

AD-A165 122

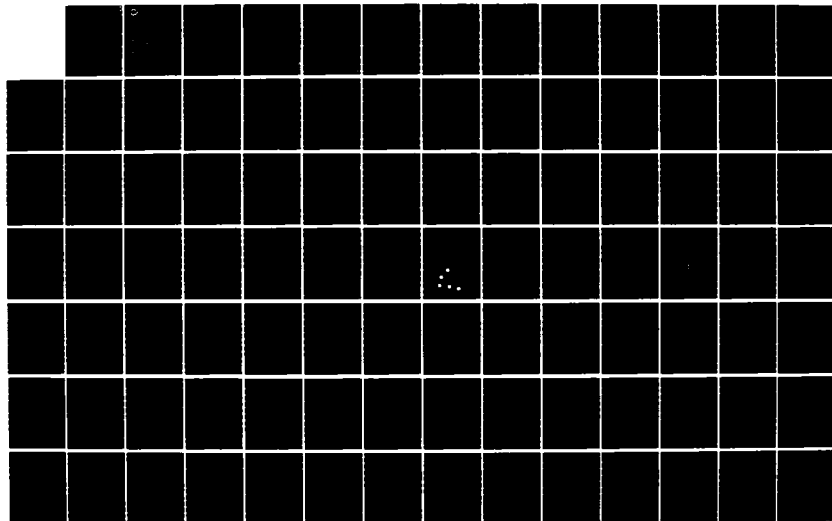
ADA (TRADEMARK) TRAINING CURRICULUM: INTRODUCTION TO
SOFTWARE ENGINEERING M102 TEACHER'S GUIDE(U) SOFTECH
INC WALTHAM MA 1986 DAAB07-83-C-K506

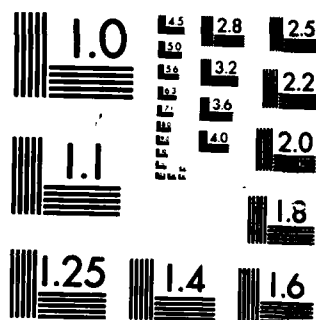
1/6

UNCLASSIFIED

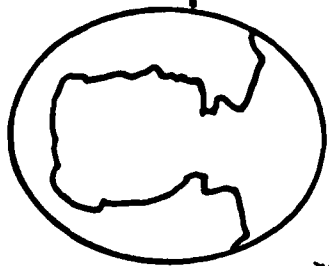
F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



AD-A165 122

Ada® Training Curriculum

DTIC FILE COPY

Introduction To Software Engineering M102 Teacher's Guide

86 3 12 058

U.S. Army Communications-Electronics Command
(CECOM)

Contract DAAB07-83-C-K506

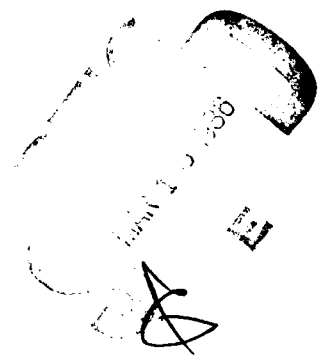
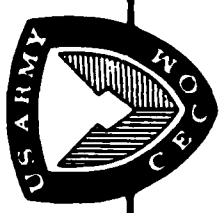
Prepared By:

SOFTECH, INC.
460 Totten Pond Road
Waltham, MA 02154

Ada is a registered trademark of the U.S. Government, Ada Joint Program Office

*Approved For Public Release/Distribution Unlimited

1986



Handwritten note:
Softech - 4/14/86
93-844

INSTRUCTOR NOTES

DURATION: 2 DAYS

GOALS:

1. DEVELOP A CONCEPTUAL UNDERSTANDING OF SOFTWARE ENGINEERING
2. OVERVIEW UNDERSTANDING OF SOFTWARE ENGINEERING METHODS
3. ESTABLISH A RELATIONSHIP BETWEEN SOFTWARE ENGINEERING AND Ada

THE FOLLOWING ARE NOT GOALS:

1. TO TEACH HOW TO USE ANY SPECIFIC DEVELOPMENT METHODOLOGY OR TOOL

INTENDED AUDIENCE:

SOFTWARE ENGINEERING (PROGRAMMERS, QA, INTEGRATION, TECHNICAL LEADS)

TELL CLASS ABOUT Ada CURRICULUM; PASS OUT HANDOUT.

BRIEFLY DISCUSS Ada AND THE ALS.

VG 744.1

1-1

Copyright by SofTech, Inc. 1984. This material may be reproduced by or for the U.S. Government pursuant to the copyright license under DAR clause 7-104.9(a) (May 81).

INTRODUCTION TO SOFTWARE ENGINEERING

M102

VG 744.1

Accession No.			
Author			
Title			
Subject			
Dist	A-1		



INSTRUCTOR NOTES

Contents:

DAY 11-

I. 1) INTRODUCTION -
2) BACKGROUND AND MOTIVATION;

II. 2) SOFTWARE ENGINEERING AND ITS GOALS -
3) GOALS ^{and} SOFTWARE ENGINEERING PRINCIPLES;

III. 3) HOW THESE GOALS ARE ACHIEVED -

4. THE LIFE-CYCLE MODEL TO DEVELOP SYSTEMS,
5. MIL-STDs AND DOCUMENTATION,
6. INTRODUCTION TO METHODS AND TOOLS,
7. REQUIREMENTS ANALYSIS: INTRODUCTION, and
8. ANALYSIS METHOD OVERVIEW;

DAY 12-

IV. 4) HOW THESE GOALS ARE ACHIEVED (CONTINUED) -

9. DESIGN OVERVIEW,
10. DESIGN METHOD OVERVIEW,
11. DETAILED DESIGN TECHNIQUES,
12. IMPLEMENTATION OVERVIEW,
13. STRUCTURED PROGRAMMING,
14. TESTING APPROACHES, and
15. SOFTWARE MANAGEMENT;

SUMMARY-

16. RELATIONSHIP OF SOFTWARE ENGINEERING TECHNIQUES TO Ada

VG 744.1

11-1

Handwritten notes:
Requirements!
Instructional Goals
Methods; Goals

TOPICS TO BE COVERED

INTRODUCTION: BACKGROUND AND MOTIVATION

SOFTWARE ENGINEERING AND ITS GOALS

ACHIEVING SOFTWARE ENGINEERING GOALS

SOFTWARE ENGINEERING AND Ada

PART I

INTRODUCTION

VG 744.1

INSTRUCTOR NOTES

THEME: SOFTWARE ENGINEERING IS AN APPROACH (OR SET OF APPROACHES) TO ADDRESSING
THE "SOFTWARE CRISIS"

PURPOSE: TO MOTIVATE THE STUDY OF SOFTWARE ENGINEERING AND METHODOLOGIES

REFERENCES: J.D. MUSA (EDITOR), "STIMULATING SOFTWARE ENGINEERING PROGRESS - A REPORT
OF THE SOFTWARE ENGINEERING PLANNING GROUP", ACM SIGSOFT SOFTWARE
ENGINEERING NOTES, VOL 8, NO. 2 APRIL 1983

SECTION 1

BACKGROUND AND MOTIVATION

VG 744.1

INSTRUCTOR NOTES

NO COMMON, SIMPLE DEFINITION FOR SOFTWARE ENGINEERING.

SEEWG = SOFTWARE ENGINEERING ENVIRONMENT WORKING GROUP, WORKING GROUP
TO DEFINE REQUIREMENTS FOR A NAVY STANDARD SOFTWARE ENGINEERING
ENVIRONMENT.

SOME DEFINITIONS

- "SOFTWARE ENGINEERING IS THE APPLICATION OF SCIENCE AND MATHEMATICS BY WHICH THE CAPABILITIES OF COMPUTER EQUIPMENT ARE MADE USEFUL TO MAN VIA COMPUTER PROGRAMS, PROCEDURES AND ASSOCIATED DOCUMENTATION."

BOEHM 1981

- "... A SOFTWARE ENGINEERING PROCESS IS A SET OF ACTIVITIES FOR DEVELOPING AND MODIFYING SOFTWARE THROUGH ITS LIFE CYCLE."

SEEWG REPORT 1982

- "A METHODOLOGY IS A REPEATABLE HUMAN PROCEDURE WHICH SUPPORTS SOME ASPECT OF AN ACTIVITY."

SEEWG REPORT 1982

INSTRUCTOR NOTES

THIS IS THE DEFINITION OF THE SOFTWARE CRISIS THAT MOTIVATED THE DEVELOPMENT OF ADA.

GIVE SOME PERSONAL EXAMPLES OF SYSTEMS YOU HAVE WORKED ON OR ASK THE CLASS FOR SOME.

MOTIVATION FOR SOFTWARE ENGINEERING

(SOFTWARE CRISIS)

• SOFTWARE FOR COMPLEX MILITARY SYSTEMS

- IS USUALLY LATE
- COSTS MORE THAN ORIGINALLY ESTIMATED
- DOES NOT WORK TO ORIGINAL SPECIFICATIONS
- IS UNRELIABLE
- IS DIFFICULT AND COSTLY TO MAINTAIN

INSTRUCTOR NOTES

TALK TO EACH BULLET, GIVING THE CLASS PERSONAL EXPERIENCE OR TRY TO GET THEM TO RELATE THEIR EXPERIENCES.

NOTE: DURING THIS SECTION YOU ARE TRYING TO SET THE STAGE FOR LATER PARTICIPATION BY THE CLASS. MAKE THEM FEEL AS IF THEY "OWN" THESE PROBLEMS.

MOTIVATION FOR SOFTWARE ENGINEERING

(ADDITIONAL PROBLEMS)

- SOFTWARE IS NOT REUSABLE ON DIFFERENT SYSTEMS
- PROLIFERATION OF METHODS, LANGUAGES AND ARCHITECTURES
- METHODS AND LANGUAGES NOT SUITED FOR CURRENT APPLICATIONS
- SUPPLY OF QUALITY SOFTWARE PERSONNEL NOT ABLE TO MEET
CURRENT SOFTWARE DEMAND
- SOFTWARE TASKS ARE MORE COMPLEX NOW, BUT NO WIDELY USED
METHODS AND TOOLS TO DEAL WITH THE PROBLEM EXIST
- LACK OF ADEQUATE MANAGEMENT AND SOFTWARE DEVELOPMENT
METHODS/TOOLS

INSTRUCTOR NOTES

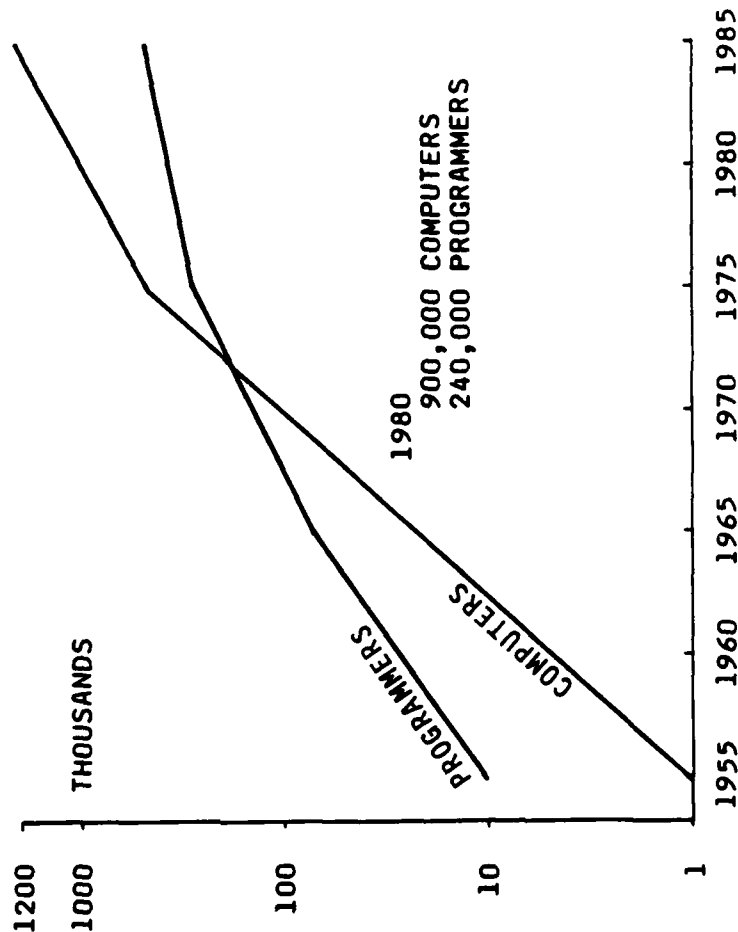
THE KEY POINT IS TO NOTE THE RATE OF GROWTH.

ENVIRONMENT FACING SOFTWARE ENGINEERING

• DEMAND FOR SOFTWARE OUTSTRIPPING DEVELOPMENT CAPABILITY

TOTAL U.S. DATA PROCESSING INDUSTRY EXPENDITURES

Year	Expenditure (billions of 1970 dollars)	Percent of GNP
1970	21	2.1
1975	41	3.2
1980	82	5.2
1985	164	8.3



• PRODUCTIVITY ADVANCES ARE NOT KEEPING UP

SOURCE: REPORT OF THE SOFTWARE ENGINEERING GROUP, 1983

VG 744.1

INSTRUCTOR NOTES

PRODUCTIVITY IS A MEASURE OF HOW EFFICIENTLY WE CAN PRODUCE SOFTWARE (NOTE THE SLOW RATE OF IMPROVEMENT).

PRODUCTION IS A MEASURE OF HOW MUCH THE DEMAND FOR SOFTWARE HAS INCREASED.

RELATIVE PROGRAMMER PRODUCTIVITY AND TOTAL U.S. YEARLY
CODE PRODUCTION (NORMALIZED TO 1955)

YEAR	PRODUCTIVITY	PRODUCTION
1955	1	1
1960	1.6	5
1965	2.0	16
1970	2.3	38
1975	2.7	59
1980	3.1	85
1985	3.6	119

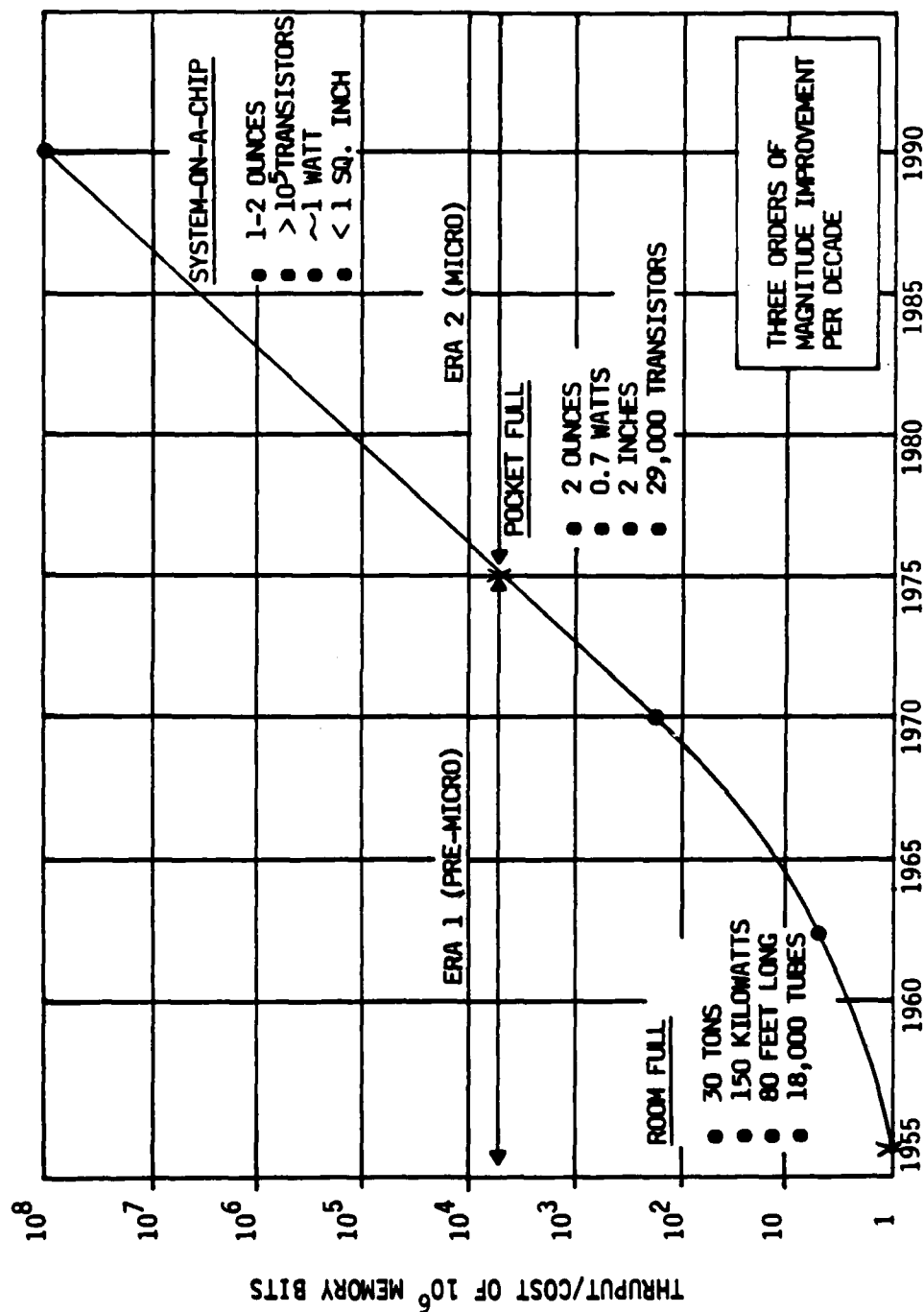
- WE HAVE A SERIOUS DEFICIENCY
- NEED MORE SOFTWARE DEVELOPERS
- MUST MAKE EXISTING ONES MORE PRODUCTIVE

INSTRUCTOR NOTES

POINT OUT THAT HARDWARE TECHNOLOGY IS GROWING SO FAST THAT WE DON'T KNOW HOW TO PROPERLY UTILIZE IT.

NOTE THE Y AXIS IS A WAY OF MEASURING THE COMBINED EFFECT OF FASTER HARDWARE TECHNOLOGIES AND HIGHER DENSITY MICRO-CIRCUIT TECHNOLOGIES.

COMPUTER HARDWARE TECHNOLOGY RELATIVE COST EFFECTIVENESS



SOURCE: REPORT OF THE SOFTWARE ENGINEERING GROUP, 1983

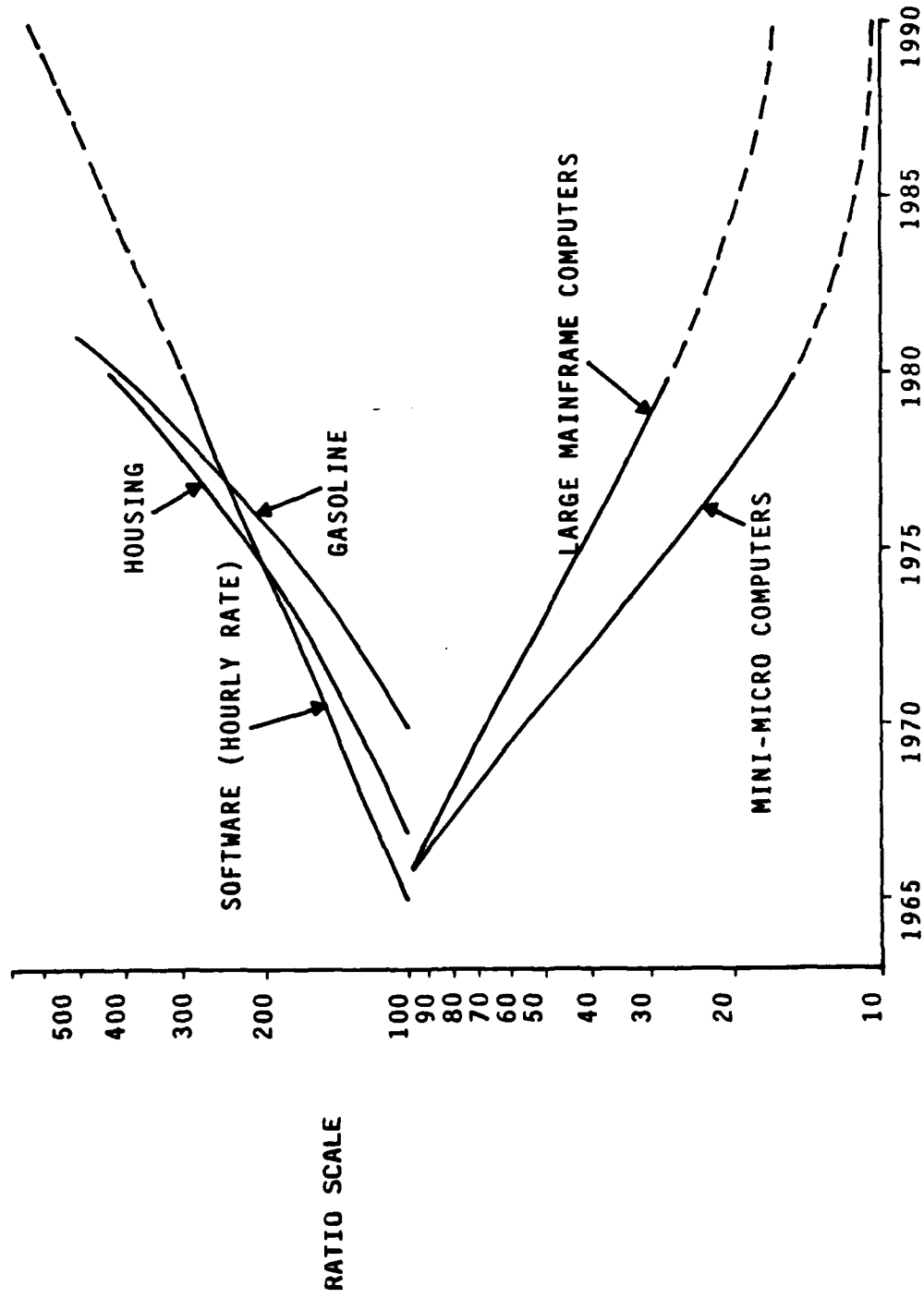
VG 744.1

1-6

INSTRUCTOR NOTES

NOTE THE DIRECTION OF THE VARIOUS COST FACTORS NOT THE ACTUAL VALUES.

SOFTWARE VS. HARDWARE COST TRENDS



SOURCE: REPORT OF THE SOFTWARE ENGINEERING GROUP, 1983

VG 744.1

1-7

INSTRUCTOR NOTES

VG 744.1

2-1

SECTION 2

SOFTWARE ENGINEERING GOALS

VG 744.1

INSTRUCTOR NOTES

THERE EXISTS AN ALMOST LIMITLESS LIST OF POTENTIAL GOALS FOR SOFTWARE ENGINEERING TO ATTAIN. THESE GOALS ARE NOT INDEPENDENT AND NOT ALL ARE APPROPRIATE IN ALL SITUATIONS. WE NEED TO IDENTIFY THE DEPENDENCIES AND RELATIONSHIPS THAT EXIST, UNDERSTAND HOW OVERALL SYSTEM OBJECTIVES INFLUENCE THEIR IMPORTANCE, AND ALSO UNDERSTAND THAT SOME GOALS ARE MUTUALLY EXCLUSIVE OR AT LEAST EXHIBIT TENDENCIES IN THAT DIRECTION.

POTENTIAL SOFTWARE ENGINEERING GOALS

CORRECTNESS	USABILITY	FLEXIBILITY	INTEROPERABILITY
ACCEPTABILITY	OPERABILITY	ADAPTABILITY	COMPLEXITY
COMPLETENESS	HUMAN FACTORS	EXTENSIBILITY	MODULARITY
CONSISTENCY	COMMUNICATIVENESS	ACCESSIBILITY	STRUCTUREDNESS
EXPRESSION	CONVERTIBILITY	EXPANDABILITY	UNIFORMITY
VALIDITY	DOCUMENTATION	AUGMENTABILITY	SELF-CONTAINEDNESS
PERFORMANCE	UNDERSTANDABILITY	MODIFIABILITY	TIME
RELIABILITY	CLARITY	TESTABILITY	
AVAILABILITY	LEGIBILITY	ACCOUNTABILITY	
ACCURACY	SELF-DESCRIPTIVENESS	COST	
ROBUSTNESS	MAINTAINABILITY	PORTABILITY	
PRECISION	STABILITY	TRANSFERABILITY	
TOLERANCE	MANAGEABILITY	COMPATIBILITY	
EFFICIENCY	CONCISENESS	REUSABILITY	
INTEGRITY	REPAIRABILITY	GENERALITY	
SECURITY	SERVICEABILITY	UTILITY	
PRIVACY			

INSTRUCTOR NOTES

THESE OBJECTIVES ARE COMMON TO ALL SYSTEM DEVELOPMENTS. THERE ARE OTHER OBJECTIVES, SUCH AS BUILDING A SECURE (HANDLING OF CLASSIFIED DATA) SYSTEM THAT ONLY APPLY IN SPECIAL SITUATIONS. BUILDING A SYSTEM WITH HUMAN SAFETY CONSIDERATIONS WOULD BE ANOTHER SPECIAL OBJECTIVE. THESE SPECIAL OBJECTIVES IMPLY ADDITIONAL GOALS.

FOR EACH OF THESE FUNDAMENTAL OBJECTIVES ON THE NEXT THREE VIEWGRAPHS WE ARE GOING TO EXAMINE ITS IMPLICATIONS AND THE SPECIFIC SOFTWARE ENGINEERING GOALS THAT SUPPORT THESE OBJECTIVES.

FUNDAMENTAL SOFTWARE ENGINEERING OBJECTIVES

- BUILD A SYSTEM THAT MEETS USER EXPECTATIONS
- BUILD A SYSTEM THAT CAN ACCOMMODATE CHANGE
- BUILD A SYSTEM WITHIN AVAILABLE RESOURCES

INSTRUCTOR NOTES

THIS IS CERTAINLY AN OBJECTIVE THAT ALL SYSTEMS MUST MEET, BUT SURPRISINGLY FEW DO. MOST OF OUR FAILURE IS IN FRONT-END ERRORS OF NOT PROPERLY UNDERSTANDING WHAT THE SYSTEM WILL BE USED FOR AND TRANSLATING THESE NEEDS INTO A SPECIFICATION THAT CAN BE REVIEWED AND APPROVED BY THE END-USER AND THEN USED BY THE IMPLEMENTORS TO GUIDE THE DEVELOPMENT. IN A TYPICAL SYSTEM IMPLEMENTATION THERE ARE SEVERAL LEVELS OF DOCUMENTATION (I.E., SYSTEM SPEC, SOFTWARE SPEC, DESIGN SPEC, ETC.). EACH LEVEL OF DOCUMENTATION IS A NEW OPPORTUNITY FOR THE SYSTEM TO DIVERGE FROM END-USER NEEDS. PRODUCING A CORRECT SYSTEM REQUIRES UNDERSTANDABLE DOCUMENTS THAT ALLOW THE END-USER'S DESCRIPTION OF THE SYSTEM TO BE TRACED THROUGH EACH LEVEL DOWN TO THE IMPLEMENTATION.

RELIABILITY CAN BE ASSURED IF THE REQUIREMENTS AT EACH LEVEL CAN BE TESTED OR VERIFIED. THIS IS MORE THAN JUST BEING ABLE TO BE UNDERSTOOD. TESTABILITY REQUIRES THAT WE BE PRECISE AND SPECIFIC AND DEAL WITH MEASURABLE QUANTITIES. VERIFIABILITY REQUIRES ORGANIZATIONAL AIDS AND APPROACHES THAT ALLOW US TO IDENTIFY INCONSISTENCIES, INCOMPLETENESS AND OTHER ERRORS.

OBJECTIVE #1 - BUILD A SYSTEM THAT MEETS USER EXPECTATIONS

- SYSTEM MUST BE CORRECT AND RELIABLE
- CORRECTNESS MUST BE TRACEABLE FROM SPECIFICATION, TO DESIGN, AND TO CODE
- INSURING CORRECTNESS REQUIRES UNDERSTANDABILITY OF SYSTEM DESCRIPTIONS
- RELIABILITY REQUIRES TESTABILITY AND VERIFIABILITY

INSTRUCTOR NOTES

VERY FEW SOFTWARE SYSTEMS ARE DEVELOPED AND DEPLOYED AND THEN LEFT UNCHANGED FOR THE REMAINDER OF THEIR USEFUL LIFE. MOST OF THE DOLLARS SPENT ON ANY WEAPON SYSTEM ARE SPENT IN MODIFYING IT TO MEET NEW OR CHANGING REQUIREMENTS. EVEN FOR SYSTEMS THAT REMAIN UNCHANGED, THERE IS USUALLY THE NEED TO MAKE SOME MODIFICATIONS TO CORRECT ERRORS FOUND DURING THE INITIAL DEVELOPMENT. TYPICALLY IT COSTS FAR MORE PER LINE OF CODE TO MODIFY SOFTWARE THAN IT DOES FOR THE INITIAL DEVELOPMENT - THIS IS BECAUSE THE SOFTWARE WAS NOT DESIGNED TO BE EASY TO CHANGE.

EACH OF THE FOLLOWING FOUR GOALS, WHILE RELATED, HAS A SLIGHTLY DIFFERENT EMPHASIS:

- MAINTAINABILITY - EMPHASIZES EASE OF FINDING AND FIXING ERRORS - USUALLY SMALL IN SCALE
- MODIFIABILITY - EMPHASIZES THE EASE OF ACCOMMODATING CHANGES OF ANY SIZE OR TYPE
- TRANSPORTABILITY - EMPHASIZES THE EASE OF MOVING THE CODE TO ANOTHER COMPUTER SYSTEM (MACHINE INDEPENDENCE)
- REUSABILITY - EMPHASIZES THE ABILITY TO USE COMPONENTS ON A NEW (AND POSSIBLY DIFFERENT) APPLICATION

WE MUST ALWAYS BE ABLE TO UNDERSTAND SOMETHING BEFORE WE CAN CHANGE IT OR REUSE IT.

OBJECTIVE #2 - BUILD A SYSTEM THAT CAN ACCOMMODATE CHANGE

- OVER 75% OF SOFTWARE DOLLARS SPENT ON MAINTENANCE
- CHANGE OFTEN NECESSARY TO ACHIEVE INITIAL OPERATIONAL CAPABILITY
- CHANGE OCCURS OVER THE LIFE-CYCLE AS ENVIRONMENT CHANGES
- THIS OBJECTIVE RELATED TO SEVERAL GOALS:
 - MAINTAINABILITY
 - MODIFIABILITY
 - TRANSPORTABILITY
 - REUSABILITY
- APPLIES TO DOCUMENTS AND CODE
- IMPLIES AN UNDERLYING GOALS OF UNDERSTANDABILITY

INSTRUCTOR NOTES

EFFICIENCY IS ONE GOAL THAT CAN OFTEN BE IMPROVED LATE IN THE LIFE-CYCLE, PARTICULARLY IF THE SOFTWARE HAS MET ITS MODIFIABILITY GOAL. WE CAN TUNE THE 20% OF SOFTWARE THAT TAKES 80% OF THE RESOURCES DURING THE SYSTEM TEST PHASE.

THE MOST OBVIOUS EXAMPLES OF PRODUCTIVITY ENHANCING TOOLS ARE HIGH-LEVEL LANGUAGES, BUT GOOD PLANNING CAN PROBABLY SAVE MORE MONEY AND SCHEDULE TIME THAN USING A HIGH LEVEL LANGUAGE.

OBJECTIVE #3 - BUILD A SYSTEM WITHIN AVAILABLE RESOURCES

- MUST ADDRESS BOTH RUNTIME AND IMPLEMENTATION RESOURCES
- RUNTIME RESOURCE LIMITATIONS REQUIRE EFFICIENCY
 - BOTH STORAGE AND CPU
- IMPLEMENTATION CONSTRAINTS REQUIRE PRODUCTIVITY
- EFFICIENCY CAN OFTEN BE IMPROVED LATE IN DEVELOPMENT
 - 20% OF CODE USES 80% OF CPU TIME
- PRODUCTIVITY IS ENHANCED BY METHODS, TOOLS, PLANNING

INSTRUCTOR NOTES

AT THE START OF A SYSTEM DEVELOPMENT IT IS IMPORTANT TO ESTABLISH THE RELATIVE IMPORTANCE OF THE SYSTEM'S OBJECTIVES - IS THE ABILITY TO RESPOND TO CHANGE MORE IMPORTANT THAN GETTING IT RIGHT IN THE FIRST PLACE? THE RELATIVE IMPORTANCE OF OBJECTIVES ALLOWS US TO TRADE-OFF THE INDIVIDUAL GOALS. FORTUNATELY MANY GOALS ARE MUTUALLY COMPATIBLE. THE TOUGHEST TRADE-OFF IS OFTEN WITH EFFICIENCY. AS A GENERAL RULE OF THUMB WE SHOULD NOT BE OVERLY CONCERNED WITH EFFICIENCY EARLY ON. IT IS HARD EARLY ON TO PREDICT SYSTEM BOTTLENECKS AND MAJOR RESOURCE USERS. BY STRESSING EFFICIENCY TOO EARLY WE LOSE UNDERSTANDABILITY AND MODIFIABILITY AND OFTEN DEGRADE EFFICIENCY BECAUSE WE LOSE OUR ABILITY TO TUNE THE SYSTEM.

SOFTWARE ENGINEERING GOAL CONFLICTS

- SOME GOALS ARE MUTUALLY COMPATIBLE
 - UNDERSTANDABILITY, MODIFIABILITY
- CONFLICTING GOALS REQUIRE TRADE-OFFS
 - EFFICIENCY VS. UNDERSTANDABILITY
 - PRODUCTIVITY VS. EFFICIENCY
 - RELIABILITY VS. PRODUCTIVITY
- TRADE-OFFS BETWEEN GOALS BASED ON SYSTEM OBJECTIVES
- TRADE-OFFS MUST BE AGREED TO BY ALL PARTICIPANTS
 - USER, MANAGER, IMPLEMENTORS

INSTRUCTOR NOTES

II. S.E. GOALS AND PRINCIPLES.

THIS SECTION SHARPENS OUR FOCUS ON SOFTWARE ENGINEERING BY JUST DISCUSSING OBJECTIVES AND GOALS AND THEN BY INTRODUCING THE PRINCIPLES OF SOFTWARE ENGINEERING THAT ALLOW US TO ACHIEVE THESE GOALS.

IN SECTION 3 WE WILL REVIEW VARIOUS METHODS AND TOOLS FOR ACHIEVING THESE GOALS. WHEN WE DISCUSS THESE METHODS AND TOOLS WE WILL IDENTIFY THE PRINCIPLES THAT THEY ARE BASED ON.

ALLOW ABOUT 45 MINUTES TO 1 HOUR FOR THIS SECTION.

PART II

SOFTWARE ENGINEERING GOALS AND PRINCIPLES

VG 744.1

INSTRUCTOR NOTES

THIS SECTION DEFINES SOME BASIC ENGINEERING PRINCIPLES THAT WORK TOWARD ACHIEVING
OUR FUNDAMENTAL GOALS.

SECTION 3

SOFTWARE ENGINEERING PRINCIPLES

VG 744.1

INSTRUCTOR NOTES

NOT ALL OF THESE PRINCIPLES ADDRESS THE ENTIRE LIFE-CYCLE; SOME ARE LIMITED TO FRONT-END CONSIDERATIONS, AND OTHERS APPLY ONLY TO DESIGN AND IMPLEMENTATION.

SOME OF THESE PRINCIPLES ARE WIDELY ACCEPTED AND USED; OTHERS ARE STILL CONTROVERSIAL AND, WHILE INTUITIVELY APPEALING, HAVE NOT BEEN WIDELY USED AND PROVEN "UNDER FIRE."

AS WE REVIEW THE SPECIFIC TOOLS AND METHODS DURING LATER SECTIONS OF THIS COURSE, WE WILL IDENTIFY AND DISCUSS THE SPECIFIC PRINCIPLES UPON WHICH EACH OF THESE TOOLS OR METHODS IS BASED.

SOFTWARE ENGINEERING PRINCIPLES

- ARE THE TECHNIQUES FOR ATTAINING SOFTWARE GOALS
- APPLY TO DIFFERENT PHASES OF LIFE-CYCLE
- REPRESENT AN EVOLVING CONSENSUS
 - NOT ALL PROVEN OR ACCEPTED THROUGHOUT INDUSTRY
- WILL RELATE TO SPECIFIC TOOLS AND METHODS DURING LATER SECTIONS OF THE COURSE

INSTRUCTOR NOTES

GO THROUGH EACH PRINCIPLE AND ITS BRIEF DEFINITION. EACH IN TURN WILL BE COVERED
IN MORE DETAIL.

SUPPRESSION OF DETAIL MEANS THAT YOU IDENTIFY AND OMIT -- FOR THE MOMENT --
NONESSENTIAL DETAILS.

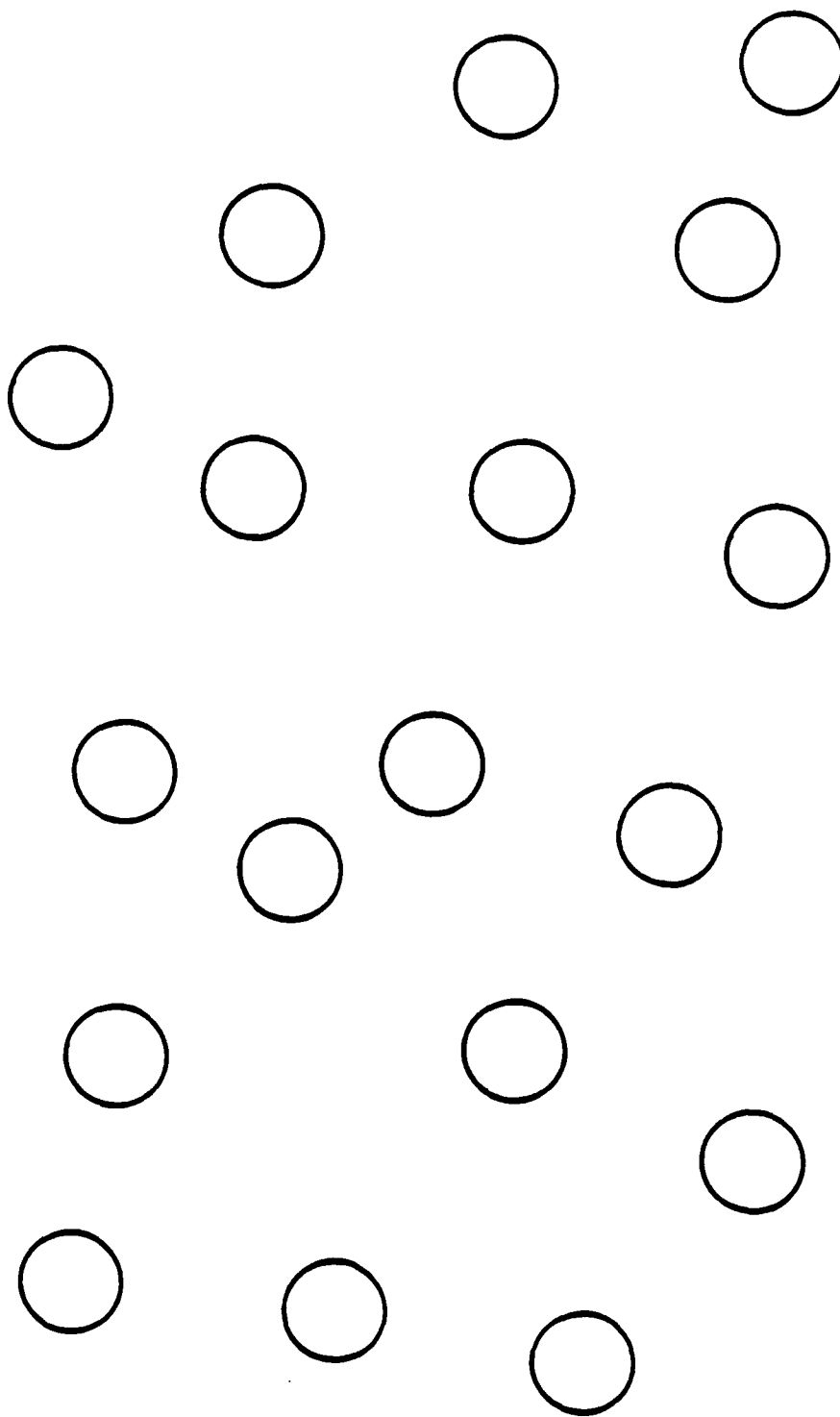
●	STRUCTURING	--	RELATING PARTS AND SUBPARTS
●	MODULARITY	--	RATIONAL STRUCTURING
●	ABSTRACTION	--	SUPPRESSION OF DETAIL
●	HIDING	--	MAKING SOME OF THE "HOWS" UNKNOWN TO OTHER PARTS OF THE SYSTEM
●	SEPARATION OF CONCERNS	--	PHYSICALLY COLLECTING RELATED THINGS AND SEPARATING UNRELATED THINGS
●	UNIFORMITY	--	CONSISTENCY
●	FORMALISM	--	AVOID PROSE, BE PRECISE AND CONCISE

INSTRUCTOR NOTES

INSTRUCTOR:

HOLD EACH SLIDE UP FOR EXACTLY 2 SECONDS. HAVE CLASS TRY TO COUNT CIRCLES,
WRITING DOWN THEIR ANSWERS. EVERYONE SHOULD COME CLOSER TO THE CORRECT ANSWER
(18) ON THE SECOND TRY, BECAUSE SHADING ENCODED INDIVIDUAL CIRCLES INTO COUNTABLE
GROUPS.

STRUCTURING



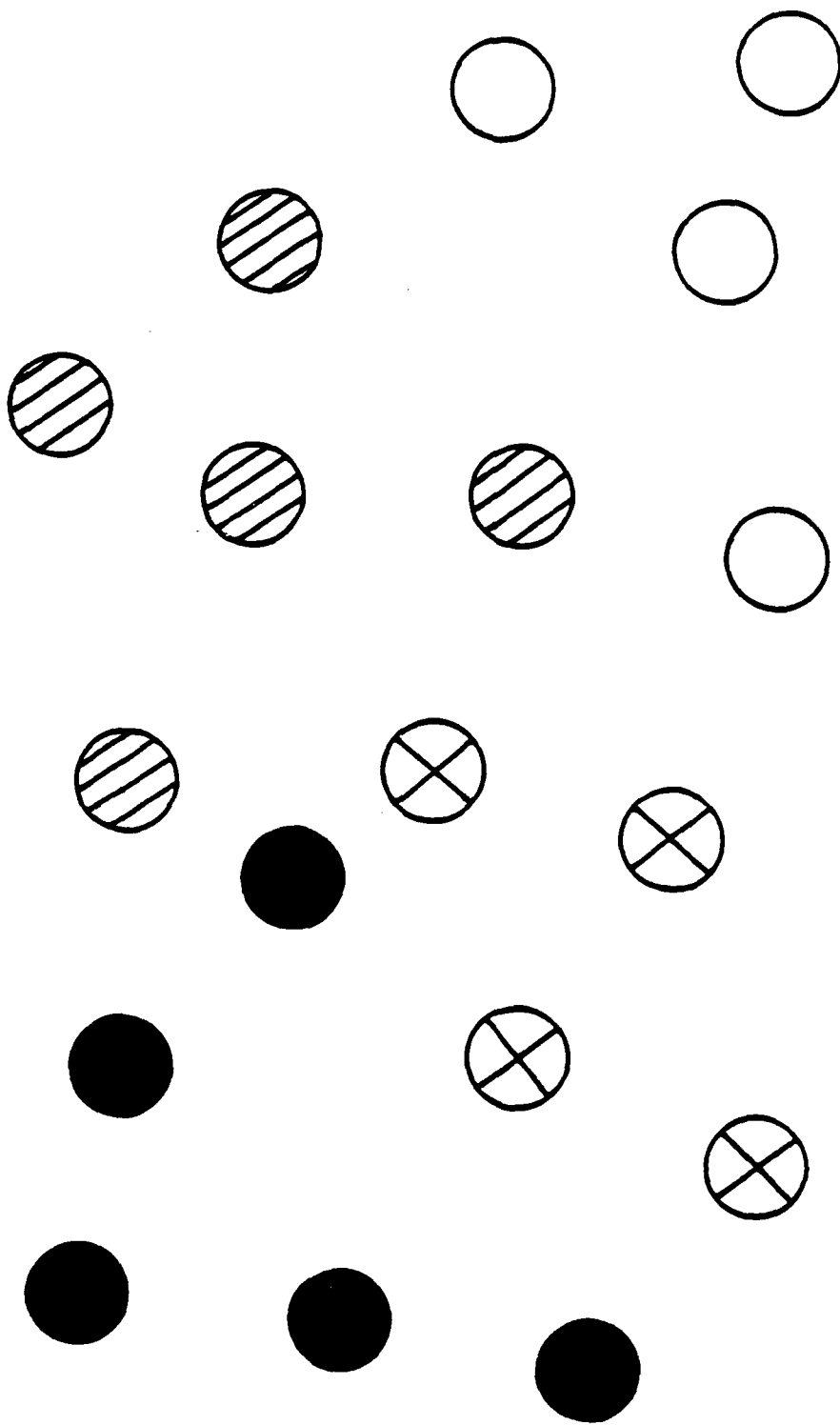
INSTRUCTOR NOTES

REFER TO PREVIOUS INSTRUCTOR NOTE.

VG 744.1

3-41

NOW, COUNT THE CIRCLES



INSTRUCTOR NOTES

AS THE PREVIOUS EXAMPLE POINTED OUT, INTRODUCING SOME STRUCTURING -- IN THIS CASE DIVIDING THE OBJECTS INTO CLASSES -- HAS ENHANCED OUR UNDERSTANDABILITY - AT LEAST AS FAR AS COUNTING THE OBJECTS GOES.

THERE ARE VARIOUS TYPES OF STRUCTURING WHICH ARE APPLICABLE IN DIFFERENT SITUATIONS. SOME ORGANIZATIONAL STRUCTURING IS USED IN DEVELOPING DOCUMENTS - HIERARCHY AND PARTITIONING INTO CLASSES OR CATEGORIES ARE EXAMPLES. OTHER FORMS OF STRUCTURING ARE USED IN IMPLEMENTATIONS (DESIGN AND CODE) EXAMPLES ARE NETWORKS, TASKING STRUCTURES, AND STRUCTURED LANGUAGES.

ANY STRUCTURING MECHANISM WORKS BY ESTABLISHING RULES AND CONVENTIONS. THESE CONVENTIONS USUALLY RESULT IN SOME LOSS OF FLEXIBILITY AND GENERALITY. WE DO PAY SOME PRICE FOR THE INCREASE IN UNDERSTANDABILITY - EVERY ONCE IN A WHILE THE STRUCTURE GETS IN THE WAY.

STRUCTURING

- STRUCTURING REDUCES COMPLEXITY, AIDS UNDERSTANDING
- STRUCTURING COMES IN DIFFERENT FORMS
 - HIERARCHICAL
 - NETWORK
 - PROCESS/TASKING
 - "STRUCTURED PROGRAMMING LANGUAGES"
 - CLASSES, TYPES, CATEGORIES
- ESTABLISHES RULES AND CONVENTIONS
 - REDUCES FLEXIBILITY

INSTRUCTOR NOTES

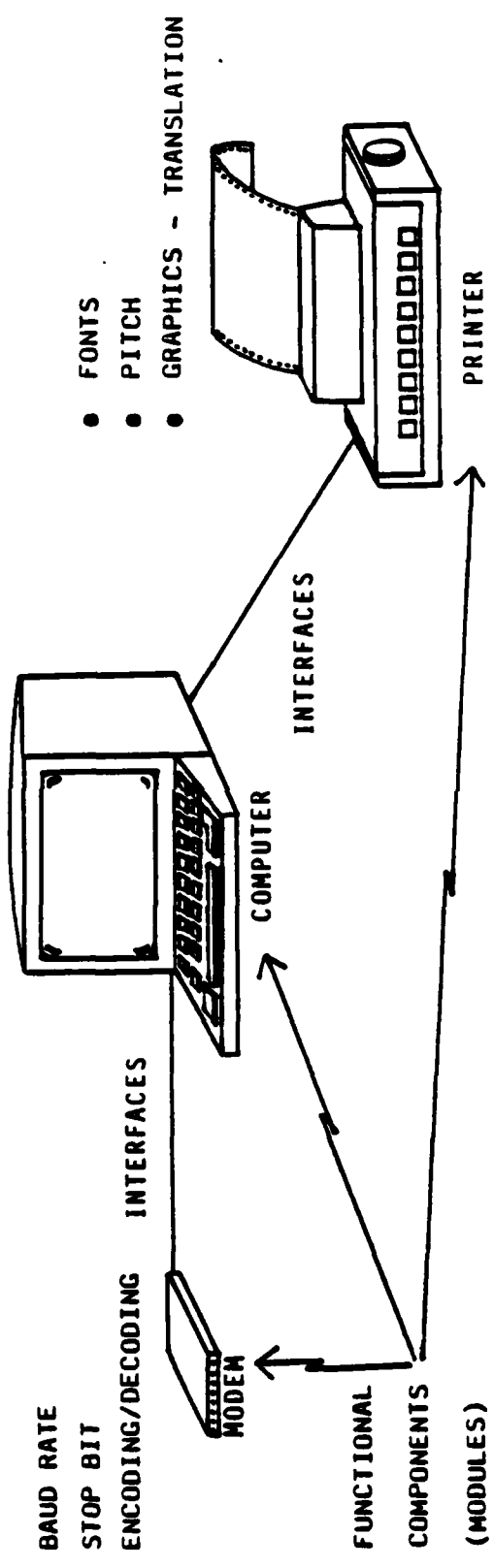
STRUCTURING BREAKS A SYSTEM INTO PARTS, BUT MODULARITY MAKES THE STRUCTURING HAVE PURPOSE.

MODULARITY GOES HAND-IN-HAND WITH THE DEVELOPMENT OF EXPLICIT INTERFACES BETWEEN THE MODULES. THESE EXPLICIT INTERFACES ALLOW THE SEPARATE MODULES TO BE DEVELOPED INDEPENDENTLY, POSSIBLY IN PARALLEL.

MODULARITY

A HARDWARE EXAMPLE:

- LINE OVERFLOW
- PIXELS
- LINE GRAPHICS
- TEXT GRAPHICS



- EACH MODULE HAS A SPECIFIC FUNCTION WITH WELL-DEFINED INTERFACES

INSTRUCTOR NOTES

ABSTRACTION AIDS UNDERSTANDING BY SUPPRESSING UNNECESSARY DETAIL WHICH ALLOWS ESSENTIAL CHARACTERISTICS TO STAND OUT. FOR THE HIGH LEVEL LANGUAGE TO WORK, ALL THE OTHER LEVELS MUST WORK - WE TRUST THIS TO BE TRUE, BUT WE DON'T HAVE TO KNOW ANYTHING ABOUT THESE OTHER LEVELS.

THIS SAME ABSTRACTION OCCURS IN MANY OTHER SITUATIONS - WE DON'T NEED TO KNOW HOW A STEERING WHEEL WORKS TO DRIVE A CAR OR A BOAT. WE LEARN TO ABSTRACT THE DETAILS AND ASSOCIATE CLOCKWISE TURNING OF THE WHEEL WITH A RIGHT TURN.

ABSTRACTION

procedure Sort is

...

begin

...

end Sort;

- HIGH LEVEL LANGUAGE LEVEL



- MACHINE LANGUAGE LEVEL



- REGISTER TRANSFER LEVEL



- CIRCUIT LEVEL



- PHYSICS LEVEL

• EXPRESSES A PARTICULAR USAGE VIEWPOINT

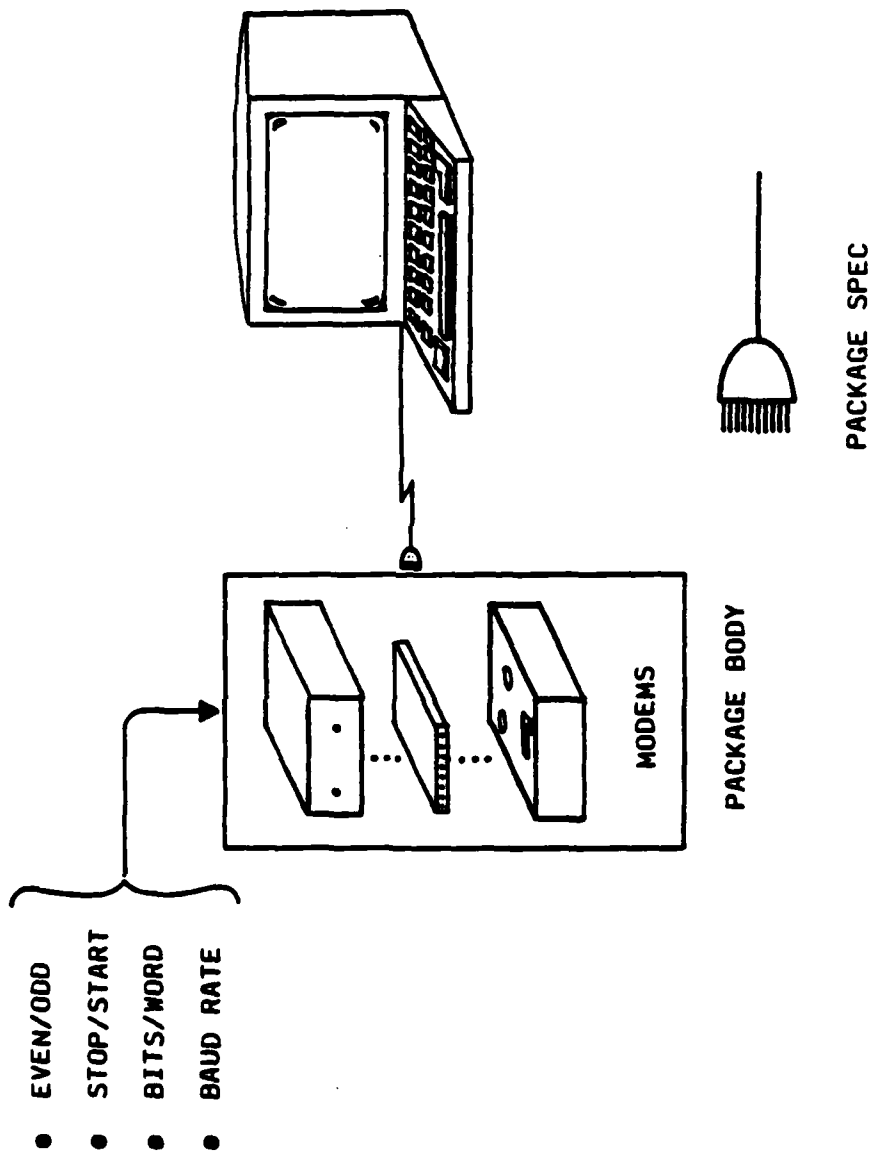
• SUPPRESSES UNRELATED DETAIL

INSTRUCTOR NOTES

STRESS THAT HIDING IS A PRINCIPLE THAT WE ALL USE INTUITIVELY. HARDWARE ENGINEERS HAVE DONE A MUCH BETTER JOB OF USING IT THAN HAVE SOFTWARE ENGINEERS. WHENEVER WE PRODUCE MODULES WITH WELL DEFINED INTERFACES, WE ARE USING HIDING - WE HAVE HIDDEN EVERYTHING THAT IS NOT EXPLICITLY INCLUDED IN THE INTERFACE.

THE MODEM EXAMPLE IS A CLASSIC HARDWARE EXAMPLE OF HIDING. WE KNOW NOTHING ABOUT THE INTERNAL DESIGN OF THE MODEM - VERY DIFFERENT IMPLEMENTATION APPROACHES CAN BE USED AND WE STILL CAN REPLACE ONE MODEM WITH A DIFFERENT ONE BECAUSE THEY BOTH MEET THE SAME INTERFACE - SHOWN HERE GRAPHICALLY, AS MEETING THE SAME PACKAGE SPEC. THE PACKAGE BODY - THE IMPLEMENTATION - CAN BE COMPLETELY DIFFERENT.

HIDING



• THE INTERFACE "HIDES" THE BRAND OR TYPE OF MODEM.

INSTRUCTOR NOTES

THESE PRINCIPLES ARE NOT USUALLY USED INDEPENDENTLY BUT IN FACT ARE RELATED AND USED TOGETHER IN MODERN SE METHODOLOGIES. MODULARITY, ABSTRACTION, AND HIDING ARE CLOSELY RELATED AND FORM THE BASIS FOR THE DESIGN APPROACH EMBODIED IN THE SOFTWARE COST PRODUCTION PROJECT. THIS PROJECT WAS LED BY AND IS BASED ON EARLIER RESEARCH BY DAVID PARNAS. WE WILL DISCUSS THIS METHODOLOGY IN THE METHODS AND TOOLS PART OF THE COURSE. (REVIEW THAT MATERIAL FOR A BETTER UNDERSTANDING.) BRIEFLY THE SCRP USES AS THEIR MODULARIZATION CRITERIA EXPECTED CHANGE. THE ANALYSTS AND DESIGNERS ATTEMPT TO PREDICT WHAT ASPECTS OF THE SYSTEM ARE LIKELY TO CHANGE - THIS IS BASED ON PAST EXPERIENCE AND SOME KNOWLEDGE OF FUTURE PLANS. THINGS LIKE CHANGES TO SENSORS AND EXTERNAL DEVICES ARE EXAMPLES OF LIKELY CHANGES. EACH MODULE IN THE SYSTEM IS DESIGNED AROUND EACH INDEPENDENT CHANGE OR GROUP OF RELATED CHANGES. THE MODULE "HIDES" THE IMPACT OF THE CHANGE BY PRESENTING AN "ABSTRACT" INTERFACE TO THE REST OF THE SYSTEM WHICH CAN STILL BE MET EVEN IF THE EXPECTED CHANGE TAKES PLACE. IN OTHER WORDS, THE INTERFACE ONLY PRESENTS THE ESSENTIAL CHARACTERISTICS OF A SENSOR THAT IS LIKELY TO BE REPLACED. IT HIDES ANY ARBITRARY DETAILS THAT MAY VARY BETWEEN DIFFERENT MAKES OR VERSIONS.

MODULARITY / ABSTRACTION / HIDING

- USED TOGETHER TO DEFINE ABSTRACT MODULE SPECIFICATIONS
- KEY COMPONENT OF THE SOFTWARE COST REDUCTION PROJECT (SCRIP)
(PARNAS) METHODOLOGY
- MODULARIZATIONS STRATEGY BASED ON EXPECTED CHANGE
- A MODULE HIDES EFFECTS OF INDEPENDENT CHANGE
- (ABSTRACT) MODULE INTERFACE MAKES VISIBLE ONLY THOSE ASPECTS OF A
MODULE THAT ARE UNLIKELY TO CHANGE

INSTRUCTOR NOTES

"SEPARATION OF CONCERNS" IS ALSO AN IMPORTANT ASPECT OF THE SCRP METHODOLOGY. THE TERM WAS COINED BY DIJKSTRA IN HIS BOOK, A DISCIPLINE OF PROGRAMMING. THE GOAL IS TO INSURE THAT THE IMPACT OF CHANGE IS MINIMIZED AND THAT WE ALWAYS KNOW WHERE TO GO TO FIND THE ANSWER TO QUESTIONS. FOR EXAMPLE, HARDWARE INTERFACES ARE DESCRIBED WITHOUT MAKING ASSUMPTIONS ABOUT THE FUNCTIONALITY OF THE SYSTEM; THE HARDWARE INTERFACE DESCRIPTION WOULD REMAIN UNCHANGED EVEN IF THE FUNCTION CHANGED. THIS SEPARATION OCCURS AT MULTIPLE LEVELS; AT A HIGH LEVEL WE SEPARATE FUNCTIONS FROM BEHAVIOR FROM HARDWARE INTERFACE (I/O MAPPING) FROM TIMING, ETC. WITHIN EACH SUBJECT WE USE A STANDARD TEMPLATE TO FURTHER SEPARATE THE DETAILS IN EACH SUBJECT AREA. ALLOW FOR EASE OF ACCESSING INFORMATION, IDENTIFIES AREAS NEEDING CHANGE, AND CREATES A "FILL-IN-THE-BLANKS" APPROACH TO WRITING SPECS ONCE THE TEMPLATES HAVE BEEN DEFINED. THIS LEADS TO AN UNSTATED PRINCIPLE "STATE QUESTIONS BEFORE TRYING TO ANSWER THEM."

SEPARATION OF CONCERNS

- ALSO KNOWN AS LOCALIZATION
- GOALS
 - REDUCE THE IMPACT OF CHANGE
 - ALWAYS KNOW WHERE TO FIND ANSWERS TO QUESTIONS
- SEPARATE (FOR EXAMPLE)
 - FUNCTIONALITY FROM BEHAVIOR
 - INPUT/OUTPUT BIT REPRESENTATIONS FROM LOGICAL MEANING
 - TIMING
 - ACCURACY
 - EXPECTED CHANGES
 - INPUT/OUTPUT MAPPING FROM FUNCTIONALITY
- USE STANDARD TEMPLATES FOR ADDITIONAL LEVEL OF SEPARATION

INSTRUCTOR NOTES

THIS IS AN EXAMPLE OF A BLANK SCRP INPUT DATA FORM. POINT OUT THAT AS AN EXAMPLE OF HIGH LEVEL SEPARATIONS OF CONCERNS THERE IS NO MENTION OF HOW THIS INPUT IS USED - WHAT FUNCTIONS DEPEND ON IT, OR HOW IT AFFECTS BEHAVIOR.

AT A LOWER LEVEL IT PROVIDES A SET OF QUESTIONS TO ASK AND ANSWER ABOUT EACH HARDWARE INTERFACE. AT THIS LOW LEVEL THE "ESSENTIAL CHARACTERISTICS" SUCH AS UNITS AND RANGE OR THE POSSIBLE VALUES (ON OR OFF) FOR VALUE ENCODING ARE SEPARATED FROM THE DATA REPRESENTATION (I.E., THE MAPPING TO BITS) AND THE INSTRUCTION SEQUENCE NEEDED TO ACQUIRE THE DATA.

I/O DATA TEMPLATE EXAMPLE

INPUT DATA ITEM:

ACRONYM:

HARDWARE:

DESCRIPTION:

CHARACTERISTICS OF VALUES:

UNITS:

RANGE:

ACCURACY:
RESOLUTION:

VALUE ENCODING:

OR

INSTRUCTION SEQUENCE:

DATA REPRESENTATION:

TIMING CHARACTERISTICS:

INSTRUCTOR NOTES

THIS PRINCIPLE IS MORE GENERAL AND HIGH LEVEL THAN THE OTHERS. SPECIFIC EXAMPLES ARE SUCH THINGS AS NAMING CONVENTIONS WHICH SUPPORT UNDERSTANDABILITY AND CONFIGURATION MANAGEMENT. SPECIFIC Ada EXAMPLES OF CONSTRUCTS WHICH SUPPORT UNIFORMITY ARE DATA TYPES, OVERLOADING OF OPERATORS AND GENERICS (WHICH ALLOWS ALL SIMILAR OPERATIONS TO HAVE THEIR SIMILARITIES MADE EXPLICIT).

DEFINING AND USING A LIMITED TASKING MODEL, AS Ada HAS DONE, IS ANOTHER EXAMPLE OF UNIFORMITY.

THE USE OF STANDARD TEMPLATES AS DISCUSSED UNDER SEPARATION OF CONCERNS IS ANOTHER EXAMPLE OF UNIFORMITY.

UNIFORMITY

- USE OF DATA TYPES, OVERLOADING, GENERICS
- CONSISTENCY OF TERMINOLOGY (NAMES)
- USE OF LIMITED DESIGN STRUCTURES - TASKING MODEL
- USE OF STANDARD TEMPLATES AND TABLES

INSTRUCTOR NOTES

THE PRINCIPLE OF FORMALISM COMES IN SEVERAL FLAVORS. TO SOME IT MEANS AN APPROACH BASED ON MATHEMATICAL TECHNIQUES THAT WILL LEAD TO AUTOMATED GENERATION OF CODE AND/OR AUTOMATIC VERIFICATION OF CORRECTNESS. TO OTHERS IT IS REPRESENTED BY CONSISTENT USE OF LABELS AND TEMPLATES WITH SOME RULES ON ALLOWED OPERATORS AND POSSIBLE NEW SYMBOLOGY (I.E., USE OF SPECIAL CHARACTERS). SOME TECHNIQUES USE A FORMAL LANGUAGE (MUCH LIKE A PROGRAMMING LANGUAGE) TO EXPRESS THINGS LIKE REQUIREMENTS - PSC/PSM IS A GOOD EXAMPLE OF THIS TYPE OF FORMALISM.

