

AD-A165 382

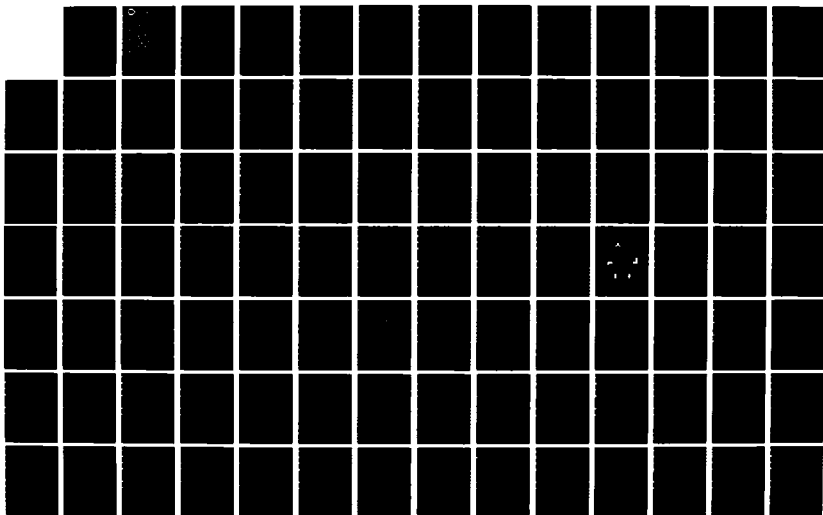
ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 3
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

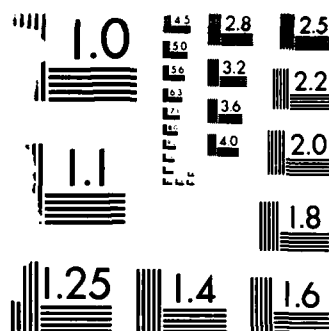
1/3

UNCLASSIFIED

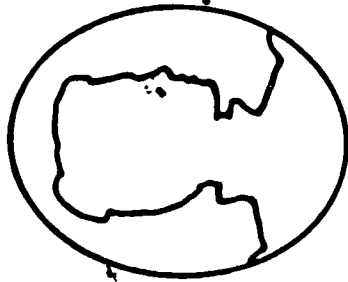
F/G 5/9

NL





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



Ada® Training Curriculum

AD-A165 302

Software Engineering Methodologies

M201

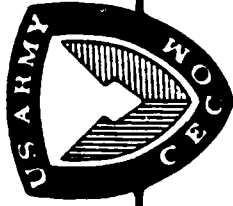
Teacher's Guide

Volume III

DTIC FILE COPY

①

1986



DTIC
ELECTE

MAR 12 1986



B

*Supp A144258
AD*

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

Contract DAAB07-83-C-K506

Prepared By:

SOFTECH, INC.

460 Totten Pond Road
Waltham, MA 02154

INSTRUCTOR NOTES

THEME: PDL's ARE RAPIDLY BECOMING THE REQUIRED (PREFERRED) METHODS FOR EXPRESSING
A DETAILED SOFTWARE DESIGN.

PURPOSE: TO SHOW THE IMPORTANT FEATURES OF ADA PDL.

REFERENCE: VAN LEER, P. "TOP DOWN DEVELOPMENT USING A PROGRAM DESIGN LANGUAGE" IBM
SYSTEMS JOURNAL VOL. 15, NO. 2; 1976

IEEE GUIDELINES FOR AN ADA PDL; 1985

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).

Section 21

PROGRAM DESIGN LANGUAGES

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
BY-	
Distribution/	
Availability Codes	
AVAIL CODES	
Dist	Special
A-1	



VG 778.1

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.

PRECISION ←————→ EXPRESSION

ALWAYS COMPETE.

PROPER BALANCE IS UNKNOWN.

- PDL'S GREW OUT OF THE PROBLEMS ON 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

WHILE THESE QUALITIES ARE DIFFICULT TO DEFINE PRECISELY OR QUANTIFY, THEY ARE USEFUL IN COMPARING ONE DESIGN LANGUAGE TO ANOTHER. FOR EXAMPLE, A DESIGN LANGUAGE WHICH LEADS TO A READABLE DESIGN DESCRIPTION IS BETTER THAN ONE WHICH PRODUCES UNREADABLE DESIGNS. EACH ORGANIZATION SHOULD HAVE ITS OWN DEFINITIONS AND MEASUREMENT STANDARDS FOR THE ABOVE TERMS CONSISTENT WITH ITS PARTICULAR BUSINESS AND DEVELOPMENT ENVIRONMENT.

QUALITIES OF A DESIGN EXPRESSED IN A PDL

- READABLE
- UNDERSTANDABLE
- ANALYZABLE
- TRACEABLE
- VALIDATABLE
- RELIABLE
- MINIMUM COUPLING
- MAXIMUM COHESIVENESS
- COMPLETE
- PRECISE
- VERIFIABLE
- STRUCTURED
- CONSISTENT

INSTRUCTOR NOTES

THESE CHARACTERISTICS ARE DEPENDENT UPON THE METHODOLOGY USED.

VG 778.1

21-61

111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

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



PDL CHARACTERISTICS

- PDL MUST BE ABLE TO EXPRESS PRODUCT RELATED INFORMATION ...
 - CHARACTERISTICS - INCLUDES PERFORMANCE CONSTRAINTS, SIZE RESTRICTIONS, THE TARGET ENVIRONMENT, RELIABILITY, AND SECURITY REQUIREMENTS.
 - PERFORMANCE - INCLUDES CRITICAL TIMING, FREQUENCY, CAPACITY AND OTHER LIMITATIONS.
 - FAULT TOLERANCE - INCLUDES ERROR DETECTION, DIAGNOSIS, AND ERROR HANDLING; BACKUP AND RECOVERY; RELIABILITY, AND REDUNDANCY.
 - SECURITY - INCLUDES MULTILEVEL SECURITY CONSTRAINTS, SET/USE ACCESS, RESTRICTIONS, BREACH DETECTION AND HANDLING.
 - DISTRIBUTION - GEOGRAPHICAL DISTRIBUTION OF PROCESSING, DATA STORAGE AND ACCESS.
 - ADAPTATION - INCLUDES REQUIREMENTS PERTAINING TO ACCOMMODATION OF DIFFERING LEVELS OF USER EXPERTISE.
 - ASSERTIONS - DESCRIPTION OF A CONTEXT WITHIN WHICH AN ALGORITHM EXISTS. FORMAL ASSERTIONS PROVIDE A MECHANISM FOR DESCRIBING THIS CONTEXT FOR DESIGN EVALUATION PURPOSES.
- PDL MUST BE ABLE TO EXPRESS DEVELOPMENT PROCESS RELATED INFORMATION ...
 - TRACEABILITY - METHOD OF TRACING PRODUCT REQUIREMENTS TO ONE OR MORE DESIGN ELEMENTS. DURING THE DEVELOPMENT OF A SYSTEM, VARIOUS KINDS OF REQUIREMENTS MUST BE RECORDED. IN ADDITION TO FUNCTIONAL REQUIREMENTS, THERE ARE PERFORMANCE, SECURITY, LOGISTICS, AND OTHER REQUIREMENTS. FOLLOWING EACH DESIGN ACTIVITY, EACH REQUIREMENT IS "TRACED" TO ONE OR MORE DESIGN ELEMENTS.
 - ORGANIZATION - INCLUDES DIVISION OF WORK AND ALLOCATION TO TEAM MEMBERS; I.E. WORK BREAKDOWN STRUCTURING.
 - PLANNING - INCLUDES RESOURCE ESTIMATION AND SCHEDULING.
 - TRACKING - INCLUDES DESIGN UNIT COMPLETION, PROJECT PROGRESS, AND PRODUCT VERSION CONTROL.

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

INSTRUCTOR NOTES

THE EXAMPLE, TAKEN FROM THE ADA LANGUAGE REFERENCE MANUAL [LRM], SHOWS HOW PACKAGES, SPECIFICATIONS, BODIES AND STUBS ALL WORK TOGETHER TO DEFER DECISIONS.

REFER BACK TO THE PREVIOUS CHART.

ADA PDL EXAMPLE

(DEFERRED IMPLEMENTATION DECISIONS)

```
procedure TOP is
  type REAL is digits 10;
  R,S: REAL := 1.0;

  package FACILITY is
    P1: constant := 3.14159 26536;
    function F(X: REAL) return REAL;
    procedure G(Y,Z: REAL);
  end FACILITY;

  package body FACILITY is separate;
    -- postpone decision on implementation of F and G
  procedure TRANSFORM(U: in out REAL) is separate;
    -- postpone decision on implementation of TRANSFORM

begin -- TOP
  TRANSFORM(R);
  ...
  FACILITY.G(R,S);
end TOP;
```

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

THIS SLIDE AND THE 6 THAT FOLLOW ALL SHOW SAMPLES OF AN ADA PDL WALKTHRU THEN WITH THE CLASS EMPHASIZING THE READABILITY OF THE DESIGN WHEN EXPRESSED IN ADA.

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

EXPANSION
TO

```

procedure SORT (TABLE : in out TABLE_TYPE) is
  SWAPPED_ITEMS : Boolean;
  TEMP : ITEM_TYPE;
begin
  if TABLE'LENGTH > 1 then
    SWAPPED_ITEMS := TRUE;
    -- loop while the table is not sorted
    while SWAPPED_ITEMS /= FALSE;
      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;

```

IMPLEMENTATION

INSTRUCTOR NOTES

POINT OUT HOW BETWEEN REFINEMENT MORE DETAILED AND SPECIFIC THEN THE DESIGN BECOMES.

AN ADA PDL

(STEP WISE REFINEMENT)

INITIAL DESIGN

procedure EMITTER_IDENTIFICATION is

begin

 // this procedure searches the emitter identification table trying to
 match the characteristics of a newly identified emitter//;
end EMITTER_IDENTIFICATION

REFINEMENT 1

procedure EMITTER_IDENTIFICATION (NEW in : EMITTER; CLASS
 out : EMITTER_CLASS) is

-- this procedure searches the emitter identification table
-- trying to match the characteristics of a newly identified
-- emitter
 // search emitter table comparing on PULSE_WIDTH, PRI, JITTER,
 and FREQUENCY//;
 if // more than one emitter match occurred // then
 // perform pair wise ambiguity resolution//;
 end if

 // set outputs and return//;
end EMITTER_IDENTIFICATION;

INSTRUCTOR NOTES

AN ADA PDL
(STEP-WISE REFINEMENT)

REFINEMENT 2

```

procedure EMITTER_IDENTIFICATION (NEW : In EMITTER;
    CLASS : out EMITTER_CLASS;
    AMBIGUOUS : out boolean)
is
-- this procedure searches the emitter identification table
-- trying to match the characteristics of a newly identified
-- emitter. If an ambiguous match is made, a best guess
-- emitter class is returned as well as an ambiguity
-- flag.
    TYPE EMITTER_TABLE is // collection of EMITTER Objects //;
    EMITTER_IDS : EMITTER_TABLE;
    BEST_BET, CHALLENGER : EMITTER_CLASS;
    EMITTER_MATCH : EMITTER_TABLE;

begin
    -- assume an accurate match
    AMBIGUOUS := TRUE;
    -- search emitter table
    for I in 1.. // # of emitter ID's // loop
        if // PULSE_WIDTH, PRI, JITTER, and FREQUENCY are within
            range of the Ith EMITTER ID's parameters // then
            // add Ith EMITTER_ID to the EMITTER_MATCH table //;
        end if
    end loop;
    --
    -- check for 0, 1, or multiple possible emitter matches
    --

```

INSTRUCTOR NOTES

VG 778.1

21-141

AN ADA PDL

(STEP-WISE REFINEMENT) (Continued)

```
if // # of EMITTER_MATCHES = 0 // then
    // call new emitter handler and issue error status //;
elseif // 1 EMITTER_MATCH // then
    // assign matched_class emitter to CLASS //;
else
    --
    -- perform pair wise ambiguity resolution
    --
    // assign 1st and 2nd EMITTER_MATCH to BEST_BET and
    CHALLENGER //;
    for I in 1..// number of possible EMITTER_MATCHES-1 //
        loop
            RESOLVE_AMBIGUITY (BEST_BET, CHALLENGER, NO_RESOLUTION);
            if NO_RESOLUTION then
                AMBIGUITY := TRUE;
            end if;
            // assign next EMITTER_MATCH to CHALLENGER //;
        end loop;
        // assign the reigning BEST_BET to CLASS as the identified
        emitter //;
    end if;

    end EMITTER_IDENTIFICATION;
```

INSTRUCTOR NOTES

ON ABSTRACT DATA TYPE DESCRIBE THE ROLE OF THE PRIVATE PART OF THE PACKAGE SPECIFICATION.

AN ADA PDL
(CHARACTERIZING DATA)

```

type COLOR is (red, white, blue);
type FREQUENCY is // 0 to 110 MH with a 1 hertz delta//;

subtype RADIO is FREQUENCY //in AM range//;
subtype INDEX is INTEGER //within table size//;

type FM_RADIO is new FREQUENCY //in FM range//;
type HAIR_COLOR is new COLOR;

```

NORMAL
DATA
TYPING

```

package GLOBAL_DATA is
  type FREQUENCY is //500 to 1500 kilocycle with 1 kilocycle
    delta//;
  type MODE is (initialization, integration, acquisition,
    SELF_TEST);
  MODE Switch: MODE;
  ERROR_FLAG: boolean;
end GLOBAL_DATA;

```

DATA
POOLS
OR
COLLECTION

```

Package BANKING is
  type KEY is limited private;
  type DOLLARS is private
  procedure EARN (PAY: in out DOLLARS);
  procedure OPEN (K: out KEY);
  procedure CLOSE (K : in out KEY);
  procedure DEPOSIT (K: in KEY; MONEY: in out DOLLARS);
  procedure WITHDRAW (K: in KEY; MONEY: in out DOLLARS);

private
  type DOLLARS is NATURAL;
  type KEY is access DOLLARS;
end BANKING;

```

ABSTRACT
DATA
TYPE

INSTRUCTOR NOTES

VG 778.1

21-161

AN ADA PDL (EXPRESSING FLOW OF CONTROL)

SELECTION	<pre>-- using AIRCRAFT STATUS as an indicator -- of an attack plane's mode of operation case AIRCRAFT STATUS is when //position is not known// => //do nothing//; when //not busy// => //select a potential target//; ACQUIRE_TARGET (//aircraft, target//); when //target is lost// => //put target on search list//; when //target is located// => STRIKE_TARGET; when others => //indicate status error//; //send warning message to the operator//; end case;</pre>
LOOP, EXIT ON CONDITION	<pre>loop //compare entry with the incoming NAME//; exit when //match occurs//; //get the next entry from the NAME table//; end loop;</pre>
ITERATION ON COUNT	<pre>for I in reverse 1.. //all unverified signals// loop //get a signal//; if //frequency is in friendly range// then IDENTIFY_FRIEND (//the current signal//); else IDENTIFY_FOE (//the current signal//); end if; end loop;</pre>
ITERATION WHILE CONDITION	<pre>while //character is alpha numeric// loop //send a character to the printer//; //construct a character from input stream//; end loop;</pre>

INSTRUCTOR NOTES

POINT OUT THE USE OF ADA EXCEPTION HANDLING FEATURE TO EXPRESS ABNORMAL CONDITION
PROCESSING WITHIN A DESIGN.

AN ADA PDL

(EXPRESSING ALGORITHMS)

{ { }

COMPUTATION
WITH ERROR
HANDLING

```
CRITICAL_CALCULATION
begin
  DIRECTION : = (SIN (X) * COS(Y) * DIRECTION +
    GRADIENT * SIN(Y)) * SPEED;
exception
  when CONSTRAINT_ERROR => RECOMPUTE_DIRECTION;
end CRITICAL_CALCULATION;
```

```
procedure PUSH (E : in FLOAT;
  S : in out STACK) is
```

```
begin
  if //the STACK is full// then
    //report overflow via an exception//;
  else
    //increase the top of stack indicator//;
    //place E on the stack//;
    end if;
  end PUSH;
```

DATA
STRUCTURE
OPERATION

```
function MAXIMUM (FIRST, SECOND : in FLOAT)
  return FLOAT is
```

```
begin
  if //FIRST is greater or even equal to SECOND// then
    return FIRST;
  else
    return SECOND;
  end if;
end if;
```

MATH
FUNCTION

INSTRUCTOR NOTES

AN ADA PDL

(EXPRESSING TIMING)

```
delay 60; -- suspends task for one minute
delay 20 x CYCLES; -- where CYCLE is a constant of
                    type DURATION
```

```
loop
  EXECUTE PASS 1;
  ATTACH DEVICE;
  EXECUTE PASS 2;
  delay .0039062-MY_TIME; -- where MY_TIME is
                          -- execution time of
                          -- this code loop. Thus, this
                          -- loop executes 256 times per second
end loop;
```

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 PDLs ARE NOT EXPRESSIVE
ENOUGH FOR REAL TIME DESIGN

INSTRUCTOR NOTES

THEME: OTHER METHODS ARE AVAILABLE AS ALTERNATIVES TO FLOW CHARTS TO EXPRESS
DETAIL DESIGNS.

PURPOSE: A QUICK OVERVIEW OF SEVERAL METHODS.

REFERENCES: KATZAN, H. "SYSTEMS DESIGN AND DOCUMENTATION: AN INTRODUCTION TO THE HIPO
METHOD" VAN NOSTRAND REINHOLD, NY; 1976

STAY, J. "HIPO AND INTEGRATED PROGRAM DESIGN" IBM SYSTEMS JOURNAL VOL. 15,
NO. 2; 1976

NASSI, I., SHNEIDERMAN, B. "FLOWCHART TECHNIQUES FOR STRUCTURED
PROGRAMMING" SIGPLAN VOL. 8, NO. 8; AUGUST 1973

MARCA, D. "A METHOD FOR SPECIFYING STRUCTURED PROGRAMS" SIGSOFT VOL. 4, NO.
3; JULY 1979

Section 22
GRAPHICAL DETAILED DESIGN METHODS -
HIPO TOOL AND NSSF TOOL

INSTRUCTOR NOTES

HIERARCHICAL INPUT PROCESSING OUTPUT (HIPO) WAS CREATED BY IBM TO DOCUMENT SOFTWARE MORE OR LESS AFTER THE FACT.

"PACKAGE" IS A HIPO KEYWORD. DON'T CONFUSE WITH Ada PACKAGE.

HIPO IS ORIENTED TOWARDS GIVING A TOP DOWN, STEP BY STEP 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.

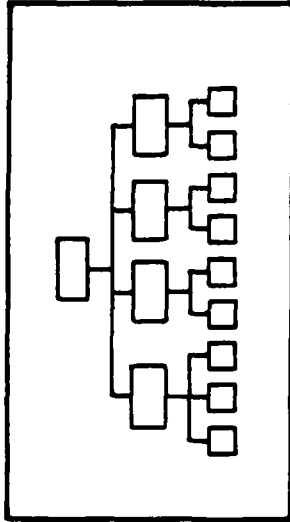
INSTRUCTOR NOTES

- NOTE TREE STRUCTURE OF VTOC
- NOTE SKETCHINESS OF OVERVIEW DIAGRAMS
- NOTE DETAIL OF DIAGRAMS

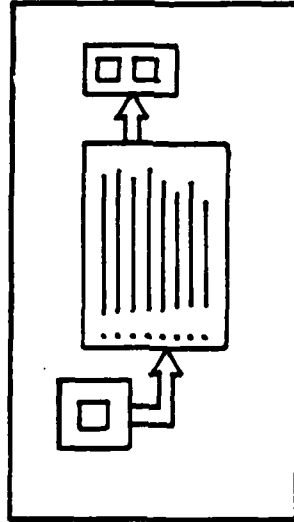
OVERVIEW DIAGRAMS SHOW CREATIONS AND CONSUMPTIONS OF FILES, DETAIL DIAGRAMS, SHOW RECORDS OR FIELDS.

A HIPO PACKAGE

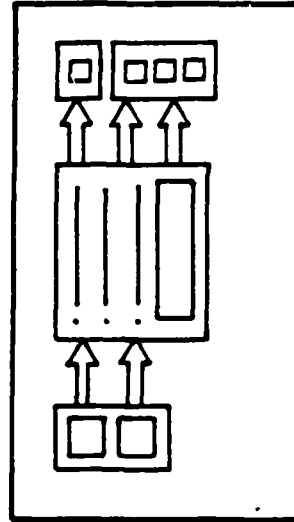
- 1 A VISUAL TABLE OF CONTENTS
(HIERARCHY)



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



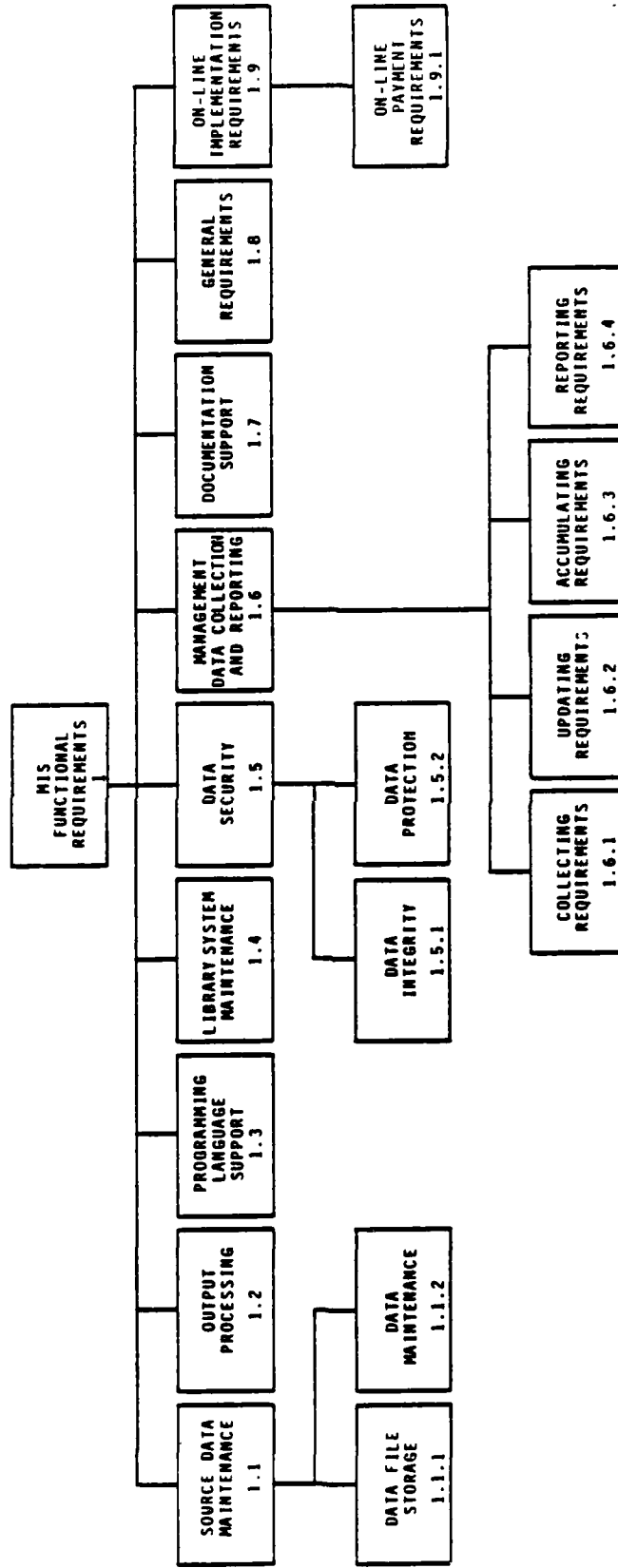
- 3 DETAIL DIAGRAMS
(FOR LOW LEVELS IN THE VTOC)



INSTRUCTOR NOTES

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

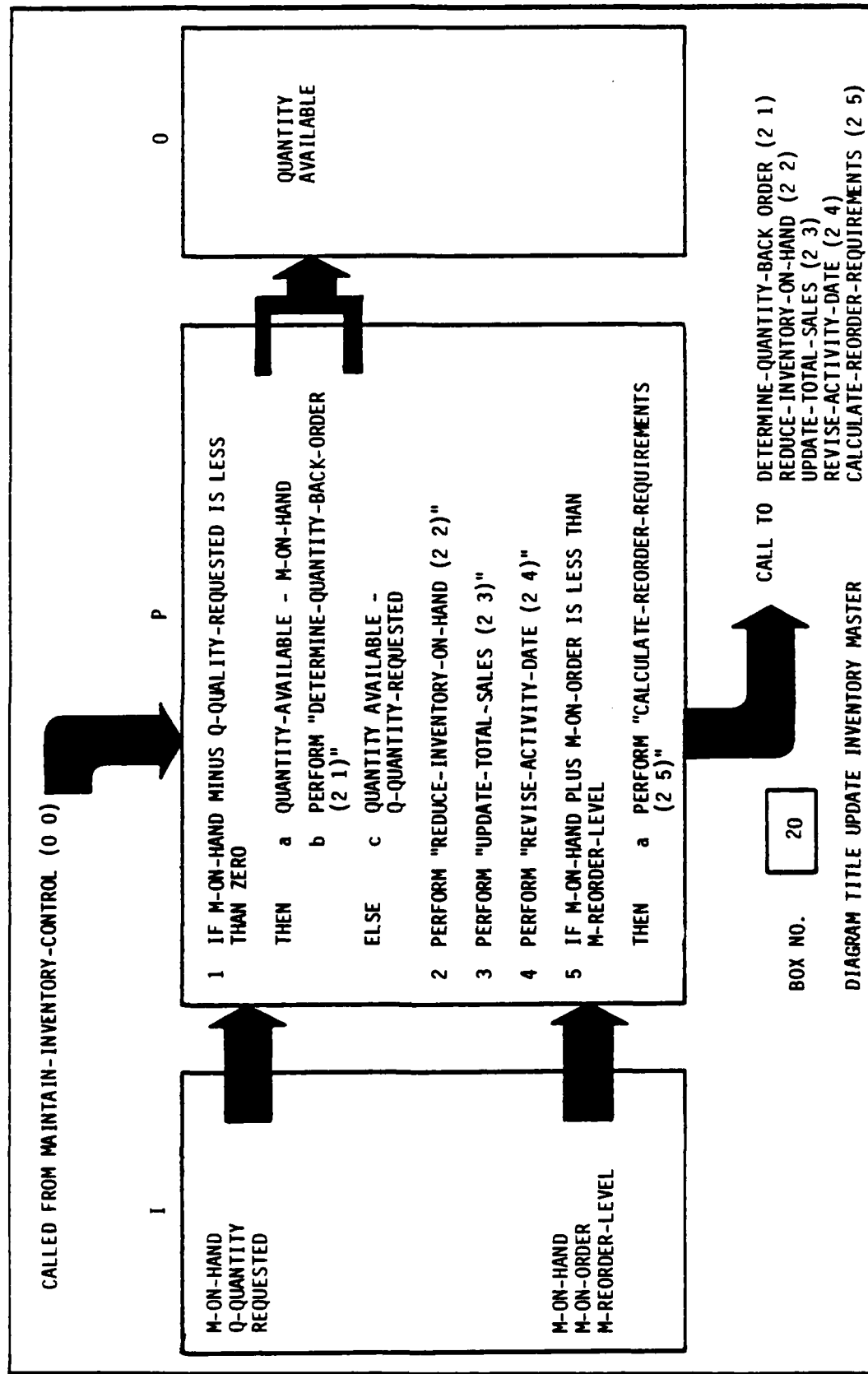
VISUAL TABLE OF CONTENTS



INSTRUCTOR NOTES

- NOTE THE "CALLED FROM" AND "CALL TO" ARROWS OF THE PROCESS BOX.
- NOTE THE SKETCHINESS OF INPUTS AND OUTPUTS.

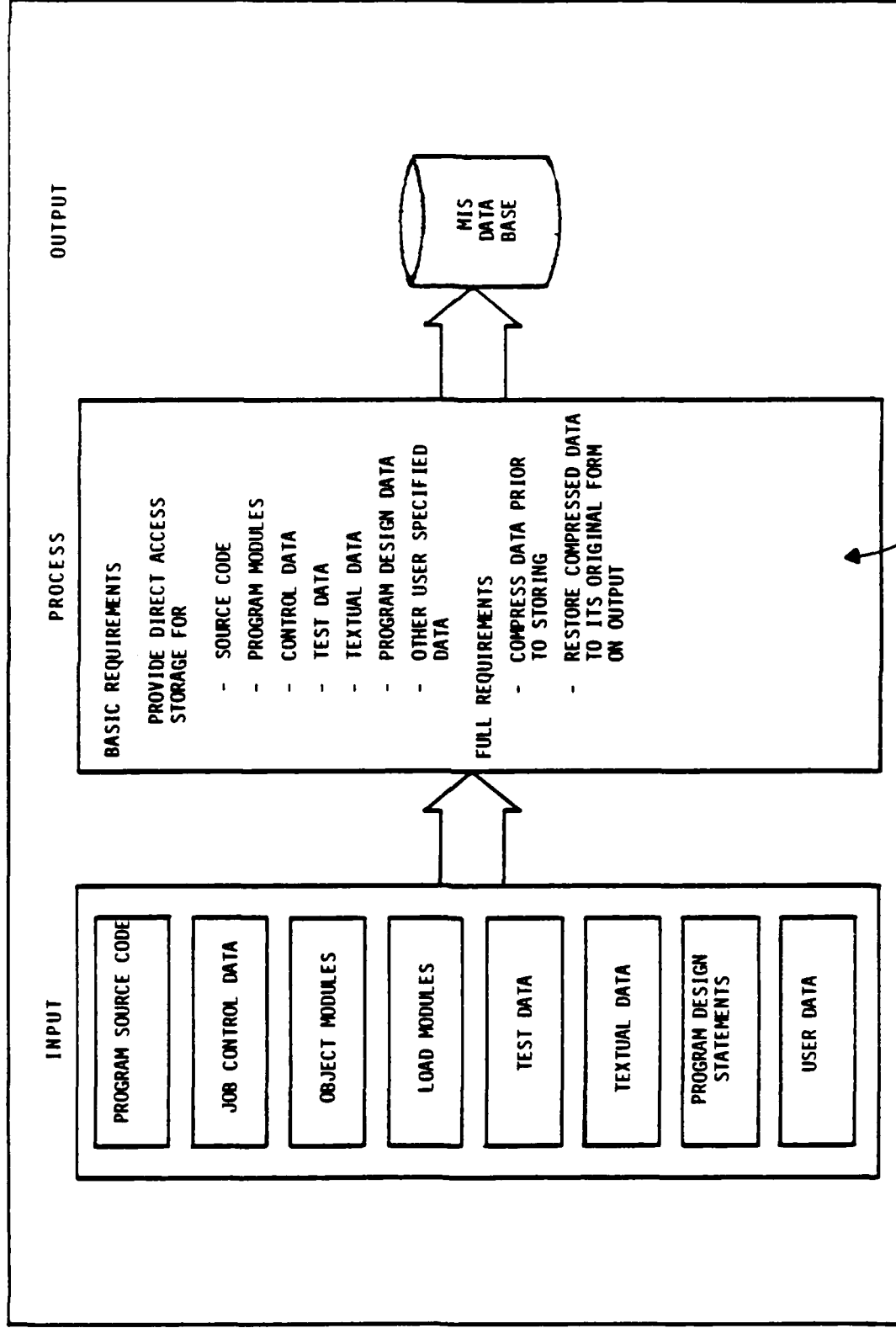
A DETAILED DIAGRAM



INSTRUCTOR NOTES

- NOTE THE DETAIL HERE, WHERE ESPECIALLY DATA IS BROKEN INTO ITS CONSTITUENT AND PHYSICAL PARTS.
- AT THIS LEVEL: FILES, FORMS AND RECORDS APPEAR.

