A Reproduced Copy

PROCEEDINGS FROM THE FIFTH ANNUAL SOFTWARE ENGINEERING WORKSHOP

HELD ON

NOVEMBER 24, 1980

AT

GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

NASA
National Aeronautics and Space Administration

PROCEEDINGS

OF

FIFTH ANNUAL SOFTWARE ENGINEERING WORKSHOP

Organized by:

Software Engineering Laboratory
GSFC

November 24, 1980

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

FIFTH ANNUAL SOFTWARE ENGINEERING WORKSHOP

NASA/Goddard Space Flight Center
Building 3 Auditorium
November 24, 1980

8:20 a.m. INTRODUCTORY REMARKS — F. McGarry/Goddard Space Flight Center

8:30 a.m. PANEL NO. 1 — "THE SOFTWARE ENGINEERING LABORATORY"
DISCUSSANT: H. Hecht/SoHaR, Incorporated

- F. McGarry (NASA/GSFC) — "An Approach to Measuring Software Technology"
- J. Page (CSC) — "Impacts of Experiments and Software Technology Changes in a Production Environment"
- V. Basili (University of Maryland) — "Measuring the Effects of Specific Software Methodologies Within the SEL"

10:00 a.m. BREAK

10:15 a.m. PANEL NO. 2 — "SOFTWARE COST/RESOURCE MODELING"
DISCUSSANT: L. Putnam/Quantitative Software Management

- J. Golden/J. Mueller/B. Anselm (Xerox) — "Software Cost Estimating: Craft or Witchcraft"
- R. Tausworthe (JPL) — "Deep Space Network Software Cost Estimation Model"
- R. Lawler (Boeing) — "Software Quality Tradeoff Measurement"

12:15 p.m. LUNCH

1:15 p.m. PANEL NO. 3 — "SOFTWARE RELIABILITY"
DISCUSSANT: J. Musa/Bell Laboratories

- J. Logan (TRW) — "Application of a Reliability Model to Requirements Error Analysis"
- A. Goel/J. Soenjoto (Syracuse University) A. Sukert (RADC) — "Economically Optimum Computer Software Maintenance Strategies"
- M. Horn/W. Thompson (Columbia Research Corporation) — "A Comparison of Results Obtained From Established Software Reliability Measurement Models"

2:45 p.m. BREAK

3:00 p.m. PANEL NO. 4 "MEASURING THE DEVELOPMENT PROCESS"
DISCUSSANT: L. Belady/IBM

- S. Sheppard/E. Kruesi (General Electric) B. Curtis (ITT) — "Symbology and Spatial Arrangement: Effects on the Comprehension of Software Specifications"
- R. Cruickshank/J. Gaffney (IBM) — "Software Design Coupling and Strength Metrics"
- S. Moy (Logicon) — "A Tool for Software Design Evaluation"

4:30 p.m. ADJOURN

PANEL #1

THE SOFTWARE ENGINEERING LABORATORY

F. McGarry, NASA Goddard Space Flight Center
J. Page, Computer Sciences Corporation
V. Basili J. Bailey, University of Maryland

N82 24001

THE SOFTWARE ENGINEERING LABORATORY:
An Approach To Measuring Software Technology

F. McGarry
NASA/Goddard Space Flight Center

Over the past several years, we have seen the advent of newer and more disciplined approaches to the overall task of software development. Such approaches as structured design and development methodologies, improved management techniques, software metrics and measures, automated development tools, refined resource estimation and reliability models, as well as numerous other disciplines have been impacting the strategies of building, maintaining, and estimating the software process and product.

Although the software development community has been presented with these numerous software development methodologies, each claiming seemingly to be more effective than the other, it has not been clearly understood, at least in the NASA/Goddard environment, what effects these techniques may have on different phases of the software development process. We have not really understood if structured programming, automated tools, organizational changes, resource models, or any of the other technologies would have any impact (either favorable or unfavorable) on our own local software development process. It has also been quite apparent that it is not a trivial task to define what is a "better" software product. For these reasons, the Software Engineering Laboratory (SEL) was created at NASA/Goddard in conjunction with the Computer Sciences Department at The University of Maryland and with the Computer Sciences Corporation – the prime on/off-site support contractor for the subject environment at Goddard. The SEL set out to accomplish three basic tasks:

(1) Provide a means of measuring and understanding the software development process at NASA/Goddard.

(2) Measure the effects of available software development techniques as applied to applications programs at NASA.

(3) Define, develop, and/or refine those methodologies, models, and tools which will have a favorable impact on the software process and product at NASA.

The SEL was created in late 1976, and it spent most of the first full year defining factors and pertinent information that could reasonably and reliably be extracted from software development tasks. It also defined an experimental process by which varying technologies could be applied to different projects so that the methodologies, models, and tools could be measured and evaluated. The experiments involved the application of different sets of software methodologies to different applications projects within the NASA/GSFC environment.

In order to support the efforts of measuring effects of software techniques, it was necessary to define and implement an extensive monitoring process by which the details of all aspects of the software development process and product could be extracted for analysis. This involved the implementation of a software data collection mechanism by which the necessary detailed history of all aspects of a development project could be archived.

The data collection process has involved five sources of information for the more than 30 projects that have been closely monitored since 1977. These sources include:

Data Collection Forms – which are filled out by programmers, managers, technical support, etc.

Automated Computer Accounting Information – such as number of runs, CPU time, etc.

Automated Tools – such as code analyzers.

Subjective Management Data – such as level to which certain approaches were followed.

Personal Interviews – used as a follow up to the forms.

These sources have provided data on about 35 development projects and have resulted in a data base of historical data approaching 45 megabytes of information.

The projects that have been involved in the experiments of the SEL have ranged in size from 2,000 lines of source to about 120,000 lines of source with most of the projects falling in the 40,000 to 60,000 size range. All of the subject software has been developed to support flight dynamics requirements of various spacecraft missions supported by Goddard. None of the experimental data were extracted from synthetic projects.

Before committing resources toward the attempted measurement of the software process and software product, the obvious question that arises is, "Why should we attempt to perform software measurements?" There are three reasons for such an effort:

(1) To better understand the process in our particular environment.

(2) To provide some rationale for setting standards or guidelines within a particular software development community.

(3) To advance state-of-the-art technology in the overall software development process.

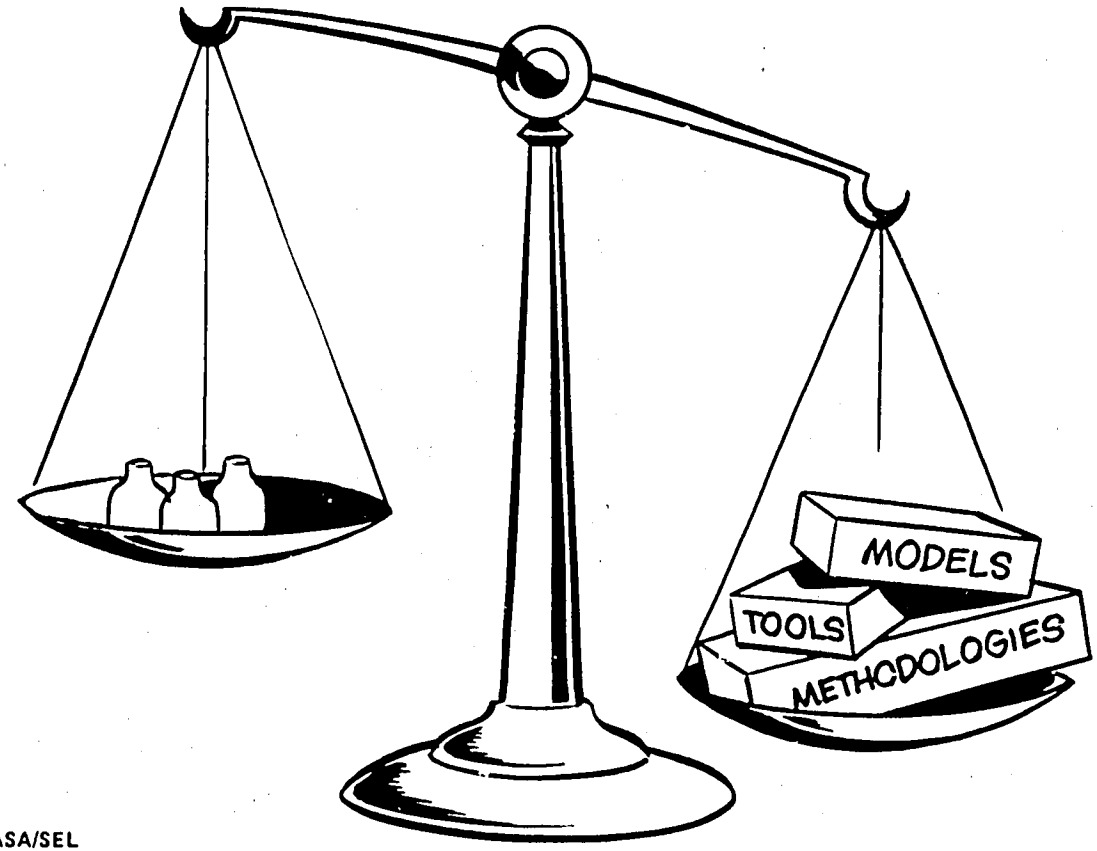
To support these goals of the software measurement process, there are some basic requirements that must be fulfilled in order to successfully arrive at any conclusions pertaining to the software being measured. First, we must attempt to understand approaches to existing software methodologies and technologies so that we may clearly define just what we are trying to measure; i.e., we must have access to some software technology base. Second, we must have access to a test environment so that the techniques of interest may be included in realistic experiments. We cannot assume that synthetic environments using synthetic projects would provide valid measures of any software technology being studied. The third requirement for supporting the measurement process is some training medium through which experimental subjects (whether they be managers or programmers) can be taught proper utilization of software techniques which are to be studied. The final requirement and certainly one of the most evasive and ill defined is a set of measures by which we can gauge both the software development process as well as the product.

An attempt has been made in the SEL to provide the essential components needed in the measurement of software technology. Through 4 years of extracting information from development projects, the SEL has learned that the measurement process is not only difficult, but it is expensive. In order to collect information, process the software data, and analyze them, an overhead as much as 25 percent has been encountered with the projects under study. Although this overhead may seem extreme, it has found that there are immediate and subtle benefits from the overall measurement process which greatly reduce the actual total cost of the experimentation.

Based on the SEL experiences, the importance of collecting software development data and measuring development methodologies cannot be overemphasized. Before we can ever expect to improve the process by which we develop software as well as the software product itself, we must understand our own environment as well as the possible effects that changing technologies may have on this environment.

THE VIEWGRAPH MATERIALS
for the
F. McGARRY PRESENTATION FOLLOW

AN APPROACH TO MEASURING SOFTWARE TECHNOLOGY



NASA/SEL

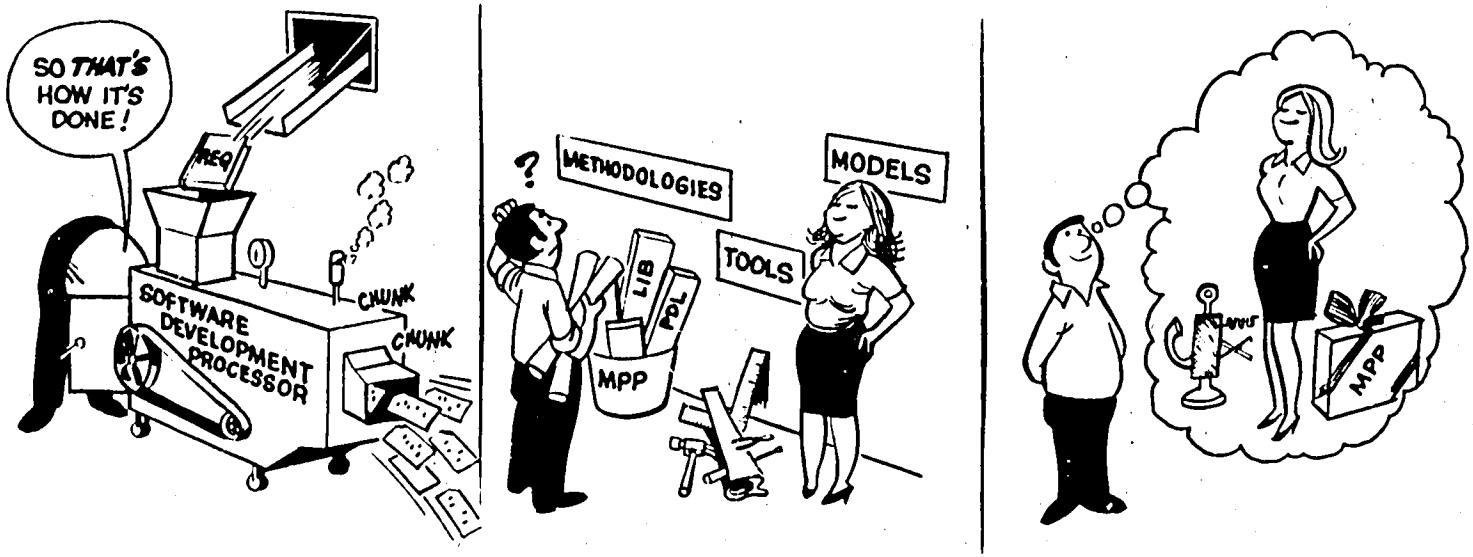
MEASURING SOFTWARE TECHNOLOGY

WHY?

1 TO UNDERSTAND

2 TO PROVIDE LOGICAL BASIS FOR SELECTION

3 TO ADVANCE TECHNOLOGY



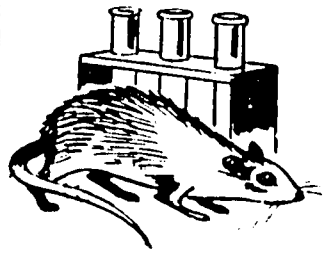
NASA/SEL

MEASURING SOFTWARE TECHNOLOGY

MAJOR REQUIREMENTS

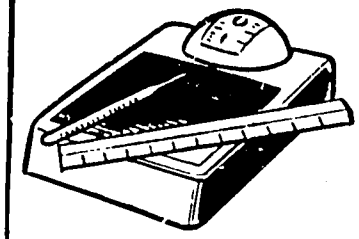
1 **TECHNOLOGY BASE**

2 **TEST ENVIRONMENT**



3 **TRAINING MEDIUM**

4 **MEASURES**



NASA/SEL

MEASURING SOFTWARE TECHNOLOGY

THE SEL APPROACH

- FORM TECHNOLOGY BASE (TEAM)
  - UNIVERSITY OF MARYLAND
  - GSFC
  - CSC

⇨ APPROACHES
IDEAS
MEASURES

- CREATE LABORATORY ENVIRONMENT
  - CSC & GSFC PROGRAMMERS
  - DEFINE CANDIDATE PROJECTS
  - PROVIDE DATA STORAGE/ANALYSIS MEDIUM

⇨ FORMS FOR DATA COLLECTION
STORAGE SYSTEM
PROCESSING PROCEDURES

- CALIBRATE DEVELOPMENT PROCESS
  - EXTRACT DEVELOPMENT DATA
  - IMPACT APPLICATIONS TASKS (SCREENING TASKS)

⇨
- PROFILES OF SOFTWARE
  - STRENGTHS/WEAKNESSES OF EXPERIMENTS
  - REFINEMENT TO DATA PROCESSING
  - BASIC EVALUATION OF MODELS, METRICS

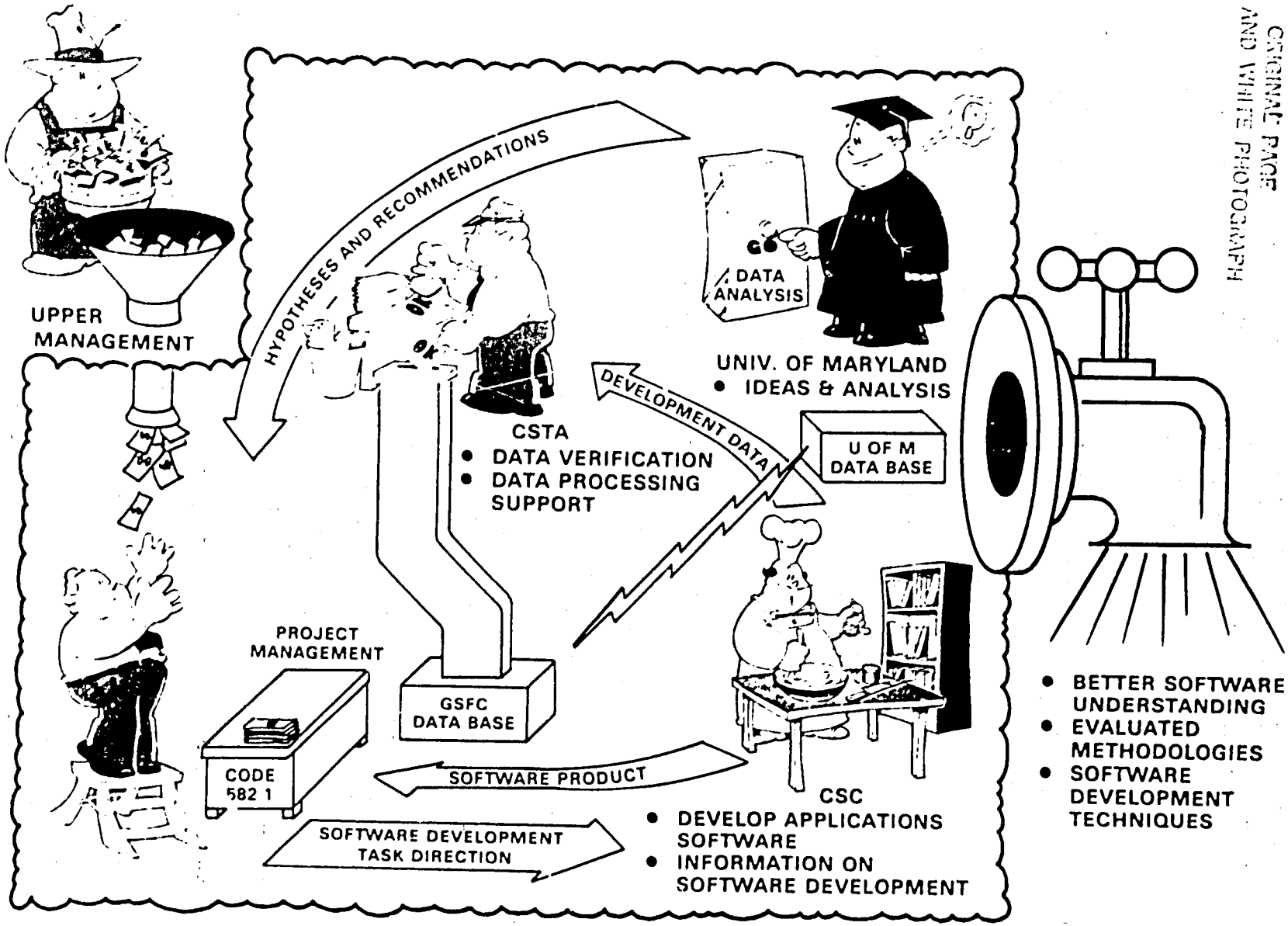
- PERTURB NEW TASKS
  - APPLY SOME TRAINING
  - EXTRACT ADDITIONAL DATA (SEMI-CONTROLLED TASKS)

⇨
- INITIAL MEASURES OF APPROACHES
  - MORE DETAILED PROFILES
  - MORE DETAILED EVALUATION OF MODELS, METRICS

- PERTURB NEW TASKS - REFINE APPROACHES
-

⇨
-
-
-

STRUCTURE OF THE SEL



F. McGarry
NASA/GSFC

MEASURING SOFTWARE IN THE SEL

BASIS FOR ANALYSIS

- LABORATORY EXPERIMENTS ..... 20-25 PROJECTS
- INFORMATION MONITORED ..... 1.3 MILLION L.O.C.
- PROGRAMMERS/MANAGERS REPRESENTED ..... 75 PEOPLE
- DATA EXTRACTED ..... 40 m BYTES ON DATA BASE (15,000 FORMS)
  - FORMS
  - TOOLS
  - SUBJECTIVE
- METHODOLOGIES APPLIED ..... 75 QUALIFYING PARAMETERS
  - VARIOUS MODELS,
  - TOOLS

MEASURING SOFTWARE TECHNOLOGY

SOME INDICATIONS FROM SEL EXPERIENCES

- COST
- GENERAL EXPERIENCES
- PAYOFFS

MEASURING SOFTWARE TECHNOLOGY

COST

(AS % OF TASKS BEING MEASURED)

	SEL EXPERIENCES	COULD BECOME
● OVERHEAD TO TASKS (EXPERIMENTS) ● FORMS ● MEETINGS ● TRAINING ● ●	5 - 10%	5%
● DATA PROCESSING ● COLLECTING/VALIDATING FORMS ● ORGANIZING DATA ● ENCODING INFORMATION ● ●	10 - 12%	6 - 8%
● ANALYSIS OF INFORMATION ● MEASURING METHODOLOGIES ● DESIGNING EXPERIMENTS ● DESIGNING ANALYSIS TOOLS ● DEFINING MEASURES ●	15 - 25%	10%
● SUPPORT SOFTWARE ● DATA BASE ● CODE ANALYZERS ● REPORT GENERATORS	2 MY- THEN 1 MY/YR	½ MY/Y

NASA/SEL

MEASURING SOFTWARE TECHNOLOGY

DESIRABLE SUPPORT INFORMATION/DATA

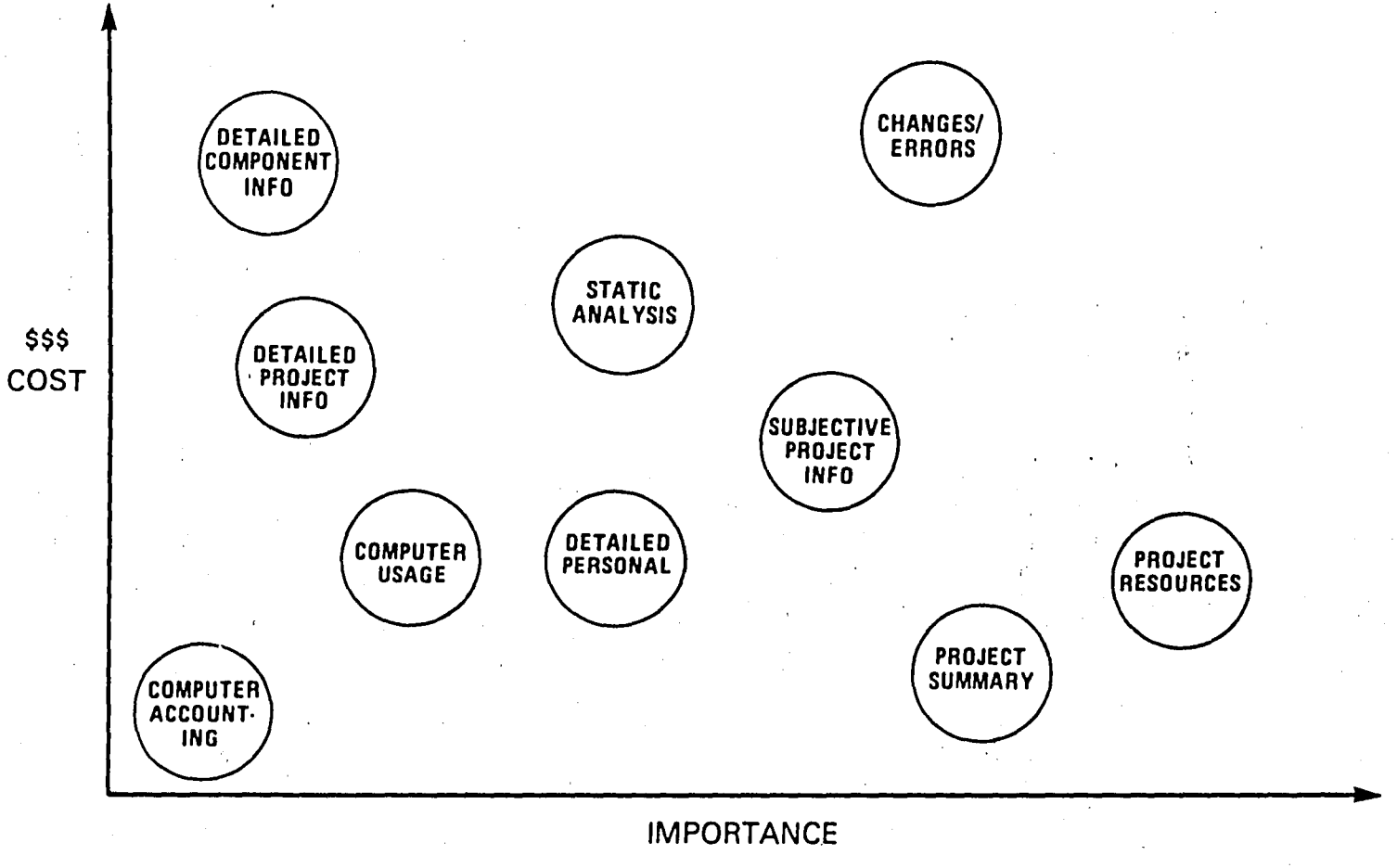
	SOURCE
● PROJECT RESOURCE	
PROJECT LEVEL	(M)
COMPONENT LEVEL	(P)
● STATIC ANALYSIS	(T)
● PERSONAL	(P)
● CHANGE/ERROR	(P)
● DETAILED COMPONENT	(P)
● SUBJECTIVE PROJECT INC.	(M)
● PROJECT SUMMARY DESCRIPTION	(M)
● COMPUTER ACCOUNTING	(C)
● COMPUTER USE	(P)

- PROGRAMMER (P)
- MANAGER (M)
- TOOL (T)
- COMPUTER (C)

F. McGarry
NASA GSFC

EXPERIENCES FROM THE SEL

INFORMATION FOR MEASURING



NASA/SEL

MEASURING SOFTWARE TECHNOLOGY

SOME THOUGHTS FROM THE SEL

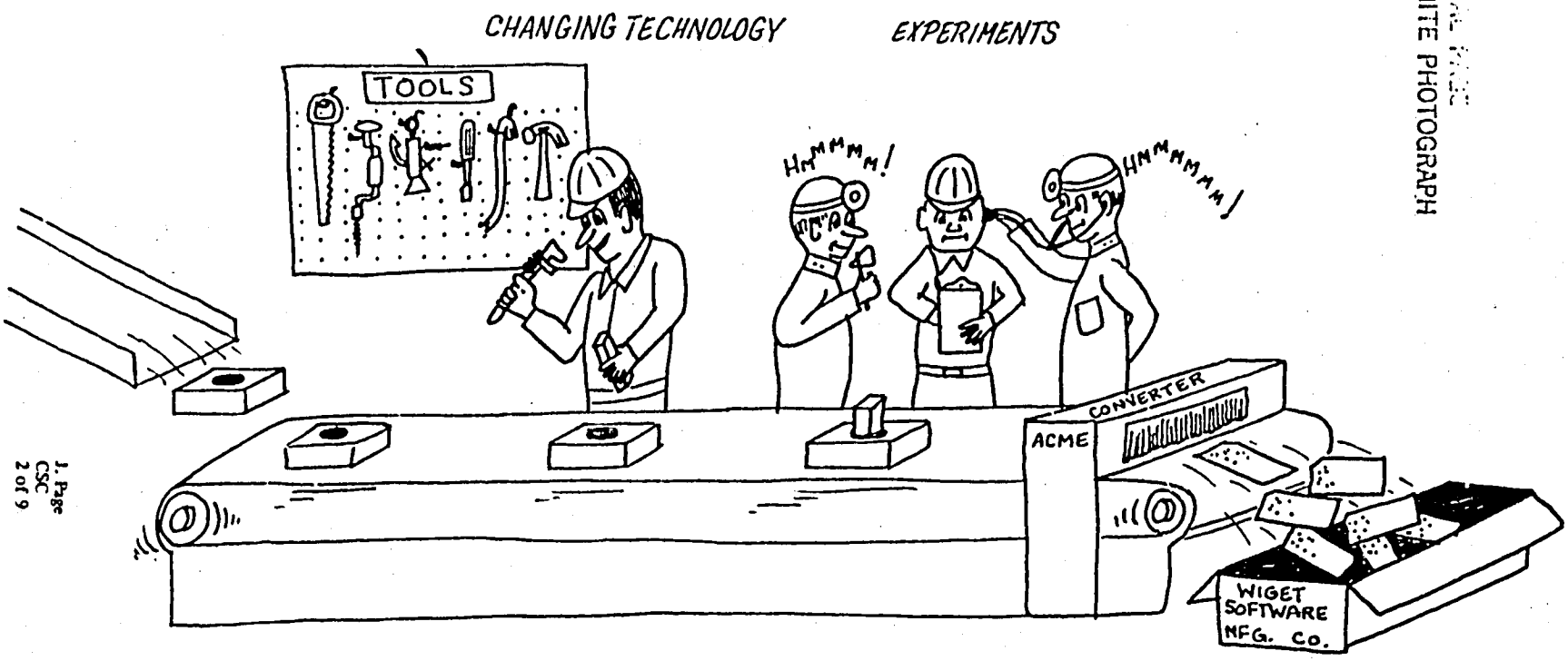
- AVOID DROWNING IN DETAIL . . . "CAN'T SEE THE FOREST . . . "
- TAKE TEST SUBJECTS INTO YOUR CONFIDENCE
- SUBJECTIVE MANAGEMENT INFORMATION — VERY, VERY IMPORTANT
- DON'T DRAW CONCLUSIONS FROM SINGLE PROJECT
- DON'T BE OVERLY CONCERNED ABOUT "PSYCHOLOGICAL" FACTORS
- REAL TIME FEEDBACK NEARLY IMPOSSIBLE, BUT . . .
- AVOID EXTENSIVE— "WE NEED MORE DATA"
  "WE NEED MORE EXPERIMENTS
  "WE HAVE TO VALIDATE THE INFORMATION"
- START WITH GENERALITIES - THEN REFINE

THE SOFTWARE ENGINEERING LABORATORY:
Impacts of Experiments and Software Technology Changes in a Production Environment

J. Page
Computer Sciences Corporation

THE VIEWGRAPH MATERIALS
for the
J. PAGE PRESENTATION FOLLOW

EFFECTS OF EXPERIMENTS & CHANGING TECHNOLOGY IN A PRODUCTION ENVIRONMENT



BLACK AND WHITE PHOTOGRAPH

J. Page
CSC
2 of 9

CHARACTERISTICS OF THE SOFTWARE ENVIRONMENT

TYPE SOFTWARE

- SCIENTIFIC, GROUND BASED
- FORTRAN (SOME ALC)
- AVERAGE 50K L.O.C.
- TYPICAL 18 MONTH DEVELOPMENT
- LARGE SCALE COMPUTER USAGE

DEVELOPMENT APPROACH

- NO FORMAL STANDARDS ENFORCED
- INTERNAL DEPARTMENT APPROACHES UTILIZED
- SOME BATCH - SOME INTERACTIVE
- NO STRICT CUSTOMER STANDARDS
- HEAVY INTERACTION WITH CUSTOMER
- NO HEAVY USAGE OF DEVELOPMENT TOOLS
- COST ESTIMATES BASED ON HISTORY + 'SMART' PEOPLE
- IN GENERAL - TOP DOWN BUILD APPROACH
- TEAMS OF 4-10 PEOPLE

ORGANIZATION

- SEPARATE SOFTWARE SUPPORT SECTION
- TEAMS FORMED WITHIN ONE SECTION
- OTHER DEPARTMENTS PROVIDE ANALYTICAL SUPPORT
- CLOSE WORKING RELATION WITH CUSTOMER (GSFC)
- OCCASIONALLY TEAMS INCLUDE GSFC PERSONNEL

DEVELOPMENT TEAM

- SCIENTIFIC - COLLEGE BACKGROUND
- 3-4 YEARS EXPERIENCE
- GENERALLY HAVE WORKED ON ONE SIMILAR PROJECT

EXPERIMENTS AND CHANGING TECHNOLOGY AFFECTING THE SOFTWARE TEAMS

- DETAILED 'DATA' REQUIRED (RECORD KEEPING) FORMS FORMS FORMS
- FORMAL TRAINING
- ALTER STANDARD DEVELOPMENT APPROACHES
- UTILIZATION OF NEW TOOLS REQUIRED
- NEW/ADDITIONAL TEAM INTERFACES
- ADDITIONAL CONTRACT (TASK) REQUIREMENTS

UTILIZING ADVANCED SOFTWARE DEVELOPMENT APPROACHES (CSC/SEL)
STEPS TAKEN

- CANDIDATE PROJECTS IDENTIFIED WITHIN CSC
- FORMAL TRAINING OF SPECIFIC METHODOLOGY (TIME CRITICAL)
- PLAUSIBLE SUBSET OF 'LEARNED' APPROACH SELECTED
- MANAGEMENT REINFORCEMENT APPLIED
- METHODOLOGY ADJUSTED TO SUIT THE PROBLEM
- APPROACHES SUBJECTIVELY EVALUATED
- RETRAIN – ADJUST – RETRY

IMPLEMENTING SOFTWARE EXPERIMENTS AT CSC
STEPS TAKEN

- DEFINE SCOPE OF INFORMATION REQUIRED
- CREATED SUPPORT TASKS (DATA PROCESSING, ANALYSIS, ...)
- IN HOUSE TRAINING TO CANDIDATE PROJECTS (FORMS, RATIONALE, PLANS, ...)
- REQUIRE DETAILED DATA BE PROVIDED FROM TASK
- REQUIRE USAGE OF SELECTED TOOLS
- CARRY OUT SCREENING EXPERIMENTS
- TRAINING
- MORE TASKS WITH APPROACHES ADJUSTED

EFFECTS OF THE CHANGING TECHNOLOGY IN THE CSC ENVIRONMENT

- PRODUCTIVITY ---------- A METHODOLOGY APPROACH + 5-10 %
  THE METHODOLOGY APPROACH + 10 %
  CORRECT PEOPLE STRUCTURE + 10-15 %

- RELIABILITY ---------- DEFINITELY DOESN'T HURT
  TO EARLY TO TELL QUANTITATIVE EFFECTS

- APPROACH TO SOFTWARE ---------- ADOPTING SELECT TECHNIQUES (AS STANDARD)
  USAGE OF RESOURCE MODEL APPROACHES AS SUPPORT
  LITTLE EFFECT FROM TOOLS

- GENERAL ---------- CSC CREATED SOFTWARE ENG. SECTION
  LEARNING CURVE SURPRISINGLY ABSENT
  PEOPLE BECOME DISCOURAGED WITHOUT REINFORCEMENT
  HAS KEPT CUSTOMER HAPPY (GSFC)
  SHOWN NEED FOR EDUCATIONAL REINFORCEMENT
  PROGRAMMERS RELUCTANT TO BLIND ACCEPTANCE

EFFECTS OF EXPERIMENTS

- PRODUCTIVITY ---------- ADDED DISCIPLINE TO PROCESS
  10 % OVERHEAD MINIMIZED BY SIDE BENEFITS

- RELIABILITY ---------- ?

- PSYCHOLOGICAL ---------- NO EVIDENCE OF HAWTHORN EFFECT
  INITIAL RELUCTANCE - THEN GRADUAL 'ACCEPTANCE'
  RARELY BOOSTS MORALE
  HAS CAUSED SOME DISCONTENT

- APPROACHES TO SOFTWARE ------ PROVIDED AN AWARENESS OF SOFTWARE TECHNOLOGY
  IDENTIFIED PLAUSIBLE METHODOLOGIES
  BASIS FOR GENERATING SOFTWARE GUIDELINES

EFFECTS OF EXPERIMENTS AND CHANGING TECHNOLOGY
SUMMARY

- EXPERIMENTS DO NOT HAVE TO BE OVERLY COSTLY
- EXPERIMENTS SHOULD BE USED TO SUPPORT METHODOLOGY SELECTION
- PROGRAMMERS DO NOT WANT BLIND ACCEPTANCE
- REINFORCEMENT - KEY TO ANY SUCCESS
- METHODOLOGIES - DEFINITE ADVANTAGES
  CANNOT IMPROVE THE PRODUCT
- MODELS - STILL VERY UNCERTAIN AS TO PARTICULAR
  SUPPORT FOR AN ENVIRONMENT
- TOOLS - GREAT POTENTIAL - BUT STILL MATURING -
  MINIMAL HELP TO DATE

THE SOFTWARE ENGINEERING LABORATORY:
Measuring the Effects of Software Methodologies Within the Software Engineering Laboratory

V. Basili and J. Bailey
University of Maryland

THE VIEWGRAPH MATERIALS
for the
V. BASILI PRESENTATION FOLLOW

WE HAVE EXAMINED A SET OF PROJECTS

- DEALING WITH GROUND SUPPORT SOFTWARE
- RANGING FROM 2K TO 101K DEVELOPED SOURCE LINES
- DURATION RANGING FROM 4.6 TO 17.4 MONTHS
- EFFORT RANGING FROM 5 TO 138 STAFF MONTHS
- AVERAGE STAFF SIZE FROM 1 TO 8 PEOPLE
- PRODUCTIVITY FROM 413 TO 1068 DEVELOPED SOURCE
  - LINES/STAFF MONTH WITH AN AVERAGE OF 668 DEVELOPED
  - SOURCE LINES/STAFF MONTH
- DATA COVERS DESIGN THROUGH ACCEPTANCE TEST
- INCLUDES MANAGER, PROGRAMMER AND SUPPORT STAFF

PROJECTS VARY WITH RESPECT TO THE SET OF SOFTWARE DEVELOPMENT TECHNIQUES USED AND THE EXTENT TO WHICH THEY WERE USED

- THERE WAS FORMAL TRAINING FOR SOME PROJECTS

EACH PROJECT WAS RATED WITH RESPECT TO

- A LARGE SET OF FACTORS
- COVERING ENVIRONMENT, METHODOLOGY, EXPERIENCE, PERFORMANCE, ETC.
- VALUES WERE GIVEN ON A SIX POINT SCALE
- RATINGS WERE SUBJECTIVE
- RELATIVE TO THE LOCAL ENVIRONMENT
- DONE NEAR END OF PROJECT WITHOUT KNOWLEDGE OF THE PRODUCTIVITY RESULTS
- BY NASA (McGARRY), CSC (PAGE) AND UNIVERSITY OF MARYLAND (BASILI)

RELATIONSHIP BETWEEN PRODUCTIVITY AND VARIOUS FACTORS

- NO SIGNIFICANT RELATIONSHIP BETWEEN PRODUCTIVITY AND SIZE

  (NO POINT IN CATEGORIZING BY SIZE)

- METHODOLOGY FACTORS

  FOR ALL THAT SHOWED A DIFFERENCE AMONG PROJECTS, THE CORRELATIONS BETWEEN METHODOLOGY AND PRODUCTIVITY:

PDL	0.26
FORMAL DESIGN REVIEW*	0.62
DESIGN FORMALISM	0.38
DESIGN DECISION NOTES*	0.62
DESIGN WALK-THROUGH	0.28
CODE WALK-THROUGH	0.19
CODE READING*	0.58
TOP DOWN DESIGN	-0.19
STRUCTURED CODE	0.02
LIBRARIAN USE**	0.52
CHIEF PROGRAMMER TEAM*	0.62
FORMAL TEST PLANS**	0.51
HEAVY MANAGEMENT INVOLVEMENT	-0.09
FORMAL TRAINING*	0.58
TOP DOWN CODE	0.29

*SIG. < 0.01
**SIG. < 0.05

OTHER FACTORS

- TRIED THE FOLLOWING:
  - CUSTOMER INTERFACE COMPLEXITY
  - CUSTOMER ORIGINATED PROGRAM DESIGN CHANGES
  - COMPLEXITY OF: APPLICATION PROCESSING, PROGRAM FLOW, INTERNAL COMMUNICATION, EXTERNAL COMMUNICATION, DATA BASE COMPLEXITY, JERRY'S GENERAL COMPLEXITY RATING
  - CONSTRAINTS: I/O CAPABILITY, TIMING, MAIN STORE
  - PROGRAMMING GROUP EXPERIENCE: MACHINE FAMILIARITY, LANGUAGE FAMILIARITY, APPLICATION EXPERIENCE, SAME TYPE BEFORE
  - HARDWARE CHANGES DURING DEVELOPMENT
  - PERCENT REAL TIME OR INTERACTIVE*
  - PERCENT PROGRAMMER INVOLVED IN SPECIFICATIONS
- ALL BUT ONE SHOWED NO SIGNIFICANT CORRELATION WITH PRODUCTIVITY

*SHOWED SIGNIFICANT DIFFERENCE AT 0.05 LEVEL IN WRONG DIRECTION (I. E., HIGHER % REAL TIME → HIGHER PRODUCTIVITY)

BASED UPON A SIMILAR STUDY BY DOUG BROOKS (IBM/FSD)

- TRIED TO SEE IF METHODOLOGY HAD A SIGNIFICANT EFFECT ON PRODUCTIVITY
- USED A STATISTICAL TEST TO SEE IF THE PROJECTS WITH HIGH METHODOLOGY USE CAME FROM A DIFFERENT ENVIRONMENT (WITH RESPECT TO PRODUCTIVITY) THAN THE PROJECTS WITH A LOW METHODOLOGY USE
- THE DATA USED WAS BASED UPON A RELATIVE RANKING RATHER THAN AN ABSOLUTE RATING
- THE APPROACH WAS
  - DIVIDE THE RATINGS FOR EACH TECHNIQUE INTO 3 CATEGORIES: LOW (-1), MEDIUM (0), HIGH (1) (DONE TO OFFSET DIFFERENCES IN SCALES)
  - ADD THE RATINGS TO GET A CUMULATIVE METHODOLOGY RATING
  - DIVIDE PROJECTS INTO GROUPS BASED UPON THEIR RATING AND ANALYZE USING THE MANN-WHITTNEY-U TEST (NONPARAMETRIC STATISTICS)

RESULTS

GROUP:	LOW	MEDIUM	HIGH
RATINGS:	(-11, -9, -9, -9)	(2, 2, 2, 1, 0, -1, -3, -3)	(12, 11, 8, 5, 5, 3)
PRODUCTIVITY:	535 DL/SM	660 DL/SM	768 DL/SM

RESULT:

LOW DIFFERENT FROM MEDIUM U HIGH (SIG. AT 0.05)

HIGH DIFFERENT FROM MEDIUM U LOW (SIG. AT 0.03)

GROUP:	LOW	HIGH
RATINGS:	(-11, -9, -9, -9, -3, -3, -1)	(0, 1, 2, 2, 2, 3, 5, 5, 8, 11, 12)
PRODUCTIVITY:	602 DL/SM	710 DL/SM

RESULT:

LOW DIFFERENT FROM HIGH (SIG. AT 0.05)

CONCLUSION

WE CAN SHOW THAT METHODOLOGY HAS A SIGNIFICANT POSITIVE EFFECT ON PRODUCTIVITY.

PANEL #2

SOFTWARE COST/RESOURCE MODELING

J. Golden/J. Mueller/B. Anselm, Xerox Corporation
R. Tausworthe, Jet Propulsion Laboratory
R. Lawler, Boeing Aerospace Company

D2

N82 24002

SOFTWARE COST/RESOURCE MODELING:

John R. Golden, James R. Mueller, Barbara Anselm
Xerox Corporation, Rochester, New York

The purpose of this paper is to briefly review the methodology and model developed by Larry Putnam[1] and present some results of their application at Xerox.

In The Mythical Man-Month,[2] Brooks stated that time and effort are not linearly related. Putnam's software costing model, based on Norden[3] Rayleigh product life cycle concepts, enables managers to define time and effort quantitatively. The tradeoff laws governing attempts to reduce development time by the adding of more people become highly non-linear, effort being proportional to the inverse fourth power of development time; and some development times are not possible at all.

The key attributes of Putnam's approach include the showing of trade-offs between time and effort, establishing a schedule (dynamic) for loading, establishing the feasible development range, a theoretical and empirical basis, an associated computational model, and the potential establishing of a baseline against which new technologies for software development can be measured.

Methodology Summary (Detailed discussion in references 1 and 6.)

Putnam used the ideas developed by Norden[4] describing the project life beginning with the observation that the rate of doing work (manpower) is equal to the work remaining times a linear function of time which Norden called the "pace" of the work (also called a linear "learning" curve by Parr[5]). The "pace" factor has dimensions of reciprocal time so in some sense corresponds to instantaneous natural frequencies of the project team. The equation states that the rate of accomplishment slows as the work remaining decreases and increases with time. That is, people tend to increase their "pace" more in response to the time remaining rather than the work remaining.

If y (t) is the work done at time t, K is the total work to be done, and t_d is the development time, the differential equation is

$$\dot{y} = \frac{dy}{dt} = 2at(K - y) \quad \text{where } a = 1/2t_d^2 \tag{1}$$

A particular solution of (1) is y = K, and the general solution of the reduced equation is $y = c\exp(-at^2)$ so that for y(0) = 0, we have

$$y = K(1 - e^{-at^2}) \tag{2}$$

or in differential form for the power curve,

$$\dot{y} = 2atK\, e^{-at^2} \tag{3}$$

A key property of Putnam's approach was the development of the software equation which relates the principal parameters of development time, total effort, system size and the development environment. Putnam linearized (3) by using logarithmic plots of $\dot{y}\,t$ versus t^2 and noted that systems with similar levels of software work or "difficulty" lay along lines of constant intercept, $K\,t_d^2$. Thus, he asserted that K/t_d^2 is a fundamental system parameter related to difficulty, that is, the software work to be done (not the size of the system).

It is reasonable to expect productivity to be the same only for systems of the same difficulty. To determine the relationship of productivity and difficulty, Putnam took data from hundreds of systems and plotted productivity, $P = S/K$ (where S is the number of source lines of code), versus difficulty, $D = K/t_d^2$, such that $P = cD^x$. He found x to be approximately -2/3. Solving for S, we have

$$S = cK^{1/3}t_d^{4/3} \quad (4)$$

Equation (4) is called the software equation and c is a constant related to the technology level of the systems development environment.

It is the software equation that indicates the non-linearity of Brook's law: Effort is proportional to the inverse fourth power of development time.

Finally, to examine the effect of time on difficulty, Putnam calculated the difficulty gradient which consists of directional derivatives of D with respect to t_d and K. The magnitude of this gradient is dominated by the term in the negative development time direction (t_d) and is approximately $2K/t_d^3$ which tells us that difficulty increases as time decreases. The trace of constant gradient, $K = bt_d^3$, where b is an empirical constant, when inserted into the software equation (4) yields the minimum development time:

$$S = cb^{1/3}t_d^{7/3} \quad (5)$$

The constants b (there are five in Putnam's current model corresponding to varying system complexity) determine the minimum development time for a given technology constant and system size by using (5). The corresponding K may then be calculated using (4).

Thus, the only way to effect the minimum possible development time given a system of a certain size and complexity is by improving the development environment with new methodologies and software tools and not by adding more people!

Results From Several Systems at Xerox

A preliminary examination of the applicability of Putnam's model to the Xerox environment has been conducted using Project Status Report data and results from interviews with the first line managers for four development projects (Figure 1). The estimates of time and effort predicted by the model for the minimum development were close to the actual figures for two of the four projects (A and B). The third project, C, exceeded the estimated minimum development time by 20% and required approximately half the effort which the model would predict as required for the minimum time. This shows the inverse fourth power law of time versus effort since $1.2^4 = 2.07$. The D project exceeded the minimum time by 15% to further illustrate the tradeoff.

This evidence of the time/effort tradeoff illustrates the potential savings attainable when development time is extended by a few months.

Figure 2 shows the actual manloading for the B project compared with that estimated by the Putnam model. These results, similar to those experienced in working with the other projects, show that the Norden-Rayleigh model basically describes the manloading pattern actually experienced by Xerox.

That allowing more time results in a lower cost may seem counterintuitive, for many managers see costs increasing with increasing time. But these cases are associated with overruns usually

demonstrating poor estimating originally, or the refusal to effectively manage the inevitable changes required during development. Thus, when a longer time is planned, the result can be significantly lower cost. Conversely, if too many people are applied too soon, the result will be higher cost with no positive effect on time.

If, however, time is of the essence, the model will show the resulting cost and warn of potential problems as the minimum time is approached. Perhaps the most important feature of the model is the estimate of minimum development time. This gives us a tool to demonstrate the risk of artifically imposed dates so prevalent in systems projects. Management can be appraised quantitatively of the inherent risk of compressing schedules.

The importance of breaking down a project into more manageable prices is clear, for not only will the system be cheaper, but early releases of those pieces will allow benefits to accrue earlier.

If a project gets into difficulty, do not try to add more people, but consider descoping. If this is not feasible, accept the slip and learn from it. The Putnam model tells us that the application of people to software development should follow the curve shown in Figure 2, or as "Brooks[2] second law" states: Adding more folks to a late project makes it later.

It is important to understand that this model is not a substitute for project management but rather is a key input, namely the software component. Physical factors such as hardware acquisition and human factors such as training must be accounted for externally.

One difficult and unresolved problem with the use of this or any similar model is system sizing, that is, extimating lines of code. It follows that a good requirements and system specification is needed. We believe the use of structured analysis and design will allow the appropriate correlations to be computed. For example, the number of process bubbles on a fully decomposed data flow diagram and the number of data elements may replace lines of code as key inputs.

SUMMARY: That a significant stride toward making software cost estimating and management "craft" rather than "witchcraft" is indicated in the following summary of our key points:

1. Time and effort are not linearly related; effort is proportional to the inverse fourth power of development time. Therefore, time is very expensive. Improving the development environment can offset this expense.

2. Productivity (source lines per work-year) is not constant but is a function of system difficulty and the development environment.

3. There is an intrinsic minimum development time which, given a system to be built of a certain size and complexity, can only be affected by improving the development environment with new methodologies and software tools and not by adding more people.

4. The importance of implementing new methodologies is clear, for with a good estimate of size and complexity, the model gives predictive indications for time, effort and loading and therefore, can more readily accommodate change.

	DEVELOPMENT TIME – CALENDAR MONTHS		EFFORT-MAN MONTHS		
	PUTNAM MINIMUM TIME ESTIMATE	ACTUAL DATA	PUTNAM ESTIMATE FOR MINIMUM TIME	ACTUAL DATA	PUTNAM VALUE FOR ACTUAL TIME
A	12.0	11.0	124	167	176
B	12.7	13	83	82	76
C	12.8	15	304	150	161
D	10.2	11.7	70	51	41

A and B were schedule driven (note closeness to minimum time).
C was resource driven, that is, was not at minimum time.
Tradeoff is shown in "Putnam Value for Actual Time" column.