THE GOAL OF FORMALISM IS TO ALLOW DETAILED, PRECISE, AND CONCISE SPECIFICATION TO BE DEVELOPED THAT CAN BE REVIEWED FOR COMPLETENESS AND CORRECTNESS.

IN GENERAL THESE FORMALISMS ARE A REPLACEMENT FOR PROSE WHICH IS RECOGNIZED TO BE AMBIGUOUS AND OFTEN REDUNDANT.

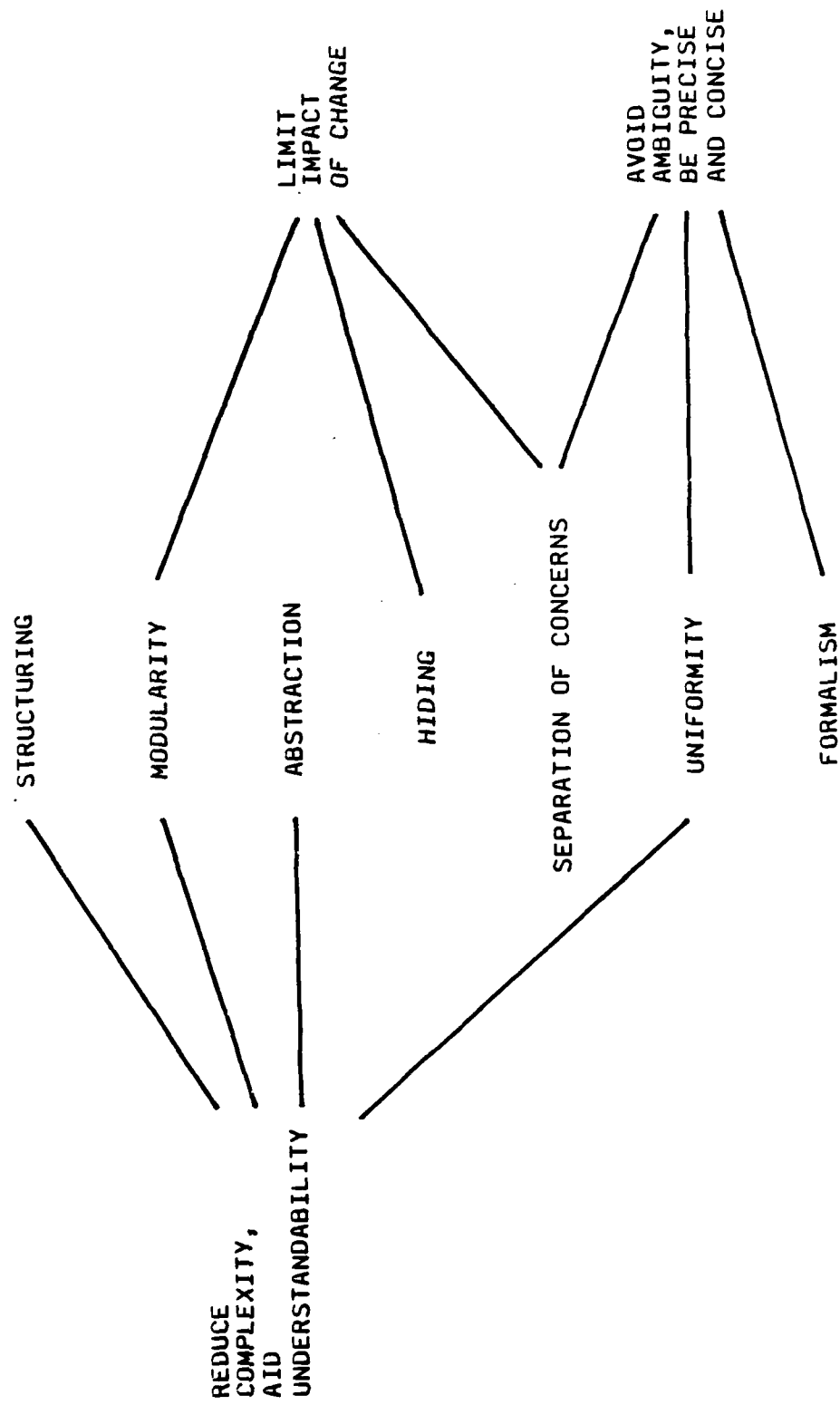
FORMALISM

- SUPPORTS PRECISION, CONCISENESS, COMPLETENESS
- MAY BE BASED ON
 - MATHEMATICS
 - FORMAL LANGUAGE
- MAY USE TABLES, TEMPLATES
- IMPLIES A REDUCTION IN STANDARD PROSE
- LEADS TO
 - MACHINE PROCESSING OF SPECIFICATION
 - MANUAL OR AUTOMATED REVIEW AND VERIFICATION
 - PROOFS OF CORRECTNESS

INSTRUCTOR NOTES

THIS IS JUST A SUMMARY OF THESE PRINCIPLES WITH THE MOST IMPORTANT FOCUS OF THESE PRINCIPLES (IN TERMS OF OUR GOALS AND OBJECTIVES) POINTED OUT.

SOFTWARE ENGINEERING PRINCIPLES



INSTRUCTOR NOTES

- IN THIS SECTION WE ARE GOING TO COVER

1) SOFTWARE LIFE-CYCLE

2) METHODS AND TOOLS FOR EACH PHASE OF LIFE-CYCLE

3) TESTING

4) SOFTWARE MANAGEMENT

- SUBSECTION 2 OF THIS SECTION IS BY FAR THE LARGEST.

PART III

ACHIEVING SOFTWARE ENGINEERING GOALS

VG 744.1

INSTRUCTOR NOTES

THEME: THE SOFTWARE DEVELOPMENT PROCESS CAN BE EXPRESSED IN TERMS OF A SEQUENCE OF PHASES.

PURPOSE: TO PROVIDE THE FRAMEWORK IN WHICH THE METHODOLOGIES WILL BE PRESENTED IN.

REFERENCE: "A SOFTWARE ENGINEERING ENVIRONMENT FOR THE NAVY", REPORT OF THE NAVMAT SOFTWARE ENGINEERING ENVIRONMENT WORKING GROUP, MARCH 1983.

SECTION 4

THE SOFTWARE LIFE CYCLE

VG 744.1

INSTRUCTOR NOTES

THE CONCEPT OF A SOFTWARE DEVELOPMENT LIFE-CYCLE HAS EVOLVED OVER THE LAST 20 YEARS. IT IS VIEWED DIFFERENTLY BY DIFFERENT INDIVIDUALS, ORGANIZATIONS, AND INDUSTRIES, OFTEN BECAUSE THE TYPES OF PRODUCTS THAT ARE PRODUCED ARE RADICALLY DIFFERENT OR BECAUSE OF THE IMPORTANCE OR NON-IMPORTANCE OF THE SOFTWARE USED IN THE PRODUCT DEVELOPMENT.

SOFTWARE LIFE CYCLE

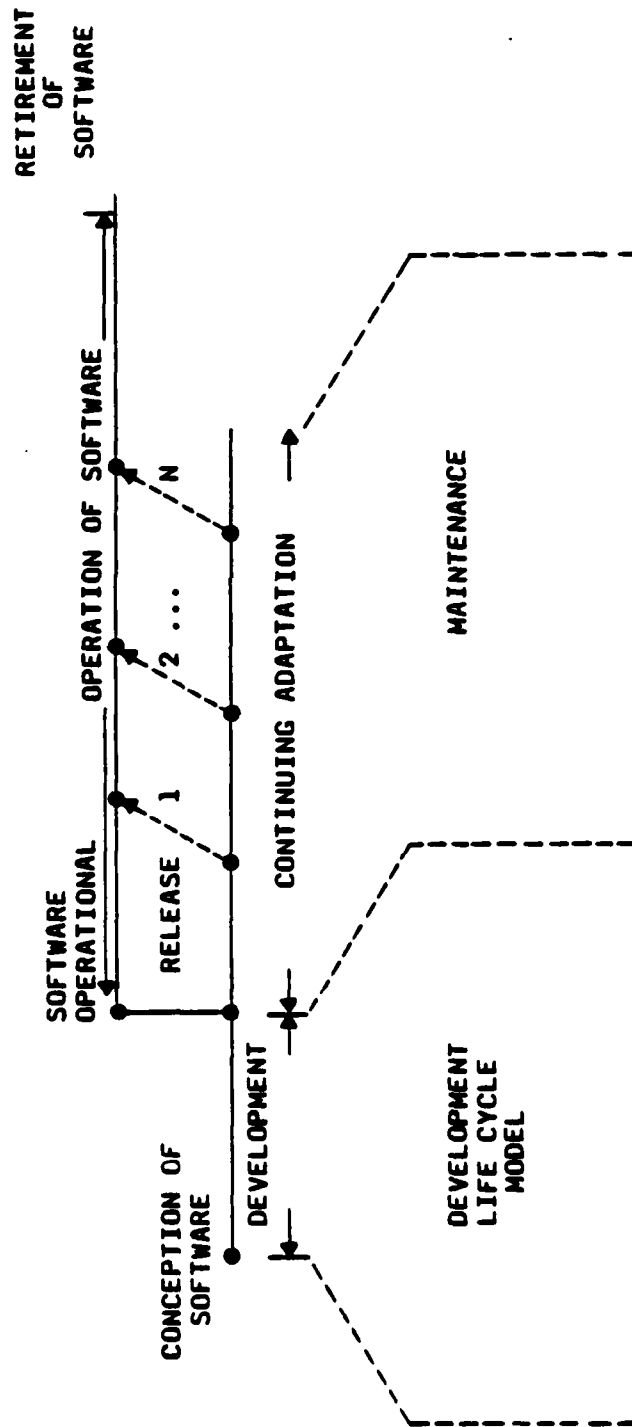
- LIFE CYCLE ORGANIZES THE ACTIVITIES OF BUILDING SOFTWARE
- SOFTWARE DEVELOPMENT IS BROKEN INTO PHASES
- LIFE CYCLE SUMMARIZES SOFTWARE DEVELOPMENT
- THE "LIFE CYCLE" IS NOT STANDARD IN THE INDUSTRY

INSTRUCTOR NOTES

THE TERM "LIFE-CYCLE" PROVIDES US WITH A WAY OF LOOKING AT THE TASKS OF BUILDING A SYSTEM. IT IS NEEDED SO THAT WE CAN TALK ABOUT ASPECTS OF SOFTWARE DEVELOPMENT IN A COMMON LANGUAGE, AND SO THAT WE CAN DETERMINE THE EFFECTS OF CHANGES TO OUR DEVELOPMENT PROCESS.

NOTE THAT SOFTWARE HAS A LIFE OF ITS OWN THAT EXTENDS WELL BEYOND THE CODING OF AN APPLICATION.

THE LIFE OF SOFTWARE



SOURCE: SEEWG REPORT, 1983

VG 744.1

4-2

INSTRUCTOR NOTES

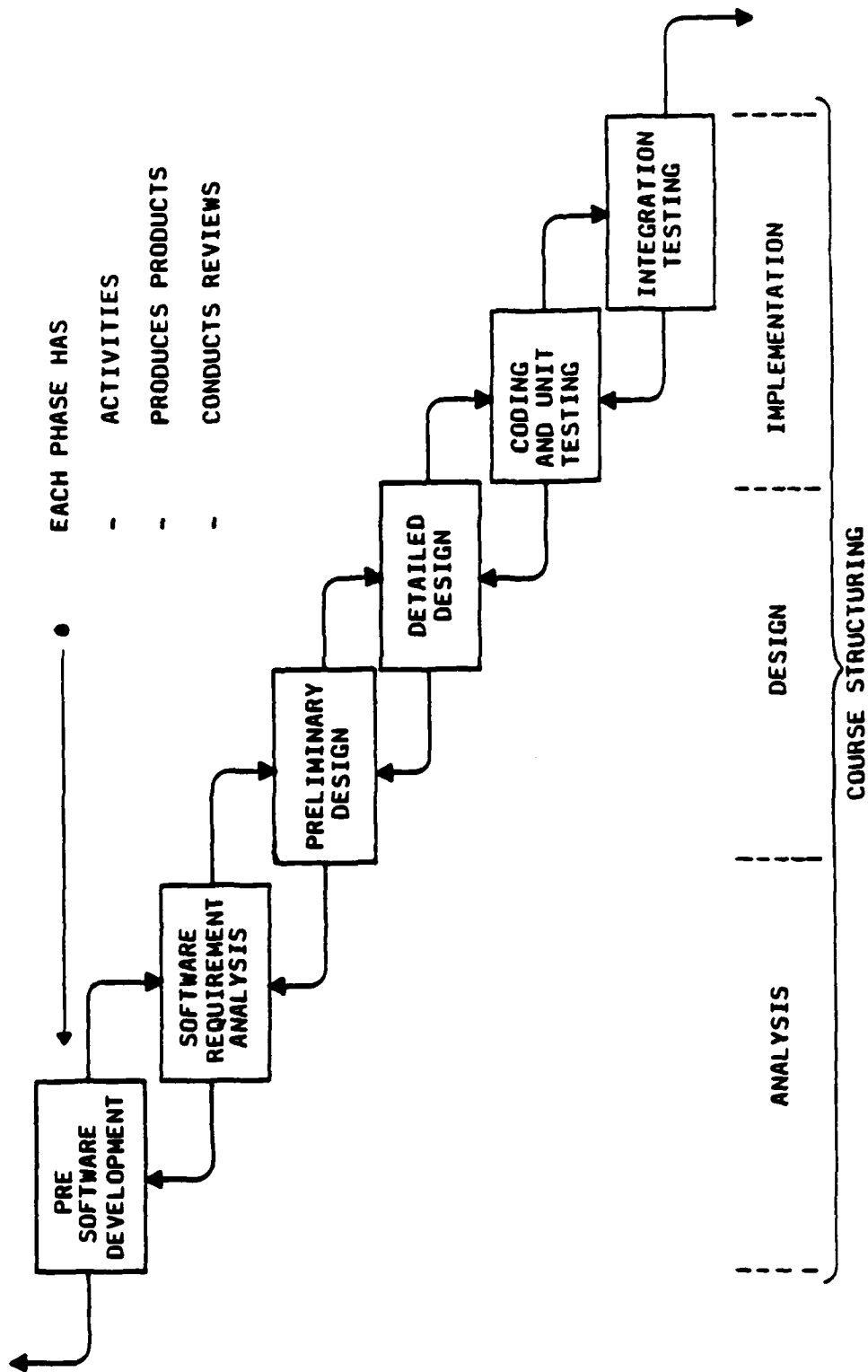
THIS IS REALLY DIVIDING UP THE RESPONSIBILITY OF THE SYSTEM DEVELOPMENT AMONG DIFFERENT GROUPS TO MAKE THE DEVELOPMENT PROCESS TRACTABLE. SOME DIVIDE IT UP INTO 6 PIECES, SOME INTO 4 OR 8 PIECES.

NOTE THE MAPPING TO THE COURSE STRUCTURE. THIS PICTURE IS A MORE CLASSICAL VIEW OF LIFE-CYCLE. THIS MAY (AND PROBABLY WILL) DIFFER FROM THE LIFE-CYCLE MODEL UNDERSTOOD BY EACH STUDENT. THIS IS OFTEN CALLED THE "WATERFALL" MODEL. OUT OF EACH PHASE A DELIVERABLE IS USUALLY PRODUCED, WHICH IS USED AS INPUT INTO THE NEXT PHASE. A REAL SYSTEM OR SOFTWARE DEVELOPMENT IS NEVER THIS CLEAN. THERE EXIST FEEDBACK LOOPS BETWEEN PHASES, THE DELIVERABLES NEED REVIEWS AND CHANGES BEFORE ACCEPTANCE, THE DESIGN PHASE MAY BE SUB-DIVIDED INTO MANY DESIGN PHASES OF INCREASED DETAIL, ETC.

THE KEY WORDS ARE:

- ANALYSIS - THE "WHAT" PHASE
- DESIGN - THE "HOW" PHASE
- IMPLEMENTATION - THE "BUILD" PHASE

DEVELOPMENT LIFE CYCLE MODEL



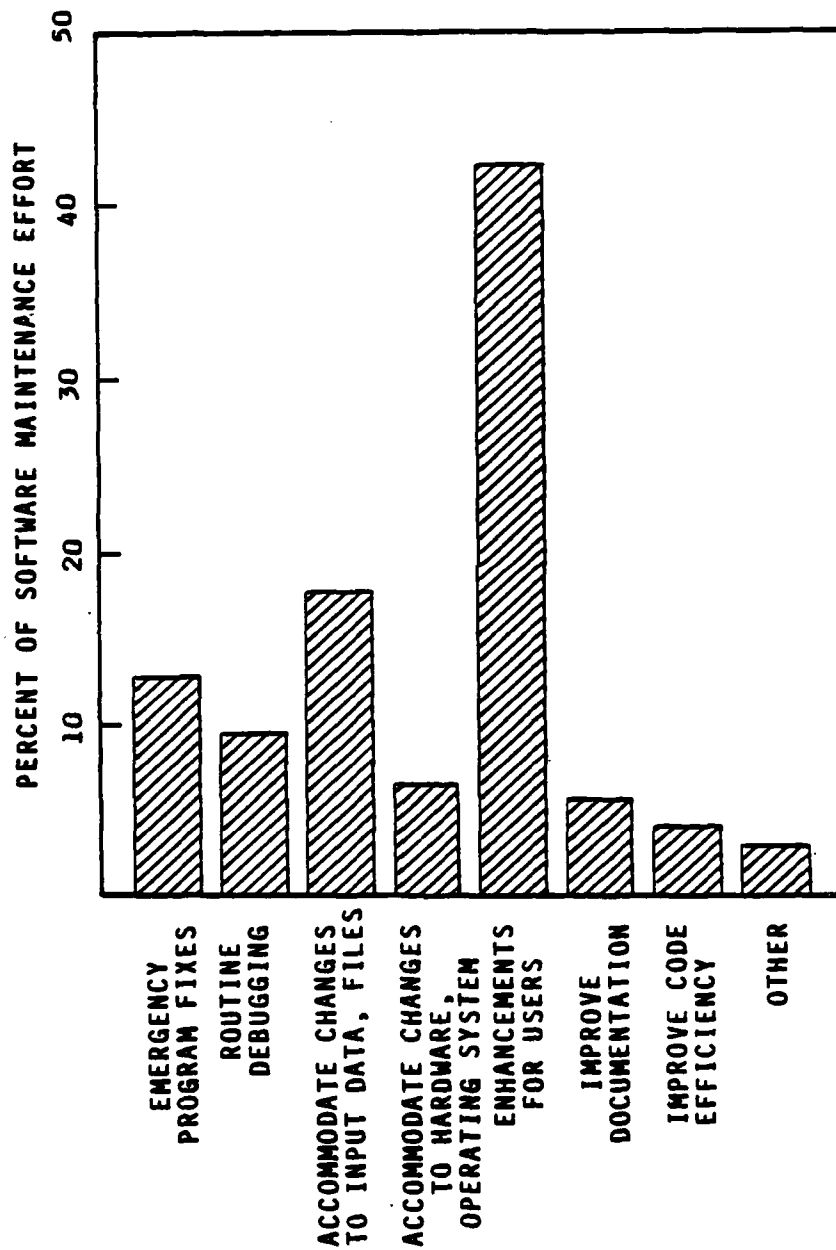
INSTRUCTOR NOTES

MAINTENANCE ISN'T THE BEST TERM FOR WHAT GOES ON AFTER DEVELOPMENT, SOME PEOPLE USE THE
WORD

"EVOLUTION"

DUE TO THE EMPHASIS ON ENHANCEMENTS ETC.

SOFTWARE MAINTENANCE ACTIVITIES



SOURCE: BOEHM, SOFTWARE ENGINEERING ECONOMICS, 1983

INSTRUCTOR NOTES

POINT OUT THAT NO MODEL CAN TOTALLY REPRESENT THE REAL WORLD.

THE CRITICAL LIMITATION IS BULLET 3 - CORRECTNESS ANALYSIS.

SHORTCOMINGS OF THIS MODEL

- FINITE ACTIVITY PHASES ARE NOT REALISTIC
 - MOST ACTIVITIES DO NOT HAVE A CLEARLY DEFINED START OR END POINT
- STATIC NATURE OF THE OUTPUTS OF A PREVIOUS ACTIVITY
 - NO OUTPUT CAN BE CONSIDERED AS COMPLETE; IT WILL NEED SOME MODIFICATION IN THE NEXT ACTIVITY
 - FEEDBACK PATH HELPS TO A LIMITED DEGREE
- CORRECTNESS ANALYSIS IS PRIMARILY DONE IN TESTING ACTIVITIES
- MANAGEMENT IS NOT EXPLICITLY SHOWN

INSTRUCTOR NOTES

EMPHASIZE CONTINUOUS MANAGEMENT AND CORRECTNESS ANALYSIS.

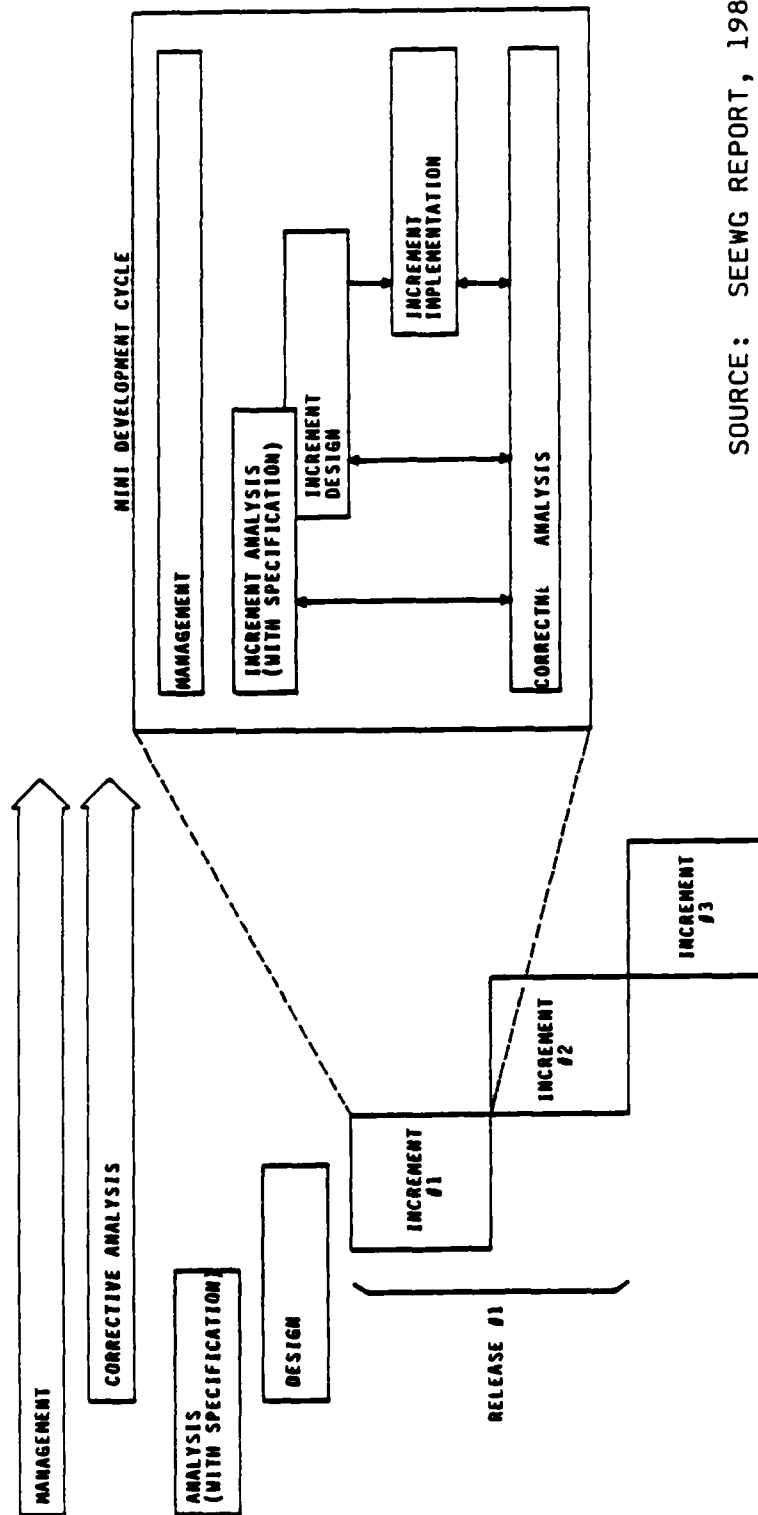
VG 744.1

4-61

A MODEL THAT ADDRESSES THE SHORTCOMINGS

● INCREMENTAL DEVELOPMENT

- BUILD SOFTWARE SYSTEM IN SMALL MANAGEABLE INCREMENTS
- EACH INCREMENT ADDS NEW FUNCTIONS TO THE SYSTEM



SOURCE: SEEWG REPORT, 1983

INSTRUCTOR NOTES

IN THIS SECTION WE WILL DISCUSS THE LIFE CYCLE THAT IS DEFINED IN MIL-STD-2167 (FORMERLY KNOWN AS MIL-STD-SDS). WE WILL ALSO BRIEFLY REVIEW THE DOCUMENTATION REQUIREMENTS OF THIS STANDARD AND HIGHLIGHT THE DIFFERENCES BETWEEN SDS AND PREVIOUS STANDARDS.

SECTION 5

MIL-STD-2167 (SDS)
LIFE CYCLE
AND
DOCUMENTATION

VG 744.1

INSTRUCTOR NOTES

THIS FIGURE SHOWS THE SYSTEM LEVEL LIFE CYCLE PHASES INTO WHICH SOFTWARE DEVELOPMENT WILL FIT. SYSTEM DEVELOPMENT TAKES PLACE EITHER DURING FULL SCALE DEVELOPMENT AND/OR PRODUCTION AND DEPLOYMENT.

THIS DIAGRAM SHOWS THE SYSTEM AND SOFTWARE LIFE CYCLE PHASES WITHIN FULL SCALE DEVELOPMENT.

DOD-STD-2167 LIFE CYCLE

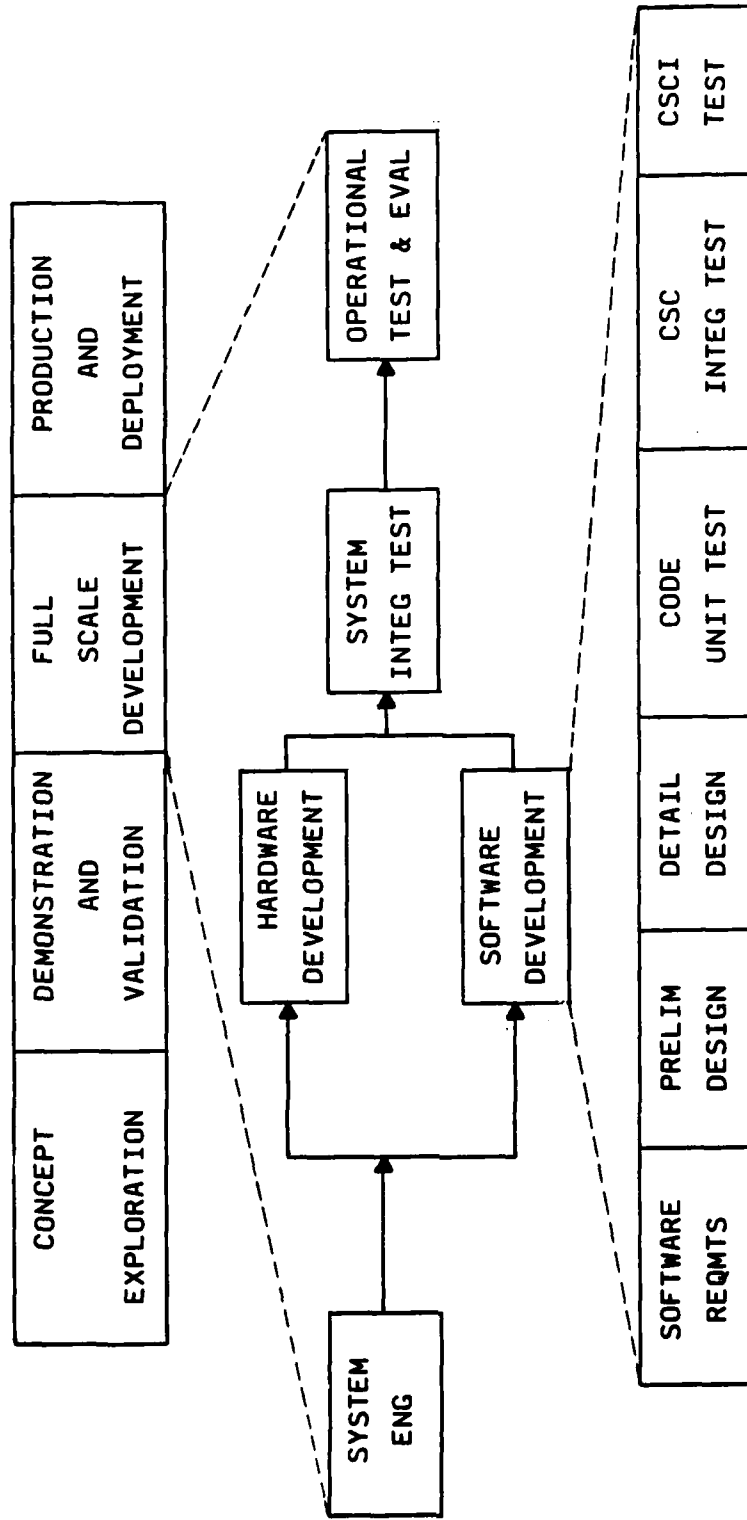


FIGURE 1

INSTRUCTOR NOTES

THIS FIGURE SHOWS THE SOFTWARE PORTION OF THE DEVELOPMENT CYCLE AND THE POSSIBLE DOCUMENTS - NOTE THAT NOT ALL OF THESE DOCUMENTS WILL BE USED ON MOST PROJECTS. THE KEY SHOWS WHICH DOCUMENTS ARE OFTEN INCLUDED IN OTHER DOCUMENTS.

POINT OUT THE REVIEWS THAT TAKE PLACE AT THE VARIOUS PLACES AND THE BASELINES THAT ARE CREATED.

PHASES

PHASES	PRELIMINARY DESIGN	DETAILED DESIGN	CONSTRUCTION AND UNIT TESTING	CSC INTEGRATION AND TESTING	CSC - TESTING

PRODUCTS

REVISIONS SUBJECTS

BASELINE, DEVELOPMENTAL/AL CONFIGURATION

KEY

- PRIMARY DOCUMENT OR CODE
- MAY BE INCLUDED IN FOLLOWING
- PRIMARY DOCUMENT
- SUPPORT DOCUMENT MAY BE VENDOR SUPPLIED
- ENTERED INTO BASELINE
- ENTERED INTO DEVELOPMENTAL CONFIGURATION
- CMSD COMPUTER RESOURCES INTEGRATED SUPPORT

DEVELOPMENTAL CONFIGURATION

INSTRUCTOR NOTES

THESE ARE NOT ALL OF THE SDS DOCUMENTS - BUT ARE PROBABLY THOSE THAT WOULD BE REQUIRED ON ANY MODERATE SIZED PROJECT. THE OCD AND SSS ARE ESSENTIALLY NEW DOCUMENTS - PREVIOUS MIL-STDS HAVEN'T REQUIRED ANY SIMILAR SOFTWARE DOCUMENT.

THE SRS IS EQUIVALENT TO THE OLD PPS, CPPS, OR B-SPEC OF PREVIOUS STANDARDS.

THE OLD C-SPEC, PDS, OR CPDS HAS BEEN SPLIT INTO TWO DOCUMENTS, THE STLDO AND THE SDDD.

THE CRISD IS A COMPLETELY NEW DOCUMENT WHICH ADDRESSES LIFE CYCLE MAINTENANCE SUPPORT REQUIREMENTS.

KEY MIL-STD-2167 DOCUMENTS

<u>NAME</u>	<u>PURPOSE</u>
OPERATIONAL CONCEPT DOCUMENT (OCD)	DESCRIBES THE MISSION OF THE SOFTWARE AND ITS OPERATIONAL AND SUPPORT ENVIRONMENTS
SYSTEM/SEGMENT SPECIFICATION (SSS)	SPECIFIES THE FUNCTIONAL, PERFORMANCE, AND INTERFACE REQUIREMENTS FOR A SYSTEM OR A SEGMENT OF A SYSTEM
SOFTWARE DEVELOPMENT PLAN (SDP)	DESCRIBES THE ORGANIZATION AND PROCEDURES TO BE USED BY THE CONTRACTOR IN PERFORMING SOFTWARE DEVELOPMENT
SOFTWARE REQUIREMENTS SPECIFICATION (SRS)	SPECIFIES IN DETAIL THE COMPLETE REQUIREMENTS (FUNCTIONAL, INTERFACE, PERFORMANCE, QUALIFICATION, ETC.) OF A PARTICULAR COMPUTER SOFTWARE CONFIGURATION ITEM (CSCI)
INTERFACE REQUIREMENTS SPECIFICATION (IRS)	SPECIFIES THE REQUIREMENTS FOR ONE OR MORE INTERFACES BETWEEN A PARTICULAR CSCI AND OTHER CONFIGURATION ITEMS OR CRITICAL ITEMS

INSTRUCTOR NOTES

VG 744.1

5-4i

KEY MIL-STD-2167 DOCUMENTS

NAME

PURPOSE

SOFTWARE TEST PLAN (STP)	DEFINES THE TOTAL SCOPE OF TESTING FOR A PARTICULAR CSCI
SOFTWARE TOP LEVEL DESIGN DOCUMENT (STLDD)	DESCRIBES THE STRUCTURE AND ORGANIZATION OF A PARTICULAR CSCI
COMPUTER RESOURCES INTEGRATED SUPPORT DOCUMENT (CRISD)	PROVIDES INFORMATION THAT IS REQUIRED TO PERFORM LIFE CYCLE SUPPORT FOR THE CONTRACTUALLY DELIVERED SOFTWARE
SOFTWARE TEST DESCRIPTION (STD)	IDENTIFIES THE INPUT DATA, EXPECTED OUTPUT DATA, AND EVALUATION CRITERIA THAT COMPRISE THE TEST CASES FOR ALL OF THE FORMAL TESTS OF A CSCI
SOFTWARE DETAILED DESIGN DOCUMENT (SDDD)	DESCRIBES IN DETAIL THE STRUCTURE AND ORGANIZATION OF A PARTICULAR CSCI
SOFTWARE TEST PROCEDURES (STP)	DESCRIBES DETAILED PROCEDURES FOR THE PERFORMANCE OF FORMAL TESTS ON A CSCI

INSTRUCTOR NOTES

THIS IS A RELEVANT DISCUSSION AS LONG AS MIL-STD-2167 IS CONSIDERED "NEW". POINT OUT THAT IN PREVIOUS STANDARD REVISIONS (I.E., WS 8506 TO SECNAVINST 3560.1 TO MIL-STD-1679) VERY LITTLE CHANGE OCCURRED IN THE DOCUMENTATION REQUIREMENTS. MIL-STD-2167 (SDS) REPRESENTS THE FIRST MAJOR CHANGE TO THE WAY WE DOCUMENT SYSTEMS IN THE PAST 10-15 YEARS (DEPENDING ON THE SERVICE).

IN GENERAL, MIL-STD-2167 PLACES MUCH MORE EMPHASIS ON METHODOLOGY, TOOLS, AND TECHNIQUES FOR EACH PHASE OF THE LIFE CYCLE - REQUIRING THE APPROACH FOR EACH PHASE TO BE DOCUMENTED IN THE SDP OR SSPM.

NOTE THAT WE WILL ADDRESS MIL-STD-2167 REQUIREMENTS AS WE COVER THE METHODS FOR EACH PHASE OF THE LIFE CYCLE.

MAJOR CHANGES IN MIL-STD-2167

- REQUIRE STRUCTURED REQUIREMENTS ANALYSIS TOOLS AND/OR TECHNIQUES
- REQUIRES DESIGN METHODOLOGY OR TOOL, PDL REQUIRED FOR DETAILED DESIGN
- ALL DOCUMENTS ARE CHANGED IN FORMAT AND CONTENT FROM PREVIOUS STANDARDS
- MORE EMPHASIS ON INTERFACE DEFINITIONS
- DOCUMENTATION OF METHODOLOGIES, TOOLS, AND TECHNIQUES TO BE USED ARE REQUIRED (SDP OR SSPM)

AD-A165 122

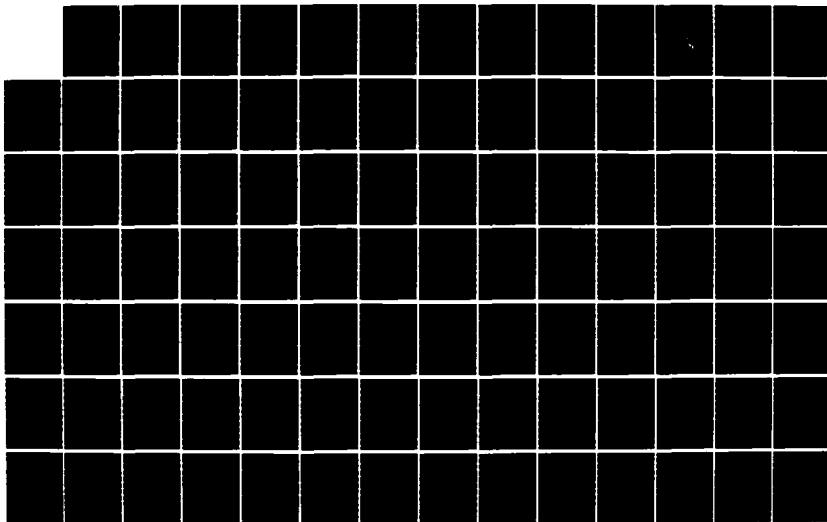
ADA (TRADEMARK) TRAINING CURRICULUM: INTRODUCTION TO
SOFTWARE ENGINEERING A102 TEACHER'S GUIDE(U) SOFTECH
INC WALTHAM MA 1986 DAB07-83-C-K506

2/6

UNCLASSIFIED

F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

PURPOSE: TO MOTIVATE FOLLOWING SECTIONS.

REFERENCE: "Ada METHODOLOGIES: CONCEPTS AND REQUIREMENTS", (METHODMAN DOCUMENT) DoD
Ada JOINT PROGRAM OFFICE, NOV. 1982.

SECTION 6

INTRODUCTION TO METHODS AND TOOLS

VG 744.1

INSTRUCTOR NOTES

THIS PICTURE ILLUSTRATES VERY ABSTRACTLY WHAT THE SOFTWARE DEVELOPMENT PROCESS IS ALL ABOUT

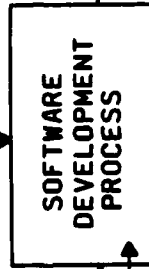
"TAKING A SET OF NEEDS AND PRODUCING A SOFTWARE SYSTEM THAT MEETS THE NEED UNDER VARIOUS CONTROLS AND CONSTRAINTS

METHODOLOGIES GUIDE THE WAY THE PROCESS IS PERFORMED.

OUR SCOPE OF CONTROL

MILITARY STANDARDS,
PROCEDURES, BUDGETS

CONTROL/
CONSTRAINTS



EFFECTIVE AND
RELIABLE
SOFTWARE
(OR SYSTEM)

NEEDS

- METHODOLOGIES AND TOOLS ARE THE ONLY THINGS WE CAN AFFECT

INSTRUCTOR NOTES

GO OVER EACH POINT CAREFULLY. THIS SETS THE STAGE FOR THE STUDENTS. IT GIVES THEM
ADDITIONAL WAYS TO LOOK AT THE METHODOLOGIES WE WILL BE DISCUSSING IN THIS COURSE.

ATTRIBUTES OF METHODOLOGIES

- A WELL-CONCEIVED METHODOLOGY HAS
 - CREATIVE ASPECTS
 - INTELLECTUAL ASPECTS
 - CLERICAL ASPECTS
 - MECHANICAL ASPECTS
- GOOD SOFTWARE ENGINEERING METHODOLOGIES SHOULD
 - IMPROVE EFFECTIVENESS AND PRODUCTIVITY OF SOFTWARE DEVELOPMENT ACTIVITIES
 - RESULT IN THE CREATION OF RELIABLE SOFTWARE
 - FIT TOGETHER TO FORM AN INTEGRATED SET OF METHODS
 - SEPARATE THE CREATIVE ASPECTS FROM THE MECHANICAL ASPECTS
 - PROMOTE AUTOMATION OF THE CLERICAL ASPECTS OF SOFTWARE DEVELOPMENT

INSTRUCTOR NOTES

MORE MOTIVATION. BE POSITIVE AND EMPHASIZE THE IMPORTANCE OF BOTH BULLETS.

WHY LEARN METHODOLOGIES?

(QUALITY VIEWPOINT)

- INCREASED EFFORT IN THE EARLIER ACTIVITIES OF A DEVELOPMENT WILL BE REFLECTED IN REDUCED COSTS FOR TESTING AND MAINTENANCE*
 - PREVENT ERRORS FROM BEING INTRODUCED
 - DETECT ANY ERRORS AT EARLIEST POSSIBLE TIME
- BY APPLYING A SET OF METHODOLOGIES YOU WILL ACHIEVE HIGHER QUALITY SOFTWARE THAT FULFILLS THE NEEDS

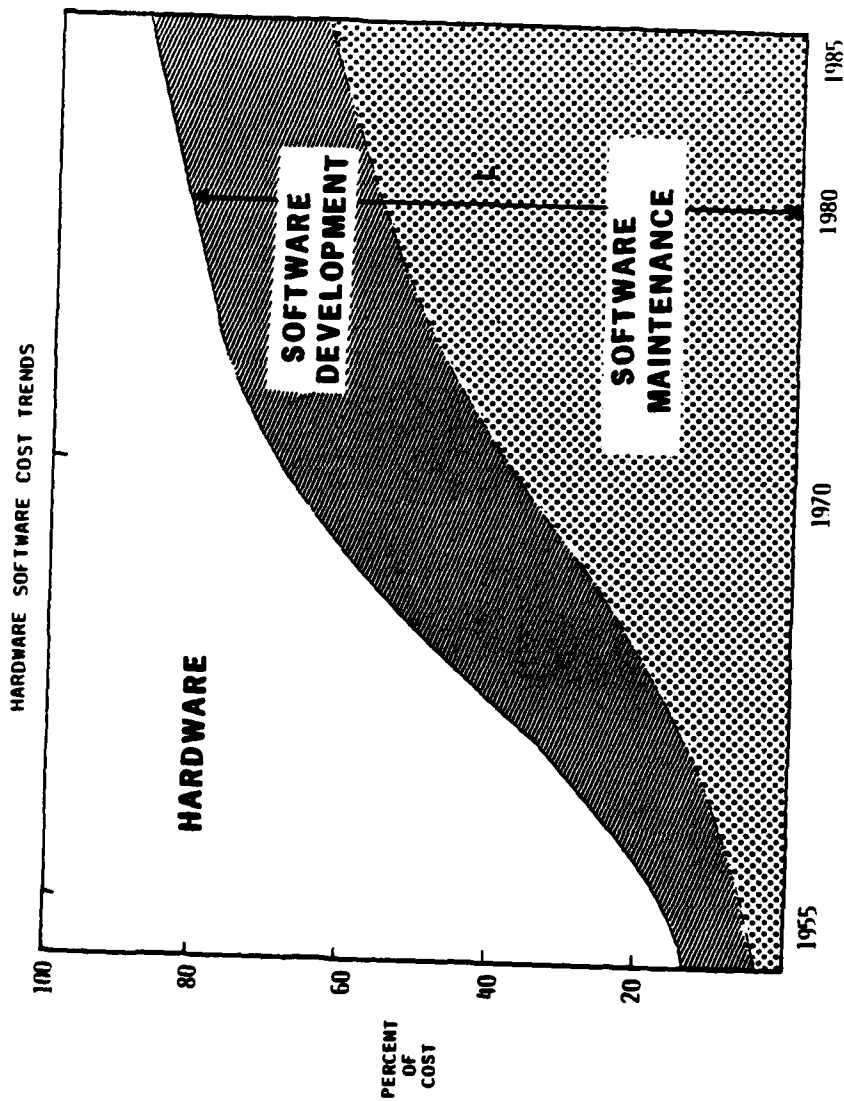
*REPAIR, ADAPTATION AND ENHANCEMENT THAT OCCURS AFTER INITIAL DEVELOPMENT.

INSTRUCTOR NOTES

THIS SLIDE IS CLOSELY LINKED WITH THE PREVIOUS ONE. RE-EMPHASIZE THE IMPORTANCE OF METHODOLOGIES TO MINIMIZE MAINTENANCE COSTS IN THE FUTURE.

WHY LEARN METHODOLOGIES? (COST VIEWPOINT)

- BY 1985 COMPUTER AND INFORMATION PROCESSING WILL REPRESENT 8.3% OF THE GNP



- IN 1980 L $\approx$ \$40,000,000,000

VG 744.1

INSTRUCTOR NOTES

TELL THE CLASS THAT THEY WILL SEE THE DETAILS OF THE DoD-STD-SDS REQUIREMENTS IN THE INTRODUCTORY SECTIONS TO ANALYSIS, DESIGN AND IMPLEMENTATION.

WHY LEARN METHODOLOGIES
(CONSTRAINT VIEWPOINT)

- FUTURE SOFTWARE DEVELOPMENTS WILL BE SO COMPLEX THAT WITHOUT A WELL-DEFINED SET OF METHODS THEY WILL BE IMPOSSIBLE TO ACCOMPLISH
- NEW MILITARY STANDARDS ARE REQUIRING THE USE OF A METHODOLOGY
 - DoD-STD-SDS
- THE POOL OF EXPERIENCED AND QUALIFIED SOFTWARE DEVELOPERS IS LIMITED
 - USE METHODOLOGIES TO IMPROVE PRODUCTIVITY

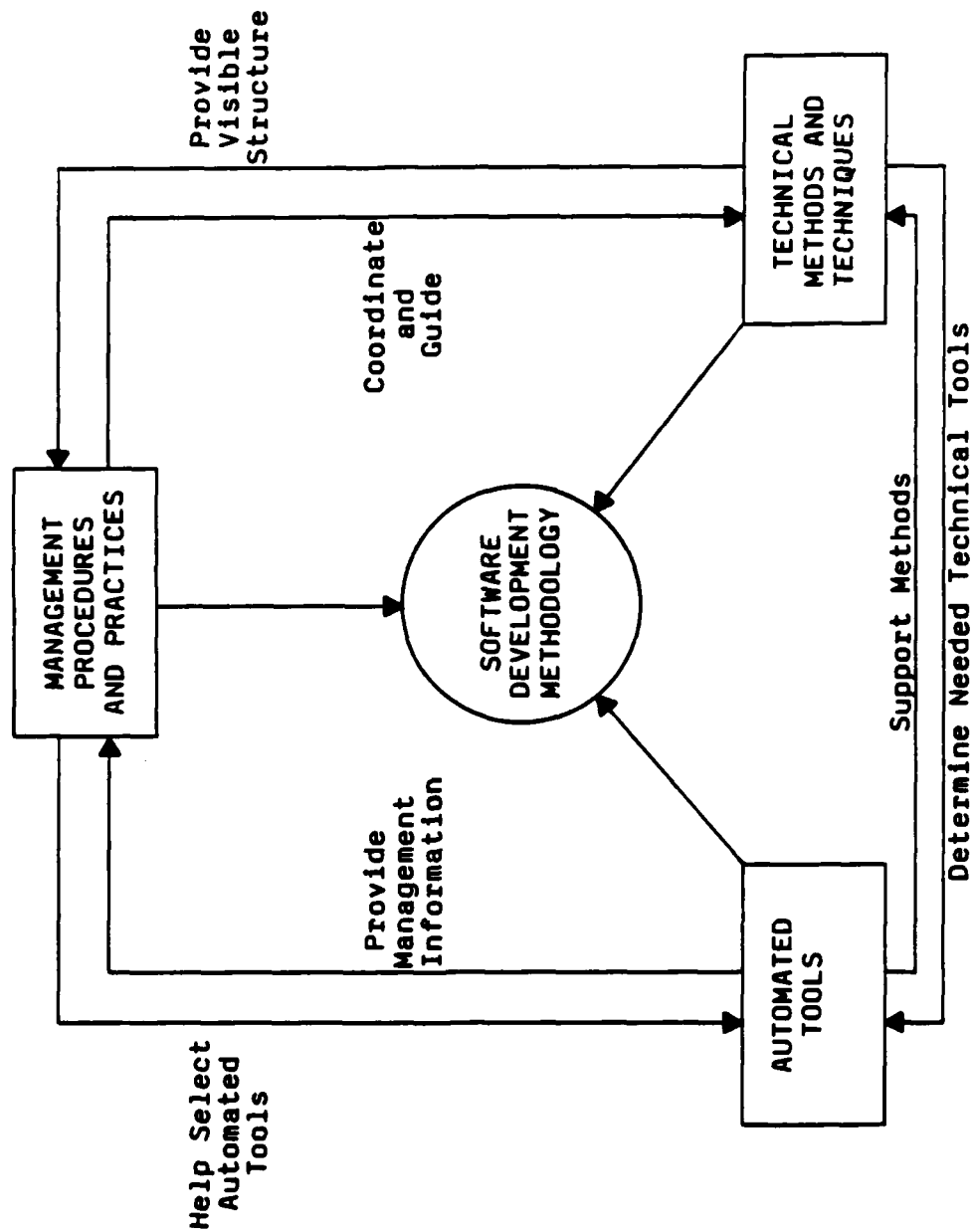
INSTRUCTOR NOTES

NOTE THAT ALL THREE ASPECTS MUST BE ADDRESSED IN A METHODOLOGY.

WALKTHROUGH THE DIAGRAM FROM METHODS, TO MANAGEMENT PRACTICES THEN AUTOMATED TOOLS.

EMPHASIZE THAT THIS COURSE FOCUSES ON THE TECHNICAL METHODS.

ASPECTS OF A METHODOLOGY (IDEAL VIEW)



SOURCE: METHODMAN, 1982

INSTRUCTOR NOTES

THE NEXT TWO SLIDES CHARACTERIZE AN IDEAL DEVELOPMENT METHODOLOGY WHICH WE COULD
USE AS A MODEL TO MEASURE THE METHODOLOGIES WE WILL COVER.

WE USE THIS IN THE WRAP-UPS FOR ANALYSIS AND DESIGN TO COMPARE THE VARIOUS METHODS
WE WILL DISCUSS.

REQUIREMENTS FOR A SOFTWARE DEVELOPMENT METHODOLOGY

(IDEAL VIEW)

- A METHODOLOGY SHOULD:
 - COVER THE ENTIRE DEVELOPMENT PROCESS, SIMPLIFYING TRANSITIONS BETWEEN PROJECT PHASES
 - ENHANCE COMMUNICATION AMONG ALL INTERESTED PERSONS AT ALL STAGES OF DEVELOPMENT
 - SUPPORT PROBLEM ANALYSIS AND UNDERSTANDING
 - SUPPORT BOTH TOP-DOWN AND BOTTOM-UP APPROACHES TO SOFTWARE DEVELOPMENT
 - SUPPORT SOFTWARE VALIDATION AND VERIFICATION THROUGH THE DEVELOPMENT PROCESS
 - FACILITATE THE CAPTURE OF DESIGN, IMPLEMENTATION, AND PERFORMANCE CONSTRAINTS

INSTRUCTOR NOTES

VG 744.1

6-81

REQUIREMENTS FOR A SOFTWARE DEVELOPMENT METHODOLOGY

(IDEAL VIEW CONTINUED)

- A METHODOLOGY ALSO SHOULD:
 - SUPPORT THE SOFTWARE DEVELOPMENT ORGANIZATION
 - SUPPORT THE EVOLUTION OF A SYSTEM THROUGHOUT ITS EXISTENCE
 - BE SUPPORTED BY AUTOMATED TOOLS
 - MAKE THE EVOLVING SOFTWARE PRODUCT VISIBLE AND CONTROLLABLE AT ALL STAGES OF DEVELOPMENT
 - BE TEACHABLE AND TRANSFERABLE
 - BE OPEN-ENDED

INSTRUCTOR NOTES

EMPHASIZE TO THE CLASS THAT THEY WILL HAVE TO COMBINE METHODS TO MAKE ANY OF THEM EFFECTIVE.

VG 744.1

6-91

METHODOLOGY COURSE VIEW

- NO SINGLE EXISTING METHODOLOGY MEETS THE "IDEAL" REQUIREMENTS

- USED IN COMBINATION WITH SOME GLUE WE CAN COME CLOSE

- IN THE COURSE YOU WILL

- GET AN OVERVIEW OF A LOT OF METHODS

- GET SOME INSIGHT INTO HOW THEY FIT THIS "IDEAL" SET OF

REQUIREMENTS

INSTRUCTOR NOTES

THEME: ANALYSIS IS CRITICAL IN THE DEVELOPMENT OF SOFTWARE, THUS WE MUST HAVE GOOD METHODS TO ADDRESS AND IDENTIFY THE REAL SOFTWARE REQUIREMENTS

PURPOSE: FOCUS THE STUDENTS' ATTENTION ON THE CRITICAL ASPECTS OF REQUIREMENTS ANALYSIS

- REFERENCES:
1. FREEMAN, P., " A PERSPECTIVE ON REQUIREMENTS ANALYSIS AND SPECIFICATION" IBM DESIGN '79 SYMPOSIUM PROCEEDINGS; APRIL 1979
 2. WEINBERG, G., "RETHINKING SYSTEMS ANALYSIS AND DESIGN" LITTLE, BROWN, MA; 1982
 3. DoD-STD-SDS, PROPOSED MILITARY STANDARD FOR DEFENSE SYSTEM SOFTWARE DEVELOPMENT, DEC. 1983 (DoD-STD-1267)

SECTION 7

REQUIREMENTS ANALYSIS INTRODUCTION

VG 744.1

INSTRUCTOR NOTES

A REQUIREMENT IS A BINDING CONDITION WHICH STATES A MANDATORY CHARACTERISTIC OF AN ABSTRACT OR PHYSICAL OBJECT.

REQUIREMENTS MAY HAVE DIFFERENT FORMS: A SPECIFIC DESCRIPTION, A CONSTRAINT, AN EVALUATION CRITERIA FOR JUDGING QUALITY, OR IT MAY BE IMPLIED BY CONTEXT.

THERE'S ALWAYS MORE THAN ONE PARTY INVOLVED. ACHIEVING CONSENSUS IS AN IMPORTANT ASPECT OF REQUIREMENTS ANALYSIS. STRESS THAT REQUIREMENTS ANALYSIS IS A HUMAN ENDEAVOR.

ANALYSIS IMPLIES BOTH REQUIREMENTS ANALYSIS
AND SPECIFICATIONS

- A REQUIREMENT IS ...
 - AN EXPRESSION OF NEED
 - AN IMPOSED DEMAND
 - SOMETHING SOMEONE WILL PAY FOR
- REQUIREMENTS FORM A "CONTRACT BETWEEN USER AND DEVELOPERS"
- REQUIREMENTS ARE DOCUMENTED IN REQUIREMENTS SPECIFICATION(S)

INSTRUCTOR NOTES

THE ANALYSIS IS USUALLY CONDUCTED WHILE AT THE SPECIFICATION AND/OR DESIGN LEVEL

REQUIREMENTS ANALYSIS

- REQUIREMENTS ANALYSIS IS THE PROCESS BY WHICH THE FEASIBILITY OF REQUIREMENTS ARE DETERMINED, PRIOR TO THE DEVELOPMENT OF THE SYSTEM
- THE ANALYSIS IS TYPICALLY PERFORMED BY A SENIOR SYSTEMS EXPERT OR ANALYST WHO ESTABLISHES AND DOCUMENTS THE REQUIREMENTS

INSTRUCTOR NOTES

GOOD ANALYSTS CAN EXPRESS THE REAL REQUIREMENTS, AS OPPOSED TO THE STATED OR PERCEIVED ONES. IT IS A DIFFICULT JOB, AND MUST BE CAREFULLY THOUGHT OUT.

THE ANALYST IS THE BRIDGE BETWEEN
USERS AND DEVELOPERS

• AN ANALYST WILL:

- POSTPONE DESIGN DECISIONS
- PUT HIMSELF IN THE USER'S CHAIR
- QUESTION THE RATIONALE
- UNDERSTAND THE REAL PROBLEM
- CONSIDER SEVERAL SOLUTIONS
- CLEARLY COMMUNICATE REQUIREMENTS AS WELL AS THE RESULTING IMPLEMENTATION

INSTRUCTOR NOTES

MANY SYSTEMS HAVE BEEN BUILT, TESTED AND MADE OPERATIONAL, ONLY TO BE NOT USED BECAUSE THEY DIDN'T MEET THE "REAL" REQUIREMENTS, ALTHOUGH THEY MET THE STATED REQUIREMENTS. REQUIREMENTS ARE HARD TO FORMULATE CORRECTLY!

CONSEQUENCES OF WRONG REQUIREMENTS

- WRONG REQUIREMENTS CAN CAUSE ...

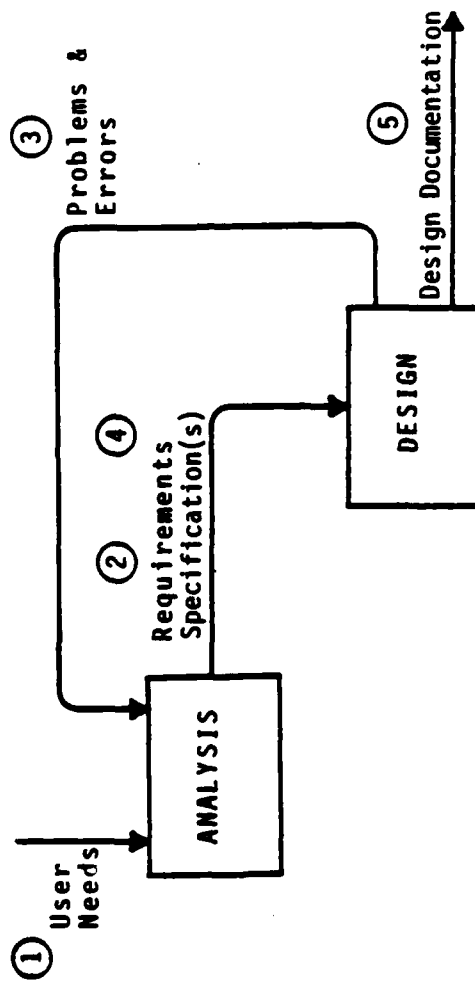
- SYSTEM REJECTION
- SYSTEM PATCHING OR RETROFIT
- INSTALLATION OF A DANGEROUS SYSTEM
- FAILURE OF THE PROJECT
- LOSS OF FUTURE BUSINESS

INSTRUCTOR NOTES

POINT OUT THAT OFTEN SOME LEVEL OF DESIGN IS NEEDED DURING ANALYSIS IN ORDER TO DETERMINE REQUIREMENTS FEASIBILITY.

ANALYSIS AND DESIGN ARE INTERRELATED

• ANALYSIS AND DESIGN ITERATE ...



INSTRUCTOR NOTES

POINT OUT THAT DESIGNERS ALSO CONTRIBUTE TO THE ANALYSIS PHASE. THEY CITE PROBLEMS AND ERRORS FROM THEIR VIEWPOINT OF FEASIBILITY, AND CONVEY THIS TO THE ANALYST. ALSO STATE THAT THERE MAY BE MULTIPLE REQUIREMENT SPECIFICATIONS (SOFTWARE, HARDWARE, SYSTEM, ETC.) THAT NEED TO BE CREATED AND ITERATED. CONFLICTS AMONG THEM NEED TO BE SPOTTED BY THE ANALYST.

DESIGNER'S ROLE IN ANALYSIS

- DURING ANALYSIS, DESIGNERS IDENTIFY ...

- AMBIGUITIES

- INCONSISTENCIES

- INCOMPLETENESS

- POTENTIAL TROUBLE AREAS

- REQUIREMENTS HAVING DISPROPORTIONATE
COST/SCHEDULE IMPACT

INSTRUCTOR NOTES

THE SOFTWARE SPECIFICATION REVIEW CONSISTS OF REVIEWING BOTH THE SOFTWARE REQUIREMENTS
AND INTERFACE REQUIREMENTS SPECIFICATION DOCUMENTS.

MIL-STO-SDS VIEW OF ANALYSIS

- SDS ANALYSIS PRODUCTS

- RECORDS OF DOCUMENT REVIEWS, STUDIES, ACTION ITEMS
AND OTHER ANALYSIS INTERMEDIATE INFORMATION
- SOFTWARE REQUIREMENTS SPECIFICATION
- INTERFACE REQUIREMENTS SPECIFICATION

- SDS ANALYSIS REVIEWS

- SOFTWARE SPECIFICATION REVIEW

INSTRUCTOR NOTES

THE NEXT FEW SLIDES PROVIDE CURRENT THINKING FROM DoD'S POINT OF VIEW ON THE ROLE OF ANALYSIS.

VG 744.1

7-81

DOD-STD-SDS VIEW OF ANALYSIS

- THE DOD STANDARD FOR DEFENSE SYSTEM SOFTWARE DEVELOPMENT (MIL-STD-SDS) ESTABLISHES
 - UNIFORM SOFTWARE DEVELOPMENT PROCESS
 - PRACTICES DEMONSTRATED TO BE COST-EFFECTIVE FROM A SOFTWARE LIFE CYCLE PERSPECTIVE
 - DOCUMENTATION REQUIREMENTS
 - REVIEW REQUIREMENTS
- SDS ANALYSIS GOAL
 - ESTABLISH COMPLETE SET OF FUNCTIONAL PERFORMANCE AND INTERFACE REQUIREMENTS FOR EACH SOFTWARE ITEM
- SDS ANALYSIS ACTIVITIES
 - ANALYZE EXISTING NEEDS OR REQUIREMENTS DOCUMENTS FOR ADEQUACY, TESTABILITY, UNDERSTANDABILITY, AND COMPLETENESS
 - USE A STRUCTURED REQUIREMENTS ANALYSIS TOOL OR TECHNIQUE
 - DEVELOP SPECIFICATION SET

INSTRUCTOR NOTES

VG 744.1

8-1



SECTION 8

ANALYSIS METHODS OVERVIEW

VG 744.1

INSTRUCTOR NOTES

IN THIS SECTION WE WILL PRESENT AN INTERMEDIATE LEVEL OF DISCUSSION ON EACH OF FIVE ANALYSIS TECHNIQUES. FOR EACH OF THESE TECHNIQUES WE WILL PRESENT SOME BACKGROUND INFORMATION AND THEN SHOW EXAMPLES OF HOW THE TECHNIQUE IS USED TO REPRESENT VARIOUS TECHNICAL ASPECTS OR CHARACTERISTICS SUCH AS FUNCTIONAL HIERARCHY, CONTROL FLOW, MAJOR COMPONENTS, OR WHATEVER THE TECHNIQUE IS KNOWN FOR.

FINALLY, WE WILL PRESENT A CRITIQUE OF THE TECHNIQUE ADDRESSING ITS COVERAGE OF OUR PRINCIPLES AND GOALS.

OVERVIEW

- SADT - STRUCTURED ANALYSIS AND DESIGN TECHNIQUE
- SREM - SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY
- PSL/PSA - PROBLEM STATEMENT LANGUAGE/PROBLEM STATEMENT ANALYSIS
- SSA - STRUCTURED SYSTEM ANALYSIS
- SCRP - SOFTWARE COST REDUCTION PROJECT

INSTRUCTOR NOTES

IN ADDITION TO SOME GENERAL BACKGROUND INFORMATION WE WILL BE PRESENTING EXAMPLES OF ACTUAL SADT DIAGRAMS TO DEMONSTRATE EACH OF THESE CAPABILITIES.

SADT

BACKGROUND

BASIC SYNTAX/PRECEDENCE, CONTROL FEEDBACK

FUNCTIONAL HIERARCHY/DECOMPOSITION, INTERFACES

DATA FLOW/DATA HIERARCHY

REVIEW PROCEDURES

CRITIQUE

INSTRUCTOR NOTES

SADT CAN BE VIEWED AS A REFINEMENT AND FORMALISM OF THE TYPE OF BLOCK DIAGRAMMING THAT SEEMS TO BE SECOND NATURE TO MOST ENGINEERS. SADT ADDS SOME RULES AND CONVENTIONS THAT ALLOW LARGER AND MORE COMPLEX PROBLEMS TO BE ADDRESSED WITH MORE CONCISENESS THAN WOULD OTHERWISE BE POSSIBLE.