AN OVERVIEW DIAGRAM



SOME PEOPLE USE A PDL HERE

INSTRUCTOR NOTES

GO OVER QUICKLY - POINT OUT HIGHLIGHTS.

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 SEE10: 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 EXTM 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, EXIMMAIN LOCAL CALLS, EXIMPITL	

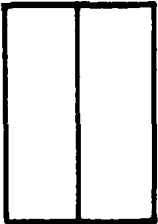
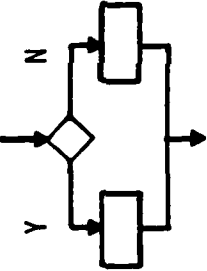
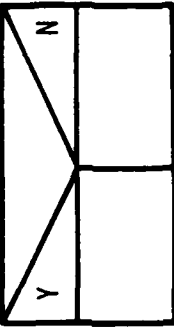
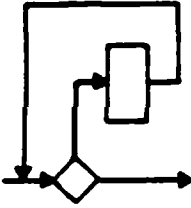
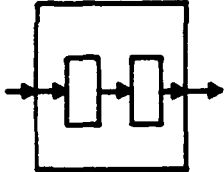
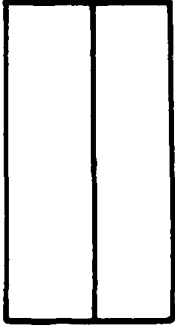
NSSF OVERVIEW

- 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

COMPARE NSSF TO CONVENTIONAL FLOWCHART GRAPHICS.

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

NSSF GRAPHICS
(REPRESENTING PROGRAM STRUCTURE)

STRUCTURE	FLOWCHART GRAPHICS	NSSF GRAPHICS
SEQUENCE		
SELECTION		
REPETITION		
HIERARCHY		

INSTRUCTOR NOTES

- TELL CLASS THAT NO ROOM TO WRITE MEANS DECOMPOSITION IS REQUIRED. THIS IS AN EXAMPLE OF A COMPLEXITY METRIC FEW TOOLS OR METHODS HAVE.

CROWDED GRAPHICS INDICATES
TOO MUCH COMPLEXITY --
TIME TO DECOMPOSE INTO A
NEW MODULE.

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

THEME: TO PUT ALL OF THE DESIGN METHODS INTO PERSPECTIVE.

PURPOSE: TO PROVIDE A COMPARISON BETWEEN THE VARIOUS METHODS TO AID IN SELECTION OF
A SET OF DESIGN METHODS.

REFERENCE: METHODMAN - SEE ANALYSIS WRAP-UP FOR REFERENCE.

Section 23

DESIGN WRAP-UP

VG 778.1

DESIGN PHASE COVERAGE

DESIGN PHASE

TRANSLATION OF REQUIREMENTS TO ARCHITECTURAL DESIGN	- ARCHITECTURAL DESIGN AND ASSESSMENT - MODULAR STRUCTURE OF THE DESIGN	- DETAILED DESIGN - INSIDES OF THE MODULES
1	2	3

EMPHASIS

SCRIP			
OBJECT-ORIENTED DESIGN			1 2
STRUCTURED DESIGN			2
JACKSON DESIGN			2
WARNIER-ORR			2
HOS			1 2 3
HIPO			3
NSSF			3
PDL			3
METRICS			2

INSTRUCTOR NOTES

THESE ARE THE DEFINITIONS OF DESIGN METHODS CHARACTERISTICS TO BE USED ON THE NEXT SLIDE. GO OVER THESE QUICKLY, BUT RELATE ONE OR TWO TO SPECIFIC TECHNIQUES BEFORE GOING TO THE CHART.

TECHNICAL CHARACTERISTICS (DESIGN)

- FUNCTION HIERARCHY - A SITUATION CAN BE REPRESENTED IN WHICH A FUNCTION AT ONE LEVEL IS ACTUALLY COMPOSED OF SEVERAL INTERCONNECTED FUNCTIONS THAT EXIST AT A LEVEL OF GREATER DETAIL.
- DATA HIERARCHY - THE CONCEPT OF DATA CLASSES; A SITUATION CAN BE REPRESENTED IN WHICH A SET OF DATA AT ONE LEVEL IS ACTUALLY COMPOSED OF SEVERAL INTERRELATED PIECES OF DATA THAT EXIST AT A LEVEL OF GREATER DETAIL.
- INTERFACES - THE CONCEPT OF HAVING DISTINCT AND WELL-DEFINED BOUNDARIES BETWEEN PROCESSES OR SETS OF DATA. SOFTWARE SYSTEMS AND THE LARGE INFORMATION SYSTEMS OF WHICH THEY ARE A PART SHOULD CONTAIN MANY DISTINCT PARTS. EACH PART SHOULD HAVE A CLEAR DEFINITION SO THAT IT CAN BE DEALT WITH AS A SEPARABLE UNIT. IF THIS IS THE CASE, THEN WE HAVE INTERFACES, OR CONNECTIONS, BETWEEN THE VARIOUS PARTS.
- CONTROL FLOW - REPRESENTATION OF THE SEQUENCE IN WHICH PROCESSES WILL TAKE PLACE.
- DATA FLOW - REPRESENTATION OF THE FLOW OF INFORMATION TYPES BETWEEN VARIOUS PROCESSING ELEMENTS AND/OR STORAGE ELEMENTS IN THE SYSTEM.
- DATA ABSTRACTION - THE CONCEPT OF HIDING INFORMATION ABOUT THE IMPLEMENTATION OF A DATA TYPE AND PROVIDING A SET OF IMPLEMENTATION-INDEPENDENT FUNCTIONS FOR USE OF THE DATA TYPE.
- PROCEDURAL ABSTRACTION - AN ALGORITHM FOR CARRYING OUT SOME OPERATION IS ABSTRACTED TO A SINGLE NAME THAT CAN BE USED TO INVOKE A PROCEDURE WITHOUT KNOWING THE DETAILS OF ITS IMPLEMENTATION. (TRANSACTIONS ARE AN INSTANCE OF THIS CASE.)
- PARALLELISM - A SITUATION IN WHICH TWO OR MORE COOPERATING SEQUENTIAL PROCESSES ARE CONCURRENTLY IN EXECUTION.
- SAFETY - THE AVOIDANCE OF RUNTIME FAILURES WHICH COULD LEAD TO THE LOSS OF LIFE OR THE OCCURRENCE OF OTHER CATASTROPHIC CONSEQUENCES.
- RELIABILITY - THE ABSENCE OF ERRORS THAT LEAD TO SYSTEM FAILURE.
- CORRECTNESS - FIDELITY TO HIGHER LEVEL SPECIFICATIONS.

INSTRUCTOR NOTES

PICK ONE OR TWO ROWS, THEN COLUMNS, AND INVESTIGATE IN DETAIL.

TECHNICAL CHARACTERISTICS

	FUNCTION HIERARCHY	DATA HIERARCHY	INTERFACES	CONTROL FLOW	DATA FLOW	DATA ABSTRACTION	PROCEDURAL ABSTRACTION	PARALLELISM	CORRECTNESS
SCRIP	M	M	H	H	M	H	H	H	H
OBJECT-ORIENTED	L	H	H	L	L	H	L	M	L
STRUCTURED DESIGN	H	L	M	M	M	L	H	L	L
JACKSON DESIGN	M	H	M	M	M	M	M	M	L
WARNIER-ORR	M	M	L	M	M	L	L	L	L
HOS	H	L	M	M	M	M	M	M	H
HIPO	M	L	H	M	L	L	M	L	L
NSSF	L	L	L	H	L	L	M	L	L
PDL	M	M	H	H	M	H	H	H	M
METRICS									H

LEVEL OF SUPPORT

H - HIGH

M - MODERATE

L - LOW

BLANK - NO SUPPORT

INSTRUCTOR NOTES

AGAIN, GO THROUGH THESE QUICKLY, AND PICK ONE OR TWO AND THE APPROPRIATE TECHNIQUES TO ILLUSTRATE AND MOTIVATE THE NEXT SLIDE.

USAGE CHARACTERISTICS (DESIGN)

- UNDERSTANDABILITY - HOW EASY IT IS FOR SOMEONE WHO IS INTERESTED IN THE SYSTEM BEING DEVELOPED, BUT NOT ESPECIALLY KNOWLEDGEABLE IN THE TECHNIQUE, TO UNDERSTAND THE RESULTS OF THE DEVELOPMENT.
- TRANSFERABILITY - THE DEGREE TO WHICH THE METHOD OR TOOL CAN BE SUCCESSFULLY TAUGHT TO PERSONNEL NOT PREVIOUSLY FAMILIAR WITH IT. THIS INCLUDES NOT ONLY HOW EASY IT IS TO TEACH AND LEARN, BUT ALSO HOW WELL FORMULATED IT IS.
- REUSABILITY - THE EASE WITH WHICH PREVIOUSLY CREATED DESIGNS, CODE, OR OTHER WORK PRODUCTS CAN BE REUSED IN A NEW PROJECT.
- COMPUTER SUPPORT - EXISTENCE OF AUTOMATED TOOLS WHICH CAN BE EASILY OBTAINED AND WHICH AID IN THE USE OF THE METHODOLOGY OR SOME OF ITS STEPS.
- LIFE-CYCLE RANGE - THE SPAN OF PHASES IN THE DEVELOPMENT LIFE-CYCLE OVER WHICH THE TOOLS OR METHOD CAN BE USEFULLY APPLIED. THIS MUST BY NECESSITY BE AN APPROXIMATE MEASURE.
- TASK RANGE - THE SPAN OF TASKS TO WHICH THE ITEM MAY BE USEFULLY APPLIED. TRADITIONAL CLASSIFICATIONS OF SOFTWARE APPLICATIONS (BUSINESS, SCIENTIFIC, SYSTEMS, ETC.) ARE NOT VERY USEFUL FOR THIS EVALUATION. THERE ARE SCIENTIFIC AND BUSINESS PROBLEMS THAT SHARE THE SAME PROBLEMS WHILE THERE ARE OTHER TASKS THAT SAY, WITHIN THE BUSINESS CATEGORY, ARE QUITE DISSIMILAR.
- COHESIVENESS - THE EXTENT TO WHICH THE TECHNICAL METHODS, MANAGEMENT PROCEDURES, AND AUTOMATED TOOLS MAY BE COMBINED TO SUPPORT THE METHODOLOGY.
- EXTENT OF USAGE - A JUDGEMENT AS TO HOW WIDESPREAD THE CURRENT USAGE OF THE METHODOLOGY IS.

INSTRUCTOR NOTES

VG 778.1

23-51

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

USAGE CHARACTERISTICS
(DESIGN) - continued

- EASE OF PHASE TRANSITION - THE EXTENT TO WHICH INFORMATION DEVELOPED AT ONE PHASE OF DEVELOPMENT SUPPORTS WORK TO BE DONE AT A SUBSEQUENT PHASE (E.G., FROM ANALYSIS TO DESIGN).
- DECISION HIGHLIGHTING - THE ABILITY OF A TECHNIQUE TO MAKE VISIBLE AND HIGHLIGHT KEY TECHNICAL OR PROJECT DECISIONS (E.G., CHOICE OF DATA STRUCTURES, NEARNESS TO COMPLETION OF A PROJECT); THE ABILITY OF A TECHNIQUE TO ILLUMINATE THE CONSEQUENCES OF A DEVELOPMENT DECISION.
- VALIDATION - THE EXTENT TO WHICH THE METHODOLOGY ASSISTS IN THE DETERMINATION OF SYSTEM CORRECTNESS.
- REPEATABILITY - THE EXTENT TO WHICH SIMILAR RESULTS ARE OBTAINED WHEN THE METHODOLOGY (OR AN INCLUDED ASPECT) IS APPLIED MORE THAN ONCE TO THE SAME PROBLEM.
- EASE OF CHANGE TO WORK PRODUCTS - THE AMOUNT OF EFFORT REQUIRED TO MODIFY A WORK PRODUCT WHEN SOME ASPECT OF REQUIREMENTS, SPECIFICATION, OR DESIGN IS CHANGED.

INSTRUCTOR NOTES

PICK ONE OR TWO ROWS, THEN COLUMNS, AND INVESTIGATE IN DETAIL.

NOTE, THIS CHART EMPHASIZES SUITABILITY.

USAGE CHARACTERISTICS

	EMBEDDED SYSTEMS	SCIENTIFIC/ ENGINEERING	OPERATING SYSTEMS	TOOLS	BUSINESS	ARTIFICIAL INTELLIGENCE	SMALL SIZE	MEDIUM SIZE	LARGE SIZE
SCRIP	W	W	W	S	S	W	I	S	W
OBJECT-ORIENTED	S	S	S	S	I	S	W	S	I
STRUCTURED DESIGN	S	S	S	S	W	S	S	W	S
JACKSON DESIGN	W	S	S	S	W	S	S	W	W
WARNIER-ORR	W	S	S	S	W	S	S	W	I
HOS	W	W	W	S	W	S	I	W	W
HIPO	S	S	S	S	S	S	S	W	W
NSSF	S	S	S	S	S	S	W	S	I
PDL	W	W	S	S	S	S	S	W	W
METRICS	S	S	S	S	S	S	W	W	S

LEGEND
 W - WELL SUITED
 S - SATISFACTORY
 I - INAPPROPRIATE

INSTRUCTOR NOTES

NOTE, THIS CHART EMPHASIZES LEVEL OF SUPPORT.

USAGE CHARACTERISTICS

	UNDERSTAND- ABILITY	TRANSFERABILITY	REUSABILITY	COMPUTER SUPPORT	EXTENT OF USAGE	EASE OF PHASE TRANSITION	DECISION HIGHLIGHTING	VALIDATION	REPEATABILITY	EASE OF CHANGE OF WORK PRODUCTS
SCRIP	M	M	H	M	M	H	H	M	H	M
OBJECT-ORIENTED	H	H	H	M	M	L	H	M	H	M
STRUCTURED DESIGN	H	M	M	L	H	M	M	M	M	L
JACKSON DESIGN	M	M	M	M	H	H	M	M	M	L
WARNIER-ORR	M	M	M	L	M	L	M	M	M	L
HOS	M	M	H	H	M	H	H	H	L	L
HIPO	M	M	M	M	H	M	L	L	L	M
NSSF	M	M	M	L	L	L	L	L	L	L
PDL	H	M	M	H	H	M	H	M	M	M
METRICS	M	M		L	L	L		H		

LEVEL OF SUPPORT
H - HIGH
M - MODERATE
L - LOW
BLANK - NO SUPPORT

INSTRUCTOR NOTES

THIS IS AN AREA OVERLOOKED, BUT IS IMPORTANT FOR THE EVENTUAL SUCCESS OF A METHODOLOGY.

MANAGEMENT CHARACTERISTICS (DESIGN)

- MANAGEABILITY - THE DEGREE TO WHICH THE METHOD OR TOOL PERMITS STANDARD MANAGEMENT TECHNIQUES OF ESTIMATION, IN-PROCESS STATUS CHECKING AND CONTROL TO BE APPLIED. THE EVALUATIONS ARE BASED ON THE EXISTENCE OF WELL-DEFINED STEPS AND INTERMEDIATE PRODUCTS WHICH MAKE MANAGEMENT POSSIBLE AND THE EXISTENCE OF MANAGEMENT TECHNIQUES APPLYING SPECIFICALLY TO THE METHOD AT HAND.
- TEAMWORK - THE EXTENT TO WHICH THE METHODOLOGY AND THE DEVELOPMENT ENVIRONMENT AID, RATHER THAN HINDER, TEAMWORK.
- PHASE DEFINITIONS - THE IDENTIFICATION WITHIN THE METHODOLOGY OF DEVELOPMENT PHASES THAT REPRESENT INTERMEDIATE STAGES OF THE DEVELOPMENT PROCESS.
- WORK PRODUCTS - THE DOCUMENTS AND SYSTEMS THAT RESULT FROM APPLICATION OF THE METHODOLOGY.
- CONFIGURATION MANAGEMENT - THE WAY IN WHICH THE METHODOLOGY AND/OR ITS TOOLS PROVIDES FOR ORGANIZATION, TRACKING, AND MAINTENANCE OF THE EMERGING WORK PRODUCTS, INCLUDING CONTROL OF RELEASES AND MULTIPLE VERSIONS.
- EXIT CRITERIA - THE WAY IN WHICH PHASES OF DEVELOPMENT AND WORK PRODUCTS ARE DEFINED TO PROVIDE EXPLICIT STOPPING OR EXIT CRITERIA FOR EACH STAGE OF DEVELOPMENT.
- SCHEDULING - THE METHODOLOGICAL SUPPORT FOR PROJECT SCHEDULING.
- COST ESTIMATION - THE METHODOLOGICAL SUPPORT FOR COST ESTIMATION.

INSTRUCTOR NOTES

TRY TO GET THE STUDENTS TO ANSWER "HOW DO THESE TECHNIQUES HELP MANAGEMENT OF A PROJECT?"

VG 778.1

23-9i

MANAGEMENT CHARACTERISTICS

	MANAGEABILITY	TEAMWORK	PHASE DEFINITION	CONFIGURATION MANAGEMENT	EXIT CRITERIA
SCRIP	M	M	M	M	H
OBJECT-ORIENTED	L	M	L	L	L
STRUCTURED DESIGN	M	L	M	L	M
JACKSON DESIGN	M	M	M	M	M
WARNIER-ORR	M	M	M	M	M
HOS	M	M	M	M	H
HIPO	M	L	L	M	M
NSSF	L	L	L	L	L
PDL	M	M	M	M	H
METRICS					

LEVEL OF SUPPORT

H - HIGH

M - MODERATE

L - LOW

BLANK - NO SUPPORT

INSTRUCTOR NOTES

THIS CHART SHOWS THE TECHNIQUES SUPPORT Ada CONCEPTS AND FEATURES.

SUPPORT OF Ada UNDERLYING CONCEPTS AND FEATURES

	USE OF STRUCTURED CONSTRUCTS	STRONG DATA TYPING	INFORMATION HIDING	DATA ABSTRACTION	CONCURRENT PROGRAMMING	EXCEPTION HANDLING	GENERIC FACILITIES	MACHINE DEPENDENCY FACILITIES	PACKAGES	ABSOLUTE AND RELATIVE PRECISION	ORIENTED TOWARD EMBEDDED SYSTEMS
SCRP	M	H	H	H	H	H	M	H	H	M	I
OBJECT-ORIENTED	M	H	H	H	M	M	L	M	H	M	I
STRUCTURED DESIGN	M	L	L	L	L	L	L	L	L	L	M
JACKSON DESIGN	M	M	L	M	M	L	L	L	L	L	M
WARNIER-ORR	M	M	L	M	M	L	L	L	L	L	M
HOS	H	M	L	L	M	L	M	L	M	L	M
HIPO	M	L	L	L	L	M	L	L	L	M	M
NSSF	M	L	L	L	L	L	L	L	L	L	M
PDL	H	H	H	H	H	H	H	H	H	H	I

LEVEL OF SUPPORT

H - HIGH
M - MODERATE
L - LOW
BLANK - NO SUPPORT

INSTRUCTOR NOTES

ENTERTAIN QUESTIONS, BUT DON'T GET STUCK ON DETAILS. KEEP TALK ON A MORE GENERAL LEVEL
RATHER THAN SPECIFIC.

REMINDERS

- ITERATE - WORK IN THE GOOD IDEAS, WORK OUT THE BAD
- ABSTRACT - SEPARATE SIMILARITIES AND DIFFERENCES
- MECHANIZE - BUILD WORKHORSES
- DECOMPOSE - BREAK INTO MODULES CONSISTENTLY
- SYNTHESIZE - BRING TOGETHER SIMILAR FUNCTIONS AND STRUCTURES
- USE METRICS - TO EVALUATE THE QUALITY OF YOUR DESIGN
- DOCUMENT - EVERYTHING!

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

SECTION 24

IMPLEMENTATION INTRODUCTION

VG 778.1

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.

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 TRASHING 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.

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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

IMPLEMENTATION SCOPE

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

- STEP-WISE PROGRAM COMPOSITION

- PROGRAM FAMILIES

- DATA ABSTRACTIONS AND TYPES

- DATA STRUCTURES

- FUNDAMENTAL ALGORITHMS

- TIME/SPACE TRADEOFFS

- ACCEPTABILITY VS. CORRECTNESS

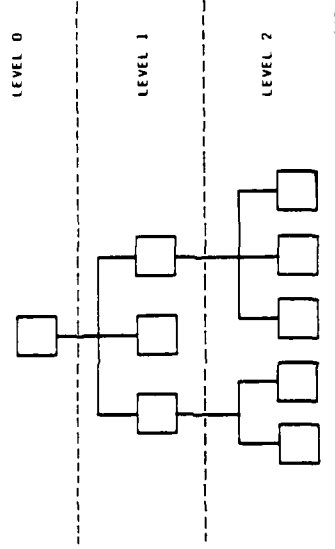
INSTRUCTOR NOTES

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 COMPOSITION (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.

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.

NO-A165 302

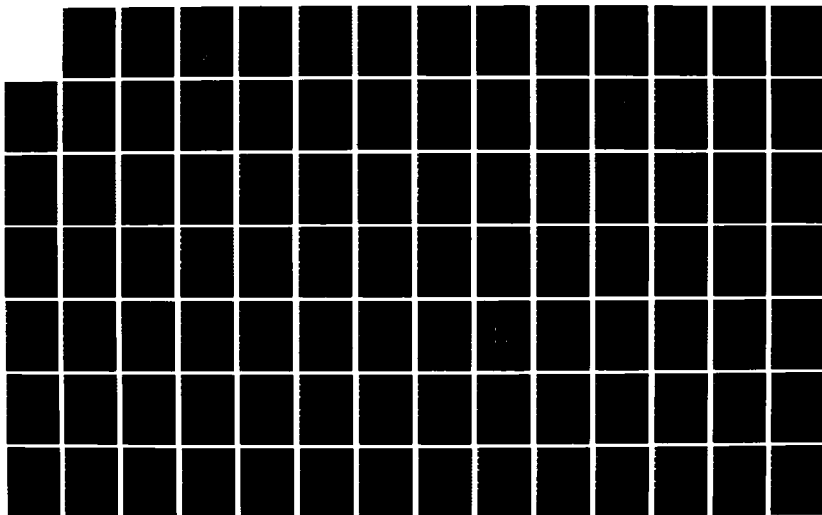
ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 3
(U) SOFTECH INC WALTHAM MA 1986 DAB807-83-C-K506

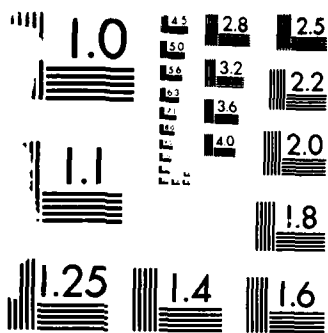
275

UNCLASSIFIED

F/G 5/9

NL

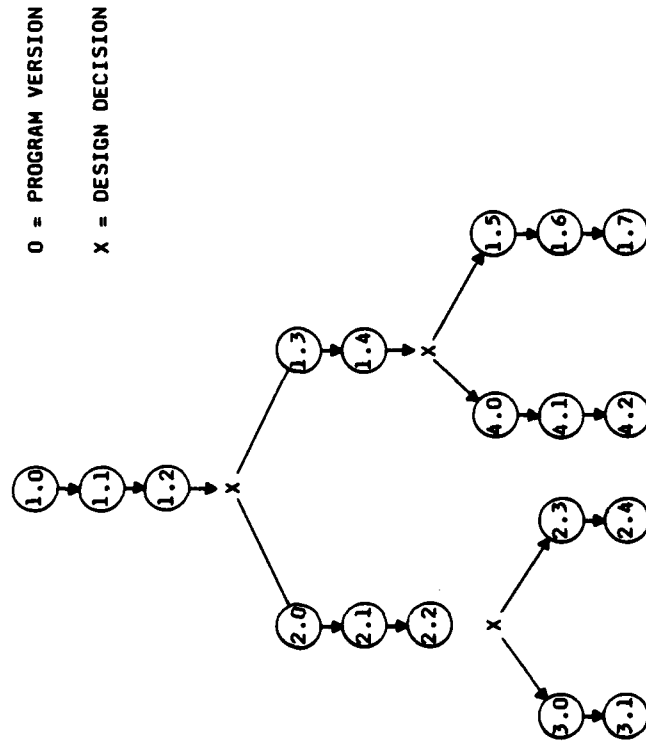




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

IMPLEMENTATION ISSUES (PROGRAM FAMILIES)

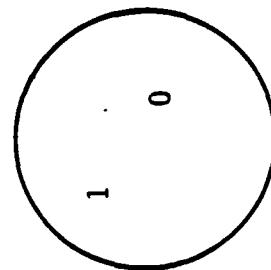
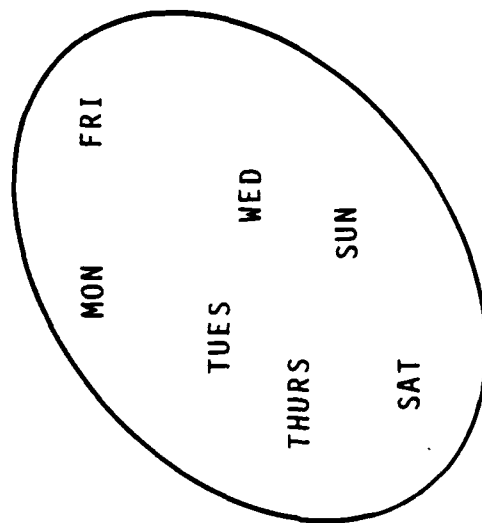
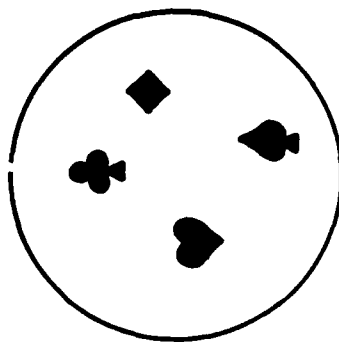
- PROGRAM FAMILIES WAS A CONCEPT THAT STATED THAT A PROGRAM EVOLVES INTO A SET OF RELATED PROGRAMS:



- VERY FEW HAVE WORRIED ABOUT PROGRAM FAMILIES. BUT TODAY'S CONCERNS ABOUT THE EVOLUTION OF INDIVIDUAL MODULES AS THEY SIMULTANEOUSLY PARTICIPATE IN SEVERAL SOFTWARE PRODUCTS BRINGS THIS ISSUE AGAIN INTO CENTER STAGE.

INSTRUCTOR NOTES

A PICTURE OF THESE TYPES COULD BE:



BOOLEAN

IMPLEMENTATION ISSUES
(DATA ABSTRACTIONS AND TYPES)

- DATA ABSTRACTIONS AND TYPES CATEGORIZED 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.

IMPLEMENTATION ISSUES

(DATA STRUCTURES)

- DATA STRUCTURES CONCERNS ITSELF WITH ORGANIZING 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.

IMPLEMENTATION ISSUES

(FUNDAMENTAL ALGORITHMS)

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

- FIBONACCI SEQUENCE GENERATION

- FACTORIAL COMPUTATION

- MATRIX MULTIPLICATION

- LINKED-LIST ALLOCATION

- TREE MANIPULATION

- MONITORS

- INSERTION SORT

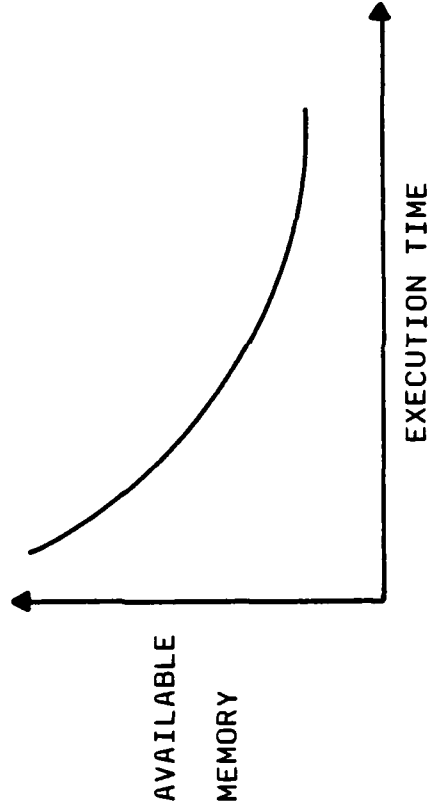
- QUICK SORT

INSTRUCTOR NOTES

AS AVAILABLE MEMORY DECREASES, PROGRAM EXECUTION SPEED DROPS. THEY ARE PROPORTIONATELY RELATED TO EACH OTHER.