Figure 1. Comparison of Resource Estimates, Putnam vs. Xerox Actual

MANLOADING ESTIMATED AND ACTUAL

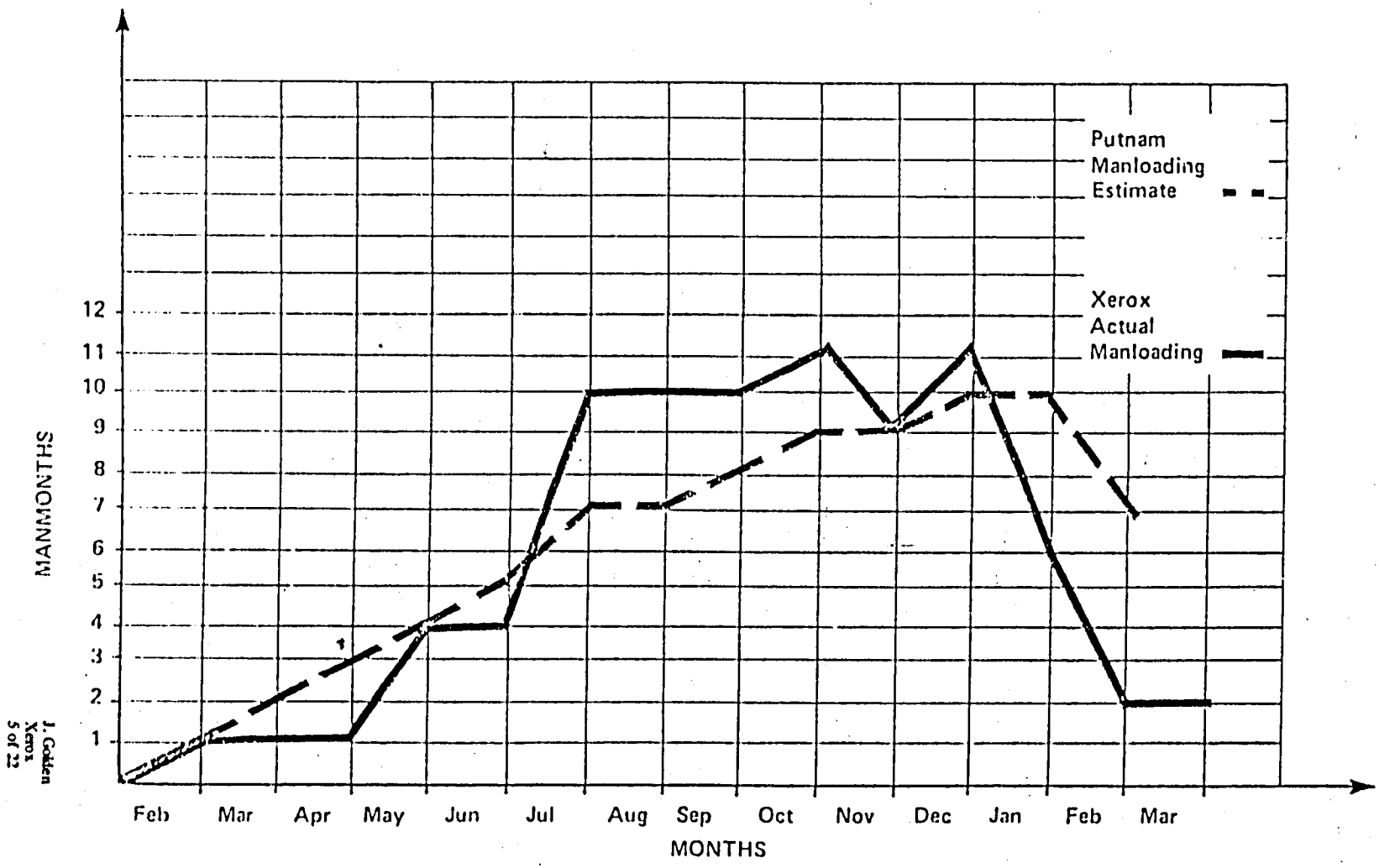


Figure 2

REFERENCES

1. Putnam, L. H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions On Software Engineering, SE-4:4, July 1978, pp. 345-360.

2. Brooks, F. P. Jr., The Mythical Man-Month, Reading MA; Addison Wesley, 1975.

3. Norden, Peter V., "Useful Tools for Project Management," Management of Production, M. K. Stan (Ed.), Penguin Books, Inc., Baltimore, MD, 1970.

4. Norden, Peter V., "Project Life Cycle Modelling: Background and Application of the Life Cycle Curves," U.S. Army Life Cycle Workshop, 1977.

5. Parr, F. N., "An Alternative to the Rayleigh Curve Model for Software Development Effort," IEEE Transactions On Software Engineering, pp. 291-296, Vol. SE-6, No. 3, May 1980.

6. Meyers, Ware, "A Statistical Approach to Scheduling Software Development," Computer, Dec., 1978, pp. 23-25.

7. Golden, J. R., "Effect of Development Time on Error Rate," Private Correspondence to L. Putnam, October 23, 1979.

THE VIEWGRAPH MATERIALS
for the
J. GOLDEN PRESENTATION FOLLOW

AGENDA

- REVIEW OF PUTNAM'S SOFTWARE EQUATION AND IMPLICATIONS
- SOME RESULTS AT XEROX
- EFFECT ON DEVELOPMENT TIME ON RELATIVE ERROR

THE NORDEN-RAYLEIGH DIFFERENTIAL EQUATION

- RATE OF WORK ACCOMPLISHMENT IS EQUAL TO THE WORK REMAINING TIMES THE "PACE"

LET

$Y(t)$ = WORK COMPLETED AT TIME

K = TOTAL WORK

t_d = DEVELOPMENT TIME

THEN

$$\frac{dY}{dt} = \frac{t}{t_d^2}(K - Y) \qquad (1)$$

AND

$$Y(t) = K(1 - e^{-t^2/2t_d^2}) \qquad (2)$$

$$\frac{dY}{dt} = \frac{Kt}{t_d^2} e^{-t^2/2t_d^2} \qquad (3)$$

LINEARIZE (3) BY TAKING LOGARITHMS AND PLOTTING

dY/dt VERSUS t^2

$$\log\left(\frac{\dot{Y}}{t}\right) = \log\frac{K}{t_d^2} - t^2/2t_d^2$$

$\frac{K}{t_d^2}$ REPRESENTS "DIFFICULTY" - - -

A FUNDAMENTAL SYSTEM PARAMETER

PRODUCTIVITY (P = S/K) IS A FUNCTION OF DIFFICULTY, THAT IS, PRODUCTIVITY IS NOT A CONSTANT NOR CAN IT BE DETERMINED BY MANAGEMENT

$$P = cD^{-x} \quad \text{OR} \quad \frac{S}{K} = c\left(\frac{K}{t_d^2}\right)^{-x}$$

PUTNAM FOUND $x \simeq 2/3$

SOLVING FOR S WE HAVE THE <u>SOFTWARE EQUATION</u>

$$S = cK^{1/3}t_d^{4/3} \qquad (4)$$

OR

$$K = S^3/c^3t_d^4$$

TIME AND EFFORT ARE NOT LINEARLY INTERCHANGEABLE

BROOKS' FIRST LAW

AND THE SOFTWARE EQUATION SHOWS US HOW NON-LINEAR IT IS.

WHAT IS THE EFFECT OF TIME ON DIFFICULTY?

CALCULATE GRADIENT

$$\nabla D = -\frac{2K}{t_d^2}(\hat{t}_d) + \left(\frac{1}{t_d^2}\right)(\hat{K})$$

TERM IN NEGATIVE DEVELOPMENT TIME DIRECTION DOMINATES THE GRADIENT.

SUBSTITUTING THE TRACE OF CONSTANT GRADIENT, $K = BT_D^3$, WHERE B IS AN EMPIRICAL CONSTANT, INTO THE SOFTWARE EQUATION (4) YIELDS THE EQUATION FOR MINIMUM DEVELOPMENT TIME:

$$S = cB^{1/3}t_d^{7/3} \qquad (5)$$

THUS, THE ONLY WAY TO EFFECT THE MINIMUM POSSIBLE DEVELOPMENT TIME IS BY IMPROVING THE ENVIRONMENT AND NOT BY ADDING MORE PEOPLE.

OR

ADDING MORE FOLKS TO A LATE PROJECT MAKES IT LATER.

BROOKS' SECOND LAW

Comparison of Resource Estimates
Putnam vs. Xerox Actual

	DEVELOPMENT TIME - CALENDAR MONTHS		EFFORT-MAN MONTHS		
	PUTNAM MINIMUM TIME ESTIMATE	ACTUAL DATA	PUTNAM ESTIMATE FOR MINIMUM TIME	ACTUAL DATA	PUTNAM VALUE FOR ACTUAL TIME
A	12.0	11.0	124	167	176
B	12.7	13	83	82	76
C	12.8	15	304	150	161
D	10.2	11.7	70	51	41

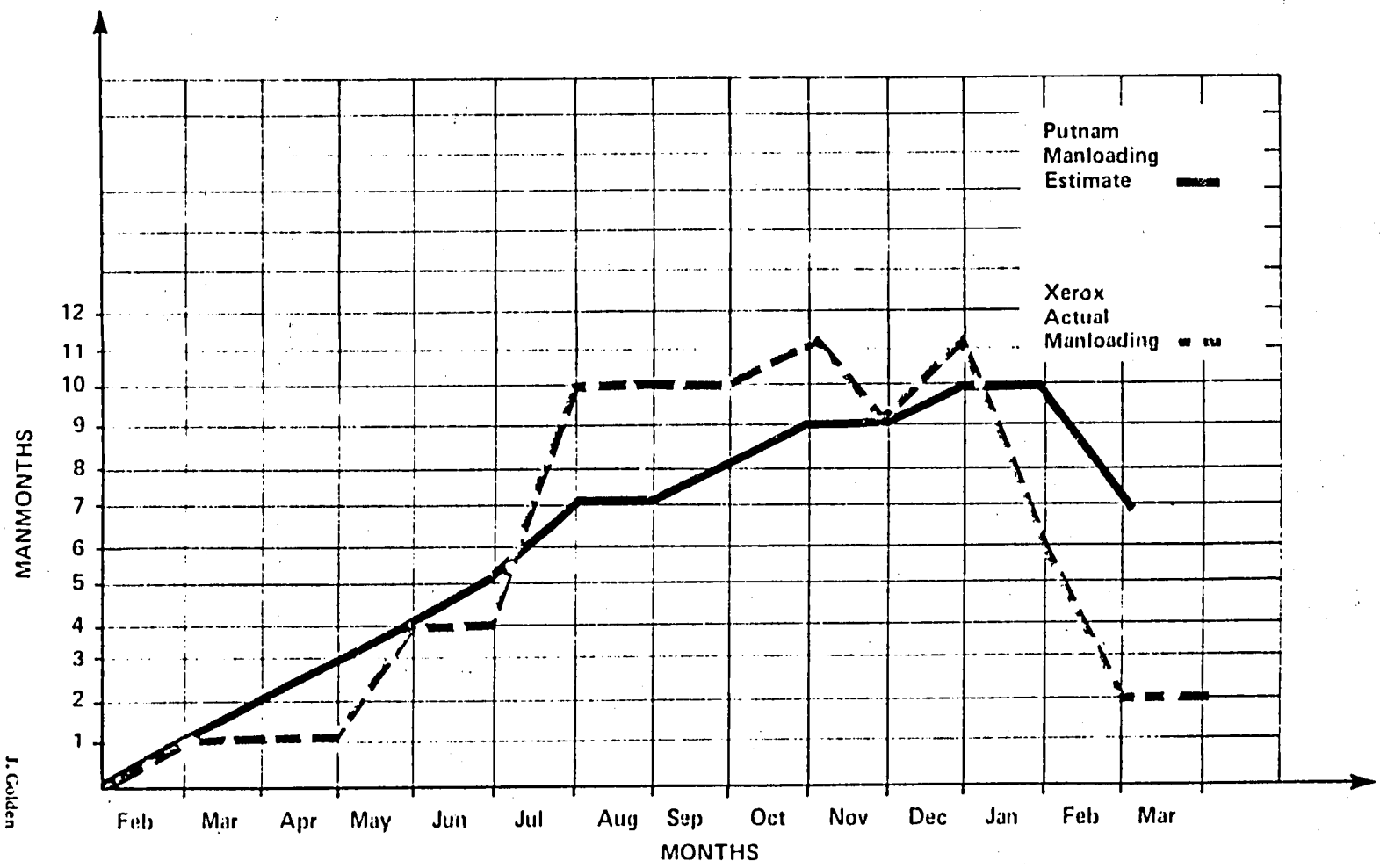
FIGURE 2 COMPARES ACTUAL LOADING WITH RAYLEIGH LOADING.

THESE RESULTS ARE SIMILAR TO OTHERS IN XEROX SUGGESTING THAT THE RAYLEIGH LOADING IS CLOSE TO WHAT WE DO.

CONCLUSIONS

- PLANNING A LONGER TIME RESULTS IN SIGNIFICANTLY LOWER COST (INVERSE FOURTH POWER LAW).
- APPLYING TOO MANY PEOPLE TOO SOON RESULTS IN HIGHER COST - PERIOD! USE THE RAYLEIGH LOADING CURVE.
- PAY ATTENTION TO THE MINIMUM DEVELOPMENT TIME.
- BREAK PROJECTS INTO SMALLER PIECES.
- USE THE TRADEOFF LAW IN PLANNING.
- RECOGNIZE PRODUCTIVITY IS NOT CONSTANT BUT IS A FUNCTION OF SYSTEM DIFFICULTY AND THE DEVELOPMENT ENVIRONMENT.
- RECOGNIZE THE DYNAMIC NATURE OF SYSTEM BUILDING AND USE THE MODEL TO HELP MANAGE CHANGE.

MANLOADING ESTIMATED AND ACTUAL



RESULTS FROM SEVERAL SYSTEMS AT XEROX

- PROJECTS A AND B ARE CLOSE TO MINIMUM TIME
- PROJECT C EXCEEDS MINIMUM TIME BY 20% AND REQUIRED HALF THE EFFORT THE MODEL WOULD PREDICT FOR MINIMUM.
- TRADEOFF SEEMS "REAL"

REFER TO FIGURE 1

- SPEND RESOURCES ON IMPROVING THE ENVIRONMENT TO HAVE REAL IMPACT.
- TRY TO DE-SCOPE A PROJECT IN TROUBLE RATHER THAN ADDING MORE PEOPLE. (I AM NOT OPTIMISTIC ABOUT THIS PROCESS IF SYSTEM WAS NOT WELL DESIGNED.)
- PUTNAM'S MODEL IS NOT A SUBSTITUTE FOR PROJECT MANAGEMENT BUT RATHER IS A KEY INPUT.
- SYSTEM SIZING POSES A MAJOR DIFFICULTY; LINES OF CODE IS FELT TO BE WANTING.
- STRUCTURED ANALYSIS MAY BE AN ANSWER TO ENVIRONMENT AND SIZING.

EFFECT OF DEVELOPMENT TIME ON RELATIVE ERROR

CALCULATE THE EFFECT OF THE DEVELOPMENT TIME, t_d, ON THE RELATIVE ERROR, E = S1/S, USING PUTNAM'S SOFTWARE EQUATION:

$$S = cK^{1/3}t_d^{4/3}$$

S IS TOTAL SOURCE LINES

S1 IS SOURCE LINES IN ERROR

TO DO THIS WE CALCULATE THE TOTAL DERIVATIVE OF E WITH RESPECT TO t_d

$$S = cK^{1/3}t_d^{4/3} \qquad \text{SOFTWARE EQUATION}$$

$$\frac{dE}{dt_d} = \frac{dE}{dS}\frac{dS}{dt_d}$$

$$= -\frac{S_1}{S^2}\frac{dS}{dt_d} \qquad \text{SINCE } E = S1/S$$

$$= -\frac{S_1}{S^2}\, cK^{1/3}\frac{4}{3}\, t_d^{4/3}$$

$$= -\frac{ES}{S^2}\, cK^{1/3}\frac{4}{3}\, t_d^{1/3} \qquad (ES = S1)$$

$$= -\, E\, cK^{1/3}\frac{4}{3}\, t_d^{1/3}/cK^{1/3}t_d^{4/3}$$

$$= -\frac{4}{3}\, E/t_d$$

$$\frac{dE}{E} = -\frac{4}{3}\frac{dt_d}{t_d} \qquad \text{SEPARATING VARIABLE}$$

$$\log E = -\frac{4}{3}\log t_d + c_1 \qquad \text{INTEGRATING}$$

$$E = c_2/t_d^{4/3}$$

N82 24003

SOFTWARE COST/RESOURCE MODELING:
Deep Space Network Software Cost Estimation Model

Robert J. Tausworthe
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91103

ABSTRACT

This paper presents a parametric software cost estimation model prepared for JPL Deep Space Network (DSN) Data Systems implementation tasks. The resource estimation model incorporates principles and data from a number of existing models, such as those of the General Research Corp., Doty Associates, IBM (Walston-Felix), Rome Air Force Development Center, University of Maryland, and Rayleigh-Norden-Putnam. The model calibrates task magnitude and difficulty, development environment, and software technology effects through prompted responses to a set of approximately 50 questions. Parameters in the model are adjusted to fit JPL software life-cycle statistics. The estimation model output scales a standard DSN Work Breakdown Structure skeleton, which is then input to a PERT/CPM system, producing a detailed schedule and resource budget for the project being planned.

*The research reported in this paper was performed at the Jet Propulsion Laboratory of the California Institute of Technology sponsored by the National Aeronautics and Space Administration, under contract NAS7-100.

1. INTRODUCTION

The early-on estimation of required resources and schedule for the development and maintenance of software has been one of the least precise aspects of the software life cycle, and yet, an orderly and rational attempt is necessary to plan and organize an implementation effort. Such an approach implies the need for a resource and schedule model that accepts as input the technical requirements to be achieved; the enormity of the task; the physical, environmental, human, and management constraints assumed or known to be in effect; the history base of similar and dissimilar experience; the means, alternatives, and technology available to the task; and a theory which is capable of correlating these parameters with measured results.

The least precise of such models is one which relies entirely on experience, intuition, and luck. It is sometimes referred to as a "WAG", or "Wildly Aspiring Guess."* When a more formalized, mathematical model with some statistical verification can be formulated, the model appelation is upgraded to "Scientific WAG," or "SWAG".

The prediction of human programming behavior is a problem in estimating a series of events in a stochastic process governed by an unknown probability function. The goal of a SWAG model is therefore to predict the events in such a way as to produce minimum variance (or risk). The optimum SWAG model can predict only to the limit imposed by the statistical distribution actually characterizing the human activity.

The optimal SWAG model would require the precise quantification of all technical, environmental, management, and human behavioristic parameters, and would combine these into a mathematical formula producing maximum likelihood results. Lacking this precise quantification, the best that one may hope for in a SWAG model is that it accommodate the principal factors effecting the estimate variance (or risk) in a way that reduces the variance (or risk) from what it would be, had that factor not been included.

There are a number of SWAGs in existence. Fourteen software cost estimation models are summarized in Ref. 1, and nine were evaluated for use by the Jet Propulsion Laboratory in Ref. 2. None of these, by itself, seemed to the author to contain sufficient range of application and adaptability to the diverse kinds of software being produced at the Jet Propulsion Laboratory, as to quantify the relevant cost factors and risks with sufficient accuracy to be useful. Taken all together, however, these models seemed to possess, in their union, the potential for as good a SWAG as could be obtained at the current state of the art.

An IBM study (Walston-Felix, Ref. 3) reported the analysis of 50 software projects with respect to 68 variables believed to influence productivity. Of these, 29 showed a significant, high correlation with productivity, and were included in their estimation model.

A number of other models (General Research Corp., Doty Associates, TRW, Air Force Electronic Systems Division, Tecolote, Aerospace Corp., et. al., reported in Ref. 1, and the University of Maryland, Ref. 4) provide productivity data with best-fit curves using many fewer parameters.

*More often, the acronym is somewhat differently derived, but with the same general connotation.

Still other models, notably the Rayleigh-Norden-Putnam model (Ref. 5, 6) presuppose a few-parameter, specific mathematical model, which is then calibrated using available industry data to provide tradeoffs between effort, duration, and risk.

Several models (e.g., Wolverton, Ref. 7) proceed to detail resource expenditures into the various phases of activity. This TRW model additionally compensates for task difficulty and some environmental factors.

Some of the models are fully automated, such as PRICE-S (Ref. 8), SLIM (Ref. 9), and SLICE (Ref. 10). The others appear to be calculative, or perhaps small programs, to be used by project planning staffs.

The software cost model described in this paper is fully automated; it borrows and extends features from many of the models above. It utilizes 7 factors from the GRC model, 29 factors from, or similar to, the Walston-Felix model, and incorporates an inherited (or existing) code model due to the author. It utilizes the PERT estimation technique to estimate the expected size and variance of the software elements to be produced. It utilizes a modification of the Rayleigh-Norden-Putnam model to check on the confidence factors associated with the resultant resource estimates. It applies the estimated effort, staff, and duration to a standard Work Breakdown Structure (WBS) developed for JPL Deep Space Network (DSN) software tasks, and automatically produces a task budget and schedule to be used at either the initial system/subsystem planning, software implementation planning, or software maintenance planning stages of a project.

There are about 70 parameters within the model which relate to productivity, duration, staffing level, documentation, and computer resources. Another 70 parameters divide the total estimated effort among WBS subtasks: an additional 70 relate total duration to subtask durations. Subtask precedences are preset (but adjustable), and drive a PERT/Critical-Path-Method scheduling algorithm.

The outputs of the model include estimates and variance values for program size, staff productivity, effort, duration, staffing, documentation, and computer resources, together with confidence level checking, a complete scheduling early/late start/finish and float-time budget and a Gantt chart (schedule bar chart) of the planned activities.

All parameters in the automated model are easily altered by a simple text editing process, without recompilation of the programs. For all its seeming complexity, the model itself is simple to use. Only a series of questions relating to size and environment need to be answered.

The ensuing sections of this paper describe the model in more detail. The values of parameters used currently are subject to modification and refinement as further calibration and usage of the model proceeds.

Concerning accuracy: at this writing, insufficient data has been collected from DSN projects to optimize the parameters of the model to fit DSN productivity, duration, etc. Therefore, the model accuracy is unknown, as pertaining to DSN prediction. However, the model does fit industry statistics (or can be made to fit any of the cited source models) by proper parameter selection. For this reason, it is felt that the few JPL data points that have been factored in will yield accuracy figures as good as; or perhaps better than, the other models in their stated environments.

The model is currently being used to estimate and plan all new programming activities involving DSN Data Systems implementations.

2. MODEL DESCRIPTION

The Software Cost and Resource Estimation Model (the acronum SCAREM is not used!) overview appears in Figure 1 in data-flow-diagram format. Program size and staff productivity factors are separately estimated and then combined to estimate effort, staffing, duration, documentation, and computer resources. The model produces uninflated dollar costs for documentation and computer resources. Both estimated mean and variance values for all resources are output.

Estimated values are presented in the automated model to the user as advice. The user is admonished to use these figures, adding sufficient margin to ensure project completion within a desired confidence value.

The model then accepts any two of the three parameters: effort, average staff, and duration. These entries are checked against a model akin to the Rayleigh-Norden-Putnam model, but altered to conform with power-law fits to measured data. A confidence factor is computed to the resources entered. The user may alter the input estimates, if desired, for another check.

Once acceptable resources and duration have been decided, the model proceeds to produce a standard DSN software implementation work breakdown structure and schedule without further input from the user.

2.1 Estimation of Program Size

The size of the software task is measured in "equivalent" Kilo-Source-Lines of Executable Code (KSLEC). A source line of executable code is defined basically as a source language statement occupying one physical line in the source medium that results in generation of object code, reservation of storage, or definition of data type. Comments are excluded, as are statements merely defining labels and equivalences of identifiers. If several basic statements may appear on one physical source line, each such statement should be added separately into the KSLEC count.

Source lines of new code are weighted differently than lines of reused code, in proportion to the relative amount of effort required to adapt the inherited code to the current task. Even deleted lines of code contribute to the programming effort, and therefore increase the "equivalent" KSLEC count.

The programming tasks involved with the generation of new code and reuse of existing code are depicted in Figure 2. The efforts to specify, produce, document, and test a new line of code is normalized to unity; the lines of code added, changed, deleted, and retested-only contribute to the equivalent line count according to relative effort. The extent of existing-module modification is measured by the number of lines added in, the number changed, and the number deleted. The equivalent lines of code produced is then defined to be a weighted linear sum of the lines of code in each category.

The assumptions with respect to each component are the following:

(a) New code is subjected to the entire standard implementation process.

(b) The recognition of the reuse of existing code is made in the preliminary design phase, so code added, changed or deleted from modules goes only through subsequent phases.

(c) Added code takes the same effort as new code in corresponding phases where activity takes place.

FIGURE 1. DSN SOFTWARE COST ESTIMATION MODEL DATA FLOWDIAGRAM

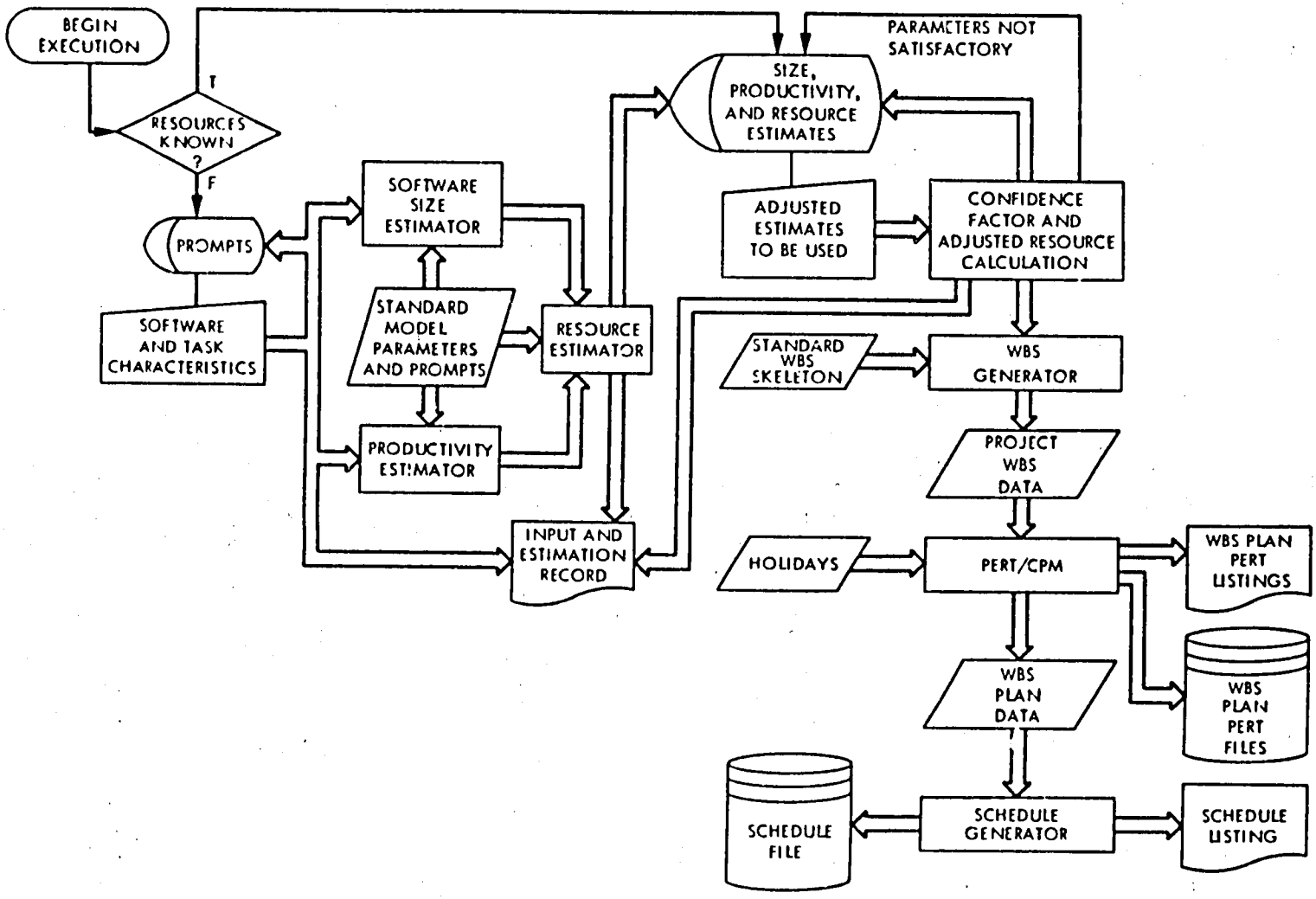
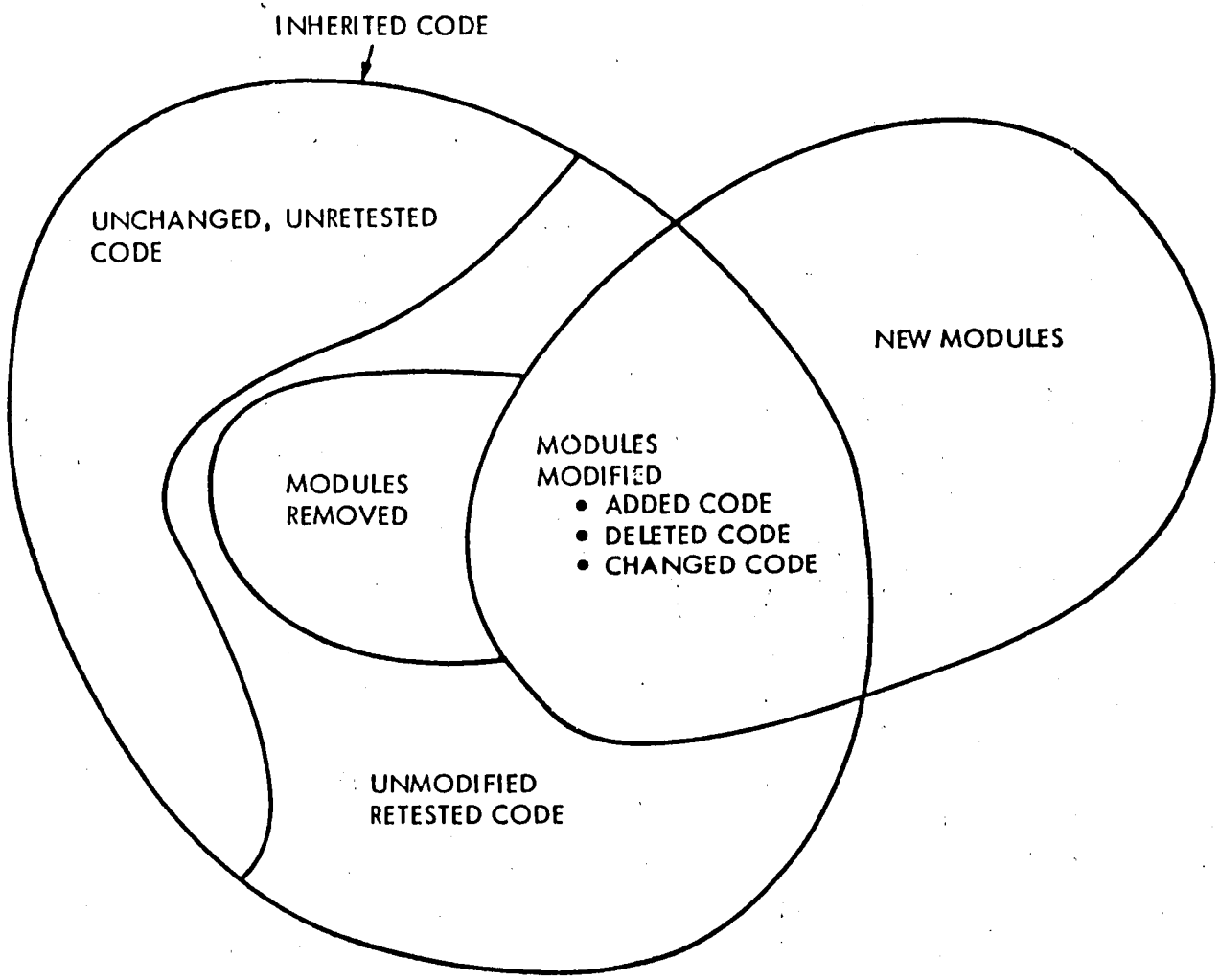


FIGURE 2

SOFTWARE IMPLEMENTATION TASKS RELATED TO NEW AND INHERITED CODE ACTIVITIES



(d) Changed code requires the same design and testing effort as new code, but less documentation and coding effort.

(e) Deleted lines from existing modules require reduced design, coding, and documentation effort, and no testing of the deleted lines.

(f) Any module changed is completely retested and requalified.

(g) Deleted modules require reduced architectural, interface, and detailed design considerations than new code; only that coding effort required to remove the unwanted code contributes to the coding time; no testing of the deleted code is possible; and documentation effort involves removal of entire sections of existing material and minor clean up. Retesting is covered in modules which interfaced with the deleted module.

(h) Retested unmodified code requires revalidation of the interface design and retesting efforts only.

Each of the size parameters is estimated by the PERT technique (Ref. 11) which produces a maximum likelihood value and variance.

The estimated value for the "equivalent lines of code" is composed of the weighted sum of the maximum likelihood estimate for each parameter, and its standard deviation is the weighted root-sum-square of the individual deviations. The weights are determined as the relative effort required for each category of code.

2.2 Estimation of Productivity

In this model, the productivity P is defined as total equivalent KSLEC (here denoted L) produced divided by the total staff work effort (here denoted W),

$$P = L/W, \text{ KSLEC/staff-month}$$

A number of data bases (e.g., Refs. 3, 12, 13) have shown that L and W are well correlated through a power-law relationship

$$W = L^a/P_1$$

where P_1 is the average 1-KSLEC productivity rate.

The value of P_1 is set by primarily technology and environment. In fact, industry studies show that P_1 may vary by as much as 50:1 (or more!) as a function of such factors. The value of a, however, in each environment where data is available, shows a relative constancy, at a value near unity.

It seems intuitive, all other things being equal, that productivity cannot increase as the program size rises; however, the least-square-power-law fits to data bases yield a-values of 0.91 (IBM), 0.991 (Doty), 0.986 (Univ. of MD), and 0.975 (RADC). The consistency of these figures therefore seems to indicate that utilization of tools and technology takes place on larger projects (where it counts) more than on smaller projects.

The value for a currently being used is unity. Thus, the model may tend to be somewhat conservative.

Several models have contributed to the formula by which P_I is calculated, principally those of GRC (Ref. 14) and IBM (Ref. 3). The form of P_I is:

$$P_I = P_0 A$$

where P_0 is a constant factor, and A is a technology and environmental adjustment factor. The value of A is computed from factors judged to be significant,

$$A = A_1 A_2 \ldots A_n$$

where each factor A_i ranges between a maximum value and a minimum value for that factor, depending on the extent to which the factor is present in the software task. The ratio of A_{max} to A_{min} is currently adjusted to span a 50:1 ratio of productivity.

2.3 Estimation of Implementation Task Duration

The IBM, University of Maryland, and RADC statistics suggest that the average duration T required for L KSLEC and W staff-months effort is approximated by

$$T = T_1 W^b$$

where T_1 is the 1-KSLEC average duration, and b is a time-factor exponent found from industry statistics. The value used in the DNS model, $T_1 = 4.8$, was adjusted to fit limited available DSN data, and $b = 0.356$ was the average power-law for the more extensive IBM, RADC, and GRC data.

2.4 Average Staff

The average staff, in persons, results from manipulation of the duration equation,

$$S = W/T = (1/T_1)\, W^{1-b}$$

The staffing exponent, $1 - b = 0.644$, implied in the DSN model compares with measured values averaging 0.629 across industry.

2.5 Documentation Sizing and Cost

IBM statistics showed a nearly linear relationship between pages of documentation and lines of code, whereas the University of Maryland measured almost a square-root relationship. DSN experience over 6 Mark-III Data System programs revealed an exponent about midway in between (0.83). A study of maintenance user needs (Ref. 15) recommended that documentation be about 40-50 pages per KSLEC for programs in the 30 KSLEC vicinity. The formula used in the model for the number of pages of documentation is

$$D = D_1 L^d$$

The model currently uses $D_1 = 90$ and $d = 0.83$ to match the DSN experience and guidelines.

The documentation cost is found by a straight dollar-per-page rate; a figure of $30 page is used in the current model.

2.6 Computer Resources

IBM and TRW give statistical figures for computer time costs as functions of lines of code and total effort, and also as a fraction of total cost. The DSN, however, mostly has dedicated mini-computers for which operational costs are not assessed to the implementation task on a usage basis. TRW does, however, also estimate a linear relationship between CPU time required per machine code instruction, of about 25.2 CPU hours per thousand instructions.

The DSN model computer CPU resources as

$$C = C_1 L^c$$

The exponent value c = 0.96, given by Walston and Felix (who give dollar costs, rather than CPU time), is adopted to account for the general trend of CPU time with program size.

If CPU dollar cost is relevant, the model computes this at a straight dollar-per-CPU-hour figure (zero in the DSN model, but a parameter is available for other applications).

2.7 Confidence Level Computation

The values predicted by the SWAG model are average values based on statistics taken over many projects. The estimated values represent but one set of resources that, on the average, produce the intended success. However, effort and time can be traded (to some extent) to produce other equally valid project scenarios.

The Rayleigh-Norden-Putnam model has a time-effort relationship for checking the reasonability of resource values other than the average values produced by the SWAG. However, in order to use this model, several adaptations were felt to be indicated.

First, the basic differential equation was modified to accommodate a nonlinear "pace" factor (or learning curve). The model assumes the work equation to be of the form

$$w' = At^r(K - w)$$

in which w = w(t) is the cumulative work effort up to time t, K is the total life-cycle effort, and r is the "pace-of-work" exponent.

Second, the model assumes the following parametric form of the software equation:

$$L = c_p w^p t^q$$

The factor f = q/p sets the time-effort tradeoff law. Putnam uses p = 1/3, q = 4/3 (f = 4), and such a value may well be valid for large projects in his data base. However, for the smaller projects typifying the DSN, an f factor which would require only 1.5 times the effort (3 times the staff) to reduce schedule by a factor of 2 seems to be more within the DSN experience (more data is needed here).

Based on these modifications, it is possible to solve for p, q, and r, and c_p in terms of the parameters of observed average power-law relationships between L, W, and t.

f = 0.585 q = 0.484 c_p = 0.0621 A
p = 0.828 r = 1.81

This model is used to compute confidence for any values of L, W, and T proposed for the project. The software equation is used to estimate the margin over or under the average project figures. By assuming that the statistics are log-normal (verified by the RADC data base), the confidence factor in producing L KSLEC in W staff-months effort and T months duration is

$$\text{Conf} = P\left\{L_{act} \leq L,\ w \leq W,\ t \leq T\right\}$$

Computation of the confedence level involves finding the optimum operating point on the software-equation curve for margin calculation, and then numeric integration of the normal probability function.

One interesting relavation to the author was that the probability of success in not expending more than W staff-months of effort and not requiring more T months duration, for the average SWAG estimate case, is only about 25%. Moreover, a significant amount of bias in W and T is required to raise the confidence to 50%.

Management should not despair, however. What the confidence limit indicates for average projects is that one out of four will go OK; the others will require some form of management intervention, in the form of schedule or resource extension.

3. TYPICAL OPERATION OF THE MODEL

Blank forms and/or CRT prompted inputs are used to specify the parameters needed by the cost model program. Outputs are selectable, and include a WBS task file, PERT plan, and schedule. The WBS task file can be edited to modify task titles, precedences, allocated resources, or durations in addition, new tasks may be entered and actual completion or need dates may be affixed, so that the PERT and schedule portions of the program form a project detailed planning and control tool.

Typical inputs and outputs are shown in Appendix A.

4. SUMMARY AND CONCLUSION

The Software Cost Model reported here is the first of a series of planned refinements. As the model is used and as performance data are collected, no doubt changes will be made: adjustments of parameters, alteration of formulas, modifications of formats, new input data types, and additional kinds of outputs. Extensions currently envisioned are the automated transfer of the WBS data-base generated by the model into the project control system currently being used in DSN Data Systems implementation projects, and the refinement of the model to include non-linear scaling of overall effort and duration into individual task requirements.

If the model, even now, seems complex, then it is justly so, for the factors which affect human performance are generally complex and unpredictable, except in statistical terms. One sample function chosen from a stochastic ensemble is hardly ever "average" or "typical". One must expect variations between actual behavior and predictions made by any model.

The directions for the future are to refine the model for greater accuracy (within human performance estimation capacity limits), to extend the utility of the model throughout the entire life cycle, and to provide the basis for indicating where, and in what form, new software technology is needed.

REFERENCES

1. "Quantitive Software Models," Report SRR-1, Data and Analysis Center for Software, RADC, Griffiss AFB, N.Y., March 1979.

2. Galorath, D. D., and Reifer, D. J., "Analysis of the State-of-the-Art of Parametric Software Cost Modeling," Report SMC-7R-006, Software Management Consultants, Torrance, CA, August 1980.

3. Walston, C. E., and Felix, C. P., "A Method of Programming Measurement and Estimation," IBM System Journal, Vol. 16, No. 1, 1977.

4. Freburger, K., and Basili, V. R., "The Software Engineering Laboratory: Relationship Equations," Tech. Rept. TR-764, SEL-3, NSG-5123, University of Maryland, Computer Science Center, College Park, MD. 20742, May, 1979.

5. Norden, P. V., "Project Life Cycle Modeling," Software Life-Cycle Management Workshop, United States Army Computer Systems Command, August, 1977, pp. 217-306.

6. Putnam, L. H., "Influence of the Time-Difficulty Factor in Large-Scale Software Development," Software Life-Cycle Management Workshop, United States Army Computer Systems Command, August 1977, pp. 307-312.

7. Wolverton, R. W., "The Cost of Developing Large Scale Software," IEEE Transactions on Computers, Vol. C-23, No. 6, June, 1974.

8. Freiman, F. R., "PRICE Software Model," RCA PRICE Systems Publication, Morestown, N.J., November 1977.

9. "Software Life-Cycle Management (SLIM) Estimating Model," Quantitative Software Management, Inc., McLean, Virginia, 1978.

10. Kustanowitz, A. L., "System Life Cycle Estimation (SLICE)," IEEE First International Computer Software and Applications Conference, Chicago, Ill.

11. The author is unable to trace the definitive reference for this work. The usage cited is exposed in Putnam, L. H., "Example of an Early Sizing, Cost and Schedule estimate for an Application Software System," COMPSAC '78, November 13-16, 1978.

12. Graver, C. A., et al., "Cost Reporting Elements and Activity Cost Tradeoffs for Defense System Software," CR-1-72/1, General Research Corp., Santa Barbara, Calif., May 1977.

13. Nelson, Richard, "Software Data Collection and Analysis," (Partial Report), Rome Air Force Development Center, Data and Analysis Center for Software, Griffiss AFB, NY, Sept. 1978.

14. Carriere, N. M., and Thibodeau, R., "Development of a Logistics Software Cost Estimating Technique for Foreign Military Sales," General Research Corp., CR-3-839, Santa Barbara, CA, June 1979.

15. Tausworthe, Robert C., "Preparation Guide for Class B Software Specification Documents," Report #79-56, Jet Propulsion Laboratory, Pasadena, Calif., October 1979.

APPENDIX A
EXAMPLE OF OPERATION

This Appendix contains a sample of the sequence of inputs and outputs from the Deep Space Network Software Cost Estimation Model.

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
Date Estimated: 14NOV80
Model Data Version 1.3 31OCT80

Answer the following items to the best of your estimation.