SADT WAS CREATED IN THE EARLY 70S AND HAS BEEN USED ON HUNDREDS OF COMMERCIAL AND MILITARY APPLICATIONS.

SADT IS USUALLY TAUGHT IN A 5 OR 10 DAY COURSE AND HAS BEEN PICKED UP BY MANY PRACTITIONERS WITHOUT ANY FORMAL TRAINING.

SADT INCLUDES EXTENSIVE PROCEDURES FOR MANAGING THE FLOW OF INFORMATION, FOR CONDUCTING INTERVIEWS, AND FOR INSURING TIMELY AND THOROUGH REVIEW OF INFORMATION IT HAS PRODUCED.

THESE PROCEDURES FACILITATE COMMUNICATION BY POTENTIALLY LARGE GROUPS OF ENGINEERS ON COMPLEX PROBLEMS - INSURES A CONSENSUS AT ALL LEVELS OF SYSTEM DEFINITION.

THE AIR FORCE OWNED VERSION IS KNOWN AS IDEF₀ ICAM (INTEGRATED COMPUTER AIDED MANUFACTURING) DEFINITION METHOD. TALK ABOUT BEHAVIOR MODELING AT YOUR OWN RISK.

SADT

- REPRESENTS A FORMALISM OF TRADITIONAL BLOCK DIAGRAMMING
- USED FOR OVER 10 YEARS ON A WIDE RANGE OF APPLICATIONS
- SIMPLE, LIMITED SYNTAX EASILY LEARNED AND USED
- INCLUDES MANAGEMENT AND REVIEW PROCEDURES
- SUPPORTS COMMUNICATION AND CONSENSUS BY LARGE DIVERSE GROUPS
- ALSO KNOWN AS IDEF₀
- PRIMARILY AN ANALYSIS TOOL BUT ALSO USED FOR ARCHITECTURAL DESIGN
- TECHNIQUE RECENTLY EXTENDED TO MODEL BEHAVIOR (STIMULUS/RESPONSE)

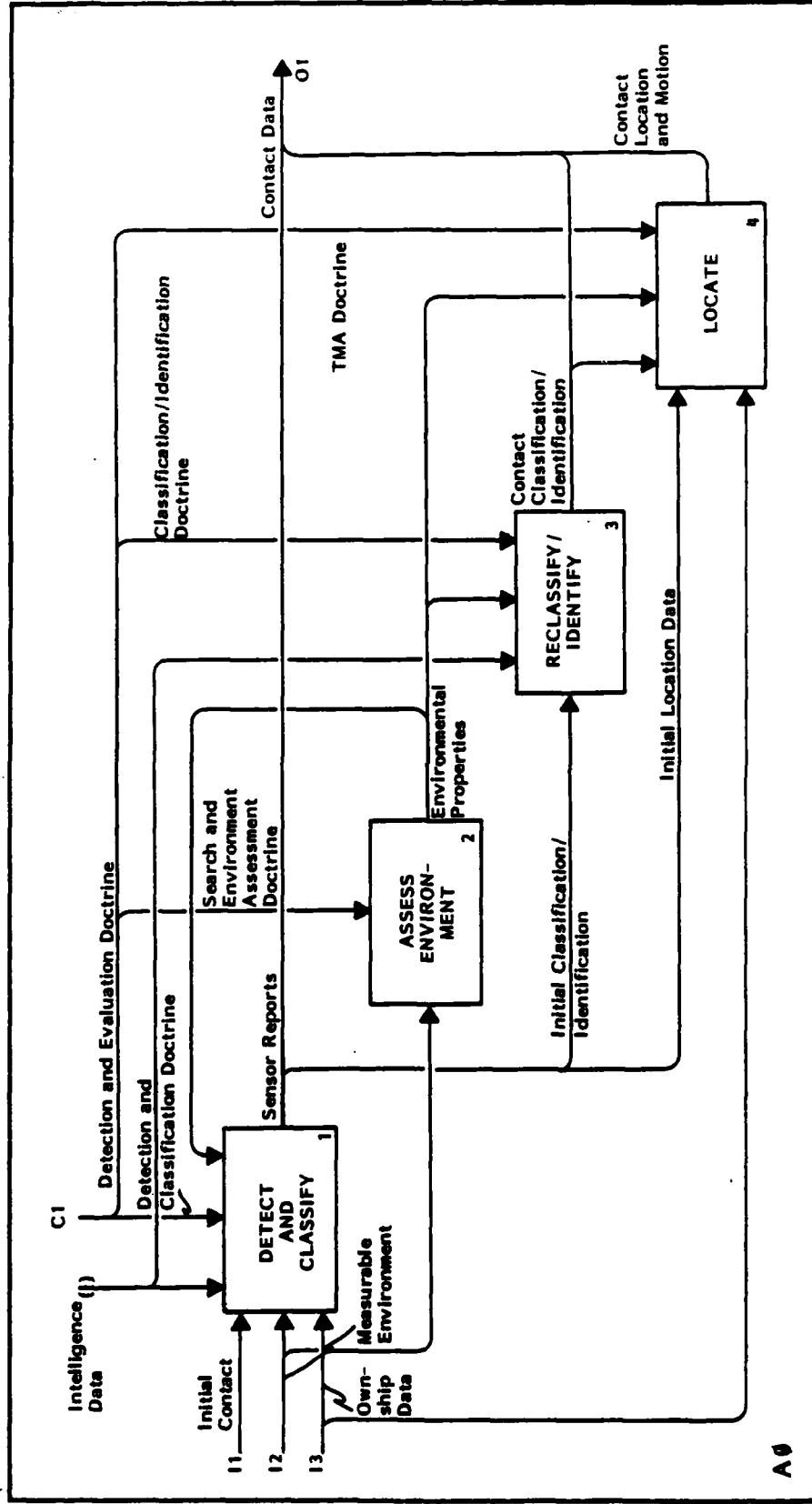
INSTRUCTOR NOTES

REVIEW THE BASIC BOXES AND ARROWS SYNTAX. POINT OUT THAT IF IT'S NOT CLEAR WHETHER SOMETHING IS AN INPUT OR A CONTROL THAT IT SHOULD BE SHOWN AS A CONTROL.

ON THE LOWER SADT DIAGRAM POINT OUT

1. SHOWS THE PRECEDENCE RELATIONSHIP BETWEEN BOXES 1, 2 AND 4. THE OUTPUTS OF BOXES 1 AND 2 MUST BE AVAILABLE AS AN INPUT TO BOX 4 BEFORE IT CAN PROCESS.
2. SHOWS THAT THE OUTPUT OF BOX 1 CONTROLS OR CONSTRAINS THE WAY BOX 2 IS PERFORMED. ALSO THE OUTPUT OF BOX 2 CONTROLS BOX 3.
- 3.. ONE OF THE POSSIBLE OUTPUTS OF BOX 4 (LAST BACK-ON) FEEDS BACK TO CONTROL BOX 1.

BASIC SYNTAX/PRECEDENCE, CONTROL, FEEDBACK



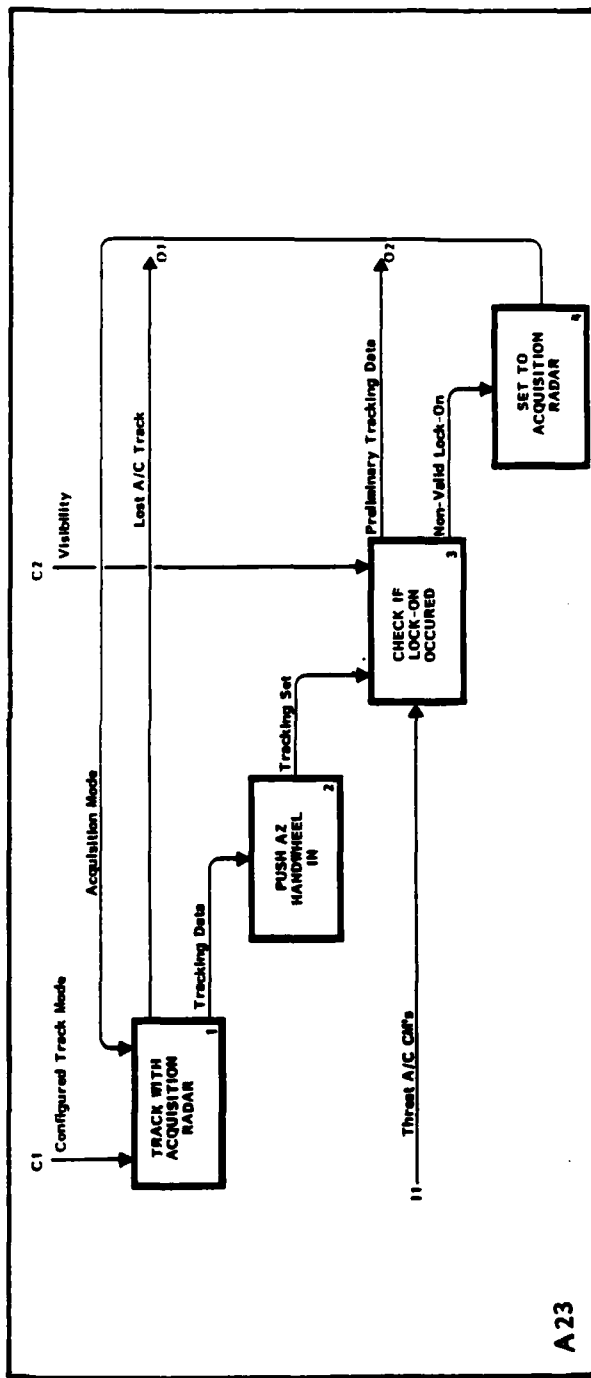
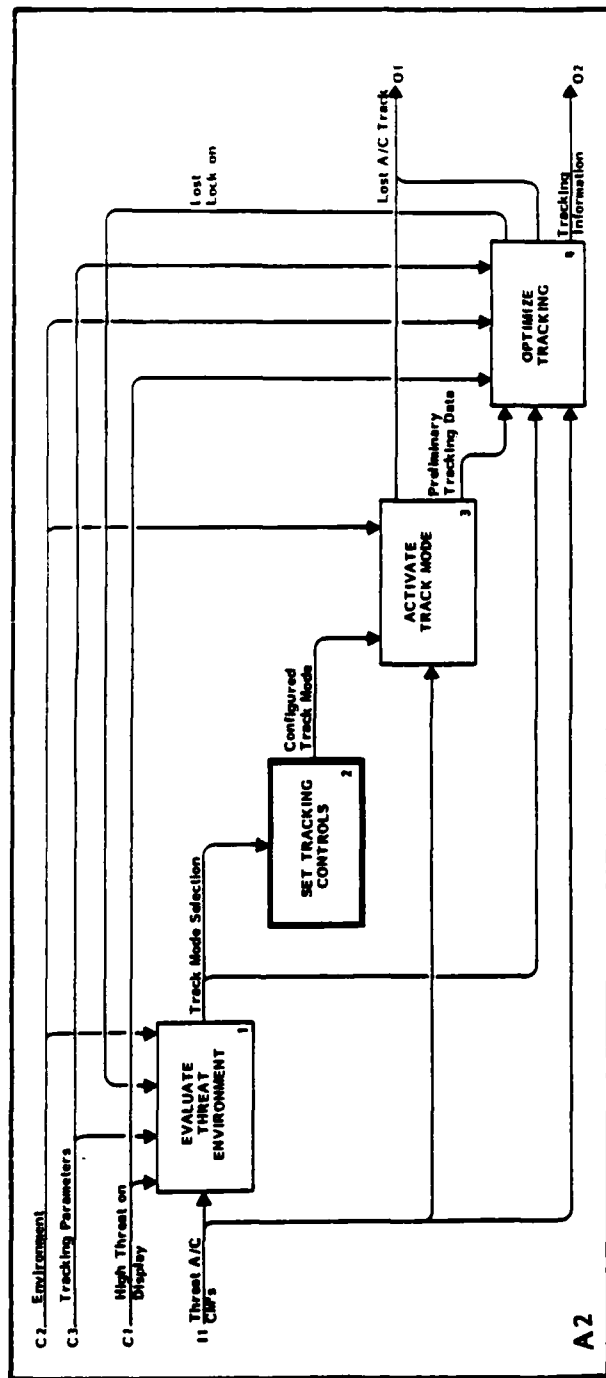
INSTRUCTOR NOTES

IN THIS EXAMPLE WE SEE HOW SADT REPRESENTS FUNCTIONAL HIERARCHY. THE BOTTOM DIAGRAM IS THE DECOMPOSITION OF BOX 3 ACTIVATE TRACK MODE ON THE UPPER DIAGRAM.

POINT OUT THAT EACH OF THE INTERFACES TO BOX 3 ON THE UPPER DIAGRAM IS INHERITED BY THE BOTTOM DIAGRAM AND THAT THEY ARE LABELED WITH ICOMS (INPUT, CONTROL, OUTPUT, MECHANISM) WHICH REFLECT THE POSITION OF THE ARROW ON THE UPPER DIAGRAM. SADT REQUIRES THIS INTERFACE CONSISTENCY BETWEEN HIERARCHICAL LAYERS.

POINT OUT THE DIAGRAM NAMING SCHEME - DIAGRAM A23 IS A DECOMPOSITION OF BOX 3 ON DIAGRAM A2.

FUNCTIONAL HIERARCHY/DECOMPOSITION, INTERFACES



INSTRUCTOR NOTES

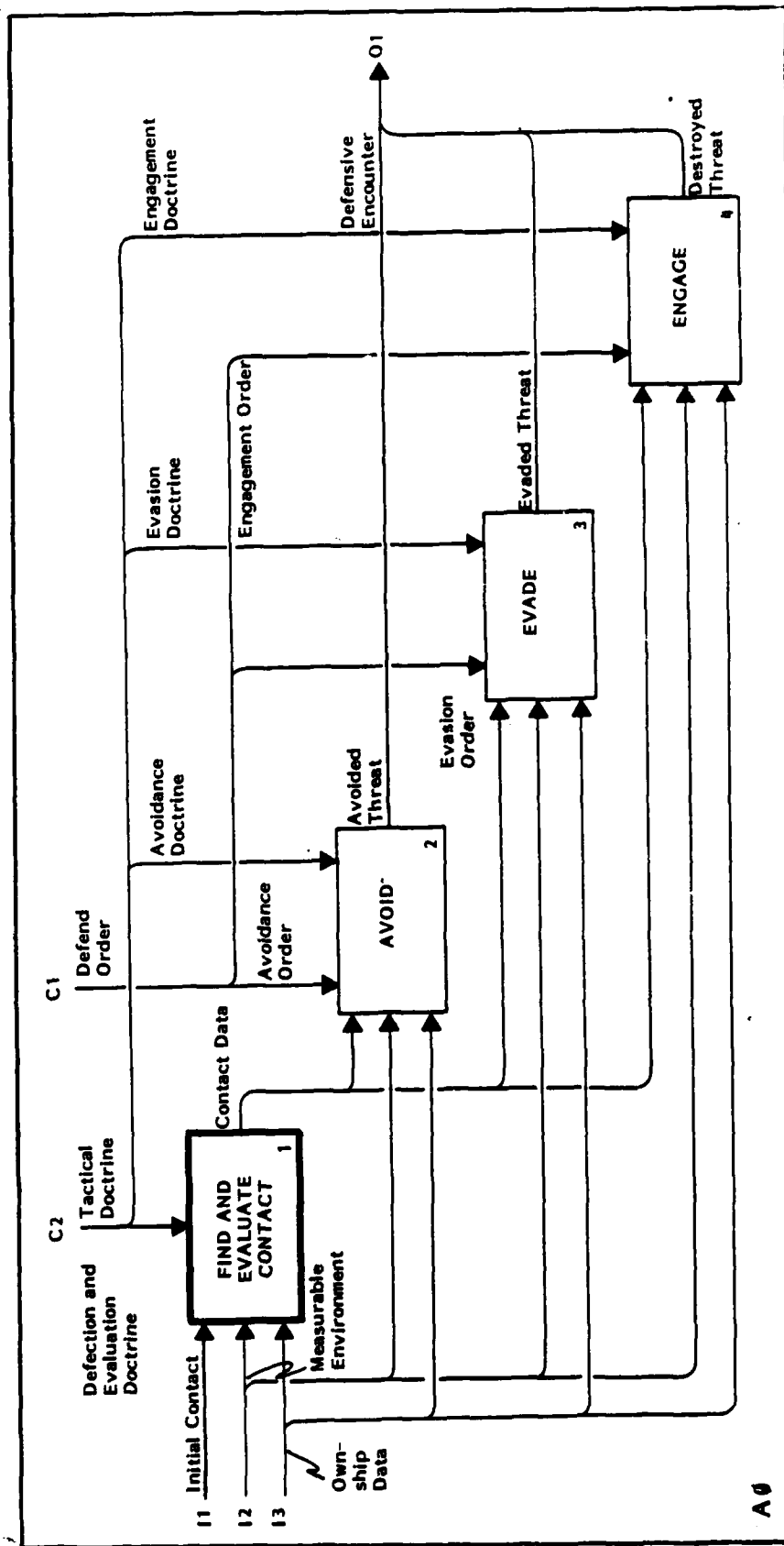
THIS DIAGRAM SHOWS BOTH DATA FLOW (WITH THE DATA BEING TRANSFORMED AS IT FLOWS THROUGH THE FUNCTIONS) AS WELL AS DATA DECOMPOSITION.

INITIAL CONTACT (11) IS TRANSFORMED INTO CONTACT DATA AND THEN INTO EITHER AVOIDED THREAT, EVADED THREAT, OR DESTROYED THREAT DEPENDING ON CONTROLS OUTSIDE OF THIS DIAGRAM.

THREE EXAMPLES OF DATA DECOMPOSITION ARE SHOWN

1. TACTICAL DOCTRINE IS COMPOSED OF DETECTION AND EVALUATION DOCTRINE, AVOIDANCE DOCTRINE, EVASION DOCTRINE, AND ENGAGEMENT DOCTRINE.
2. DEFEND ORDER IS COMPOSED OF EITHER AVOIDANCE ORDER (MISPLACED), EVASION ORDER, OR ENGAGEMENT ORDER.
3. DEFENSIVE ENCOUNTER IS COMPRISED OF EITHER AVOIDED THREAT, EVADED THREAT, OR DESTROYED THREAT.

DATA FLOW/DATA DECOMPOSITION



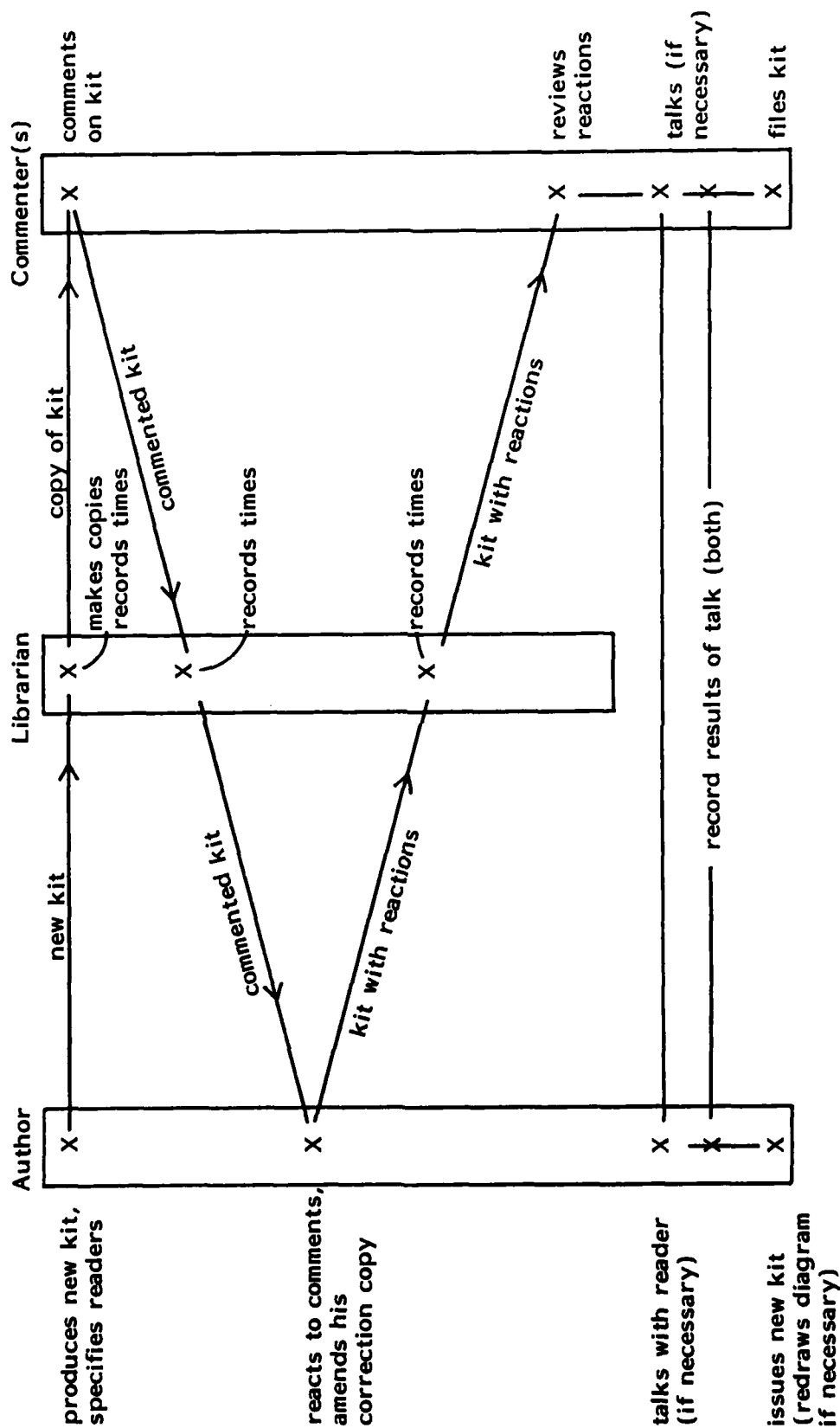
INSTRUCTOR NOTES

THE READER/AUTHOR CYCLE IS ONE EXAMPLE OF A REVIEW PROCEDURE THAT IS PART OF THE MANAGEMENT OR PROCEDURAL ASPECTS OF SADT.

WALK THE AUDIENCE THROUGH THE STEPS OF THE CYCLE. STRESS THE FACT THAT A READER "OWNS" HIS COPY OF A KIT AND THAT IN THE END THE COPIES OF THE KITS RESIDE WITH THE READERS (COMMENTERS).

POINT OUT THAT THESE PROCEDURES CAN BE USED WITH PROJECTED MATERIAL OTHER THAN SADT KITS.

REVIEW PROCEDURES



INSTRUCTOR NOTES

THE THREE IMPORTANT PRINCIPLES ON WHICH SADT ARE BUILT ARE STRUCTURING (STRICT TOP-DOWN HIERARCHY), ABSTRACTION, AND MODULARITY. ABSTRACTION IS SUPPORTED BY THE TOP-DOWN DECOMPOSITION WHICH FORCES ANY FUNCTION OR ACTIVITY TO INITIALLY BE DESCRIBED IN A SINGLE BOX. MODULARITY IS IMPOSED VIA THE DECOMPOSITION PROCESS WHICH ALWAYS CREATES 3 TO 6 NEW COMPONENTS WITH CLEARLY DEFINED INTERFACES. SADT IS LESS FORMAL THAN SOME TECHNIQUES, BUT MORE FORMAL THAN AD HOC BLOCK DIAGRAMMING TECHNIQUES OR FUNCTION AND CONTROL FLOW DIAGRAMS. SYNTAX RULES (I.E., CONTROL AND INPUT ARROWS) AND HIERARCHY RULES ARE EXAMPLES OF THESE FORMALISMS - MANY OTHERS EXIST.

THE PRIMARY GOAL OF SADT IS UNDERSTANDABILITY - REACHING A CONSENSUS AMONG A GROUP OF COMMUNICATING ANALYSTS. CORRECTNESS AND VERIFIABILITY OF THE SYSTEM DESCRIPTION ARE MUCH BETTER THAN WITH AD HOC DIAGRAMMING TECHNIQUES, BUT ARE NOT AS RIGOROUS AS SOME OTHERS.

THE HIERARCHICAL DECOMPOSITION RULES AID IN TRACING A REQUIREMENT THROUGH ALL LEVELS OF A SYSTEM DESCRIPTION AIDING THE MAINTAINABILITY OF THE OVERALL SYSTEM.

SADT

PRINCIPLES

- STRUCTURING
 - TOP DOWN, HIERARCHICAL
- MODULARITY
 - DECOMPOSITION RULES
- ABSTRACTION
 - MULTIPLE LEVEL DESCRIPTOR
- FORMALISM
 - REFINES INTUITIVE BLOCK
DIAGRAMMING TECHNIQUE
WITH HIERARCHY AND SYNTAX
RULES

GOALS

- UNDERSTANDABILITY
 - USES STRUCTURING, ABSTRACTION,
AND MODULARITY TO AID
COMMUNICATION
- CORRECTNESS, VERIFIABILITY
 - INTUITIVE NATURE OF DIAGRAMS AND,
REVIEW OF CORRECTNESS
- MAINTAINABILITY, TRACEABILITY
 - HIERARCHICAL DECOMPOSITION ALLOWS
EASY FOCUSING ON AREAS OF
INTEREST

INSTRUCTOR NOTES

FOR SREM WE WILL HIGHLIGHT THE MAJOR COMPONENTS OF THE SYSTEM WHICH ARE THE RSL LANGUAGE (A SUPERSET OF PSL TO SOME EXTENT), R-NETS WHICH ARE A GRAPHICAL LANGUAGE FOR EXPRESSING STIMULUS/RESPONSE FLOW OF CONTROL NETWORKS, THE RADX ANALYZER WHICH PERFORMS CONSISTENCY AND COMPLETENESS ANALYSIS (MUCH LIKE PSA), AND FINALLY THE FUNCTIONAL SIMULATION SUPPORT.

SREM

- BACKGROUND

- OVERVIEW

- RSL LANGUAGE

- R-NETS

- RADX ANALYSIS

- FUNCTIONAL SIMULATION

- CRITIQUE

INSTRUCTOR NOTES

INSTRUCTOR SHOULD READ ONE OF THE MANY MACK ALFORD PAPERS ON SUBJECT. SREM DEVELOPED INITIALLY AS AN AUTOMATED TOOL SPECIFICALLY FOR REAL TIME SYSTEMS. R-NETS PROVIDE A CIRCUIT-LIKE DESCRIPTION OF THE INPUT STIMULUS TO OUTPUT RESPONSE NATURE OF THE SYSTEM. RSL IS IN MANY WAYS SIMILAR TO PSL.

SYSTEM AUTOMATICALLY GENERATES SIMULATION TEMPLATES FOR COMPONENTS THAT ARE THEN COMPLETED BY FOLLOWING IN PASCAL CODE.

HAS BEEN USED ON SEVERAL LARGE TRW PROJECTS.

SREM

- AUTOMATED REQUIREMENTS DEFINITION AND ANALYSIS SYSTEM
- DEVELOPED BY TRW FOR ARMY (BMD)
- MACK ALFORD, TRW, HUNTSVILLE IS POINT OF CONTACT
- TECHNIQUE DEVELOPED SPECIFICALLY FOR REAL TIME SYSTEMS
- HOSTED ON VAX/VMS BASED SYSTEM
- PROVIDES AUTOMATED CONSISTENCY CHECKING
- ASSISTS IN DEVELOPING SIMULATION
- RSL SIMILAR TO PSL
- R-NETS TRACE STIMULUS/RESPONSE NETWORK
- BEING EXTENDED TO SUPPORT DISTRIBUTED PROCESSING APPLICATIONS

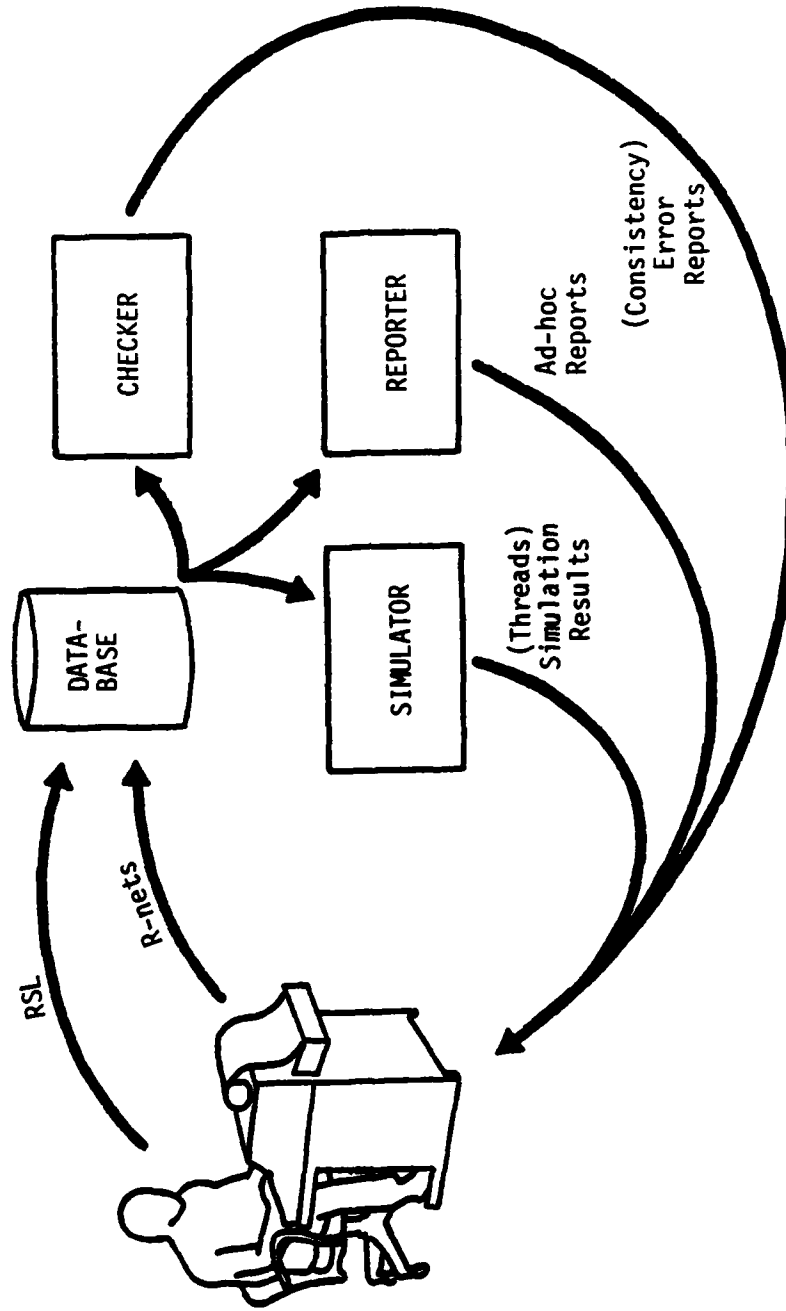
INSTRUCTOR NOTES

SREM BEGINS WITH THE TRANSLATION AND DECOMPOSITION OF SYSTEM LEVEL REQUIREMENTS. AS PART OF THE METHODOLOGY, A SET OF SOFTWARE SUPPORT TOOLS WERE IMPLEMENTED TO AUTOMATE MANY OF THE PREVIOUSLY MANUAL ACTIVITIES ASSOCIATED WITH REQUIREMENTS ENGINEERING. THESE SOFTWARE TOOLS FORM THE REQUIREMENTS ENGINEERING AND VALIDATION SYSTEM (REVS); REVS PROCESSING IS ACCOMPLISHED BY EXPRESSION OF THE SOFTWARE REQUIREMENTS IN THE STRUCTURED, FORMAL REQUIREMENTS SPECIFICATION LANGUAGE (RSL). A KEY CONCEPT OF REVS IS THAT ALL REQUIREMENTS ARE TRANSLATED INTO A CENTRAL DATABASE CALLED THE ABSTRACT SYSTEM SEMANTIC MODEL (ASSM). THE RSL STATEMENTS THEMSELVES ARE NOT STORED IN THE ASSM. INSTEAD, THEY ARE TRANSLATED INTO REPRESENTATIONS OF THE INFORMATION CONTENT OF THE REQUIREMENTS STATEMENTS. THIS PROVIDES AN EFFICIENT AND FLEXIBLE MEANS OF MAINTAINING A LARGE SOFTWARE SPECIFICATION IN A RELATIVELY SMALL COMPUTER DATABASE.

RSL EXPRESSES REQUIREMENTS IN AN UNAMBIGUOUS, MACHINE-PROCESSABLE LANGUAGE, AS OPPOSED TO FREE FORM (AND FREE CONTENT) ENGLISH. SUPPORT SOFTWARE IS AVAILABLE TO AUTOMATICALLY PROCESS THE REQUIREMENTS STATEMENTS AND PERFORM A WIDE RANGE OF NEEDED ACTIVITIES (E.G., IDENTIFY SYNTAX ERRORS USING THE RSL TRANSLATOR).

REQUIREMENTS ENGINEERING AND VALIDATION SYSTEMS (REVS) CONSISTS OF A COMPUTER PROGRAM WITH A DATABASE. THE REQUIREMENTS INFORMATION WRITTEN FOR A SYSTEM WILL BE STORED IN THIS DATABASE AND A LIST OF ERRORS PROVIDED. THE DATABASE CAN'T BE CORRECTED UNTIL ALL DETECTED ERRORS ARE REMOVED. TOOLS ARE AVAILABLE TO MECHANICALLY GENERATE A SIMULATION FOR THE SYSTEM AND TO PROVIDE AUTOMATIC DOCUMENTATION.

OVERVIEW



INSTRUCTOR NOTES

THE LANGUAGE, REQUIREMENTS STATEMENT LANGUAGE (RSL), IS FAIRLY SIMPLE IN APPEARANCE. IT IS VERY "ENGLISH-LIKE" IN STRUCTURE, HAS A RESTRICTED VOCABULARY AND GRAMMAR, AND IS VERY READABLE; IT IS ALSO A MACHINE READABLE LANGUAGE. THIS CHART IS AN EXAMPLE OF RSL. HERE DATA ARE DECLARED AND GIVEN A NAME. THERE ARE VARIOUS ADJECTIVES SUCH AS DESCRIPTION, MAXIMUM VALUE, MINIMUM VALUE, AND UNITS. THE RELATIONSHIPS BETWEEN THAT PARTICULAR PIECE OF DATA AND OTHER ITEMS IS ALSO SHOWN, E.G., "INPUT TO," "PASSED BY." FOR EXAMPLE, THE MESSAGE "OPERATOR_INQUIRY" IS PASSED BY THE "REMOTE_TERMINAL_INPUT_INTERFACE." IT IS MADE UP OF DATA "OPERATOR_ID," "QUERY_TYPE," AND "ITEM_NAME."

THE LANGUAGE ITSELF IS A PRIMITIVE LANGUAGE WHICH CONTAINS FOUR CONCEPTS: 1) ELEMENTS WHICH ARE THE NOUNS OF THE SYSTEM, SUCH AS DATA, MESSAGES, AND INTERFACES; 2) ATTRIBUTES OF ELEMENTS, SUCH AS DESCRIPTION, MAXIMUM VALUE, MINIMUM VALUE; 3) RELATIONSHIPS BETWEEN ELEMENTS SUCH AS INPUT TO, OUTPUT FROM; AND 4) STRUCTURES. RSL ALSO INCLUDES THE ABILITY TO EXTEND THE LANGUAGE AND ADD NEW CONCEPTS. FOR EXAMPLE, "DATA" CAN BE CHANGED TO "INFORMATION," "MAXIMUM VALUE" COULD BE CALLED "MAXIMUM_ALLOWED_VALUE." IF A NEW MEANINGFUL CONCEPT IS NEEDED IT CAN BE ADDED.

REQUIREMENT STATEMENT LANGUAGE

• RSL IS SIMPLE, PRECISE, AND NON-AMBIGUOUS:

DATA:	RANGE_ESTIMATE,
DESCRIPTION:	"THIS DATA ITEM REPRESENTS THE CURRENT ESTIMATE OF SLANT RANGE TO THE TARGET."
MAXIMUM_VALUE:	100000,
MINIMUM_VALUE:	100,
UNITS:	METERS,
INPUT TO:	PREDICT_INTERCEPT,
OUTPUT FROM:	ESTIMATE_TARGET_STATE,
MESSAGE:	OPERATOR_INQUIRY,
PASSED BY:	REMOTE_TERMINAL_INPUT_INTERFACE,
MADE BY:	OPERATOR_ID, QUERY_TYPE, ITEM_NAME,

INSTRUCTOR NOTES

THIS LIST IDENTIFIES THE POSSIBLE TYPES FOR EACH OF THE FOUR PRIMITIVE LANGUAGE CONCEPTS. THERE ARE 21 TYPES OF ELEMENTS, 23 RELATIONSHIPS, 21 ATTRIBUTES, AND 3 STRUCTURES. THIS IS NOT TO INFER THAT ALL TYPES OF ONE CONCEPT CAN BE USED FOR ANY OR ALL OF THE OTHER TYPES. THE LEGITIMATE CORRESPONDENCE CAN BE FOUND IN APPENDIX D OF THE USERS MANUAL.

THE "KERNEL" IS THE NUCLEUS OF INFORMATION CONSISTING OF THE REQUIREMENTS EXPRESSED IN RSL AND USING THE TYPES OF CONCEPTS LISTED ON THIS CHART.

RSL KEYWORDS

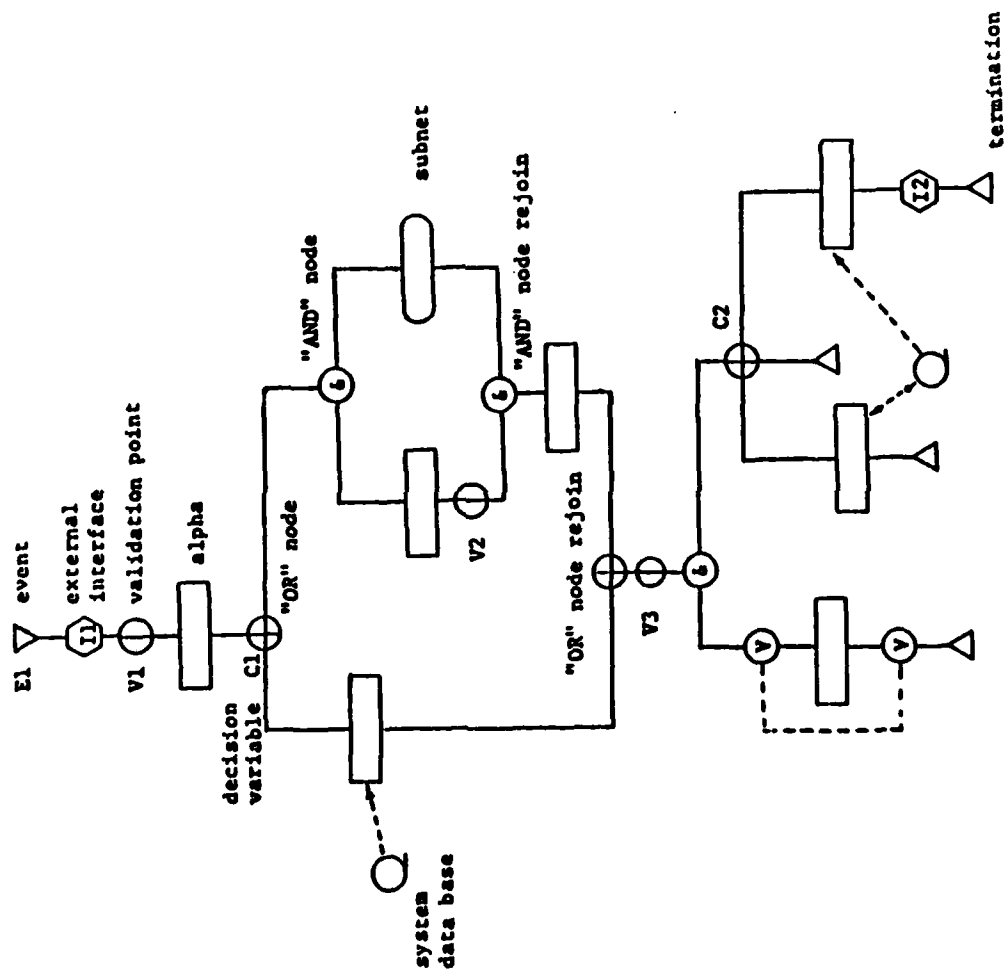
<u>21 ELEMENTS</u>	<u>23 RELATIONSHIPS</u>	<u>21 ATTRIBUTES</u>	<u>3 STRUCTURES</u>
ALPHA	ASSOCIATES	ALTERNATIVES	R NETS
DATA	COMPOSES	ARTIFICIALITY	SUBNETS
DECISION	CONNECTS TO	BETA	VALIDATION_PATH
ENTITY CLASS	CONSTRAINS	CHOICE	
ENTITY_TYPE	CONTAINS	COMPLETENESS	
EVENT	CREATES	DESCRIPTION	
FILE	DELAYS	ENTERED BY	
INPUT INTERFACE	DESTROYS	GAMMA	
MESSAGE	DOCUMENTS	INITIAL_VALUE	
ORIGINATING REQUIREMENT	ENABLES	LOCALITY	
OUTPUT INTERFACE	EQUATES TO	MAXIMUM TIME	
PERFORMANCE_REQUIREMENT	FORMS	MAXIMUM_VALUE	
R NET	IMPLEMENTS	MINIMUM TIME	
SOURCE	INCLUDES	MINIMUM_VALUE	
SUBNET	INCORPORATES	PROBLEM	
SUBSYSTEM	INPUTS	RANGE	
SYNONYM	MAKES	RESOLUTION	
UNSTRUCTURED REQUIREMENT	ORDERS	TEST	
VALIDATION_PATH	OUTPUTS	TYPE	
VALIDATION_POINT	PASSES	UNITS	
VERSION	RECORDS	USE	
	SETS		
	TRACES TO		
	(PLUS THEIR COMPLEMENTS)		

INSTRUCTOR NOTES

HERE'S THE STANDARD R-NET SYNTAX ...

- THESE GRAPHS CAN ALSO BE EXPRESSED IN RSL AS WELL AS BEING ENTERED GRAPHICALLY.

R-NET NOTATION



INSTRUCTOR NOTES

THIS SAMPLE RADX PRINTOUT ILLUSTRATES THE RADX COMMANDS REQUIRED TO LIST ALL MESSAGES NOT FORMED BY AN ALPHA AND AN EXAMPLE OF SUCH A MESSAGE DETECTED BY RADX. USING RADX FOR THIS PURPOSE ELIMINATES THE HUMAN ERROR IF DONE BY HAND, AND DECREASES THE AMOUNT OF TIME REQUIRED.

RADX IS TO SREM WHAT PSA IS TO PSL/PSA.

RADX PROCESSING OF THESE COMMANDS IDENTIFIES ERRORS

```
[RADX COMMAND =
SET OUT_MSG = MESSAGE THAT PASSED OUTPUT_INTERFACE,
-----
SET COUNT = 3

[RADX COMMAND =
SET OUT_MSG_NOT_FORMED = OUT_MSG THAT IS NOT FORMED.
-----
SET COUNT = 1
[RADX COMMAND =
LIST OUT_MSG_NOT_FORMED.
-----
MESSAGE: REQUEST_NEXT_ENGINE.
MADE BY:
DATA: NEXT_CHANNEL_NO
DATA: NEXT_TIME,
PASSED THROUGH:
OUTPUT_INTERFACE: TO_MULTI,

[RADX COMMAND =
SET MSG_NOT_PASSED = MESSAGE THAT IS NOT PASSED.
-----
SET COUNT = 0

[RADX COMMAND =
LIST MSG_NOT_PASSED.
-----
[RADX COMMAND =
SET MULTI_PASSED_MESSAGE = MESSAGE THAT IS MULTIPLE PASSED.
-----
SET COUNT = 0
```

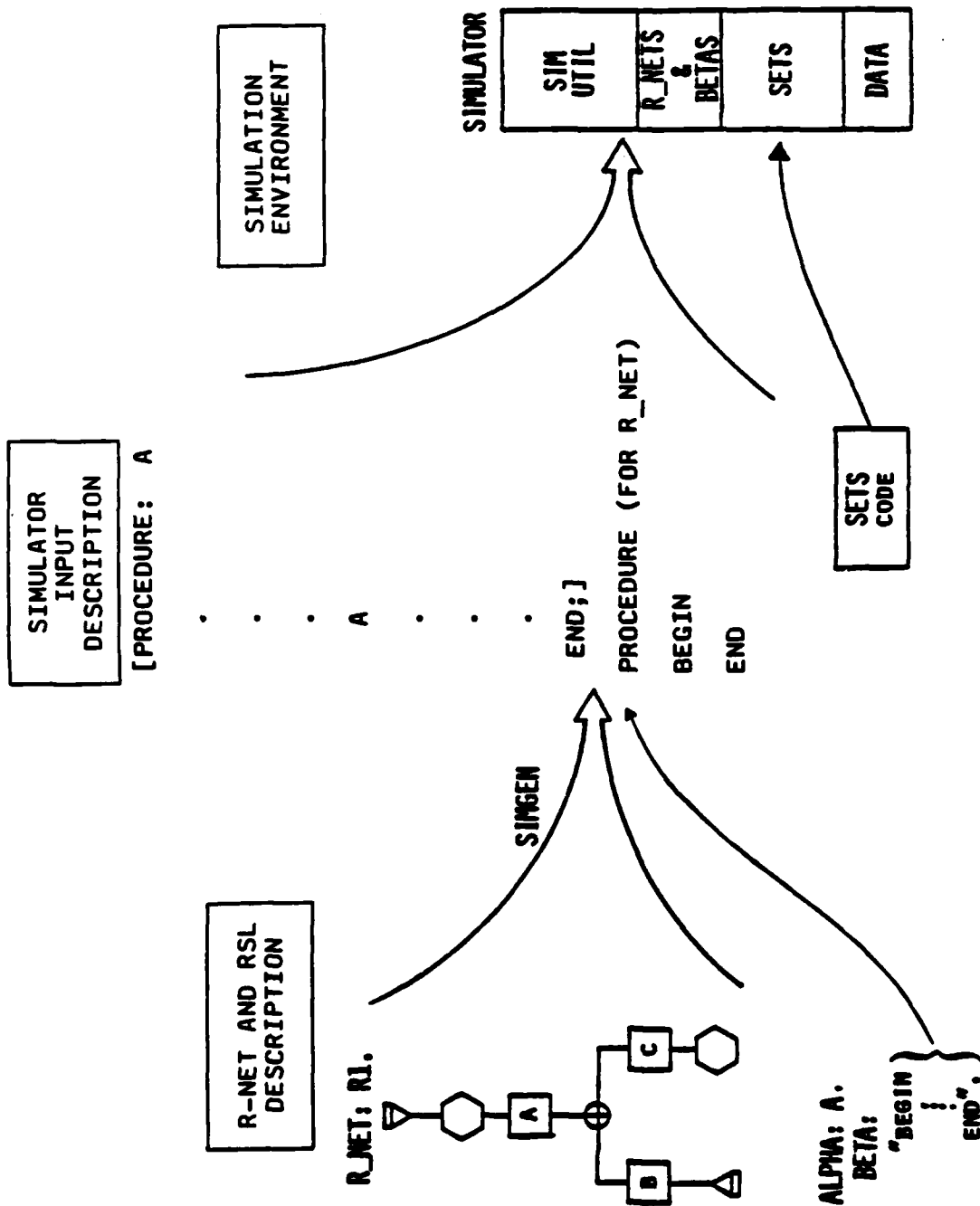
INSTRUCTOR NOTES

THE APPROACH TO FUNCTIONAL SIMULATION IS SHOWN. THE FIRST STEP IS TO DEFINE A BETA FOR EACH ALPHA ON THE R_NET STRUCTURE IN ORDER OF EXECUTION (FIRST TO LAST). THIS SHOULD BE DONE FOR EACH R_NET IN THE SAME ORDER AS THE R_NETS APPEAR, E.G., INITIALIZATION WOULD PROBABLY BE FIRST.

A BETA IS A PASCAL PROCEDURE WHICH MODELS THE TRANSFORMATION OF THE DATA AND FILES INPUT TO AN ALPHA INTO THE DATA AND FILES OUTPUT FROM IT. THE PURPOSE OF A BETA IS TO IMPLEMENT THE DATA FLOW REQUIRED FOR FUNCTIONAL SIMULATION IN ORDER TO DETERMINE THE SUFFICIENCY OF A FUNCTIONAL SPECIFICATION. THUS, IF THE PURPOSE OF AN ALPHA IS TO CHECK MESSAGE VALIDITY, THE BETA IS THE FUNCTIONAL SIMULATION OF HOW THIS SHOULD BE ACCOMPLISHED. HOW BETAS ARE WRITTEN IN PASCAL CODE WILL BE PRESENTED IN SUBSEQUENT CHARTS.

THE R_NETS, BETAS, AND DATA DEFINITIONS ARE INPUT INTO SIMGEN TO GENERATE PASCAL PROCEDURES AND DECLARATIONS. SETS IS USER SUPPLIED AND FUNCTIONALLY DEFINES THE OPERATION OF THE SUBSYSTEMS. THE PASCAL PROCEDURES AND DECLARATIONS ARE COMBINED WITH SIMULATION UTILITIES AND SETS TO FORM A SIMULATION PACKAGE.

DYNAMIC FUNCTIONAL VALIDATION APPROACH



INSTRUCTOR NOTES

SREM PROVIDES AN INTERESTING CONTRAST TO SADT. SREM IS MUCH MORE FORMAL AND IS AUTOMATED, THEREFORE IT IS MUCH BETTER AT ATTAINING CORRECTNESS, AND VERIFIABILITY THAN SADT. WHAT IT GIVES UP IS UNDERSTANDABILITY BECAUSE IT ISN'T AS INTUITIVE AND DOESN'T SUPPORT ABSTRACTION IN THE SAME WAY.

IT DOES PROVIDE LOOKS FOR SIMULATION AND IS MORE EXPLICIT IN SHOWING INHERENT PARALLELISM. THESE PROVIDE MANY BENEFITS INCLUDING EFFICIENCY ANALYSIS.

SREM

PRINCIPLES

- FORMALISM
 - RSL, R-NETS
- MODULARITY
 - SUBNETS, RSL ENTITIES
- STRUCTURING
 - R-NET SYNTAX, RSL
- SEPARATION OF CONCERNS
 - RSL ENTITIES AND RELATIONSHIPS
 - RSL/R-NET PARTITIONING
- UNIFORMITY
 - ENFORCED BY AUTOMATED CHECKING

GOALS

- CORRECTNESS, VERIFIABILITY, TESTABILITY
 - SUPPORTED BY FORMAL LANGUAGE, CONSISTENCY CHECKING, DYNAMIC SIMULATION CAPABILITY
- MAINTAINABILITY, MODIFIABILITY
 - AIDED BY AUTOMATED DATABASE EXTENSIVE REPORTS
- RELIABILITY
 - CONSISTENCY CHECKING AND SIMULATION
- EFFICIENCY
 - CAN BEGIN PARALLELISM AND TUNING ANALYSIS

INSTRUCTOR NOTES

AS WITH SREM WE WILL HIGHLIGHT THE MAJOR COMPONENTS OF PSL/PSA WHICH ARE THE LANGUAGE
AND THE ANALYZER.

VG 744.1

8-181

PSL/PSA

- BACKGROUND

- OVERVIEW

- PSL

- PSA

- CRITIQUE

INSTRUCTOR NOTES

- NOTE THAT PSL/PSA DOESN'T HELP YOU FORMALIZE THE REQUIREMENTS: IT'S JUST A TOOL.
- PSL/PSA IS BASED ON WHAT'S CALLED AN ENTITY/RELATIONSHIP MODEL. BASICALLY, RELATIONSHIPS (SPECIFIED AND IMPLIED) BETWEEN OBJECTS ARE DESCRIBED AND ARE RETRIEVABLE.
- THE TOOL IS CLAIMED TO BE METHODOLOGY INDEPENDENT. HOWEVER, BECAUSE OF THE RELATIONSHIPS AND OBJECTS ABLE TO BE DESCRIBED, AND THE PSL/PSA APPLICATION GUIDEBOOK, AN AD HOC METHODOLOGY EXISTS.
- PSL/PSA WAS ORIGINALLY DEVELOPED (CALLED URL/URA BY THE U.S. AIR FORCE) AS A DOCUMENTATION TOOL. OTHER USES OF THE TOOL FOLLOWED.
- PSL/PSA IS A DESCRIPTION VEHICLE. THAT'S WHERE ITS POWER LIES.

PSL/PSA BACKGROUND

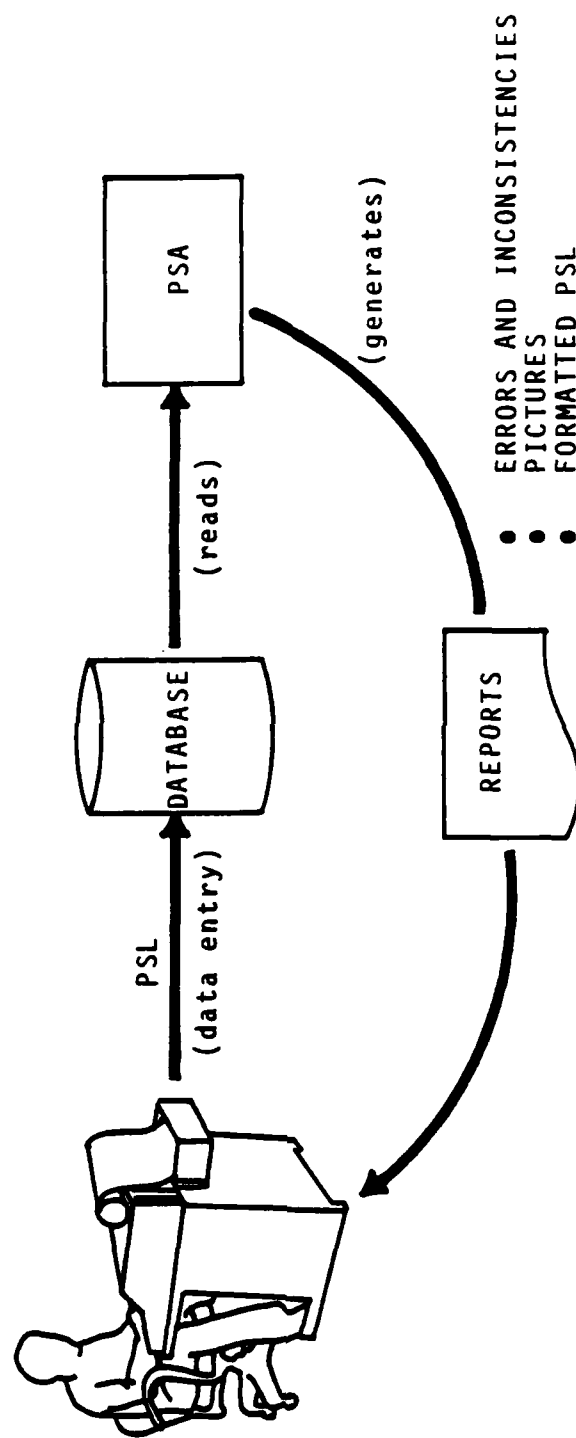
- PSL/PSA WAS DEVELOPED AS AN APPROACH TO IMPROVING SYSTEM/SOFTWARE DEVELOPMENT
- BASED ON AN ENTITY RELATIONSHIP MODEL
- BASED ON THREE PREMISES:
 - MORE EFFORT AND ATTENTION SHOULD BE DEVOTED TO "FRONT-END" PORTIONS OF THE DEVELOPMENT PROCESS
 - MAKE MAXIMUM USE OF AUTOMATION DUE TO LARGE AMOUNT OF INFORMATION THAT MUST BE HANDLED
 - PUT AUTOMATION IN CORRECT PERSPECTIVE - EMPHASIZE NEED FOR DOCUMENTATION
- USE OF PSL/PSA SIMILAR TO THAT OF SREM
- DEVELOPED BY DAN TEICHROEW AT UNIVERSITY OF MICHIGAN
- FAIRLY WIDELY USED ON LARGE SYSTEMS TO RECORD AND TRACK INTERFACES

INSTRUCTOR NOTES

PSL CAN BE INPUT IN A RANDOM MANNER SINCE IT'S NON-PROCEDURAL. PSA WILL STRUCTURE THE INPUT CORRECTLY.

PSA CAN PRODUCE A WIDE VARIETY OF REPORTS IDENTIFYING INCONSISTENCIES AND VARIOUS INTERRELATIONSHIPS OF THE COMPONENTS DESCRIBED IN PSL. THE PICTURE REPORTS ARE VERY BASIC.

INFORMATION CYCLE



INSTRUCTOR NOTES

JUST A QUICK WALKTHROUGH HERE. NOTE THE KEYWORD TITLES ON THE LEFT. HIGHLIGHT THE HIGHLY TEXTUAL FORMAT.

MENTION THAT THE "NOTE" IS ONLY INTERNAL COMMENTING AND IS NOT CHECKED BY THE TOOL.
"STATUS CODE" HAS TO BE DESCRIBED ELSEWHERE TO BE CHECKED.

POINT OUT THAT SUBPARTS SHOWS THE HIERARCHICAL DECOMPOSITION OF THIS PROCESS (FUNCTION)
AND PART OF SHOWS WHAT PROCESS (FUNCTION) THIS ENTITY WAS DECOMPOSED FROM.

WHAT IT LOOKS LIKE

PSL FORMATTED
PROBLEM STATEMENT FRAGMENT

PROCESS

HOURLY-EMPLOYEE-PROCESSING:

GENERATES : PAY-STATEMENT, ERROR-LISTING, HOURLY-EMPLOYEE-REPORT;
RECEIVES : TIME-CARD;
SUBPARTS ARE : HOURLY-PAY-CHECK-VALIDATION, HOURLY-EMP-UPDATE,
HOURLY-REPORT-ENTRY-GENERATION,
HOURLY-PAYCHECK-PRODUCTION;
PART OF : PAYROLL-PROCESSING;
DERIVES : PAY-STATEMENT
USING : TIME-CARD, HOURLY-EMPLOYEE-RECORD;
DERIVES : HOURLY-EMPLOYEE-REPORT
USING : TIME-CARD, HOURLY-EMPLOYEE-RECORD;
DERIVES : ERROR-LISTING
USING : TIME-CARD, HOURLY-EMPLOYEE-RECORD;
PROCEDURE : 1. COMPUTE GROSS PAY FROM TIME CARD DATA.
2. COMPUTE TAX FROM GROSS PAY.
3. SUBTRACT TAX FROM GROSS PAY TO OBTAIN NET PAY.
4. UPDATE HOURLY EMPLOYEE RECORD ACCORDINGLY.
5. UPDATE DEPARTMENT RECORD ACCORDINGLY.
6. GENERATE PAYCHECK.

NOTE:

HAPPENS : IF STATUS CODE SPECIFIES THAT THE EMPLOYEE DID NOT WORK
TRIGGERED BY : THIS WEEK, NO PROCESSING WILL BE DONE FOR THIS EMPLOYEE;
TERMINATION-CLAUSES : NUMBER-OF-PAYMENTS TIMES PER PAY-PERIOD;
SECURITY IS : HOURLY-EMP-PROCESSING-EVENT;
NEW-EMPLOYEE-PROCESSING-EVENT;
COMPANY-ONLY;

INSTRUCTOR NOTES

- (a) AN OBJECT IS ANYTHING GIVEN A PSL NAME BY THE PSL/PSA USER.
- (b) USING THESE OBJECT TYPES AND RELATIONSHIPS IN THIS MANNER, STRUCTURES CAN BE DESCRIBED SUCH AS HIERARCHICAL TREES.
- (c) MENTION THAT THERE ARE OVER 20 OBJECT AND 50 RELATIONSHIPS IN PSL/PSA.

MAJOR PSL KEYWORDS

<u>OBJECTS</u>	<u>RELATIONSHIPS</u>
INPUT	PART OF
OUTPUT	CONTAINED
INTERFACE	USES
PROCESS	DERIVES
SET	UPDATES
ELEMENT	RECEIVES
GROUP	GENERATES
RELATION	CONSISTS
ENTITY	CONSUMES
SYSTEM-PARAMETER	PERFORMED BY
INTERVAL	MAKES...
CONDITION	TERMINATION
EVENT	INCEPTION
	HAPPENS
	TRIGGERS

INSTRUCTOR NOTES

PSA IS THE "HEART" OF THE SYSTEM. OVER 50 REPORTS ARE AVAILABLE FROM WHICH TO ANALYZE THE SYSTEM.

USERS CAN ALSO ADD THEIR OWN REPORTS.

PROBLEM STATEMENT ANALYZER (PSA) REPORTS

- DATABASE MODIFICATION REPORTS
 - RECORD CHANGES THAT HAVE BEEN MADE
 - PROVIDE DIAGNOSTICS AND WARNINGS OF POSSIBLE DATA INCONSISTENCIES
- REFERENCE REPORTS
 - NAME LIST REPORT IDENTIFIES ALL OBJECTS, THEIR TYPE AND DATE OF LAST CHANGE
 - FORMATTED PROBLEM STATEMENT REPORT LISTS ALL PROPERTIES AND RELATIONSHIPS FOR A PARTICULAR OBJECT
 - DICTIONARY REPORT GIVES INFORMATION ABOUT DATA WITHIN THE SYSTEM IN A DATA DICTIONARY FORMAT
- SUMMARY REPORTS
 - DATABASE SUMMARY REPORT PROVIDES PROJECT MANAGEMENT INFORMATION
 - STRUCTURE REPORT SHOWS SYSTEM HIERARCHY
 - EXTENDED PICTURE REPORT SHOWS DATA FLOW GRAPHICALLY

INSTRUCTOR NOTES

THE FORMATTER THAT CAN BE USED TO GENERATE REQUIREMENT SPECIFICATIONS CAN ALSO BE MODIFIED BY THE USER TO MEET HIS SPECIFIC NEEDS. OTHERS HAVE ADDED THEIR OWN FORMATTER.

PROBLEM STATEMENT ANALYZER (PSA) REPORT

- ANALYSIS REPORTS
 - CONTENTS COMPARISON REPORT ANALYZES SIMILARITY OF INPUTS AND OUTPUTS
 - DATA PROCESS INTERACTION REPORT DETECTS GAPS IN INFORMATION FLOW OR UNUSED DATA OBJECTS
 - PROCESS CHAIN REPORT SHOWS THE DYNAMIC BEHAVIOR OF THE SYSTEM
- PSA CAN BE USED TO EXTRACT IN A SEMI-AUTOMATED WAY INFORMATION NEEDED FOR THE REQUIREMENTS SPECIFICATION DOCUMENTS

INSTRUCTOR NOTES

THIS IS AN EXAMPLE OF THE TYPE OF CONDENSED REPORT THAT PSA CAN PRODUCE. THIS IS SHOWING THE INTERACTIONS BETWEEN DATA OBJECTS AND PROCESSES - DATA IS ITEMIZED ON ONE AXIS AND PROCESSES ON THE OTHER. THE D, F, AND R ARE ABBREVIATIONS FOR THE RELATIONSHIPS (DERIVES, RECEIVES) BETWEEN THE PROCESS AND THE DATA.

DATA PROCESS INTERACTION MATRIX

VG 744.1

INSTRUCTOR NOTES

WE SHOULD EXPECT THE PRINCIPLES AND GOALS FOR PSL/PSA TO BE QUITE SIMILAR TO SREM AND THEY ARE. SREM WITH ITS BUILT IN AIDS FOR SIMULATION AND ITS R-NETS HAS MORE SUPPORT FOR TESTABILITY AND GOES FURTHER THAN PSL IN ACTUALLY DESCRIBING FUNCTIONALITY AND BEHAVIOR.

COMPARED TO SADT, PSL CAN BE SEEN TO BE LESS INTUITIVELY UNDERSTANDABLE AND TO HAVE LESS SUPPORT FOR ABSTRACTION.

PSL/PSA

PRINCIPLES

- FORMALISM
 - PSL
- SEPARATION OF CONCERNS, STRUCTURING
 - PSL ENTITIES AND RELATIONSHIPS
- UNIFORMITY
 - PSA CONSISTENCY CHECKING REPORTS
- MODULARITY
 - SPECIFIC PSL RELATIONSHIPS (I.E., SUBPARTS)

GOALS

- CORRECTNESS, VERIFIABILITY
 - PSA REPORTS
- MAINTAINABILITY, MODIFIABILITY
 - AIDED BY AUTOMATED DATABASE PSA REPORTS
- RELIABILITY
 - CONSISTENCY CHECKING BY PSA

INSTRUCTOR NOTES

IN THIS SECTION WE HIGHLIGHT THE (PREDOMINANT) GRAPHIC TECHNIQUES WHICH MAKE UP SSA.

