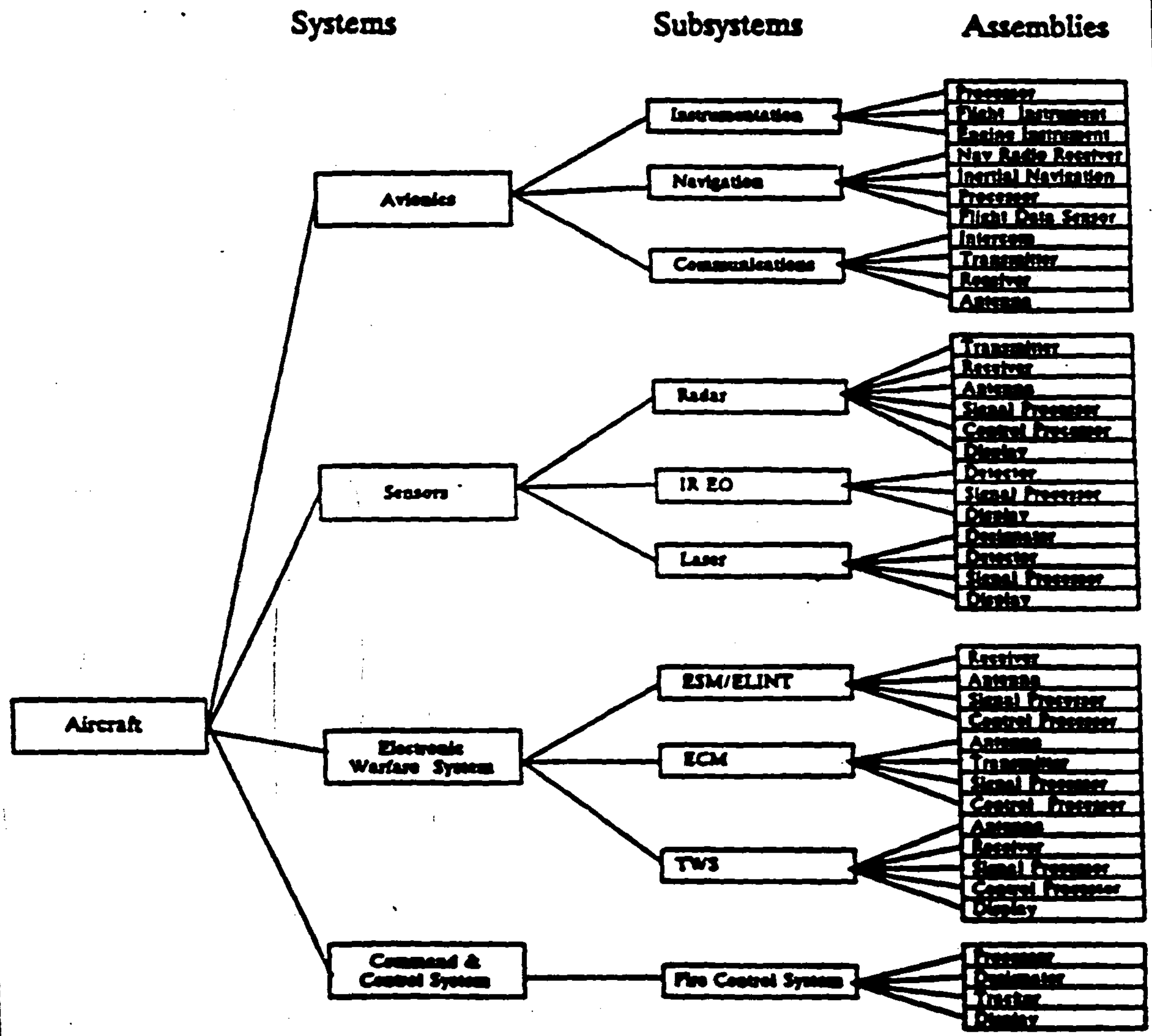
THREE MAJOR SYNTHESIS SUBPROCESSES



LIBRARY CONTENTS



TARGET APPLICATIONS FOR DOMAIN ANALYSIS – AIRPLANE EXAMPLE



SOFTWARE PRODUCTIVITY CONSORTIUM

ESSENCE OF DOMAIN ANALYSIS

- Each application area must be analyzed and characterized by standard *designs* or *architectures* that capture the way that many systems in that area could reasonably be built.

- The application engineer must be able to state his needs in application terms and have those needs mapped appropriately to an instance of the standard design.

- The design instance can be realized by specification of a set of parts from a reuse library and a set of rules for combining those parts.

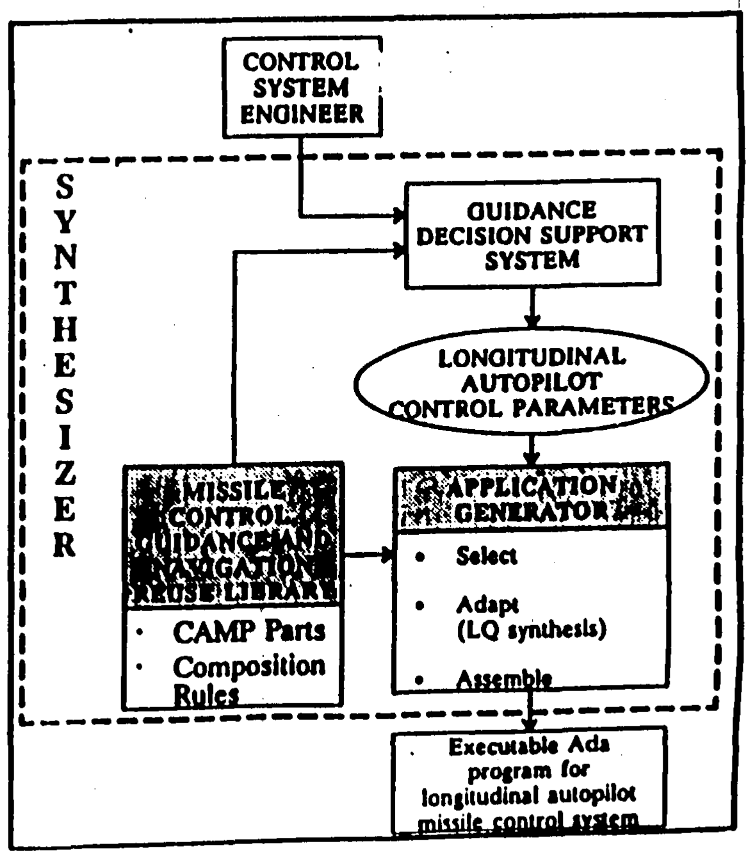
SYNTHESIS SUBPROCESS - SCAVENGING

- Many systems with software have portions amenable to adaptation for reuse.
- Scavenging these systems for reusable parts involves:
  - Extraction
  - Generalization
  - Standardization
  - Certification
  - Cataloging and storing in reuse libraries.

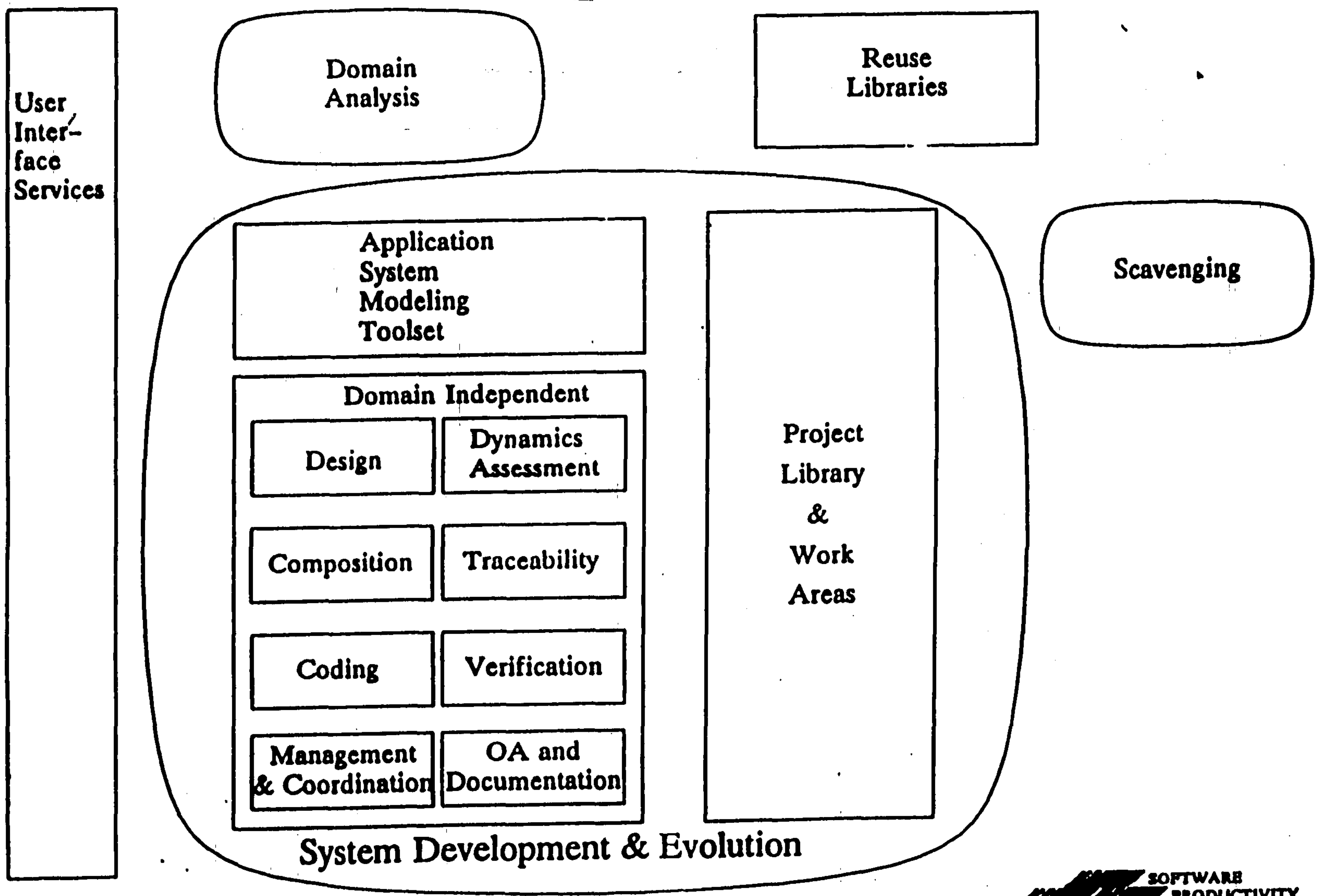
A MISSILE GUIDANCE SYNTHESIS PROTOTYPE TOOL

An example of the application of reuse, prototyping, and synthesis using a reuse library in a specific domain

- Based on U.S. Air Force "Common Ada Missile Packages" (CAMP) parts
- Initially demonstrates a longitudinal autopilot control system
- Aids understanding of the economics of reuse



PARTS OF SYNTHESIS/SYNTHESIZER

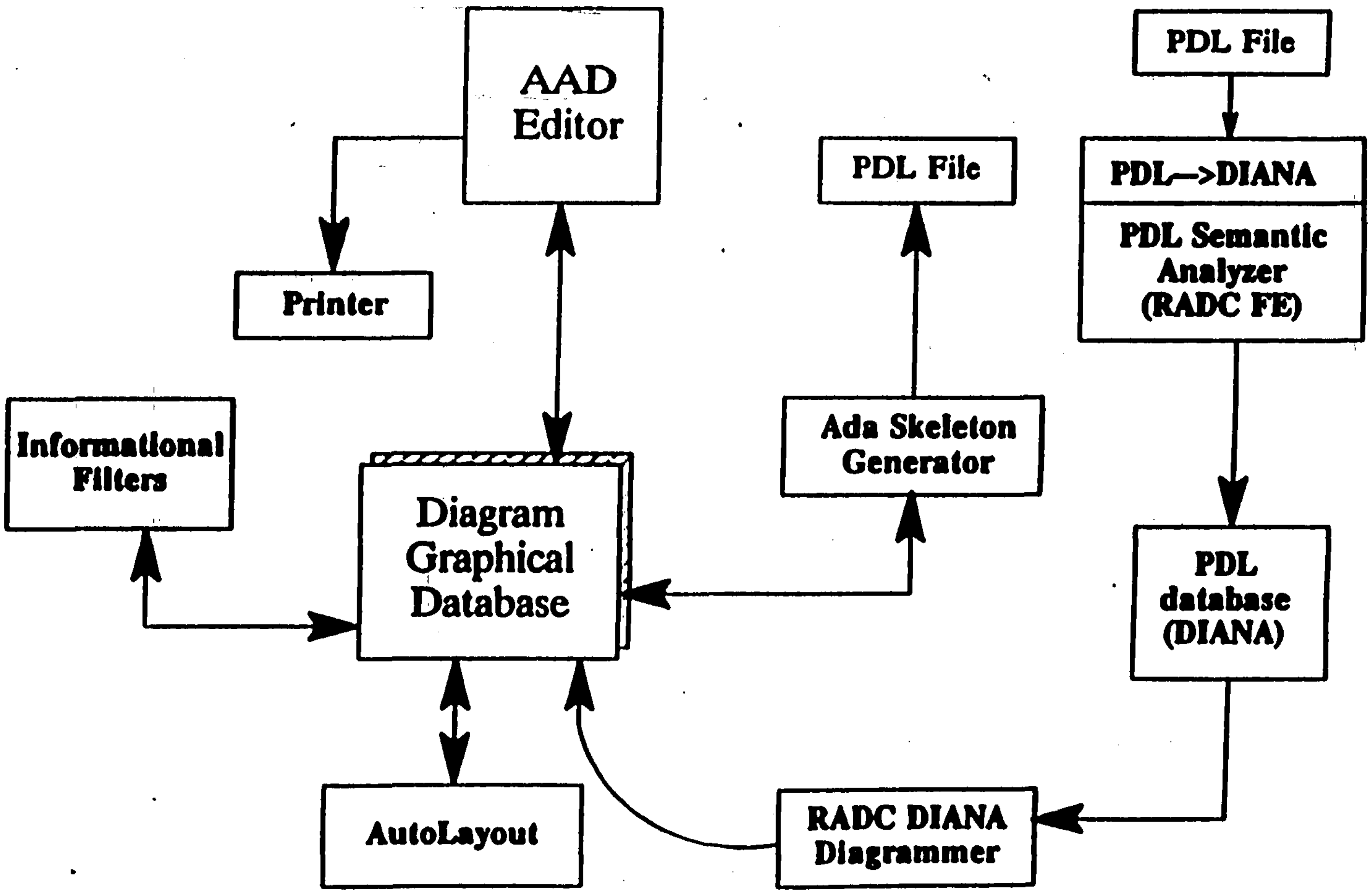


SOFTWARE PRODUCTIVITY CONSORTIUM

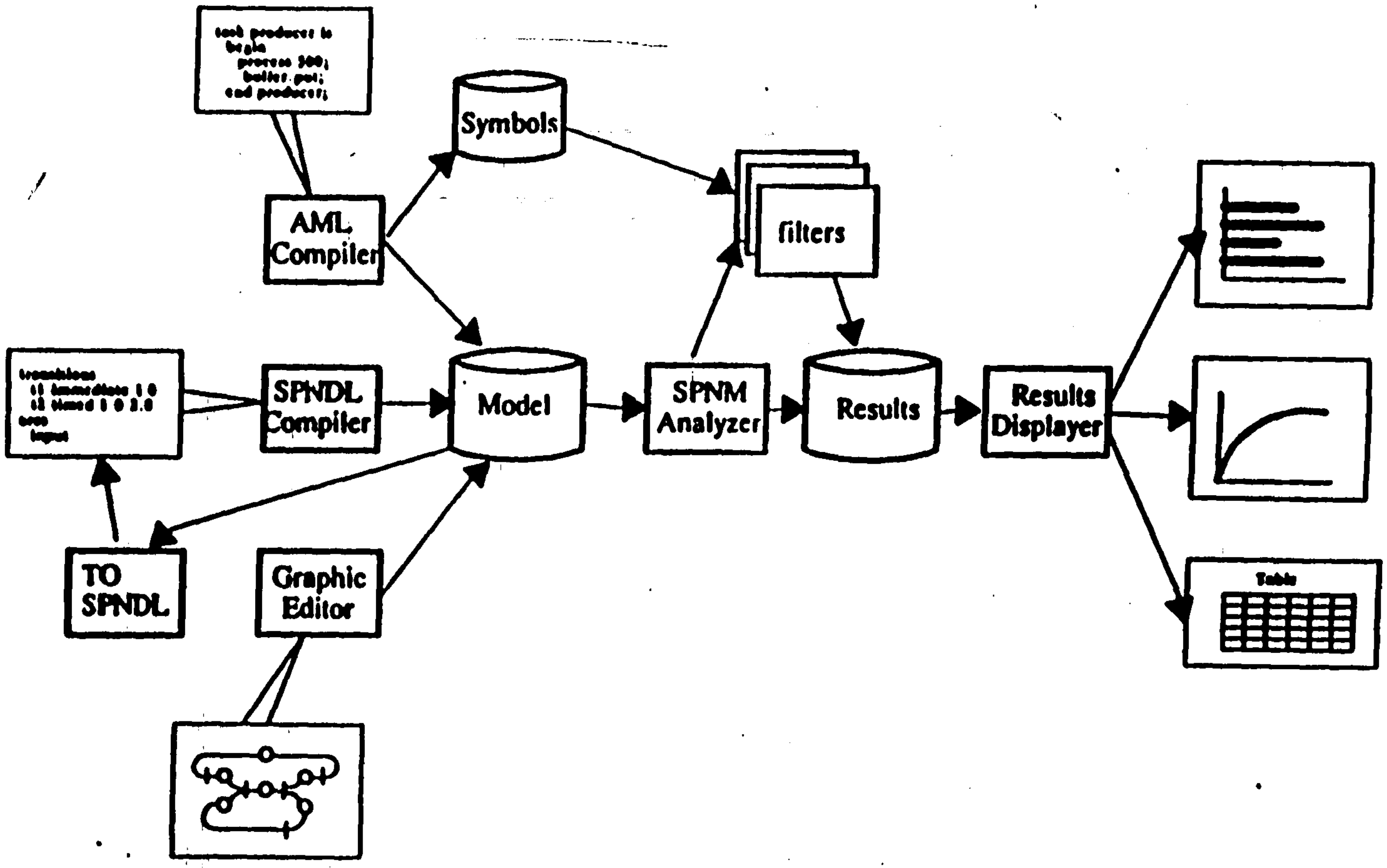
A METHODOLOGY FOR PARTS SPECIFICATION AND MODEL ASSEMBLY IS EVOLVING

- Based On NRL Software Cost Reduction Methodology
  - Information Hiding Module Families
  - Abstract Interfaces
- Accommodates Ada Packaging And Tasking Concepts
  - Tasking Guidelines Evolved (ADARTS)
- Initial Guidebooks Written And In Use