IMPLEMENTATION ISSUES
(TIME/SPACE TRADEOFFS)

- THE TIME/SPACE TRADEOFF MADE CLEAR THAT MORE MEMORY USAGE MEANT FASTER PROGRAM EXECUTION, AND LESS MEMORY USAGE MEANT PROGRAMS WORKED MORE SLOWLY. THIS HAS ALWAYS REMAINED TRUE.



INSTRUCTOR NOTES

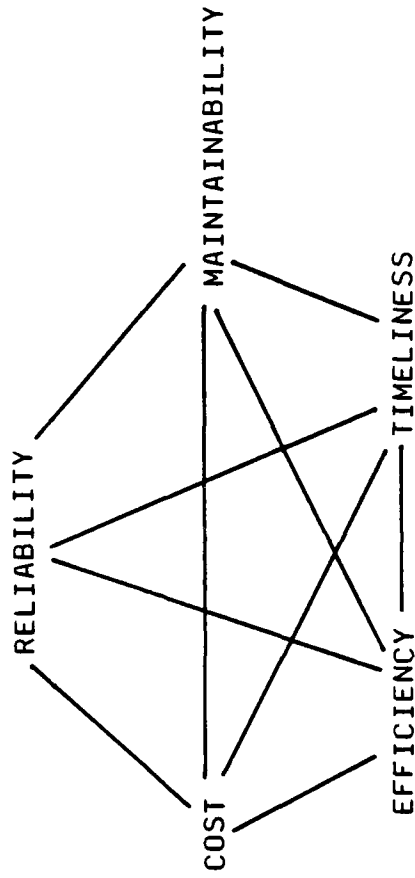
SPEND SOME TIME ON THIS SLIDE. IT IS IMPORTANT.

TALK ABOUT THE GUIDANCE AVAILABLE TODAY (E.G. ON SOFTWARE COST MODELS) TO MAKE THESE
TRADE-OFFs.

IMPLEMENTATION ISSUES

(EFFICIENCY)

- BUT MANY HAVE TAKEN THE CONCEPT AND MADE EFFICIENCY OF PARAMOUNT IMPORTANCE. THIS IS FALLACIOUS, FOR EFFICIENCY IS JUST ANOTHER ENGINEER GOAL. IT MUST BE EVALUATED WITH, AND TRADEOFF AMONG, THE OTHER SOFTWARE ENGINEERING GOALS:

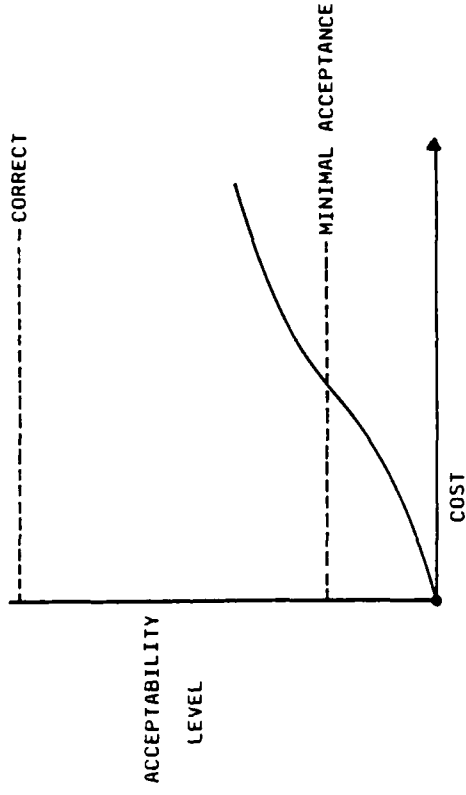


INSTRUCTOR NOTES

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

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.

DOD-STD-SDS VIEW OF IMPLEMENTATION

- SDS IMPLEMENTATION ACTIVITIES

- UNITS ARE CODED IN A SYSTEMATIC, TOP-DOWN HIERARCHICAL SEQUENCE
- MUST CODE UNITS IN ACCORDANCE 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 INTEGRATE ITEM IN ACCORDANCE WITH SOFTWARE TEST PLAN

INSTRUCTOR NOTES

NOTICE THE FORMAL REVIEW CYCLE.

000-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?"

CODING STANDARDS

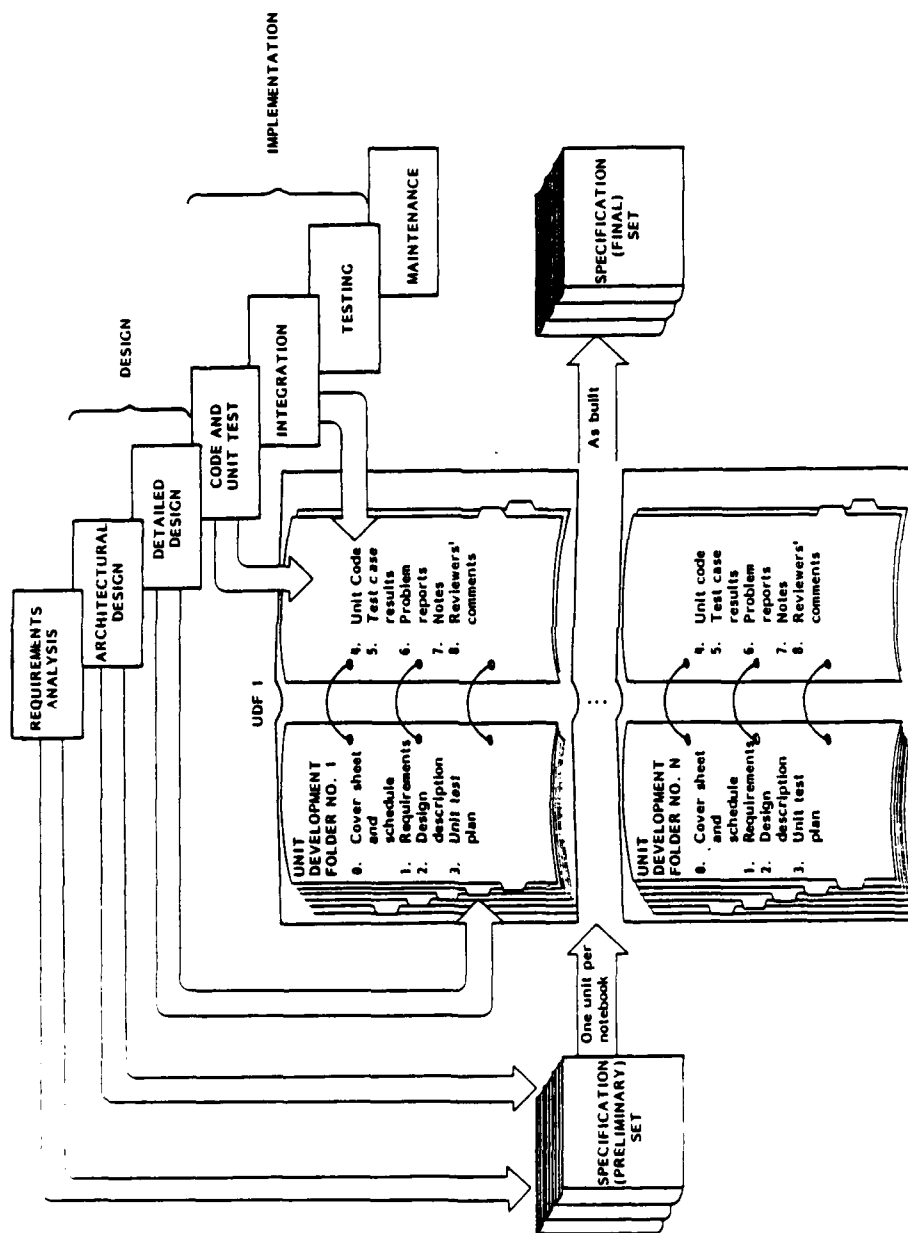
(CONTINUED)

- 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.

UNIT DEVELOPMENT FOLDERS



• PROVIDES A UNIFORM AND VISIBLE COLLECTION POINT FOR ALL UNIT INFORMATION

SOURCE: BEOHM, SOFTWARE ENGINEERING ECONOMICS, 1983

VG 778.1

24-15

INSTRUCTOR NOTES

THIS IS THE SET-UP OR VIEW WHICH WILL BE TAKEN FOR THE UPCOMING SECTIONS.

IMPLEMENTATION PERSPECTIVES AND FORMATS

- PRIMARY PERSPECTIVES CONSIDERED DURING IMPLEMENTATION
 - CODING STRATEGIES
 - COMPLEXITY MANAGEMENT
 - QUALITY ASSURANCE
 - TESTING STRATEGIES
- PRIMARY FORMATS
 - GUIDELINES
 - MEETING-ORIENTED
 - PROCEDURAL

INSTRUCTOR NOTES

TRY TO REFER BACK TO THESE PERSPECTIVES AND FORMATS WHILE TEACHING THE OTHER SECTIONS.

IMPLEMENTATION METHODOLOGIES TO BE PRESENTED

<u>METHODOLOGY</u>	<u>PERSPECTIVE</u>	<u>FORMAT</u>
• STRUCTURED PROGRAMMING	• CODING STRATEGIES	• GUIDELINES
• PROGRAM COMPLEXITY	• COMPLEXITY MANAGEMENT	• GUIDELINES
• PROGRAM CORRECTNESS	• QUALITY ASSURANCE	• PROCEDURAL
• TESTING	• TESTING STRATEGIES	• GUIDELINES • PROCEDURAL
• REVIEW	• QUALITY ASSURANCE	• MEETING ORIENTED • GUIDELINES

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 25
STRUCTURED PROGRAMMING

VG 778.1

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.

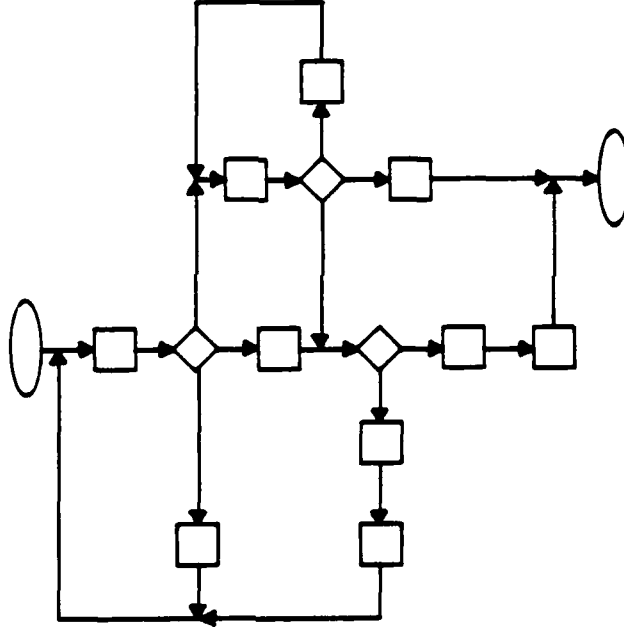
- STRUCTURED PROGRAMMING IS A COLLECTION OF TECHNIQUES WHICH EVOLVED TO ANSWER CONCERNS ABOUT THE PROGRAMS WHICH PEOPLE WRITE.

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 EDSGER 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

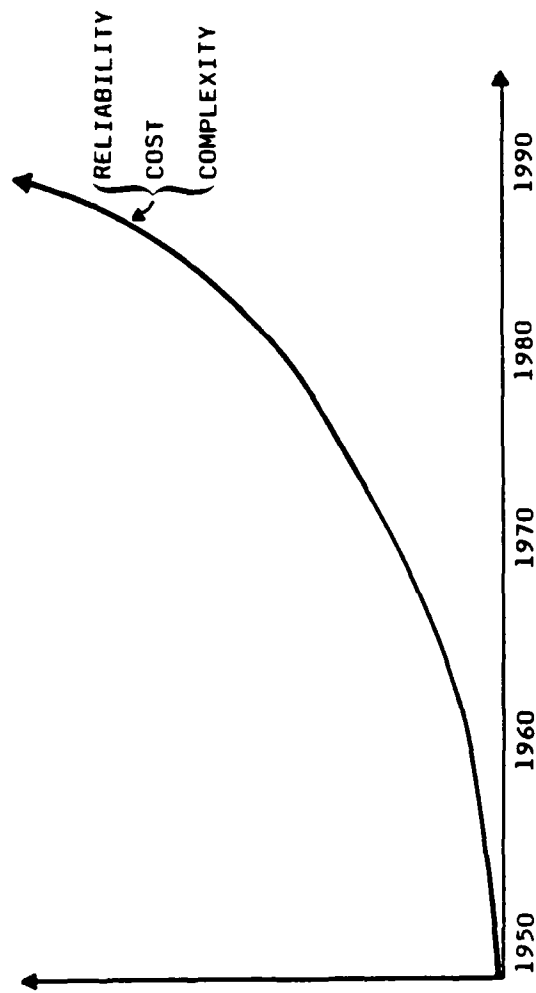
INSTRUCTOR NOTES

ALL THE FACTORS OF RELIABILITY, COST, AND COMPLEXITY ARE SKYROCKETING; EVEN TODAY.

STRUCTURED PROGRAMMING

SCOPE

- STRUCTURED PROGRAMMING DEALS WITH MORE THAN STRUCTURED CONTROL FLOWS ...
STRUCTURED PROGRAMMING EVOLVED FROM THE NEED TO MAKE RELIABLE AND COST-EFFECTIVE SOFTWARE.



- Ada's SCOPE PARTLY EVOLVED FROM STRUCTURED PROGRAMMING CONCEPTS
 - Ada EXTENDED THE CONCEPTS

INSTRUCTOR NOTES

ANY PROGRAM CAN BE WRITTEN USING 3 BASIC CONTROL STRUCTURES.

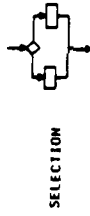
THE PAPER WAS "FLOW DIAGRAMS, TURING MACHINES, AND LANGUAGES WITH ONLY TWO FORMATION RULES," CACM May 1966.

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

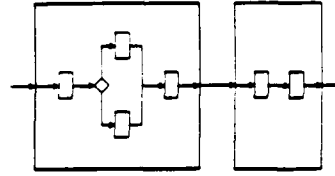
CONTROL STRUCTURING RULES/GUIDELINES

- BOHM AND JACCOPINNI PROVED THAT ANY LOGIC FLOW COULD BE REDUCED TO THREE SIMPLE RULES ...

- THE THREE BASIC STRUCTURING RULES THEY STATED WERE:



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

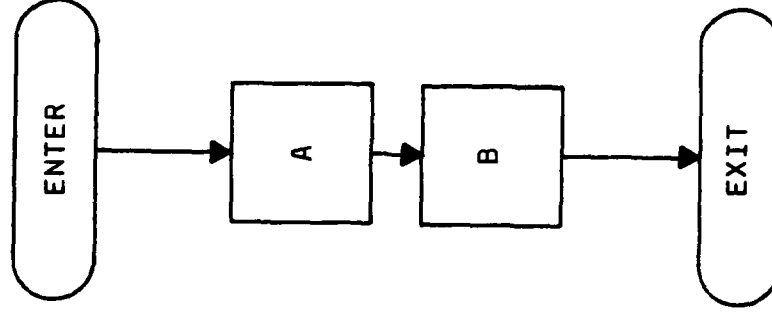


INSTRUCTOR NOTES

REMEMBER, SDS ONLY ALLOWS 5 CONTROL CONSTRUCTS.

DOD-STD-SDS AND STRUCTURE PROGRAMMING

(SEQUENCE)



SOURCE: DOD-STD-SDS, 1984

VG 778.1

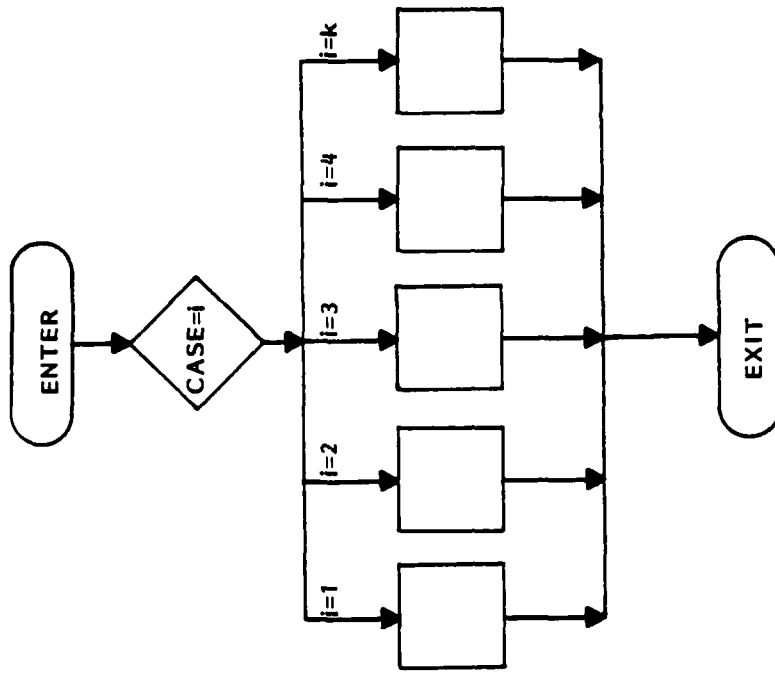
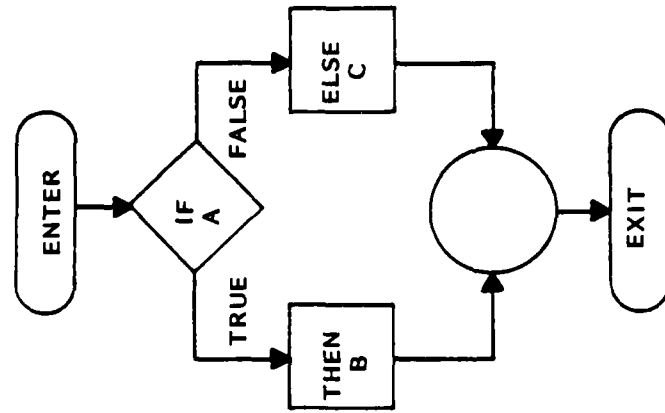
SEQUENCE CONSTRUCT

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 778.1

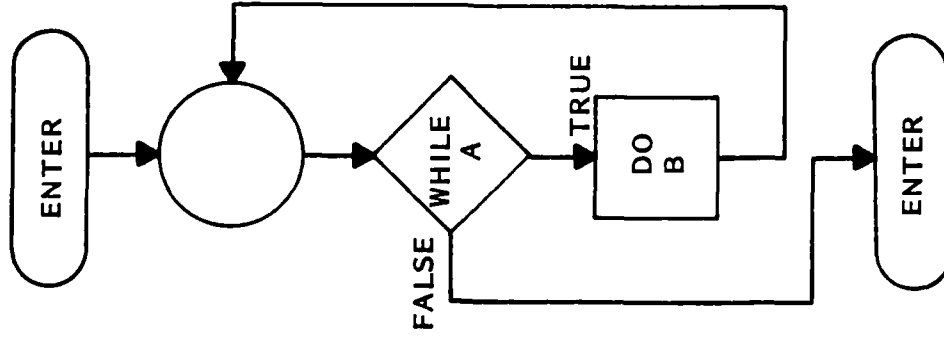
CASE CONSTRUCT

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

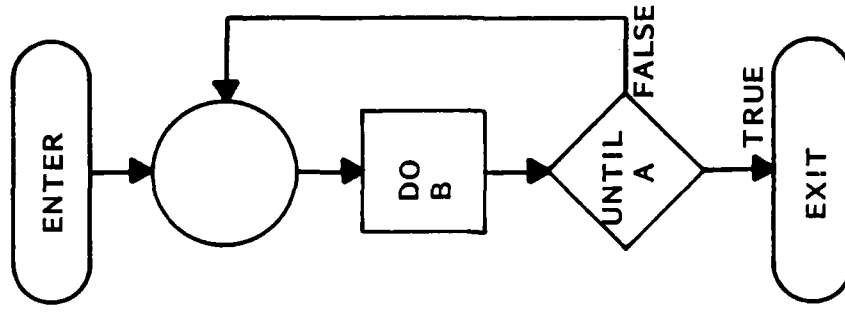
(REPETITION)



DO-WHILE CONSTRUCT

SOURCE: DOD-STD-SDS, 1984

VG 778.1



DO-UNTIL CONSTRUCT

INSTRUCTOR NOTES

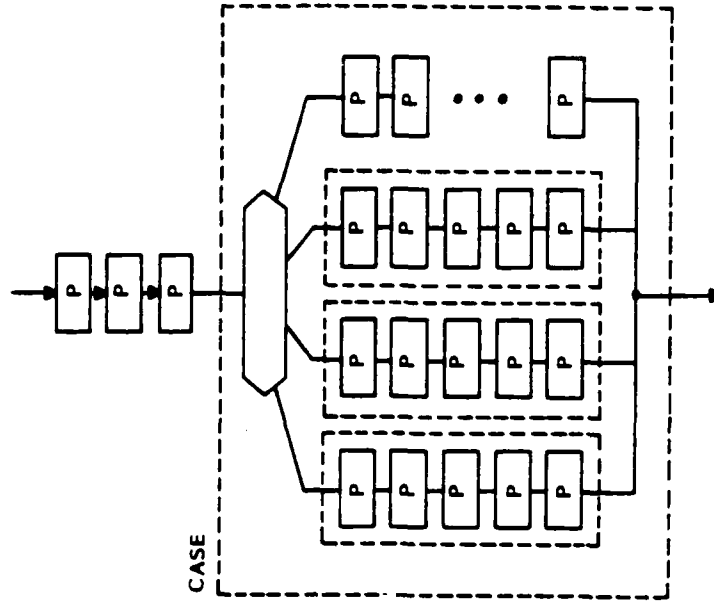
THIS IS A RATHER OBVIOUS BUT IMPORTANT EXAMPLE OF A CASE CONSTRUCT. SPEND A LITTLE TIME
HERE AND SEE IF ALL CONDITIONS ARE SPECIFIED.

Ada AND STRUCTURED PROGRAMMING (SEQUENCE)

```

GET(USER_COMMAND);
NEW_LINE;
PUT_LINE("Command accepted");
case USER_COMMAND is
    when DISABLE =>
        PUT("Enter sensor name:");
        GET(NAME);
        NEW_LINE;
        COLLECTION OF SENSORS.DISABLE(SENSOR => NAME);
        PUT_LINE("Sensor disabled");
    when ENABLE =>
        PUT("Enter sensor name:");
        GET(NAME);
        NEW_LINE;
        COLLECTION OF SENSORS.ENABLE(SENSOR => NAME);
        PUT_LINE("Sensor enabled");
    when RECORD STATUS =>
        PUT("Enter sensor name:");
        GET(NAME);
        NEW_LINE;
        COLLECTION OF SENSORS.FORCE_RECORD
            (OF_SENSOR => NAME);
        PUT_LINE("Sensor status set");
    when SET LIMITS =>
        PUT("Enter sensor name:");
        GET(NAME);
        NEW_LINE;
        PUT("Enter lower limit:");
        GET (LOW_BOUND);
        NEW_LINE;
        PUT_LINE("Lower limit accepted");
        PUT("Enter upper limit:");
        GET(HIGH_BOUND);
        NEW_LINE;
        PUT_LINE("Upper limit accepted");
        COLLECTION OF SENSORS.SET_THE_LIMITS
            (FOR_SENSOR => NAME,
             LOW_LIMIT => LOW_BOUND,
             HIGH_LIMIT => HIGH_BOUND);
        PUT_LINE("Limits set");
end case;

```



- EXAMPLE SHOWS
- SEQUENCE
- HIERARCHICAL NESTING
- SELECTION (CASE STATEMENTS)

INSTRUCTOR NOTES

ASK CLASS,

"WHEN WOULD YOU USE ONE OVER THE OTHER?"

Ada AND STRUCTURED PROGRAMMING

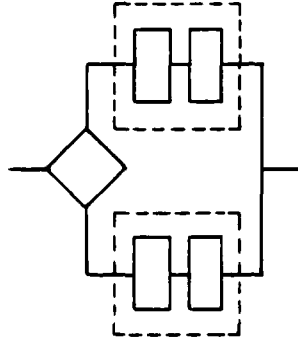
(SELECTION)

IF STATEMENT

```

if VALVE_RECORD(1).OPEN then
    VALVE_RECORD(2).OPEN := TRUE;
    VALVE_RECORD(3).OPEN := FALSE;
else
    VALVE_RECORD(2).OPEN := FALSE
    VALVE_RECORD(3).OPEN := TRUE;
end if;

```

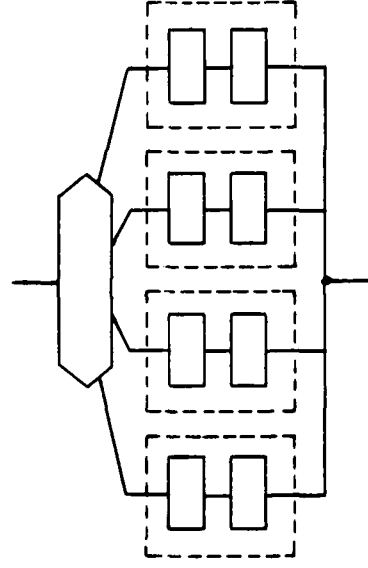


CASE STATEMENT

```

case PROCESS_STATE is
    when RUNNING => SCHEDULER_TABLE(RUNNING) := 1
                    IS ACTIVE := TRUE;
    when READY   => SCHEDULER_TABLE(READY) :=
                    SCHEDULER_TABLE(READY) + 1;
                    IS ACTIVE := FALSE;
    when BLOCKED => SCHEDULER_TABLE(BLOCKED) :=
                    SCHEDULER_TABLE(BLOCKED) + 1;
                    IS ACTIVE := FALSE;
    when DEAD    => SCHEDULER_TABLE(DEAD) :=
                    SCHEDULER_TABLE(DEAD) + 1;
                    IS ACTIVE := FALSE;
end case;
VG 778.1

```



INSTRUCTOR NOTES

EXPLAIN THE DIFFERENCES CAREFULLY.

Ada AND STRUCTURED PROGRAMMING (REPETITION)

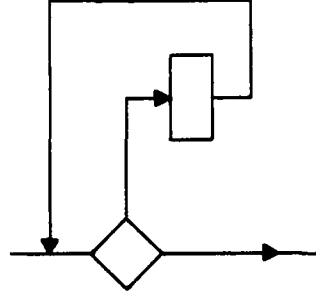
FOR LOOP

```

for INDEX in RUNNING .. DEAD
loop
    SCHEDULER_TABLE(INDEX) := 0
end loop;

for INDEX in reverse TOTAL_VALVES
loop
    VALVE_RECORD(INDEX).OPEN := FALSE;
end loop;

```



WHILE LOOP

```

while not END_OF_FILE(DATA_FILE)
loop
    ACTIVE_RECORDS := ACTIVE_RECORDS + 1;
    READ(DATA_FILE, ITEM => DATA(INDEX));
end loop;

```

• FOR LOOP IS THE MORAL EQUIVALENT
OF A DO LOOP

INSTRUCTOR NOTES

DON'T GET BOGGED DOWN HERE. JUST ILLUSTRATE ONE OR TWO FUNCTION CALLS.

Ada AND STRUCTURED PROGRAMMING

- Ada GOES BEYOND JUST STRUCTURED CONTROL; IT ALSO PROVIDES STRUCTURED DATA
- ABSTRACT DATA TYPE EXAMPLE OF STRUCTURED DATA

```

package COMPLEX is
  type NUMBER is record
    REAL_PART : FLOAT;
    IMAGINARY_PART : FLOAT;
  end record;
  function "+" (A, B : in NUMBER) return NUMBER;
  function "-" (A, B : in NUMBER) return NUMBER;
  function "*" (A, B : in NUMBER) return NUMBER;
  function "/" (A, B : in NUMBER) return NUMBER;
end COMPLEX;

package body COMPLEX is
  function "+" (A, B : in NUMBER) return NUMBER is
    RESULT : NUMBER;
  begin
    RESULT.REAL_PART := A.REAL_PART + B.REAL_PART;
    RESULT.IMAGINARY_PART := A.IMAGINARY_PART + B.IMAGINARY_PART;
    return RESULT;
  end "+";
  function "-" (A, B : in NUMBER) return NUMBER is
  begin
    return NUMBER'(REAL_PART => A.REAL_PART - B.REAL_PART,
                   IMAGINARY_PART => A.IMAGINARY_PART - B.IMAGINARY_PART);
  end "-";
  function "*" (A, B : in NUMBER) return NUMBER is
    RESULT : NUMBER;
  begin
    RESULT.REAL_PART := (A.REAL_PART * B.REAL_PART) -
      (A.IMAGINARY_PART * B.IMAGINARY_PART);
    RESULT.IMAGINARY_PART := (A.REAL_PART * B.IMAGINARY_PART) +
      (A.IMAGINARY_PART * B.REAL_PART);
    return RESULT;
  end "*";
  end COMPLEX;

```


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: MANAGING THE COMPLEXITIES ASSOCIATED WITH THE DEVELOPMENT OF SOFTWARE ARE
KEY TO ALL PHASES OF THE LIFE CYCLE.