VG 744.1

8-271

SSA

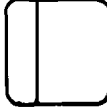
- BACKGROUND
- DATA FLOW DIAGRAMS
- DATA DICTIONARY
- GANE AND SARSON DIAGRAMS AND DATA DESCRIPTION
- CRITIQUE

INSTRUCTOR NOTES

YOURDON DATA FLOW DIAGRAMS ARE ONE OF THE MOST WIDELY USED STRUCTURED ANALYSIS TECHNIQUES. THEY HAVE BEEN POPULARIZED THROUGH SEVERAL TEXTS BY TOM DIMARCO. THESE DFD DIAGRAMS ARE USED TO BUILD HIERARCHICAL MODELS MUCH LIKE SADT.

THE DATA DICTIONARY USES A LIMITED SET OF LOGICAL OPERATORS TO PRODUCE AN ACCURATE AND CONCISE DESCRIPTION OF DATA - THIS EMPHASIS REFLECTS THE WIDESPREAD USE OF THIS TECHNIQUE ON COMMERCIAL DATA PROCESSING APPLICATIONS. RECENT EXTENSIONS HAVE ATTEMPTED TO ADDRESS REAL TIME SYSTEMS BY INCLUDING STATE TRANSITION TABLES AND CONCEPTS SUCH AS EVENTS AND TRIGGERS.

ANOTHER SIMILAR TECHNIQUE HAS BEEN CREATED BY GANE AND SARSON AND USED "SQUARE" BUBBLES

- RECTANGLES WITH ROUNDED CORNERS  AND A SIMILAR BUT LESS PRECISE
DICTIONARY

SSA

- DEVELOPED BY YOURDON AND DEMARCO - SEVERAL TEXTS AVAILABLE
- BUILD MODELS MUCH LIKE SADT
- DATA DICTIONARIES USE A SPECIFIC SET OF OPERATORS
 - = HIERARCHY (COMPRISES)
 - + SEQUENCE (AND)
 - [] SELECTION
 - { } REPETITION
 - () OPTIONAL
- TECHNIQUE IS WIDELY USED IN COMMERCIAL APPLICATION
 - HAS BEEN EXTENDED FOR REAL TIME APPLICATIONS
- GANE AND SARSON HAVE SIMILAR TECHNIQUE WITH SLIGHTLY DIFFERENT GRAPHICS

AD-A165 122

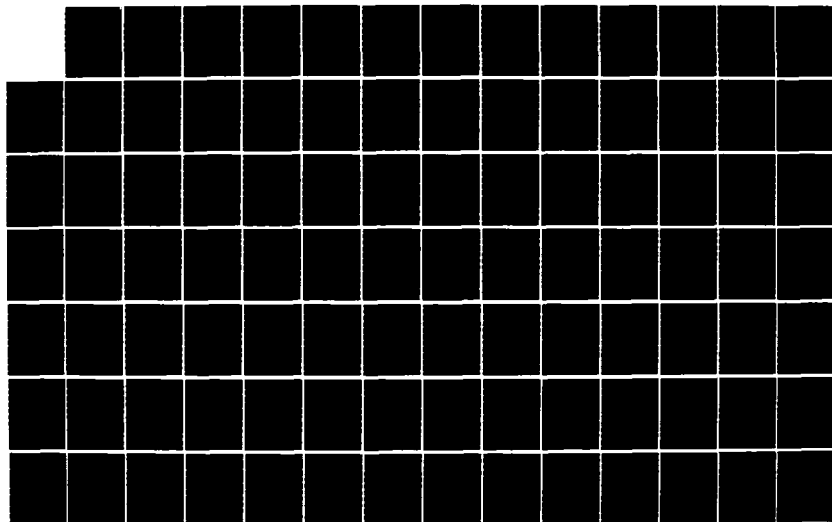
ADA (TRADEMARK) TRAINING CURRICULUM: INTRODUCTION TO
SOFTWARE ENGINEERING M102 TEACHER'S GUIDE(U) SOFTECH
INC WALTHAM MA 1986 DADB07-83-C-K506

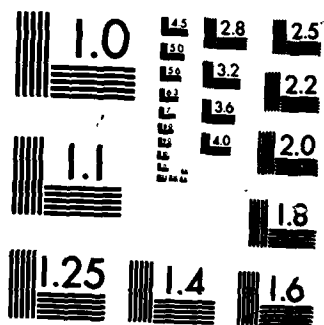
3/6

UNCLASSIFIED

F/G 5/9

NL



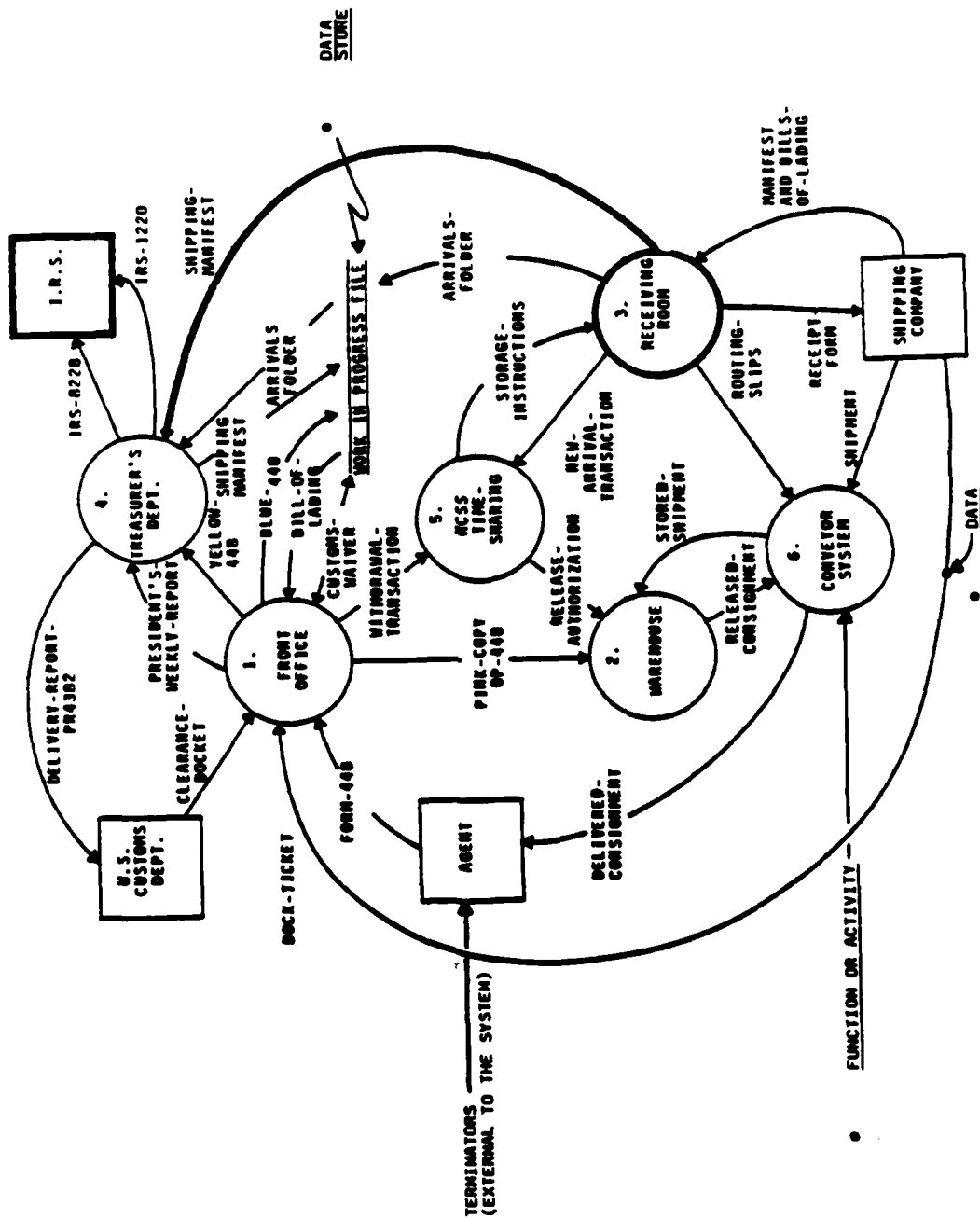


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

- THE SQUARE IS AN EXTERNAL SYMBOL. IT IS THE SOURCE AND/OR DESTINATION OF DATA OUTSIDE THE SYSTEM.
- THE ARROW IS THE DIRECTION OF DATA FLOW. IT'S THE PATHWAY ALONG WHICH DATA MOVES.
- THE "BUBBLE" (CIRCLE) IS THE PROCESS SYMBOL. IT'S JUST A FUNCTION WHICH TRANSFORMS THE DATA.
- THE PARALLEL LINES ARE THE DATA STORE SYMBOL. IT'S THE PLACE IN THE SYSTEM WHERE DATA IS STORED IN SOME WAY (IT MAY BE PEOPLE, CARDS, DISKS, BOOKS, ETC.).

DEMARCO DATA FLOW DIAGRAM



INSTRUCTOR NOTES

- A DATA DICTIONARY DOCUMENTS IN A RIGOROUS WAY THE STRUCTURE OF DATA FLOWS IN THE DIAGRAMS

- DEMARCO DATA DICTIONARY SYMBOLS:

= HIERARCHY (COMPRISES)

+ SEQUENCE (AND)

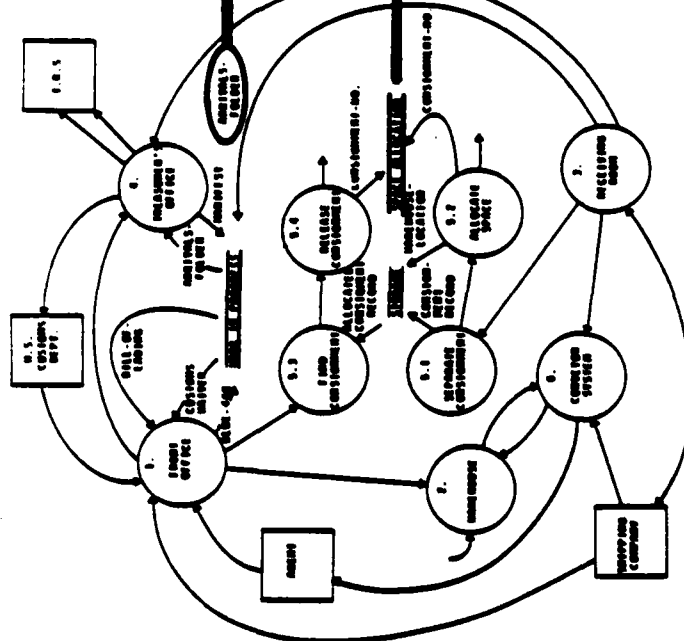
[] SELECTION

{ } REPETITION

() OPTIONAL

DEMARCO

DATA FLOW DIAGRAM



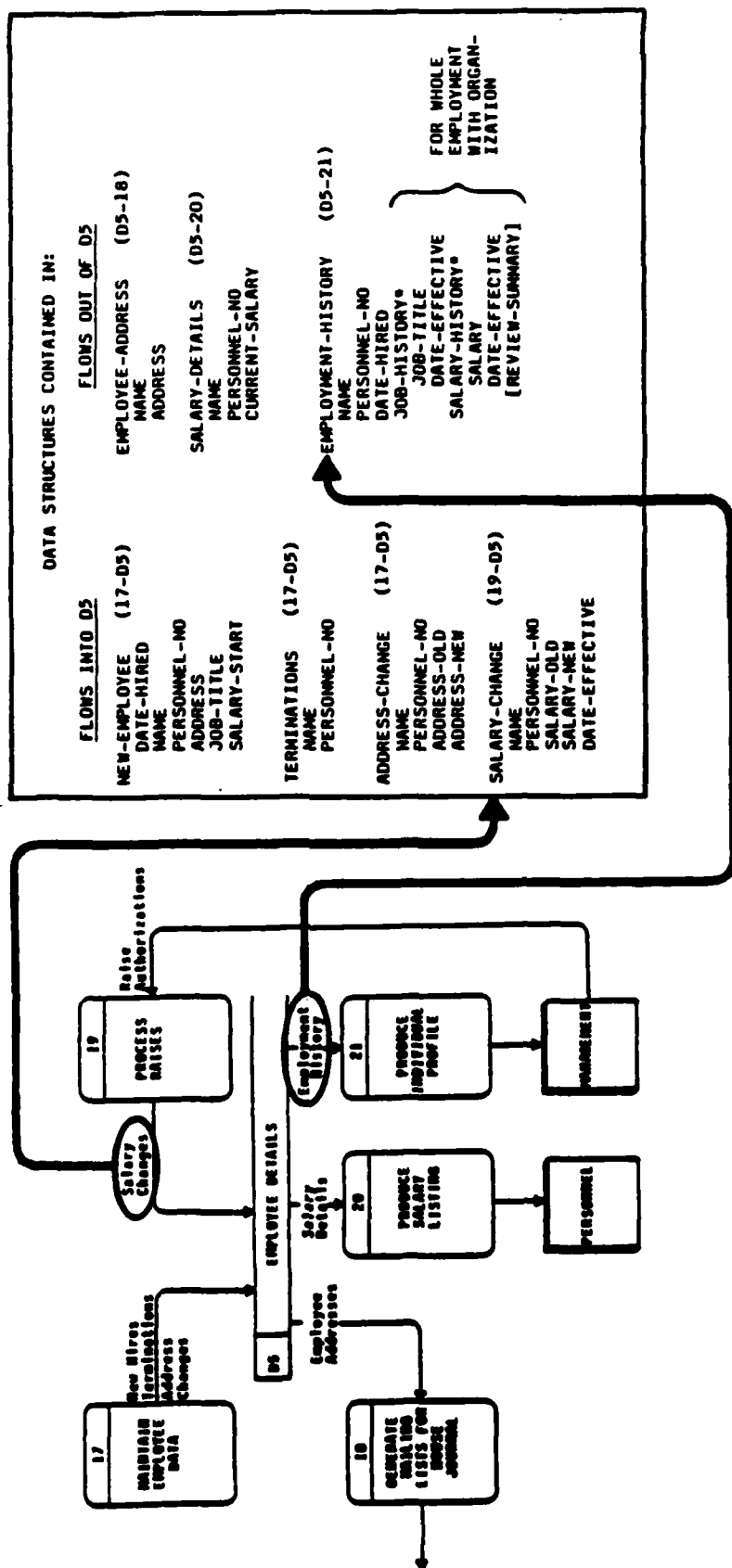
DATA DICTIONARY ENTRIES

- WORK-IN-PROGRESS
 - "FILE OF ALL INFO ABOUT RECEIVED GOODS"
 - [SHIP-NAME • ARRIVAL-FOLDER • (MANIFEST)]
- ARRIVAL FOLDER
 - [BILL-OF-LADING • (ALLOCATION-TICKET) • (CUSTOMS-WAIVER) • (FORM-448)]
- BILL-OF-LADING
 - CONSIGNMENT-NO. • DESTINATION • SHIPPED-WEIGHT • AGENT-NAME • SHIP-NAME • CONSIGNMENT-VALUE • [PARCEL-NO. • PARCEL-DESCRIPTION • PARCEL-VALUE • CUSTOMS-CODE]
- ALLOCATION-TICKET
 - CONSIGNMENT-NO. • WAREHOUSE-LOCATION • RECEIVED-WEIGHT
- CUSTOMS-WAIVER
 - CONSIGNMENT-NO. • AUTHORIZATION-CODE
- FORM-448
 - CONSIGNMENT-NO. • AGENT-NAME • [PARCEL-NO. • PARCEL-DESCRIPTION]
- MANIFEST
 - SHIP-NAME • ARRIVAL-DATE • [CONSIGNMENT-NO. • CONSIGNMENT-VALUE]
- STORAGE-FILE
 - "FILE SHOWING HOW CONSIGNMENTS STORED"
 - [CONSIGNMENT-NO. • NUMBER-OF-SPACES-ALLOCATED • WAREHOUSE-LOCATION • [PARCEL-NO.]]
- SPACE-FILE
 - "FILE OF SPACE ALLOCATION IN WAREHOUSE"
 - [WAREHOUSE-LOCATION • CONSIGNMENT-NO.]

INSTRUCTOR NOTES

- GANE AND SARSON PICTURES ...
 - USE BUBBLE CHART SYNTAX, BUT BUBBLES ARE "SQUARE"
 - EMPHASIZE ONLY DATA FLOWS, SOURCES AND SINKS
 - ARE HIERARCHIC, BUT LACKS DIAGRAM-TO-DIAGRAM INTERFACES
 - USE DEFINED TERMINOLOGY
- GANE AND SARSON DATA DICTIONARY ENTRIES ARE JUST LISTS OF ATTRIBUTES

DATA DESCRIPTION



INSTRUCTOR NOTES

WE WOULD EXPECT SSA TECHNIQUES TO BE VERY SIMILAR TO SADT - MANY PEOPLE CONSIDER THESE TWO TECHNIQUES TO BE INTERCHANGEABLE. HOWEVER, THERE ARE DIFFERENCES AND THESE AFFECT THE RELATIVE IMPORTANCE OF PRINCIPLES AND GOALS. SADT HAS A MORE RIGOROUS MODULARITY CRITERIA - SSA DOESN'T HAVE ANY 3-6 BOX DECOMPOSITION RULES. SSA USES BUBBLES WITHOUT ANY DISTINCTION BETWEEN CONTROLS AND INPUTS SO IN THIS ASPECT SSA IS LESS FORMAL. HOWEVER, SSA'S DATA DICTIONARY IS AN ADDITIONAL FORMALISM OVER WHAT SADT OFFERS.

SSA

PRINCIPLES

- STRUCTURING
 - TOP-DOWN HIERARCHICAL
- ABSTRACTION
 - MULTIPLE LEVEL DECOMPOSITION
- MODULARITY
 - DECOMPOSITION RULES
- FORMALISM
 - SPECIALTY DATA DICTIONARY

GOALS

- UNDERSTANDABILITY
 - USES STRUCTURING AND ABSTRACTION TO AID UNDERSTANDING
- CORRECTNESS, VERIFIABILITY
 - INTUITIVE NATURE OF DIAGRAMS AIDS REVIEW
- MAINTAINABILITY, TRACEABILITY
 - HIERARCHICAL DECOMPOSITION
ALLOWS EASY FOCUSING ON
AREAS OF INTEREST

INSTRUCTOR NOTES

FOR SCRP WE WILL HIGHLIGHT OR PRESENT EXAMPLES OF THE KEY CONCEPTS AND TECHNIQUES USED
IN THIS METHODOLOGY.

VG 744.1

8-331

SCRIP

- BACKGROUND

- SEPARATION OF CONCERNS

- MODE DEFINITIONS

- MODE TRANSITIONS

- FUNCTION DEFINITIONS

INSTRUCTOR NOTES

THE SCRP IS AN OUTGROWTH OF THE WORK DONE BY DAVID PARNAS. THIS PROJECT IS AN ATTEMPT TO APPLY SOFTWARE ENGINEERING PRINCIPLES TO A REAL, EXISTING SYSTEM - THE ONBOARD FLIGHT PROGRAM (OFF) OF THE A-7E. THIS IS THE ONLY FULL SCALE EXPERIMENT AIMED AT DEMONSTRATING THE FULL LIFE CYCLE BENEFITS OF MODERN SOFTWARE ENGINEERING PRINCIPLES. THE PROJECT IS CURRENTLY IN THE INTEGRATION PHASE. SEPARATION OF CONCERNS AND BEING AS FORMAL AS POSSIBLE ARE THE TWO BASIC PRINCIPLES APPLIED TO THE REQUIREMENTS SPECIFICATION PHASE.

THE SEPARATION OF CONCERNS INCLUDES SEPARATING INPUT/OUTPUT MAPPING FROM FUNCTION DESCRIPTIONS AS WELL AS FUNCTIONALITY FROM BEHAVIOR, TIMING, AND ACCURACY.

A VERY FUNDAMENTAL DIFFERENCE WITH THIS APPROACH IS THAT ANY OUTPUT OF THE SYSTEM IS ONLY CONTROLLED OR SET) BY ONE FUNCTION - A 1 TO N RELATIONSHIP EXISTS BETWEEN FUNCTIONS AND OUTPUTS. THIS CREATES LOTS OF SMALL FUNCTIONS.

MODES ARE USED TO SIMPLIFY THE DESCRIPTION OF THE FUNCTIONS AND TO MAKE THE OVERALL COMPLEXITY OF THE SYSTEM EASIER TO COMPREHEND.

SCRIP

- DEVELOPED BY NRL AND NWC, LED BY DAVID PARNAS
- TECHNIQUE BEING USED ON THE REDEVELOPMENT OF THE A-7E OFP
- BASED ON
 - SEPARATION OF CONCERNS
 - FORMALISM
 - ADDITIONAL DESIGN RELATED PRINCIPLES ADDRESSED LATER
- INPUTS AND OUTPUTS DESCRIBED INDEPENDENT OF FUNCTIONS
- 1 TO N RELATIONSHIP EXISTS BETWEEN FUNCTIONS AND OUTPUTS - VERY DIFFERENT FROM TRADITIONAL SPECIFICATIONS
- USES TEMPLATES AND TABLES FOR ADDITIONAL FORMALISM, SEPARATION OF CONCERNS
- USES PRECISELY DEFINED CONCEPTS OF MODES AND EVENTS
- MODES USED TO SIMPLIFY FUNCTION DESCRIPTIONS

INSTRUCTOR NOTES

SEPARATION OF CONCERNS IS A VERY IMPORTANT PRINCIPLE - MOST CURRENT DOCUMENTS INTERMIX CONCERNS. ANY CHANGE TO THE SYSTEM HAS A RIPPLE EFFECT THROUGH THE DOCUMENTS. THIS IS WHERE WE ARE WASTING MOST OF OUR SOFTWARE DOLLARS. THIS PRINCIPLE APPLIES TO ALL DOCUMENTS AND TO THE CODE AS WELL.

AT A HIGH LEVEL, WE ORGANIZE THE DOCUMENT SO THAT INTERFACE (INPUT/OUTPUT MAPPING) INFORMATION IS SEPARATED FROM FUNCTIONAL, WHICH IS SEPARATED FROM TIMING, WHICH IS SEPARATED FROM ACCURACY.

AT A LOWER LEVEL WITHIN EACH OF THESE CATEGORIES WE USE TEMPLATES TO PROVIDE A STANDARD ORGANIZATION TO THE DETAILED INFORMATION.

SEPARATION OF CONCERNS

TOP LEVEL - DOCUMENT TABLE OF CONTENTS

0. INTRODUCTION
1. DISTINGUISHING CHARACTERISTICS OF THE COMPUTER ENVIRONMENT
2. INPUT AND OUTPUT DATA ITEMS
3. MODES OF OPERATION
4. TIME - INDEPENDENT DESCRIPTION OF FUNCTIONS
5. TIMING REQUIREMENTS
6. ACCURACY CONSTRAINTS ON FUNCTIONS
7. UNDESIRABLE EVENT RESPONSES
8. REQUIRED SUBSETS
9. EXPECTED TYPES OF CHANGES
10. GLOSSARY OF ABBREVIATIONS, ACRONYMS, TECHNICAL TERMS, INDICES, DICTIONARY
11. SOURCES OF FURTHER INFORMATION

LOWER LEVEL - TEMPLATE FOR DATA ITEM

INPUT DATA ITEM: MODE ROTARY SWITCH

ACRONYM: /MODEROT/

HARDWARE: TC-2 PANEL

DESCRIPTION: /MODEROT/ INDICATES THE SETTING OF THE MODE ROTARY SWITCH, A SIX POSITION ROTARY SWITCH ON THE TC-2 PANEL

SWITCH NOMENCLATURE: PRES, POS, DEST, MARK, RNG/BRG, D-BHT, ALT-MSLP

DATA TYPE: +ENUMERATION+
CHARACTERISTICS OF VALUES

VALUE ENCODING:

\$None\$	(000000),
\$PRESPOS\$	(100000),
\$DEST\$	(010000),
\$MARK\$	(001000),
\$RNG/BRG\$	(000100),
\$DBHT\$	(000010),
\$ALTMSLP\$	(000001)

INSTRUCTION SEQUENCE: READ 196 (CHANNEL 6)

DATA REPRESENTATION: TC-2 PANEL INPUT WORD 3, BIT 0-5

TIMING CHARACTERISTICS: /MODEROT/ = \$None\$ INDICATES THAT THE SWITCH IS IN TRANSITION BETWEEN TWO POSITIONS

COMMENTS: THE MODE ROTARY SWITCH HAS GROWTH CAPABILITY TO EIGHT POSITIONS

INSTRUCTOR NOTES

THIS TABLE SHOWS FOR EACH MODE WHAT VALUES WILL BE PRESENT FOR THE VARIOUS INPUTS AND TEXT MACROS THAT CAN CAUSE MODE CHANGES. A TABLE LIKE THIS CAN BE SCANNED TO IDENTIFY INCONSISTENT OR OVERLAPPING DEFINITIONS. GRUMMAN HAS INTRODUCED A SLIGHTLY DIFFERENT AND EXPANDED TABLE WHICH HAS A 1 OR 0 IN EACH TABLE ENTRY ALLOWING EASIER CROSS CHECKING.

TABLE 3.3-a: NAVIGATION MODE CONDITIONS

Condition Mode	/IMSMODE/=	/ACAIRB/=	Align. stage completed	!latitude:	Other
DIG	\$Norm\$ OR \$Gndal\$	\$Yes\$	!CA stage:	!s 700	!Doppler up: AND !IMS up:
DI	\$Iner\$ OR \$Norm\$ OR \$Gndal\$	\$Yes\$	!CL stage:	!s 800	!Doppler used: AND !IMS up:
I	\$Iner\$ OR \$Norm\$ OR \$Gndal\$	X	!CL stage:	!s 800	NOT !Doppler used: AND !IMS up:
Mag sl	\$Magsl\$	X	X	!s 800	!IMS up:
Grid	\$Grid\$	X	X	X	!IMS up:
UDI	\$Iner\$	\$Yes\$	NOT !CL stage:	!s 800	!IMS up: AND !Doppler used: AND !pitch small: AND !roll small:
OLB	\$Iner\$ OR \$Norm\$ OR \$Gndal\$ OR	X	NOT !CL stage:	!s 800	X
IMS fail	X	X	X	X	!IMS down:
PolarDI	\$Iner\$ OR \$Norm\$ OR \$Gndal\$	\$Yes\$	!CL stage:	X	!Doppler used: AND !IMS up:
PolarI	\$Iner\$ OR \$Norm\$ OR \$Gndal\$	X	!CL stage:	X	NOT !Doppler used: AND !IMS up:

INSTRUCTOR NOTES

THIS TABLE SHOWS THE POSSIBLE MODE TRANSITIONS - THE ACTUAL CAUSE IS SHOWN IN THE NEXT SLIDE. EACH ROW/COLUMN INTERSECTION WITH A NUMBER SHOWN IS A POSSIBLE MODE TRANSITION. THE NUMBER IS THE ROW AND COLUMN (ALSO INDICATES THE EXITING MODE # AND THE ENTERING MODE #) AND IS USED AS AN IDENTIFIER IN THE FOLLOWING SLIDE TO IDENTIFY THE CAUSING EVENT.

TRANSITION TABLE 3.4: TRANSITIONS BETWEEN ALIGNMENT, CALIBRATION AND NAVIGATION MODES

EXITED MODE

ENTERED MODE	*Lauto cal*	*Sauto cal*	*Landaln*	*SINSaln*	*OI Update*	*HUDaln*
Lautocal						
Sautocal						
Landaln	3-1	2-2	1-3	2-4		3-6
SINSaln			3-3	3-4		
OIupdate		4-2	4-3	4-4		
HUDaln	5-1		5-3			5-6
Airaln	6-1					6-6
DIG						
DI						
I						
UDI			10-3	10-4	10-5	10-6
OLB			12-3	12-4		
Mag sl			13-3	13-4		12-6
Grid			14-3	14-4		
IMS fail			15-3			
PolarDI						
PolarI			17-3	17-4	17-5	
Grtest			18-3	18-4		

INSTRUCTOR NOTES

THIS MODE TRANSITION EVENT SECTION OR TABLE SHOWS THE EXACT CAUSE (OR POTENTIAL CAUSES)
FOR EACH MODE TRANSITION IDENTIFIED IN THE PREVIOUS SLIDE.

VG 744.1

8-381

VERSION 3

```

1-3  *Lautocal* TO *Landaln*
      @T(:present position entered!)
1-7  *Lautocal* TO *Airaln*
      @T(/ACAIRB/=Yes$) WHEN (:Doppler up!)
1-10 *Lautocal* TO *I*
      @T(:ND stage: completed) WHEN (:latitude: lseq 80o)
1-12 *Lautocal* TO *OLB*
      @T(:present position entered!) WHEN (/IMSMODE/-=$Norm$ OR $Iner$)
      @T(/ACAIRB/=Yes$) WHEN (:Doppler down!)
      @T(:present position entered!)
1-13 *Lautocal* TO *Mag sl*
      @T(:present position entered: WHEN(/IMSMODE/=$Magsl$)
1-14 *Lautocal* TO *Grid*
      @T(:present position entered!) WHEN(/IMSMODE/=$Grid$)
1-15 *Lautocal* TO *IMS fail*
      @T(:IMS down!)
1-17 *Lautocal* TO *PolarI*
      @T(:ND stage: completed) WHEN (:latitude: gt 80o)
1-18 *Lautocal* TO *Grtest*
      @T(/PNLTEST/=$TEST$)
2-2  *Sautocal* TO *Sautocal*
      @T(NOT :SINS valid: for more than 2 sec)
2-4  *Sautocal* TO *SINSaln*
      @T(:present position entered!)
2-7  *Sautocal* TO *Airaln*
      @T(/ACAIRB/=Yes$) WHEN (:Doppler up!)
2-10 *Sautocal* TO *I*
      @T(:ND stage: completed) WHEN (:latitude: lseq 80o)
2-12 *Sautocal* TO *OLB*
      @T(/ACAIRB/=Yes$) WHEN (:Doppler down!)
      @T(:present position entered!) WHEN (/IMSMODE/-=$Norm$ OR $Iner$)

```

INSTRUCTOR NOTES

ALL FUNCTION DESCRIPTIONS INDICATE THE ASSOCIATED OUTPUT DATA ITEMS PRODUCED BY THE FUNCTION, THUS PROVIDING A CROSS REFERENCE TO THE DATA ITEM DESCRIPTIONS. THE LIST OF MODES IS TO PROVIDE THE READER WITH AN OVERVIEW OF WHEN THE FUNCTION IS PERFORMED, THIS OVERVIEW IS REFINED IN THE FUNCTION DESCRIPTION PORTION OF THE TEMPLATE.

THE EVENT TABLE SHOWS THE EVENTS (I.E., A TRANSITION OF A CONDITION) THAT REQUEST THE FUNCTION TO BE PERFORMED AND THE MANNER IN WHICH THE OUTPUT IS TO BE PRODUCED. SYMBOLIC NAMES ARE USED FREELY TO PROVIDE AN INDEPENDENCE BETWEEN VALUES AND NAMES.

4.1.5 Demand Function Name: Change scale factor

Modes in which function required:

Alignment: *Lautocal*, *Sautocal*, *Landaln*, *SINSaln*,
 OUpdate, *HUDaln*
 Navigation: *DIG*, *DI*, *I*, *UDI*, *OLB*, *PolarDI*,
 PolarI

Output data item: //IMSSCAL//

Function Requested and Output Description:

EVENT TABLE 4.1-h: WHEN THE SCALE FACTOR IS CHANGED

MODES	EVENTS	
Lautocal *Landaln* *O1 Update*	@T(In mode) WHEN (/IMSSCAL// = \$Coarse\$)	X
HUDaln	@T(In mode) WHEN (/IMSMODE/ = \$Gndal\$ AND //IMSSCAL// = \$Coarse\$)	@T(In mode) WHEN (/IMSMODE/ = (\$Norm\$ OR \$Iner\$) AND //IMSSCAL// = \$Fine\$)
Sautocal *SINSaln* *Airaln* All navigation modes listed	X	@T(In mode) WHEN (/IMSSCAL// = \$Fine\$)
ACTION	//IMSSCAL// = \$Fine\$	//IMSSCAL// = \$Coarse\$

INSTRUCTOR NOTES

SCRIP IS VERY DIFFERENT FROM OTHER METHODS AND THEREFORE WE EXPECT TO SEE A DIFFERENT EMPHASIS ON GOALS AND UNDERLYING PRINCIPLES.

THE UNIQUE USE OF SEPARATION OF CONCERNS AND A DIFFERENT TYPE OF FORMALISM RESULT IN A HIGHLY MAINTAINABLE SPECIFICATION - THE IMPACT OF ANY CHANGE TO THE SYSTEM IS WELL CONTROLLED AND LOCALIZED.

UNDERSTANDABILITY IS ENHANCED BECAUSE THE SPECIFICATION IS PARTITIONED INTO MANY SMALL EASILY GRASPED COMPONENTS, HOWEVER THE "BIG PICTURE" IS NOT AS EASY TO GRASP AS WITH SADT, BUT A MUCH MORE DETAILED, PRECISE, CONCISE DESCRIPTION IS POSSIBLE WITH SCRIP.

SADT AND SCRIP CAN BE COMBINED TO GET THE BEST OF BOTH WORLDS - WRITE TO NEWPORT FOR DETAILS.

SCRIP

PRINCIPLES

- SEPARATION OF CONCERNS
 - MAXIMIZED AT MULTIPLE LEVELS
- FORMALISM, UNIFORMITY
 - LIMITED SET OF PRIMITIVE CONSTRUCTS, HEAVY USE OF TABLES AND TEMPLATES
- MODULARITY, ABSTRACTION
 - PARTITIONING INTO MODES, AND SMALL FUNCTIONS
- STRUCTURING
 - ATTAINED THROUGH SEPARATION OF CONCERNS

GOALS

- MAINTAINABILITY, MODIFIABILITY
 - SEPARATION OF CONCERNS LIMITS IMPACT OF CHANGES
- CORRECTNESS, VERIFIABILITY, TESTABILITY
 - FORMALISM AND STANDARD TABLES AND TEMPLATES FACILITATE REVIEW
- UNDERSTANDABILITY
 - SEPARATION OF CONCERNS PARTITIONS COMPLEXITY TO IMPROVE OVERALL UNDERSTANDABILITY
- PRODUCTIVITY
 - SEPARATION OF CONCERNS ALLOWS INEXPERIENCED ENGINEERS TO CONTRIBUTE EFFECTIVELY

INSTRUCTOR NOTES

THEME: DESIGN CONSISTS OF TWO SEPARATE SETS OF ACTIVITIES WHICH TAKE THE OUTPUT OF THE ANALYSIS (AND SPECIFICATION) PHASE OF THE LIFE CYCLE AND CREATES A "PAPER" MODEL OF THE SOFTWARE.

PURPOSE: TO PROVIDE INSIGHT INTO THE MAJOR ASPECTS OF THE DESIGN PHASE OF THE LIFE CYCLE.

REFERENCES: WEINBERG, G., "RETHINKING SYSTEMS ANALYSIS AND DESIGN"
LITTLE, BROWN, MA; 1982

SECTION 9

DESIGN OVERVIEW

VG 744.1

INSTRUCTOR NOTES

DESIGN IS A BLUEPRINT OF MODULES AND THEIR INTERCONNECTIONS. THE DESIGN PROCESS ALLOCATES THE FUNCTIONAL REQUIREMENTS TO A DESIGN STRUCTURE, PRESENTING THE FORM OR DESIGN STRUCTURE ALONG WITH SOME INDICATION OF WHERE THE VARIOUS FUNCTIONS ARE TO BE PERFORMED. THE STRUCTURE AND ALLOCATION SHOULD ALLOW THE PERFORMANCE AND ANALYTIC REQUIREMENTS TO BE FACTORED IN AND VERIFIED AS CONSTRAINTS.

THE INTERFACES BETWEEN THE COMPONENTS OF THE STRUCTURE ARE ALSO DEFINED AT THIS TIME. (MORE EMPHASIS IS PLACED ON THESE INTERFACES (E.G. PARNAS) THAN OTHERS (E.G. STRUCTURED DESIGN)).

THE ACTIVITY OF DESIGN IS A MODELING ACTIVITY. A DESIGN LEAVES CERTAIN DETAILS OUT UNTIL LATER. THERE IS A NEED TO BE ABLE TO COMPREHEND A LARGE AMOUNT OF THE SYSTEM TO BE SURE THAT THE GOALS OF THE PARTICULAR DESIGN STRATEGY (WHICH VARY) ARE BEING MET. DURING LATER STAGES OF THE DESIGN PROCESS DETAIL WILL BE ADDED AS WE EXAMINE SMALLER PARTS OF THE DESIGN.

THE DIFFERENT MODELING TECHNIQUES WE WILL COVER ENCOURAGE US TO LEAVE OUT DIFFERENT THINGS. LEAVING OUT THE DETAILS IS THE HARDEST PART OF THE DESIGN PROCESS.

WHAT IS DESIGN?

- DESIGN ...
 - TRANSLATES REQUIREMENTS SPECIFICATIONS INTO A BLUEPRINT OF THE SYSTEM
 - IS A MODEL OF THE SOFTWARE
 - IS DONE BY DESIGNERS
- FOR OUR PURPOSE DESIGN CONSISTS OF TWO MAJOR SUBPHASES
 - ARCHITECTURAL DESIGN (PRELIMINARY DESIGN)
 - DETAILED DESIGN

INSTRUCTOR NOTES

DESIGNERS DEAL WITH MORE OF THE SYSTEM AT ONE TIME THAN IMPLEMENTERS. THEY CONCENTRATE ON GLOBAL ISSUES, POSTPONING DECISIONS HAVING ONLY LOCAL SCOPE. THE DESIGNER PARTITIONS THE SYSTEM IN A MANNER THAT HIDES DECISIONS LIKELY TO CHANGE SEPARATELY INTO INDIVIDUAL MODULES.

CLEAN INTERFACES REALLY MEAN FULLY DEFINED INTERFACES. THE SIMPLER THESE INTERFACES ARE - FEWER OPTIONS, FEWER PARAMETERS - THE MORE LIKELY IT IS TO BE FOR THEM TO BE FULLY DEFINED. COMPLEX INTERFACES PROBABLY MEAN TOO MANY FUNCTIONS IN A SINGLE MODULE.

THE DESIGNER COMMUNICATES WITH THE ANALYST EITHER VERBALLY OR BETTER YET, THROUGH DOCUMENTATION TO MAP FUNCTIONAL REQUIREMENTS (AS WELL AS OTHERS SUCH AS RELIABILITY, MODIFIABILITY, ETC.). THE DESIGNER WORKS WITH IMPLEMENTERS TO ASSESS PERFORMANCE AND OTHER GOALS IN ORDER TO MAKE TRADE-OFFS AMONG THEM.

THE INTERFACE BETWEEN DESIGNERS AND ANALYSTS IS PROBABLY MORE DIFFICULT THAN BETWEEN DESIGNERS AND IMPLEMENTERS. DESIGNERS USUALLY WERE IMPLEMENTERS ONCE AND UNDERSTAND THE AREA WELL.

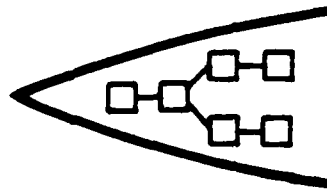
DESIGNER'S ROLE

- DESIGNERS ...
 - POSTPONE IMPLEMENTATION DECISIONS
 - HIDE THEIR DECISIONS INSIDE MODULES
 - WORRY ABOUT THE SOFTWARE'S STRUCTURE
 - SPECIFY ALGORITHMS AND CONTROL FLOW
 - MAKE CLEAN INTERFACES
 - COMMUNICATE WITH ANALYSTS AND IMPLEMENTERS

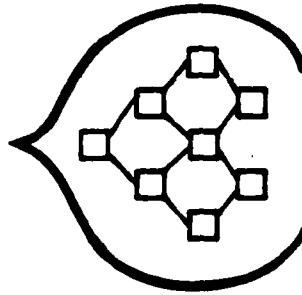
- THE DESIGNER IS THE BRIDGE BETWEEN ANALYSTS AND IMPLEMENTERS

INSTRUCTOR NOTES

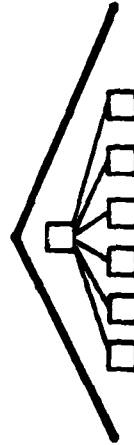
IF YOU WANT TO HAVE A GOOD ARGUMENT ON REQUIREMENTS VERSUS DESIGN JUST VISIT AN ORGANIZATION WHERE ONE GROUP DOES ONE AND ANOTHER DOES THE OTHER. "WHAT" VS "HOW" SOUNDS SIMPLE BUT IT IS VERY HARD TO GET SPECIFIC ABOUT WHAT SOMETHING DOES WITHOUT PROPOSING A HOW. ADDING TO THE DIFFICULTIES IS THE FACT THAT PEOPLE GO FROM IMPLEMENTER TO DESIGNERS TO ANALYSTS. THEY HAVE A HARD TIME FORGETTING WHAT THEY PREVIOUSLY DID. THE DESIGN "BLUEPRINT" MUST DEPICT THE OVERALL STRUCTURE:



TOWER
(BAD)



MOSQUE
(GOOD)



PANCAKE
(BAD)

SOME MEANS OF JUDGING OVERALL STRUCTURE IS IMPORTANT - GRAPHICS ARE NICE AND PREFERRED BUT OTHER METHODS ARE ACCEPTABLE (E.G., INDENTED HIERARCHICAL DIAGRAM).

GENERAL GUIDELINES FOR ARCHITECTURAL DESIGN

- THE BOUNDARY BETWEEN REQUIREMENTS AND DESIGN IS UNCLEAR (AND ALWAYS WILL BE)
 - IMPLIES ITERATION BETWEEN ANALYSIS AND DESIGN
- NO SINGLE METHOD GIVES A COMPLETE PROCESS ON HOW TO GO FROM REQUIREMENTS TO DESIGN
 - SOFTWARE COST REDUCTION PROJECT (SCRIP) COMES CLOSE
- ALWAYS BUILD A PICTURE OF THE SOFTWARE ARCHITECTURE
 - MAKE ARCHITECTURE VISIBLE
 - SHOW INTERRELATIONSHIP BETWEEN COMPONENTS THAT MAKE UP ARCHITECTURE
- MAKE YOUR GOAL A SET OF DELIVERABLES

INSTRUCTOR NOTES

THESE ARE THE ELEMENTS REQUIRED BY SDS. EACH DELIVERED ITEM MUST BE TECHNICALLY
ADEQUATE, UNDERSTANDABLE AND COMPREHENSIVE.

DoD-STD-SDS VIEW OF ARCHITECTURAL DESIGN

- SDS PRELIMINARY DESIGN ACTIVITIES
 - ASSIGN FUNCTIONAL AND PERFORMANCE SOFTWARE REQUIREMENTS TO TOP LEVEL SOFTWARE COMPONENTS
 - USE TOP-DOWN DESIGN APPROACH
 - ESTABLISH TOP-LEVEL DESIGN
 - DETERMINE FLOW OF DATA AND EXECUTION OF CONTROL BETWEEN COMPONENTS
 - SPECIFY TOP-LEVEL ALGORITHMS
 - USE A PROGRAM DESIGN LANGUAGE (PDL) AT TOP-LEVEL DESIGN
 - DEVELOP TEST PLANS
 - DEVELOP USER MANUALS (IF NEEDED)

INSTRUCTOR NOTES

SDS CONTINUED ...

VG 744.1

9-51

DoD-STS-SDS VIEW OF ARCHITECTURAL DESIGN

- SDS PRELIMINARY DESIGN PRODUCTS
 - RECORDS OF REVIEWS, ACTION ITEMS, ETC.
 - SOFTWARE TOP-LEVEL DESIGN DOCUMENT
 - SOFTWARE TEST PLAN
 - COMPUTER SYSTEM OPERATOR'S MANUAL, SOFTWARE USER'S MANUAL, COMPUTER DIAGNOSTIC MANUAL (IF NEEDED)
- SDS PRELIMINARY DESIGN REVIEWS
 - A PRELIMINARY DESIGN REVIEW
 - REVIEWS DESIGN PRODUCTS
 - DEMONSTRATES TRACEABILITY BACK TO THE REQUIREMENTS

INSTRUCTOR NOTES

STRESS THAT DETAILED DESIGN IS A TRANSITIONARY ACTIVITY.

- (a) REVIEW THE DIFFERENCES BETWEEN REQUIREMENT AND SPECIFICATIONS ON ONE HAND (i.e., DETERMINE WHAT IS TO BE DONE) AND DESIGN ON THE OTHER (i.e., HOW IT IS TO BE DONE, THE ANALYSIS OF VARIOUS ALTERNATIVES, AND THE TRADE-OFFS MADE BASED ON THE CONSTRAINTS SPECIFIED).
- (b) DESIGN ACTIVITY IS DIVIDED INTO 2 DISTINCT PHASES: ARCHITECTURAL AND DETAILED:
 - (1) ARCHITECTURAL EMPHASIZES STRUCTURE OF THE SYSTEM, THE DECOMPOSITION OF THE SYSTEM INTO MODULES, AND THE PRECISE SPECIFICATION OF THE INTERFACES BETWEEN THEM.
 - (2) DETAILED DESIGN EMPHASIZES THE SELECTION AND EVALUATION OF THE ALGORITHMS NECESSARY TO CARRY OUT THE LOGICAL STEPS SPECIFIED FOR INDIVIDUAL MODULES.
- (c) DESIGN IS A VERY ITERATIVE PROCESS.

ASPECTS OF DETAILED DESIGN

- DETAILED DESIGN SPANS THE GAP FROM ARCHITECTURAL DESIGN
TO CODE
- THERE IS CONTROVERSY ABOUT WHEN DETAILED DESIGN BEGINS AND
ENDS AND WHAT FORM IT TAKES
- EVERYONE AGREES THAT DETAILED DESIGN PRODUCES SPECIFICATIONS
SUFFICIENT TO CODE FROM

INSTRUCTOR NOTES

THE LINES ARE EVEN FUZZIER BETWEEN DETAILED DESIGN AND ARCHITECTURAL DESIGN AND CODING, THAN BETWEEN REQUIREMENTS AND ARCHITECTURAL DESIGN.

THE "CONTROVERSY" EXISTS PRECISELY BECAUSE DESIGN IS AN ITERATIVE PROCESS. BOTH DETAILED AND ARCHITECTURAL DESIGN SERVE TO TIGHTEN UP ANY LOOSE POINTS IN THE SPECIFICATIONS, AND HELPS DEFINE FOR THE PROGRAMMER THE STEPS NEEDED TO BE INCLUDED IN THE ACTUAL IMPLEMENTATION. ITERATION BETWEEN THE SUB-PHASES AFFECTS THE DISTINCTION BETWEEN THE TWO.

DETAILED DESIGN METHODOLOGIES

- MOST ARCHITECTURAL DESIGN METHODS STOP WHEN THE MODULE OR UNIT LEVEL IS REACHED

- SO MANY DESIGN METHODOLOGIES ADDRESS DESIGN ISSUES INSIDE

A MODULE EMPHASIZING ONE OR MORE OF THE FOLLOWING:

- PHYSICAL STORAGE (FILES, DATABASES, ETC.)
- LOGIC FLOW
- ALGORITHM
- DATA STRUCTURE

INSTRUCTOR NOTES

AGAIN, EACH DELIVERED ITEM MUST BE TECHNICALLY ADEQUATE, UNDERSTANDABLE, AND COMPREHENSIVE.

DoD-STD-SDS VIEW OF DESIGN

- SDS DETAILED DESIGN ACTIVITIES
 - REFINE TOP-LEVEL SOFTWARE COMPONENTS TO LOWER-LEVELS
 - PERFORM UNTIL A SINGLE, NON-DIVISIBLE FUNCTION IS DEFINED
 - USE A TOP-DOWN APPROACH
 - USE BOTTOM-UP APPROACH ONLY ON CRITICAL LOWER-LEVEL UNITS (I.E. COMMON ROUTINES, EXECUTIVE, ETC.)
 - USE A PROGRAM DESIGN LANGUAGE
 - ESTABLISH SOFTWARE DEVELOPMENT FOLDERS FOR ALL SOFTWARE UNITS CONTAINING:
 - UNIT REQUIREMENTS
 - DESIGN INFORMATION
 - CODE
 - TEST PROCEDURES
 - UNIT SCHEDULE
 - STATUS
 - DEVELOP TEST FOR EACH UNIT
 - DEVELOP INTEGRATION TEST CASES

INSTRUCTOR NOTES

SDS DESIGN VIEW CONTINUED ...

VG 744.1

9-91

DoD-STD-SDS VIEW OF DESIGN

- SDS DETAILED DESIGN PRODUCTS
 - SOFTWARE DETAILED DESIGN DOCUMENT
 - INTERFACE DESIGN DOCUMENT
 - DATABASE DESIGN DOCUMENT
 - SOFTWARE TEST DESCRIPTION
 - SOFTWARE DEVELOPMENT FOLDER
 - SOFTWARE PROGRAMMER'S MANUAL*
 - FIRMWARE SUPPORT MANUAL*
- SDS DETAILED DESIGN REVIEWS
 - CRITICAL DESIGN REVIEW OF PRODUCTS ABOVE

*PRODUCED IF REQUIRED BY PROCURING AGENCY

INSTRUCTOR NOTES

VG 744.1

10-1

SECTION 10

DESIGN METHODS

VG 744.1

INSTRUCTOR NOTES

AS WITH THE ANALYSIS TECHNIQUES WE WILL PRESENT AN INTERMEDIATE LEVEL DESCRIPTION OF THESE FIVE TECHNIQUES DISCUSSING THEM BY COMPONENTS OR GIVING EXAMPLES OF THE MAJOR TECHNIQUES OR FEATURES.

AGAIN, WE WILL PRESENT A CRITIQUE USING OUR PRINCIPLES AND GOALS AS A YARDSTICK.

OVERVIEW

- SCRP - SOFTWARE COST REDUCTION PROJECT
- OBJECT ORIENTED
- SD - STRUCTURED DESIGN
- JACKSON/WARNIER-ORR
- HOS - HIGHER ORDER SOFTWARE

INSTRUCTOR NOTES

IN THIS SECTION WE WILL HIGHLIGHT SEVERAL ASPECTS OF THE SCRP DESIGN TECHNIQUES. THESE ARE PRIMARILY TEXTUAL AND TABULAR IN NATURE. SEVERAL PAPERS EXIST COVERING THE SCRP MODULARIZATION APPROACH AND THEIR TECHNIQUE FOR DEFINING ABSTRACT INTERFACES. NO PAPERS COMPLETELY COVER THE TRACEABILITY ISSUE. ALL SCRP DESIGN DOCUMENTS ARE AVAILABLE.

SCRIP

- BACKGROUND
- ABSTRACTION, MODULARITY, HIDING
- GENERIC MODULE HIERARCHY
- DESIGN TRACEABILITY
- DESIGN EXAMPLE
- ABSTRACT INTERFACE TEMPLATE
- CRITIQUE

INSTRUCTOR NOTES

NOTE - READ SCRP PAPERS ON MODULE GUIDE AND ABSTRACT INTERFACES.

THIS PROJECT HAS SUCCESSFULLY APPLIED THE PRINCIPLE OF INFORMATION HIDING TO A COMPLEX SYSTEM AND HAS DESIGNED AND DOCUMENTED THE SYSTEM USING THIS APPROACH. ALL DOCUMENTS ARE READILY AVAILABLE AND SHOW HOW PRINCIPLE WAS APPLIED.

THIS APPROACH REQUIRES THE ANALYST AND DESIGNER TO PREDICT WHAT REQUIREMENTS ARE LIKELY TO CHANGE AND THEN TO PARTITION THE MODULES OF THE SYSTEM SO THAT THE EFFECT OF THESE CHANGES IS LIMITED (I.E., IF WE EXPECT A PARTICULAR SENSOR TO POSSIBLY BE REPLACED, THEN WE DESIGN A MODULE WHICH PROVIDES AN INTERFACE TO THE REST OF THE SYSTEM TO PROVIDE THE ESSENTIAL INFORMATION OF THE SENSOR IN A MANNER THAT WON'T BE CHANGED IF WE CHANGE THE SPECIFIC SENSOR).

BECAUSE OF THE SEPARATION OF CONCERNS HAS BEEN APPLIED TO BOTH THE FUNCTIONAL REQUIREMENTS DESCRIPTION AND THE DESIGN DESCRIPTION IT IS POSSIBLE TO BUILD IN AN EXTREMELY HIGH DEGREE OF TRACEABILITY FROM REQUIREMENTS TO DESIGN.

THE HIGHER LEVEL OF THE DESIGN DECOMPOSITION ARE APPLICABLE TO A WIDE CLASS OF EMBEDDED SYSTEMS. DAVE PARNAS HAS BET HIS RIGHT ARM THAT THE FIRST LEVEL DECOMPOSITION IS UNIVERSALLY APPLICABLE AND HIS LEFT TOE THAT THE SECOND LEVEL IS UNIVERSALLY REUSABLE.

SCRIP BACKGROUND

- MODULAR DECOMPOSITION BASED ON INFORMATION HIDING
- INDEPENDENT MODULES HIDE DECISIONS THAT ARE LIKELY TO CHANGE INDEPENDENTLY
- MODULES ARE DESCRIBED USING STANDARD ABSTRACT INTERFACE TEMPLATE
 - ABSTRACT BECAUSE IT ONLY SHOWS THAT PART OF THE INTERFACE TO THE MODULE THAT WILL NOT CHANGE WHEN A HIDDEN DECISION (INFORMATION) CHANGES
- EXTREMELY HIGH DEGREE OF TRACEABILITY FROM FUNCTIONAL REQUIREMENTS TO DESIGN DESCRIPTION
- HIGH LEVEL DESIGN DECOMPOSITION; FIRST TWO LEVELS ARE CONSIDERED TO BE REUSABLE FOR ALL EMBEDDED APPLICATIONS

INSTRUCTOR NOTES

DISCUSS THE FACT THAT THE SCRP HAS PRODUCED PAPERS AND SEVERAL DESIGN SPECIFICATIONS WHICH SHOW MANY WORKED OUT EXAMPLES OF THESE PRINCIPLES. THE VARIOUS TRADE-OFFS THAT WERE CONSIDERED AS WELL AS THE FINAL CONCLUSION ARE WELL DOCUMENTED.

THE ABSTRACT INTERFACE ONLY SHOWS YOU WHAT YOU NEED TO KNOW TO USE A MODULE - JUST AS A ROADMAP ONLY SHOWS YOU WHAT YOU NEED TO KNOW TO DRIVE A CAR - NOT WHERE TREES AND STORES ARE - MAP ALSO HIDES IRRELEVANT DETAILS SO THEY DON'T HAVE TO BE CHANGED WHENEVER SOMETHING THAT DOESN'T EFFECT THE DRIVER IS CHANGED.

SCRIP VIEW OF ABSTRACTION, MODULARITY AND HIDING

- PARTITION SYSTEM ON THE BASIS OF EXPECTED CHANGES
 - HIDE INDEPENDENT SECRETS IN SEPARATE MODULES
 - ABSTRACT INTERFACE ONLY SHOWS UNCHANGING ASPECTS OF THE INTERFACE
 - ABSTRACT INTERFACE IS TO MODULE, AS ROAD MAP IS TO WORLD
- REQUIRES FORETHOUGHT ABOUT EXPECTED CHANGE
 - BASED ON EXPERIENCE WITH SIMILAR SYSTEMS

INSTRUCTOR NOTES

THESE TOP TWO (OF THREE) LEVELS OF THE DESIGN DECOMPOSITION ARE "LOGICAL" OR ORGANIZATIONAL IN NATURE. THE DESIGN DOCUMENTS ARE ORGANIZED THIS WAY - THEY AID THE IMPLEMENTOR OR MAINTAINER IN FINDING WHERE A PARTICULAR FUNCTION HAS BEEN IMPLEMENTED. YOU WON'T NECESSARILY FIND THAT THE CODE GROUPING REFLECTS THE TOP TWO LEVELS - IT REFLECTS ONLY THE LOWEST (THIRD) LEVEL GROUPING.

THE FIRST LEVEL FOR CERTAIN AND PROBABLY THE SECOND AS WELL REPRESENTS A "GENERIC" DESIGN PARTITIONING (I.E., ONE THAT WOULD APPLY TO ALMOST ANY EMBEDDED REAL TIME SYSTEM). DAVE PARNAS HAS BET HIS "RIGHT ARM" ON THE APPLICABILITY OF THE FIRST LEVEL AND THE "BIG TOE OF HIS LEFT FOOT" ON THE SECOND LEVEL. THIS GENERIC ORGANIZATION IS ESSENTIAL TO THE REUSEABILITY OF SOFTWARE IN GENERAL.

VG 744.1

10-51

SCRIP DESIGN DECOMPOSITION
(GENERIC MODULE HIERARCHY GUIDELINES)

- HIERARCHY OF MODULES ALLOW RELEVANT MODULES TO BE IDENTIFIED BY THE READER
- GENERIC HIERARCHY PROVIDES A STARTING POINT FOR "ALL" SOFTWARE IMPLEMENTATIONS

FIRST LEVEL HIERARCHY

SECOND LEVEL HIERARCHY

HARDWARE HIDING MODULE

EXTENDED COMPUTER MODULE

DEVICE INTERFACE MODULE

BEHAVIOR HIDING MODULE

FUNCTION DRIVER MODULE

SHARED SERVICES MODULE

SOFTWARE DECISION MODULE

APPLICATION DATA MODULE
PHYSICAL MODEL MODULE
DATA BANKER MODULE
SYSTEM GENERATION MODULE
SOFTWARE UTILITY MODULE

INSTRUCTOR NOTES

THIS SLIDE SHOWS A PART OF AN INDEX FROM ONE OF THE ABSTRACT MODULE INTERFACE SPECIFICATION (DESIGN DOCUMENTS) OF THE A-7E PROJECT. ON THE LEFT WE SEE THE INDEX OF THE SECTIONS OF THIS DESIGN DOCUMENT. THE RIGHT COLUMN IS THE SECTION NUMBER OF THE CORRESPONDING REQUIREMENT FROM THE SOFTWARE REQUIREMENTS SPECIFICATION. THE NEXT SLIDE WILL SHOW THE ACTUAL DESIGN EXAMPLE FOR FD.8-2 - THE HIGHLIGHTED ENTRY.

AS AN ADDITIONAL POINT TO MENTION - MODES AND TEXT MACROS DEFINED IN THE REQUIREMENTS SPECIFICATION ARE MAPPED ONE FOR ONE TO THE ACTUAL IMPLEMENTATION - I.E., THERE IS A PROGRAM THAT RETURNS TEXT MACRO VALUES OR THE CURRENT MODE. INTERRUPTS CAN BE SCHEDULED WHEN MODES CHANGE OR TEXT MACRO VALUES CHANGE.

DESIGN TRACEABILITY

FD.7.1.5	Set HUD in-range cue mode	4.3.11.0 - 3
FD.7.1.6	Control HUD Lower Solution Cue (LSC)	
FD.7.1.6.1	Set HUD LSC mode	4.3.11.0, 3
FD.7.1.6.2	Set HUD LSC position	4.3.11.1
FD.7.1.7	Control HUD Pullup Anticipation Cue (PUAC)	
FD.7.1.7.1	Set HUD PUAC mode	4.3.8
FD.7.1.7.2	Set HUD PUAC position	4.3.8
FD.7.1.8	Set HUD pullup cue mode	4.3.9
FD.7.1.9	Control HUD Upper Solution Cue (USC)	
FD.7.1.9.1	Set HUD USC mode	4.3.11.0, 3
FD.7.1.9.2	Set HUD USC position	4.3.11.2
FD.7.1.10	Set HUD symbol blink rate	None.
FD.7.2	HUD Value Indicators	
FD.7.2.1	Set HUD altitude display	4.3.3
FD.7.2.2	Set HUD heading display	4.3.6
FD.7.2.3	Set HUD pitch & roll displays	4.3.7, 10
FD.7.2.4	Control HUD vertical velocity/acceleration displays.	
FD.7.2.4.1	Enable vert. vel. and accel. displays	4.3.12.0
FD.7.2.4.2	Set vert. accel. display	4.3.12.1, 2
FD.7.2.4.3	Set vert. vel. display	4.3.12.1

FD.8 IMS Functions

FD.8.1	Switch IMS computer control on/off	4.1.2
FD.8.2	Set IMS velocity measurement scale	4.1.5
FD.8.3	Adjust alignment of IMS platform x, y, and z axes	
FD.8.3.1	Perform small X and Y axis adjustments	4.1.4, 4.1.6
FD.8.3.2	Perform large X and Y axis adjustments	4.1.4, 4.1.6
FD.8.3.3	Adjust z axis	4.1.4, 4.1.6
FD.8.4	Initialize IMS velocities	4.6.48
FD.8.5	Set IMS reconfiguration values	None.

INSTRUCTOR NOTES

THIS IS A SAMPLE OF A DESIGN DESCRIPTION FOR A SOFTWARE FUNCTION. THE FORMAT IS VERY SIMILAR TO THAT USED IN THE REQUIREMENTS SPECIFICATION AND IS READ THE SAME WAY - IF WE ARE IN THE MODE IDENTIFIED IN THE COLUMN ON THE LEFT AND ANY EVENT IDENTIFIED IN A COLUMN OCCURS THEN THE OUTPUT VALUE IS SET TO THE VALUE AT THE BOTTOM OF THAT COLUMN.

UP NEAR THE TOP - UNDER OUTPUT PRODUCED, THIS TEMPLATE CONTAINS THE ACTUAL NAME OF AN ACCESS PROGRAM AND THE TYPE OF PARAMETER ACCEPTED BY THAT PROGRAM WHEN SETTLING THIS VALUE. THE HIGH DEGREE OF TRACEABILITY BETWEEN REQUIREMENTS AND DESIGN CAN BE SEEN BY COMPARING THIS TEMPLATE TO THE FUNCTION TEMPLATE CONTAINED IN THE ANALYSIS ON SCRP.

DESIGN EXAMPLE

FD.8.2 DEMAND FUNCTION DESCRIPTION: Set the IMS velocity measurement scale.

Mnemonic: +FD IMS_SCALE_D+

Output produced:	Type	Access program
Item	imscale	+DI S IMS_SCALE+
IMS scale		

Function definition:

Event Table FD.8.2-a -- Changing the IMS Velocity Measurement Scale

MODES		EVENTS	
Landaln			
Lautocal	@T(In mode)		X
Ol Update			
HUDaln	@T(In mode) WHEN (!+IMS mode+! = \$Cndal\$)	@T(In mode) WHEN(!+IMS mode+! = (\$Norm\$ OR \$Iner\$))	
Airaln			
Sautocal			
SINSaln			
DI			@T(In mode)
DIG	X		
I			
OLB			
PolarDI			
PolarI			
UDI			
Output value:		\$Fine\$	\$Coarse\$

INSTRUCTOR NOTES

EVERY MODULE INTERFACE IS DOCUMENTED USING A STANDARD TEMPLATE. THESE TEMPLATES CAN RUN TO MANY PAGES IN LENGTH. IN ADDITION TO DESCRIBING THE FUNCTION OR PROCEDURE NAMES, EVENTS (OR INTERRUPTS) TO BE GENERATED AND ALL PARAMETER TYPES (AS WOULD BE COVERED IN AN Ada PACKAGE SPECIFICATION) MANY OTHER TOPICS ARE COVERED. THESE INCLUDE THE EFFECTS OF A FUNCTION CALL, ANY TRADEOFFS THAT WERE CONSIDERED DURING THE DESIGN, SPECIFICALLY WHAT HAS BEEN HIDDEN, AND ANY UNDERLYING ASSUMPTIONS NOT EXPECTED TO CHANGE - THESE ARE ONLY THE MOST IMPORTANT ASPECTS - THE ACTUAL TEMPLATE HAS TWELVE (12) SECTIONS.

INTERFACE TEMPLATE

- STANDARD TEMPLATE USED FOR ALL INTERFACES
- IDENTIFIES
 - ACCESS FUNCTIONS
 - EVENTS
- STATES BASIC ASSUMPTIONS - UNLIKELY TO CHANGE
- DOCUMENTS IMPLEMENTATION TRADEOFFS
- DEFINES FUNCTION EFFECTS
- IDENTIFIES HIDDEN INFORMATION

INSTRUCTOR NOTES

THE SCRP WAS INSTIGATED TO DEMONSTRATE THAT THE USE OF ABSTRACT MODULE INTERFACES, WHERE THE MODULARIZATION CRITERIA IS BASED ON INFORMATION HIDING, WOULD REDUCE LIFE CYCLE MAINTENANCE COSTS - DATA HAS ALREADY BEEN COLLECTED TO VERIFY THIS. THUS THE IMPORTANCE OF THESE PRINCIPLES AND GOALS.