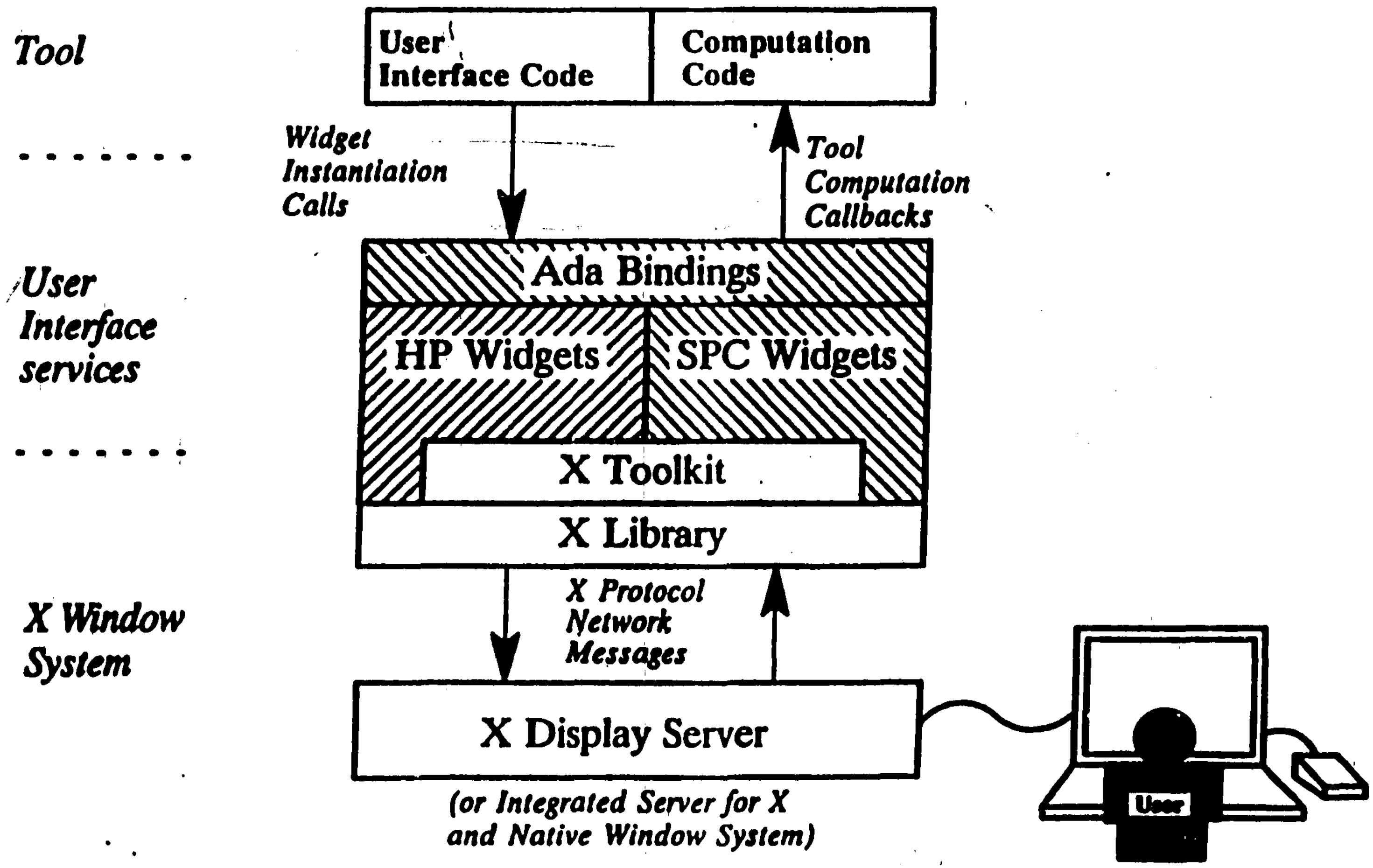
PRODUCT SET 1A STRUCTURE



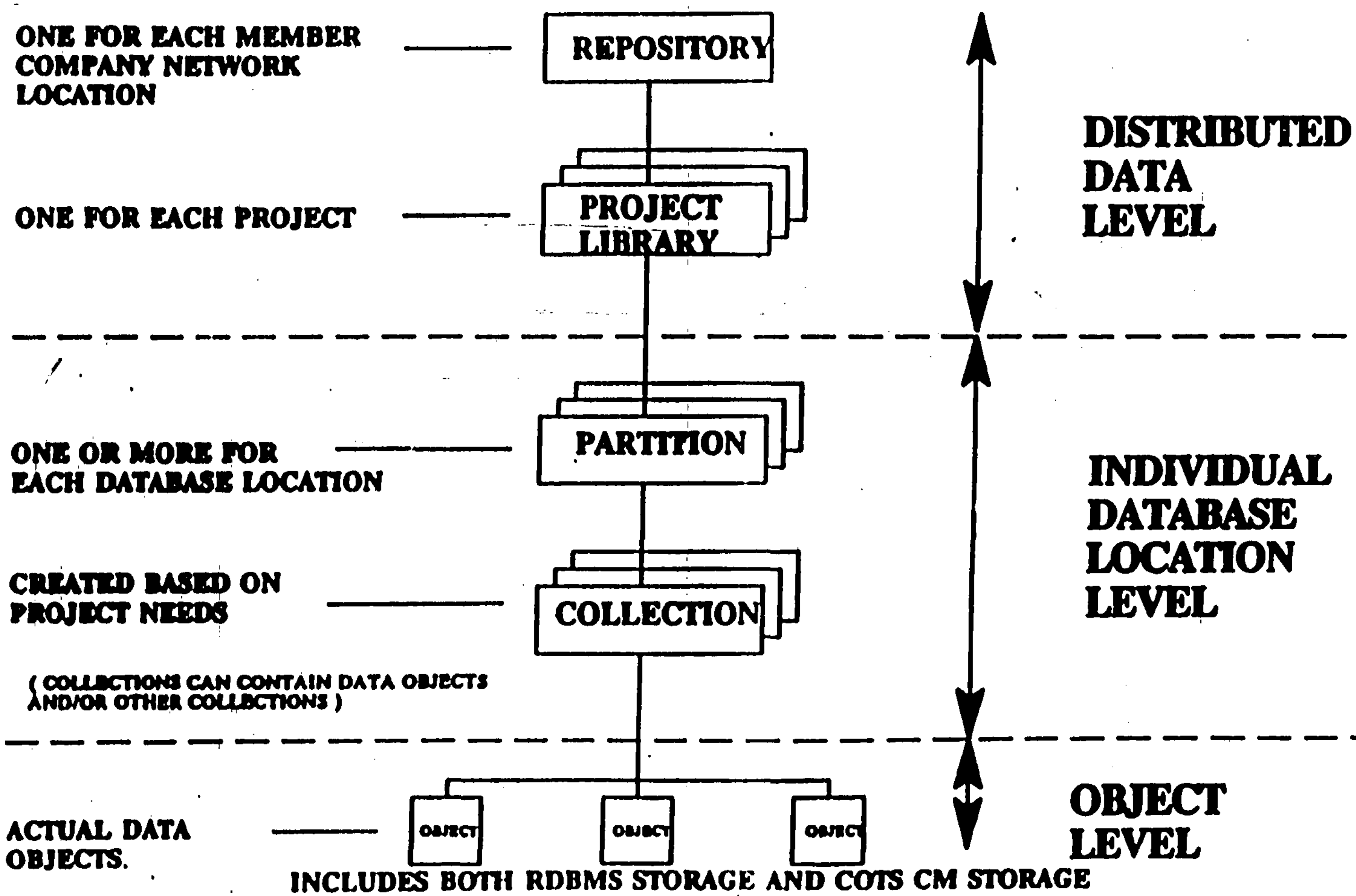
DYNAMICS ASSESSMENT TOOLSET COMPONENTS



UIS ARCHITECTURE

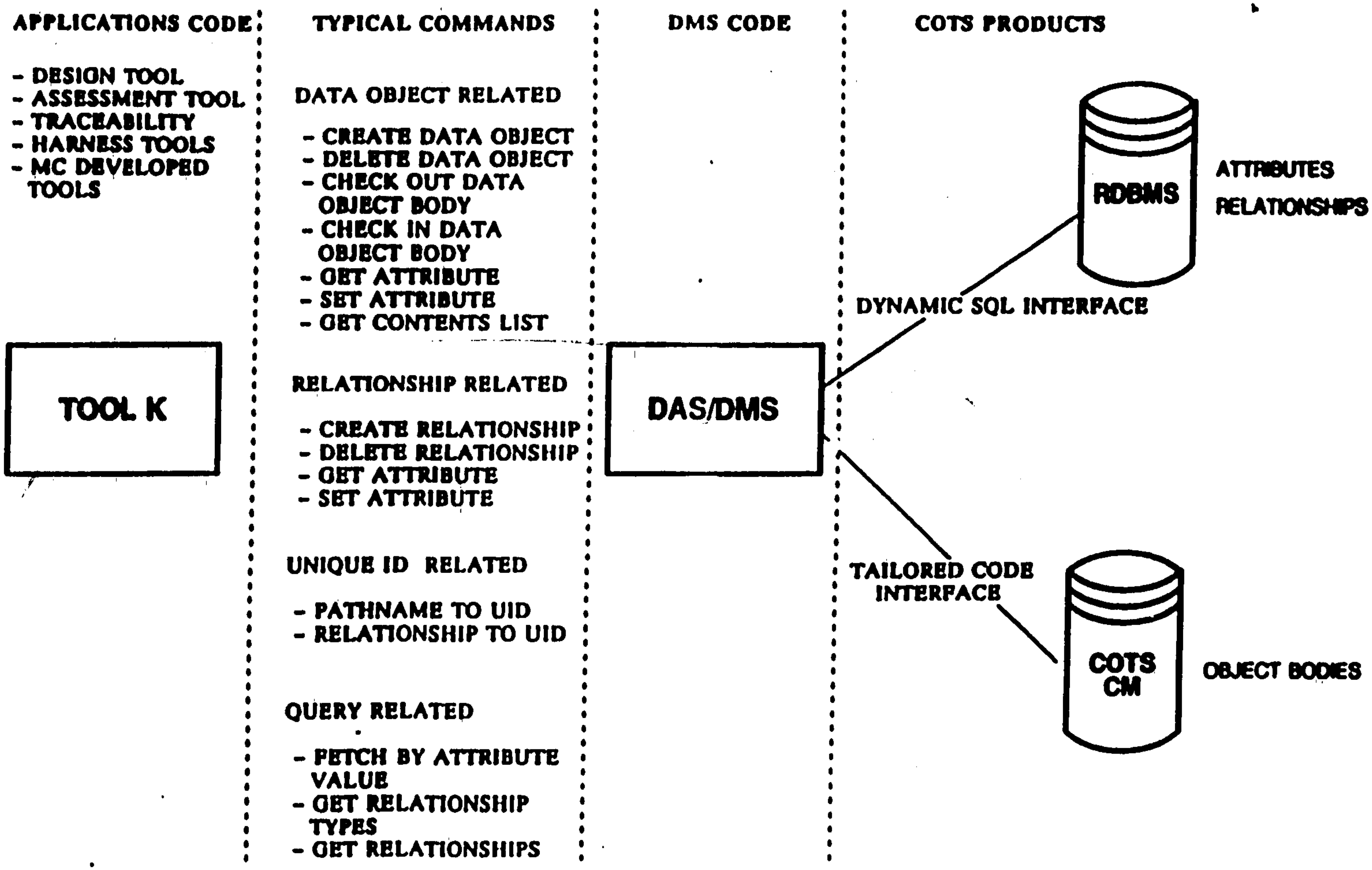


THE LAYERED REPOSITORY CONCEPT



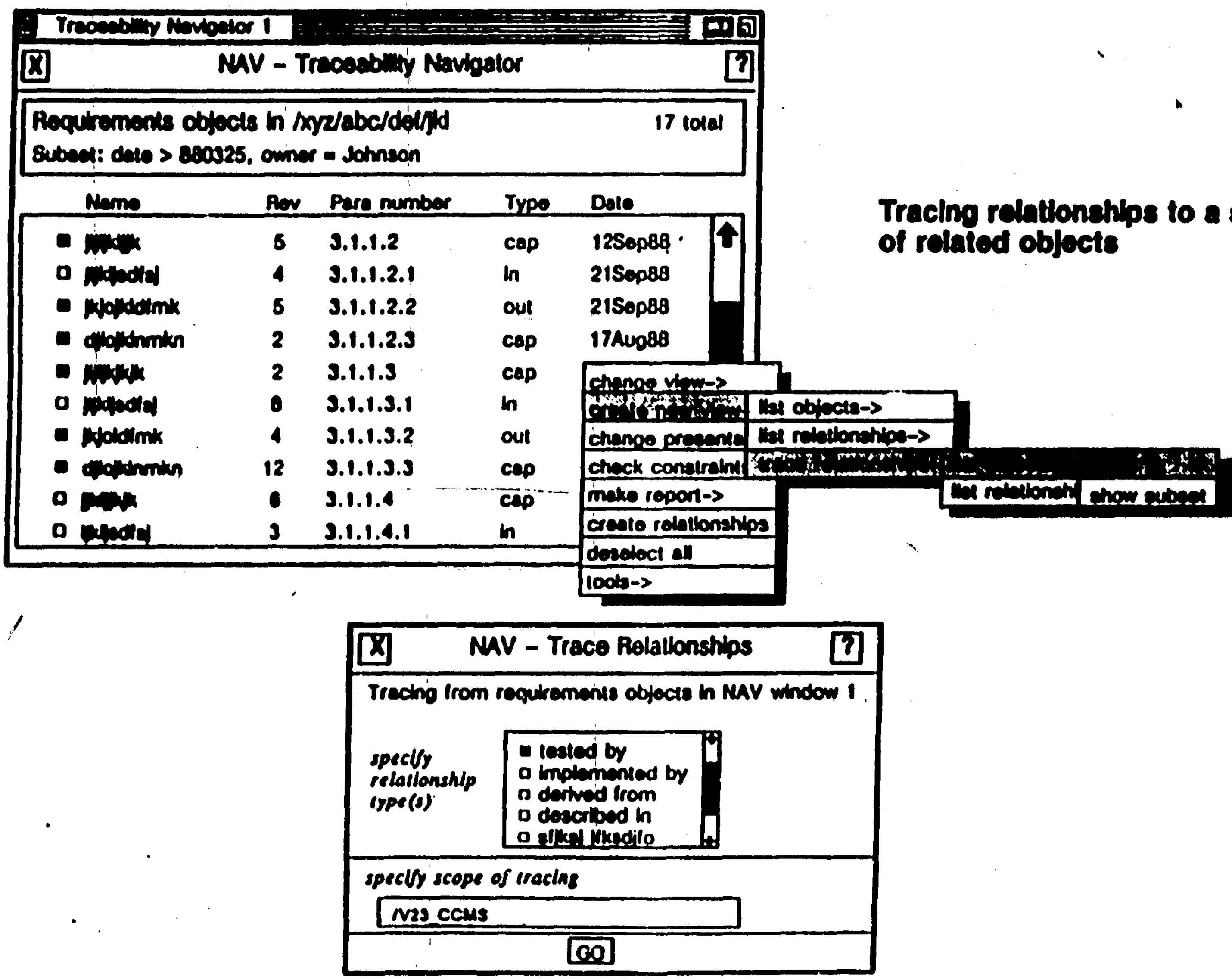
SOFTWARE PROD[illegible]TIVITY

TYPICAL PROJECT LIBRARY ACCESS



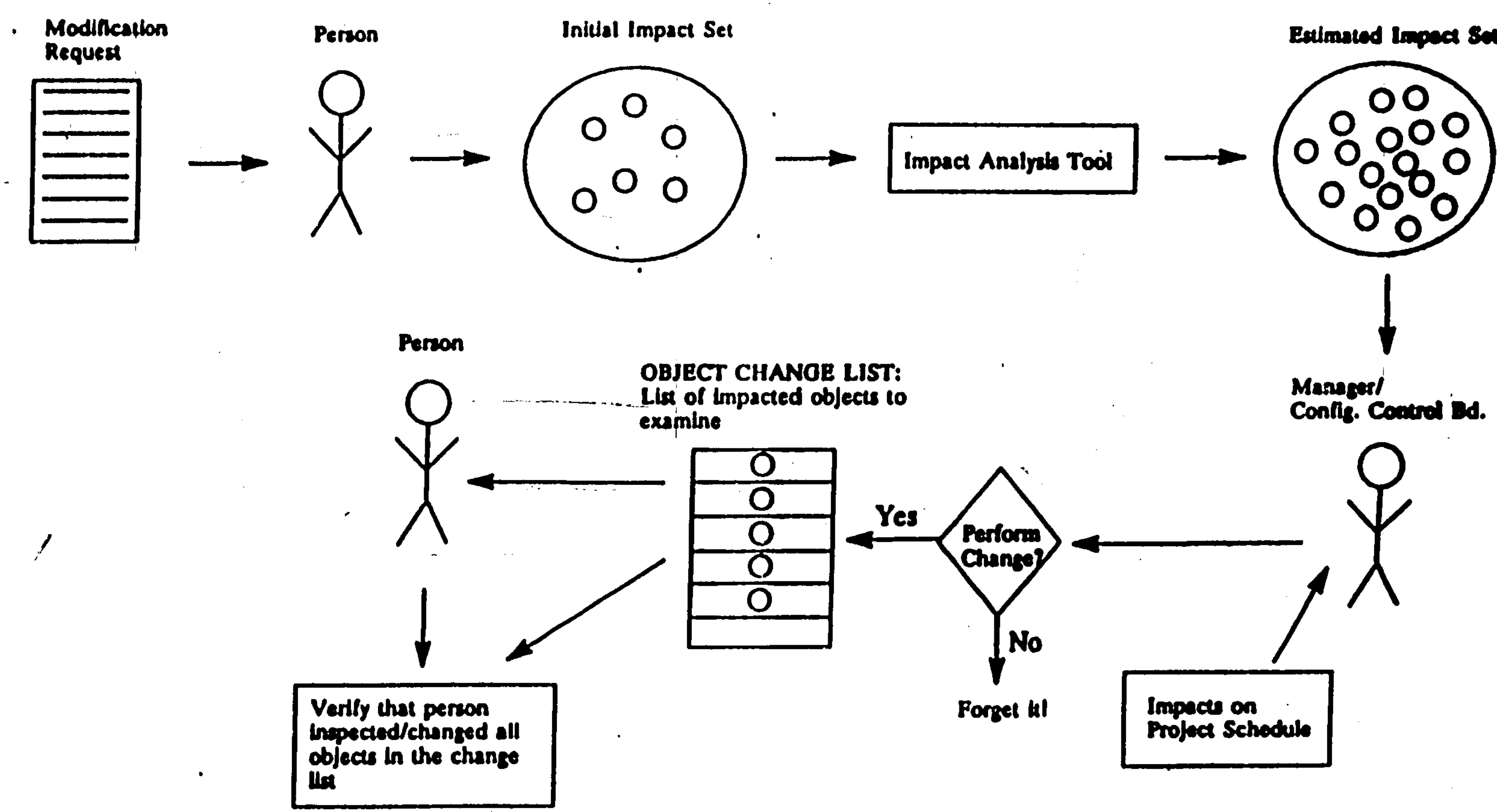
SOFTWARE PRODUCTIVITY

Tracing relationships to a set of related objects



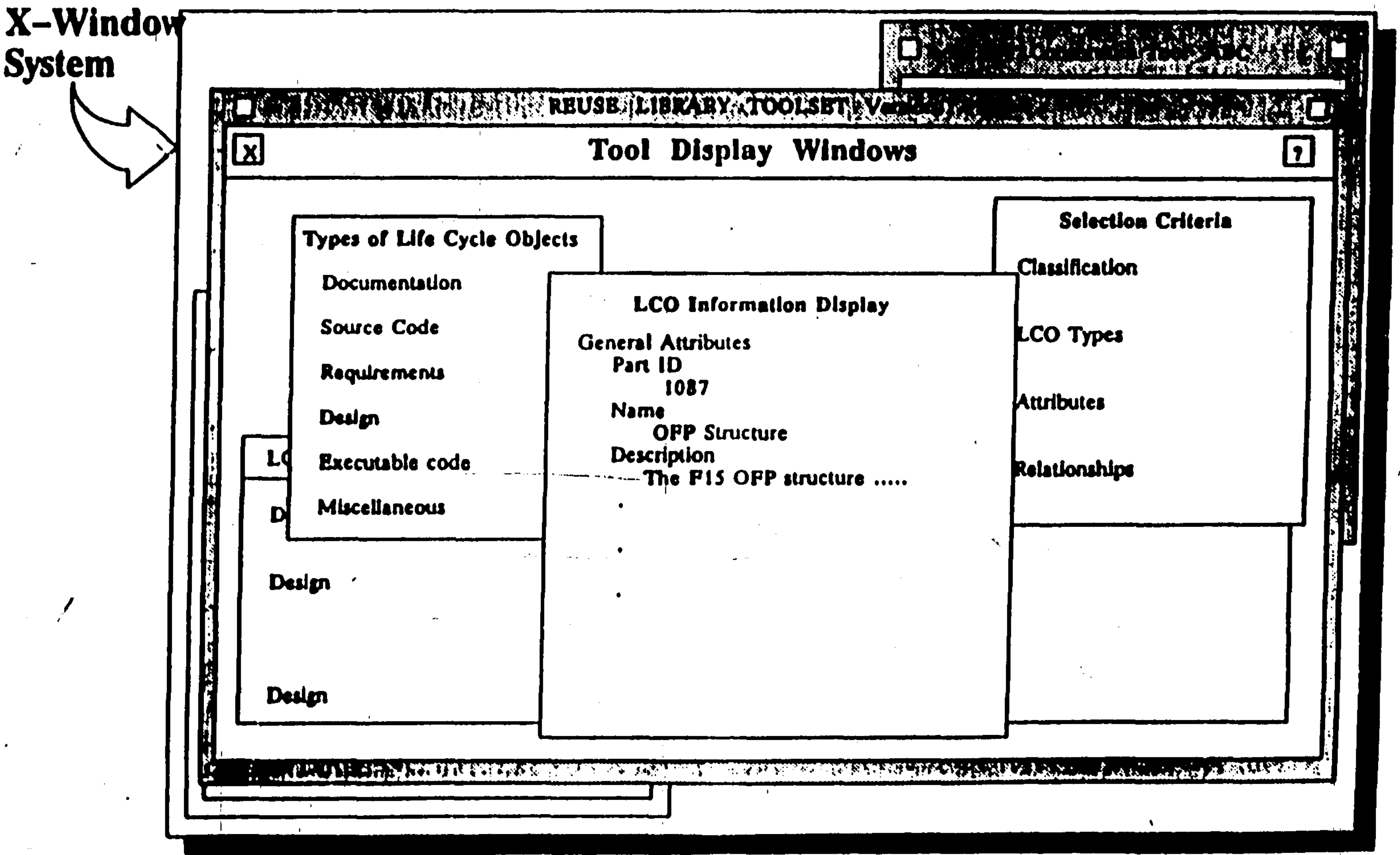
SOFTWARE PRODUCTIVITY

IMPACT ANALYSIS TOOLSET OVERVIEW



SOFTWARE PRODUCTIVITY

CANDIDATE USER INTERFACE FOR RLT



N91
19726
UNCLAS

Concurrent

N91-19726

205500

P15

A New Generation of Real-Time DOS Technology for Mission-Oriented System Integration and Operation

E. Douglas Jensen

Concurrent Computer Corporation

Westford, MA
(508) 692-6200
edj@cs.cmu.edu, uunet.uu.net!masscomp!jensen

University of Houston RICIS and NASA/Johnson Space Center
Symposium on
Integrated Computing Environments for Large, Complex Systems

November 10, 1988

Outline

- System Integration and Operation (SIO) Requirements
- New Generation Technical Approaches for SIO

System Integration and Operation (SIO) OS Requirements

- Real-time
- Distribution
- Survivability
- Adaptability

SIO Application Requirements

Real-Time

- The application, and thus the OS, activities have various types of stringent *time constraints* (e.g., hard and soft deadlines) for their completion, which are part of the correctness criteria of the activities because they are critical to mission success and the survival of human life and property

- SIO is a dynamic and stochastic environment
    - a high percentage of the activities are aperiodic with critical time constraints
    - not all periodic and aperiodic time constraints can always be met, in which case application-specified recourse must be taken

- Activities have dynamic (time- and context-dependent) relative importance (functional criticality) as well as urgency (time criticality)

- The performance of the system, and of its OS, must be optimized for high-stress exception cases, such as emergencies (e.g., due to faults, errors, and failures, or even hostile attack)

SIO Application Requirements

Distribution

- Each system consists of many subsystems containing single- and multiple-processor machines which, for technical and logistical reasons, are loosely interconnected (i.e., via i/o paths such as buses or links) —

  in some systems, the subsystems may be physically dispersed across tens or even hundreds of meters

- These interconnected machines constitute a single integrated computing system, dedicated to a particular application, executing complex distributed programs

- A multiplicity of such systems communicate application data and status among one another, and are implicitly or explicitly coordinated in their mission activities —

  the distances among systems may be hundreds of meters

- System integration and operation is automated, and under the control of a (human) hierarchical command authority

SIO Application Requirements

Survivability

- The computing system must tolerate conditions far more severe than those encountered in non-real-time contexts
  - some systems are subject to hostile attack, so their hardware faults tend to be clustered in space and time
  - different systems have a wide variety of mission periods for which there is no single robustness approach: from hours to decades
  - limited or no repairs may be possible during the mission
  - the system usually has to remain in non-stop service during recovery from faults
  - extreme safety concerns: system failure may jeopardize the mission, human life, and property
- Because the hardware and software are distributed, there are multiple independent fault modes
- Overloads, faults, and resource contention are inevitably dynamic and stochastic
- Optimal performance under exceptional stress is the *raison d'etre* of the system

SIO Application Requirements

Adaptability

- Application limitations often demand maximum computing capability for the allowable size, weight, and power, which argues for special-purpose hardware and OS;

  but there is not just one set of fixed computing requirements

- There are many widely divergent real-time SIO applications, and the high costs of developing their computing systems argues for generality, standardization, and re-usability of the hardware and OS

- The computing requirements for any particular application evolve continuously over the entire lifetime of the system because
  - the application is extremely complex and difficult to understand
  - the application environment varies with time
  - technology advances rapidly

