ADVANCED SOFTWARE TEST METHODS

Detailed Notes and Viewgraphs

Presented By:

Dr. Edward F. Miller, Jr.

May 11, 1988

Prepared For Presentation At:

QUALITY WEEK
Marriott Fisherman's Wharf
San Francisco, California

SR Job No. 1107

Software Research, Inc.
625 Third Street
San Francisco, CA 94107-1997 USA

Phone: (415) 957-1441 -- Telex: 340-235 -- Fax: (415) 957-0730

MOTIVATIONS FOR SOFTWARE QUALITY ASSURANCE

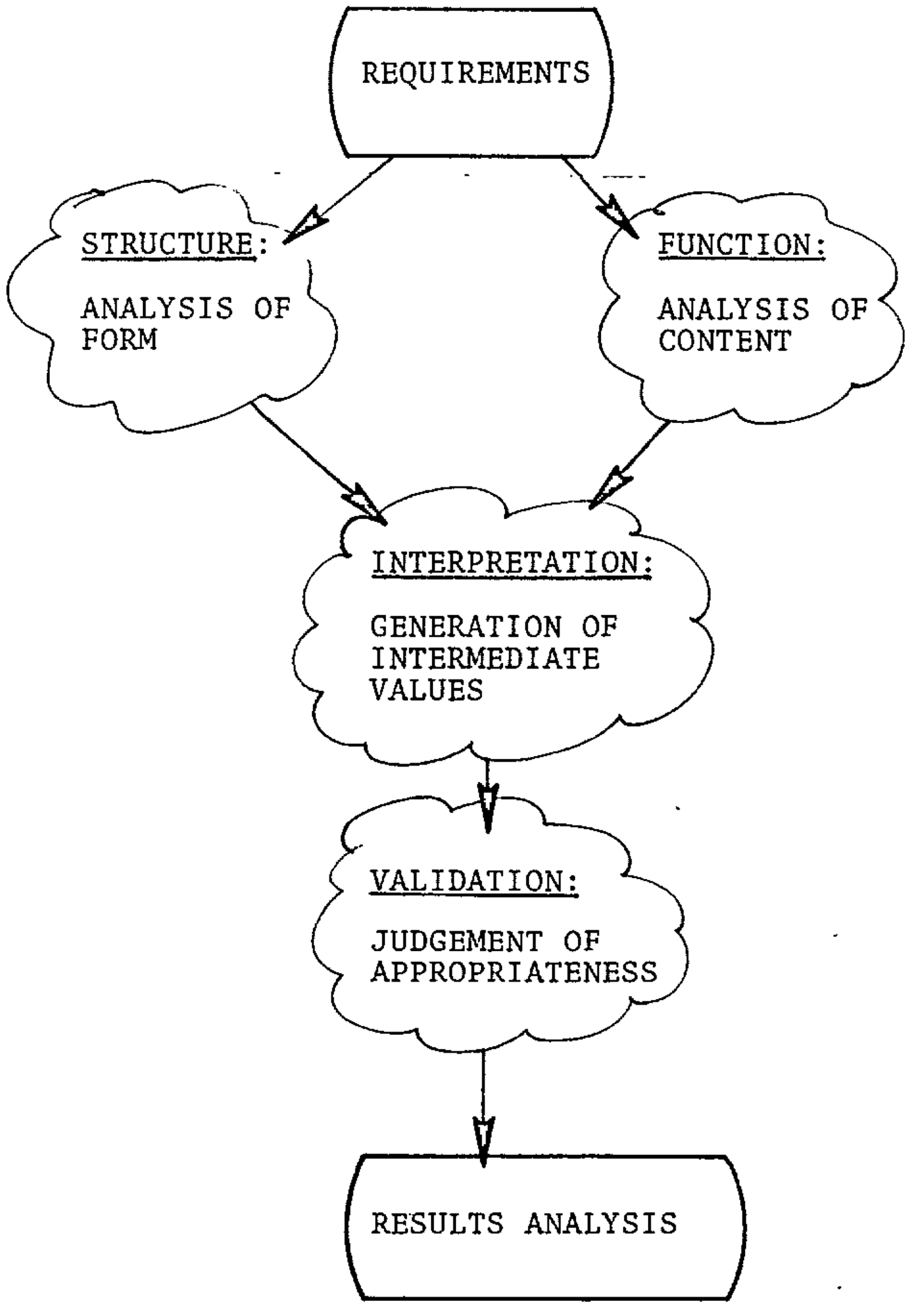
- **EFFECTIVE UTILITY**
  - Does a software system do what it is supposed to do?
  - Does it not do what it's not supposed to do?
  - What are its actual properties?
- **JUDGEMENT REQUIRED FOR QUALITY**
- **PERSPECTIVE OF OPINION MAKER (JUDGE)**
  - Software expert
  - Software user (Engineer)
  - Public user
  - Public non-user
- **WHAT IS THE COST OF A SOFTWARE ERROR?**
  - Direct costs
  - Indirect costs
  - Human costs
  - Liability

SOFTWARE RESEARCH

Comparison of Hardware and Software Complexity

	Software	Hardware
1	Simple "tr" (translate function)	A PLA or combinational circuit
2	Some mathematical operation like matrix multiply	Memory address computation logic and arithmetic
3	Language Compiler	Complete CPU (instruction interpreter & executer)
4	Operating system	Complete Computer System (CPU, memory, input/output controllers)

THE TECHNICAL BASIS OF VALIDATION TECHNIQUES

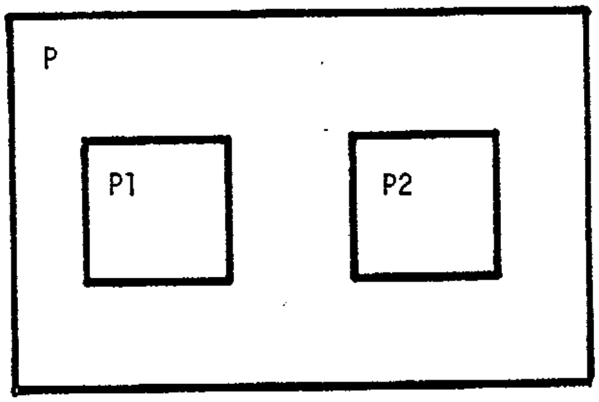


SOFTWARE RESEARCH

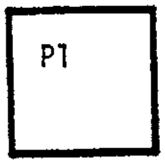
SOME "PRINCIPLES" OF PROGRAM TESTING

- **SEPARABILITY**
  - Testing a thing composed of two parts can be done by testing the thing's parts.
- **REPEATABILITY**
  - Any test of a module has to be repeatable.
  - Non-repeatability implies non-deterministically.
- **MEASURABILITY**
  - It doesn't do any good to do something if you can't measure the effect of what you've done.
- **FINITENESS**
  - Any test that never stops is not really a test at all.
- **FUNCTIONAL NECESSITY**
  - Every part of a software system has to have some purpose else it need not be part of the software system.
- **DISTINGUISHABILITY**
  - Two identical tests are no better than one of them.
- **THE ENVIRONMENT IS PART OF THE INPUT!**

THE SEPARABILITY PRINCIPLE OF TESTING



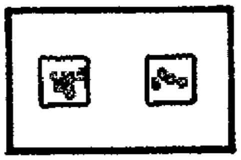
TEST ALL THREE MODULES AS A UNIT



(1) TEST P1

(2) TEST P2

P - (P1 + P2)



(3) TEST P WITHOUT THE EFFECT OF P1 AND P2

(BUILD WITH PROVEN COMPONENTS)

(ASSUMES NO SIDE EFFECTS)

SOFTWARE RESEARCH

REQUIREMENTS-BASED TEST PLANNING

GOAL:

ESTABLISH TEST REQUIREMENTS FROM ORIGINAL SYSTEM REQUIREMENTS:

- MODULE TEST
- SUBSYSTEM TEST
- SYSTEM (ACCEPTANCE) TEST

PROVIDE ASSURANCE OF COMPLETENESS OF TESTING

IDENTIFY MISSING/EXTRA TESTS

METHOD:

REQUIREMENTS TRACING:

- LIST OF REQUIREMENT FEATURES
- FEATURES COVERED BY TESTS
- "R1" METRIC MEASURES:
  - MISSED REQUIREMENTS
  - MISSING TESTS
  - EXTRA TESTS

AFTER CREATION OF DATABASE, ANALYSIS OF INFORMATION FOR CURRENT R1 VALUE

PAYOFFS:

"EASY" METHODOLOGY, SIMPLE TOOL TO BUILD

HARD TO APPLY RELIABLY FOR BIG SYSTEMS

PROBLEMS AND SOLUTIONS:

RELIES ON HUMAN INTERFACE

POSSIBLE HIGH DEGREE OF AUTOMATION (AUTOMATIC COLLECTION OF REQUIREMENTS' HIT)

LARGE DATABASE IF MODULE, SUBSYSTEM, AND SYSTEM TESTING HANDLED

CAUSE EFFECT GRAPHS AS REQUIREMENTS-BASED TEST PLANNER

GOAL:

USE FIRST ORDER FORMAL LOGIC AS BASIS FOR DEFINING INPUT/OUTPUT RELATIONS

GENERATE TEST PLANS (INPUTS AND OUTPUTS) AUTOMATICALLY

METHOD:

MANUAL OR AUTOMATED SUPPORT FOR IDENTIFICATION OF INPUT AND OUTPUT STATES

MANUAL OR AUTOMATED SUPPORT FOR CAUSE EFFECT GRAPH (CEG) GENERATION

MECHANICAL GENERATION OF TESTS (PATH SENSITIZATION)

PAYOFFS:

FULLY MECHANICAL OPERATION OF CEG SYSTEM

EXTERNAL SPECIFICATIONS OF TESTOBJECT BEHAVIOR

PROBLEMS AND SOLUTIONS:

CAPACITY LIMITED TO APPROX. 75 CAUSES + EFFECTS

TOO MUCH DEPENDENCE ON USER'S CHOICE OF INPUT AND OUTPUT STATES

NO WAY TO DESCRIBE CERTAIN SOFTWARE-ESSENTIAL CONSTRUCTIONS (E.G. LOOPS)

FINITE STATE MODELS AS REQUIREMENTS-BASED TEST PLANNER

GOAL:

EXTENDED MODEL OF SYSTEM AS BASIS FOR BEHAVIOR DESCRIPTION

TESTS OF MODEL BECOME ACCEPTANCE TESTS OF SYSTEM

METHOD:

MODEL EXPECTED PROGRAM BEHAVIOR WITH STATE TRANSITION DIAGRAMS

TEST ALL TRANSITIONS IN DIAGRAM

TEST ALL STATE/INPUT POSITIONS IN TABLE

PAYOFFS:

MECHANICAL TEST GENERATION, INPUT/OUTPUT DESCRIPTION

ASSURED REPRESENTABILITY OF ANY SYSTEM CONSTRUCT (WITH ASSOCIATED COMPLEXITY)

PROBLEMS AND SOLUTIONS:

BASED ON AUTOMATA THEORY: POWERFUL BUT COMPLEX TECHNIQUE

LIMITED TO PERHAPS 50-100 TOTAL STATES

CAPABILITY FOR MODEL TO HAVE "MEMORY" BUT DIFFICULT TO REPRESENT ITERATION

COMBINATORIC GROWTH IN NUMBER OF TESTS

DESIGN BASED TEST PLANNING

GOAL:

DEVISE TESTS FROM SOME PART OF EARLY SYSTEM DESIGN DESCRIPTION:

- NS CHARTS
- MJS CHARTS
- DATA DICTIONARY

OR, DESIGN TESTS FROM PDL

METHOD:

TAKE ADVANTAGE OF STRUCTURAL INFORMATION IN DESIGN

ORGANIZE STRUCTURALLY SOUND, FUNCTIONALLY ACCURATE TESTS

EXPAND BY MANUAL OR AUTOMATED ANALYSIS OF "FLOW"

PAYOFFS:

CERTAINTY OF COMPLETENESS

AUTOMATED ASSISTANCE

PROBLEMS AND SOLUTIONS:

COMBINATORICS MAY BE A LIMITING (WITH OR WITHOUT AUTOMATION)

ESSENTIAL LINK TO "EXTERNAL SPECIFICATION" MAY BE LACKING

TEST PLANNING FROM PDL OR FROM LIVE CODE

GOAL:

EXPLOIT EXISTING STRUCTURAL INFORMATION IN PDL, OR IN CODE, TO ASSIST IN TEST PLANNING

METHOD:

CONSTRUCT DIGRAPH FROM STRUCTURED OBJECT

REDUCE DIGRAPH TO DESCRIPTIONS OF INTERNAL STRUCTURE (HIERARCHICAL DECOMPOSITION)

DERIVE COMPREHENSIVE AND COVERING TEST SETS

USE PATH DESCRIPTIONS TO PLAN TESTS

PAYOFFS:

MAY BE CONNECTED TO PROGRAMMING TASK

POSSIBLE AUTOMATIC SELECTION OF INPUT/ OUTPUT INFORMATION

UNAMBIGUOUS OUTPUTS

PROBLEMS AND SOLUTIONS:

TESTING ONLY STRUCTURE, NOT FUNCTIONS

USE WITH HIGH-LEVEL BEHAVIOR MODELS ONLY?

INSPECTION/REVIEW TECHNIQUE AUTOMATION

GOAL:

SUPPORT INSPECTION METHODS FOR DESIGN, TEST PLANS, AND PROGRAM CODE

METHOD:

PROVIDE AUTOMATED SUPPORT DURING INSPECTION PROCESS

ASSISTANCE IN APPLYING RULES

POSSIBLE MECHANIZATION IN DESIGNING AND/OR PRESENTING RULES

ASSISTANCE IN RECORDKEEPING

ASSISTANCE IN RE-INSPECTION

PAYOFFS:

INCREASED PRODUCTIVITY

INCREASED ACCURACY, REPEATABILITY

PROBLEMS AND SOLUTIONS:

INTERJECTION OF "MORE MACHINERY" INTO ALREADY LABOR-INTENSIVE SITUATION

HAVE A "PC" IN THE INSPECTION LOOP TO ACT AS SECRETARY

POSSIBLE "EXPERT SYSTEM" APPLICATION

STATIC TESTING

GOAL:

APPLY STATIC TESTING (SOURCE BASED) METHODS TO CANDIDATE SOFTWARE

METHOD:

REQUIRES SPECIAL STATIC ANALYZER SYSTEM

ALLEGATION SET MUST BE CHOSEN CAREFULLY

ALLEGATIONS MUST BE BASED ON EXPERIENCE WITH REAL-WORLD AREAS

PAYOFFS:

AFTER INITIAL CAPITAL COST, VERY HIGH RETURN (VERY LOW COST/DEFECT)

REPLACES PROGRAMMERS' ATTENTIVENESS

PROBLEMS AND SOLUTIONS:

AUTOMATED METHOD MAY BECOME A CRUTCH FOR PROGRAMMER/ANALYST

LANGUAGE DEPENDENT SYSTEM, RULESET

POSSIBLE EXPERT SYSTEM APPLICATION?

DYNAMIC MODULE TESTING -- TEST CAPTURE/PLAYBACK

GOAL:

PROVIDE AUTOMATED CAPTURE OF ACTUAL TEST SESSIONS

ASSURE AUTOMATIC, 100% PERFECT, SESSION PLAYBACK

METHOD:

INTERCEPT TESTERS' KEYBOARD ACTIVITY

INTERCEPT SCREEN ACTIVITY (USUALLY ON TESTERS' COMMAND)

GENERATE SUPPORTING KEYSAVE FILES THAT CAN BE PLAYED BACK

PAYOFFS:

STRONG BASE FOR REGRESSION TESTING

ASSURED REPEATABILITY OF TESTS

POSSIBLE KEYSAVE FILE EDITING (FOR INCREASED SIMPLICITY, EFFICIENCY OF KEYSAVE FILES)

PROBLEMS AND SOLUTIONS:

TIMING AMBIGUITIES CAN ALTER TEST BEHAVIOR -- USE FAITHFUL TIME RECORDING

DATA VOLUME IS SUBSTANTIAL IF TOO MANY SCREEN IMAGES ARE SAVED

DYNAMIC MODULE TESTING -- TEST COMPLETENESS ANALYSIS

GOAL:

ASSURE A COMPREHENSIVE TEST SET, ACCORDING TO SOME REPEATABLE MEASURE

"CONVERGENCE TESTING" TO COMPLETE, DIVERSIFY TEST SET

METHOD:

TEMPORARY SOURCE PROGRAM INSTRUMENTATION

RUNTIME DATA COLLECTION (POSSIBLY INTERACTIVE)

POST-TEST DATA REDUCTION, NOT-HIT ANALYSIS

PAYOFFS:

LOW COST METHOD

IDENTIFICATION OF UNDER TESTED REGIMES

HIGH AVAILABILITY OF TEST TOOLS

PROBLEMS AND SOLUTIONS:

DATA BURDEN CAN BE LARGE IF CARE IS NOT TAKEN IN PLANNING STAGES

TESTS ONLY STRUCTURE, NOT FUNCTIONS (STRUCTURAL TESTS MAY BE A GOOD APPROXIMATION TO FUNCTIONAL TESTS)

DYNAMIC MODULE TESTING -- TEST FILE GENERATION

GOAL:

CREATION OF FILES OF TEST DATA

RIGHT FORMAT FOR TESTED PROGRAM

VARIABLE CONTENTS, USER SELECTABLE

METHOD:

TEST FILE GENERATOR SYSTEM

- DESCRIPTOR FILE
- VALUES FILE
- INSTRUCTIONS FILE

OUTPUT PROCESSING COMBINES THREE FILES, GENERATES INSTANCES OF TEST DATA FILE

PAYOFFS:

FORCES DEFINITION OF OUTPUTS QUICKLY, EARLY IN TESTING PROCESS

CAN BE ACCOMPLISHED FROM EXISTING OUTPUT (BY EDITING)

PROBLEMS AND SOLUTIONS:

COMBINATORIC GROWTH OF NUMBER OF TESTS POSSIBLE

RANDOM SELECTION OF VALUES MAY BE DECEPTIVE, NOT PROVIDING ENOUGH COVERAGE?

DYNAMIC MODULE TESTING -- RELIABLE TEST DATA ASSESSMENT

GOAL:

GIVEN A SET OF STRUCTURALLY SOUND TEST, ASSURE THE THEORETICAL RELIABILITY OF THE TEST DATA VALUES

METHOD:

REQUIRES ANALYSIS OF SOURCE PROGRAM AND TEST SET

TESTS DATA RELIABILITY CRITERIA ARE WELL ESTABLISHED:

- AT BOUNDARY VALUES
- AT LIMIT VALUES
- FOR ITERATIONS
- NEAR "SWITCH" VALUES
- UNIQUE INPUT/OUTPUT VALUES
- ETC.

SHOULD BE ABLE TO IDENTIFY "UNRELIABLE" TESTS BY DETAILED ANALYSIS

PAYOFFS:

ENHANCED CONFIDENCE IN SETS OF TESTS

ELIMINATION OF USELESS OR REDUNDANT TESTS

POSSIBLE IMPLICATIONS ON NUMERICAL PRECISION

PROBLEMS AND SOLUTIONS:

RELIES TOO HEAVILY ON STRUCTURE

QUESTIONS CONCERNING NUMERIC PRECISION (ROUNDOFF, TRUNCATION, ETC.)

DYNAMIC MODULE TESTING -- TESTBED GENERATION

GOAL:

CONSTRUCT UNIT TEST ENVIRONMENT AUTOMATICALLY FROM SOURCE CODE

METHOD:

SOURCE PROGRAM ANALYZER AND TEST BED GENERATOR SYSTEM:

- AUTOMATIC TEST TARGET CALL GENERATED
- STUBS GENERATED
- GLOBAL DATA SIMULATED
- USER INTERACTIVE CONTROL

CLOSE CONNECTION TO COMPILER SYSTEM

PAYOFFS:

EASES PROGRAMMING, TESTING TASKS

COVERAGE ANALYSIS POSSIBLY AUTOMATIC

STRONG CONNECTION TO INTERACTIVE DEBUG SYSTEMS

PROBLEMS AND SOLUTIONS:

INVESTMENT COST

GENERALITY (LANGUAGE, SYSTEM DEPENDENCE)

PORTABILITY

HOW TO LOCATE "RIGHT" ELEMENTS OF PROGRAM SUPPORT ENVIRONMENT

DYNAMIC INTERFACE TESTING

GOAL:

HAVE ALL THE INTERFACES BEEN FULLY TESTED

METHOD:

FULLY TESTED INTERFACE MEANS:

- CONTROL VARIABLES TRIED
- DATA VALUES TRIED:
  - INPUT
  - OUTPUT
  - INPUT & OUTPUT

INSTRUMENTATION OF INTERFACES ALONE

"I1" METRIC MEASURES COMPLETENESS

PAYOFFS:

IMMEDIATE KNOWLEDGE OF UNEXERCISED AREAS

PROBLEMS AND SOLUTIONS:

WHAT PERCENTAGE OF ERRORS ARISE FROM THIS AREA?

CAPITAL INVESTMENT IN TOOL SYSTEM

POSSIBLE INTRODUCTION OF DATA FLOW ANOMALIES IF INTERFACE CONTROL TESTING IS NOT DONE CORRECTLY

DYNAMIC SYSTEM TESTING

GOAL:

HAVE ALL REQUIRED SYSTEM FEATURES AND FACILITIES BEEN EXERCISED SUCCESSFULLY?

METHODS:

(1) "BLACK BOX" TESTING:

SEE REQUIREMENTS BASED TESTING METHODS

(2) SOURCE LEVEL INSTRUMENTATION OF FUNCTION CALL PAIRS:

AUTOMATIC CALL-PAIR IDENTIFICATION

RUNTIME DATA COLLECTION

POST-TEST ANALYSIS OF DATA

"S1" MEASURE APPLIED TO ASSESS COMPLETENESS

PAYOFFS:

MECHANICAL VERIFICATION OF POWER, SOPHISTICATION OF TESTS

IDENTIFICATION OF FUNCTION CALL PAIRS NOT EXERCISED (RELATED TO INTERFACE TESTING)

PROBLEMS AND SOLUTIONS:

EXTRA RUNNING TIME AND SYSTEM COMPLEXITY

SIZE GROWTH OF APPROX. 10%

REGRESSION TESTING

GOAL:

HAVE ALREADY-TESTED FUNCTIONS BEEN RE-TESTED SUCCESSFULLY?

ARE NEW FUNCTIONS INTRODUCED?

ARE EXISTING FUNCTIONS DELETED?

METHOD:

ORGANIZE TESTS FOR AUTOMATIC REGRESSION EXECUTION

BUILD MATRIX IDENTIFYING STRUCTURE VERSUS TESTS WHICH EXERCISE STRUCTURE

RE-EXECUTE AND CHECK ONLY NEEDED TESTS (TYPICALLY 1%-10% OF THE TOTAL)

PAYOFFS:

POSSIBLE 10-100:1 REDUCTION IN RETESTING TIME REQUIREMENTS

SECONDARY BENEFITS FROM WELL ORGANIZED TEST DATA

PROBLEMS AND SOLUTIONS:

LARGE AMOUNTS OF TESTS DATA REQUIRED

ADDITIONAL ANALYSIS BURDEN

POSSIBLE TROUBLE HANDLING UNEXPECTED SYSTEM "ABORTS", OTHER ANOMALOUS OUTPUTS NOT EASILY AUTOMATABLE

MAINTENANCE TESTING -- CHANGE ANALYSIS

GOAL:

RELATE STRUCTURE OF CHANGES IN PROGRAM TO NEEDED RE-TESTING

METHOD:

IDENTIFY "STRUCTURE UNIT" THAT IS KNOWN TO CONTAIN ALL OF THE CHANGE:

ADDITION	(+)
DELETION	(-)
MODIFICATION	(o)

IDENTIFY TESTS WHICH ENTER/EXIT THE AFFECTED REGION

PAYOFFS:

VERY EFFICIENT RE-TESTING SCHEME

DETAILED KNOWLEDGE OF SYSTEM STRUCTURE AVAILABLE

PROBLEMS AND SOLUTIONS:

MAY BE TOO DETAILED IF SYSTEM IS LARGE

MAY BE UNNEEDED IF SYSTEM IS SMALL

MAY REQUIRE TOO MANY TESTS

NOT NECESSARILY RELATED TO FUNCTIONAL TESTING

LANGUAGE VALIDATION TESTING

GOAL:

IDENTIFY PROBLEMS IN COMPILER

METHOD:

FULL VALIDATION -- USE VALIDATION SUITE

PARTIAL VALIDATION -- APPLY SELECTED PARTS

TOUCH-TEST VALIDATION -- ASSURE REQUIRED BEHAVIOR OF ALL FEATURES ONE-BY-ONE

SANITY TESTING -- SIMPLE "COHERENCE" TESTING

PAYOFFS:

PACKAGED SUITES READILY AVAILABLE

MOST LANGUAGE ANOMALIES IDENTIFIED EARLY

SUBTLE DEFECTS DISCOVERED

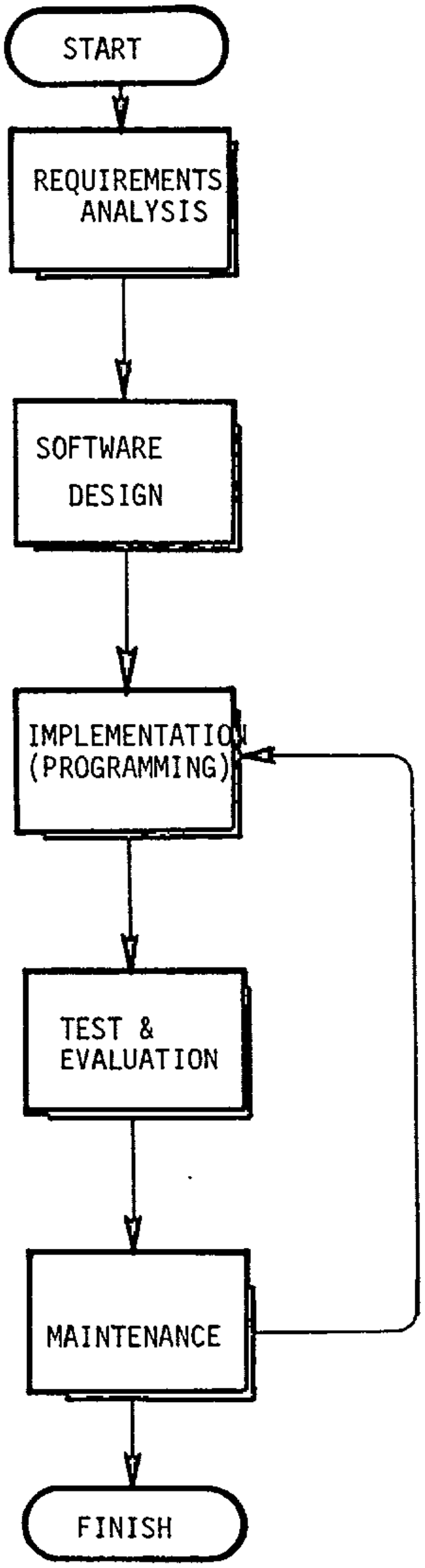
PROBLEMS AND SOLUTIONS:

NOT NECESSARILY STRONG STRUCTURAL TESTS

LENGTHY TO ACCOMPLISH EVEN FOR SIMPLE, WELL KNOWN COMPILERS

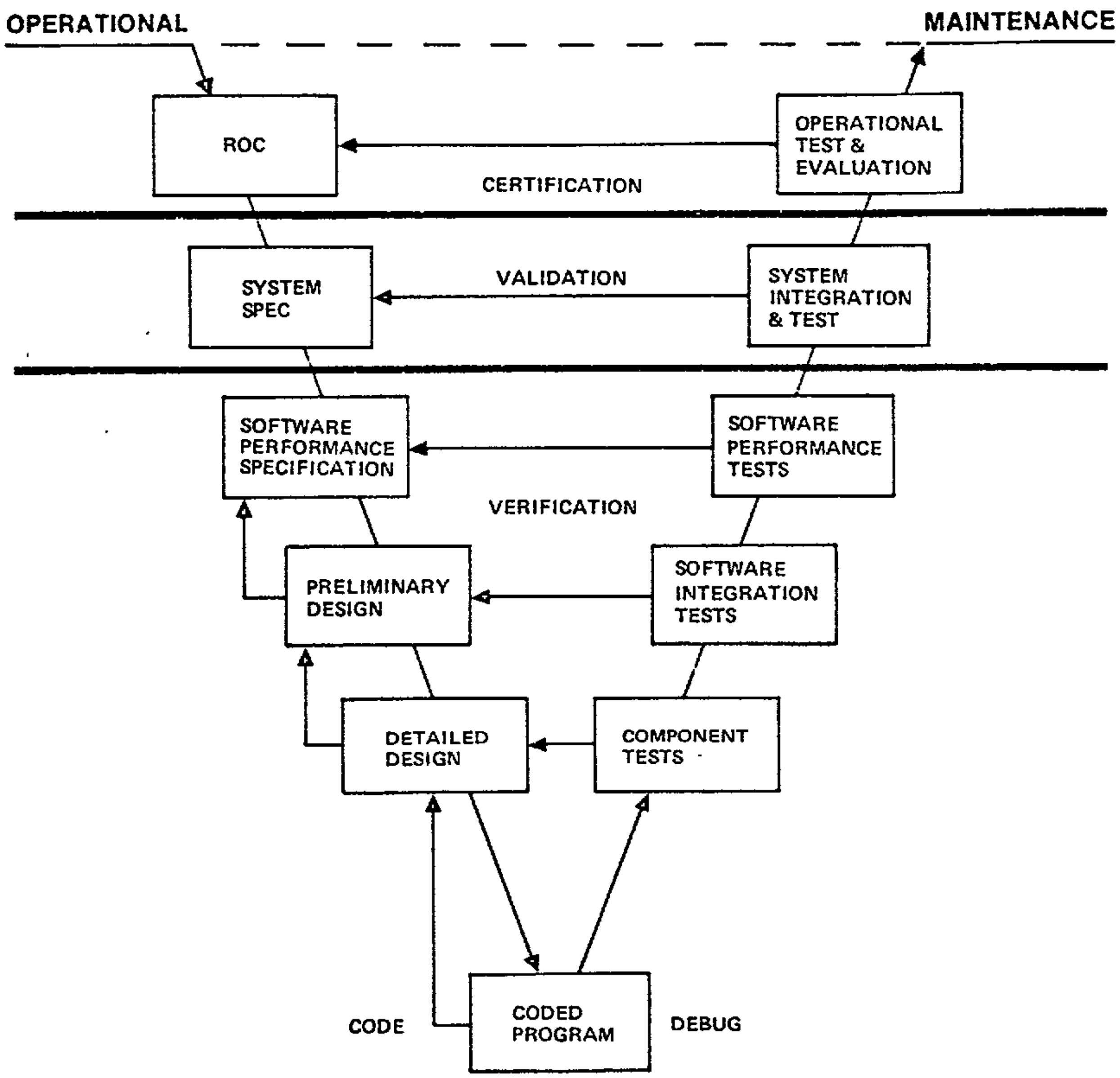
SOFTWARE RESEARCH

CLASSICAL SOFTWARE DEVELOPMENT METHODOLOGY STAGES:



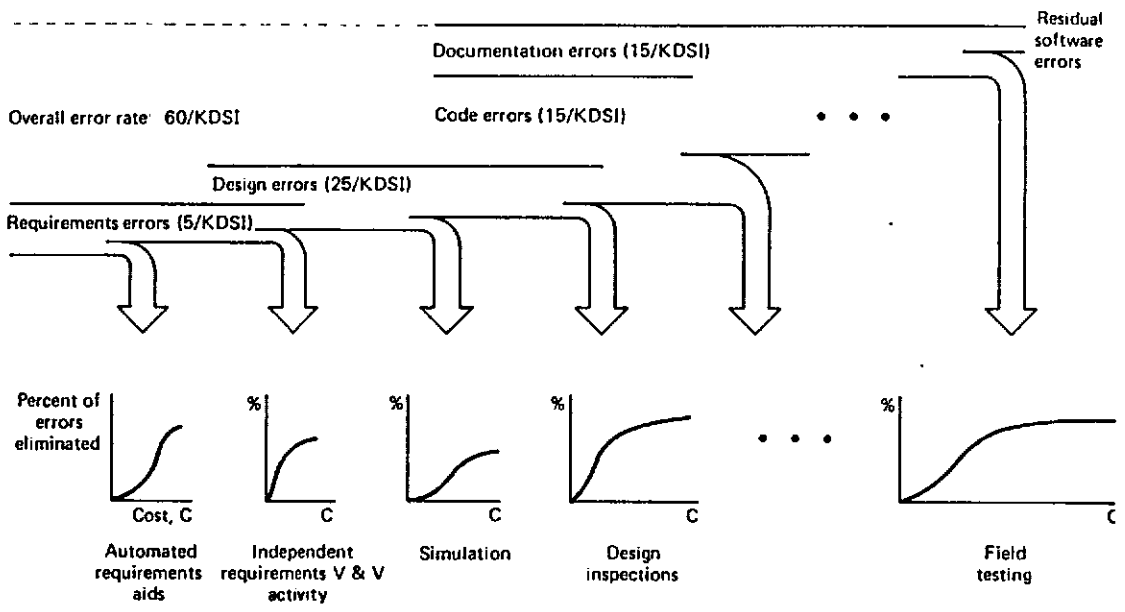
SOFTWARE RESEARCH

VERIFICATION, VALIDATION & CERTIFICATION



SOFTWARE RESEARCH

THE BASIC SOFTWARE ERROR INTRODUCTION/REMOVAL MODEL


Residual software errors
Documentation errors (15/KDSI)
Overall error rate 60/KDSI
Code errors (15/KDSI)
Design errors (25/KDSI)
Requirements errors (5/KDSI)
Percent of errors eliminated
%
Cost, C
C
Automated requirements aids
Independent requirements V & V activity
Simulation
Design inspections
Field testing

Source: Boehm, Software Engineering Economics, 1982

ERROR CATEGORY

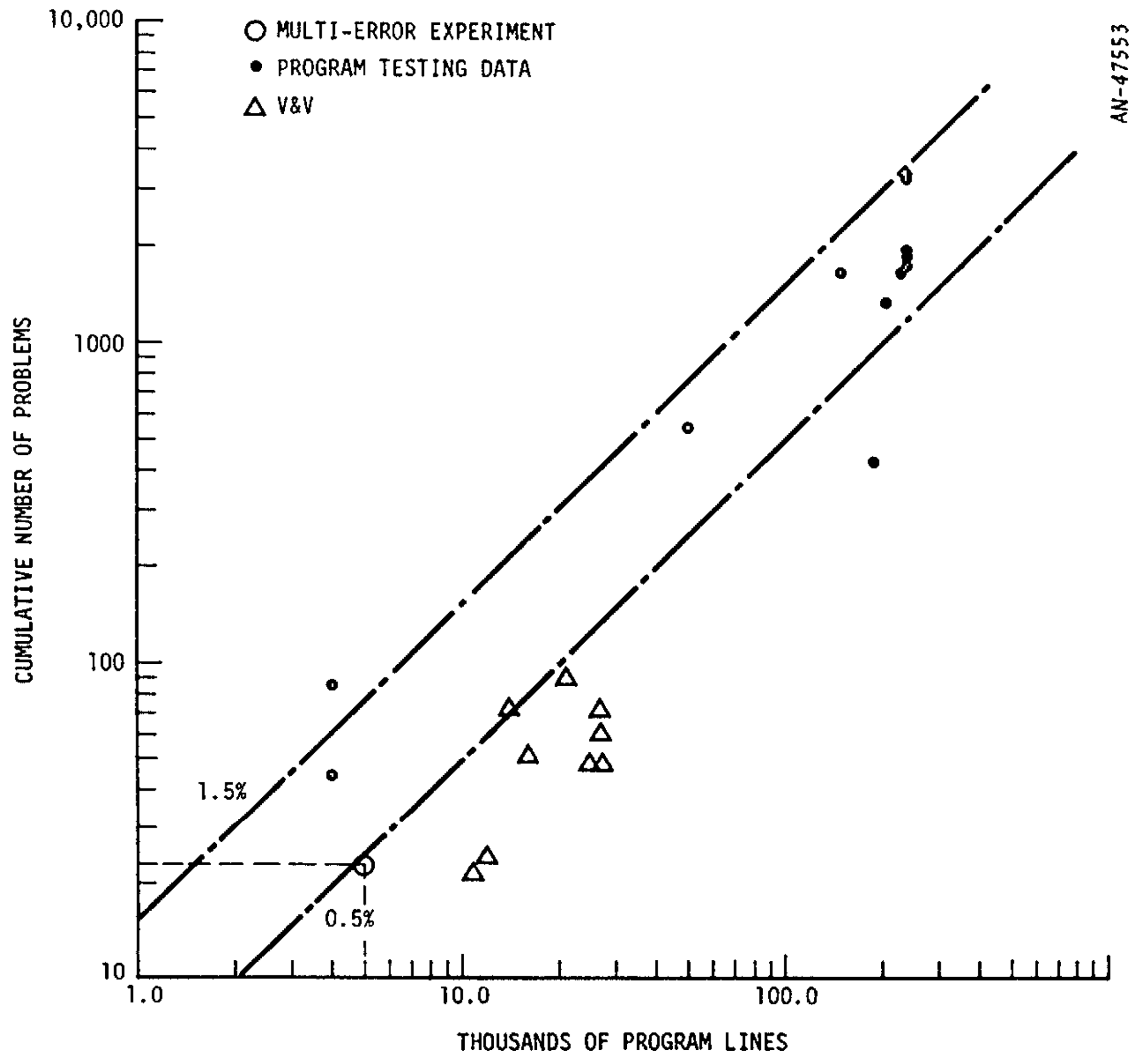
A000		COMPUTATIONAL ERRORS
	A100	Incorrect operand in equation
	A200	Incorrect use of parenthesis
	A300	Sign convention error
	A400	Units or data conversion error
	A500	Computation produces an over/under flow
	A600	Incorrect/inaccurate equation used
	A700	Precision loss due to mixed mode
	A800	Missing computation
	A900	Rounding or truncation error
B000		LOGIC ERRORS
	B100	Incorrect operand in logical expression
	B200	Logic activities out of sequence
	B300	Wrong variable being checked
	B400	Missing logic or condition tests
	B500	Too many/few statements in loop
	B600	Loop iterated incorrect number of times (including endless loop)
	B700	Duplicate logic
C000		DATA INPUT ERRORS
	C100	Data read with incorrect format
	C200	Incorrect input bus protocol
	C300	Data read from wrong device/file
	C400	Data read to wrong location
D000		DATA HANDLING ERRORS
	D100	Data initialization not done
	D200	Data initialization done improperly
	D300	Variable used as a flag or index not set properly
	D400	Variable referred to by the wrong name (A100?)
	D500	Bit manipulation done incorrectly
	D550	Scaling error
	D600	Incorrect variable type
	D700	Data packing/unpacking error
	D800	Sort error
	D900	Subscripting error
E000		DATA OUTPUT ERRORS
	E100	Data output to wrong device
	E200	Data output in wrong format
	E300	Incorrect output bus protocol
	E400	Data read from wrong location

SOFTWARE RESEARCH

F000		INTERFACE ERRORS
	F100	Wrong subroutine called
	F200	Call to subroutine not made or made in wrong place
	F300	Subroutine arguments not consistent in type, units, order
	F400	Subroutine called is nonexistent
	F500	Software/data base interface error
	F600	Software/software interface error
G000		DATA DEFINITION ERRORS
	G100	Data not properly defined/dimensioned
	G200	Data referenced out of bounds
	G300	Data being referenced at incorrect location
	G400	Data pointers not incremented properly
H000		DATA BASE ERRORS
	H100	Data not initialized in data base
	H200	Data initialized to incorrect value
	H300	Data units are incorrect
J000		OTHER
	J100	Cycle time limit exceeded
	J200	Memory storage limit exceeded
	J300	Wrong data states at time of concurrent voting
	J400	Timing error between I/O and CPU; I/O synchronization

ERROR ANALYSIS QAT-11-13

ERRORS IN DELIVERED SOFTWARE



△ DATA FROM RUBEY ET AL., LOGICON, 1975

● DATA COMPILED BY BALKOVICH, GRC, 1977

○ MULTI-ERROR EXPERIMENT, GRC, 1979

Reference: Gannon, et. al., "Experimental Evaluation of Software Testing,"

SOFTWARE RESEARCH

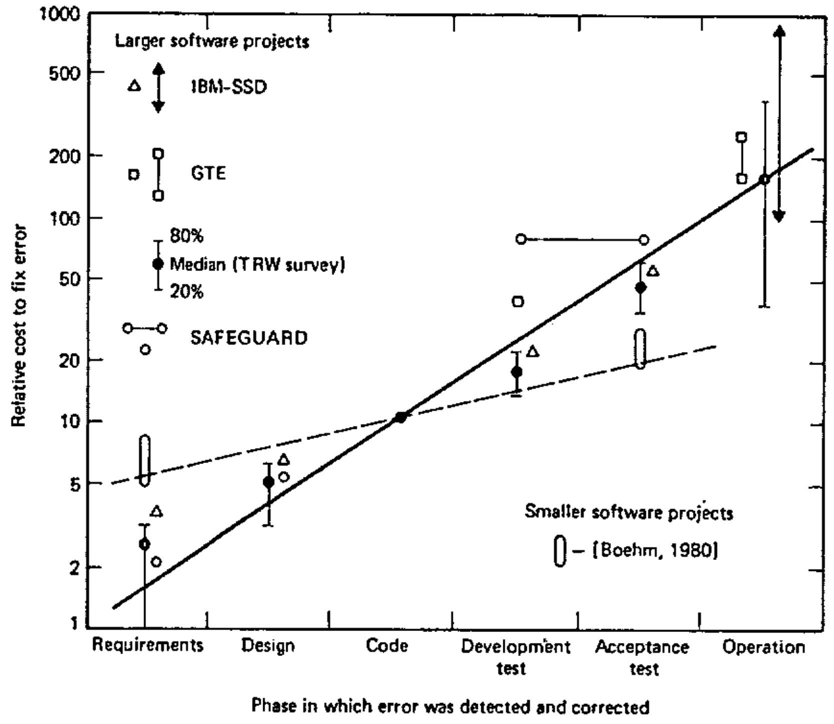
EFFECT OF MODULARIZATION ON ERROR RATES

Number of Lines of Executable Code	Number of Modules in System 1	2	5
100	1.75%	1.60%	1.30%
1000	2.43%	2.22%	1.95%
10000	3.17%	2.94%	2.65%

Citation: M. Lipow, "Number of Faults Per Line of Code," IEEE Trans. Software Engineering, Vol. SE-8, No. 4, July 1982.

SOFTWARE RESEARCH

COCOMO DATABASE REPRESENTATION OF COST-TO-FIX OR CHANGE SOFTWARE THROUGHOUT LIFE CYCLE



SOURCE: Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

CODE INSPECTION METHODS

- **ORIGIN OF CODE INSPECTIONS**
  - "Structured Programming" and allied software engineering technologies of the 1970's
- **ESSENTIAL ELEMENTS OF CODE INSPECTION**
  - Independent view of quality of software
  - Typical inspection team has various roles:

    MODERATOR - coach or key person
    DESIGNER - someone who understands the software
    CODER - person who wrote the program
    TESTER - person responsible for testing the program
- **TYPICAL RULES**
  - LOGIC: Missing, Wrong, and Extra Segments
  - PREDICTED TESTING BEHAVIOR: Common branches taken?
  - INTERCONNECTION: All links checked?
- **TYPICAL RESULTS**
  - 80% of available error population round per inspection cycle
  - 82% found during non-dynamic test; 18% found with unit test data
  - Rates range between 539 and 898 NCSSs per hour for design review and first code inspection.

REFERENCE: M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, 1976.

ASSEMBLER LANGUAGE INSPECTION RULES (CONTINUED)

Data Area Specifications:

11. Check that DSECTs correspond in format to the data which they represent.

12. If modifications have been made to a data structure, e.g., addition of fields within the structure (control block), check that required alignments are still preserved. Use particular care in the case of control blocks iteratively generated via conditional assembly logic.

    Even if the first block is OK, subsequent blocks may not start on the same type of boundary, causing program failure only when operating on blocks other than the first.

Preferred Coding Standards:

13. Insure that extended mnemonics are used whenever possible rather than hand coded condition code masks.

14. Check that a save area exists if required and is set up according to the prevailing operation system conventions (e.g. forward or backward pointers, etc.). If available, a system macro should be used to establish save area linkages (e.g. the OS SAVE macro).

15. Check that register usage conforms to the prevailing standards applicable to the project, if any. If no special standards are in use, then operating system standards should be applied (e.g., for OS, R13 is the save area pointer; R14 the return address; R15 the entry point address; R1 the parameter list pointer; R10 and R11 the parameter registers).

16. Check that EQUATEs are all meaningfully defined; in particular, check that register EQUATEs such as "R5 EQU 5" are not redefined as a short cut method of introducing changes, as would be the case if the above example were changed to "R5 EQU 6" in order to free register 5, assigning its current use to register 6.

17. Check that instruction level documentation adds meaning to the code, for example in the instruction "SR R5,R5 ZERO R5", the comment "ZERO R5" adds nothing to the content of the instruction. Compare: "SR R5,R5 ASSUME NO REQUESTS PENDING".

SOFTWARE RESEARCH

ASSEMBLER LANGUAGE INSPECTION RULES

Base Registers and Addressability:

1. Check that base registers defined by USINGs are all loaded at the appropriate time, i.e. before first attempted use.
2. Check that all temporary base registers are DROPped when no longer needed.
3. Check to ensure that base registers cannot be destroyed during execution particularly via calls to subroutines or across CSECT boundaries.
4. Check that all intended entry points are defined by ENTRY statements. Use the External Symbol Table Dictionary to verify their external status.
5. Check for operation code misspellings that will nevertheless be accepted by the assembler because the misspelling is another valid assembler instruction for which the operands have the same format as the intended instruction.
6. Check that Load Multiple (LM) picks up the desired sequence of full words and that they are placed into the expected registers.
7. Check that loop control mechanisms (BCT/BCTR, BXLE, BXH) do not cause looping one more time than expected, or one less.
8. Ensure that CLI is not used when TM is really required, i.e., check that bit switches are not confused with byte switches.
9. Check that the EX instructions are set up correctly, in particular in the case of a variable length move operation (MVC subject instruction).that I less than the length desired for the move be loaded into the first operand register of the EX.
10. Check that register 2 has not been unwittingly destroyed by a TR (translate) or (TRT Translate and Test) instruction.

ASSEMBLER LANGUAGE INSPECTION RULES (CONTINUED)

Miscellaneous:

18. Check that expressions representing lengths are specified correctly.

19. Check that all possible cases of conditional assembly parameters are generating the code that is expected. An assembly should be produced for all major cases and the logic of each compared with a card image printout of the source statements.

20. Check system macro calls to insure that keyword parameters are not specified as positional parameters, and vice versa.

    For macros accepting mixed format (i.e. both positional and keyword parameters) a keyword parameter written in positional form might be accepted as meaning something else than intended.

SOURCE: IBM TR 21.630m 3 May 1976

CODE INSPECTION MODULE DETAIL REPORT

Date ______________

Module: ______________________ Component/Application ______________

	MAJOR			MINOR			TOTAL
	M	W	E	M	W	E	
LO: Logic							
TB: Test and Branch							
EL: External Linkages							
RU: Register Usage							
SU: Storage Usage							
DA: Data Area Usage							
PU: Program Language Usage							
PE: Performance							
MN: Maintainability							
DE: Design Error							
PR: Prologue							
CC: Code Comments							
OT: Other							
TOTAL							

REINSPECTION REQUIRED? _____ (Y or N)

SOFTWARE RESEARCH

INSPECTION & REVIEW FORMS

QAT-22-6

SUMMARY INSPECTION REPORT

INITIAL DESIGN ☐ DETAILED DESIGN ☐ CODE ☐

Date ____________

To: Design manager ______________________ Development manager ______________________

Subject: Inspection report for ______________________ Inspection date ____________

Application ______________________

Component(s) ______________________

			Work Performed By		ELOC/NCSS Added, Modified, Deleted									Inspection Person-Hours (X.X)				
														Actual		Estimated		
		Full or Part	Initial Designer ☐	Detailed Designer ☐	Est. Pre.			Est. Post.			Rework							
Module Name	New or Mod.	Insp.	Detailed Designer ☐ / Programmer ☐	Programmer ☐ / Tester ☐	A	M	D	A	M	D	A	M	D	Over-view & Prep.	Insp. Meetg.	Re-work	Follow-up	Component
				Totals														

Reinspection required? ________ Length of inspection (clock hours and tenths) ______________________

Reinspection by (date) ________ Additional modules ______________________

DCR ID's written ______________________

Problem summary: Major ________ Minor ________ Total ________

Errors in changed code: Major ______ Minor ______ Errors in base code: Major ______ Minor ______

Initial Designer	Detailed Designer	Programmer	Team Leader	Other	Moderator's Signature

SOFTWARE RESEARCH

CODE INSPECTION ERROR ANALYSIS

Error Type		Error Category			Total Errors	Error %
		Missing	Wrong	Extra		
CC	Code Comments	5	17	1	23	6.6
DA	Data Area Usage	3	21	1	25	7.2
DE	Design Error	31	32	14	77	22.1
EL	External Linkages	7	9	3	19	5.5
LO	Logic	33	49	10	92	26.4
MN	Maintainability	5	7	2	14	4.0
OT	Other					
PE	Performance	3	2	5	10	2.9
PR	Prologue	25	24	3	52	14.9
PU	Prog. Lang. Usage	4	9	1	14	4.0
RU	Register Usage	4	2		6	1.7
SU	Storage Usage	1	8		9	2.7
TB	Test and Branch	2	5		7	2.0
		123	185	40	348	100.0

SOFTWARE RESEARCH

STATIC ANALYSIS — SYMBOLIC ANALYSIS OF PROGRAMS

- **GOAL**

  Static "Interpretation" of Program Behavior at the Programming Language Level

  NOTE: This process makes a number of assumptions about the environment, the properties (primarily determinism) of the programming language behavior, and the "meaning" of results.

- **TECHNIQUE**

  * *Choose a Path*: This requires specifying the symbolic outcomes of some of the program predicates, in turn based on knowledge of the intended/expected program behavior.
  * *Perform Symbolic Interpretation of Actions Along Chosen Path*: This produces a "formula" set that describes the computation the program performs on the specified path.
  * *Study Resulting Input/Output Relationship Against Specification*

- **PROBLEMS**

  * *Combinatorics* - The number of possible paths, or the path formulas in the presence of iteration become large and/or complicated, apparently exponentially with program size.
  * *Logical Choices* - Difficult to make in practical cases.
  * *Human Interaction Design* - How to communicate effectively to human user.
  * Others?

- **PROGNOSIS**

  Most Promising Method, Much Research Needed.

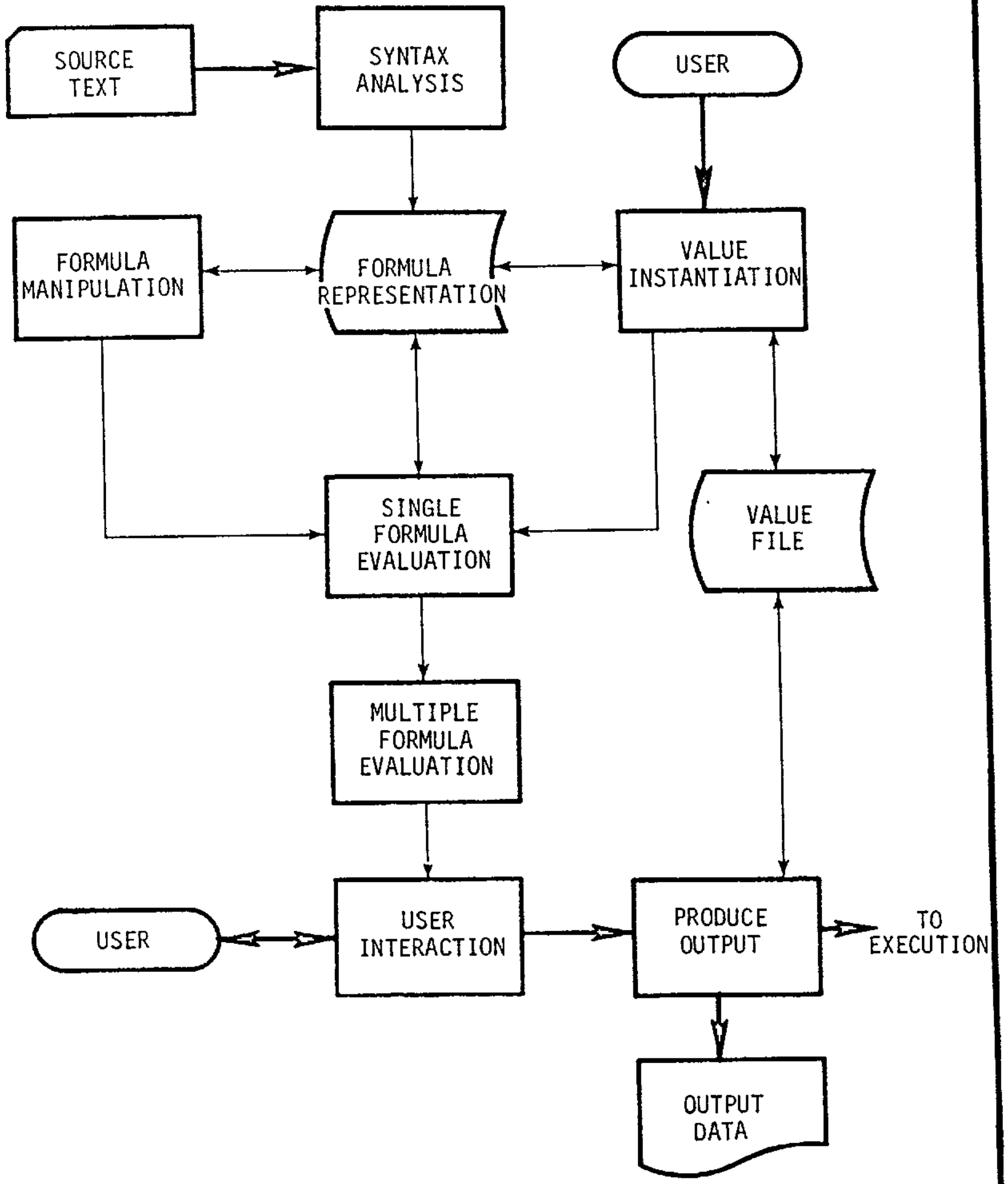
REFERENCES:

W. E. Howden, "Symbolic Testing and the DISSECT Symbolic Evaluation System," *IEEE Trans. Software Engineering*, July 1977.

L. Clarke, Current work at University of Massachusetts

SOFTWARE RESEARCH

SYSTEM FLOW WITHIN A SYMBOLIC EVALUATION/EXECUTION SYSTEM



SOFTWARE RESEARCH

SYMBOLIC ANALYSIS APPLIED TO NUCLEAR POWER PLANT SYSTEM

- **GOAL**

  Show how symbolic evaluation techniques can be applied now.

- **OUTLINE OF EXPERIMENT**
  - Software written in FORTRAN/IFTRAN.
  - Symbolic evaluator developed on ARPANET:
    - — uses MACSYMA (a lisp-based system)
    - — displays "formulas" to user
  - User compares original and implemented formulas for equality.