PURPOSE: TO PROVIDE A MIX OF TECHNIQUES FOR MANAGING COMPLEXITY IN PROGRAMMING.

REFERENCES: KNUTH, D. "FUNDAMENTAL ALGORITHMS" ADDISON WESLEY, MA; 1973

MCGOWAN, C., KELLY, J. "TOP-DOWN STRUCTURED PROGRAMMING TECHNIQUES" VAN
NOSTRAND REINHOLD, NY; 1975

Section 26
PROGRAM COMPLEXITY MANAGEMENT

VG 778.1

INSTRUCTOR NOTES

THERE IS NO CORRELATION BETWEEN THE LIMIT OF PERCEPTION CATEGORIES AND THE LIMIT OF DESCRIPTORS.

WHY CONSIDER PROGRAM COMPLEXITY MANAGEMENT?

- BECAUSE PEOPLE ARE LIMITED IN THE WAY THEY THINK, THEY MODEL ...
 - DATA BY DESCRIPTORS AND THE RULES THAT ACT UPON IT
 - ACTIVITIES BY CATEGORIZING DESCRIPTORS AND INTEGRATING THEM INTO RULES
- THE NUMBER OF PERCEPTION CATEGORIES HAS A LIMIT FROM AROUND 5 TO AROUND 9.
- THE NUMBER OF DESCRIPTORS THAT CAN BE PUT INTO A SINGLE CATEGORY ALSO HAS A LIMIT FROM AROUND 5 TO AROUND 9.
- THESE LIMITS GIVE HUMANS ONLY A SMALL TOOL KIT TO HANDLE COMPLEX PROBLEMS.

INSTRUCTOR NOTES

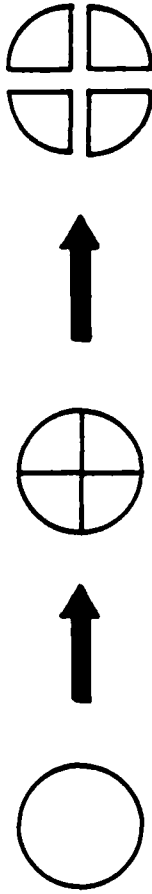
THESE TECHNIQUES ARE WAYS OF HELPING ORGANIZE COMPLEXITY.

PROGRAM COMPLEXITY MANAGEMENT TECHNIQUES

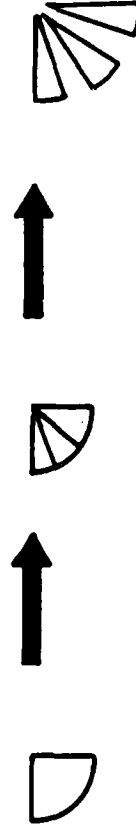
- TECHNIQUES HAVE BEEN DEVELOPED TO MODEL PORTIONS OF A SYSTEM IN WAYS THAT DO NOT TAX THE LIMITS OF HUMAN THINKING.
- SOME EARLY PROGRAMMING TECHNIQUES WERE ...
 - STEP-WISE REFINEMENT
 - DECISION TABLES
 - DECISION TREES
 - FINITE-STATE MAPS
- IN GENERAL THESE TECHNIQUES MAKE THE COMPLEXITY ASSOCIATED WITH A PORTION OF A PROGRAM MORE APPROACHABLE.

STEP-WISE REFINEMENT

- STEP-WISE REFINEMENT IS THE BREAKING OF A PROBLEM INTO SEVERAL SMALLER PIECES:



- THEN EACH PIECE CAN BE INVESTIGATED SEPARATELY, AND POTENTIALLY BROKEN INTO STILL SMALLER PIECES:



- THE NUMBER OF PIECES IN EACH BREAK UP HAS A LIMIT WITHIN THE BOUNDARIES OF HUMAN COMPLEXITY LIMITS.
- MOST OF THE ANALYSIS AND DESIGN METHODS UTILIZE STEP-WISE REFINEMENT

INSTRUCTOR NOTES

POINT OUT THAT THE "TOP" IS NOT NECESSARILY ALWAYS THE BEGINNING.

ALSO RECALL THE TECHNIQUE OF ITERATION.

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

STEP-WISE REFINEMENT

- PROGRAMMERS START AT THE "TOP" OF THE SYSTEM AND BREAK IT APART SLOWLY AS THEY FOLLOW THE STEP-WISE REFINEMENT PARADIGM.
- OFTEN, THE HARDEST TASK IN STEP-WISE REFINEMENT IS FINDING THE RIGHT "TOP."
- NO METHODOLOGY TELLS YOU WHAT THE "TOP" OF THE SYSTEM IS.
- SOMETIMES, THE ONLY WAY TO MAKE A "TOP" IS TO CREATE A SCENARIO (BLOW-BY-BLOW ACCOUNT) AND GENERALIZED TO A HIGHER LEVEL PATTERN.

STEP-WISE REFINEMENT EXAMPLE

HERE'S A SCENARIO FOR ADDING TWO NUMBERS:	AND HERE'S A MORE GENERALIZED PROCEDURE:
592 +236 ??? 1 LOOK AT RIGHTMOST COLUMN 2 ADD 2 TO 6 GIVING 8 3 ENTER 8 IN RIGHTMOST COLUMN OF ANSWER 4 MOVE TO SECOND COLUMN FROM RIGHT 5 ADD 9 TO 3 GIVING 12 6 SEPARATE 12 INTO 2 AND A CARRY OF 1 7 ENTER 2 IN SECOND COLUMN FROM RIGHT TO ANSWER 8 MOVE TO THIRD COLUMN FROM RIGHT 9 ADD 5 TO 2 AND THE CARRY OF 1, GIVING 8 10 ENTER 8 IN THIRD COLUMN FROM RIGHT OF ANSWER	CLEAR CARRY SET CURRENT-COLUMN TO RIGHT-COLUMN REPEAT UNTIL NO DIGITS IN CURRENT-COLUMN ADD CARRY AND DIGITS IN COLUMN SPLIT RESULT INTO ANSWER-DIGIT AND CARRY ENTER ANSWER-DIGIT IN CURRENT-COLUMN OF ANSWER MOVE CURRENT-COLUMN LEFT BY 1 ENDREPEAT

- BUT WATCH OUT!! A SINGLE SCENARIO MAY NOT GIVE ENOUGH CLUES!

THIS COULD HAVE BEEN
EXPRESSED IN A PDL

INSTRUCTOR NOTES

AFTER READING THIS, ASK SOMEONE TO REPEAT IT WITHOUT LOOKING AT IT.

DECISION TABLES

- IN PROGRAMMING, DETAILS LEAVE US REELING IN CONFUSION, FOR EXAMPLE ...

WE LIMIT THE AMOUNT AN EMPLOYEE CAN SAVE, DEPENDING ON HIS SALARY AND HIS LENGTH OF SERVICE WITH US. AN EMPLOYEE CAN PUT AWAY UP TO 5%, 6% OR 7% OF THE FIRST \$15,000 OF HIS SALARY IF HE'S BEEN WITH US 1 YEAR, 2 YEARS OR MORE, RESPECTIVELY. IF HE'S BEEN WITH US 1 YEAR, HE CAN PUT AWAY UP TO 4% OF THE NEXT \$10,000. TWO-YEAR WORKERS GET 5% OF ANY AMOUNT FROM \$15,000 TO \$30,000 AND 4% AFTER THAT. LONG-SERVICE PEOPLE GET TO SAVE UP TO 7% OF THEIR FIRST \$15,000 AND THEN 6% AND 5% AFTER THAT.

- HOW WELL DOES THIS TEXTUAL MODEL DESCRIBE THE RULES?

DECISION TABLES

TRANSLATING THE TEXTUALLY EXPRESSED RULES TO TABULAR FORMAT PUTS RULES
IN PERSPECTIVE

LENGTH OF SERVICE (YEARS)	1	1	1	1-2	1-2	1-2	2	2	2
SALARY (THOUSANDS)	<15	15-25	>25	<15	15-30	>30	<15	15-30	>30
PERCENTAGE ALLOWED %	5	4	?	6	5	4	7	6	5

DECISION TABLES

- DECISION TABLES ARE USED TO IDENTIFY ACTIONS (RATHER THAN SIMPLY NUMBERS) WHICH RESULT FROM A CERTAIN COMBINATION OF CONDITIONS.
- IN GENERAL, DECISION TABLES ARE OF THE FORM:

CONDITION 1	T	T	T	T	F	F	F
CONDITION 2	T	T	F	F	T	F	F
CONDITION 3	T	F	T	T	F	T	F
.							
.							
.							
ACTION 1							
ACTION 2							
ACTION 3							
ACTION 6							
.							
.							
.							

- EACH POSSIBLE COMBINATION OF CONDITIONS GETS ITS OWN COLUMN IN THE TABLE. THE ACTIONS TO BE TAKEN ARE MARKED IN THE COLUMNS.

INSTRUCTOR NOTES

DECISION TABLES EXAMPLE

- A DECISION TABLE CAN REPRESENT THE RULES FOR FILLING OUT THE 1979 FEDERAL INCOME TAX GUIDE PRIOR TO WRITING A PROGRAM TO AUTOMATE THE

CALCULATIONS:

- TEXTUALLY STATEMENT - YOU MUST USE SCHEDULE B IF YOU HAD MORE THAN \$400 IN INTEREST OR DIVIDENDS OR YOU HAD A FOREIGN ACCOUNT. IF YOU HAD MORE THAN \$400 IN INTEREST, FILL OUT PART I OF SCHEDULE B. IF YOU HAD MORE THAN \$400 IN DIVIDENDS, FILL OUT PART II. IF YOU FILLED OUT PART I OR PART II OR YOU HAD A FOREIGN ACCOUNT, FILL OUT PART III.

DECISION TABLES

EXAMPLE

STEP 1:

IDENTIFY ALL THE CONDITIONS IN THE PROBLEM AND ALL THE POSSIBLE VALUES OF EACH CONDITION. (DON'T WORRY ABOUT THE ACTIONS RIGHT NOW.)

<u>CONDITIONS</u>	<u>VALUES</u>
INTEREST > \$400	Y OR N
DIVIDENDS > \$400	Y OR N
FOREIGN ACCOUNT	Y OR N

STEP 2:

WORK OUT HOW MANY COMBINATIONS OF CONDITIONS THERE ARE IN TOTAL BY MULTIPLYING THE NUMBER OF VALUES FOR EACH CONDITION.

IN THIS CASE, THERE ARE A TOTAL OF 8 COMBINATIONS OF CONDITIONS ($8 = 2 \times 2 \times 2$).

INSTRUCTOR NOTES

THESE STEPS ARE, IN REALITY, DONE VERY QUICKLY. HOWEVER, IF THERE ARE MANY DECISIONS, THIS TECHNIQUE CAN BE LABORIOUS.

DECISION TABLES EXAMPLE

STEP 3:

WRITE DOWN ALL OF THE INDEPENDENT ACTIONS.

ACTIONS

DO NOTHING
FILL OUT PART I
FILL OUT PART II
FILL OUT PART III

STEP 4:

DRAW A SKELETON DECISION TABLE WITH THE CONDITIONS AT THE TOP AND THE ACTIONS AT THE BOTTOM. DRAW AS MANY COLUMNS AS THERE ARE COMBINATIONS OF CONDITIONS. NUMBER THE COLUMNS.

	1	2	3	4	5	6	7	8
CONDITIONS { INTEREST > \$400 DIVIDENDS > \$400 FOREIGN ACCOUNT								
ACTIONS { DO NOTHING FILL OUT PART I FILL OUT PART II FILL OUT PART III								

INSTRUCTOR NOTES

VG 778.1

26-121

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.

DECISION TABLES

STEP 5:

FILL IN THE CONDITION VALUES:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------|---|---|---|---|---|---|---|---|
| INTEREST > \$400 | Y | Y | Y | Y | N | N | N | N |
| DIVIDENDS > \$400 | Y | Y | N | N | Y | Y | N | N |
| FOREIGN ACCOUNT | Y | N | Y | N | Y | N | Y | N |
| DO NOTHING | | | | | | | | |
| FILL OUT PART I | | | | | | | | |
| FILL OUT PART II | | | | | | | | |
| FILL OUT PART III | | | | | | | | |

STEP 6:

MARK APPROPRIATE ACTION(S) FOR EACH COMBINATION OF CONDITIONS.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------|---|---|---|---|---|---|---|---|
| INTEREST > \$400 | Y | Y | Y | Y | N | N | N | N |
| DIVIDENDS > \$400 | Y | Y | N | N | Y | Y | N | N |
| FOREIGN ACCOUNT | Y | N | Y | N | Y | N | Y | N |
| DO NOTHING | X | X | X | X | | | | X |
| FILL OUT PART I | X | X | | | X | | | |
| FILL OUT PART II | X | X | X | X | X | X | X | |
| FILL OUT PART III | X | X | X | X | X | X | X | |

STEP 7: IMPLEMENT IN Ada

INSTRUCTOR NOTES

ASK CLASS, "WHERE ARE TABLES NOT USEFUL?"

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.

DECISION TABLES

- TO SUM UP, A DECISION TABLE
 - IS A USEFUL TOOL WHERE COMPLICATED CONDITIONS APPLY OR WHERE COMPLEX DECISIONS ARE TO BE MADE
 - IS USED TO FIND INCONSISTENCIES OR MISSING ACTIONS
 - IS USED TO ORGANIZE ONE'S THOUGHTS PRIOR TO WRITING CODE
 - CAN BE USED AS DOCUMENTATION WITHIN THE CODE
- DECISION TABLES ARE TYPICALLY IMPLEMENTED WITH SOME COMBINATION OF THE STRUCTURED CONTROL CONSTRUCTS IN PREVIOUS SECTION.

INSTRUCTOR NOTES

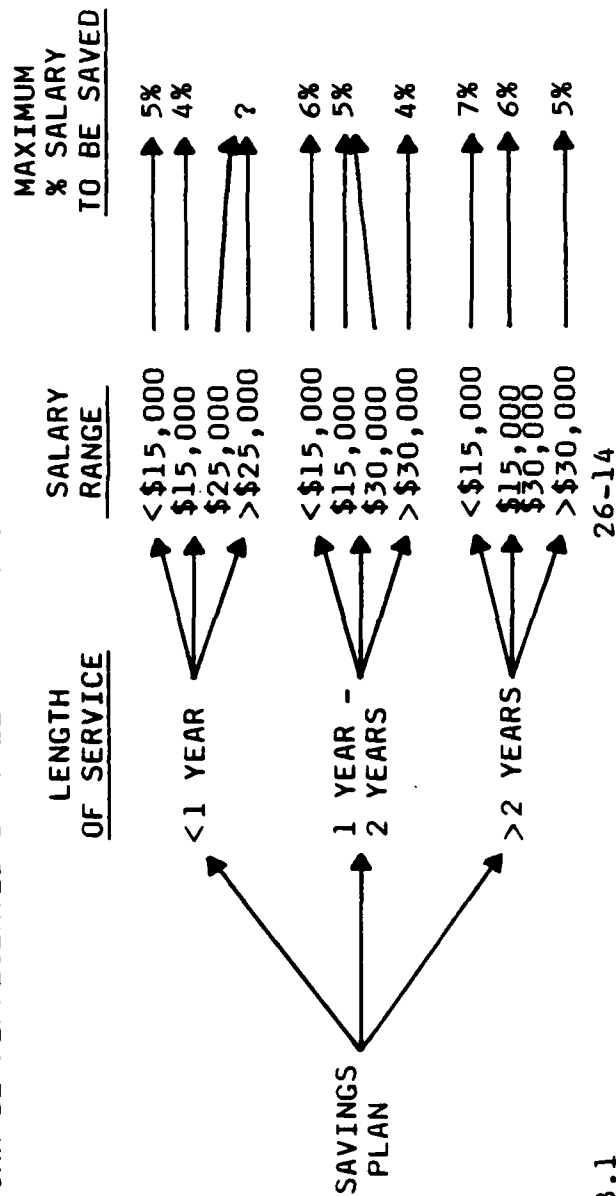
TREES AND TABLES REPRESENT THE SAME THING.

DECISION TREES

- A DECISION TREE IS A MORE GRAPHIC FORM OF A DECISION TABLE
- FOR EXAMPLE, THE TABLE OF EMPLOYEE SAVINGS RATES ...

| LENGTH OF SERVICE (YEARS) | 1 | 1 | 1 | 1-2 | 1-2 | 1-2 | 2 | 2 | 2 |
|---------------------------|-----|-------|-----|-----|-------|-----|-----|-------|-----|
| SALARY (THOUSANDS) | <15 | 15-25 | >25 | <15 | 15-30 | >30 | <15 | 15-30 | >30 |
| PERCENTAGE ALLOWED % | 5 | 4 | ? | 6 | 5 | 4 | 7 | 6 | 5 |

- CAN BE REPRESENTED IN TREE FORM BY ...



INSTRUCTOR NOTES

DECISION TREES

- MANY PEOPLE FIND DECISION TREES TO BE MORE IMMEDIATE THAN DECISION TABLES
 - EXPLICITLY SHOW RELATIONSHIP BETWEEN CAUSE AND EFFECT (CONDITION AND ACTION)
- SOMETIMES, TREES ARE EASIER TO IMPLEMENT THAN TABLES. SO PICK A REPRESENTATION (TREE OR TABLE) MOST SUITABLE FOR PROGRAMMING. (YOU CAN ALWAYS DOCUMENT IT IN THE OTHER FORM IF YOU WISH.)

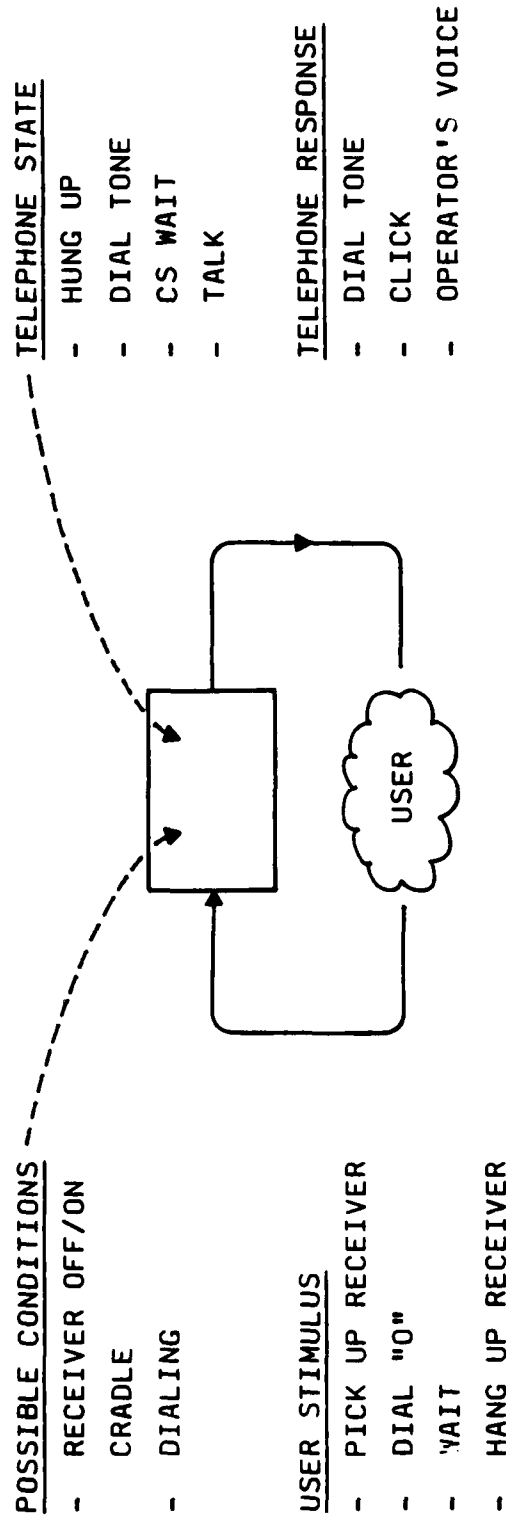
INSTRUCTOR NOTES

A BLACK BOX IS A MACHINE WHOSE ACTIONS ARE HIDDEN FROM THE OBSERVER. ONLY BY OBSERVING THE BOX'S RESPONSE TO INPUTS CAN ONE MODEL ITS BEHAVIOR. A SCENARIO IS THE OBSERVING OF OUTPUTS BASED ON GIVEN INPUTS.

STATE THAT FOR THIS EXAMPLE WE AREN'T USING A TRUE BLACK BOX. WE NEED TO HAVE SOME INFORMATION ON HOW OUR TELEPHONE SYSTEM WORKS FOR THIS EXAMPLE TO END UP BEING RIGHT. FURTHER, THESE "INPUTS" ARE REALLY ACTIONS, NOT CONTROL VARIABLES. THE "WAIT" INPUT IS REALLY A COMMUNICATIONS SWITCHING WAIT. PICKING UP THE RECEIVER GIVES AN INSTANT DIAL TONE WITH NO WAIT.

FINITE STATE MAPS

- FINITE STATE MAPS MODEL THE DYNAMIC BEHAVIOR OF A PORTION OF A SYSTEM, BY SHOWING ITS STATES AND TRANSFORMATIONS
- MANY REAL TIME OR EMBEDDED APPLICATIONS HAVE A PORTION THAT CAN EASILY BE REPRESENTED AS A FSM (I.E. COMMUNICATIONS PROTOCOLS)
- FINITE STATE MAPS DESCRIBE "BLACK BOXES." FOR EXAMPLE, HERE'S A BLACK BOX SCENARIO OF HOW TO USE THE TELEPHONE TO TALK WITH THE OPERATOR ...



INSTRUCTOR NOTES

THERE ARE THREE INPUTS, EACH WITH A BINARY STATE, OR 2^3 POSSIBLE COMBINATIONS. A STATE EXISTS IF SOMETHING HAPPENS AT AN OUTPUT GIVEN AN INPUT. 0 MEANS THAT CONDITION IS NOT TRUE.

| INPUT | pick up
rec | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
|-------------------|----------------|---|---|---|---|---|---|----|---|---|
| "0" | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| CS wait | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| possible
state | * | * | * | * | * | * | * | ** | * | * |
| OUTPUT | dial tone | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| click | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| op voice | 0* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

* This is a state because by definition we need a "zeroth" state.

**In real world, but not this problem, this would be another state (i.e., a wait state).

FINITE STATE MAPS

EXAMPLE

- A TABLE IS DRAWN, DEFINING EACH STATE OF THE PHONE USING A SPECIFIC COMBINATION OF INPUTS AND OUTPUTS ...

| PROPERTY
STATE | CONDITIONS | | OUTPUT |
|-------------------|---------------------------|----------|--------------|
| | RECEIVER
OFF/ON CRADLE | DIALING | |
| 0. HUNG UP | ON | NO | NONE |
| 1. DIAL TONE | OFF | NO | DIAL
TONE |
| 2. CS WAIT | OFF | DIAL "0" | CLICK |
| 3. TALK | OFF | NO | VOICE |

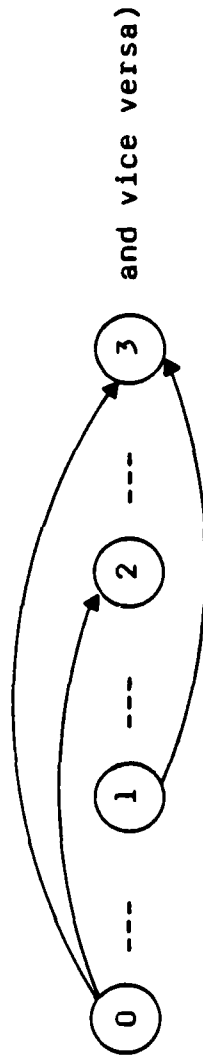
- LASTLY, A MAP IS DRAWN, USING THE SYMBOLS ...



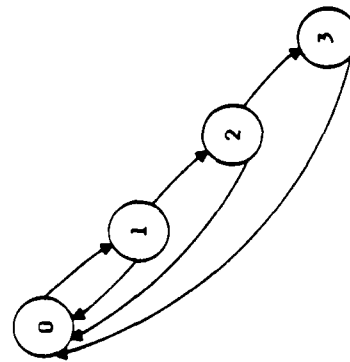
TRANSITION (AN ACTION THAT GETS YOU OUT OF ONE STATE AND INTO ANOTHER)

INSTRUCTOR NOTES

THUS WE HAVE FOUR STATES. WE NEED TO DRAW THE TRANSITIONS BETWEEN THEM. THERE ARE 12 POSSIBLE TRANSITIONS (6 FORWARD, 6 BACKWARDS, i.e.

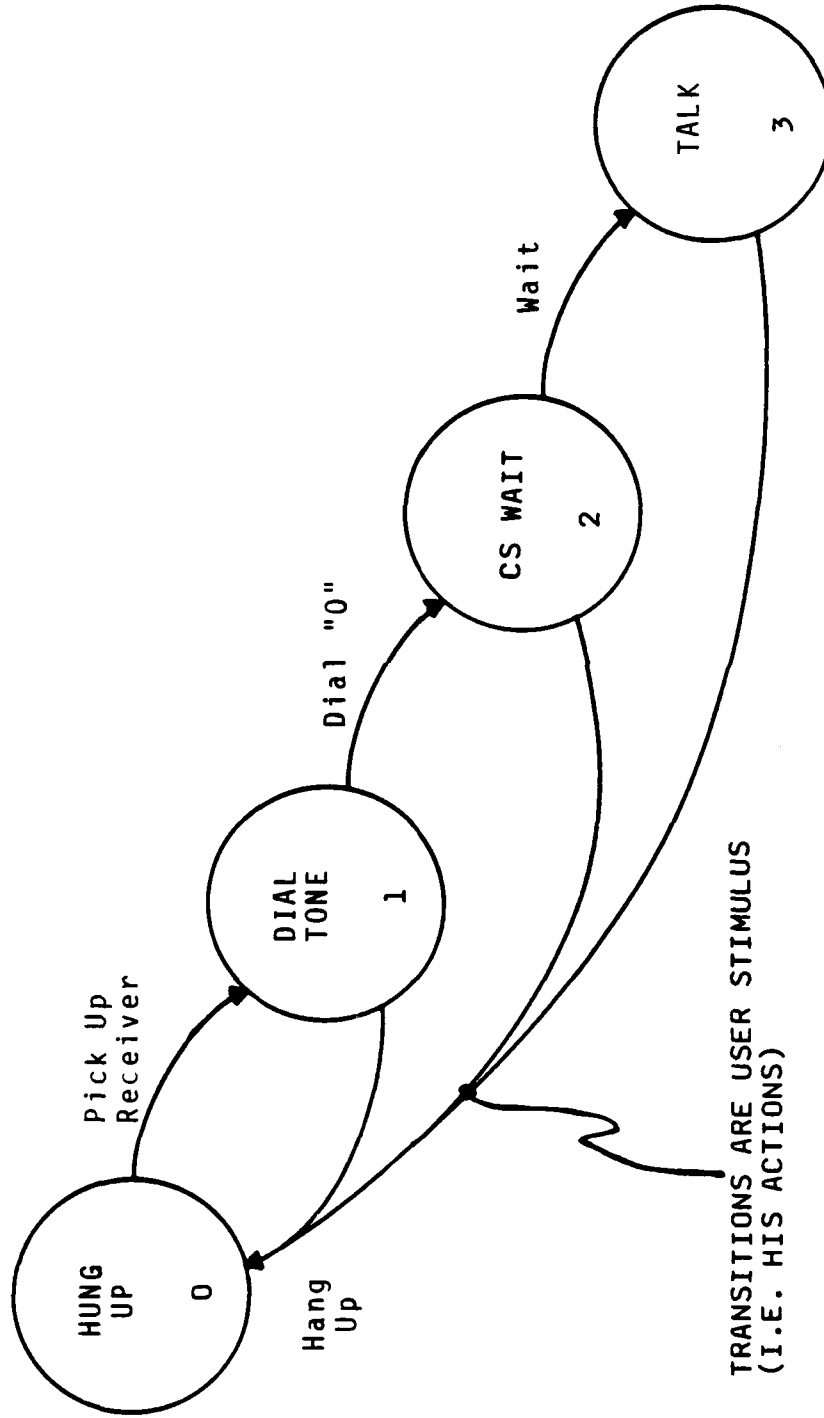


NOW DRAW THE FS MAP AS FOLLOWS, DESCRIBING WHY ONLY CERTAIN TRANSITIONS ARE ALLOWED.



FINITE STATE MAPS

EXAMPLE



INSTRUCTOR NOTES

FSM'S REPRESENT THE DYNAMIC NATURE OF A SYSTEM, WHILE DECISION TREES AND TABLES ARE MORE
STATIC IN NATURE.