AN ANALYSIS OF THE SCRP DOCUMENTATION WILL SHOW HOW THIS MODULARIZATION APPROACH AND THE USE OF SEPARATION OF CONCERNS PROVIDES FOR AN EXTREMELY HIGH DEGREE OF TRACEABILITY FROM REQUIREMENTS TO DESIGN AND A DESIGN THAT IS EASILY UNDERSTOOD AND ONE IN WHICH ANY DESIGNER CAN QUICKLY LOCATE HIS/HER AREA OF CONCERN.

THIS FORMALISM AND UNIFORMITY MAKE THE DESIGN REVIEWABLE FOR CORRECTNESS.

AGAIN THE HIGH LEVEL OF MODULARITY AND THE SEPARATION OF CONCERN ALLOWS FOR MANY TASKS WHICH CAN BE PURSUED IN PARALLEL, SOME BY RELATIVELY JUNIOR ENGINEERS AND THEN QUICKLY REVIEWED BY MORE EXPERIENCED ENGINEERS.

PRINCIPLES

- ABSTRACTION, MODULARITY, HIDING
 - SYSTEM PARTITIONED INTO HUNDREDS OF MODULES USING INFORMATION HIDING AND ABSTRACT INTERFACES
- SEPARATION OF CONCERNS
 - USED AT SEVERAL LEVELS
- UNIFORMITY, FORMALISM
 - LIMITED SET OF PRIMITIVE CONSTRUCTS, HEAVY USE OF TABLES AND TEMPLATES
- STRUCTURING
 - ATTAINABLE THROUGH MODULARIZATION STRATEGY

GOALS

- MAINTAINABILITY, MODIFIABILITY
 - MODULARIZATION PRINCIPLES MINIMIZE IMPACT OF CHANGE
- TRACEABILITY, UNDERSTANDABILITY
 - SEPARATION OF CONCERNS, MODULAR STRUCTURE FACILITATE TRACING FROM REQUIREMENTS TO DESIGN
- CORRECTNESS, VERIFIABILITY, TESTABILITY
 - FORMALISM, STANDARD TABLES AND TEMPLATES FACILITATE REVIEWS
- PRODUCTIVITY
 - SEPARATION OF CONCERNS, ABSTRACT INTERFACES, AND MODULARITY ALLOW FOR PARALLEL DEVELOPMENT BY ENGINEERS AT VARIED LEVELS OF EXPERIENCE.

INSTRUCTOR NOTES

OBJECT ORIENTED DESIGN IS PRIMARILY CONCERNED WITH DETAILED DESIGN OF LARGE SYSTEMS OR THE OVERALL DESIGN OF SMALL SYSTEMS. WE WILL INTRODUCE THE TERMINOLOGY, PRESENT THE BASIC METHOD, SHOW AN EXAMPLE OF THE TYPE OF INFORMATION PRODUCED AND SUMMARIZE.

OBJECT ORIENTED DESIGN

- OVERVIEW

- DEFINITIONS

- METHOD

- EXAMPLE

- SUMMARY

INSTRUCTOR NOTES

THIS TECHNIQUE WAS FORMALIZED BY GRADY BOOCH IN HIS BOOK "SOFTWARE ENGINEERING IN Ada."
THERE ARE SEVERAL PAPERS BY HIM AND OTHERS ON THIS TOPIC.

ABSTRACT DATA TYPES - SET OF VALUES, DATA STRUCTURES AND THE ASSOCIATED OPERATION
RELATED TO THAT DATA TYPE.

OBJECT-ORIENTED DESIGN

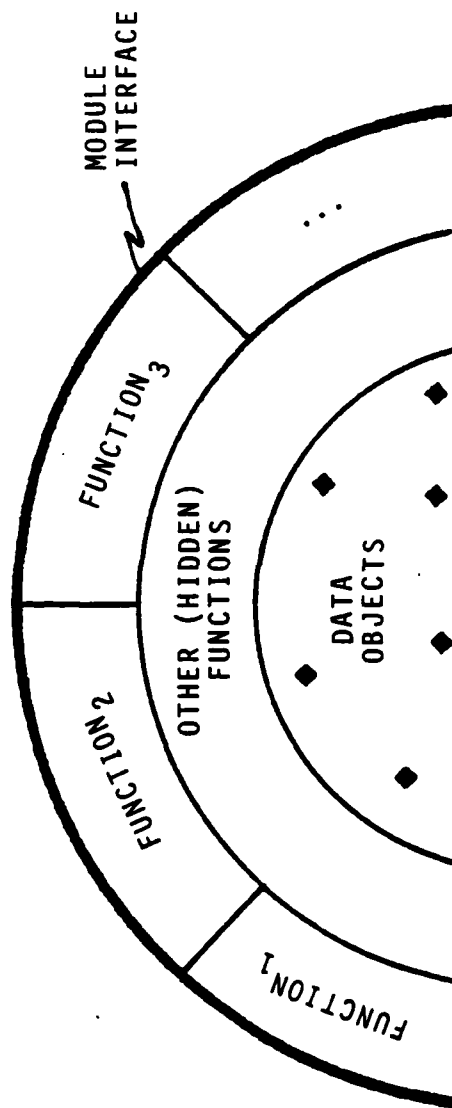
- EVOLVED FROM WORK DONE BY G. BOOCH AND INTEL CORPORATION ON METHODS TO EXPRESS Ada SOFTWARE MODULE DESIGNS
- BASED ON THE SAME CONCEPTS AS SCRIP DESIGN METHODS
 - ABSTRACTION
 - MODULARITY
 - HIDING
- DIFFERS FROM SCRIP DESIGN METHOD IN ...
 - EMPHASIS ON ABSTRACTION OF DATA (ABSTRACT DATA TYPES)
 - CRITERIA FOR MODULARIZATION BASED ON DATA TYPE AND OPERATIONS
 - BOTTOM-UP VS. TOP-DOWN VIEW OF DESIGN

INSTRUCTOR NOTES

"MODULES" ARE COLLECTIONS (PACKAGES) OF Ada PROCEDURES OR FUNCTIONS THAT (BY SOMETIMES CALLING OTHER (HIDDEN) FUNCTIONS) OPERATE ON A SET OF CLOSELY RELATED DATA OBJECTS. FUNCTIONS USUALLY EXIST FOR CREATING, DESTROYING, EXAMINING, AND MANIPULATING DATA OBJECTS.

OBJECT-ORIENTED DESIGN

- "Ada" OBJECT-ORIENTED DESIGN BUILDS A SYSTEM OF MODULES, EACH OPERATING ON A CLASS OF DATA OBJECTS:



- A CLASS OF DATA OBJECTS WOULD BE CHARACTERIZED AS AN ABSTRACT DATA TYPE
 - A SET OF VALUES
 - A DATA STRUCTURE
 - THE ALLOWED OPERATIONS ON THE DATA STRUCTURE AND SET OF VALUES

INSTRUCTOR NOTES

ONE NEEDS TO IDENTIFY A VERBAL DESCRIPTION OF THE PROBLEM.

DATA OBJECTS - TYPICALLY NOUNS IN THE PROBLEM STATEMENT.

OPERATIONS ON OBJECTS - TYPICALLY VERBS IN THE PROBLEM STATEMENT.

ESTABLISH INTERFACES - DEFINE IN A FORMAL SENSE THE INPUTS/OUTPUTS OF EACH
OPERATION.

OBJECT-ORIENTED DESIGN

- METHOD

- DEVELOP AN INFORMAL STRATEGY
- IDENTIFY DATA OBJECTS
- IDENTIFY OPERATIONS ON THOSE OBJECTS
- ESTABLISH INTERFACES
 - PACKAGE INTO "MODULES"
 - EXPRESS IN TERMS OF A PROGRAM DESIGN LANGUAGE (Ada)

INSTRUCTOR NOTES

RELATE THE GRAPHIC AND PDL DESCRIPTION.

DESCRIBE WHAT A FUNCTION AND PROCEDURE ARE.

INQUIRY_OPERATIONS IMPLEMENTS AN ABSTRACT DATA TYPE FOR COMMAND

COMMAND SET OF VALUES

- COLLECT_STATISTICS
- LIST_DATA
- QUIT
- SELECT_UNIT

COMMAND OPERATIONS

- IMPLEMENTED WITH PROC. AND FTNS

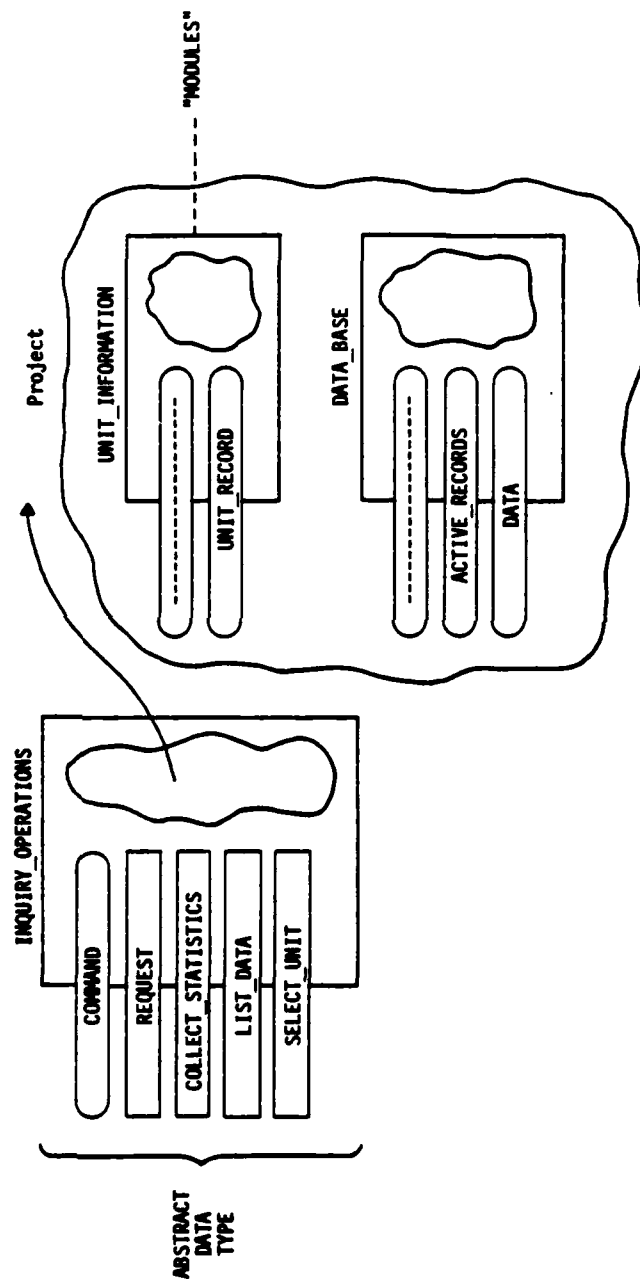
OBJECT-ORIENTED DESIGN EXAMPLE

• ESTABLISH INTERFACES

```

with Project;
use Project;
package Inquiry_Operations is
  type Command is (Collect_Statistics, List_Data, Quit, Select_Unit);
  function Request return Command;
  procedure Collect_Statistics;
  procedure List_Data;
  procedure Select_Unit;
end Inquiry_Operations;

```



INSTRUCTOR NOTES

THIS METHOD HAS BEEN EXTENDED BY OTHERS (RAY BUHR FOR ONE) AND MAY BECOME AN IMPORTANT METHODOLOGY IN THE FUTURE DUE TO IT'S STRONG LINK TO Ada.

VG 744.1

10-151

LE

OBJECT-ORIENTED DESIGN SUMMARY

- RELIES ON DATA ABSTRACTIONS
- PRODUCES MODULES
- HIDES DESIGN DECISIONS
 - DATA STRUCTURE
 - ALGORITHM SELECTION
- GOAL —→ GOOD MODULE SPECIFICATIONS
- BOTTOM UP APPROACH WORKS WELL ON SMALL PROBLEMS OR ON SEGMENTS OF A LARGER PROBLEM DECOMPOSED BY OTHER METHODS

INSTRUCTOR NOTES

DIFFERENCES IN CRITERIA FOR MODULARITY (COMPARED TO SCRP) SUPPORT DIFFERENT GOALS. SCRP ATTEMPTS TO IDENTIFY LIKELY CHANGES TO THIS SYSTEM AND BUILDS MODULES TO HIDE EFFECTS. OBJECT ORIENTED USES A MORE GENERALIZED APPROACH AND BUILD MODULES AROUND DATA STRUCTURES. THIS IS MORE LIKELY TO BUILD MODULES USABLE ON A DIFFERENT SYSTEM, BUT MAY NOT ACCOMMODATE CHANGES TO THIS SPECIFIC SYSTEM AS EASILY.

REUSABILITY AND TRANSPORTABILITY DON'T CONFLICT WITH MAINTAINABILITY AND MODIFIABILITY - THEY REFLECT A CHANGE IN EMPHASIS.

OBJECT ORIENTED

PRINCIPLES

- ABSTRACTION, MODULARITY, HIDING
 - SYSTEM PARTITIONED AROUND DATA
- OBJECTS - USES Ada PACKAGE SPECIFICATIONS FOR ABSTRACT INTERFACE
- UNIFORMITY, FORMALISM
 - INHERENT IN THE USE OF Ada SYNTAX AND SEMANTICS
- STRUCTURING
 - USES Ada'S STRUCTURING CONSTRUCTS

GOALS

- REUSABILITY, TRANSPORTABILITY
 - ATTEMPTS TO BUILD GENERIC REUSABLE COMPONENTS
- CORRECTNESS, VERIFIABILITY, TESTABILITY
 - USE OF Ada SYNTAX ALLOWS FOR COMPILER AND TOOL PROCESSING, AND CONSISTENCY CHECKING

INSTRUCTOR NOTES

IN THIS SECTION ON SD WE WILL INTRODUCE DATA FLOW GRAPHS AND STRUCTURE CHARTS AND TO INTRODUCE THE TWO TECHNIQUES FOR TRANSITIONING BETWEEN THEM.

VG 744.1

10-171

STRUCTURED DESIGN

- OVERVIEW
- DATA FLOW GRAPHS (DFG)
- DFG EXAMPLE
- DFG SUMMARY
- STRUCTURE CHART
- STRUCTURE CHART EXAMPLE
- TRANSLATION PROCESS
- TRANSFORM-CENTERED DESIGN
- TRANSACTION-CENTERED DESIGN
- SUMMARY
- CRITIQUE

INSTRUCTOR NOTES

THIS METHODOLOGY IS ATTRIBUTED TO YOURDON AND CONSTANTINE. IT IS CONCERNED WITH MODULARIZATION, INTERCONNECTIVITY, AND THE FLOW OF DATA. WE ARE NOT INTERESTED IN THE DETAILS OF THE MODULE INTERNALS YET.

TRANSFORMS AND TRANSACTIONS WILL BE EXPLAINED SOON.

THE CONNECTIONS BETWEEN MODULES SHOULD BE SIMPLE AND WELL DEFINED.

OVERVIEW

- STRUCTURED DESIGN IS A METHOD FOR DEVELOPING A HIGH-LEVEL BLUEPRINT OF A SOFTWARE SYSTEM
- KEY CONCEPTS:
 - A SYSTEM CAN BE DESIGNED AFTER UNDERSTANDING ITS DATA FLOW
 - DESIGNS ARE BUILT AROUND EITHER THE MAJOR DATA TRANSFORMS OR TRANSACTION CENTERS OF A SYSTEM
 - DESIGN QUALITY CAN BE IMPROVED BY EVALUATING INTRAMODULE UNITY AND INTERMODULE CONNECTIONS
- USES A GRAPHICAL NOTATION SUPPLEMENTED WITH ANNOTATION TO EXPRESS
 - DATA FLOW (USES BUBBLE CHARTS)
 - STRUCTURAL DESIGN (USES STRUCTURE CHARTS)

INSTRUCTOR NOTES

BUBBLE CHARTS AND DATA FLOW DIAGRAMS ARE OFTEN USED INTERCHANGEABLY.

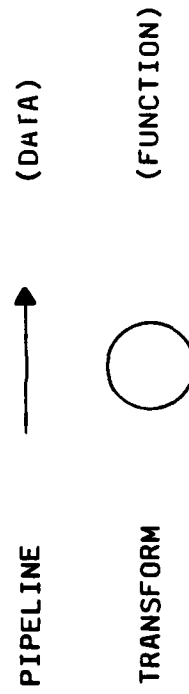
VG 744.1

10-19i

DATA FLOW GRAPHS
(BUBBLE CHARTS)

- DATA FLOW GRAPHS ARE NETWORKS OF TRANSFORMATIONS CONNECTED BY DATA PIPELINES.
- TRANSFORMATIONS ARE FUNCTIONS THAT CHANGE THEIR INPUT DATA INTO OUTPUT.
- PIPELINES ARE CONTINUOUS STREAMS OF DATA, BE THEY INPUT OR OUTPUT.

BUBBLE CHART SYNTAX



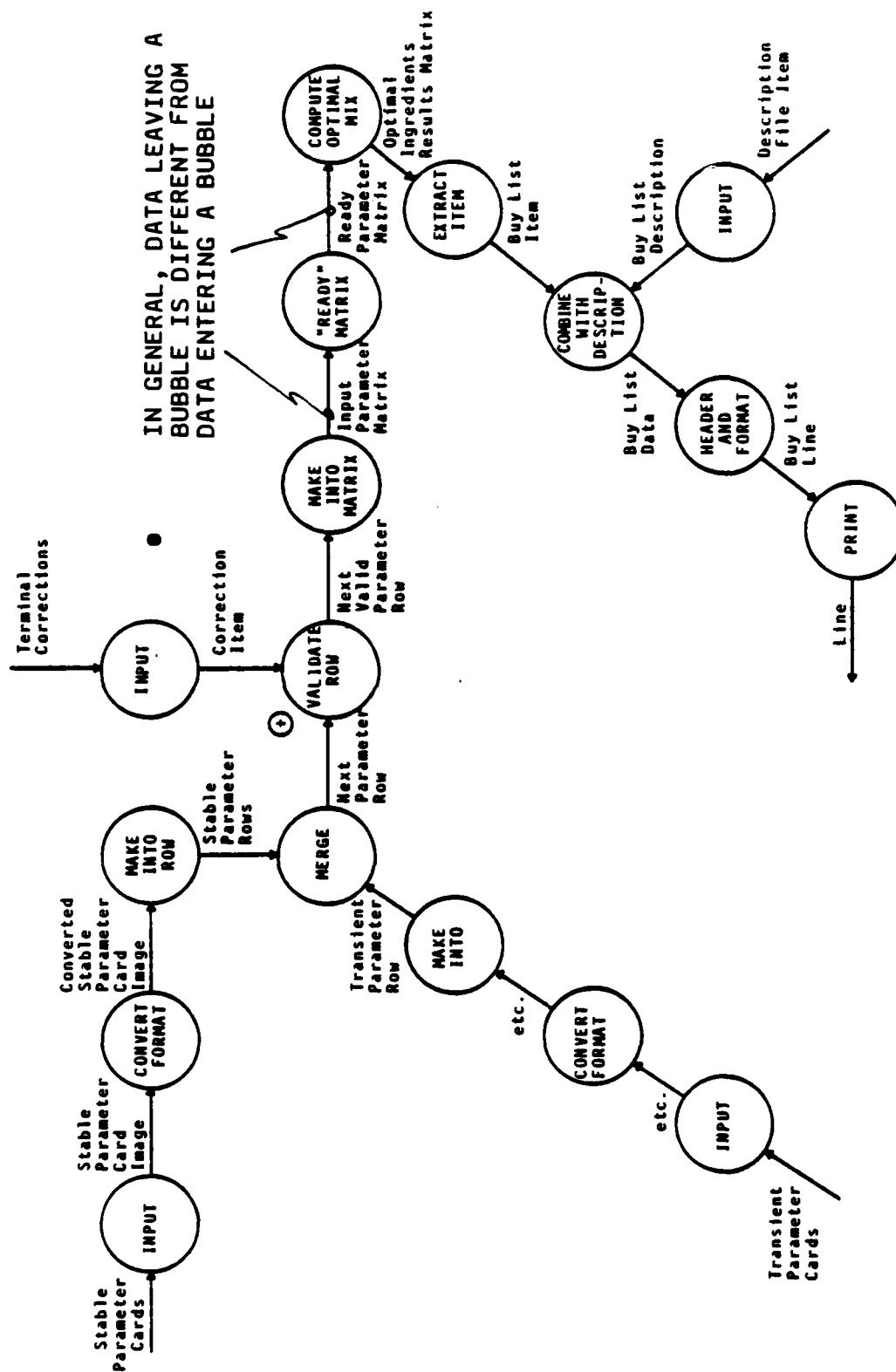
INSTRUCTOR NOTES

EXAMPLE REFERENCE: YOURDON, E. AND CONSTANTINE, L.; "STRUCTURED DESIGN,"
PRENTICE-HALL, 1979.

NOTE THE EDP NATURE OF THE PROBLEM, BUT INDICATE TO THE CLASS THAT MAJOR PORTIONS OF
MILITARY SYSTEMS HAVE SIMILAR DATA FLOWS (ONLY THE ARROWS AND BUBBLE LABELS WILL DIFFER).

DATA FLOW GRAPH

- ARROWS CONNECT BUBBLES TOGETHER INTO A NETWORK:



INSTRUCTOR NOTES

VG 744.1

10-211

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

DATA FLOW GRAPHS SUMMARY

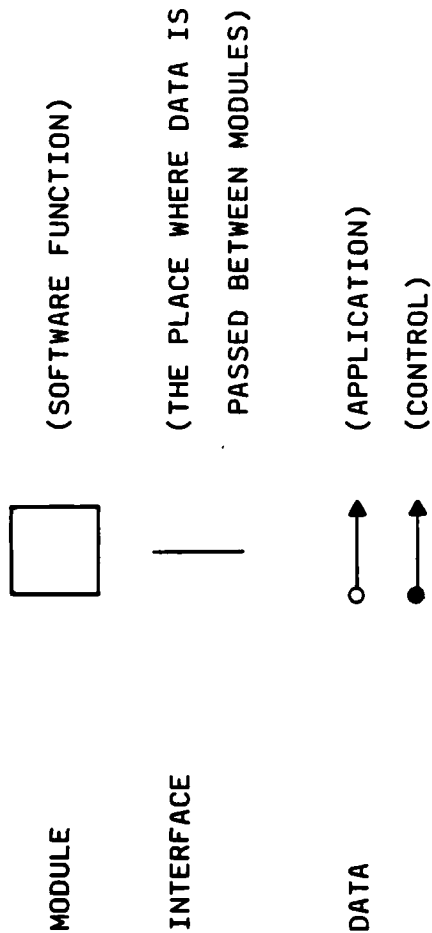
- DATA FLOW GRAPHS CAN BE DERIVED FROM SOFTWARE REQUIREMENTS BY A DECOMPOSITION PROCESS
- DATA FLOW GRAPHS ARE NETWORKS OF TRANSFORMS CONNECTED BY PIPELINES OF CONTINUOUS DATA STREAMS
- THEY DO NOT SHOW ...
 - DATA DECOMPOSITION
 - DATA HIERARCHY
 - PROCEDURAL DETAILS
 - CONTROL LOGIC
 - ACTIVATION OR TERMINATION CONDITIONS

INSTRUCTOR NOTES

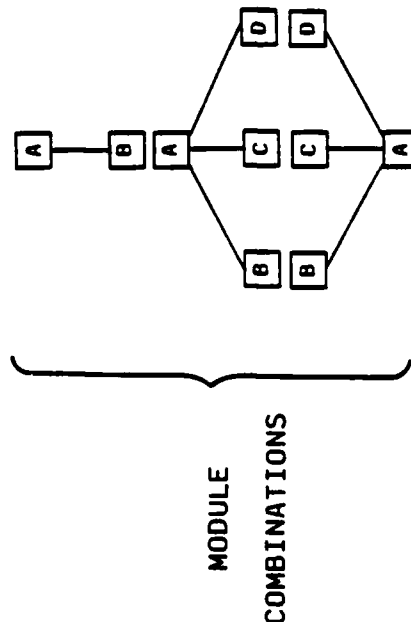
THESE SYMBOLS ARE NOT LANGUAGE SPECIFIC. POINT OUT DIFFERENCES BETWEEN THE "FILLED"
ARROW AND "EMPTY" ARROW.

STRUCTURE CHART

- THE BASIC SYNTAX, USED MOST OFTEN, COMPRISES FOUR SYMBOLS ...



- BASIC STRUCTURAL ASPECTS FROM VG 778.1 / 16-9



MODULE A CALLS MODULE B.

MODULE A CALLS MODULES B, C, AND D.

MODULE A IS CALLED BY MODULES B, C, AND D.

INSTRUCTOR NOTES

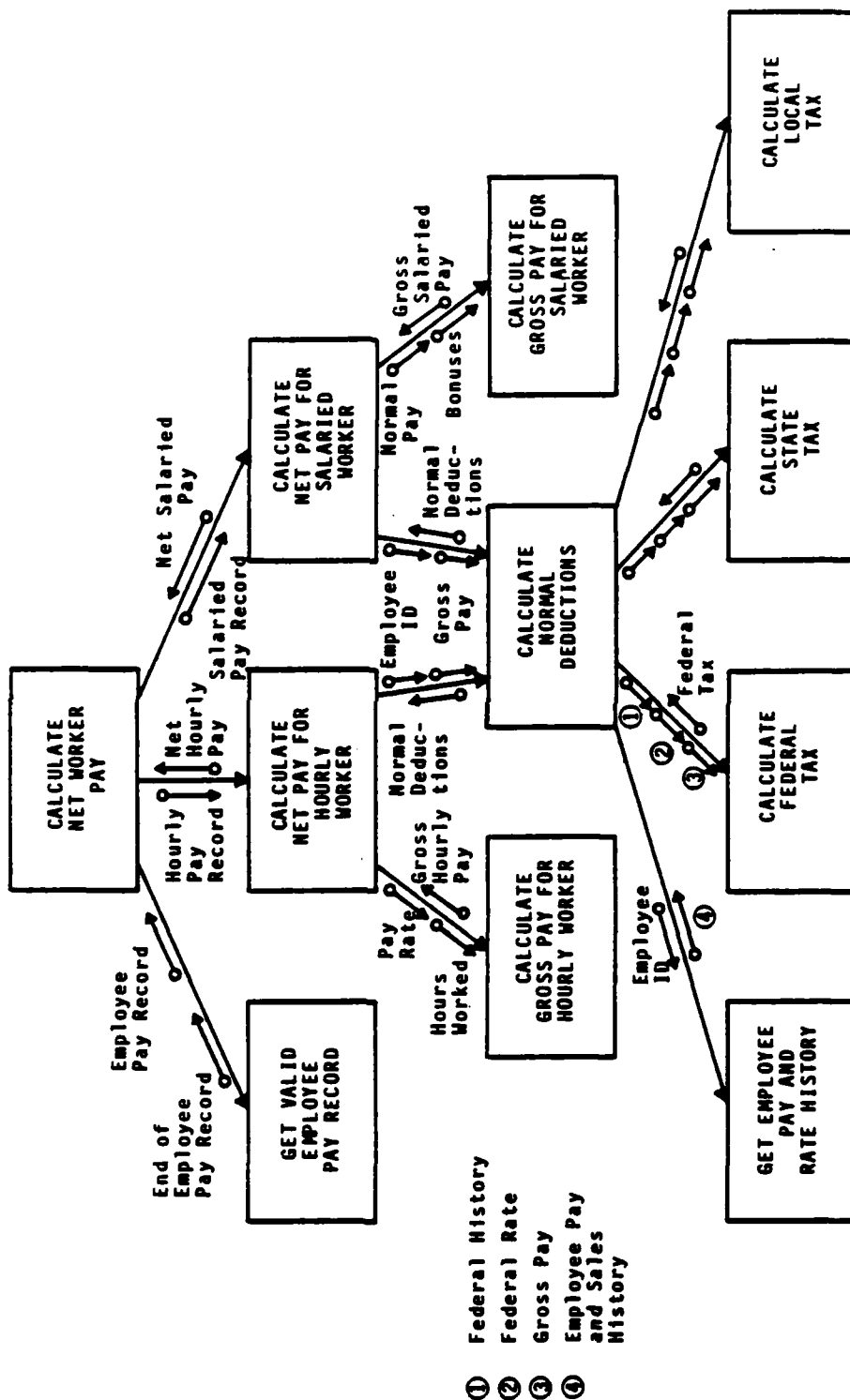
WHETHER THE CALLED MODULES ARE PHYSICALLY CONTAINED IN THE CALLER MODULE OR ANOTHER MODULE IS IMMATERIAL AT THIS POINT.

WITHOUT THE ARROWS, THIS IS CALLED A CALLING HIERARCHY DIAGRAM. AT THE MINIMUM THERE SHOULD BE ONE FOR EVERY SYSTEM.

THIS IS A GOOD TECHNIQUE FOR DESCRIBING THE DESIGN: IT WILL SUPPORT ANALYSIS OR DESIGN QUALITY.

STRUCTURE CHART

• THE TOTAL EFFECT IS A HIERARCHY OF MODULES ...



INSTRUCTOR NOTES

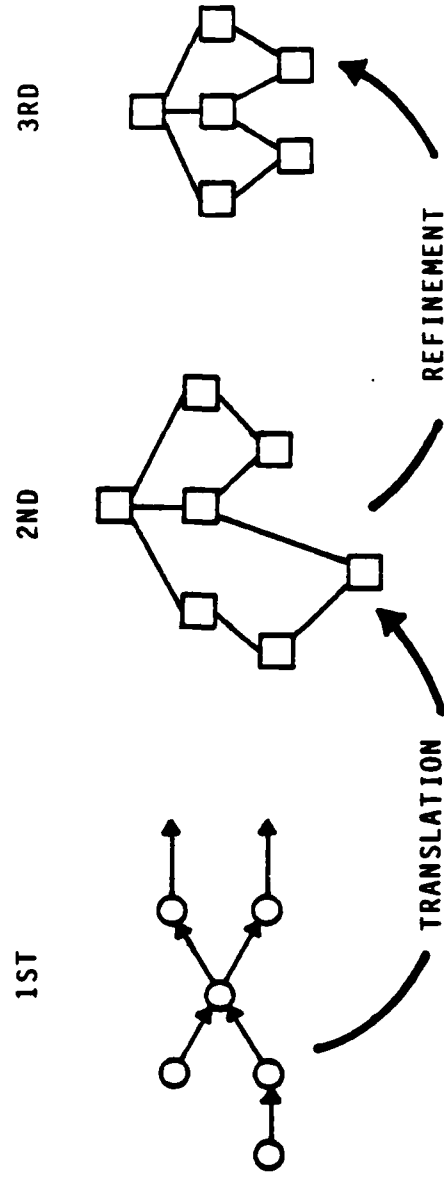
THESE 2 TECHNIQUES ARE TO HELP IN THE TRANSLATION. THE MAJOR TECHNIQUE USED IS
TRANSFORM CENTERED DESIGN.

VG 744.1

10-24i

TRANSLATION PROCESS

- TWO TECHNIQUES GUIDE THE DESIGNER DURING THE GENERAL PROCESS:
 - TRANSFORM-CENTERED
 - TRANSACTION-CENTERED
- AFTER THE TRANSLATION PROCESS, THE STRUCTURE CHART IS EVALUATED AND REFINED ACCORDING TO METRICS ...



INSTRUCTOR NOTES

EMPHASIZE THE ALMOST COOKBOOK NATURE OF THE PROCEDURES.

NOTE THAT IDENTIFYING THE MAJOR OR HIGHEST TRANSFORM IS A SUBJECTIVE JUDGEMENT - SOMETHING TO BE GAINED FROM EXPERIENCE. UNDERLYING THIS APPROACH IS AN ASSUMPTION THAT THE MAJOR TRANSFORM WON'T CHANGE SO IT'S SAFE TO DESIGN AROUND IT.

TRANSFORM-CENTERED DESIGN

TRANSFORM-CENTERED DESIGN PROCEDURE:

1. DRAW THE DATA FLOW GRAPH.
2. LABEL WHERE THE HIGHEST TRANSFORMS ARE.
3. MAKE THE HIGHEST TRANSFORMS THE TOP OF THE DESIGN.
(I.E. TOP OF THE STRUCTURE CHART)
4. EVERYTHING TO THE LEFT OF THE TRANSFORMS IS
CONSIDERED INPUT (AFFERENT).
5. EVERYTHING TO THE RIGHT OF THE TRANSFORMS IS
CONSIDERED OUTPUT (EFFERENT).

AD-A165 122

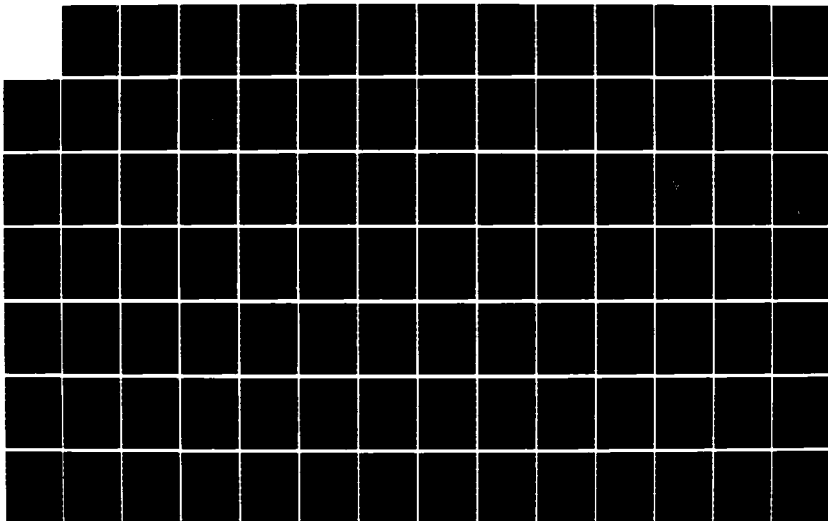
ADA (TRADEMARK) TRAINING CURRICULUM: INTRODUCTION TO
SOFTWARE ENGINEERING M102 TEACHER'S GUIDE(U) SOFTECH
INC WALTHAM MA 1986 DAAB07-83-C-K506

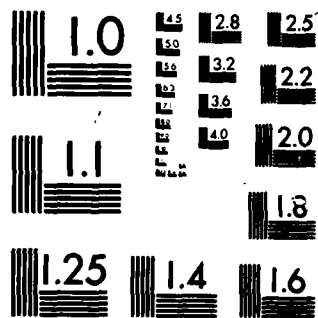
4/6

UNCLASSIFIED

F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

TRANSACTION CENTERS ARE EASIER TO FIND - IT'S JUST A MATTER OF COUNTING ARROWS - AS REQUIREMENTS CHANGE THE TRANSACTION CENTER MAY WANT TO CHANGE CREATING POTENTIAL DESIGN CHANGES.

VG 744.1

10-261

TRANSACTION-CENTERED DESIGN

- TRANSACTION-CENTERED DESIGN PROCEDURE:

1. DRAW THE DATA FLOW GRAPH.
2. FIND THE BUBBLE WITH THE HIGHEST VOLUME OF TRANSACTION DISPATCHING THAT IS ALSO A HIGH-LEVEL TRANSFORM.
3. MAKE THIS BUBBLE THE TOP OF THE DESIGN.
(I.E. TOP OF STRUCTURE CHART)

INSTRUCTOR NOTES

ASK WHY REAL TIME APPLICATIONS DON'T WORK WELL.

ANSWER: REAL TIME APPLICATIONS AREN'T DATA FLOW INTENSIVE - THEY ARE CONTROL FLOW
 INTENSIVE SO STRUCTURING AROUND DATA FLOW CAN LEAD TO PROBLEMS.

STRUCTURED DESIGN SUMMARY

- PROVIDES METHODS FOR REPRESENTING SOFTWARE ARCHITECTURAL DESIGN
- PROVIDES GUIDANCE AS TO MANNER IN WHICH MODULARITY IS DETERMINED
- WORKS VERY WELL ON MEDIUM SIZED APPLICATIONS WHICH RESULT IN SEQUENCE PROGRAMS
- WORKS MARGINALLY IN REAL TIME APPLICATIONS

INSTRUCTOR NOTES

AS THE NAME OF THIS TECHNIQUE IMPLIES STRUCTURING ACCORDING TO WELL DEFINED RULES IS ONE OF THE IMPORTANT UNDERLYING PRINCIPLES - THE MODULAR STRUCTURE USES ABSTRACTION AND GRAPHICAL REPRESENTATIONS TO AID IN THE UNDERSTANDABILITY OF THE SYSTEM.

THE DATA FLOW GRAPHS AND STRUCTURE CHARTS HAVE WELL DEVELOPED SYNTAX AND SEMANTICS RULES REFLECTING A REASONABLE (BUT NOT EXTREME) DEGREE OF UNIFORMITY AND FORMALISM.

THE COUPLING AND COHESION METRICS ARE SPECIFICALLY AIMED AT INSURING THAT THE DESIGN AVOIDS PROBLEMS THAT WOULD IMPACT ITS MODIFIABILITY.

STRUCTURED DESIGN

PRINCIPLES

- STRUCTURING, MODULARITY
 - PROCESS DEVELOPS MODULAR, HIERARCHICAL STRUCTURE
- ABSTRACTION
 - DATA FLOW GRAPHS DECOMPOSITION AND STRUCTURE CHARTS HIERARCHY BOTH SUPPORT ABSTRACTION
- UNIFORMITY, FORMALISM
 - BOTH DATA FLOW GRAPHS AND STRUCTURE CHARTS HAVE WELL DEVELOPED SYNTAX AND SEMANTIC RULES

GOALS

- UNDERSTANDABILITY
 - GRAPHIC NATURE OF TECHNIQUES AID UNDERSTANDING
- MAINTAINABILITY, MODIFIABILITY
 - STRATEGIES ARE INTENDED TO PRODUCE MAINTAINABLE DESIGNS
 - COMPILING AND COHESION RULES SUPPORT MODIFIABILITY
- TRACEABILITY
 - STRATEGIES PROVIDE TRACEABLE DESIGN

INSTRUCTOR NOTES

THIS SECTION PRESENTS AN INTRODUCTION TO JACKSON STRUCTURED PROGRAMMING (JSP) AND WARNIER-ORR. COMMON ELEMENTS ARE COVERED AS PART OF JSP - THE MAIN DIFFERENCES ARE IN THE SYNTAX.

JACKSON/WARNIER-ORR

- JSP OVERVIEW
- JSP METHOD
- JSP EXAMPLE
- LIMITATIONS
- STRUCTURE CLASH
- WARNIER-ORR OVERVIEW
- WARNIER-ORR SYNTAX
- SUMMARY
- CRITIQUE

INSTRUCTOR NOTES

JACKSON'S VIEW OF REALITY IS A SET OF PROCESSES TRANSFORMING DATA STRUCTURES WHICH ARE INHERENT IN THE PROBLEM. THIS DATA IS USED TO MODEL THE PROBLEM.

NOTICE THE SUPPORT LENT TO BUSINESS SYSTEMS WHICH ARE DATA BASE ORIENTED.

JACKSON STRUCTURED PROGRAMMING OVERVIEW

- WAS DEVELOPED IN THE EARLY 70'S
- IS SUITABLE FOR SMALL-MEDIUM PROGRAM DESIGNS
- IS VERY WELL TESTED
- HAS STRUCTURING PRINCIPLES APPLICABLE FOR JSD
- DEVELOPED FOR COBOL SHOPS EXTENDED TO REAL TIME APPLICATIONS
- CONCERNED WITH DESIGNING A PARTICULAR PROGRAM
- A RELIABLE AND REPEATABLE METHOD
- STARTS WITH DATA AND FINISHES WITH A PROGRAM

INSTRUCTOR NOTES

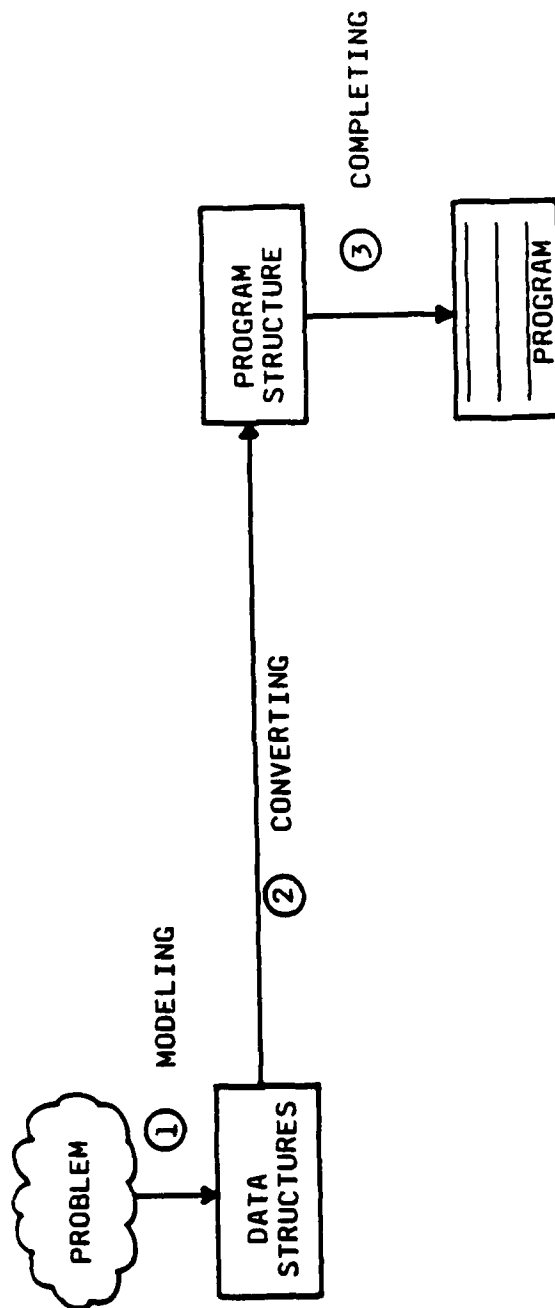
JSP IS A 3-STEP METHOD.

- ① FIND THE DATA RELATIONSHIPS THROUGH YOUR OWN UNDERSTANDING OF THE PROBLEM.
DRAW THE DATA STRUCTURES.
- ② CONVERT THESE TO A SKELETON PROGRAM STRUCTURE.
- ③ FILL IN THE SKELETON.

JACKSON DOESN'T HELP MUCH WITH FINDING THE DATA STRUCTURES. THE KEY IS THAT WE MUST
FIND A PROGRAM STRUCTURE THAT MATCHES THE INPUT AND OUTPUT DATA STRUCTURE.

JACKSON STRUCTURED PROGRAMMING METHOD

- THESE STRUCTURES ARE BUILT ACCORDING TO A COOKBOOK-LIKE METHOD ...



- MODELING: DEFINE INPUT AND OUTPUT DATA STRUCTURE USING JACKSON'S GRAPHIC NOTATION.
- CONVERTING: CREATE A GENERAL PROGRAM STRUCTURE (SAME GRAPHICS) THAT MATCHES THE STRUCTURES OF BOTH THE INPUT AND OUTPUT.
- COMPLETING: LIST ELEMENTARY OPERATIONS AND ASSIGN THESE TO MODULES.

INSTRUCTOR NOTES

MAKE SURE THE CLASS UNDERSTANDS THE DIAGRAMS OR THEY WILL QUICKLY BE LOST.

GOAL:

SUMMARIZE TRUCK ACTIVITY FOR A MILITARY MOTOR POOL.

INPUT:

TRUCK FILE, CONTAINING RECORDS (IN TRUCK NUMBER ORDER) THAT TELL ABOUT THE TRUCK'S: SOURCE, DESTINATION, AND LOAD.

OUTPUT:

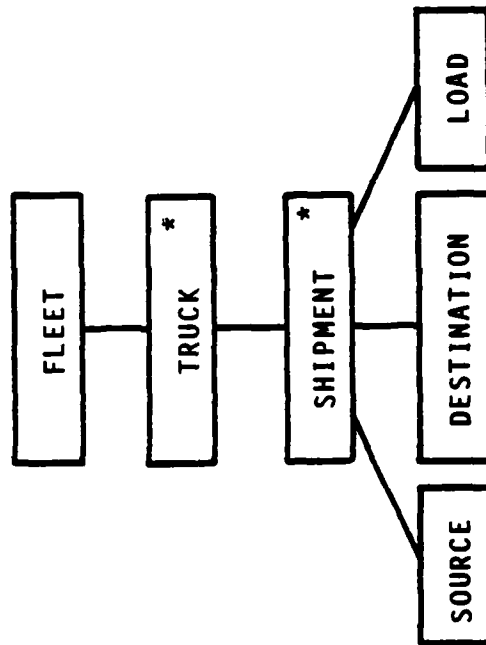
A REPORT, WITH A LONE TITLE PAGE, SUMMARIZING EACH TRUCK'S SHIPMENT ACTIVITY ON A SINGLE REPORT LINE.

EXAMPLE

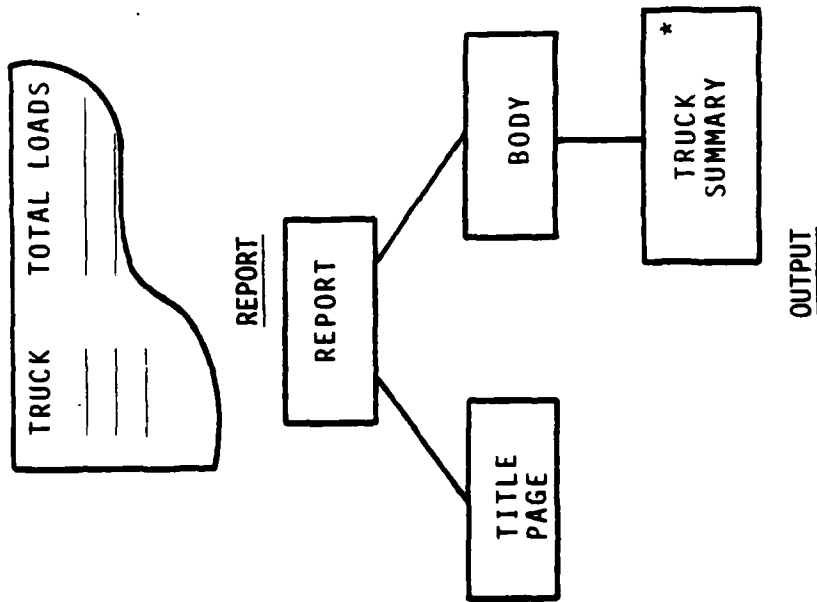
- MODELING: DEFINE INPUT AND OUTPUT DATA STRUCTURES



TRUCK FILE

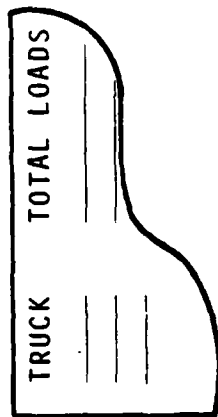


INPUT



REPORT

OUTPUT

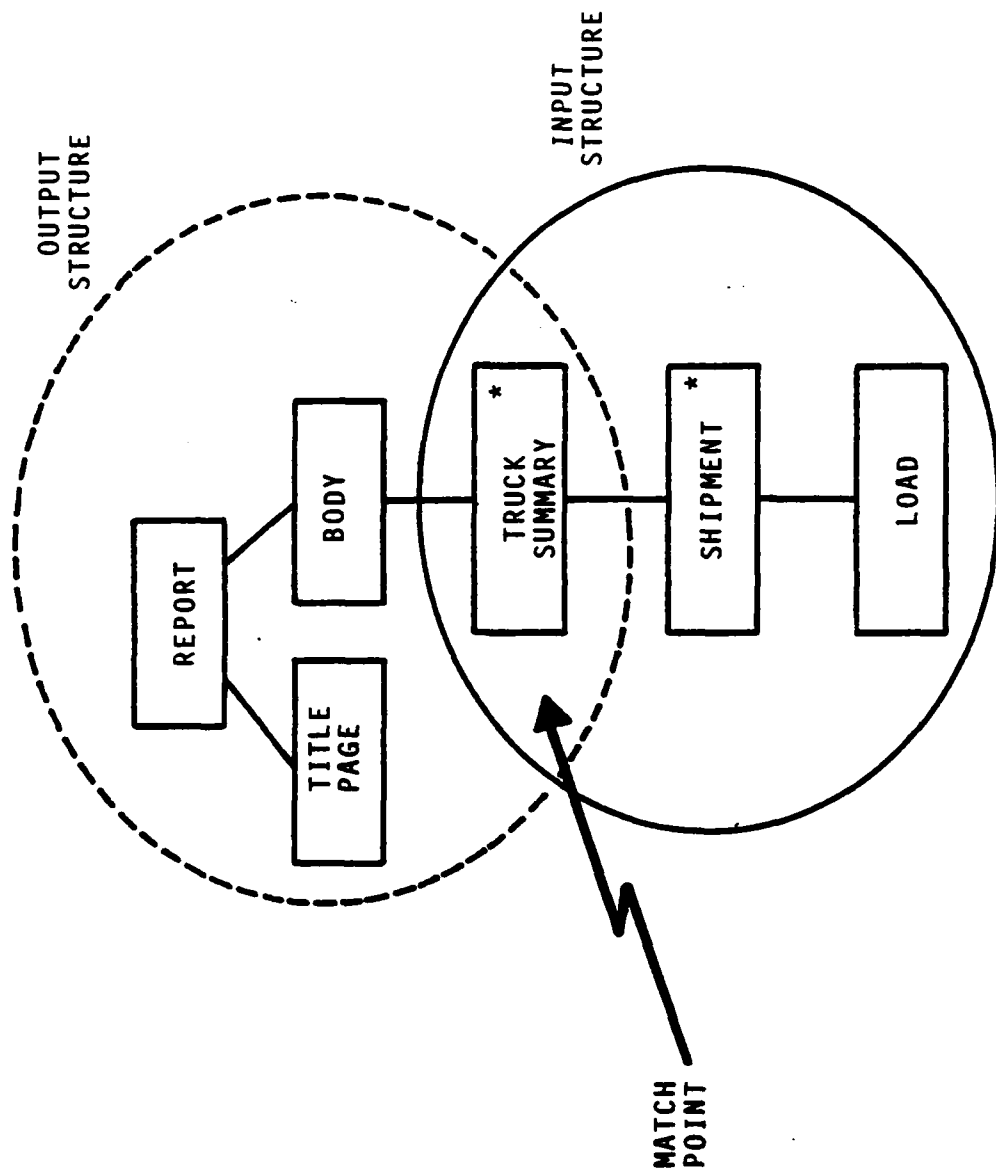


INSTRUCTOR NOTES

NOTE WE HAVE MADE SOME SIMPLIFYING ASSUMPTIONS ABOUT THE DETAIL ON THE OUTPUT REPORT.
THE MATCH POINT SHOWS THE INTERSECTION OF THE INPUT AND OUTPUT DATA STRUCTURES - IN THIS
CASE INFORMATION ON A PER TRUCK BASIS.

EXAMPLE

- CONVERTING: CREATE A GENERAL PROGRAM STRUCTURE THAT MATCHES THE DATA STRUCTURES

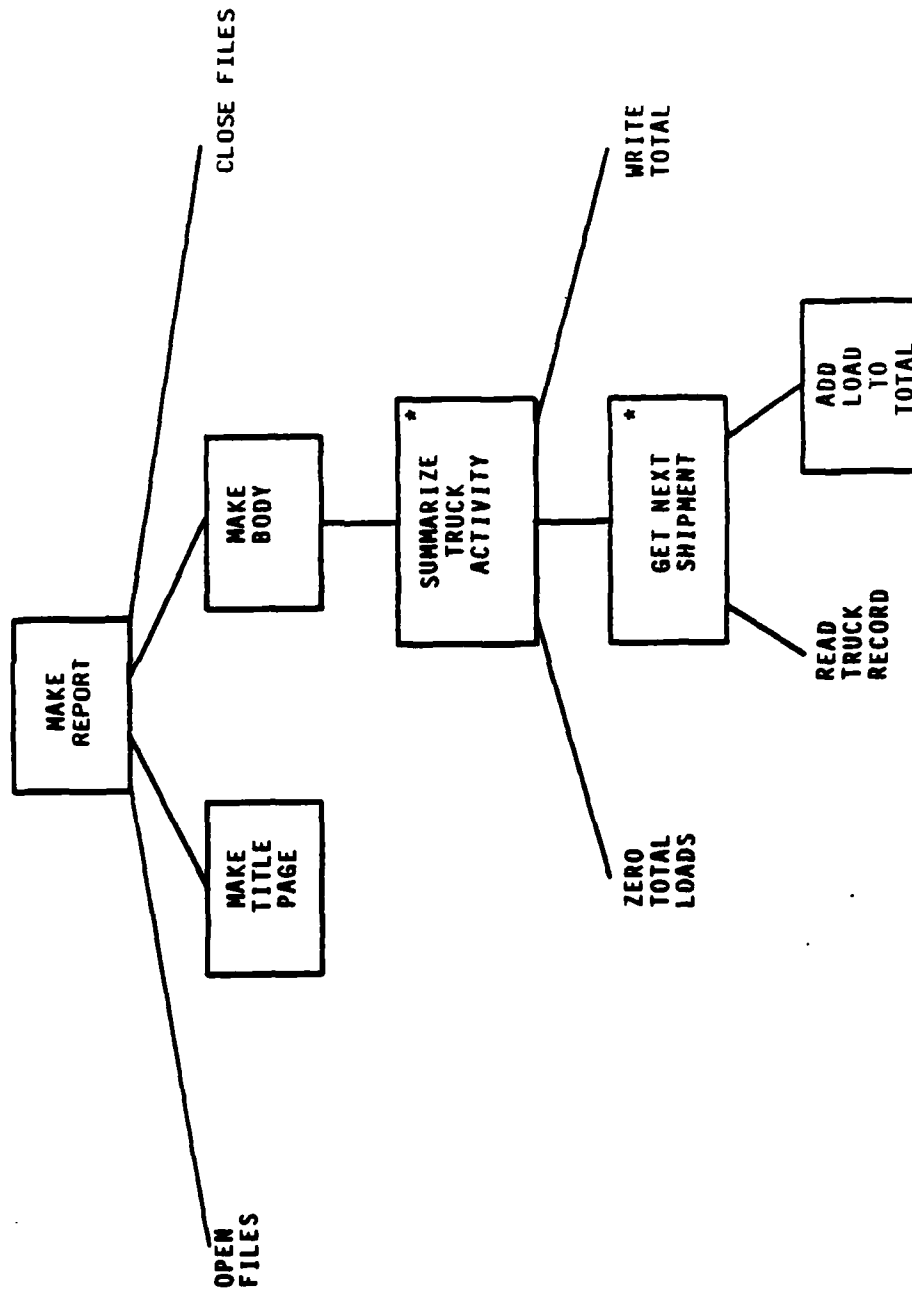


INSTRUCTOR NOTES

THIS IS THE PROCESS: WE ARE NO LONGER ADDRESSING THE DATA STRUCTURE. YET THE STRUCTURE MODELS THE DATA STRUCTURE. THE COMPLETION IS THE ADDITION OF THE FILE MANAGEMENT PROCESSES AND THE INITIALIZATION PHASE. NOTATION IS THE SAME.

EXAMPLE

- COMPLETING: LIST ELEMENTARY OPERATIONS AND ASSIGN THEM TO MODULES



INSTRUCTOR NOTES

WE WILL ONLY SHOW AN EXAMPLE OF STRUCTURE CLASH, SO EXPLAIN BACKTRACKING ON THIS SLIDE.

VG 744.1

10-351

LIMITATIONS

- NOT ALL PROBLEMS ARE SO SIMPLE ...
 - PROGRAMS MUST MAKE PARTIAL DECISIONS BEFORE HAVING A COMPLETE SET OF DATA (SOLUTION USE BACKTRACKING)
 - MAKE AN ASSUMPTION ABOUT WHAT IS COMING NEXT
 - CONTINUE PROCESSING UNTIL YOU FIND OUT YOUR ASSUMPTION WAS WRONG
 - UNDO ANY DAMAGE AND MAKE A NEW ASSUMPTION BASED ON THE ADDITIONAL INFORMATION
 - THE STRUCTURE OF INPUT AND OUTPUT DATA DO NOT MATCH (SOLUTION RESOLVES STRUCTURE CLASH)
- LARGE PROGRAMS HAVE A VERY LARGE NUMBER OF DATA STRUCTURES MAKING METHODS HARD TO HANDLE

INSTRUCTOR NOTES

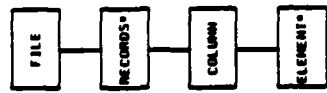
POINT OUT THAT THE OUTPUT PROGRAM IS "INVERTED" FROM THE OUTPUT FORMAT.

THE IN-CORE DATA STRUCTURE HOLDS ALL THE DATA READ IN BEFORE THE DATA IS WRITTEN OUT.

THERE ARE OTHER FORMS OF STRUCTURE CLASH USING AN INTERMEDIATE FILE OR ASYNCHRONOUS TASKS, COUPLED WITH BUFFERS ARE TWO OTHERS.

STRUCTURE CLASH

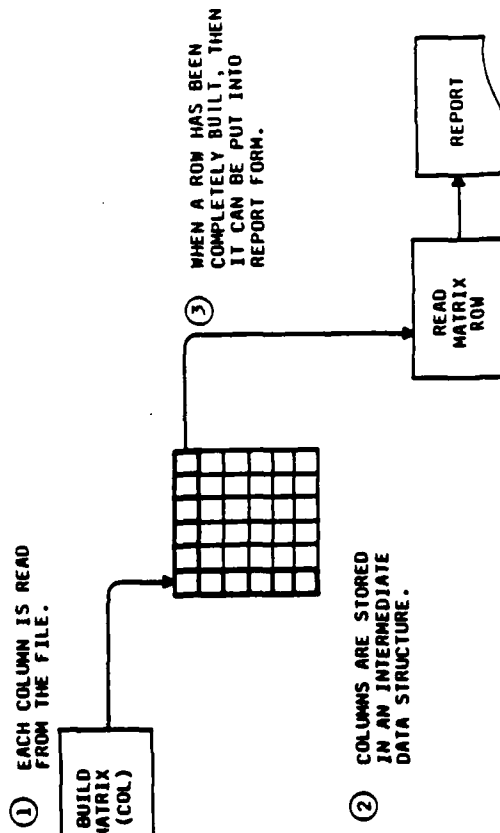
INPUT: A MATRIX IS STORED IN COLUMN-MAJOR ORDER.



OUTPUT: A REPORT MUST BE READ IN ROW-MAJOR ORDER.



PROGRAM INVERSION APPROACH



INSTRUCTOR NOTES

WARNIER-ORR IS CONSIDERED TO BE VERY SIMILAR TO JACKSON. WARNIER AND JACKSON DEVELOPED THEIR IDEAS AROUND THE SAME TIME. ORR EXTENDED THESE AND POPULARIZED THEM. IT USES A DIFFERENT FORM OF GRAPHICS BUT USES JACKSON COMPATIBLE GRAPHICS FOR DATA AND FUNCTIONS.

VG 744.1

10-371

OVERVIEW

- THE WARNIER-ORR METHOD ...

- WAS DEVELOPED IN FRANCE BY JEAN WARNIER
- WAS MODIFIED AND PUBLICIZED BY KEN ORR
- USES GRAPHICS TO MODEL BOTH DATA AND FUNCTIONS
- DEPARTS FROM THE BOX AND ARROW NOTATION

INSTRUCTOR NOTES

THERE ARE THREE (3) ELEMENTS TO THE WARNIER SYNTAX ...

NAMES ARE EITHER

- NOUNS -- FOR DATA

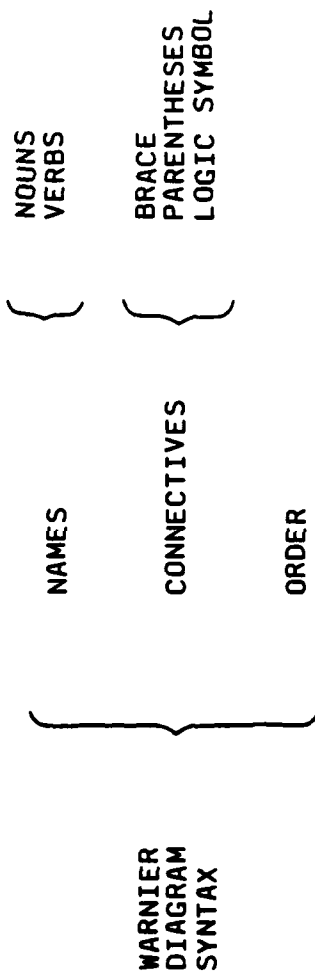
- VERBS -- FOR FUNCTIONS

THERE ARE THREE (3) KINDS OF GRAPHIC SYMBOLS THAT CONNECT (I.E. STRUCTURE) NAMES ...

BRACES AND LOGIC SYMBOLS DEFINE RELATIONSHIPS, WHEREAS PARENTHESES DEFINE ATTRIBUTES.

WARNIER DIAGRAM

SYNTAX



INSTRUCTOR NOTES

VG 744.1

10-391

SUMMARY

- WARNIER-ORR METHOD CAN BE VIEWED AS A SUBSET OF JACKSON STRUCTURED PROGRAMMING USING A DIFFERENT NOTATION
- MUST STILL USE JACKSON CONCEPTS TO DEVELOP FINAL PROGRAM STRUCTURE:
 - MATCHING STRUCTURES
 - RESOLVING CLASHES
 - COMPLETING THE PROGRAM

INSTRUCTOR NOTES

TECHNIQUES PROVIDE A COOKBOOK APPROACH FOR DEVELOPING A MODULAR STRUCTURE BASED ON DATA STRUCTURES. HIGH DEGREE OF TRACEABILITY AND UNDERSTANDABILITY IS INHERENT IN THE STRAIGHT-FORWARD COOKBOOK APPROACH.

CAN BE A PRODUCTIVE TECHNIQUE FOR DATA PROCESSING APPLICATIONS.

PRINCIPLES

- STRUCTURING, MODULARITY
 - DERIVED FROM DATA STRUCTURE
- UNIFORMITY, FORMALISM
 - SYNTAX AND SEMANTICS RULES FOR HANDLING COMPLEX DATA STRUCTURES

GOALS

- UNDERSTANDABILITY, TRACEABILITY
 - STRUCTURE OF PROGRAM RELATES DIRECTLY TO STRUCTURE OF DATA
- PRODUCTIVITY
 - A COOKBOOK APPROACH FOR DATA HANDLING APPLICATIONS
- MAINTAINABILITY, MODIFIABILITY
 - CHANGES TO INPUT OR OUTPUT MAP DIRECTLY TO CORRESPONDING PROGRAM STRUCTURE CHANGES

INSTRUCTOR NOTES

IN THIS SECTION WE WILL INTRODUCE THE IMPORTANT CONCEPTS OF HOS AND BRIEFLY DISCUSS AN AUTOMATED TOOL (USE.IT) THAT SUPPORTS IT.

HIGHER ORDER SOFTWARE (HOS)

- OVERVIEW
- FUNDAMENTAL BASIS
- CONTROL STRUCTURE EXAMPLES
- RADAR SYSTEM EXAMPLE
- USE.IT
- CRITIQUE

INSTRUCTOR NOTES

- HOS HELPS MODEL SYNCHRONOUS, ASYNCHRONOUS, NETWORKED, REAL TIME, INTERRUPT-DRIVEN, RECURSIVE, AND INTERACTIVE SYSTEMS.
- IT HAS BEEN AROUND IN SOME FORM SINCE THE MID-60'S, AND HAS BEEN USED BY NASA SINCE THE APOLLO SPACE PROGRAM. EXTENSIVE USE IN SHUTTLE PROGRAM.
- IT IS VERY HIERARCHICAL IN NATURE.