  NOTE: Differences between computed and actual formulas are mistakes. These are highly visible because special formula formatting methods are used to enhance differences.

- **RESULTS THUSFAR**

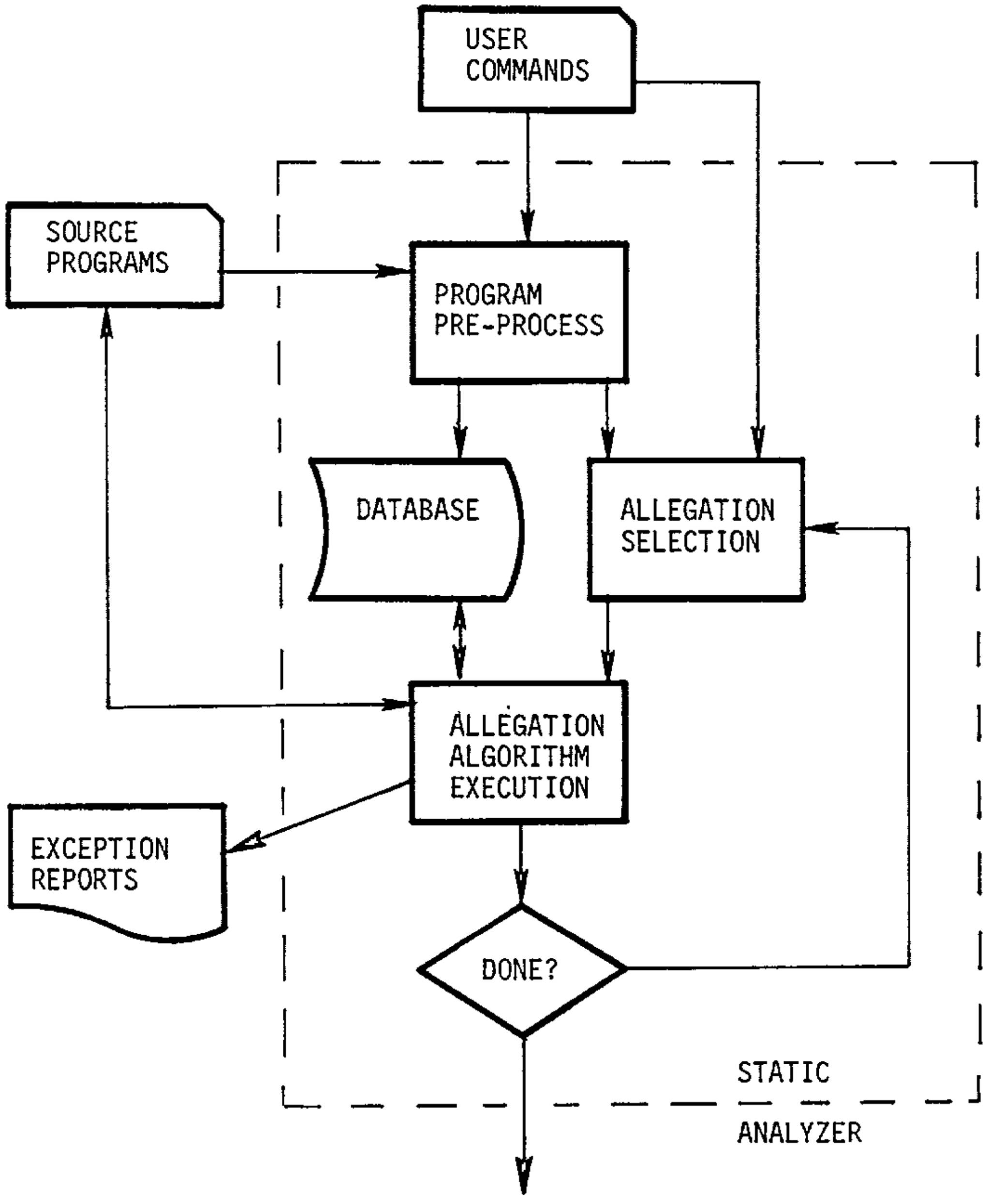
  (Final control software not yet available.)
  - High expectations from systematic analysis
  - Some "errors" already found in preliminary analysis

- **PROGNOSIS**

  Good results expected based on current estimates.

REFERENCE: C. V. Ramamoorthy, et. al., "A Systematic Approach to the Development and Validation of Critical Software For Nuclear Power Plants," *Proc. 1979 International Conference on Software Engineering*, Munich, West Germany, September 1979.

STRUCTURE OF STATIC ANALYSIS SYSTEM



SOFTWARE RESEARCH

EXAMPLE OF UNIX/LINT STATIC ANALYZER FOR "C" PROGRAMS

```
"ald.c", line 26: warning: old-fashioned initialization: use =
"ald.c", line 186: sflag undefined
"ald.c", line 186: warning: sflag may be used before set
"ald.c", line 186: warning: sflag unused in function main
"ald.c", line 137: warning: n unused in function main
"ald.c", line 256: warning: old-fashioned assignment operator
"ald.c", line 299: warning: illegal combination of pointer and integer
"ald.c", line 651: warning: illegal pointer combination
"ald.c", line 742: warning: struct/union or struct/union pointer required
"ald.c", line 876: undefined structure or union
"ald.c", line 876: warning: illegal member use: n_name
"ald.c", line 936: warning: function lookloc has return(e); and return;
_putw, arg. 2 used inconsistently       "ald.c"(630)  ::  "ald.c"(742)
chmod returns value which is always ignored
dseek, arg. 1 used inconsistently       "ald.c"(747)  ::  "ald.c"(288)
enter returns value which is sometimes ignored
error: variable # of args.      "ald.c"(911)  ::  "ald.c"(151)
error, arg. 3 used inconsistently       "ald.c"(911)  ::  "ald.c"(464)
fclose returns value which is always ignored
fread, arg. 1 used inconsistently       "/usr/lib/lint/llib-lc"(74)  ::  "ald.c"(770)
link returns value which is always ignored
loadl returns value which is sometimes ignored
lookloc, arg. 1 used inconsistently     "ald.c"(926)  ::  "ald.c"(613)
mget, arg. 1 used inconsistently        "ald.c"(709)  ::  "ald.c"(247)
mput, arg. 1 used inconsistently        "ald.c"(735)  ::  "ald.c"(493)
signal returns value which is sometimes ignored
tget returns value which is sometimes ignored
unlink returns value which is always ignored
$
```

SOFTWARE RESEARCH

A "TYPICAL" COST/BENEFIT ANALYSIS FOR STATIC ANALYSIS

- FACES has uncovered approximately 1 "error" per 200 FORTRAN statements in NASA/Huntsville application.
- Conservatively it costs $10 per statement to bring software to the point where it can be processed by a static analysis system.
- It is estimated to cost $100 to repair a mistake using manual methods (once it is found).
- FACES costs approximately 10 cents per statement in a commercial environment.
- Typical Situation†:
    * 20,000 statement program
    * FACES discovers 100 errors at a cost of $2000
    * Manual identification/repair would cost $10,000
    * Manual repair (new statement rates) would cost $1000
    * Saving is: $10,000 - $($2000 + $1000) = $7000
    * Implication: Using faces at a cost of $2000 results in a $7000 savings
    * Benefit(saving)/cost = **3.5**

† SOURCE: Wendel & Kleir, *FORTRAN Error Detection Through Static Analysis*, 1977.

STATIC ANALYSIS — PROGRAM PROVING METHODS

- **GOAL**

  Mathematical approach to "proving" the correspondence between a program and its formal specification.

- **TYPES OF CORRECTNESS**
  - Total Correctness
  - Partial Correctness
  - Path Correctness

- **TECHNIQUE**
  - Define set of verification conditions.
  - Prove consistency, using contradiction proof method, that the verification is consistent with program and all other assumptions.

- **ASSUMPTIONS**
  - Environment
  - Programming Language
  - Operating System
  - Validity of Proof

  NOTE: Some programs proved correct in the literature have been shown to actually contain errors.

  NOTE: Failure of Proof Method can be due to failure in prover, verification conditions, environment understanding, etc.

- **LARGEST PROGRAMS PROVED**
  - 1700 NCSS Assembly Language Interpreter, 43 errors

    NOTE: Approximately 85% of these errors could be found with simpler testing methods.
  - USC/ISI's "PROVE OFF", seeking thorough proof of 2000 NCSS System.
  - Typical cost \$50 - \$500/NCSS is expected range.

REFERENCES:

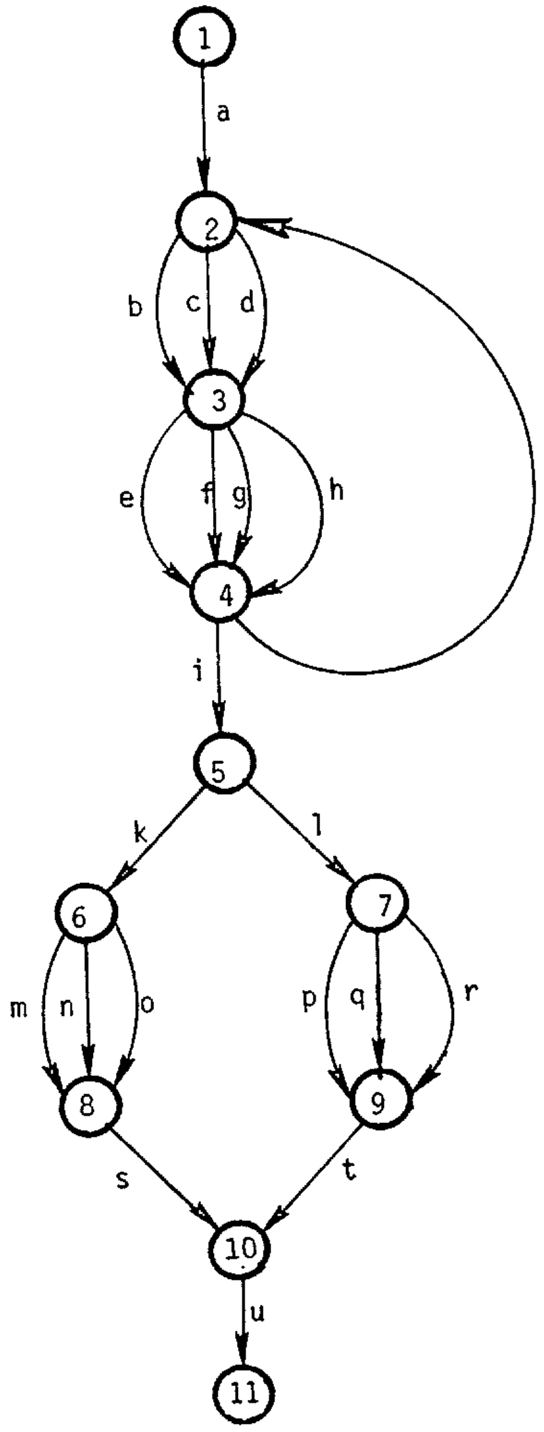
S. L. Hantler and J. C. King, "An Introduction to Proving Correctness of Programs," *ACM Computing Surveys*, September 1976

S. L. Gerhart and L. Yelowitz, "Observations of Fallibility in Applictations of Modern Programming Methodologies," *IEEE Trans. Software Engineering*, Sept. 1976

SOFTWARE RESEARCH

LEVELS OF SOFTWARE TEST PLANNING

- **REQUIREMENTS BASED TEST PLANNING**
  - Requirements analysis
  - Test plans
  - Requirements coverage
- **EARLY-DESIGN BASED TEST PLANNING**
  - Test plans
  - Documentation strategy
- **DESIGN/PSEUDOCODE BASED TEST PLANNING**
  - Operates from design embryo
  - Takes advantage of existing structure
  - Automatable function
- **CODE BASED TEST PLANNING**
  - White box testing
  - Black box testing
  - Gray box testing



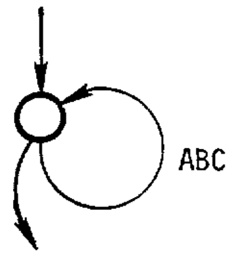
```
a
REPEAT
   CASE OF ( )
   CASE ( )
      b
   CASE ( )
      c
   CASE ( )
      d
   END CASE
   CASE OF ( )
   CASE ( )
      e
   CASE ( )
      f
   CASE ( )
      g
   CASE ( )
      h
   END CASE
UNTIL ( )
i
IF ( )
   CASE OF ( )
   CASE ( )
      m
   CASE ( )
      n
   CASE ( )
      o
   END CASE
   s
ELSE
   CASE OF ( )
   CASE ( )
      p
   CASE ( )
      q
   CASE ( )
      r
   END CASE
   t
END IF
u
```

SOFTWARE RESEARCH

IMPACT OF AN ESCAPE STATEMENT

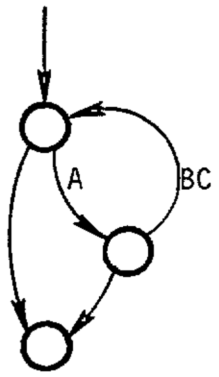
(1) WHILE WITHOUT ESCAPE

```
WHILE (p)
   A
   B
   C
END WHILE
```



(2) WHILE WITH ESCAPE

```
WHILE (p)
   A
   IF (p1)
      ESCAPE
   END IF
   B
   C
END WHILE
```

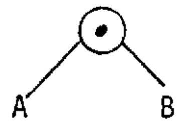


SOFTWARE RESEARCH

PROGRAM DECOMPOSITION PRIMITIVES

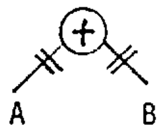
Succession:

```
A
B
```



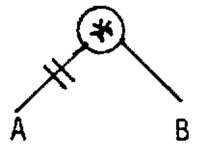
Alteration:

```
IF (p)
   A
ELSE
   B
ENDIF
```



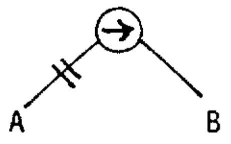
Iteration:

```
WHILE (p)
   A
ENDWHILE
   B
```



Invocation:

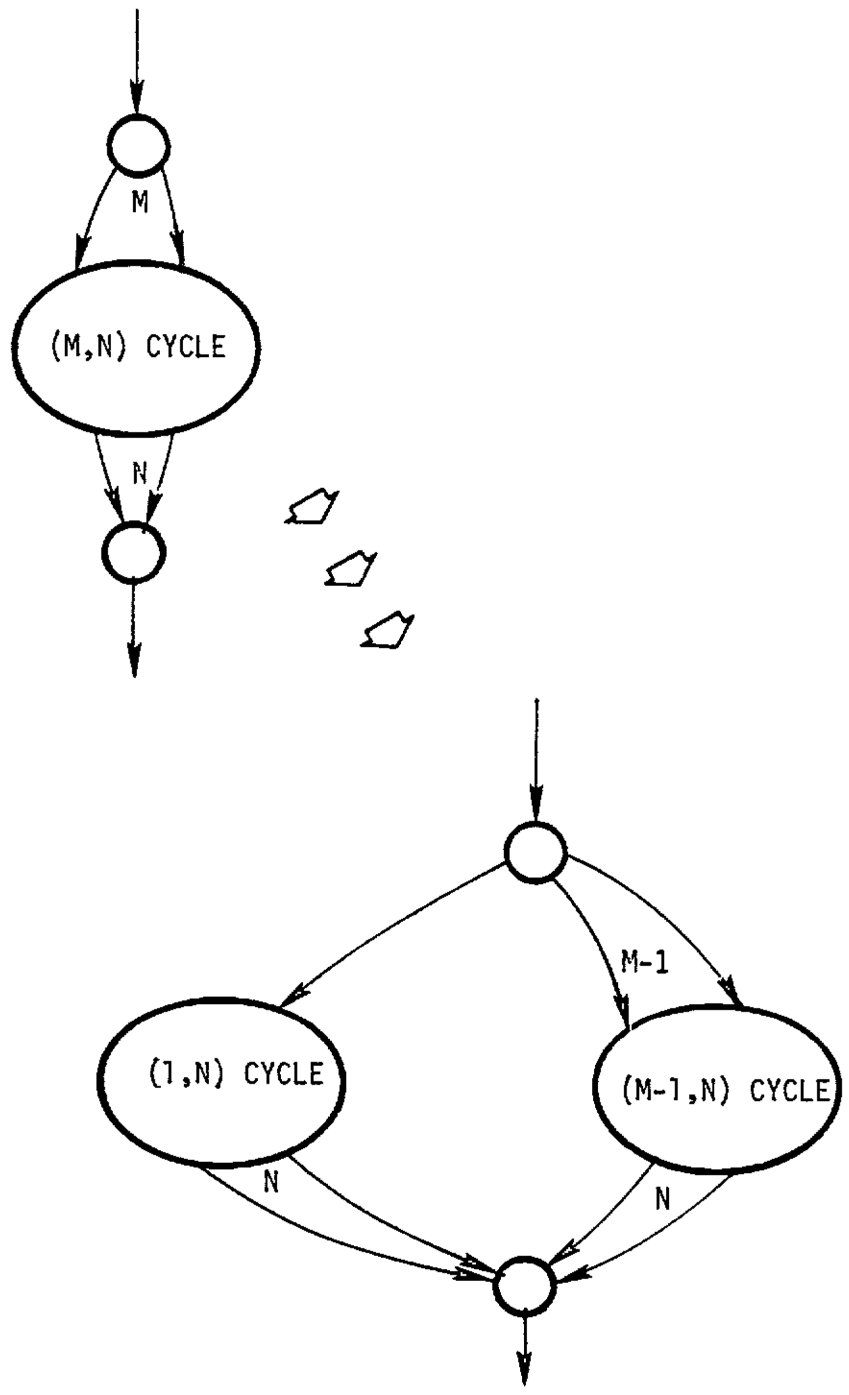
```
CALL A(...)
B
```



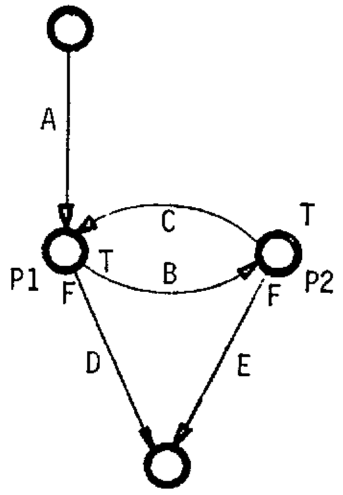
REDUCTION TECHNIQUES FOR (M,N) CYCLES

- Any digraph can be reduced to a form that involves only (1,1) cycles.
- It may be necessary to add edges and nodes.
    - Repeated Statements
    - Duplicated Labels
- Basic Algorithm
    - Reduce (M,N) cycle to (M-1,N) cycle and a (1,N) cycle.
    - This is done by copying the (1,N) cycle.
    - Reduce a (1,N) cycle to a (1,N-1) cycle and a (1,1) cycle.
    - This is done by "splitting" a node.
    - Continue until only (1,1) cycles remain.

REDUCTION OF (M,N) CYCLE TO (1,N) CYCLE AND (M-1,N) CYCLE



EXAMPLE OF PROGRAM INTERPRETATION/CONVERSION OF (1,2) CYCLE



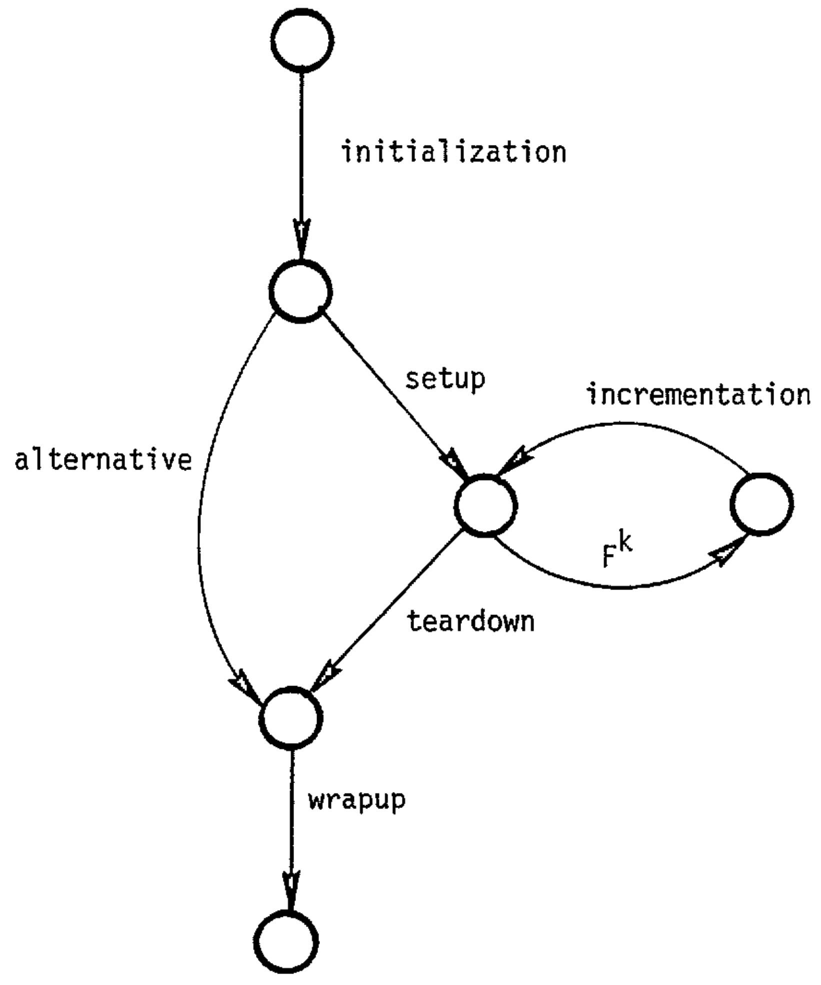
```
A
tag = "TRUE"
temp = P1
WHILE ( temp AND tag )
      B
      IF (P2)
             C
             temp = P1
      ELSE
             tag = "FALSE"
      END IF
END WHILE
IF ( tag )
      D
ELSE
      E
END IF
```

SOFTWARE RESEARCH

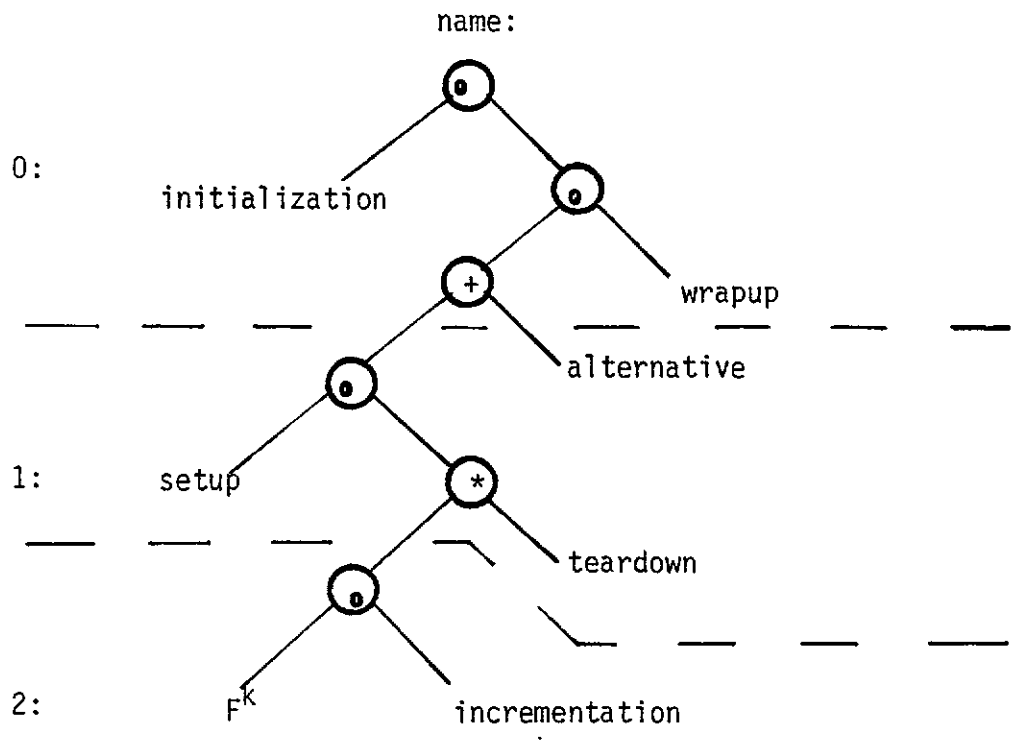
GENERIC EXAMPLE (CONDITIONED ITERATION)

```
SUBROUTINE name ( arguments )
declarations
initialization
IF ( initial-test )
     setup
     WHILE ( termination-condition )
          $F^k$
          incrementation
     END WHILE
     teardown
ELSE
     alternative
END IF
wrapup
RETURN
END
```

GRAPH OF CONTROL FLOW FOR GENERIC EXAMPLE (CONDITIONED ITERATION)



HIERARCHICAL DECOMPOSITION OF GENERIC EXAMPLE



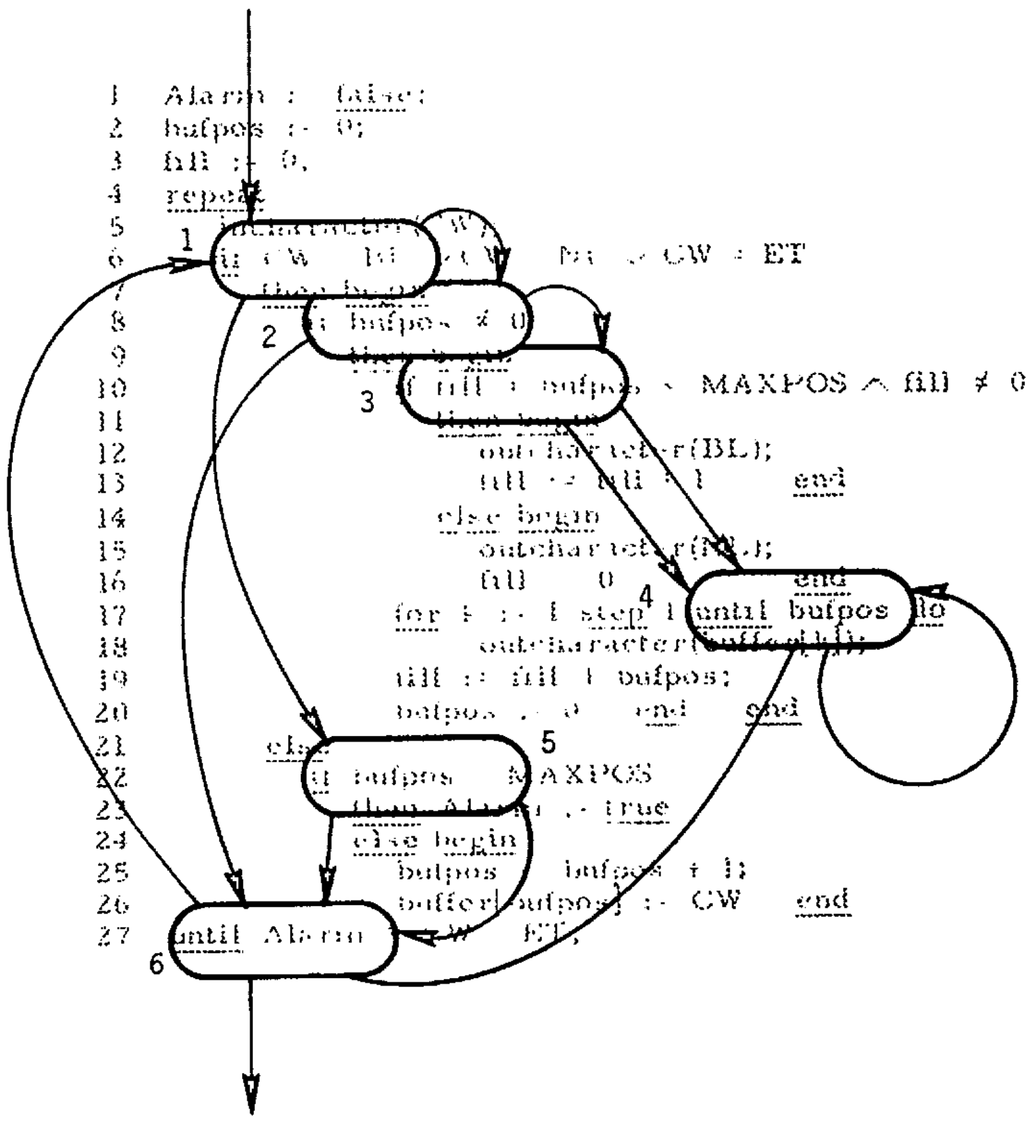
0th DECISIONAL LEVEL: (SEE BELOW)