  and the application system lifetime can be decades

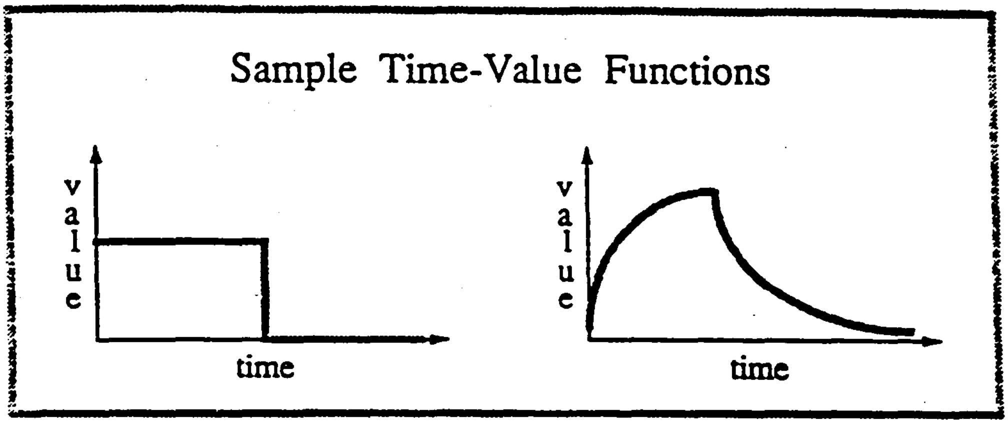
New Generation Technical Approaches for System Integration and Operation

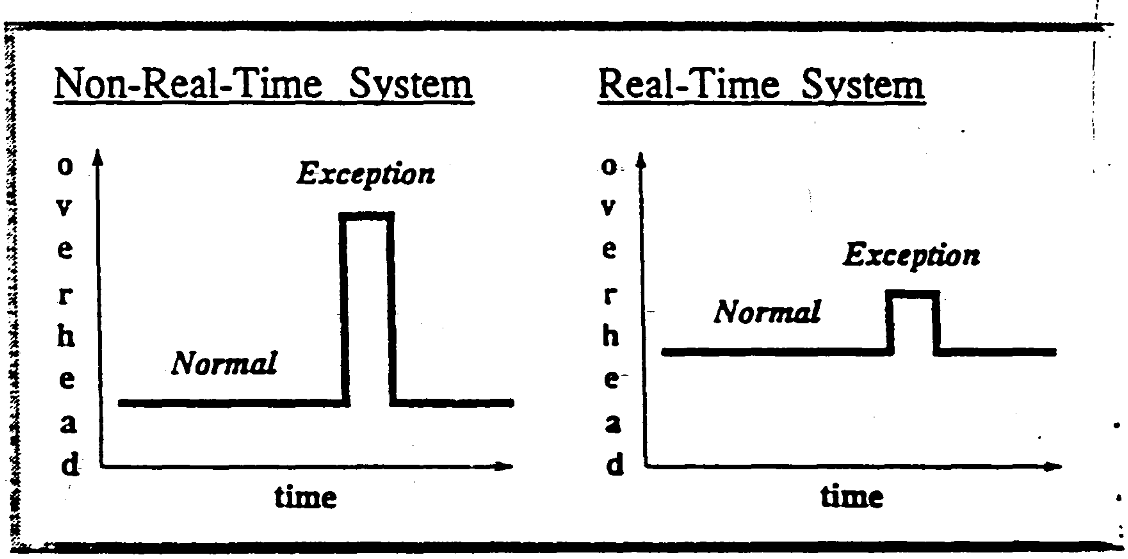
- Real-Time
- Distribution
- Survivability
- Adaptability

New Generation Technical Approaches for System Integration and Operation

Real-Time

- Manage all physical and logical resources directly with actual application-specified time constraints as expressed by *time-value functions* for all activities —
    - manages periodic and aperiodic activities in an integrated, uniform manner
    - distinguishes between urgency and importance
    - allows not only hard deadlines but also a wide variety of soft (i.e., residual value) time constraints
    - accommodates dynamic variability and evolution of both periodic and aperiodic time constraints
    - provides behavior which is as deterministic as desired and affordable
    - handles overloads gracefully according to application-specified policies
    - supports the clean-up of computations which fail to satisfy their time constraints, to avoid wasting resources and executing improperly timed actions
    - employs the same block-structured, nested, atomic commit/abort mechanisms as for transactions
- Optimize performance for exception cases


Sample Time-Value Functions
value
time
value
time


Non-Real-Time System
overhead
Exception
Normal
time
Real-Time System
overhead
Exception
Normal
time

New Generation Technical Approaches for System Integration and Operation

Distribution

- Provide a new programming model which is well-suited for writing large-scale, complex real-time distributed software:

  objects (passive abstract data types — code plus data), in which there may be any number of concurrent control points; and *threads* (loci of control point execution) which move among objects via *operation invocation*

- A thread is a distributed computation which transparently (and reliably) spans physical nodes, carrying its local state and attributes for timeliness, robustness, etc.;

  these attributes are used at each node to perform resource management on a system-wide basis in the best interests (i.e., to meet the time constraints) of the whole distributed application

- Distributed computations must explicitly maintain consistency of data and correctness of actions, despite asynchronous real concurrency (and multiple independent hardware faults) — to accomplish this requires (at the kernel level, because the OS must itself be distributed)
  - real-time transaction mechanisms for atomicity, application-specific concurrency control, and permanence
  - system- and user-supplied commit and abort handlers

New Generation Technical Approaches for System Integration and Operation

Survivability

- The survivability properties and approaches include
    - graceful degradation: *best-effort* resource management policies; dynamic reconfiguration of objects
    - fault containment: data encapsulation (objects); object instances in private address spaces; capabilities
    - consistency of data, correctness of actions: concurrency control objects; resource tracking; thread maintenance; abort blocks; real-time transaction mechanisms (atomicity, concurrency control, permanence)
    - high availability of services and data: object replication; dynamic reconfiguration of objects
- The survivability features are presented through the programming model as a set of mechanisms which can be selected and combined as desired — their cost is proportional to their power
- Transactions
    - are scheduled according to the same real-time policies as are all other resources
    - allow application-specific commit and abort handlers

New Generation Technical Approaches for System Integration and Operation

Adaptability

- Adhere to the philosophy of *policy/mechanism separation*:
  - have a kernel of primitive mechanisms from which everything else is constructed according to a wide possible range of application-specific policies to meet particular functionality, performance, and cost objectives
  - provide these mechanisms at the optimal level of functionality — i.e., both necessary and sufficient to create large scale, complex real-time distributed systems
- Encourage application-specific information to be exploited statically and dynamically — e.g.,
  - special-purpose objects can be migrated into the kernel
  - references to objects can be monitored for locality
  - any attributes can be carried along with threads
  - special hardware augmentations can be objects
  - concurrency control and abort handlers can be special
  - resource management policies are application-defined
- Employ elastic resource management which flexes to tolerate variability in loading, timing, etc.