1. How much new code is to be produced (completely new modules)?
   - Maximum value, kilo-lines executable source(99% confidence level)? 3.5
   - Expected value, kilo-lines executable source? 3.3
   - Minimum value, kilo-lines executable source(99% confidence level)? 3.1

2. How much code exists in modules requiring modification?
   - Maximum value, kilo-lines executable source(99% confidence level)? 6.9
   - Expected value, kilo-lines executable source? 6.6
   - Minimum value, kilo-lines executable source(99% confidence level)? 6.3

3. How much code will be deleted from these existing modules?
   - Maximum value, kilo-lines executable source(99% confidence level)? .4
   - Expected value, kilo-lines executable source? .3
   - Minimum value, kilo-lines executable source(99% confidence level)? .2

4. How much code will be added to these existing modules?
   - Maximum value, kilo-lines executable source(99% confidence level)? .7
   - Expected value, kilo-lines executable source? .6
   - Minimum value, kilo-lines executable source(99% confidence level)? .4

5. How much code will be changed in other ways in these modules?
   - Maximum value, kilo-lines executable source(99% confidence level)? 1.2
   - Expected value, kilo-lines executable source? .9
   - Minimum value, kilo-lines executable source(99% confidence level)? .7

6. How much code will be deleted as entire modules from existing code?
   - Maximum value, kilo-lines executable source(99% confidence level)? 1.4
   - Expected value, kilo-lines executable source? 1.3
   - Minimum value, kilo-lines executable source(99% confidence level)? 1.2

7. How much of the remaining existing code must be retested?
   - Maximum value, kilo-lines executable source(99% confidence level)? 2.1
   - Expected value, kilo-lines executable source? 1.9
   - Minimum value, kilo-lines executable source(99% confidence level)? 1.5

8. Expected percentage of code to be developed actually delivered (0-90, 91-99, 100)? 91-99

9. How many different kinds of input/output data items per 1000 lines of new or modified code(>80, 16-80, 0-15)? 16-80

10. Overall complexity of program and data base architecture (high, medium, low)? MEDIUM

11. Complexity of code logical design(high, medium, low)? LOW

12. What percent of the programming task is in Assembly language? 9

13. What percent of the new or modified code must be storage-optimized? 9

14. What percent of the new or modified code must be timing-optimized?	9
15. What percent of the total programming task is 'easy'?	20
16. What percent of the total programming task is 'hard'?	30
17. When is work to start, on the(FRD/FDD, SRD, SDD)?	FRD/FDD
18. What percent of the total program requirments will be established and stable before design, and will not be altered before delivery?	80
19. What percent of the requirements are likely to change slightly before delivery, but will do so under baseline change control?	10
20. What percent of the requirements are likely to change more drastically before delivery, but will do so under baseline control?	5
21. Complexity of program functional requirements(high, medium, low)?	LOW
22. Expected user involvement in requirements definition (much, some, none)?	MUCH
23. Customer experience in application area(much, none, some)?	SOME
24. Customer/implementor organizational interface complexity (high, normal, low)?	NORMAL
25. Interfaces with other SW development projects or organizations (many, few, none)?	FEW
26. Efficiency of implementing organization(poor, ok, good)?	GOOD
27. Overall implementation personnel qualifications and motivation (low, average, high)?	HIGH
28. Percentage of programmers doing functional design who will also be doing development(<25, 25-50, >50)?	25-50
29. Previous programmer experience with application of similar or greater size and complexity(minimal, average, extensive)?	AVERAGE
30. What is the average staff experience, in years, obtained from work similar to that required in the task being estimated?	6
31. Previous experience with operational computer to be used (minimal, average, extensive)?	MINIMAL
32. Previous experience with programming language(s) to be used (minimal, average, extensive)?	MINIMAL
33. Use of top-down methodology(low, medium, high)?	HIGH
34. Use of structured programmer team concepts(low, medium, high)?	HIGH
35. Use of Structured Programming(low, medium, high)?	HIGH

36. Use of design and code inspections(low, QA, peer)?	QA
37. Classified security environment for computer(yes, , no)?	NO
38. Hardware under concurrent development(much, some, none)?	NONE
39. Percent of work done at primary development site (<70, 70-90, >90)?	70-90
40. Development computer access mode(remote, scheduled, demand)?	DEMAND
41. Percent of development computer access availability(<30, 30-60, >60)?	30-60
42. Quality of SW development tools and environment(poor, ok, good)?	OK
43. Maturity of system and support software(buggy, ok, good)?	OK
44. Overall adverse constraints on program design (severe, average, minimal)?	MINIMAL
45. Is the program real-time, multi-task(chiefly, some, no)?	NO
46. SW to be adaptable to multiple computer configurations or environments (yes, , no)?	NO
47. Adaptation required to change from development to operational environment(much, some, minimal)?	MINIMAL

```
Estimated Overall Parameters:
                                                  =average value
                                        +1-sigma                    -1-sigma

Adjusted Lines of code= 6182 SLEC
 6280           6085
Effort= 26.5 person-months
 45.8           15.3
Staff productivity= 233 SLEC/staff-month
 404            135
Duration= 15.7 months
 19.0           12.9
Avg. Staff=  1.7
  2.9           1.0
Docume  ation=  537 pages      $16.1K
  645           448            $19.4K              $13.4K
Computer CPU time=  319 hours    $0.0K
  478           212              $0.0K               $0.0K

Use these figures to arrive at Effort, Duration, and Staffing
requirements. Include factors to provide acceptible risk
and confidence levels.


Values specified are:
   Kilo-lines of code=                                                  6.18
   Effort (person-months):                                              32.0
   Duration (months):                                                   16.0
   Average staff (persons):                                              2.0

    For the numbers you have entered, a reasonableness check indicates that
the average project would produce 7303 lines of code, using 32 staff-months
of resources and 16 months of duration, with an average staff of 2 persons,
for a productivity of 228 SLEC/staff-month.

The level of confidence in delivering 6182 lines of code,
on-time and within resources= 33 %.

Is output to be saved in a file?                                          Y

Name of output file to be created:                                    VCEDIT

Schedule start date:                                                17NOV80


Select desired outputs and output media, or enter RETURN only for
defaults. Defaults are 1A, 2A, and 3A. Choices are:

                  1=Gantt Chart                 A=file
                  2=PERT data, 132 width        B=line printer
                  3=PERT data, 80 width

                                                                  1B,2B,3A
Choice(s):
```

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

CODE	TASK	WHO	EFF	E-START	L-FINSH	FLT
*0.	START		0	17NOV80	17NOV80	0
*1.	Sys Plans, Reqts, & Design		0	9JAN81	9JAN81	0
* 1.1	Define Subsys Reqts		15	17NOV80	3DEC80	0
* 1.2	FRD		0	8DEC80	8DEC80	0
! 1.2.1	Write all sections		4	17NOV80	5DEC80	9
* 1.2.2	Edit and release FRD		2	5DEC80	8DEC80	0
* 1.3	Level B Review		3	8DEC80	10DEC80	0
* 1.4	Define Sys Architecture		18	10DEC80	31DEC80	0
* 1.5	FDD		0	7JAN81	7JAN81	0
! 1.5.1	Write all sections		4	10DEC80	6JAN81	12
* 1.5.2	Edit and release		2	6JAN81	7JAN81	0
* 1.6	Level C Review		3	7JAN81	9JAN81	0
*2.	SW Planning and Reqts		0	16FEB81	16FEB81	0
* 2.1	Define Software Reqts		25	9JAN81	4FEB81	0
* 2.2	SRD		0	12FEB81	12FEB81	0
! 2.2.1	Write all sections		4	9JAN81	11FEB81	21
* 2.2.2	Edit and release		2	11FEB81	12FEB81	0
* 2.3	Level D Review		2	12FEB81	16FEB81	0
*3.	SW Architecture and Design		0	6APR81	6APR81	0
* 3.1	Define SW architecture		31	16FEB81	19MAR81	0
* 3.2	SDD		0	2APR81	2APR81	0
! 3.2.1	Write all sections		4	16FEB81	1APR81	30
* 3.2.2	Edit and release		2	1APR81	2APR81	0
! 3.3	System Interface Design		22	16FEB81	1APR81	20
* 3.4	Level E Review		2	2APR81	6APR81	0
*4.	SW Detailed Design & Prod		0	22DEC81	22DEC81	0
! 4.1	SSD		0	23FEB82	11MAR82	12
! 4.1.1	Write Sections 1,2,3		6	6APR81	18DEC81	175
* 4.1.2	Write Section 4		18	6APR81	24APR81	0
! 4.1.3	Write Section 5		43	24APR81	16DEC81	142
! 4.1.4	Write Section 6		4	6APR81	18DEC81	176
! 4.1.5	Write Section 7		9	6APR81	16DEC81	173
! 4.1.6	Edit and release		6	16FEB82	11MAR82	12
! 4.2	SDM		0	22FEB82	11MAR82	13
* 4.2.1	Write preliminary draft		8	24APR81	4MAY81	0
! 4.2.2	Complete all sections		9	12JUN81	18DEC81	133
! 4.2.3	Edit and release		4	16FEB82	11MAR82	13
* 4.3	High-level Design Review		2	29MAY81	2JUN81	0
* 4.4	Module Production & Integ		0	16DEC81	18DEC81	0
* 4.4.1	Executive and control		18	4MAY81	29MAY81	0
* 4.4.2	I/O Modules		18	2JUN81	26JUN81	0
* 4.4.3	Interface handlers		18	26JUN81	23JUL81	0

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

CODE	TASK	WHO	EFF	E-START	L-FINSH	FLT
* 4.4.4	Function A		18	23JUL81	17AUG81	0
* 4.4.5	Function B		18	17AUG81	11SEP81	0
* 4.4.6	Function C		17	11SEP81	6OCT81	0
* 4.4.7	Function D		17	6OCT81	29OCT81	0
* 4.4.8	Function E		17	29OCT81	23NOV81	0
* 4.4.9	Function F		17	23NOV81	18DEC81	0
4.5	Special Tasks		0	10JUN81	18DEC81	132
4.5.1	Support software		12	2JUN81	18DEC81	132
4.5.2	Other		6	2JUN81	18DEC81	135
* 4.6	Acceptance Readiness Rvw		2	18DEC81	22DEC81	0
*5.	SW Test and Transfer		0	18MAR82	18MAR82	0
* 5.1	Verification tests		28	22DEC81	1FEB82	0
5.2	Contingency		25	9APR81	11MAR82	218
5.3	STT		0	19FEB82	18MAR82	19
5.3.1	Write all sections		14	2JUN81	1FEB82	159
5.3.2	Edit and release		2	18FEB82	11MAR82	14
* 5.4	Acceptance tests		20	1FEB82	18FEB82	0
* 5.5	Demonstration tests		22	18FEB82	11MAR82	0
* 5.6	Transfer, CDE to COE		7	11MAR82	18MAR82	0
6.	Mgt Tasks and Milestones		0	16DEC80	18MAR82	313
6.1	CDE Activities		37	17NOV80	18MAR82	313
* 6.2	Develop prelim budget		3	3DEC80	5DEC80	0
* 6.3	Develop Sys Impl Plan		3	31DEC80	6JAN81	0
* 6.4	Draft Software Impl Plan		6	4FEB81	11FEB81	0
* 6.5	Revise Impl Plan		12	19MAR81	1APR81	0
6.6	QA Audit		26	18FEB82	18MAR82	6
*FINISH			0	18MAR82	18MAR82	0

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

CODE	TASK	WHO	EFF ORT	START EARLY DAY	START EARLY DATE	START LATE DAY	START LATE DATE	FINISH EARLY DAY	FINISH EARLY DATE	FINISH LATE DAY	FINISH LATE DATE	FLOAT TIME DAYS
*0.	START		0	0	17NOV80	0	17NOV80	0	17NOV80	0	17NOV80	0
*1.	Sys Plans, Reqts, & Design		0	32	9JAN81	32	9JAN81	32	9JAN81	32	9JAN81	0
* 1.1	Define Subsys Reqts		15	0	17NOV80	0	17NOV80	10	3DEC80	10	3DEC80	0
* 1.2	FRD		0	13	8DEC80	13	8DEC80	13	8DEC80	13	8DEC80	0
1.2.1	Write all sections		4	0	17NOV80	9	2DEC80	3	20NOV80	12	5DEC80	9
* 1.2.2	Edit and release FRD		2	12	5DEC80	12	5DEC80	13	8DEC80	13	8DEC80	0
* 1.3	Level B Review		3	13	8DEC80	13	8DEC80	15	10DEC80	15	10DEC80	0
* 1.4	Define Sys Architecture		18	15	10DEC80	15	10DEC80	27	31DEC80	27	31DEC80	0
* 1.5	FDD		0	30	7JAN81	30	7JAN81	30	7JAN81	30	7JAN81	0
1.5.1	Write all sections		4	15	10DEC80	27	31DEC80	17	12DEC80	29	6JAN81	12
* 1.5.2	Edit and release		2	29	6JAN81	29	6JAN81	30	7JAN81	30	7JAN81	0
* 1.6	Level C Review		3	30	7JAN81	30	7JAN81	32	9JAN81	32	9JAN81	0
*2.	SW Planning and Reqts		0	58	16FEB81	58	16FEB81	58	16FEB81	58	16FEB81	0
* 2.1	Define Software Reqts		25	32	9JAN81	32	9JAN81	50	4FEB81	50	4FEB81	0
* 2.2	SRD		0	56	12FEB81	56	12FEB81	56	12FEB81	56	12FEB81	0
2.2.1	Write all sections		4	32	9JAN81	53	9FEB81	34	13JAN81	55	11FEB81	21
* 2.2.2	Edit and release		2	55	11FEB81	55	11FEB81	56	12FEB81	56	12FEB81	0
* 2.3	Level D Review		2	56	12FEB81	56	12FEB81	58	16FEB81	58	16FEB81	0
*3.	SW Architecture and Design		0	93	6APR81	93	6APR81	93	6APR81	93	6APR81	0
* 3.1	Define SW architecture		31	58	16FEB81	58	16FEB81	81	19MAR81	81	19MAR81	0
* 3.2	SDD		0	91	2APR81	91	2APR81	91	2APR81	91	2APR81	0
3.2.1	Write all sections		4	58	16FEB81	88	30MAR81	60	18FEB81	90	1APR81	30
* 3.2.2	Edit and release		2	90	1APR81	90	1APR81	91	2APR81	91	2APR81	0
3.3	System Interface Design		22	58	16FEB81	78	16MAR81	70	4MAR81	90	1APR81	20
* 3.4	Level E Review		2	91	2APR81	91	2APR81	93	6APR81	93	6APR81	0
*4.	SW Detailed Design & Prod		0	273	22DEC81	273	22DEC81	273	22DEC81	273	22DEC81	0
4.1	SSD		0	315	23FEB82	327	11MAR82	315	23FEB82	327	11MAR82	12
4.1.1	Write Sections 1,2,3		6	93	6APR81	268	15DEC81	96	9APR81	271	18DEC81	175
* 4.1.2	Write Section 4		18	93	6APR81	93	6APR81	107	24APR81	107	24APR81	0
4.1.3	Write Section 5		43	107	24APR81	249	16NOV81	129	27MAY81	271	18DEC81	142
4.1.4	Write Section 6		4	93	6APR81	269	16DEC81	95	8APR81	271	18DEC81	176
4.1.5	Write Section 7		9	93	6APR81	266	11DEC81	98	13APR81	271	18DEC81	173
4.1.6	Edit and release		6	312	18FEB82	324	8MAR82	315	23FEB82	327	11MAR82	12
4.2	SOM		0	314	22FEB82	327	11MAR82	314	22FEB82	327	11MAR82	13
* 4.2.1	Write preliminary draft		8	107	24APR81	107	24APR81	113	4MAY81	113	4MAY81	0
4.2.2	Complete all sections		9	133	2JUN81	266	11DEC81	138	9JUN81	271	18DEC81	133
4.2.3	Edit and release		4	312	18FEB82	325	9MAR82	314	22FEB82	327	11MAR82	13
* 4.3	High-level Design Review		2	131	29MAY81	131	29MAY81	133	2JUN81	133	2JUN81	0
* 4.4	Module Production & Integ		0	271	18DEC81	271	18DEC81	271	18DEC81	271	18DEC81	0
* 4.4.1	Executive and control		18	113	4MAY81	113	4MAY81	131	29MAY81	131	29MAY81	0
* 4.4.2	I/O Modules		18	133	2JUN81	133	2JUN81	151	26JUN81	151	26JUN81	0
* 4.4.3	Interface handlers		18	151	26JUN81	151	26JUN81	169	23JUL81	169	23JUL81	0

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

CODE	TASK	WHO	EFF ORT	START EARLY DAY	START EARLY DATE	START LATE DAY	START LATE DATE	FINISH EARLY DAY	FINISH EARLY DATE	FINISH LATE DAY	FINISH LATE DATE	FLOAT TIME DAYS
* 4.4.4	Function A		18	169	23JUL81	169	23JUL81	186	17AUG81	186	17AUG81	0
* 4.4.5	Function B		18	186	17AUG81	186	17AUG81	203	11SEP81	203	11SEP81	0
* 4.4.6	Function C		17	203	11SEP81	203	11SEP81	220	6OCT81	220	6OCT81	0
* 4.4.7	Function D		17	220	6OCT81	220	6OCT81	237	29OCT81	237	29OCT81	0
* 4.4.8	Function E		17	237	29OCT81	237	29OCT81	254	23NOV81	254	23NOV81	0
* 4.4.9	Function F		17	254	23NOV81	254	23NOV81	271	18DEC81	271	18DEC81	0
4.5	Special Tasks		0	139	10JUN81	271	18DEC81	139	10JUN81	271	18DEC81	0
4.5.1	Support software		12	133	2JUN81	265	10DEC81	139	10JUN81	271	18DEC81	132
4.5.2	Other		6	133	2JUN81	268	15DEC81	136	5JUN81	271	18DEC81	132
* 4.6	Acceptance Readiness Rvw		2	271	18DEC81	271	18DEC81	273	22DEC81	273	22DEC81	135
*5.	SW Test and Transfer		0	312	18MAR82	332	18MAR82	332	18MAR82	332	18MAR82	0
* 5.1	Verification tests		28	273	22DEC81	273	22DEC81	299	1FEB82	299	1FEB82	0
5.2	Contingency		25	96	9APR81	314	22FEB82	109	28APR81	327	11MAR82	218
5.3	STT		0	313	19FEB82	332	18MAR82	313	19FEB82	332	18MAR82	19
5.3.1	Write all sections		14	133	2JUN81	292	21JAN82	140	11JUN81	299	1FEB82	159
5.3.2	Edit and release		2	312	18FEB82	326	10MAR82	313	19FEB82	327	11MAR82	14
* 5.4	Acceptance tests		20	299	1FEB82	299	1FEB82	312	18FEB82	312	18FEB82	0
* 5.5	Demonstration tests		22	312	18FEB82	312	18FEB82	327	11MAR82	327	11MAR82	0
* 5.6	Transfer, CDE to COE		7	327	11MAR82	327	11MAR82	332	18MAR82	332	18MAR82	0
6.	Mgt Tasks and Milestones		0	19	16DEC80	332	18MAR82	19	16DEC80	332	18MAR82	313
6.1	CDE Activities		37	0	17NOV80	313	19FEB82	19	16DEC80	332	18MAR82	313
* 6.2	Develop prelim budget		3	10	3DEC80	10	3DEC80	12	5DEC80	12	5DEC80	0
* 6.3	Develop Sys Impl Plan		3	27	31DEC80	27	31DEC80	29	6JAN81	29	6JAN81	0
* 6.4	Draft Software Impl Plan		6	50	4FEB81	50	4FEB81	55	11FEB81	55	11FEB81	0
* 6.5	Revise Impl Plan		12	81	19MAR81	81	19MAR81	90	1APR81	90	1APR81	0
6.6	QA Audit		26	312	18FEB82	318	26FEB82	326	10MAR82	332	18MAR82	6
*FINISH			0	332	18MAR82	332	18MAR82	332	18MAR82	332	18MAR82	0

WBS Version 1.2 30OCT80 PAGE 1

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

	TASK	1980 NOV DEC	1981 JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC	1982 JAN FEB MAR APR	MAN DYS	FINISH DATE
*0.	START	A			0	17NOV80
*1.	Sys Plans, Reqts, & Design	.A			0	9JAN81
* 1.1	Define Subsys Reqts	==A			15	3DEC80
* 1.2	FRD	.A			0	8DEC80
1.2.1	Write all sections	A=v			4	5DEC80
* 1.2.2	Edit and release FRD	=A			2	8DEC80
* 1.3	Level B Review	.A			3	10DEC80
* 1.4	Define Sys Architecture	.==A.			18	31DEC80
* 1.5	FDD	.A			0	7JAN81
1.5.1	Write all sections	.A=Sv			4	6JAN81
* 1.5.2	Edit and release	=A			2	7JAN81
* 1.6	Level C Review	.A			3	9JAN81
*2.	SW Planning and Reqts	A			0	16FEB81
* 2.1	Define Software Reqts	.==A			25	4FEB81
* 2.2	SRD	.A			0	12FEB81
2.2.1	Write all sections	.A==v			4	11FEB81
* 2.2.2	Edit and release	.A			2	12FEB81
* 2.3	Level D Review	.=A			2	16FEB81
*3.	SW Architecture and Design	A			0	6APR81
* 3.1	Define SW architecture	====A			31	19MAR81
* 3.2	SDD	A			0	2APR81
3.2.1	Write all sections	A====Sv			4	1APR81
* 3.2.2	Edit and release	A			2	2APR81
3.3	System Interface Design	==A=S=v			22	1APR81
* 3.4	Level E Review	A			2	6APR81
*4.	SW Detailed Design & Prod		A.		0	22DEC81
4.1	SSD			A=v	0	11MAR82
4.1.1	Write Sections 1,2,3	=A=========v			6	18DEC81
* 4.1.2	Write Section 4	===A.			18	24APR81
4.1.3	Write Section 5	====A=========S===v			43	18DEC81
4.1.4	Write Section 6	=A=========v			4	18DEC81
4.1.5	Write Section 7	=A=========Sv			9	18DEC81
4.1.6	Edit and release			=A=v	6	11MAR82
4.2	SOM			A=v	0	11MAR82
* 4.2.1	Write preliminary draft	=A			8	4MAY81
4.2.2	Complete all sections	=A=========Sv			9	18DEC81
4.2.3	Edit and release			=A=v	4	11MAR82
* 4.3	High-level Design Review	=A			2	2JUN81
* 4.4	Module Production & Integr		A		0	18DEC81
* 4.4.1	Executive and control	===A.			18	29MAY81
* 4.4.2	I/O Modules	===A.			18	26JUN81
* 4.4.3	Interface handlers	====A.			18	23JUL81

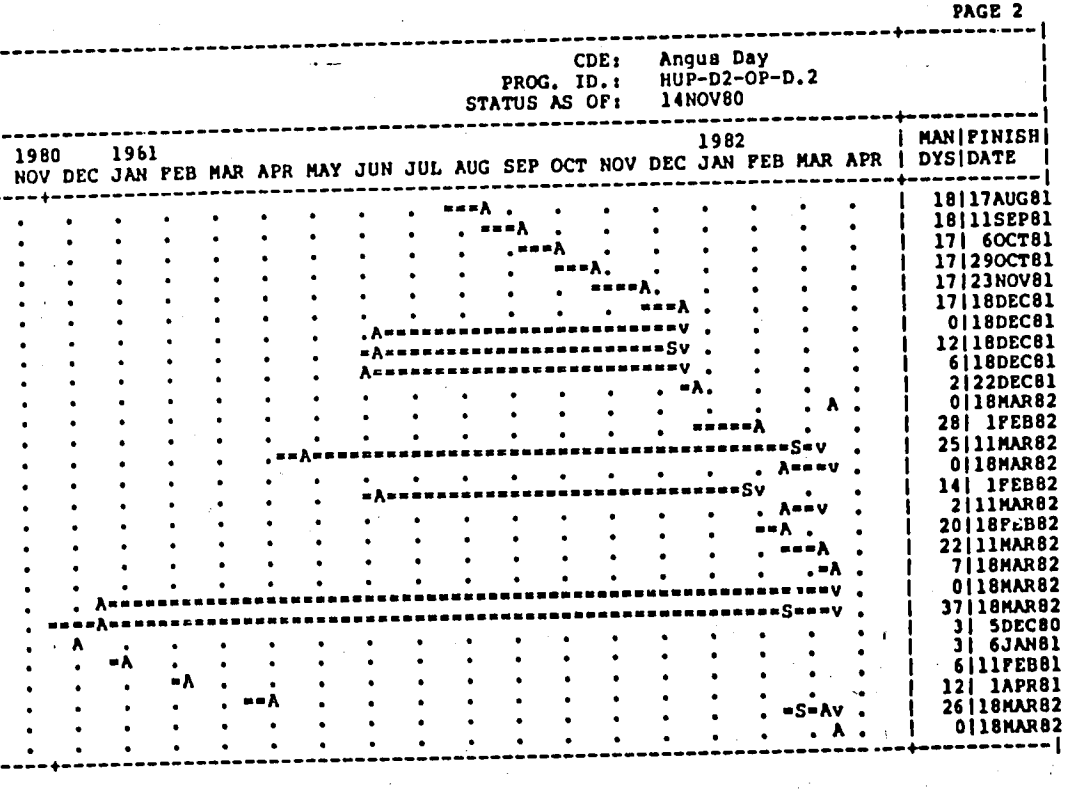
LEGEND: =-ACTIVITY A-EARLY FINISH
S-LATE START *v-LATE FINISH*
<-EVENTS PAST V-ACTUAL FINISH *-CRITICAL PATH ITEM

NOTE: ABSENCE OF $, S, A, OR v INDICATES MERGED DATES

WBS Version 1.2 30OCT80

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80



	TASK	MAN DYS	FINISH DATE
* 4.4.4	Function A	18	17AUG81
* 4.4.5	Function B	18	11SEP81
* 4.4.6	Function C	17	6OCT81
* 4.4.7	Function D	17	29OCT81
* 4.4.8	Function E	17	23NOV81
* 4.4.9	Function F	17	18DEC81
4.5	Special Tasks	0	18DEC81
4.5.1	Support software	12	18DEC81
4.5.2	Other	6	18DEC81
* 4.6	Acceptance Readiness Rvw	2	22DEC81
*5.	SW Test and Transfer	0	18MAR82
* 5.1	Verification tests	28	1FEB82
5.2	Contingency	25	11MAR82
5.3	STT	0	18MAR82
5.3.1	Write all sections	14	1FEB82
5.3.2	Edit and release	2	11MAR82
* 5.4	Acceptance tests	20	18FEB82
* 5.5	Demonstration tests	22	11MAR82
* 5.6	Transfer, CDE to COE	7	18MAR82
6.	Mgt Tasks and Milestones	0	18MAR82
6.1	CDE Activities	37	18MAR82
* 6.2	Develop prelim budget	3	5DEC80
* 6.3	Develop Sys Impl Plan	3	6JAN81
* 6.4	Draft Software Impl Plan	6	11FEB81
* 6.5	Revise Impl Plan	12	1APR81
6.6	QA Audit	26	18MAR82
*FINISH		0	18MAR82

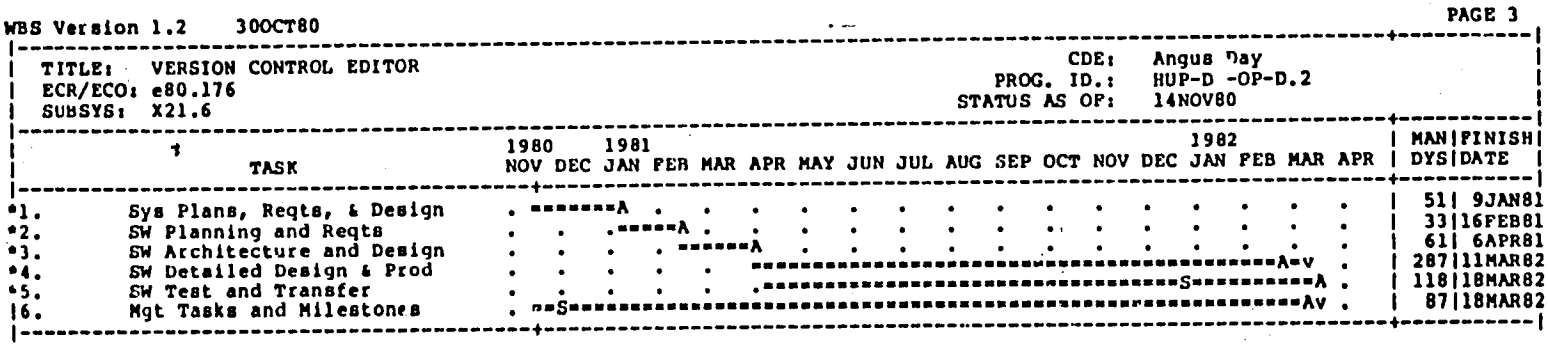
LEGEND: =-ACTIVITY A-EARLY FINISH
S-LATE START v-LATE FINISH
<-EVENTS PAST V-ACTUAL FINISH *-CRITICAL PATH ITEM

NOTE: ABSENCE OF $, S, A, OR v INDICATES MERGED DATES

WBS Version 1.2 30OCT80

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D -OP-D.2
STATUS AS OF: 14NOV80



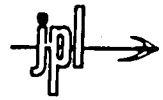
LEGEND: =-ACTIVITY A-EARLY FINISH
S-LATE START v-LATE FINISH
<-EVENTS PAST V-ACTUAL FINISH *-CRITICAL PATH ITEM

NOTE: ABSENCE OF $, S, A, OR v INDICATES MERGED DATES

THE VIEWGRAPH MATERIALS
for the
R. TAUSWORTHE PRESENTATION FOLLOW

DEEP-SPACE NETWORK SOFTWARE COST AND RESOURCE ESTIMATION MODEL PROTOTYPE

Robert C. Tausworthe



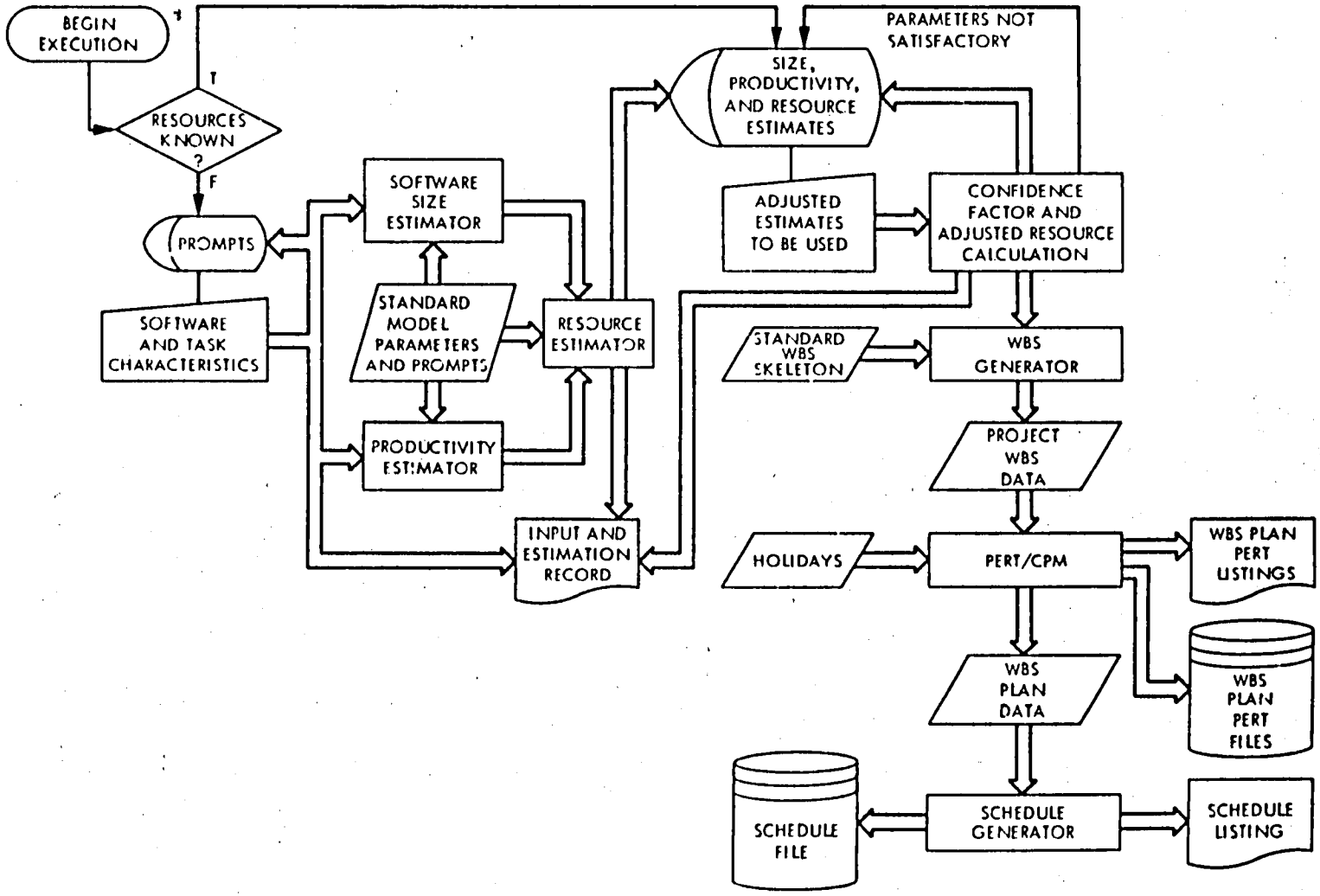
DSN SOFTWARE COST ESTIMATION MODEL

- COMBINES AND REFINES A NUMBER OF MODELS
- USES ABOUT 60 SIZING AND PRODUCTIVITY FACTORS TO PREDICT RESOURCES, DURATION, STAFFING, COMPUTER CPU, AND DOCUMENTATION REQUIREMENTS
- COMPUTES CONFIDENCE FACTOR FOR DELIVERING ON-TIME AND WITHIN BUDGET FOR ANY PROPOSED EFFORT AND DURATION
- PRODUCES SCHEDULE AND RESOURCE ESTIMATES FOR ABOUT 50 INDIVIDUAL TASKS MAKING UP A PROJECT
- IS SPECIALIZED TO OUTPUT A STANDARD WORK BREAKDOWN STRUCTURE (WBS) FOR DSN DATA SYSTEMS TASKS
- PERMITS ALL MODEL PARAMETERS TO BE CHANGED BY SIMPLE EDITING PROCESS

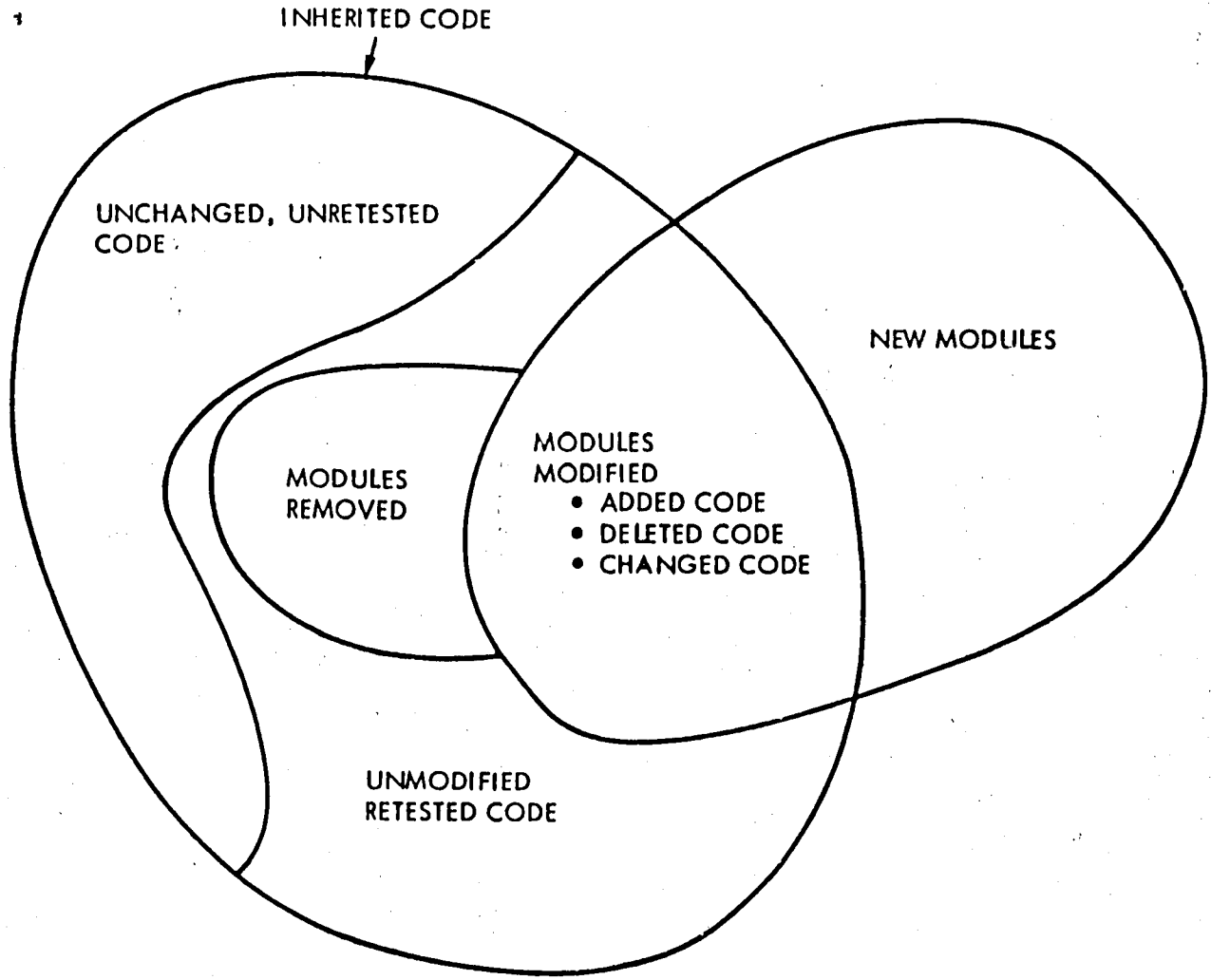
PARAMETER VALUES
PRINCIPAL SOURCES

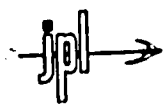
- IBM DATA (WALSTON-FELIX)
- RADC DATA BASE (NELSON)
- UNIVERSITY OF MARYLAND (FREBERGER-BASILI)
- GENERAL RESEARCH CORP (CARRIERE-THIBODEAU)
- TRW (WOLVERTON)
- QUANTITATIVE SOFTWARE MANAGEMENT, INC. (PUTNAM)
- JPL WBS ARCHIVE DATA

DATA FLOW DIAGRAM



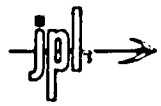
CODE SIZING MODEL





PRODUCTIVITY FACTORS
INPUTS

- CODE AND DATA BASE COMPLEXITY
- CODING LANGUAGE AND DEVELOPMENT TOOLS
- STAFF EXPERTISE AND EXPERIENCE
- REQUIREMENTS COMPLEXITY AND STABILITY
- USE OF MODERN PROGRAMMING TECHNIQUES
- SPONSOR AND USER INVOLVEMENT
- QUALITY AND MATURITY OF SYSTEM AND SUPPORT SOFTWARE
- ADVERSE DESIGN CONSTRAINTS
- ORGANIZATION AND WORK ENVIRONMENT
- HARDWARE DEVELOPMENT CONCURRENCY
- REAL-TIME VS NON-REAL-TIME PROCESSING



OUTPUTS

- RESOURCE ESTIMATES AND DEVIATIONS
- CONFIDENCE FACTORS FOR SUCCESS
- SCALED STANDARD WBS SOURCE FILE
- PERT-PROCESSED TASK LISTING
- GANTT (SCHEDULE) CHART

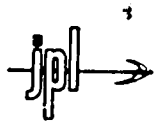
BASIC MODEL CALCULATIONS

EFFORT, STAFF MONTHS	$W = L^a/P_1$
PRODUCTIVITY, KSLEC/STAFF-MONTH	$P = P_1 L^{1-a}$
PRODUCTIVITY FACTOR	$P_1 = P_0 A_1 .. A_n$
TIME DURATION, MONTHS	$T = T_1 W^b$
AVERAGE STAFF, PERSONS	$S = W/T = W^{1-b}/T_1$
CPU TIME, HOURS	$C = C_1 L^c$
DOCUMENTATION, PAGES	$D = D_1 L^d$

WHERE L IS COMPUTED FROM SIZE MODEL

A_i ARE COMPUTED FROM PRODUCTIVITY FACTORS

P_0, T_1, C_1, D_1 ARE PRESET FROM MEASURED STATISTICS



CONFIDENCE CALCULATIONS

RAYLEIGH-NORDEN-PUTNAM FORM: $\dot{w} = At^r (K - w)$

SOFTWARE EQUATION: $\bar{L} = c_p w^p t^q$

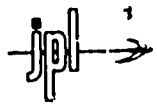
CONF (L, W, T) = $Pr \{\lambda \leq L, w \leq W, t \leq T\}$

ASSUMES:

(1) p, q, r CHOSEN SO SOFTWARE EQUATION FITS POWER LAW FORMULAS

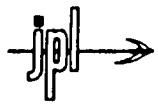
(2) q/p SET TO GIVE OBSERVED TIME-EFFORT TRADEOFF IN DSN

(3) $pr \{t \mid w, \lambda\}$, $pr \{w \mid \lambda\}$, and $pr \{\lambda\}$ are LOG-NORMAL DENSITIES



USAGE

- PLANNING - CURRENTLY BEING USED TO ESTIMATE RESOURCE NEEDS OF ALL SOFTWARE TASKS IN DEEP SPACE NETWORK CONSOLIDATION PROJECT
- PROJECT CONTROL SYSTEM - INTEGRATION WITH WBS RESOURCE ACCOUNTING AND REPORTING SYSTEM CAN REDUCE CONTROL SYSTEM OVERHEAD
- TECHNOLOGY EVALUATION - MODEL CAN POINT OUT WHERE RESOURCES ARE ACTUALLY GOING, AND WHERE ADVANCED SOFTWARE TECHNOLOGY IS NEEDED, AND WHAT BENEFITS FOR THAT TECHNOLOGY WILL BE

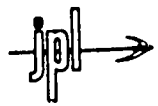


CURRENT STATUS

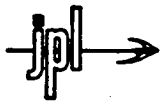
- PROTOTYPE IMPLEMENTED ON 8080-CP/M MICRO-COMPUTER FOR EVALUATION
- TECHNICAL REPORT NEAR COMPLETION
- SUMMARY ARTICLE AVAILABLE
- APPLICATIONS MANUAL BEING WRITTEN

FUTURE PLANS

- MODEL CALIBRATION USING PERFORMANCE DATA FROM PROJECTS NOW IN PROGRESS
- TRANSLATION AND INTEGRATION INTO PROJECT CONTROL SYSTEM
- MODIFICATION TO ACCOMMODATE PROPOSED JPL SOFTWARE STANDARD STANDARD LIFE CYCLE AND PRACTICES
- MODEL EXTENSION AND REFINEMENT TO INCORPORATE HUMAN ACTIVITY MODELS
- UTILIZATION TO FORECAST TECHNOLOGY IMPACTS ON SOFTWARE PRODUCTIVITY



TYPICAL MODEL USAGE EXAMPLE

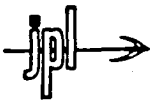


TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

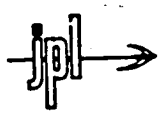
CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
Date Estimated: 14NOV80
Model Data Version 1.3 31OCT80

Answer the following items to the best of your estimation.

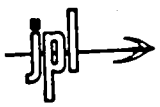
1.	How much new code is to be produced (completely new modules)?	
	Maximum value, kilo-lines executable source(99% confidence level)?	3.5
	Expected value, kilo-lines executable source?	3.3
	Minimum value, kilo-lines executable source(99% confidence level)?	3.1
2.	How much code exists in modules requiring modification?	
	Maximum value, kilo-lines executable source(99% confidence level)?	6.9
	Expected value, kilo-lines executable source?	6.6
	Minimum value, kilo-lines executable source(99% confidence level)?	6.3
3.	How much code will be deleted from these existing modules?	
	Maximum value, kilo-lines executable source(99% confidence level)?	.4
	Expected value, kilo-lines executable source?	.3
	Minimum value, kilo-lines executable source(99% confidence level)?	.2
4.	How much code will be added to these existing modules?	
	Maximum value, kilo-lines executable source(99% confidence level)?	.7
	Expected value, kilo-lines executable source?	.6
	Minimum value, kilo-lines executable source(99% confidence level)?	.4
5.	How much code will be changed in other ways in these modules?	
	Maximum value, kilo-lines executable source(99% confidence level)?	1.2
	Expected value, kilo-lines executable source?	.9
	Minimum value, kilo-lines executable source(99% confidence level)?	.7
6.	How much code will be deleted as entire modules from existing code?	
	Maximum value, kilo-lines executable source(99% confidence level)?	1.4
	Expected value, kilo-lines executable source?	1.3
	Minimum value, kilo-lines executable source(99% confidence level)?	1.1
7.	How much of the remaining existing code must be retested?	
	Maximum value, kilo-lines executable source(99% confidence level)?	2.1
	Expected value, kilo-lines executable source?	1.9
	Minimum value, kilo-lines executable source(99% confidence level)?	1.5
8.	Expected percentage of code to be developed actually delivered (0-90, 91-99, 100)?	91-99
9.	How many different kinds of input/output data items per 1000 lines of new or modified code(>80, 16-80, 0-15)?	16-80
10.	Overall complexity of program and data base architecture (high, medium, low)?	MEDIUM
11.	Complexity of code logical design(high, medium, low)?	LOW
12.	What percent of the programming task is in Assembly language?	9
13.	What percent of the new or modified code must be storage-optimized?	9



14. What percent of the new or modified code must be timing-optimized?	9
15. What percent of the total programming task is 'easy'?	20
16. What percent of the total programming task is 'hard'?	30
17. When is work to start, on the(FRD/FDD, SRD, SDD)?	FRD/FDD
18. What percent of the total program requirments will be established and stable before design, and will not be altered before delivery?	80
19. What percent of the requirements are likely to change slightly before delivery, but will do so under baseline change control?	10
20. What percent of the requirements are likely to change more drastically before delivery, but will do so under baseline control?	5
21. Complexity of program functional requirements(high, medium, low)?	LOW
22. Expected user involvement in requirements definition (much, some, none)?	MUCH
23. Customer experience in application area(much, none, some)?	SOME
24. Customer/implementor organizational interface complexity (high, normal, low)?	NORMAL
25. Interfaces with other SW development projects or organizations (many, few, none)?	FEW
26. Efficiency of implementing organization(poor, ok, good)?	GOOD
27. Overall implementation personnel qualifications and motivation (low, average, high)?	HIGH
28. Percentage of programmers doing functional design who will also be doing development(<25, 25-50, >50)?	25-50
29. Previous programmer experience with application of similar or greater size and complexity(minimal, average, extensive)?	AVERAGE
30. What is the average staff experience, in years, obtained from work similar to that required in the task being estimated?	6
31. Previous experience with operational computer to be used (minimal, average, extensive)?	MINIMAL
32. Previous experience with programming language(s) to be used (minimal, average, extensive)?	MINIMAL
33. Use of top-down methodology(low, medium, high)?	HIGH
34. Use of structured programmer team concepts(low, medium, high)?	HIGH
35. Use of Structured Programming(low, medium, high)?	HIGH



36. Use of design and code inspections(low, QA, peer)?	QA
37. Classified security environment for computer(yes, , no)?	NO
38. Hardware under concurrent development(much, some, none)?	NONE
39. Percent of work done at primary development site (<70, 70-90, >90)?	70-90
40. Development computer access mode(remote, scheduled, demand)?	DEMAND
41. Percent of development computer access availability(<30, 30-60, >60)?	30-60
42. Quality of SW development tools and environment(poor, ok, good)?	OK
43. Maturity of system and support software(buggy, ok, good)?	OK
44. Overall adverse constraints on program design (severe, average, minimal)?	MINIMAL
45. Is the program real-time, multi-task(chiefly, some, no)?	NO
46. SW to be adaptable to multiple computer configurations or environments (yes, , no)?	NO
47. Adaptation required to change from development to operational environment(much, some, minimal)?	MINIMAL



Estimated Overall Parameters:

=average value

	+1-sigma	-1-sigma
Adjusted Lines of code= 6182 SLEC		
6280	6085	
Effort= 26.5 person-months		
45.8	15.3	
Staff productivity= 233 SLEC/staff-month		
404	135	
Duration= 15.7 months		
19.0	12.9	
Avg. Staff= 1.7		
2.9	1.0	
Documentation= 537 pages	$16.1K	
645	$19.4K	$13.4K
448		
Computer CPU time= 319 hours	$0.0K	
478	$0.0K	$0.0K
212		

Use these figures to arrive at Effort, Duration, and Staffing requirements. Include factors to provide acceptible risk and confidence levels.

Values specified are:

Kilo-lines of code=	6.18
Effort (person-months):	32.0
Duration (months):	16.0
Average staff (persons):	2.0

For the numbers you have entered, a reasonableness check indicates that the average project would produce 7303 lines of code, using 32 staff-months of resources and 16 months of duration, with an average staff of 2 persons, for a productivity of 228 SLEC/staff-month.

The level of confidence in delivering 6182 lines of code, on-time and within resources= 33 %.

Is output to be saved in a file? Y

Name of output file to be created: VCEDIT

Schedule start date: 17NOV80

Select desired outputs and output media, or enter RETURN only for defaults. Defaults are 1A, 2A, and 3A. Choices are:

1=Gantt Chart	A=file
2=PERT data, 132 width	B=line printer
3=PERT data, 80 width	

Choice(s): 1B,2B,3A

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

CODE	TASK	WHO	EFF ORT	START EARLY DAY	START EARLY DATE	START LATE DAY	START LATE DATE	FINISH EARLY DAY	FINISH EARLY DATE	FINISH LATE DAY	FINISH LATE DATE	FLOAT TIME DAYS
*0.	START		0	0	17NOV80	0	17NOV80	0	17NOV80	0	17NOV80	0
*1.	Sys Plans, Reqts, & Design		0	32	9JAN81	32	9JAN81	32	9JAN81	32	9JAN81	0
* 1.1	Define Subsys Reqts		15	0	17NOV80	0	17NOV80	10	3DEC80	10	3DEC80	0
* 1.2	FRD		0	13	8DEC80	13	8DEC80	13	8DEC80	13	8DEC80	0
| 1.2.1	Write all sections		4	0	17NOV80	9	2DEC80	3	20NOV80	12	5DEC80	9
* 1.2.2	Edit and release FRD		2	12	5DEC80	12	5DEC80	13	8DEC80	13	8DEC80	0
* 1.3	Level B Review		3	13	8DEC80	13	8DEC80	15	10DEC80	15	10DEC80	0
* 1.4	Define Sys Architecture		18	15	10DEC80	15	10DEC80	27	31DEC80	27	31DEC80	0
* 1.5	FDD		0	30	7JAN81	30	7JAN81	30	7JAN81	30	7JAN81	0
| 1.5.1	Write all sections		4	15	10DEC80	27	31DEC80	17	12DEC80	29	6JAN81	12
* 1.5.2	Edit and release		2	29	6JAN81	29	6JAN81	30	7JAN81	30	7JAN81	0
* 1.6	Level C Review		3	30	7JAN81	30	7JAN81	32	9JAN81	32	9JAN81	0
*2.	SW Planning and Reqts		0	58	16FEB81	58	16FEB81	58	16FEB81	58	16FEB81	0
* 2.1	Define Software Reqts		25	32	9JAN81	32	9JAN81	50	4FEB81	50	4FEB81	0
* 2.2	SRD		0	56	12FEB81	56	12FEB81	56	12FEB81	56	12FEB81	0
| 2.2.1	Write all sections		4	32	9JAN81	53	9FEB81	34	13JAN81	55	11FEB81	21
* 2.2.2	Edit and release		2	55	11FEB81	55	11FEB81	56	12FEB81	56	12FEB81	0
* 2.3	Level D Review		2	56	12FEB81	56	12FEB81	58	16FEB81	58	16FEB81	0
*3.	SW Architecture and Design		0	93	6APR81	93	6APR81	93	6APR81	93	6APR81	0
* 3.1	Define SW architecture		31	58	16FEB81	58	16FEB81	81	19MAR81	81	19MAR81	0
* 3.2	SDD		0	91	2APR81	91	2APR81	91	2APR81	91	2APR81	0
| 3.2.1	Write all sections		4	58	16FEB81	88	30MAR81	60	18FEB81	90	1APR81	30
* 3.2.2	Edit and release		2	90	1APR81	90	1APR81	91	2APR81	91	2APR81	0
| 3.3	System Interface Design		22	58	16FEB81	78	16MAR81	70	4MAR81	90	1APR81	20
* 3.4	Level E Review		2	91	2APR81	91	2APR81	93	6APR81	93	6APR81	0
*4.	SW Detailed Design & Prod		0	273	22DEC81	273	22DEC81	273	22DEC81	273	22DEC81	0
| 4.1	SSD		0	315	23FEB82	327	11MAR82	315	23FEB82	327	11MAR82	12
| 4.1.1	Write Sections 1,2,3		6	93	6APR81	268	15DEC81	96	9APR81	271	18DEC81	175
* 4.1.2	Write Section 4		18	93	6APR81	93	6APR81	107	24APR81	107	24APR81	0
| 4.1.3	Write Section 5		43	107	24APR81	249	16NOV81	129	27MAY81	271	18DEC81	142
| 4.1.4	Write Section 6		4	93	6APR81	269	16DEC81	95	8APR81	271	18DEC81	176
| 4.1.5	Write Section 7		9	93	6APR81	266	11DEC81	98	13APR81	271	18DEC81	173
| 4.1.6	Edit and release		6	312	18FEB82	324	8MAR82	315	23FEB82	327	11MAR82	12
| 4.2	SOM		0	314	22FEB82	327	11MAR82	314	22FEB82	327	11MAR82	13
* 4.2.1	Write preliminary draft		8	107	24APR81	107	24APR81	113	4MAY81	113	4MAY81	0
| 4.2.2	Complete all sections		9	133	2JUN81	266	11DEC81	138	9JUN81	271	18DEC81	133
| 4.2.3	Edit and release		4	312	18FEB82	325	9MAR82	314	22FEB82	327	11MAR82	13
* 4.3	High-level Design Review		2	131	29MAY81	131	29MAY81	133	2JUN81	133	2JUN81	0
* 4.4	Module Production & Integ		0	271	18DEC81	271	18DEC81	271	18DEC81	271	18DEC81	0
* 4.4.1	Executive and control		18	113	4MAY81	113	4MAY81	131	29MAY81	131	29MAY81	0
* 4.4.2	I/O Modules		18	133	2JUN81	133	2JUN81	151	26JUN81	151	26JUN81	0
* 4.4.3	Interface handlers		18	151	26JUN81	151	26JUN81	169	23JUL81	169	23JUL81	0

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

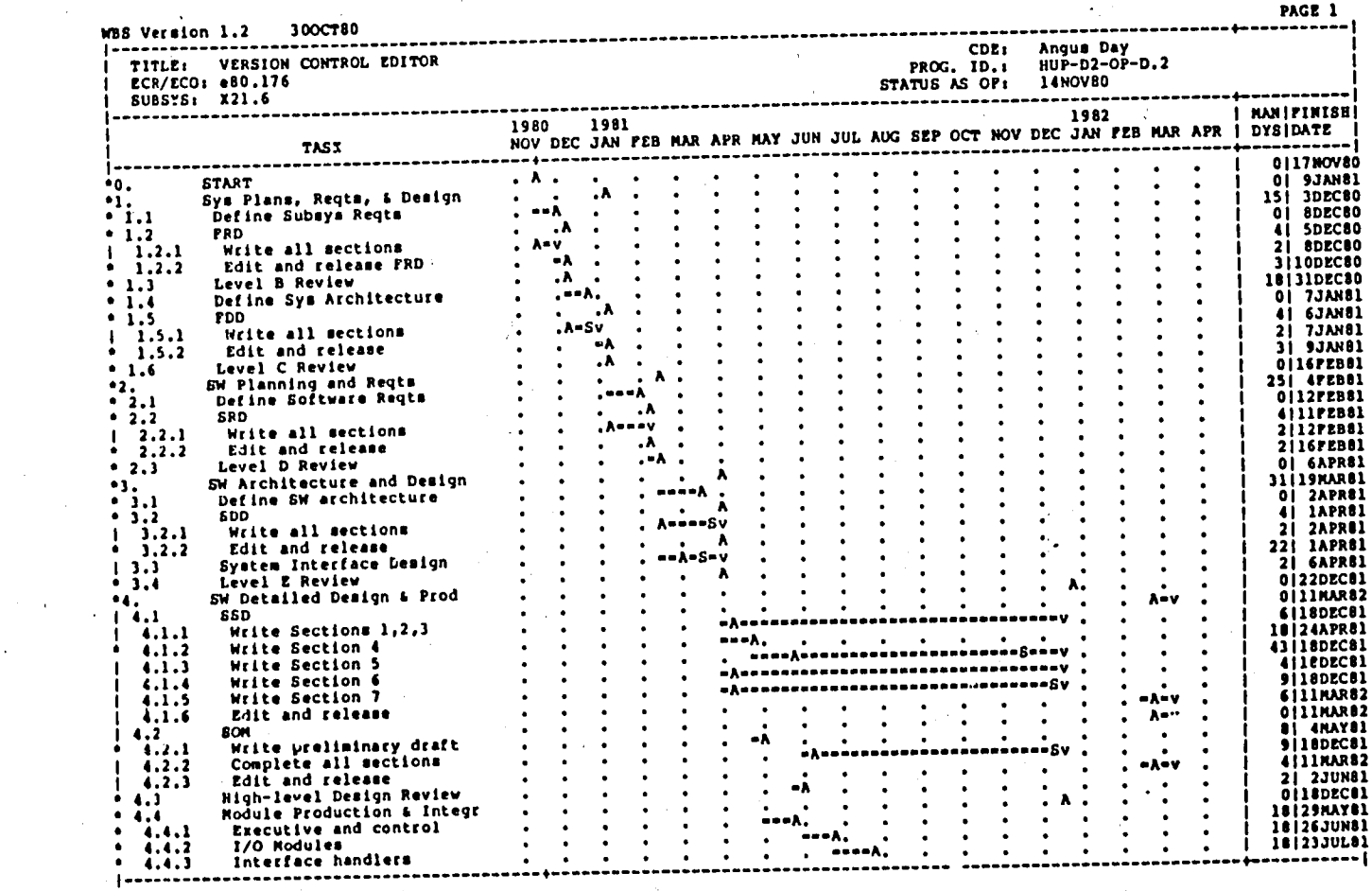
CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