1st DECISIONAL LEVEL: TEST initialization AND alternative AND wrapup

TEST setup AND teardown

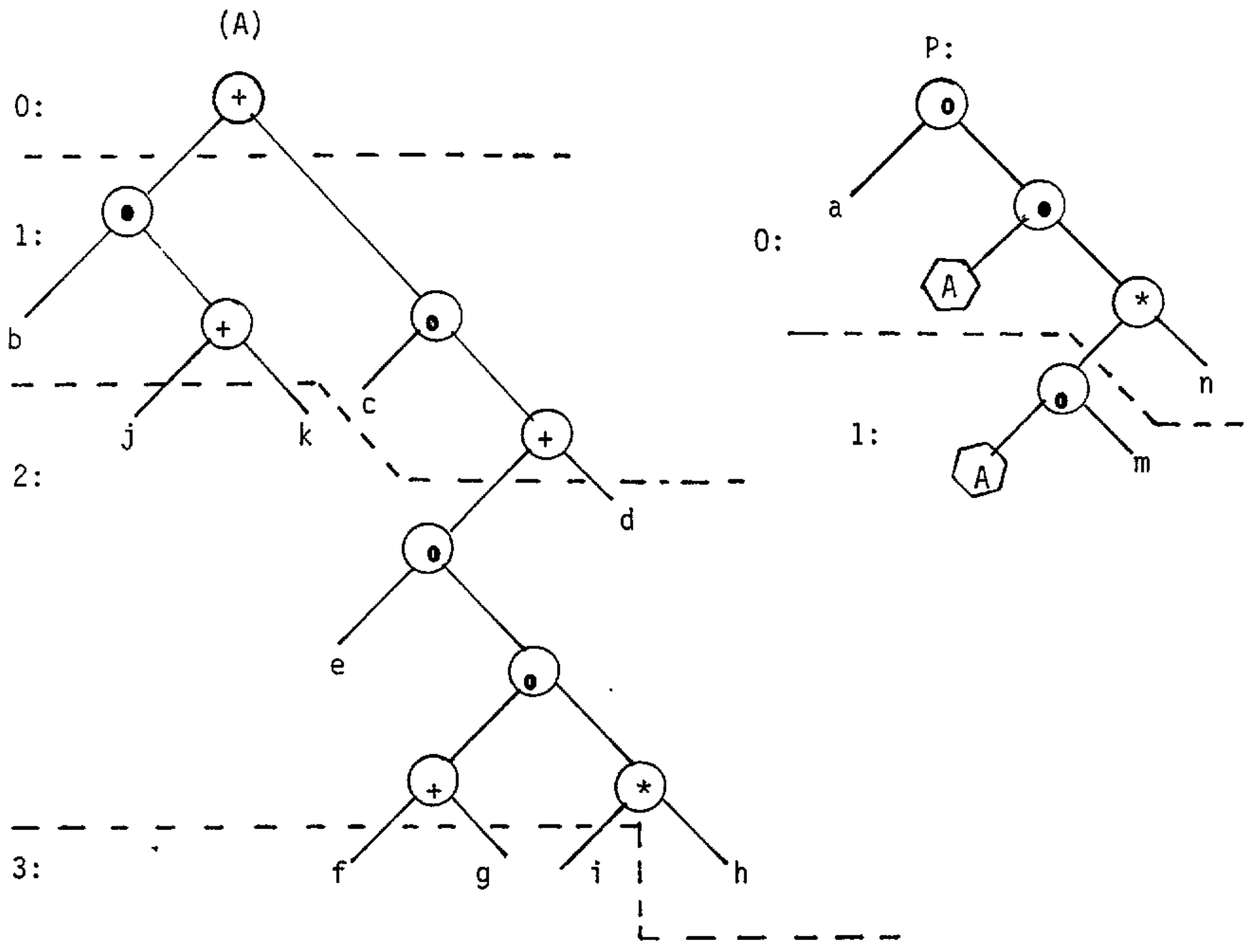
2nd DECISIONAL LEVEL: TEST F^k AND incrementation

EXAMPLE BY NAUR SHOWING DIRECTED GRAPH STRUCTURE IN OVERLAY



```
Alarm := false;
bufpos := 0;
fill := 0;
repeat
   incharacter(CW);
   if CW = BL ∨ CW = NL ∨ CW = ET
   then begin
      if bufpos ≠ 0
      then begin
         if fill + bufpos < MAXPOS ∧ fill ≠ 0
         then begin
            outcharacter(BL);
            fill := fill + 1      end
         else begin
            outcharacter(NL);
            fill := 0      end
         for k := 1 step 1 until bufpos do
            outcharacter(buffer[k]);
         fill := fill + bufpos;
         bufpos := 0   end   end
   else
      if bufpos = MAXPOS
      then Alarm := true
      else begin
         bufpos := bufpos + 1;
         buffer[bufpos] := CW   end
until Alarm ∨ CW = ET;
```

HIERARCHICAL DECOMPOSITION OF NAUR'S EXAMPLE



NOTES

- THE SEGMENT "A" IS COPIED ONTO THE TREE FOR P IN TWO LOCATIONS
- EACH ELEMENT OF "A" RESIDES AT TWO LEVELS OF DECISIONAL DEPTH

IMPLIED TESTING SCHEME FOR NAUR'S EXAMPLE BASED ON DECOMPOSITION TREE

- Test *A* in pieces in the following way:
    - Test for *j* and *k*.
    - Test for *d*.
    - Test for *f* and *g*.
    - Test the iteration on *i*.
- Test *P* - *A* in the following way:
    - Test for *n*.
    - Test the iteration on *A*.
- Test *P* in the following way:
    - Test *P* with *A* at zeroth decisional level.
    - Test *P* with *A* at first decisional level.

IHD EXAMPLE (brfex1.*)

PATH SET:

```
1 2 9 10 25 26
1 2 9 11 (13) 12 (15 (17 18 20) 16 21 23) 14 24 25 26
1 2 9 11 (13) 12 (15 (17 18 20) 16 22 23) 14 24 25 26
1 2 9 11 (13) 12 (15 (17 19 20) 16 21 23) 14 24 25 26
1 2 9 11 (13) 12 (15 (17 19 20) 16 22 23) 14 24 25 26
1 2 9 11 (13) 12 (15 16 21 23) 14 24 25 26
1 2 9 11 (13) 12 (15 16 22 23) 14 24 25 26
1 2 9 11 (13) 12 14 24 25 26
1 2 9 11 12 (15 (17 18 20) 16 21 23) 14 24 25 26
1 2 9 11 12 (15 (17 18 20) 16 22 23) 14 24 25 26
1 2 9 11 12 (15 (17 19 20) 16 21 23) 14 24 25 26
1 2 9 11 12 (15 (17 19 20) 16 22 23) 14 24 25 26
1 2 9 11 12 (15 16 21 23) 14 24 25 26
1 2 9 11 12 (15 16 22 23) 14 24 [illegible]
1 2 9 11 12 14 24 25 [illegible]          [illegible] 23) 14 24 25 26
1 3 (5 6 8) 4 [illegible]          [illegible] 20) 16 22 23) 14 24 25 26
1 3 [illegible]          [illegible] (17 19 20) 16 21 23) 14 24 25 26
            [illegible] 12 (15 (17 19 20) 16 22 23) 14 24 25 26
      [illegible] 4 9 11 (13) 12 (15 16 21 23) 14 24 25 26
[illegible] (5 7 8) 4 9 11 (13) 12 (15 16 22 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 (13) 12 14 24 25 26
1 3 (5 7 8) 4 9 11 12 (15 (17 18 20) 16 21 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 12 (15 (17 18 20) 16 22 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 12 (15 (17 19 20) 16 21 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 12 (15 (17 19 20) 16 22 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 12 (15 16 21 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 12 (15 16 22 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 12 14 24 25 26
1 3 4 9 10 25 26
1 3 4 9 11 (13) 12 (15 (17 18 20) 16 21 23) 14 24 25 26
1 3 4 9 11 (13) 12 (15 (17 18 20) 16 22 23) 14 24 25 26
1 3 4 9 11 (13) 12 (15 (17 19 20) 16 21 23) 14 24 25 26
1 3 4 9 11 (13) 12 (15 (17 19 20) 16 22 23) 14 24 25 26
1 3 4 9 11 (13) 12 (15 16 21 23) 14 24 25 26
1 3 4 9 11 (13) 12 (15 16 22 23) 14 24 25 26
1 3 4 9 11 (13) 12 14 24 25 26
1 3 4 9 11 12 (15 (17 18 20) 16 21 23) 14 24 25 26
1 3 4 9 11 12 (15 (17 18 20) 16 22 23) 14 24 25 26
1 3 4 9 11 12 (15 (17 19 20) 16 21 23) 14 24 25 26
1 3 4 9 11 12 (15 (17 19 20) 16 22 23) 14 24 25 26
1 3 4 9 11 12 (15 16 21 23) 14 24 25 26
1 3 4 9 11 12 (15 16 22 23) 14 24 25 26
1 3 4 9 11 12 14 24 25 26
```

IHD EXAMPLE (brfex1.*)

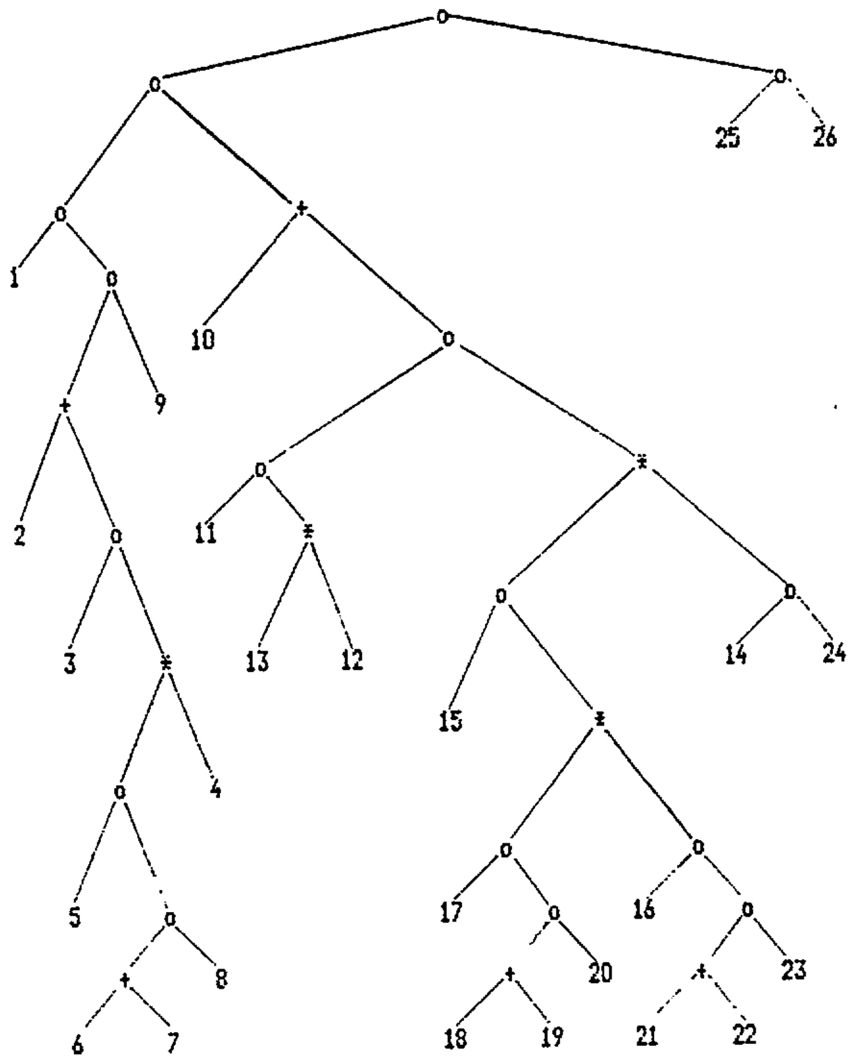
DIGRAPH: COVER SET:

```
 26
  0   36
 36   44
 36   37
 37   44
 37   38
 38   41
 38   41
 41   37
 44   50
 50  136
 50   52
 52   62
 52   52
 62  113
 62   66
 66   80
 66   67
 67   71
 67   71
 71   66
 80   83
 80   83
 83   62
113  136
136  151
151    0
```

```
1 2 9 10 25 26
1 3 (5 6 8) 4 9 11 (13) 12 (15 (17 18 20) 16 21 23) 14 24 25 26
1 3 (5 7 8) 4 9 11 (13) 12 (15 (17 19 20) 16 22 23) 14 24 25 26
```

IHD EXAMPLE (brfex1.*)

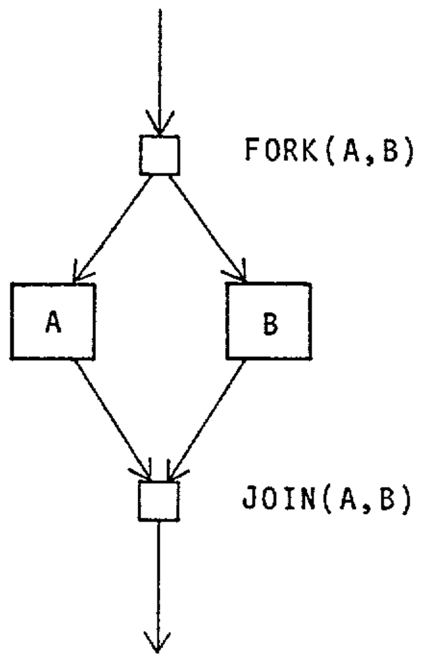
TREE:



SOFTWARE RESEARCH

CONCURRENT PROGRAMS (FORK/JOIN PRIMITIVES)

- Apply principle of separability
- Assumes call FORK/JOIN (A,B)
- Tests required are:
  - TEST A
  - TEST B
  - TEST FORK(A,B)
  - TEST JOIN(A,B)



COOPERATING PROGRAMS

- Two (or more) programs intercommunicating in a rigorous intercommunications scheme
- Test by splitting the communication paths
- Note that standard communications primitives are *designed* so that they survive the dislocation required by testing.

$$\begin{tikzcd}
\boxed{A} \arrow[r, shift left] & \boxed{B} \arrow[l, shift left]
\end{tikzcd}$$

$$\begin{tikzcd}
I_A \arrow[r] & \boxed{A} \arrow[r] & \boxed{B} \arrow[r] & O_B
\end{tikzcd}$$

$$\begin{tikzcd}
I_B \arrow[r] & \boxed{B} \arrow[r] & \boxed{A} \arrow[r] & O_A
\end{tikzcd}$$

CAUSE/EFFECT GRAPHING

- **CAUSE/EFFECT GRAPH**

  A method for expressing the relationships between:

  * *Causes*: Explicit and implicit input conditions
  * *Effects*: Responses by the program (output conditions)

- **PROCEDURE**

  Construct a cause/effect graph (roughly) as follows:

  * Identify from the program specifications all implicit and explicit causes.
    - Assumes a verbal, or at least English-language, level of specification.
    - May require detailed study.
  * Assign each cause a number.
  * Repeat for program effects.
  * Identify all relationships between causes and effects, using the potential relationships:
    - and, or, not, exor...
    - if (cause) then (effect)
      if (cause) then (intermediate term)
  * Draw a graph representing these relationships.
  * Design tests based on a decision table representing all of the legitimate cause/effect relationships.
  * Verify that all of the program predicate outcomes (C1 measure) have occurred at least once.

MYERS' EXAMPLE OF CAUSE/EFFECT GRAPHS

PROBLEM STATEMENT

3.4 The CHANGE Subcommand

The CHANGE subcommand is used to modify a character string in the "current line" of the file being edited.

3.4.1 Inputs

The syntax of the subcommand is:

C /string1/string2

String1 represents the character string you want to replace. It can be from 1 through 30 characters long and can contain any characters except "/". String2 represents the character string that is to replace string1. It can be from 0 through 30 characters long and can contain any characters except "/". If string2 is omitted (zero length), string1 is simply deleted.

At least one blank must follow the command name "C".

3.4.2 Outputs

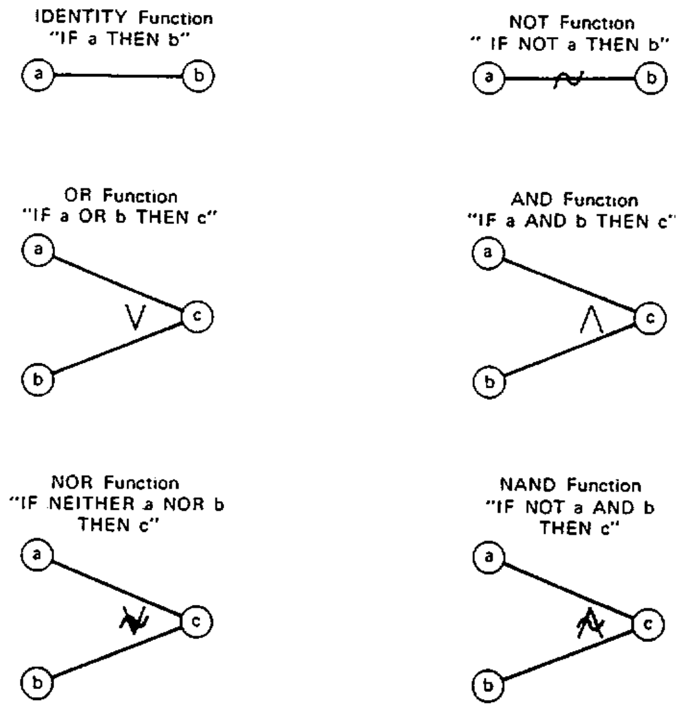
The changed line is printed on the terminal if the command is successful. If the change cannot be made because string1 cannot be found in the current line, the message "NOT FOUND" is printed. If the command syntax is incorrect, the message "INVALID SYNTAX" is printed.

3.4.3 System Transformations

If the syntax is valid and string1 can be found in the current line, then string1 is removed from the line and string2 is inserted in its place. The line is expanded or contracted as necessary based on the length differences between string1 and string2. If the command syntax is invalid, or if string1 cannot be found in the current line, the line is not changed.

MYERS' EXAMPLE OF CAUSE/EFFECT GRAPHS

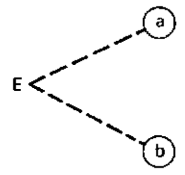
CAUSE/EFFECT RELATIONSHIPS AND REPRESENTATION



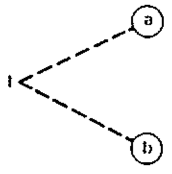
MYERS' EXAMPLE OF CAUSE/EFFECT GRAPHS

CAUSE/EFFECT CONSTRAINTS AND REPRESENTATION

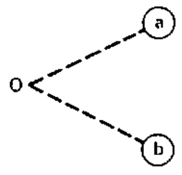
EXCLUSIVE Constraint
"AT MOST ONE OF a b CAN BE INVOKED"



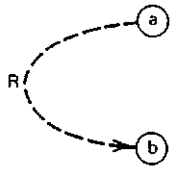
INCLUSIVE Constraint
'AT LEAST ONE OF a, b MUST BE INVOKED'



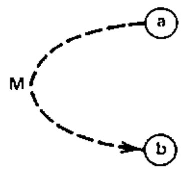
ONE–ONLY–ONE Constraint
"ONE AND ONLY ONE OF a, b CAN BE INVOKED"



REQUIRES Constraint
"IF a IS INVOKED THEN b MUST BE INVOKED"

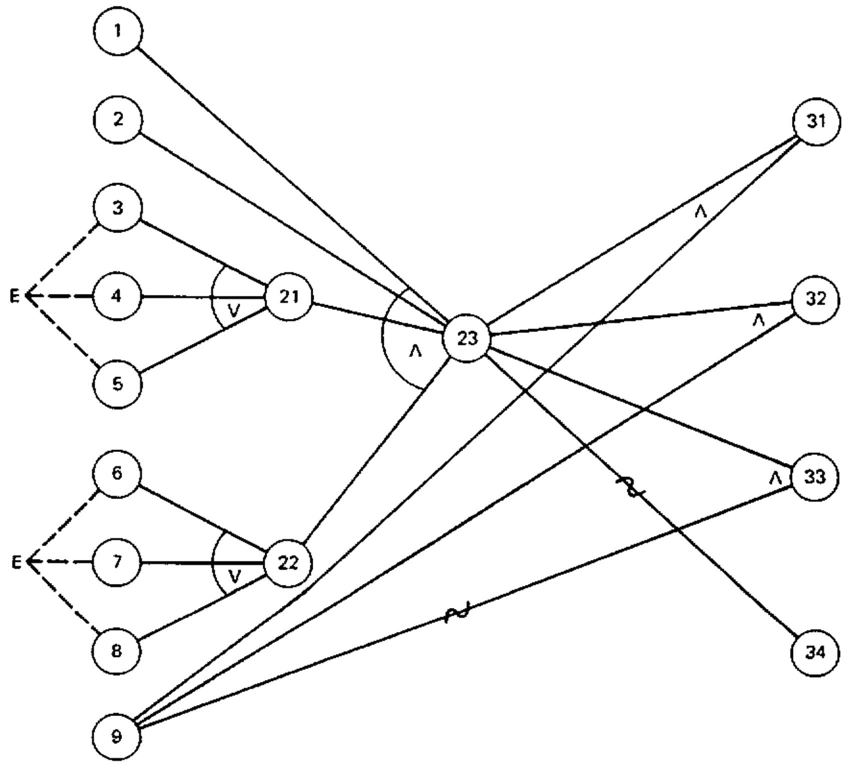


MASKS Constraint
"EFFECT a MASKS OBSERVANCE OF EFFECT b"



MYERS' CAUSE/EFFECT GRAPHS EXAMPLE

FINAL CAUSE/EFFECT GRAPH



CAUSES (INTERMEDIATE RELATIONSHIPS) EFFECTS

MYERS' EXAMPLE OF CAUSE/EFFECT GRAPHS

RESULTING DECISION TABLE

TESTS

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
CAUSES	1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	S
	2	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	S	I
	3	I	I	I	S	S	S	S	S	S	I	I	I	S	S	S	S	S	S	S	S	S	S
	4	S	S	S	I	I	I	S	S	S	S	S	S	I	I	I	S	S	S	S	S	S	S
	5	S	S	S	S	S	S	I	I	I	S	S	S	S	S	S	I	I	I	S	I	I	I
	6	I	S	S	I	S	S	I	S	S	I	S	S	I	S	S	I	S	S	S	S	S	S
	7	S	I	S	S	I	S	S	I	S	S	I	S	S	I	S	S	I	S	S	S	S	S
	8	S	S	I	S	S	I	S	S	I	S	S	I	S	S	I	S	S	I	I	S	I	I
	9	I	I	I	I	I	I	I	I	I	S	S	S	S	S	S	S	S	S	X	X	X	X
EFFECTS	31	P	P	P	P	P	P	P	P	P	A	A	A	A	A	A	A	A	A	A	A	A	A
	32	P	P	P	P	P	P	P	P	P	A	A	A	A	A	A	A	A	A	A	A	A	A
	33	A	A	A	A	A	A	A	A	A	P	P	P	P	P	P	P	P	P	A	A	A	A
	34	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	P	P	P	P

S: Suppressed
I: Invoked
X: Don't care
A: Absent
P: Present

SOFTWARE RESEARCH

THEORETICAL RESULTS

- **GOAL**

  State whether defects have been reliably removed by test or tests.

- **RELIABLE TESTING THEOREM**

  - *Theorem States*: There are Test Data Selection Criteria such that when a program operates on a subdomain D of the entire input domain S and also meets requirements of theorem, then the program can be pre-edited to operate on *all* inputs.
  - *Issue Is:* **How much less than all possible inputs is minimum needed for assurance of defect-free software?**

- **CURRENT STATUS**

  - *Some programs have reliable tests, some do not.*
  - *Under certain technical restrictions testing is known to be reliable against non-structural errors.*
  - *Some errors are very difficult to find reliably.*

  NOTE: Any error must be made manifest by some combination of inputs, but the problem is to determine what that set of inputs is without encountering combinatoric limits on testing complexity.

- **RESEARCH METHODS**

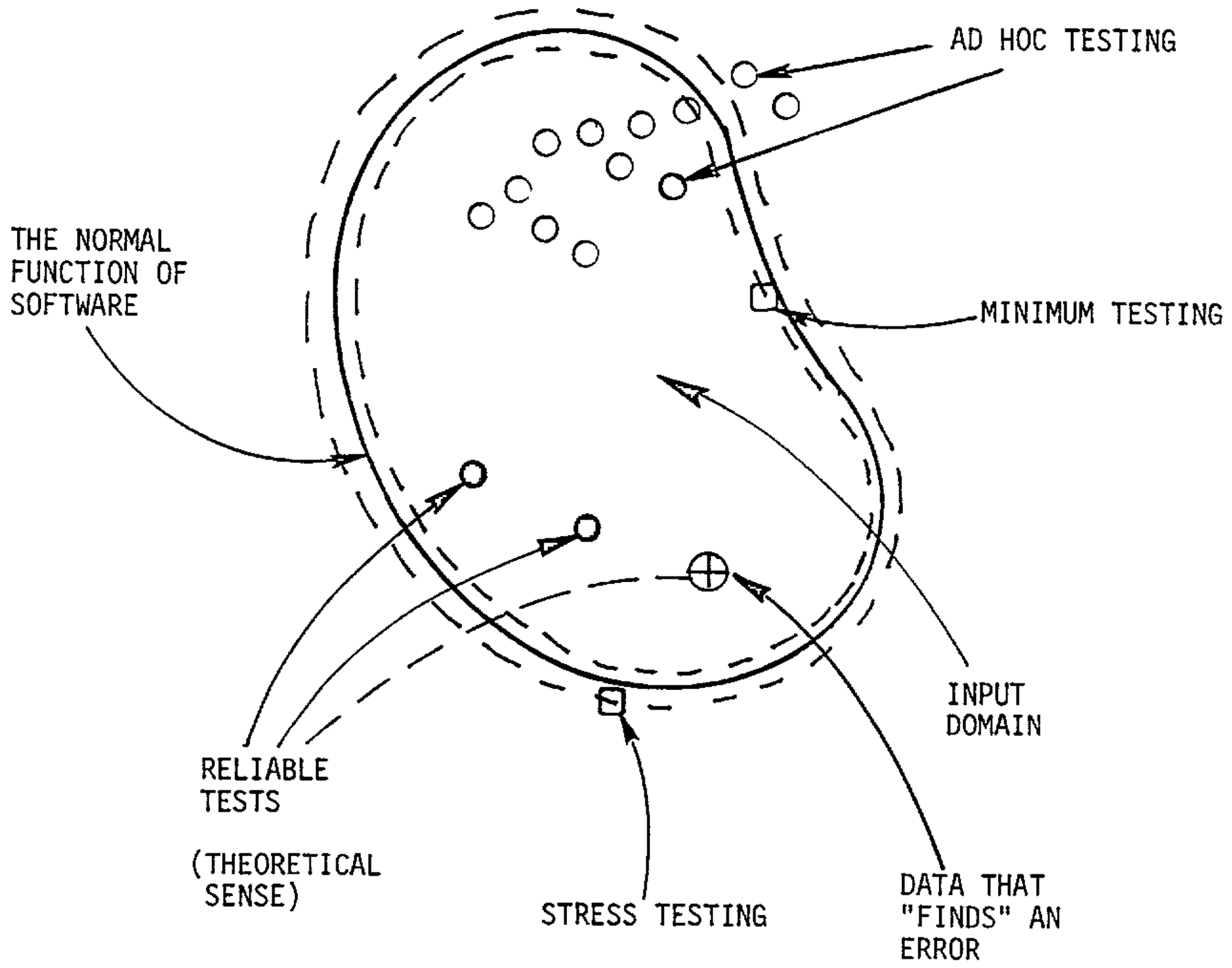
  - *Domain Refinement*
  - *How data characterizes programs*

REFERENCES:

W.E. Howden, "Reliability of Path Analysis Testing Strategy," IEEE Trans. Software Engineering, June, 1975.