FINITE STATE MAPS

- FINITE STATE MAPS ARE EASY FOR PEOPLE TO READ AND UNDERSTAND
- DECISION TABLES AND FINITE STATE MAPS SHARE SOME COMMON ATTRIBUTES
 - BOTH EXPRESS CONDITIONS VS. ACTIONS
 - A TABULAR REPRESENTATION OF A FSM CAN BE TRANSLATED TO A DECISION TABLE TO AID IN PROGRAMMING

INSTRUCTOR NOTES

AD-A165 302

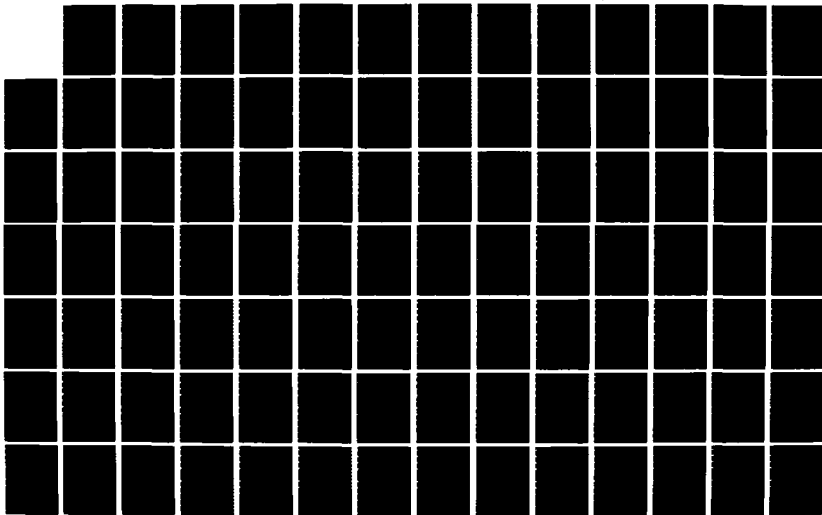
ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 3
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

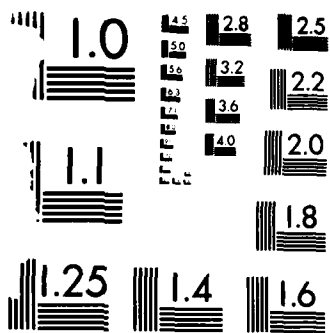
1/5

UNCLASSIFIED

F/G 5/9

NL





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

SUMMARY

- PROGRAMS ARE COMPLEX.
- HUMANS HAVE LIMITS TO THEIR THINKING.
- TECHNIQUES WERE DEVELOPED TO COPE WITH COMPLEXITY BY
LIVING WITHIN HUMAN LIMITS:
 - STEP-WISE REFINEMENT
 - DECISION TABLES
 - DECISION TREES
 - FINITE STATE MAPS

INSTRUCTOR NOTES

THEME: INTEREST IN "PROVING" PROGRAMS ARE CORRECT IS GROWING, THE TECHNOLOGY ISN'T
HERE YET, BUT IT COULD BE HERE SOON.

PURPOSE: TO PROVIDE A PRACTICAL APPROACH TO APPLYING PROGRAM CORRECTNESS TECHNIQUES
TO Ada PROGRAMS.

REFERENCES: DIJKSTRA, E. "GUARDED COMMANDS, NONDETERMINACY AND FORMAL DERIVATION OF
PROGRAMS" CACM VOL. 18, NO. 5; AUGUST 1975

MILLS, H. "HOW TO WRITE CORRECT PROGRAMS AND KNOW IT" CONFERENCE ON
RELIABLE SOFTWARE PROCEEDINGS ACM SIGPLAN; 1975

WIRTH, N. "ON THE COMPOSITION OF WELL-STRUCTURED PROGRAMS" ACM COMPUTING
SURVEYS VOL. 6, NO. 4; DECEMBER 1974

Section 27
PROGRAM CORRECTNESS

VG 778.1

INSTRUCTOR NOTES

STATE THAT YOU ARE USING CONCEPTS OF PROGRAM CORRECTNESS TO SHOW HOW GOOD CODE IS
PRODUCED - NOT REALLY DISCUSSING FORMAL PROGRAM CORRECTNESS FIELD.

PROGRAM CORRECTNESS

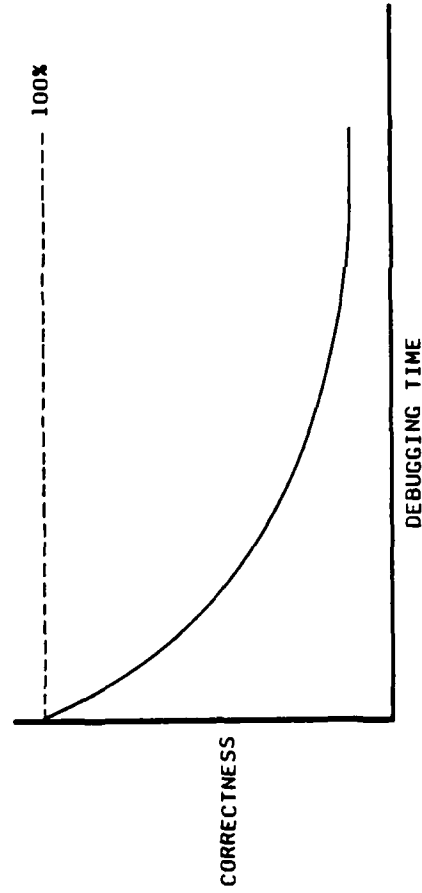
- CORRECTNESS - MEANS THE PROGRAM WILL WORK FOR ANY COMBINATION OF INPUTS
 - ANY MEANS ALL. FOR EXAMPLE, TO VERIFY THE ADDITION OF TWO INTEGER NUMBERS FROM 1 TO 1,000,000 WOULD TAKE 1,000,000,000 TESTS:
- TESTING FOR CORRECTNESS IS NOT FEASIBLE
 - COMBINATIONS OF TESTS EXPAND EXPONENTIALLY
 - TESTING ONLY SHOWS THE PRESENCE OF ERRORS, NOT THEIR ABSENCE
- CORRECTNESS IS A MEASURE OF THE ABSENCE OF ERRORS
- A WEALTH OF THEORY EXISTS THAT IS JUST NOW SHOWING LIMITED PRACTICAL USE

INSTRUCTOR NOTES

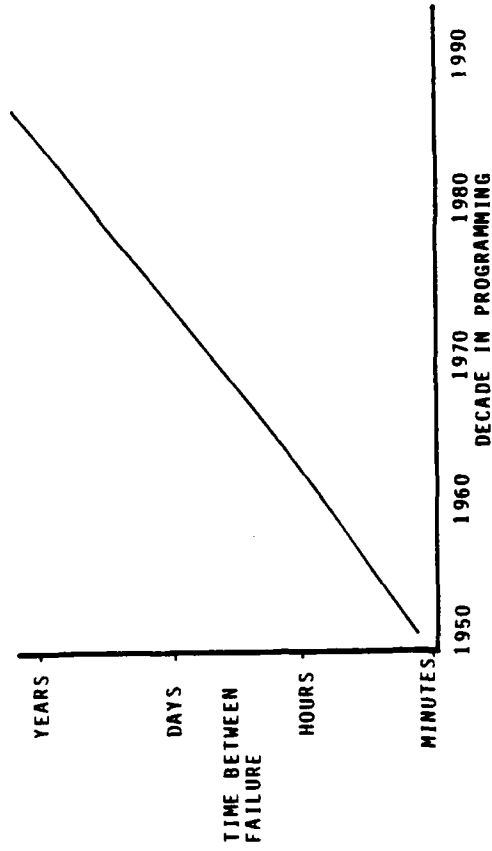
DON'T CONFUSE CORRECTNESS WITH OPERABILITY. A PROGRAM MAY BE CORRECT, BUT STILL NOT
USABLE BY THE INTENDED USER.

PROGRAM CORRECTNESS MOTIVATION

- IT IS EASIER TO DEBUG WHEN THERE ARE FEWER ERRORS ...



- RELIABILITY REQUIREMENTS ARE GETTING STRICTER ...



INSTRUCTOR NOTES

A MAJOR REASON FOR USING CORRECTNESS ANALYSIS IS THAT IT IS GETTING CHEAPER TO BUILD
"CORRECT" PROGRAMS THAN TO FIX INCORRECT ONES.

HEURISTIC = RULE OF THUMB.

BUILDING IN CORRECTNESS

- IT IS EASIER TO BUILD CORRECTNESS IN THAN TO ...
 - TEST IT IN
 - GET THE DEFECT OUT
- THERE ARE SOME SIMPLE HEURISTICS ...
 - ASSERTIONS -- ONE OF THE FIRST NOTED BENEFITS OF COMMENTING CODE WAS OPPORTUNITY TO DOCUMENT ASSERTIONS ABOUT PROGRAMS
 - INVARIANT -- IS THE SITUATION THAT IS ALWAYS TRUE ABOUT THE LOOP; BEFORE AND AFTER
 - GUARDS -- A GUARD ENSURES THE SAFETY OF CARRYING OUT EACH STEP OF A LOOP

INSTRUCTOR NOTES

AN ASSERTION IS SOME RELATIONSHIP THAT THE PROGRAMMER KNOWS TO BE TRUE.

POINT OUT THAT THE EXAMPLE IS TRIVIAL, BUT DOES SHOW WHAT IS MEANT BY AN ASSERTION.

ASSERTIONS

- ASSERTION - IS A STATEMENT (COMMENT) ABOUT WHAT HAS
HAPPENED SO FAR.
(AS OPPOSED TO WHAT IS ABOUT TO HAPPEN.)

B = 5;

A = B + 7;

-- { A IS NOW 12 AND B IS NOW 5. }

- ASSERTIONS CAN BE ENGLISH STATEMENTS ABOUT THE RESULT OF
AN EVALUATION OF A CONDITIONAL.

INSTRUCTOR NOTES

ASSERTIONS ARE MEANT TO CONVEY INFORMATION. THERE IS NO RESTRICTION AS HOW TO EXPRESS ONE.

ASSERTIONS

- ASSERTIONS ARE THE "STORY SO FAR."
- THE ONE-IN, ONE-OUT CONTROL STRUCTURES OF STRUCTURED PROGRAMMING ENABLES US TO MAKE GUARANTEED ASSERTIONS
- ASSERTIONS HELP CLARIFY THE ELSE PARTS OF CONDITIONALS ESPECIALLY NESTED ONES.
- ASSERTIONS WHEN USED WITH VERIFIER TOOLS PROVIDE SOME OPPORTUNITIES TO PROVE PROGRAM CORRECTNESS

INSTRUCTOR NOTES

THE LOOP INVARIANT IS A TECHNIQUE THAT WILL HELP US CODE LOOPS CORRECTLY.

INVARIANTS

- PERHAPS THE TOUGHEST CONTROL STRUCTURE TO GET RIGHT IS THE LOOP. PROBLEMS INCLUDE ...
 - CODE LOOPING INDEFINITELY
 - CODE DOES ONE TOO MANY OR TOO FEW ITERATIONS
- ALL THESE PROBLEMS ARE THE RESULT OF IMPROPERLY UNDERSTANDING THE LOOP INVARIANT
- FROM AN INVARIANTS STAND POINT A LOOP IS A STEP-BY-STEP PROCEDURE TO GO FROM A BEGINNING TO A CONCLUSION

INSTRUCTOR NOTES

THE INVARIANT NEVER CHANGES. "SUM" IS ALWAYS EQUAL TO THE SUMMATION OF THE Jth ELEMENT IN THE ARRAY "A."

- SUM IS SUMMATION OF 1st J ELEMENTS OF THE LOOP.

- AT THE BOTTOM OF THE LOOP $J > 10$, IN FACT J IS 11.

- WHAT'S TRUE AT THE BOTTOM OF THE LOOP IS THE NEGATION OF THE CONDITION AND INVARIANT.

INVARIANTS

FOR EXAMPLE, THE LOOP:

```
J := 0;  
SUM := 0;  
WHILE J < 10 LOOP  
    J := J + 1;  
    SUM = SUM + A (J);  
END LOOP;
```

HAS THE INVARIANT:

$$\text{SUM} = \sum_{i=1}^J A(i)$$

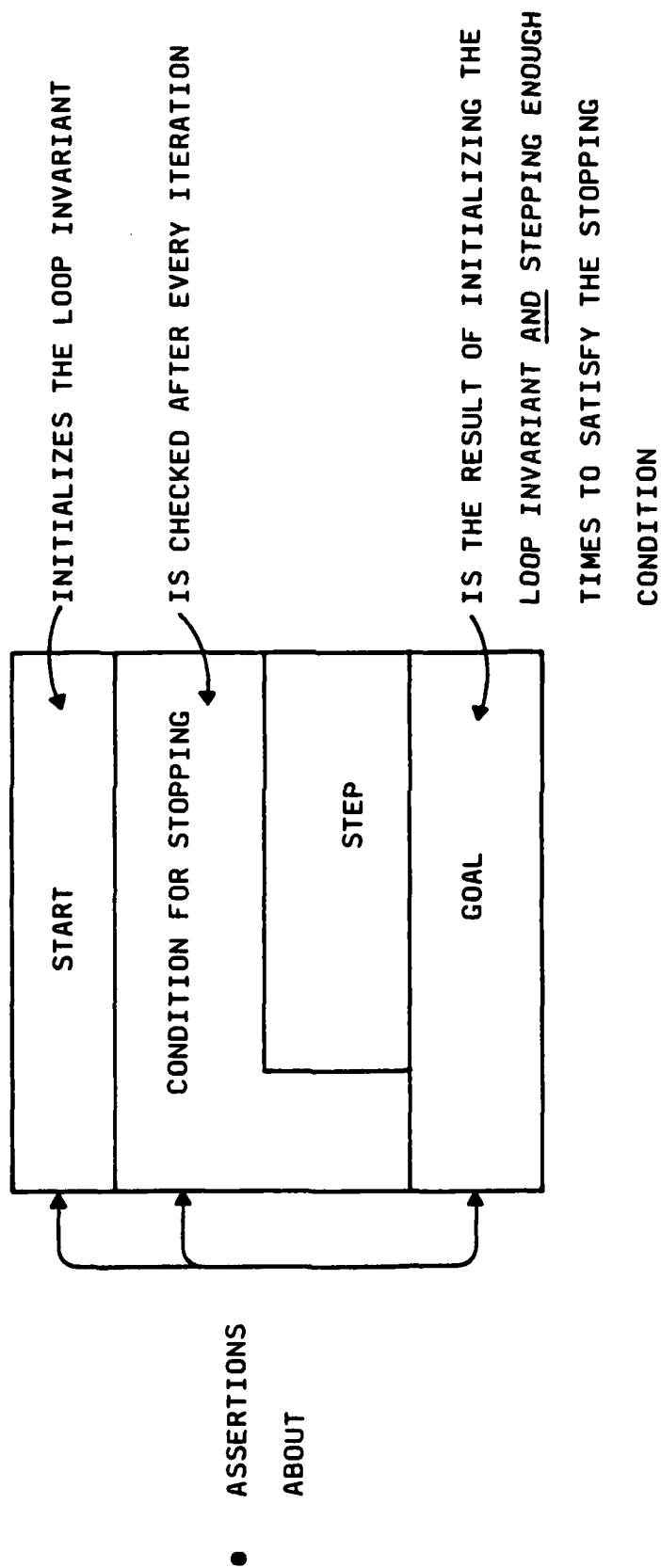
INITIALIZATION SETS UP THE INVARIANT:

WHEN J = 0, SUM = 0

INSTRUCTOR NOTES

EVERY POINT MUST BE ABLE TO BE VERIFIED.

INvariants



INSTRUCTOR NOTES

A GUARD "ALLOWS" US TO CONTINUE.

GUARDS

- EACH STEP OF A LOOP HAS TWO PARTS ...
 - ONE THAT MAKES PROGRESS TOWARDS THE GOAL
 - ONE THAT PRESERVES THE LOOP INVARIANT
- A GUARD IS THE OPPOSITE OF THE CONDITION FOR STOPPING

INSTRUCTOR NOTES

DON'T GO THROUGH THIS IN MUCH DEPTH. IT IS STRAIGHTFORWARD AND IT WILL BE INVESTIGATED
IN DETAIL IN THE FOLLOWING SLIDE.

PROGRAM CORRECTNESS EXAMPLE

• CODE FRAGMENT MANIPULATING A STACK ...

```
Make_Empty (Stack);  
Count := 0;  
Push (Tree, Onto => Stack);  
while not Is_Empty (Stack) loop  
  
    Pop (T, Off_Of => Stack);  
    if Is_Leaf (T) then  
        Count := Count + 1;  
    else  
        Push (Left_Subtree (T), Onto => Stack);  
        Push (Right_Subtree (T), Onto => Stack);  
        end if;  
    end loop;
```

INSTRUCTOR NOTES

THE ASSERTIONS ARE IN THE BOXES. EACH ASSERTION TELLS WHAT HAS HAPPENED (OR SHOULD HAVE HAPPENED) SO FAR.

PROGRAM CORRECTNESS - AN OPTIMIST'S VIEW

- IN THE FUTURE, SPECIFICATIONS WILL BE COMPLETELY FORMAL AND RIGOROUS

[REQUIREMENTS DEFINITION] WILL CULMINATE IN A COMPLETE, UNAMBIGUOUS,
AND PROVABLY CONSISTENT SPECIFICATION FOR THE ENTIRE END PRODUCT ...

INSTEAD OF PRETTY PICTURES AND DRAWINGS, [THE SPECIFICATION WILL
CONSIST OF] A COLLECTION OF DEFINITIONS, MATHEMATICAL FORMULAS, AND
LOGICAL PROOFS ...

C.A.R. HOARE

"PROGRAMMING: SORcery OR SCIENCE?"
IEEE SOFTWARE, APRIL 1984

- PROGRAMMERS WELL-VERSED IN LOGIC WILL DEVELOP A PROGRAM AND A PROOF OF CORRECTNESS
HAND-IN-HAND
- PROVEN PROGRAMS ARE GUARANTEED TO MEET THEIR SPECIFICATIONS, SO IT WILL BE
UNNECESSARY TO PROVIDE FOR SOFTWARE ERROR RECOVERY

INSTRUCTOR NOTES

ON THE OTHER HAND, OTHERS BELIEVE CORRECTNESS ISN'T WORTH THE TIME, TROUBLE, OR EXPENSE,
AND ISN'T COMPLETE ANYWAY.

PROGRAM CORRECTNESS - A PESSIMIST'S VIEW

- VERIFIABILITY IS NOT A PRIMARY GOAL OF MOST PROGRAMMING LANGUAGES, INCLUDING Ada
 - HARD TO WRITE PROOF RULES FOR PARAMETER PASSING, SHARED VARIABLES, ETC.
 - DEFINITION OF Ada IS LARGE AND COMPLEX

- MACHINE ARITHMETIC DOESN'T OBEY ARITHMETIC AXIOMS:

EXAMPLES:

$$a*(b+c) = a*b + a*c$$

$$(a/b)*b = a$$

- EMBEDDED SYSTEMS ARE TOO LARGE AND COMPLEX TO PROVE CORRECT
- EVEN FOR NONEMBEDDED SYSTEMS, INPUT-OUTPUT ASSERTIONS ARE EXTREMELY COMPLEX AND DIFFICULT EVEN FOR A TRAINED LOGICIAN TO GET RIGHT
 - COMPLETENESS
 - NOTATION

INSTRUCTOR NOTES

MOREOVER, THE COMPUTER FIELD ISN'T ONLY POPULATED WITH C.A.R. HOARE'S.

PROGRAM CORRECTNESS - A PESSIMIST'S VIEW (CONTINUED)

- THERE EXIST CORRECT PROGRAMS THAT CAN'T BE PROVEN CORRECT
- VERIFIED SOFTWARE CAN FAIL
 - COMPILER, OPERATING SYSTEM, HARDWARE ERRORS
 - INPUT DATA VIOLATING INPUT ASSUMPTIONS
 - NONFUNCTIONAL REQUIREMENTS (PERFORMANCE)
- FEW PEOPLE WITHOUT GRADUATE DEGREES IN COMPUTER SCIENCE WILL BE CAPABLE OF USING FORMAL VERIFICATION
 - FACILITY IN READING, WRITING, AND MANIPULATING LOGICAL FORMULAS
 - DIFFICULTY OF FORMULATING INVARIANTS
- DEVELOPING A PROOF WHILE WRITING A PROGRAM ENTAILS VOLUMINOUS, ERROR-PRONE FORMULA MANIPULATION

INSTRUCTOR NOTES

THESE CONCLUSIONS ARE ABOUT THE "BEST" COMPROMISES BETWEEN THE OPTIMIST'S AND
PESSIMIST'S VIEWS.

PROGRAM CORRECTNESS - A PRAGMATIST'S VIEW

- IDENTIFY A VERIFIABLE SUBSET OF Ada.
 - AVOID DIFFICULT LANGUAGE FEATURES, ENCOURAGE GOOD STYLE
- ACCEPT INCOMPLETENESS
 - MOST "ACCIDENTALLY CORRECT" PROGRAMS CAN'T BE PROVEN CORRECT
 - MOST POORLY WRITTEN PROGRAMS CAN'T BE PROVEN CORRECT
 - PROGRAMS USING CERTAIN LANGUAGE FEATURES (E.G. SHARED VARIABLES) CAN'T BE PROVEN CORRECT
- PROVE SPECIFIC, LIMITED PROPERTIES OF PROGRAMS
 - NO NEED TO FORMULATE COMPLEX INPUT/OUTPUT ASSERTIONS FOR A FULL PROGRAM
 - SOME PRECONDITIONS/POSTCONDITIONS CAN BE FORMULATED AUTOMATICALLY
 - VERIFY PROGRAM COMPONENTS INDEPENDENTLY AND COMPOSE PROOFS TO VERIFY LARGE SYSTEMS

INSTRUCTOR NOTES

HOW ABOUT CORRECTNESS AND Ada?

Ada AND PROGRAM CORRECTNESS

- PRINCIPAL GOALS OF Ada INCLUDE:

- CONSTRUCTION OF RELIABLE PROGRAMS
- EASE OF MAINTENANCE

- THESE GOALS CAN BE ADVANCED BY FORMALLY PROVING THAT AN

Ada PROGRAM WILL BEHAVE IN A CERTAIN WAY

- CORRECT IMPLEMENTATION
- MODIFICATIONS THAT DON'T VIOLATE INITIAL ASSUMPTIONS

INSTRUCTOR NOTES

THESE ARE THE THINGS WE CAN PROVE ABOUT Ada PROGRAMS. EACH WILL BE REVIEWED IN DETAIL.

SPECIFIC PROVABLE PROPERTIES OF Ada PROGRAMS

- ABSENCE OF CERTAIN FORMS OF ERRONEOUSNESS
- ABSENCE OF UNANTICIPATED EXCEPTIONS
- CORRECTNESS OF NARROWLY-DEFINED PROGRAM COMPONENTS
- CORRECTNESS OF DATA ABSTRACTIONS PROVIDED BY PACKAGES
- NUMERIC PROPERTIES

INSTRUCTOR NOTES

FOR AN Ada ORIENTED CLASS, SPEND SOME TIME AND GIVE AN EXAMPLE. OTHERWISE, SKIM THROUGH THESE POINTS.

PROVING THE ABSENCE OF CERTAIN FORMS OF ERRONEOUSNESS

- STRAIGHTFORWARD FOR PROGRAMS THAT FOLLOW CERTAIN REASONABLE PROGRAMMING PRACTICES:
 - NO PATHS ALONG WHICH A SCALAR VARIABLE IS EXAMINED BEFORE IT IS GIVEN A VALUE
 - RESTRICTED USE OF GLOBAL VARIABLES
 - RESTRICTED USE OF EXPRESSIONS WITH SIDE-EFFECTS
 - NO USE OF THE SAME RECORD, OR SLICES OF THE SAME ARRAY, AT MORE THAN ONE ACTUAL PARAMETER POSITION IN THE SAME SUBPROGRAM CALL
 - NO DIRECT USE OF THE SUPPRESS PRAGMA, UNCHECKED CONVERSION, UNCHECKED DEALLOCATION, OR SHARED VARIABLES
- COMPONENTS THAT DEPART FROM THESE PRACTICES CAN STILL BE PROVABLY NONERRONEOUS
- BENEFITS OF PROVING THE ABSENCE OF ERRONEOUSNESS:
 - FILLS IN THE GAPS IN COMPILE-TIME CHECKING
 - MAKE PROGRAMS SAFER TO RUN AND SAFER TO PORT
 - PRECONDITIONS AND POSTCONDITIONS CAN BE GENERATED AUTOMATICALLY

INSTRUCTOR NOTES

IF ONE CAN DO ALL ERRONEOUS PROOFS, APPLY SOME TO THESE AREAS.

MOST PROMISING AREAS FOR NON-ERRONEOUSNESS PROOFS

- EVALUATION OF A SCALAR VARIABLE WITH AN UNDEFINED VALUE
 - VERIFICATION IS MORE POWERFUL THAN STRAIGHTFORWARD DATA FLOW (LIVE/DEAD, DEF/REF) ANALYSIS, BECAUSE IT CONSIDERS THE CONDITIONS UNDER WHICH A POINT IN A PROGRAM IS REACHED
- OCCURRENCE OF AN ERROR SITUATION FOR WHICH RUN-TIME CHECKS HAVE BEEN SUPPRESSED
 - SUPPRESSION OF CHECKS CAN BE ONE OF THE MOST EFFECTIVE WAYS TO SPEED UP A CRITICAL PART OF A PROGRAM, BUT IT IS ALSO ONE OF THE MOST DANGEROUS
 - IF AN EFFECTIVE VERIFIER IS AVAILABLE, MANAGERS CAN FORBID THE USE OF THE SUPPRESS PRAGMA WITHOUT PROOF OF ITS SAFETY
- USE OF AN ALLOCATED VARIABLE AFTER IT HAS BEEN DEALLOCATED
 - VERIFICATION VALUABLE FOR VALIDATING LEGITIMATE USES OF UNCHECKED DEALLOCATION

INSTRUCTOR NOTES

NOT ALL FORMS OF ERRONEOUSNESS CAN BE HANDLED.

FORMS OF ERRONEOUSNESS THAT PRESENT VERIFICATION PROBLEMS

- AN UNCHECKED CONVERSION THAT VIOLATES THE PROPERTIES THAT ARE GUARANTEED BY THE LANGUAGE FOR OBJECTS OF THE RESULT TYPE
 - ILL-DEFINED
- UPDATE OF A SHARED VARIABLE BETWEEN SYNCHRONIZATION POINTS IF ANOTHER TASK UPDATES OR EXAMINES THE VARIABLE BETWEEN THE SAME SYNCHRONIZATION POINTS
 - IT MAY BE DIFFICULT TO RELATE DYNAMIC SYNCHRONIZATION POINTS BETWEEN TWO TASKS TO POINTS IN THE PROGRAM TEXT. (COMPLICATIONS INCLUDE ENTRY FAMILIES; SELECT STATEMENTS; AND ENTRY CALLS AND ACCEPT STATEMENTS WITHIN CONDITIONAL STATEMENTS.)
 - IMPOSSIBILITY OF ERRONEOUSNESS MAY BE DUE TO IMPLEMENTATION-DEPENDENT SCHEDULING CHARACTERISTICS
 - VERIFICATION OF CONCURRENT PROGRAMS USING SHARED VARIABLES IS NOT WELL UNDERSTOOD
- ADDRESS CLAUSES THAT CAUSE OBJECTS OR PROGRAM UNITS TO BE OVERLAID, OR THAT CAUSE THE SAME INTERRUPT TO BE LINKED TO MORE THAN ONE ENTRY
 - IMPLEMENTATION-DEPENDENT

INSTRUCTOR NOTES

THESE MAY NOT BE AS USEFUL.

OTHER FORMS OF ERRONEOUSNESS WHOSE ABSENCE CAN BE PROVEN

- INVOCATION OF A SUBPROGRAM IN A SITUATION WHERE ITS EFFECT DEPENDS ON WHETHER COMPOSITE PARAMETERS ARE PASSED BY COPYING OR BY REFERENCE
- ASSIGNMENT TO A COMPONENT IN A RECORD VARIANT, IF EXECUTION OF THE ASSIGNMENT STATEMENT HAS THE SIDE-EFFECT OF CHANGING THE RECORD'S DISCRIMINANT
- PASSING A COMPONENT IN A RECORD VARIANT AS A COMPOSITE OR IN OUT OR OUT PARAMETER TO A SUBPROGRAM, IF EXECUTION OF THE SUBPROGRAM CHANGES THE RECORD'S DISCRIMINATION

INSTRUCTOR NOTES

THIS ASSURES THE PROGRAMMER EXCEPTIONS ARE HANDLED THE WAY HE HAD INTENDED.

VERIFYING THE ABSENCE OF UNANTICIPATED EXCEPTIONS

- PROGRAMMER SPECIFIES THE EXCEPTIONS THAT HE ANTICIPATES EACH SUBPROGRAM MAY RAISE
- VERIFIER PROVES THAT NO OTHER EXCEPTIONS CAN BE RAISED
- BENEFITS:
 - PRECONDITIONS AND POSTCONDITIONS CAN BE GENERATED AUTOMATICALLY
 - PROGRAMMER ALERTED TO BOUNDARY CONDITIONS AND OTHER POSSIBLE EXCEPTIONS HE HASN'T ACCOUNTED FOR
- DIFFICULTIES:
 - IMPLEMENTATION-DEPENDENT EXCEPTIONS (`IO_Exceptions.Use_Error`, `Program_Error`)
 - "UNPREDICTABLE" EXCEPTIONS (`Storage_Error`, I/O exceptions dependent on existence and content of external files, `IO_Exceptions.Device_Error`)

INSTRUCTOR NOTES

VERIFYING DATA ABSTRACTIONS ARE NOT A DIFFICULT THING TO DO, AND IS HIGHLY BENEFICIAL.

VERIFYING DATA ABSTRACTIONS

- IN Ada, A DATA ABSTRACTION USUALLY CORRESPONDS TO A PACKAGE PROVIDING A PRIVATE TYPE AND OPERATIONS ON THAT TYPE
- BEHAVIOR OF THE TYPE CAN BE DESCRIBED BY AXIOMS, AND THE AXIOMS CAN BE USED TO VERIFY PARTS OF THE PROGRAM THAT USE THE DATA ABSTRACTION
- VERIFYING THE DATA ABSTRACTION CONSISTS OF PROVING THAT THE PACKAGE'S SUBPROGRAMS OBEY THE AXIOMS
- BENEFITS:
 - WELL UNDERSTOOD
 - SUPPORTED BY Ada PACKAGES AND VISIBILITY RULES
 - ALLOWS COMPOSITION OF PROOFS
 - FACILITATES A LIBRARY OF TESTED, FORMALLY SPECIFIED DATA ABSTRACTIONS

INSTRUCTOR NOTES

THIS IS PROBABLY NOT WORTH DOING IN MOST CASES.