OVERVIEW

- HIGH ORDER SOFTWARE ...
 - GREW FROM THE APOLLO MOON-MISSION EFFORT
 - DEALS WITH SOFTWARE SYSTEMS
 - WAS DEVELOPED FOR REAL TIME SYSTEMS
 - EMPHASIS IS ON GENERATION OF PROVABLY CORRECT SOFTWARE
 - SIMPLIFIES WRITING SOFTWARE THAT IMPLEMENTS MATHEMATICAL EQUATIONS
 - HAS BEEN USED AS AN ANALYSIS AS WELL AS DESIGN METHOD

INSTRUCTOR NOTES

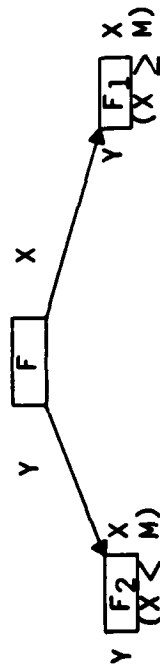
- (a) AXIOMS CONTROL DECOMPOSITIONS. PARENTS CAN INVOKE ONLY ITS OFFSPRING, CONTROLS THE ORDER OF THEIR INVOCATION, AND CONTROLS THEIR INPUT AND OUTPUT (i.e., ACCESS RIGHTS).
- (b) ALL CONTROL STRUCTURES ARE DERIVED FROM THE AXIOMS.
- (c) DATA TYPES AUTOMATICALLY DEFINE PRIMITIVE OPERATIONS ON DATA.
- (d) FUNCTIONS REPRESENT SPECIFIC TRANSFORMATIONS OF INPUT VALUES OF DATA TYPE MEMBERS TO OUTPUT VALUES OF DATA TYPE MEMBERS.
- (e) THE AXIOMATIC NATURE OF HOS ALLOWS THE OUTPUTS TO BE TRANSFORMED INTO REPRESENTATIONS TO BE USED AS INPUTS TO JACKSON, PSL/PSA, SADT, SREM, AND PETRI NETS.

HIGHER ORDER SOFTWARE
FUNDAMENTAL BASIS

- HIERARCHY REPRESENTS DECOMPOSITION OF SOFTWARE FROM HIGHEST LEVEL (TOP) TO PRIMITIVE OR PREVIOUSLY DEVELOPED FUNCTIONS.
- THE SIX AXIOMS THAT MAKE FUNCTION DECOMPOSITION MATHEMATICALLY PRECISE:
 - AXIOM 1: A GIVEN MODULE CONTROLS THE INVOCATION OF THE SET OF FUNCTIONS ON ITS IMMEDIATE, AND ONLY ITS IMMEDIATE, LOWER LEVEL.
 - AXIOM 2: A GIVEN MODULE CONTROLS THE RESPONSIBILITY FOR ELEMENTS OF ONLY ITS OWN OUTPUT SPACE.
 - AXIOM 3: A GIVEN MODULE CONTROLS THE OUTPUT ACCESS RIGHTS TO EACH SET OF VARIABLES WHOSE VALUES DEFINE THE ELEMENTS OF THE OUTPUT SPACE FOR EACH IMMEDIATE, AND ONLY EACH IMMEDIATE, LOWER LEVEL FUNCTION.
 - AXIOM 4: A GIVEN MODULE CONTROLS THE INPUT ACCESS RIGHTS TO EACH SET OF VARIABLES WHOSE VALUES DEFINE THE ELEMENTS OF THE INPUT SPACE FOR EACH IMMEDIATE, AND ONLY EACH IMMEDIATE, LOWER LEVEL FUNCTION.
 - AXIOM 5: A GIVEN MODULE CONTROLS THE REJECTION OF INVALID ELEMENTS OF ITS OWN, AND ONLY ITS OWN, INPUT SET.
 - AXIOM 6: A GIVEN MODULE CONTROLS THE ORDERING OF EACH TREE FOR THE IMMEDIATE, AND ONLY THE IMMEDIATE, LOWER LEVEL.
- HIERARCHY CAN BE USED WITH PROGRAM GENERATOR TO "AUTOMATICALLY" PRODUCE "CORRECT" PROGRAMS.

INSTRUCTOR NOTES

FOR "OR", WE HAVE



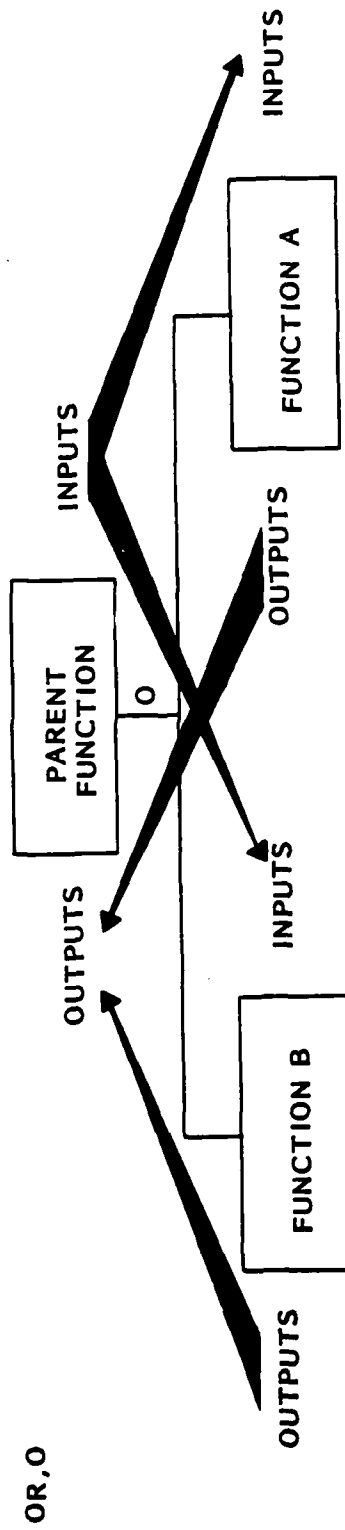
FIRST, X IS CHECKED TO SEE IF IT IS $\geq M$ (IF IT IS $Y = F_1(X)$).

IF IT IS NOT, $Y = F_2(X)$.

WE ARE ONLY SHOWING EXAMPLES OF ONE PRIMITIVE AND ONE NON-PRIMITIVE STRUCTURE. THERE ARE ALSO PRIMITIVE JOIN AND INCLUDE STRUCTURES AND NON-PRIMITIVE COOR AND COINCLUDE STRUCTURES.

PRIMITIVE CONTROL STRUCTURE

(OR)



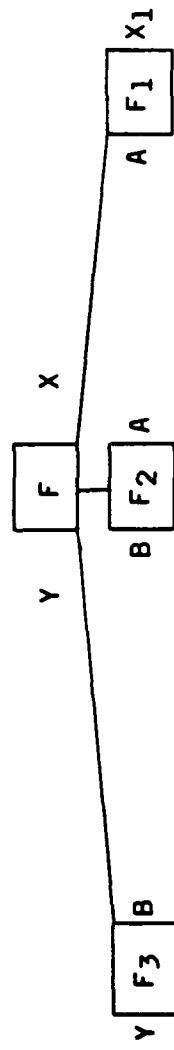
FUNCTION A OR FUNCTION B IS PERFORMED

- INPUTS TO ALL FUNCTIONS ARE IDENTICAL (INCLUDING ORDER).
- ALL INPUTS ARE USED IN A PARTITION FUNCTION P WHICH CHOOSES WHICH OFFSPRING TO EXECUTE.
- OUTPUTS FROM ALL FUNCTIONS ARE IDENTICAL (INCLUDING ORDER).

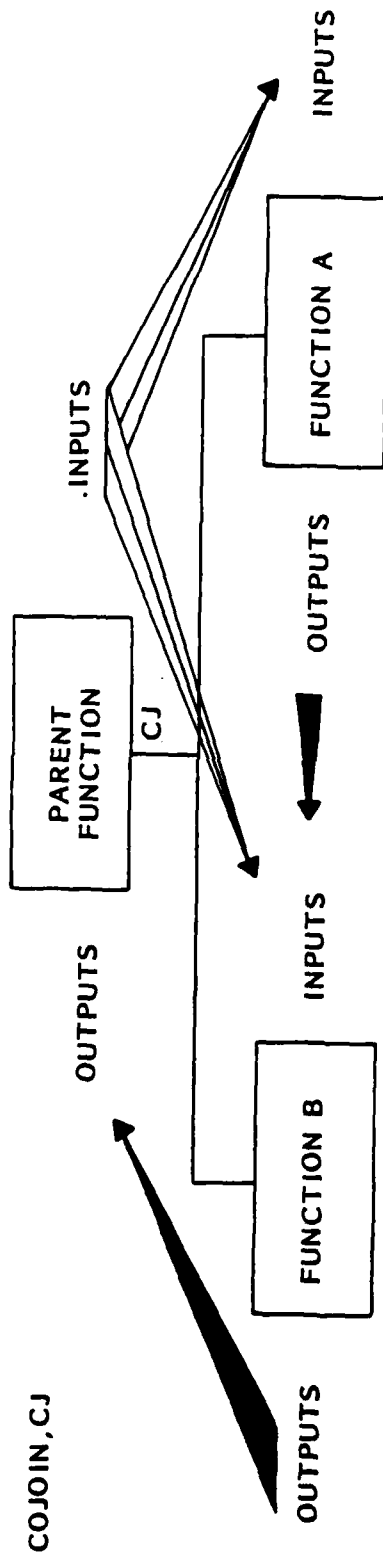
SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

FOR COJOIN,



NON-PRIMITIVE CO-CONTROL STRUCTURE
(COJOIN)



FUNCTION A IS PERFORMED FIRST, THEN FUNCTION B

- INPUTS TO RIGHT OFFSPRING ARE A SUBSET OF THE PARENT'S INPUTS.
- INPUTS TO LEFT OFFSPRING COME FROM EITHER THE PARENT'S INPUTS OR THE
 - OUTPUTS OF THE RIGHT OFFSPRING.
- OUTPUTS FROM PARENT ARE IDENTICAL TO THE OUTPUTS FROM THE LEFT OFFSPRING.

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

VG 744.1

10-45

INSTRUCTOR NOTES

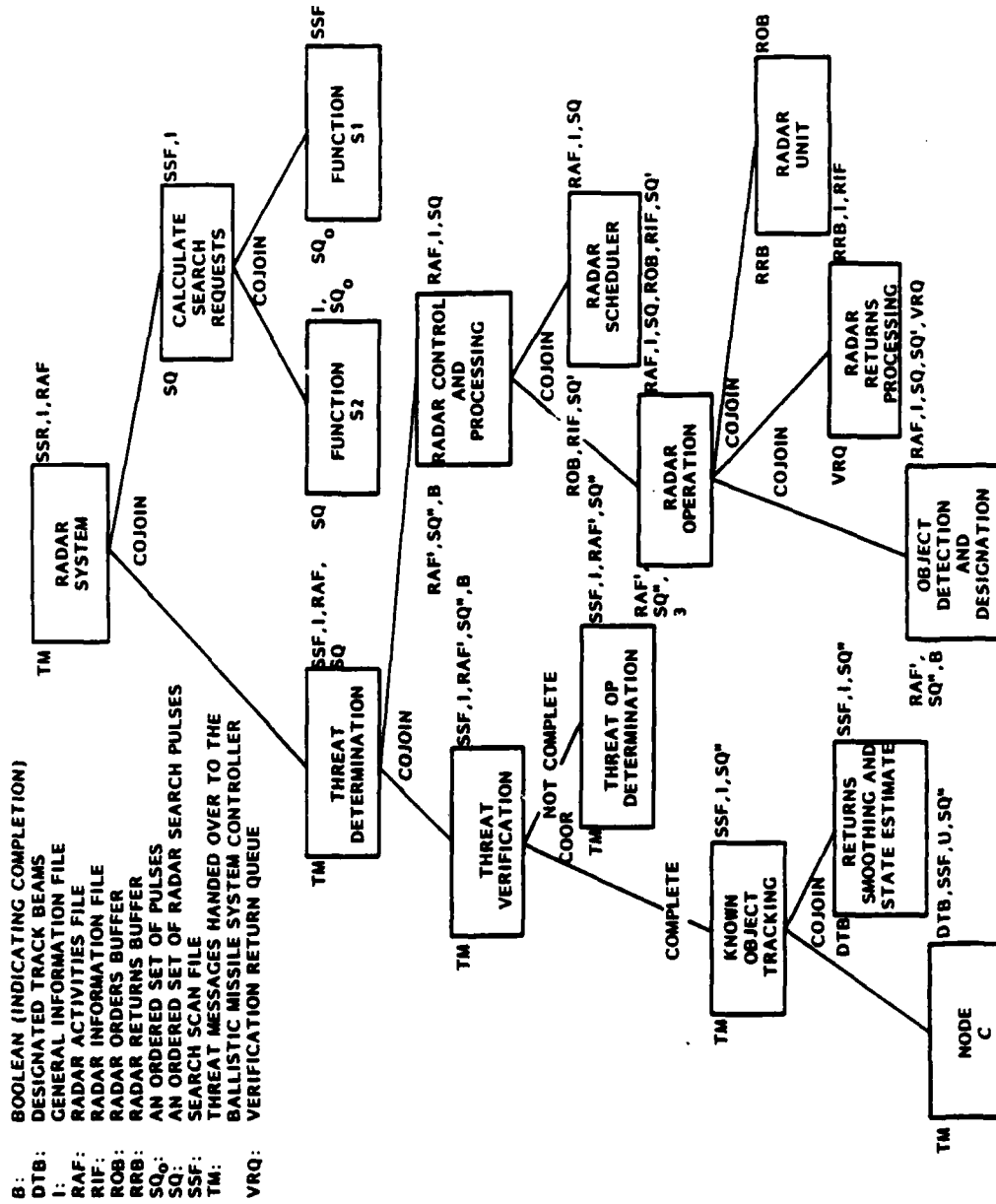
GO THROUGH QUICKLY. NEXT SLIDE SHOWS THREAT DETERMINATION LEAF DETAIL

VG 744.1

10-461

RADAR SYSTEM

EXAMPLE



SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

SOFTWARE FOR APPLYING SUCH TECHNIQUES REQUIRES THE FOLLOWING COMPONENTS

1. A LANGUAGE FOR EXPRESSING FUNCTIONS AND THEIR DECOMPOSITION INTO OTHER FUNCTIONS
2. AN INTERACTIVE SCREEN FACILITY FOR CONSTRUCTING AND MANIPULATING THE CONTROL MAPS, AND ALLOWING THE USER TO CORRECT ERRORS INTERACTIVELY
3. A LIBRARY OF DATA TYPES, PRIMITIVE FUNCTIONS, AND PREVIOUSLY DEFINED MODULES
4. AN ANALYZER ROUTINE FOR AUTOMATICALLY CHECKING THAT ALL THE RULES THAT GIVE PROBABLY CORRECT LOGIC HAVE BEEN FOLLOWED
5. A GENERATOR THAT AUTOMATICALLY GENERATES PROGRAM CODE

AUTOMATION OF HOS

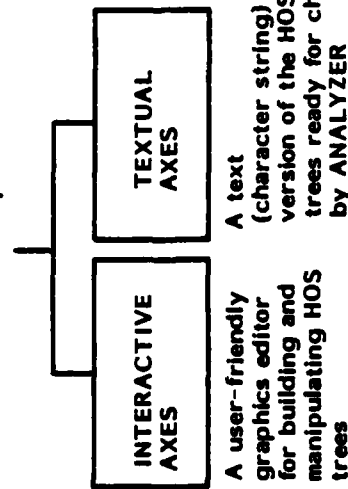
(USE.IT)

AXES is a language for describing HOS function hierarchies (with their data types and control structures)

RAT converts the logic which has been checked by ANALYZER into program code ready for execution



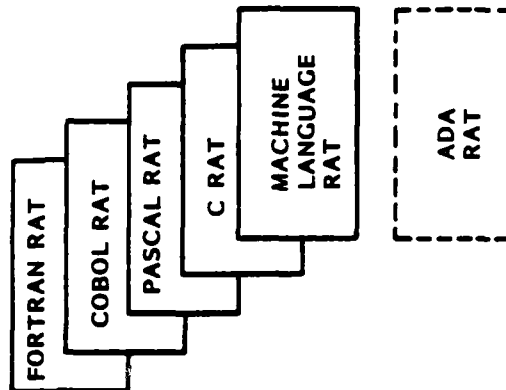
AXES has two components:



INTERACTIVE AXES
A user-friendly graphics editor for building and manipulating HOS trees

TEXTUAL AXES
A text (character string) version of the HOS trees ready for checking by ANALYZER

Different versions of RAT produce code in different languages



(UNDER DEVELOPMENT)

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

10-47

VG 744.1

INSTRUCTOR NOTES

HOS IS THE MOST FORMAL OF ANY OF THE TECHNIQUES WE HAVE LOOKED AT AND IS ABLE TO PROVIDE THE MOST ASSISTANCE IN PRODUCING CORRECT, VERIFIABLE, AND RELIABLE SYSTEMS. PRODUCTIVITY IS ENHANCED BY THE AUTOMATED USE.IT TOOL.

VG 744.1

10-48i

HOS

PRINCIPLES

- FORMALISM, UNIFORMITY
 - PROVABLY CORRECT AXIOMATIC BASIS
- STRUCTURING, MODULARITY
 - HIERARCHICAL STRUCTURE BASED ON STRICT DECOMPOSITION RULES

GOALS

- CORRECTNESS, VERIFIABILITY
 - FORMAL, MATHEMATICAL BASISFACILITATES VERIFYING CORRECTNESS
- RELIABILITY
 - ORIENTED SPECIFICALLY FOR HIGH RELIABILITY SYSTEMS
- PRODUCTIVITY
 - AUTOMATION ASSISTS IN GENERATING CODE FROM DESIGN DESCRIPTION

INSTRUCTOR NOTES

VG 744.1

11-1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

SECTION 11

DETAILED DESIGN

VG 744.1

INSTRUCTOR NOTES

WE'RE PRESENTING THREE DETAILED DESIGN TECHNIQUES. TWO, HIPO AND NSSF HAVE BEEN AROUND A WHILE AND ARE BASICALLY GRAPHICAL IN NATURE.

PDL HAS EVOLVED FROM OLD PSEUDO-CODE TECHNIQUES AND IS BECOMING A HIGHLY VISIBLE APPROACH THAT MAY WELL BE MANDATED ON MANY PROJECTS.

DETAILED DESIGN TECHNIQUES

- PDL - PROGRAM DESIGN LANGUAGE
- HIPO - HIERARCHAL INPUT PROCESSING OUTPUT
- NSSF - NASSI-SCHNEIDERMAN STRUCTURED FLOWCHARTS

INSTRUCTOR NOTES

WE AREN'T GOING TO DEFINE A SINGLE PDL. WE ARE GOING TO GIVE YOU AN INTRODUCTION TO THE ISSUES INVOLVED WITH DEFINING AND USING PDL'S.

THE DIFFICULTY IN UPDATING FLOWCHARTS REALLY LED TO PDL. IN MAINTAINING A SYSTEM IT IS FOOLISH TO RELY ON FLOWCHARTS. SO A MECHANISM WAS NEEDED TO: 1) DESCRIBE THE HIGH LEVEL STRUCTURE AND 2) BE UPDATED EASILY.

STATE THAT MANY ELEMENTS OF DESIGN ARE GRAPHIC IN NATURE. TEXTUAL LANGUAGES, LIKE PDL, NATURALLY FALL SHORT OF MEETING ALL THE NEEDS OF A DESIGNER.

PROGRAM DESIGN LANGUAGE OVERVIEW

- IS A TEXTUAL LANGUAGE, PRECISE ENOUGH TO DESCRIBE SOFTWARE, YET EXPRESSIVE ENOUGH TO DESIGN WITH.



ALWAYS COMPETE.

PROPER BALANCE IS UNKNOWN.

- PDL'S GREW OUT OF THE PROBLEMS OF TRYING TO KEEP UP-TO-DATE FLOWCHARTS OF THE SYSTEM.
- PDL'S WILL BE REQUIRED FOR DOCUMENTING DETAIL DESIGNS.

INSTRUCTOR NOTES

A DESIGN LANGUAGE IS A TEXTUAL REPRESENTATION FOR THE PRECISE EXPRESSION OF PROGRAM OR SYSTEM DESIGNS.

IN USING THE IEEE PDL STANDARD, THERE IS A "RECOMMENDED PRACTICE" THAT GOES ALONG WITH IT. IT PROVIDES GENERAL GUIDANCE IN USING THE PDL, THE FEATURES OF AN Ada PDL, DESIGN LANGUAGE CHARACTERISTICS, MANAGEMENT ISSUES, ETC.

Ada AS A PROGRAM DESIGN LANGUAGE

- IEEE HAS PRODUCED A DRAFT SET OF GUIDELINES FOR USE OF Ada AS A DESIGN LANGUAGE.
 - ONLY "REAL" STANDARD FOR MODERN PDL.
 - USED AS A BASIS FOR THIS SECTION
- IEEE MOTIVATING GOALS FOR USE OF Ada AS A PDL ARE TO:
 - UTILIZE THE POWER OF THE Ada PROGRAMMING LANGUAGE IN THE DESIGN PROCESS.
 - ENHANCE COMMUNICATION BY USING THE SAME LANGUAGE NOTATION THROUGHOUT THE LIFE CYCLE.
 - SUPPORT QUALITY SOFTWARE DESIGN BY FOCUSING ON APPROPRIATE LEVELS OF DESIGN DETAIL.
 - CAPITALIZE ON THE EMERGING AVAILABILITY OF Ada TOOLS AND INDUSTRY SUPPORT FOR THE Ada LANGUAGE.
 - PROVIDE A MECHANISM THAT SUPPORTS THE TRANSITION TO Ada BASED SOFTWARE ENGINEERING PRACTICE.
 - PROVIDE A BASIS FOR STANDARDIZATION.

INSTRUCTOR NOTES

THE USE OF A DESIGN LANGUAGE CAN INCREASE PRODUCTIVITY IN A NUMBER OF WAYS. FIRST, AS LONG AS DESIGN DOCUMENTATION CAN BE WRITTEN IN MACHINE READABLE FORM, AUTOMATED TOOLS CAN BE CREATED TO CHECK FOR COMPLETENESS AND CONSISTENCY, AND TO AUTOMATE DEVELOPMENT ACTIVITIES. SECOND, SUCH A LANGUAGE FACILITATES COMMUNICATION BETWEEN VARIOUS MEMBERS OF A PROJECT TEAM. FINALLY, IT IS ADVANTAGEOUS TO HAVE A SINGLE NOTATION THAT CAN BE USED THROUGHOUT ALL ACTIVITIES OF THE SOFTWARE LIFE CYCLE.

USE OF A DESIGN LANGUAGE CAN INCREASE SOFTWARE QUALITY BY FACILITATING THE EARLY DETECTION AND CORRECTION OF ERRORS BY HELPING TEAM MEMBERS TO COMMUNICATE THROUGH A COMMON LANGUAGE, AND BY PROVIDING A MECHANISM FOR VERIFICATION OF DESIGN AND THE TRANSLATION OF THE DESIGN NOTATION. A DESIGN LANGUAGE PROVIDES A VEHICLE BOTH FOR COMMUNICATIONS AND FOR SUPPORTING VARIOUS ACTIVITIES OF THE PRODUCT LIFE CYCLE, AND SHOULD BE HUMAN ENGINEERED, PRECISE, ANALYZABLE, VERIFIABLE AND SUPPORTIVE OF VARIOUS PROGRAMMING METHODOLOGIES.

RATIONALE FOR USING A PDL

- TO INCREASE PRODUCTIVITY
 - PDLs ARE MACHINE READABLE; TOOLS CAN CHECK FOR COMPLETENESS, CONSISTENCY AND CONFORMANCE TO STANDARDS.
 - PDLs DEFINE A COMMON LANGUAGE TO ENHANCE COMMUNICATIONS WITHIN A PROJECT.
- TO IMPROVE SOFTWARE QUALITY
 - PDLs FACILITATE EARLY DETECTION AND CORRECTION OF ERRORS BY AIDING IN COMMUNICATION AND VERIFICATION.
- TO MINIMIZE THE RISKS INVOLVED IN DEVELOPING AND MAINTAINING SOFTWARE
 - PDLs MAKE DESIGNS VISIBLE EARLY.
 - COMMON PDL AND IMPLEMENTATION LANGUAGE REDUCES TRAINING NEEDS.

INSTRUCTOR NOTES

A DESIGN METHODOLOGY IS A BODY OF PROCEDURES AND TECHNIQUES WHICH CAN BE USED TO CONSTRUCT A SYSTEM DESIGN. THE DESIGN ITSELF IS AN ABSTRACTION OF THE PROPOSED SYSTEM; WHEN THE SYSTEM IS IMPLEMENTED ACCORDING TO THE DESIGN, IT IS EXPECTED TO PERFORM ACCORDING TO THE SYSTEM SPECIFICATION AND TO SATISFY THE SYSTEM REQUIREMENTS. THE DESIGN LANGUAGE SHOULD SUPPORT THE DESCRIPTION OF A PRODUCT AT THE APPROPRIATE LEVELS OF DETAIL FOR EACH ACTIVITY OF A GIVEN METHODOLOGY.

PDL REQUIREMENTS

- A PDL MUST SUPPORT ...
 - ABSTRACTION - EMPHASIZING PARTICULAR CONCEPTS WHILE SUPPRESSING UNNECESSARY DETAIL.
 - DECOMPOSITION - DIVIDING A LARGE SYSTEM INTO SMALLER, MORE MANAGEABLE PIECES WHILE MAINTAINING A FIXED LEVEL OF DETAIL.
 - INFORMATION HIDING - ISOLATING DESIGNATED INFORMATION TO CONTROL THE DEPENDENCE ON A PARTICULAR IMPLEMENTATION.
 - STEPWISE REFINEMENT - PROGRESSIVELY ADDING DETAIL TO A DESIGN DESCRIPTION.
 - MODULARIZATION - ISOLATING PORTIONS OF A SYSTEM INTO LOGICALLY SEPARABLE PARTS SUCH THAT A CHANGE IN ONE PART HAS MINIMAL IMPACT ON OTHER PARTS.

NOTE: THESE ARE SOME OF THE DOD'S REQUIREMENTS FOR Ada

INSTRUCTOR NOTES

THESE CHARACTERISTICS ARE DEPENDENT UPON THE METHODOLOGY USED.

VG 744.1

11-61

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

PDL CHARACTERISTICS

- A PROGRAM DESIGN LANGUAGE SHOULD ADDRESS ALL OF THE FOLLOWING:
 - EXPRESSIVE POWER
 - ALGORITHM DESIGN
 - DATA STRUCTURES
 - SUPPLEMENTARY INFORMATION
 - CONNECTIVITY
 - MANAGEMENT OF COMPLEXITY
 - HUMAN FACTORS
 - ANALYZABILITY
 - RELATIONSHIP BETWEEN THE PDL AND IMPLEMENTATION LANGUAGE
 - TOOL IMPACT ON THE PROGRAM DESIGN LANGUAGE
 - IMPLEMENTATION CONSIDERATIONS

INSTRUCTOR NOTES

THIS CHART ILLUSTRATES THAT SIGNIFICANT AMOUNTS OF DESIGN INFORMATION ARE EXPRESSIBLE USING Ada CONSTRUCTS. IF ANOTHER LANGUAGE WAS CHARTED, E.G. FORTRAN, THE CHART WOULD BE MUCH LESS FILLED IN.

MAPPING OF Ada PDL FEATURES TO DESIGN CHARACTERISTICS

DESIGN CHARACTERISTICS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
ABSTRACTION	*	*	*	*		*		*	*	*		*		*	*	*	
DATA FLOW		*	*				*		*						*	*	*
CONTROL FLOW	*	*	*						*		*						*
ALGORITHM		*	*						*		*						*
DATA STRUCTURE DESIGN	*	*				*	*		*	*			*	*	*	*	
DATA REFERENCE	*	*	*						*								
INTERFACES	*	*	*	*	*	*	*		*	*	*	*	*				
DEPENDENCIES	*	*	*	*	*	*	*	*	*	*	*	*	*				
REUSABILITY	*	*		*		*	*		*					*	*	*	
DEFERRED DECISIONS	*	*		*		*		*	*	*				*	*		

Ada LANGUAGE FEATURES AND CONSTRUCTS

A	PACKAGES	J	PRIVATE
B	SUBPROGRAMS	K	EXCEPTIONS
C	TASKS	L	REP SPECS
D	SPEC/BODY	M	ALLOCATIONS
E	PRAGMA	N	GENERICS
F	TYPES	O	PREDEFINED PACKAGES
G	OBJECTS	P	NAMING
H	STUBS/IS SEPARATE	Q	STATEMENTS
I	COMMENTS		

SOURCE: IEEE PDC GUIDELINES, 1985

VG 744.1

11-7

INSTRUCTOR NOTES

IN AN Ada DESIGN LANGUAGE, COMMENTS PROVIDE AN INDISPENSABLE MECHANISM FOR ADDING NEW CONSTRUCTS TO THE DESIGN LANGUAGE TO ANNOTATE THE DESIGN.

UNSTRUCTURED COMMENTS SHOULD BE USED FOR NATURAL LANGUAGE EXPLANATION OF STATEMENTS MADE IN THE Ada LANGUAGE. THESE COMMENTS CAN BE USED TO INDICATE ANY INFORMATION REQUIRED IN THE DESIGN PROCESS WHEN FORMAL STRUCTURES ARE NOT REQUIRED.

A STRUCTURED COMMENT, OR ANNOTATION, PROVIDES EXPLANATORY DESIGN INFORMATION. IT IS IDENTIFIED BY A SENTINEL CHARACTER, WORD OR PHRASE IMMEDIATELY FOLLOWING THE DOUBLE DASH THAT INDICATES A COMMENT TO AN Ada COMPILER (E.G. "--!" OR "--*"). THE SENTINEL CHARACTER HAS TWO FUNCTIONS. THE FIRST IS TO HIGHLIGHT THE COMMENT AS BELONGING TO THE FORMAL STRUCTURE OF THE DESIGN LANGUAGE, ALERTING ANY REVIEWER OF THE DESIGN DOCUMENTATION THAT THE INFORMATION THAT FOLLOWS IS OF SPECIAL SIGNIFICANCE. THE SECOND FUNCTION IS TO INDICATE TO TOOLS WHICH PROCESS THE DESIGN THAT THE TOOL MAY BE REQUIRED TO ACT UPON THE INFORMATION GIVEN IN THE COMMENT. ASSOCIATED WITH THIS SYMBOL ARE SEMANTIC RULES INDICATING WHETHER THE COMMENT APPLIES TO THE PRECEDING OR TO THE FOLLOWING Ada CONSTRUCT AND WHAT SORT OF CONSTRUCTS ARE ALLOWED AFTER THE SYMBOL ACCORDING TO THE CONTEXT IN WHICH IT APPEARS. ALTHOUGH THE STRUCTURED COMMENT IS A RESTRICTION IN TERMS OF THE SYNTAX OF THE Ada LANGUAGE, IT PROVIDES A MECHANISM FOR EXTENDING THE DESIGN LANGUAGE SEMANTICS TO INCLUDE ADDITIONAL DESIGN ORIENTED INFORMATION.

ROLE OF COMMENTS IN Ada PDL

- UNSTRUCTURED COMMENTS SHOULD BE USED FOR NATURAL LANGUAGE EXPLANATIONS
 - FOR HUMAN TO HUMAN COMMUNICATION
 - FOR RATIONALE
 - USE NORMAL COMMENT DELIMITERS
- STRUCTURED COMMENTS (ANNOTATION) PROVIDE EXPLANATORY DESIGN INFORMATION
 - EXTENDS Ada SYNTAX TO EXPRESS DESIGN INFORMATION NOT EASILY EXPRESSED IN Ada

// an English statement of conditions //

--* HIGH LEVEL DESIGN

--*

--*

--*

--* END HIGH LEVEL DESIGN

↓
INCLUDES A STATEMENT OF
HIGH LEVEL DESIGN

INSTRUCTOR NOTES

USE THIS SLIDE TO SHOW THE READABILITY OF AN Ada PDL AND THE TRACEABILITY TO THE Ada IMPLEMENTATION.

NOTATION

// ~~~~~ // - AN ILLEGAL Ada CONSTRUCT TO EXPRESS DESIGN INFORMATION THAT WILL BE LATER EXPANDED OR THAT IS MORE INFORMATIVE EXPRESSED IN ENGLISH THAN IN Ada.

AN ADA PDL (SAMPLE)

```

procedure SORT (TABLE : in out TABLE_Type) is
begin
  if // TABLE has more than one entry // then
    while // TABLE is not sorted // loop
      for // each pair of entries in TABLE // loop
        if // 1st entry is greater than 2nd
          then
            //exchange the two entries//;
          end if;
        end loop;
      end loop;
    end if;
  end SORT;

```

DESIGN

EXPANDS
TO

```

procedure Sort (Table : in out Table_Type) is
  Swapped_Items : Boolean;
  Temp : Item_Type;
begin -- Sort
  if Table'Length > 1 then
    Swapped_Items := True;
    -- loop_while the table is not sorted
    while not Swapped_Items loop
      Swapped_Items := False;
      for I in Table'First..Index_Type'Pred(Table'Last)
        loop
          if Table(I) > Table(Index_Type'Succ(I))
            then
              Swapped_Items := True;
              Temp := Table(I);
              Table(I) := Table(Index_Type'Succ(I))
              Table(Index_Type'Succ(I)) := Temp;
            end if;
          end loop; -- for each pair of entries
        end loop; -- for sorting Table
      end Sort;
    end if;
  end if;
end Sort;

```

IMPLEMENTATION

INSTRUCTOR NOTES

- IF NO ONE CHIMES UP FOR DISCUSSION, AS THE QUESTION "SHOULD Ada BE USED AS A PDL?"

PDL SUMMARY

- PDL'S HAVE A PLACE IN DESIGN, THEY ...
 - CAN INCREASE PRODUCTIVITY
 - CAN IMPROVE QUALITY
 - CAN MINIMIZE RISK
 - WILL BE REQUIRED BY DOD-STD-SDS.
- PDL'S HAVE DRAWBACKS ...
 - PEOPLE WILL CODE INSTEAD OF DESIGN
 - FULLY COMPILABLE PDL'S ARE NOT EXPRESSIVE
ENOUGH FOR REAL TIME DESIGN

INSTRUCTOR NOTES

PDL IS REALLY A TOOL NOT A TECHNIQUE - IT IS RELATIVELY FORMAL AND UNIFORM, ESPECIALLY IF COMPATIBILITY WITH Ada SYNTAX RULES IS A REQUIREMENT. THE ABILITY TO ATTAIN MODULARITY, STRUCTURING, AND ABSTRACTION ARE INHERITED FROM THE LANGUAGE BASE FOR THE PDL. THESE AND OTHER PRINCIPLES, SUCH AS HIDING AND SEPARATION OF CONCERNS, CAN BE UTILIZED BUT ARE NOT REALLY FACILITATED OR ENFORCED THROUGH ANY METHODOLOGY. VARIOUS METHODOLOGIES, SUCH AS OBJECT ORIENTED DESIGN, WILL PROBABLY BE CREATED TO ADD A RECOMMENDED APPROACH TO USING THIS TOOL.

BECAUSE PDL IS ONLY A TOOL, IT ALONE DOES NOT HELP US IN ATTAINING MANY OF OUR GOALS. ONLY THOSE THAT DERIVE FROM ITS FORMALITY, SUCH AS VERIFIABILITY AND CORRECTNESS, ARE EASILY ATTAINED. UNDERSTANDABILITY IS IMPROVED OVER THE HIGH LEVEL LANGUAGE ITSELF, BUT STILL SUFFERS WHEN COMPARED TO GRAPHICAL OR MORE ABSTRACT TECHNIQUES.

PDL

PRINCIPLES

- FORMALISM, UNIFORMITY
 - ALMOST AS HIGH AS FOR A PROGRAMMING LANGUAGE
- MODULARITY, STRUCTURING, ABSTRACTION
 - SAME AS FOR A HIGH LEVEL LANGUAGE
- HIDING, SEPARATION OF CONCERNS
 - CAN BE SUPPORTED, DOESN'T COME AUTOMATICALLY

GOALS

- VERIFIABILITY, CORRECTNESS
 - CAN USUALLY BE MACHINE PROCESSED
- UNDERSTANDABILITY
 - IMPROVED OVER HIGH LEVEL LANGUAGE

INSTRUCTOR NOTES

HIPO WAS DEVELOPED IN THE EARLY 70S BY IBM. IT WAS USED MORE TO DOCUMENT STABLE SOFTWARE FOR UNDERSTANDABILITY BY MAINTAINERS OR USERS THAN AS A TECHNIQUE USED BY DESIGNERS IN THE MIDDLE OF A DEVELOPMENT CYCLE. THIS IS BECAUSE THE GRAPHICS ARE RELATIVELY ELABORATE AND COSTLY TO PRODUCE.

THE EXACT TECHNIQUE HASN'T REALLY CAUGHT ON OUTSIDE OF IBM BUT MANY VARIATIONS HAVE BEEN DEVELOPED, STILL CALLED HIPO, BUT WITH LESS GRAPHICS AND POSSIBLY THE USE OF A PDL FOR THE PROCESSING DESCRIPTION.

HIPO

OVERVIEW

- HIPO SEES SOFTWARE AS
 - A HIERARCHY OF FUNCTIONS, EACH HAVING AN INPUT-PROCESS-OUTPUT NATURE
- HIPO
 - IS A DOCUMENTATION TOOL DEVELOPED BY IBM
 - DOCUMENTS PROCEDURAL DETAILS OF SOFTWARE
 - COLLECTS INFORMATION INTO PACKAGES

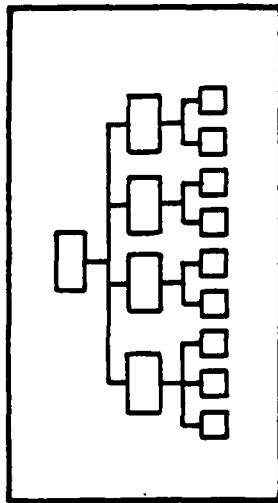
INSTRUCTOR NOTES

THE EMPHASIS WITH HIPO IS ON GRAPHICS TO AID UNDERSTANDABILITY. THE HIERARCHY IS REPRESENTED BY A HIERARCHICAL VISUAL TABLE OF CONTENTS (VTOC). THESE ARE SIMILAR TO BUT NOT AS REFINED AS THE STRUCTURED ANALYSIS STRUCTURE CHARTS.

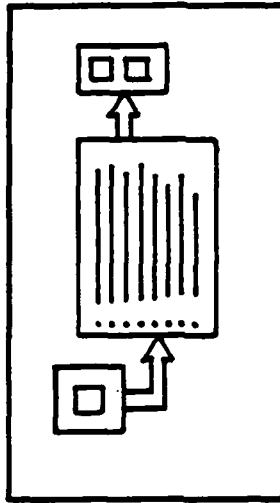
THE OVERALL PROCESSING OF THE SYSTEM IS DESCRIBED IN AN OVERVIEW DIAGRAM WHICH SHOWS INPUTS, PROCESSING, AND OUTPUTS. THESE OVERVIEW DIAGRAMS SOMETIMES USE RELATIVELY SOPHISTICATED GRAPHICS. EACH MODULE (FROM THE VTOC) IS REPRESENTED BY A DETAILED DIAGRAM. THESE HAVE LESS GRAPHICS BUT STILL USE BOLD ARROWS TO ASSOCIATE INPUTS AND OUTPUTS WITH PROCESSING.

HIPO

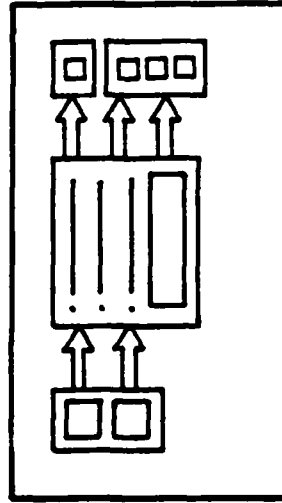
- 1 A VISUAL TABLE OF CONTENTS
(HIERARCHY)



- 2 OVERVIEW DIAGRAMS
(FOR HIGH LEVELS IN THE VTOC)



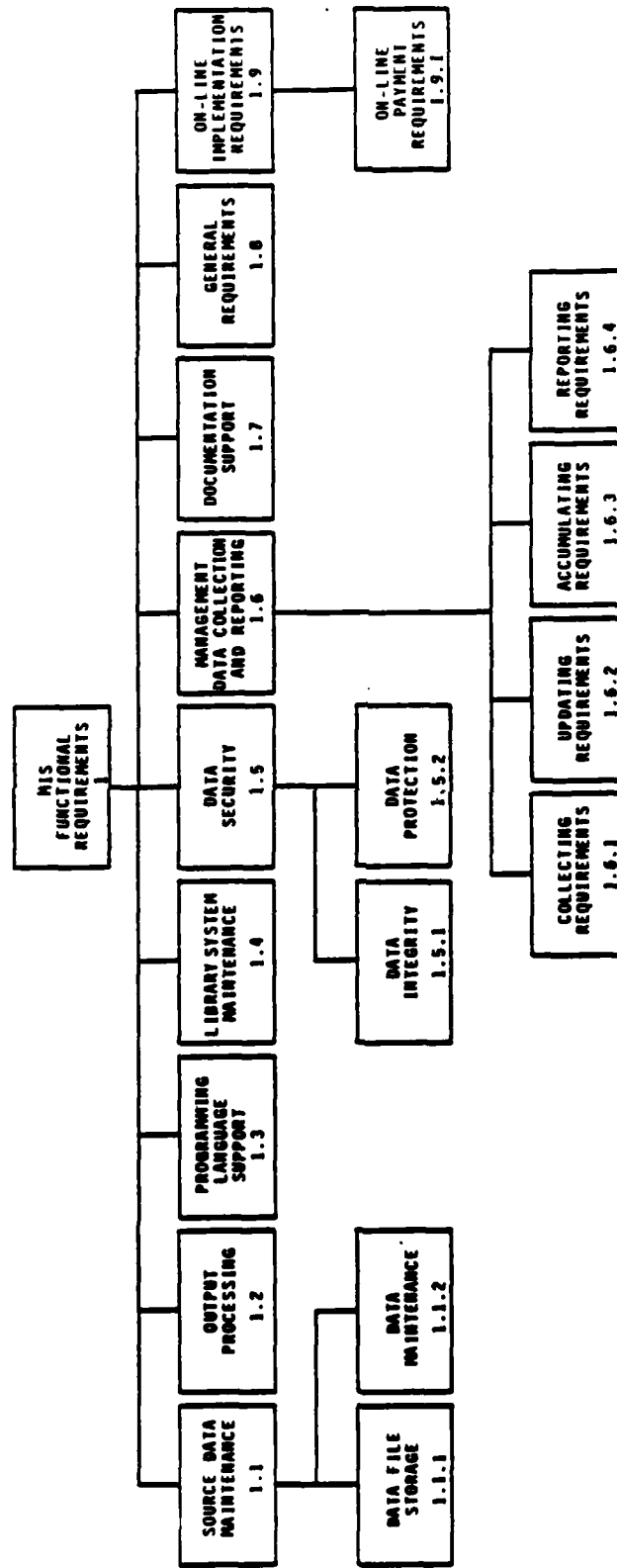
- 3 DETAIL DIAGRAMS
(FOR LOW LEVELS IN THE VTOC)



INSTRUCTOR NOTES

- NOTE THAT THIS ONLY SHOWS SIMPLE HIERARCHY. CONTRAST THIS TO A CONSTANTINE STRUCTURE CHART.
- NOTE THAT THE VTOC IS THE BIG PICTURE.

VISUAL TABLE OF CONTENTS



INSTRUCTOR NOTES

GO OVER QUICKLY - POINT OUT HIGHLIGHTS.

THIS IS A SAMPLE OF A NON-IBM HIPO. A LOT OF OTHER COMPANIES USE IT THIS WAY WITHOUT ANY GRAPHICS - THIS CAN ALL BE DONE WITH A TEXT EDITOR AND A STANDARD LINE PRINTER AND SIMPLE SOFTWARE.

HIPO SAMPLE

TITLE: EXECUTIVE EXIMMC INTERRUPT MANAGEMENT MONITOR CLOCK INTERRUPT HANDLER
PURPOSE: TO HANDLE THE OCCURRENCE OF A MONITOR CLOCK INTERRUPT

INPUTS	PROCESSING	OUTPUTS
FROM SEIO: STATUS REGISTERS PROGRAM COUNTER EXECUTIVE REGISTERS FROM EXEC COMMON: EXAVNXTM - NEXT MONITOR CLOCK TIME EXAIMCLK - STARTING ADDRESS OF MONITOR CLOCK AREA	SAVE EXECUTIVE REGISTERS SAVE STATUS REGISTERS AND PROGRAM COUNTER CALL EXIMPITL (PROGRAM COUNTER, STATUS REGISTERS) TO ESTABLISH RETURN LINKAGE ENABLE INTERRUPTS IF (TIME ON TOP OF AREA = 0) THEN INCREMENT MONITOR CLOCK FLAG (EXAFMON) TO SIGNAL EXIM THAT TIME IS UP CURRENT MONITOR CLOCK TIME = NEXT MC TIME CLEAR NEXT MONITOR CLOCK TIME ENDIF IF (CURRENT MONITOR CLOCK TIME > 0) THEN IF (CURRENT MONITOR CLOCK TIME < MAX HARDWARE MC TIME) THEN SET MONITOR CLOCK TO LEAST SIGNIFICANT WORD OF CURRENT TIME CLEAR CURRENT MC TIME ELSE SET MONITOR CLOCK TO MAX HARDWARE MC TIME DECREMENT HIGH ORDER WORD OF CURRENT TIME ENDIF ENDIF RESTORE EXECUTIVE REGISTERS RETURN THROUGH LINKAGE ESTABLISHED BY EXIMPITL	TO EXEC COMMON: EXAFMON - MONITOR CLOCK SERVICE FLAG EXAVNXTM - NEXT MONITOR CHECK TIME EXAIMCLK - STARTING ADDRESS OF MONITOR CLOCK QUEUE
	GLOBAL CALLS, EXTIMAIN LOCAL CALLS, EXIMPITL	

INSTRUCTOR NOTES

THE PRIMARY USE OF THE ORIGINAL IBM TECHNIQUE WAS AIMED AT IMPROVING THE MAINTAINABILITY OF COMPLEX SYSTEMS BY IMPROVING THE UNDERSTANDABILITY OF THEIR DOCUMENTATION. THIS TECHNIQUE DOES USE HIERARCHICAL STRUCTURING AND ENCOURAGES MODULARITY BUT DOESN'T PROVIDE ANY ADDITIONAL MODULARIZATION CRITERIA - IT CAN AND HAS BEEN USED WITH OTHER DESIGN METRICS (COHESION AND BINDING) AS THE MODULARIZATION CRITERIA. ONE OF THE BIGGEST AIDS TO UNDERSTANDABILITY IS THE USE OF GRAPHICS - THIS ALSO MAKES IT COSTLY AND IS PROBABLY WHY IT HAS NOT BECOME WIDESPREAD IN ITS ORIGINAL FORM.

HIPO

PRINCIPLES

- STRUCTURING, MODULARITY
 - VTOC PROVIDES MECHANISM
- ABSTRACTION
 - GRAPHICS AND VTOC SUPPORT THIS

GOALS

- UNDERSTANDABILITY, MAINTAINABILITY
 - MAJOR EMPHASIS IS ON PRODUCING UNDERSTANDABLE DOCUMENTATION FOR MAINTENANCE
 - TRACEABILITY
 - MULTIPLE LEVELS OF DESCRIPTION
- PROVIDE SOME DEGREE OF ASSISTANCE

VG 744.1

11-16

INSTRUCTOR NOTES

NSSF IS A WAY OF DESCRIBING THE ALGORITHMIC STRUCTURE OF A PROGRAM. IT WORKS WELL ONLY
ON STRUCTURED PROGRAMMING.

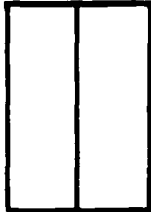
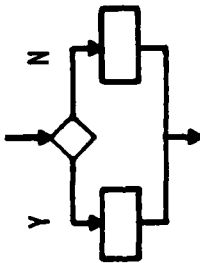
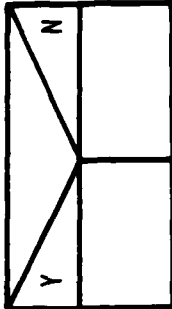
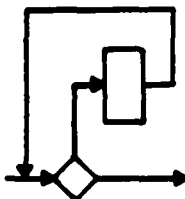
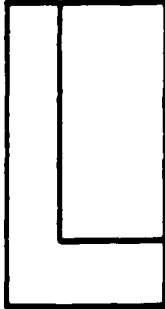
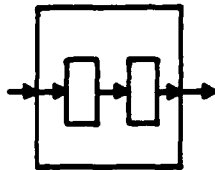
NSSF

- NASSI-SCHNEIDERMAN STRUCTURED FLOWCHARTS
 - DOCUMENT THE PROCEDURAL DETAILS OF SOFTWARE WITH IMPROVED FLOWCHART GRAPHICS
- BASED ON THE ASSUMPTION THAT FLOW CHARTING GRAPHICS CAN BE SIMPLIFIED TO REPRESENT STRUCTURED PROGRAMMING CONSTRUCTS
- NSSF GRAPHICS HELP DESIGNERS
 - REPRESENT PROGRAM STRUCTURES
 - IDENTIFY COMPLEXITY IN A MODULE

INSTRUCTOR NOTES

- COMPARE NSSF TO CONVENTIONAL FLOWCHART GRAPHICS.
- BE CAREFUL OF THE HIERARCHY EXAMPLE: THINK OF IT AS LOOKING "INSIDE" THE BIG BOX.

NSSF

STRUCTURE	FLOWCHART GRAPHICS	NSSF GRAPHICS
SEQUENCE		
SELECTION		
REPETITION		
HIERARCHY		

INSTRUCTOR NOTES

NSSF IS VERY USEFUL IN TEACHING STRUCTURED PROGRAMMING AND FOR DOCUMENTATION. IT IS EXPENSIVE TO PRODUCE, THUS AN AUTOMATED TOOL IS NEEDED TO EXTRACT THE NSSF FROM THE CODE. WITHOUT SUCH A TOOL, IT WILL BE DIFFICULT TO KEEP THEM UP TO DATE.

NSSF SUMMARY

- IMPROVES OLD FLOWCHART GRAPHICS,
- IS GOOD FOR MODULE-SIZED SOFTWARE,
- CAN IDENTIFY COMPLEXITY IN A MODULE.

REMEMBER:

- THERE ARE SEVERAL OTHER STRUCTURED ALTERNATIVES TO FLOWCHARTS.

INSTRUCTOR NOTES

AGAIN THIS IS ONLY A TOOL NOT A TECHNIQUE OR METHODOLOGY. NSSF IS PROBABLY ONE OF THE MOST UNDERSTANDABLE TOOLS FOR REPRESENTING FLOW OF CONTROL (AT A DETAILED LEVEL) AVAILABLE.

NSSF

PRINCIPLES

- STRUCTURING
 - AT A LOW LEVEL, PROGRAM STRUCTURING ONLY
- UNIFORMITY, FORMALISM
 - THE GRAPHICAL CONSTRUCTS

GOALS

- UNDERSTANDABILITY
 - REPRESENTS PROGRAM FLOW OF CONTROL GRAPHICALLY

INSTRUCTOR NOTES

THEME: IMPLEMENTATION IS MORE THAN JUST CODING!

PURPOSE: TO IDENTIFY THE KEY ISSUES ONE MUST CONSIDER DURING THE IMPLEMENTATION
PHASE OF THE LIFE CYCLE.

REFERENCE: DOD-STD-SDS

VG 744.1

12-1

SECTION 12

IMPLEMENTATION OVERVIEW

VG 744.1

INSTRUCTOR NOTES

THE IMPLEMENTATION PHASE SHOULD CREATE A SYSTEM WHICH IMPLEMENTS THE DESIGN CORRECTLY AND MEETS THE RESOURCE, ACCURACY, AND PERFORMANCE CONSTRAINTS IN THE SPECIFICATION. IMPLEMENTATION TRANSLATES DESIGN INTO THE LANGUAGE OF THE TARGET COMPUTER, THE DATABASE, AND OTHER PRODUCTS NEEDED TO CREATE THE OPERATIONAL SYSTEM.

VG 744.1

12-11

IMPLEMENTATION PHASE

- THE IMPLEMENTATION PHASE CONSISTS OF
 - CODING (I.E. TRADITIONAL PROGRAMMING)
 - UNIT TESTING
 - SOFTWARE INTEGRATION AND TEST
 - SYSTEM TESTING
- WHEN OTHERS TALK OF SOFTWARE DEVELOPMENT THEY NORMALLY MEAN IMPLEMENTATION
 - CURRENTLY WE SPEND A LOT OF TIME THRASHING IN THIS PHASE
- IF ANALYSIS AND DESIGN ARE DONE PROPERLY, IMPLEMENTATION IS A WELL DEFINED, EASY PROCESS

INSTRUCTOR NOTES

EACH OF THESE TECHNIQUES CAN HELP CREATE CLEAR, UNDERSTANDABLE PROGRAMS.

VG 744.1

12-21

IMPLEMENTATION SCOPE

- THE IMPLEMENTATION PHASE CONCERNS ITSELF WITH ISSUES LIKE ...

- STEP-WISE PROGRAM DECOMPOSITION

- PROGRAM FAMILIES

- DATA ABSTRACTIONS AND TYPES

- DATA STRUCTURES

- FUNDAMENTAL ALGORITHMS

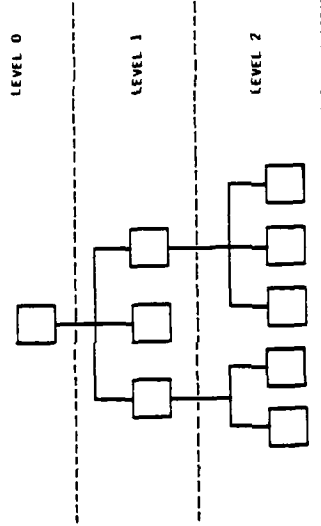
- ACCEPTABILITY VS. CORRECTNESS

INSTRUCTOR NOTES

HERE, THIS VISUAL TABLE OF CONTENTS DEPICTS THE SUBPROGRAMS CREATED DUE TO STEP-WISE REFINEMENT. WE'VE ALREADY SEEN THIS CONCEPT IN THE DESIGN PHASE.

IMPLEMENTATION ISSUES
(STEP-WISE REFINEMENT)

- STEP-WISE PROGRAM DECOMPOSITION (REFINEMENT) STATED THAT PROGRAMS SHOULD BE WRITTEN IN LEVELS; THE HIGHEST LEVEL CALLING ONE OR MORE SUBPROGRAMS AT THE NEXT LOWEST LEVEL:



- EARLY PROGRAMMING WAS CONTENT TO JUST WRITE A CALL STATEMENT: THAT ALONE WAS CONSIDERED THINKING IN STEP-WISE REFINEMENT TERMS.
- TODAY, STEP-WISE REFINEMENT IS USED PRIMARILY IN THE DESIGN PHASE, WHERE THE TECHNIQUE SHAPES THE SYSTEM'S ARCHITECTURE.

AD-A165 122

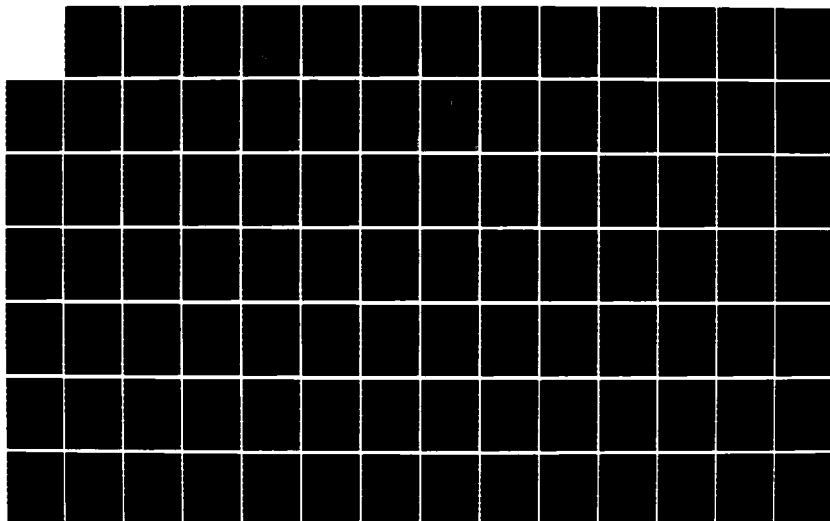
ADA (TRADEMARK) TRAINING CURRICULUM: INTRODUCTION TO
SOFTWARE ENGINEERING M102 TEACHER'S GUIDE(U) SOFTECH
INC WALTHAM MA 1986 DRA807-83-C-K506

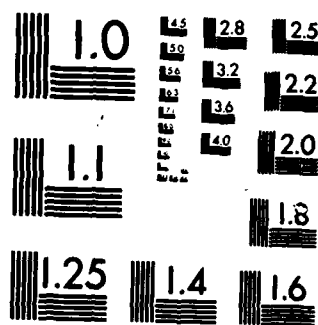
3/6

UNCLASSIFIED

F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1063-A

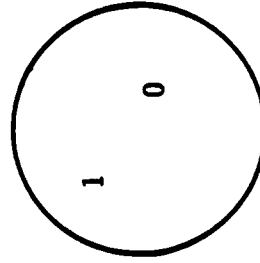
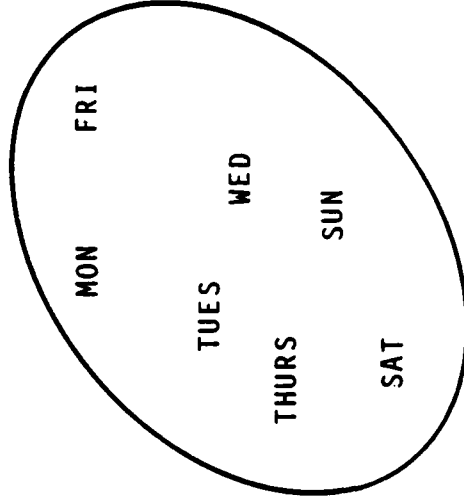
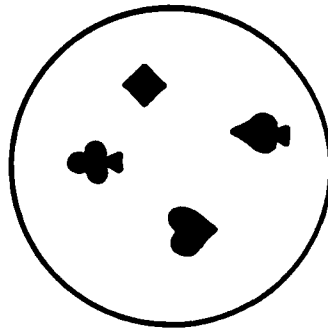
INSTRUCTOR NOTES

HERE, THE NOTATION 1.1 MEANS VERSION 1 PROGRAM 1. THE NOTATION "2" MEANS THIS IS A NEW PROGRAM, SPAWNED FROM ITS ANCESTOR (PROGRAM 1) BY THE CHANGING OF A DESIGN DECISION.

- PROGRAM FAMILIES WORK AT THE MODULE LEVEL. IT DESCRIBES AMOUNT AND TYPES OF IMPACTS WHICH ARE ON THE STRUCTURE OF THE PROGRAM.
- STRUCTURED PROGRAMMING AND PROGRAM FAMILIES ARE NEITHER EQUIVALENT OR CONTRADICTORY. BOTH USE STEP-WISE REFINEMENT TO ENCOURAGE ONE TO MAKE DESIGN DECISIONS EARLY ON.

INSTRUCTOR NOTES

A PICTURE OF THESE TYPES COULD BE:



BOOLEAN

IMPLEMENTATION ISSUES

(DATA ABSTRACTIONS AND TYPES)

- DATA ABSTRACTIONS AND TYPES CATEGORIZE PROGRAM VARIABLES AND THEIR VALUES INTO COLLECTIONS HAVING STRONGLY-RELATED CHARACTERISTICS:

TYPE SUIT = (CLUB, DIAMOND, HEART, SPADE);

TYPE DAY = (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
SATURDAY, SUNDAY);

TYPE BOOLEAN = (TRUE, FALSE);

INSTRUCTOR NOTES

DATA STRUCTURES IMPLY RULES ON HOW TO ACCESS THE DATA THEY CONTAIN.

VG 744.1

12-61

IMPLEMENTATION ISSUES

(DATA STRUCTURES)

- DATA STRUCTURES CONCERN THEMSELVES WITH THE ORGANIZATION OF DATA IN WAYS THAT COMPUTERS CAN PROCESS. FOR EXAMPLE ...

- SCALARS - SINGLE ITEMS (ATOMS) HAVING ONLY A SINGLE VALUE
AT ANY GIVEN TIME

- RECORDS - A LINEAR COLLECTION OF SCALARS OF DIFFERENT TYPE

- ARRAYS - A LINEAR COLLECTION OF SCALARS OF THE SAME TYPE

- STACK - A DYNAMIC ARRAY WHERE ELEMENTS ARE TAKEN OFF OF, AND
PUT ON AT, THE SAME END

- QUEUE - A DYNAMIC ARRAY WHERE ELEMENTS ARE PUT ON AT ONE
END AND TAKEN OFF THE OTHER END

INSTRUCTOR NOTES

SEE THE KNUTH SERIES FOR MORE EXAMPLES.

VG 744.1

12-71

IMPLEMENTATION ISSUES

(FUNDAMENTAL ALGORITHMS)

- FUNDAMENTAL ALGORITHMS CONCERN THEMSELVES WITH STANDARD WAYS OF CALCULATING NUMBERS AND MANIPULATING DATA STRUCTURES. SOME OF THEM ARE ...

- FIBONACCI SEQUENCE GENERATION

- FACTORIAL COMPUTATION

- MATRIX MULTIPLICATION

- LINKED-LIST ALLOCATION

- TREE MANIPULATION

- MONITORS

- INSERTION SORT

- QUICK SORT

INSTRUCTOR NOTES

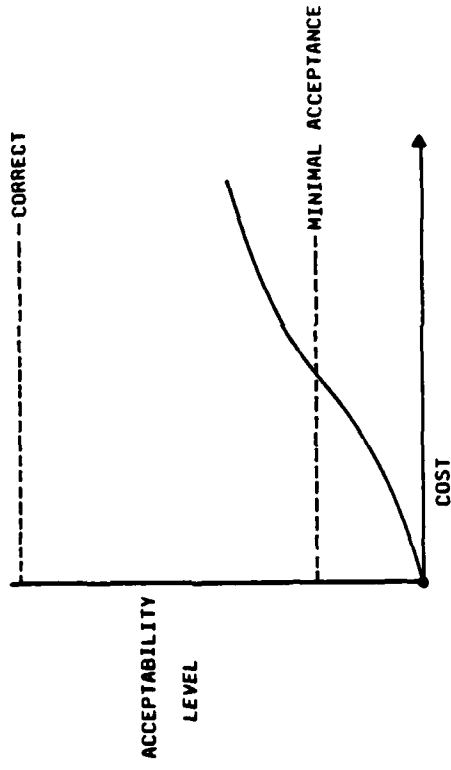
IT COSTS A GREAT DEAL TO GET MORE THAN A MINIMAL LEVEL OF ACCEPTABILITY. ATTAINING
CORRECTNESS IS IMPOSSIBLE THESE DAYS.

VG 744.1

12-81

ACCEPTABILITY VS. CORRECTNESS

- THE ISSUE OF ACCEPTABILITY STATED THAT, SINCE CORRECTNESS COULD NOT BE ATTAINED OF SOFTWARE, TESTING UP TO A LEVEL OF ACCEPTABILITY WAS DEEMED AN ETHICAL PROGRAMMING PRACTICE. THE SAME IS TRUE TODAY.



- THIS IS WHY TESTING, V&V AND QUALITY ASSURANCE HAVE EVOLVED INTO DISCIPLINES OF THEIR OWN -- TO MEET THE NEED OF VALIDATING PROGRAMS TO THE HIGHEST DEGREE OF ACCEPTABILITY POSSIBLE.

INSTRUCTOR NOTES

THIS IS SDS'S REQUIREMENTS FOR IMPLEMENTATION. NOTE HOW THEY RELATE TO THE PREVIOUS TOPICS.

VG 744.1

12-91

DOD-STD-SDS VIEW OF IMPLEMENTATION

- SDS IMPLEMENTATION ACTIVITIES

- CODE UNITS IN A SYSTEMATIC, TOP-DOWN HIERARCHICAL SEQUENCE
- MUST CODE UNITS IN ACCORDANCE WITH CODING STANDARDS
- MAINTAIN UNIT DEVELOPMENT FOLDERS ON ALL UNITS
- TEST EACH UNIT
- DEVELOP INTEGRATION TEST PROCEDURES
- DEVELOP PROCEDURES FOR FORMAL (ACCEPTANCE) TESTING
- INTEGRATE UNITS IN A SEQUENCE BY WHICH TOP-LEVEL SOFTWARE ARE INTEGRATED BEFORE LOWER-LEVEL
- TEST INTEGRATED ITEM IN ACCORDANCE WITH SOFTWARE TEST PLAN

INSTRUCTOR NOTES

NOTICE THE FORMAL REVIEW CYCLE.