J.B. Goodenough and S.L. Gerhart, "Toward A Theory of Test Data Selection," IEEE Trans. Software Engineering, June 1975.

SOME RELATIONSHIPS BETWEEN SOFTWARE FUNCTION AND LEVEL OF TESTING



STATEMENT TYPES AND CORRESPONDING FUNCTIONS

Statement	Functions
1. Assignment	Data access, data storage, arithmetic expression
2. Conditionals	Data access, arithmetic expression, relational expression, boolean expression
3. Loops	Loop entry functions: data access, arithmetic expression, relational expression, boolean expression
	Loop exit functions: data access, arithmetic expression, relational expression, boolean expression
	Index initialization: data access, data storage, arithmetic expression
	Indexing: data access, data storage, arithmetic expression

Reference: Howden, 1980

RELIABLE TEST DATA FOR SIMPLE ERRORS IN STATEMENT FUNCTIONS

Functions	Reliable Test Data
1. Data access	Unique value for variable
2. Data storage	New value for variable
3. Arithmetic expression	Evaluates to non-zero quantity
4. Relational expressions (of the form E_1 r E_2)	Tests that evaluate E_1 and E_2 so that $E_1 < E_2$, $E_1 = E_2$ and $E_1 > E_2$; tests that evaluate E_1 and E_2 so that for (i) $r = <, \geq$: $E_2 - E_1$ is maximal and < 0 $E_2 - E_1$ is minimal and ≥ 0 (ii) $r = >, \leq$: $E_2 - E_1$ is minimal and > 0 $E_2 - E_1$ is maximal and ≤ 0 (iii) $r = =, \neq$: $E_2 - E_1 = 0$.
5. Boolean expressions (of the form $B(E_1, E_2, \ldots, E_n)$)	Tests that evaluate E_i, $1 \leq i \leq n$, so that all possible combinations of True and False are generated

Reference: Howden, 1980

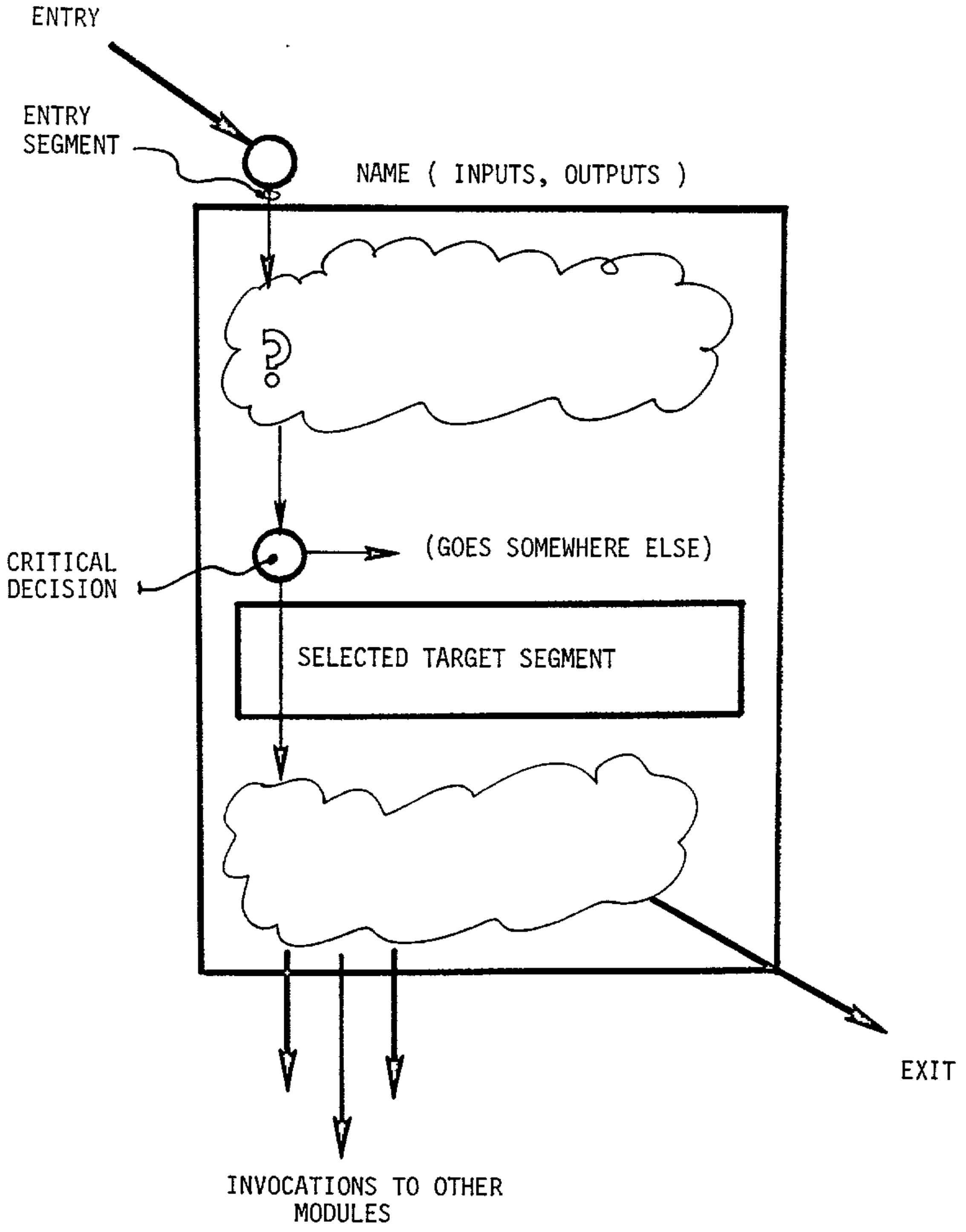
SOFTWARE RESEARCH

TEST DATA GENERATION PROBLEM

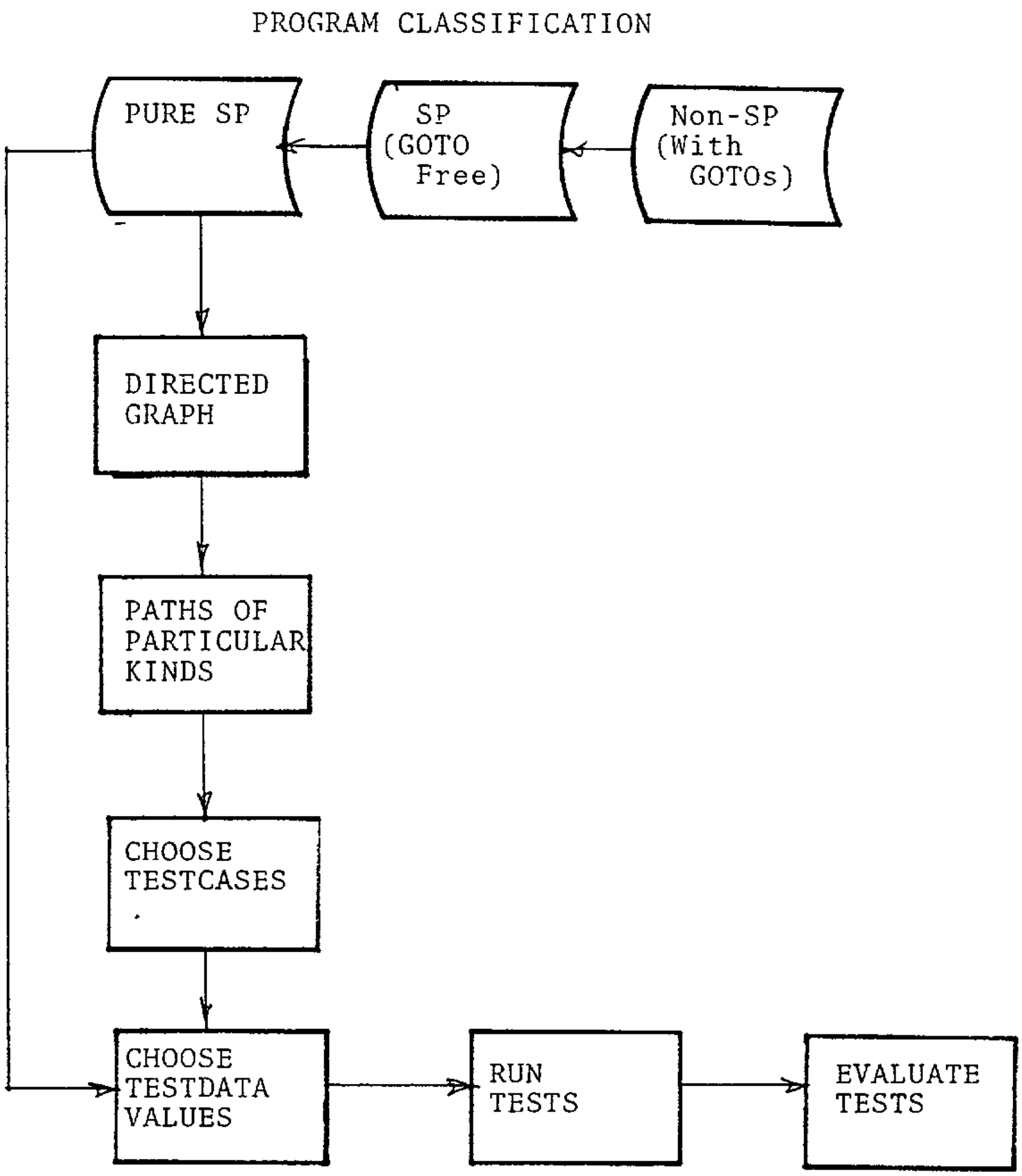
- **GENERAL STATEMENT**
  - Find test data that forces a program to execute a previously unexercised segment.
  - Choose data values automatically.
  - Unsolvable in general.
- **APPROACH**
  - Choose candidate path.
  - Analyze formulas (path conditions).
  - Solve set of inequalities.
- **LIMITATIONS**
  - Combinatorics
  - Complexity
  - Non-Linearity

SOFTWARE RESEARCH

A REPRESENTATION OF THE TEST DATA GENERATION PROBLEM

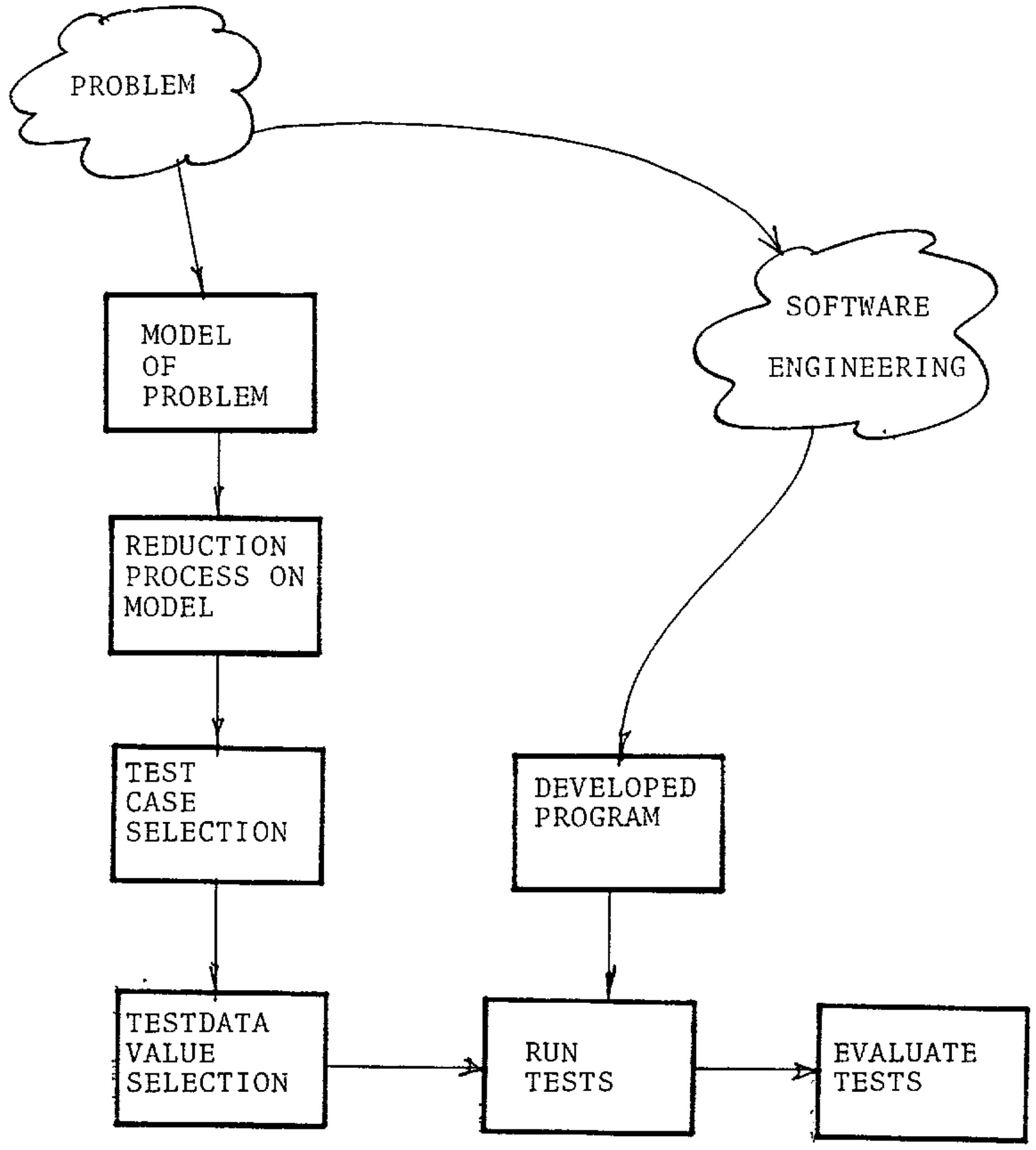


WHITE BOX TESTING -- SUMMARY



SOFTWARE RESEARCH

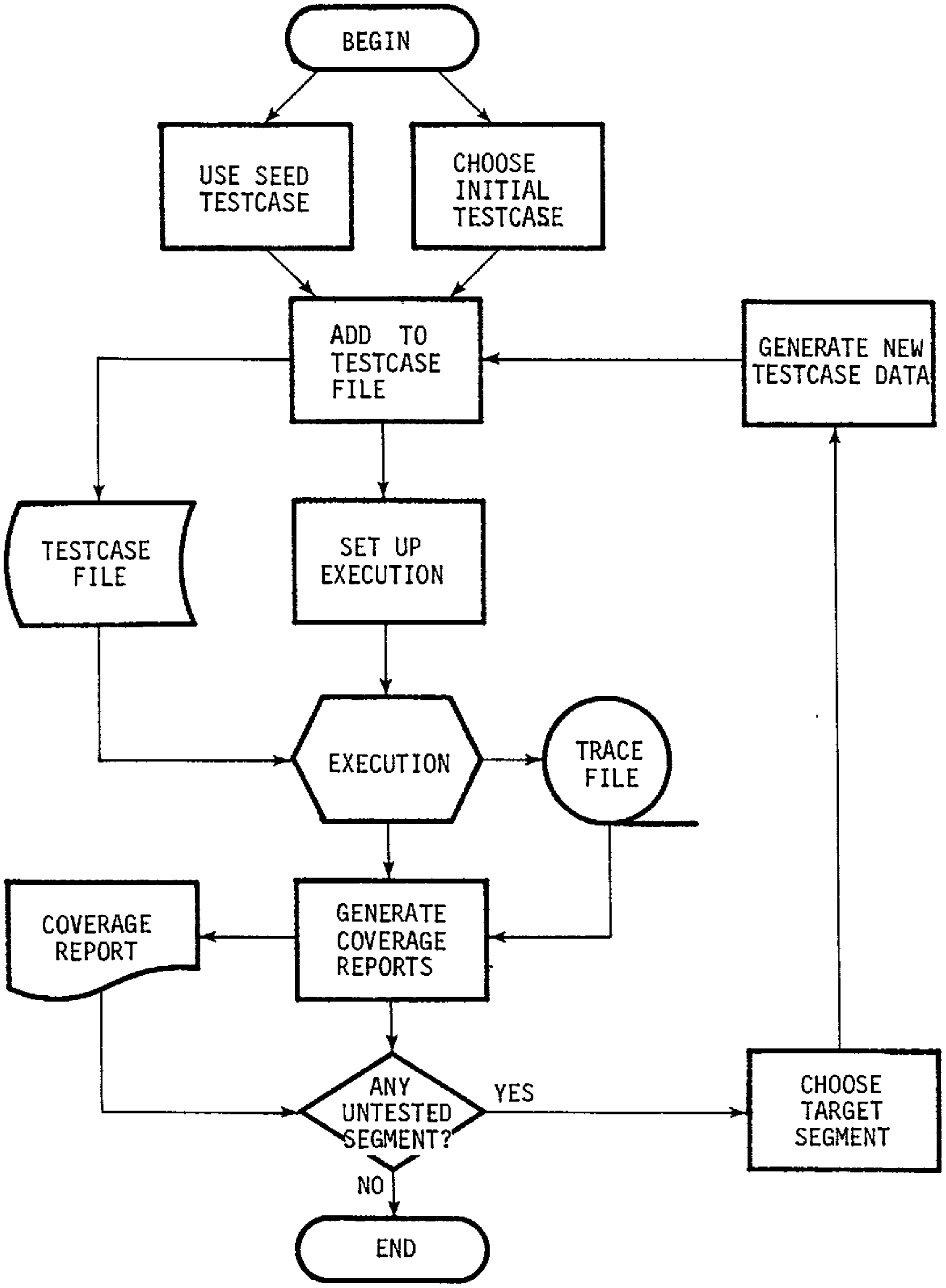
BLACK BOX TESTING -- SUMMARY



LEVELS OF TESTING METHODOLOGY

- **SINGLE MODULE TESTING**
  - Comprehensive exercise of single program(s)
  - Exhaustive investigation of behavior of module
  - Maximum level of quality assurance
  - "No system is better tested than the level of testing attained for the least-tested module."
- **MULTIPLE MODULE (SUB-SYSTEM) TESTING**
  - Demonstration of functional behavior
  - Integration of proven/tested modules into coherent sub-system
  - Localization of computational resource
- **INTERFACE TESTING**
  - Demonstration of quality of interaction between subsystems
  - Protection of subsystems from each other
- **SYSTEM TESTING**
  - Formal acceptance testing and/or certification of software system
  - Overall demonstration of function
  - Assessment of service-ability, future "reliability," other measures or robustness

SINGLE MODULE SEGMENT TESTING METHODOLOGY



SOFTWARE RESEARCH

STRUCTURE BASED COVERAGE MEASURES -- MODULE LEVEL

Module Level Coverage Measures

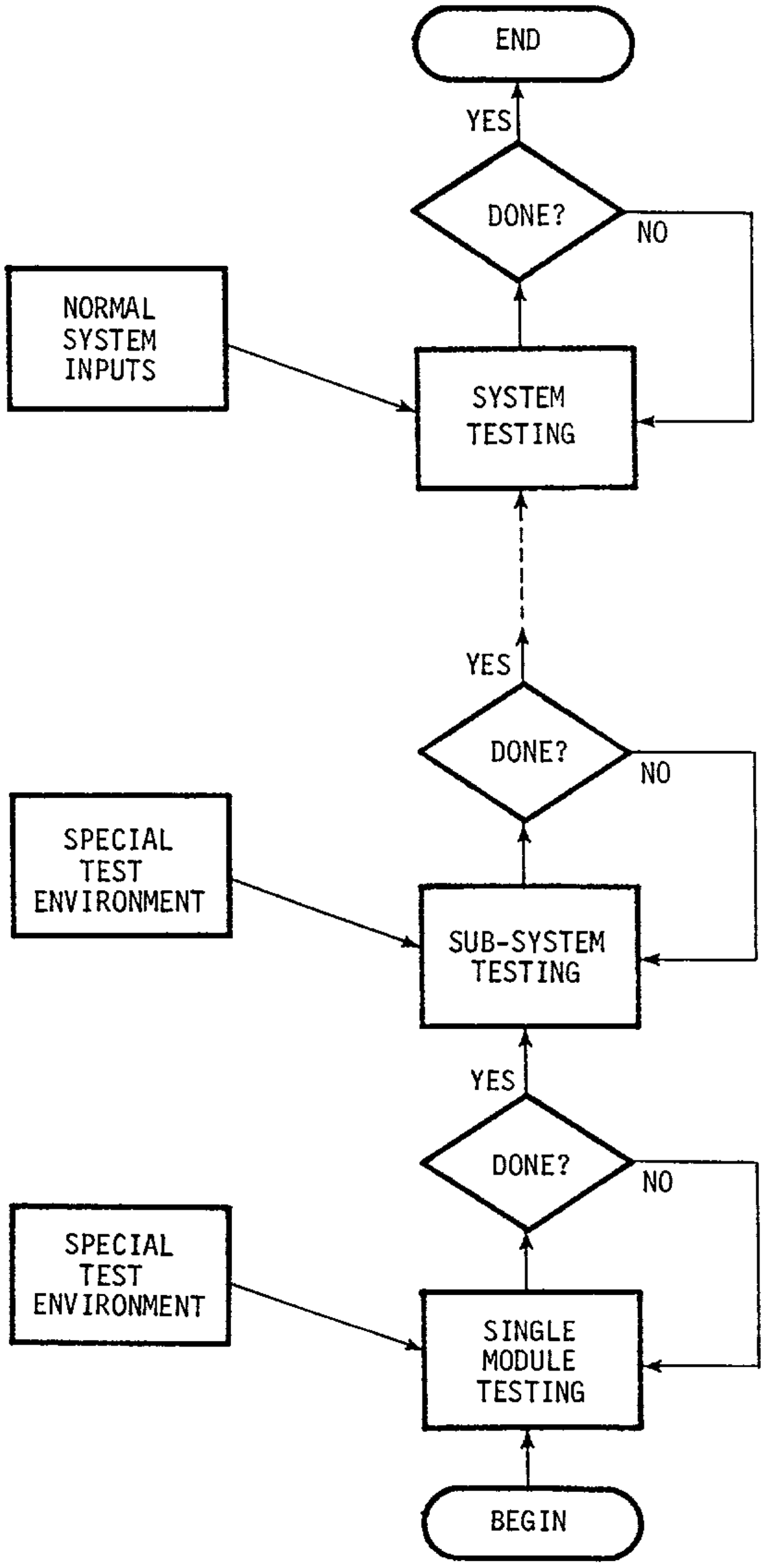
Name	Short Description	Comments
C0	Execute all statements in a a program.	Historically this is what most programmers "think" is the right level of testing, but it may leave out many segments.
C1-	Execute all non-null segments in each program.	This measure is close to the full C1 measure (below) and may in some cases be equivalent to it.
C1	Execute all segments in each program.	This is the basic measure of testing coverage now advocated by most experts. It has the intuitive benefit of attempting to exercise each "part" of a program.
C1+	C1 and also all interior and exterior features of iterations.	This measure extends C1 to include some of the basic properties of program iterations or loops in a way similar to that in proof of correctness.
C1p	C1 and each relational term to each possible outcome.	This measure extends C1 by requiring that each relational expression in any logical expression be exercised to each possible outcome. That is, predicates must be broken into their simple parts and each part tested.
C2	C1 and also one exterior and an upper and lower interior test.	This measure extends C1+ so that three properties of each iteration are checked: no iteration, a lower iteration count, and an upper iteration count. Each must be achieved on successive encounters of the loop.
C3	C2 plus each different non-iterative paths.	This extends C2 to include all of the non-iterative paths within the program structure. This may be difficult to achieve in practice.
Cik	C1 plus one test for each iteratiion i = 1, 2,...,k times.	This measure requires that each cycle in the program be executed a fixed number of times, i = 1, 2, 3, ..., k, where k is normally set to an upper bound of k = 2.

SOFTWARE RESEARCH

STRUCTURE BASED COVERAGE MEASURES -- SYSTEM LEVEL

Name	Short Description	Comments
S0	Invoke all modules at least once.	This measure is insufficient to assure full exercise of a software system structure.
S1	All invocations to modules exercised at least once.	This is the minimum useful system level structural exercise measure.
S2	All invocations to a module for each possible value of logical expression (actual) parameters.	This measure extends S1 to account for the case when the actual parameter list has a logical expression, and requires that each possible outcome be exercised (similar to C1).
S2p	All invocations to a module for each possible logical outcome.	This measure extends S1 to include the case when the module has alternate logical outcomes, such as RETURN i or error modes (language dependent).
Sd	Every module down to a prespecified decisional depth.	This measure tries to require tests that execute the "most complex" parts of a software system, as measured by the decisional depth of the most deeply constrained segment.
St	All calling chains from the top module to any other module.	This measure requires that each distinct calling chain from the topmost module to every other module in the software system.
S3	One invocation for all major equivalence classes possible.	This measure tries to capture "one test for each different equivalence class of invocation input," a kind of inter-modular data-exercise measure.