CODE	TASK	WHO	EFF ORT	START EARLY DAY	START EARLY DATE	START LATE DAY	START LATE DATE	FINISH EARLY DAY	FINISH EARLY DATE	FINISH LATE DAY	FINISH LATE DATE	FLOAT TIME DAYS
* 4.4.4	Function A		18	169	23JUL81	169	23JUL81	186	17AUG81	186	17AUG81	0
* 4.4.5	Function B		18	186	17AUG81	186	17AUG81	203	11SEP81	203	11SEP81	0
* 4.4.6	Function C		17	203	11SEP81	203	11SEP81	220	6OCT81	220	6OCT81	0
* 4.4.7	Function D		17	220	6OCT81	220	6OCT81	237	29OCT81	237	29OCT81	0
* 4.4.8	Function E		17	237	29OCT81	237	29OCT81	254	23NOV81	254	23NOV81	0
* 4.4.9	Function F		17	254	23NOV81	254	23NOV81	271	18DEC81	271	18DEC81	0
4.5	Special Tasks		0	139	10JUN81	271	18DEC81	139	10JUN81	271	18DEC81	132
4.5.1	Support software		12	133	2JUN81	265	10DEC81	139	10JUN81	271	18DEC81	132
4.5.2	Other		6	133	2JUN81	268	15DEC81	136	5JUN81	271	18DEC81	135
* 4.6	Acceptance Readiness Rvw		2	271	18DEC81	271	18DEC81	273	22DEC81	273	22DEC81	0
*5.	SW Test and Transfer		0	332	18MAR82	332	18MAR82	332	18MAR82	332	18MAR82	0
* 5.1	Verification tests		28	273	22DEC81	273	22DEC81	299	1FEB82	299	1FEB82	0
5.2	Contingency		25	96	9APR81	314	22FEB82	109	28APR81	327	11MAR82	218
5.3	STT		0	313	19FEB82	332	18MAR82	313	19FEB82	332	18MAR82	19
5.3.1	Write all sections		14	133	2JUN81	292	21JAN82	140	11JUN81	299	1FEB82	159
5.3.2	Edit and release		2	312	18FEB82	326	10MAR82	313	19FEB82	327	11MAR82	14
* 5.4	Acceptance tests		20	299	1FEB82	299	1FEB82	312	18FEB82	312	18FEB82	0
* 5.5	Demonstration tests		22	312	18FEB82	312	18FEB82	327	11MAR82	327	11MAR82	0
* 5.6	Transfer, CDE to COE		7	327	11MAR82	327	11MAR82	332	18MAR82	332	18MAR82	0
6.	Mgt Tasks and Milestones		0	19	16DEC80	332	18MAR82	19	16DEC80	332	18MAR82	313
6.1	CDE Activities		37	0	17NOV80	313	19FEB82	19	16DEC80	332	18MAR82	313
* 6.2	Develop prelim budget		3	10	3DEC80	10	3DEC80	12	5DEC80	12	5DEC80	0
* 6.3	Develop Sys Impl Plan		3	27	31DEC80	27	31DEC80	29	6JAN81	29	6JAN81	0
* 6.4	Draft Software Impl Plan		6	50	4FEB81	50	4FEB81	55	11FEB81	55	11FEB81	0
* 6.5	Revise Impl Plan		12	81	19MAR81	81	19MAR81	90	1APR81	90	1APR81	0
6.6	QA Audit		26	312	18FEB82	318	26FEB82	326	10MAR82	332	18MAR82	6
*FINISH			0	332	18MAR82	332	18MAR82	332	18MAR82	332	18MAR82	0

WBS Version 1.2 30OCT80

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

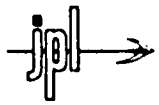
CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80



	TASK	MAN DYS	FINISH DATE
*0.	START	0	17NOV80
*1.	Sys Plans, Reqts, & Design	0	9JAN81
* 1.1	Define Subsys Reqts	15	3DEC80
* 1.2	FRD	0	8DEC80
1.2.1	Write all sections	4	5DEC80
* 1.2.2	Edit and release FRD	2	8DEC80
* 1.3	Level B Review	3	10DEC80
* 1.4	Define Sys Architecture	18	31DEC80
* 1.5	FDD	0	7JAN81
1.5.1	Write all sections	4	6JAN81
* 1.5.2	Edit and release	2	7JAN81
* 1.6	Level C Review	3	9JAN81
*2.	SW Planning and Reqts	0	16FEB81
* 2.1	Define Software Reqts	25	4FEB81
* 2.2	SRD	0	12FEB81
2.2.1	Write all sections	4	11FEB81
* 2.2.2	Edit and release	2	12FEB81
* 2.3	Level D Review	2	16FEB81
*3.	SW Architecture and Design	0	6APR81
* 3.1	Define SW architecture	31	19MAR81
* 3.2	SDD	0	2APR81
3.2.1	Write all sections	4	1APR81
* 3.2.2	Edit and release	2	2APR81
3.3	System Interface Design	22	1APR81
* 3.4	Level E Review	2	6APR81
*4.	SW Detailed Design & Prod	0	22DEC81
4.1	SSD	0	11MAR82
4.1.1	Write Sections 1,2,3	6	18DEC81
* 4.1.2	Write Section 4	18	24APR81
4.1.3	Write Section 5	43	18DEC81
4.1.4	Write Section 6	4	18DEC81
4.1.5	Write Section 7	9	18DEC81
4.1.6	Edit and release	6	11MAR82
4.2	SOM	0	11MAR82
* 4.2.1	Write preliminary draft	8	4MAY81
4.2.2	Complete all sections	9	18DEC81
4.2.3	Edit and release	4	11MAR82
* 4.3	High-level Design Review	2	2JUN81
* 4.4	Module Production & Integr	0	18DEC81
* 4.4.1	Executive and control	18	29MAY81
* 4.4.2	I/O Modules	18	26JUN81
* 4.4.3	Interface handlers	18	23JUL81

LEGEND: --ACTIVITY A-EARLY FINISH
S-LATE START v-LATE FINISH
<-EVENTS PAST V-ACTUAL FINISH *-CRITICAL PATH ITEM

NOTE: ABSENCE OF $, S, A, OR v INDICATES MERGED DATES



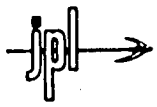
WBS Version 1.2 30OCT80 PAGE 2

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e20.176
SUBSYS: X21.6
CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

	TASK	1980 NOV	DEC	1981 JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	1982 JAN	FEB	MAR	APR	MAN DYS	FINISH DATE
* 4.4.4	Function A	.	.	.	.	.	.	.	.	.	===A	.	.	.	.	.	.	.	.	18	17AUG81
* 4.4.5	Function B	.	.	.	.	.	.	.	.	.	.	===A	.	.	.	.	.	.	.	18	11SEP81
* 4.4.6	Function C	.	.	.	.	.	.	.	.	.	.	.===A	.	.	.	.	.	.	.	17	6OCT81
* 4.4.7	Function D	.	.	.	.	.	.	.	.	.	.	.	===A.	.	.	.	.	.	.	17	29OCT81
* 4.4.8	Function E	.	.	.	.	.	.	.	.	.	.	.	.	====A.	.	.	.	.	.	17	23NOV81
* 4.4.9	Function F	.	.	.	.	.	.	.	.	.	.	.	.	.	===A	.	.	.	.	17	18DEC81
4.5	Special Tasks	.	.	.	.	.	.	.	.A	==========	==========	==========	==========	==========	==========	==========	==========	==========	v	0	18DEC81
4.5.1	Support software	.	.	.	.	.	.	.	=A	==========	==========	==========	==========	==========	==========	==========	==========	==========	Sv	12	18DEC81
4.5.2	Other	.	.	.	.	.	.	.	A=	==========	==========	==========	==========	==========	==========	==========	==========	==========	v	6	18DEC81
* 4.6	Acceptance Readiness Rvw	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=A.	.	.	.	2	22DEC81
*5.	SW Test and Transfer	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	. A	0	18MAR82
* 5.1	Verification tests	.	.	.	.	.	.	.	.	.	.	.	.	.	.	====A	.	.	.	28	1FEB82
5.2	Contingency	.	.	.	.	.	.==A	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	S=v	.	25	11MAR82
5.3	STT	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	A===v	.	0	18MAR82
5.3.1	Write all sections	.	.	.	.	.	.	=A	==========	==========	==========	==========	==========	==========	==========	==========	Sv	.	.	14	1FEB82
5.3.2	Edit and release	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	A==v	.	2	11MAR82
* 5.4	Acceptance tests	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	==A	.	.	20	18FEB82
* 5.5	Demonstration tests	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	===A	.	22	11MAR82
* 5.6	Transfer, CDE to COE	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.=A	.	7	18MAR82
6.	Mgt Tasks and Milestones	.	. A	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	v	0	18MAR82
6.1	CDE Activities	.	===A	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	==========	S=v	.	37	18MAR82
* 6.2	Develop prelim budget	.	A	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	5DEC80
* 6.3	Develop Sys Impl Plan	.	.	=A	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	6JAN81
* 6.4	Draft Software Impl Plan	.	.	.	=A	.	.	.	.	.	.	.	.	.	.	.	.	.	.	6	11FEB81
* 6.5	Revise Impl Plan	.	.	.	.	==A	.	.	.	.	.	.	.	.	.	.	.	.	.	12	1APR81
6.6	QA Audit	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=S=Av	.	26	18MAR82
*FINISH		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	A	0	18MAR82

LEGEND: =-ACTIVITY A-EARLY FINISH
S-LATE START v-LATE FINISH
<-EVENTS PAST V-ACTUAL FINISH *-CRITICAL PATH ITEM

NOTE: ABSENCE OF $, S, A, OR v INDICATES MERGED DATES



WBS Version 1.2 30OCT80

TITLE: VERSION CONTROL EDITOR
ECR/ECO: e80.176
SUBSYS: X21.6

CDE: Angus Day
PROG. ID.: HUP-D2-OP-D.2
STATUS AS OF: 14NOV80

	TASK	1980 NOV DEC	1981 JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC	1982 JAN FEB MAR APR	MAN DYS	FINISH DATE
*1.	Sys Plans, Reqts, & Design	=======A	. . . . .	.	51	9JAN81
*2.	SW Planning and Reqts	.=====A	. . . . .	.	33	16FEB81
*3.	SW Architecture and Design	.=====A	. . . . .	.	61	6APR81
*4.	SW Detailed Design & Prod	.===========	==========A=v	.	287	11MAR82
*5.	SW Test and Transfer	.===========	==========S=========A	.	118	18MAR82
6.	Mgt Tasks and Milestones	==S==========	==========Av	.	87	18MAR82

LEGEND: =-ACTIVITY A-EARLY FINISH
S-LATE START v-LATE FINISH
<-EVENTS PAST V-ACTUAL FINISH *-CRITICAL PATH ITEM

NOTE: ABSENCE OF $, S, A, OR v INDICATES MERGED DATES

D7

N82 24004

SOFTWARE COST/RESOURCE MODELING:
Software Quality Tradeoff Measurement

R. W. Lawler
Boeing Aerospace Company

1.0 INTRODUCTION

In the early 70's the concept of software quality focused on the property of software reliability - freedom from errors. Subsequent research [McCall (1)] has identified eleven other software quality factors and has developed a system of metrics to predict/assess the degree of presence of the various qualities in software. The metrics have been validated for some quality factors. However, the concepts and validation have focused primarily on the software subsystem and have largely ignored some of the total system aspects such as the computer hardware, operating system, communications network, and other noncomputing elements of a total system. This paper develops a conceptual framework for treating software quality from a total system perspective. Examples are given to show how system quality objectives may be allocated to hardware and software; to illustrate trades among quality factors, both hardware and software, to achieve system performance objectives; and, to illustrate the impact of certain design choices on software functionality.

2.0 STATUS OF SOFTWARE QUALITY FRAMEWORK

Figure 2.0-1 summarizes the definitions of software quality developed by McCall. Criteria and metrics have been defined for many of these quality factors. McCall validated metrics for four quality factors, namely

reliability (design and implementation);
maintainability (design and implementation);
flexibility (implementation; and,
portability (implementation).

The sample size used for validation was too small to establish acceptable confidence limits. The significance of his work was that relationships between metrics do exist and correspond to intuition. There is ample need to perform similar validations for these and other quality factors.

3.0 THE SYSTEM QUALITY PERSPECTIVE FOR DISTRIBUTED SYSTEMS

The Software Quality Perspective can also be formulated from a total system point of view. That is, computer hardware, communications network, operating system, and even non computing elements are considered in a system perspective on Quality.

In fact a major problem that plagues the automated system design process is the lack of a framework for including software in the process of allocating system requirements and quality goals. A meaningful and practical software quality framework for distributed systems must address this problem. Three examples of this impact are:

(1) System quality factors may change the emphasis of a software quality factor, criterion, or metric by either increasing or decreasing its importance in the context of distributed systems. For example, software fault tolerance receives increased emphasis in a distributed system, whereas in a single-processor system the emphasis is usually placed on error recovery.

FACTORS	DEFINITION
CORRECTNESS	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
RELIABILITY	Extent to which a program can be expected to perform its intended function with required precision.
EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
USABILITY	Effort required to learn, operate, prepare input, and interpret output of a program.
MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
TESTABILITY	Effort required to test a program to incure it performs its intended function.
FLEXIBILITY	Effort required to modify an operational program.
PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
REUSABILITY	Extent to which a program can be used in other applications – related to the packaging and scope of the functions that programs perform.
INTEROPERABILITY	Effort required to couple one system with another.

Figure 2.0-1. Software Quality Factor Definitions

(2) System quality factors may require modification or additions to software quality factor concepts. For example, the system quality of survivability requires the addition of a software quality of survivability.

(3) Requirements for system quality may lead to increased software functionality. For example, the operating system may be required to incorporate dynamic reconfigurability to achieve system reliability requirements.

There is a need to define system Quality Factors and Criteria. Figure 3.0-1 shows a partial compilation of these Quality Factors and Criteria. Figure 3.0-2 proposes definitions for some of the system quality factors and criteria.

The figures show that

(1) there are additional quality factors for systems e.g. safety and availability for systems

(2) the system quality perspective adds a large number of new criteria, e.g. distributedness

(3) the definitions of system quality factors and criteria may differ from the software definitions, e.g. maintainability

There is a relationship between system quality factors and criteria and software quality factors and criteria. Figure 3.0-3 gives examples of this correspondence.

For example, the system quality factor of availability implies software quality factors of correctness and maintainability. High quality in these factors insures that the system will seldom fail due to a software error and if one occurs the fault will be corrected quickly. Availability in a distributed system also implies software reliability criteria of fault recovery and fault tolerance. These properties insure that the system continues to run if a processor or communication link fails, and that once repair is made the failed component rejoins the network quickly.

Some system quality factors correspond exactly to their software quality counterpart, e.g. integrity. But other quality factors do not. For example, system portability implies low power, light weight, and compact. These are not software qualities, but they imply that there are constraints on the software in the form of low power and low speed logic; limited facilities for data storage, need for firmware rather than software; limited facilities for data entry and display; and electromagnetic communication links. In this environment the quality factors of efficiency, usability, and integrity would be emphasized.

If a system is highly distributed with respect to the control logic of the software, testability of the software system and response time of the processing will be important considerations.

One of the major impacts of systen. quality factors is that they place requirements for increased functionality on the software. Figure 3.0-4 illustrates the impact of a set of system Quality factors on a distributed operating system. Collectively the eight factors require the addition of software in eight functional areas.

The considerations that are made in allocating a system quality factor to hardware, software, firmware, and operating system for a distributed system may differ from those for a single processor system. Figures 3.0-5 and 3.0-6 are examples of allocating system quality factors of efficiency and reliability to system components.

System Quality Factors	Criteria	
Availability	Distributedness	Completeness
Correctness	File Availability	Consistency
Maintainability	Remote Testability	Accuracy
Reliability	Extensibility	Error Tolerance
Flexibility	Parallelism	Simplicity
Testability	Tasking Efficiency	Modularity
Portability	Communications Efficiency	Generality
Usability	Multilevel Security	Expandability
Efficiency	Homogeneity	Instrumentation
Integrity	Virtuality	Self Descriptiveness
Interoperability	Endurance	Execution Efficiency
Survivability	Reconfigurability	Access Control
Reusability	Dispersion	Access Audit
Safety	Independence	Operability
	Graceful Degradation	Training
	Communicativeness	Conciseness
	Commonality	Warnings
	Damage Potential	

Figure 3.0-1. Candidate System Quality Factors and Criteria

System Factor or Criteria	Definition
Availability	1 - MTTR/MTBF where MTTR is Mean Time to Repair MTBF is Mean Time Between Failure
Survivability	Probably that a distributed system still performs essential functions after one or more nodes and communication links are destroyed or fail.
Distributedness	Degree to which communication connected processors, data bases, users, and/or processes are geographically or logically separated.
Maintainability	Mean Time to repair

Figure 3.0-2. Candidate Definitions of System Quality Factors and Criteria

System	Software	
Quality Factors/Criteria	Quality Factors	Criteria
Availability	Maintainability Correctness	Fault Tolerance Fault Recovery
Integrity	Integrity	No Change
Portability	Efficiency Usability Integrity	No Change
Distributiveness	Testability	Response Time

Figure 3.0–3. Correspondence of System and Software Quality. Factors and Criteria.

Heterogeneous Distributed System Quality Factors	Resource Monitoring/ Scheduling	Error Control & Reconfiguration	Access Control	Interprocessor Communication	Uniform Command Language	System Wide File System	Intercomponent Translations	Data Base Control
Survivability	X	X						
Usability					X	X		
Efficiency	X							
Interoperability				X		X	X	
Integrity			X					
Flexibility		X						
Expandability	X	X						
Reliability	X	X						X

Figure 3.0–4. System Quality Factors Impact on Operating System Functionality

	EFFICIENCY			
	HARDWARE	FIRMWARE	OP. SYSTEM	APP. S/W
SINGLE PROCESSOR	High Speed Components	Increase Speed	Priority Tasking	Optimized Code
DISTRIBUTED SYSTEM	Low Speed Components High-Speed Communication	Increase Speed	Synchronization	Exploitation of Parallelism

Figure 3.0-5. Allocation of Efficiency to System Components

	RELIABILITY			
	HARDWARE	FIRMWARE	OP. SYSTEM	APP. S/W
SINGLE PROCESSOR	High Reliability Parts	No Impact	Fault Recovery	No Impact
DISTRIBUTED SYSTEM	Low Reliability Parts Redundancy	No Impact	Fault Tolerance Fault Recovery	No Impact

Figure 3.0-6. Allocation of Reliability to System Components

This type of analysis highlights the difference between single processor systems and distributed systems with respect to system and software quality. It also identifies the trades among the system components.

4.0 MANAGING SOFTWARE QUALITY ACQUISITION

The system Quality perspective influences the activities of a software acquisition manager. There are several issues namely

(1) the issues that arise between a system acquisition manager and a software acquisition manager in allocation of quality goals to software;

(2) the issues that arise between a software acquisition manager and a hardware acquisition manager in ensuring that hardware and software implementations of the quality goals are compatible;

(3) the issues that arise during a competitive phased acquisition between the software acquisition manager and the contractors;

(4) the issues that arise when an existing distributed system is being modified or expanded to include new functions;

(5) the issues that arise when a distributed system is being modernized.

These issues are illustrated by the following scenarios.

4.1 System Availability Scenario

In this example computer system availability is the only system quality factor that is specified. System availability however is determined by system maintainability and system reliability. The system acquisition manager allocates budgets to maintainability and reliability. There are an infinite number of combinations of reliability and maintainability that will achieve the system availability goal. The system acquisition manager usually identifies a relevant range of combinations based on current state of technology, total life cycle cost, a feasible acquisition and operation expenditure profile, and a desired initial operation date.

The budgets are further broken down into hardware and software budgets. At this point the hardware acquisition manager and the software acquisition manager interact to develop a feasible and compatible design strategy.

The software manager breaks down the software reliability and software maintainability budgets further. The software reliability budget is allocated to fault tolerance and fault recovery. Fault tolerance contributes positively toward reliability in that the system continues to be available even though there are hardware faults.

Fault recovery is closely related to hardware maintainability because the system is not available until the hardware fault is repaired and the computer system restarted, and the data processing system resumes processing at the point where the hardware failure occurred. Recovery may include restoration of files, recovery of lost transactions, notification to system users, etc. The relative emphasis between fault tolerance and fault recovery must be compatible with the hardware architecture. For example if the topology of the system is such that a single hardware failure causes a system failure, then the software should emphasize fault recovery rather than fault tolerance.

The software maintainability budget is allocated to two quality factors, maintainability and correctness. It is assumed that the system will not change after its initial development and installation. If change were expected flexibility would be an important factor for system maintainability, provided the time between the identification of a new requirement or function and the incorporation of that function into the system is included in the system availability concept.

To meet the software maintainability budget tradeoffs between correctness and maintainability are made. The tradeoffs must include consideration of the feasible software expenditure profile and the desired initial operation date.

In a competitive phased acquisition the software acquisition manager establishes a feasible range and the competing contractors each propose a strategy for meeting the budget range. Suppose a contractor proposes to use "off the shelf" software that meets 80% of the requirements. To assess the risk the software acquisition manager would check the correctness, the maintainability, and the flexibility ratings of the proposed software. If the correctness rating is high but the maintainability and flexibility ratings are low, the contractors proposal would be judged risky, i.e. there is a good chance that the proposed software can't be used at all.

Figure 4.0–1 summarizes the issues exposed by this example and the relationship between the software acquisition manager and the system acquisition manager, the hardware acquisition manager, and the competing contractor.

4.2 Multifunction System Scenario

One way of characterizing distributed systems is the term "multifunction system". That is, the system performs multiple functions; the responsibility for performing those functions is usually distributed among several physically separated processors or processing complexes; and the execution of any one function will normally require cooperation among several physically separated processors/complexes with distinct capabilities. This is analogous to any large system with multiple subsystems cooperating to perform system functions.

From this point of view, quality requirements for the various subsystems are derived by: examining the quality requirements for each function; allocating a portion of the requirement for each function to each subsystem involved in the execution of that function; and then combining the allocations for one subsystem to determine its quality requirement(s).

For example, a navigation function for a military application being implemented in a distributed system may have high reliability, efficiency, and correctness requirements. These requirements will be allocated differently to different parts of the system. The correctness requirement would most likely be allocated to the software and possibly to the processor (e.g. the need for floating point precision) and the sensors (e.g. precision and accuracy).

The reliability and efficiency requirements will be allocated among the system elements in line with performing navigation: a certain amount of accuracy would be required of the sensors; the communication links from the sensors to the processors would most likely be redundant for reliability and require a minimum data transfer rate for efficiency; the processors would also likely be redundant and also require a minimum processing speed; and the software will require some type of resource management scheme to insure navigation function reliability.

If after the system is implemented a weapon delivery function is added which uses some of the same sensors, the same communication links, and the same processors the developed system quality will be impacted. For example the addition of this function may overload the communication system. Additionally the weapon delivery function will have quality requirements appropriate to its needs and different than the quality requirements for the navigation function. Moreover, after the new quality requirements are allocated the various elements, it will be necessary to combine the requirements from the different functions in order to arrive at the quality requirements for individual elements.

It is then necessary to perform trade studies in order to iteratively arrive at an optimum design solution. For example, the reliability allocation to the software might be decreased (with a corresponding decrease in cost) if management of the redundant communication link and any redundant sensors was placed in microprocessors which front–ended the main processors–provided that the cost to implement and maintain microprocessors with the required reliability did not exceed the cost budget for that subfunction.

5.0 SOFTWARE QUALITY IMPACT ON DEVELOPMENT COST

From Figure 2.0-1 it is clear that the more quality is built into the software, the more operational life costs will be reduced. It is also clear that there are trade offs among software quality factors –

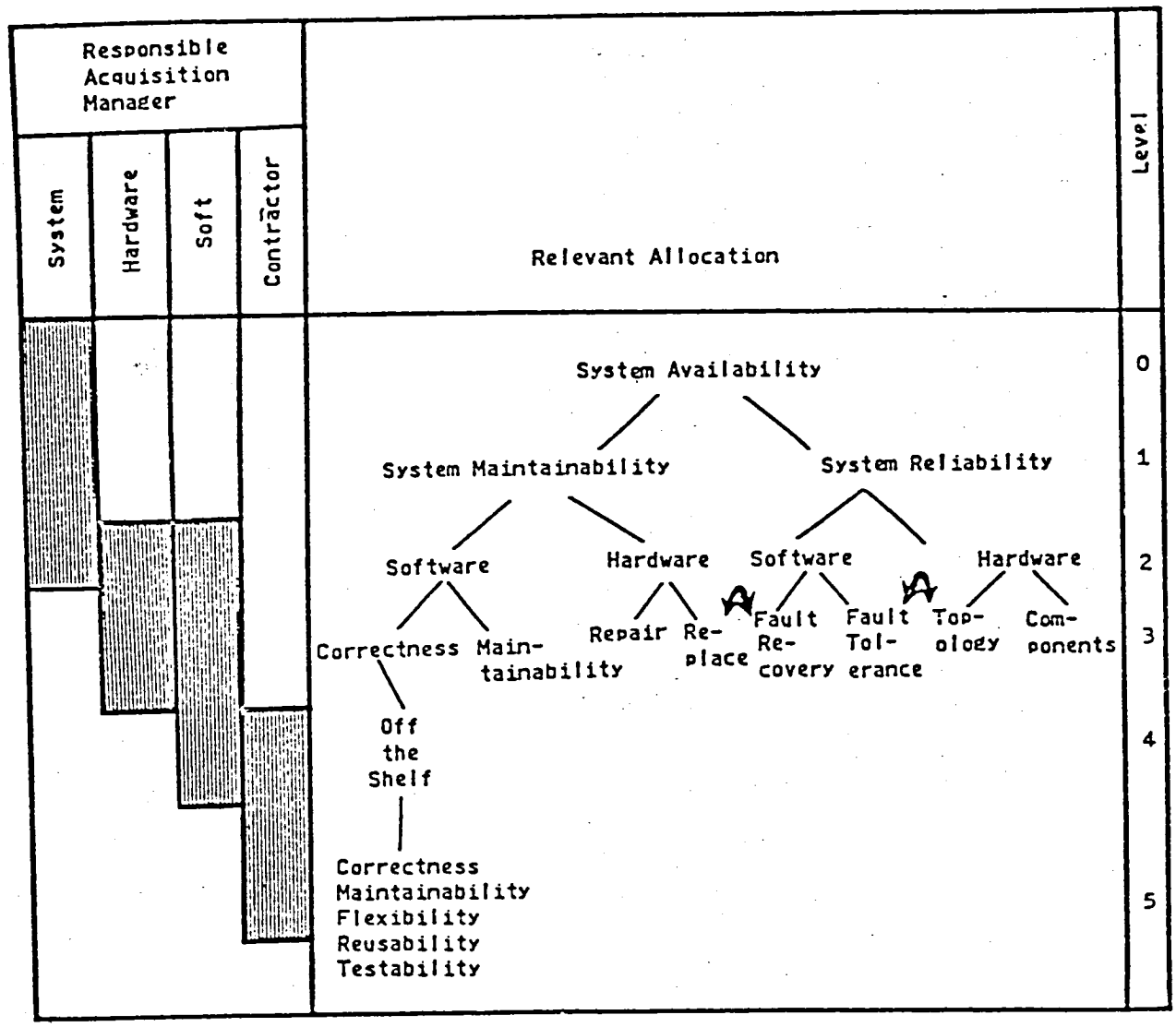


FIGURE 4.0-1 Trades and Allocations Relevant to System Availability Quality

for example it is generally accepted that increasing software efficiency will reduce its portability, flexibility, and usability. But the impact of software quality on development cost has not been explored. Currently software costing models do not account for the impact of software quality on development costs.

Figures 5.0-1a through 5.0-1d illustrate four alternate theories for this impact. They are:

(1) Software quality if productive – The greater the degree of software quality the lower development cost. i.e. Software quality is the key to increasing software development productivity;

(2) Software quality is free - Software development costs and software quality are independent. The level of quality attained is a consequence of the development methodology. In this case the interest in measuring software quality is partially directed at determining a superior development methodology;

(3) Software quality costs - Enhance quality is achieved only at increased cost. In this case there is a tradeoff between degree of quality and life cycle cost. Quality measurement is directed at determining when to stop increasing quality;

(4) Software quality is a combination of (1), (2), and (3). i.e. In the low range theory (1) holds, in the mid range theory (2) holds, and in the high range theory (3) holds.

Intuitively Theory (4) appears to be more realistic than the other three because

(1) If a software system ranks extremely low in maintainability and reliability during the development process it will take an "infinite" amount of time to correct an "infinite" number of errors. Even a small increase in quality will have a beneficial effect; hence the direction of section one is correct.

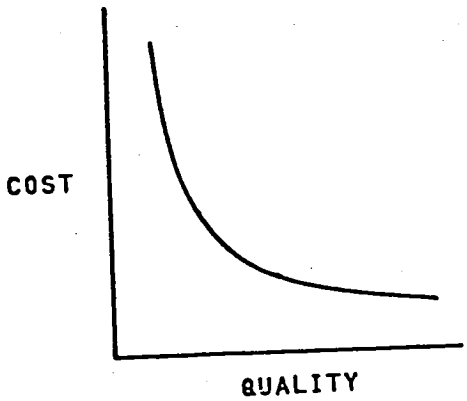
(2) A large number of programs of equivalent size lie within a narrow cost range. The quality of the programs vary widely. i.e. Section 2 of the curve has the correct slope.

(3) When programs utilize a large percentage of CPU time and/or memory capacity the cost of the software grows exponentially. But when utilization is below 50% efficiency is not a concern and traditionally software costs are reduced dramatically. Figure 5.0-2 illustrates a PRICE -S estimation of software costs as utilization increases from 60% to 95%. The curve was derived from data in Gaffney 80.

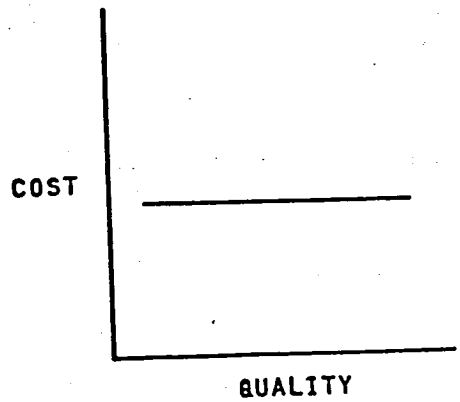
The rising portion of curve 5.0-1d is of interest because that is where trades between development cost and operational life costs are made. If all software quality factor curves are exponential like the PRICE -S model.

6.0 CONCLUSIONS & RECOMMENDATIONS

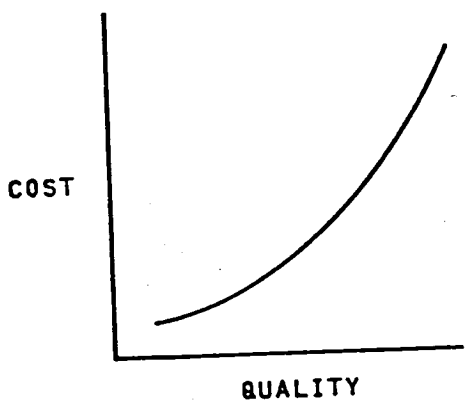
Measurement of the Quality of Software for Specific factors that are important in the operational phase provides a rational method of selecting among alternative off-the-shelf software packages. This technique will satisfy user concerns. But the developer needs to be able to determine the amount of quality that can be built into the software cost effectively. The systems approach can indicate the relative emphasis to be placed on software quality factors. But models are needed to help determine when it is cost effective to terminate improvement of software quality factors. An operational approach, such as tracking the marginal improvement in software quality can be used as a stopgap measure.



5.0-1a Productive Theory



5.0-1b Free Theory



5.0-1c Quality Costs Theory

5.0-1d Composite Theory

Figure 5.0-1 Alternate Theories for the Cost of Building Quality Factors into Software

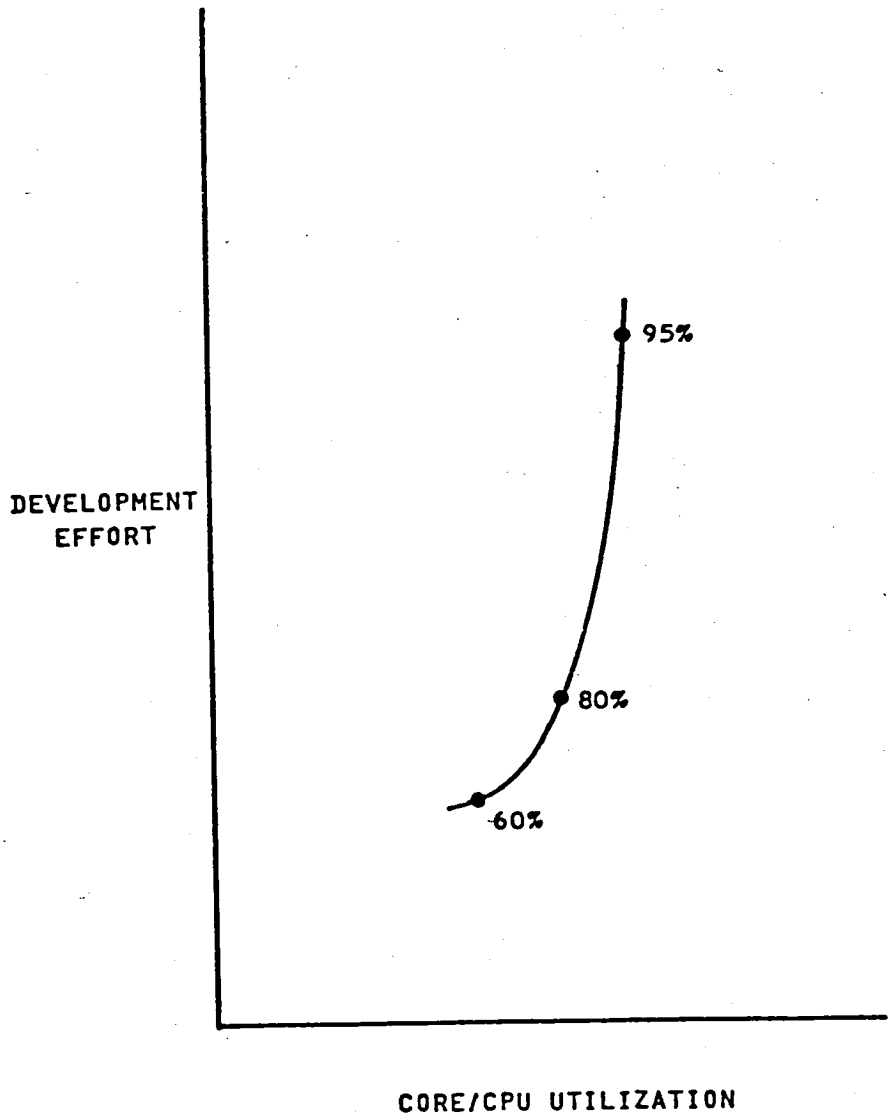


Figure 5.0-2 Impact of Utilization on
Software Development Cost PRICE-S

The cost development curve in the high quality range appears to be exponential for a single quality factor. But models for determining the development cost impact of a mix of quality factors requires a model that factors in the correlations among quality factors.

PANEL #3
SOFTWARE RELIABILITY

J. Logan, TRW
A. Goel/J. Soenjoto, Syracuse University and A. Sukert, Rome Air Development Center
M. Horn/W. Thompson, Colombia Research Corporation

N82 24005

SOFTWARE RELIABILITY:
Application of a Reliability Model To Requirements Error Analysis

J. Logan
TRW

Although requirements errors were identified several years ago as a major source of software problems encountered in software integration and testing, most reliability studies have been concerned with other issues and have made little contribution to solving requirements problems. Application of a software reliability model having a well defined correspondence to computer program properties to requirements error analysis has identified requirements error categories relatable to program structural elements and to their effect on program execution.

Analysis of B-5 type software requirement specifications has confirmed that errors of these types do occur.

The software reliability model represents in precise terms the software reliability definition: the probability that a program will compute required output values under specified conditions. In the model, the specified operational conditions are represented in terms of the set E of all possible inputs, the set E_i of input values for a specific execution, and the probability of choosing E_i as an input. Correct computation is defined as the actual computed value of (E_i) being equal to the required output value $\hat{f}(E_i)$ within a specified tolerence ϵ_i.

The model represents a software requirements specification in terms of the function $\hat{f}$ the program is required to compute on the input domain set $\hat{E}$. The requirement specification generally will partition the input set $\hat{E}$ into input subsets (input domain partitions) $\hat{G}_j$ and the function $\hat{f}$ into functions $\hat{f}_j$ on $\hat{G}_j$. These requirements elements correspond to program elements:

- $\hat{E}$ corresponds to the program input domain E
- $\hat{G}_j$ correspond to input domain subsets $\hat{G}_j$
- $\hat{f}_j$ corresponds to the logic path L_j executed for inputs from G_j and which computes the function f_j on G_j

Requirements errors arise from:

- incomplete specification of
  - the input domain $\hat{E}$,
  - the input domain partition subsets $\hat{G}_j$, or
  - the functions $\hat{f}_j$ on $\hat{G}_j$.
- incorrect specification of these same items or
- association of a function $\hat{f}_j$ with the wrong input domain partition $\hat{G}_k$.

The B-5 software requirement specification format is compatible with the model, for it presents the requirement specifications for a program module in terms of:

- Inputs
  - Specifying $\hat{E}$ and $\hat{G}_j$
- Processing
  - Specifying the functions $\hat{f}_j$ to be computed
- Outputs
  - Specifying the output values $\hat{f}_j(E_i)$

The requirement error categories provide a systematic means for examining the requirement specification for errors, aiding quick identification of them.

Here is a simplified example of a B-5 type requirement specification for a program module named Event Processor. Although this example is not a real example, it closely approximates the problems found in actual requirements reviews.

The Event Processor is required to log event occurrence and select for each event occurrence the task to be performed in accordance with prescribed action criteria.

The module is required to process two inputs, the event identifier and a task action table. Two processing functions are specified:

- on occurrence of an event, select the task meeting task action criteria, and
- post the event, time of event occurrence, and the task selected in the event log.

One output, the updated event log, is specified.

Some of the problems in this requirement specification are shown on the following viewgraphs.

The first problem identified is incomplete specification of the input Task Action Table. Only the name of this input is specified. While the name indicates it is a table, the structure of the table and the range of values allowed as elements of the table are not specified. Presumably, definition of this table is left to the software designer.

The second problem identified is incomplete specification of a processing function. The task action criteria, presumably algorithms for selecting the task, are not defined. Possibly the incompletely specified task action table contains a tabular representation of the criteria, again for the software designer to invent. While this may result in a correct design, generally, the software designer understands the problem the program is intended to compute less well than the requirement specifier and therefore is more likely to design a program computing the wrong problem.

The third problem identified is reference in a processing function to an input variable time not on the input list.

The fourth problem identified is a missing function. Presumably (again incomplete input specification) the task action table contains an entry for each expected event. The missing function is the processing to be performed in an event identifier not present in the task action table is presented to the program.

These four problems are not the only problems in this specification, but time does not permit discussion of the others.

THE VIEWGRAPH MATERIALS
for the
J. LOGAN PRESENTATION FOLLOW

RELIABILITY MODEL APPLICATION TO REQUIREMENTS ERRORS

REQUIREMENTS ERRORS

- MAJOR SOURCE OF SOFTWARE PROBLEMS

RELIABILITY MODEL

- IDENTIFIES REQUIREMENTS ERROR CATEGORIES

ANALYSIS OF B-5 SPECIFICATIONS

- CONFIRM ERROR CATEGORIES

SOFTWARE RELIABILITY MODEL

SOFTWARE RELIABILITY DEFINITION

- PROBABILITY THAT PROGRAM WILL COMPUTE REQUIRED OUTPUT VALUES UNDER SPECIFIED CONDITIONS

SPECIFIED OPERATIONAL CONDITIONS

- E : SET OF ALL POSSIBLE INPUTS
- E_i : INPUT TO SPECIFIC EXECUTION
- P_i : PROBABILITY OF CHOOSING E_i

CORRECT COMPUTATION

- $F(E_i)$: ACTUAL COMPUTED VALUE
- $\hat{F}(E_i)$: REQUIRED VALUE
- $|\hat{F}(E_i) - F(E_i)| \leq \epsilon_i$: CORRECT COMPUTATION

MODEL REPRESENTATION OF SOFTWARE REQUIREMENTS

PROGRAM IS REQUIRED TO

- COMPUTE FUNCTION $\hat{F}$ FOR ALL INPUTS IN $\hat{E}$

REQUIREMENTS PARTITION

- $\hat{E}$ INTO SUBSETS $\hat{G}_j$
- $\hat{F}$ INTO FUNCTIONS $\hat{F}_j$ ON $\hat{G}_j$

REQUIREMENTS ELEMENTS CORRESPOND PROGRAM ELEMENTS

- $\hat{E}$ CORRESPONDS TO INPUT DOMAIN E
- $\hat{G}_j$ CORRESPONDS TO INPUT DOMAIN SUBSET G_j
- $\hat{F}_j$ CORRESPONDS TO LOGIC PATH L_j COMPUTING FUNCTION F_j ON G_j

REQUIREMENT SPECIFICATION SPECIFIES

- $\hat{E}$
- $\hat{G}_j$
- $\hat{F}_j$ ON $\hat{G}_j$

REQUIREMENTS ERRORS ARISE FROM

INCOMPLETE SPECIFICATION OF

- $\hat{E}$
- $\hat{G}_j$
- $\hat{F}_j$ ON $\hat{G}_j$

INCORRECT SPECIFICATION OF

- $\hat{E}$
- $\hat{G}_j$
- $\hat{F}_j$ ON $\hat{G}_j$

ASSOCIATION OF

- $\hat{F}_j$ WITH $\hat{G}_k$

APPLICATION TO B-5 SOFTWARE REQUIREMENT SPECIFICATION

B-5 FORMAT IS COMPATIBLE WITH MODEL

- INPUTS CORRESPONDS TO $\hat{E}$ AND $\hat{G}_j$
- PROCESSING CORRESPONDS TO $\hat{F}_j$
- OUTPUTS CORRESPONDS TO $\hat{F}_j(E_i)$

REQUIREMENT ERROR CATEGORIES

- AID QUICK IDENTIFICATION OF ERRORS

B-5 REQUIREMENT SPECIFICATION EXAMPLE

NAME: EVENT PROCESSOR

- REQUIRED TO LOG EVENT OCCURRENCE AND SELECT TASK IN ACCORDANCE WITH ACTION CRITERIA

INPUTS

- EVENT ID
- TASK ACTION TABLE

PROCESSING

- ON OCCURRENCE OF EVENT, SELECT TASK MEETING TASK ACTION CRITERIA
- POST EVENT, TIME OF EVENT OCCURRENCE, AND TASK SELECTED IN EVENT LOG

OUTPUTS

- UPDATED EVENT LOG

INCOMPLETE SPECIFICATION OF INPUT

NAME: EVENT PROCESSOR

- REQUIRED TO LOG EVENT OCCURRENCE AND SELECT TASK IN ACCORDANCE WITH ACTION CRITERIA

INPUTS

- EVENT ID
- TASK ACTION TABLE (NOT DEFINED)

PROCESSING

- ON OCCURRENCE OF EVENT, SELECT TASK MEETING TASK ACTION CRITERIA
- POST EVENT, TIME OF EVENT OCCURRENCE, AND TASK SELECTED IN EVENT LOG

OUTPUTS

- UPDATED EVENT LOGS

INCOMPLETE SPECIFICATION OF PROCESSING FUNCTION

NAME: EVENT PROCESSOR

- REQUIRED TO LOG EVENT OCCURRENCE AND SELECT TASK IN ACCORDANCE WITH ACTION CRITERIA

INPUTS

- EVENT ID
- TASK ACTION TABLE

PROCESSING

- ON OCCURRENCE OF EVENT, SELECT TASK MEETING TASK ACTION CRITERIA (NOT DEFINED)
- POST EVENT, TIME OF EVENT OCCURRENCE, AND TASK SELECTED IN EVENT LOG

OUTPUTS

- UPDATED EVENT LOG

PROCESSING REFERENCE OF VARIABLE NOT ON INPUT LIST

NAME: EVENT PROCESSOR

- REQUIRED TO LOG EVENT OCCURRENCE AND SELECT TASK IN ACCORDANCE WITH ACTION CRITERIA

INPUTS

- EVENT ID
- TASK ACTION TABLE

PROCESSING

- ON OCCURRENCE OF EVENT, SELECT TASK MEETING TASK ACTION CRITERIA
- POST EVENT, TIME (NOT ON INPUT LIST) OF EVENT OCCURRENCE, AND TASK SELECTED IN EVENT LOG

OUTPUTS

- UPDATED EVENT LOG

PROCESSING FUNCTION NOT DEFINED FOR INPUT PARTITION

NAME: EVENT PROCESSOR

- REQUIRED TO LOG EVENT OCCURRENCE AND SELECT TASK IN ACCORDANCE WITH ACTION CRITERIA

INPUTS

- EVENT ID
- TASK ACTION TABLE

PROCESSING

- ON OCCURRENCE OF EVENT, SELECT TASK MEETING TASK ACTION CRITERIA
- POST EVENT, TIME OF EVENT OCCURRENCE, AND TASK SELECTED IN EVENT LOG
- (PROCESSING FOR EVENT ID NOT IN TASK ACTION TABLE IS NOT DEFINED)

OUTPUTS

- UPDATED EVENT LOG

SOFTWARE RELIABILITY:
Optimum Maintenance Policies for Hardware-Software Systems

Amrit L. Goel and J. Soenjoto
Syracuse University
Alan Sukert
Rome Air Development Center

The objective of this presentation is to develop an analytical model for a hardware-software system at a macro level in order to calculate the optimum maintenance policies. A markov model of a hardware-software system is developed. We assume that the system consists of hardware and software components which are subject to random failures and random maintenance times. Distributions of the time to next failure and maintenance times (of both hardware and software components) are assumed to be exponential. Next we explain the maintenance and failure rate giving some factors on which they depend. Some operational performance measures, such as distribution of time to next failure, probability of successful operation for a specified time, system availability, distribution of number of failures by a specified time are developed next and studied. Afterwards, we develop a cost model which incorporates the cost of failures, system maintenance activities, and down time. Then we formulate the problem as a non-linear optimization problem with or without constraints on repair rates and system availability to compute the optimum maintenance policies.

ECONOMICALLY OPTIMUM HARDWARE/SOFTWARE MAINTENANCE STRATEGIES

by

Amrit L. Goel[1], Jopie B. Soenjoto[2], Alan N. Sukert[3]

Continued increase in the use of computer systems in a wide variety of applications has necessitated a greater emphasis on the development and deployment of cost-effective and reliable hardware/software systems. Several models have been developed and tested during the past seven years to quantify the performance of such software systems.

In this presentation, we summarize our previous results and then present a Markovian model for the operational phase of a hardware/software system. Several performance measures, such as average system availability, and number of software and hardware failures by time t, are developed and studied. A cost model is proposed which incorporates the cost of failures, system unavailability and maintenance. Optimum maintenance policies are computed by formulating the problem as a nonlinear optimization problem with or without constraints on repair rates and system availability.

[1]Professor, I.E.&O.R., and School of Computer and Information Science, Syracuse University.
[2]Research Assistant, I.E.&O.R., Syracuse University.
[3]ISIS, Rome Air Development Center, Griffiss Air Force Base, Rome, New York, 13441.