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

VERIFYING NUMERIC PROPERTIES

- BASE AXIOMS FOR REAL ARITHMETIC ON MODEL INTERVALS
- PROVE ASSERTIONS ABOUT ACCURACY OF RESULTS
- BENEFITS:
 - FORMALIZE PROOF OF NUMERIC PROPERTIES
 - PROVE PORTABILITY
 - PROVE ABSENCE OF Numeric_Error
- PROBLEMS:
 - MODEL ARITHMETIC TENDS TO BE FAR MORE PESSIMISTIC THAN REALITY, BUT
Ada SEMANTICS DO NOT JUSTIFY ANY STRONGER METHOD
 - MAY BE IMPOSSIBLE TO PROVE CORRECTNESS OF CORRECT ALGORITHMS

INSTRUCTOR NOTES

SUMMARY

- PROOF OF TOTAL CORRECTNESS INFEASIBLE, ESPECIALLY FOR
 - A LANGUAGE WITH ALL OF Ada'S FEATURES
 - PROBLEMS AS COMPLEX AS EMBEDDED SYSTEMS SOFTWARE
 - A WORKFORCE WITH LIMITED EXPERTISE IN FORMAL LOGIC
- SIGNIFICANT BENEFITS CAN BE DERIVED FROM APPLYING VERIFICATION TECHNIQUES TO MORE LIMITED PROBLEMS. BENEFITS INCLUDE:
 - PROGRAMMING DISCIPLINE
 - FORMAL SEMANTIC INTERFACES
 - RELIABLE PROGRAMS
 - RELIABLE MAINTENANCE
 - RELIABLE SOFTWARE REUSE
- THESE MORE LIMITED PROBLEMS SEEM FEASIBLE
 - IMPACT OF Ada'S COMPLEXITY IS CONTROLLED
 - DEMANDS ON USER EXPERTISE ARE REDUCED

21. 202 1E 202 1P 202 771 202 801 202 810 202 825 202 835 202 839 202 844 202 845 202 846 202 847 202 848 202 849 202 850 202 851 202 852 202 853 202 854 202 855 202 856 202 857 202 858 202 859 202 860 202 861 202 862 202 863 202 864 202 865 202 866 202 867 202 868 202 869 202 870 202 871 202 872 202 873 202 874 202 875 202 876 202 877 202 878 202 879 202 880 202 881 202 882 202 883 202 884 202 885 202 886 202 887 202 888 202 889 202 890 202 891 202 892 202 893 202 894 202 895 202 896 202 897 202 898 202 899 202 900 202 901 202 902 202 903 202 904 202 905 202 906 202 907 202 908 202 909 202 910 202 911 202 912 202 913 202 914 202 915 202 916 202 917 202 918 202 919 202 920 202 921 202 922 202 923 202 924 202 925 202 926 202 927 202 928 202 929 202 930 202 931 202 932 202 933 202 934 202 935 202 936 202 937 202 938 202 939 202 940 202 941 202 942 202 943 202 944 202 945 202 946 202 947 202 948 202 949 202 950 202 951 202 952 202 953 202 954 202 955 202 956 202 957 202 958 202 959 202 960 202 961 202 962 202 963 202 964 202 965 202 966 202 967 202 968 202 969 202 970 202 971 202 972 202 973 202 974 202 975 202 976 202 977 202 978 202 979 202 980 202 981 202 982 202 983 202 984 202 985 202 986 202 987 202 988 202 989 202 990 202 991 202 992 202 993 202 994 202 995 202 996 202 997 202 998 202 999 202 1000

[illegible][illegible][illegible]

Section 28

TESTING APPROACHES

VG 778.1

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.

RELATIONSHIP OF TESTING AND OTHER ERROR REMOVAL TECHNIQUES

PHASE IN WHICH ERROR WAS INTRODUCED

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

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 GUIDELINES THAT SHOULD BE FOLLOWED FAIRLY STRICTLY.

TESTING PRINCIPLES

- SPECIFY TEST CASE RESULTS
- DON'T TEST YOUR OWN CODE
- REVIEW TEST CASE RESULTS
- TEST ERRONEOUS INPUTS
- DEVELOP MAINTAINABLE TEST CASES
- SUCCESSFUL TESTS FIND ERRORS

INSTRUCTOR NOTES

THESE ARE THE MOST COMMON TEST CASES. EACH WILL BE EXPLAINED IN SUBSEQUENT SLIDES.

TEST-CASE DESIGN

- BLACK-BOX

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

- WHITE-BOX

- LOGIC - COVERAGE TESTING

INSTRUCTOR NOTES

PRIMARY TESTING PROCEDURE.

EQUIVALENCE PARTITIONING

- REQUIRES EXTERNAL, FUNCTIONAL SPECIFICATION
- PARTITION EXTERNAL CONDITIONS INTO VALID AND INVALID EQUIVALENCE CLASSES
 - "ITEM COUNT CAN BE FROM 1 TO 999"
 - 1 VALID CLASS 1 < ITEM COUNT < 999
 - 2 INVALID CLASSES: ITEM COUNT < 1, ITEM COUNT > 999
 - "COLOR CAN BE RED OR GREEN"
 - 2 VALID CLASSES, 1 INVALID CLASS
- ASSIGN UNIQUE MEMBERS TO EACH CLASS
- WRITE NEW TEST CASES COVERING AS MANY VALID CLASSES AS POSSIBLE
- WRITE SEPARATE TEST CASE FOR EACH INVALID TEST CASE

INSTRUCTOR NOTES

SPECIAL CASE TESTING.

VG 778.1

28-111

BOUNDARY VALUE ANALYSIS

- USUALLY USED TO AUGMENT EQUIVALENCE PARTITIONING
- ADD TESTS TO CHECK "ENDS" OF VALID RANGE: ITEM COUNT = 1,
ITEM COUNT = 999
- ADD TESTS TO CHECK LIMITS IN NUMBER OF PARAMETERS
- FOCUS ON OUTPUT VALUES AS WELL
- SEARCH FOR OTHER BOUNDARIES

INSTRUCTOR NOTES

GIVES VERY GOOD COVERAGE - TIME CONSUMING HOWEVER, ONE COULD USE DECISION TABLES
DEVELOPED PREVIOUSLY FOR THE IMPLEMENTATION TO HELP GET STARTED.

CAUSE EFFECT GRAPHING

- BREAKS SPECIFICATION INTO SUBSETS
- WITHIN SUBSET IDENTIFY "CAUSES" AND "EFFECTS"
- DRAW BOOLEAN GRAPH LINKING CAUSES AND EFFECTS
- ANNOTATE WITH CONSTRAINTS TO IDENTIFY IMPOSSIBLE COMBINATIONS
- CONVERT TO DECISION TABLE
- EACH COLUMN BECOMES A TEST CASE

INSTRUCTOR NOTES

REQUIRES THE TESTER TO HAVE A FULL UNDERSTANDING OF THE SYSTEM. NOT REALISTIC IN LARGE SYSTEMS.

ERROR GUESSING

- INTUITIVE AND AD-HOC
- IDENTIFY ERROR-PRONE SITUATIONS
- EXAMPLES - FOR A SORT ROUTINE
 - INPUT LIST IS EMPTY
 - INPUT LIST CONTAINS ONE ITEM
 - ALL ENTRIES HAVE SOME VALUE
 - INPUT LIST ALREADY SORTED

INSTRUCTOR NOTES

WHILE VERY COMPLETE, A DETAILED KNOWLEDGE OF THE SYSTEM IS REQUIRED. OFTEN THE DOCUMENTATION IS NOT COMPLETE ENOUGH TO DO WHITE-BOX TESTING.

LOGIC - COVERAGE TESTING (WHITE-BOX)

- STATEMENT COVERAGE
 - EXECUTE EACH STATEMENT AT LEAST ONCE
- DECISION COVERAGE
 - EACH BRANCH DECISION TAKES ON ALL POSSIBLE
VALUE
- CONDITION COVERAGE
 - EACH CONDITION (IN DECISION) TAKES ON ALL
POSSIBLE VALUES
- DECISION/CONDITION COVERAGE
 - ALL COMBINATIONS OF ABOVE
- MULTIPLE CONDITION COVERAGE

INSTRUCTOR NOTES

TESTING STRATEGY IS IMPORTANT TO UNDERSTAND WHAT IT IS YOU'RE TESTING FOR, AND WHAT YOU AREN'T. ALSO, IT POINTS OUT GAPS IN TEST COVERAGE.

TESTING STRATEGY

- START WITH CAUSE - EFFECT GRAPHING (IF POSSIBLE)
- USE BOUNDARY VALUE ANALYSIS
- IDENTIFY EQUIVALENCE CLASSES
- USE ERROR GUESSING TO SUPPLEMENT
- USE LOGIC COVERAGE TO EVALUATE ABOVE TESTS - ADD
TEST TO COMPLETE COVERAGE

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

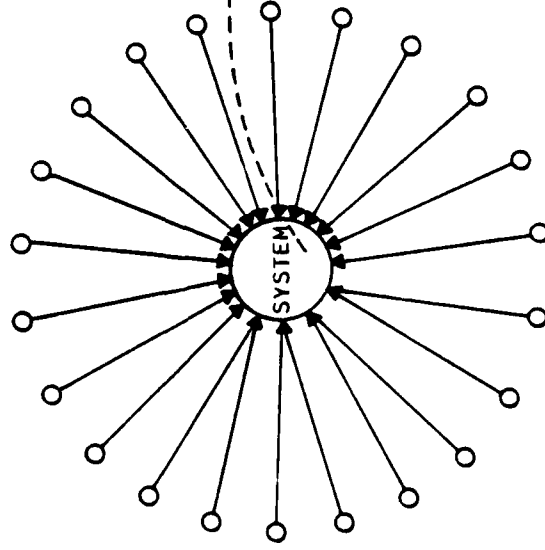
INSTRUCTOR NOTES

PROBABLY THE MOST COMMON APPROACH USED. ALSO THE ONE THAT HAS THE LEAST CHANCE OF
SUCCESS AS THE SYSTEM GROWS LARGER IN SIZE.

INTEGRATION TESTING APPROACHES

(BIG BANG)

- COMBINE ALL MODULES INTO A SYSTEM AND TEST THE CONGLOMERATE



SUMMARY OF EVENTS

"THE DEADLINE IS NOT MET, AND THE WORKERS
CONTINUE TO STUMBLE THROUGH THE TESTING PHASE
UNTIL EITHER THEY WEAR DOWN THE PROBLEMS OR
THE PROBLEMS WEAR THEM DOWN."

- THIS IS THE OLDEST APPROACH WHICH EVOLVED FROM UNSTRUCTURED TESTING
(DEBUGGING) OF SMALL PROGRAMS

INSTRUCTOR NOTES

DISCOURAGE THE USE OF THE BIG BANG APPROACH.

INTEGRATION TESTING APPROACHES

(BIG BANG)

• PROBLEMS

- NO WAY TO PREDICT WHAT KIND (EASY OR HARD) OF ERROR WILL OCCUR NEXT
- ERRORS ARE DIFFICULT TO IDENTIFY
- FIXING ONE ERROR SPAWNS NEW ERRORS
- NEVER SURE WHEN TESTING WILL FINISH

CONSEQUENCES

- MANAGEMENT OF PEOPLE AND COMPUTER RESOURCES IS MADE DIFFICULT
- NO FLEXIBILITY IN THE APPROACH
- BIG SYSTEMS NEVER QUITE WORK

INSTRUCTOR NOTES

THIS METHOD IS POSSIBLY THE ONLY WAY TO HANDLE LARGE SYSTEMS. BUT IT DOES TAKE TIME AND CAREFUL THOUGHT.

INTEGRATION TESTING APPROACHES

(INCREMENTAL)

- CAME ABOUT WHEN LARGE PROGRAMS STARTED BEING BUILT
- USEFUL FOR MEDIUM TO COMPLEX PROGRAMS
- OFFERS SOME CONTROL OF THE TESTING PROCESS
- BASIC APPROACH
 1. WRITE AND TEST EACH MODULE SEPARATELY
 2. COMBINE TWO MODULES TOGETHER INTO A COMBINATION
 3. TEST THE COMBINATION
 4. COMBINE ANOTHER MODULE WITH THE COMBINATION
 5. GO TO STEP 3

INSTRUCTOR NOTES

THERE ARE VERY FEW DISADVANTAGES TO THIS METHOD. ANOTHER ADVANTAGE IS THAT THE SYSTEM DEVELOPMENT CAN BE STAGGERED TO PROVIDE THE MODULES FOR TESTING IN PROPER ORDER, THUS MAKING THE DEVELOPMENT MORE EFFICIENT.

INTEGRATION TESTING APPROACHES

(INCREMENTAL TESTING)

- ADVANTAGES INCLUDE

- INTERFACE DISCREPANCIES CAN BE EASILY IDENTIFIED AND CORRECTED

- YOU CAN PICK WHICH MODULE TO TEST NEXT

- ERRORS ARE LOCALIZED TO THE NEW MODULE AND ITS INTERCONNECTIONS

- FIXING AN ERROR DOESN'T SPAWN NEW ERRORS

- TESTING IS DONE WHEN THE LAST MODULE IS TESTED

INSTRUCTOR NOTES

ENCOURAGE THIS APPROACH.

VG 778.1

28-21i

PROGRAM CORRECTNESS EXAMPLE
(ANNOTATED WITH ASSERTIONS)

```

[True]
{0 + leaves_in_tree (Iree) = leaves_in_tree (Iree)}
Make_Empy (Stack);
[leaves_in_stack (Stack) + leaves_in_tree (Iree) = leaves_in_tree (Iree)]
Count := 0;
{Count +
  leaves_in_stack (Stack) +
  leaves_in_tree (Iree) = leaves_in_tree (Iree)}
Push (Iree, Onto => Stack);

[Count + leaves_in_stack (Stack) = leaves_in_Iree (Iree)]
while not Is_Empy (Stack) loop

  {Count +
    leaves_in_stack ([stack with top item removed]) +
    leaves_in_tree ([top item on stack]) = leaves_in_tree (Iree)}
  Pop (I, Off_Of => Stack);
  {Count +
    leaves_in_stack (Stack) +
    leaves_in_tree (I) = leaves_in_tree (Iree)}
  If Is_Leave (I) then
    [Count + 1 + leaves_in_stack (Stack) = leaves_in_tree (Iree)]
    Count := Count + 1;
  else
    {Count +
      leaves_in_stack (Stack) +
      leaves_in_tree (Left_Subtree (I)) +
      leaves_in_tree (Right_Subtree (I)) = leaves_in_tree (Iree)}
    Push (Left_Subtree (I), Onto => Stack);
    {Count +
      leaves_in_stack (Stack) +
      leaves_in_tree (Right_Subtree (I)) = leaves_in_tree (Iree)}
    Push (Right_Subtree (I), Onto => Stack);
  end if;

  [Count + leaves_in_stack (Stack) = leaves_in_tree (Iree)]
end loop;

[Count = leaves_in_tree (Iree)]

```

INSTRUCTOR NOTES

SOME COMPUTER SCIENTISTS, ESPECIALLY THOSE OF A MATHEMATICAL BENT, SEE CORRECTNESS AS THE (ONLY ?) WAVE OF THE FUTURE.

AD-A165 302

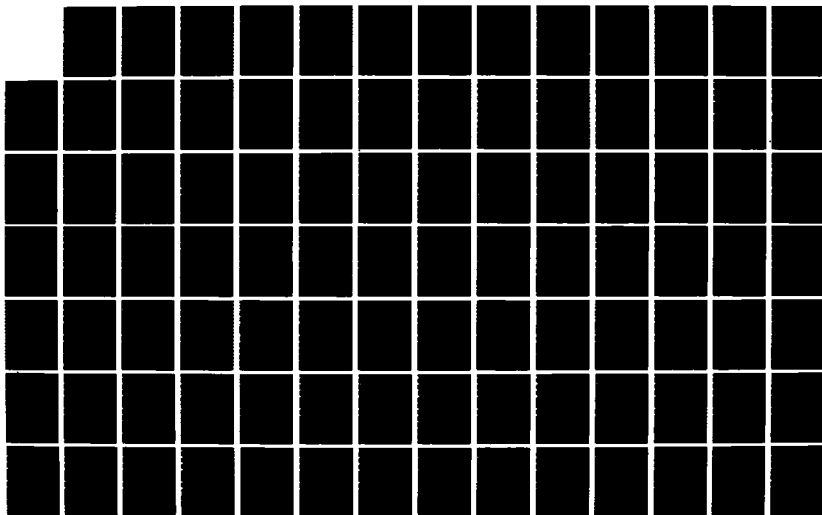
ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 3
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

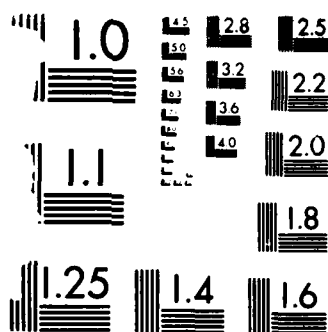
4/5

UNCLASSIFIED

F/G 5/9

NL





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

INTEGRATION TESTING APPROACHES
(INCREMENTAL TESTING)

• CONSEQUENCES ARE

- STAFFING AND COMPUTER RESOURCE NEEDS CAN BE BETTER PREDICTED
- TROUBLE IN A SUBSYSTEM CAN BE IDENTIFIED EARLIER
- APPROACH IS FLEXIBLE (NEW AREA CAN BEGIN BEING TESTED WHILE SUSPENDING AN OLD AREA)
- GREATER CONFIDENCE THAT THE SYSTEM WORKS

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.

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

STUBS ARE ALSO USED IN THE DESIGN OF PROGRAMS.

VG 778.1

28-241

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

TOP-DOWN INTEGRATION

- STUBS (ALIAS DUMMY MODULES) ARE STAND-INS FOR NOT-YET-WRITTEN SUBORDINATE MODULES.
- A STUB MAY:
 - DO NOTHING - NOT VERY USEFUL, BUT EASY TO IMPLEMENT.
 - DISPLAY A TRACE MESSAGE - TO ENABLE A TESTER TO SEE WHAT'S HAPPENING TO THE PROGRAM.
 - CONVERSE WITH AN INTERACTIVE TERMINAL - TO ALLOW, PERHAPS, A TESTER TO ENTER TEST DATA.
 - RETURN A CONSTANT NUMBER, RANDOM NUMBER OR A NUMBER FROM A TABLE.
 - BE A SIMPLER VERSION OF THE REAL MODULE - THIS IS HALF-WAY BETWEEN A STUB AND THE REAL THING.
 - TEST OR DISPLAY ITS DATA - TO ALLOW THE TESTER A FURTHER CHANCE TO MONITOR THE PROGRAM.

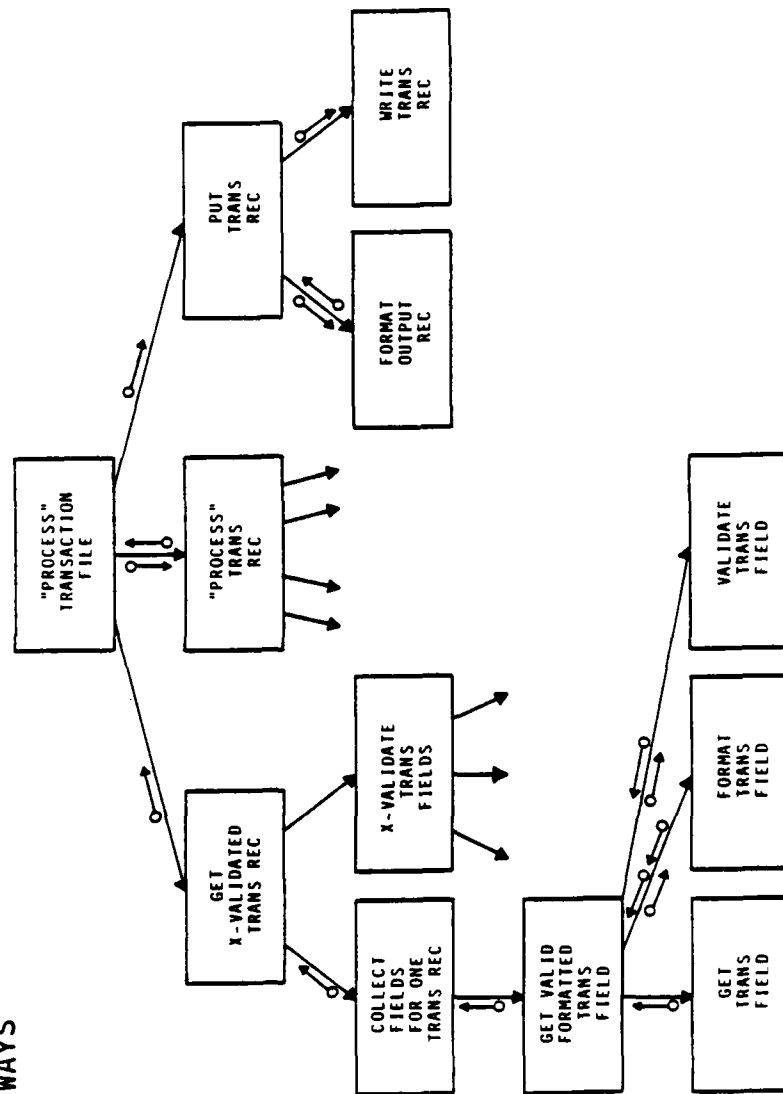
INSTRUCTOR NOTES

POINT OUT THE INPUTS AND OUTPUTS REQUIRED FOR EACH UNIT.

TOP-DOWN INTEGRATION EXAMPLE

• HOW MIGHT THIS DESIGN BE TESTED TOP-DOWN?

- THREE WAYS



INSTRUCTOR NOTES

USE THE CASE THAT BEST FITS THE TESTING DESIRED.

VG 778.1

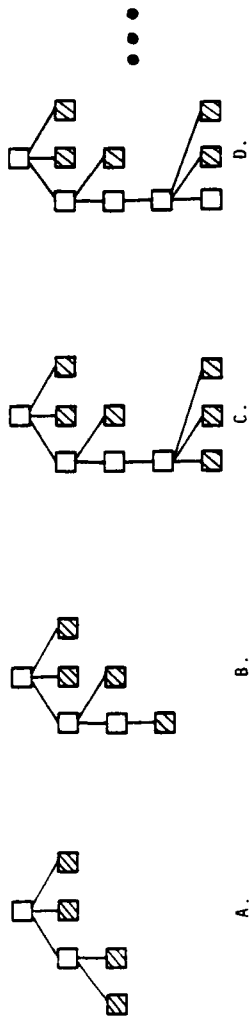
28-261

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

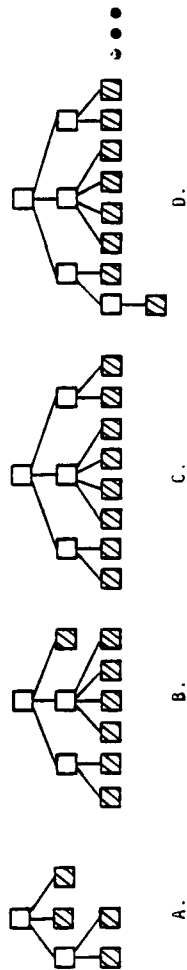
TOP-DOWN INTEGRATION

EXAMPLE

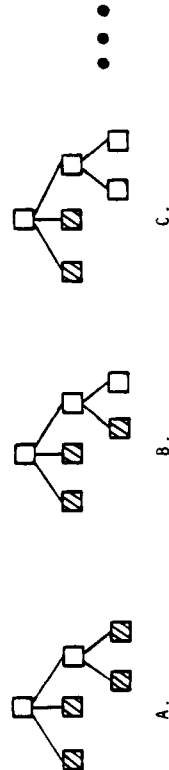
1. DEPTH FIRST (TEST ALL INPUT) ...



2. BREADTH FIRST (TEST MAJOR FUNCTIONS FIRST) ...



3. BACKWARDS DEPTH FIRST (TEST OUTPUT) ...



 = STUB

INSTRUCTOR NOTES

TOP-DOWN INTEGRATION IS THE PREFERABLE APPROACH.

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.
- IMPLEMENTORS GET A BOOST FROM SEEING SOMETHING WORKING.
- CODING AND TESTING CAN BEGIN BEFORE DESIGN IS FINISHED.

INSTRUCTOR NOTES

TOP-DOWN IS ALSO EASIER ON MANAGEMENT.

TOP-DOWN INTEGRATION ADVANTAGES

- MANAGEMENT AND MONITORING IS MORE STRAIGHT FORWARD. FOR EXAMPLE:
 - MANAGERS CAN ESTIMATE RESOURCES REQUIRED TO IMPLEMENT A WHOLE DESIGN BY SUMMING THE ESTIMATES FOR EACH PART OF THE DESIGN.
 - FEEDBACK IS BY SOLID DELIVERABLES RATHER THAN BY NON-OPERATIONAL CODE OF UNKNOWN QUALITY.
 - PROGRESS IS MEASURED IN MORE USEFUL QUANTITY: INCH PEBBLES, NOT MILESTONES.

INSTRUCTOR NOTES

DRIVERS ARE USUALLY HARDER TO WRITE THAN STUBS.

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 HARNESSES) ARE STAND-INS FOR NOT-YET-WRITTEN SUPERORDINATE MODULES.

INSTRUCTOR NOTES

VG 778.1

28-30i

23



2

15

2,341



五

21

23

5

1

3

三

3

1

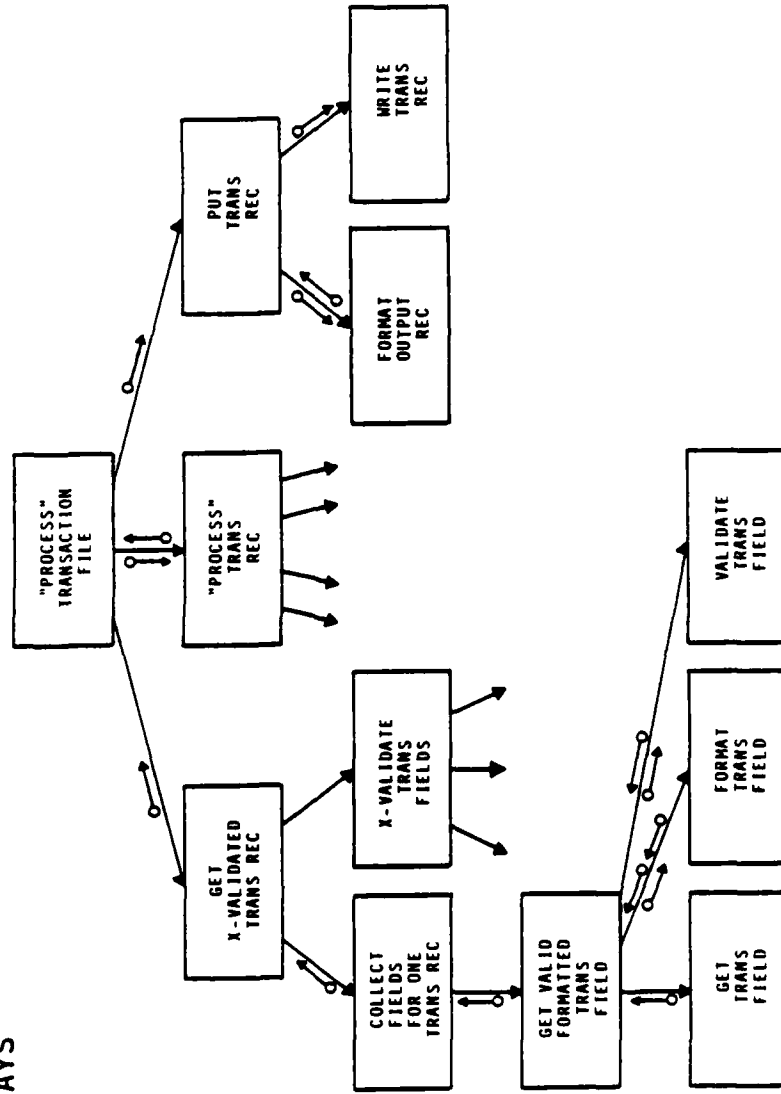
...

2

BOTTOM-UP INTEGRATION

• HOW MIGHT THIS DESIGN BE TESTED BOTTOM UP?

- TWO WAYS

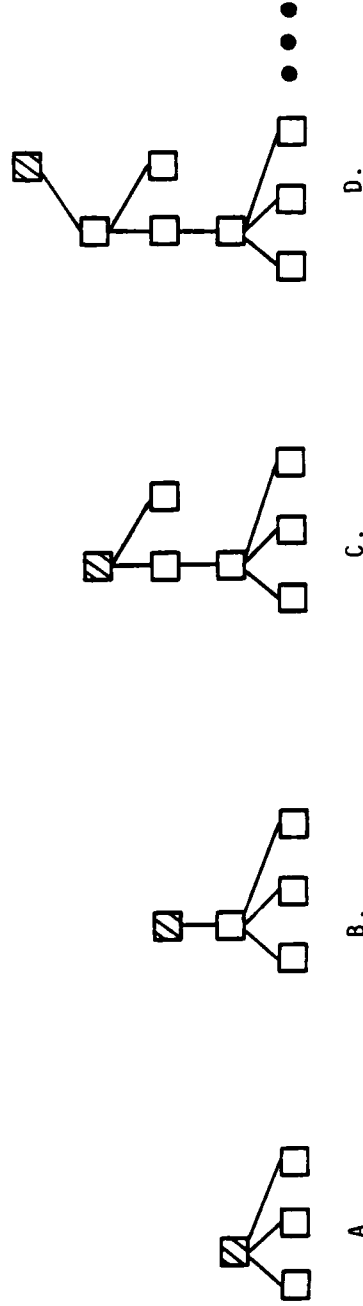


INSTRUCTOR NOTES

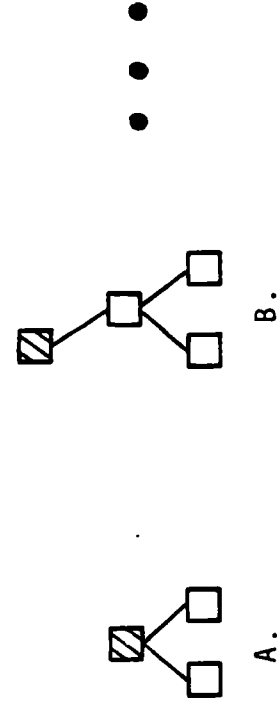
PICK THE MOST SUITED APPROACH.