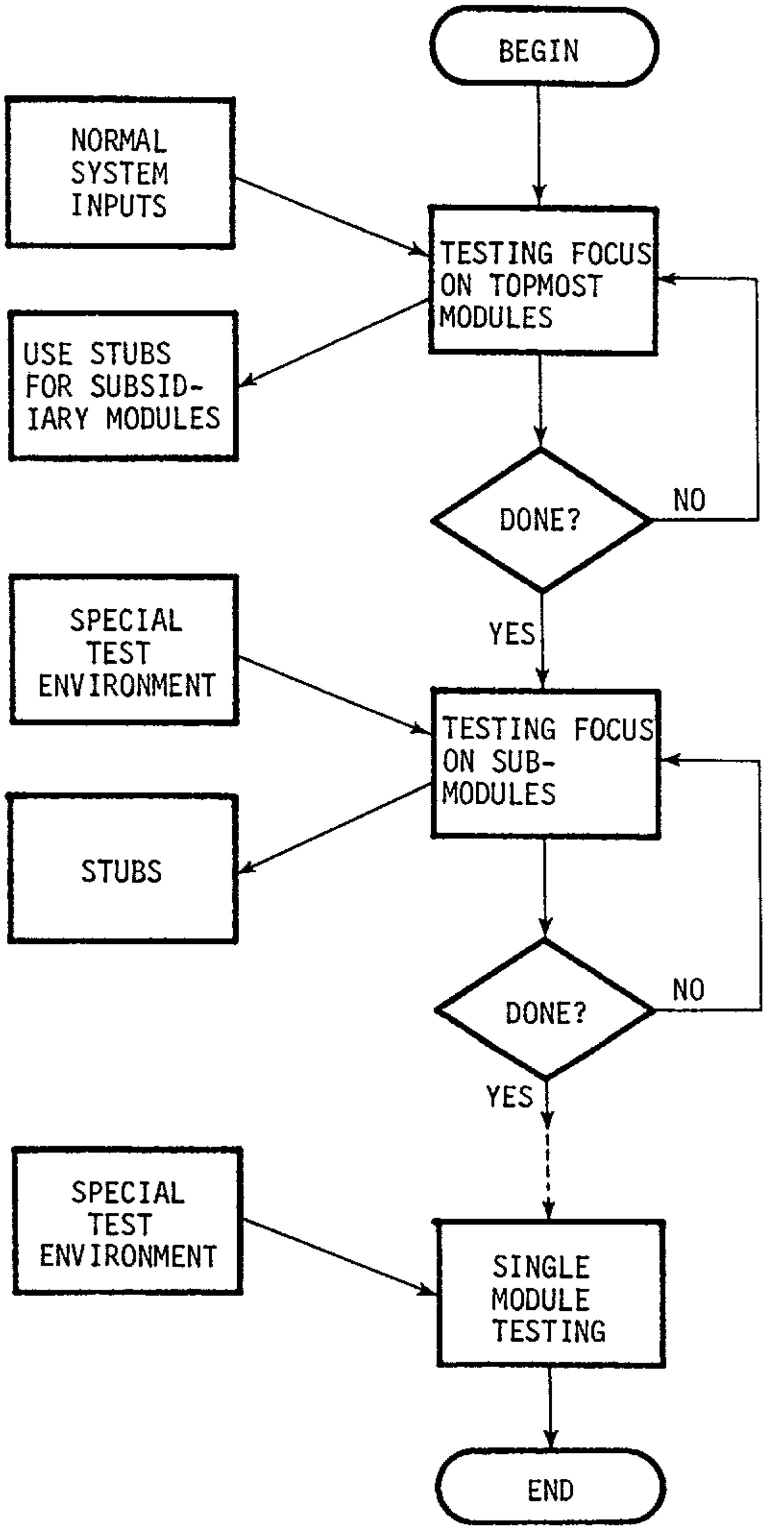
BOTTOM-UP TESTING METHODOLOGY



A-231-5

SOFTWARE RESEARCH

TOP-DOWN TESTING METHODOLOGY



A-231-4

TCAT/C -- EXAMPLE COVERAGE ANALYSIS

ANALYSIS OF A SMALL "C" IMPLEMENTED SOFTWARE SYSTEM

STATISTICS ON "C" SYSTEM TESTED:

15 "C" MODULES

241 SEGMENTS

AVERAGE OF 16.07 SEGMENTS/MODULE

APPROXIMATELY 1500 LINES OF "C" CODE

STATISTICS ON THE SET OF TESTS:

32 SEPARATE TESTS

TEST EFFICIENCY: 7.53 SEGMENTS/TEST

LEAST COVERAGE OBTAINED IN ONE TEST:	2.49%
MOST COVERAGE OBTAINED IN ONE TEST:	70.12%

AVERAGE COVERAGE PER TEST: 50%

INITIAL C1 VALUE:	36.51%
FINAL C1 VALUE:	94.19%
LEAST INVOKED MODULES:	1 TIME (1 MODULE)
MOST INVOKED MODULES:	906 TIMES (2 MODULES)
LEAST TESTED MODULE:	"UPDATE" WITH 76.47% (3 INVOCATIONS)
MOST TESTED MODULE:	"GENDATA" WITH 100.00% (167 INVOCATIONS, 38 SEGMENTS, 6346 SEGMENT HITS)

SOFTWARE RESEARCH

COVERAGE ANALYSIS EXAMPLE QAT-50-2

```
RESULTS AFTER TEST NO. 1

TCAT COVERAGE ANALYZER,  COVER VERSION 1.8 (80 COLUMN)
(C) COPYRIGHT 1984 BY SOFTWARE RESEARCH ASSOCIATES

+----------------------------------------------------------------------------+
I                          I        THIS TEST          I  CUMULATIVE SUMMARY  I
+                          +---------------------------+----------------------+
I                          I          NO. OF           I          NO. OF      I
I MODULE         NUMBER OF I NO. OF   SEGMENTS  C1%    I NO. OF   SEGMENTS C1% I
I NAME:          SEGMENTS: I INVOKES  HIT       COVER  I INVOKES  HIT    COVER I
+----------------------------------------------------------------------------+
I MAIN                  48 I     1      15     31.25   I     1      15   31.25 I
I MY_FOPEN               5 I     2       2     40.00   I     2       2   40.00 I
I BUILDTBL              55 I     1      31     56.36   I     1      31   56.36 I
I READCOM                5 I     1       4     80.00   I     1       4   80.00 I
I ENTERDATA             11 I    71       5     45.45   I    71       5   45.45 I
I GENDATA               38 I    11      21     55.26   I    11      21   55.26 I
I LOOKUP                 5 I    11       4     80.00   I    11       4   80.00 I
I PRINTNUM               9 I    11       6     66.67   I    11       6   66.67 I
I RANGE                 25 I     0       0      0.00   I     0       0    0.00 I
I ITOA                   7 I     0       0      0.00   I     0       0    0.00 I
I REVERSE                3 I     0       0      0.00   I     0       0    0.00 I
I GENRAND                5 I     0       0      0.00   I     0       0    0.00 I
I TOUCHFILE              5 I     0       0      0.00   I     0       0    0.00 I
I UPDATE                17 I     0       0      0.00   I     0       0    0.00 I
I FILECOPY               3 I     0       0      0.00   I     0       0    0.00 I
+----------------------------------------------------------------------------+
I TOTALS               241 I   109      88     36.51   I   109      88   36.51 I
+----------------------------------------------------------------------------+

CURRENT TEST MESSAGE (SAVED IN ARCHIVE):

RUNTIMEVERSION1.2. LAST UPDATED ON 6-12-84
```

SOFTWARE RESEARCH

COVERAGE ANALYSIS EXAMPLE QAT-50-3

RESULTS AFTER TEST NO. 10

```
TCAT COVERAGE ANALYZER, COVER VERSION 1.8 (80 COLUMN)
(C) COPYRIGHT 1984 BY SOFTWARE RESEARCH ASSOCIATES

+----------------------------------------------------------------------------+
I                              I       THIS TEST         I  CUMULATIVE SUMMARY  I
+                              +-------------------------+----------------------+
I                              I          NO. OF         I          NO. OF      I
I MODULE          NUMBER OF   I NO. OF   SEGMENTS  C1%  I NO. OF   SEGMENTS  C1%  I
I NAME:           SEGMENTS:   I INVOKES  HIT     COVER  I INVOKES  HIT     COVER  I
+----------------------------------------------------------------------------+
I MAIN                  48 I      1        3     6.25 I     10       19    39.58 I
I MY_FOPEN               5 I      1        2    40.00 I     17        4    80.00 I
I BUILDTBL              55 I      1       30    54.55 I      9       48    87.27 I
I READCOM                5 I      2        5   100.00 I     15        5   100.00 I
I ENTERDATA             11 I     71        5    45.45 I    355        9    81.82 I
I GENDATA               38 I      0        0     0.00 I     56       33    86.84 I
I LOOKUP                 5 I      0        0     0.00 I     53        5   100.00 I
I PRINTNUM               9 I      0        0     0.00 I     52        8    88.89 I
I RANGE                 25 I      0        0     0.00 I      4       17    68.00 I
I ITOA                   7 I      0        0     0.00 I    211        5    71.43 I
I REVERSE                3 I      0        0     0.00 I    211        3   100.00 I
I GENRAND                5 I      0        0     0.00 I      0        0     0.00 I
I TOUCHFILE              5 I      0        0     0.00 I      0        0     0.00 I
I UPDATE                17 I      0        0     0.00 I      0        0     0.00 I
I FILECOPY               3 I      0        0     0.00 I      0        0     0.00 I
+----------------------------------------------------------------------------+
I TOTALS               241 I     76       45    18.67 I    993      156    64.73 I
+----------------------------------------------------------------------------+

CURRENT TEST MESSAGE (SAVED IN ARCHIVE):

RUNTIMEVERSION1.2. LAST UPDATED ON 6-12-84
```

SOFTWARE RESEARCH

COVERAGE ANALYSIS EXAMPLE QAT-50-4

```
RESULTS AFTER TEST NO. 20

TCAT COVERAGE ANALYZER, COVER VERSION 1.8 (80 COLUMN)
(C) COPYRIGHT 1984 BY SOFTWARE RESEARCH ASSOCIATES

+----------------------------------------------------------------------------+
I                              I       THIS TEST         I  CUMULATIVE SUMMARY   I
+                              +-------------------------+-----------------------+
I                              I          NO. OF         I          NO. OF       I
I MODULE           NUMBER OF   I NO. OF   SEGMENTS  C1%  I NO. OF   SEGMENTS  C1% I
I NAME:            SEGMENTS:   I INVOKES  HIT     COVER  I INVOKES  HIT     COVER I
+----------------------------------------------------------------------------+
I MAIN                    48   I    1      15     31.25  I   20      20     41.67 I
I MY_FOPEN                 5   I    2       2     40.00  I   33       4     80.00 I
I BUILDTBL                55   I    1      41     74.55  I   19      52     94.55 I
I READCOM                  5   I    6       4     80.00  I   50       5    100.00 I
I ENTERDATA               11   I   20       9     81.82  I  622       9     81.82 I
I GENDATA                 38   I    7      21     55.26  I   89      37     97.37 I
I LOOKUP                   5   I    7       5    100.00  I   82       5    100.00 I
I PRINTNUM                 9   I    6       6     66.67  I   80       8     88.89 I
I RANGE                   25   I    2      19     76.00  I   13      23     92.00 I
I ITOA                     7   I   74       7    100.00  I  515       7    100.00 I
I REVERSE                  3   I   74       3    100.00  I  515       3    100.00 I
I GENRAND                  5   I    0       0      0.00  I    0       0      0.00 I
I TOUCHFILE                5   I    0       0      0.00  I    0       0      0.00 I
I UPDATE                  17   I    0       0      0.00  I    0       0      0.00 I
I FILECOPY                 3   I    0       0      0.00  I    0       0      0.00 I
+----------------------------------------------------------------------------+
I TOTALS                 241   I  200     132     54.77  I 2038     173     71.78 I
+----------------------------------------------------------------------------+

CURRENT TEST MESSAGE (SAVED IN ARCHIVE):

RUNTIMEVERSION1.2. LAST UPDATED ON 6-12-84
```

SOFTWARE RESEARCH

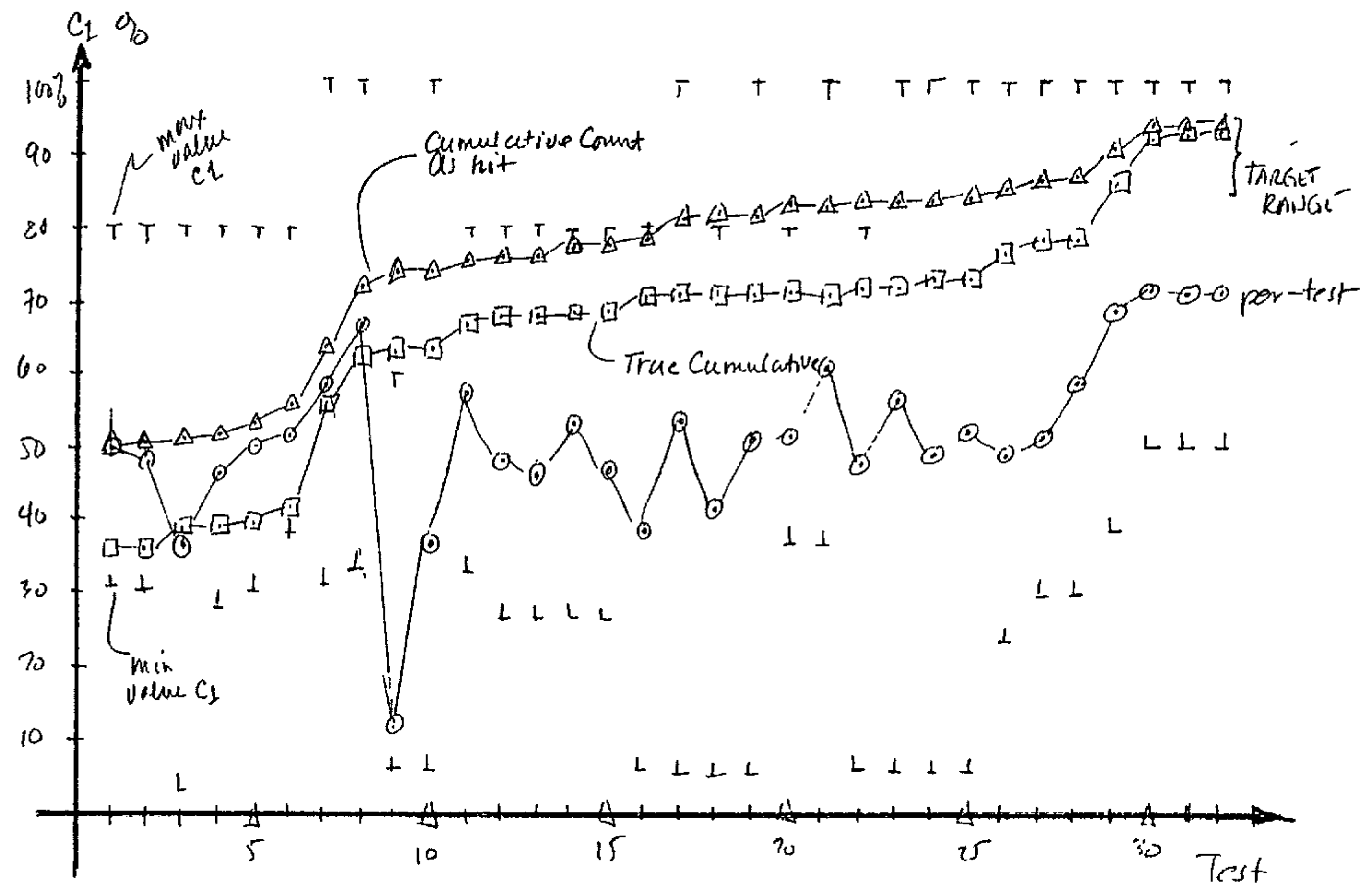
COVERAGE ANALYSIS EXAMPLE QAT-50-5

```
RESULTS AFTER TEST NO. 32

TCAT COVERAGE ANALYZER,  COVER VERSION 1.8 (80 COLUMN)
(C) COPYRIGHT 1984 BY SOFTWARE RESEARCH ASSOCIATES

+-------------------------------------------------------------------------------+
I                           I         THIS TEST          I   CUMULATIVE SUMMARY  I
+                           +----------------------------+-----------------------+
I                           I          NO. OF            I          NO. OF       I
I MODULE          NUMBER OF I NO. OF   SEGMENTS    C1%   I NO. OF   SEGMENTS  C1%I
I NAME:           SEGMENTS: I INVOKES  HIT       COVER   I INVOKES  HIT     COVERI
+-------------------------------------------------------------------------------+
I MAIN                  48  I     1      24      50.00   I    32      43    89.58 I
I MY_FOPEN               5  I     8       4      80.00   I    73       5   100.00 I
I BUILDTBL              55  I     1      47      85.45   I    31      53    96.36 I
I READCOM                5  I     2       4      80.00   I    82       5   100.00 I
I ENTERDATA             11  I    31       9      81.82   I   896       9    81.82 I
I GENDATA               38  I    11      28      73.68   I   167      38   100.00 I
I LOOKUP                 5  I     9       4      80.00   I   150       5   100.00 I
I PRINTNUM               9  I     9       8      88.89   I   147       9   100.00 I
I RANGE                 25  I     1      17      68.00   I    28      25   100.00 I
I ITOA                   7  I    11       5      71.43   I   906       7   100.00 I
I REVERSE                3  I    11       3     100.00   I   906       3   100.00 I
I GENRAND                5  I     0       0       0.00   I    22       5   100.00 I
I TOUCHFILE              5  I     0       0       0.00   I     1       4    80.00 I
I UPDATE                17  I     1      13      76.47   I     3      13    76.47 I
I FILECOPY               3  I     1       3     100.00   I     3       3   100.00 I
+-------------------------------------------------------------------------------+
I TOTALS               241  I    97     169      70.12   I  3447     227    94.19 I
+-------------------------------------------------------------------------------+

CURRENT TEST MESSAGE (SAVED IN ARCHIVE):

RUNTIMEVERSION1.2. LAST UPDATED ON 6-12-84
```

SOFTWARE RESEARCH

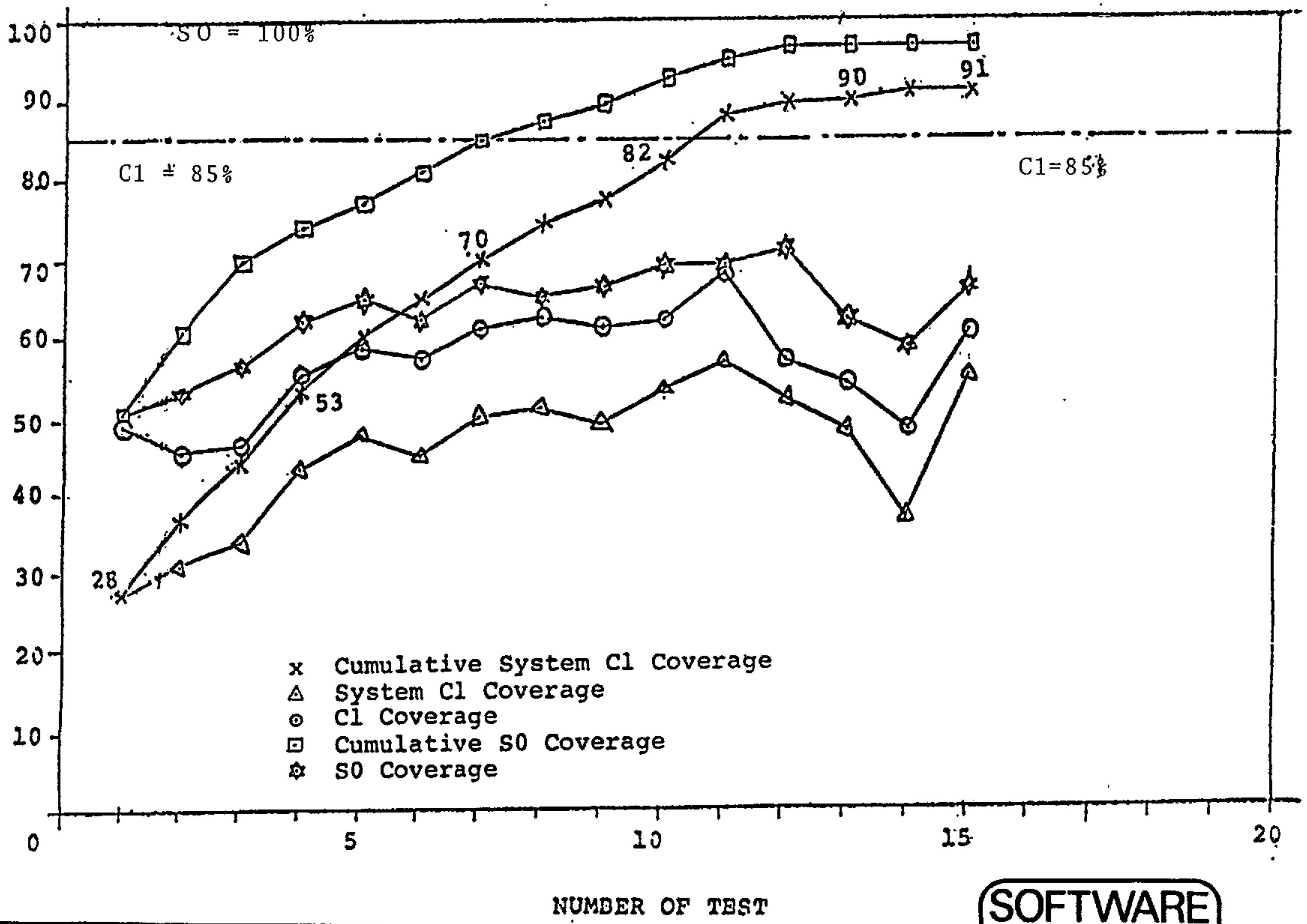
COVERAGE ANALYSIS EXAMPLE QAT-50-6

GRAPH OF OVERALL RESULTS



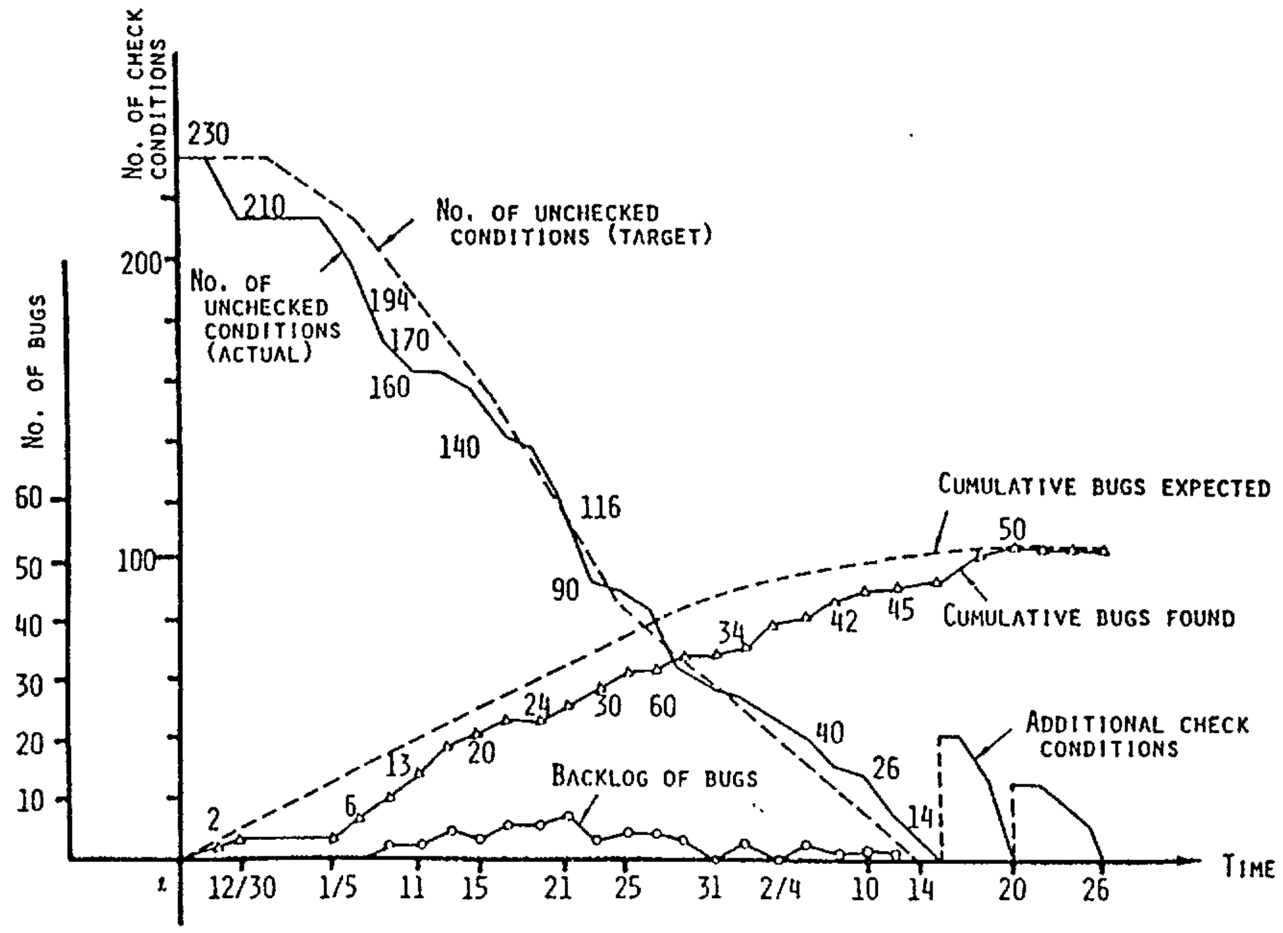
SOFTWARE RESEARCH

C1 GRAPH OF FM



SOFTWARE RESEARCH

NUMBER OF DEFECTS VS. BRANCH COVERAGE



Source: Reifer, 5th Annual Pacific Northwest Software Quality, '87

C1 ANALYSIS OF SMALL COBOL PROGRAM

- **BACKGROUND FACTS**
  - 2391 lines of text
  - 767 sentences
  - 371 segments
  - Initial coverage achieved: 63% C1
  - Final coverage achieved: 87.1% C1
- **QA METHODOLOGY USED**
  - Search for tests for untested segments.
  - Identify defects after each test.
  - Rerun tests upon correction of defect.
  - Minimal formal recordkeeping.
- **PRIOR HISTORY OF COBOL PROGRAM**
  - Less than six months operational use
  - Some defects found in operational use
  - Need for higher quality
- **RESULTS**
  - Defect discovery rate:
    - 1.3% of lines of text
    - 3.91% of sentences
  - Total defects found: 30
  - Untested segments: 6
  - Cost estimate: %40-60/defect

TYPICAL ACTIVITY BASED ON HIGH COVERAGE USING JAVS

- **PILOT PROJECT:**

  To apply existing (prototype) tools and related Quality Assurance methodology to practical problem.

- **BACKGROUND**

  Central Flow Control (CFC) Software for FAA written in JOVIAL/J2 Dialect

  * Approximately 23,700 statements processed.
  * 98+% C1 coverage attained.
  * Three stages of software evaluation: Unit testing level, subsystem testing level, and system/acceptance testing level.

- **RESULTS**

  * 3.57% NCSS unit testing errors
  * 0.26% NCSS subsystem testing errors
  * 0.08% NCSS system testing errors
  * 3.91% NCSS overall deficiency discovery rate

REFERENCE: P. C. Belford, R. A. Berg, and T. L. Hannan, "Central Flow Control Software Development: A Case Study of the Effectiveness of Software Engineering Techniques," *Proc. 1979 Int'l. Conference on Software Engineering*, September 1979.

DYNAMIC TESTING — C1 BASED APPROACH

- **GOAL: Thorough Exercise of Program(s)**

  NOTE: C1 is defined as the percentage of logical segments in a program that are exercised by any one test. The *normal* goal is to achieve an aggregate value of 100% C1 over a series of tests. 85% C1 is sometimes acceptable in practice for various technical reasons.

- **MECHANISM: Integrated, Automated Testbed (Test Harness) to Support Major Bookkeeping Functions Needed by Software Test Engineer**

  NOTE: Typical systems have been built to include the functions listed below:

  * Automatic C1 coverage analysis
  * Assistance in setting input values and evaluating output values
  * Centralized statistics gathering for multiple tests
  * Some form of results comparison (automated)

- **SOME GUIDELINE FACTS**

  * Number of Segments is approximately 25% of KLOC.
  * No more than one test per Segment is normally required, with typically 2 - 8 Segments "retired" per test.
  * 85% C1 level is relatively easy to achieve, 100% C1 may require some "exceptions".
  * Most "off-the-assembly-line" programs achieve between 25 - 50% C1 coverage.

REFERENCE: E. F. Miller and W. E. Howden, "Software Testing and Validation Methods," IEEE Computer Society, September 1978.