THE VIEWGRAPH MATERIALS
of the
A. GOEL PRESENTATION FOLLOW

OBJECTIVES

- DEVELOP AN ANALYTICAL FRAMEWORK FOR ASSESSING HARDWARE-SOFTWARE PERFORMANCE AT A MACRO LEVEL.
- EXPLORE TRADEOFFS BETWEEN PERFORMANCE MEASURES
- DETERMINE OPTIMUM PARAMETRIC VALUE
  - WITHOUT CONSTRAINTS
  - WITH CONSTRAINTS
- PROVIDE INSIGHTS INTO ROLES OF SYSTEM PARAMETERS IN DETERMINING PERFORMANCE

MARKOV MODEL FOR A HARDWARE-SOFTWARE SYSTEM

- SYSTEM CONSISTS OF HARDWARE AND SOFTWARE COMPONENTS
- THESE ARE SUBJECT TO RANDOM FAILURES
- MAINTENANCE TIMES ARE RANDOM

A MARKOV MODEL

- FAILURES, MAINTENANCE ASSUMED RANDOM WITH EXPONENTIAL DISTRIBUTIONS
- KEY PARAMETERS:
  - S/W FAILURE RATE (λ_i)
  - H/W FAILURE RATE (β)
  - S/W MAINTENANCE RATE (μ_i)
  - H/W REPAIR RATE (γ)

DISTRIBUTION OF FAILURE AND MAINTENANCE TIMES

- T_i - TIME TO NEXT SOFTWARE FAILURE
- W_i - SOFTWARE MAINTENANCE TIME
- U - TIME TO NEXT HARDWARE FAILURE
- V - HARDWARE MAINTENANCE TIME

cdf of $T_i = 1 - e^{-i\lambda t}$

cdf of $W_i = 1 - e^{-\mu it}$

cdf of $U = 1 - e^{-\beta t}$

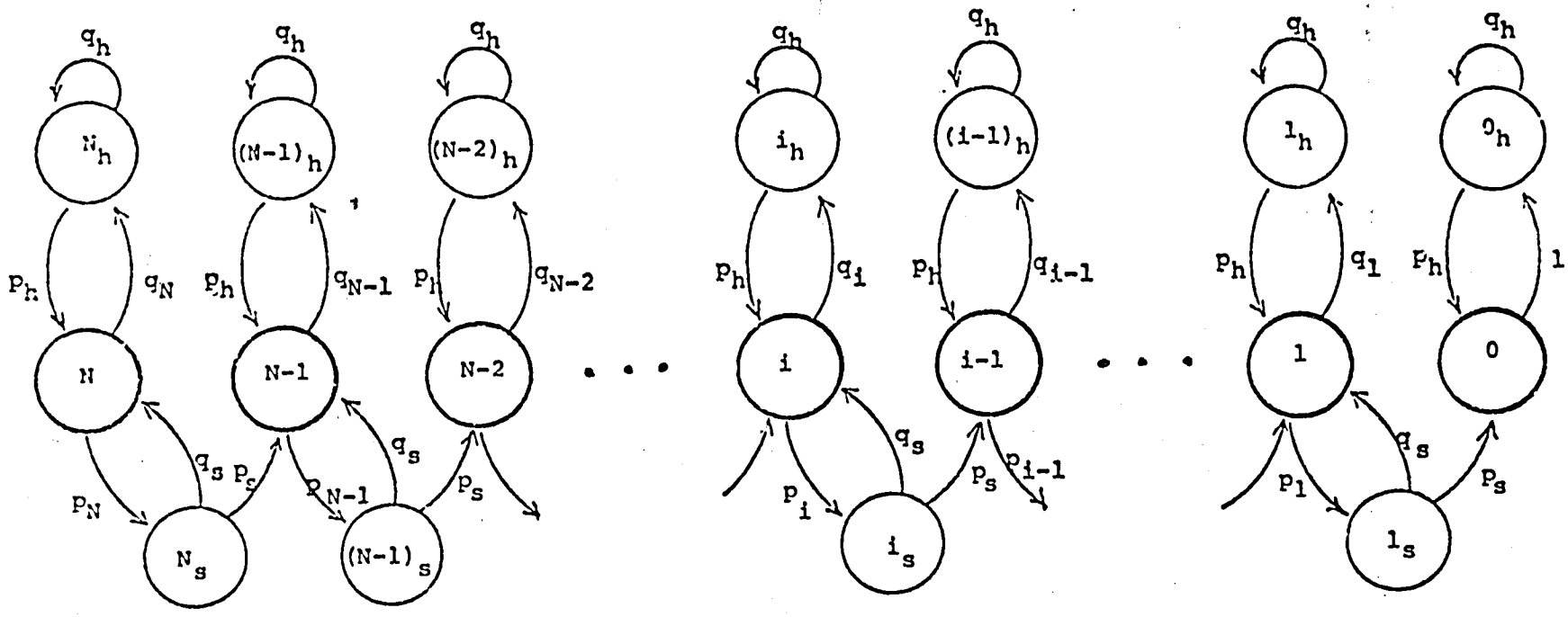
cdf of $V = 1 - e^{-\gamma t}$

FAILURE RATES:

- DETERMINE TIMES BETWEEN FAILURES
- DEPEND ON:
  - CODE QUALITY
  - TESTING LEVEL
  - REDUNDANCY
  - FAULT TOLERANCE
  - WORK LOAD

MAINTENANCE

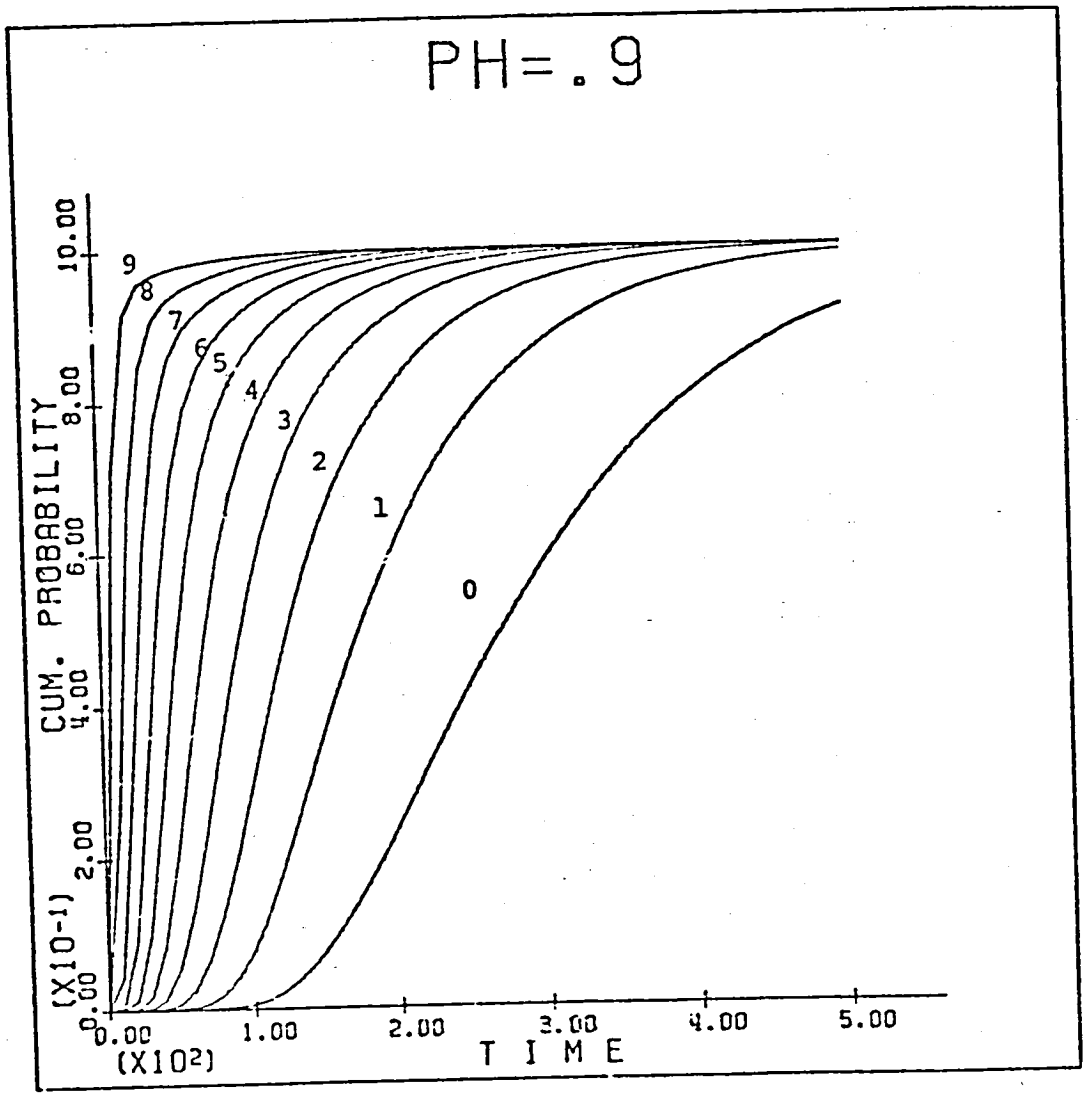
- INCLUDES ALL ACTIVITIES REQUIRED TO BRING THE SYSTEM BACK INTO OPERATION AFTER A FAILURE
- DOES NOT INCLUDE ENHANCEMENT OF CAPABILITIES
- DEPENDS ON
  - QUALITY OF PERSONNEL
  - S/W DEVELOPMENT METHODOLOGY
  - SYSTEM RECOVERABILITY



Diagrammatic Representation of Transitions Between States of X(t).

SOME USEFUL OPERATIONAL PERFORMANCE MEASURES

- DISTRIBUTION OF TIME TO NEXT FAILURE
- PROBABILITY OF SUCCESSFUL OPERATION FOR A SPECIFIED TIME
- SYSTEM AVAILABILITY
- DISTRIBUTION OF NUMBER OF FAILURES BY A SPECIFIED TIME

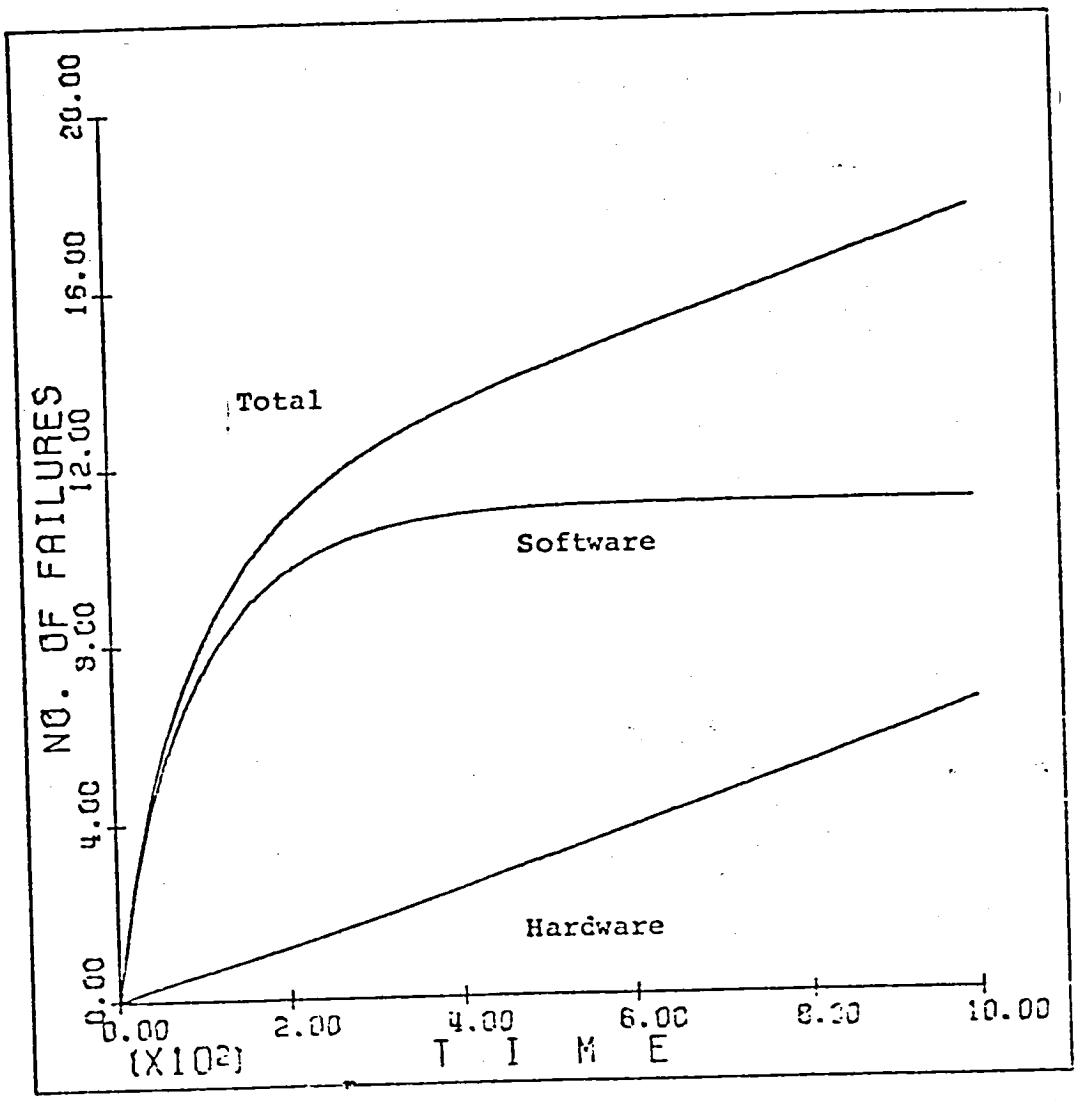


$\lambda = .02$ $\beta = .01$ $p_s = .9$

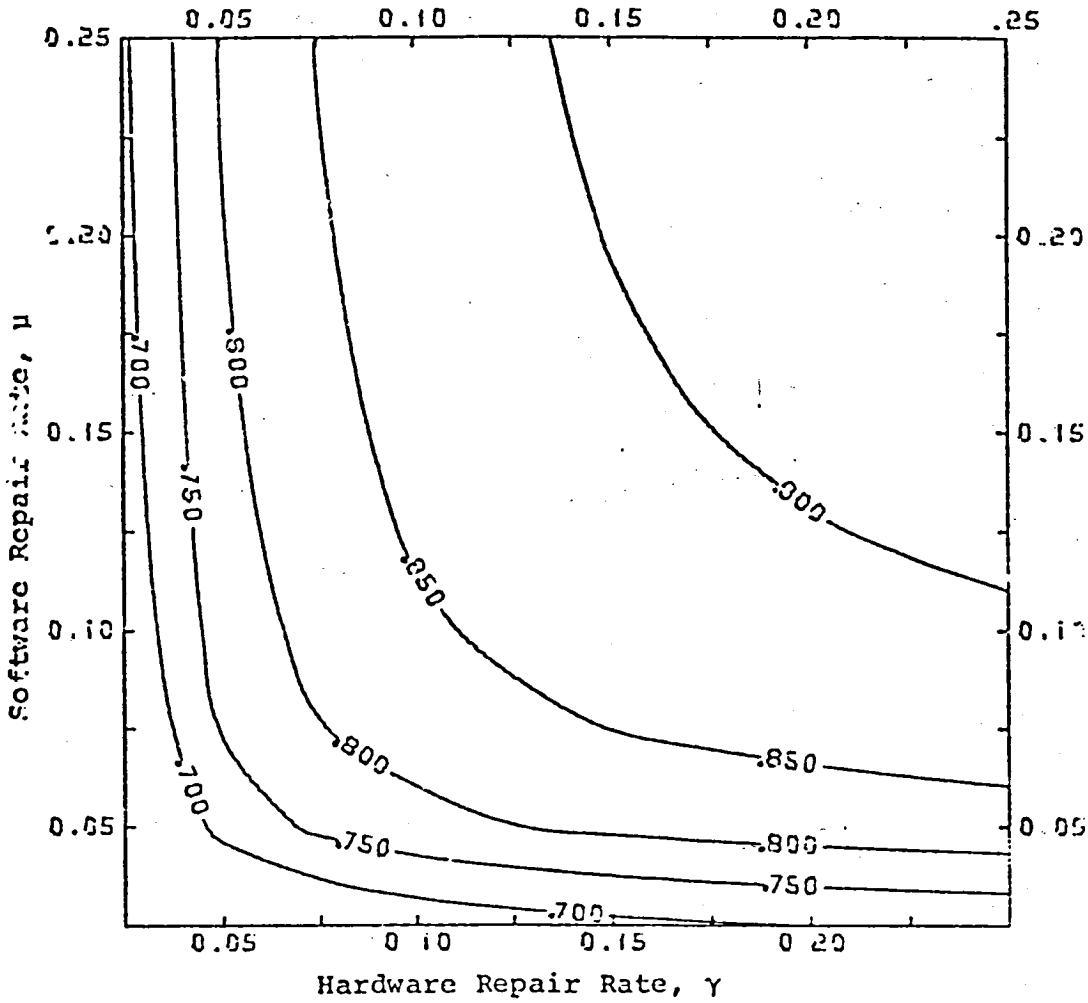
$\mu = .05$ $\gamma = .025$ $p_h = .9$

$N = 10,$ $n = 9,8,7,\ldots,2,1,0$

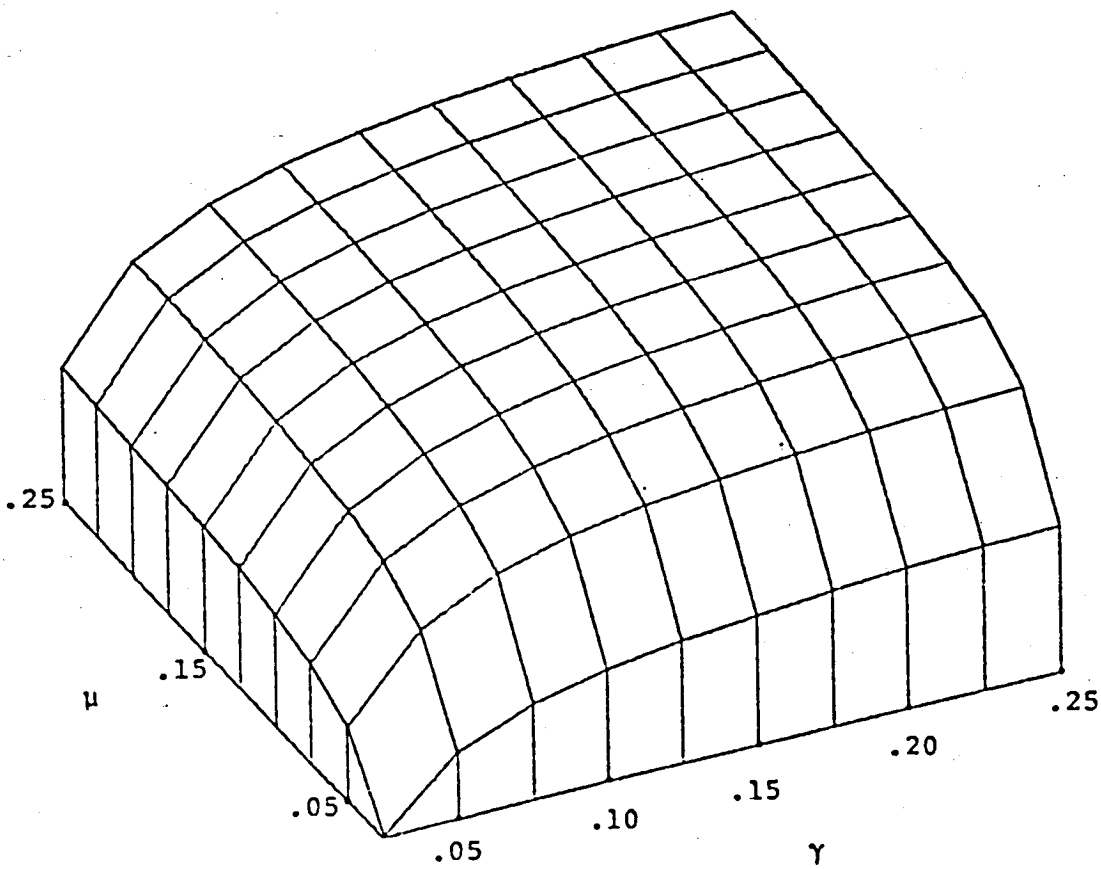
Cumulative Distribution Function of First Passage Time to n.



Expected Cumulative Number of Failures.



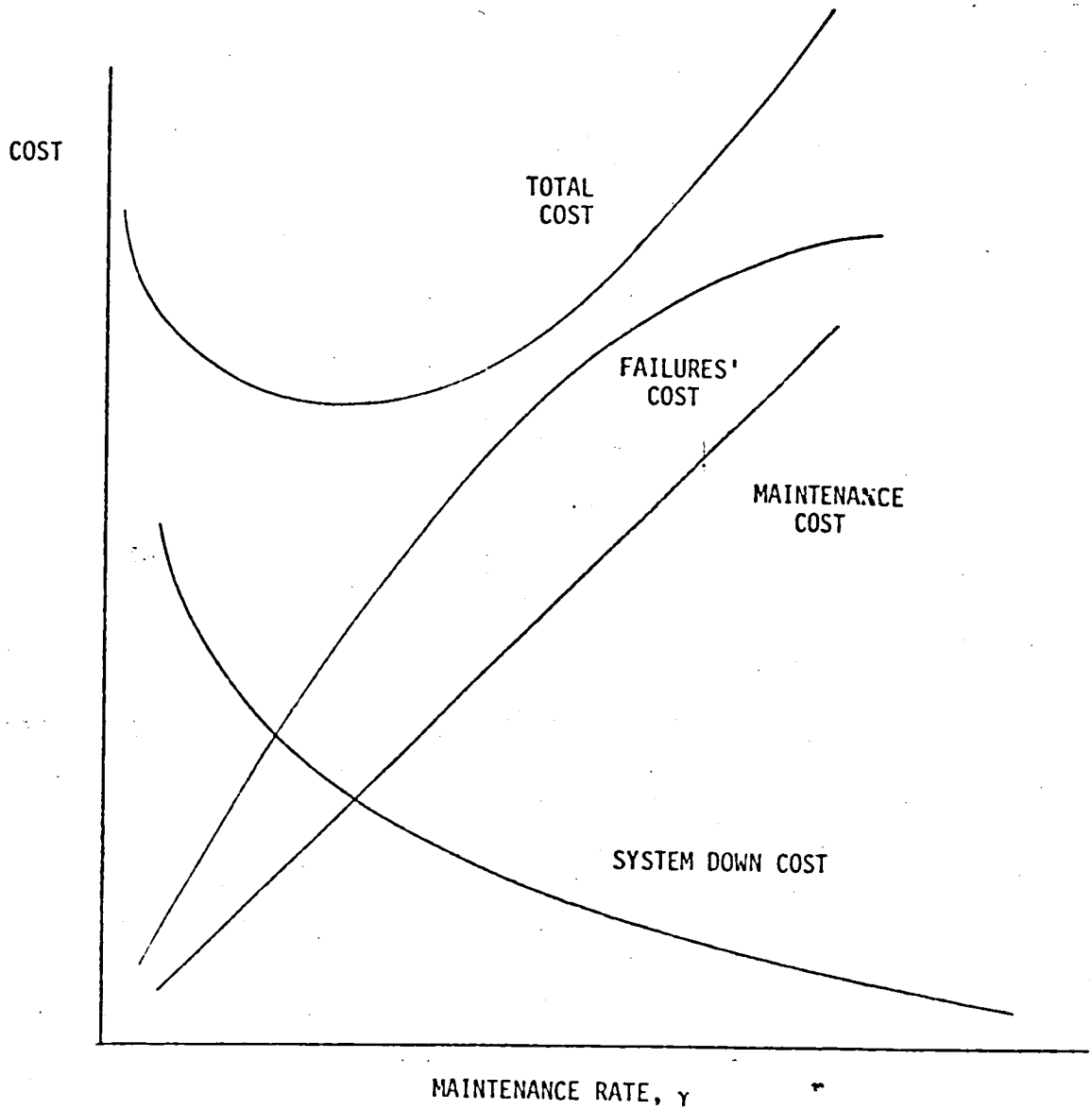
Contours of Average Availability vs. Repair Rates: Hardware-Software System (β = .01, λ = .05, t = 500).

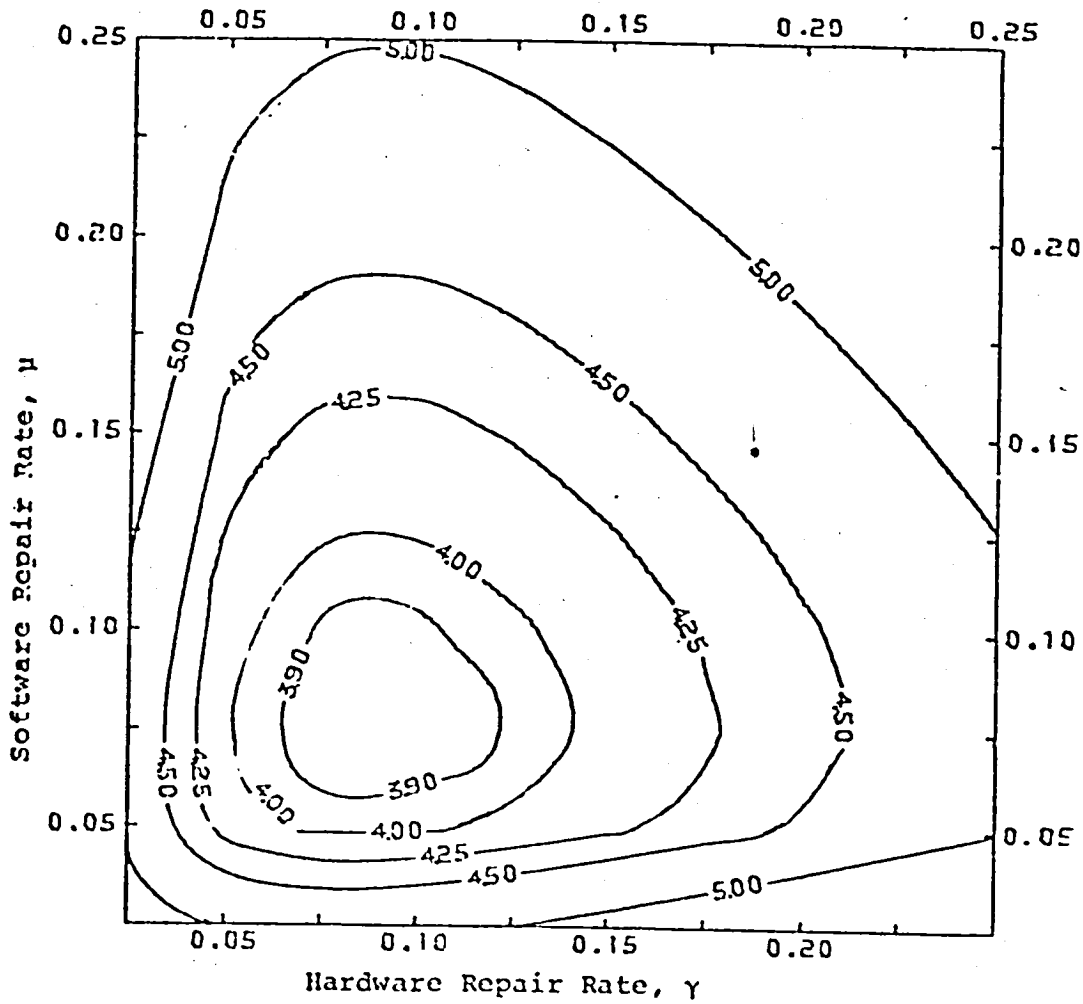


Surface of Average Availability vs. Repair Rates: Hardware-Software System (β = .01, λ = .05, t = 500).

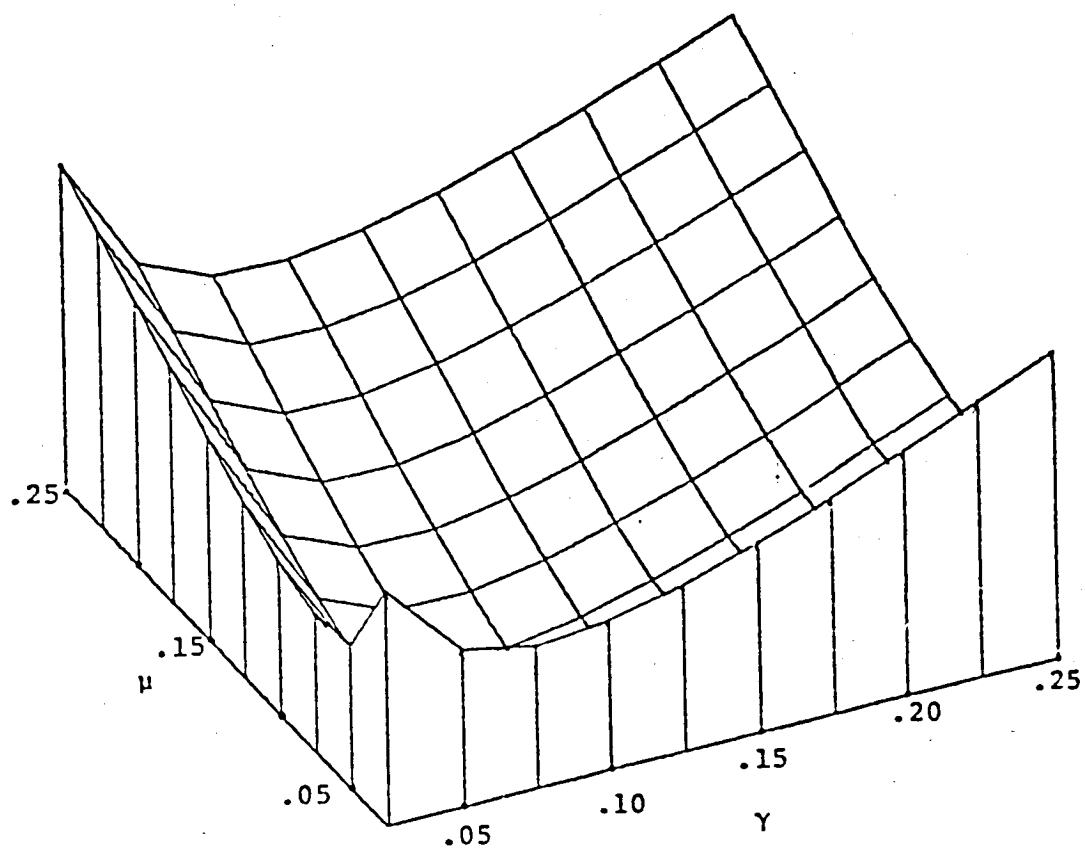
OPERATIONAL COST MODELS

- INCLUDE COSTS OF
  - FAILURES (HARDWARE OR SOFTWARE)
  - MAINTENANCE ACTIVITY AND PERSONNEL
  - SYSTEM DOWNTIME
- PERFORMANCE MEASURES THAT AFFECT COST:
  - EXPECTED NUMBER OF FAILURES
  - SYSTEM UNAVAILABILITY


COST
TOTAL
COST
FAILURES'
COST
MAINTENANCE
COST
SYSTEM DOWN COST
MAINTENANCE RATE, y



Contours of Expected Total Cost/Unit Time vs. Repair Rates: Hardware-Software System ($\beta = .01$, $\lambda = .05$, $t = 500$, cost factors = 10).



Expected Total Cost/Unit Time vs. Repair Rates: Hardware-Software System (δ = .01, λ = .05, t = 500).

OPTIMIZATION OF OPERATIONAL COST

- FOR GIVEN COST FACTORS, EXPECTED TOTAL COST IS SOLELY DETERMINED BY THE FAILURE AND MAINTENANCE RATES.
- FOR A GIVEN SYSTEM, FAILURE OR MAINTENANCE RATES CAN BE OPTIMIZED.

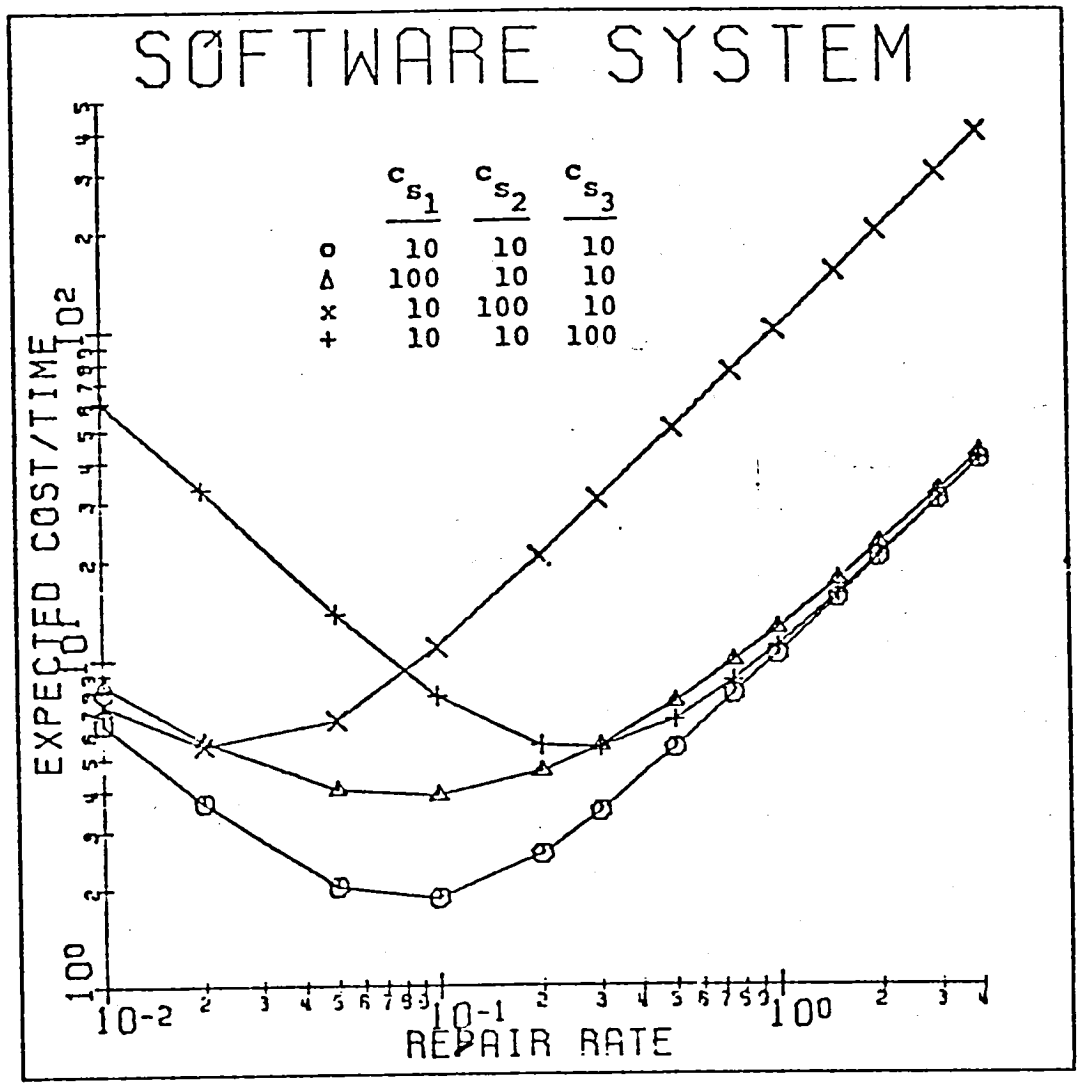
OPTIMIZATION PROCEDURE

- FUNCTIONS INVOLVED ARE NON-LINEAR
- 2ND AND HIGHER ORDER DERIVATIVES ARE ALMOST IMPOSSIBLE TO OBTAIN
- WE USE THE FOLLOWING METHODS:
  - POWELL'S SEARCH
  - VARIABLE METRIC METHOD AND DFP ALGORITHM

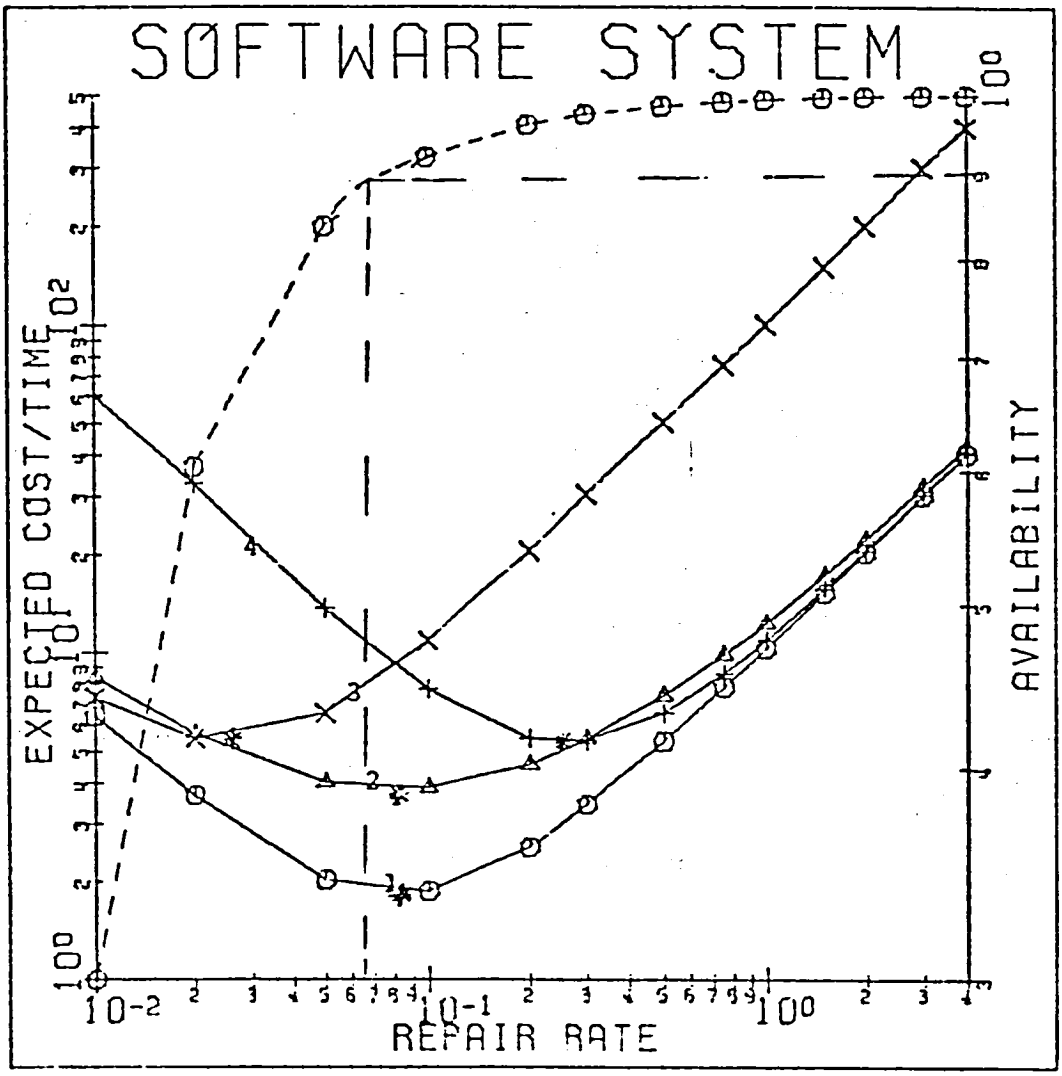
  } UNCONSTRAINED

  - LAGRANGIAN FUNCTIONS AS UNCONSTRAINED FOR USING VMM

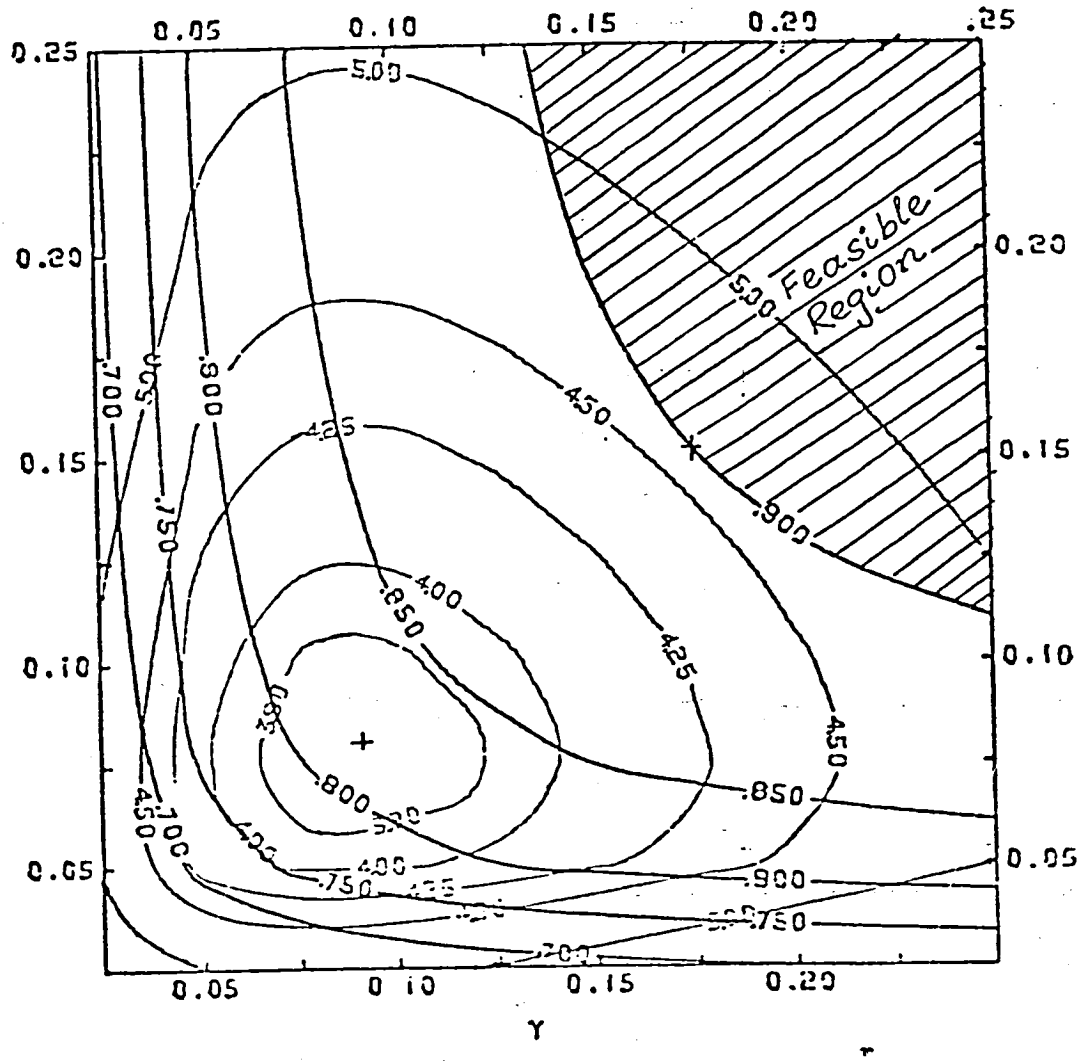
  } CONSTRAINED



Expected Total Cost/Unit Time vs. Repair Rate for Different Cost Factors ($\lambda = .05$, $t = 500$).



Expected Total Cost/Unit Time (for Different Cost Factors) and Average Availability vs. Repair Rate (λ = .05, t = 500).



Expected TotalCost/Unit Time and Average Availability vs. Repair Rates: Hardware-Software System (β = .01, λ = .05, t = 500).

CONCLUSIONS

- MODEL PROVIDES AN ANALYTICAL FRAMEWORK FOR ASSESSMENT
- NEEDS REFINEMENTS, RELAXATION OF ASSUMPTIONS
- NEEDS VALIDATION
- TOO MACRO?
- FURTHER WORK ON OPTIMIZATION ALGORITHMS DESIRABLE

N82 24006

SOFTWARE RELIABILITY:
A Comparison of Results Obtained from Established Software Reliability Models

Martin H. Horn and William E. Thompson
Columbia Research Corporation, Arlington, Virginia

ABSTRACT

The software error detection process may be described as a stochastic process within the general Poisson family, but is distinguished by having rates which are changed by events, i.e. detection and correction of software errors.

Two of the best known models of the software error detection process are here compared, the Jelinski-Moranda model [1] and a Bayes inference model [2, 3]. Simulation techniques are used to generate software-related system failure data which is analyzed by both models. Point estimates and confidence limits are compared.

It is demonstrated that uncertainty may be considerable for reasonable sample sizes and should certainly be considered in any application of these techniques. It is further demonstrated that the Jelinski-Moranda model is extremely sensitive to failure of data to follow the model's internal assumptions, often not providing any point estimates, a factor which may limit its usefulness in many real-world situations. The Bayes model is shown to be able to respond to the introduction of additional errors in the software correction process, a condition where error counting models such as the Jelinski-Moranda generally fail to converge.

INTRODUCTION

A "malfunction" or "failure" of a computerized system is any response unacceptable to an objective and consistent user. Each system failure falls within one of three categories: a hardware-related system failure (caused by such factors as error in hardware design or construction, or failure of component in service); a software-related system failure (caused by error in design or implementation of computer programs, data or documentation) and the ambiguous or unknown failure which cannot immediately be placed in either the hardware-related or software-related categories.

The overall reliability of the system may be defined as the probability that the system will function throughout a given interval of time without a failure, given that it was functioning at the start of the interval. "Software reliability" is then defined as the probability that the system will function throughout a given time interval without experiencing a software-related system failure. The mechanisms of hardware-related and software-related system failure are considered to be independent point processes following a Poisson distribution with a constant rate parameter. This rate parameter, for software-related system failures, is denoted by λ throughout this study and will be the parameter solved for in the reliability models. The reciprocal of λ is the expected or mean time to the next software-related system failure, assuming no further correction of software errors, and thus may be related to "software reliability" as already defined.

In the development of a hardware-software system, software may be tested or debugged using test hardware which may not be the environment in which it eventually will operate. In the test environment, software errors are detected through exercise of the different program branches. The discovery of a software error in such a test process is defined, for the purposes of these reliability models, as equivalent to a software-related failure of the complete system in the field.

THE MODELS

The two models investigated in this paper estimate the rate parameter λ in two different ways, both using as input the observed times between software-related system failures. The time to the first software-related system failure is denoted as x_1; the time between the first and second failures is x_2; the time between the $(n-1)$th and nth failures is x_n. The sum of all x_i, from $i = 1$ to $i = n$, is the total elapsed time after n software-related system failures have occurred (Figure 1).

A. Jelinski-Moranda Model

Jelinski and Moranda, in 1972, were among the first to offer a mathematical formulation for software reliability prediction. This model is one of a family of classical generalized Poisson models. It assumes that a given software package contains a finite number N of errors initially.

The fundamental assumption of the Jelinski-Moranda model is that λ is proportional to the number of errors currently present in the software, and that each error, once detected through a software-related system failure, is corrected immediately with no errors introduced in the correction process. As each error is removed, λ will decline in steps, but it is a constant rate parameter between each detection and removal of an error (Figure 2).

The Jelinski-Moranda algorithm for estimating the value of N is:

$$\sum_{i=1}^{n} \frac{1}{N-(i-1)} = \frac{n \sum X_i}{N \sum X_i - \sum (i-1) X_i} \qquad (1)$$

(n, N are positive integers with $N \geq n$)

Once a best estimate for N is found, the proportionality constant ϕ which represents one error's contribution to the failure rate is computed as:

$$\phi = \frac{n}{N \sum X_i - \sum (i-1) X_i} \qquad (2)$$

The rate parameter λ is then calculated from:

$$\lambda = \phi[N-(n-1)] \qquad (3)$$

The computer algorithm used in the solution of these equations first evaluates the left- and right-hand sides of equation (1) for $N = n$. It then evaluates these expressions for $N = n + 1$, $N = n + 2$, etc., establishing curves describing the left-hand and right-hand sides as functions of N. When an N is found where these curves intersect—or close to such an intersection if they do not intersect at an integer value of N—this N is the estimate of the number of errors originally presented and is used in calculation of ϕ and λ.

Should iteration of this procedure reach a preset maximum N without the curves intersecting, no further iteration is carried out and a message "NO SOLUTION FOUND" is printed. (In most

runs this maximum N was set at 200. In several runs, the maximum was increased to 400 with no significant change in the probability of not reaching a solution; it was concluded that in most cases when no solution is found the equation will not converge, rather than converge on an N higher than the limit.)

The failure times x_i were generated through a random number routine for input to the model to simulate a series of error detections. The x_i have an exponential distribution with a rate parameter $\lambda_o = \phi_o[N_o - (i-1)]$ where the "true" values N_o and ϕ_o are input at the start of the run and the X_i computed from:

$$r_i = 1 - e - \phi_o X_i (N_o - (i-1)) \tag{4}$$

where the r_i is the random numbers uniformly distributed over [0, 1].

After each "failure occurrence" the series of x_i so far generated would be used in a Jelinski-Moranda estimate of N and λ. If the equation (1) did in fact converge, these estimates would be printed out.

Figure 3 represents a typical output from a Jelinski-Moranda run. The horizontal axis denotes n, the number of software-related system failures which have already occurred, while the vertical axis represents λ as computed through the Jelinski-Moranda model at each successive n. The graph compares the λ computed through the software development process to the "true" λ_o defined in terms of the input parameters ϕ_o and N_o by:

$$\lambda_o = \phi_o(N_o - (n-1)) \tag{5}$$

This follows from the fundamental assumption that λ is dependent on the number of errors remaining in the program, and that each error is corrected immediately upon discovery. Equation (4) may thus be restated as

$$x_n = \frac{1}{\lambda_o(n)} \ln(1 - r_n) - 1 \tag{4}$$

Although λ_o is a step function which is constant between changes in n, it is shown in Figure 3 as a constantly declining function of n.

There are gaps in the graph of λ as a function of n, as computed through the Jelinski-Moranda model. Where no λ appears for a given n, Figure 3, no solution was computed for that n because equation (1) failed to converge.

B. Bayes Model

The second reliability model considered in this study is based on the Bayes inference procedure. The procedure assumes that λ is a random variable and locates confidence limits of that variable from the posterior distribution function of λ.

The Bayes inference procedure computes a posterior density function based on a previously computed prior density function and the experimental data obtained since the computation of the prior. At a given time T_o after k_o errors have been detected, the prior may be defined as:

$$p(\lambda) = \frac{(\lambda T_o)^{k_o} e^{-\lambda T_o}}{\Gamma(k_o + 1)} \quad \text{[Reference 2]} \tag{6}$$

where Γ represents the gamma function ($\Gamma(k_o + 1) = k_o!$ if k_o is an integer ≥ 0).

It is assumed here that the software in question does in fact contain further errors: that at a given time T' beyond the T_o the number k of errors already detected will increase to $k' = k_o + k$. At this time the posterior density function for λ may be represented as:

$$f(\lambda \mid k, T) = \frac{(T' + T_o)\,(\lambda(T' + T_o))^{(k' + k_o)}\, e^{-\lambda(T' + T_o)}}{\Gamma(k' + k_o + 1)} \tag{7}$$

The distribution function is generated by numerical integration of this density function using Simpson's rule. The density function is used as the prior in evaluation of the posterior density function for the next test interval where after another interval of time T' a number k' of errors is detected through software-related system failures.

The distribution function, defined as:

$$F(\lambda' \mid k, T) = \int_o^{\lambda'} f(\lambda \mid k, T)\, d\lambda \tag{8}$$

is used to compute upper and lower confidence limits and a medium value for λ. The medium provides a point estimate for λ, while the confidence limits are a measure of the uncertainty of this point estimate. For all the illustrations used in this study, upper and lower 80% confidence limits are employed–defining an 80% probability that the true λ is between them. Integral (8) is evaluated for values of λ' ranging from zero to one, generally in intervals of 0.01. Where the values of the integral is closest to 0.1, the upper limit of integration λ' is printed out as the lower 80% confidence limit. The medium is that λ' where the integral value is closest to 0.5, and the upper 80% confidence limit that λ' where the value of the integral is closest to 0.9 (Figure 4).