Alpha Program Management Overview

- Alpha originated at CMU-CSD as part of the Archons Project on real-time distributed computer architectures and operating systems—Doug Jensen was the Principal Investigator

- As part of a long-continuing "Think—Do" cycle, new concepts and techniques were created, based on the PI's 15 years of industrial R&D experience with real-time computer systems,

  then many of these were embodied in a feasibility test vehicle: the Alpha real-time decentralized OS

- The Alpha prototype ("Release 1")
  - lead by Duane Northcutt, with a team of five programmers for about three years
  - written for (homebrew) multiprocessor Sun workstations connected via Ethernet
  - consists of a high-functionality kernel, some system-layer functions, some software development tools
  - installed at General Dynamics/Ft. Worth in 1987 and demonstrated to many DoD agencies with a real-time C^2 application
  - numerous technical reports now becoming available

Alpha Program Management Overview

(continued)

- Alpha Release 2
  - intended to make the technology externally accessible, on reproduceable hardware platform, and further develop it
  - kernel interface spec subcontracted by CMU to Kendall Square Research, which Jensen later joined

    substantial functional enhancements were included
  - initial detailed design subcontracted to Concurrent when Jensen moved there
  - continuing research and remainder of design and implementation is part of a pending procurement
  - Jensen's Ph.D. students continuing research at CMU
  - pre-release available mid-CY89, release at end of CY89
  - portable, open, multi-vendor hosted
- Release 3
  - significant enhancements over Release 2
  - release at end of CY90

N91
19727
UNCLAS

MISSION OPERATIONS DIRECTORATE
FACILITY AND SUPPORT SYSTEMS DIVISION

REQUIREMENTS ANALYSIS FUNDAMENTALS
NOVEMBER 9, 1988

MICHAEL J. SEE

INTRODUCTION

ADVANCED PROJECTS SECTION

- ELEMENT WITHIN MISSION OPERATIONS DIRECTORATE

- RESPONSIBILITIES

  - DEVELOP/COORDINATE USER REQUIREMENTS FOR GROUND INFORMATION SYSTEMS SOFTWARE (E.G., MISSION CONTROL CENTER UPGRADE) AND TRANSMIT TO DEVELOPER.

  - REPRESENT OPERATIONS COMMUNITY (USERS) TO DEVELOPER.

  - REPRESENT DEVELOPER TO USERS.

  - DEVELOP/PROTOTYPE USER APPLICATIONS.

  - PROVIDE CONFIGURATION MANAGEMENT OVERSIGHT FOR USER APPLICATIONS.

PROBLEM

- SOFTWARE PRODUCTS OF THE CURRENT DEVELOPMENT PROCESS OFTEN DO NOT FULLY MEET "TRUE" USER NEEDS UPON DELIVERY.
  - DELIVERY OF NEEDED CAPABILITIES IS DELAYED.
  - COST OF CORRECTING SYSTEMS AFTER DELIVERY IS HIGH.

- PROBLEM IS ROOTED IN REQUIREMENTS DEFINITION AND ANALYSIS PROCESS.

CAUSES

- REQUIREMENTS DEFINITION FOR CONTEMPORARY INFORMATION SYSTEMS IS INHERENTLY DIFFICULT.
  - HIGH HUMAN/COMPUTER INTERACTION
  - APPLICATIONS DEVELOPED BY USER COMPLICATES APPLICATION INTERFACE REQUIREMENTS DEVELOPMENT
- REQUIREMENTS CHANGE RAPIDLY.
  - USER POPULATION IS DYNAMIC.
  - USER APPLICATIONS ARE CONSTANTLY EVOLVING.
  - NEW PROGRAMS (E.G., SPACE STATION) INTRODUCE NEW OPERATIONS CONCEPTS.
  - NEW TECHNOLOGY IS CONSTANTLY EMERGING.
  - EXPERIENCE WITH CURRENT SYSTEM UNCOVERS NEW REQUIREMENTS.

CAUSES (CONTINUED)

- REQUIREMENTS ARE OFTEN INCOMPLETE/CONFLICTING DUE TO DIVERSITY OF USER COMMUNITY.
  - TASKS
  - FLIGHT SYSTEMS (E.G., DISCRETE VS. ANALOG, TELEMETRY VS. TRAJECTORY)
  - USER EXPERIENCE LEVEL
- REQUIREMENTS ARE EASILY MISINTERPRETED [illegible] ELOPER.
  - USERS ORGANIZATIONALLY SEPARATED FROM DEVELOPERS.
  - WRITTEN DESCRIPTIONS OF VISUAL SYSTEMS IS INADEQUATE.
- THESE CONDITIONS ARE NOT UNIQUE TO NASA MISSION OPERATIONS.

INTRODUCTION TO SESSION 1

REQUIREMENTS ANALYSIS FUNDAMENTALS

- "REQUIREMENTS ANALYSIS, DOMAIN KNOWLEDGE, AND DESIGN," COLIN POTTS/MCC SOFTWARE TECHNOLOGY PROGRAM

  SUGGESTS INNOVATIVE METHODOLOGY TO:

  - ACCOMMODATE CHANGING/CONFLICTING REQUIREMENTS.

  - SYSTEMATIZE TRANSLATION OF REQUIREMENTS INTO DESIGN, REDUCING MISINTERPRETATION.

  - IMPROVE REQUIREMENTS COMPLETENESS.

  - ENHANCE TRACEABILITY.

INTRODUCTION TO SESSION 1 (CONTINUED)

- "KNOWLEDGE-BASED REQUIREMENTS ANALYSIS FOR AUTOMATING SOFTWARE DEVELOPMENT," LAWRENCE MARKOSIAN/REASONING SYSTEMS, INC.

  PROPOSES NEW SOFTWARE DEVELOPMENT PARADIGM THAT:

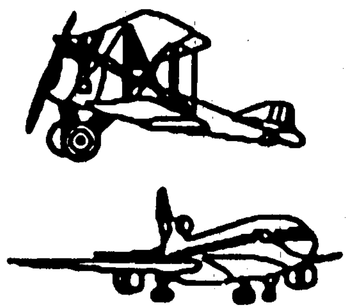
  - AUTOMATES DERIVATION OF IMPLEMENTATIONS FROM REQUIREMENTS, REDUCING MISINTERPRETATION.
  - INCREASES DEVELOPMENT PRODUCTIVITY.
  - VALIDATES FORMALIZED REQUIREMENTS.
  - ENHANCES TRACEABILITY.

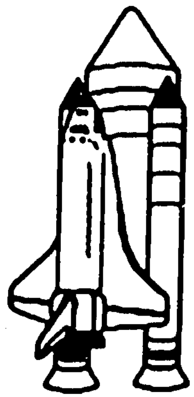
SOFTWARE ENGINEERING AS AN ENGINEERING DISCIPLINE

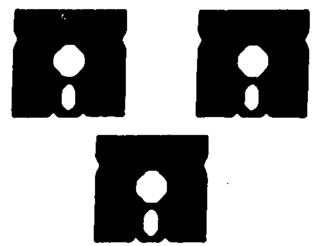
ROBERT B. MacDONALD
MISSION SUPPORT DIRECTORATE
NASA/JOHNSON SPACE CENTER

T1110

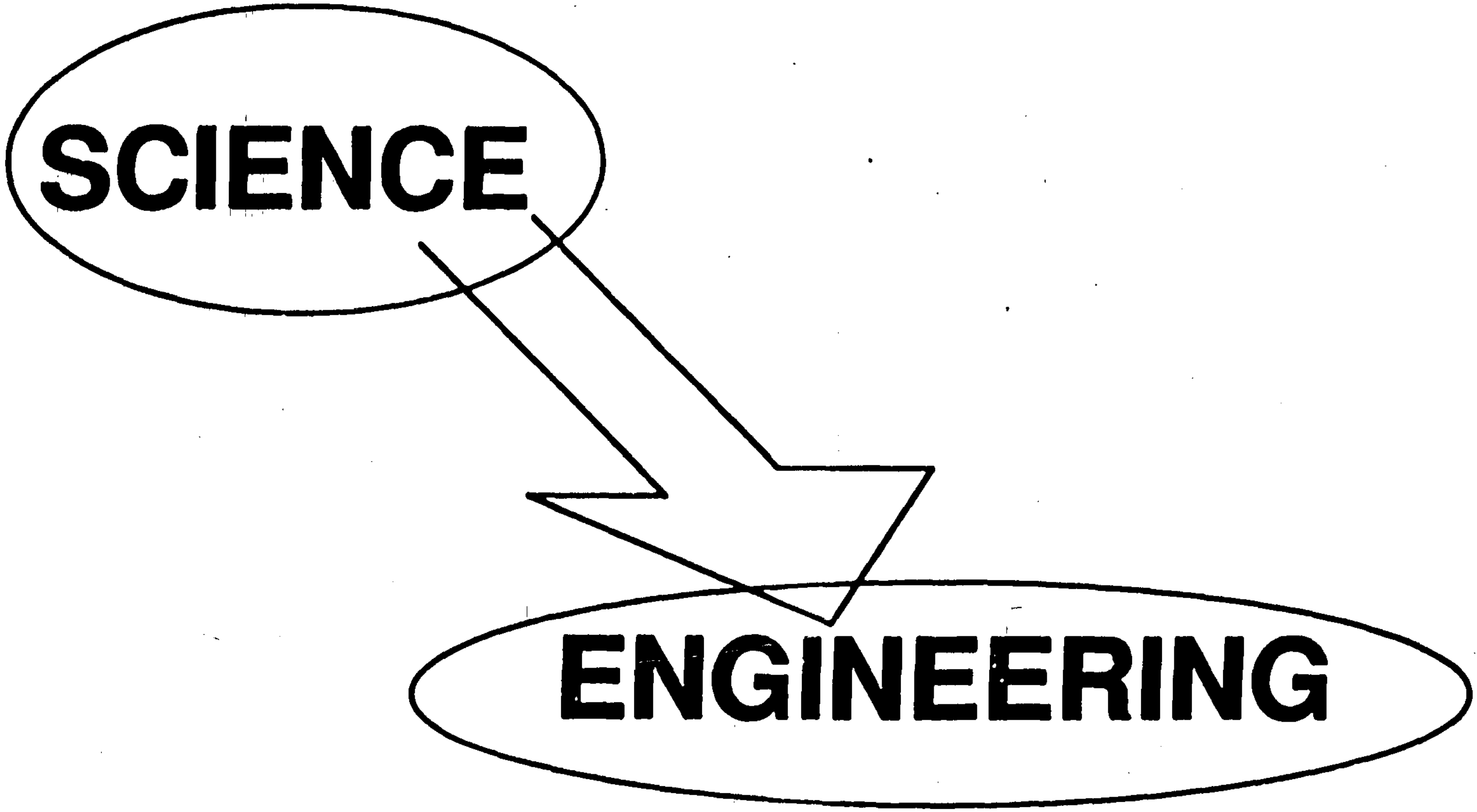
BACKGROUND







SCIENCE TO ENGINEERING

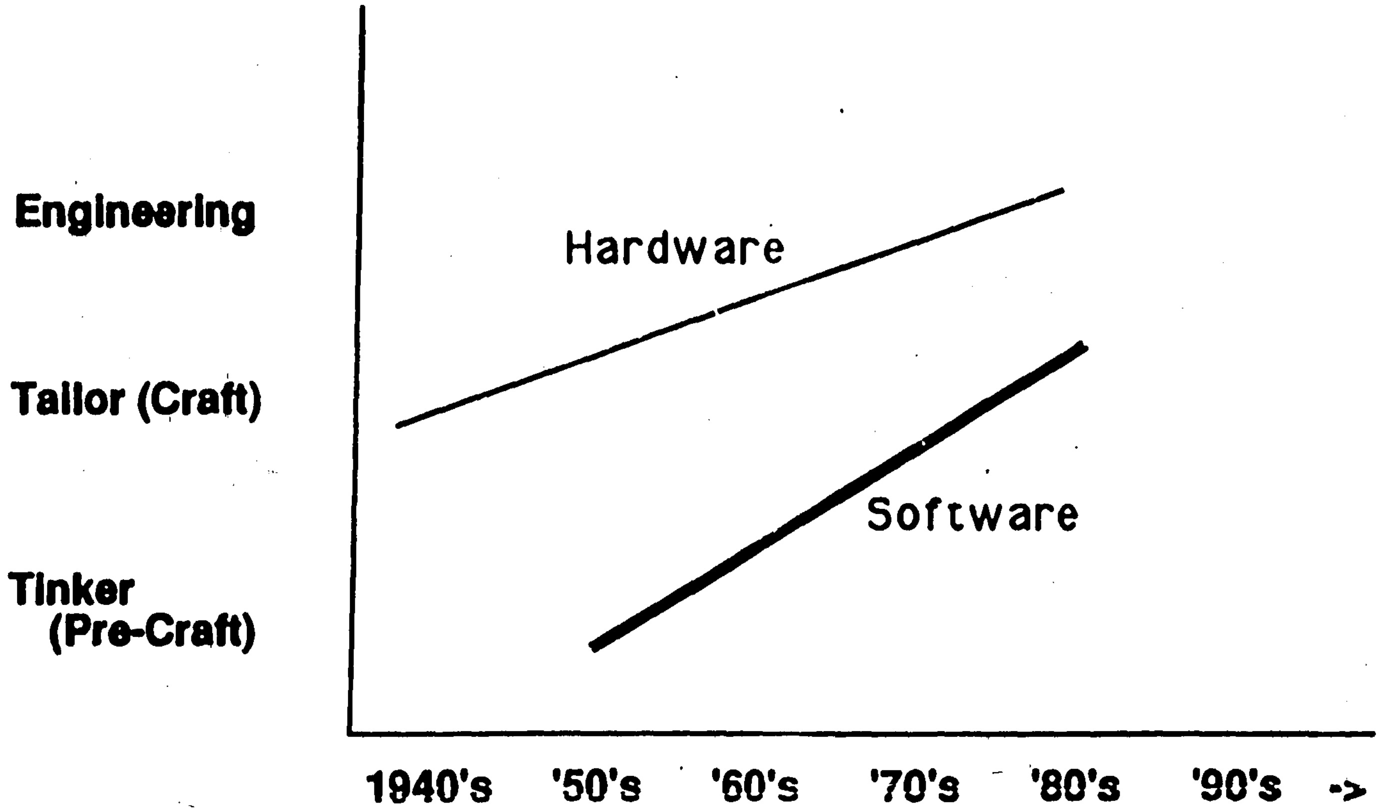


DEFINITIONS OF ENGINEERING

"...application of mathematical and scientific 'bodies of knowledge' as captured by predictive models, laws, etc. to the problem of designing and constructing economical and reliable systems which fulfill some real need."