VG 744.1

12-101

DOD-STD-SDS VIEW OF IMPLEMENTATION

- SDS IMPLEMENTATION PRODUCTS
 - SOURCE AND OBJECT FOR EACH UNIT
 - SOFTWARE TEST PROCEDURES (FORMAL TESTS)
 - UNIT DEVELOPMENT FOLDERS
 - TEST REPORTS
 - SOFTWARE PRODUCT SPECIFICATION
- SDS IMPLEMENTATION REVIEWS
 - REVIEW OF SOURCE CODE, TEST PROCEDURES AND TEST RESULTS
 - FUNCTIONAL CONFIGURATION AUDIT
 - DEMONSTRATES THAT SOFTWARE WAS TESTED AND MEETS ALL REQUIREMENTS IN SOFTWARE SPECIFICATIONS
 - PHYSICAL CONFIGURATION AUDIT
 - DEMONSTRATES THAT SOFTWARE PRODUCT SPECIFICATION IS COMPLETE

INSTRUCTOR NOTES

ASK THE CLASS HOW THEY FEEL ABOUT THE CODING STANDARDS. NOTE, STRUCTURED PROGRAMMING IS
COMING IN THE NEXT SECTION.

DOD-STD-SDS MINIMUM LEVEL CODING STANDARDS

- ALL CODE SHALL BE WRITTEN IN HIGHER LEVEL LANGUAGE
 - WAIVER REQUIRED FOR USE OF ASSEMBLY LANGUAGE
 - EVEN IF YOU GET A WAIVER THE OTHER STANDARDS MUST BE MET
- ONLY FIVE CONTROL CONSTRUCTS SHALL BE USED
 - SEQUENCE
 - IF-THEN-ELSE
 - DO-WHILE
 - DO-UNTIL
 - CASE
- SOURCE CODE FOR EACH UNIT SHALL NOT EXCEED ...
 - ON THE AVERAGE, 100 STATEMENTS
 - AT MOST, 200 STATEMENTS

INSTRUCTOR NOTES

ASK THE CLASS "DO YOU THINK THESE STANDARDS ARE NECESSARY OR USABLE?"

VG 744.1

12-121

CODING STANDARDS

(CONTINUED)

- UNITS SHALL EXHIBIT THE FOLLOWING CHARACTERISTICS:
 - UNITS SHALL NOT SHARE TEMPORARY STORAGE LOCATIONS
 - A UNIT SHALL PERFORM A SINGLE FUNCTION
 - CODE SHALL NOT BE MODIFIED DURING EXECUTION
 - ALL UNIT SOURCE CODE SHALL INCLUDE A PROLOGUE, DECLARATIVE STATEMENTS, THEN EXECUTABLE STATEMENTS OR COMMENTS
 - EXCEPT FOR ERROR EXITS, UNITS SHALL HAVE A SINGLE ENTRY AND A SINGLE EXIT
- USE MEANINGFUL NAMES AND MAKE ALL PARAMETERS SYMBOLIC
- AVOID MIXED MODE OPERATIONS
- USE PARAGRAPHING, BLOCKING, AND INDENTING TO ENHANCE READABILITY

INSTRUCTOR NOTES

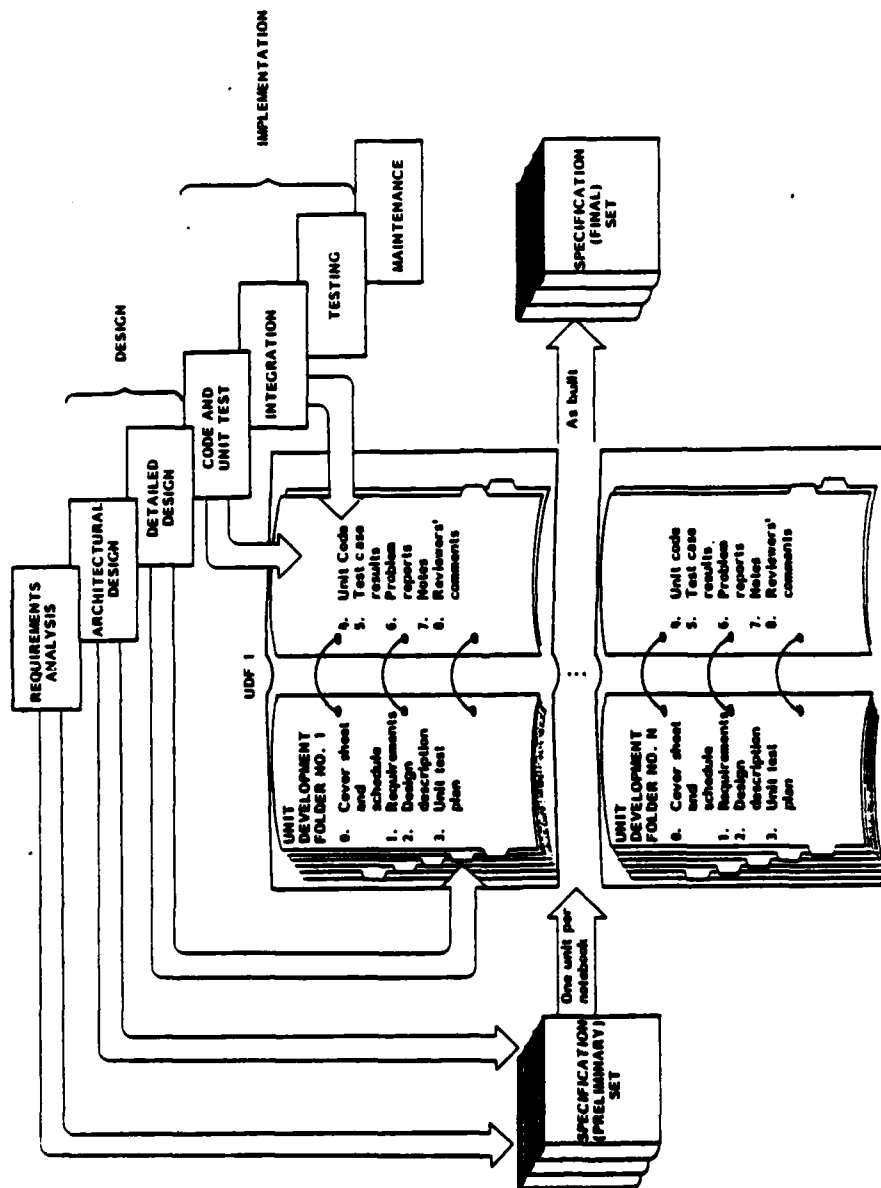
FOLDERS ARE A CONVENIENT WAY OF KEEPING TRACK OF RELATED INFORMATION.

VG 744.1

12-131



UNIT DEVELOPMENT FOLDERS



- PROVIDE A UNIFORM AND VISIBLE COLLECTION POINT FOR ALL UNIT INFORMATION

SOURCE: BOEHM, SOFTWARE ENGINEERING ECONOMICS, 1983

VG 744.1

12-13

INSTRUCTOR NOTES

THEME: TO RELATE THE EARLY 70'S CONCEPTS OF STRUCTURED PROGRAMMING TO MODERN

REQUIREMENTS AND Ada.

PURPOSE: TO PROVIDE A LIMITED VIEW INTO STRUCTURED PROGRAMMING.

REFERENCE: WIRTH, N. "PROGRAM DEVELOPMENT BY STEPWISE REFINEMENT" CACM VOL. 14, NO. 4;
DECEMBER 1971

DAHL, O., DIJKSTRA, E., HOARE, C. "STRUCTURED PROGRAMMING" ACADEMIC PRESS,
LONDON; 1972

SECTION 13

STRUCTURED PROGRAMMING

VG 744.1

INSTRUCTOR NOTES

- GIVE SOME MOTIVATION BY DESCRIBING IN GENERAL WHAT STRUCTURED PROGRAMMING IS ABOUT AND WHY IT'S USEFUL. A FEW DIFFERENT DEFINITIONS MAY BE HELPFUL. FOR EXAMPLE,
- EMPHASIZE THAT STRUCTURED PROGRAMMING IS A METHODOLOGY. USING UNIQUE CONTROL STRUCTURES (AND NO GO TO'S) AND NESTING CODE DO NOT BY THEMSELVES CONSTITUTE STRUCTURED PROGRAMMING. ONE ALSO NEEDS AN ORGANIZATIONAL METHOD TO COLLECTIVELY USE THESE TECHNIQUES BY EVERYONE ON A PROJECT, SUCH AS IBM'S PROGRAMMING TEAMS.
- IT IS LANGUAGE INDEPENDENT, BUT CERTAIN LANGUAGES HELP MORE THAN OTHERS, E.G. Ada. STEP-WISE REFINEMENT IS THE KEY.
- IT IS NOT A RELIGION (OF NOT USING GO TO'S). IT IS MEANT TO ENHANCE READABILITY, RELIABILITY, PROGRAMMER EFFICIENCY, ETC. THE ABSENCE OR RARE OCCURRENCE OF GO TO'S IS NO MORE THAN A SYMPTOM OF STRUCTURED PROGRAMMING.

STRUCTURED PROGRAMMING

MOTIVATION

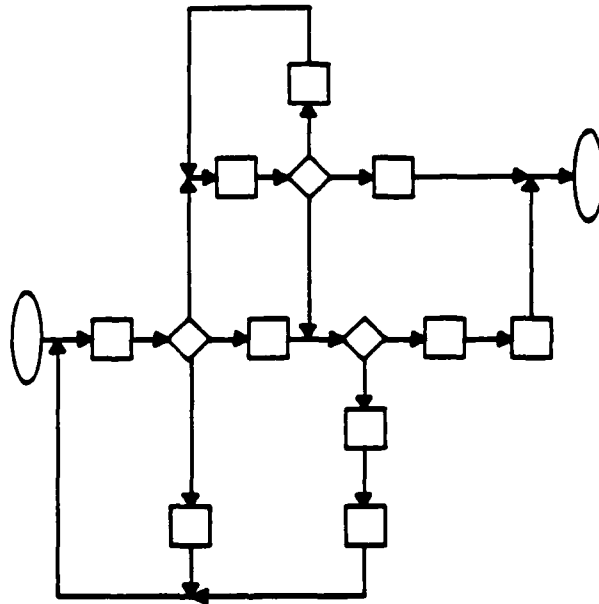
- PROGRAMMING IS A MODELING ACTIVITY AND MODELS MUST BE REVIEWED BY PEOPLE ...
 - TO VERIFY CORRECTNESS OF THE APPROACH
 - TO FIND ERRORS
 - TO SHARE TECHNIQUES
- PROGRAMS MUST BE DESIGNED AND IMPLEMENTED TO BE READ AND UNDERSTOOD BY PEOPLE, NOT JUST COMPUTERS.
- STRUCTURED PROGRAMMING IS A COLLECTION OF TECHNIQUES WHICH EVOLVED; IT CONCERNS ITSELF WITH PROGRAMS PEOPLE MAKE.

INSTRUCTOR NOTES

- SOFTWARE IS ALWAYS READ MORE OFTEN THAN IT IS WRITTEN.
- STRUCTURED PROGRAMMING KEEPS THINGS SIMPLE, THUS ENHANCING THE LIKELIHOOD THAT THINGS WILL BE CORRECT.
- THIS DIAGRAM REQUIRES A GREAT AMOUNT OF WORK TO DETERMINE WHAT CONDITION HOLDS AT A SPECIFIC POINT. MOREOVER, IT'S NOT CLEAR WHERE THAT POINT EXISTS.

MOTIVATION

- EARLY PROGRAMS WERE JUST WRITTEN WITH NO THOUGHT OF HUMAN CONCERNS. A TYPICAL FLOW DIAGRAM OF SUCH A PROGRAM LOOKED LIKE SPAGHETTI ...



- STRUCTURED PROGRAMMING PROVIDES RULES/GUIDELINES FOR "STRUCTURING" PROGRAMS.

INSTRUCTOR NOTES

ALL THE PREVIOUS METHODOLOGIES HAVE AS THEIR ROOTS STRUCTURED PROGRAMMING. THE FOUNDER IS EDGER DIJKSTRA. ALTHOUGH IT STARTED OUT AS A REACTION AGAINST "GO TO" TYPE PROGRAMMING, IT NOW HAS NO STANDARD DEFINITION.

STRUCTURED PROGRAMMING

- A DEFINITION ...

"STRUCTURED PROGRAMMING IS THE ART OF ORGANIZING COMPLEXITY, MASTERING
MULTITUDES, AND ITS RESULTING BASTARD CHAOS AS EFFECTIVELY AS POSSIBLE."

DIJKSTRA

INSTRUCTOR NOTES

ANY PROGRAM CAN BE WRITTEN USING 3 BASIC CONTROL STRUCTURES.

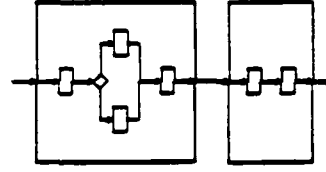
THE PAPER WAS "FLOW DIAGRAMS, TURING MACHINES, AND LANGUAGES WITH ONLY TWO FORMATION RULES," CACM May 1966.

CONTROL STRUCTURING RULES/GUIDELINES

- BOEHM AND JACCOPINNI PROVED THAT ANY LOGIC FLOW COULD BE REDUCED TO THREE SIMPLE RULES ...
- THE THREE BASIC STRUCTURING RULES THEY STATED WERE:



- ANY OF THESE RULES CAN BE NESTED HIERARCHICALLY ...

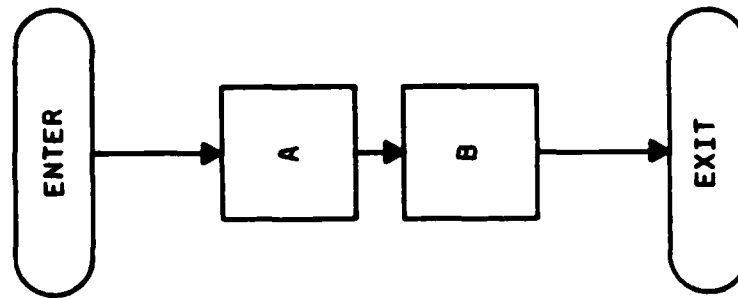


INSTRUCTOR NOTES

REMEMBER, SDS ONLY ALLOWS 5 CONTROL CONSTRUCTS.

DOD-STD-SDS AND STRUCTURE PROGRAMMING

(SEQUENCE)



SEQUENCE CONSTRUCT

SOURCE: DOD-STD-SDS, 1984

VG 744.1

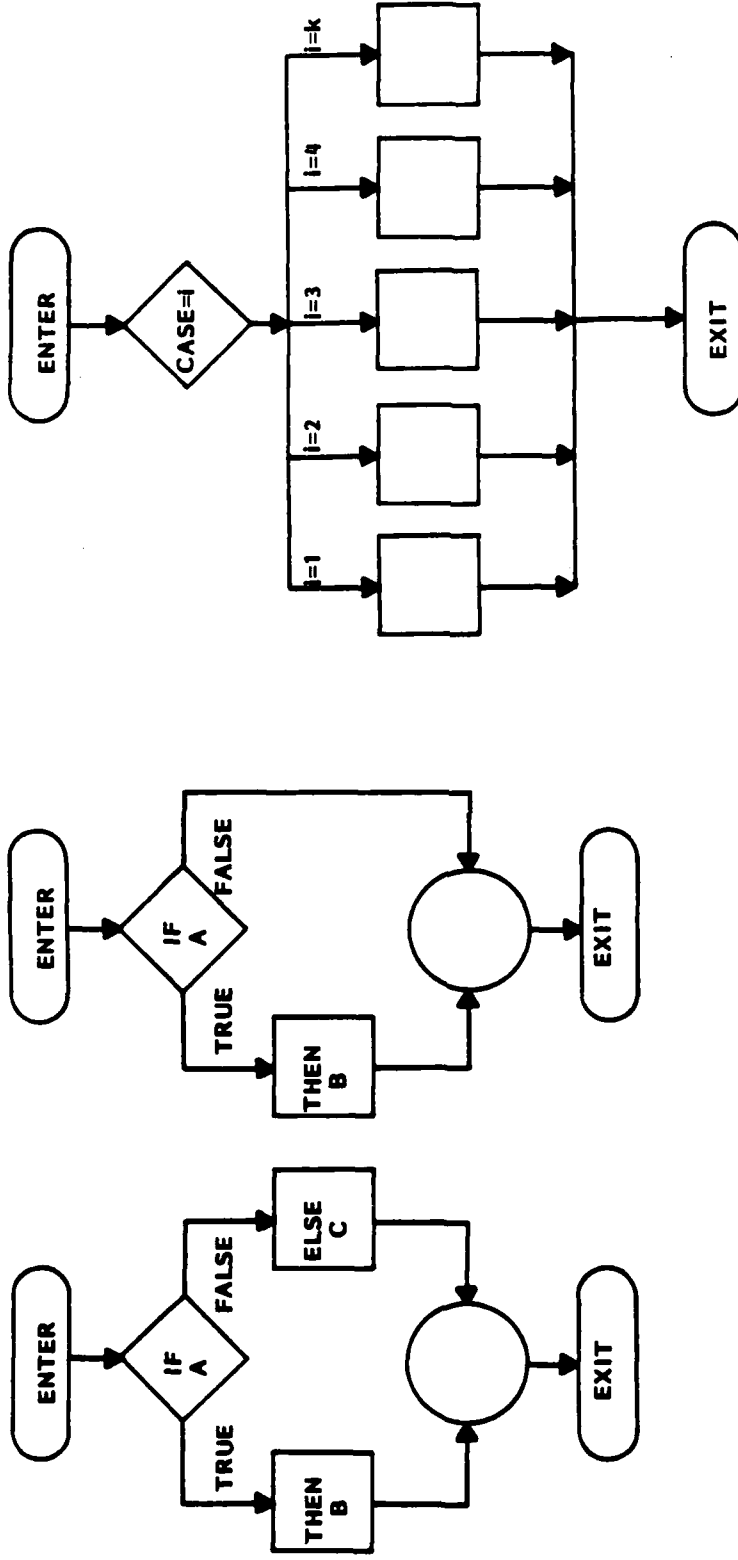
13-5

INSTRUCTOR NOTES

NOTICE ESPECIALLY THE CASE STATEMENT. ALL CONDITIONS MUST BE SPECIFIED. THERE IS NO
DEFAULT CONDITIONS. CERTAIN IMPLEMENTATIONS OF PASCAL COULD NOT MEET THE SDS STANDARD.

DOD-STD-SDS AND STRUCTURED PROGRAMMING

(SELECTION)



IF-THEN-ELSE CONSTRUCT

SOURCE: DOD-STD-SDS, 1984

VG 744.1

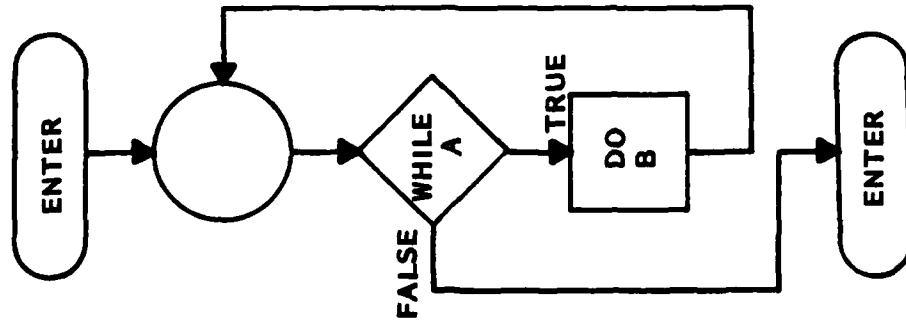
CASE CONSTRUCT

13-6

INSTRUCTOR NOTES

POINT OUT THE DIFFERENCES TO THE CLASS, AND HOW THEY ARE BOTH DIFFERENT TO A WHILE-DO
(TO BE SEEN LATER).

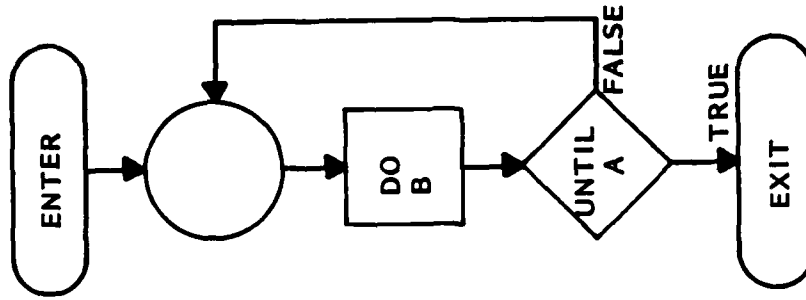
DOD-STD-SDS AND STRUCTURED PROGRAMMING (REPETITION)



DO-WHILE CONSTRUCT

SOURCE: DOD-STD-SDS, 1984

VG 744.1



DO-UNTIL CONSTRUCT

INSTRUCTOR NOTES

WHILE THE THREE FLOW OF CONTROL CONSTRUCTS OF BOHM AND JACOBINE ARE SUFFICIENT, IT HAS BEEN DETERMINED THAT A LIMITED NUMBER OF ADDITIONAL CONSTRUCTS AND UNDERSTANDABILITY. THESE EXAMPLES SHOW WHAT SDS ALLOWS AND WHAT Ada IMPLEMENTS DIRECTLY. NOTE THAT Ada CAN ALSO IMPLEMENT THE Do_Until CONSTRUCT USING A For_Loop WITH AN EXIT STATEMENT AT THE END OF THE LOOP. ALSO NOTE THAT Ada ALLOWS ADDITIONAL CONTROL CONSTRUCTS (I.E., NESTED EXIST STATEMENTS AND EXIT STATEMENTS AT ARBITRARY PLACES IN LOOPS) THAT ARE NOT ALLOWED BY SDS.

CONTROL CONSTRUCTS

MIL-STD-SDS

Ada

•	SEQUENCE	•	SEQUENCE
•	If_Then_Else	•	if-then-else
•	Do_While	•	for loop
•	Do_Until	•	while loop
•	Case	•	case

INSTRUCTOR NOTES

MOST PEOPLE TODAY AGREE THAT STRUCTURED PROGRAMMING IS A SET OF TOOLS AND TECHNIQUES TO SIMPLIFY PROGRAMS AND THEREFORE INCREASE RELIABILITY.

VG 744.1

13-91

SUMMARY

- THE GOAL OF STRUCTURED PROGRAMMING HAS ALWAYS BEEN TO DEVELOP
A STANDARD SET OF RULES FOR STRUCTURING DATA AND ALGORITHMS
INTO PROGRAMS.

INSTRUCTOR NOTES

THEME: SOFTWARE TESTING IS MORE THAN DEBUGGING CODE.

PURPOSE: TO PROVIDE AN OVERVIEW OF THE MAJOR STRATEGIES
INVOLVED IN SOFTWARE TESTING.

REFERENCE: MILLER, E., HOWDEN, W. "TUTORIAL: SOFTWARE TESTING
AND VALIDATION TECHNIQUES" IEEE, EHO 138-8; 1978

SECTION 14

TESTING APPROACHES

VG 744.1

INSTRUCTOR NOTES

- TESTING IS IN THE PROCESS OF EXECUTING A PROGRAM WITH THE INTENT OF FINDING ERRORS.
- DEBUGGING IS WHAT YOU DO AFTER YOU HAVE DISCOVERED AN ERROR -- IDENTIFYING THE EXACT CAUSE OF IT, WITH THE OPTION TO CORRECT IT.
- THE GOALS OF TESTING ARE TO FIND ERRORS - NOT TO NOT FIND ERRORS, AND TO INCREASE OUR CONFIDENCE LEVEL THAT OUR PROGRAM WILL PERFORM CORRECTLY.

TESTING

- OFTEN TESTING AND DEBUGGING ARE CONFUSED
 - TESTING -- SHOWING THAT NO ERRORS ARE PRESENT.
 - DEBUGGING -- IS IDENTIFYING AND CORRECTING AN ERROR.
- TESTING UNCOVERS MANY KINDS OF ERRORS ...
 - LOGIC - MISTAKES WHEN CODING FROM THE DESIGN
 - OVERLOAD - WHEN DATA STRUCTURES ARE FILLED TO CAPACITY
 - TIMING - COORDINATION AMONG PARALLEL PROCESSES
 - THROUGHPUT - PROCESSING SPEED
 - CAPACITY - MEMORY LIMITS
 - RECOVERY - WHAT HAPPENS WHEN THE SYSTEM FAILS
 - SYSTEM SOFTWARE - ASSUMPTIONS ABOUT THE SOFTWARE

INSTRUCTOR NOTES

NOTE HOW WE MOVE FROM FORMAL (CORRECTNESS) TO INFORMAL (REVIEWS) TECHNIQUES.

VG 744.1

14-21

RELATIONSHIP OF TESTING AND OTHER ERROR REMOVAL TECHNIQUES

TECHNIQUE	PHASE IN WHICH ERROR WAS INTRODUCED		
	<u>ANALYSIS</u>	<u>DESIGN</u>	<u>IMPLEMENTATION</u>
• PROGRAM CORRECTNESS			X
• TESTING			
- STRUCTURAL		X	X
- FUNCTIONAL	X		X
• REVIEWS			
- REQUIREMENTS	X		
- DESIGN	X	X	
- CODE		X	X

X - INDICATED TECHNIQUE IS EFFECTIVE IN ERROR REMOVAL

INSTRUCTOR NOTES

POINT OUT THE IMPORTANCE OF USING THE RIGHT TECHNIQUE FOR THE RIGHT JOB.

RELATIONSHIP OF TESTING AND OTHER ERROR REMOVAL TECHNIQUES

EFFECTIVENESS OF TECHNIQUES

IMPLEMENTATION ERROR TYPES	TESTING			
	CODE REVIEWS	PROGRAM CORRECTNESS	STRUCTURAL	FUNCTIONAL
COMPUTATIONAL	MEDIUM	HIGH	HIGH	MEDIUM
LOGIC	MEDIUM	HIGH	HIGH	MEDIUM
INPUT AND OUTPUT	HIGH	LOW	MEDIUM	HIGH
DATA HANDLING	HIGH	MEDIUM	LOW	HIGH
INTERFACE	HIGH	LOW	HIGH	MEDIUM
DATA DEFINITION	MEDIUM	MEDIUM	LOW	MEDIUM
DATABASE	HIGH	LOW	MEDIUM	MEDIUM

INSTRUCTOR NOTES

IF THE TEST RESULT IS CONSISTENT WITH THE EXPECTED RESULT, THE COMPONENT IS DEEMED
CORRECT IN THE LIMITED CONTEXT OF THE TEST.

FOR COMPLEX PROGRAMS, THE LIFE CYCLE MAINTENANCE ASPECTS ALSO BECOME IMPORTANT IN
CHOOSING THE PROPER TESTING STRATEGY. AREAS THAT CHANGE OFTEN MAY USE A DIFFERENT TYPE
OF TESTING THAN AN AREA WHICH DOESN'T.

TESTING

- A TESTED PROGRAM IS ONE WHICH YOU HAVE NOT YET FOUND THE CONDITIONS THAT MAKE IT FAIL.
- TESTING METHODS MUST BE ESTABLISHED ACCORDING TO EACH PROJECT.
 - TYPE OF SYSTEM DIFFERS
 - SIZE OF PROGRAMS VARY
- PRIMARY CATEGORIES OF TESTING
 - UNIT LEVEL
 - INTEGRATION

INSTRUCTOR NOTES

TESTING COSTS GO UP ALMOST EXPONENTIALLY AS PROGRAMS BECOME LARGER. THUS TESTING IS OFTEN NOT DONE TO THE LEVEL REQUIRED. MOREOVER, IN LARGE SYSTEMS INTEGRATION TESTING AS WELL AS UNIT TESTING IS REQUIRED.

TESTING

PROGRAM TYPE	PROGRAM CHARACTERISTICS	TESTING CHARACTERISTICS
SIMPLE PROGRAMS	• LESS THAN 1000 STATEMENTS • ONE PROGRAMMER FOR ABOUT 1 YEAR • NO INTERACTIONS WITH OTHER PROGRAMS	• ANY METHOD WILL DO • EASY TO FIND AND FIX BUGS
MEDIUM PROGRAMS	• LESS THAN 10,000 STATEMENTS • 1-5 PROGRAMMERS FOR LESS THEN 2 YEARS • FEW INTERACTIONS WITH OTHER PROGRAMS	• EFFORT OF 6-12 MONTHS • BRUTE FORCE STILL WORKS, BUT COSTLY • ORGANIZED TESTING CAN BE EFFECTIVE • STILL EASY TO LOCATE BUGS
COMPLEX PROGRAMS	• LESS THAN 100,000 STATEMENTS • 5-20 PROGRAMMERS FOR ABOUT 2-3 YEARS • OFTEN INTERACTS WITH OTHER PROGRAMS • 100-1,000 MODULES • SEVERAL SUBSYSTEMS	• COORDINATION AMONG SEVERAL TEAMS NEEDED • TURNOVER OF PERSONNEL • REQUIRES GOOD RECORDS • RETESTING REQUIRED TO VERIFY REQUIREMENT CHANGES • HARD TO LOCATE ERRORS • ORGANIZED TESTING REQUIRED
NEARLY IMPOSSIBLE PROGRAMS	• OVER 100,000 BUT OFTEN OVER 1,000,000 • MORE THAN 100 PROGRAMMERS, BUT SOMETIMES 1,000 • BETWEEN 5-10 YEAR DEVELOPMENT TIME • 1,000 - 10,000 MODULES • COMPLEX INTERACTIONS WITH OTHER PROGRAMS	• ORGANIZED TESTING MANDATORY • EXTENSIVE TESTING TO MINIMIZE FAILURE • SOMETIMES BEYOND THE STATE-OF-THE-ART IN TESTING

INSTRUCTOR NOTES

UNIT TEST REQUIRES THE TESTING OF THE LOWEST UNIT OF SOFTWARE INDEPENDENTLY OF OTHER PROGRAMS WHICH INTERACT WITH IT.

VG 744.1

14-61

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

UNIT TESTING

- UNIT TESTING IS THE COMPLETE VERIFICATION
OF AN INDIVIDUAL MODULE

- UNIT TESTING REQUIRES US TO CONSIDER
 - TESTING APPROACHES
 - TESTING PRINCIPLES
 - TEST CASE DESIGN
 - TESTING STRATEGY

INSTRUCTOR NOTES

BLACK-BOX TESTING USES THE SPECIFICATION TO DEVELOP TEST CASES AND IS MOST APPROPRIATE
FOR SYSTEM TESTING.

WHITE-BOX TESTING USES DESIGN INFORMATION TO DEVELOP TEST CASES AND IS MOST APPROPRIATE
FOR COMPONENT TESTING.

TESTING APPROACHES

- BLACK-BOX TESTING

- DOES NOT USE DESIGN KNOWLEDGE
- REQUIRES EXTERNAL SPECIFICATION
- SPECIFY INPUTS/OBSERVE OUTPUTS

- WHITE BOX TESTING

- BASED ON INTERNAL DESIGN LOGIC
- REQUIRES UNDERSTANDING (DOCUMENTATION) OF DESIGN
- EMPHASIZES PATH/BRANCH COVERAGE

INSTRUCTOR NOTES

THESE ARE THE MOST COMMON TEST CASES.

EQUIVALENCE PARTITIONING

1. PARTITION EXTERNAL SPECIFICATIONS INTO VALID AND INVALID EQUIVALENCE CLASSES (I.E., VALID CLASS - 1 ITEM COUNT 999; INVALID CLASSES ITEM COUNT 1 AND ITEM COUNT 999)
2. ASSIGN UNIQUE NUMBERS TO EACH CLASS
3. WRITE TEST CASES WITH EACH TEST COVERING AS MANY VALID CLASSES AS POSSIBLE
4. WRITE A SEPARATE TEST USE FOR EACH INVALID CLASS

BOUNDARY VALUE - AUGMENT EQUIVALENT PARTITIONING TO TEST LEGAL BOUNDARY VALUES

COURSE-EFFECT GRAPHING

1. IDENTIFY SPECIFICATION SUBSETS, SUBDIVIDE INTO COURSE AND EFFECTS
2. DRAW BOOLEAN CAUSE - EFFECT GRAPHIC - ANNOTATE WITH CONSTRAINTS
3. CONVERT GRAPH TO DECISION TABLE
4. DEVELOP A TEST CASE FOR EACH COLUMN IN DECISION TABLE

NOTE: THIS PROVIDES A COOKBOOK APPROACH TO COVERING THE FUNCTIONALITY OF A SPECIFICATION

TEST-CASE DESIGN

- BLACK-BOX

- EQUIVALENCE PARTITIONING
- BOUNDARY-VALUE ANALYSIS
- CAUSE - EFFECT GRAPHING
- ERROR GUESSING

- WHITE-BOX

- LOGIC - COVERAGE TESTING

INSTRUCTOR NOTES

INTEGRATION TESTING HELPS ASSURE THAT THE MODULES OR UNITS WHEN PUT TOGETHER DO INDEED
WORK AS A SYSTEM.

INTEGRATION TESTING

- INTEGRATION TESTING IS THE "COMPLETE" VERIFICATION OF THE SET OF MODULES THAT MAKE UP THE SYSTEM
 - BUILDS ON PREVIOUSLY UNIT TESTED MODULES
- INTEGRATION TESTING REQUIRES US TO CONSIDER
 - TESTING APPROACHES
 - INTEGRATION STRATEGIES

INSTRUCTOR NOTES

AT THIS POINT, THE IMPORTANCE OF BOTH UNIT TESTING AND THE INTEGRATION STRATEGY COMES INTO PLAY.

INTEGRATION STRATEGIES

- THE WAY MODULES ARE COMBINED DURING INCREMENTAL TESTING IS CALLED INTEGRATION.

- INTEGRATION TAKES ONE OF TWO FORMS ...

- TOP DOWN

- BOTTOM UP

INSTRUCTOR NOTES

THIS STRATEGY IS VERY NATURAL TO A SYSTEMS DEVELOPMENT EFFORT.

VG 744.1

14-111

TOP-DOWN STRATEGY

- IMPLEMENT THE TOP MODULE OF THE DESIGN FIRST
- SIMULATE THE MODULE'S SUBORDINATES BY STUB MODULES
- GRADUALLY WORK DOWN THE DESIGN, REPLACING STUBS BY REAL MODULES AND INTRODUCING NEW STUBS WHERE NECESSARY

INSTRUCTOR NOTES

TOP-DOWN INTEGRATION IS THE PREFERABLE APPROACH.

VG 744.1

14-121

TOP-DOWN INTEGRATION ADVANTAGES

- IMPORTANT FEEDBACK IS PROVIDED TO THE USER IN THE BEST POSSIBLE WAY.
- THE USER MAY TAKE DELIVERY OF SEVERAL SKELETON VERSIONS OF THE SYSTEM, WHICH WILL SMOOTH HIS TRANSITION FROM OLD SYSTEM TO NEW.
- THE PROJECT SHOULD BE IN BETTER POLITICAL SHAPE IF IT FALLS BEHIND SCHEDULE.
- MAJOR INTERFACES ARE TESTED EARLY AND OFTEN.
- IMPLEMENTERS GET A BOOST FROM SEEING SOMETHING WORKING.
- CODING AND TESTING CAN BEGIN BEFORE DESIGN IS FINISHED.

INSTRUCTOR NOTES

DRIVERS ARE USUALLY HARDER TO WRITE THAN STUBS.

VG 744.1

14-131

BOTTOM-UP INTEGRATION STRATEGY

- BOTTOM-UP INTEGRATION IS, IN GENERAL, THE INVERSE OF TOP-DOWN INTEGRATION ...
- IMPLEMENT A MODULE AT THE BOTTOM OF THE DESIGN.
- SIMULATE THE MODULE'S SUPERORDINATES BY A DRIVER.
- GRADUALLY WORK UP THE DESIGN REPLACING DRIVERS BY REAL MODULES.
- DRIVERS (ALIAS TEST HARNESES) ARE STAND-INS FOR NOT-YET-WRITTEN SUPERORDINATE MODULES.

INSTRUCTOR NOTES

DRIVERS CAN FORCE A MODULE TO OPERATE UNDER UNUSUAL OR ILLEGAL COMBINATIONS OF INPUTS.
A MODULE ONLY OPERATES ON "EXPECTED" DATA. YOU NEED TO KNOW MORE ABOUT THE BEHAVIOR OF
A MODULE WHEN ITS BEEN TESTED BOTTOM UP.

BOTTOM-UP INTEGRATION

PLUSES AND MINUSES

- + STARTS OFF WITH GOOD PARALLEL DEVELOPMENT.
- PARALLEL DEVELOPMENT GETS HARDER TO MANAGE AS TEAMS REACH THE TOP OF THE DESIGN.
- + TESTS PHYSICAL I/O INTERFACES EARLY.
- TEST MAJOR INTERNAL INTERFACES LATE.
- + USEFUL FOR SOME CRITICAL MODULES.
- CODING AND TESTING CANNOT BEGIN BEFORE DESIGN FINISHES.
- SKELETON SYSTEMS DIFFICULT TO PRODUCE.

INSTRUCTOR NOTES

VG 744.1

14-151

SUMMARY

TOP-DOWN VS. BOTTOM-UP

TOP-DOWN

BOTTOM-UP

-	BEST FOR TESTING MAJOR INTERFACES EARLY	-	BEST FOR TESTING PHYSICAL I/O EARLY
-	ASSUMES LOW LEVELS IN DESIGN WILL WORK AS PLANNED	-	ASSUMES LOW LEVELS WILL INTEGRATE AS PLANNED
-	CONTAINS NO SURPRISES	-	VERIFIES LOW-LEVEL ASSUMPTIONS
•	IN THE REAL WORLD, A SINGLE STRATEGY WILL NOT WORK. A PROJECT NEEDS THE RIGHT MIXTURE OF BOTH.		

INSTRUCTOR NOTES

THEME: THERE ARE SEVERAL ASPECTS TO SOFTWARE MANAGEMENT

PURPOSE: PROVIDE AN OVERVIEW OF THE MAJOR SOFTWARE MANAGEMENT DISCIPLINES

REFERENCE: BARRY BOEHM, SOFTWARE ECONOMICS

SECTION 15

SOFTWARE MANAGEMENT

VG 744.1

INSTRUCTOR NOTES

TOPICS TO BE COVERED INCLUDE

- (1) SOFTWARE PLANNING AND TRACKING TECHNIQUES AND TOOLS,
- (2) SOFTWARE COST ESTIMATION - BASED PRIMARILY ON BARRY BOEHM'S WORK AND HIS COCOMO MODEL,
- (3) QUALITY MANAGEMENT INCLUDING VALIDATION AND VERIFICATION TECHNIQUES, AND
- (4) CONFIGURATION MANAGEMENT - WHAT IT IS AND HOW TO GO ABOUT IT.

SOFTWARE MANAGEMENT

- MUST MANAGE THE PROCESS (PROJECT) AND THE PRODUCT (DELIVERABLES)
- ASPECTS TO CONSIDER
 - PLANNING AND TRACKING
 - COST ESTIMATION
 - QUALITY MANAGEMENT
 - CONFIGURATION MANAGEMENT

A vertical strip of film showing a repeating pattern of a textured surface and a dark rectangular block. The texture appears to be a fine, irregular pattern, possibly a close-up of a material or a biological surface. The dark rectangular block is positioned in the center of each frame, creating a rhythmic visual sequence.

15-2i

SOFTWARE PLANNING AND TRACKING

- PLANNING IS THE MOST BASIC FUNCTION OF MANAGEMENT

- DECIDES "WHAT TO DO",

- "HOW TO DO IT",

- "WHEN TO DO IT" AND

- "WHO IS TO DO IT."

- TRACKING REPORTS "HOW IT IS GOING"

- PLANNING INVOLVES

- SETTING OBJECTIVES

- BREAKING THE WORK INTO TASKS

- ESTABLISHING SCHEDULES AND BUDGETS

- ALLOCATING RESOURCES

- SETTING STANDARDS

- SELECTING FUTURE COURSES OF ACTION

INSTRUCTOR NOTES

NOTE THE EMPHASIS ON BOTH THE PROJECT (I.E., MONEY, SCHEDULE, STAFFING, ETC.) AND ON THE PRODUCT (I.E., ATTRIBUTES OF THE SPECIFICATION, DESIGN, AND CODE).

WE NEED TO ESTABLISH QUANTITATIVE ESTIMATES IN BOTH AREAS AND HAVE WAYS OF TRACKING AND COMPARING OUR ACTUALS. ALSO NEED CONTINGENCY PLANS WHEN ESTIMATES AND ACTUALS DIVERGE.

SOFTWARE PLANNING AND TRACKING

(CONTINUED)

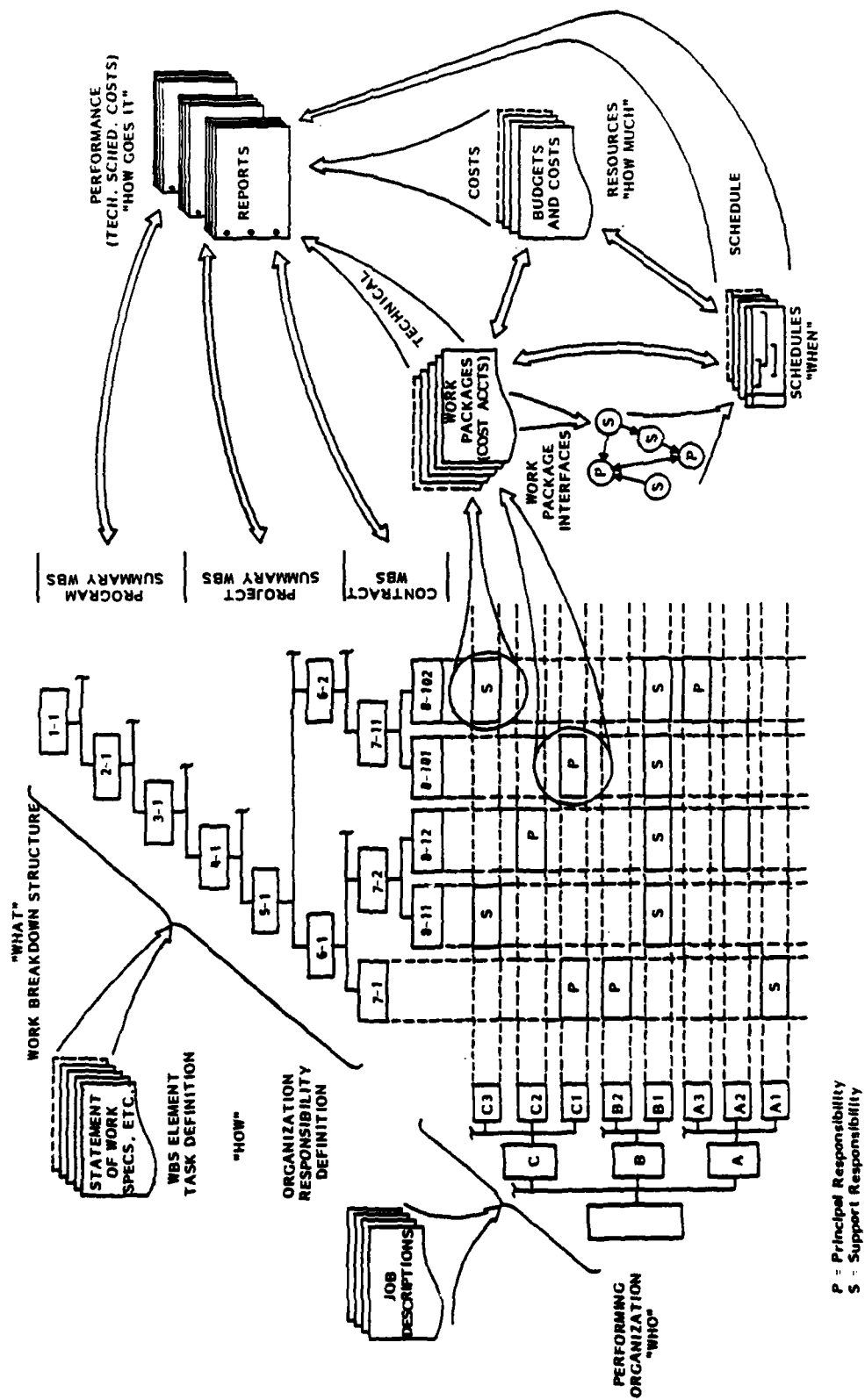
- TWO PRIMARY ASPECTS TO BE TRACKED
 - THE SOFTWARE PRODUCT
 - THE SOFTWARE PROJECT
- PURPOSE
 - MEASURE AND EVALUATE PROGRESS
 - IDENTIFY PROBLEMS EARLY WHILE THERE IS STILL TIME TO TAKE CORRECTIVE ACTION
- KEYS TO SUCCESSFUL TRACKING
 - PERFORM BASE TRACKING ON DELIVERABLES (I.E., SOMETHING THAT CAN BE MEASURED AS COMPLETE OR NOT)
 - ESTABLISH MILESTONES FOR ALL DELIVERABLES
 - HAVE AN ESTIMATE TO COMPARE ACTUALS TO (RELIES ON PLANNING)

INSTRUCTOR NOTES

PLANNING ACTIVITIES HAVE TO WORK WITHIN TYPICAL MATRIX ORGANIZATIONS. THE PROJECT STRUCTURE REFLECTED BY THE WBS IS SHOWN ON TOP. THE PERFORMING ORGANIZATION IS SHOWN ON THE LEFT. THE INTERSECTIONS DEFINE THE SPECIFIC WORK PACKAGES (PEOPLE AND TASKS) THAT CONTRIBUTE TO THE DELIVERED PRODUCT.

SCHEDULES DEFINE "WHEN" THESE WORK PACKAGES GET PERFORMED.

SOFTWARE PLANNING AND TRACKING (STRUCTURING TECHNIQUES)



INSTRUCTOR NOTES

UNFORTUNATELY WE ARE NOT USUALLY THE MASTERS OF OUR MAJOR GOALS AND OBJECTIVES. MANY OF SOFTWARE'S DIFFICULTIES ARE THE RESULT OF EXTERNALLY IMPOSED COST AND SCHEDULE CONSTRAINTS. COST AND SCHEDULE OBJECTIVES ARE ALMOST ALWAYS SET IN FAIRLY CLEAR AND EASILY UNDERSTOOD TERMS. QUALITY OBJECTIVES ARE OFTEN SET IN VAGUE, EASILY MISUNDERSTOOD TERMS - THIS LEADS TO DISAPPOINTED CUSTOMERS.

THE NEXT SEVERAL SLIDES WILL ADDRESS SPECIFIC TECHNIQUES FOR DEFINING SCHEDULES AND IDENTIFYING MILESTONES, MONITORING COSTS AND ASSESSING QUALITY. IN GENERAL WE HAVE TO DO THE BEST JOB WE CAN WITHIN THE EXTERNAL CONSTRAINTS, WITH EMPHASIS ON BEING ABLE TO IDENTIFY AND RECTIFY PROBLEMS AS EARLY AS POSSIBLE.

SETTING OBJECTIVES AND SELECTING

FUTURE COURSES OF ACTION

- OBJECTIVES ARE OFTEN ALREADY SET FOR US
 - COST GOALS
 - SCHEDULE "DROP DEAD DATES"
 - QUALITY STANDARDS

- MANAGEMENT STRATEGIES FOR SETTING OBJECTIVES AND SELECTING COURSES OF ACTION VARY GREATLY AND DEPEND ON:
 - ORGANIZATION CONSTRAINTS
 - EXPERIENCE OF MANAGER
 - NATURE OF THE MARKET PLACE OR INDUSTRY TYPE OR DoD COMPONENT

INSTRUCTOR NOTES

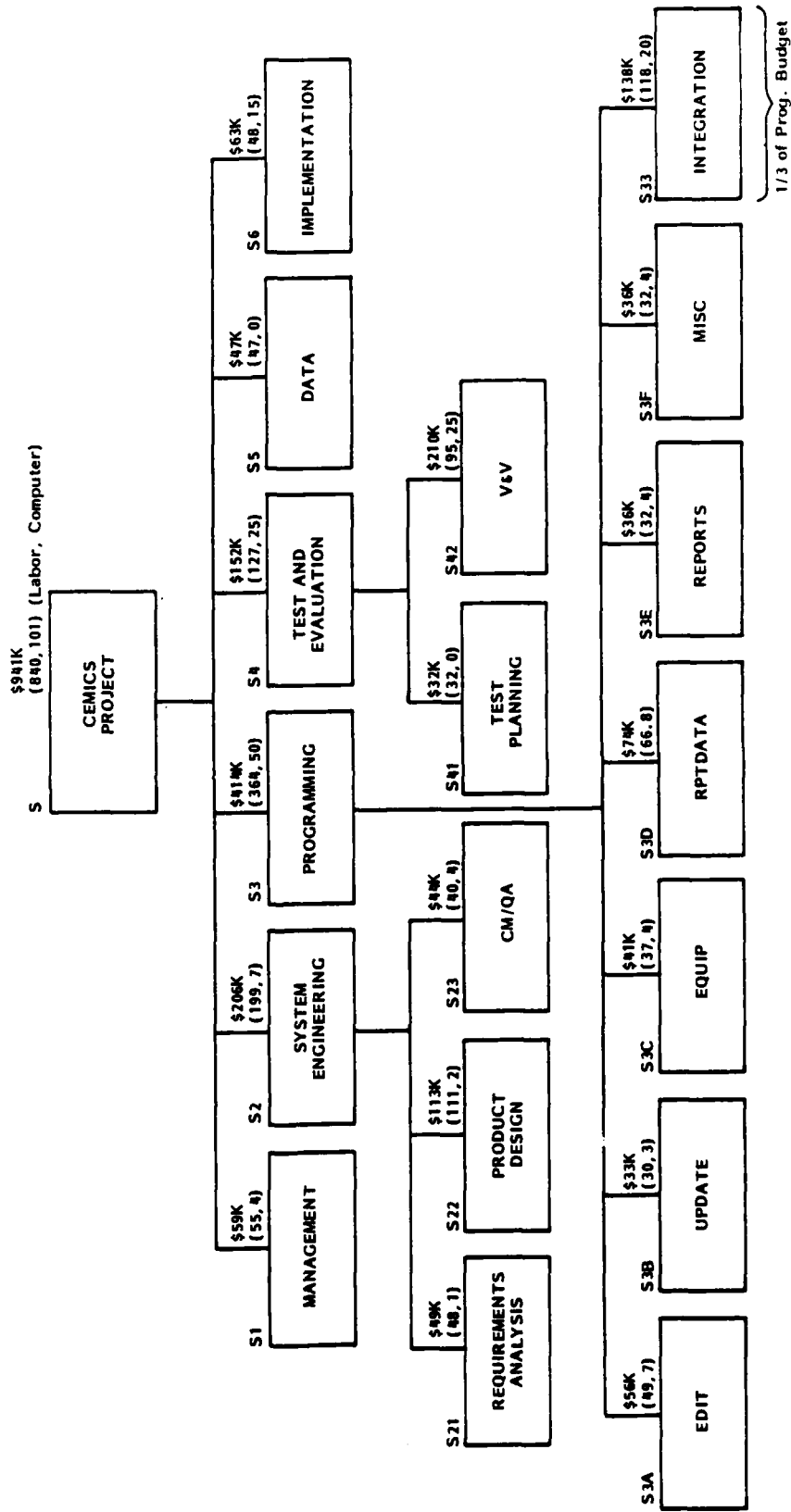
THE WBS REPRESENTS THE TASK WE MUST PERFORM IN A HIERARCHICALLY STRUCTURED FORM. THIS EXAMPLE SHOWS THREE LEVELS, BUT ON LARGE PROJECTS MORE LEVELS ARE OFTEN USED.

MANY OTHER ASPECTS OF THE PROJECT ARE RELATED TO OR MAY BE DRIVEN BY THE WBS. WE MAY BE REQUIRED TO PRODUCE A PLAN FOR EACH FIRST LEVEL ITEM, OR TO TRACK STATUS AS SOME PARTICULAR LEVEL (I.E., THE THIRD).

COMMON PROBLEMS WITH WBSS ARE OMISSIONS (IT WON'T GET DONE) AND REDUNDANCY (TWO SPECIAL GROUPS MAY BE DOING IT).

THE WBS

- CHARACTERISTICS OF THE WORK BREAKDOWN STRUCTURE (WBS)
 - A TREE STRUCTURE THAT REPRESENTS ALL TASKS AND SERVICES NEEDED IN DEVELOPING SOFTWARE
 - PRESENTS TASKING AT SUMMARY AS WELL AS DETAIL LEVEL



INSTRUCTOR NOTES

MILESTONES PROVIDE VISIBILITY INTO A PROJECT FOR THE CUSTOMER THAT IS PROCURING A SYSTEM. MORE MILESTONES (BOTH FORMAL AND INFORMAL) ARE BETTER, BUT MILESTONES IMPOSE ADDITIONAL COSTS. FORMAL MILESTONES REQUIRE ADDITIONAL, FORMAL PAPERWORK BY A COMPANY'S CONTRACTS DEPARTMENT - ADDING TO COSTS. GOOD MILESTONES ARE ASSOCIATED WITH ACTIVITIES THAT ARE INHERENTLY NECESSARY (I.E., PRODUCING A SPECIFICATION FOR A SYSTEM). A BAD MILESTONE IS ONE THAT REQUIRES A LOT OF WORK THAT IS NOT DIRECTLY NEEDED TO PRODUCE THE SYSTEM.

A TENDENCY IN PROJECTS THAT ARE EXPERIENCING PROBLEMS IS TO EITHER ESTABLISH TOO MANY MILESTONES (AND THUS INHIBIT PROGRESS ON THE SYSTEM DEVELOPMENT) OR TO DELETE ALL MILESTONES (SO AS TO NOT "WASTE" ANY MORE RESOURCES). A PROJECT THAT IS IN TROUBLE REQUIRES "BETTER" MILESTONES NOT NECESSARILY MORE OR LESS.

MILESTONES

- FORMAL MILESTONES ARE REQUIRED BY CONTRACT OR AGREEMENT WITH CUSTOMER
 - REQUIREMENTS REVIEW
 - DESIGN REVIEWS
 - CONFIGURATION AUDITS
 - ACCEPTANCE TESTS

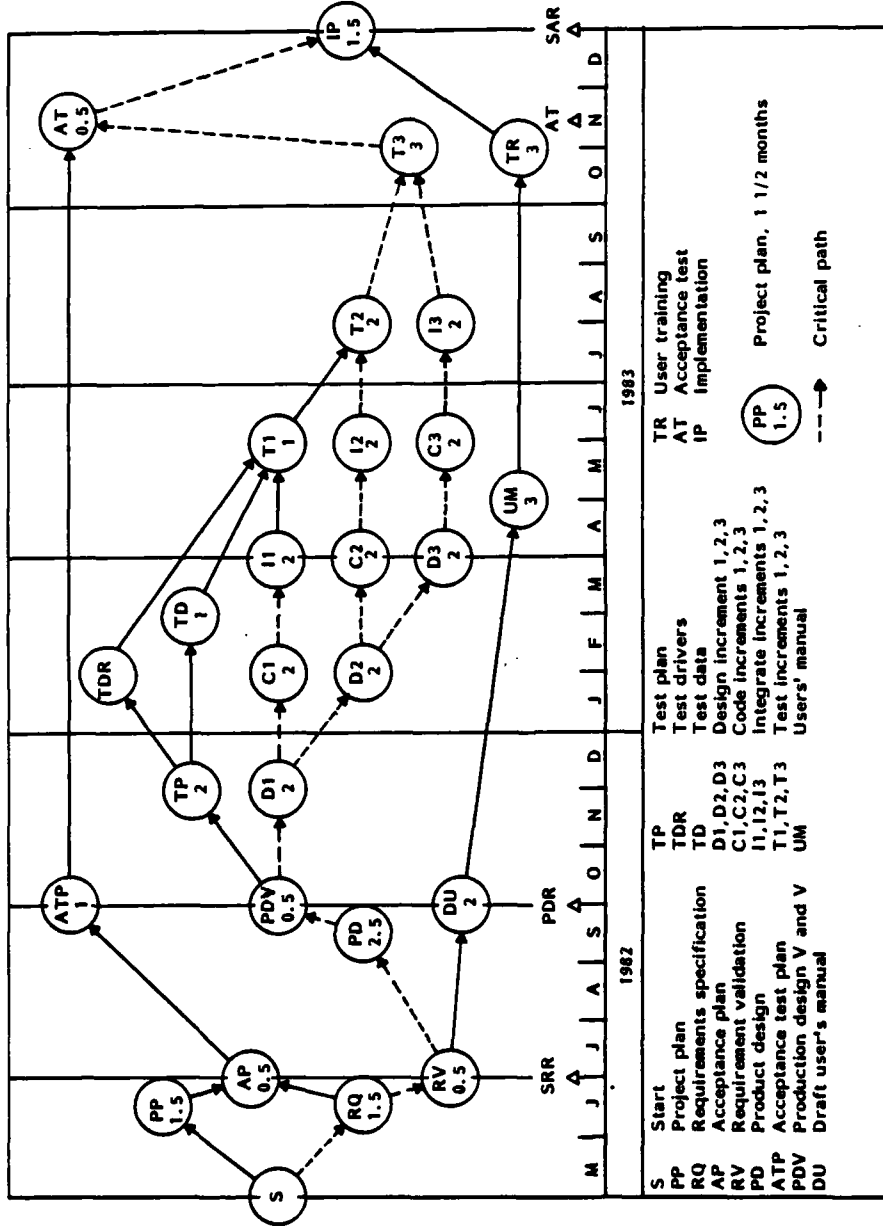
- INFORMAL MILESTONES CONSIST OF MEETINGS AND DEMONSTRATIONS TO PROVIDE VISIBILITY INTO THE PROJECT
 - STATUS REVIEW MEETINGS
 - IN PROCESS REVIEWS
 - IN PROCESS DEMONSTRATIONS

INSTRUCTOR NOTES

PERT CHARTS, SUCH AS THIS EXAMPLE, SHOW THE DEPENDENCIES BETWEEN TASKS AND THE IMPACTS THESE DEPENDENCIES HAVE ON THE TIME LINE FOR THE OVERALL PROJECT.

PERT CHARTS REQUIRE AN ESTIMATE OF THE TIME DURATION FOR EACH TASK (SHOWN AS THE NUMBER INSIDE EACH TASK BUBBLE) AND AN IDENTIFICATION OF THE PRECEDENCE RELATIONSHIPS BETWEEN TASKS (SHOWN AS THE ARROWS BETWEEN TASKS). FROM THIS INFORMATION A PERSON (OR IN THIS CASE A PROGRAM) CAN PRODUCE A PERT CHART. ON THIS CHART THE CRITICAL PATH (THE PATH ON WHICH ANY TASK SLIP WILL LENGTHEN THE OVERALL SCHEDULE) IS SHOWN AS A DASHED LINE.

SCHEDULE (USING PERT/CPM)



• PROVIDES A VIEW OF OVERALL SCHEDULE AND INTER TASK DEPENDENCIES

INSTRUCTOR NOTES

GANTT CHARTS SHOW OVERALL PROJECT SCHEDULES, WITHOUT ANY OF THE INTERDEPENDENCIES SHOWN ON A PERT CHART.

THESE CHARTS SHOW WHAT HAS HAPPENED (TRIANGLES THAT ARE FILLED IN) AND WHAT IS STILL SCHEDULED (OPEN TRIANGLES). THESE TRIANGLES USUALLY REPRESENT MILESTONES OF ONE FORM OR ANOTHER.

[illegible]

PROVIDES A VIEW OF OVERALL SCHEDULE AND COMPLETION STATUS

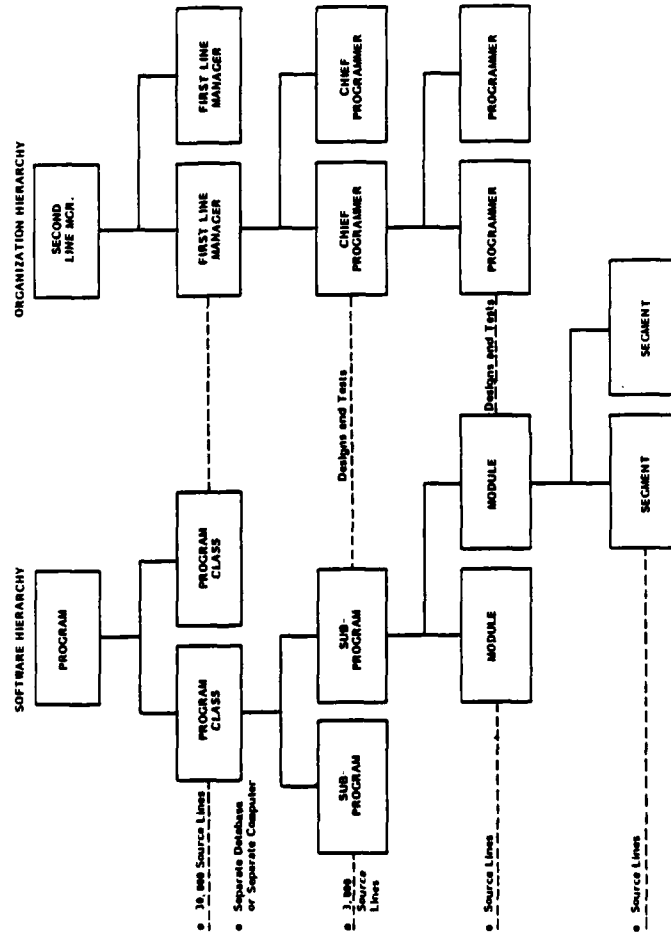
INSTRUCTOR NOTES

NOTE THAT WHAT WE ARE RECOMMENDING HERE IS TO LET THE SOFTWARE STRUCTURE (WHICH MAY ALSO BE THE WBS STRUCTURE) INFLUENCE THE ORGANIZATION STRUCTURE. ALL TO OFTEN WE END UP WITH SOFTWARE STRUCTURES WHICH HAVE BEEN INFLUENCED BY THE EXISTING ORGANIZATIONAL STRUCTURE

- THIS CAN RESULT IN INEFFICIENT SOFTWARE.

ALLOCATION OF MANPOWER

- MAIN CONSIDERATIONS
 - ORGANIZE IN TEAM ORIENTED WAY
 - ASSIGN TASKS TO MINIMIZE DEPENDENCIES
 - ASSIGN A COHESIVE PIECE TO A TEAM
 - DEFINE RESPONSIBILITIES CLEARLY

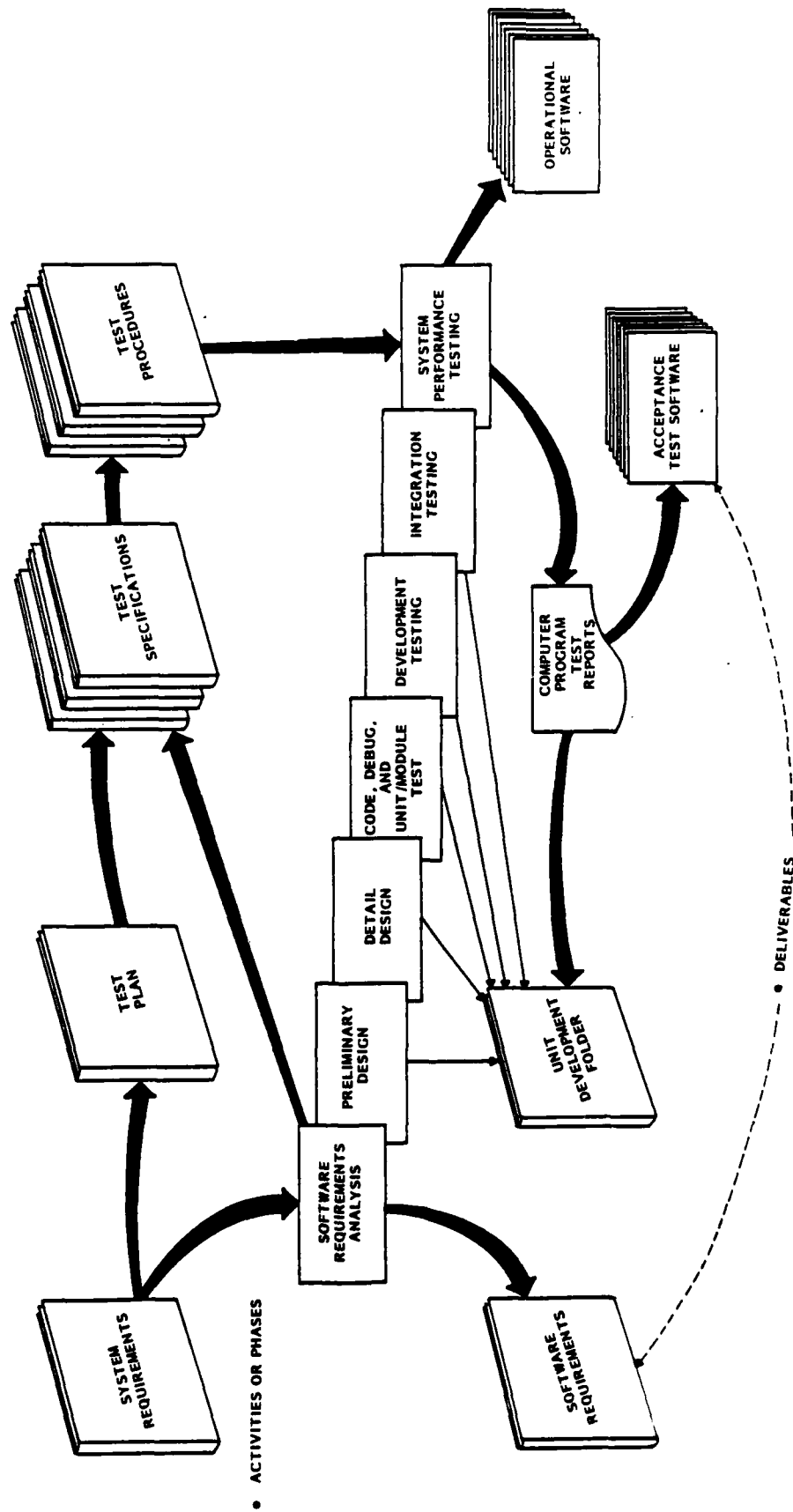


- MANPOWER ALLOCATION IN GENERAL FOLLOWS SOFTWARE STRUCTURE

INSTRUCTOR NOTES

NOTE THAT THIS IS SHOWING GENERIC SOFTWARE DELIVERABLES - IT IS NOT SPECIFICALLY DERIVED FROM MIL-STD-2167 (SDS) OR ANY OTHER STANDARD.