BOTTOM-UP INTEGRATION

1. TEST DETAILS OF INPUT LOGIC ...



2. TEST DETAILS OF OUTPUT LOGIC ...



 = DRIVER

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 778.1

28-331

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: THE QUALITY OF VARIOUS PRODUCTS DEVELOPED DURING THE SOFTWARE LIFE IS DETERMINED BY THE DEGREE A REVIEW THEY RECEIVE DURING THEIR DEVELOPMENT.

PURPOSE: TO PROVIDE A SET OF GUIDELINES FOR EFFECTIVE REVIEWS.

REFERENCES: FREEDMAN, D., WEINBERG, G. "WALKTHROUGHS, INSPECTIONS, AND TECHNICAL REVIEWS" LITTLE, BROWN, MA; 1977
YOURDON, E. "STRUCTURED WALKTHROUGHS" YOURDON, NYC; 1977

Section 29
METHODS OF REVIEW

VG 778.1

INSTRUCTOR NOTES

THIS AREA COVERS THE FULL RANGE OF REVIEWS DURING THE LIFE CYCLE IN A FAIRLY GENERAL MANNER.

REVIEWS

- QUESTIONS OFTEN ASKED ABOUT REVIEWS

- WHY REVIEW?
- WHO PARTICIPATES?
- HOW ARE REVIEWS ORGANIZED?
- HOW ARE REVIEWS RUN?
- SHOULD REVIEWS REPEAT?
- WHAT KIND OF REVIEWS ARE THERE?
- WHAT SHOULD WE LOOK FOR?

- REMEMBER REVIEWS ARE REQUIRED IN ALL PHASES OF A
SOFTWARE DEVELOPMENT

INSTRUCTOR NOTES

EVEN THOUGH WE DON'T LIKE TO ADMIT IT, EVERYONE MAKES MISTAKES.

WHY REVIEW?

- BECAUSE TO ERR IS HUMAN.
 - REVIEWS ARE ESTABLISHED TECHNIQUES FOR REMOVING ERRORS
- SOFTWARE ENGINEERING IS A MODELING PROCESS. MODELS CONTAIN ERRORS. REVIEW IDENTIFIES AND CORRECTS ERRORS.
- THREE CATEGORIES OF ERRORS CAN EXIST IN MODELS.
 - INACCURACIES - MAKE THE MODEL LESS USEFUL
 - OMISSIONS - MAKE THE MODEL INEFFECTIVE OR DANGEROUS TO USE
 - EXTRAS - GET IN THE WAY, MAKING THE MODEL HARDER TO USE

INSTRUCTOR NOTES

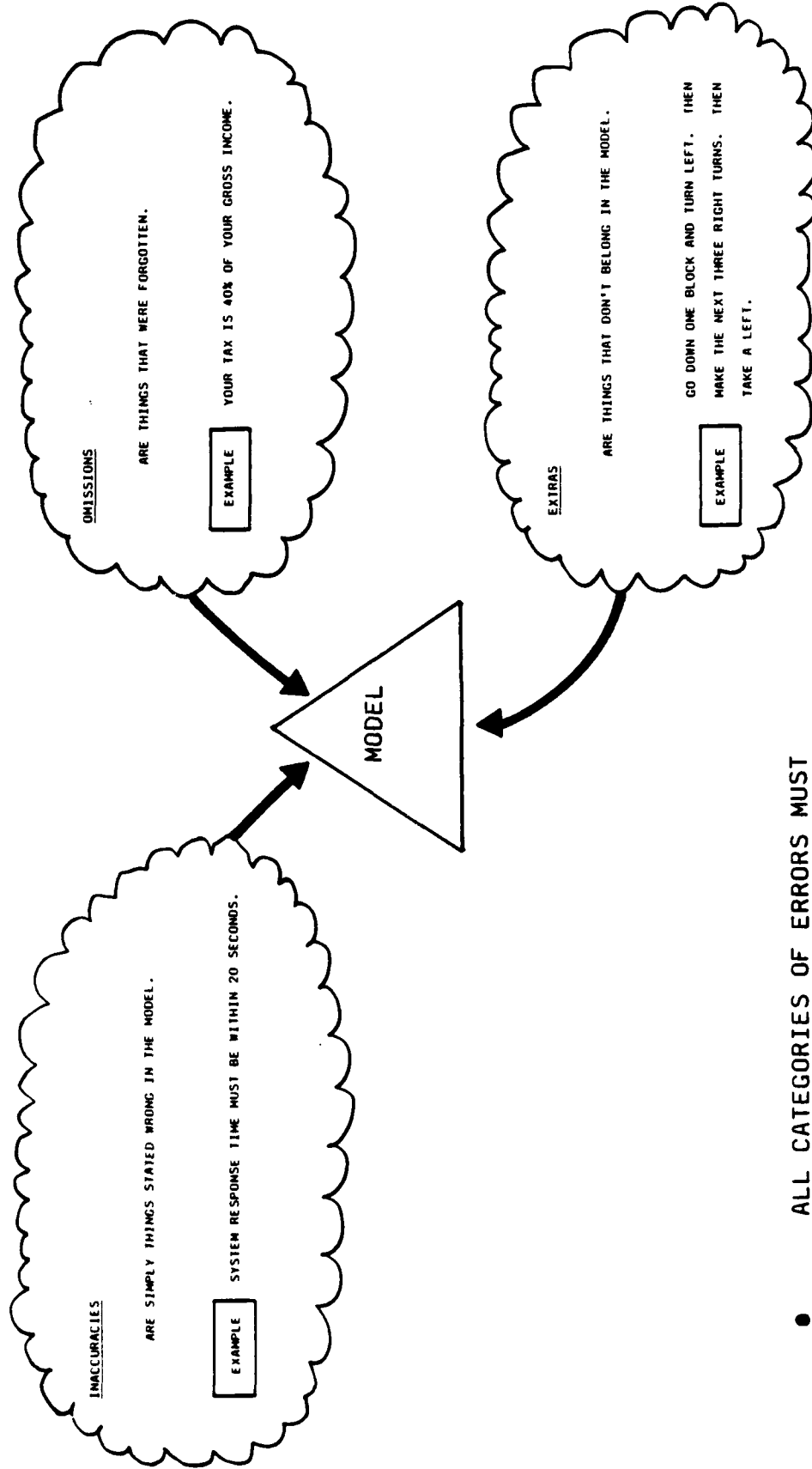
INACCURACIES - WRONG STATEMENTS ARE DANGEROUS (E.G. MISPLACED DECIMAL
POINT BLOWS UP A ROCKET)

OMISSIONS - WHAT YOU FORGOT IS ALWAYS MORE DANGEROUS THAN WHAT YOU
WROTE DOWN (E.G. WHAT HAPPENS IF LIGHTENING HITS THE
AIRPLANE AND THE SOFTWARE FAILS?)

EXTRAS - EXTRA JUNK JUST MAKES THE MODEL HARDER TO READ, OR
CREATES EXTRA WORK.

ALL TYPES OF ERRORS ARE EQUALLY BAD FROM THE REVIEW PROCESS VIEWPOINT.

WHY REVIEW?



- ALL CATEGORIES OF ERRORS MUST BE REMOVED

INSTRUCTOR NOTES

IN MOST REVIEWS THE PARTICIPANTS ARE ASSIGNED THESE ROLES TO HELP FOCUS THE REVIEW ACTIVITY.

DON'T SET UP ORGANIZATIONALLY A "WATCH-DOG" TEAM FOR REVIEW. THEY WILL START TO WORRY ABOUT THEIR ROLE INSTEAD OF THE REVIEW.

WHO PARTICIPATES?

- IN A TYPICAL REVIEW THE FOLLOWING ROLES MUST BE FILLED BY SELECTED PARTICIPANTS.

- THE LEADER
- THE RECORDER
- SOMEONE OLD
- SOMEONE NEW
- SOMEONE BORROWED
- SOMEONE BLUE

- IN OUR DISCUSSION WE ASSUME THE REVIEW IS HELD AS A MEETING NOT AS AN INFORMAL DISCUSSION OR INDIVIDUAL REVIEW.

INSTRUCTOR NOTES

- GOAL - THE LEADER OBTAINS A GOOD REVIEW OR REPORTS WHY GOOD REVIEW WASN'T ATTAINED IF THERE ARE PROBLEMS.
- MANAGERS AND TEAM LEADERS CARRY CONFLICTS OF INTEREST INTO THE REVIEW.
- REQUIRES GOOD MEETING SKILLS.

OFFENSIVE PEOPLE SHOULD NOT LEAD THE REVIEW. BUT AN OCCASIONAL AND WELL-TIMED REMARK OR GESTURE THAT GETS RESULTS IS VERY IMPORTANT.

WHO PARTICIPATES?

• THE LEADER

- FOCUSES ON OBTAINING A GOOD REVIEW
- IS NEVER A MANAGER OR TEAM LEADER
- PREPARES THE REVIEW KITS
- CONDUCTS AN ORDERLY REVIEW
- KEEPS THE MEETING ON TRACK
- MAKES SURE EVERYONE PARTICIPATES
- MAKES SURE ALL IMPORTANT POINTS ARE COVERED

INSTRUCTOR NOTES

THE RECORDER DOESN'T PROVIDE A "BLOW-BY-BLOW" ACCOUNT. HE FOCUSES ON THE REAL ISSUES.

WHO PARTICIPATES?

• THE RECORDER

- PRODUCES AN ACCURATE REPORT OF THE REVIEW SESSION
- MAY TAKE MINUTES (PRIVATE)
- MAY RECORD ON FLIP CHARTS (PUBLIC)
- MAY BE A TECHNICIAN OR A LIBRARIAN
- IS NEVER THE SAME PERSON AT EVERY REVIEW

INSTRUCTOR NOTES

SHOULD BE AN ACTIVE, RESPECTED PERSON. NOT SOMEONE WHO TELLS "WAR-STORIES", OR SAYS
CONSTANTLY "THIS WAS HOW WE USED TO DO IT, TEN YEARS AGO."

WHO PARTICIPATES?

• SOMEONE OLD

- IS A SENIOR TECHNICAL PERSON IN THE PROJECT
- IS A ROLE MODEL FOR JUNIOR STAFF
- PROVIDES TECHNICAL DEPTH
- ACTS IN A TEACHING CAPACITY

INSTRUCTOR NOTES

SHOULD NOT BE A SHY PERSON, AND SHOULD BE MADE TO FEEL HE CAN CONTRIBUTE.

WHO PARTICIPATES?

• SOMEONE NEW

- IS A JUNIOR TECHNICAL PERSON IN THE PROJECT
- LEARNS FROM SENIOR STAFF
- LENDS A FRESH VIEW TO THE MATERIAL
- SOMETIMES CONTRIBUTES NOVEL IDEAS

INSTRUCTOR NOTES

SOMETIMES THIS PERSON IS FROM AN AREA THAT WILL BE EVENTUALLY EFFECTED BY THE SYSTEM
BEING REVIEWED.

WHO PARTICIPATES?

• SOMEONE BORROWED

- COMES FROM OUTSIDE THE PROJECT
- LENDS A FRESH VIEW TO THE MATERIAL
- SOMETIMES CONTRIBUTES NOVEL IDEAS
- LEARNS NEW TECHNIQUES FROM OTHERS

INSTRUCTOR NOTES

REMEMBER, THIS PERSON'S WORK IS UNDER REVIEW, NOT ATTACK. NOR IS HE GIVING A PRESENTATION.

WHO PARTICIPATES?

• SOMEONE BLUE

- IS THE PERSON WHO'S WORK IS BEING REVIEWED
- IS THE AUTHORITY ON THE WORK
- ANSWERS QUESTIONS ABOUT THE MATERIAL
- LEARNS ABOUT THE WORK'S QUALITY

INSTRUCTOR NOTES

TEAM LEADERS MUST NEVER CONDUCT A REVIEW OF THEIR OWN TEAM'S WORK.

HOW ARE REVIEWS ORGANIZED?

- TWO VERY DIFFERENT WAYS ...
 - AS A TEAM - RESPONSIBLE FOR OBTAINING A CONSENSUS ON THE QUALITY OF WORK BEING REVIEWED.
 - AS A GROUP INDIVIDUALS - RESPONSIBLE FOR GIVING THEIR TECHNICAL OPINIONS AND ADVICE TO THE AUTHOR.
- TEAM CHARACTERISTICS
 - MEET AND REACH CONSENSUS
 - COMMUNICATE VERBALLY
 - PUT TOGETHER SPECIFICALLY TO REACH A PROJECT GOAL
- INDIVIDUAL CHARACTERISTICS
 - GIVE WRITTEN FEEDBACK TO AUTHOR(S)
 - REVIEW AT AN AUTHOR'S DISCRETION

INSTRUCTOR NOTES

THIS SECTION FOCUSES ON BOTH TEAM AND INDIVIDUAL REVIEWS.

HOW ARE REVIEWS RUN?

• REVIEWS ARE RUN LIKE SUCCESSFUL MEETINGS; THEY ...

- ARE PLANNED
- HAVE PREPARED MATERIALS
- HAVE INFORMED PARTICIPANTS
- ARE STRUCTURED AND CONTROLLED
- ARE RECORDED

INSTRUCTOR NOTES

EACH STEP IS VERY IMPORTANT. EACH IS NECESSARY, NONE ARE SUFFICIENT.

HOW ARE REVIEWS RUN?

PLANNING

- ESTABLISHES THE GOAL OF THE REVIEW
- DEFINES WHY THE GOAL MUST BE REACHED
- MEANS MATERIALS ARE PACKAGED AND SENT TO PARTICIPANTS IN A TIMELY MANNER
- MEANS EVERYONE CAN PREPARE FOR THE REVIEW

PREPARING

- MEANS EVERYONE KNOWS ...

INFORMING

- WHY THE REVIEW IS BEING HELD
- WHERE IT WILL TAKE PLACE
- WHEN IT WILL OCCUR
- WHAT WILL BE COVERED
- WHO WILL BE THERE

STRUCTURING

- MEANS THAT THERE ARE PROCEDURES THAT GUIDE EVERYONE'S THOUGHTS TO A COMMON GOAL ...
- PRESENT THE EVIDENCE
- POLL FOR INTERPRETATIONS
- DEFINE A PLAN OF ACTION

CONTROLLING

- MEANS THE MEETING STAYS ON TRACK ...
- NO ARGUMENTS
- NO UNNECESSARY TANGENTS
- RULE OUT THE OVERLY CRITICAL
- RULE OUT THE OVERLY PRAISEWORTHY
- PROPER ALLOCATION OF TIME TO EACH MATTER ON THE AGENDA

RECORDING

- MEANS EVERYONE REMEMBERS ...
- RESOLUTIONS
- ACTION ITEMS
- THE MEETING'S HISTORY

INSTRUCTOR NOTES

SINCE NO ONE LIKES TO REPEAT REVIEWS, THE PARTICIPANTS MUST BE ENCOURAGED TO SAY "OKAY, IT'S GOOD UP TO THIS POINT. FIX THE REST, AND WE'LL REVIEW IT AGAIN."

INDIVIDUAL REVIEWS NORMALLY ONLY CONCENTRATE ON SMALL PORTIONS OF AN AUTHOR'S WORK.

SHOULD REVIEWS REPEAT?

● TEAM REVIEWS REPEAT ONLY IF ...

- A QUORUM IS NOT ACHIEVED
- TIME LIMIT WAS EXCEEDED
- THE PRODUCT REQUIRES MAJOR WORK
- A CONSENSUS WAS NOT ACHIEVED

● INDIVIDUAL REVIEWS REPEAT ...

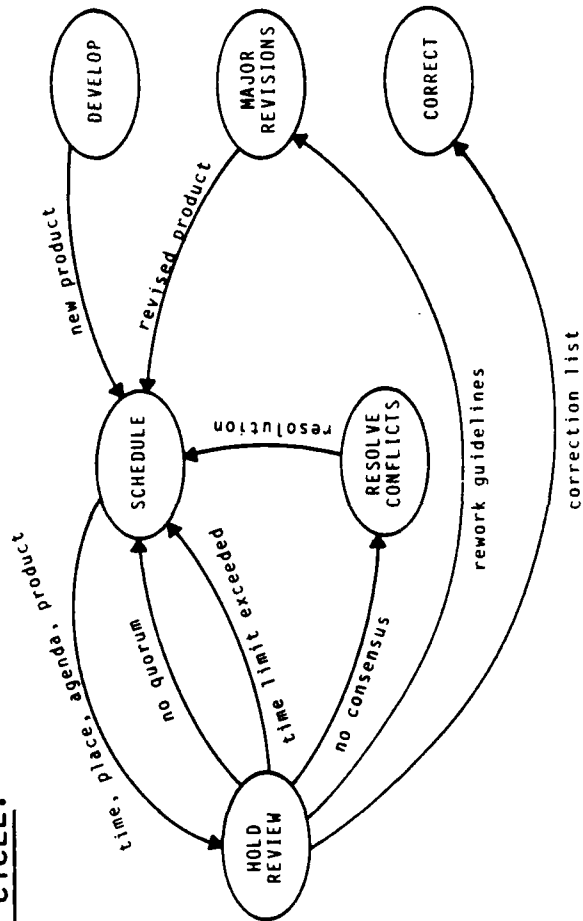
- RARELY ONCE
- USUALLY BETWEEN 2 AND 4 TIMES
- UNTIL THE AUTHOR HAS ACHIEVED CONSENSUS
WITH REVIEWERS

INSTRUCTOR NOTES

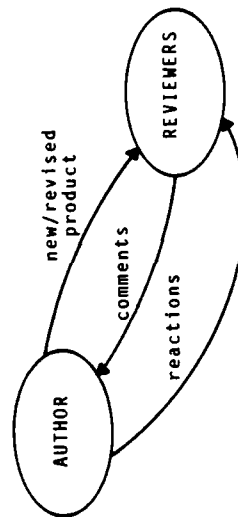
NOTE THE MULTIPLE DYNAMICS OF THE TEAM REVIEW.

SHOULD REVIEWS REPEAT?

• THE TEAM REVIEW CYCLE:



• THE INDIVIDUAL REVIEW CYCLE:



INSTRUCTOR NOTES

THE GENERIC ASPECTS OF REVIEW ARE ALWAYS PRESENT, BUT MUST BE TAILORED TO THE PRODUCTS
AND CIRCUMSTANCE ENCOUNTERED.

WHAT KINDS OF REVIEWS ARE THERE?

- AS MANY AS THERE ARE PRODUCTS PRODUCED BY THE PROJECT, AT LEAST ...

- SPECIFICATION REVIEWS
- DESIGN REVIEWS
- CODE REVIEWS
- TEST PLAN REVIEWS
- DOCUMENTATION REVIEWS

- FOR EACH REVIEW YOU SHOULD CONSIDER

- WHY IT'S DONE
- WHO ATTENDS
- WHAT CRITERIA EVALUATES QUALITY
- HOW ITS GOALS ARE TO BE ACHIEVED

INSTRUCTOR NOTES

SPECIFICATION REVIEWS ARE VERY IMPORTANT TO KEEP THE PROJECT FROM POSSIBLY MAKING GRIEVOUS ERRORS LATER ON.

WHAT KINDS OF REVIEWS ARE THERE?

SPECIFICATION REVIEWS

WHY

TO CATCH OMISSIONS AND MISINTERPRETATIONS OF USER REQUESTS.

WHO

USERS, ANALYSTS AND DESIGNERS ATTEND.

WHAT

COMPLETENESS, CORRECTNESS, CONSISTENCY, PRECISION, TRACEABILITY

HOW

- CONCENTRATE ON A PARTICULAR USER VIEWPOINT (I.E. FACTION).

- REMOVE AMBIGUITIES BY:

- REREADING WITH DIFFERENT STRESS
- SUBSTITUTING SYNONYMS
- REDRAWING
- WORK OUT AN EXAMPLE

INSTRUCTOR NOTES

DESIGN REVIEWS ARE USUALLY THE LAST PLACE TO CATCH ERRORS BEFORE THE "RUBBER MEETS THE ROAD."

WHAT KIND OF REVIEWS ARE THERE?

DESIGN REVIEWS

WHY

TO VERIFY ALL REQUIREMENTS ARE SATISFIED IN A FEASIBLE WAY.

WHO

ANALYSTS, DESIGNERS AND IMPLEMENTERS.

WHAT

SUBSYSTEM AND USER INTERFACES, UNNECESSARY CONSTRAINTS, INFEASIBLE STRATEGIES.

HOW

- COMPARE ALTERNATE STRATEGIES.
- ESTABLISH EVALUATION METRICS.
- RECORD ISSUES AND POSSIBLE ALTERNATE SOLUTIONS.

INSTRUCTOR NOTES

BY THIS TIME, THE REVIEW SHOULD ONLY BE CONCERNED WITH THE CODE, NOT THE DESIGN. IF DESIGN PROBLEMS APPEAR, ANOTHER DESIGN REVIEW IS PROBABLY NECESSARY.

WHAT KIND OF REVIEWS ARE THERE?

CODE REVIEWS

WHY

TO VERIFY CORRECT IMPLEMENTATION OF THE DESIGN.

WHO

DESIGNERS, IMPLEMENTERS, MAINTAINERS

WHAT

SUBPROGRAM INTERFACES, DATA STRUCTURES, MACHINE UTILIZATION,
STYLE, COMMENTS.

HOW

- ADHERENCE TO STANDARDS.
- DESCRIPTIONS (E.G. PACKAGE SPECS) BEFORE WORKINGS
(E.G. PACKAGE BODIES).
- TRACE BACK TO DESIGN AND TO SPECIFICATION.

INSTRUCTOR NOTES

THIS SHOULD BE THE TIME TO VERIFY THAT THE CORRECT TEST STRATEGIES HAVE BEEN SELECTED.

WHAT KIND OF REVIEWS ARE THERE?

TEST PLAN REVIEWS

WHY

TO VERIFY TEST DETAILS SATISFY ACCEPTANCE CRITERIA.

WHO

ANALYSTS, USERS, DESIGNERS, IMPLEMENTERS, MAINTAINERS.

WHAT

A TEST FOR: EVERY REQUIREMENT, SYSTEM BEHAVIOR, SYSTEM FUNCTION,
ERROR SITUATION, BACKUP AND RECOVERY SITUATION

HOW

- TRACE BACK TO SPECIFICATION, DESIGN AND CODE.
- CLUSTER TESTS INTO CATEGORIES (TO VERIFY COVERAGE).

INSTRUCTOR NOTES

THESE REVIEWS ARE OFTEN SKIMMED OVER, BUT ARE VERY NECESSARY. IF THE DOCUMENTATION IS POOR OR INCOMPLETE, AT MAINTENANCE TIME THE COST OF FIXES WILL ASSUREDLY SKYROCKET.

WHAT KIND OF REVIEWS ARE THERE?

DOCUMENTATION REVIEWS

WHY

TO ENSURE TECHNICAL CORRECTNESS ON HOW THE SYSTEM IS OPERATED
AND MAINTAINED.

WHO

USERS, MAINTAINERS, IMPLEMENTERS, TECHNICAL WRITERS.

WHAT

EVIDENCE OF OPERATIONAL TESTS, ERROR MESSAGES, USER INTERFACE
DESCRIPTION, INSTALLATION.

HOW

- TRACE BACK TO SPECIFICATION.
- VERIFY TEST CASE FOR EACH USER INTERFACE FEATURE.
- FORM ON ERROR LIST AND ENFORCE ASSOCIATED CORRECTIONS.

INSTRUCTOR NOTES

THERE IS TOO MUCH TO COVER IN THIS LAST SECTION, SO HIT WHATEVER IMPORTANT POINTS:

(1) YOU WANT TO MAKE, AND (2) THAT ARE APPROPRIATE FOR YOUR AUDIENCE. TELL STUDENTS TO CONSULT THE FREEDMAN AND WEINBERG TEXT FOR MORE DETAILS.

POINT OUT THAT MOST ORGANIZATIONS WILL DEVELOP FROM THESE TYPES OF CHECKLISTS A UNIQUE SET OF GUIDELINES.

WHAT SHOULD WE LOOK FOR?

● SPECIFICATION REVIEW CHECKLIST/TECHNIQUES ...

1. VARY THE STRESS PATTERN IN A SENTENCE TO REVEAL POSSIBLE ALTERNATIVE MEANINGS.
2. WHEN A TERM IS DEFINED EXPLICITLY SOMEWHERE, TRY SUBSTITUTING THAT DEFINITION IN PLACE OF THE TERM.
3. WHEN A STRUCTURE IS DESCRIBED IN WORDS, TRY TO SKETCH A PICTURE OF THE STRUCTURE BEING DESCRIBED.
4. WHEN A STRUCTURE IS DESCRIBED BY A PICTURE, TRY TO REDRAW THE PICTURE IN A FORM THAT EMPHASIZES DIFFERENT ASPECTS.
5. WHEN THERE IS AN EQUATION, TRY EXPRESSING THE MEANING OF THE EQUATION IN WORDS.
6. WHEN A CALCULATION IS SPECIFIED OR IMPLIED IN WORDS, TRY EXPRESSING IT IN AN EQUATION.
7. WHEN A CALCULATION IS SPECIFIED, WORK AT LEAST TWO EXAMPLES BY HAND AND GIVEN THEM AS EXAMPLES IN THE SPECIFICATION.
8. LOOK FOR STATEMENTS THAT IN ANY WAY IMPLY CERTAINTY AND THEN ASK FOR PROOF.
9. WHEN YOU ARE SEARCHING BEHIND CERTAINTY STATEMENTS, PUSH THE SEARCH BACK AS MANY LEVELS AS ARE NEEDED TO ACHIEVE THE KIND OF CERTAINTY A COMPUTER WILL NEED.
10. BE ON THE LOOKOUT FOR WORDS THAT ARE SUPPOSED TO BE PERSUASIVE, SUCH AS CERTAINLY, THEREFORE, CLEARLY, OBVIOUSLY, OR AS ANY FOOL CAN PLAINLY SEE.
11. WATCH FOR VAGUE WORDS, SUCH AS SOME, SOMETIMES, OFTEN, USUALLY, ORDINARILY, CUSTOMARILY, MOST, OR MOSTLY.
12. WHEN LISTS ARE GIVEN, BUT NOT COMPLETED, MAKE SURE THAT THERE IS A COMPLETE UNDERSTANDING OF THE NATURE OF THE SUBSEQUENT ITEMS. WATCH OUT FOR ETC., AND SO FORTH, AND SO ON, OR SUCH AS.
13. IN ATTEMPTING TO CLARIFY LISTS, AS IN (12), WE SOMETIMES STATE A RULE. BE SURE THAT THE RULE DOESN'T CONTAIN UNSTATED ASSUMPTIONS.
14. THE ABOVE PROBLEMS (11, 12, AND 13) MIGHT HAVE BEEN CAUGHT BY LOOKING FOR LISTS WITHOUT EXAMPLES OR EXAMPLES THAT ARE TOO FEW OR TOO SIMILAR TO EACH OTHER TO EXPLICATE THE RULE.
15. BEWARE OF VAGUE VERBS, SUCH AS HANDLED, PROCESSED, REJECTED, SKIPPED, OR ELIMINATED.
16. PASSIVE VOICE CONSTRUCTIONS ARE ALSO TRAPS. SINCE THE PASSIVE VOICE DOESN'T NAME AN ACTOR, IT'S EASY TO OVERLOAD HAVING ANYBODY DO THE WORK.
17. BE ESPECIALLY ON THE LOOKOUT FOR COMPARATIVES WITHOUT REFERENTS. ALWAYS ASK, "COMPARED WITH WHAT?"
18. PRONOUNS ARE OFTEN CLEAR TO THE WRITER AND NOT TO THE READER. KEY ON THE OCCURRENCES OF IT, HE, SHE, THEY, HIS, HERS, ITS, THEIR, WE, US, OUR, YOU, YOUR, AND TRY TO FIND THE REFERENT AND STATE IT EXPLICITLY.

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

• ARCHITECTURAL DESIGN REVIEW CHECKLIST ...

1. ARE THE PRELIMINARY DESIGN OBJECTIVES CLEARLY STATED?
2. DOES THE PRELIMINARY DESIGN DOCUMENT CONTAIN A DESCRIPTION OF THE PROCEDURE THAT WAS USED TO DO PRELIMINARY DESIGN OR IS THERE A REFERENCE TO SUCH A PROCEDURE?
3. IS THERE A LIST OF THE FUNCTIONS THAT ARE TO BE PROVIDED BY THE COMPUTING SYSTEM?
4. IS THERE A MODEL OF THE USER INTERFACE TO THE COMPUTING SYSTEM?
5. ARE THERE MODELS AND/OR DESCRIPTIONS OF ALL OTHER INTERFACES TO THE COMPUTING SYSTEM?
6. IS THERE A HIGH LEVEL FUNCTIONAL MODEL OF THE PROPOSED COMPUTING SYSTEM?
7. ARE THE MAJOR IMPLEMENTATION ALTERNATIVES AND THEIR EVALUATIONS REPRESENTED IN THE DOCUMENT?
8. IS THERE A RECOMMENDATION FROM THE PRELIMINARY DESIGN TEAM TO IMPLEMENT ONE OF THE ALTERNATIVES?
9. IS THE RECOMMENDATION OF THE PRELIMINARY DESIGN TEAM ADEQUATELY SUPPORTED?
10. DOES THE INFORMATION PRESENTED IN THE PRELIMINARY DESIGN DOCUMENT AND DURING THE PRELIMINARY DESIGN REVIEW GIVE YOU CONFIDENCE THAT THE COMPUTING SYSTEM CAN BE IMPLEMENTED TO SATISFY THE REQUIREMENTS TO SUCH AN EXTENT THAT YOU WOULD USE THE PROPOSED SYSTEM?