"... establishment and use of sound engineering principles to obtain economical software that is reliable and works efficiently on real machines"

DEVELOPMENT OF COMPUTING: A PERSONAL MODEL



SOFTWARE ENGINEERING

- ◊ UNIVERSI PROGRAMS
- ◊ INDUSTRIAL PROGRAMS

CHALLENGES

HOW TO GET 4-5 YEARS OF CONTENT PACKED INTO A CURRICULUM

VERIFYING THE DESIGN

DEVELOP MODELS OF A MATURE DISCIPLINE, E.G. MAXWELL'S EQUATION OR NEWTON'S LAWS

N91

19728

UNCLAS

Empirical Studies of Software Design: Implications for SEEs

Herb Krasner
Manager, Software Process Research
Lockheed Software Technology Center
Austin, Texas

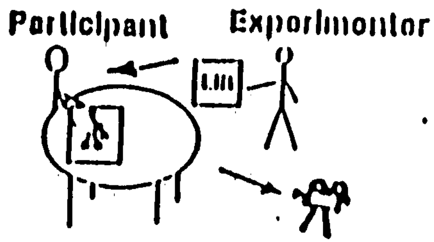
Lockheed
Missiles & Space Company, Inc.
Software Technology Center #11758

Empirical Research on the Software Process

Individual level

LIFT experiment

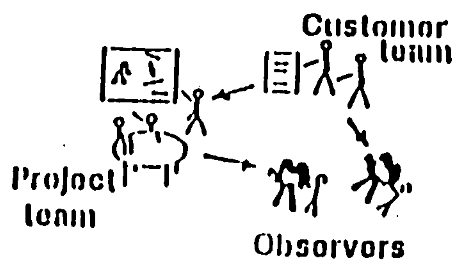
8 experienced programmers designing the control structure for a set of elevators during an intense 2 hr. session



Team level

Object server exp.

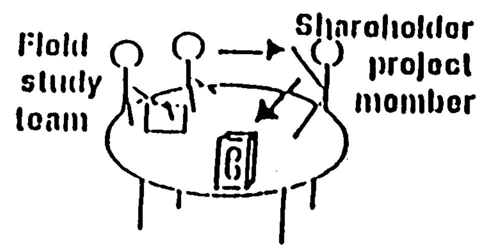
Videotaped team meetings from a 7 mo. effort to design and build a tool to support object oriented programming



Project level

Field study

Detailed interviews with key members of 18 large development projects to model their decision-making and communication process



Results of the Field Study

- Observations about commonality/difference of projects
- Identification of five areas of organizational breakdown (within that sixteen specific problems)
- Implications for process modeling
- Mapping of problems onto lower-level phenomena

"You need to understand, this project isn't the way we develop software at our company."

Characteristics of Projects Studied

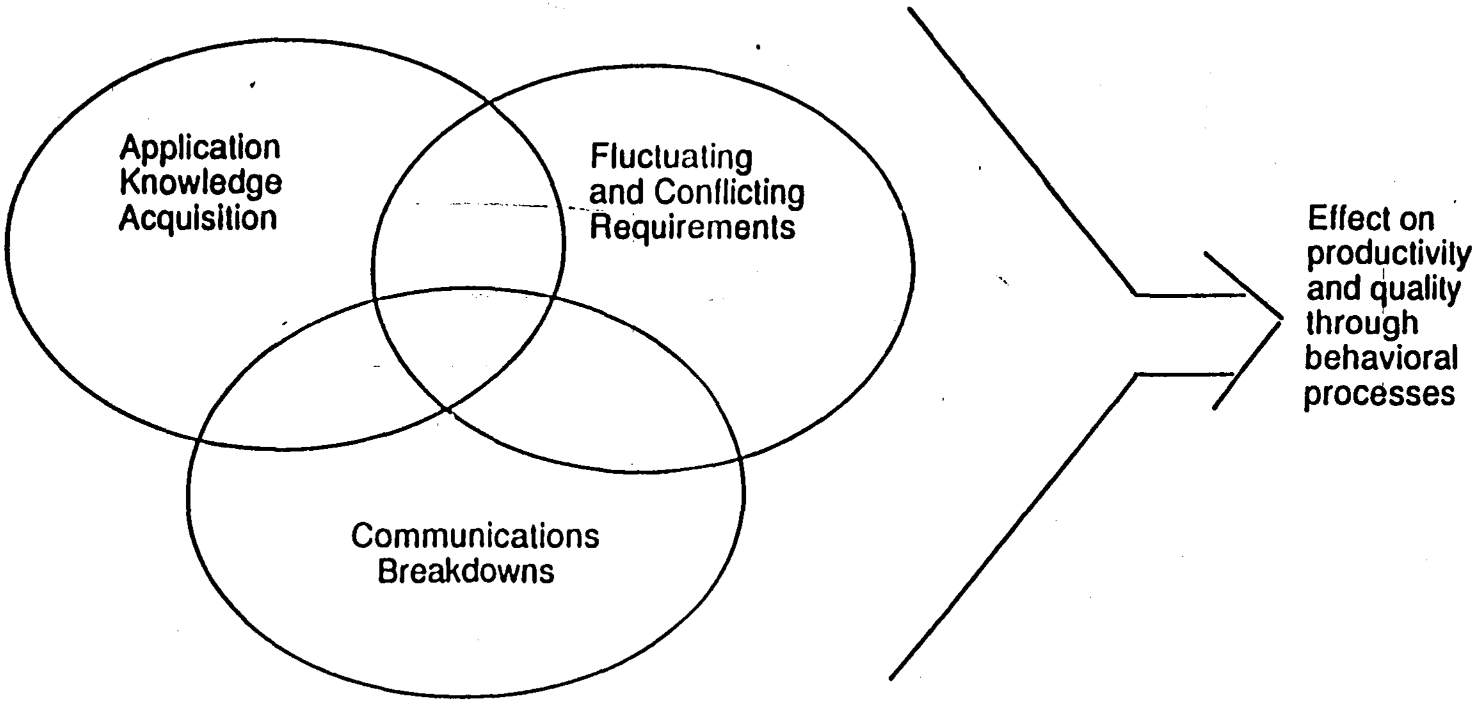
Project	Stage of Life Cycle	KLOC	Characteristics: Real time	Dist. Sys.	Emb. Sys.	Gov.	Application
1	Terminated	-					Support Software
2	Development	24	✔		✔		Radio Control
3	Development	50	✔	✔	✔		Process Control
4	Development	50	✔				Operating System
5	Design	70					CAD
6	Development	130				✔	CAD
7	Development	150+	✔		✔	✔	Avionics
8	Maintenance	194	✔			✔	C^3
9	Development						Compiler
10	Maintenance	250					Run-time Library
11	Development	350+					Compiler
12	Maintenance	400					Transaction Proc.
13	Design	500	✔	✔			Telephony
14	Maintenance	725					Operating System
15	Development	1000	✔	✔			Telephony
16	Maintenance	50k+	✔	✔	✔	✔	Radar, C^3
17	Requirements	100k+	✔	✔	✔	✔	C^3, Life Support

Summary of Results from MCC Field Study*

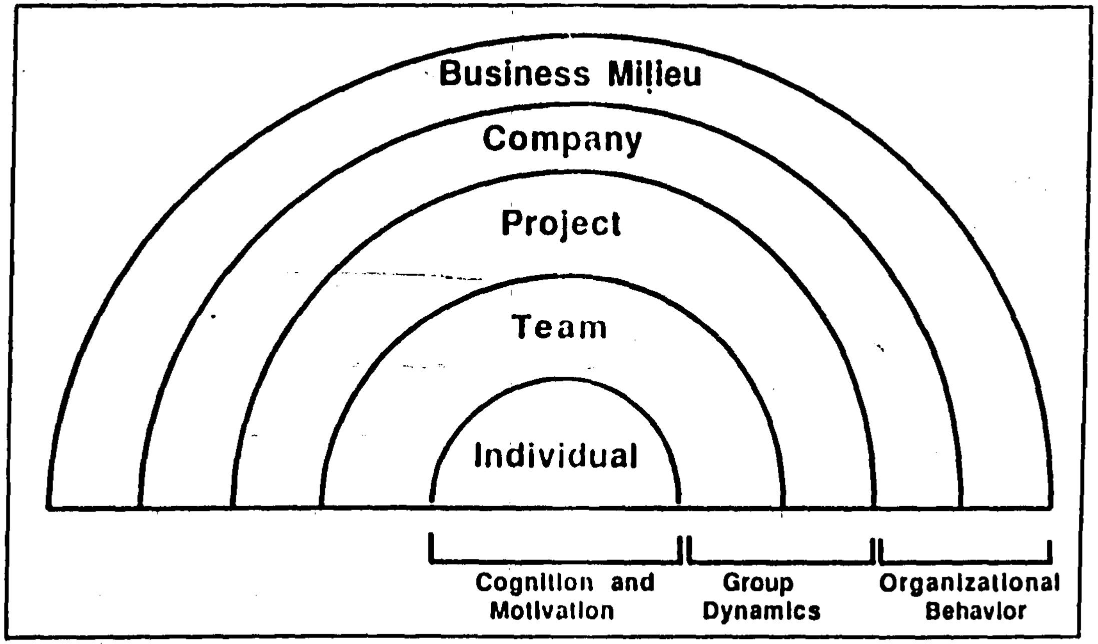
- Analysis of three significant problems
- Layered behaviorial model of software processes
- Conclusions and implications

* Paper appearing in this months CACM

Analysis of Three Significant Problems in Software Design for Large Systems



Layered Behavorial Model of Software Processes



Implications of Field Study Results

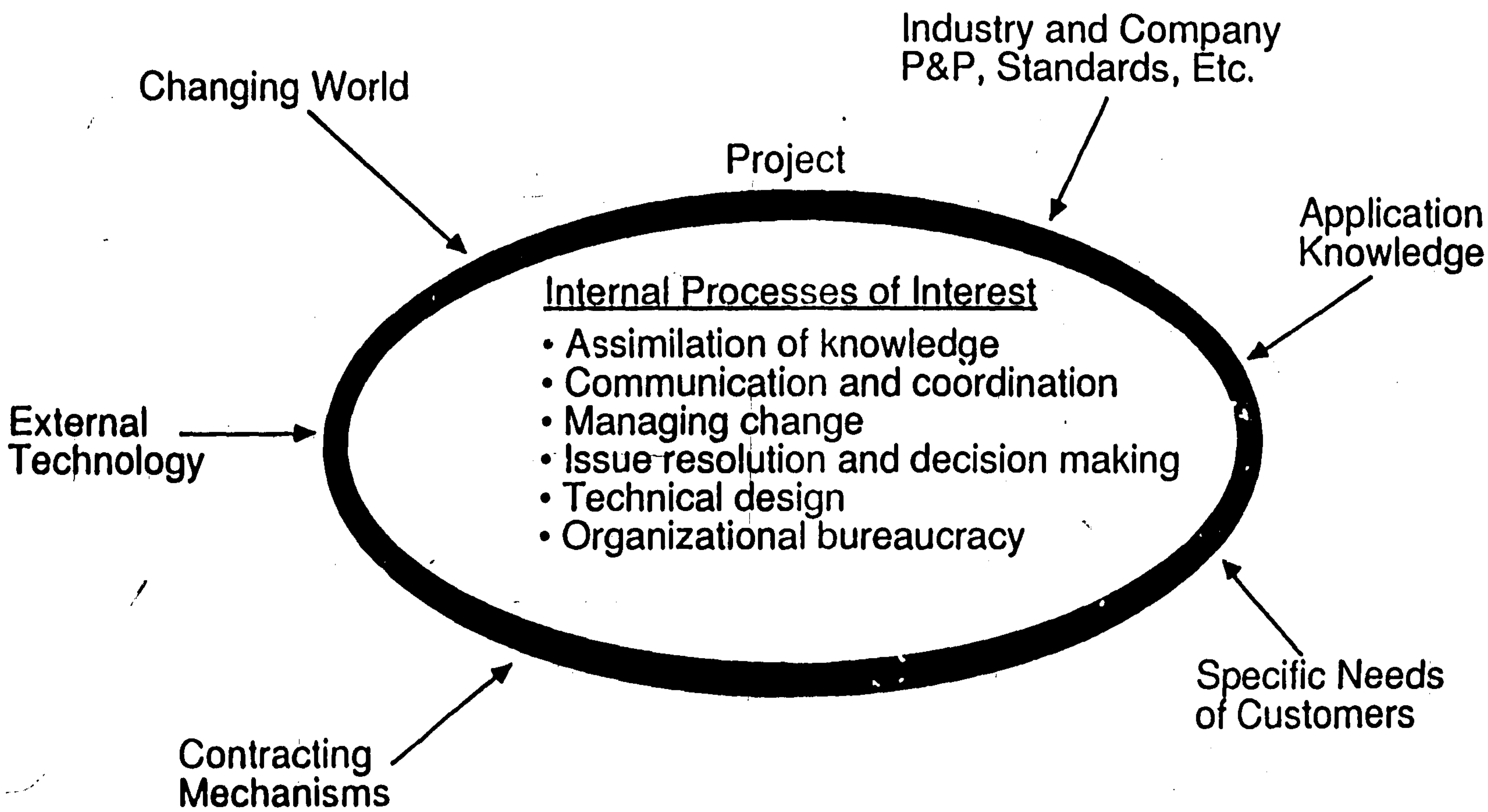
- For Software Technology
  - Environment support needed for:
    - Knowledge integration
    - Change facilitation
    - Broad communication and coordination
  - Beginnings of an empirical model to measure improvement for a tool/practice

- For Project Management
  - Expertise is the primary determinant, new ways of effectively organizing should be pursued
  - Key role players identified and described:
    superconceptualizer, diagnostician, gatekeeper, boundary spanner
  - Coordination by shared model of process, product

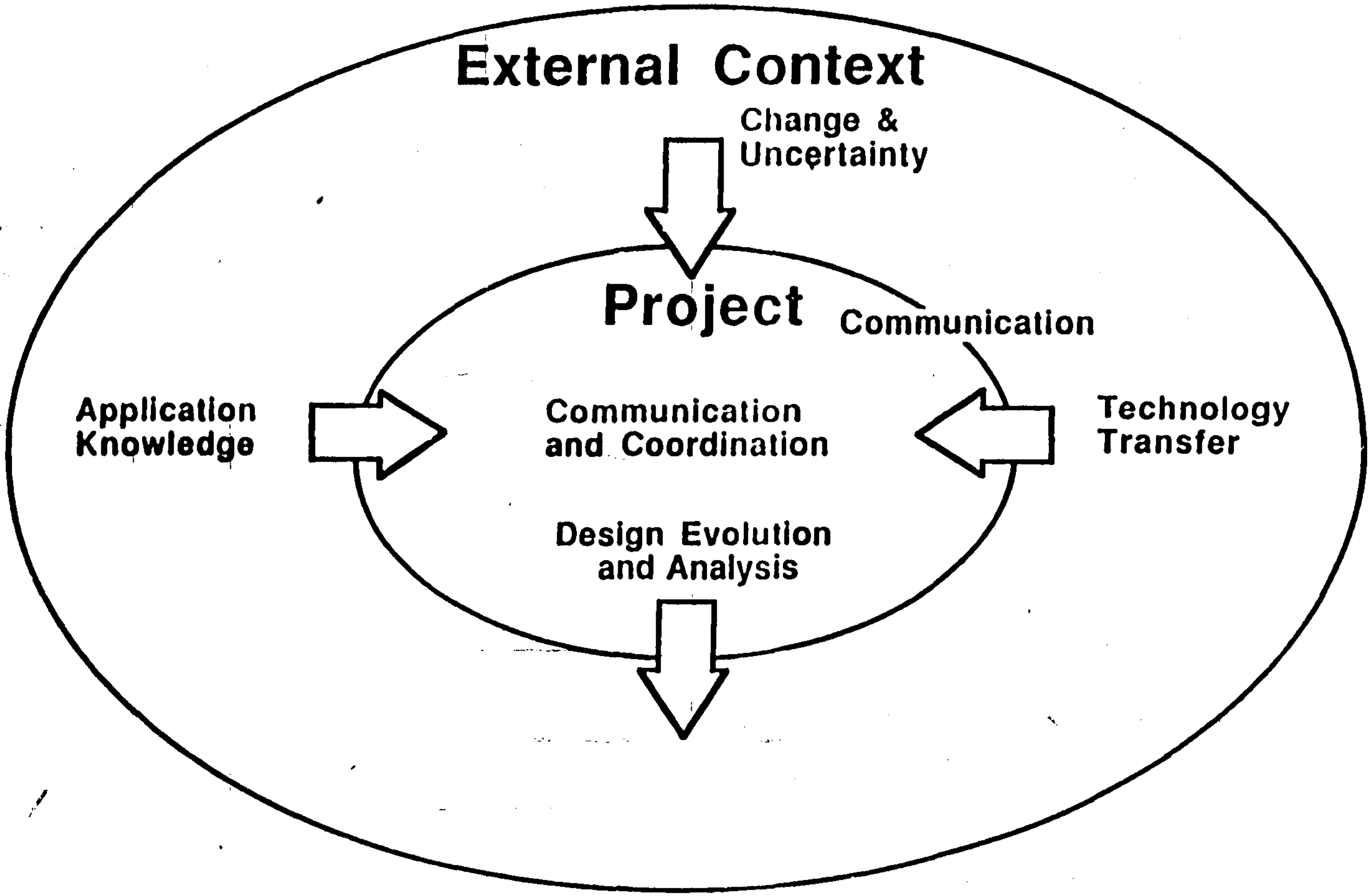
- For Software Process Models
  - Difference between prescriptive and actual processes
  - Current process models do not reflect:
    learning, technical communication, requirements negotiation, and customer interaction
  - Framework for an "ideal" process model emerging