Data is generated for the Bayes analysis by the same simulation process used for the Jelinski–Moranda analysis. The Bayes k and the Jelinski–Moranda n are the same, both equal to the number of detected errors up to the Bayes T, which is the sum of the Jelinski–Moranda x_i.

Figure 5 is a plot of a typical Bayes analysis using the model just described. As in Figure 3, the horizontal axis represents the number of errors already detected and the vertical axis represents the computed values of λ. The curves, from the bottom, represent the lower confidence limit, median and upper confidence limit respectively. The true rate, λ_o, is shown. In this case λ_o obeys the Jelinski–Moranda assumption of immediate error correction. The confidence limits converge toward a given value as more errors are discovered, narrowing the range in which the true λ is likely to be found.

PROCEDURE FOR COMPARISON MODELS

For each comparison between these models, a series of "failure times" was generated and stored on tape so that the identical data could be analyzed using both models.

Figure 6 is an overlay of Figures 3 and 5: a direct comparison of the Jelinski-Moranda and Bayes model estimates of λ for the same input data. In this case the data was generated using Equation (4): the Jelinski-Moranda assumption of immediate error correction holds. As might be expected, the more data is available the closer both models estimate to the "true" λ_o.

However, as n approaches 100 (the input N_o) the Bayes model converges to approximately halfway between the original λ_o and zero (the theoretical λ when all errors have been corrected).

Figure 7 is a similar run with different input data. Again, the x_i are computed using the Jelinski-Moranda hypothesis of immediate error correction; again the Bayes model converges to a value higher than λ_o as errors are corrected; again there is a considerable range of n where no Jelinski-Moranda estimate is available. Comparison of Figures 6 and 7 shows that the Jelinski-Moranda value does not appear where λ as indicated by the Bayes model is an increasing function of n. This corresponds to an increase in the rate of error discovery per unit time—directly contradicting the Jelinski-Moranda hypothesis of decreasing λ as errors are corrected. Figure 7 also plots the "arithmetic mean" defined as n/T or failures per unit time. This curve closely follows the Bayes median.

MODEL COMPARISON FOR VARIOUS INPUT HYPOTHESIS

Figure 8 is another run in which the Jelinski-Moranda hypothesis was followed in generation of input data. The first "failure" occurred at an abnormally long time after start of testing, as compared to λ_o. As in all other runs, the time to the first failure is used to compute the first Bayes prior, and thus the prior λ is abnormally low. As further "failures" occurred, the rate λ was observed as increasing since it had been low at the start. The Bayes model followed this observed increase and converged toward the given λ_o. However no solution was obtained from the Jelinski-Moranda model at any point of this run. This is further indication that the Jelinski-Moranda model does not provide any estimate of λ when the trend of the rate is upward.

In all the examples presented so far, the input to the simulation process has conformed to the Jelinski-Moranda assumption that a software error is corrected immediately upon detection. However, in the "real world" this may not always be true: an attempted correction may not remove an error, or may introduce a new error. The simulation process was therefore modified to allow for this possibility.

In Figure 9 is shown the result of a run in which, upon detection of a software error, a 75% probability was assumed that the error would be corrected. The other 25% of the time the correction attempt was unsuccessful: the number of errors remaining in the program was unchanged.

To provide this hypothesis, a random number was generated; if this number was less than or equal to 0.75 the number n' of errors corrected would decrease by 1, otherwise n' would remain unchanged. The failure rate λ_o in Equations (4) and (5) would then be dependent on n', the number of errors actually corrected, which may now be different from n (or i), the number of software-related system failures observed (Figure 10).

$$\lambda_o = \phi_o(N_o - (n' - 1)) \tag{9}$$

$$r_i = 1 - e^{-x_i \lambda_o(i)} \tag{10}$$

No solution was found anywhere on this run by the Jelinski-Moranda model. In the several runs made using this hypothesis, no Jelinski-Moranda solutions were found except in those cases where

the immediate correction hypothesis was obeyed for the first several iterations ($n' = n$); the Jelinski-Moranda model would produce an estimate of λ for these iterations but not after the first instance of non-correction. Once the model stopped producing estimates, it would not resume at any time. The Bayes model is shown to converge – and since the decrease of λ_0 is slower than before, the Bayes model here used, which searches for a constant failure rate, converges "better" to λ_0 than it did when the Jelinski-Moranda hypothesis was followed.

Figure 11 shows a run similar to Figure 9, but in which the probability existed of the introduction of further errors during the software correction process. A random number determined whether n' as defined above decreased (error corrected), increased (error not corrected and new error introduced) or remained the same (error not corrected, or corrected but new error introduced) (Figure 12). Equations (9) and (10) were again used, once an n' was established, to generate "failure times". The result is the same as the previous run: the Jelinski-Moranda model is usable to handle this deviation from its internal assumptions and does not reach a solution anywhere.

In another tested hypothesis, correction of errors follows the Jelinski-Moranda assumption for the first fifteen errors detected: each is corrected immediately. The next five errors are not corrected: five software-related system failures occur and no successful error correction is made. After these five failures, a correction is finally achieved and the process again follows the Jelinski-Moranda assumption. In Figure 13, λ_0 is shown to decline from $n = 1$ to $n = 15$. From $n = 15$ to $n = 20$, λ_0 remains constant (the number of errors corrected, n', remains at 15 while the number of failures, n, increases). After $n = 20$, λ_0 declines again, at the same rate as before $n = 15$ as all errors are again corrected immediately. The Jelinski-Moranda model produces estimates of λ up until $n = 15$ – the point at which its hypothesis is violated. Thereafter, it does not converge to a solution – even after the region of non-correction has been left. Again the Bayes model is shown as better able to follow a changing failure rate, even though it does not conform to the Bayes internal hypothesis of a constant rate.

Figure 14 is a result of using a hypothesis similar to that of Figure 13, but with additional errors introduced. From $n = 1$ to $n = 20$, the Jelinski-Moranda hypothesis is followed. From $n = 21$ to $n = 25$, one new error is introduced at each failure: as n increased by 1, n' decreases by 1. After $n = 25$, the Jelinski-Moranda hypothesis is again followed. The graph shows λ_0 decreasing until $n = 20$, then increasing until n reaches 25, after which it decreases again at the same rate as before. As in Figure 13, the Jelinski-Moranda model produced estimates of λ until the point at which its hypothesis was violated, and did not converge again even once its hypothesis was again valid.

In Figure 15, an opposite perturbation was applied to the Jelinski-Moranda hypothesis. A correction was assumed to follow each error detection; however it was possible to correct more than one error at a time. (A second error may be discovered in examination of the code in the process of correction of the original error which caused a failure.) If one error was corrected, n' increases by 1; if two errors were corrected at one time, n' increases by 2 as n increases by 1. Here the Jelinski-Moranda model was able to reach estimates of λ although they appear to be uniformly low; the Bayes model converges to a high value.

ASSERTIONS RESULTING FROM COMPARISON OF MODELS

It has been shown that the Jelinski-Moranda model will provide an estimate of λ when the assumptions underlying the model are valid, but any perturbation of these assumptions which has the effect of increasing the rate λ_0 above that otherwise encountered will cause this model not

to converge to an estimate. On the other hand, the Bayes constant-rate model which is compared to it will always provide a point estimate and confidence limits, but may converge to a value above λ_0 as n increases.

Even where the Jelinski-Moranda hypothesis holds, but the actual rate of error detection is shown to (temporarily) increase, the Jelinski-Moranda model will not reach a solution (Figures 6, 7, 8). The assertion suggests itself that this non-convergence of the Jelinski-Moranda model may be inherent in the definition of the model; that it may be proven, from the definition of the model, that it will not converge if the trend in the x_i is decreasing, indicating that failure occurrence frequency is increasing.

Sukert, et al. [5] have demonstrated that the maximum likelihood estimator of the Jelinski-Moranda N does not exist in the case where the time interval x_i between error detections is constant or decreasing as n increases, a finding which agrees with the assertions made here that no Jelinski-Moranda solution is found when the trend of the x_i is decreasing (corresponding to an increase in rate of error detection).

CONCLUSIONS: FURTHER DEVELOPMENT

The Bayes model, even in the current constant-rate assumption, has been shown able to respond to perturbations in error discovery rates which cause the Jelinski-Moranda model to fail to reach an estimate of λ.

The Jelinski-Moranda model is one of several error count models, some of which assume an increasing rate and some a decreasing. Moranda [6] has discussed the application of several of these models, but none of them is suitable as a single reliability model which will follow any change in rate of failure occurrence or error detection. Littlewood [3] and Goel [4] have developed more flexible Bayes models, allowing increasing and decreasing λ_0. Development is continuing at CRC on modifications to the Bayes model which will permit it to follow a declining rate trend if there is in fact such a trend. Such a model would recognize a trend and modify the prior in accordance with the trend, so as not to be influenced by extreme fluctuation of rate early in the software development process, although in no way would it stop production of estimates should the trend be reversed.

REFERENCES

1. Jelinski, Z. and Moranda, P. B., "Software Reliability Research," in Statistical Computer Performance Evaluation, Ed: W. Freiberger, Academic Press, New York, pp. 465–484, 1972.

2. Thompson, W. E. and Chelson, P. O., "On the Specifications and Testing of Software Reliability," 1980 Proceedings Annual Reliability and Maintainability Symposium, IEEE Catalog Number 80 Ch 1513-1R; January 1980, pp. 379-383.

3. Littlewood, B. and Verall, J. L., "A Bayesian Reliability Growth Model for Computer Software," J. Royal Statistical Soc. (Series C, Applied Statistics), Vol. 22, 3, pp. 332-346, 1973.

4. Goel, A. K. and Okumoto, K., "Bayesian Software Prediction Models, Vol. 1: An Imperfect Debugging Model for Reliability and Other Quantitative Measures of Software Systems," RADC-TR-78-155, Rome Air Development Center, New York, 1978.

5. Sukert, A., Schafer, R., and Angus, J., "Software Reliability Model Validation," 1980 Proceedings Annual Reliability and Maintainability Symposium, January 1980, pp. 191-199.

6. Moranda, P., "Event-Altered Rate Models for General Reliability Analysis," IEEE Transactions on Reliability, Vol. R-28, No. 5, December 1979, pp. 376-381.

BIOGRAPHIES

Martin H. Horn
Columbia Research Corporation
2531 Jefferson Davis Highway
Arlington, Virginia 22202 USA

Mr. Horn is a member of the technical staff of Columbia Research Corporation. He is responsible for developing automated data processing application programs in the areas of hardware and software reliability, availability and maintainability. Among these have included several applications of computer simulation for reliability/availability measurement. He received his M.S. (physics) from the University of Maryland in 1974 and his B.A. (physics) from Johns Hopkins University in 1972.

William E. Thompson
Columbia Research Corporation
2531 Jefferson Davis Highway
Arlington, Virginia 22202 USA

Dr. Thompson is the Director of Reliability Activity of Columbia Research Corporation. He and his technical staff are responsible for the development, acquisition management, and quality assurance techniques for computer hardware and software systems for various DOD and commercial agencies. Dr. Thompson has also taught and performed research in mathematical statistics and reliability. He has published more than forty papers related to various aspects of software reliability and testing. He received his Ph.D. (mathematics) from Purdue University in 1959, his M.S.E.E. from the University of New Mexico in 1965, his M.S. (statistics) from Purdue University in 1956, and his B.A. (physics-chemistry) from the University of Evansville in 1952.

THE VIEWGRAPH MATERIALS
for the
M. HORN PRESENTATION FOLLOW

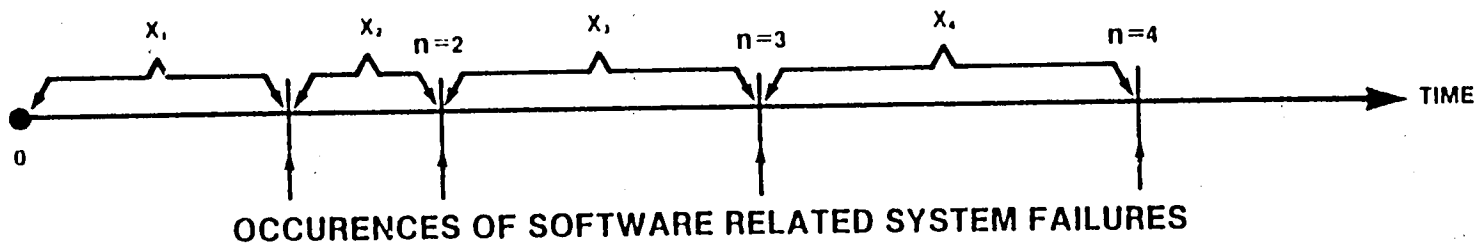


FIGURE 1: Definition of n and x_1

JELINSKI-MORANDA MODEL ASSUMPTIONS

1) EACH ERROR IS CORRECTED IMMEDIATELY UPON DETECTION. NO NEW ERROR INTRODUCED.

2) λ IS PROPORTIONAL TO NUMBER OF ERRORS REMAINING IN SOFTWARE. λ DECLINES BY CONSTANT AMOUNT Φ AFTER EACH ERROR IS CORRECTED.

3) λ IS CONSTANT BETWEEN DETECTIONS OF ERRORS.

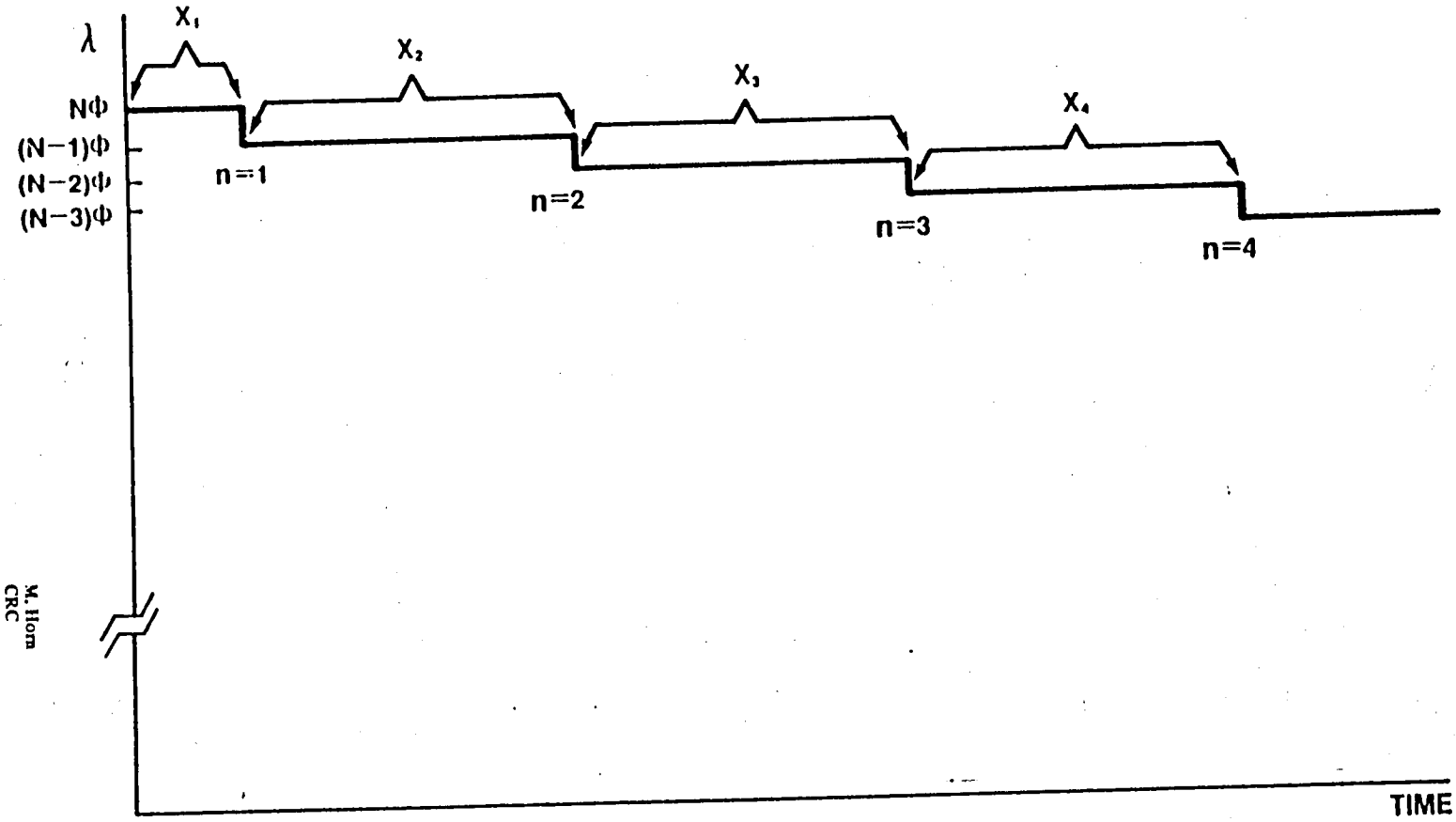


FIGURE 2: Jelinski-Moranda hypothesis

RESULT OF JELINSKI—MORANDA ESTIMATE
N_0=100 Φ_0=.001

λ_0 FOLLOWS JELINSKI—MORANDA HYPOTHESIS OF IMMEDIATE ERROR CORRECTION.

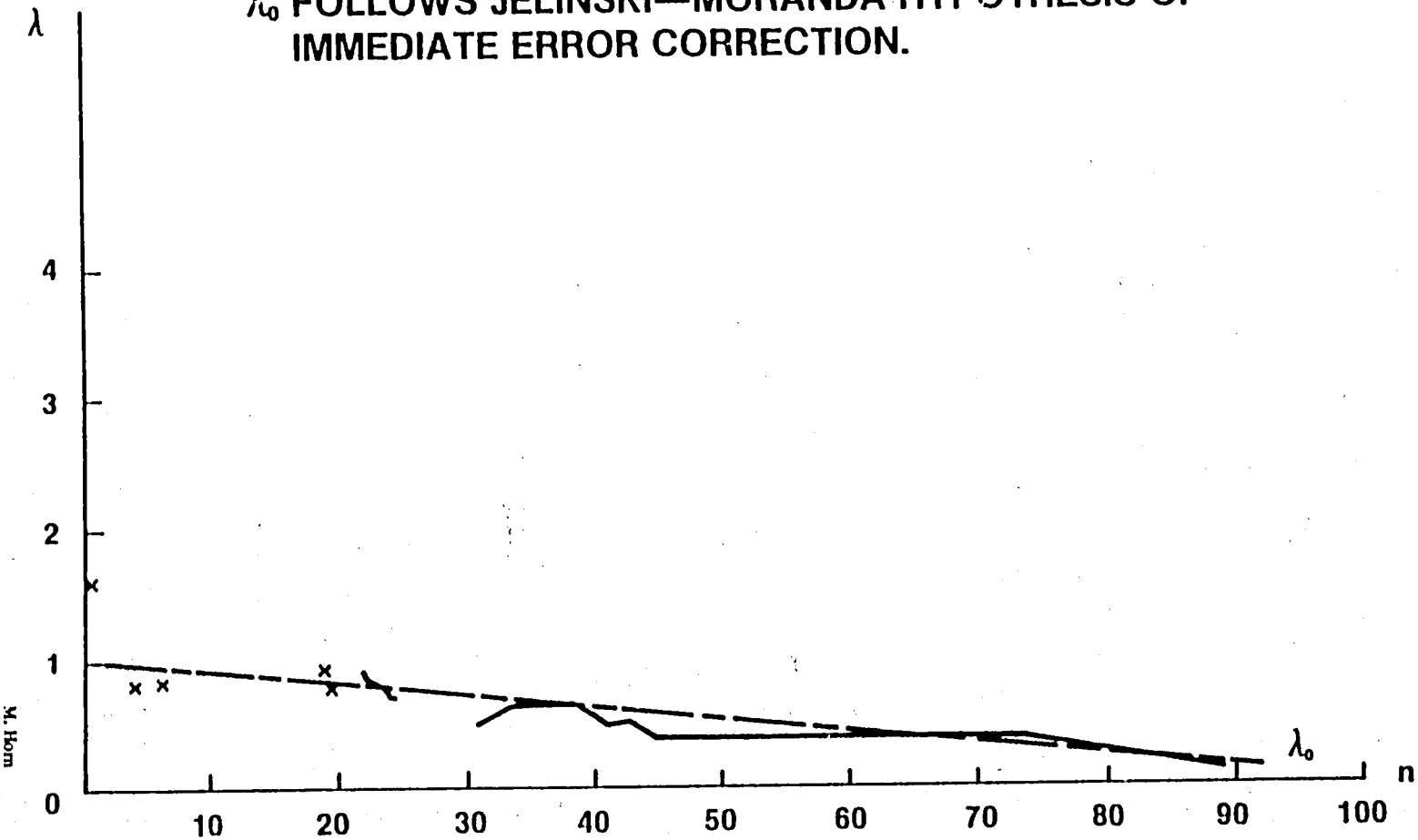


FIGURE 3

NUMBER OF OBSERVED SOFTWARE—RELATED SYSTEM FAILURE

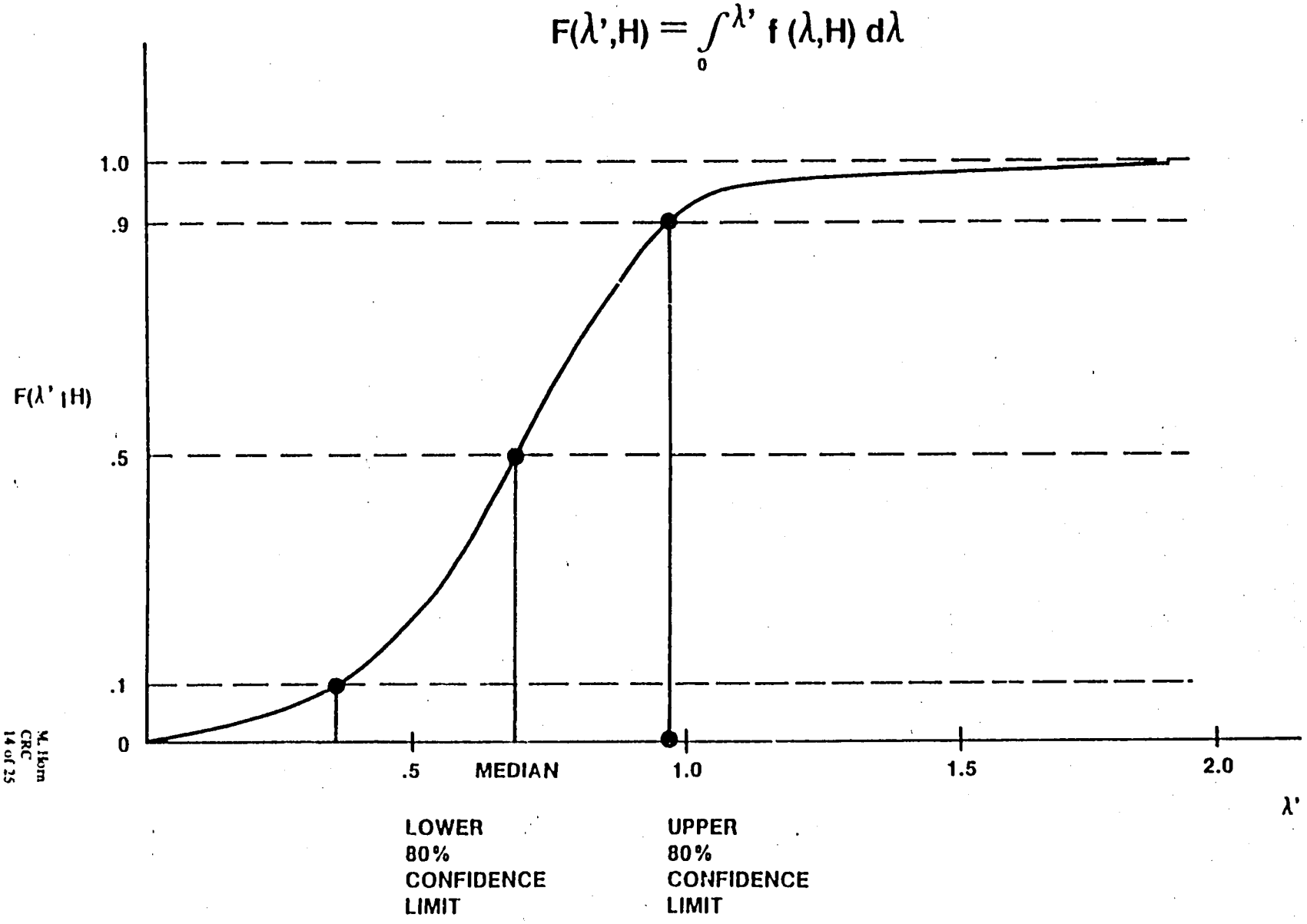


FIGURE 4: Bayes Confidence Limits and Median

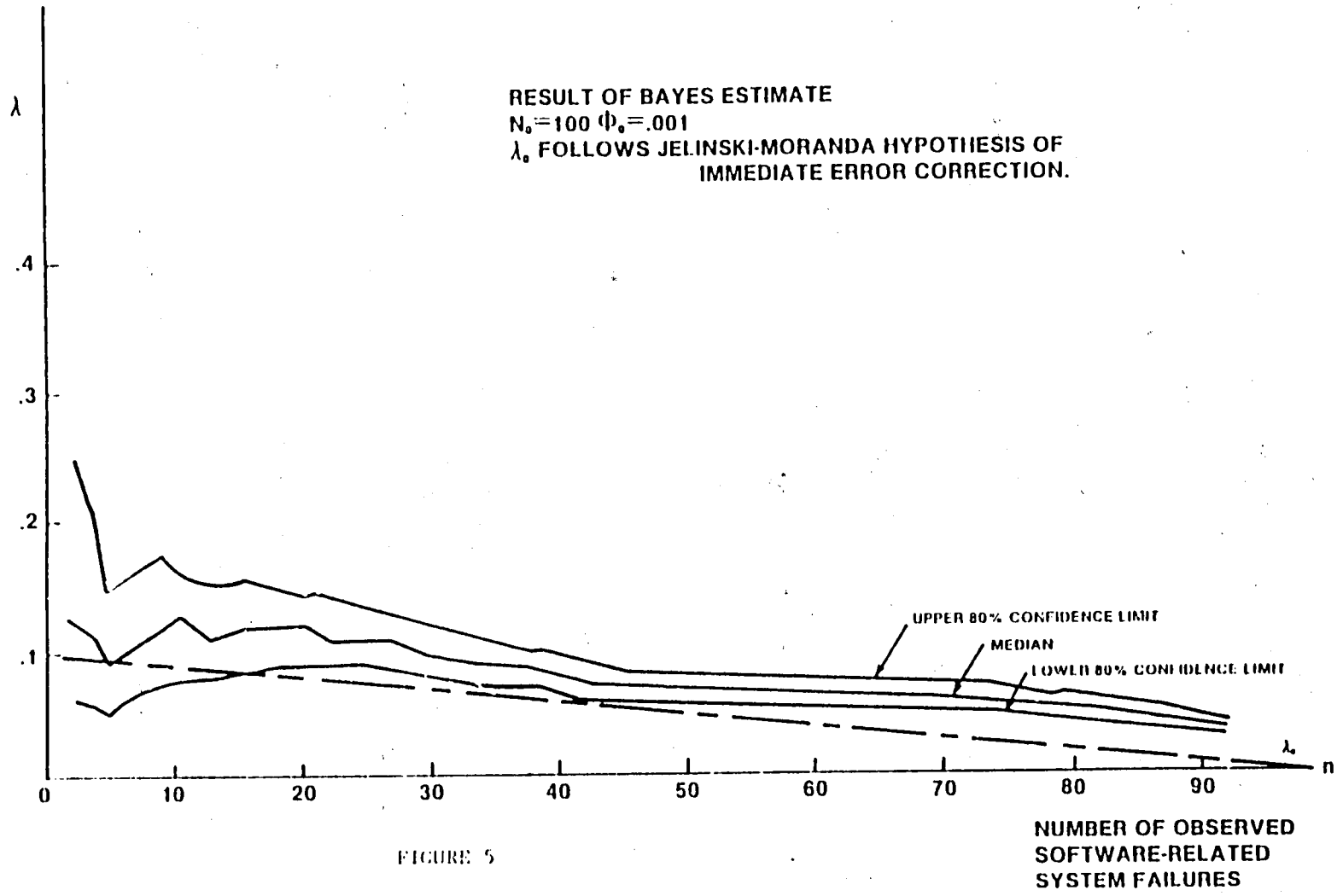

RESULT OF BAYES ESTIMATE
N_0=100 Φ_0=.001
λ_0 FOLLOWS JELINSKI-MORANDA HYPOTHESIS OF IMMEDIATE ERROR CORRECTION.
λ
.4
.3
.2
.1
0
10
20
30
40
50
60
70
80
90
n
UPPER 80% CONFIDENCE LIMIT
MEDIAN
LOWER 80% CONFIDENCE LIMIT
λ_0
NUMBER OF OBSERVED SOFTWARE-RELATED SYSTEM FAILURES

FIGURE 5

COMPARISON OF BAYES AND JELINSKI—MORANDA

N_0=100 Φ_0=001

λ_0 FOLLOWS JELINSKI—MORANDA HYPOTHESIS OF IMMEDIATE ERROR CORRECTION.

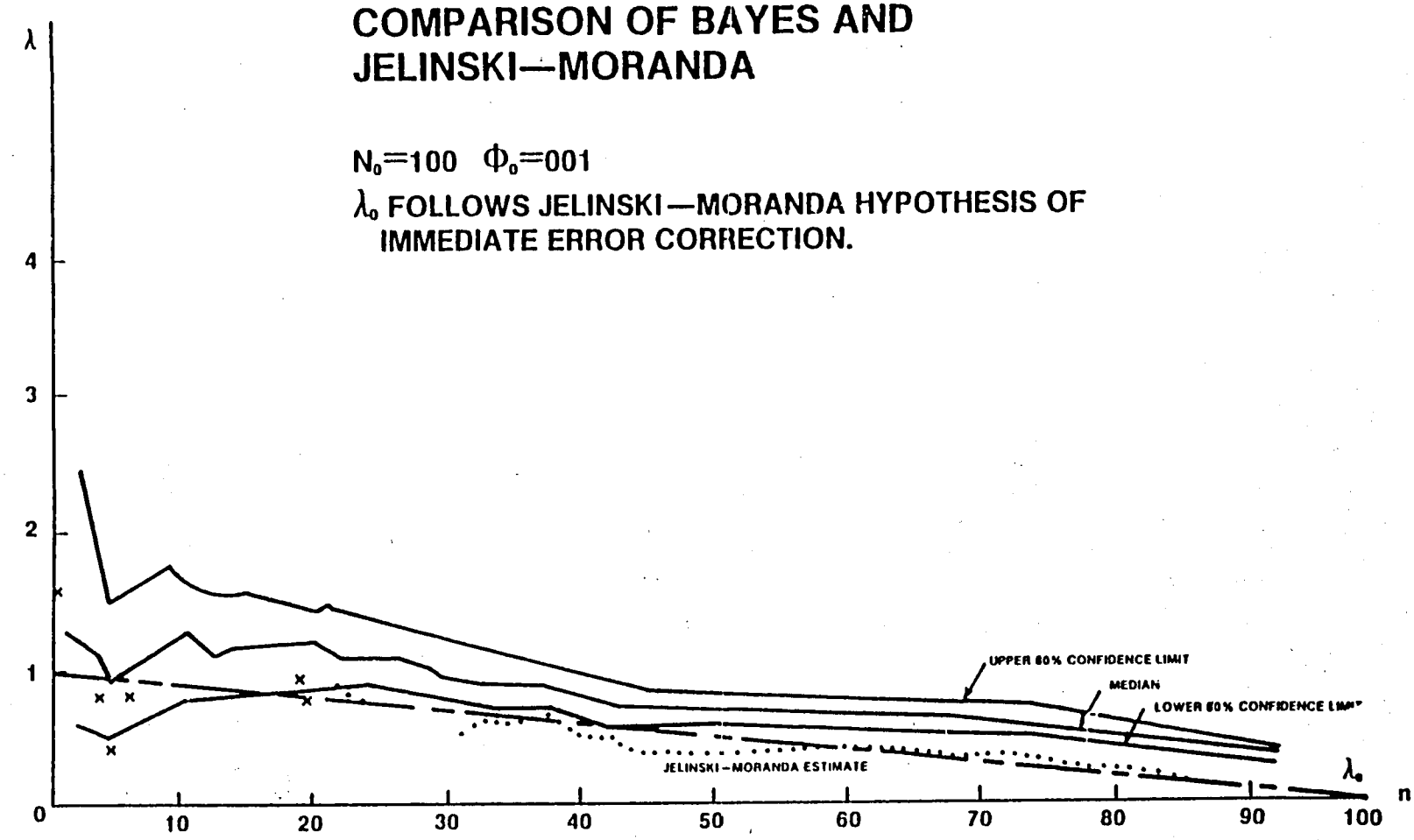
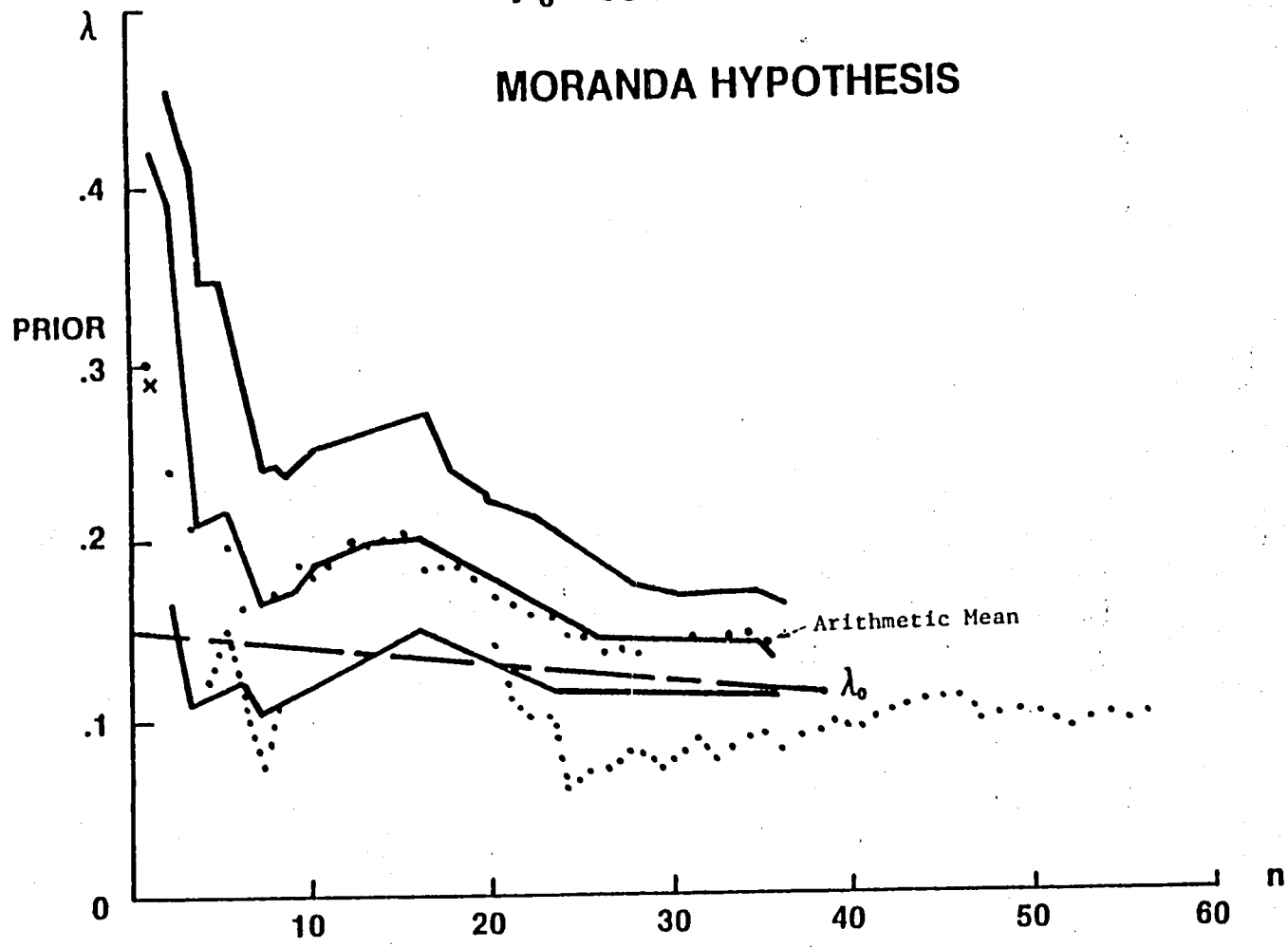


FIGURE 6

NUMBER OF OBSERVED SOFTWARE—RELATED SYSTEM FAILURES



M. Horn
CRC
17 of 25

FIGURE 7

$N_0 = 150$

$\Phi_0 = .001$

λ_0 FOLLOWS JELINSKI—MORANDA HYPOTHESIS

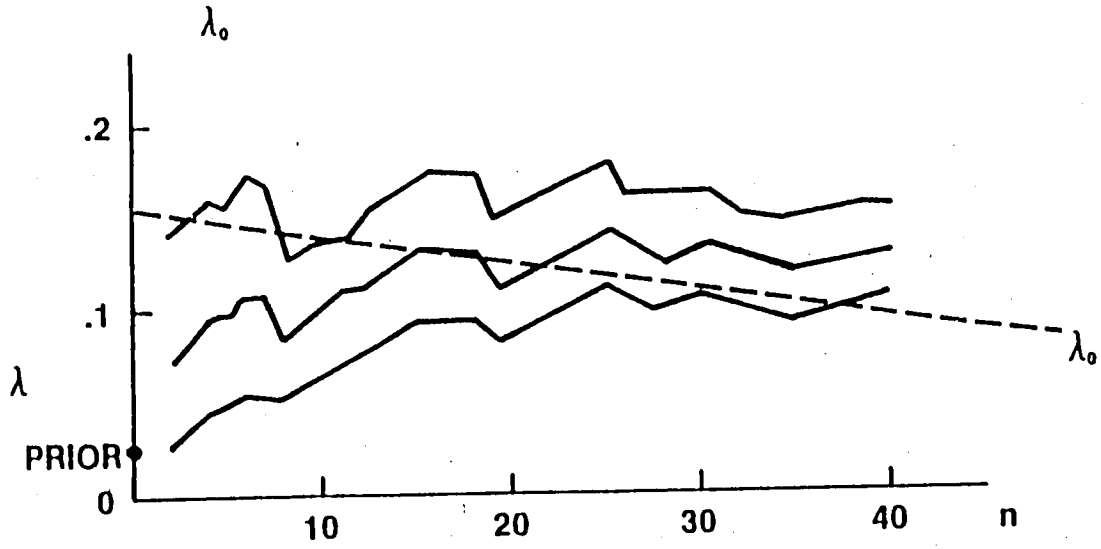


FIGURE 3

$N_0 = 100$

$\Phi_0 = .001$

ERRORS UNCORRECTED AT RANDOM
(25% PROBABILITY OF NO CORRECTION)

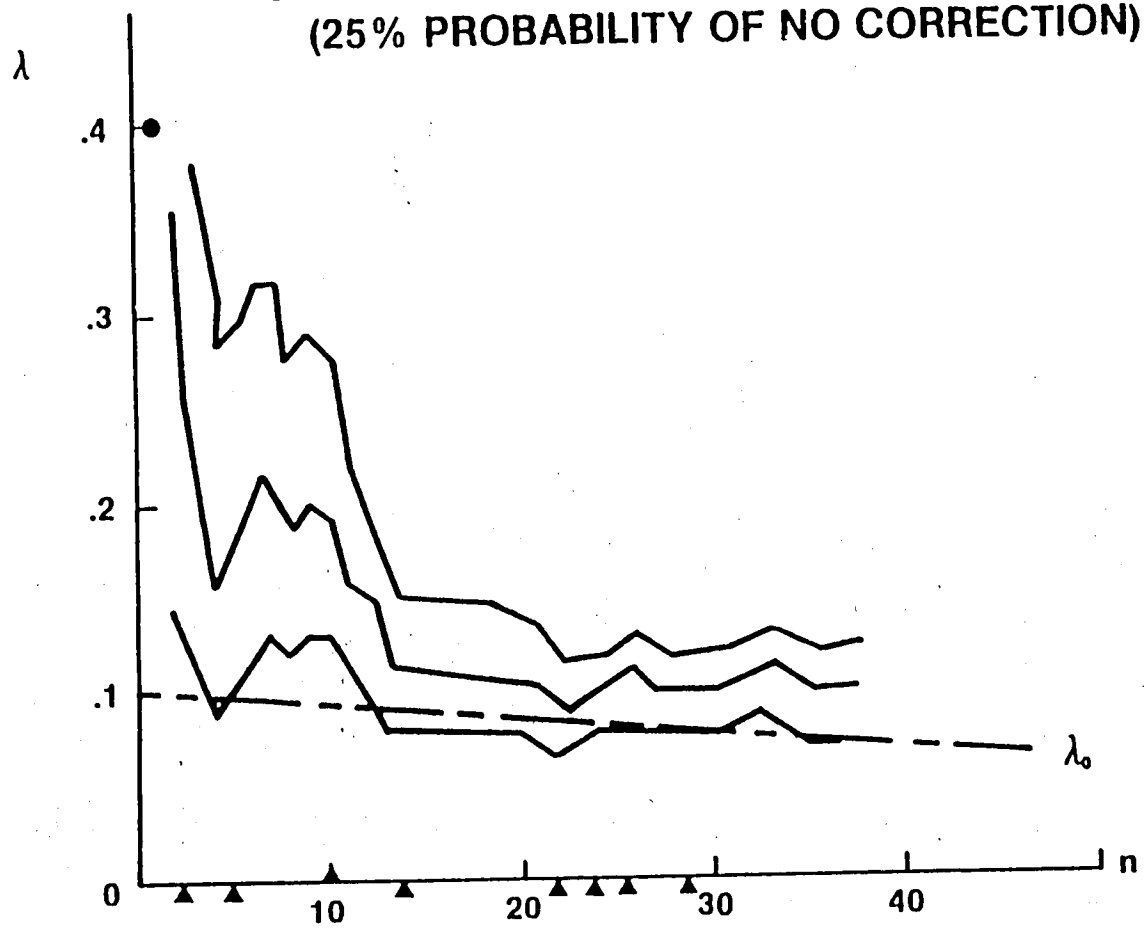


FIGURE 9

NOT ALL ERRORS CORRECTED UPON DETECTION

JELINSKI-MORANDA HYPOTHESIS HOLDS ONLY WHEN AN ERROR IS IN FACT CORRECTED.

λ IS NOW PROPORTIONAL TO NUMBER OF ERRORS REMAINING–WHICH MAY NOT BE EQUAL TO N-n.