SOFTWARE DEVELOPMENT DELIVERABLES



ND-A165 122

ADA (TRADEMARK) TRAINING CURRICULUM: INTRODUCTION TO
SOFTWARE ENGINEERING M102 TEACHER'S GUIDE(U) SOFTECH
INC WALTHAM MA 1986 DADB07-83-C-K506

6/6

UNCLASSIFIED

F/G 5/9

NL

END

P. P. M. M.

1170



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

UDFs ARE SORT OF A "GRASS ROOTS" CONCEPT THAT ARE KNOWN BY SEVERAL OTHER NAMES (SOFTWARE DEVELOPMENT FILE, MODULE DEVELOPMENT FILE, MODULE DEVELOPMENT FOLDER, AND MANY OTHERS). IT IS REALLY A CONFIGURATION MANAGEMENT SCHEME FOR INTERMEDIATE SOFTWARE DEVELOPMENT WORK PRODUCTS.

IN SOME ORGANIZATIONS THE UDF IS LEVELED OFF FROM DEVELOPER TO INTEGRATION AND TEST WHEN THE "UNIT" HAS PASSED ITS UNIT TEST.

MUCH UDF INFORMATION IS A CANDIDATE FOR KEEPING ON-LINE, SO THAT IT CAN BE PROCESSED BY VARIOUS TOOLS AND CAN BE USED TO GENERATE STATUS REPORTS.

UNIT DEVELOPMENT FOLDER

- A FORMALIZED PROGRAMMER'S NOTEBOOK
 - PROVIDES A UNIFORM AND VISIBLE COLLECTION POINT FOR ALL UNIT DOCUMENTATION AND CODE
 - PROVIDES MANAGEMENT VISIBILITY INTO THE DEVELOPMENT PROCESS
- UDF CONTAINS FOR A UNIT
 - REQUIREMENTS
 - DESIGNS
 - TEST PLAN
 - CODE
 - TEST RESULTS
 - PROBLEM REPORTS
 - NOTES
 - REVIEWERS' COMMENTS
- IN ADDITION A UDF SUMMARIZES THE STATUS OF A UNIT

INSTRUCTOR NOTES

MUCH OF THE CRITICISM OF SOFTWARE FOR ALWAYS EXCEEDING ITS COST ESTIMATES IS BASED ON OUR POOR ABILITY TO PRODUCE AND JUSTIFY RELIABLE ESTIMATES. THIS IS AN AREA OF SOFTWARE ENGINEERING THAT NEEDS SUBSTANTIAL IMPROVEMENT.

TO BE ABLE TO PRODUCE ESTIMATES WE NEED

- (1) COST MODELS - (I.E., THE RELATIONSHIP BETWEEN LINES OF CODE AND COST, BETWEEN LANGUAGE AND COST, BETWEEN EXPERIENCE AND COST, ETC.)
- (2) COST MONITORING PROCEDURES - BECAUSE FUTURE PROJECT COSTS ARE MOST CLOSELY RELATED TO COSTS-TO-DATE.
- (3) COST HISTORY MAINTENANCE - THIS IS HOW WE BUILD OUR MODELS AND OUR PROJECTION ABILITIES - WITHOUT GOOD COST DATA FROM SIMILAR PROJECTS WE HAVE ALMOST NO ABILITY TO MAKE RELIABLE PROJECTIONS.

SOFTWARE COST ESTIMATION

- MOTIVATION

- EVERY PRODUCT MUST BE ENGINEERED TO COST
- COSTS MUST BE PREDICTED

- SOFTWARE COST ESTIMATION IS THE PROCESS OF DETERMINING THE FULL COST OF THE DEVELOPMENT OF SOFTWARE

- SOFTWARE COST ESTIMATION REQUIRES:

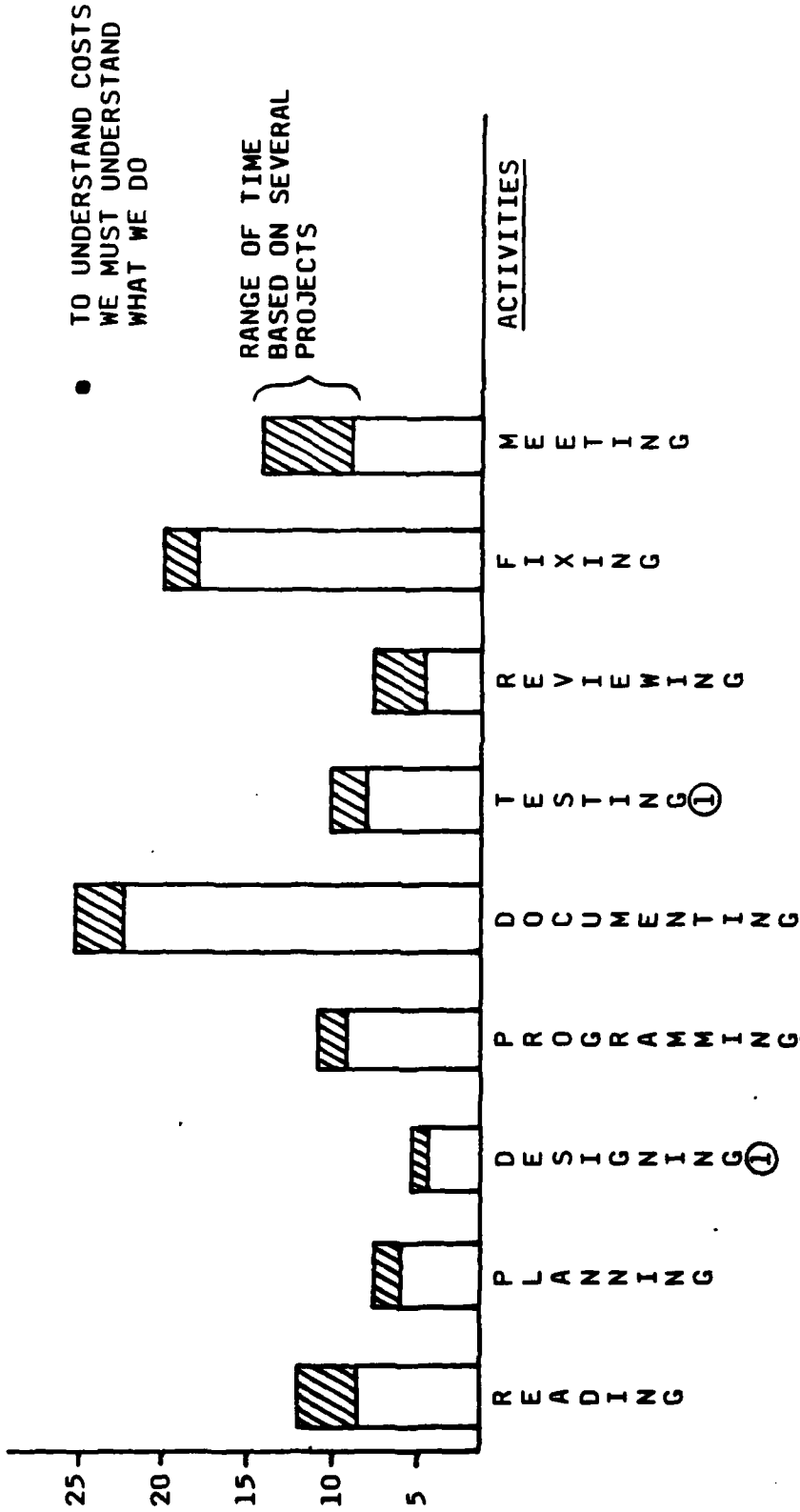
- COST MODELING TECHNIQUES
- COST MONITORING PROCEDURES
- COST HIERARCHY MAINTENANCE

INSTRUCTOR NOTES

THIS GRAPHIC POINTS OUT HOW WE REALLY DON'T HAVE MUCH INSIGHT INTO WHAT SOFTWARE ENGINEERS DO WITH THEIR TIME. WE TALK ABOUT DESIGN, CODE AND TEST AS THEIR PRINCIPLE ACTIVITIES, BUT IN REALITY THIS IS ONLY 25 - 30%.

NOTE THAT JUST DELETING AN EXPENSIVE ACTROCITY (DOCUMENTING) DOESN'T REDUCE COSTS - IT JUST MOVES IT ELSEWHERE (FIXING).

ACTIVITIES OF A SOFTWARE PROJECT



① TRADITIONAL ACTIVITIES INCLUDED IN COST ESTIMATES

• TRADITIONAL ESTIMATES TYPICALLY ONLY ACCOUNT FOR 25% TIME ASSOCIATED WITH A SOFTWARE PROJECT

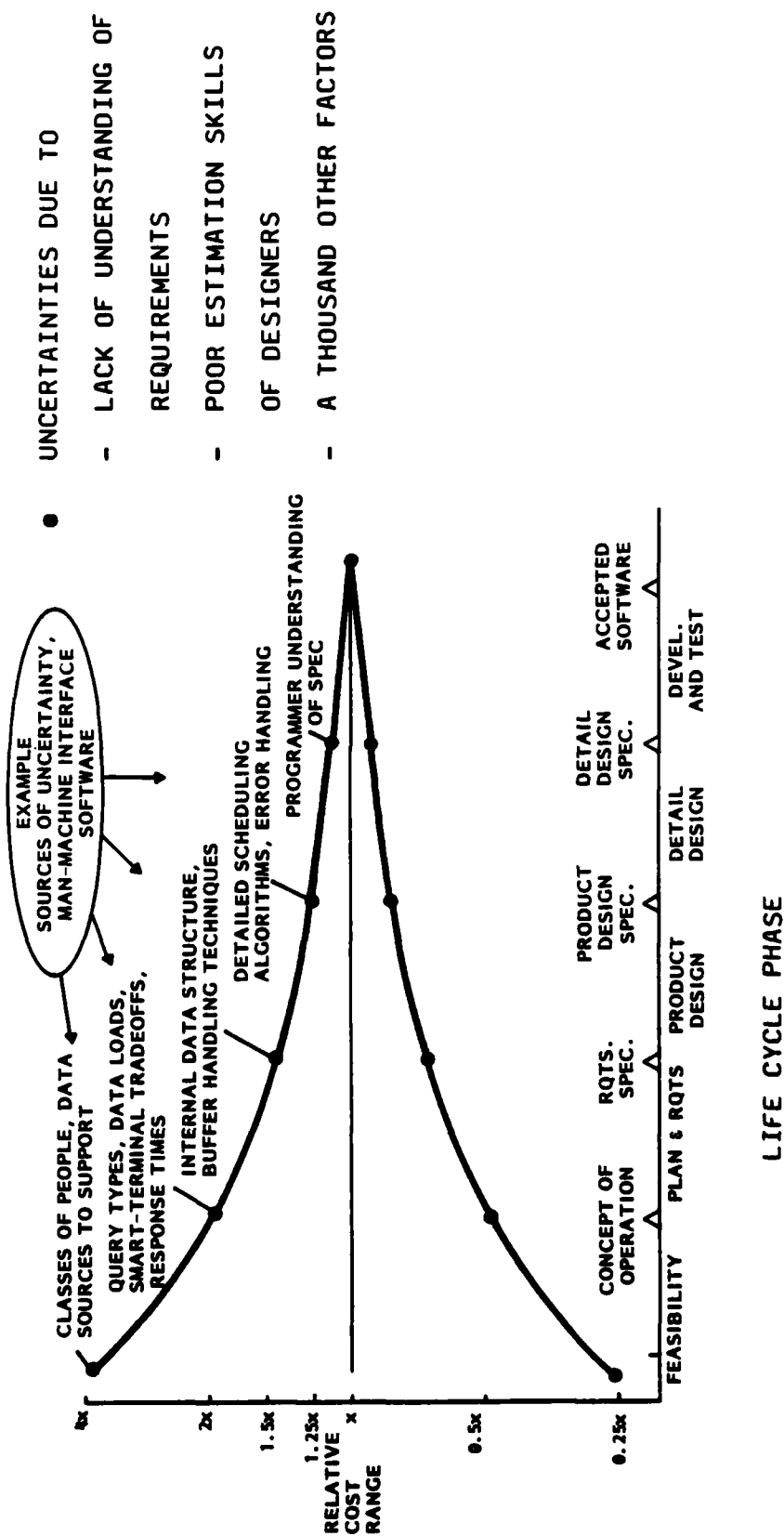
INSTRUCTOR NOTES

THIS DIAGRAM DRAMATICALLY SHOWS THE RELATIVE ACCURACY OF COST ESTIMATE DEPENDING ON THE PHASE OF A PROJECT. DURING EARLY FEASIBILITY PHASES, IT IS NOT UNUSUAL FOR ESTIMATES TO BE HIGH OR LOW BY A FACTOR OF 4 (I.E., ESTIMATING ANYWHERE FROM \$4 MILLION TO \$250,000 FOR WHAT IS ULTIMATELY A \$1 MILLION DOLLAR JOB). EVEN IN THE MIDDLE OF THE ANALYSIS PHASE, AT THE POINT WE WOULD EXPECT A FAIRLY GOOD REQUIREMENTS SPECIFICATION TO BE AVAILABLE WE COULD STILL BE OFF BY A FACTOR OF 2.

VG 744.1

15-151

SOFTWARE COST ESTIMATION EXPECTED ACCURACY VS PHASE



SOURCE: BOEHM, SOFTWARE ENGINEERING ECONOMICS, 1982

VG 744.1

15-15

INSTRUCTOR NOTES

COCOMO - CONSTRUCTIVE COST MODEL WAS DEVELOPED BY BARRY BOEHM AND IS THE SUBJECT OF HIS BOOK ON SOFTWARE ENGINEERING ECONOMICS. COCOMO IS AVAILABLE IN SEVERAL LEVELS OF COMPLEXITY (BASIC, INTERMEDIATE, DETAILED). THE MODEL IS BASED ON EXTENSIVE EMPIRICAL DATA.

LIKE ALL SUCH MODELS CERTAIN PARAMETERS PLAY A VERY SIGNIFICANT ROLE IN THE ACCURACY OF RESULTS - THE MOST IMPORTANT PARAMETER IS THE ESTIMATED NUMBER OF LINES OF CODE - NEXT MOST IMPORTANT ARE CHARACTERISTICS OF THE PRODUCT BEING DEVELOPED (I.E., ITS COMPLEXITY, SIZE AND SPEED CONSTRAINTS, ETC.) AND THE QUALITIES OF THE STAFF AND CHARACTERISTICS PROJECT SUCH AS SCHEDULE AND FACILITIES.

A SOFTWARE COST ESTIMATION TECHNIQUE - COCOMO

- COCOMO IS BASED ON EMPIRICAL DATA
 - OVER 63 MAJOR PROJECTS
 - ACCURATELY MODELS SOFTWARE COSTS TO 20% OF ACTUAL COSTS 80% OF THE TIME
- SOFTWARE COSTS DETERMINATION DEPENDS ON
 - ESTIMATING SOURCE LINES OF CODE
 - CHARACTERIZING THE SOFTWARE PRODUCT
 - CHARACTERIZING THE SOFTWARE PROJECT
 - USING THE COCOMO EQUATIONS

INSTRUCTOR NOTES

STRESS THE IMPORTANCE OF GETTING AN ACCURATE ESTIMATE OF SOURCE LINES OF CODE AND THE DIFFICULTY. WHAT WE NEED IS SOME MODELS TO HELP US EXTRAPOLATE FROM A REQUIREMENTS SPECIFICATION OR A DESIGN DESCRIPTION TO AN ESTIMATE OF SOURCE LINES OF CODE. SUCH MODELS DON'T EXIST.

A SPECIFICATION TECHNIQUE LIKE SCRP (A-7E) OFFERS PROMISE IN THIS AREA BECAUSE OF ITS UNIQUE DEFINITIONS FOR THINGS LIKE FUNCTIONS, MODES, AND DATA ITEMS. IF ALL SPECIFICATIONS WERE WRITTEN THIS WAY WE COULD PROBABLY DEVELOP ESTIMATION TECHNIQUES WHICH DERIVED AN ESTIMATE OF SOURCE LINES OF CODE FROM A COUNT OF FUNCTIONS, MODES, INPUT DATA ITEMS AND OTHER QUANTIFIABLE ASPECTS OF THE SPECIFICATION.

KDSI = THOUSANDS OF DELIVERED SOURCE INSTRUCTIONS.

ESTIMATING SOURCE LINES OF CODE

- NO GOLDEN RULES FOR THIS

- BASE ESTIMATES ON

- PAST COMPLETED PROGRAMS WITH HIGH DEGREES OF COMMONALITY
- EXPERIENCE OF A COST ESTIMATION GROUP
- GOOD ENGINEERING JUDGMENT

- ESTIMATES SHOULD BE MADE AND TRACKED TO ACTUALS ALL THROUGH THE PROJECT

- ESTIMATE EXPRESSED IN TERMS OF KDSIS

INSTRUCTOR NOTES

THESE CHARACTERIZATIONS ARE THE MOST SIGNIFICANT INFLUENCE OF THE BASIC MODEL. SOFTWARE COSTS ARE ALWAYS CHARACTERIZED BEING EXPONENTIALLY RELATED TO THE SIZE OF THE PRODUCT. WHICH CATEGORY OUR PRODUCT FALLS INTO AFFECTS A CONSTANT MULTIPLIER AND THE ACTUAL EXPONENT. (YOU CAN WORK OUT A COUPLE OF EXAMPLES TO SHOW HOW THE EXPONENT CAUSES SIGNIFICANT NON-LINEAR GROWTH AS SIZES GET LARGE.)

2.8 (40) ^{1.2}	=	234 MM	FOR 40,000 LINE EMBEDDED SYSTEM
2.8 (80) ^{1.2}	=	538 MM	FOR 80,000 LINE EMBEDDED SYSTEM
2.8 (400) ^{1.2}	=	3712 MM	FOR 400,000 LINE EMBEDDED SYSTEM
3.2 (400) ^{1.05}	=	1727 MM	FOR 400,000 LINE ORGANIC SYSTEM
3.2 (40) ^{1.05}	=	154 MM	FOR 40,000 LINE ORGANIC SYSTEM

THESE FIGURES SHOW THAT EMBEDDED SYSTEMS ARE MUCH MORE EXPONENTIALLY AFFECTED BY SIZE THAN ORGANIC SYSTEMS.

CHARACTERIZING THE SOFTWARE PRODUCT

- CATEGORIZE THE SOFTWARE
 - ORGANIC MODE
 - LIGHTLY CONSTRAINED
 - FAMILIAR SITUATION
 - EXAMPLE - DATA REDUCTION SOFTWARE
 - EMBEDDED MODE
 - TIGHTLY CONSTRAINED
 - UNFAMILIAR SITUATION
 - EXAMPLE - AIRCRAFT COLLISION AVOIDANCE SOFTWARE
 - SEMI-DETACHED MODE
 - BETWEEN ORGANIC AND EMBEDDED
 - EXAMPLE - TRAINER SOFTWARE

- NOMINAL COST IN MAN-MONTHS
 - ORGANIC : $MM_{NOM} = 3.2 \text{ (KDSI)}^{1.05}$
 - SEMI-DETACHED : $MM_{NOM} = 3.0 \text{ (KDSI)}^{1.12}$
 - EMBEDDED : $MM_{NOM} = 2.8 \text{ (KDSI)}^{1.20}$

INSTRUCTOR NOTES

THESE ARE ADDITIONAL MULTIPLICATIVE FACTORS THAT ARE USED IN COMING UP WITH THE FINAL MAN-MONTH ESTIMATE. NOTE THAT THE MOST IMPORTANT FACTORS ARE THE ONES WITH THE WIDEST RANGE NOT NECESSARILY THE LARGEST OR SMALLEST ABSOLUTE NUMBER (I.E., ANALYST CAPABILITY, PRODUCT COMPLEXITY).

WHETHER THE RANGE IS INCREASING OR DECREASING ISN'T IMPORTANT - JUST A NOTATIONAL CONVENIENCE - MUST SEE DETAILED DESCRIPTION TO UNDERSTAND.

CHARACTERIZING THE SOFTWARE PROJECT

•	COST DRIVERS IN SOFTWARE RESULT IN EFFORT MULTIPLIERS	
-	REQUIRED SOFTWARE RELIABILITY	[0.75 .. 1.40]
-	DATABASE SIZE	[0.94 .. 1.16]
-	PRODUCT COMPLEXITY	[0.70 .. 1.65]
-	EXECUTION TIME CONSTRAINT	[1.00 .. 1.66]
-	MAIN STORAGE CONSTRAINT	[1.00 .. 1.56]
-	VIRTUAL MACHINE VOLATILITY	[0.87 .. 1.30]
-	COMPUTER TURNAROUND TIME	[0.87 .. 1.15]
-	ANALYST CAPABILITY	[1.46 .. 0.71]
-	APPLICATION EXPERIENCE	[1.29 .. 0.82]
-	PROGRAMMING CAPABILITY	[1.42 .. 0.70]
-	VIRTUAL MACHINE EXPERIENCE	[1.21 .. 0.90]
-	PROGRAMMING LANGUAGE EXPERIENCE	[1.14 .. 0.95]
-	USE OF MODERN METHODS	[1.24 .. 0.82]
-	USE OF TOOLS	[1.24 .. 0.83]
-	REQUIRED DEVELOPMENT SCHEDULE	[1.23 .. 1.10]

• EFFORT MULTIPLIER

- = (PRODUCT OF EACH OF THE FACTORS ABOVE AS SELECTED FOR YOUR PROJECT).

INSTRUCTOR NOTES

THESE ARE THE FINAL STEPS OF ACTUALLY COMING UP WITH A PRICE TAG. IS THE PRODUCT OF ALL THE CHARACTERIZATIONS FROM THE PREVIOUS SLIDE.

STRESS THE IMPORTANCE OF THE ASSUMPTIONS IN PARTICULAR INSTRUCTION COUNT AND THE DIFFICULTY IN GETTING A GOOD ESTIMATE FOR KDSI.

ALSO POINT OUT THE CUMULATIVE EFFECT OF ALL THE OTHER CHARACTERIZATIONS.

USING THE COCOMO EQUATIONS

- ESTIMATING EFFORT IN MAN-MONTHS

$$MM = MM_{NOM} \cdot \pi$$

- ESTIMATING COST IN DOLLARS

$$COST = MM \cdot COST_{MM}$$

WHERE $COST_{MM}$ = AVERAGE COST PER MAN-MONTH FOR THE PROJECT

- REMEMBER ESTIMATES ARE ONLY AS ACCURATE AS YOUR ASSUMPTIONS

- DELIVERED INSTRUCTION COUNT
- CHARACTERISTICS OF PRODUCT AND PROJECT

INSTRUCTOR NOTES

NOTE THAT WHILE WE USUALLY TALK ABOUT QUALITY IN GENERAL TERMS WE CAN ESTABLISH QUANTITATIVE MEASURES SUCH AS NUMBER OF ERRORS PER 1000 LINES OF CODE OR PERCENT OF TRACEABLE REQUIREMENTS OR COMPLEXITY MEASURES ON SOFTWARE MODULES.

THE LAST POINT IS THE ONE TO BE STRESSED ON THIS SLIDE - QUALITY IS THE RESULT OF PLANNING AND MUST BE INTEGRATED INTO THE ENTIRE DEVELOPMENT PROCESS - QUALITY CAN'T JUST BE TESTED IN AT THE END OF A PROJECT.

SOFTWARE QUALITY MANAGEMENT

- WHAT IS "QUALITY"?

- AN ESSENTIAL NATURE OR CHARACTERISTIC
- THE DEGREE OF EXCELLENCE

- WHAT IS "SOFTWARE QUALITY"?

- DEGREE OF EXCELLENCE EXHIBITED BY A SOFTWARE PRODUCT (WHERE A SOFTWARE PRODUCT CONSISTS OF PROGRAMS AND OTHER RELEVANT DOCUMENTATION NEEDED TO SPECIFY, DESIGN, CODE, TEST, USE AND MAINTAIN THOSE PROGRAMS)

- FOR SOFTWARE QUALITY MANAGEMENT TO BE SUCCESSFUL IT MUST BE TREATED AS AN INTEGRAL PART OF THE DEVELOPMENT PROCESS

INSTRUCTOR NOTES

VERIFICATION AND VALIDATION AND SOFTWARE QUALITY ASSURANCE ARE SOMETIMES USED INTERCHANGEABLY. IN FACT VERIFICATION AND VALIDATION ARE TECHNIQUES FOR ASSURING SOFTWARE QUALITY (THE GOAL).

VERIFICATION IS A PROCESS PERFORMED AT EACH STEP AND COVERS JUST THAT ONE STEP.

VALIDATION IS AN END-TO-END PROCESS WHICH ATTEMPTS TO MAKE SURE THE COMPLETED SYSTEM MEETS ITS OVERALL OBJECTIVES.

SOFTWARE QUALITY MANAGEMENT IS A PROCESS OF

VERIFICATION AND VALIDATION

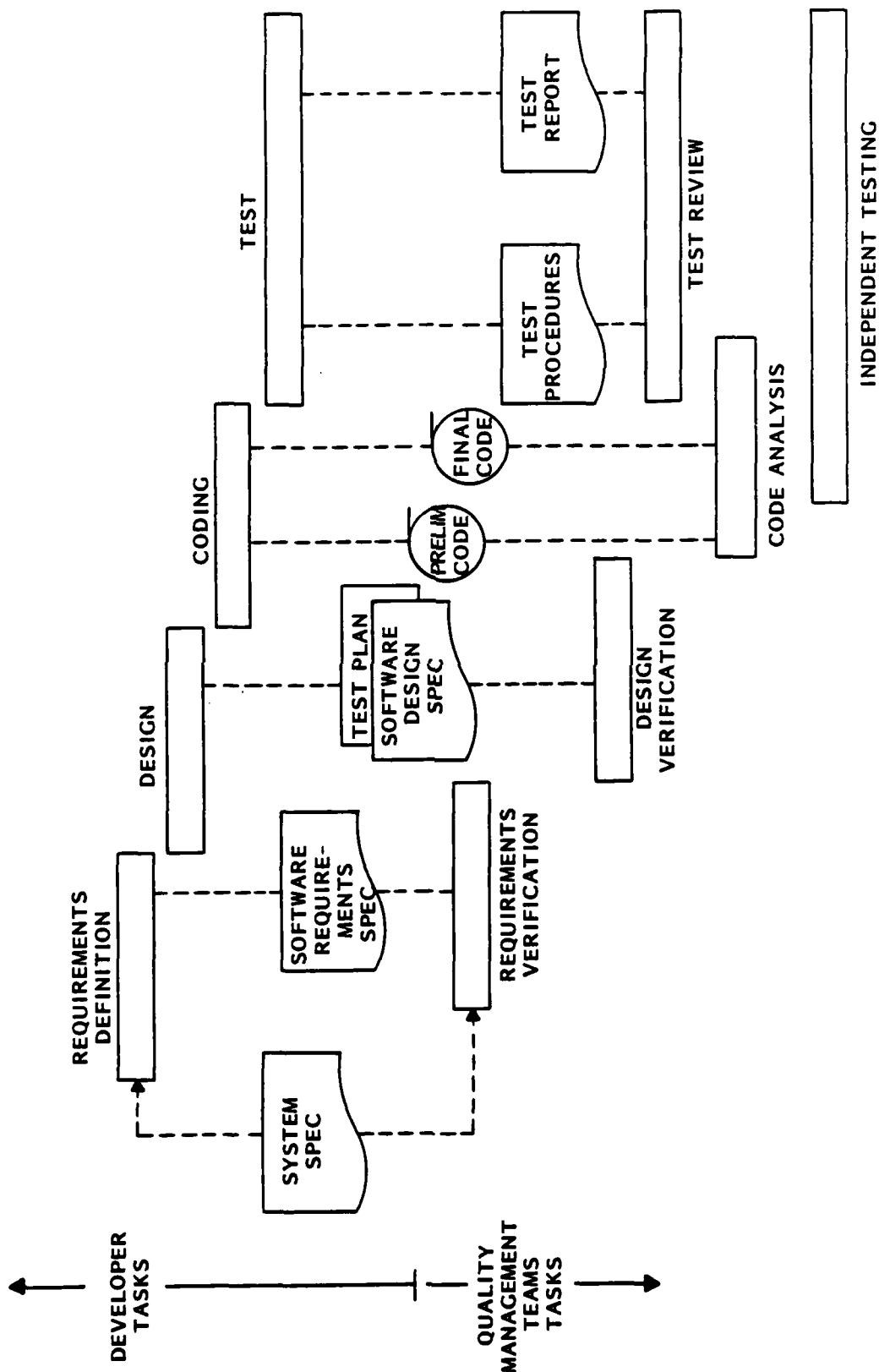
- VERIFICATION - ITERATIVE PROCESS OF DETERMINING WHETHER EACH STEP OF THE DEVELOPMENT PROCESS FULFILLS ALL THE REQUIREMENTS LEVIED BY THE PREVIOUS STEP.
 - IS THE PRODUCT COMPLETE?
 - IS THE PRODUCT CONSISTENT?

- VALIDATION - EVALUATION, INTEGRATION AND TEST ACTIVITIES CARRIED OUT AT THE SYSTEM LEVEL TO ENSURE THE PRODUCT IS MEETING ALL REQUIREMENTS.
 - IS THE RIGHT PROBLEM BEING SOLVED?
 - IS THE SOLUTION ACCURATE?
 - IS THE PRODUCT USABLE?

INSTRUCTOR NOTES

THIS SLIDE SHOWS THE VERIFICATION STEPS (REQUIREMENTS VERIFICATION, DESIGN VERIFICATION, CODE ANALYSIS, AND TEST REVIEW). ALSO SHOWN IS INDEPENDENT TESTING WHICH IS THE VALIDATION TASK. IT WOULD PROBABLY BE APPROPRIATE FOR THE SYSTEM SPECIFICATION TO BE SHOWN AS PROVIDING INPUTS TO THIS INDEPENDENT TESTING TASK.

VERIFICATION AND VALIDATION TASKS



INSTRUCTOR NOTES

MOST OF THESE TECHNIQUES ARE REVERIFICATION TECHNIQUES - THEY ARE APPLIED TO ONE OR MORE STEPS, BUT EACH APPLICATION ONLY COVERS THAT PARTICULAR STEP.

THE FINAL THREE TECHNIQUES ARE PART OF THE VALIDATION TASK.

VERIFICATION AND VALIDATION TECHNIQUES

VERIFICATION AND VALIDATION TECHNIQUE	LIFE CYCLE PHASE				RELATIVE COST
	ANALYSIS	DESIGN	IMPLEMENTATION		
			CODE	TEST	
REQUIREMENTS TRACEABILITY	X	X	X	X	MEDIUM
INTERFACE ANALYSIS	X	X	X	X	MEDIUM
QUALITY ASSESSMENT	X	X	X	X	MEDIUM
SECURITY EVALUATION	X	X	X	X	LOW
INDEPENDENT STRUCTURED ANALYSIS	X	X			HIGH
FAILURE MODES & EFFECTS ANALYSIS	X	X	X	X	MEDIUM
DESIGN WALKTHROUGH	X	X	X	X	LOW
PROTOTYPING		X	X		HIGH
MODULE INDEPENDENCE ANALYSIS		X	X		MEDIUM
CODE INSPECTION			X		LOW
CODE WALKTHROUGH			X		LOW
STATIC CODE ANALYSIS			X		LOW
DYNAMIC CODE ANALYSIS			X	X	HIGH
TEST PLANS/PROCEDURES EVALUATION				X	LOW
INDEPENDENT TESTING				X	MEDIUM
TEST WITNESSING/EVALUATION				X	LOW

INSTRUCTOR NOTES

NOTE THAT THESE SOFTWARE QUALITY FACTORS ARE VERY SIMILAR TO THE GOALS WE IDENTIFIED AND DISCUSSED EARLIER AND MEASURED ALL OF OUR OTHER TECHNIQUES AGAINST - THIS DRIVES HOME THE IMPORTANCE OF QUALITY BEING INTEGRATED INTO THE OVERALL PROCESS.

WE ARE REVISITING THESE GOALS HERE BECAUSE THEY ARE NORMALLY THOUGHT OF AS BEING PART OF THE QUALITY ASSURANCE ASPECTS OF PROJECT MANAGEMENT.

THE FOLLOWING SPLIT INTO SOFTWARE QUALITY FACTORS, OBJECTIVES, AND QUALITY CRITERIA ARE FROM THE DISCIPLINE CALLED QUALITY METRICS - A MORE SCIENTIFIC APPROACH TO QUALITY ASSURANCE.

HOW DO YOU ASSESS QUALITY

• DEFINITION OF VARIOUS SOFTWARE QUALITY FACTORS

- CORRECTNESS
EXTENT TO WHICH A PROGRAM SATISFIES ITS SPECIFICATIONS AND FULFILLS THE USER'S MISSION OBJECTIVES.
- RELIABILITY
EXTENT TO WHICH A PROGRAM CAN BE EXPECTED TO PERFORM ITS INTENDED FUNCTION WITH REQUIRED PRECISION.
- EFFICIENCY
THE AMOUNT OF COMPUTING RESOURCES AND CODE REQUIRED BY A PROGRAM TO PERFORM A FUNCTION.
- INTEGRITY
EXTENT TO WHICH ACCESS TO SOFTWARE OR DATA BY UNAUTHORIZED PERSONS CAN BE CONTROLLED.
- USABILITY
EFFORT REQUIRED TO LEARN, OPERATE, PREPARE INPUT, AND INTERPRET OUTPUT OF A PROGRAM.
- MAINTAINABILITY
EFFORT REQUIRED TO LOCATE AND FIX AN ERROR IN AN OPERATIONAL PROGRAM.
- TESTABILITY
EFFORT REQUIRED TO TEST A PROGRAM TO ENSURE IT PERFORMS ITS INTENDED FUNCTION.
- PORTABILITY
EFFORT REQUIRED TO TRANSFER A PROGRAM FROM ONE HARDWARE CONFIGURATION AND/OR SOFTWARE SYSTEM ENVIRONMENT TO ANOTHER.
- REUSABILITY
EXTENT TO WHICH A PROGRAM CAN BE USED IN OTHER APPLICATIONS - RELATED TO THE PACKAGING AND SCOPE OF THE FUNCTIONS THAT PROGRAMS PERFORM.
- INTEROPERABILITY
EFFORT REQUIRED TO COUPLE ONE SYSTEM WITH ANOTHER.

INSTRUCTOR NOTES

NOTE AGAIN THAT THESE OBJECTIVES ARE SIMILAR TO THOSE WE DISCUSSED AT THE BEGINNING OF THIS COURSE, BUT ARE BROADER - SOME OF THESE ARE NOT UNIVERSALLY APPLICABLE. IN ANY SPECIFIC PROJECT WE PRIORITIZE OUR SYSTEM'S OBJECTIVES AND THEN USE THESE OBJECTIVES TO HELP US PRIORITIZE THE QUALITY FACTORS - THESE ARE THE FACTORS WE WILL TRY TO MAXIMIZE IN ORDER TO MEET OUR OBJECTIVES.

THE PROCESS DOESN'T END HERE - IN THIS FORMAL FIELD OF QUALITY METRICS WE NEXT ATTEMPT TO IDENTIFY THE QUALITY CRITERIA (QUANTIFIABLE, MEASURABLE THINGS). THE ASSUMPTION HERE IS THAT THE QUALITY FACTORS CAN'T BE DIRECTLY MEASURED - BUT THE QUALITY CRITERIA CAN BE MORE EASILY MEASURED.

QUALITY FACTORS HAVE A DIRECT RELATIONSHIP
TO OVERALL SYSTEM OBJECTIVES

OBJECTIVES

QUALITY FACTORS

• SAFETY (HUMAN LIVES AFFECTED)	_____●_____	RELIABILITY CORRECTNESS TESTABILITY
• HIGH SYSTEM COST	_____●_____	RELIABILITY FLEXIBILITY
• LONG LIFE CYCLE	_____●_____	MAINTAINABILITY PORTABILITY FLEXIBILITY
• REAL TIME APPLICATION	_____	EFFICIENCY
• ON-BOARD APPLICATION	_____●_____	RELIABILITY EFFICIENCY
• PROCESS CLASSIFIED DATA	_____	INTEGRITY
• INTER-RELATED SYSTEMS	_____	INTEROPERABILITY

INSTRUCTOR NOTES

THESE TWO SLIDES IDENTIFY AND DEFINE THE QUALITY CRITERIA.

NOTE THAT TRACEABILITY (WHICH WE HAD IN THE INTRODUCTION IS INCLUDED AS A GOAL) IS DEFINED HERE AS A CRITERIA, BECAUSE THERE ARE MEANS TO MEASURE IT, WHERE CORRECTNESS IS A QUALITY FACTOR BECAUSE IT IS HARD TO QUANTIFY.

VG 744.1

15-271

CRITERIA MEASURED TO ASSESS SOFTWARE QUALITY

- TRACEABILITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE A THREAD FROM THE REQUIREMENTS TO THE IMPLEMENTATION WITH RESPECT TO THE SPECIFIC DEVELOPMENT AND OPERATIONAL ENVIRONMENT.
- COMPLETENESS
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FULL IMPLEMENTATION OF THE FUNCTIONS REQUIRED.
- CONSISTENCY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE UNIFORM DESIGN AND IMPLEMENTATION TECHNIQUES AND NOTATION.
- ACCURACY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE THE REQUIRED PRECISION IN CALCULATIONS AND OUTPUTS.
- ERROR TOLERANCE
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE CONTINUITY OF OPERATION UNDER NON-NOMINAL CONDITIONS.
- SIMPLICITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE IMPLEMENTATION OF FUNCTIONS IN THE MOST UNDERSTANDABLE MANNER. (USUALLY AVOIDANCE OF PRACTICES WHICH INCREASE COMPLEXITY.)
- MODULARITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE A STRUCTURE OF HIGHLY INDEPENDENT MODULES.
- GENERALITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE BREADTH TO THE FUNCTIONS PERFORMED.
- EXPANDABILITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR EXPANSION OF DATA STORAGE REQUIREMENTS OR COMPUTATIONAL FUNCTIONS.
- INSTRUMENTATION
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR THE MEASUREMENT OF USAGE OR IDENTIFICATION OF ERRORS.

INSTRUCTOR NOTES

MORE DEFINITIONS OF QUALITY CRITERIA - AGAIN WE SEE EFFICIENCY (ONE OF OUR ORIGINAL GOALS) IDENTIFIED AS A CRITERIA - BECAUSE IT IS MEASURABLE.

CRITERIA MEASURED TO ASSESS SOFTWARE QUALITY

(CONTINUED)

- SELF-
DESCRIPTIVENESS
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE EXPLANATION OF THE IMPLEMENTATION OF A FUNCTION.
- EXECUTION
EFFICIENCY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR MINIMUM PROCESSING TIME.
- STORAGE EFFICIENCY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR MINIMUM STORAGE REQUIREMENTS DURING OPERATION.
- ACCESS CONTROL
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR CONTROL OF THE ACCESS TO SOFTWARE AND DATA.
- ACCESS AUDIT
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR AN AUDIT OF THE ACCESS TO SOFTWARE AND DATA.
- OPERABILITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT DETERMINE OPERATION AND PROCEDURES CONCERNED WITH THE OPERATION OF THE SOFTWARE.
- TRAINING
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE INITIAL FAMILIARIZATION OR TRANSITION FROM CURRENT OPERATION
- COMMUNICATIVENESS
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE USEFUL INPUTS AND OUTPUTS WHICH CAN BE ASSIMILATED.
- SOFTWARE SYSTEM
INDEPENDENCE
THOSE ATTRIBUTES OF THE SOFTWARE THAT DETERMINE ITS DEPENDENCY ON THE SOFTWARE ENVIRONMENT (OPERATING SYSTEMS, UTILITIES, INPUT/OUTPUT ROUTINES, ETC.)
- MACHINE
INDEPENDENCE
THOSE ATTRIBUTES OF THE SOFTWARE THAT DETERMINE ITS DEPENDENCY ON THE HARDWARE SYSTEM.

INSTRUCTOR NOTES

VG 744.1

15-29i

CRITERIA MEASURED TO ASSESS SOFTWARE QUALITY

(CONTINUED)

- COMMUNICATIONS COMMONALITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE THE USE OF STANDARD PROTOCOLS AND INTERFACE ROUTINES.
- DATA COMMONALITY
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE THE USE OF STANDARD DATA REPRESENTATIONS.
- CONCISENESS
THOSE ATTRIBUTES OF THE SOFTWARE THAT PROVIDE FOR IMPLEMENTATION OF A FUNCTION WITH A MINIMUM AMOUNT OF CODE.

INSTRUCTOR NOTES

THIS MATRIX SHOWS THE INTERRELATIONSHIPS BETWEEN THE CRITERIA AND THE QUALITY FACTORS THAT THEY CAN EFFECT (BOTH POSITIVELY AND NEGATIVELY).

REMEMBER THE QUALITY FACTORS ARE WHAT WE ARE AFTER AS INHERENTLY DESIRABLE ATTRIBUTES - THE QUALITY CRITERIA ARE MEASURABLE AND LEAD TO IMPROVING QUALITY FACTORS - THE QUALITY CRITERIA ARE NOT INHERENTLY DESIRABLE (FROM AN END-USER POINT OF VIEW).

EFFECT OF CRITERIA ON SOFTWARE QUALITY FACTORS

QUALITY FACTORS CRITERIA	EFFECT OF CRITERIA ON SOFTWARE QUALITY FACTORS										
	CORRECTNESS	RELIABILITY	EFFICIENCY	INTEGRITY	USABILITY	MAINTAINABILITY	TESTABILITY	FLEXIBILITY	PORTABILITY	REUSABILITY	INTEROPERABILITY
TRACEABILITY	o					o	o	o		o	
COMPLETENESS	o	o			o						
CONSISTENCY	o	o				o	o	o		o	
ACCURACY		o	•		o						
ERROR TOLERANCE	o	o	•		o						
SIMPLICITY	o	o	o			o	o	o	o	o	o
MODULARITY			•			o	o	o	o	o	o
GENERALITY		•	•	•				o		o	o
EXPANDABILITY			•					o		o	
INSTRUMENTATION			•		o	o	o				
SELF-DESCRIPTIVENESS			•			o	o	o	o	o	
EXECUTION EFFICIENCY			o						•		
STORAGE EFFICIENCY			o				•		•		
ACCESS CONTROL			•	o	o			•			•
OPERABILITY			•		o					o	
TRAINING					o					o	
COMMUNICATIVENESS			•		o	o	o	o		o	
SOFTWARE SYSTEM INDEPENDENCE			•					o	o	o	o
MACHINE INDEPENDENCE			•					o	o	o	o
COMMUNICATIONS COMMONALITY											o
DATA COMMONALITY				•						o	o
CONCISENESS	o		o			o	o				
ACCESS AUDIT			•	o							

LEGEND - ATTRIBUTES ASSOCIATED WITH CRITERIA HAVE:
 o NEGATIVE EFFECT ON QUALITY FACTOR o POSITIVE EFFECT ON QUALITY FACTOR

INSTRUCTOR NOTES

THERE ARE THREE MAJOR ASPECTS TO CM

- (1) IDENTIFYING - NAMING, COUNTING, NUMBERING, RELATIONSHIPS, ETC.
- (2) TRACKING - THE CHANGES IN THESE IDENTIFIED ENTITIES OVER TIME
- (3) STATUS REPORTING - OF THE IDENTIFICATION AND TRACKED CHANGES

CONFIGURATION MANAGEMENT

- CONFIGURATION MANAGEMENT IS THE APPLICATION OF TECHNICAL AND ADMINISTRATIVE

CONTROL TO:

- IDENTIFY AND DOCUMENT PHYSICAL CHARACTERISTICS OF CONFIGURATION ITEMS (SOFTWARE DELIVERABLES) AND THE RELATIONSHIPS AMONG THESE ITEMS
- TRACK THOSE CHARACTERISTICS AND RELATIONSHIPS
- RECORD AND REPORT CHANGE PROCESSING AND STATUS

INSTRUCTOR NOTES

ON SMALL SIMPLE SYSTEMS, WE CAN PROBABLY GET AWAY WITH ONLY CONCERNING OURSELVES WITH THE IDENTIFICATION OF CODE - NOT NECESSARILY DISTINGUISHING BETWEEN SOURCE, OBJECT, AND EXECUTABLE IMAGE. AS SYSTEMS GET LARGER AND MORE COMPLEX (AND IN PARTICULAR AS A SYSTEM IS DEPLOYED IN DIFFERENT CONFIGURATIONS) WE MUST ALSO IDENTIFY THE COMPONENTS OF THE DOCUMENTATION, THE TOOLS (COMPILERS) USED TO PROCESS THE CODE AND DATA, AND DISTINGUISH BETWEEN THE DIFFERENT FORMS OF CODE.

CONFIGURATION ITEMS AND IDENTIFICATION

- ANY ITEM WHICH IS A PRODUCT OF A SOFTWARE DEVELOPMENT ACTIVITY
 - SYSTEM DOCUMENTATION
 - SOURCE CODE
 - OBJECT CODE
 - EXECUTABLE IMAGES
 - PROGRAMMING SUPPORT TOOLS
- IT MUST BE POSSIBLE TO UNIQUELY IDENTIFY EVERY ITEM UNDER CONFIGURATION CONTROL

INSTRUCTOR NOTES

THERE ARE TWO ASPECTS TO THIS POINT

- (1) CONTROLLING WHO IS ALLOWED TO CHANGE WHAT (THIS IS MORE IMPORTANT ON AUTOMATED SYSTEMS THAN FOR MANUAL ONES), AND
- (2) KEEPING A RECORD OF THE ALLOWABLE CHANGES.

THESE CONTROLS APPLY TO THE FORMAL PART OF THE SYSTEM - THE OFFICIAL BASELINE - PRIVATE CHANGES THAT DON'T AFFECT THE SHARED OR CONTROL BASELINE - ARE USUALLY ALLOWED AND ARE NOT GENERALLY TRASHED.

CHANGE CONTROL AND TRACKING

- PROGRAMMERS SHOULD BE ALLOWED ACCESS ONLY TO THOSE CONFIGURATION ITEMS THEY NEED
- IT SHOULD BE POSSIBLE TO MAKE PRIVATE EXPERIMENTAL CHANGES THAT DO NOT AFFECT OTHER PROGRAMMERS
- MECHANISMS TO CHANGE AND DOCUMENT SOFTWARE MUST BE PROVIDED
- CHANGE DOCUMENTATION INCLUDES AT LEAST:
 - WHO MADE THE CHANGE
 - WHY THE CHANGE WAS MADE
 - DESCRIPTION OF THE CHANGE
 - DATE OF THE CHANGE

INSTRUCTOR NOTES

STATUS ACCOUNTING IS PRIMARILY CONCERNED WITH PRODUCING MANAGEMENT REPORTS - BUT IT SHOULD ALSO BE ABLE TO RESPOND TO SPECIFIC QUERIES OR ALLOW THE TRACING AND/OR RECONSTRUCTION OF ANY ITEM AT ANY TIME.

STATUS ACCOUNTING

- MECHANISM FOR MAINTAINING A RECORD OF HOW A CONFIGURATION ITEM HAS EVOLVED
- PROVIDES STATUS OF THE CONFIGURATION ITEM AT ANY TIME
- PROVIDES REPORTING CAPABILITIES
 - PROJECT STATUS VISIBILITY TO MANAGEMENT
 - CHANGE CONSISTENCY FOR CM PERSONNEL
 - REQUIRED CHANGE INFORMATION FOR PROGRAMMERS

INSTRUCTOR NOTES

WE'VE TALKED ABOUT THE PRINCIPLES - NOW WE WILL ADDRESS THE MECHANISMS FOR ACCOMPLISHING THE CM OBJECTIVES. ANY CM SYSTEM INCLUDES MANUAL PROCEDURES. SOME CM SYSTEMS ALSO INCLUDE AUTOMATED CAPABILITIES.

CM SYSTEM

- A CM SYSTEM CONSISTS OF

- A SET OF DOCUMENTED MANUAL PROCEDURES
- PROCEDURES AUTOMATICALLY ENFORCED
- AUTOMATED TOOLS

TO SUPPORT CONFIGURATION MANAGEMENT FUNCTIONS

INSTRUCTOR NOTES

NOTE THAT NOT ALL OF THESE FEATURES WOULD BE AUTOMATED IN MOST CM SYSTEMS ALTHOUGH FUTURE SYSTEMS WILL PROBABLY ACCOMPLISH THIS AUTOMATION. CURRENTLY VERY FEW SYSTEMS AUTOMATICALLY PERFORM LISTING AND RETESTING OF COMPONENTS - FASP (FACILITY FOR AUTOMATED SOFTWARE PRODUCTION) ON NADC ENVIRONMENT DOES PROVIDE THIS LAST CAPABILITY. TO BE COMPLETELY EFFECTIVE, ALL OF THESE CAPABILITIES SHOULD BE INTEGRATED, AS WELL AS AUTOMATED.

VG 744.1

15-361

CM SYSTEM
(CONTINUED)

- THE BASIC LEVEL OF SUPPORT INCLUDES:
 - PROGRAM LIBRARIES (COLLECTIONS)
 - PROBLEM/CHANGE REPORTING PROCEDURES AND FORMS (OUTSIDE CHANGES)
 - ERROR TRACKING TOOLS AND PROCEDURES (BUG FIXES)
 - DOCUMENTATION CONTROL TOOLS AND PROCEDURES (CONSISTENT CHANGES)
 - AUTOMATIC TESTING AND RETESTING OF COMPONENTS

INSTRUCTOR NOTES

THESE BASIC GOALS PRETTY WELL SUMMARIZE THE MAJOR CAPABILITIES THAT A CM SYSTEM SHOULD PROVIDE - MANY MODERN CM SYSTEMS SUCH AS THE ALS CAN PROVIDE AN AUTOMATED CAPABILITY COVERING MOST OR ALL OF EACH OF THESE GOALS.

IT IS IMPORTANT TO STRESS, HOWEVER, THAT THESE GOALS CAN ALSO BE ATTAINED BY MANUAL PROCEDURES PERFORMED BY CONSISTENT AND CONSCIENTIOUS PEOPLE.

GOALS OF A CONFIGURATION MANAGEMENT SYSTEM

- PROVIDE A CENTRAL REPOSITORY FOR SOFTWARE CONFIGURATION ITEMS
- MAINTAIN CONFIGURATION MANAGEMENT DATA FOR CONFIGURATION ITEMS
- SUPPORT QUERIES/REPORTS OF THE CONTROLLED CONFIGURATION STRUCTURE AND CONTENTS
- PROVIDE AN INTERFACE CAPABILITY TO COMPILERS AND SYSTEM GENERATION TOOLS

CONFIGURATION MANAGEMENT

- A CM SYSTEM MAINTAINS:
 - CREATION DATES, CREATOR, AND REASON FOR CREATION
 - DERIVED-FROM/DERIVES-TO RELATIONS
 - CONFIGURATION AND STATUS INFORMATION FOR BUILT SYSTEMS
 - VERSIONS OF CONFIGURATION ITEMS (A HISTORY)
 - VARIATIONS OF CONFIGURATION ITEMS (AN ALTERNATE IMPLEMENTATION)

INSTRUCTOR NOTES

IT IS IMPORTANT TO POINT OUT THAT EVEN WHEN AUTOMATED TOOLS ARE AVAILABLE (SUCH AS IN ALS) THAT AN OVERALL SCHEME OR CM STRATEGY (OR A POLICY) STILL NEEDS TO BE DEVELOPED AND ENFORCED. TOOLS USUALLY JUST PROVIDE CM CAPABILITIES, EVEN IF THEY ARE INTEGRATED. EITHER THESE TOOLS NEED TO BE CONFIGURED OR CONSTRAINED BY A MASTER TOOL OR ADDITIONAL MANAGEMENT PROCEDURES NEED TO BE PUT IN PLACE.

SUMMARY

- CONFIGURATION MANAGEMENT

- CONTROLS THE DETAILS OF CHANGE
- REQUIRES AN OVERALL SCHEME
- REQUIRES INTEGRATED TOOLS
- TRACKS VERSIONS

INSTRUCTOR NOTES

IV. SUMMARY/SOFTWARE ENGINEERING AND Ada - ALLOW 1/4 HOUR

SUMMARIZE THE DAY AND SHOW THE RELATIONSHIP OF SOFTWARE ENGINEERING WITH Ada

PART IV

SUMMARY

VG 744.1

INSTRUCTOR NOTES

VG 744.1

16-1

SECTION 16

SOFTWARE ENGINEERING AND Ada

VG 744.1

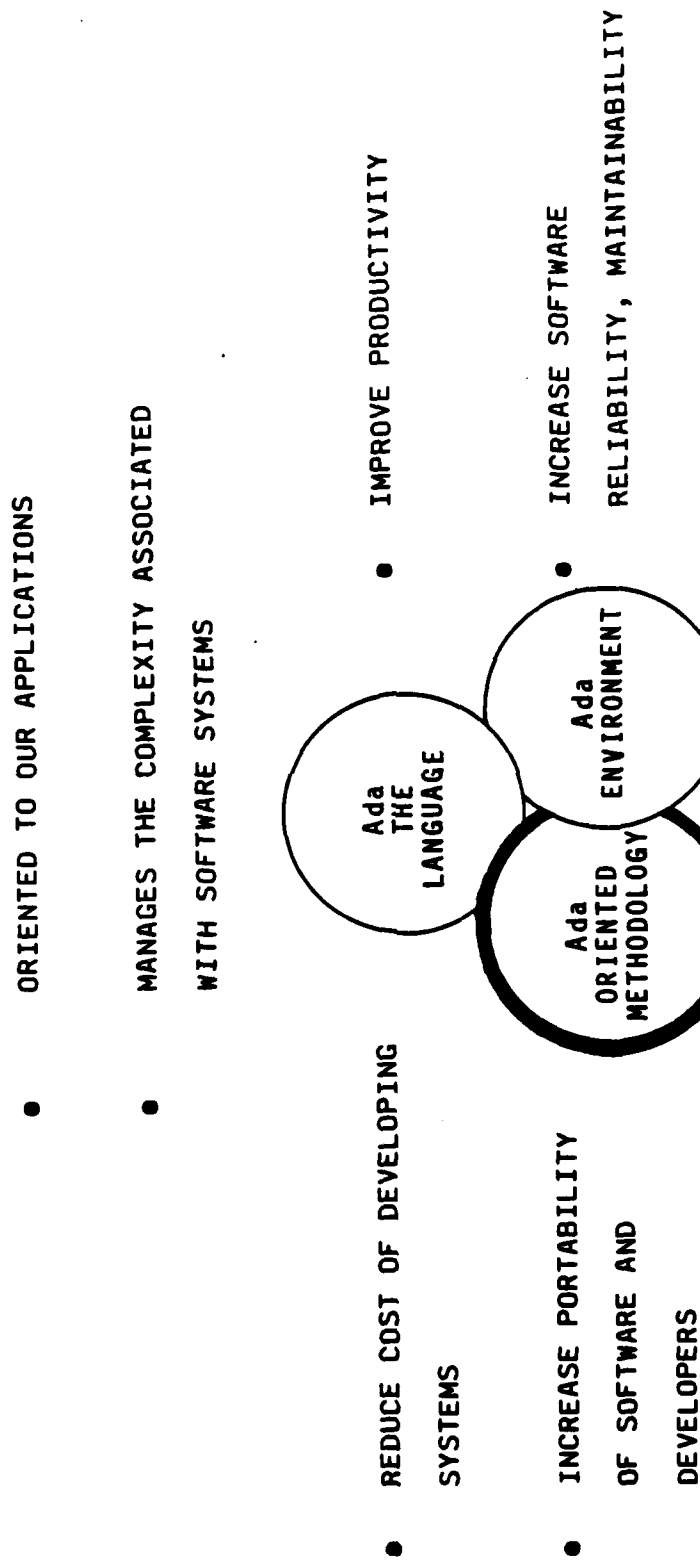
INSTRUCTOR NOTES

THIS SLIDE SHOWS HOW THE DoD VIEW THE TOTAL Ada EFFORT.

VG 744.1

16-11

RELATIONSHIP OF "Ada THE LANGUAGE" AND METHODOLOGIES



THREE ASPECTS WHICH ADDRESS THE "SOFTWARE PROBLEM"

INSTRUCTOR NOTES

THESE ARE THE STATED GOALS OF THE METHODMAN DOCUMENT, AND TO SOME EXTENT THE STARS
PROGRAM (AS OF EARLY 1985).

Ada ORIENTED METHODOLOGY

- GOAL IS TO ENCOURAGE THE USE OF AND SUPPLEMENT THE DEVELOPMENT OF THE SET OF METHODS USED IN THE PROCESS OF MANAGING, DEVELOPING, AND MAINTAINING SOFTWARE SYSTEMS

- MUST ADDRESS FULL LIFE CYCLE

- KEY ASSUMPTION - "INCREASED EFFORT IN EARLIER STAGES OF DEVELOPMENT WILL BE REFLECTED IN REDUCED COST FOR TESTING AND MAINTENANCE"

- FOCUS COMES FROM

- METHODMAN

- STARS (SOFTWARE TECHNOLOGY FOR ADAPTABLE AND RELIABLE SYSTEMS)

INSTRUCTOR NOTES

DETAILS OF METHODMAN GOALS.

VG 744.1

16-31

"METHODMAN" GOALS

GENERAL GOALS:

- BALANCE BETWEEN SIMPLICITY AND COMPLEXITY
- CONTROL OF COMPLEXITY
- RIGOR
- APPLY TO ANY PROBLEM DOMAIN

TECHNICAL GOALS:

- CRITERIA GIVEN FOR ALL TECHNICAL ASPECTS
- FORMALIZATION OF SPECIFICATIONS AND DESIGN
- VERIFICATION OF SPECIFICATION AND DESIGN DECISIONS
- PROVIDE AN EXPLICIT MODEL OF THE REAL WORLD
- OVERALL OPTIMIZATION OF LOGICAL/PHYSICAL DATA
- AND PROCESSING ARCHITECTURES
- SUPPORT DESIGN OF CONCURRENT HARDWARE AND SOFTWARE SYSTEMS

AUTOMATION GOALS:

- AUTOMATE LIFE CYCLE PROCESSES THAT ARE CONVENTIONALLY DONE MANUALLY AND REDUNDANTLY
- INTEGRATED FAMILY OF TOOLS
- PROVIDE GRAPHICAL TOOLS

PRODUCT MANAGEMENT GOALS:

- PROVIDE QUALITY CONTROL
- PROVIDE VERSION CONTROL
- PROVIDE CONFIGURATION MANAGEMENT
- PROVIDE AN EXPLICIT MODEL OF THE SOFTWARE DEVELOPMENT PROCESS

PROJECT MANAGEMENT GOALS:

- IMPROVE THE MANAGEABILITY OF SOFTWARE PRODUCTION
- IMPROVE THE EFFICIENCY OF SOFTWARE PRODUCTION
- IMPROVE THE SE PRACTICES OF PROGRAMMERS IN THE ORGANIZATION
- 100% CENTRALLY VERIFIED CONSISTENCY
- IMPROVE PRODUCTIVITY OVER THE ENTIRE LIFE CYCLE

"A GOOD METHODOLOGY"

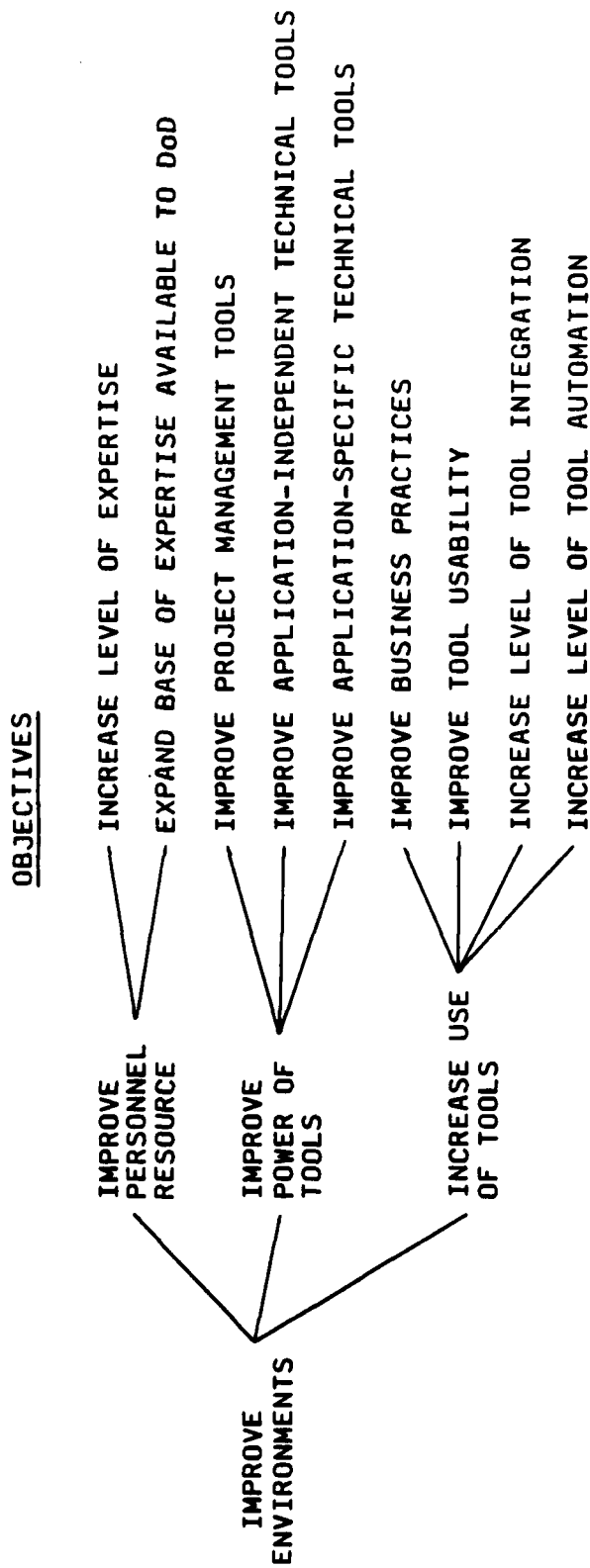
INSTRUCTOR NOTES

DETAILS OF STARS PROGRAM GOALS.

VG 744.1

16-41

THE STARS PROGRAM
WILL FOCUS Ada METHODOLOGY DEVELOPMENT



"... TO IMPROVE PRODUCTIVITY WHILE ACHIEVING GREATER SYSTEM RELIABILITY AND ADAPTABILITY"

Material: Introduction to Software Engr'g (M102), Teacher's Guide

A144236

We would appreciate your comments on this material and would like you to complete this brief questionnaire. The completed questionnaire should be forwarded to the address on the back of this page. Thank you in advance for your time and effort.

1. Your name, company or affiliation, address and phone number.

2. Was the material accurate and technically correct?

Yes ☐

No ☐

Comments:

3. Were there any typographical errors?

Yes ☐

No ☐

If yes, on what pages?

4. Was the material organized and presented appropriately for your applications?

Yes ☐

No ☐

Comments:

5. General Comments:

place
stamp
here

COMMANDER
US ARMY MATERIEL COMMAND
ATTN: AMCDE-SB (OGLESBY)
5001 EISENHOWER AVENUE
ALEXANDRIA, VIRGINIA 22233

END

FILMED



DTIC