- For Further Empirical Research on Professional Software Engineering
  - Much more to do
  - Focus on "variation" and its effect on the difference in productivity and quality outcomes
    among people, situations, and their interaction

The Software Project as an Ecological System



Five Crucial Problem Areas in Large Software Projects*



* see STP-390-86p

Overall Conclusion

The Greatest Leverage Is in Supporting the Intersection of:

The Technical Task

- Assessing customer needs
- Assimilating application knowledge
- Negotiating requirements, technology, and resources
- Identifying and exploring design assumptions/alternatives
- Decomposing and recomposing functionality
- Defining and controlling component interfaces

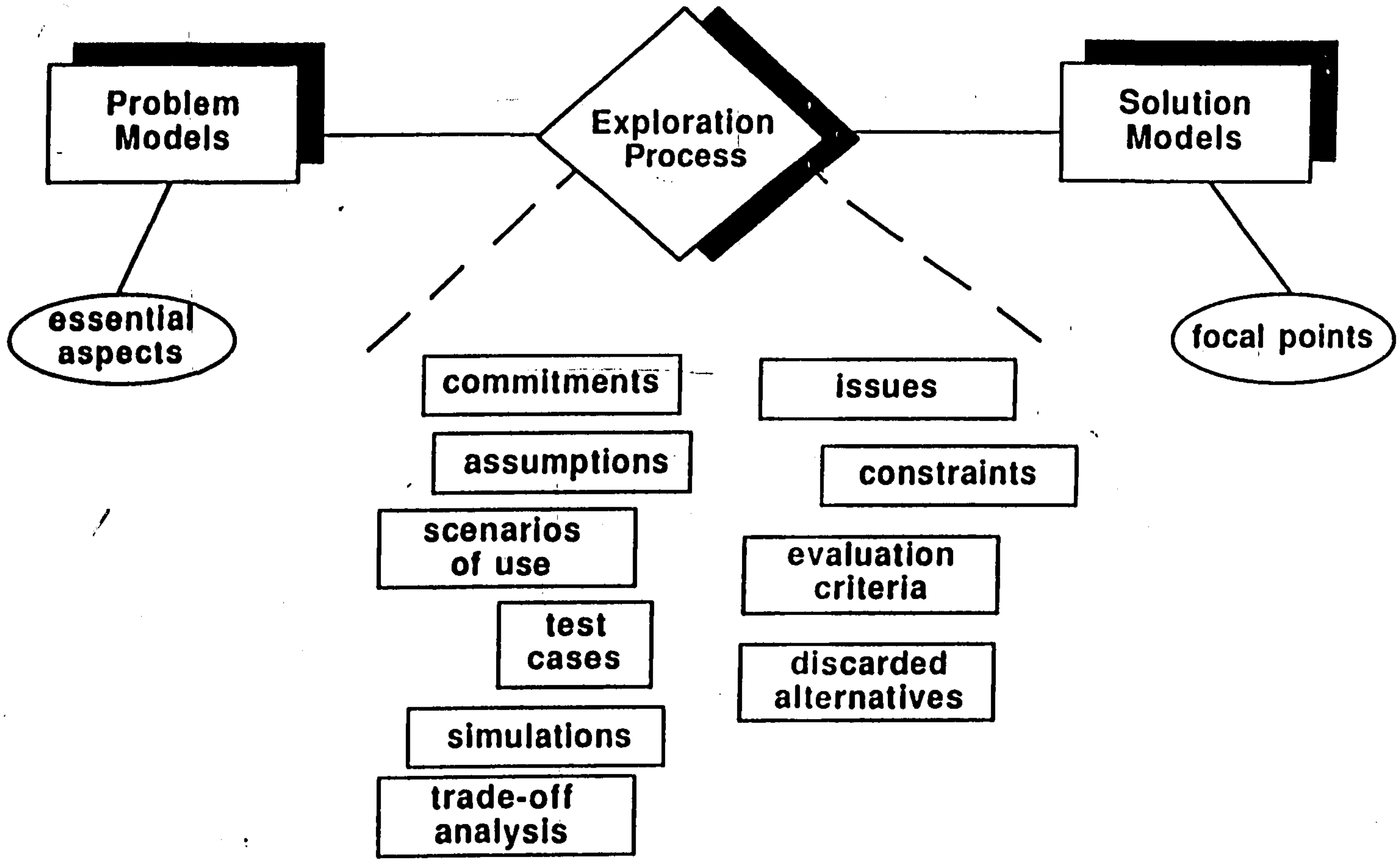
The ManagementTask

- Strategically managing system features and attributes
- Assessing and controlling risks
- Ensuring developers work from the same models

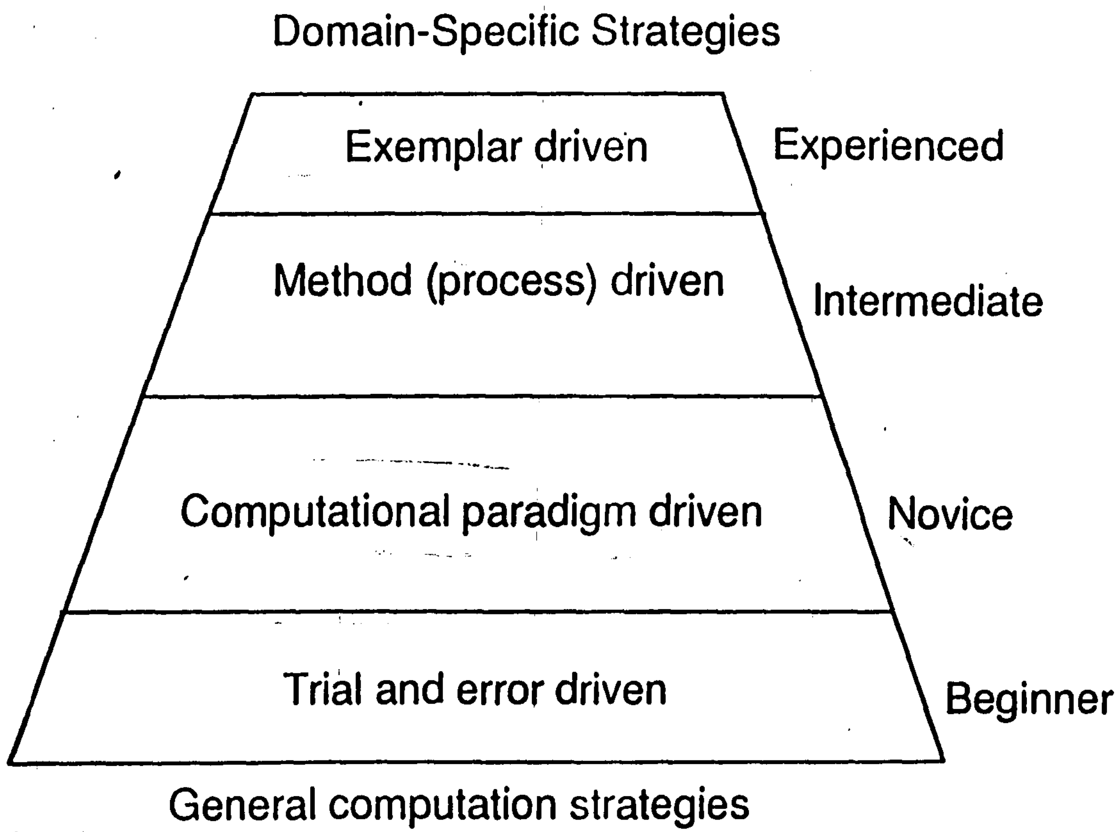
Results of the "LIFT" Study

- Observations on relative effort distribution
- Observations about individual differences
- Identification of six process breakdowns
- A cognitive model of design problem solving

Information Model of Design Exploration



Individual Differences in Software Design Strategies



Results of the Team Design Study*

- Identification of <u>conflict behavior</u> as key to achieving shared models
- Observations on the limitations of "documents"
- Observation of ombudsman to facilitate communication between customer and design teams
- Observations on the effect of *midnight prototype* creation
- Videotape identified as history capture mechanism

* being completed at U.T. - D. Walz, 1988

Future SSEs Should Contain Facilities For

1) Focus on Productivity and Quality
   - Statistical QC
   - Reduce waste and redundancy
   - Institutionalized reuse process yields component parts (via standards)

2) Process Engineering
   - Introduction of good practices, tools, etc.
   - Process definition, tailoring, monitoring, analysis, and improvement
   - Embodiment in education programs

3) Process Efficiency through Teamwork and Communication
   - Revocation of Brook's Law
   - High performance teamwork
   - "Groupware"

4) Flexible Organization Evolution
   - Coordinated technology, policy and organizational structure around process management concerns
   - Committment to improve (facilitation of change)
   - Capture of corporate domain knowledge (via issue-oriented domain analysis)
   - Negotiation-based requirements technology

5) Liveware Support
   - Variety of "experts" (stakeholders)
   - Significant variation in abilities

PUBLICATIONS

Field Study Papers

Curtis, B., Krasner,. H., and Iscoe, N. (1988), A Field Study of the Software Design Process for Large Systems, in Communications of the ACM, Vol. 31, No. 11, November, 1988

Krasner, H., Curtis, B. and Iscoe, N. (1987) Communications Breakdowns and Boundary Spanning Activites on Large Software Projects, In Proceedings of the Second Annual Conference on Empirical Studies of Programmers, Chapter 4, Ablex, Inc., Norwood, NJ.

Curtis, B., Krasner,. H., Shen, V. and Iscoe, N. (1987), On Building Process Models Under the Lamppost,In the Proceedings of the Ninth International Conference on Software Engineering, Washington, DC: IEEE Computer Society, 1987, 96-103.

Krasner, H., Shen, V., Curtis, B. and Iscoe, N. (1986) Preliminary Observations from the MCC Field Study of Large Software Projects, MCC Technical Report Number STP-390-86P.

Shen, V., Krasner, H., Curtis, B. (1986) A Field Study Plan for Developing Models of the Design Process, MCC Technical Report Number STP-115-86P.

Team Study Papers

Elam, J., Walz, D., Krasner, H., Curtis, B. (1987), A Methodology for Studying Software Design Teams: An Investigation of Conflict Behaviors in the Requirements Definition Phase,In Proceedings of the Second Annual Workshop on Empirical Studies of Programmers, Chapter 6, Ablex, Inc., Norwoood, NJ.

Walz, D. (1988), Phd Dissertation, U. of Texas, to appear

Individual Study Papers

Guindon, R., Krasner, H., Curtis, B. (1987) Breakdowns and Processes During the Early Activities of Software Design by Professionals, In Proceedings of the Second Annual Workshop on Empirical Studies of Programmers, Chapter 5, Ablex, Inc., Norwoood, NJ.

Guindon, R., Krasner, H., Curtis, B.(1987b) A Model of Cognitive Processes in Software Design: An Analysis of Breakdowns in Early Design Activities by Individuals, MCC Technical Report Number STP-283-87.

Motivational Slide for this Morning

In a study of 38 U.S. and Japanese Companies a wide variety of software management strategies were observed (Cusumano, 1987). It was concluded that Japanese firms are significantly ahead in applying a disciplined and flexible factory approach, as evidenced by:

Japan	$\frac{.26 \text{ bugs}}{1000 \text{ SLOC}}$	5% projects late	34% reuse
U.S.	$\frac{8.3 \text{ bugs}}{1000 \text{ SLOC}}$	43% projects late	15% reuse

N91

19729

UNCLAS

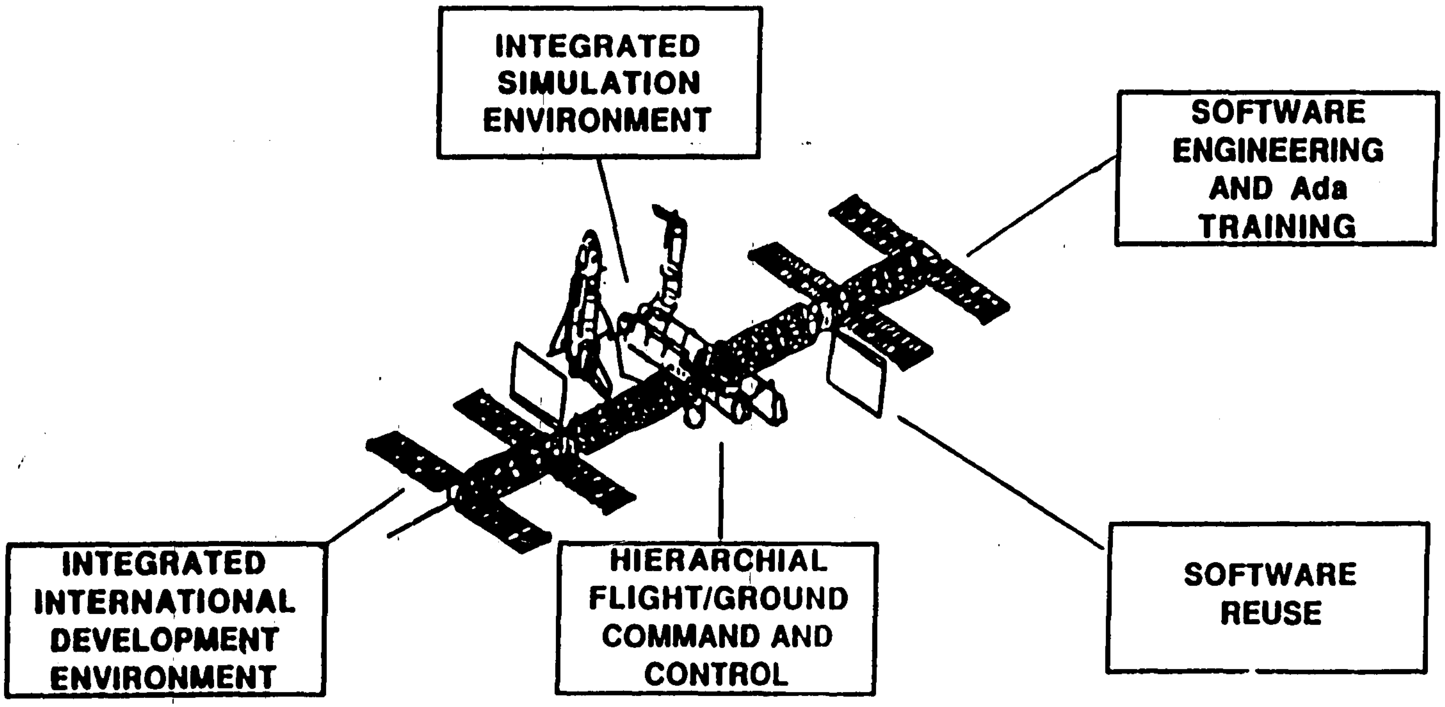
THE ROLE OF SOFTWARE ENGINEERING IN THE SPACE STATION PROGRAM

RICIS SYMPOSIUM 1988
INTEGRATED COMPUTING ENVIRONMENTS FOR LARGE, COMPLEX SYSTEMS

NOVEMBER 9-10, 1988

DANA HALL
SPACE STATION PROGRAM OFFICE
RESTON, VIRGINIA

SNAPSHOTS OF SOFTWARE ENGINEERING APPLICATIONS WITHIN THE SPACE STATION FREEDOM PROGRAM



SOFTWARE ENGINEERING AND ADA TRAINING

- **SOFTECH (RICIS) 1987 SURVEY OF NASA:**
  - OVER 150 ADA PROJECTS WITHIN 5 YEARS
  - MINIMAL EXPERIENCED PERSONNEL
  - MUCH SOFTWARE ENGINEERING AND ADA TRAINING NEEDED
  - FEW WRITTEN SOFTWARE DEVELOPMENT POLICIES

- **TRAINING RECEIVING MUCH MORE ATTENTION (but its still too little and ..... maybe too late)**