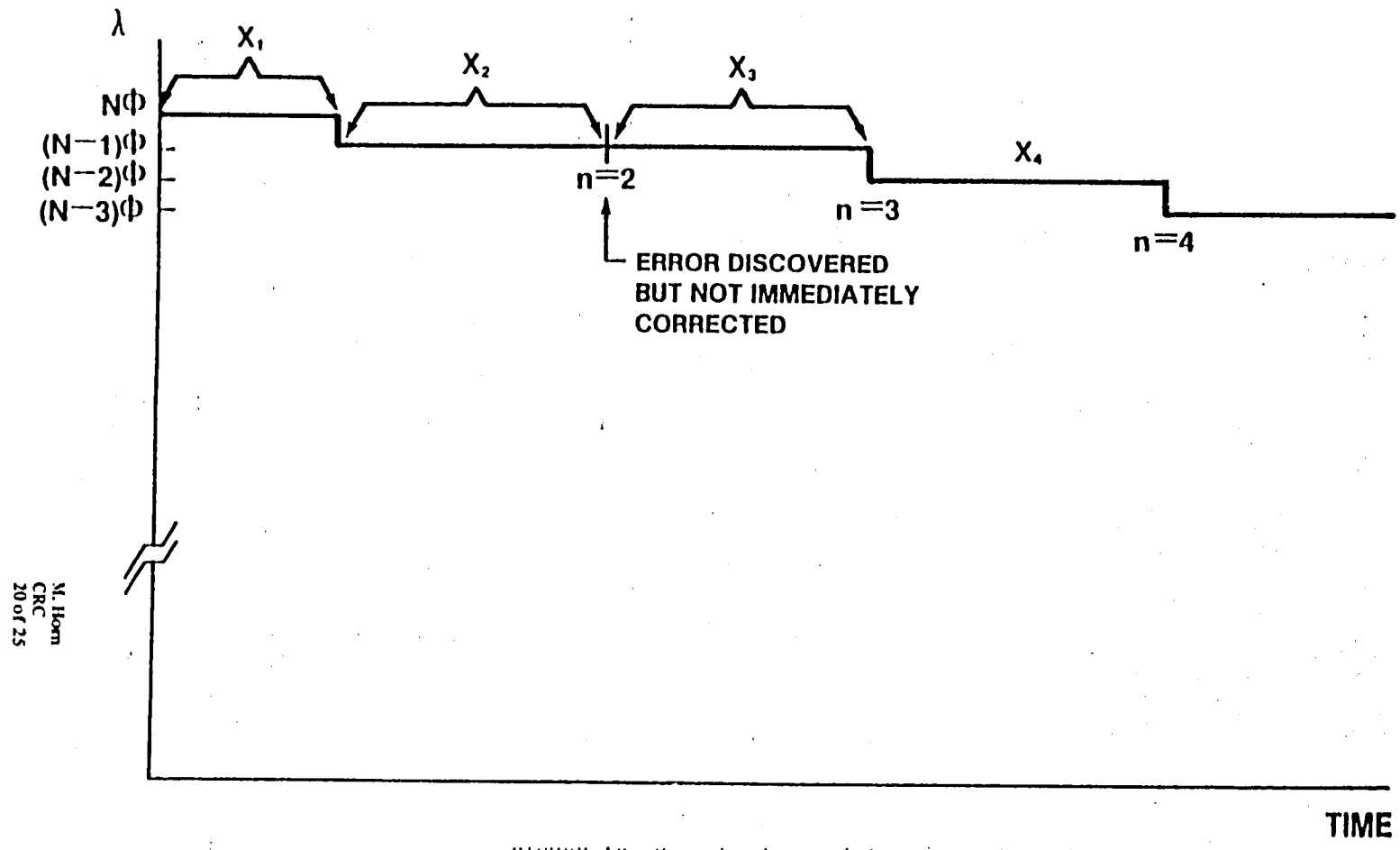


FIGURE 10: Hypothesis used in generation of Figure 9

$N_0 = 150$
$\Phi_0 = .001$

NEW ERRORS INTRODUCED AT RANDOM

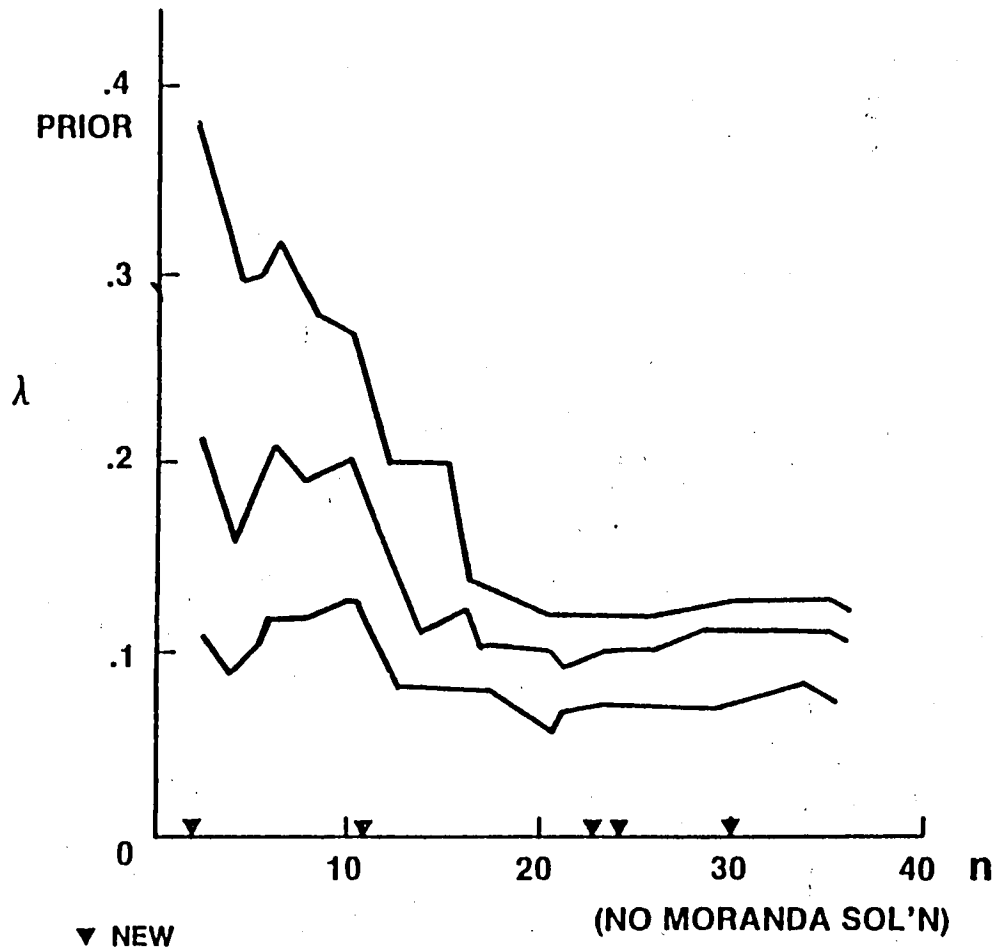


FIGURE 11

NEW ERROR MAY BE INTRODUCED IN CORRECTION PROCESS

INTRODUCTION OF NEW ERROR RAISES λ BY A FACTOR Φ

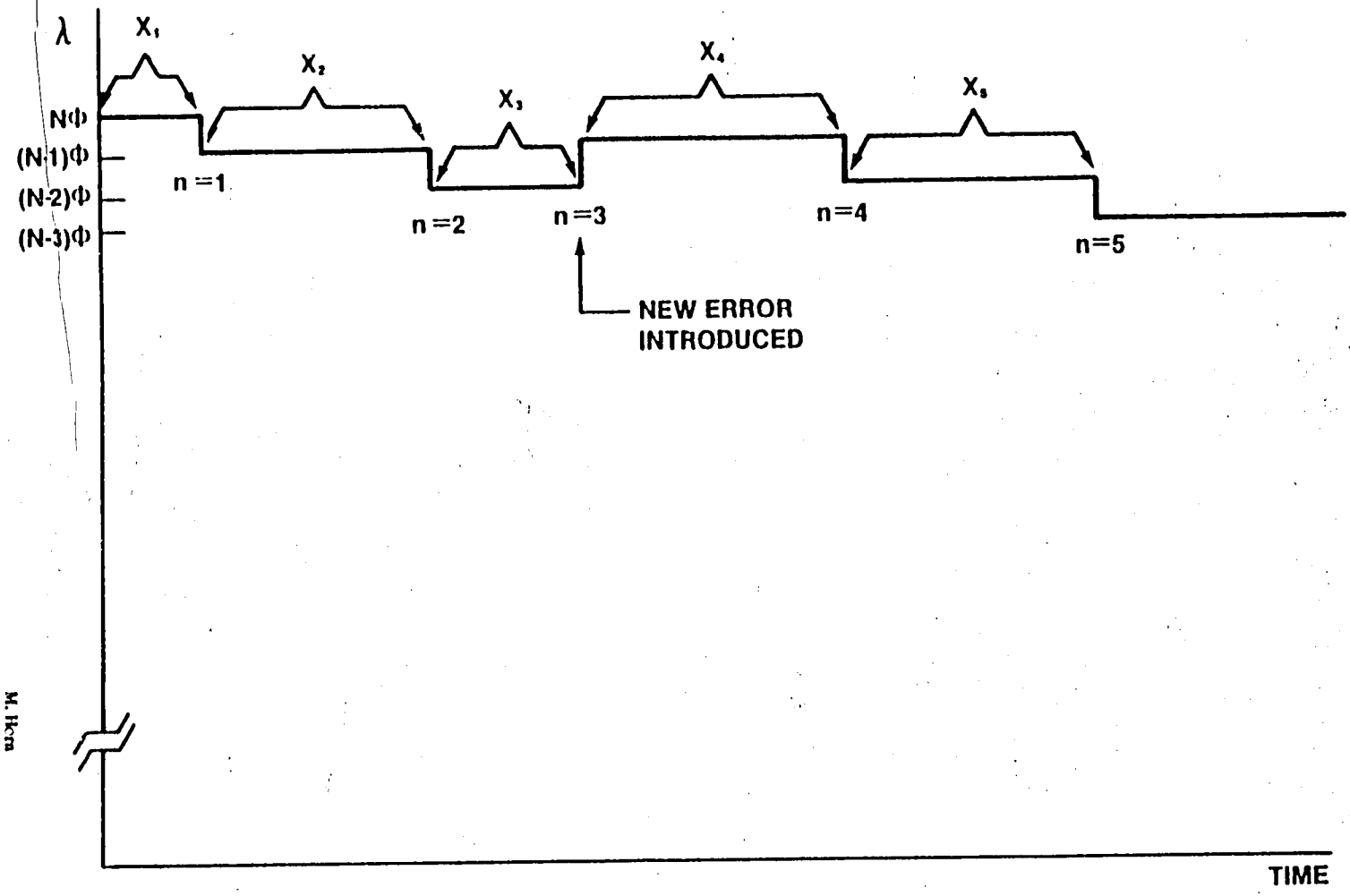


FIGURE 12: Hypothesis used in generation of Figure 11

$N_0=125$

$\Phi_0=.001$

ERROR NOT CORRECTED n =15 TO n=20

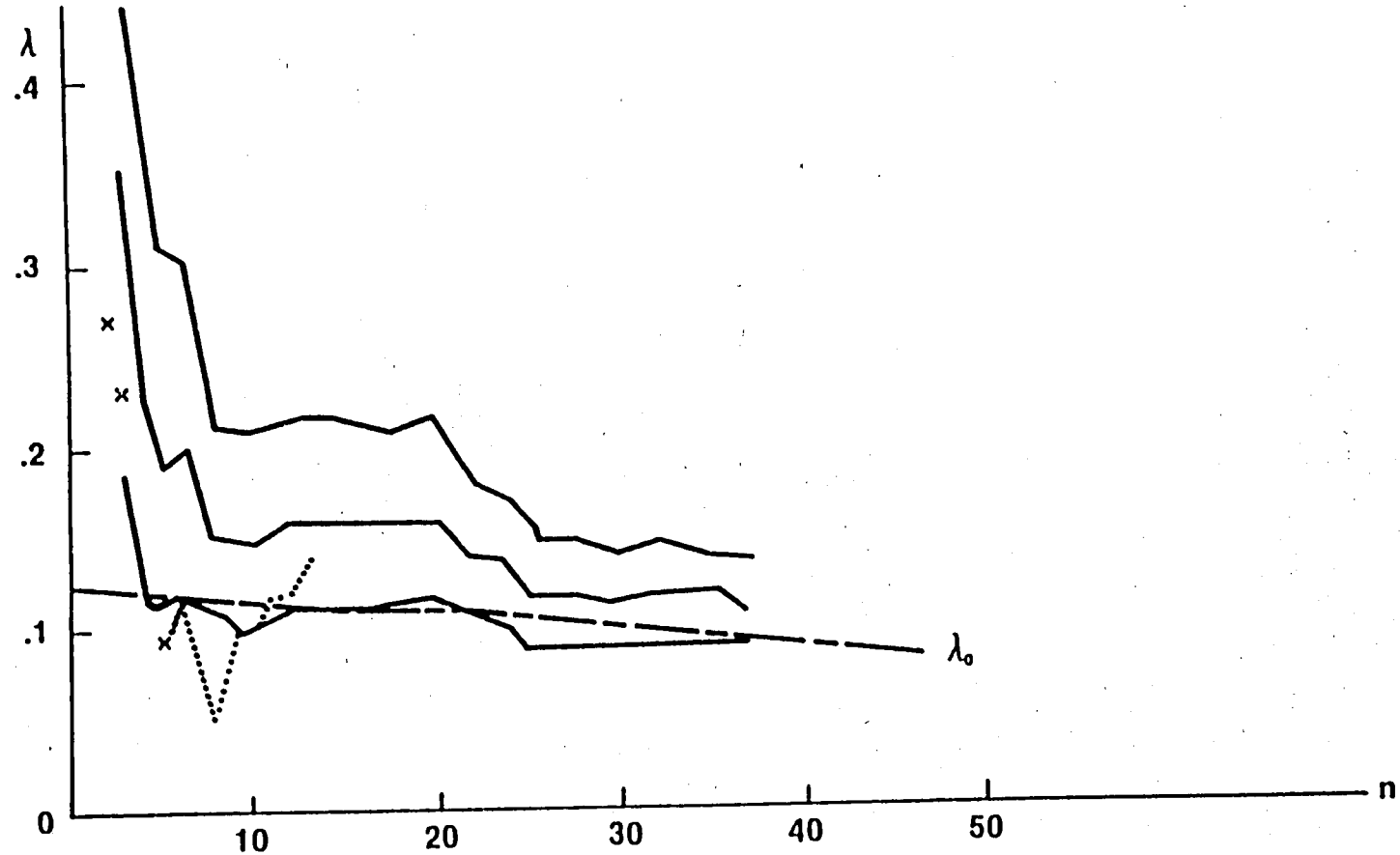


FIGURE 13

$N_0=150$
$\Phi_0=.001$

NEW ERRORS INTRODUCED n=20 TO n=25

J—M SOLUTION FOUND ONLY AT n=1 AND n=8

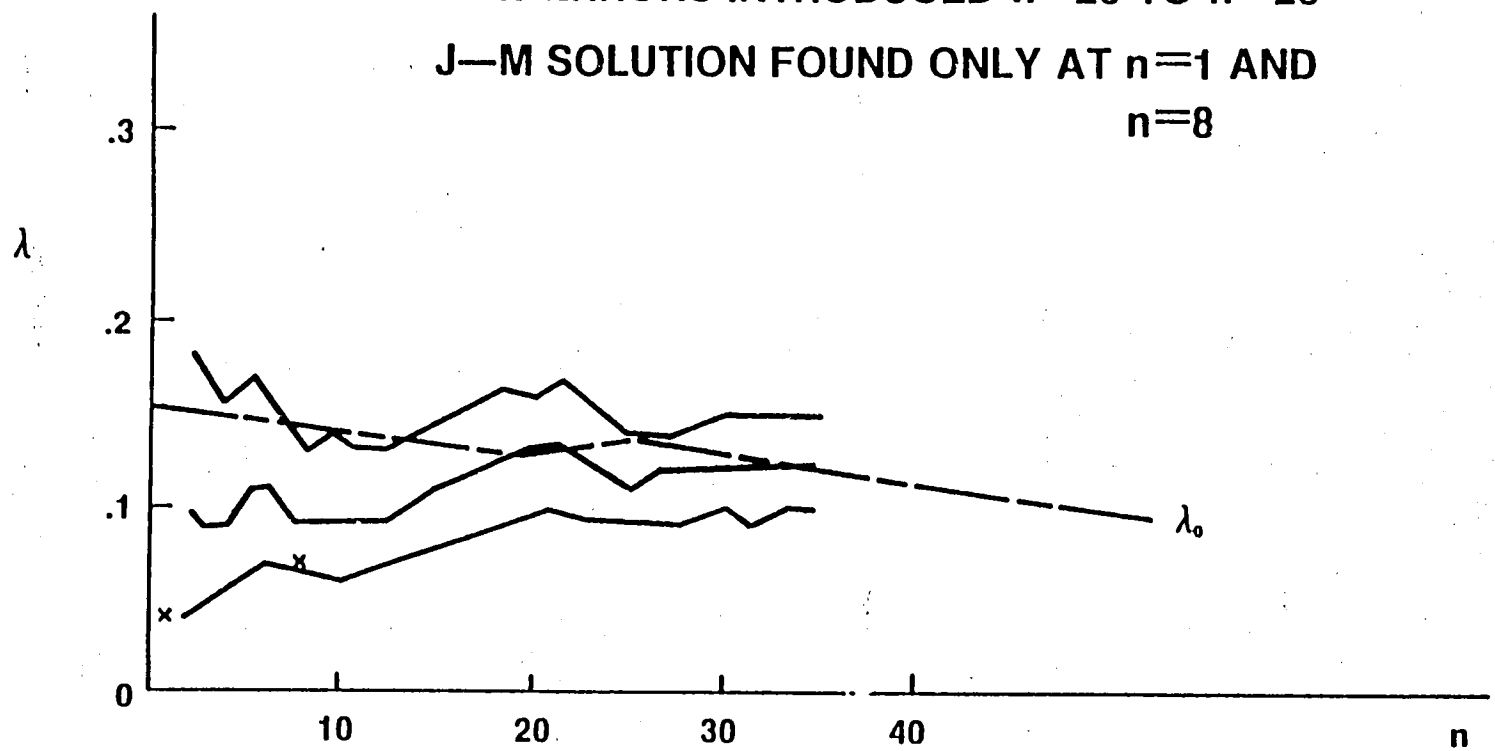


FIGURE 14

N_0=125
Φ_0=.001
POSSIBILITY OF MULTIPLE ERROR CORRECTION

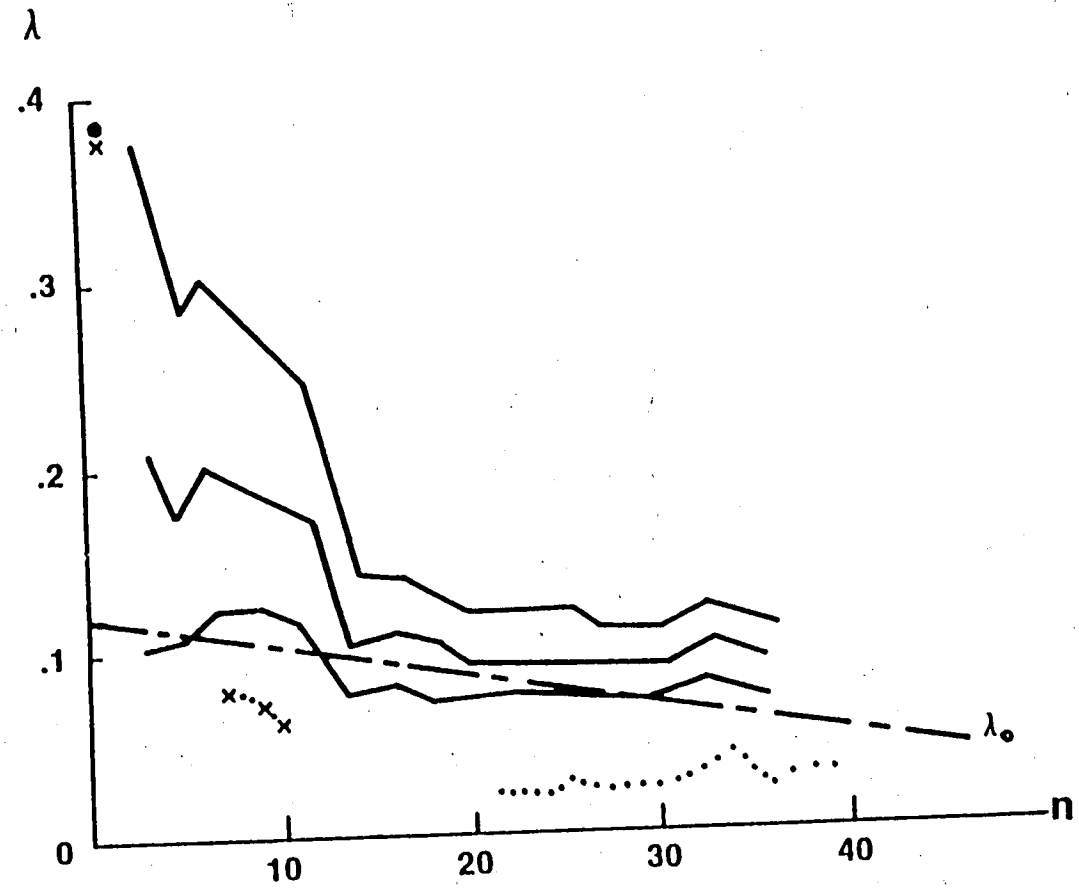


FIGURE 15

PANEL #4
MEASURING THE DEVELOPMENT PROCESS

S. Sheppard/E. Kruesi/B. Curtis, ITT
R. Cruickshank/J. Gaffney, Jr., IBM Federal Systems Division
S. Moy, Logicon, Inc.

MEASURING THE DEVELOPMENT PROCESS:
Symbology and Spatial Arrangement:
Effects on the Comprehension of Software Specifications

S. B. Sheppard, E. Kruesi, and B. Curtis
ITT

The costs and reliability of software development and maintenance are affected by how efficiently programmers can perform their tasks throughout the software life-cycle. Easily understood specifications are a primary criterion for a successful system development effort. This experiment investigated the understandability of various specification formats in a laboratory environment.

Two dimensions of specification formats were examined: the type of symbology and the spatial arrangement of the information.

Three types of symbology were evaluated: natural language, constrained language and ideograms. The natural language consisted of a description of a program in English narrative. The constrained language was a modified version of PDL (Program Design Language). The ideograms were comprised of the standard IBM flow chart symbols.

Three spatial arrangements were also evaluated: sequential, branching and hierarchical. In the sequential versions, both the flow of control and the nesting levels were arranged vertically. (The sequential arrangement is the standard format for presenting PDL and English narrative.) The branching arrangement was similar to a structured flow chart and had a vertical flow of control and horizontal nesting. The heirarchical arrangement was somewhat tree-like in nature with a horizontal flow of control and vertical nesting.

Each of the three types of symbology were presented in three spatial arrangements, resulting in nine specification formats. These nine formats were prepared for each of three small programs (approximately 50 lines of code).

Seventy-two participants were asked to study a set of specifications and then answer a series of questions from the specifications. The questions were presented interactively on a CRT. The answers and response times were recorded automatically. Each participant saw specifications for the three different programs. Across the three programs, each spatial arrangement and each type of symbology was seen once.

Eighteen questions comprised each experimental task. These included: 5 forward-tracing questions, 5 backward-tracing questions and 8 input-output questions. For the forward-tracing questions, the participants were given a set of conditions from the specifications. Their task was to trace through the specifications and find the first statement executed under those conditions. For the backward-tracing questions, they were presented with a statement. Their task was to determine the relevant conditions which held when that statement was executed. For the input-output questions, they were some data and were required to give the output values.

The level of accuracy was high (89%) and did not differ across the nine formats. Both forward- and backward-tracing questions were answered more quickly from specifications presented in constrained language or ideograms than in natural language. Forward-tracing questions were answered most quickly from a branching arrangement, and backward-tracing questions were answered more quickly from branching and hierarchical arrangements. Response times to the input-output questions did not vary significantly as a function of the type of symbology or the spatial arrangement.

When asked which type of symbology they preferred, the majority of the participants preferred the constrained language, while the fewest preferred natural language. In terms of the spatial arrangement, the branching arrangement was the most preferred while the hierarchical arrangement was the least preferred.

These results extend previous research on presentation formats by demonstrating the separate contributions of symbology and spatial arrangement to comprehension.

THE VIEWGRAPH MATERIALS
for the
S. SHEPPARD PRESENTATION FOLLOW

GENERAL ELECTRIC COMPANY  SPACE DIVISION	FIFTH ANNUAL SOFTWARE ENGINEERING WORKSHOP NOVEMBER 24, 1980	INFORMATION SYSTEMS PROGRAMS  SOFTWARE MANAGEMENT RESEARCH

C-3

SYMBOLOGY AND SPATIAL ARRANGEMENT: EFFECTS ON THE COMPREHENSION OF SOFTWARE SPECIFICATIONS

SYLVIA SHEPPARD, ELIZABETH KRUESI AND BILL CURTIS

SPONSOR: OFFICE OF NAVAL RESEARCH
ENGINEERING PSYCHOLOGY PROGRAMS

GENERAL ELECTRIC COMPANY

SPACE DIVISION

OUR APPROACH

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

- VARY THE TYPE OF SYMBOLS
  - IDEOGRAMS
  - CONSTRAINED LANGUAGE
  - NATURAL LANGUAGE
- VARY THE SPATIAL ARRANGEMENT
  - BRANCHING
  - SEQUENTIAL
  - HIERARCHICAL
- RESULT IS NINE DIFFERENT FORMATS

GENERAL ELECTRIC COMPANY

SPACE DIVISION

RESEARCH QUESTION

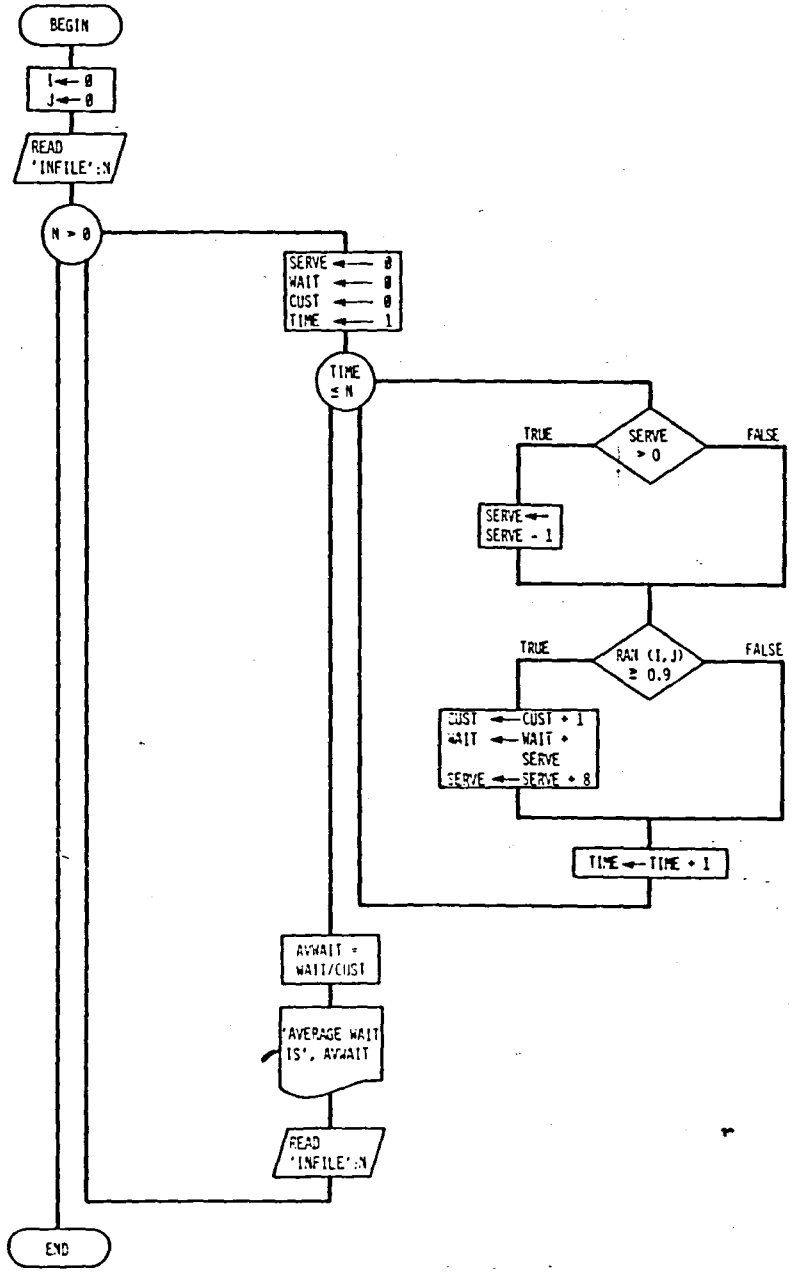
INFORMATION SYSTEMS PROGRAMS

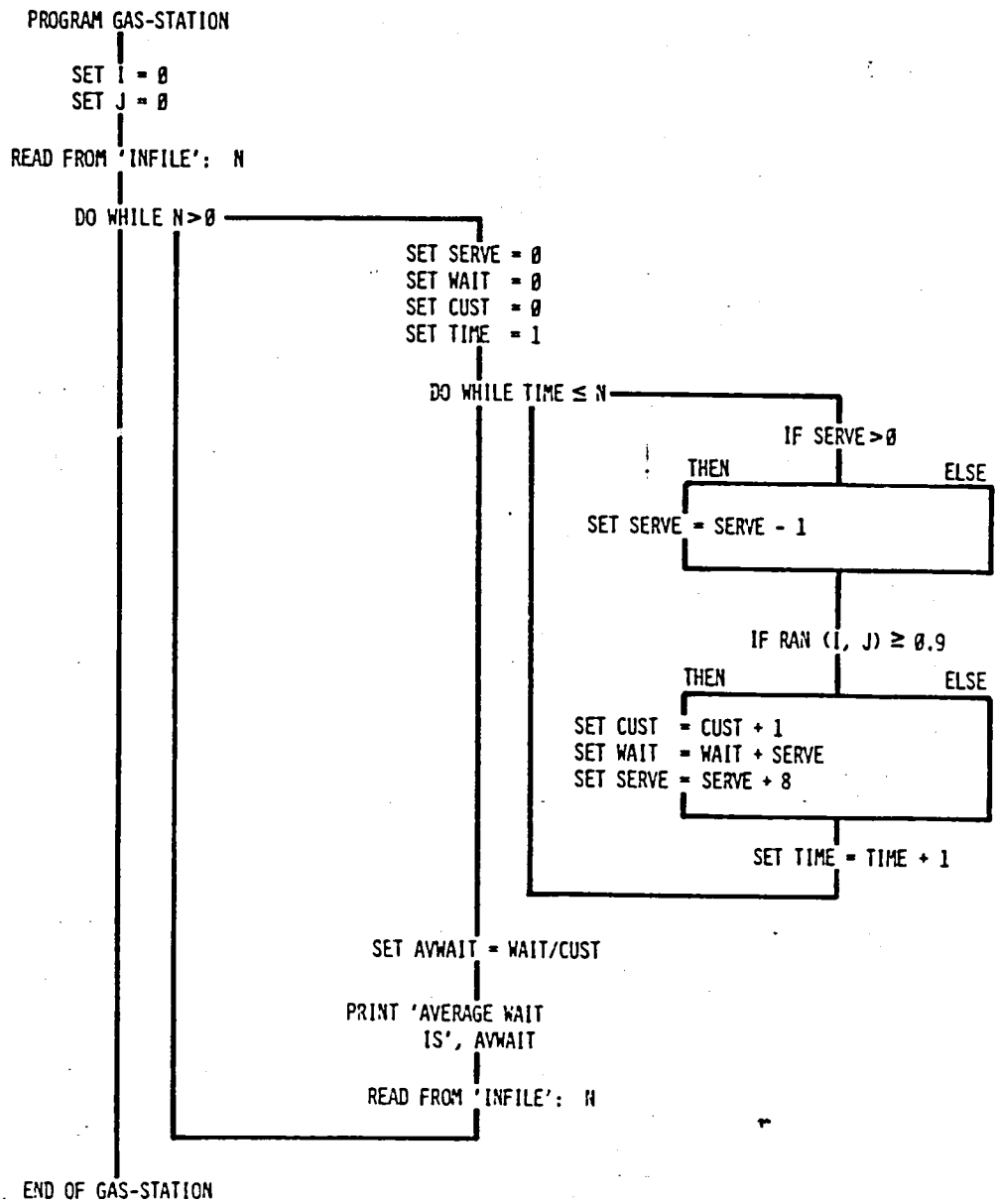
SOFTWARE MANAGEMENT RESEARCH

- WHICH IS BETTER: PROGRAM DESIGN LANGUAGE (PDL) OR FLOWCHARTS?
- ONE APPROACH: COMPARE THESE TWO REPRESENTATIONS ON ONE OR MORE TASKS (E.G., COMPREHENSION, CODING)
- THIS WILL TELL US WHICH IS BETTER BUT NOT WHY.

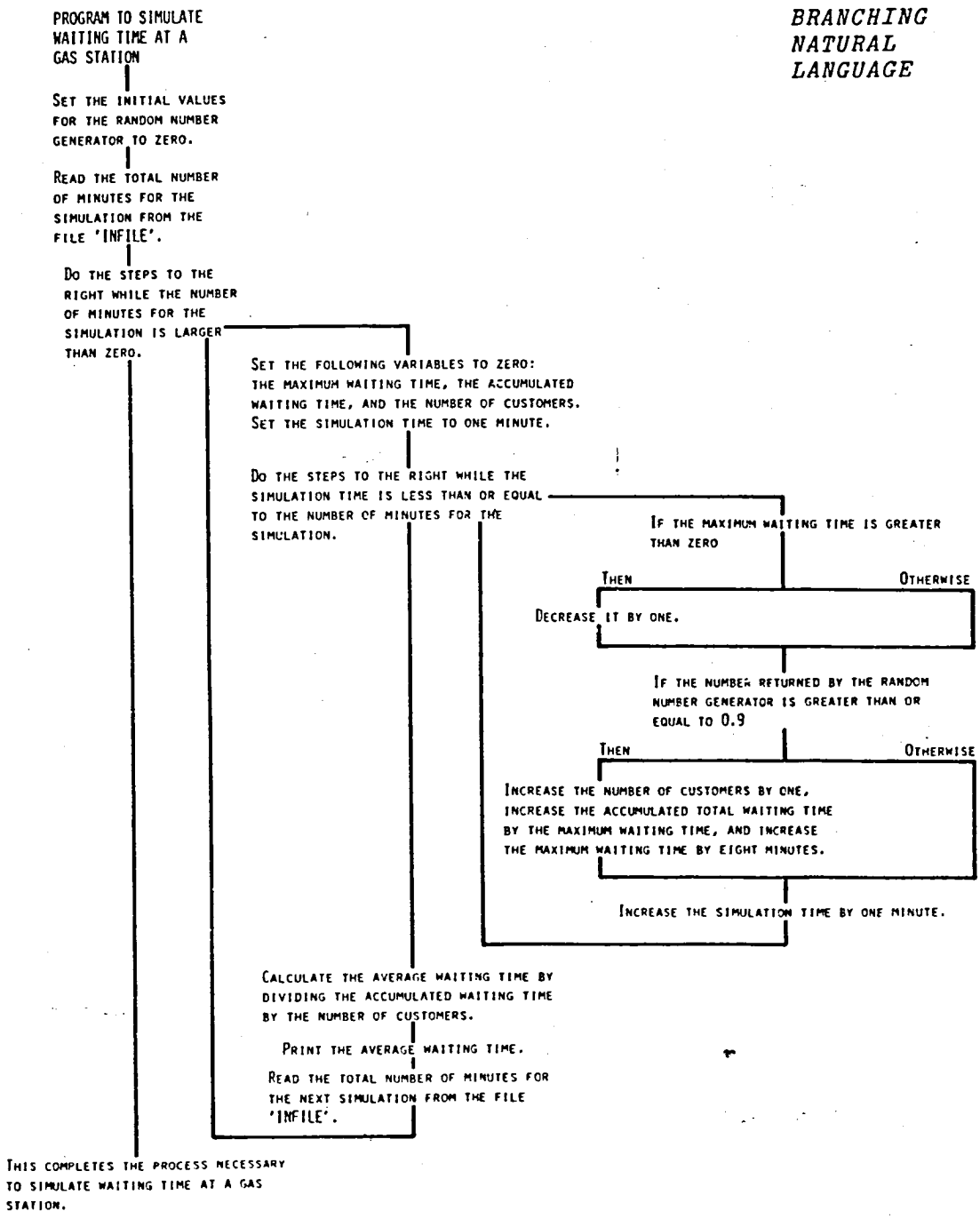
  ARE DIFFERENCES DUE TO:

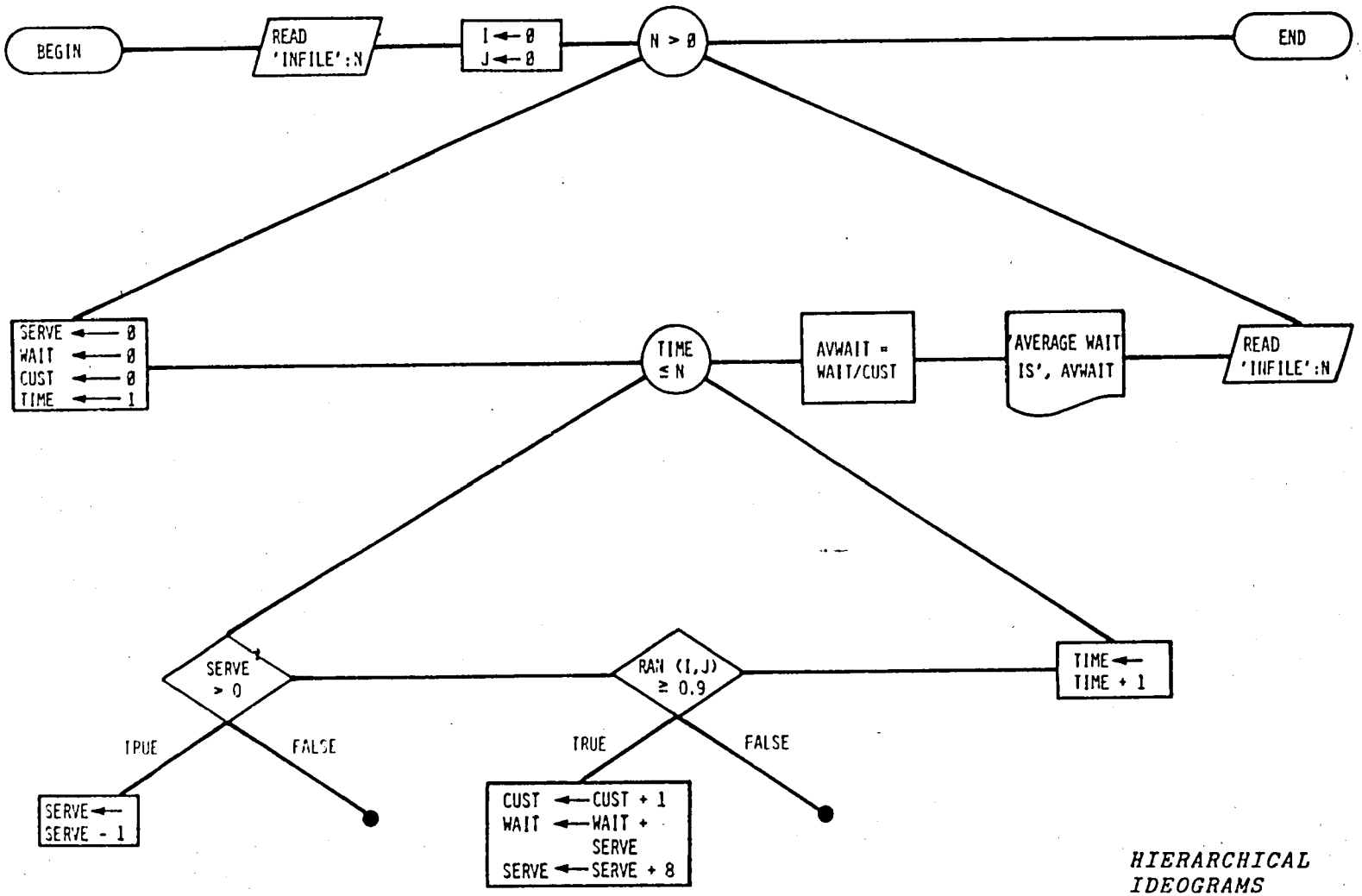
  - TYPE OF SYMBOLS (CONSTRAINED LANGUAGE VS IDEOGRAMS)
  - SPATIAL ARRANGEMENT (SEQUENTIAL VS BRANCHING)


BEGIN
I ← 0
J ← 0
READ
'INFILE':N
N > 0
SERVE ← 0
WAIT ← 0
CUST ← 0
TIME ← 1
TIME
≤ N
TRUE
SERVE
> 0
FALSE
SERVE ←
SERVE - 1
TRUE
RAN (I,J)
≥ 0.9
FALSE
CUST ← CUST + 1
WAIT ← WAIT +
SERVE
SERVE ← SERVE + 8
TIME ← TIME + 1
AVWAIT =
WAIT/CUST
'AVERAGE WAIT
IS', AVWAIT
READ
'INFILE':N
END


PROGRAM GAS-STATION
SET I = 0
SET J = 0
READ FROM 'INFILE': N
DO WHILE N>0
SET SERVE = 0
SET WAIT = 0
SET CUST = 0
SET TIME = 1
DO WHILE TIME ≤ N
IF SERVE>0
THEN
ELSE
SET SERVE = SERVE - 1
IF RAN (I, J) ≥ 0.9
THEN
ELSE
SET CUST = CUST + 1
SET WAIT = WAIT + SERVE
SET SERVE = SERVE + 8
SET TIME = TIME + 1
SET AVWAIT = WAIT/CUST
PRINT 'AVERAGE WAIT IS', AVWAIT
READ FROM 'INFILE': N
END OF GAS-STATION

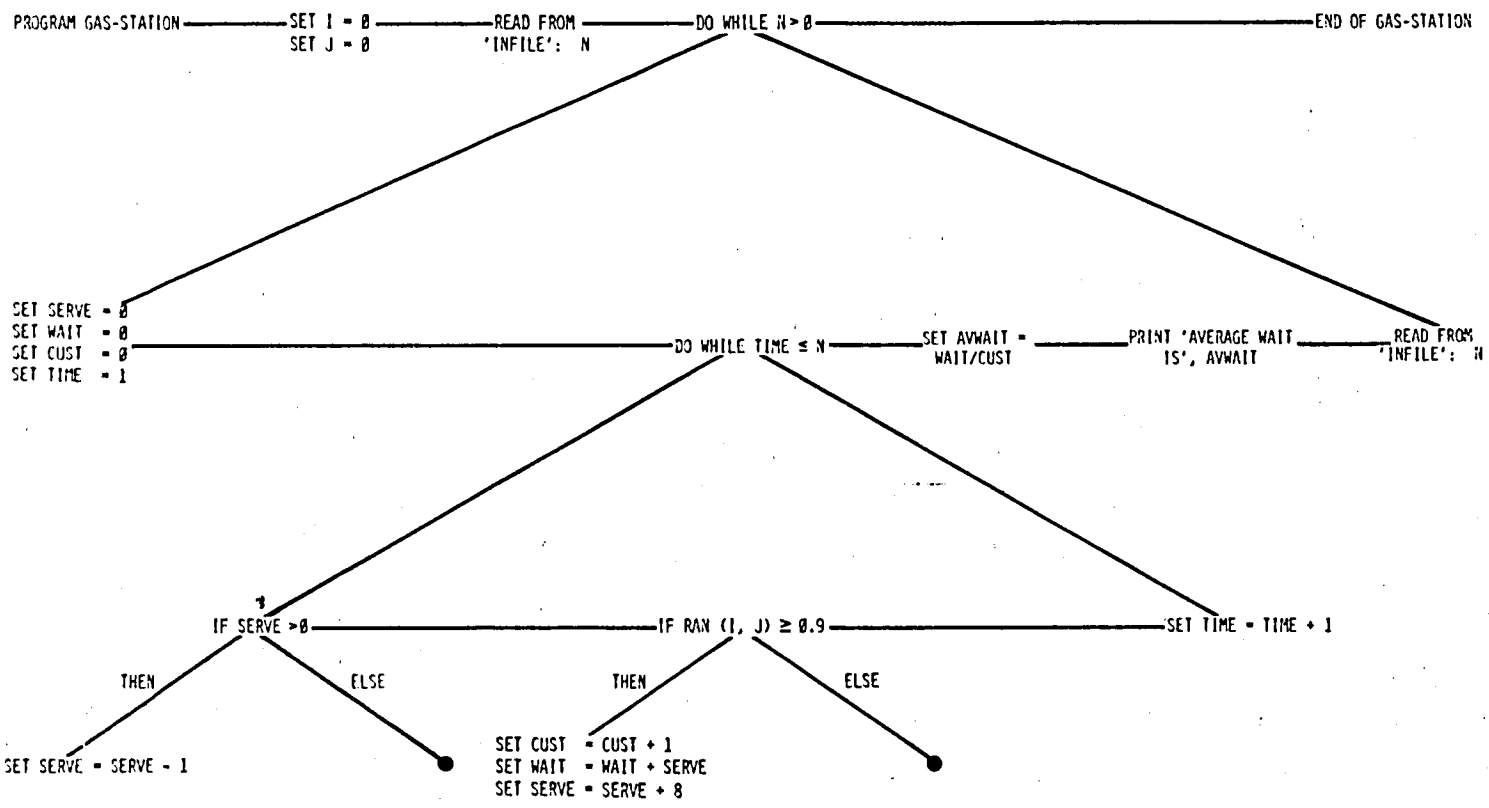
BRANCHING
NATURAL
LANGUAGE

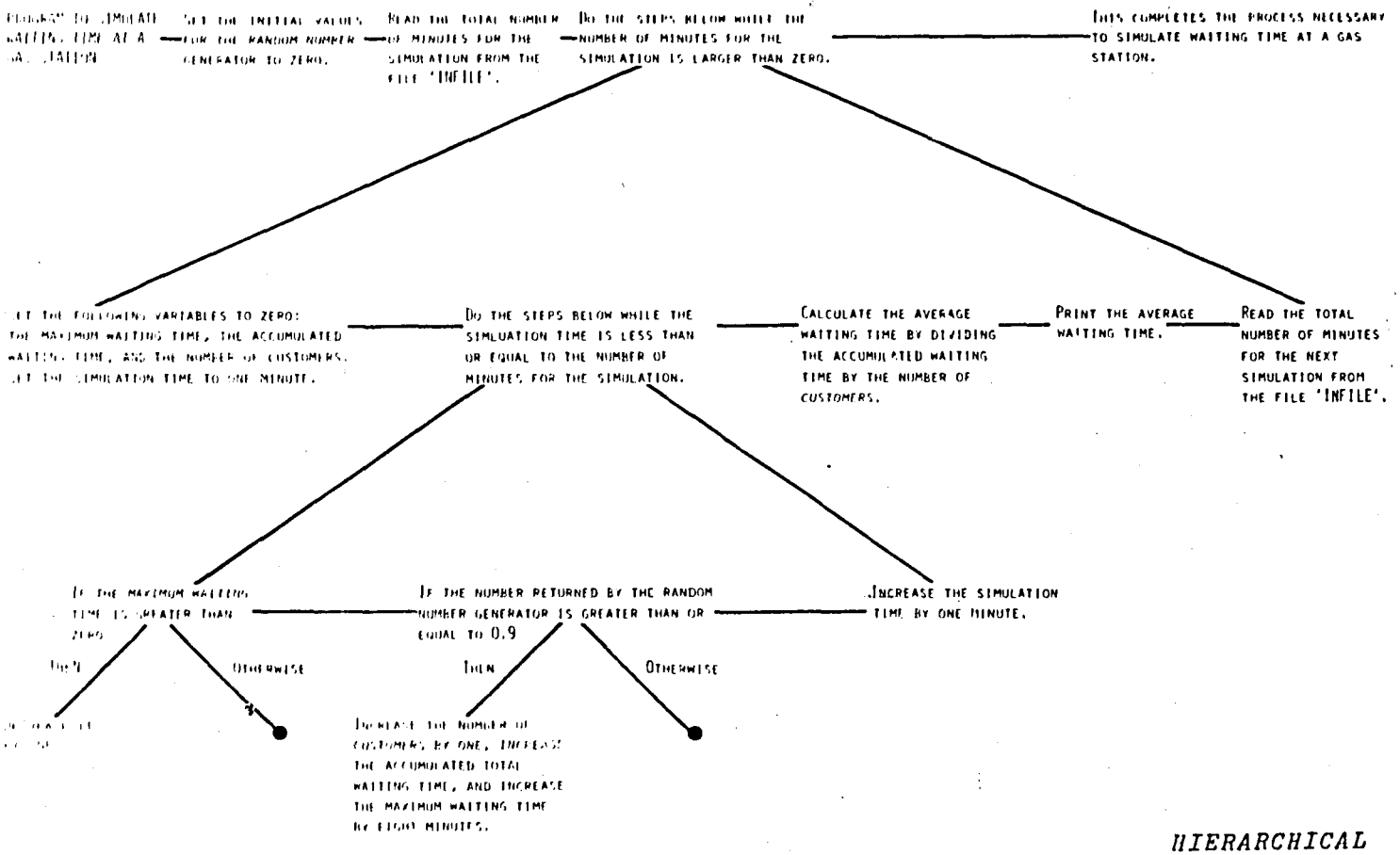




HIERARCHICAL IDEOGRAMS

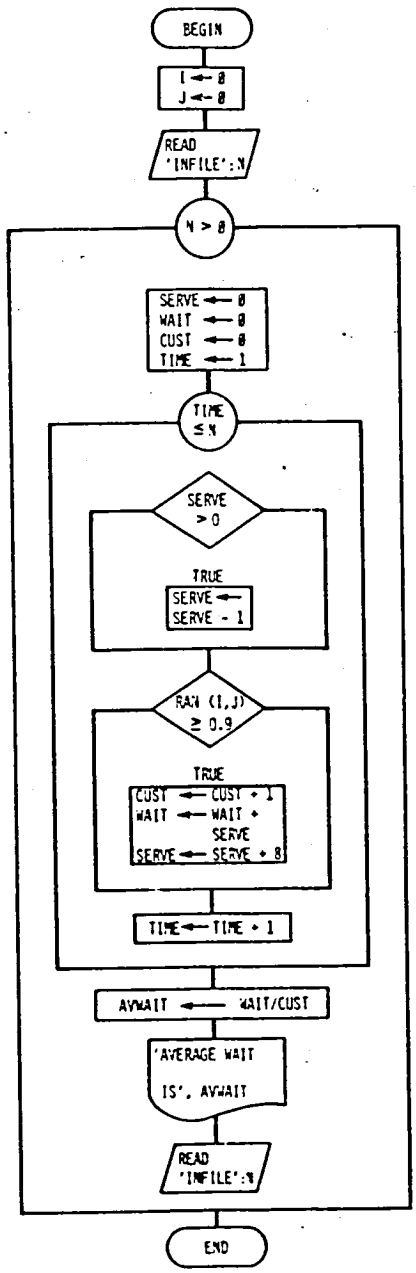
HIERARCHICAL PDL





HIERARCHICAL NATURAL LANGUAGE

ORIGINAL PAGE IS
OF POOR QUALITY


BEGIN
I ← 0
J ← 0
READ 'INFILE':N
N > 0
SERVE ← 0
WAIT ← 0
CUST ← 0
TIME ← 1
TIME ≤ N
SERVE > 0
TRUE
SERVE ← SERVE - 1
RAN (I,J) ≥ 0.9
TRUE
CUST ← CUST + 1
WAIT ← WAIT + SERVE
SERVE ← SERVE + 8
TIME ← TIME + 1
AVWAIT ← WAIT/CUST
'AVERAGE WAIT IS', AVWAIT
READ 'INFILE':N
END

SEQUENTIAL PDL

```
PROGRAM GAS STATION

SET I = 0
SET J = 0

READ FROM 'INFILE': N

DO WHILE N>0

     SET SERVE = 0
     SET WAIT = 0
     SET CUST = 0
     SET TIME = 1

     DO WHILE TIME≤N

          IF SERVE>0

             THEN

             SET SERVE = SERVE - 1

          ENDIF

          IF RAN (I,J)≥0.9

             THEN

             SET CUST = CUST + 1
             SET WAIT = WAIT + SERVE
             SET SERVE = SERVE + 8

          ENDIF

          SET TIME = TIME + 1

     ENDDO

     SET AVWAIT = WAIT/CUST

     PRINT 'AVERAGE WAIT IS', AVWAIT

     READ FROM 'INFILE': N

ENDDO

END OF GAS STATION
```

SEQUENTIAL NATURAL LANGUAGE

PROGRAM TO SIMULATE WAITING TIME AT A GAS STATION

SET THE INITIAL VALUES FOR THE RANDOM NUMBER GENERATOR TO ZERO.
READ THE NUMBER OF MINUTES FOR THE SIMULATION FROM THE FILE 'INFILE'.
DO STEPS 1 THROUGH 5 WHILE THE NUMBER OF MINUTES FOR THE SIMULATION IS LARGER THAN ZERO.

1. SET THE FOLLOWING VARIABLES TO ZERO: THE MAXIMUM WAITING TIME, THE ACCUMULATED WAITING TIME, AND THE NUMBER OF CUSTOMERS.
2. DO STEPS A THROUGH C WHILE THE SIMULATION TIME IS LESS THAN OR EQUAL TO THE NUMBER OF MINUTES FOR THE SIMULATION.
   (A) IF THE MAXIMUM WAITING TIME IS GREATER THAN ZERO, DECREASE IT BY ONE.
   (B) IF THE NUMBER RETURNED BY THE RANDOM NUMBER GENERATOR IS GREATER THAN OR EQUAL TO 0.9, INCREASE THE NUMBER OF CUSTOMERS BY ONE, INCREASE THE ACCUMULATED WAITING TIME BY THE MAXIMUM WAITING TIME, AND INCREASE THE MAXIMUM WAITING TIME BY 8 MINUTES.
   (C) INCREASE THE SIMULATION TIME BY ONE MINUTE.
3. CALCULATE THE AVERAGE WAITING TIME BY DIVIDING THE ACCUMULATED WAITING TIME BY THE NUMBER OF CUSTOMERS.
4. PRINT THE AVERAGE WAITING TIME.
5. READ THE TOTAL NUMBER OF MINUTES FOR THE NEXT SIMULATION FROM THE FILE 'INFILE'.

THIS COMPLETES THE PROCESS NECESSARY TO SIMULATE WAITING TIME AT A GAS STATION.

GENERAL ELECTRIC COMPANY

SPACE DIVISION

COMPREHENSION EXPERIMENT

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

TASK: ANSWER QUESTIONS ON-LINE WHILE STUDYING DOCUMENTATION

PARTICIPANTS: 72 PROFESSIONAL PROGRAMMERS AVERAGING 7 YEARS EXPERIENCE

3 PROGRAMS:
- SCIENTIFIC (ROCKET TRAJECTORY)
- DATA BASE (STORE INVENTORY)
- SCIENTIFIC/DATA BASE (AIRPORT SCHEDULING/SIMULATION)

COMPARISONS:
- ACCURACY OF RESPONSE
- RESPONSE TIME
- PREFERENCES

GENERAL ELECTRIC COMPANY  SPACE DIVISION	COMPREHENSION TEST	INFORMATION SYSTEMS PROGRAMS  SOFTWARE MANAGEMENT RESEARCH

- FORWARD TRACING
- BACKWARD TRACING
- INPUT-OUTPUT

GENERAL ELECTRIC COMPANY SPACE DIVISION	MEAN RESPONSE TIME IN SECONDS	INFORMATION SYSTEMS PROGRAMS SOFTWARE MANAGEMENT RESEARCH

FORWARD TRACING QUESTIONS

	NATURAL LANGUAGE	CONSTRAINED LANGUAGE	IDEOGRAMS	TOTAL
SEQUENTIAL	49.5	32.9	37.8	40.1
BRANCHING	44.8	30.4	36.0	37.1
HIERARCHICAL	43.4	41.9	38.9	41.4
TOTAL	45.9	35.1	37.6	39.5

NOTE. INDIVIDUAL CELL MEANS REPRESENT 120 RESPONSES

GENERAL ELECTRIC COMPANY

SPACE DIVISION

MEAN RESPONSE TIME IN SECONDS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

BACKWARD TRACING QUESTIONS

	NATURAL LANGUAGE	CONSTRAINED LANGUAGE	IDEOGRAMS	TOTAL
SEQUENTIAL	51.5	35.0	42.6	43.0
BRANCHING	45.3	35.6	34.6	38.5
HIERARCHICAL	43.6	36.8	35.7	38.7
TOTAL	46.8	35.8	37.6	40.1

NOTE: INDIVIDUAL CELL MEANS REPRESENT 120 RESPONSES

GENERAL ELECTRIC COMPANY

SPACE DIVISION

MEAN RESPONSE TIME IN SECONDS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

INPUT-OUTPUT QUESTIONS

	NATURAL LANGUAGE	CONSTRAINED LANGUAGE	IDEOGRAMS	TOTAL
SEQUENTIAL	43.4	41.9	38.9	41.4
BRANCHING	43.3	36.2	44.3	41.4
HIERARCHICAL	41.9	44.8	34.9	40.5
TOTAL	42.9	41.0	39.4	41.1

NOTE: INDIVIDUAL CELL MEANS REPRESENT 192 RESPONSES

PERCENT OF PREFERENCES

TYPE OF SYMBOLOGY:

CONSTRAINED LANGUAGE (PDL)	53%
IDEOGRAMS	33%
NATURAL LANGUAGE	14%
	100%

SPATIAL ARRANGEMENT:

BRANCHING	48%
SEQUENTIAL	36%
HIERARCHICAL	16%
	100%

GENERAL ELECTRIC COMPANY

SPACE DIVISION

RELATIONSHIP OF EXPERIENCE TO PERFORMANCE (AVERAGE RESPONSE TIME)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

YEARS PROGRAMMING	+.23*
NUMBER OF LANGUAGES	-.49**

*p<.05
**p<.01

GENERAL ELECTRIC COMPANY  SPACE DIVISION	CONCLUSIONS	INFORMATION SYSTEMS PROGRAMS  SOFTWARE MANAGEMENT RESEARCH

- FOR A COMPREHENSION TASK, NATURAL LANGUAGE IS NOT AS GOOD AS CONSTRAINED LANGUAGE OR IDEOGRAMS

- THE BRANCHING ARRANGEMENT WITH SUCCINCT SYMBOLS (CONSTRAINED LANGUAGE OR IDEOGRAMS) IS BEST FOR COMPREHENSION.