SOFTWARE RESEARCH

SUMMARY OF OPERATION OF TESTING FACTORY

Quantity	Total/Average
Modules	128
Statements	60881
Segments	4378
Test Cases Used	1544
Coverage Attained	89.7%
Code Violations	1296
Program Errors	190
Total Discrepancy Reports	1486
Statements/Error	40.96
Error Rate (/Statement)	2.44%

S-TCAT/C -- EXAMPLE COVERAGE ANALYSIS

SYSTEM AND INTERFACE ANALYSIS OF A "C" SOFTWARE SYSTEM

METRIC USED: S1 (% OF POSSIBLE CALL-PAIRS EXERCISED)

STATISTICS ON "C" SYSTEM TESTED:

- 15 "C" MODULES
- 65 CALL-PAIRS
- AVERAGE OF 4.33 CALL-PAIRS/MODULE
- APPROXIMATELY 1500 LINES OF "C" CODE

STATISTICS ON THE SET OF TESTS:

36 SEPARATE TESTS

TEST EFFICIENCY: 1.81 CALL-PAIRS/TEST

LEAST S1 COVERAGE OBTAINED IN ONE TEST:	6.00%
MOST S1 COVERAGE OBTAINED IN ONE TEST:	67.69%

INITIAL S1 VALUE:	41.67%
FINAL S1 VALUE:	86.15%

LEAST INVOKED MODULES:	1	TIME	(2 MODULES)
MOST INVOKED MODULES:	1128	TIMES	(2 MODULES)

AVERAGE NUMBER OF INVOKES/TEST: 267.8

```
List of blocked function names.  When a name appears in this file
S-TCAT/C does NOT instrument for this name.

abort
abs
assert
atof     atoi     atol
toupper tolower _toupper        _tolower        toascii
ctime   localtime       gmtime  asctime
isalpha isupper islower isdigit isxdigit        isalnum isspace
        ispunct isprint isgraph iscntrl isascii
cuserid
ecvt fcvt gcvt
exit
exp      log      powe    sqrt
fclose   fflush
feof     ferror   clearerr        fileno
floor    ceil     fmod    fabs
fopen    freopen fdopen
fread    fwrite
frexp    ldexp    modf
fseek    ftell    rewind
getc     getchar fgetc    getw
getenv
getgrent          getgrnam        setgrent        endgrent
getlogin
getopt
getpwent          getpwuid        getpwnam        setpwent        endpwent
gets     fgets
l3tol    ltol3
logname
malloc   realloc calloc
mktemp
monitor
nlist
perror
printf   fprintf sprintf
putc     fputc    putw
```

SOFTWARE RESEARCH

S-TCAT ANALYSIS OF TESTS   QAT-53-2

RESULTS AFTER TEST NO. 1

```
S-TCAT Coverage Analyzer.  SCOVER Version 1.85 (80 Column)
(c) Copyright 1985 by Software Research Associates

+--------------------------------------------------------------------------------+
I                            I      This Test          I   Cumulative Summary    I
+                            +-------------------------+-------------------------+
I                            I          No. Of         I          No. Of         I
I Module            Number Of I No. Of   Functions S1% I No. Of   Functions S1%  I
I Name:             Fn Calls: I Invokes  Hit     Cover I Invokes  Hit     Cover  I
+--------------------------------------------------------------------------------+
I main                     26 I      1      9    34.62 I      1      9    34.62  I
I my_fopen                  0 I      2      0   100.00 I      2      0   100.00  I
I buildtbl                 11 I      1      6    54.55 I      1      6    54.55  I
I readcom                   0 I      1      0   100.00 I      1      0   100.00  I
I enterdata                 5 I     71      2    40.00 I     71      2    40.00  I
I gendata                   5 I     11      2    40.00 I     11      2    40.00  I
I lookup                    1 I     11      1   100.00 I     11      1   100.00  I
I printnum                  0 I     11      0   100.00 I     11      0   100.00  I
+--------------------------------------------------------------------------------+
I Totals                   48 I    109     20    41.67 I    109     20    41.67  I
+--------------------------------------------------------------------------------+

Current test message (saved in archive):

t
```

S-TCAT ANALYSIS OF TESTS QAT-53-6

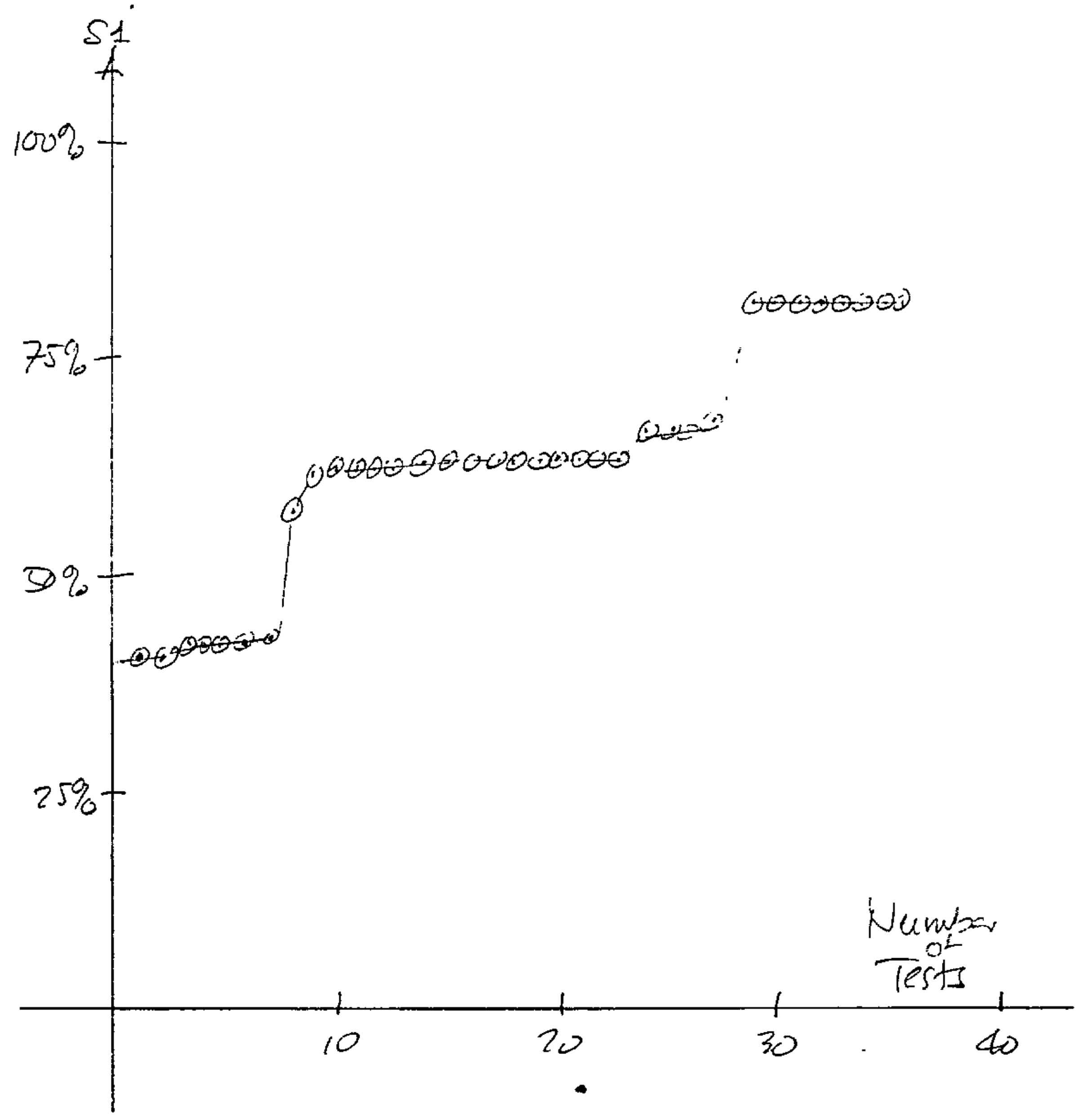
RESULTS AFTER TEST NO. 36

```
S-TCAT Coverage Analyzer.  SCOVER Version 1.85 (80 Column)
(c) Copyright 1985 by Software Research Associates

+---------------------------------------------------------------------------+
I                            I       This Test          I  Cumulative Summary I
+                            +--------------------------+---------------------+
I                            I          No. Of          I          No. Of      I
I Module           Number Of I No. Of   Functions S1%  I No. Of   Functions S1% I
I Name:            Fn Calls: I Invokes  Hit      Cover I Invokes  Hit     Cover I
+---------------------------------------------------------------------------+
I main                  26 I     1      13     50.00 I    36     19     73.08 I
I my_foren               0 I     4       0    100.00 I    76      0    100.00 I
I buildtbl              11 I     1      10     90.91 I    35     11    100.00 I
I readcom                0 I     2       0    100.00 I    93      0    100.00 I
I enterdata              5 I    31       5    100.00 I  1005      5    100.00 I
I gendata                5 I    11       4     80.00 I   190      5    100.00 I
I lookup                 1 I     9       1    100.00 I   168      1    100.00 I
I printnum               0 I     9       0    100.00 I   165      0    100.00 I
I range                  0 I     1       0    100.00 I    34      0    100.00 I
I itoa                   1 I    11       1    100.00 I  1139      1    100.00 I
I reverse                1 I    11       1    100.00 I  1139      1    100.00 I
I genrand                4 I     0       0      0.00 I    22      4    100.00 I
I touchfile              2 I     1       0      0.00 I     5      0      0.00 I
I update                 6 I     0       0      0.00 I     1      6    100.00 I
I filecopy               3 I     0       0      0.00 I     1      3    100.00 I
+---------------------------------------------------------------------------+
I Totals                65 I    92      35     53.85 I  4109     56     86.15 I
+---------------------------------------------------------------------------+

Current test message (saved in archive):

t
```

SOFTWARE RESEARCH

S-TCAT ANALYSIS OF TESTS QAT-53-7

GRAPH OF OVERALL S1 COVERAGE RESULTS



EXAMPLE COMPUTATION

ORIGINAL SITUATION

32 TESTS FOR 15 MODULES OF "C"

TOTAL LENGTH APPROXIMATELY 1500 LINES

CUMULATIVE C1 = 91.57% FROM 32 TESTS

AUTOMATED TEST RE-RUN CAPABILITY EXISTS

INDIVIDUAL TEST COVERAGE RANGE:

LOW:	C1 = 2.39%
HIGH:	C1 = 69.08%

TEST SELECTION METHOD

CHOOSE HIGHEST C1 TEST FIRST

COMPUTE ALL 2ND TEST C1 CONTRIBUTION

CHOOSE HIGHEST-CONTRIBUTION 2ND TEST

REPEAT FOR 3RD TEST, 4TH TEST, ETC.

IN CASE OF MULTIPLE CHOICES CHOOSE FIRST OCCURING INSTANCE (ARBITRARY ORDER)

CONTINUE UNTIL MAXIMUM COVERAGE IS ACHIEVED

DERIVED EFFICIENT TEST ORDER

OLD ORDER	NEW ORDER
1	TEST-29
2	TEST-20
3	TEST-27
4	TEST-32
5	TEST-6
6	TEST-13
7	TEST-9
8	TEST-12
9	TEST-28
10	TEST-3
11	TEST-4
12	TEST-10
13	TEST-14
14	TEST-15
15	TEST-16
16	TEST-17
17	TEST-18
18	TEST-19
19	TEST-21
20	TEST-22
21	TEST-24
22	TEST-25

SOFTWARE RESEARCH

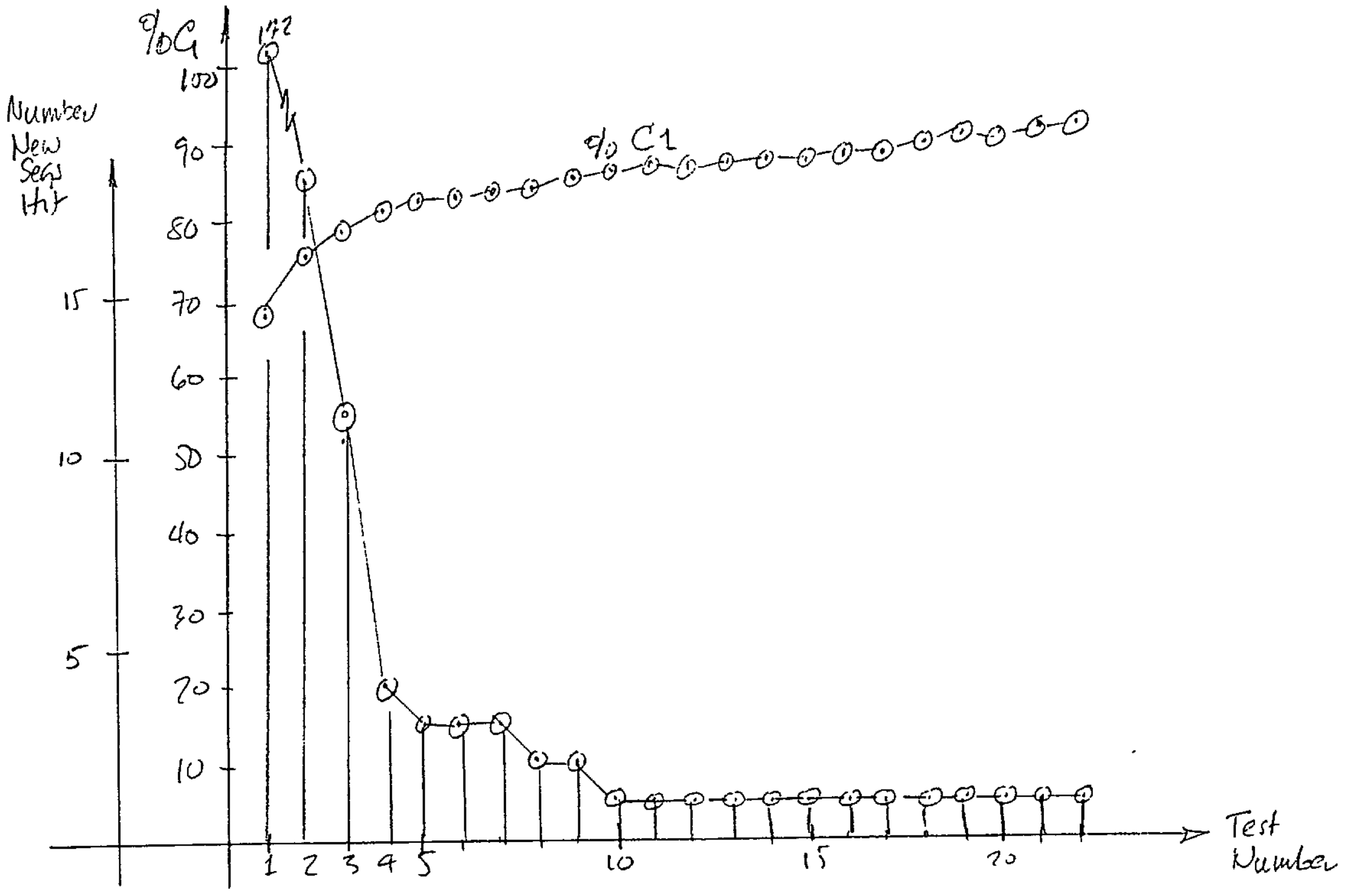
EFFICIENT ORGANIZATION OF TESTS

```
TCAT Coverage Analyzer.  COVER Version 1.8 (80 Column)
(c) Copyright 1984 by Software Research Associates

+------------------------------------------------------------------------------+
I                           I        This Test         I   Cumulative Summary  I
+                           +--------------------------+-----------------------+
I                           I            No. Of        I            No. Of     I
I Module         Number Of  I No. Of    Segments  C1%  I No. Of    Segments C1%I
I Name:          Segments:  I Invokes   Hit      Cover I Invokes   Hit    Cover I
+------------------------------------------------------------------------------+
I main                  52  I    1       27      51.92 I    1       27    51.92 I
I my_fopen               5  I    7        2      40.00 I    7        2    40.00 I
I buildtbl              55  I    1       47      85.45 I    1       47    85.45 I
I readcom                5  I    2        4      80.00 I    2        4    80.00 I
I enterdata             11  I   31        9      81.82 I   31        9    81.82 I
I gendata               42  I   11       30      71.43 I   11       30    71.43 I
I lookup                 5  I    9        4      80.00 I    9        4    80.00 I
I printnum               9  I    9        8      88.89 I    9        8    88.89 I
I range                 25  I    1       17      68.00 I    1       17    68.00 I
I itoa                   7  I   11        5      71.43 I   11        5    71.43 I
I reverse                3  I   11        3     100.00 I   11        3   100.00 I
I genrand                5  I    0        0       0.00 I    0        0     0.00 I
I touchfile              5  I    0        0       0.00 I    0        0     0.00 I
I update                17  I    1       13      76.47 I    1       13    76.47 I
I filecopy               3  I    1        3     100.00 I    1        3   100.00 I
+------------------------------------------------------------------------------+
I Totals               249  I   96      172      69.08 I   96      172    69.08 I
+------------------------------------------------------------------------------+

Current test message (saved in archive):

RuntimeVersion1.2. Last updated on 6-12-84
```

```
TCAT Coverage Analyzer.  COVER Version 1.8 (80 Column)
(c) Copyright 1984 by Software Research Associates

+--------------------------------------------------------------------------------+
I                            I       This Test          I   Cumulative Summary   I
+                            +-------------------------+------------------------+
I                            I          No. Of          I          No. Of        I
I Module           Number Of I No. Of   Segments   C1%  I No. Of   Segments   C1% I
I Name:            Segments: I Invokes  Hit       Cover I Invokes  Hit      Cover I
+--------------------------------------------------------------------------------+
I main                    52 I      1        3     5.77 I    22       45    86.54 I
I my_foren                 5 I      1        2    40.00 I    44        4    80.00 I
I buildtbl                55 I      1       37    67.27 I    21       53    96.36 I
I readcom                  5 I      5        4    80.00 I    61        5   100.00 I
I enterdata               11 I     14        9    81.82 I   628        9    81.82 I
I gendata                 42 I      0        0     0.00 I    80       42   100.00 I
I lookup                   5 I      0        0     0.00 I    70        5   100.00 I
I printnum                 9 I      0        0     0.00 I    68        9   100.00 I
I range                   25 I      2       18    72.00 I    19       25   100.00 I
I itoa                     7 I     70        7   100.00 I   618        7   100.00 I
I reverse                  3 I     70        3   100.00 I   618        3   100.00 I
I genrand                  5 I      0        0     0.00 I    18        5   100.00 I
I touchfile                5 I      0        0     0.00 I     0        0     0.00 I
I update                  17 I      0        0     0.00 I     2       13    76.47 I
I filecopy                 3 I      0        0     0.00 I     2        3   100.00 I
+--------------------------------------------------------------------------------+
I Totals                 249 I    164       83    33.33 I  2271      228    91.57 I
+--------------------------------------------------------------------------------+

Current test message (saved in archive):

RuntimeVersion1.2. Last updated on 6-12-84
```

EFFICIENT ORGANIZATION OF TESTS

QAT-52-9



SOFTWARE RESEARCH

COVERAGE CURVE OF REORDERED TESTS

TOTAL OF 249 SEGMENTS

TEST No.	INDIVIDUAL TESTS			CUMULATIVE RESULTS		
	INVOKES	HITS	C1%	INVOKES	HITS	C1%
1	96	172	69.08	96	172	69.08
2	200	134	53.82	296	188	75.50
3	74	93	37.35	370	199	79.92
4	83	171	68.67	453	203	81.53
5	56	93	37.35	509	206	82.73
6	33	96	38.55	542	209	83.94
7	2	6	2.41	544	211	84.74
8	83	84	33.73	627	213	85.54
9	454	130	52.21	1081	215	86.35
10	75	45	18.07	1156	216	86.75
11	42	84	33.73	1198	217	87.15
12	76	45	18.07	1274	218	87.55
13	80	83	33.33	1354	219	87.95
14	22	58	23.29	1376	220	88.35
15	347	86	34.54	1723	221	88.76
16	21	61	24.50	1744	222	89.16
17	164	83	33.33	1908	223	89.56
18	31	92	36.95	1939	224	89.96
19	26	70	28.11	1965	225	90.36
20	55	89	35.74	2020	226	90.76
21	87	142	57.03	2107	227	91.16
22	164	83	33.33	2271	228	91.57

GENERIC SYSTEM-LEVEL TESTING PLANS

Very often certain kinds of system characteristics can form the basis of a systematic approach to system test planning.

These categories represent some extremes that may be encountered.

- **VERTICAL SYSTEM**
  - Deep Interconnection Diagram
  - Strong Bottom-Up Dependence
  - Use Bottom-Up Testing
- **HORIZONTAL SYSTEM**
  - Flat Interconnection Structure
  - Primary Top-Down Dependence
  - Use Top-Down Testing
- **PARTIALLY IMPLEMENTED SYSTEMS**
  - Bottom-Up Implementation
  - Top-Down Implementation
  - Mixed Implementation

SOFTWARE RESEARCH

APPLIED SOFTWARE TESTING -- CASE STUDY DESCRIPTIONS

CASE STUDY FOCUS:

- TYPICAL SITUATIONS
- BEST INDUSTRIAL-STRENGTH METHODOLOGY
- AUTOMATION OF FUNCTION
- HIGH QUALITY

PRODUCTIVITY GAINS THROUGH:

- INCREASED RATE OF PRODUCTION
- LOWER COSTS TO DETECT DEFECTS
- BETTER AND LOWER-COST FUTURE TESTING

CLASSES OF CASE STUDIES

- TEST SUITE DEVELOPMENT
- COMPREHENSIVE PRODUCT TESTING
- DETAILED TECHNICAL TESTING
- VALIDATION TESTING

CASE STUDY ORGANIZATION

TEST SUITE DEVELOPMENT

- INTEPPRET FUNCTIONAL SPECIFICATIONS
- ACCOUNT FOR PASS/FAIL RATIOS
- AUTOMATIC APPLICATION

COMPREHENSIVE PRODUCT TESTING

- DESIGN FUNCTIONAL TESTS
- CHECK TEST MATPIX
- BUILD & APPLY TESTS
- REPORT DEFECTS

DETAILED TECHNICAL TESTING

- EXPLOIT PROPEPTIES OF PRODUCT
- DEFECT-PPONE MODULE IDENTIFICATION

VALIDATION TESTING

- INSPECTION
- FUNCTIONAL TESTING
- CONVERGENCE TESTING
- PEGRESSION TESTING

CASE STUDY A -- DEVELOP PROGRAMMING ENVIRONMENT TEST SUITE (REF #0890)

SITUATION:

- NEW PROGRAMMING ENVIPONMENT
- FORMAL SPECIFICATION EXISTS

REQUIREMENT:

- FULL-VALIDATION STYLE TEST SUITE
- LIMITED SUBSET OF FUNCTIONS
- KERNEL FUNCTIONS NEED MAIN ATTENTION

CONTEXT:

- NON-STANDARD HARDWARE
- NON-STANDARD LANGUAGE
- INTERNATIONAL CLIENT

SOFTWARE RESEARCH

METHODOLOGY USED

TEST PLANNING:

- FUNCTIONAL SPECIFICATION ANALYSIS
- 100% AUTOMATED TEST CONTROL PROGRAM

TEST PROPERTIES:

- SELF-CHECKING TEST FORMAT
- MANUAL VALIDATION TO SPECIFICATION

SPECIAL FEATURES:

- STANDARD PASS/FAIL REPORTING
- STANDARD ACCOUNTING

RESULTS ACHIEVED

PRODUCT:

- 157 TEST PROGRAMS
- 471 TESTS
- 175 COMMANDS, CALLS, DRIVES, FUNCTIONS TESTED
- AUTOMATED TEST EXECUTOR
- AUTOMATED RESULTS REPORTING

APPLICATION:

- APPROX. 24 DEFECTS DETECTED
- APPROX. 40 HRS TEST EXECUTION TIME
  - TESTS REQUIRE MANUAL VALIDATION
  - 8 HRS FOR RE-EXECUTION
- COST/DEFECT: $1K (ASSUMES NO VALUE FOR SYSTEM)

RE-APPLICATION

- MAJOR SYSTEM IMPROVEMENTS OBSERVED
- ONLY MINOR NEW DEFECTS FOUND

CASE STUDY B -- DEVELOP PL/I TEST SUITE (REF #1010)

SITUATION:

- NEW PL/I COMPILER
- MULTIPLE HARDWARE SYSTEMS

REQUIREMENT:

- PRE-RELEASE TESTING NEEDED
- FULL-VALIDATION NOT NEEDED

CONTEXT:

- ADVANCED FUNCTION PRODUCT
- PRIOR DEFECT-PRONE HISTORY
- SHORT SCHEDULE
- US CLIENT

METHODOLOGY USED

TEST PLANNING:

- CAREFUL STUDY OF LANGUAGE SPEC
- APPLY "TOUCH TEST" PRINCIPLE

TEST PROPERTIES:

- TEST MATRIX DEVELOPED
- SMARTS CONTROL

SPECIAL FEATURES:

- DEVELOPMENT VERSION OF COMPILER ONLY
- ERROR-PRONE ENVIRONMENT

RESULTS ACHIEVED

PRODUCT:

- 165 TEST PROGRAMS
- 2 AUXILIARY FILES
- 28 TEST SCRIPTS
- 11,167 LINES OF PL/I CODE
- 164 BASELINE FILES

APPLICATION:

- 31 DEFECTS DETECTED
- 6-8 HRS TEST EXECUTION TIME
- COST/DEFECT: $.5K (ASSUMES NO VALUE FOR PRODUCT)

SUBSEQUENT HISTORY:

- BASIS FOR COMMERCIAL PL/I PRODUCT

SOFTWARE RESEARCH

CASE STUDY C -- TEST ASSEMBLER PRODUCT (REF #0954)

SITUATION:

- MAJOR PRODUCT (COMPLETE MACRO ASSEMBLER)
- VERY-GREAT TECHNICAL SOPHISTICATION
  - MACRO PROCESSOR
  - MANY USER OPTIONS
- SOME INTERNAL TESTING COMPLETED

REQUIREMENT:

- 100% FUNCTIONAL COVERAGE IN TESTS
- AUTOMATIC OPERATION
- CONTINUAL DEFECT REPORTING

CONTEXT:

- PC/DOS ENVIRONMENT
- STANDARD CPU PART
- DOMESTIC CLIENT

METHODOLOGY USED

TEST PLANNING:

- BASED ON TECHNICAL MANUAL
- COMPREHENSIVE TEST MATRIX

TEST PROPERTIES:

- "FLAT" TEST ORGANIZATION
  - SIMPLE TESTS
  - MANY OF THEM
- MAXIMUM INDEPENDENCE OF TESTS

MECHANIZED CONTROL:

- SMARTS BASED REGRESSION
- MULTIPLE PC DEVELOPMENT
- AUTOMATED COMPARISON REQUIRED

SPECIAL FEATURES:

- MANUAL REVIEW COMPLETED
- SPECIAL PEGRESSION TECHNIQUES USED
- FULL TURNOVER OF TESTS TO CLIENT
- FULL TURNOVER OF BASELINE OUTPUTS TO CLIENT

RESULTS ACHIEVED

PRODUCT:

610 TESTS DEVELOPED

5,400 LINES OF SMARTS CONTROL FILE

APPLICATION:

160 DEFECTS FOUND

3-5 DAYS TEST EXECUTION TIME

COST/DEFECT: $.6K (ASSUMES NO VALUE FOR TEST SUITE AND REGRESSION SYSTEM)

RE-APPLICATION:

MAJOR QUALITY IMPROVEMENT

COMMERCIAL PRODUCT RELEASED

SOFTWARE RESEARCH

CASE STUDY D -- TEST HI-END PUBLISHING PRODUCT (REF #0988)

SITUATION:

- HI-END PUBLISHING PRODUCT
- SUN WORKSTATION
  - WINDOWS
  - MOUSE
  - KEYBOARD
- OBJECT-ORIENTED PROGRAMMING

REQUIREMENT:

- AUTOMATIC REGRESSION TOOL
- INITIAL TEST DESIGN
- INITIAL TEST DEVELOPMENT
- CREATION OF TEST BASELINE

CONTEXT:

- DOMESTIC CLIENT
- HI-PERFORMANCE ARCHITECTURE
- EXTREMELY SOPHISTICATED PRODUCT

METHODOLOGY USED -- TEST SUPPORT TOOL

TEST TOOL PLANNING:

- CAPBAK DESIGN BASE
- DESIGN OF INPUT CAPTURE
  - KEYBOARD
  - MOUSE
  - SUBSCREENS
- DESIGN OF REPLAY
- AUTOMATED COMPARISON

TOOL PROPERTIES:

- HIGHLY INTERACTIVE SYSTEM
- SMARTS INTEGRATION

SPECIAL FEATURES:

- SCREENSAVE OF SUBWINDOWS
- AUTOMATED COMPARISON OF WINDOWS

METHODOLOGY USED -- TEST DEVELOPMENT

TEST PLANNING:

- BASED ON EXISTING MANUAL TESTS
- REORGANIZED FOR MECHANIZED OPERATION

TEST PROPERTIES:

- 210 TESTS BUILT
- 650-950 SUBTESTS CONSIDERED
  - DEPENDS ON DEFINITION OF SUB-TEST
  - TYPICALLY, "SUB-TEST" = SUB-WINDOW

SPECIAL FEATURES:

- FEEDBACK INTO DEFECT TRACKING
- CONNECTION TO TRAINING DEPARTMENT

SOFTWARE RESEARCH

RESULTS ACHIEVED

PRODUCT:

TEST SYSTEM DOES 100% AUTOMATIC TEST REGRESSION

APPLICATION:

22 DEFECTS DETECTED

3-4 DAYS TEST EXECUTION TIME

MANUAL ASSISTANCE AND VALIDATION

SYSTEM IS INTERACTIVE

COST/DEFECT: $2.2K (ASSUMES NO VALUE ON TEST SUITE OR REGRESSION SYSTEM)

CASE STUDY E -- TEST UNIX OPERATING SYSTEM (REF #0877)

SITUATION:

- NEW HARDWARE RELEASE
- PROPRIETARY CPU
- SOFTWARE PORT PLUS SPECIAL FEATURES

REQUIREMENT:

- VALIDATION OF KERNEL
  - FUNCTIONALITY
  - PERFORMANCE
- VALIDATION OF SYSTEM PROPERTIES
- IDENTIFICATION OF USER PROBLEMS

CONTEXT:

- DOMESTIC CLIENT
- HI-TECHNOLOGY HARDWARE

METHODOLOGY USED

TEST PLANNING:

- USER DOCUMENTATION AS TEST PLAN BASE
- SPECIAL COMPATIBILITY TESTS
  - ANOTHER XENIX AS BASE
  - MANUAL VALIDATIONS

TEST PROPERTIES:

- 66 TESTS (182 SUBTESTS) OF KERNEL INTERFACE (USVS)
- 87 TESTS (150 SUBTESTS) OF LIBRARY FUNCTIONS (USVL)
- 141 TESTS (TESTING 665 SWITCHES OF 195 COMMANDS) OF UTILITY FUNCTIONS (USVU)

SPECIAL FEATURES:

- PART OF 'STANDARD TEST SUITES' FOR UNIX
- COMMERCIAL OFFERING

RESULTS ACHIEVED

PRODUCT:

- ALL TESTS RUN
- 2 CONFIGURATIONS

APPLICATION:

- 31 DEFECTS
- COST/DEFECT: $1K (ASSUMES NO VALUE FOR SUITE)

CASE STUDY F -- TEST XENIX OPERATING SYSTEM (REF #0795B)

SITUATION:

- NEW SOFTWARE RELEASE IN EARLY STAGES OF DEVELOPMENT
- VARIOUS HARDWARE CONFIGURATIONS

REQUIREMENT:

- FUNCTIONALITY VERIFICATION
- TESTING OF DRIVERS
- MULTIPLE REGRESSIONS (ON 5 RELEASES)

CONTEXT:

- DOMESTIC CLIENT
- MID-TECHNOLOGY HARDWARE

METHODOLOGY USED

TEST PLANNING:

REUSE OF EXISTING TEST SUITES

SOME NEW TESTS NEEDED

TEST PROPERTIES:

107	TOUCH TESTS OF 177 BASE COMMANDS WITH 449 SWITCHES
51	TOUCH TESTS OF 56 SOFTWARE DEVELOPMENT SYSTEM COMMANDS WITH 288 SWITCHES
20	TOUCH TESTS OF 31 TEXT PROCESSING COMMANDS WITH 107 SWITCHES
1	FULL TEST OF SYSTEM INITIALIZATION CODE
53	FULL TESTS OF CRT
10	FULL TESTS OF KEYBOARD
100	FULL TESTS OF TTY
36	FULL TESTS OF FLOPPY DISK
20	FULL TESTS OF HARD DISK
126	FULL TESTS OF SERIAL PORTS
12	FULL (ADDITIONAL) TESTS OF MULTI-PORT BOARD
6	FULL TESTS OF PARALLEL PORTS
25	FULL TESTS OF CO-PROCESSOR
6	FULL TESTS OF TIMER
7	FULL TESTS OF CLOCK
17	FULL TESTS OF MMU

SOFTWARE RESEARCH

RESULTS ACHIEVED

PRODUCT:

- TESTS RUN ULTIMATLEY ON 5 RELEASES
- MANY MACHINE CONFIGURATIONS
  - DIFFERENT MACHINES
  - SINGLE-USER
  - MULTI-USER
  - LINKED TOGETHER

APPLICATION:

- 95 ERRORS AND 3 INCIDENTS REPORTED
- 40+ HRS TEST EXECUTION TIME
- COST/DEFECT: $1K (ASSUMES NO VALUE ON SUITE)

CASE STUDY G -- TEST PATIENT ORIENTED MEDICAL PRODUCT (REF #0813)

SITUATION:

- MEDICAL PRODUCT, USED BY PATIENT
    - MEASURES BLOOD SUGAR
    - RECORDS INFORMATION OVER TIME
    - REPORTS TO CENTRAL COMPUTER

REQUIREMENT:

- FUNCTIONAL TESTING
- COVERAGE ANALYSIS
- REGRESSION TESTS

CONTEXT:

- DOMESTIC/INTERNATIONAL CLIENT
- HIGH CRITICALITY (MEDICAL PRODUCT)

METHODOLOGY USED

TEST PLANNING:

TESTS BASED ON FUNCTIONAL SPECS

TESTS ORGANIZED THROUGH TEST MATRIX

TEST PROPERTIES:

39 TESTS DEVELOPED: FUNCTIONAL + STRESS

11 CONVERGENCE TESTS

2 SYSTEM CONVERGENCE TESTS

52 TOTAL

C1 = 87% ACHIEVED

SPECIAL FEATURES:

CAPBAK USED TO RECORD ALL TESTS

MANUAL INITIATION OF REPLAY WITH CAPBAK

SOFTWARE INCIDENT REPORT SYSTEM INSTITUTED

RESULTS_ACHIEVED

PRODUCT:

- 52 TESTS BUILT, APPLIED
- TESTS PRESERVED ON KEYSAVE FILES FOR REGRESSIONS AND FOR FUTURE (FDA?) ENQUIRIES
- CODE AND TESTS UNDER CONFIGURATION CONTROL
- INTERMEDIATE COVERAGE REPORT FILES CONSERVED FOR FUTURE ENQUIRIES

APPLICATION:

- 12 DEFECTS DETECTED
- 40+ HRS TEST EXECUTION TIME
- COST/DEFECT: $2K (ASSUMES NO COST FOR TEST SUITE, KEYSAVE FILES, CONFIGURATION SYSTEM)

REGRESSION ON NEW VERSION:

- 2 DEFECTS NOTED
- 40+ HRS TEST EXECUTION TIME

SOFTWARE RESEARCH

CASE STUDY H -- TEST A MEDICAL PRODUCT (REF #1020)

SITUATION:

HARDWARE/SOFTWARE PRODUCT

USED FOR QUALITY ANALYSIS OF MEDICAL PRODUCT

PRODUCES ANALYTIC REPORTS

REQUIREMENT:

FULL VALIDATION TESTING

COVERAGE LEVELS SPECIFIED

VALIDATION SYSTEM REQUIRED

REGRESSION SYSTEM REQUIRED

CONTEXT:

HIGH CRITICALITY

MODERATE SIZE PRODUCT

METHODOLOGY USED

- INSPECTION:
  - UNIT-LEVEL INSPECTION
  - SYSTEM-LEVEL INSPECTION
- TEST PLANNING:
  - FUNCTIONAL TEST PLANNING
    - SPECIFICATIONS
    - IN PART FROM CODE
  - ALL TESTS UNDER SMARTS CONTROL
- FUNCTIONAL TESTING:
  - INITIAL MANUAL TEST VALIDATION
  - AUTOMATED DIFFERENCING
- CONVERGENCE TESTING:
  - $C1 > 95\%$ REQUIRED
  - $S1 > 99\%$ REQUIRED
- REGRESSION TESTING:
  - MULTIPLE REGRESSIONS
  - VALIDATION SYSTEM DEVELOPED

SOFTWARE RESEARCH

RESULTS ACHIEVED

PRODUCT:

- 23 FUNCTIONAL TESTS
- 15 FORMAT AND ERROR TESTS
- 13 OTHER FUNCTIONAL TESTS
- 14 COSMETIC TESTS
- 65 TOTAL
- VALIDATION SYSTEM
- 65 VALIDATED BASELINE FILES
- REGRESSION SYSTEM (SMARTS CONTROL FILE)
  - PRODUCT ITSELF REQUIRES MANUAL INTERVENTION

APPLICATION:

- INSPECTION STAGE
  - 71 MODULE ANOMALIES
  - 65 SYSTEM ANOMALIES
  - 52 ERs
- DYNAMIC TESTING
  - 11 FUNCTIONAL TESTING INCIDENTS
  - 11 CONVERGENCE TESTING INCIDENTS
- TOTAL: 74 DEFECTS DETECTED
- TEST EXECUTION TIME
  - 8-12 HRS WITHOUT PRINTING
  - 36-48 HRS WITH PRINTING
- COST/DEFECT: $.5K (ASSUMES NO VALUE FOR TEST SUITE REGRESSION SYSTEM, VALIDATION SYSTEM)

SOFTWARE RESEARCH

RELIABILITY ANALYSIS — DEFECT REMOVAL PROCESS

◦ GOAL

Model the current series of defect removal steps, predict remaining defects based on historical experience.

◦ DEFECT REMOVAL STEPS

(1) *Code Inspections/Reviews*: Standard IBM-like processing during software production

(2) *System Testing*: Pre-release examination by independent testing group

(3) *Field Testing*: Users find defect during (attempted) normal use of software

◦ MODEL USED

* Each step assumed to remove Pi fraction of total numbcr of original defects.
* Overall effectiveness is (1-P1)(1-P2)...(1-Pi).

◦ APPROXIMATE VALUES

NOTE: P1, P2, and P3 are "Company Confidential" and highly dependent on the methodologies and personnel used. Good guess is Pi = 90% in all cases. This implies overall residual defect rate approximately 0.01% NCSS.

NOTE: Original defect rate is in 2.0 - 4.0% NCSS range, with 3.0% the "best guess" value.

◦ UNANSWERED QUESTIONS

* Now many remaining defects?
* How do users feel about assisting in the defect removal process?

REFERENCE: H. Remus and S. Zilles, "Prediction and Management of Program Quality," *Proc. 1979 Int'l. Conference on Software Engineering*, Munich, West Germany, September 1979.

SOFTWARE RESEARCH

RELIABILITY ANALYSIS — NEXT ERROR DISCOVERY PREDICTION

- **GOAL**

  Define an error statistic, develop model of error behavior, compute probability of an error in the future.

- **BASIC METHODS**

  * Extension of hardware reliability assessment (application dependent)
  * Based on analysis of properties of software system itself (application dependent)

  NOTE: Software dependent model discussed later.

- **HARDWARE RELIABILITY MODELS**

  * *Constant Error Rate*: Does not match reality.
  * *Jelinski-Moranda Model*: Random error detection, error rate proportional to number of remaining faults.
  * *Schick-Wolverton Model*: Same as Jelinski-Moranda, except error rate proportional also to length of time testing (equally probable error discovery statistics in testing).
  * *Shooman Model*: Same as Jelinski-Moranda, but includes total debugging and total execution time.
  * *Schneidewind Model*: Errors assumed Poisson distributed, mean number of errors detected decreases exponentially with time (same level of testing competence), error rate proportional to remaining errors.
  * *MUSA Model*: Errors present and not found are function of total execution time of program (actual use time).

- **PROBLEMS**

  * Calibration Statistics (Model Validation)
  * Inaccuracies in modeling software as "hardware"

REFERENCE: J. C. Rault, "Quantitative Measures for Software Reliability," *Infotech State of the Art Report*, 1979.

TESTING OF EXPERT SYSTEMS

- PRINCIPLES OF EXPERT SYSTEMS
  - USER INPUT/OUTPUT INTERFACE
  - RULE SET
  - INFERENCE ENGINE
- QUALITY ISSUE IN EXPERT SYSTEMS
  - POSITIVE FAILURES
    - INCORRECT RULE
    - INCORRECT DEDUCTION
    - COMBINATION
  - NEGATIVE FAILURE
    - OMITTED RULE
    - MISSING INTERMEDIATE LEMMA
- OTHER DEFICIENCY SOURCES
  - DEFICIENCY IN INFERENCE ENGINE
  - HARDWARE DEFICIENCY
  - INSUFFICIENTLY CHECKED HUMAN INPUT
  - COMBINATION

EXPERT SYSTEM'S QUALITY ASSESSMENT

- FAILURE MODES
  - USER INPUT ERRORS
  - INCONSISTENT INPUT FACTS
  - ERRORS IN RULESET
  - INCORRECT REDUCTION(s)
  - INCORRECT VALIDATIONS
- IMPACT OF FAILURE
  - IMAGE
  - FIELD REPLACEMENT COST
  - DAMAGE COMPENSATION
  - LIABILITY
- DEGREE OF DIFFICULTY
  - RULESET PROGRAMMING
  - SYSTEM INTEGRATION
  - COMPONENT FAILURE MODES
  - USER TRAINING

USER INPUT ERRORS

FAILURE MODE:

USER TYPES INCORRECTLY

USER "FORGETS" CONTEXT

SIMPLE PILOT ERROR

EXAMPLE:

WRONG PATIENT NAME

WRONG MACHINE SITE

ESTS REMEDY:

INPUT CHECKING REQUIRES "SANITY" MODEL

CREATE ARCHIVE OF FULL-SESSION TESTS

VALIDATE TESTS INDEPENDENTLY

INCONSISTENT FACTS

FAILURE MODE:

- ATOMIC "FACTS" ARE NOT TRUE
- "COMBINATIONS OF FACTS" ARE NOT TRUE

EXAMPLE:

- ACTUAL FACTUAL DEFECT
  - INCORRECT FORMULA
  - INTERFACE ERROR

ESTS REMEDY:

- AT LEAST ONE TEST MUST USE EACH FACT
- 100% RULE COVERAGE (LR1)
- AUTOMATED REGRESSION ON EXAMPLE SESSIONS

ERROR IN RULESET

FAILURE MODE:

- MISSING, WRONG, EXTRA RULE
- INCOMPLETE CONSISTENCY CHECKING
- INCOMPLETE "EXPERT" UNDERSTANDING
- FIRING ORDER DEPENDENCE
- MISSING/INCORRECT DATA FLOW
- ESTIMATE: 20 DEFECTS/KRULE

EXAMPLE:

- THE "FLYING ZEBRA"

ESTS REMEDY:

- POSSIBLE FORMAL MANUAL INSPECTION
- FULL PATH ANALYSIS APPEARS NECESSARY (LRT)
- LR1 MAY BE AN EFFECT APPROXIMATION TO LRT

SOFTWARE RESEARCH

INCORRECT REDUCTION

FAILURE MODE:

- RULES ARE CORRECT: FAILURE IN REDUCTION
- CONVENTIONAL SW STATISTICS APPLY

EXAMPLE:

- MISSED RULE IN BACKTRACKING

ESTS REMEDY:

- USE COMMERCIAL STS METHODS
- ANALOGOUS TO COMPILER VALIDATION

INVALID OUTPUT

FAILURE MODE:

- INSUFFICIENT CHECKING OF RESULTS
- SYSTEMIC FAILURE

EXAMPLE:

- MISSED DIAGNOSIS
- INCORRECT CONFIGURATION

ESTS REMEDY:

- STRUCTURIZATION OF RULESET
- PARTITIONING OF TESTING WITHIN ESTS

SOFTWARE RESEARCH

COMMERCIAL SOFTWARE TEST SERVICES (STS)

- TURNKEY SERVICE STRUCTURE
  - CODE INSPECTION
  - SYSTEM INSPECTION (INTERFACES)
  - TEST PLANNING & FUNCTIONAL TESTS
  - C1 COMPLETION
  - S1 COMPLETION
  - REGRESSION
- COSTS
  - LOW RANGE
    - $\leq$ 20 KLOC
    - NORMAL QUALITY CODE
    - $10K-$25K/KLOC
    - 30-40 REPORTS/KLOC
  - HIGH RANGE
    - $>$ 40 KLOC
    - NORMAL TO LOW QUALITY CODE
    - $15K-$35K/KLOC
    - 20-30 REPORTS/KLOC

SOFTWARE RESEARCH

TEST ENVIRONMENT FOR EXPERT SYSTEM

- TEST SETUP
    - INDIVIDUAL TESTS
    - SCENARIOS
    - REPEATABLE
    - RANDOM INPUT
- TEST OUTPUT ANALYSIS
    - HUMAN
    - REGRESSION BASE
    - ALTERNATIVE RULE SET
- DEFICIENCY DETECTION MECHANISM
    - DEFICIENCY IN TESTS
    - DEFICIENCY IN RULESET
    - DEFICIENCY IN IMPLEMENTATION BASE
    - OTHER DEFICIENCY

QUALITY CONTROL ESTIMATES FOR EXPERT SYSTEMS

ORDINARY SYSTEMS

METRIC:

DEFECTS PER K LINES OF CODE

LIFE CYCLE ESTIMATES

DESIGN	5-20/KLOC	200-1
CODE	20-40/KLOC	
TEST	20-30/KLOC	20-1
MAINTAIN	10-35/KLOC	10-1
LIFE CYCLE	10-80/KLOC	15-1

EXPERT SYSTEMS

METRIC:

PERCENTAGE OF DEFECTIVE RULES

LIFE CYCLE ESTIMATES

PROTOTYPE	2-5%	400-1
FIRST RELEASE	1-2%	100-1
MAINTAIN	1-4%	20-1
LIFE CYCLE	1-8%	30-1

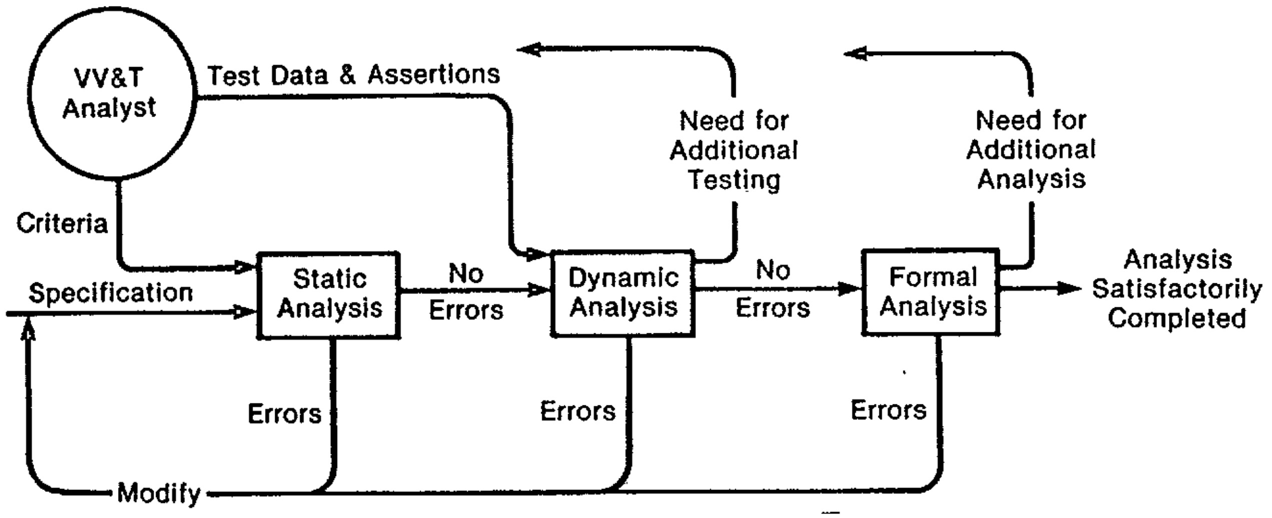
ADVANCED-CONCEPT MINIMUM ACCEPTANCE CRITERIA

GOAL: MINIMUM ACCEPTANCE CRITERIA FOR SOFTWARE AT THE UNIT LEVEL

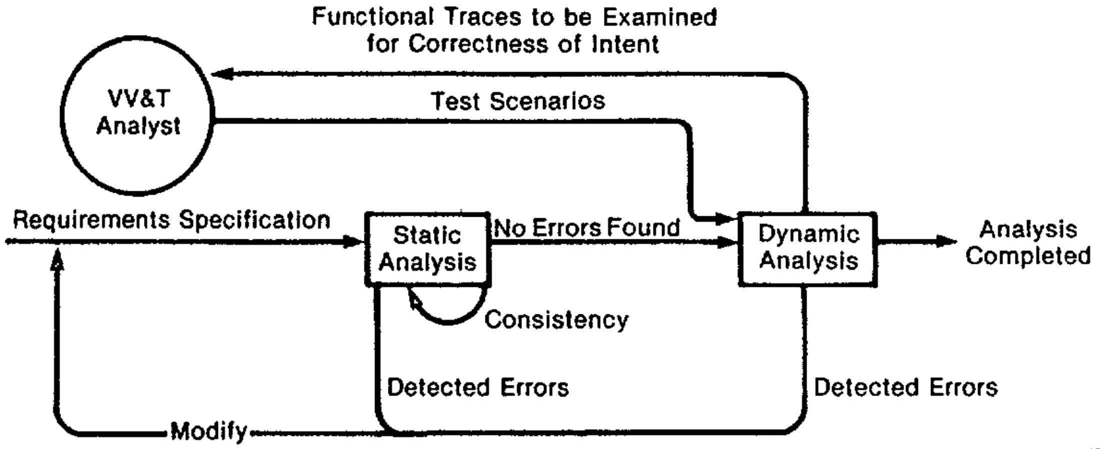
- "BEAUTIFIED" SOURCE PROGRAM LISTING, WITH IN-LINE COMMENTING
- OUTPUT FROM STATIC ANALYZER WITH EXPLANATIONS AND SUPERVISOR APPROVAL FOR ALL DISCREPANCY REPORTS
- OUTPUT FROM TEST EXECUTION VERIFICATION WITH MINIMUM TEST COVERAGE GOAL MET, OR EXPLANATIONS WITH SUPERVISOR APPROVAL FOR MISSED SEGMENTS
- OUTPUT FROM SOURCE CODE CONTROL SYSTEM SHOWING SUCCESSFUL INTEGRATION OF UNIT WITH "SYSTEM"

Source: Dr. Bud Wonsiewiez, CompSAC 82, November 1982.

FIPS PUB 101: INTEGRATED APPROACH TO VV&T



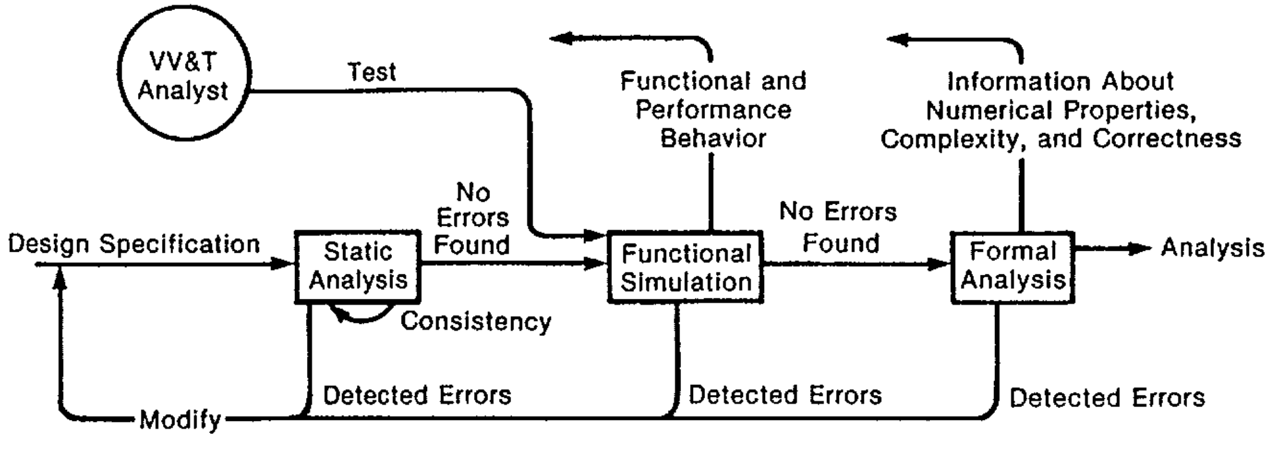
General VV&T integration strategy



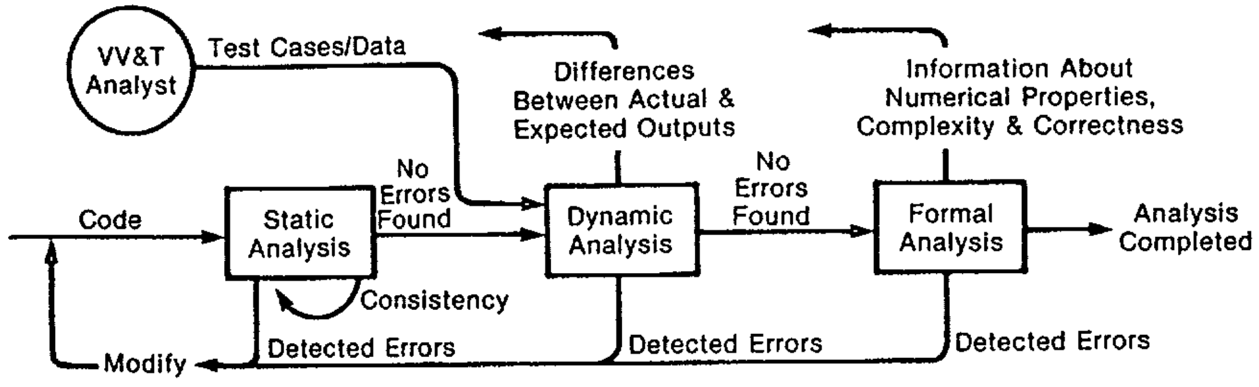
Integrated approach to requirements VV&T

SOFTWARE RESEARCH

FIPS PUB 101: INTEGRATED APPROACH TO VV&T



Integrated approach to design VV&T



Integrated approach to code VV&T

FIPS PUB 101: RECOMMENDED APPROACH (BASIC)

Phase	**Technique**
Requirements	Review
Design	Inspection
Code	Inspection Test Coverage Unit: 100% statement Integration: 100% module call System: 95% module call 100% of major logic paths
Installation	Acceptance Testing: Insure continued validity of system test
Operations and maintenance	For affected code: Inspection Test Coverage: 100% statement 100% module

Recommended techniques for lifecycle VV&T (basic approach)

FIPS PUB 101: RECOMMENDED APPROACH (COMPREHENSIVE)

Phase	**Technique**
Requirements	Inspection
Design	Interface Analysis Data Flow Analysis
Code	Assertions Standards Audit Interface Analysis Data Flow Analysis Explicit Trace-back of Code to Requirements
Installation	Acceptance Testing
Operations and maintenance	For affected code: Reapply techniques used during development

Recommended techniques for VV&T (comprehensive approach)

SOFTWARE RESEARCH

FIPS PUB 101: RECOMMENDED APPROACH (CRITICAL)

Phase	Technique
Requirements	Automated Consistency Analysis
Design	Automated Consistency Analysis Automated Simulation Proof of Critical Sections
Code	Symbolic Evaluation Proof of Critical Sections or Properties
Installation	Acceptance Testing: System Certification
Operations and maintenance	Re-do proofs that cover affected areas; retest

Recommended techniques for VV&T for critical software

SOFTWARE RESEARCH