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

DESIGN REVIEW CHECKLISTS

1. BOUNDARY OVERSIGHTS

- a. IS ANYTHING GOING TO FALL BETWEEN THE CRACKS OF "MINE" VERSUS "YOURS."
- b. IS ANYTHING GOING TO BE CLAIMED BY TWO OR MORE PARTIES?
- c. IS EACH INPUT, FUNCTION, AND OUTPUT SPECIFICALLY ADDRESSED BY A SPECIFIC, IDENTIFIABLE PART OF THE SYSTEM? CAN YOU PROVE IT?
- d. IS THERE ANY MISINTERPRETATION OF THE PERSON-MACHINE INTERFACE -- EITHER BY PERSON OR MACHINE?

2. OVERADAPTATION

- a. HAS ANY PORTION OF THIS DESIGN RECEIVED MORE EMPHASIS THAN IT SEEMS TO DESERVE? CAN YOU EXPLAIN WHY THAT HAPPENED, AND WHAT EFFECTS IT HAS HAD?
- b. IS THE DESIGN OVERLY CONSTRAINED, PERHAPS BY PAYING TOO MUCH ATTENTION TO ONE PART AT THE EXPENSE OF OTHERS?
- c. IF YOU COULD RELAX ANY SINGLE CONSTRAINT, WHICH WOULD IT BE? HOW WOULD THE DESIGN BE AFFECTED?

3. AFTERTHOUGHTS

EXAMINE THE LAST THREE THINGS ADDED TO THE DESIGN AND ANSWER THE FOLLOWING QUESTIONS FOR EACH:

- a. WHAT HAS BEEN CRAMMED IN?
- b. COULD SOMEONE TELL THAT THE CHANGE WAS NOT PART OF THE ORIGINAL CONCEPTION -- THAT IT IS A PATCH TO THE DESIGN?
- c. WHAT WASN'T CONSIDERED WHEN THIS CHANGE WAS MADE?
- d. WHAT WOULD HAPPEN IF THIS CHANGE WERE UNDONE AND LEFT OUT OF THE FINAL DESIGN?

4. VESTIGES

- a. WHAT THINGS ARE IN THE DESIGN BECAUSE "WE'VE ALWAYS DONE IT THAT WAY?" WHY ARE YOU DOING IT THAT WAY?
- b. DOES THE DESIGN REFLECT THE MACHINE ON WHICH IT WILL OPERATE? IF SO, WHY?
- c. IS YOUR DESIGN INDEPENDENT OF THE PROGRAMMING LANGUAGE THAT WILL BE USED? IF NOT, WHY NOT?

5. MISTAKES

- a. WHAT HAVE YOU FORGOTTEN?
- b. WHAT HAS BEEN DONE WRONG?
- c. DID YOU DO THE I'S?
- d. DID YOU EVER GO BACK AND CORRECT THAT PROBLEM YOU FOUND WHEN YOU WERE BUSY WITH SOMETHING MORE IMPORTANT?
- e. DO YOU HAVE ANY NOTES ON SCRAPS OF PAPER?

6. INSENSITIVITY

- a. HAVE YOU REMEMBERED THE PEOPLE WHO WILL HAVE TO USE THIS SYSTEM?
- b. HAVE YOU REMEMBERED THE PEOPLE WHO WILL HAVE TO OPERATE IT?
- c. HAVE YOU REMEMBERED THE PEOPLE WHO WILL HAVE TO REPAIR IT?
- d. IF YOU WERE ONE OF THESE PEOPLE, WHAT ONE THING WOULD YOU CHANGE IN THE DESIGN TO MAKE YOUR LIFE EASIER? WHY WASN'T THAT CHANGE MADE?

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

• CODE REVIEW CHECKLIST/QUESTIONS

FUNCTION

1. IS THERE A CONCEPT, AN UNDERLYING IDEA, THAT CAN BE EXPRESSED EASILY IN PLAIN LANGUAGE? IS IT EXPRESSED IN PLAIN LANGUAGE IN THE IMPLEMENTED CODE?
2. DOES THE FUNCTION OF THIS PART HAVE A CLEAR PLACE IN THE OVERALL FUNCTION OF THE WHOLE, AND IS THIS FUNCTION CLEARLY EXPRESSED?
3. IS THE ROUTINE PROPERLY SHELTERED, SO THAT IT MAY PERFORM ITS FUNCTION RELIABLY IN SPITE OF POSSIBLE MISUSE?

FORM

1. WHATEVER STYLE IS ADOPTED, IS IT CLEAN AND CLEAR WHEN TAKEN AS A WHOLE?
2. IS IT MEANINGFUL TO ALL CLASSES OF READERS WHO WILL SEE IT?
3. ARE THERE REPEATED CODE SEGMENTS, WHETHER WITHIN OR BETWEEN ROUTINES?
4. ARE COMMENTS USEFUL OR ARE THEY SIMPLY ALIBIS FOR POOR CODING?
5. IS THE LEVEL OF DETAIL CONSISTENT?
6. ARE STANDARD PRACTICES USED?
7. IS INITIALIZATION PROPERLY DONE, AND DOES THE ROUTINE CLEAN UP AFTER ITSELF?

ECONOMY

1. ARE THERE REDUNDANT OPERATIONS FOR WHICH THERE IS NO COMPENSATING BENEFIT?
2. IS STORAGE USE CONSISTENT, BOTH INTERNALLY AND WITH EXTERNAL SPECIFICATIONS?
3. HOW MUCH WILL IT COST TO MODIFY? (CONSIDER THE THREE MOST LIKELY FUTURE MODIFICATIONS.)
4. IS IT SIMPLE?

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

CODE REVIEW STYLE GUIDELINES

- WRITE CLEARLY -- DON'T BE TOO CLEVER.
- SAY WHAT YOU MEAN, SIMPLY AND DIRECTLY.
- USE LIBRARY FUNCTIONS.
- AVOID TEMPORARY VARIABLES.
- WRITE CLEARLY -- DON'T SACRIFICE CLARITY FOR "EFFICIENCY"
- LET THE MACHINE DO THE DIRTY WORK.
- REPLACE REPETITIVE EXPRESSIONS BY CALLS TO A COMMON FUNCTION.
- PARENTHESIZE TO AVOID AMBIGUITY
- CHOOSE VARIABLE NAMES THAT WON'T BE CONFUSED.
- AVOID UNNECESSARY BRANCHES.
- DON'T USE CONDITIONAL BRANCHES AS A SUBSTITUTE FOR A LOGICAL EXPRESSION.
- IF A LOGICAL EXPRESSION IS HARD TO UNDERSTAND, TRY TRANSFORMING IT.
- USE DATA ARRAYS TO AVOID REPETITIVE CONTROL SEQUENCES.
- CHOOSE A DATA REPRESENTATION THAT MAKES THE PROGRAM SIMPLE.
- WRITE FIRST IN AN EASY-TO-UNDERSTAND PSEUDO-LANGUAGE; THEN TRANSLATE INTO WHATEVER LANGUAGE YOU HAVE TO USE.
- MODULARIZE. USE SUBROUTINES.
- USE GOTOS ONLY TO IMPLEMENT A FUNDAMENTAL STRUCTURE.
- AVOID GOTOS COMPLETELY IF YOU CAN KEEP THE PROGRAM READABLE.
- DON'T PATCH BAD CODE -- REWRITE IT.
- WRITE AND TEST A BIG PROGRAM IN SMALL PIECES.
- USE RECURSIVE PROCEDURES FOR RECURSIVELY-DEFINED DATA STRUCTURES.
- TEST INPUT FOR PLAUSIBILITY AND VALIDITY.
- MAKE SURE INPUT DOESN'T VIOLATE THE LIMITS OF THE PROGRAM.
- TERMINATE INPUT BY END-OF-FILE OR MARKER, NOT BY COUNT.
- IDENTIFY BAD INPUT; RECOVER IF POSSIBLE.
- MAKE INPUT EASY TO PREPARE AND OUTPUT SELF-EXPLANATORY.
- USE UNIFORM INPUT FORMATS.
- MAKE INPUT EASY TO PROOFREAD.
- USE FREE-FORM INPUT WHEN POSSIBLE.
- USE SELF-IDENTIFYING INPUT. ALLOW DEFAULTS. ECHO BOTH ON OUTPUT.
- MAKE SURE ALL VARIABLES ARE INITIALIZED BEFORE USE.
- DON'T STOP AT ONE BUG.
- TAKE CARE TO BRANCH THE RIGHT WAY ON EQUALITY.
- BE CAREFUL WHEN A LOOP EXITS TO THE SAME PLACE FROM SIDE AND BOTTOM.
- MAKE SURE YOUR CODE "DOES NOTHING" GRACEFULLY.
- TEST PROGRAMS AT THEIR BOUNDARY VALUES.
- CHECK SOME ANSWERS BY HAND.
- 10.0 TIMES 0.1 IS HARDLY EVER 1.0.
- DON'T COMPARE FLOATING POINT NUMBERS SOLELY FOR EQUALITY.
- MAKE IT RIGHT BEFORE YOU MAKE IT FASTER.
- MAKE IT FAIL-SAFE BEFORE YOU MAKE IT FASTER.
- MAKE IT CLEAR BEFORE YOU MAKE IT FASTER.
- DON'T SACRIFICE CLARITY FOR SMALL GAINS IN "EFFICIENCY."
- LET YOUR COMPILER DO THE SIMPLE OPTIMIZATIONS.
- DON'T STRAIN TO RE-USE CODE; REORGANIZE INSTEAD.
- MAKE SURE SPECIAL CASES ARE TRULY SPECIAL.
- KEEP IT SIMPLE TO MAKE IT FASTER.
- DON'T DIDDLE CODE TO MAKE IT FASTER -- FIND A BETTER ALGORITHM.
- INSTRUMENT YOUR PROGRAMS. MEASURE BEFORE MAKING "EFFICIENCY" CHANGES.
- MAKE SURE COMMENTS AND CODE AGREE.
- DON'T JUST ECHO THE CODE WITH COMMENTS -- MAKE EVERY COMMENT COUNT.
- DON'T COMMENT BAD CODE -- REWRITE IT.
- USE VARIABLE NAMES THAT MEAN SOMETHING.
- USE STATEMENT LABELS THAT MEAN SOMETHING.
- FORMAT A PROGRAM TO HELP THE READER TO UNDERSTAND IT.
- DOCUMENT YOUR DATA LAYOUTS.

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

● DOCUMENTATION REVIEW CHECKLIST ...

1. HAVE ALL PHASES OF THE DOCUMENT'S LIFE CYCLE BEEN CONSIDERED?
 - a. IS THERE PROVISION FOR USER FEEDBACK?
 - b. IS THERE PROVISION FOR MAKING CHANGES?
 - c. WILL CHANGES IN THE SYSTEM CAUSE DIFFICULT OR EXPENSIVE CHANGES IN THE DOCUMENTATION?
 - d. IS THERE ADEQUATE PROVISION FOR DISTRIBUTION OF THE DOCUMENTS?
 - e. IS THERE ADEQUATE PROVISION FOR THE DISTRIBUTION OF CHANGES TO THE DOCUMENTS?
 - f. CAN DOCUMENTS BE REPRODUCED EASILY?
 - g. CAN COPYING BE PREVENTED OR CONTROLLED?
 - h. ARE THERE AVAILABLE PEOPLE TO SUPPLEMENT DOCUMENTS?
 - i. DO THE USER AND CREATORS AGREE ON THE PURPOSE OF THE DOCUMENTS?
 - j. IS THERE ADEQUATE PROVISION FOR KEEPING SUPPORT PEOPLE CURRENT AND INFORMED?
 - k. ARE TOOLS AVAILABLE (E.G., FICHE READERS, TERMINALS) FOR READING/ACCESSING/STORING THESE MATERIALS?
 - l. HAVE THE DOCUMENTS BEEN PROPERLY APPROVED?
 - m. DO THESE DOCUMENTS SHOW WHERE THEY FALL IN THE TOTAL PLAN?
 - n. DO THE DOCUMENTS INDICATE OTHER DOCUMENTS THAT MAY BE USED AS FOLLOW-UP?
2. ARE THE CONTENTS OF THE DOCUMENTS ADEQUATE?
 - a. COVERAGE OF TOPICS
 - b. CORRECTNESS
 - c. EVIDENCE
3. ARE THE MATERIALS IN THE DOCUMENTS CLEAR?
 - a. ARE EXAMPLES CLEAR?
 - b. ARE DIAGRAMS, PICTURES, OR OTHER VISUAL MATERIALS CLEAR?
 - c. IS TERMINOLOGY CLEAR?
 - d. IS WRITING STYLE CLEAR?
4. ARE THE DOCUMENTS ADEQUATELY SUPPLIED WITH REFERENCING AIDS?
 - a. IS THERE A TABLE OF CONTENTS, IF APPROPRIATE?
 - b. IS THE TABLE OF CONTENTS WELL PLACED?
 - c. IS THERE AN INDEX, IF APPROPRIATE?
 - d. IS THE INDEX WELL PLACED?
 - e. IS THE TABLE OF CONTENTS CORRECT?
 - f. IS THE INDEX CORRECT?
 - g. IS THERE A BIBLIOGRAPHY OF PREREQUISITE PUBLICATIONS?
 - h. IS THERE A BIBLIOGRAPHY OF RELATED PUBLICATIONS WHICH MAY CONTAIN FURTHER INFORMATION?
 - i. DOES THE ORGANIZATION OF THE DOCUMENTS THEMSELVES CONTRIBUTE TO THE EASE OF FINDING INFORMATION?

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

● SYSTEM TESTING REVIEW CHECKLIST ...

I. TEST PREPARATION

1. TEST TEAM MEMBERS GIVEN ASSIGNMENTS.
2. FINAL OPERATING ENVIRONMENT DEFINED.
3. ALL CONDITIONS TO BE TESTED, IDENTIFIED, AND DOCUMENTED.
4. TEST FILES CREATED FOR EXECUTION OF ALL SYSTEM FUNCTIONS.
5. TEST TRANSACTIONS DEVELOPED TO DEMONSTRATE ALL TEST CASES.
6. TEST RESULTS PREDICTED AND DOCUMENTED FOR LATER COMPARISON WITH COMPUTER OUTPUT.
7. PARALLEL SYSTEM OUTPUT IDENTIFIED FOR STANDARD OF CORRECTNESS.
8. ACCEPTABILITY AND ACCURACY STANDARDS ESTABLISHED AND DOCUMENTED FOR LATER ASSESSMENT OF RESULTS.

II. TEST OPERATIONS

1. TEST TRANSACTIONS ENTERED BY DATA ENTRY DEPARTMENT USING STANDARD PROCEDURES.
2. TESTS EXECUTED BY OPERATIONS PERSONNEL WITHOUT PROGRAMMER SUPERVISION.
3. TRANSACTION LISTING OBTAINED BEFORE TEXT EXECUTION.
4. FILE DUMPS OBTAINED PRIOR TO PROGRAM EXECUTION.
5. SYSTEM EXECUTED TO DEMONSTRATE ALL TEST CASES. (MAY REQUIRE MULTIPLE EXECUTIONS OF THE SYSTEM OR OF INDIVIDUAL MODULES.)
6. ALL HARDCOPY (REPORTS, CONTROL TOTALS) OBTAINED.
7. TRANSCRIPTION OF CONSOLE MESSAGES (AND TIMING INFORMATION) OBTAINED.
8. FILE DUMPS TAKEN AFTER TEST EXECUTION.
9. OUTPUT FROM PARALLEL RUN AVAILABLE.

III. UNIT-TEST EVALUATION

1. JCL EXECUTES PROGRAM WITHOUT ERRORS OR UNNECESSARY OPERATOR INTERVENTION, AND FILE LABELS ARE CORRECT.
2. JCL-DEFINED FILE CAPACITIES ADEQUATE, ASSUMING MODERATE EXPANSION.
3. EXECUTE TIME ACCEPTABLE ACCORDING TO PREESTABLISHED STANDARDS.
4. SUFFICIENT CORE AVAILABLE FOR EXECUTION AND MODERATE EXPANSION OF PROGRAM SIZE.
5. INPUT DATA ACCEPTED AS FORMATTED WITHOUT DATA EXCEPTION.
6. ALL LOGICAL PATHS EXECUTED CORRECTLY.
7. CREATED FILES CONFORM TO DESIGN SPECIFICATIONS.
8. FIELD SIZES IN FILES ADEQUATE: NO UNEXPECTED TRUNCATION, LOSS OF SIGNIFICANT DIGITS.
9. MATHEMATICAL ACCURACY, ROUNDING.
10. REPORTS CONFORM TO DESIGNED LAYOUT.
11. REPORT INFORMATION CONTENTS AGREES WITH TITLES, HEADERS.
12. REPORT PAGE NUMBERING ACCURATE.
13. SPELLING CORRECT ON REPORT TITLES, HEADERS.
14. FIELD SIZES ON REPORTS ADEQUATE.
15. AUDIT TRAILS (CONTROL TOTALS) ACCURATE.

INSTRUCTOR NOTES

WHAT SHOULD WE LOOK FOR?

IV. SYSTEM TEST EVALUATION

1. JCL EXECUTES SYSTEM WITHOUT ERRORS OR UNNECESSARY OPERATOR INTERVENTION, AND FILE LABELS ARE CORRECT.
2. JCL-DEFINED FILE CAPACITIES ADEQUATE, ASSUMING MODERATE EXPANSION.
3. EXECUTE TIME ACCEPTABLE ACCORDING TO PREESTABLISHED STANDARDS.
4. FILE SEQUENCES (SORT SPECS) ACCURATE THROUGHOUT SYSTEM EXECUTION.
5. NO-FILE CASE EXECUTING THROUGHOUT ALL SYSTEM STEPS.
6. EMPTY-FILE CASE EXECUTING THROUGHOUT ALL SYSTEM STEPS.
7. AVAILABLE MEANS FOR CROSS-CHECKING SYSTEM RESULTS EXHAUSTED.
8. INTERFACES WORKING ACCURATELY.
9. FILES CLOSED AT EOI (NORMAL OR DUE TO ERROR).
10. EXECUTION SEQUENCE (SYSTEM FLOW) ACCURATE AND WORKABLE.
11. AUDIT TRAILS (CONTROL TOTALS) ACCURATE.
12. ERROR-CORRECTION PROCEDURE LOOPS WORK.
13. SYSTEM RESTART PROCEDURES SATISFACTORY.
14. FILE RECOVERY PROCEDURES SATISFACTORY.
15. FILES AND REPORTS CONFORM TO OUTPUT FROM PARALLEL TEST, IF APPLICABLE.
16. OUTPUT ACCURATE AFTER MORE THAN ONE EXECUTION CYCLE.
17. END-OF-PERIOD OUTPUT ACCURATE, AND PERIOD-TOTALS RESET TO ZERO.

V. ACCEPTABILITY TEST

1. USER REVIEW OF OUTPUT.
2. JUDGEMENT OF ACCEPTABILITY ACCORDING TO PREDETERMINED STANDARDS.
3. SERVICES PROVIDED BY THE SYSTEM CONFORM TO ORIGINALLY STATED USER REQUIREMENTS.
4. CHANGES IN USER REQUIREMENTS SINCE ORIGINAL STATEMENT.
5. SYSTEM TESTING JUDGED COMPLETE.
6. USER SIGN-OFF ON OUTPUT.

INSTRUCTOR NOTES

THEME: IMPLEMENTATION CONSIST OF MORE THAN CODING

PURPOSE: A QUICK SUMMARY

Section 30

IMPLEMENTATION WRAP-UP

INSTRUCTOR NOTES

- GOALS - WHAT YOU'RE SHOOTING FOR (THE "ENDS")
- REQUIREMENTS - THE MAJOR CONSTRAINTS (THE "WHYS")
- DESIGN - THE BIG PICTURE (THE "WHATS")
- HARDWARE - THE POSSIBLE WAYS THINGS CAN BE DONE (THE "HOWS")

PERSONNEL TRAITS

- GOOD IMPLEMENTERS ARE AWARE OF ...

- PROJECT GOALS

- THE REQUIREMENTS

- THE DESIGN

- THE HARDWARE'S CAPABILITY

- THE TECHNIQUES FOR EFFECTIVE
IMPLEMENTATION

INSTRUCTOR NOTES

ENTERTAIN QUESTIONS, BUT DON'T GET STUCK ON DETAILS. KEEP TALK ON A MORE GENERAL LEVEL
RATHER THAN SPECIFIC.

REMARK THAT Ada RELATIONSHIPS WERE HANDLED IN INDIVIDUAL SECTIONS.

TECHNIQUES SUMMARY

CHARACTERISTIC TECHNIQUE	EMPHASIZES	GOAL	APPROPRIATE AUDIENCES	COVERAGE OF THE IMPLEMENTATION PHASE
STRUCTURED PROGRAMMING	CONTROL FLOW COMPLEXITY	SIMPLICITY	CODERS	THROUGHOUT
TIME/SPACE TRADEOFF	EFFICIENCY	EFFICIENCY	CODERS	THROUGHOUT
STEP-WISE REFINEMENT	HIERARCHICAL DECOMPOSITION	UNDERSTANDABILITY	DESIGNERS CODERS	THROUGHOUT
FINITE-STATE MAPS	SYSTEM BEHAVIOR	UNDERSTANDABILITY	ANALYSTS DESIGNERS CODERS	EARLY
DECISION TABLES & TREES	DECISION LOGIC COMPLEXITY	SIMPLICITY	CODERS	THROUGHOUT
ASSERTIONS	DECISION LOGIC IMPLICATIONS	UNDERSTANDABILITY	CODERS	THROUGHOUT
INVARIANTS & GUARDS	REPETITION COMPLEXITY	RELIABILITY	CODERS	THROUGHOUT
TESTING	VERIFICATION	RELIABILITY	TESTERS	LATE
INTEGRATION	CONNECTING SYSTEM PARTS	RELIABILITY	TESTERS	LATE
REVIEWS	VERIFICATION	RELIABILITY	EVERYONE	THROUGHOUT

INSTRUCTOR NOTES

- - DATA AND ALGORITHMS HAVE THE SIMPLEST STRUCTURES
- - GOES AWAY WITH SMART USE OF TIME/SPACE TRADEOFF
- - IS AN IDEAL, NEVER ATTAINABLE
- - ARE YOUR ASSUMPTIONS, SO DOCUMENT YOUR THOUGHTS
- - MAKE LOOPS RELIABLE
- - INTEGRATE TOP-DOWN = 90%, BOTTOM-UP = 10%, PHASED = 0%
- - ALL PRODUCTS, WITH PEERS, AS MUCH AS POSSIBLE

REMINDERS

- DURING IMPLEMENTATION ...
 - STRUCTURE SIMPLY
 - BE AWARE OF BLATANT INEFFICIENCY
 - DON'T INSIST ON CORRECTNESS
 - DOCUMENT ASSERTIONS
 - GUARD AGAINST UNRELIABLE LOOPS
 - TEST IN A RELIABLE MANNER
 - CONSTANTLY REVIEW YOUR WORK
- IMPLEMENTATION TECHNIQUES CAN'T BE EVALUATED USING
SAME CRITERIA AS FOR ANALYSIS AND DESIGN

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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

Section 31

INSTRUCTOR NOTES

COURSE

WRAP-UP

NOTE: THERE ARE NO LECTURE NOTES FOR THIS SECTION.

INSTRUCTOR NOTES

- THIS LAST TOPIC IS RUN AS A DISCUSSION.
- IF THE COURSE HAS BEEN RECEIVED WELL, THEN TRY TO HAVE A LIVELY DISCUSSION. OTHERWISE, KEEP THE LECTURE SHORT AND TO THE POINT.
- FIRST POLL THE CLASS FOR COMMENTS AND GENERAL QUESTIONS. GIVE THEM SOME TIME TO RESPOND. YOU CAN EVEN GIVE THE CLASS A COUPLE OF MINUTES TO REFLECT ON THE COURSE AND JOT DOWN COMMENTS AND QUESTIONS. IF THINGS ARE GOING WELL, JUST LET THE DISCUSSION FLOW. USE UP THE ALLOTTED TIME, THEN FORMALLY "SIGN OFF."
- SELECT SLIDES FOR THE INTRODUCTION SECTION (1-4) AND PHASE WRAP-UP SECTIONS IF YOU NEED TO TALK TO A SLIDE.

AD-A165 302

ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 3
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

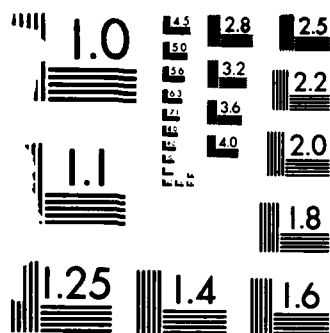
5/5

UNCLASSIFIED

F/G 5/9

NL





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

INSTRUCTOR NOTES

- IF THE COMMENTS AND QUESTIONS OF THE CLASS DON'T STIMULATE ENOUGH DISCUSSION, ASK A FEW OF YOUR OWN. THE FOLLOWING ARE THREE QUESTIONS THAT ARE LIKELY TO MAKE MOST OF THE CLASS START PARTICIPATING:

1. ARE THERE ANY NEGLECTED LIFE CYCLE PHASES?

YES, THERE ARE TWO: INSTALLATION AND MAINTENANCE. THESE PHASES ARE STILL IN A STATE OF HAVING A BUNCH OF HEURISTICS, WITH NO METHODOLOGY. THE PROBLEMS FACED IN THESE PHASES ARE VERY PROJECT SPECIFIC, AND THE SHARING OF TECHNIQUES HAS NOT BEEN HIGH ("SOFTWARE PRACTICE AND EXPERIENCE" BEING ONE OF THE ONLY PROFESSIONAL FORUMS). SO MUCH WORK IS NEEDED TO BETTER UNDERSTAND THE PROBLEMS IN THESE AREAS.

INSTRUCTOR NOTES

2.

HOW WELL DO THESE METHODOLOGIES INTEGRATE WITH EACH OTHER?

A BRIEF ANSWER: SOME WELL, SOME NOT SO WELL, SOME NOT AT ALL. INTEGRATION OF TWO METHODOLOGIES IS INFLUENCED BY THE COVERAGE OF THE LIFE CYCLE BY EACH METHODOLOGY, AND BY WHAT IS EMPHASIZED BY EACH METHODOLOGY. LET'S GIVE AN EXAMPLE OF EACH CASE:

SOME WELL

STRUCTURED DESIGN STOPS AT THE MODULE LEVEL; IT IS CONCERNED WITH ARCHITECTURAL DESIGN ISSUES. NSSF DESCRIBES WELL THE LOGIC FLOW OF MODULE. SO, NSSF DIAGRAMS COULD BE DRAWN FOR EACH STRUCTURE CHART BOX. HERE, THE FIT IS EXACT, FOR THE CONTEXT FOR EACH NSSF DIAGRAM IS A BOX ON THE ("BIG PICTURE") STRUCTURE CHART.

ASK THE CLASS FOR METHODOLOGY COMBINATION THEY THINK WOULD WORK WELL TOGETHER.

INSTRUCTOR NOTES

SOME NOT SO WELL

SADT DIAGRAMS MODEL THE FUNCTIONS AND DATA OF A SYSTEM; SO DO THE BUBBLE CHARTS OF STRUCTURED DESIGN. BUT SADT MODELS EMPHASIZE THE CONSTRAINTS AMONG THE FUNCTIONS OF A SYSTEM, WHILE BUBBLE CHARTS EMPHASIZE DATA FLOW AMONG SYSTEM FUNCTIONS. SO, AN SADT DIAGRAM CANNOT BE CONSIDERED A BUBBLE CHART FROM WHICH A STRUCTURED DESIGN STRUCTURE CHART CAN BE DRAWN. A BUBBLE CHART MUST BE DRAWN FROM THE SADT MODEL, BEFORE A STRUCTURED DESIGN CAN BEGIN. HERE, THE FIT IS NOT EXACT, BUT INTEGRATION CAN BE SUCCESSFUL (THOUGH NOT STRAIGHT-FORWARD).

ASK THE CLASS FOR COMBINATIONS THEY THINK THAT DO NOT FIT.

INSTRUCTOR NOTES

SOME NOT AT ALL

TRYING TO INTEGRATE PARNAS (SCRIP) WITH STRUCTURED DESIGN WOULD BE DISASTROUS. THE MAJOR REASON BEING WHAT EACH METHODOLOGY EMPHASIZES: PARNAS -- DATA HIDING RESULTS IN PACKAGES, STRUCTURED DESIGN -- DATA FLOWS (TRANSFORMS/TRANSACTION CENTERS) RESULT IN SYSTEM STRUCTURE. COMMON SERVICE PACKAGES ARE DEVELOPED EARLY WITH PARNAS AND LATE WITH STRUCTURED DESIGN. THE TWO METHODS WORK CROSS PURPOSES, SO THEY WOULD BE EXTREMELY DIFFICULT, IF NOT IMPOSSIBLE, TO INTEGRATE.

INSTRUCTOR NOTES

3. HOW DOES ADA FIT WITH THE METHODOLOGIES YOU'VE SEEN?

ASK THE CLASS TO IDENTIFY SOME PROBLEM AREAS IN APPLYING Ada IN THE CONTEXT OF THE METHODOLOGIES WE DISCUSS.

YOU MAY NEED THE Ada RELATIONSHIP SLIDES FROM THE VARIOUS SECTION WRAP-UPS HERE.

INSTRUCTOR NOTES

- NOW IT'S TIME TO SAY "GOODBYE." SO GIVE AN APPROPRIATE FAREWELL, LET EACH INSTRUCTOR HAVE THEIR SAY, LET THE CLASS MAKE ANY PARTING REMARKS.

Material: Software Engr'g Methodologies (M201), Volume III

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

DTIC

END

4-86