MEASURING THE DEVELOPMENT PROCESS:
Software Design Coupling and Strength Matrices

R. D. Cruickshank and J. E. Gaffney, Jr.
IBM Corporation, Federal Systems Division, Manassas, VA

OVERVIEW

Meyers[1] has defined two concepts of considerable importance to representing the degree of 'goodness' of a software design, 'coupling' and 'strength'. 'Coupling' indicates the degree of interconnectedness of modules. 'Strength' indicates the degree of cohesiveness of module function. Both of these measures relate to the ease of modification of a module. If modules have relatively low strength, then the function they effect will be relatively diffused, and changes in a function will be propagated among a plurality of modules at decreased efficiency and increased cost.

Cruickshank and Gaffney[2] have developed a set of metrics relating to the 'goodness' of a software design and its degree of completion. A (software) metric is a mathematical measure of a body of software that is sensitive to differences in software characteristics. Two of these metrics, for 'coupling' and 'strength', are described in detail here. The effect of these metrics is to transform these qualitative concepts into quantitative measures. These metrics are a tool for evaluating a design and comparing it with alternatives. These measures have been applied to a design written in a design language, Process Design Language (PDL).

STRENGTH

Meyers has defined two types of high strength modules. The first, 'functional strength', refers to a module in which all of the elements relate to the performance of a single function. The other, 'informational strength', relates to a module that performs multiple functions with a single data structure. The strength metric recognizes that 'strength' is a two-dimensional concept, in which the relative numbers of (unique) inputs and outputs are represented and the depth of processing (number of actions indicated) is included. This is necessary in order to cover the range of cases in which modules exhibit high functional or informational strength. Let Y = no. of module outputs/no. of module inputs and X = 1/total number of assignment, 'RUN' and 'CALL' PDL statements. Then S = X + jY, a complex number. For comparative purposes, S the strength = $\sqrt{X^2 + Y^2}$. As an illustrative example, a module exhibiting higher informational strength might take from types of data A, B, C, and D and do many things to them. As an example, take the following code sequence which would also appear the same in the design language, PDL:

$$0_1: = A + B + C + D$$

$$0_2: = A^2 + B^2 + C^2 + D^2$$

$$0_3: = (A/B)^* [C + 3D]$$

In this case, X = 1/3; Y = 3/4 = 0.75; and S = 0.819. A module of lower informational strength could be one that performs several unrelated functions:

$$0_1: = A + B$$

$$0_2: = C + D$$

Here, X = 1/2; Y = 2/4 = 0.5; and S = 0.707.

Note the relationship between the strength measures and the transfer function of an electronic circuit, which has a gain (rate of output/input magnitudes) and a phase (measure of delay through a circuit).

COUPLING

Coupling is a measure of the relationships among modules, whereas strength is a measure of relationships among the elements within a module. Coupling measures the degree to which two modules are interconnected. The less the coupling, the better the design, because intellectual control of design is facilitated and the propagation of changes is minimized for a given module. Let Z equal the average number of input and output items shared by this module and another one. This figure can be normalized, if desired, to vary between 0 (for high coupling) to 1 (for low coupling). If k = 1 – exp (–0.66943/Z); then if Z = 1, k = 0.988; and if Z = 0.25, k = 0.93.

OBSERVATIONS

Software metrics create a new dimension for the evaluation of software. They can greatly improve the manageability of the software development process and enhance the growth of the resultant product. The set of metrics should include ones for coupling and strength to replace the qualitative measures for these items used heretofore. The data for the mathematical measures of coupling and strength given here can be derived automatically if a system, such as PSL/PSA, developed by the University of Michigan, is employed to support the design process. Very importantly, these metrics provide a method of comparing designs and suggesting improvements to software designs.

BIBLIOGRAPHY

1. Meyers, G. J., "Reliable Software through Composite Design," Petrocelli/Charter, 1975.

2. Cruickshank, R. D., and Gaffney, J. E., Jr., "Quantitative Software Design Issues," 1980 ACM Computer Science Conference.

3. Linger, R. C., Mills, H. D., and Witt, B. L., "Structured Programming Theory and Practice," Addison-Wesley, 1979.

4. Skinners, S. M., "Modern Control Systems Theory and Application," Addison-Wesley, 1978.

THE VIEWGRAPH MATERIALS
for the
R. CRUICKSHANK PRESENTATION FOLLOW

OBJECTIVE OF OUR WORK IS TO QUANTIFY SOME MEASURES OF GOODNESS/BADNESS OF DESIGN

TWO IMPORTANT CONCEPTS* REPRESENTING THE DEGREE OF GOODNESS OF A SOFTWARE DESIGN ARE:

STRENGTH – INDICATES COHESIVENESS OF MODULE FUNCTION

COUPLING – INDICATES DEGREE OF INTERCONNECTEDNESS OF MODULES

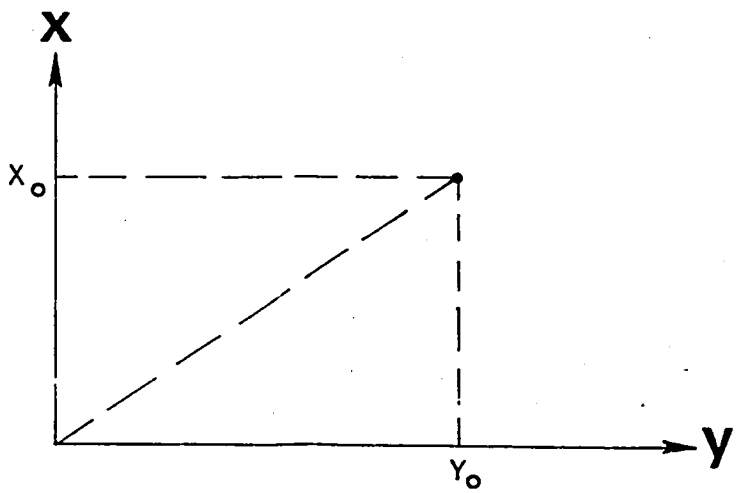
*SUGGESTED BY MYERS IN "RELIABLE SOFTWARE THROUGH COMPOSITE DESIGN" (PETROCELLI/CHARTER, 1975)

WE PROPOSE QUANTIFYING THESE CONCEPTS SO THAT ALTERNATIVE DESIGNS CAN BE COMPARED

STRENGTH METRIC

TWO DIMENSIONAL, REPRESENTS:

- RELATIVE NUMBER OF INPUTS AND OUTPUTS
- DEPTH OF PROCESSING



Y = NO. OF UNIQUE MODULE OUTPUTS
NO. OF UNIQUE MODULE INPUTS

(Y ANALOGOUS TO "GAIN" IN A CIRCUIT)

X = 1/NO. OF ASSIGNMENT STATEMENTS

(1/X ANALOGOUS TO "PHASESHIFT" IN A CIRCUIT)

$$S = X_o + JY_o$$

S = STRENGTH = $\sqrt{X^2 + Y^2}$

TWO TYPES OF HIGH STRENGTH MODULES –

FUNCTIONAL – ALL OF THE ELEMENTS RELATE TO THE PERFORMANCE OF A SINGLE FUNCTION

INFORMATIONAL – MULTIPLE FUNCTIONS PERFORMED WITH A SINGLE DATA STRUCTURE

EXAMPLE OF LOWER FUNCTIONAL STRENGTH

$R: = \sqrt{A + B}$

$Z: = C + D$

$0: = R + Z$

$X = 1/3; Y = 1/4$

$S = \sqrt{(1/3)^2 + (1/4)^2} = 0.414$

EXAMPLE OF HIGHER FUNCTIONAL STRENGTH

$0: = A + B + C + D$

$X = 1/1; Y = 1/4$

$S = \sqrt{(1/1)^2 + (1/4)^2} = 1.0308$

EXAMPLE OF HIGHER INFORMATIONAL STRENGTH

0_1: = A + B + C + D

0_2: = $A^2 + B^2 + C^2 + D^2$

0_3: = (A/B) X C + 3D

X = 1/3; Y = 3/4

S = $\sqrt{(1/3)^2 + (3/4)^2}$ = 0.81939

EXAMPLE OF LOWER INFORMATIONAL STRENGTH

0_1: = A + B

0_2: = C + D

X = 1/2; Y = 2/4

S = $\sqrt{(1/2)^2 + (2/4)^2}$ = 0.707

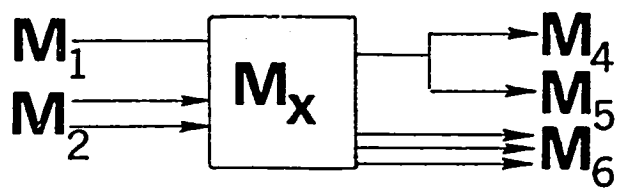
SOME STRENGTH DATA

PROCEDURE	UNIQUE INPUTS (I)	UNIQUE OUTPUTS (O)	Y=O/I	NO. OF ASSIGNS (1/X)	X	$S=\sqrt{X^2 + Y^2}$
1	5	4	.8000	11	.0083	.8051
2	6	5	.8333	9	.1111	.8407
3	7	5	.7143	10	.1000	.7213
4	5	4	.8000	5	.2000	.8246
5	4	5	1.2500	7	.1429	1.2581

COUPLING METRIC

Z = REPRESENTS AVERAGE OF INPUT AND OUTPUT ITEMS SHARED BY THIS MODULE AND ANOTHER ONE.

EXAMPLE: FOR MODULE MX



NUMBER OF ITEMS:

1 FROM M_1
2 FROM M_2
1 TO M_4
1 TO M_5
3 TO M_6

$$Z = \frac{1+2+1+1+3}{5} = \frac{8}{5} = 1.6$$

CAN COMPUTE AN AVERAGE Z OVER A SET OF MODULES

EXAMPLE: $Z = \frac{1.6 + 2.0 + 3.0}{3} = 2.2$ FOR A 3 MODULE PROGRAM

NUMERICAL STRENGTH AND COUPLING METRICS PROVIDE A MEANS FOR QUANTITIVELY COMPARING ALTERNATIVE DESIGNS

R. Cruickshank
IBM

N82 24007

MEASURING THE DEVELOPMENT PROCESS: A Tool for Software Design Evaluation

Susan S. Moy
Logicon, Inc.

ABSTRACT

This paper described the Design Metrics Evaluator (DME), a component of an automated software design analysis system developed by Logicon. The DME quantitatively evaluates software design attributes. Its use directs the analyst's attention to areas of a procedure, module, or complete program having a high potential for error.

INTRODUCTION

As the usage of computers has increased and their areas of application have broadened, software reliability has become an issue of major concern. Considerable effort has been devoted to assessing software reliability, and studies of software quality characteristics and software evaluation technologies have produced reliability models and complexity metrics based on various approaches and parameters. Nevertheless, there are still severe limitations on our ability to evaluate the quality of computer software quantitatively and automatically.

At Logicon, efforts to enhance software evaluation methodology have produced a system of software design analysis tools called LOGICFLOW. LOGICFLOW has the following components:

- A simple design language sufficiently versatile to relate high- and low-level, structured and unstructured designs
- A design language processor to detect logical errors
- A translator to translate the design language into FORTRAN or JOVIAL
- A flowcharter to generate flowcharts from design language, FORTRAN, and several assembly languages
- A cross-reference generator to create an index of all design tokens (or symbols)
- A design metrics evaluator to generate metrics that reflect the characteristics of a software design

The Design Metrics Evaluator provides a quantitative measure of design quality. It facilitates enforcement of software development standards, encourages better design by providing feedback on design quality during the design phase, and directs attention to areas where there is a high potential for errors. This early detection of errors in the design phase can reduce software production cost, improve software reliability, and contribute to timely completion of the software.

METHODOLOGY

The quality of a software design can be evaluated by comparing specific design characteristics with predetermined evaluation criteria. The approach used to develop the Design Metrics Evaluator was

to identify desirable software design characteristics, determine how to measure them, and establish a set of evaluation criteria.

The first step in developing the Design Metrics Evaluator was to identify a set of nonoverlapping, desirable attributes of software design. The pertinent literature was surveyed and all relevant attributes were extracted and recorded. Table 1 presents the list of desirable attributes compiled in this way. All members of this initial set were examined, synonymous or redundant attributes were grouped together, and the most meaningful or inclusive term in each group was chosen as the representative "name" for that group. The resulting attributes were then organized hierarchically, with attributes identified as goals at the top level and contributing factors at the lower levels.

The attributes that emerged as goals were usability, maintainability, and generality. Usable software is easy to work with and produces reliable results.

Maintainable software is understandable, modifiable, and verifiable. General software adapts readily to new conditions and different configurations. Reliability and human factors are supporting factors for the goal of usability; understandability, verifiability, and modifiability support maintainability; and reusability and transferability support generality. These supporting factors can be further broken into components as shown in Figure 1.

Once this basic hierarchy had been formulated, the definitions of the design attributes were examined and a number of design criteria were identified for each. For example, some of the criteria associated with structuredness are: Is the design modularized in accordance with the major system functions? Are the modules organized heirarchically? Are all modules separate and distinct?

After the desirable software design attributes had been identified, design language constructs were studied and measurable features such as program length, statement size, etc., were identified. A prototype Design Metrics Evaluator was developed to collect statistics automatically on the design language constructs. These statistics were formulated into an initial set of metrics. Analysis of these metrics and their relationship to the design attributes revealed that not all design characteristics identified earlier are quantitatively measurable. The metrics collected reflect the fact that structural characteristics (such as the number of unconditional branches in a procedure) are the most directly measurable. Difficult to quantify are the design characteristics related to the functional aspects of a design (for example, the accuracy of the algorithms used).

The initial set of design metrics was found to have the following limitations:

- Knowledge of the evaluation criteria is required to interpret the metrics
- The metrics are not on the same absolute scale and therefore are difficult to compare or combine

To overcome these limitations, normalization factors were developed. The design evaluation criteria established earlier were used to formulate a normalization equation for each of the measurable design characteristics. (Some of the normalization equations and their rationales are presented in Table 2.) An absolute measure ranging from 0 to 1, with 1 meaning most desirable, was developed for each factor measured by the Design Metrics Evaluator. This allowed normalized metrics to be combined to provide absolute measures for design characteristics at various levels of the hierarchy. The calculation of the simplicity metric by combining its components is also illustrated in Table 2.

The importance of different design characteristics can vary for different applications. The overall quality of a program's design can be obtained by combining characteristics of interest to that program, weighted by their importance. The assignment of weights to the various design characteristics can be subjective and can vary according to application.

THE DESIGN METRICS EVALUATOR

The Design Metrics Evaluator is an automated software tool consisting of three modules that interface through a global data base. The Metrics Module collects the design metrics during parsing and stores the results in the global data base. The Normet Module normalizes the metrics. The Display Module displays the resultant metrics, in normalized or unnormalized form.

The Design Metrics Evaluator has the capability of collecting design metrics for a procedure, a module, or a complete program. It is a user-oriented tool with six user-selectable options:

- Collection of design metrics
- Listing of the prenormalized design metrics
- Normalization of the design metrics
- Output of normalized design metrics to a file
- Output of normalized design metrics to the printer
- Collection of program statistics

Design metrics for a module or a program can be calculated with user-specified weights assigned to the design characteristics. This capability allows weights to be tailored or refined for a specific software application.

Figure 2 presents the normalized design metrics for one module of a software tool. Shown at the top are the metrics for three groups of attributes (simplicity, structuredness, and readability) for the procedures identified in the left-most column. The parenthetical numbers are the standard deviations for the module. These results indicate that, in general, the procedures of this module are of reasonable length (program length metric = 0.766). The low statement size metric of 0.289 indicates that the statements are too long. This is in general true for all the procedures, as indicated by the standard deviation of 0.186 and the statement size metric for each procedure. The flow-interruption metric of 1.0 results because there are no unconditional branches in any of the procedures.

Absolute measures on selected design characteristics for the module are shown at the bottom of Figure 2. These metrics were obtained by combining the values of their supporting factors, weighted by importance. For example, the simplicity metrics were calculated with equal weightings and the sufficiency metrics were calculated with simplicity weighted twice as much as structuredness:

$$\text{Simplicity} = 1/4\ (\text{program length} + \text{statement size} + \text{decision count} + \text{statement nesting level})$$
$$= 1/4\ (0.766 + 0.289 + 0.614 + 0.896) = 0.641$$
$$\text{Sufficiency} = 1/3\ (2\ \text{simplicity} + \text{structuredness})$$
$$= ((2)(0.641) + 0.814) = 0.699$$

The summary metrics suggest that the module is fairly well designed. The lowest value, 0.641, for the simplicity metrics reflects the large statement size and points out that this area needs to be improved. Furthermore, the maintainability metrics are higher than the usability metrics, pointing out that the design is weak in the usability area.

FUTURE WORK

The current Design Metrics Evaluator supplies a program designer with objective metrics and statistics based on design language constructs. Although many design characteristics are not quantitatively measurable, the DME output metrics provide a guideline for determining design quality.

Further work is being considered in the following areas:

- Development of systematic approach to subjective software design evaluation
- Incorporation of design metrics for data structures
- Incorporation of selected complexity metrics found in the literature

Further validation of the Design Metrics Evaluator is another area for future work. The tool has been tested on a number of different software application programs but should be tested more extensively before release for general use.

Table 1
Candidate Software Characteristics

Acceptability
Accessibility
Accountability
Accuracy
Adaptability
Augmentability
Brevity
Changeability
Clarity
Communicativeness
Compatibility
Completeness
Comprehensibility
Conciseness
Confirmability
Consistency
Convenience
Correctness
Expandability
Expressiveness
Extensibility
Flexibility
Generality
Hierarchicality
Human Factors
Improvability
Integrity
Intelligibility
Interoperability
Legibility
Linearity
Machine-Independence
Maintainability
Meaningfulness
Modifiability
Modularity
Operability
Orderedness
Portability
Precision
Readability
Reliability
Repairability
Reusability
Robustness
Ruggedness
Security
Self-Containedness
Self-Descriptiveness
Serviceability
Simplicity
Stability
Structuredness
Succinctness
Sufficiency
Terseness
Testability
Tolerance
Transferability
Understandability
Uniformity
Usability
User-Centeredness
Validity
Verifiability

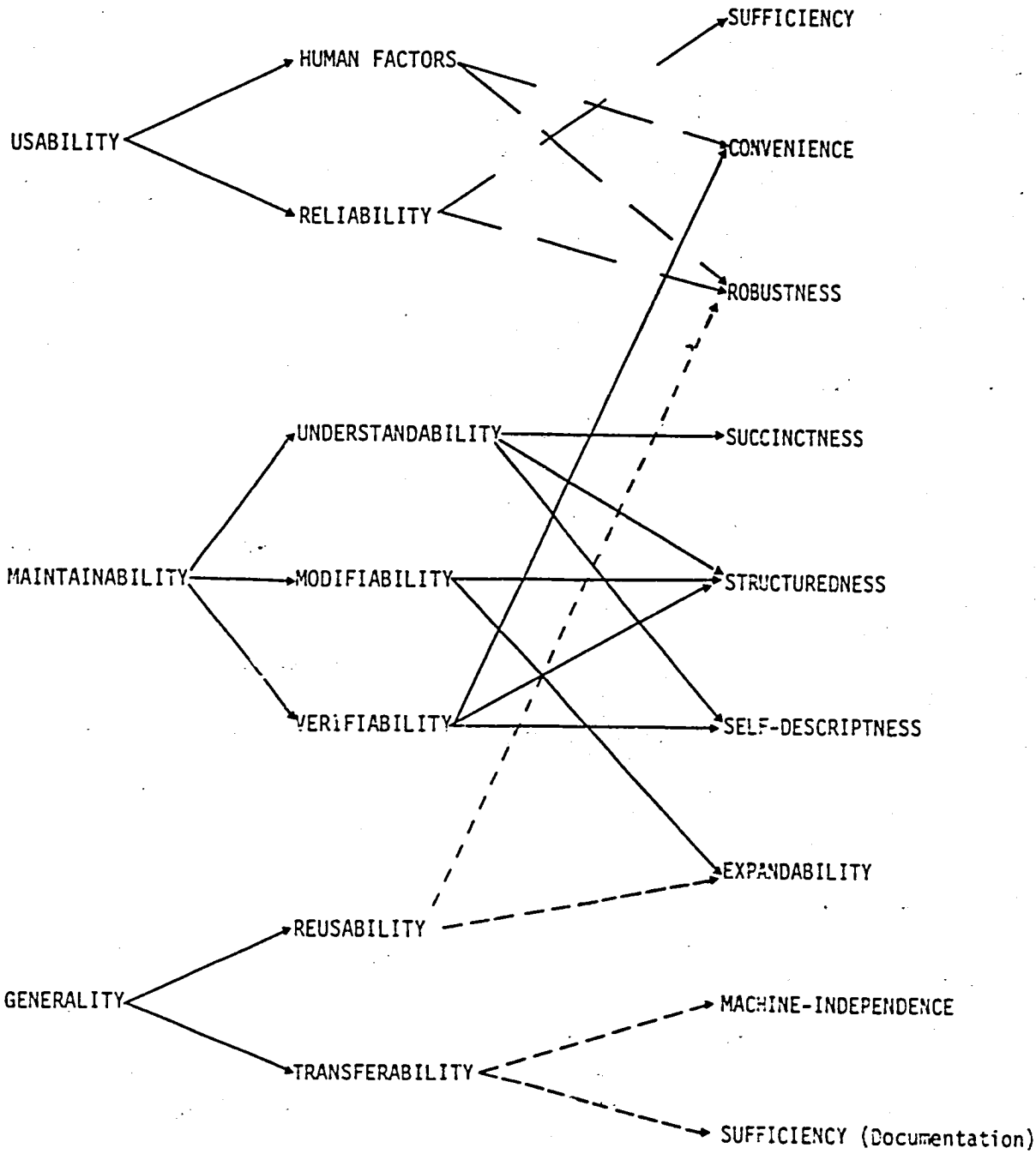


Figure 1 Hierarchical structure of good software design characteristics.

Table 2. Normalization Equations for the Components of Simplicity

Component — **Absolute Measure** — **Comments**

Procedure length

$$S(n) = \begin{cases} 1 - \frac{n}{150} & n \le 50 \\ 1 - \min\left(1, \frac{1}{3} + \frac{n-50}{n}\right) & n > 50 \end{cases}$$

where

n = number of statements

A procedure with a large number of statements is more complex than one with fewer statements. The threshold value of 35 to 100 statements has been used in various studies. Here, 50 is chosen for the reason that it is also the criterion for fitting the whole procedure onto one listing page. The piecewise continuous function was defined such that the complexity of a function with no more than 50 statements increases gradually. The slope of the function turns steeper as the number of statements goes beyond 50, and finally levels off at a complexity of 1 as the number of statements increases. Note that the complexity of 1/3 for n=50 is subjective and can be considered just as a point of reference.

Statement sizes

$$S(t) = \begin{cases} 1 - \frac{\bar{t}}{15} & \max(t) \le 5 \\ 1 - \min\left(1, \frac{1}{3} + \frac{n_1}{2n}\left(\max(0, \bar{t}-5)\right)\right) & \max(t) > 5 \end{cases}$$

where

- t = number of tokens per statement
- $\bar{t}$ = average number of tokens per statement
- n = number of statements
- n_1 = number of statements with $t > 5$

Longer statements are, in general, more complex than the shorter ones. A simple executable statement has no more than 5 tokens is the basis for this function.

Number of decisions

$$S(d) = 1 - \min\left(1, \frac{2d}{n}\right)$$

where

- n = number of statements
- d = total number of decisions = $d_1+d_2+d_3$
- d_1 = number of decisions based on the IF construct = [1/2(no. of IFs)(avg size of IFs)] + [1/2(no. of IFs)(avg no. of ELSEIFs/IF)(avg size of ELSEIFs)]*
- d_2 = number of decisions based on the LOOP construct = [1/2(no. of LOOPs)(avg no. of WHILEs/LOOP)(avg size of WHILE)]
- d_3 = number of decisions based on the CASE construct = [1/2(no. of CASEs)(avg no. of CASEIFs/CASE)(avg size of CASEIF)]

A condition statement is the most complex type of statement. A procedure that has to make more decisions is, in general, more complex than one that has to make fewer decisions. The size of the condition statement contributes to the number of decisions, e.g.,

IF ((A.EQ.B) or (A.EQ.C))

is one condition statement with three decisions. The total number of decisions divided by the number of statements provides a measure of complexity. The function is defined as such since procedures with over 50% of condition statements are too complex.

Nesting level

$$S(l) = \begin{cases} 1 - \frac{\bar{l}}{9} & \max(l) \le 3 \\ 1 - \min\left(1, \frac{1}{3} + \frac{n_2}{n}\left(\max(0, \bar{l}-3)\right)\right) & \max(l) > 3 \end{cases}$$

where

- $\bar{l}$ = average nesting level
- l = nesting level
- n = number of statements
- n_2 = number of statements with $l > 3$

A deeply nested procedure is more complex than a less nested one. The threshold value of 3 is the criterion used here.

Absolute simplicity measure $S = \frac{1}{4}(S(n) + S(t) + S(d) + S(l))$

*[a] means the integral part of the number a, e.g., [1.8] = 1, 5 [5] = 5

S. Moy
Logicon

NORMALIZED DESIGN METRICS*

SIMPLICITY METRICS

	PROGRAM LENGTH	STATEMENT SIZE	DECISION COUNT	STMT NESTING LEVEL
BAND	0.967	0.0	1.000	1.000
DELETE_I	0.867	0.502	0.800	0.979
HASH	0.900	0.380	1.000	1.000
HASH_DEC	0.747	0.458	0.789	0.919
HASHED_RE	0.800	0.247	0.600	0.937
SEARCH	0.629	0.191	0.615	0.958
SEARCH2	0.693	0.256	0.389	0.979
SEARCH_FC	0.800	0.0	0.333	0.904
SEARCH_SC	0.747	0.577	0.526	0.971
SEARCH_S	0.593	0.218	0.407	0.963
TEST_IDE	0.680	0.374	0.333	0.667
MODULE	0.766(0.115)	0.289(0.186)	0.614(0.253)	0.896(0.094)

STRUCTUREDNESS METRICS

	FLOW INTERRUPTION	MULTI EXITS
BAND	1.000	1.000
DELETE_I	1.000	0.0
HASH	1.000	1.000
HASH_DEC	1.000	1.000
HASHED_RE	1.000	0.333
SEARCH	1.000	1.000
SEARCH2	1.000	0.565
SEARCH_FC	1.000	0.0
SEARCH_SC	1.000	1.000
SEARCH_S	1.000	0.0
TEST_IDE	1.000	1.000
MODULE	1.000(0.0)	0.627(0.459)

READABILITY METRICS

	PROGRAM LENGTH	COMMENT STATEMENTS	REDUNDANT INFORMATION
BAND	0.967	1.000	1.000
DELETE_I	0.867	1.000	1.000
HASH	0.900	1.000	1.000
HASH_DEC	0.747	1.000	1.000
HASHED_RE	0.800	0.700	0.909
SEARCH	0.629	1.000	1.000
SEARCH2	0.693	0.978	1.000
SEARCH_FC	0.800	1.000	1.000
SEARCH_SC	0.747	1.000	1.000
SEARCH_S	0.593	1.000	1.000
TEST_IDE	0.680	1.000	1.000
MODULE	0.766(0.115)	0.971(0.090)	0.992(0.027)

ABSOLUTE MEASURES ON SELECTED DESIGN CHARACTERISTICS FOR THE MODULE

SIMPLICITY:	0.631
STRUCTUREDNESS:	0.813
READABILITY:	0.959
SUFFICIENCY:	0.699
RELIABILITY:	0.639
USABILITY:	0.639
SUCCINCTNESS:	0.992
UNDERSTANDABILITY:	0.915
VERIFYABILITY:	0.892
MODIFIABILITY:	0.819
MAINTAINABILITY:	0.873

*NUMBERS IN PARENTHESIS ARE VALUES OF THE STANDARD DEVIATIONS

Figure 2. Listing of Normalized Metrics for a Sample Module and Its Procedures

BIBLIOGRAPHY

1. Boehm, B. W., et al., Characteristics of Software Quality, TRW Series of Software Technology, Vol. 1, 1978.

2. Design Metrics Study: Final Report, Logicon Report SED-R79009, October 1979.

3. Design Metrics Study: Final Report, Logicon Report R:SED-80390, September 1980.

4. Gild, T., Software Metrics, Cambridge, MA: Winthrop, 1977.

5. Halstead, Elements of Software Science, New York: Elsevier North-Holland, 1977.

6. McCall, J. A., P. K. Richards, and G. F. Walters, Factors in Software Quality, Rome Air Development Center Report No. RADC-TR-77-369, November 1977.

THE VIEWGRAPH MATERIALS
for the
S. MOY PRESENTATION FOLLOW

A TOOL FOR SOFTWARE DESIGN EVALUATION

SUSAN S. MOY
LOGICON, INC.

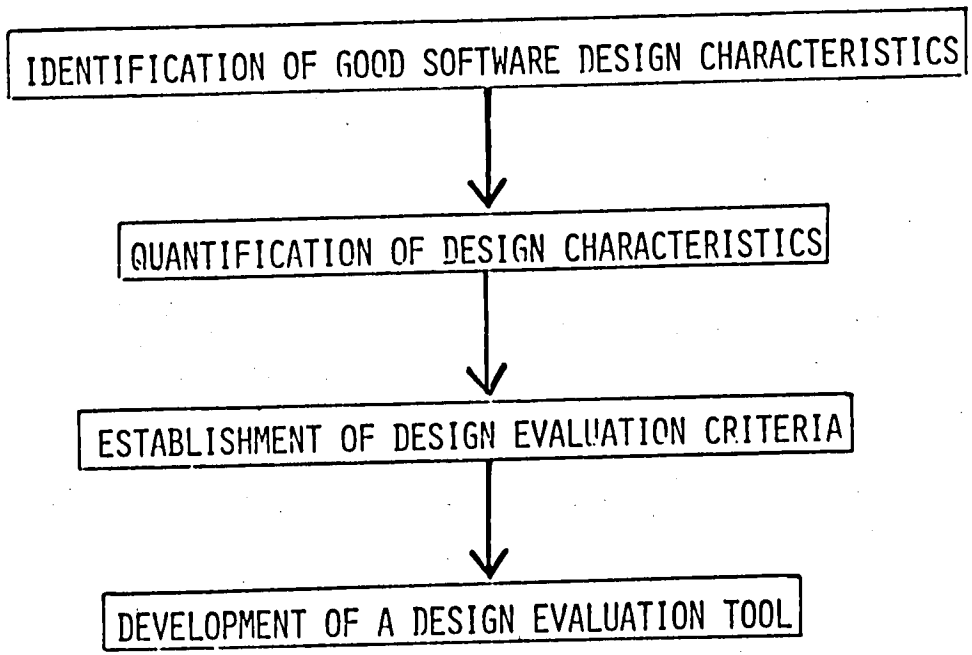
OBJECTIVE

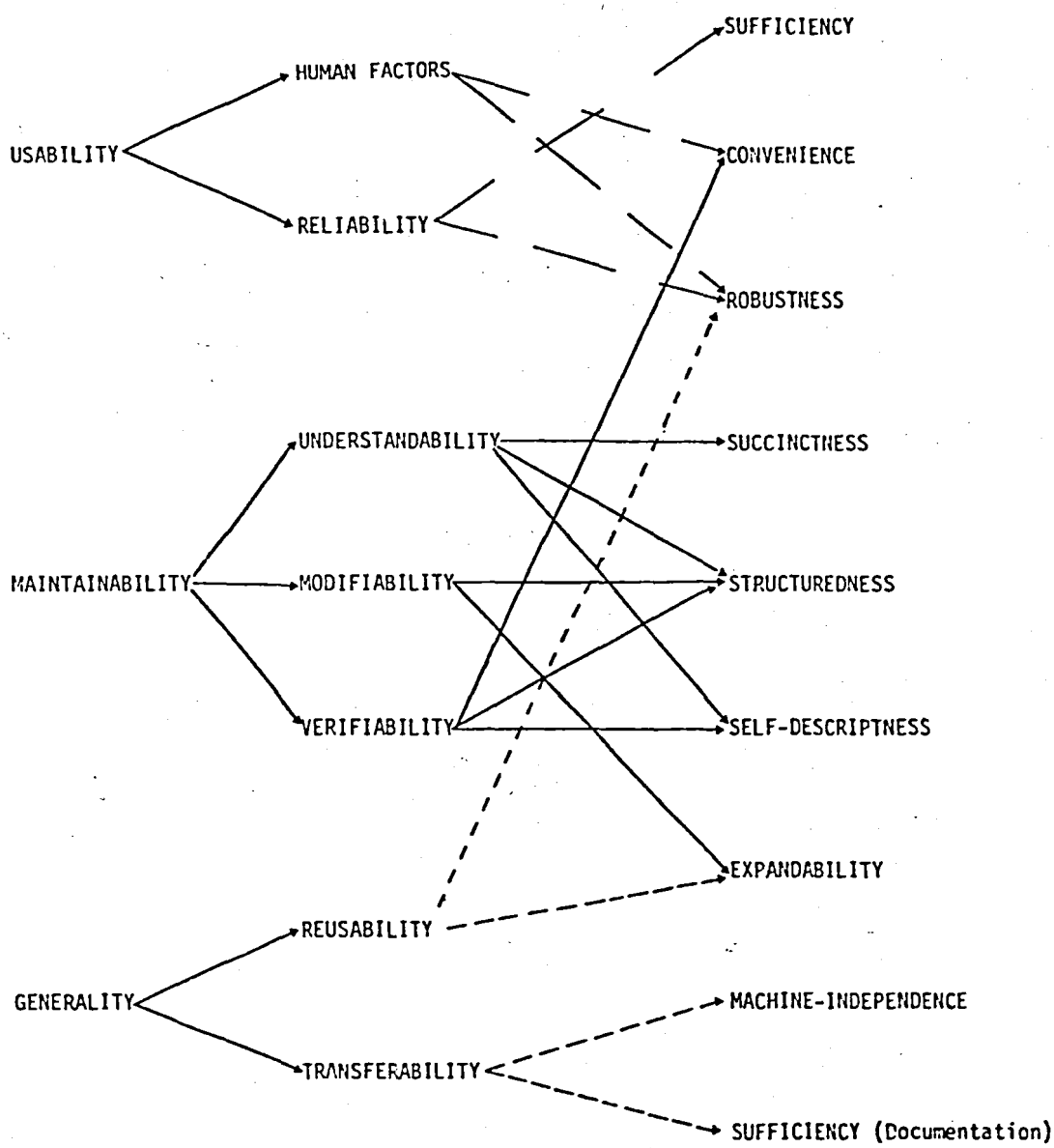
- SOFTWARE DEVELOPMENT AID
  - DESIGN EVALUATION
  - QUALITY MONITORING
- VERIFICATION AND VALIDATION TOOL
  - DESIGN ANALYSIS
  - INDICATOR OF PROBLEM AREAS

RESULTS IN

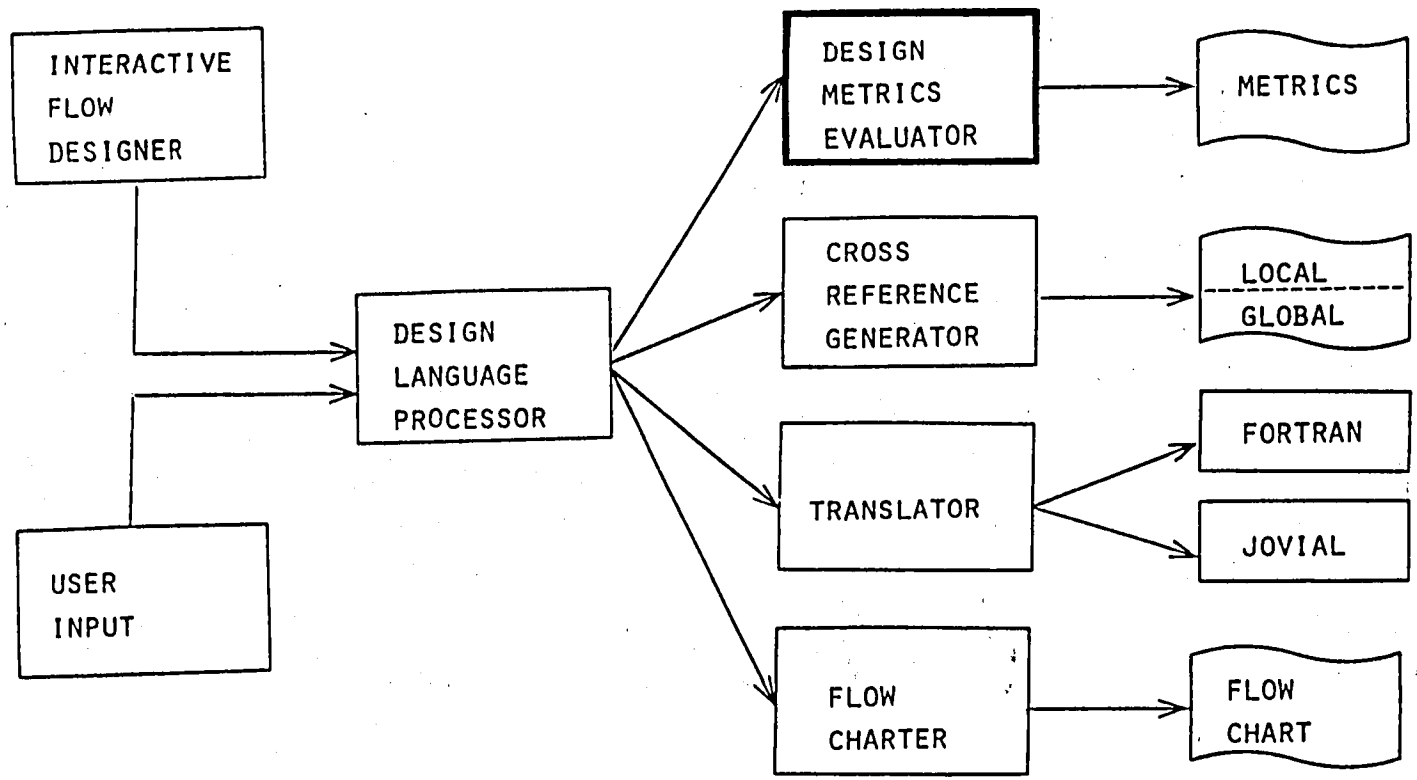
- LOWER COSTS
- MORE RELIABLE SOFTWARE
- BETTER VISIBILITY

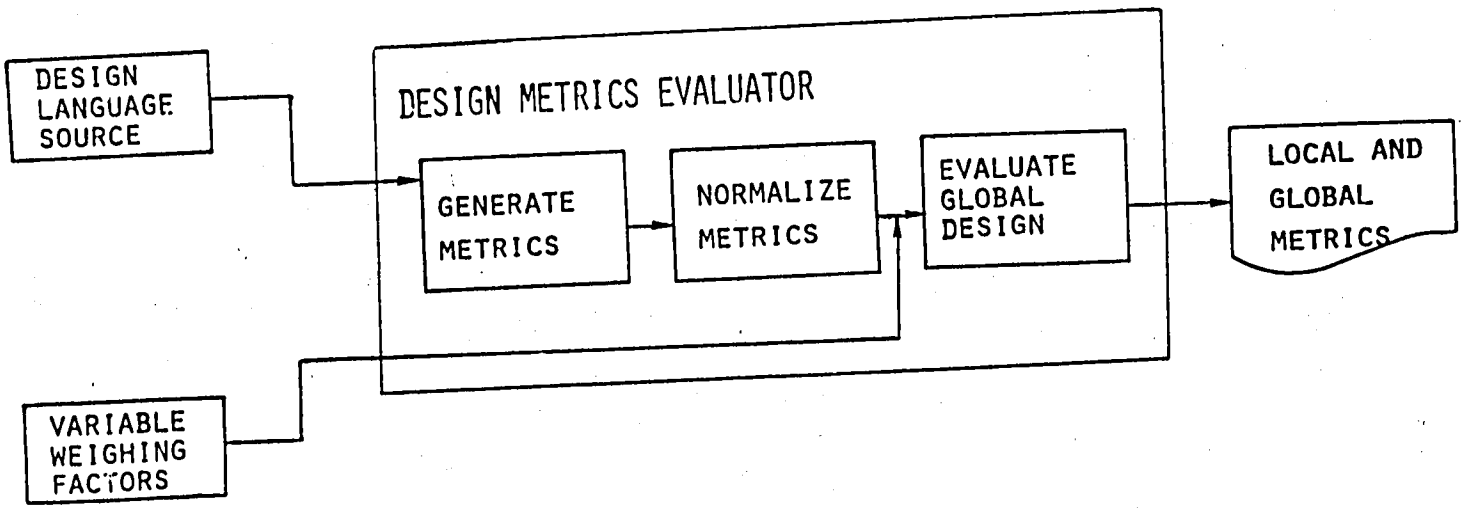
METHODOLOGY




SUFFICIENCY
HUMAN FACTORS
USABILITY
CONVENIENCE
RELIABILITY
ROBUSTNESS
UNDERSTANDABILITY
SUCCINCTNESS
MAINTAINABILITY
MODIFIABILITY
STRUCTUREDNESS
VERIFIABILITY
SELF-DESCRIPTNESS
EXPANDABILITY
REUSABILITY
GENERALITY
MACHINE-INDEPENDENCE
TRANSFERABILITY
SUFFICIENCY (Documentation)

LOGICFLOW SYSTEM




DESIGN LANGUAGE SOURCE
DESIGN METRICS EVALUATOR
GENERATE METRICS
NORMALIZE METRICS
EVALUATE GLOBAL DESIGN
LOCAL AND GLOBAL METRICS
VARIABLE WEIGHING FACTORS

Design Metrics

Procedure Metrics

	TOKEN COUNT	STATEMENT COUNT	STATEMENT SIZE	STMT NESTING LEVEL	PROC PARM COUNT	INTP	CMNT
Proc A							
B							
.							
.							
Module							

IF Construct and CASE Construct Metrics

	IFS	IF CONDITION SIZE	ELSEIFS PER IF	ELSEIF COND SIZE	CASE	CASEIFS PER CASE	CASEIF LIST SIZE	CASEIF
Proc A								
B								
.								
.								
Module								

LOOP Construct and LOOPFOR Construct Metrics

	LOOP	WHILES PER LOOP	WHILE COND SIZE	LFOR	INCR
Proc A					
B					
.					
.					
Module					

GOTO Construct and Miscellaneous Metrics

	GOTO	BKWD	EXTR	EXITPROC COUNT	LABL	REDN
Proc A						
B						
.						
.						
Module						

NORMALIZED DESIGN METRICS

SIMPLICITY METRICS

		STATEMENT COUNT	STATEMENT SIZE	DECISION COUNT	STMT NESTING LEVEL
PROCEDURE	A	0.967	0.0	1.000	1.000
	B	0.867	0.502	0.800	0.978
	.				
	.				
	.				
MODULE		0.766 (0.115)	0.289 (0.186)	0.614 (0.253)	0.896 (0.094)

STRUCTUREDNESS METRICS

		FLOW INTERRUPTION	MULTI-EXITS
PROCEDURE	A	1.000	1.000
	B	1.000	0.0
	.		
	.		
	.		
MODULE		1.000 (0.0)	0.627 (0.459)

READABILITY METRICS

		STATEMENT COUNT	COMMENT STMT	REDUNDANT INFO
PROCEDURE	A	0.967	1.000	1.000
	B	0.867	1.000	1.000
	.			
	.			
	.			
MODULE		0.766 (0.115)	0.971 (0.090)	0.992 (0.027)

ABSOLUTE MEASURES ON SELECTED DESIGN CHARACTERISTICS FOR THE MODULE

SIMPLICITY:	0.641
STRUCTUREDNESS:	0.814
READABILITY:	0.909
SUFFICIENCY:	0.699
RELIABILITY:	0.699
USABILITY:	0.699
SUCCINCTNESS:	0.992
UNDERSTANDABILITY:	0.912
VERIFIABILITY:	0.892
MODIFIABILITY:	0.814
MAINTAINABILITY:	0.873

BENEFITS

- QUALITY CONTROL AID
- EFFECTIVE VERIFICATION AND VALIDATION TOOL
- ENCOURAGES BETTER FUTURE DESIGN
- FLEXIBLE FOR VARIOUS APPLICATIONS
- COST AND TIME EFFECTIVE

FUTURE DIRECTIONS

- DATA STRUCTURES METRICS
- SUBJECTIVE DESIGN METRICS
- COMPARISON WITH METRICS IN THE LITERATURE
- EXTENSIVE VALIDATION

Alphabetical Listing of Attendees and Their Affiliations

Alford, Bill	NASA/GSFC
Arnold, Robert	Univ. of Maryland
Bailey, Angel	Univ. of Maryland
Bailey, John	Univ. of Maryland
Bailey, Theresa	NASA/GSFC
Barksdale, Joe	NASA/GSFC
Barnes, Bill	NASA/GSFC
Barth, Randy	NASA/GSFC
Basili, Vic	Univ. of Maryland
Belady, Laszlo	IBM
Bishop, Joe	NASA/HDQ
Bond, Jack	NSA
Boyles, George	NASA/Langley
Broske, Richard	CSC
Brown, Lottie	NASA/GSFC
Brown, Sarah	
Cahoon, Frank	NSA
Callender, Dave	JPL
Caron, Gary	IITRI
Cassidy, Pat	ESD
Cephas, Arnie	NASA/GSFC
Chen, Chinder	Syracuse Univ.
Church, Vic	CSC
Clarson, John	Stromberg Carlson
Clinedinst, Winston	CSC
Crowley, Jeanne	NASA/GSFC
Cruickshank, Robert	IBM
Curtis, Bill	ITT
Daniels, Herman	CSC
Dapkunas, Mimi	IITRI
Davlin, Jim	SAI
Decker, Bill	CSC
DeMeo, Joe	FAA
Drake, Fred	Ketron, Inc.
Dyer, Michael	IBM
Edwards, Betsy	NASA/GSFC
Ellis, Walter	IBM
Erickson, Dan	JPL
Fisher, Kurt	CSC
Gaffney, John	IBM
Gatterman, Bruce	SAI
Goel, Amrit	Syracuse Univ.
Golden, John	XEROX
Hannan, Sue	GE
Hannan, Tom	FAA
Haynes, Robert	ARINC
Hecht, Herb	SoHaR
Holmes, Barbara	CSTA
Horn, Martin	CRC
Hornstein, Rhoda	NASA/HDQ
Horton, Hubert	New Technology, Inc.

Houghton, Ray	NBS
Hutchens, David	Univ. of Maryland
Hwang, Vincent	Univ. of Maryland
James, Larry	Hughes Aircraft
Jamieson, Lillian	NASA/GSFC
Jones, Tony	CSTA
Kafura, Dennis	Iowa State Univ.
Knox, Richard	CSC
Kornfeld, Judith	Softech, Inc.
Kruesi, Elizabeth	GE
Kurihara, Tom	DOT
Laubenthal, Nancy	NASA/GSFC
Lawler, Robert	Boeing Aerospace
Littlewood, Bec	George Washington Univ.
Logan, James	TRW
Maione, Tony	NASA/GSFC
Masters, Tom	NSA
Mattson, Bob	CSC, Huntsville, Ala.
McGarry, Frank	NASA/GSFC
McGarry, Mary Ann	IITRI
Meltzer, Dick	GE
Miya, Gene	JPL
Moe, Karen	NASA/GSFC
Moy, Susan	LOGICON
Muckel, Jerry	NASA/GSFC
Murray, Paul	SAI
Musa, John	NASA/GSFC
Meyers, Lynn	NASA/GSFC
Nadelman, Matthew	CSC
Napjus, Chris	NSA
Ng, Edward	JPL
Noonan, Robert	College of Wm. & Mary
Ostrand, Tom	Sperry-Univac
Owings, Jan	NASA/GSFC
Perricone, Barry	Univ. of Maryland
Phillips, Tsai-Yun	Univ. of Maryland
Picasso, Gino	Univ. of Maryland
Pietras, John	MITRE
Pinsky, Sylvan	SSA
Plett, Mike	CSC
Postak, John	Doty Associates
Province, Phil	CSC
Puccinelli, Ed	NASA/GSFC
Putnam, Larry	QSM, Inc.
Ramapriyan, H. K.	NASA/GSFC
Reed, Larry	CRC
Reiter, Bob	IBM
Rich, Sybil	CRC
Sayani, Hasan	ASTEC
Scheffer, Paul	Martin Marietta
Schumacher, Lee	DOD
Schwenk, Bob	NASA/GSFC
Sheppard, Sylvia	GE

Spiegel, Andras	Ketron, Inc.
Stuart, Toni	USDA
Suddith, Steve	CSTA
Sukert, Alan	
Sullivan, Terry	CSC, Huntsville, Ala.
Super, Bill	SSA
Szulewski, Paul	Draper Lab
Tausworthe, Bob	JPL
Teague, Ellen	CSTA
Tidd, Randall	NSA
Tippett, Jim	NSA
Truss, Vivian	IITRI
Turner, Chris	IITRI
Velez, Tom	Computer Technology Assoc.
Wachs, Dick	CTA
Walston, Claude	IBM
Wyckoff, Dave	CSC
Zaveler, Saul	DOD
Zelkowitz, Marv	Univ. of Maryland
Zolnowski, Jean	Bell Labs

END

DATE

FILMED

JUL 22 1982