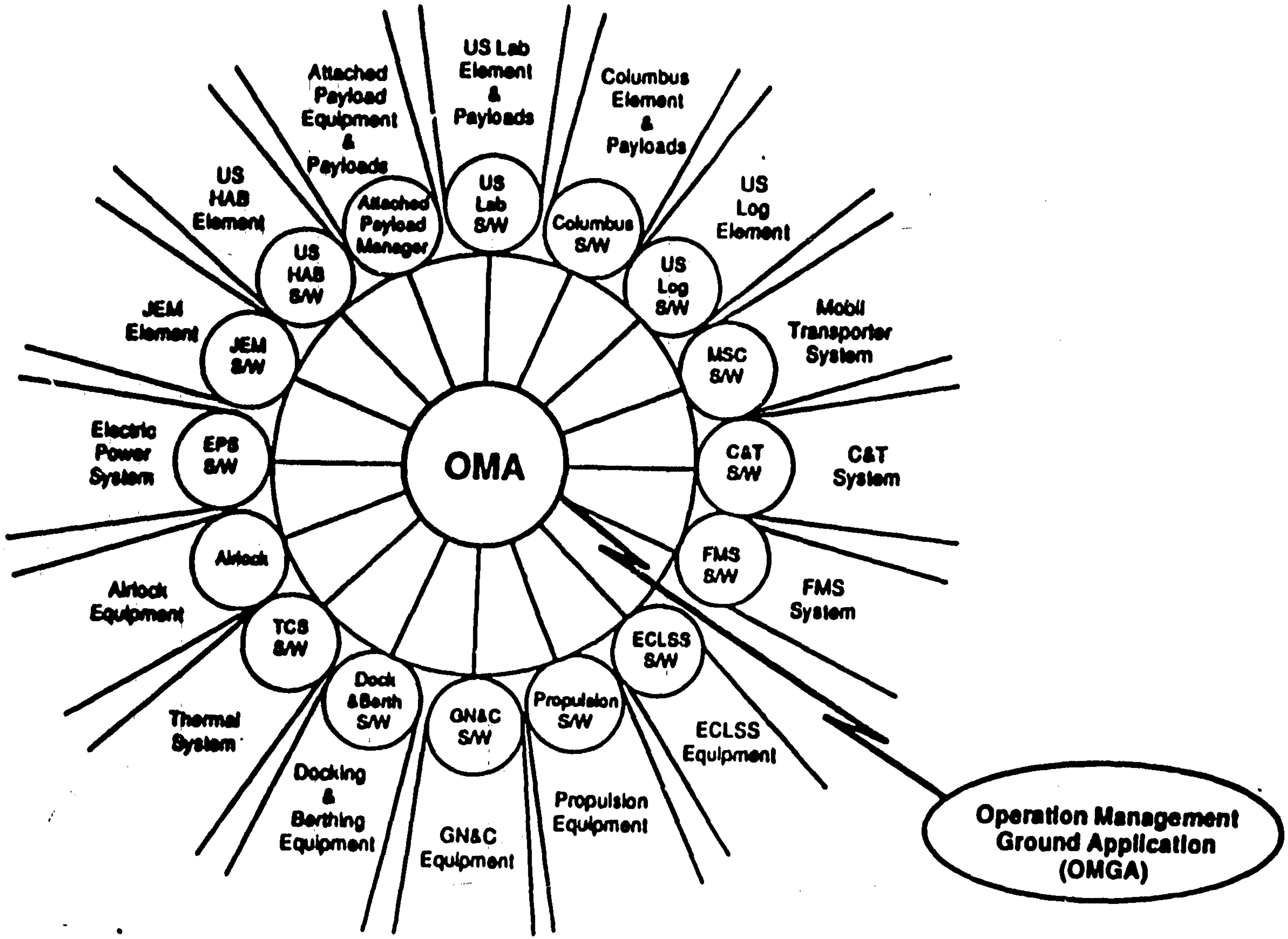
  EXAMPLES:

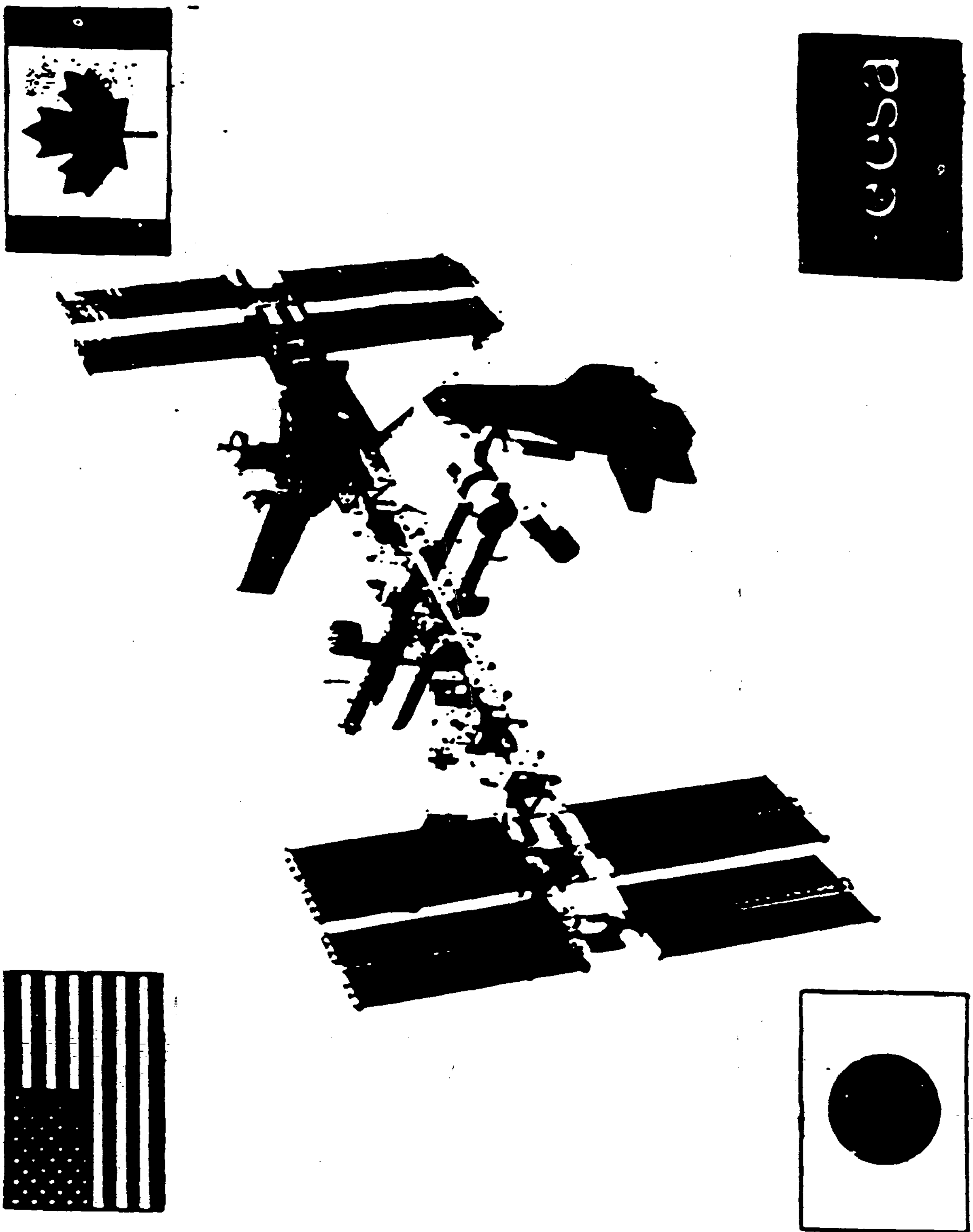
  - EXPANDED COMPUTER-BASED AND CLASS ROOM TRAINING FROM SSE
  - TOP MANAGEMENT ATTENTION

SOFTWARE REUSE

- o WANT TO CAPITALIZE ON RESEARCH AND EXPERIENCE TO DATE
  - EXAMPLES:
    - o ARMY 'RAPID' TOOL FOR LIBRARY MANAGEMENT
    - o UNIVERSITY TAXONOMY/ ATTRIBUTES WORK
- o POLICY WILL ENCOURAGE REUSE
  - COMPONENT DEVELOPMENT
  - TRY DURING RAPID PROTOTYPING
- o PIVOT POINTS
  - CONTRACTOR INCENTIVES
  - ACCOUNTABILITY AND LIABILITY

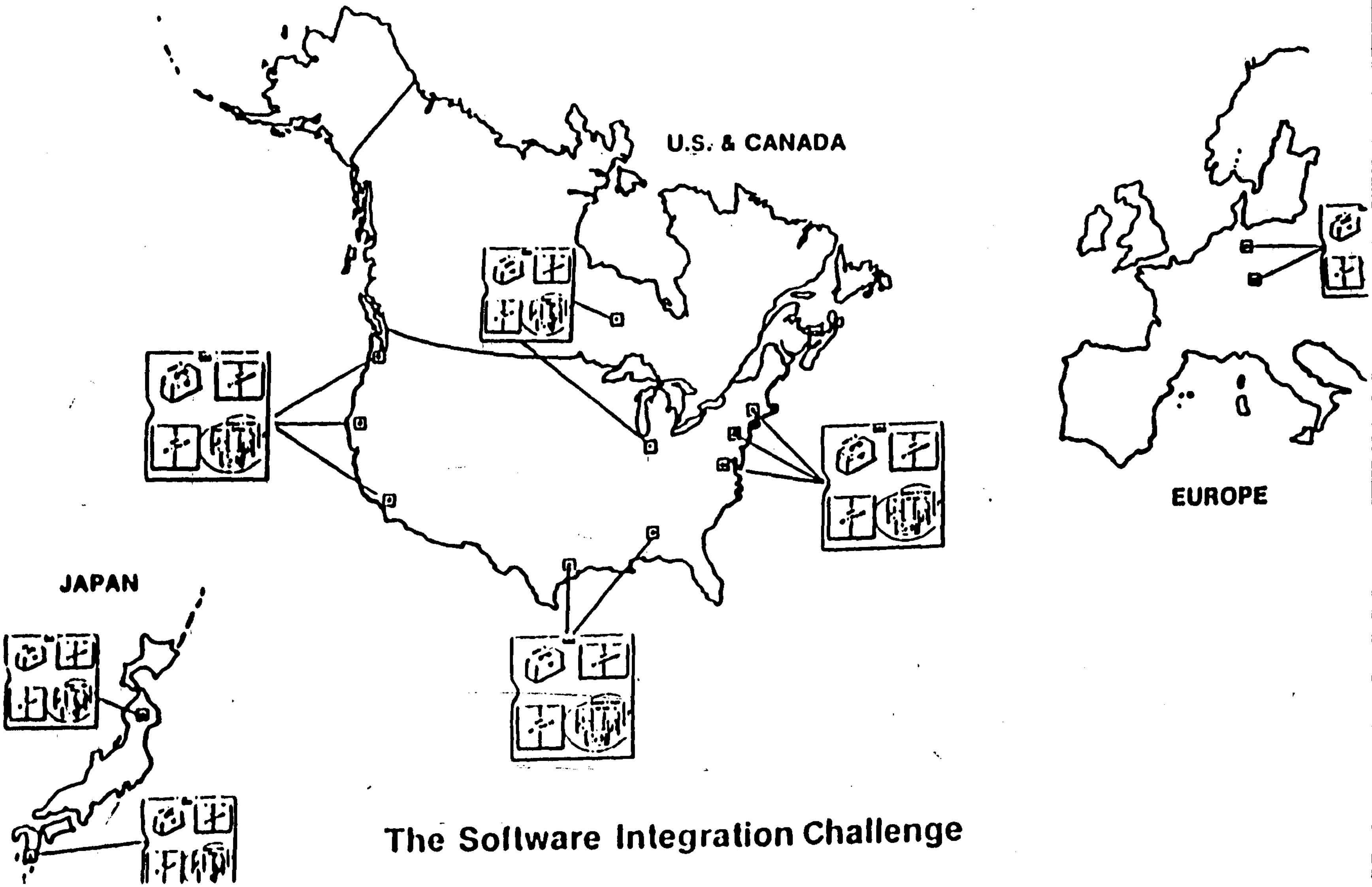
HIERARCHIAL COMMAND AND CONTROL





ORIGINAL PAGE IS
OF POOR QUALITY

PROGRAM CHARACTERISTICS
DISTRIBUTED SOFTWARE DEVELOPMENT & MAINTENANCE



The Software Integration Challenge

INTEGRATED, INTERNATIONAL ENVIRONMENTS

(HOW MUCH IS NECESSARY FOR SPACE STATION FREEDOM PROGRAM ?)

- ESA, NASDA, AND CANADA WILL PROVIDE MISSION/LIFE CRITICAL FLIGHT SOFTWARE
  - QUALITY OF PARTNER DEVELOPMENT ENVIRONMENTS UNKNOWN AND UNCONTROLLED
  - LIMITED EXPERIENCE IN MAN-RATED, COMPLEX SOFTWARE

- HOW COMMON OR INTEROPERABLE MUST BE THE DEVELOPMENT ENVIRONMENTS?..... AND HOW DO WE ANSWER THAT QUESTION?
  - WRITE TIGHT INTERFACE SPECS?

    → HOW CAN WE DETERMINE NECESSARY DATA EXCHANGES?

  - PROVIDE THE SSE TO THE PARTNERS?

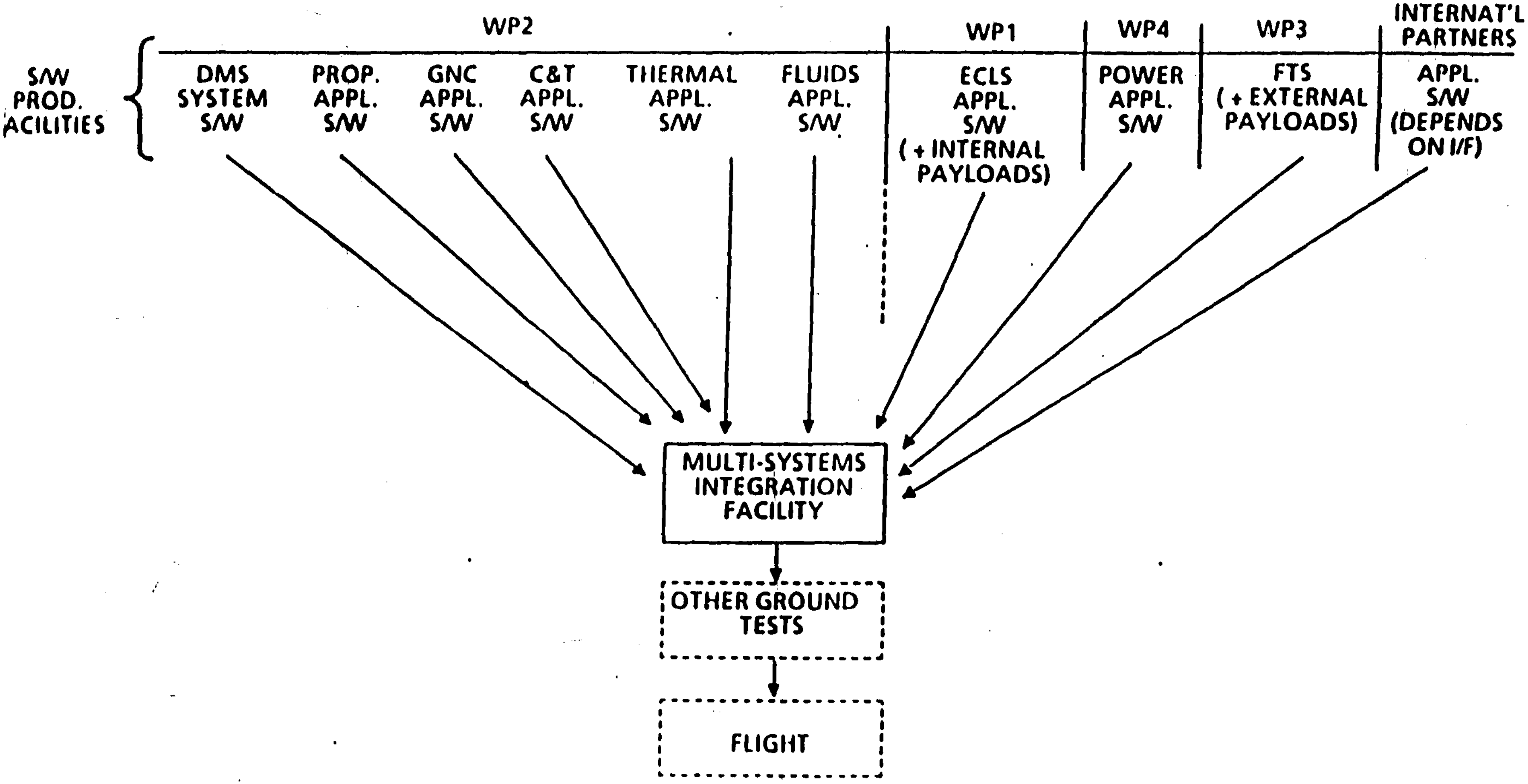
    → TECHNOLOGY TRANSFER (?)

  - SHARE A CRITICAL SUBSET?

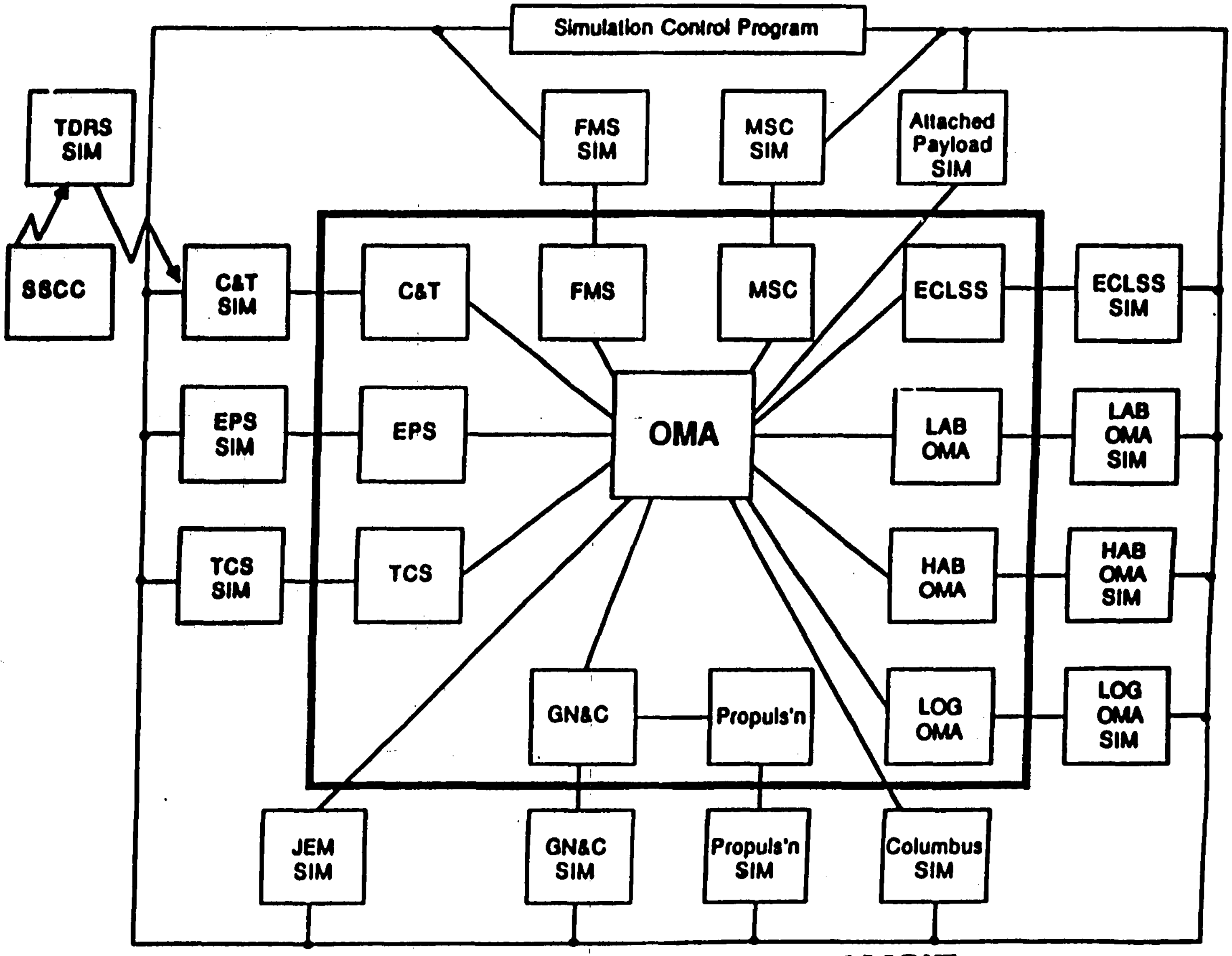
    → STANDARDS? TOOLS? ALL OR PART? ENVIRONMENTS ARE TIGHTLY INTEGRATED

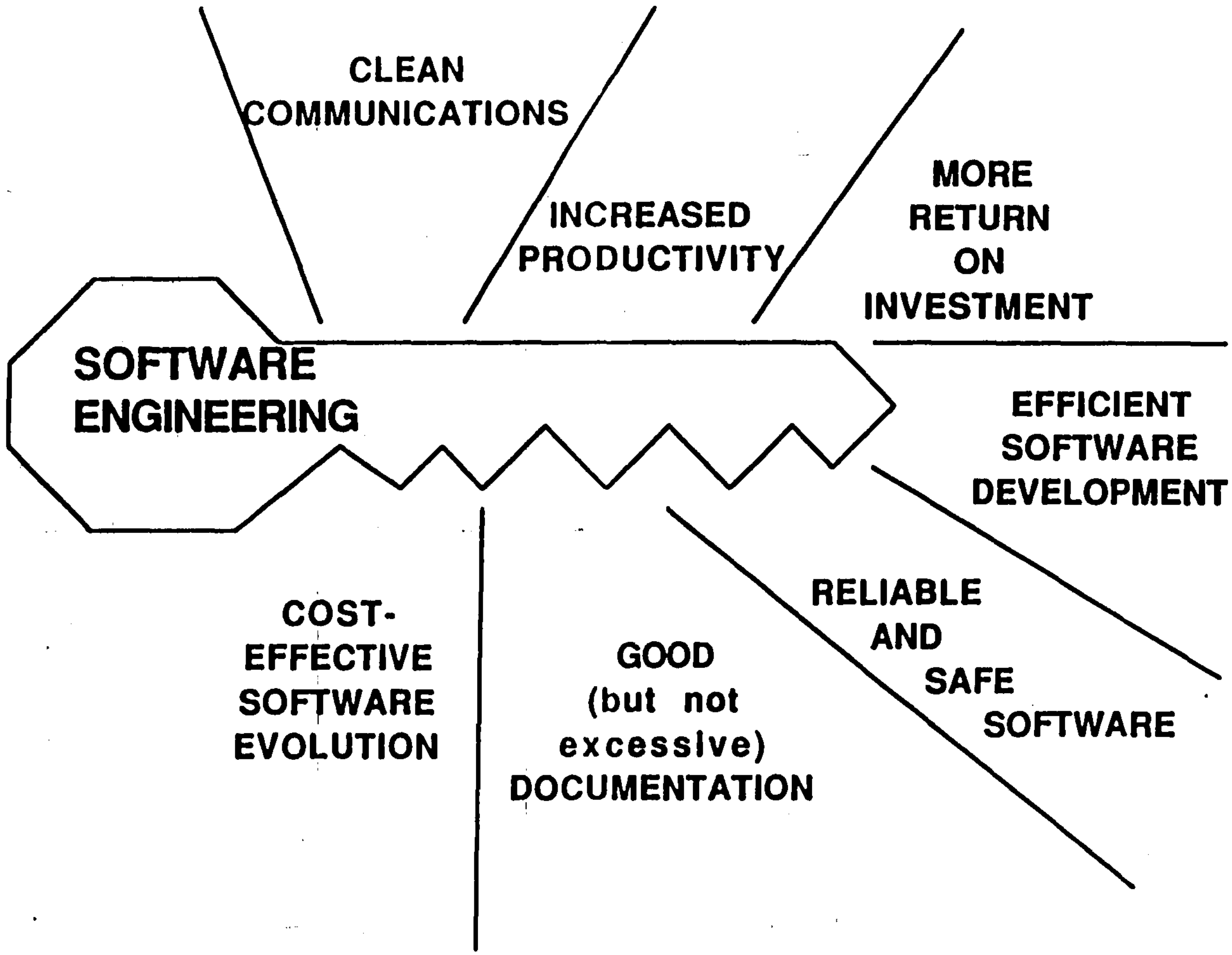
THIS ISSUE MUST BE SETTLED SOON

SOFTWARE PRODUCTION, INTEGRATION, AND MANAGEMENT



INTEGRATED SIMULATION ENVIRONMENT





SOFTWARE ENGINEERING IS THE KEY TO MAXIMIZING PROGRAM "PROFIT"

N91
19730
UNCLAS

LESSONS LEARNED FROM AN Ada CONVERSION PROJECT

N91-19730

Tim Porter

10 November 1988

SAIC COMSYSTEMS

58-61

320505

P-13

BACKGROUND

- SOFTWARE AUTOMATED VERIFICATION AND VALIDATION SYSTEM (SAVVAS)
- ORIGINALLY DEVELOPED
  - FOR VAX/VMS
  - USING DEC Ada
- PORTED FOR NASA SPACE STATION SSE
  - TO IBM 3090/VM
  - USING ALSYS Ada

BACKGROUND

- SOFTWARE AUTOMATED VERIFICATION AND VALIDATION SYSTEM (SAVVAS)
- ORIGINALLY DEVELOPED
  - FOR VAX/VMS
  - USING DEC Ada
- PORTED FOR NASA SPACE STATION SSE
  - TO IBM 3090/VM
  - USING ALSYS Ada

SOFTWARE PORTABILITY

Software portability is measured by the relative ease with which software is moved between alternative hardware, compilers, operating system, and other external interfaces.

HOW DO WE IMPROVE PORTABILITY?

- STANDARD LANGUAGE - Ada
- ISOLATION OF NON-PORTABLE CODE
- CONSTRAINTS ON LANGUAGE FEATURES
- VIRTUAL INTERFACES

HISTORY OF ADA

1972	DoD recognizes rapid growth of software costs for military systems
1975	HOLWG reviews language requirements
1979	Ada selected from language design efforts
1983	Ada established as an ANSI standard
1985	DoD spends $11 billion on software
1987	Ada mandated by DoD directive 5034.2 NASA awards Space Station SSE contract
1988	STARS Competing Primes contracts awarded
1995	DoD projected software spending is over $25 billion

ISOLATION OF NON PORTABLE CODE

- CAPITALIZE ON Ada'S FEATURES
  - PACKAGES
- CLASSES OF DEPENDENT SOFTWARE
  - INPUT/OUTPUT
  - DATABASE ACCESS
  - OPERATING SYSTEM SERVICES

SIMPLE TERMINAL INTERFACE PACKAGE

```
package SIMPLE_TERMINAL_INTERFACE is

    procedure GO_TO_POSTIION_(X, Y: in INTEGER);

    procedure DISPLAY_TEXT (MESSAGE: in STRING);

end SIMPLE_TERMINAL_INTERFACE;

with TEXT_IO; use TEXT_IO;
package body SIMPLE_TERMINAL_INTERFACE is

    procedure GO_TO_POSITION_(X, Y: in INTEGER) is
    begin
        Send the appropriate code sequence to the terminal.
        These are different for varying terminal types.

    end GO_TO_POSTIION;

    procedure DISPLAY_TEXT (MESSAGE: in STRING) is
    begin
      -- Send the message to the terminal.
      -- Including any required code sequences.

    end DISPLAY_TEXT;

end SIMPLE_TERMINAL_INTERFACE;
```

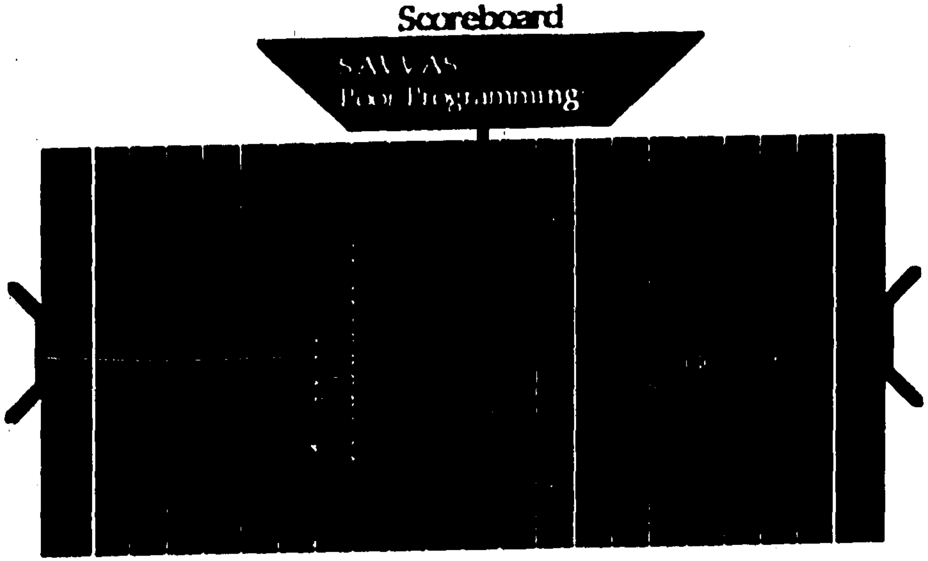
CONSTRAINTS OF LANGUAGE FEATURES

- TASKS
- PRAGMAS
- GENERICS
- EXCEPTION HANDLING

VIRTUAL INTERFACES

- DATABASE ACCESS
  - Ada/SQL
  - MODULE APPROACH
- INPUT/OUTPUT
  - X WINDOW
  - Ada-GKS
- OPERATING SYSTEM
  - CAIS
  - PCTE
  - POSIX

PLAY BY PLAY ANALYSIS

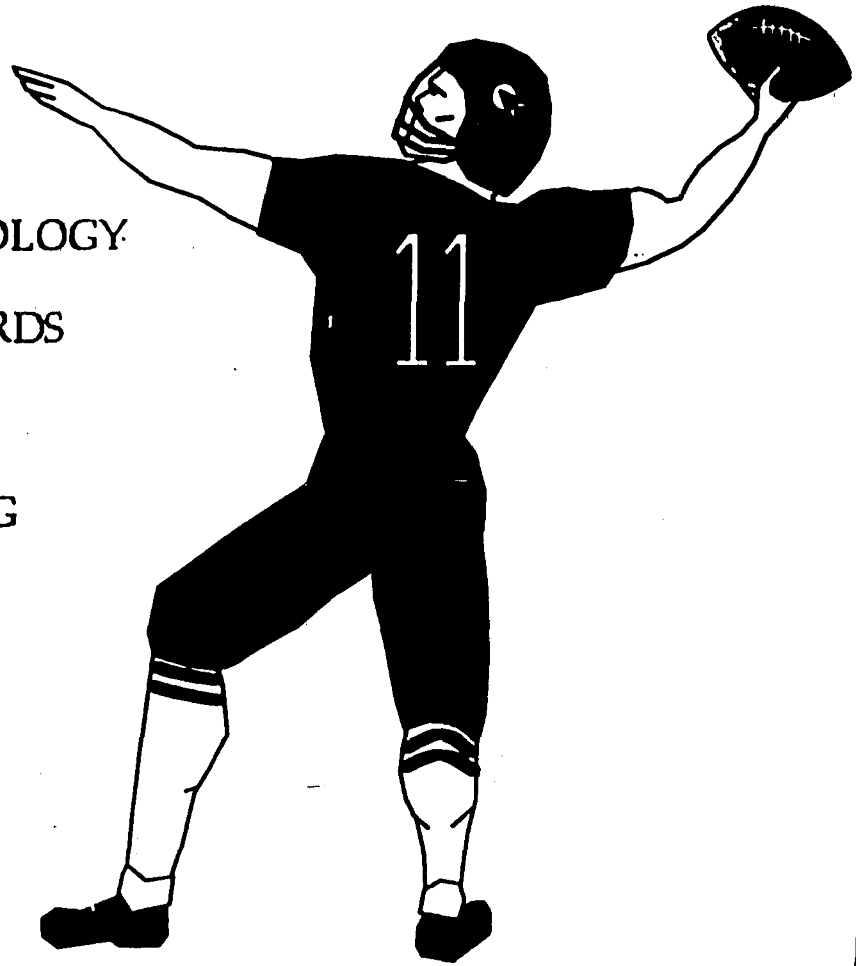


	SAVVAS	Poor Practices
• INPUT/OUTPUT	7	7
• PRAGMAS	7	14
• TRAINING EDUCATION	14	21
• EXCEPTION HANDLING	14	24
• DATABASE INTERFACE	21	24
• COMPILER COMPATIBILITY	28	31
• SOFTWARE LAYERING	35	31

ORIGINAL PAGE IS
OF POOR QUALITY

MODIFICATIONS TO THE PLAY BOOK

- CONSISTENT DESIGN METHODOLOGY
- DESIGN AND CODING STANDARDS
- VIRTUAL INTERFACES
- COMPREHENSIVE Ada TRAINING



END

DATE

FILMED

MAY 7 1991

End of Document